GLADE를 사용한 리눅스 데스크탑 어플리케이션 개발

선정필 `simje@maninet.com`

March 29th, 2000, 최종 갱신 : May 15th, 2000.

GLADE란 GNU General Public License(GPL)을 따르는 **GTK+ User Interface Builder**이다. 쉽게 얘기하면 **GTK+**라는 라이브러리를 이용하여 GUI 사용자 인터페이스를 만드는 RAD(Rapid Application Development) 툴이다. 이것이 무엇인지 그리고 이를 어떻게 구입/설치하는지 그리고 조그만 어플케이션 샘플인 텍스트 에디터를 개발하는 구체적인 단계들이 어떻게 되는지 리눅스 환경하에서, 소개 하고자 하는 것이 이 문서의 목적이다. 그리고 **GLADE**가 부족한 부분들에 대한 내용을 담고 있다. 이 문서에 대한 조언/비판/문의가 있으신 분은 이메일로 연락을 주시기 바란다. 다음 버전은 좀 더 깊숙한 내용을 다루어보도록 하겠다.

목 차

제 1 절 GLADE와 구입 및 설치

1.1 GLADE란 무엇인가?

`GLADE` 는 다른 윈도우즈(Windows NT/9x를 앞으로 이렇게 부르겠다)Visual Studio와 많은 부분이 유사하며 Widget Template Palette(윈도우즈 용어로 한다면 Control쯤에 해당하겠다) 템플리트 팔레트에 있는 위젯들을 드래그 앤 드롭으로 GUI 폼과 그 위의 GUI 객체들을 생성하고 이들의 프로퍼티들을 조정하고 시그널 핸들러(윈도우즈로 한다면 메시지 맵 함수) 등록/삭제/변경하여 C, C++ 소스(C++의 경우 addon인 **glade–**가 필요)를 생성할 수 있는 통합 개발 환경이다.

앞으로 C 소스를 생성하는 것에만 촛점을 맞춰서 얘기하겠다.

1.2 GLADE를 어디서 구할 것인가?

1.2.1 Source와 Binary

GLADE은 현재 0.5.7버전까지 release되었으며 공식 web site는 `http://glade.pn.org`이다.

다음 그림1 (GLADE 공식 홈 사이트)은 **GLADE**공식 홈 사이트 모습이다.

이곳에서 가장 최근 버전의 소스를 다운로드할 수 있으며 이곳에는 미리 만들어진 binary를 팩키징한 RedHat RPM, Debian, Slackware, NetBsd package들도 마련되어 있다. 급하신 분은 이런 binary의 팩키징을 선택할 수도 있겠다.

1.2.2 문서

아직 공식 배포되는 문서는 없다. 그러나 다음과 같은 내용들을 참조할 수 있겠다:

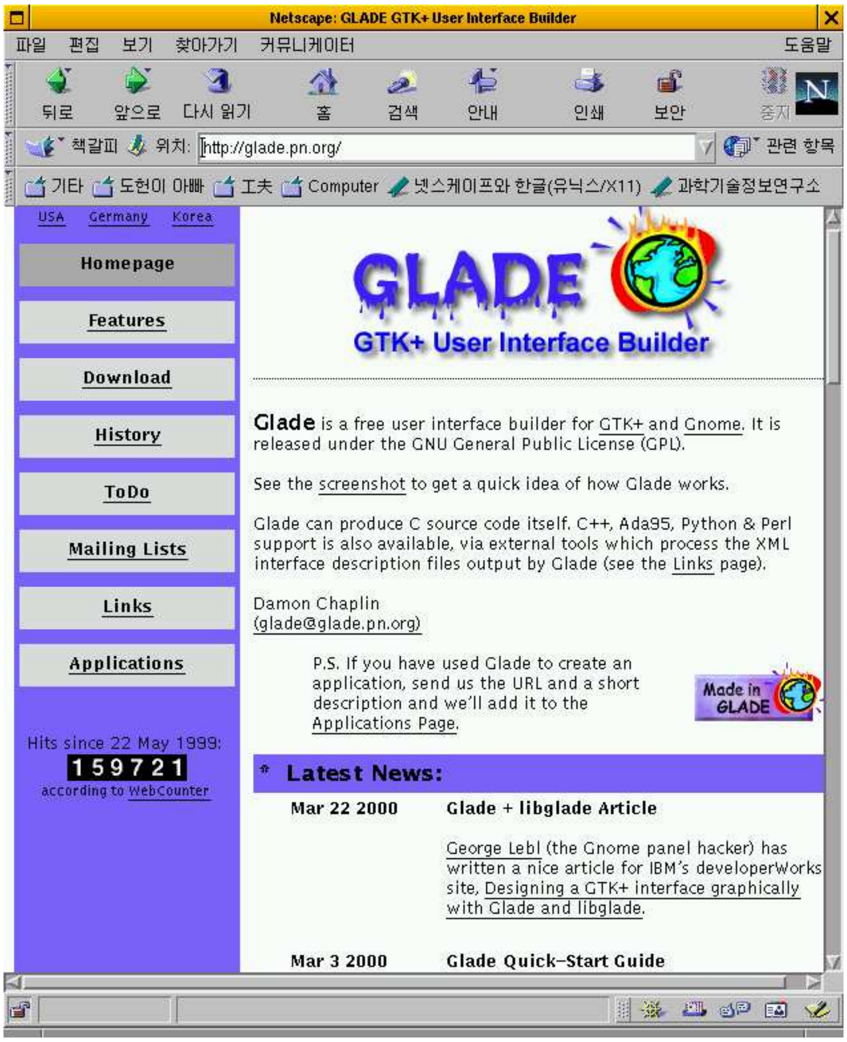
- **GLADE** 소스 자체
- **GLADE** 소스의 doc 디렉토리에 있는 파일들
- **GLADE** 소스의 examples 디렉토리에 있는 내용
- Designing a GTK+ interface graphically with Glade and libglade (`http://www-4.ibm.com/software/developer/library/gnome-glade/`)
- Glade turbo-start (`http://glade.pn.org/tutorial.txt`)

1.3 GLADE 설치

1.3.1 설치를 위해서 필요한 것들(requirements)

GLADE를 설치하려면 다음과 같은 것들이 필요하다.

- binary에도 필요한 라이브러리들



Glade is a free user interface builder for GTK+ and Gnome. It is released under the GNU General Public License (GPL).

See the screenshot to get a quick idea of how Glade works.

Glade can produce C source code itself. C++, Ada95, Python & Perl support is also available, via external tools which process the XML interface description files output by Glade (see the Links page).

Damon Chaplin
(glade@glade.pn.org)

P.S. If you have used Glade to create an application, send us the URL and a short description and we'll add it to the Applications Page.

Hits since 22 May 1999:
159721
according to WebCounter

Latest News:

Mar 22 2000	**Glade + libglade Article**
	George Lebl (the Gnome panel hacker) has written a nice article for IBM's developerWorks site, Designing a GTK+ interface graphically with Glade and libglade.
Mar 3 2000	**Glade Quick-Start Guide**

그림 1: GLADE 공식 홈 사이트

GTK+ 1.2.0 이상

– `<http://www.gtk.org>`

gnome-libs 1.0.16 이상

– Gnome 지원이 필요한 경우에 필요

Gettext 0.10.35

– `<ftp://alpha.gnu.org/gnu/gettext-0.10.35.tar.gz>` 이것은 gettext를 지원하는 어플리케이션을 작성하고자 할 때 필요.

- 소스로 설치하기 위해서 필요한 보조 도구들

Automake 1.4

– `<ftp://ftp.gnu.org/pub/gnu/automake>`, `<http://www.gnu.org/software/automake>`, `<ftp://rawhide.redhat.com/i386/RedHat/RPMS>`.

Autoconf 2.13

– `<ftp://ftp.gnu.org/pub/gnu/autoconf>`, `<http://www.gnu.org/software/autoconf>`, `<ftp://rawhide.redhat.com/i386/RedHat/RPMS>`

1.3.2 binary package 설치

RPM 등과 같은 binary형태의 팩키지는 그 설치가 용이하다. 예를 들어서 RPM은 다음과 같이 설치한다:

```
rpm -Uvh glade-0.5.7-2.i386.rpm
```

1.3.3 소스로 설치

소스로 설치하기 위해서는 automake 1.4와 autoconf 2.13이 필요하다. 이들은 대부분의 linux에 설치되어 있을 것이다. 없다면 찾아서 설치해주자. 그리고 소스로 설치하는 것은 다음과 같이, 리눅스 소스 설치의 전형적인 방법을 따르며, 그렇게 어려운 것이 아니다:

1. 맨먼저 해야 할 일은 다운 로드 받은 파일의 압축을 푸는 것이다.

   ```
   [simje@rurulala /usr/src/local/gtk]$ tar xvzf glade-0.5.5.tar.gz
   ```

2. 다음은 새로 생성된 디렉토리로 이동해서 *configure*를 실행시킨다.

   ```
   [simje@rurulala /usr/src/local/gtk]$ cd glade-0.5.5
   [simje@rurulala /usr/src/local/gtk/glade-0.5.5]$ ./configure
   ```

   여기서 특정 디렉토리에 설치될 것을 겨냥한다면 아시는 바와 같이 다음과 같이 입력해야 한다. 그렇지 않는 경우 디폴트로 `/usr/local`를 *prefix*로 설정하게 된다.

   ```
   [simje@rurulala /usr/src/local/gtk/glade-0.5.5]$ ./configure
   --prefix=/usr
   ```

3. 다음은 *make*을 실행시켜 컴파일하는 것이다.

   ```
   [simje@rurulala /usr/src/local/gtk/glade-0.5.5]$ make
   ```

4. 컴파일이 끝나면 다음과 같이 설치한다.

   ```
   [simje@rurulala /usr/src/local/gtk/glade-0.5.5]$ su -c "make install"
   ```

제 2 절 GLADE를 이용한 텍스트 에디터 개발 - 사전 단계

설치가 제대로 되었는가? 그렇다면 실행을 해보자. 이상없이 실행되었다면 이제 단순한 예제 개발을 통해서 **GLADE**를 익혀 보도록 하자:

2.1 사전 지식(preliminaries)

그냥 드래그 앤 드랍으로 개발할 수도 있지만 외형을 만드는 것보다 좀 더 깊숙히 들어갈려면 다음과 같은 사전 지식이 필요하다:

2.1.1 라이브러리

GLADE가 GUI를 만들기 위해서 많이 사용하는 라이브러리들로써 다음과 같은 것들이 있다:

GTK+

이것은 가장 기초적인 기반 지식으로써 다음과 같은 문서를 통해서 쉽게 자가 학습 할 수 있다 `<http://kldp.org/KoreanDoc/html/GtkTutorial/GtkTutorial.html>`. 그러나 좀 더 자세히 들여다 보거나 library API들을 보려면 다음 사이트를 찾아야 할 것이다. `<http://www.gtk.org>`. 이곳에는 많은 문서들과 레퍼런스들이 있다. 이 라이브러리 객체들은 대개 함수는 `gtk_`*xxx*, 메모리 객체는 `Gtk`*Xxx*, 매크로는 `GTK_`*XXX* 과 같은 형태를 가진다.

GDK

Xlib의 꼭대기에 있는 wrapper library로 고안된 라이브러리이다. 이것은 GTK+가 픽스맵이나 폰트 컬러등을 다룰 때 사용되는 것이다. 이에 대한 문서도 `<http://www.gtk.org>`에 가면 찾을 수 있다. 이 라이브러리 객체들은 대개 함수는 `gdk_`*xxx*, 메모리 객체는 `Gdk`*Xxx*, 매크로는 `GDK_`*XXX* 과 같은 형태를 가진다.

Glib

이것은 메모리 구조(링크드 리스트, 리스트, 해쉬, 캐쉬, 트리)나 타이머, 텍스트 출력과 같은 것을 다룰 때 사용되는 것이다. 이에 대한 문서도 `<http://www.gtk.org>`에 가면 찾을 수 있다. 이 라이브러리 객체들은 대개 함수는 `g_`*xxx*, 메모리 객체는 `g`*xxx*, 매크로는 `G`*XXX* 과 같은 형태를 가진다.

이를 구조화해서 그림으로 나타낸다면 다음과 같다(2 (gnome 아키텍쳐)).

2.1.2 GTK+ 튜터리얼 살펴보기

시그널과 시그널 핸들러

위의 **GTK+** 튜터리얼 페이지(`<http://kldp.org/KoreanDoc/html/GtkTutorial/GtkTutorial.html>`)를 보면 많은 기초적인 내용들에 대한 설명이 있다. 이들 중에서 가장 중요하다 싶은 것은 시그널과 시그널 핸들러(윈도우즈의 메시지 맵, 메시지 핸들러 등과 유사한 개념)이라 할 수 있겠다. 이는 대개 필요한 창을 *main* 함수(윈도우즈의 *WinMain()*함수)안에서 생성하고 디스플레이한 후, 메시지 펌프에 해당하는 *gtk_main()* 함수로 진입하는 데, 이 안에서 사용자 액션에 대한 시그널들이 생성되어서 어플리케이션에게 전달된다.

이를 위해서는 사전에 시그널과 시그널 핸들러를 맵핑시켜주는 준비 조작을 해주어야 하는 데 그 역할을 하는 것이 *gtk_signal_connect()*라는 함수를 통해서이다. **GLADE**가 만드는 소스 코드는 사용자가 **GLADE**를 통해서 등록하는 시그널들과 시그널 핸들러간의 맵핑을 어플리케이션의 메인 창을 생성하는 함수*create_window1*(이 이름은 절대적이지 않을 수 있다) 안에서 하고 있음을 앞으로 확인할 수 있을 것이다. 또한 이 함수 내에서는 어플리케이션의 메인 창위에 붙여 놓은 많은 위젯들을 생성하고 속성 설정하는 등의 일을 겸한다.

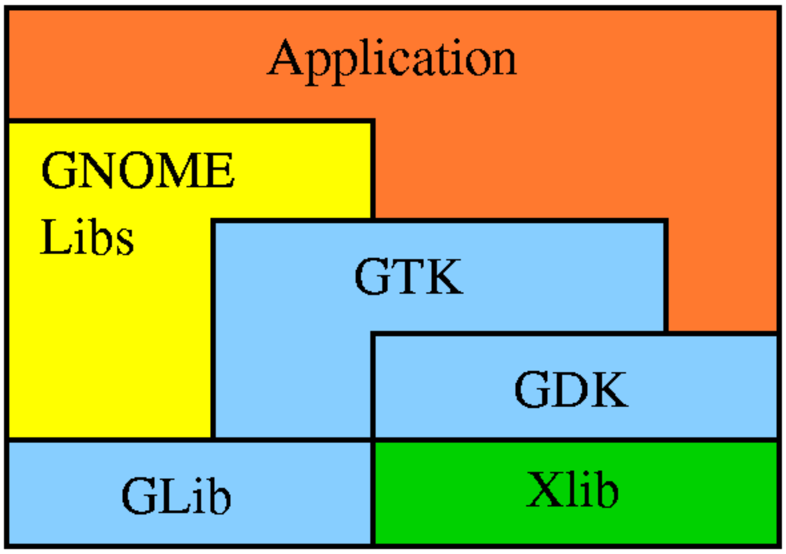


그림 2: gnome 아키텍쳐

팩킹 및 수평/수직/테이블 박스

다음으로 팩킹이라는 개념인 데 이것은 윈도우즈에는 없는 개념이다(혹시 MFC내부적으로 구현되어 처리되는지는 모르지만 말이다). 그렇게 어려운 개념은 아니고 여러가지 위젯을 색종이 잘라 붙이기 처럼 붙여 나갈 때 밑에 있는 것과 그 위에 있는 것의 바탕으로 생각하는 개념이며 주로 레이아웃에 대한 기능 지원을 한다. 이것은 윈도우즈 보다는 자바에 있는 컨테이너 및 레이아웃 개념과 유사하다고 할 수 있겠다.

GTK+를 곧이 곧대로 프로그래밍한다면 조금 꺼끄러울 수 있겠지만, **GLADE**에서는 이것을 시각적으로 처리해 준다. 수평 박스(자바 용어로 한다면 컨테이너) 및 수직 박스 그리고 테이블 박스를 제공하며 여기에다 여러가지 객체들, 즉 위젯들을 손쉽게 붙였다 떼었다 할 수 있다.

위와 같은 개념을 가지고 만들어진 어플리케이션은 다음 그림3 (GTK로 만든 어플리케이션 내부 계층 구조)과 같은 내부 계층 구조를 갖는다.

2.2 GLADE 살피기

2.2.1 세 개의 디폴트 윈도우

GLADE를 실행하면 다음과 같은 세개의 창이 디폴트로 뜬다:

주 윈도우(main glade window)

이것은 프로젝트를 구성하는 모든 창들과 대화상자들을 담고 있다. 이들 중의 하나를 더블 클릭하면 이에 대응하는 창이나 대화상자 템플리트가 뜬다. 그리고 이에 대응하는 창이나 대화상자의 프로퍼티를 설정할 수 있도록 프로퍼티 에디터 객체가 이것으로 변경된다.

위젯 팔레트 윈도우(widget palette window)

이것은 가능한 모든 위젯들의 들어 있는 팔레트이다. 새로운 창이나 대화 상자를 프로젝트에 추가하려면 팔레트에 있는 창이나 대화상자를 클릭하기만 하면된다. 이렇게 생성된 창이나 대화상자에 위젯을 붙이려면 위젯을 클릭해서 선택한 다음 더하고자 하는 위치에 클릭하면 된다.

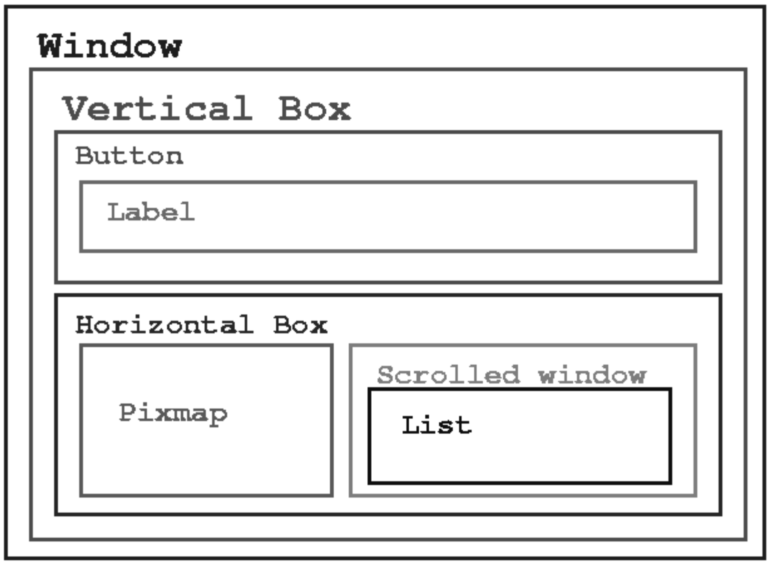


그림 3: GTK로 만든 어플리케이션 내부 계층 구조

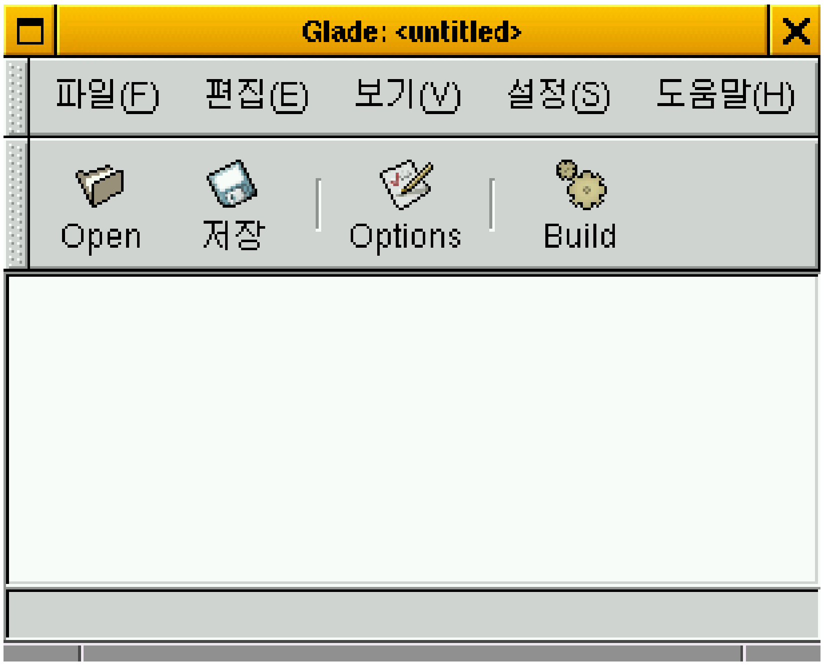


그림 4: GLADE 메인 윈도우

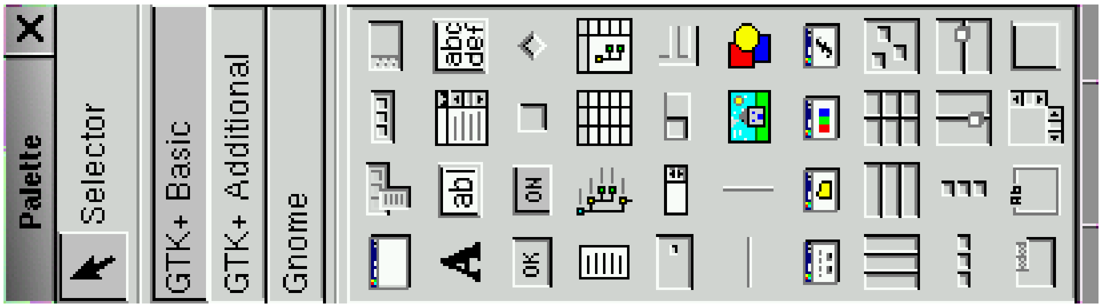
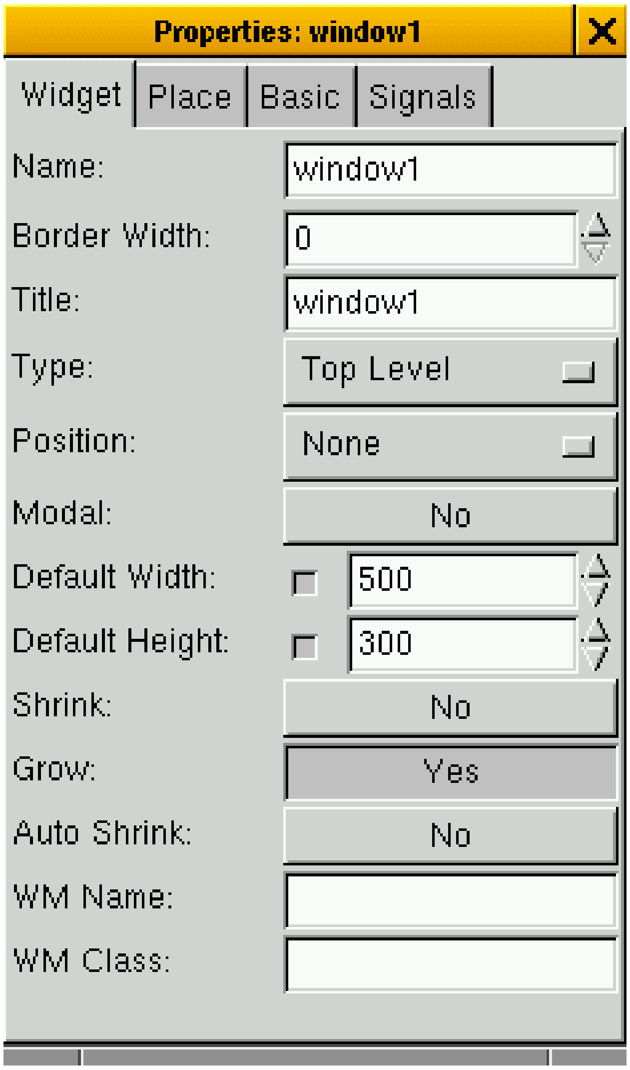


그림 5: GLADE 위젯 팔레트 윈도우



Name: window1
Border Width: 0
Title: window1
Type: Top Level
Position: None
Modal: No
Default Width: ☐ 500
Default Height: ☐ 300
Shrink: No
Grow: Yes
Auto Shrink: No
WM Name:
WM Class:

그림 6: GLADE 속성 편집기 윈도우

속성 편집기 윈도우(property editor window)

이것은 창이나 대화상자, 그리고 이들 위에 있는 위젯들의 크기 또는 텍스트 등과 같은 속성들을 변경할 때 사용한다. 주 윈도우에 있는 창이나 대화상자를 클릭하여, 또는 창이나 대화상자 위에 있는 위젯을 클릭해서, 프로퍼티를 설정할 대상 객체를 선택할 수 있다.

2.2.2 기타 보조 창들

GLADE는 다음과 같은 보조 창들을 가지고 있으며 주로 메뉴를 통해서 접근될 수 있다:

메뉴 작성 윈도우

이것은 프로젝트 창에 메뉴를 추가한 경우 이 메뉴를 클릭한후 속성 편집기 윈도우에 나타나는 *edit menu* 버튼을 누르면 나타난다. 이것을 통해서 여러분은 프로젝트의 메뉴를 편집할 수 있다.

이곳에서 생성된 메뉴는 소스에 자동으로 들어가게 된다.

*add*를 누른 후에 *Stock Item* 콤보박스를 눌러서 미리 정의된, 그리고 널리 쓰이는 메뉴를 선택할 수 있다. 이 때 *Handler*가 이에 맞춰 정형화된 꼴로 나타나게 되어 있는 데, 한글화된 버전의 경우 한글로 된 함수가 *Handler*가 되어서 소스에 깨지거나 한글 부분이 언더라인으로 채워진 함수가 나타나게 된다. 고로 *Handler* 부분은 되도록이면 영문으로 바꿔 주는 것이 좋겠다.

위젯 트리(widget tree)

이것은 프로젝트에 사용된 모든 위젯들의 트리 구조를 보여주는 윈도우이다. 이것은 주 윈도우의 *View/Show WidgetTree*을 선택해서 볼 수 있다.

2.3 분석/설계 - 텍스트 에디터 기능

우리가 만들고자 하는 텍스트 에디터는 단순한 데스크탑 어플리케이션으로써 우선 윈도우즈의 notepad를 닮았다고 생각하시면 된다. 그렇다면 이 에디터가 지원해야 하는 그래서 구현해야 하는 기능들 리스트를 살펴보자:

- 편집창에 문자 입력, 선택, 복사, 삭제, 붙이기 기능
- 편집된 내용을 저장하는 기능
- 파일을 읽어와 편집창에 디스플레이 하는 기능
- 종료 기능
- about 대화 상자 기능

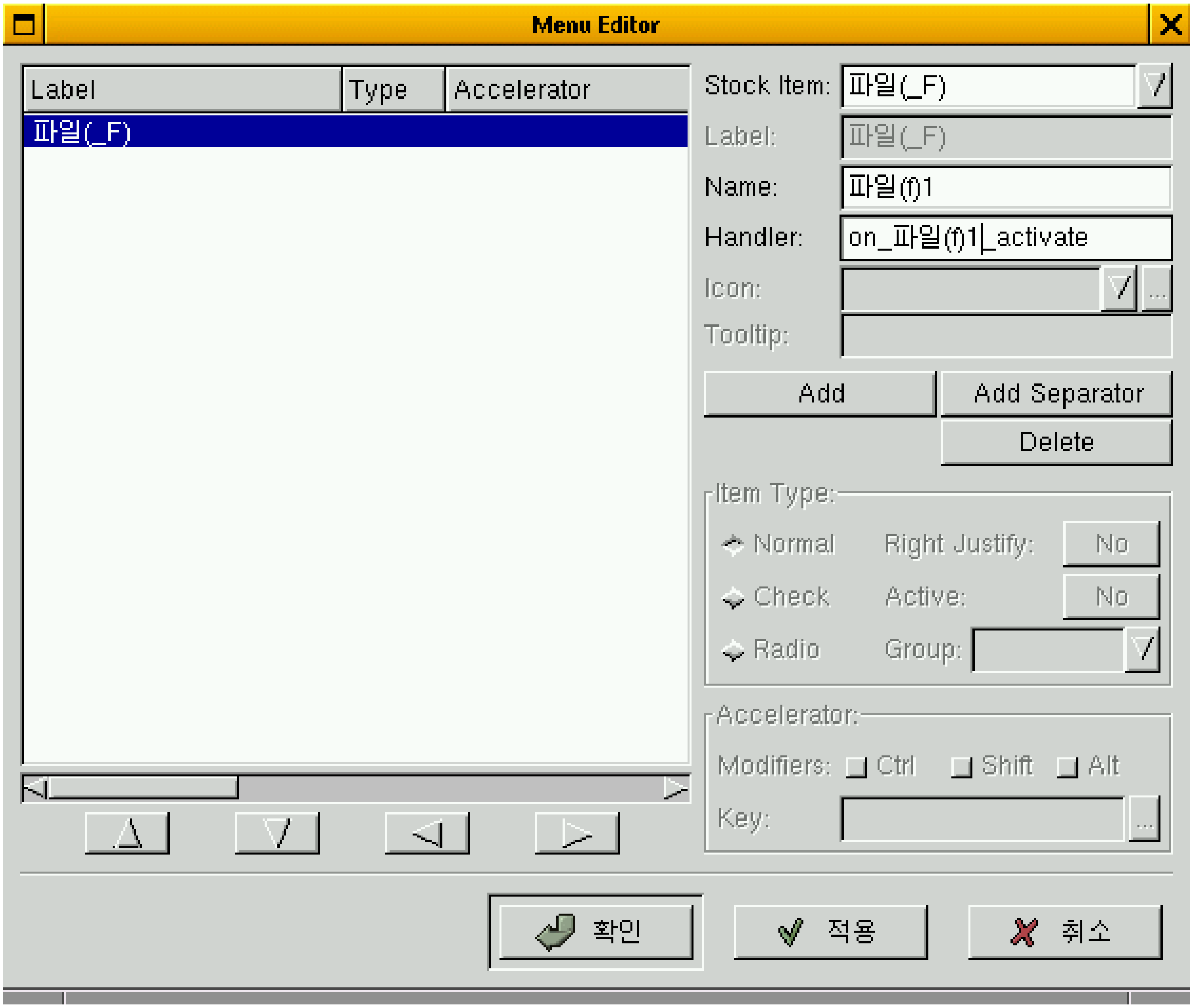
제 3 절 GLADE를 이용한 텍스트 에디터 개발 - 단계적 접근

GLADE를 실행시키면 위에서 설명한 바와 같이 세개의 창이 뜬다. 여기에서는 이런 상황에서 다음과 같은 단계별 접근으로 위에서 정의된 기능을 가진 텍스트 에디터를 개발하는 모습을 살피도록 한다:

3.1 개발 첫번째 단계 - 바탕 윈도우

바탕 윈도우 - 생성

우선 텍스트 에디터의 바탕이 되는 창을 만들어야 한다. 그럴려면 위젯 팔레트 윈도우에서 다음 그림의 왼쪽 상단의 위젯을 클릭해서 만들 수 있다.



파일(_F)

Stock Item: 파일(_F)

Label: 파일(_F)

Name: 파일(f)1

Handler: on_파일(f)1_activate

Right Justify: No

Active: No

그림 7: GLADE 메뉴 작성 윈도우

Widget Tree

- window1
  - vbox2
    - toolbar1
      - button3
      - button4
      - button5
      - button6
    - menubar1
      - 파일(f)1
        - 파일(f)1_menu
          - 새로운(n)1
          - 열기(o)1
          - 저장(s)1
          - 다른_이름으로_저장(a)1
          - 인쇄(p)1
          - 인쇄_설정(e)1
      - 편집(e)1
        - 편집(e)1_menu
          - 자르기(c)2
          - 복사(c)1
          - 붙이기(p)1
          - 모두_선택(s)1
      - 도움말(h)1
        - 도움말(h)1_menu
          - 이_프로그램은(a)1
    - entry1
    - statusbar1
- about1

그림 8: GLADE 위젯 트리

그림 9: 윈도우 위젯

그러면 다음 그림과 같은 바탕 창을 얻는다. 이 그림을 자세히 보면 사선 격자들이 촘촘히 회색으로 그려져 있는 것을 볼 수 있다. 이런 표시는 앞으로도 나오겠지만 무언가를 붙일 수 있는 곳이라는 얘기이다. 마치 접착식 앨범의 비닐을 들어 올리면 나타나는 접착제 사선과 같은 것이라고 생각할 수 있겠다.

이제는 여기에다 필요한 위젯들을 붙여야 되겠다. 먼저 필요한 것들을 생각해보면 메뉴, 툴바, 텍스트 박스, 그리고 상태 바가 있겠다. 이것을 그냥 하나씩 하나씩 붙일 수 있을까. 바로 위의 그림을 자세히 보면 붙일 수 있는 접착제 부분은 커다랗지만 구획이 나누어져 있지 않다. 즉 하나의 객체만을 붙일 수 있다는 얘기가 된다. 그러므로 어떻게 해야 할 까?

바탕 윈도우 - 메뉴바 위젯 붙이기

답은 위에서 언급한 바 있는, 다음 그림과 같은 *수직 박스*라는 위젯을 사용하면 된다. 이것을 붙이면 그 그 위에 여러개의 접착 가능한 구획이 생기게 되어 여러가지 위젯들을 붙일 수 있게 된다.

이것을 클릭하면 마우스가 십자가 모양으로 바뀐다. 이 때 이 마우스 커서를 바탕 윈도우에 대고 클릭하면 다음 그림과 같이 몇 개의 수직 박스를 만들 것인가를 묻는 대화상자가 나타난다.

여기서 우리는 4를 입력하자. 왜냐면 바탕 창에 붙이고자 하는 객체가 4개이기 때문이다. OK를 누르면 다음과 같이 바탕 창이 평등 분할되어 나타날 것이다.

자 이제는 이 네개의 구획 각각에다 메뉴바, 툴바, 텍스트 박스, 그리고 상태 바를 붙여 보도록 하자. 먼저 메뉴바를 붙이려면 다음 그림과 같이 생긴 메뉴바 widget을 위젯 팔레트 윈도우에서 클릭하여 선택한다.

그리고 바탕 창 네 구획 중 가장 위에 있는 구획을 클릭한다. 그러면 바탕 화면은 다음과 같이 변한다.

이것을 자세히 보면 접착제가 발라져 있는 부분은 3개로 줄어든 것을 알 수 있다. 조금 전에 최상단에 붙인 메뉴 부분은 접착제가 아닌 것을 또한 알 수 있다. 그러나 이부분은 별로 메뉴 같아 보이지 않다. 왜냐면 실제로 메뉴를 입력하지 않았기 때문이다. 그렇다면 메뉴를 이제 넣어 보자.

바탕 윈도우의 조금 전에 붙인 메뉴를 클릭하면 프로퍼티 윈도우가 다음 그림처럼 나타난다. 이곳을 자세히 보면 다른 위젯을 클릭해서 선택할 때와는 다르게 메뉴 편집(*Edit Menus...*)이라는 버튼이 하나 더 있다. 이것을 클릭하자.

그러면 위에서 본 *그림 5*와 같은 화면이 뜬다. 여기에 다음 그림처럼 편집해서 넣도록 하자. 단 모두 기본적으로 제공되는 *Stock Item*을 사용해서 만든 것이다. 먼저 *Add* 버튼을 눌러서 하나의 아이템을 만든 뒤에 *Stock Item* 콤보 박스를 클릭해서 미리 마련된 아이템을 선택하면 조금 전에 만들어진 것이 원하는 것으로 바뀐다. 그리고 *Name*이 한글로 나올 경우, 이것은 나중에 알게 되겠지만 소스 생성 후 소스 내부에서 모두 깨지거나 밑줄로 표기되어 뭐가 뭔지 모르게 된다. 이 때는 *Name*을 적절한 것으로 바꾸자. 즉 *새로운*이라면 *New*으로 바꾸자. 그러면 자동으로 그 아래에 있는 *Handler*가 바뀐다. 정말 좋다. 그리고 왼쪽 리스트 창에 있는 *Label*에 해당하는 것들은 그 아래에 있는 방향성 있는 화살표 등으로 제어되는 것으로써 메뉴의 깊이 및 계층 구조를 꾸밀 때 사용된다.

바탕 윈도우 - 툴바 위젯 붙이기

자 이제는 바탕 윈도우에 다음 그림과 같이 위젯 팔레트에 있는 툴바 위젯을 붙여 보도록 하자.

이것을 클릭하면 마우스 커서가 십자가 모양으로 변한다. 이것을 우리가 원하는 위치인 위에서 두번째 칸에 놓고 클릭한다. 그러면 다음과 같이 몇개의 툴바를 원하는가하는 대화 상자가 뜬다.

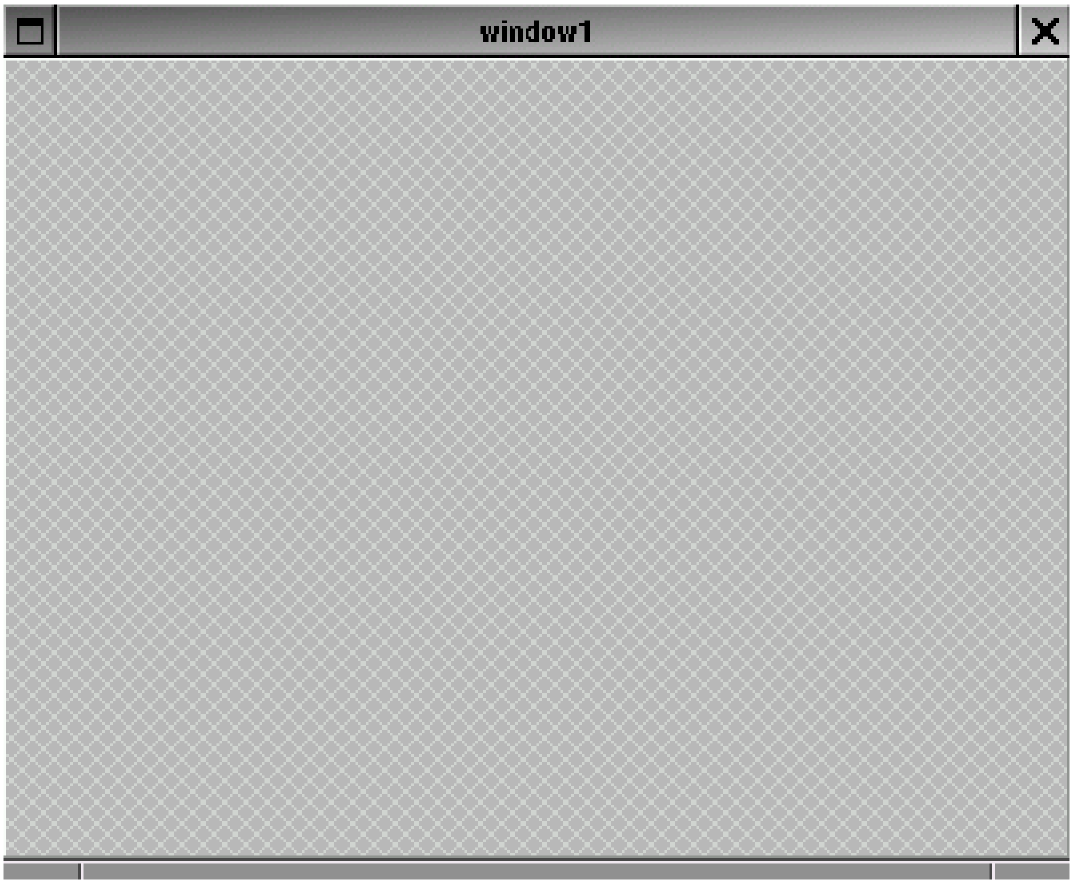


그림 10: 바탕 윈도우

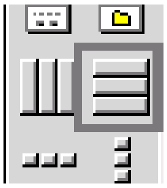
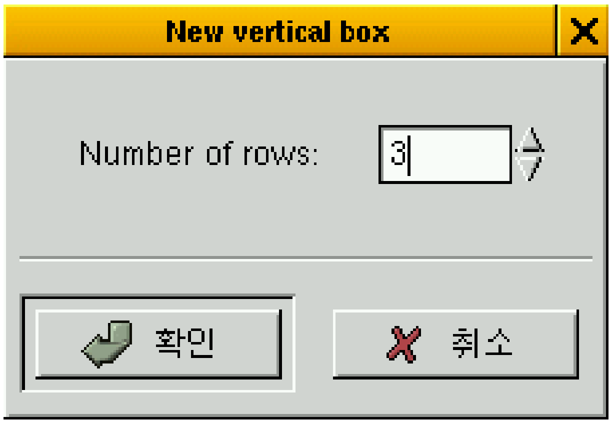


그림 11: 수직 박스 위젯



Number of rows: 3

그림 12: 수직 박스 위젯 개수를 묻는 대화상자

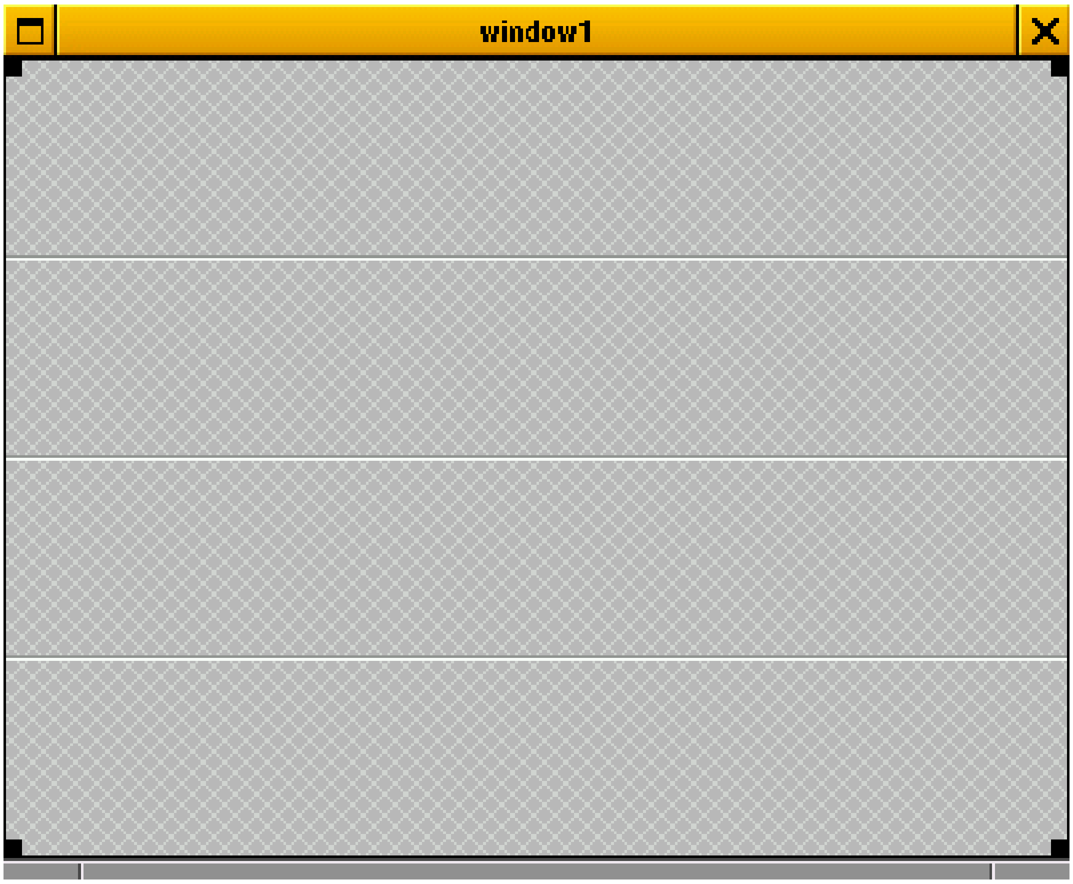


그림 13: 수직 박스를 붙인 후의 바탕 윈도우 모습

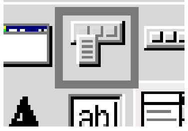
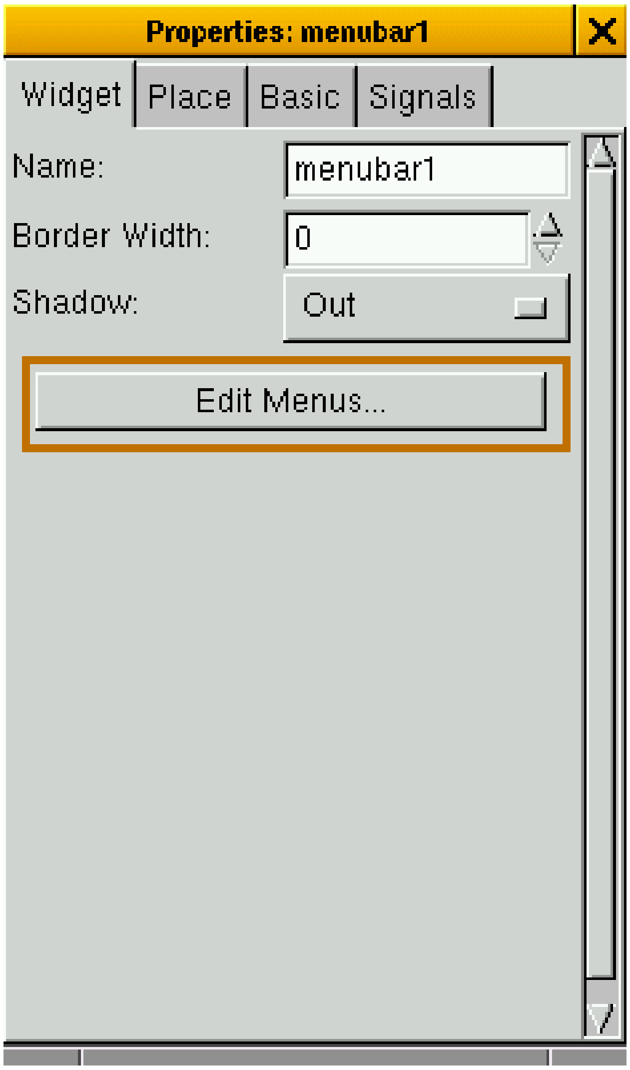


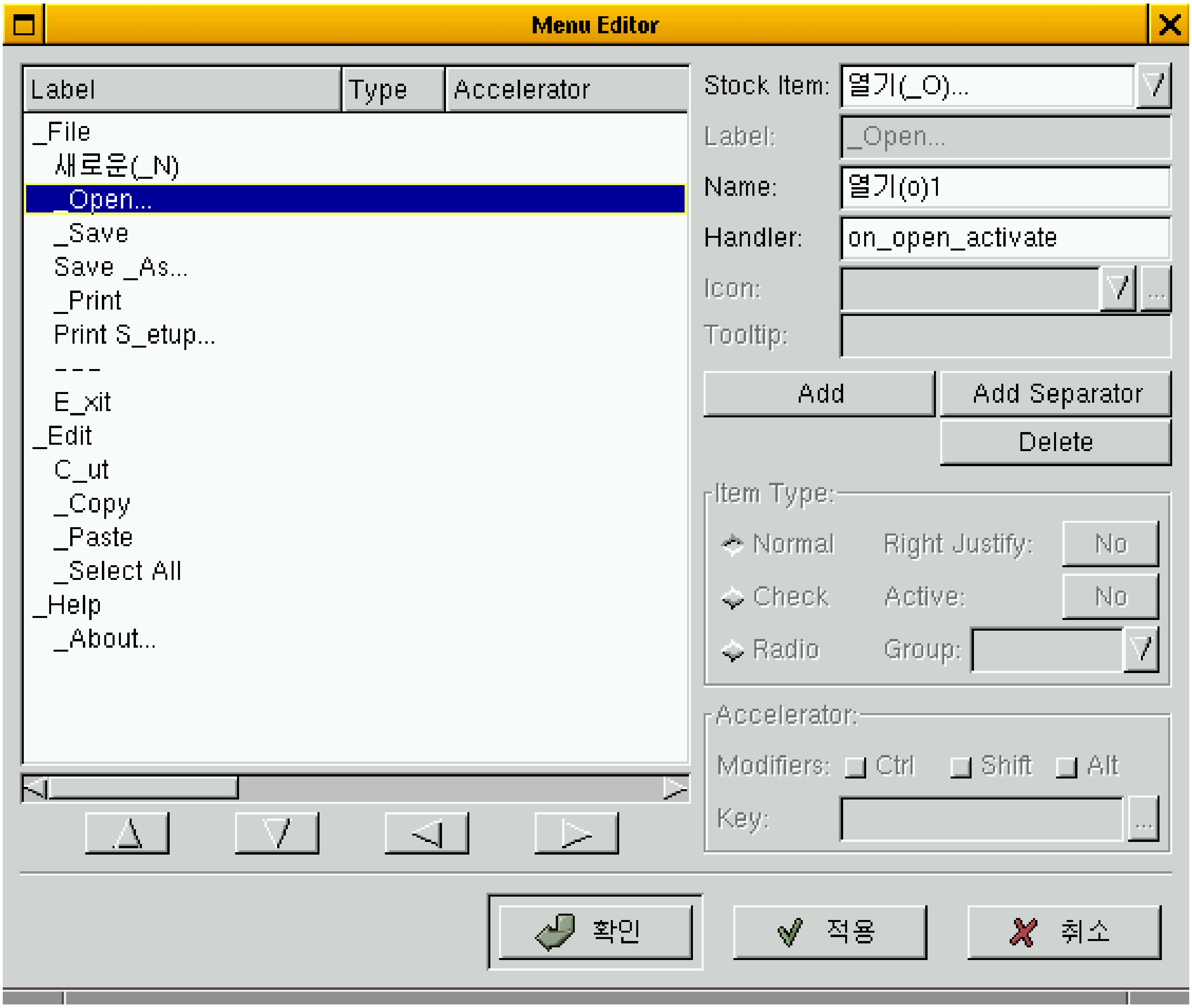
그림 14: 메뉴 위젯

그림 15: 메뉴 위젯을 붙인 후의 바탕 윈도우



Name: menubar1
Border Width: 0
Shadow: Out

그림 16: 메뉴 편집 버튼



Stock Item: 열기(_O)...

Label: _Open...

Name: 열기(o)1

Handler: on_open_activate

그림 17: 메뉴 편집 예제

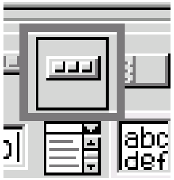


그림 18: 툴바 위젯

그림 19: 툴바의 버튼 수를 묻는 대화상자

여기에 원하는 개수(이 경우는 4개)를 입력하고 OK를 클릭하면 다음 그림 처럼 두번째 행이 4개의 새로운 접착판을 가진 것으로 변경되는 것을 볼 수 있다.

자 이제는 이 새로 생긴 조그만 4개의 접착판에다 툴바버튼들을 집어 넣어야 한다. 그럴려면 다음 그림과 같이 위젯 팔레트에서 버튼 위젯을 선택해야 한다.

이것을 선택한 후 마우스를 바탕 윈도우 두번째 행 첫번째 접착판으로 가져 가면 커서가 십자가 모양으로 변한다. 이것을 클릭하면 다음과 같이 바탕 화면이 변한다.

이 때 변경된 프로퍼티 윈도우를 자세히 보면 *Icon* 옆에 아래를 가르키는 콤보박스 버튼이 있다. 이것을 누르면 다음과 같은 미리 정의된 *Stock Icon*들이 나온다.

여기에 보이는 것 말고도 아래로 스크롤 하면 정말 많고 예쁜 아이콘들이 준비되어 있음에 놀랄 것이다. 이 중에 하나를 선택하자. 그리고 프로퍼티에서 *Label* 부분을 아이콘에 맞추어 바꾸도록 하자.. 그리고서 나머지 세개의 접착판들에 원하는 것을 넣으면 되겠다. 그러면 다음 그림과 같은 것을 얻는다.

바탕 윈도우 - 텍스트 박스 위젯 붙이기

다음은 세번째 행에다 다음과 같이 위젯 팔레트 윈도우에 있는 텍스트 박스 위젯을 선택하여 붙여 보도록 하자.

이것을 클릭한 후 마우스를 예의 바탕 윈도우 세번째 행으로 옮기면 커서가 십자가로 바뀐다. 클릭하자 그러면 다음 그림처럼 바탕 윈도우가 변한다.

그런데 프로퍼티 윈도우를 보면 *Editable*이 디폴트로 *No*으로 되어 있다. 이것은 *Yes*로 변경해야만 나중에 텍스트를 입력할 수 있다. 그런데 전체적인 크기가 마음에 들지 않을 수 있다. 그런데 내부의 위젯 크기를 변경하는 것보다 바깥 바탕 윈도우의 크기를 조정하는 것이 좋다. 다음 그림 처럼 바탕 윈도우의 크기 속성을 변경하도록 하자. 그러면 나중에 실행 결과가 이에 맞춰질 것이며 나머지 이 바탕 윈도우가 품고 있는 객체들의 크기는 적절하게 배분될 것이다.

바탕 윈도우 - 상태바 위젯 붙이기

자 이제는 마지막으로 바탕 윈도우에 다음 그림과 같은 위젯 팔레트 윈도우에 있는 상태바 위젯을 선택해서 상태바 위젯을 붙여 보도록 하자.

이것을 선택한 후 마지막 남은 접착판 위에 마우스를 가져가면 마찬가지로 십자가 모양으로 커서가 변경된다. 클릭하면 다음과 같이 바탕 윈도우가 변한다.

바탕 윈도우 - 만든 것을 저장하고 소스 생성 그리고 실행

이로써 우리는 바탕 윈도우를 만들고 이 위에 몇가지 GUI 객체들을 붙이고 약간의 조정을 해보았다. 어떤가? 그럴듯하지 않는가. 이제는 다음 그림 처럼 **GLADE**의 *File*메뉴에 있는 *Save*를 선택해보자.

그러면 다음 그림과 같은 *Project Options*라는 대화상자가 뜬다. 여기에 적절한 값을 입력하고 OK를 누르면 프로젝트가 저장된다. 단, 생성될 언어로는 C를 선택하도록 하자.

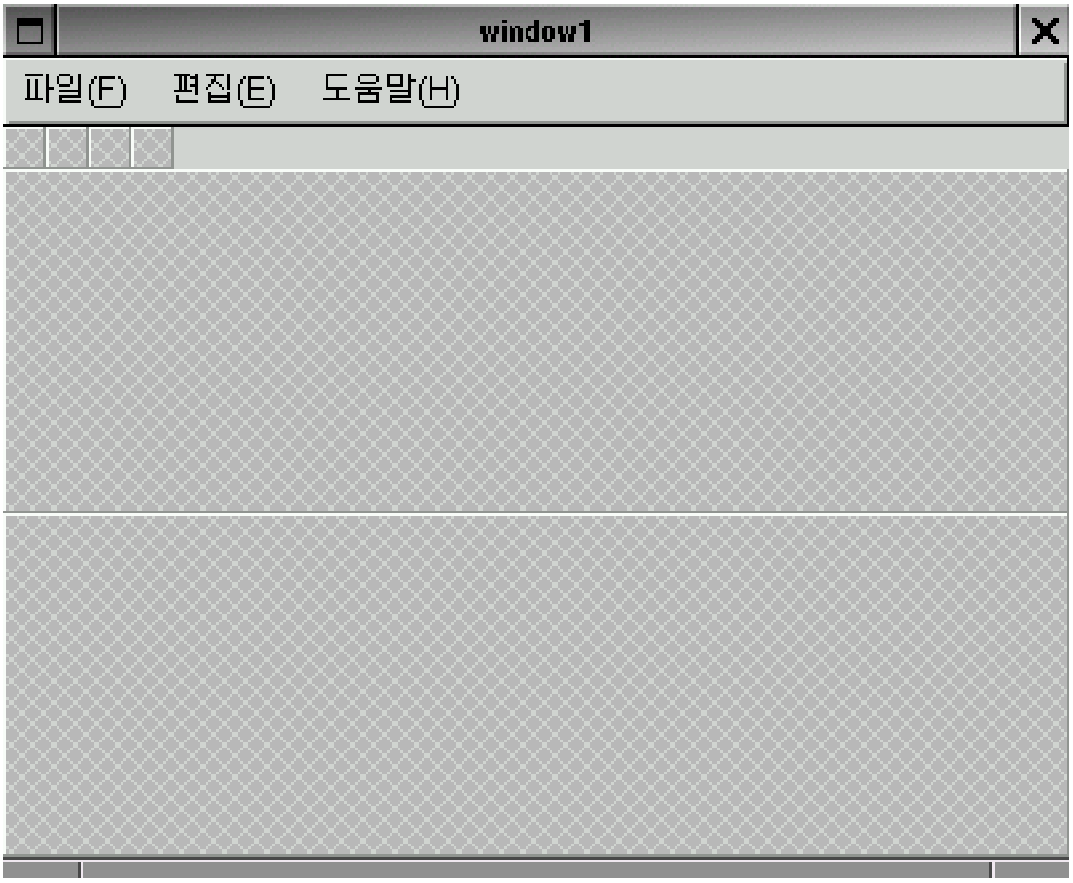


그림 20: 툴바를 붙인 후의 바탕 윈도우

그림 21: 버튼 위젯

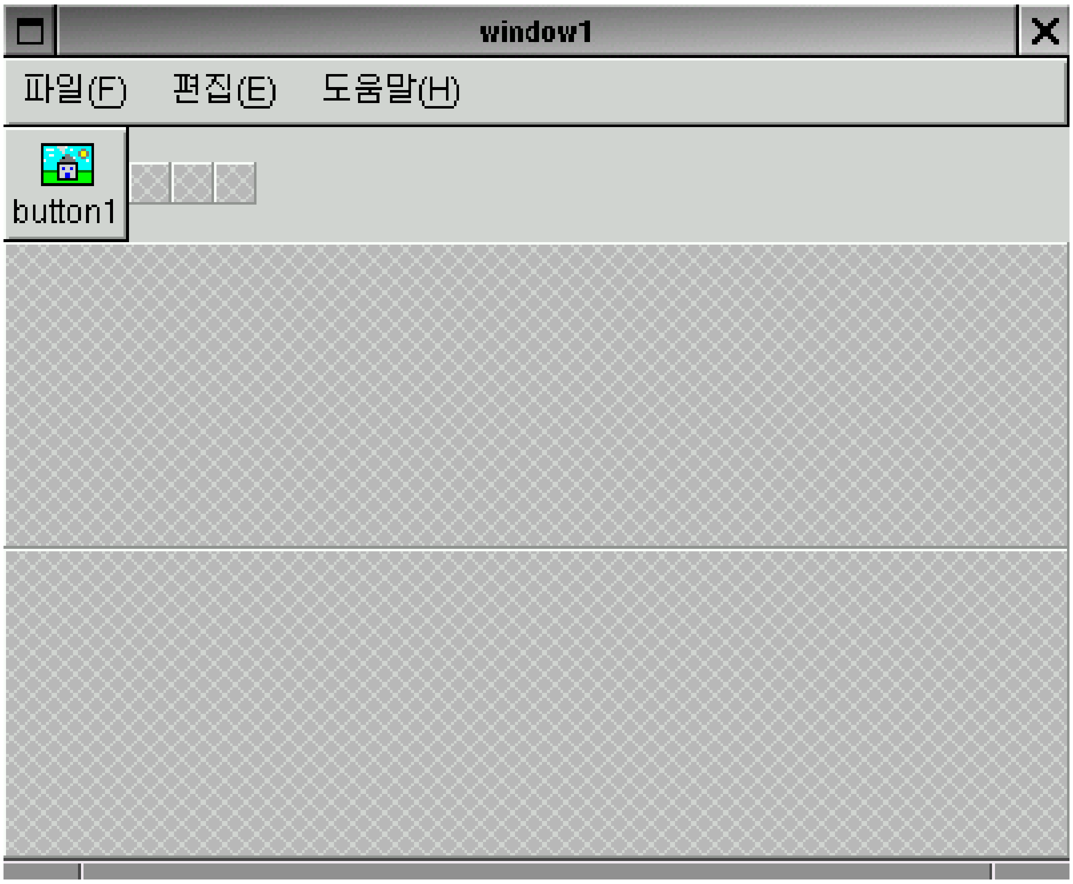


그림 22: 한개의 버튼을 붙인 후의 바탕 윈도우

그림 23: 미리 준비된 버튼들

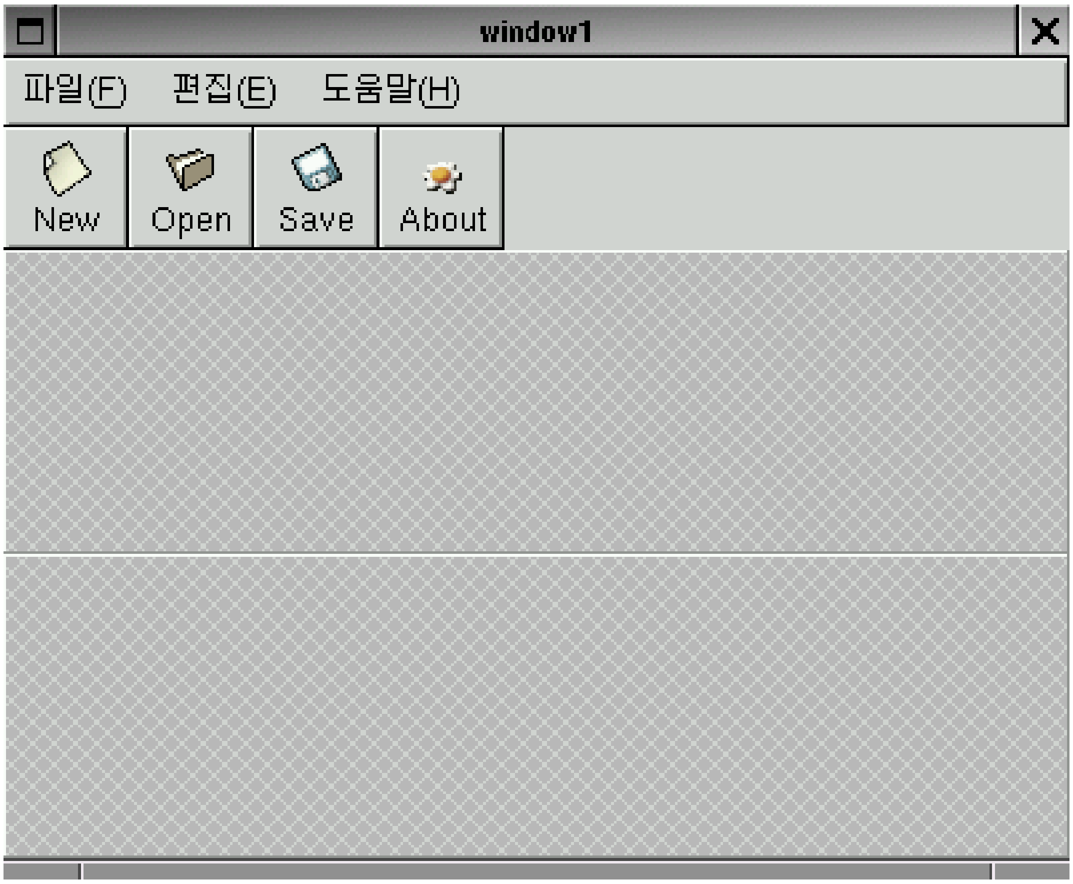


그림 24: 툴바에 모든 버튼들을 붙인 후의 바탕 윈도우

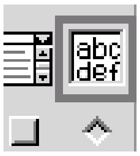


그림 25: 텍스트 박스 위젯

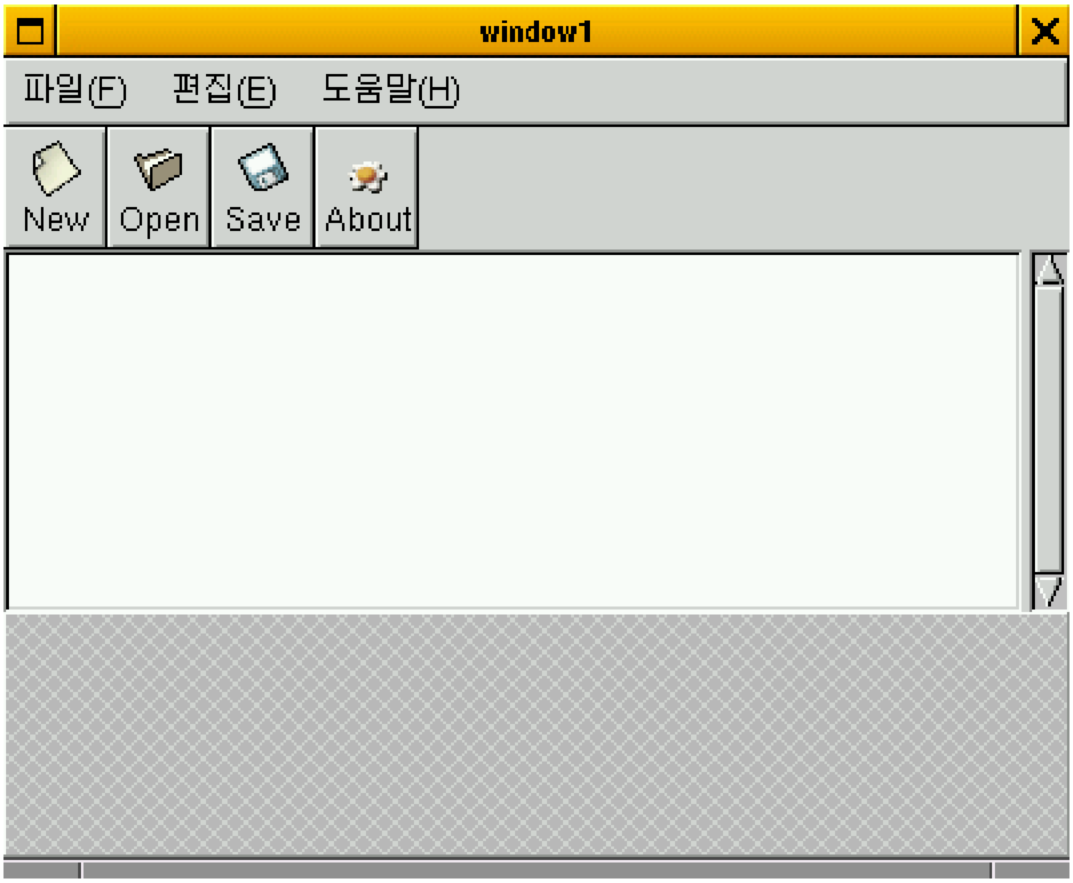
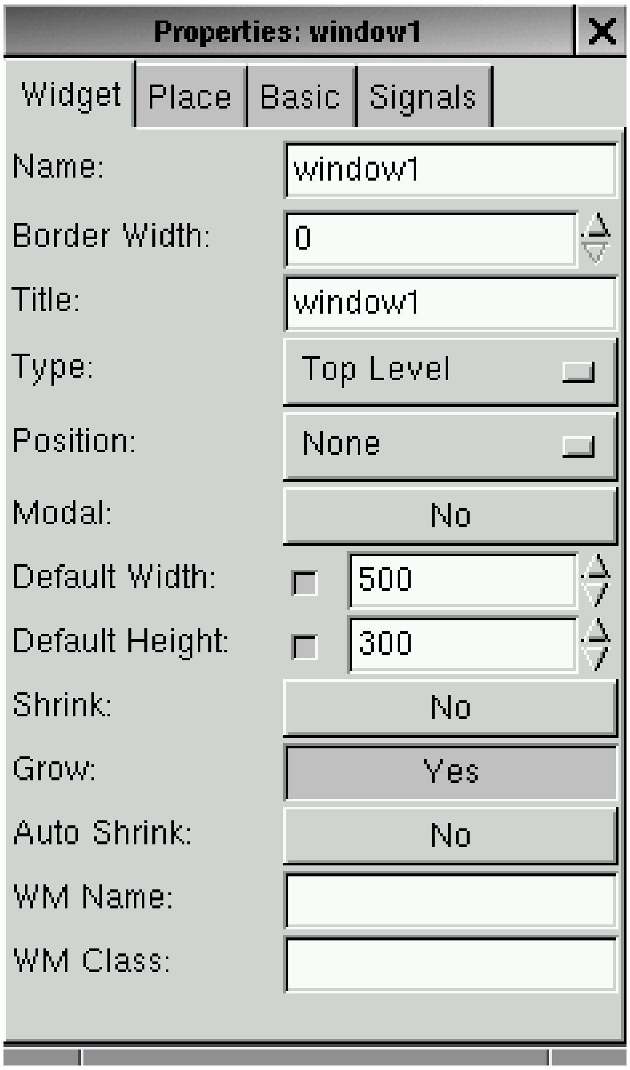


그림 26: 텍스트 박스 위젯을 붙인 후의 바탕 윈도우



Name: window1
Border Width: 0
Title: window1
Type: Top Level
Position: None
Modal: No
Default Width: ☐ 500
Default Height: ☐ 300
Shrink: No
Grow: Yes
Auto Shrink: No
WM Name:
WM Class:

그림 27: 바탕 윈도우의 크기 조정

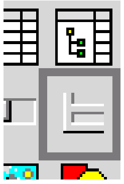


그림 28: 상태바 위젯

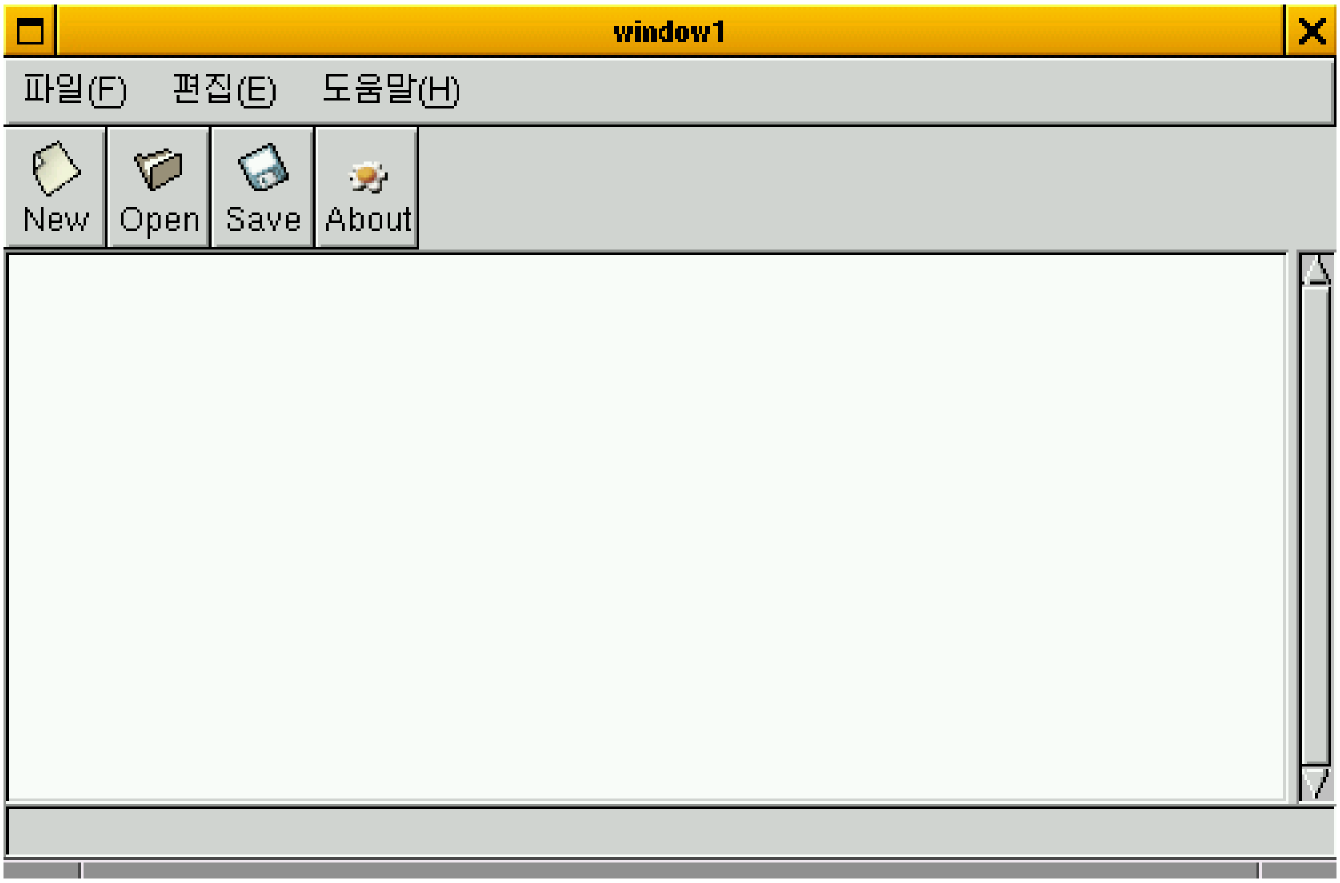
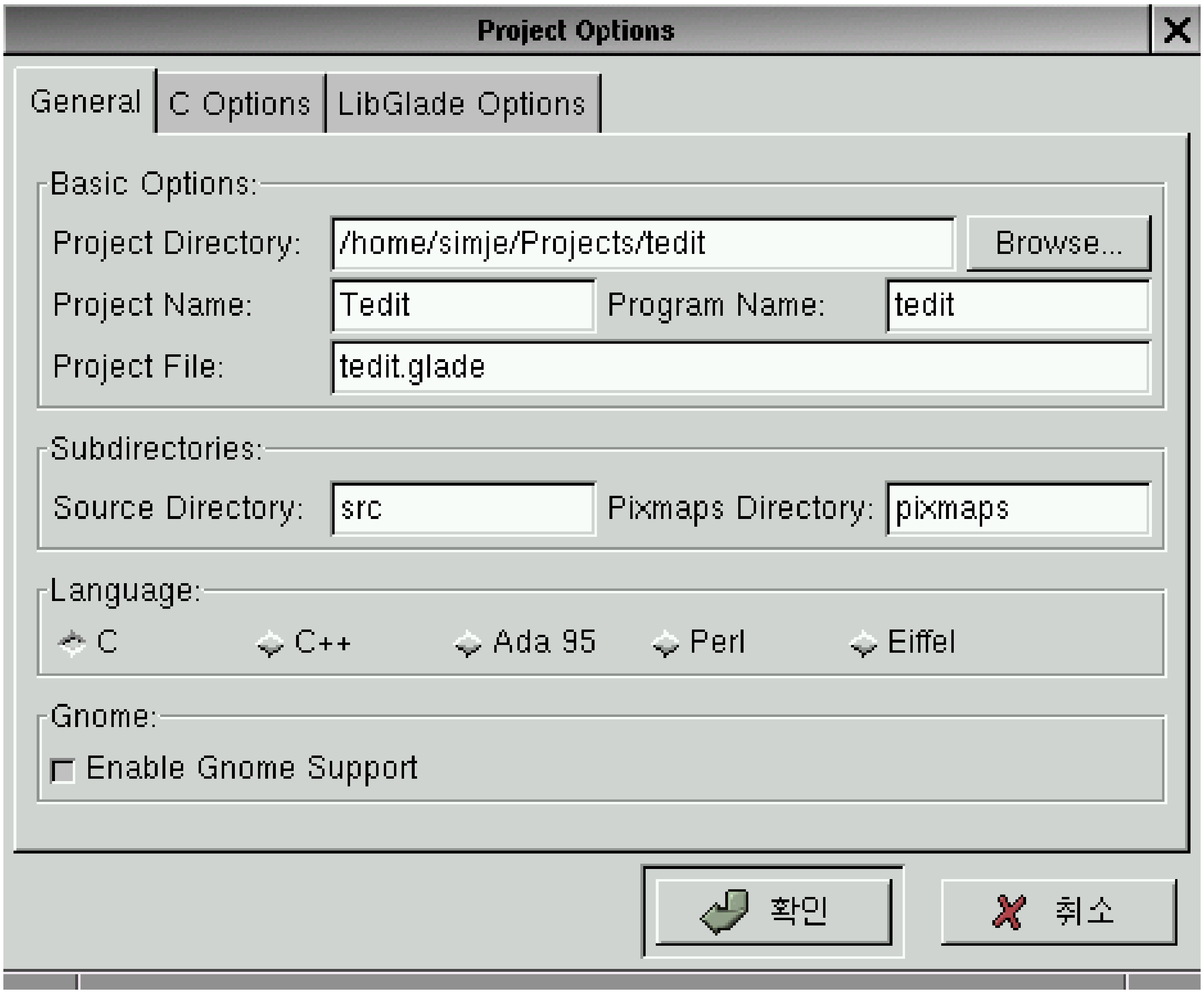


그림 29: 모든 위젯들을 붙인 후의 바탕 윈도우

그림 30: GLADE의 파일 메뉴



Basic Options:

Project Directory: /home/simje/Projects/tedit

Project Name: Tedit

Program Name: tedit

Project File: tedit.glade

Subdirectories:

Source Directory: src

Pixmaps Directory: pixmaps

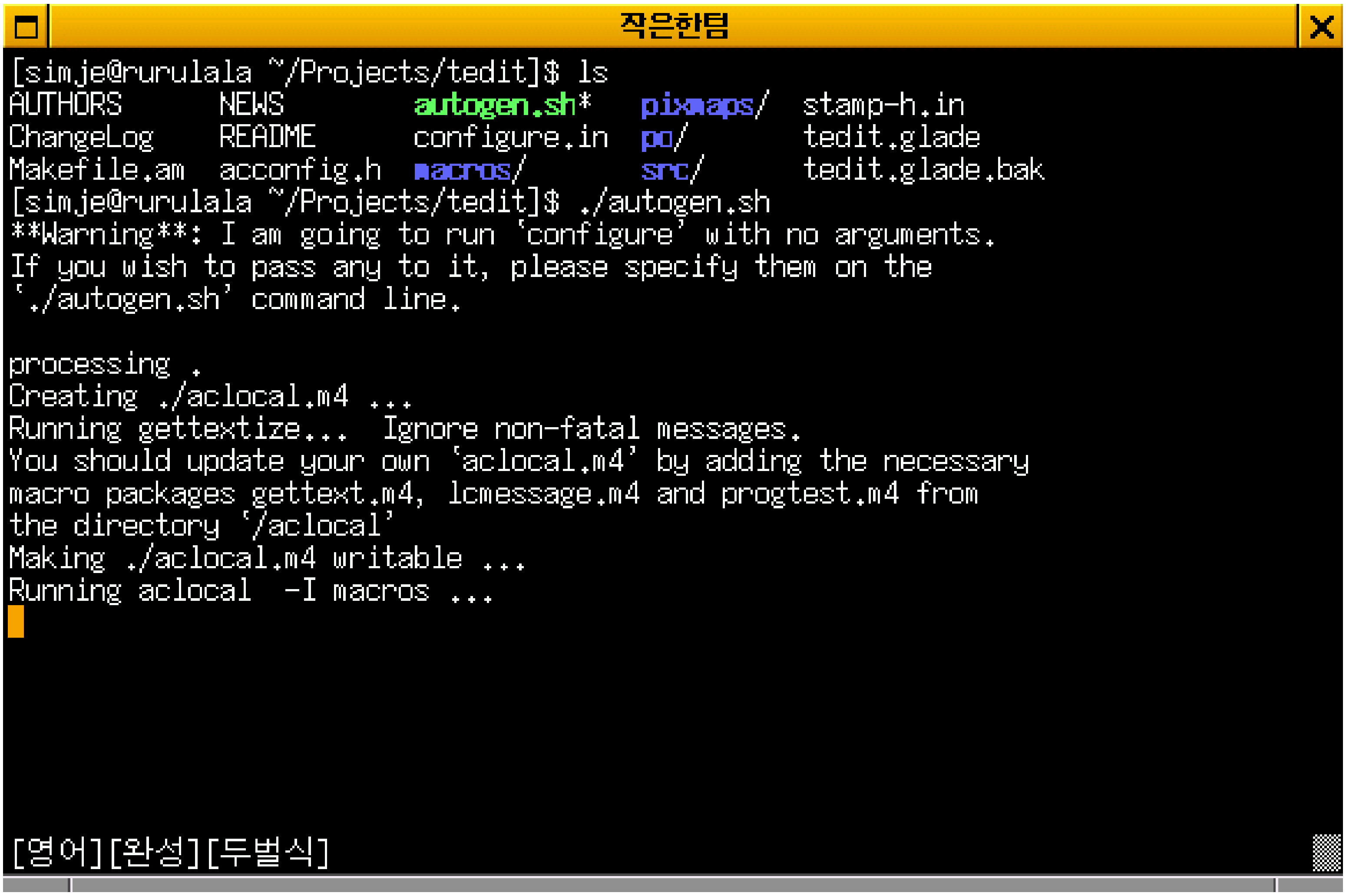
Language:

● C ○ C++ ○ Ada 95 ○ Perl ○ Eiffel

Gnome:

☐ Enable Gnome Support

그림 31: GLADE의 프로젝트 옵션 대화상자



```
[simje@rurulala ~/Projects/tedit]$ ls
AUTHORS      NEWS        autogen.sh*   pixmaps/  stamp-h.in
ChangeLog    README      configure.in  po/       tedit.glade
Makefile.am  acconfig.h  macros/       src/      tedit.glade.bak
[simje@rurulala ~/Projects/tedit]$ ./autogen.sh
**Warning**: I am going to run `configure' with no arguments.
If you wish to pass any to it, please specify them on the
`./autogen.sh' command line.

processing .
Creating ./aclocal.m4 ...
Running gettextize...  Ignore non-fatal messages.
You should update your own `aclocal.m4' by adding the necessary
macro packages gettext.m4, lcmessage.m4 and progtest.m4 from
the directory `/aclocal'
Making ./aclocal.m4 writable ...
Running aclocal  -I macros ...

[영어][완성][두벌식]
```

그림 32: `ls`명령과 `autogen.sh`실행 화면

그리고 위에서 본 *File* 메뉴의 *Build Source Code*을 선택하면 눈에 보이지 않지만 소스가 생성된다. 위의 프로젝트를 저장하는 곳에서 지정한 디렉토리에 가보면 `src`라는 디렉토리가 있는 데 여기에 모든 소스가 생성된다. 아쉽게도 바로 뭔가를 클리해서 생성된 소스를 가지고 실행 파일을 만드는 방법은 아직 없다. 그래서 어쩔수 없이 쉘을 통해서 실행 파일을 만들수밖에 없다. 프로젝트의 루트 디렉토리를 *ls*로 출력해보고 여기에 마련된 `autogen.sh`를 실행하면 환경 설정이 이루어진다.

그리고 나서 `make`을 실행하면 원하는 실행 파일을 얻을 수 있다. `make`의 결과는 `src`에 프로젝트 이름을 가진 실행 파일이 생기는 것이다. 이것을 실행한 결과는 다음과 같다.

바탕 윈도우 - 실행 후 죽이기

이제 종료해보자. 보통 하듯이 오른쪽 위에 있는 닫기 버튼을 눌러도 우리가 만든 어플리케이션은 죽지 않는다. 아뿔사. 우리는 불사신을 만들어낸 것이다. 이제 이 불사신을 죽여보도록 하자. 그럴러면 이전에 얘기한 2.1.2 (GTK 튜터리얼) 섹션에 있는 시그널을 알아야 하고 가장 기본적인 **delete_event**라는 시그널을 알아야 한다. 이것은 다른 것이 아니고 실행 파일의 *close* 메뉴[1]를 선택하거나 *title bar*에서 닫기를 선택하면 X 윈도우 관리자가 실행 파일에게 전하는 이벤트이다. 이것을 받아서 처리하려면 다음 그림과 같이 바탕 윈도우의 프로퍼티 윈도우의 여러 탭들 중에서 시그널 탭에서 시그널 핸들러 이름과 함께 등록해주어야 한다.

그러면 다음 그림과 같이 등록될 것이다.

---

[1] 이것은 창관리자가 붙인 메뉴를 말한다. 이것은 우리가 만든 어플리케이션의 메뉴와 다르다

avcd
asdfiwefkasdf
가나다라마바사

그림 33: 첫번째 실행 결과

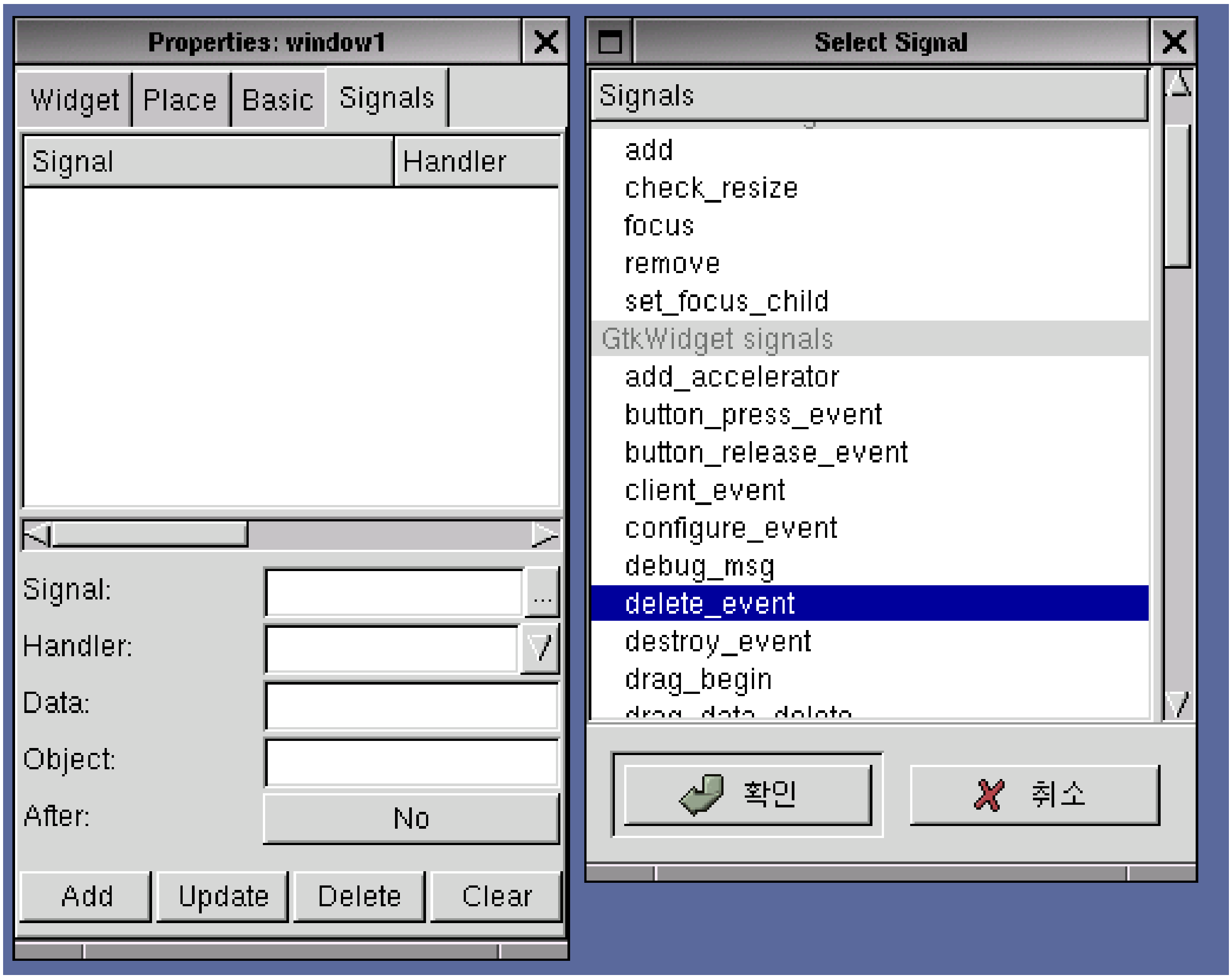
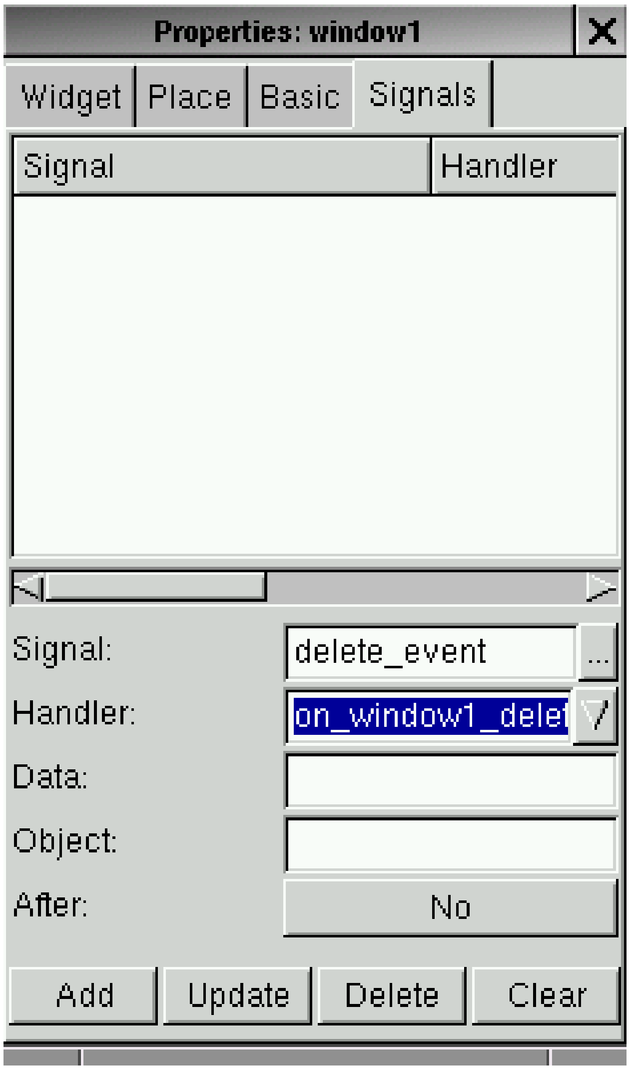


그림 34: 바탕 윈도우 window1의 delete_event



Signal: delete_event

Handler: on_window1_delet

Data:

Object:

After: No

그림 35: 등록 후의 모습

그림 36: GNOME의 About 대화상자 위젯

여기서 우리는 자동으로 시그널 핸들러 이름이 `on_window1_delete_event`가 되는 것을 볼 수 있다. 이것을 그대로 쓰자. 이것까지 그리고 생성될 내부 소스에 시그널과 시그널 핸들러를 연결시켜주는 것까지는 **GLADE**가 해주는 것이다. 자 이제 위의 스텝처럼 소스를 생성해보자. 그리고 이 시그널 핸들러 안에서 우리는 해야 할 일을 해보도록 하자. 다른 것은 아니고 다음과 같이 `src/callbacks.c`안에 있는 위의 `on_window1_delete_event` 함수 내부에 다음과 같이 입력하면 된다.

```
gboolean
on_window1_delete_event (GtkWidget *widget, GdkEvent *event,
                gpointer user_data)
{
        gtk_exit( 0 );
        return FALSE;
}
```

그리고서 `make`하고(다시 `autoget.sh`를 실행할 필요는 없다. 그러나 한번 소스를 재생성할 때마다 `make`시 모든 코드가 재컴파일될 것이다) 실행해보도록 하자. 이제 종료하면 제대로 종료될 것이다.

앞으로 우리는 GLADE 의 작업 대상이 되는 객체를 폼, 또는 템플릿이라고 부르자. 그리고 실제 실행된 후의 결과는 그 결과의 속성대로 부르기로 하자. 예를 들어서 *바탕 윈도우 폼* 에서 무엇 무엇을 작업하면 *바탕 윈도우*에 어떤 어떤 효과가 나타날 것이다라는 식으로 표현하기로 하자.

3.2 개발 두번째 단계 - about 대화상자

많은 **gnome** 프로그램들은 표준화된 룩앤필을 갖는다. 이 중에서도 메뉴, 툴바, 상태바, 그리고 about 대화상자가 두드러진다. 물론 다른 것도 있다. 예를 들면 색상표 대화 상자, 파일 선택기(윈도우즈의 File Open, Save, Save As와 유사) 등이 그것이다. 우리는 여기서 about 대화 상자를 만들고 이것을 넣어 보도록 하자.

about 대화상자 - 대화상자 만들기

대화 상자를 만들 때 **GTK+**가 제공하는 대화상자 위젯을 그대로 사용해서 그 위에 다른 여러 위젯들을 붙여서 만들어도 되고, **gnome**이 제공하는 다음과 같은 위젯을 사용하여 더 쉽게 만들 수도 있다. 우리는 이것을 택해보도록 하자.

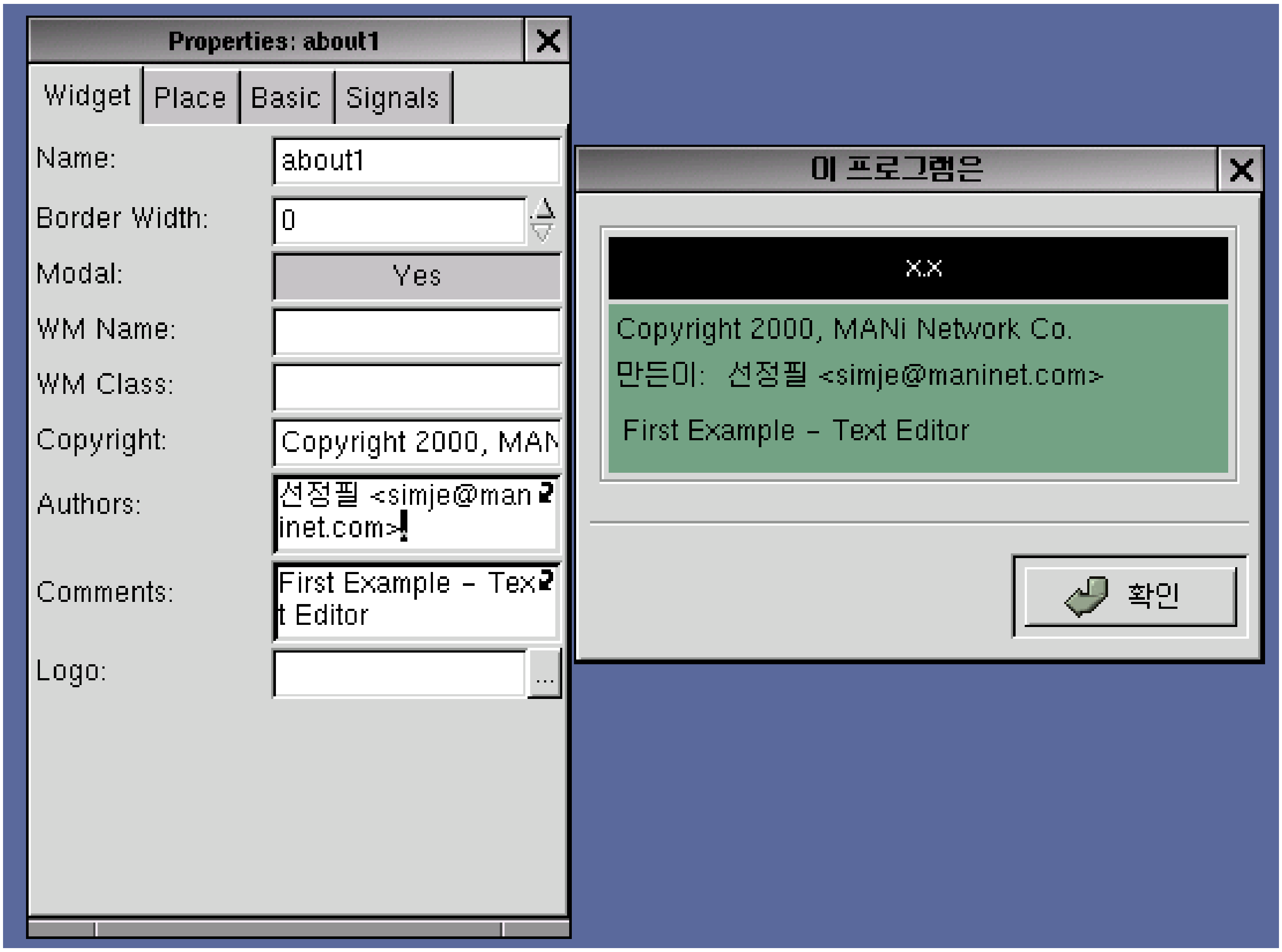
이것을 선택하면 다음고 같은 대화상자가 나타난다.

우리는 이 때, 같이 변한 프로퍼티 윈도우의 *Copyright*, *Author*, *Comments*, 그리고 *Logo*를 바꾸어서 이 about 대화상자를 단장시켜 보도록 하자. 먼저 *Copyright*와 *Author*, *Comments*는 다음 그림처럼 변경한다.

실제 똑같이 해보시면 알겠지만 이들 내용이 많아지면 대화상자의 전체 크기 등이 자동으로 늘어들었다 줄어들었다 한다.

그리고 마지막으로 *Logo*를 지정해보도록 하자. 이렇게 하려면 *Logo* 옆의 편집박스에 직접 패스를 입력해도 되고 그 옆에 있는 생략형을 클릭해서 파일 선택 대화상자를 불러 이것을 통해서 선택해도 된다.

그림 37: GNOME의 About 대화상자 초기 모습



Name: about1

Border Width: 0

Modal: Yes

WM Name:

WM Class:

Copyright: Copyright 2000, MAN

Authors: 선정필 <simje@maninet.com>

Comments: First Example - Text Editor

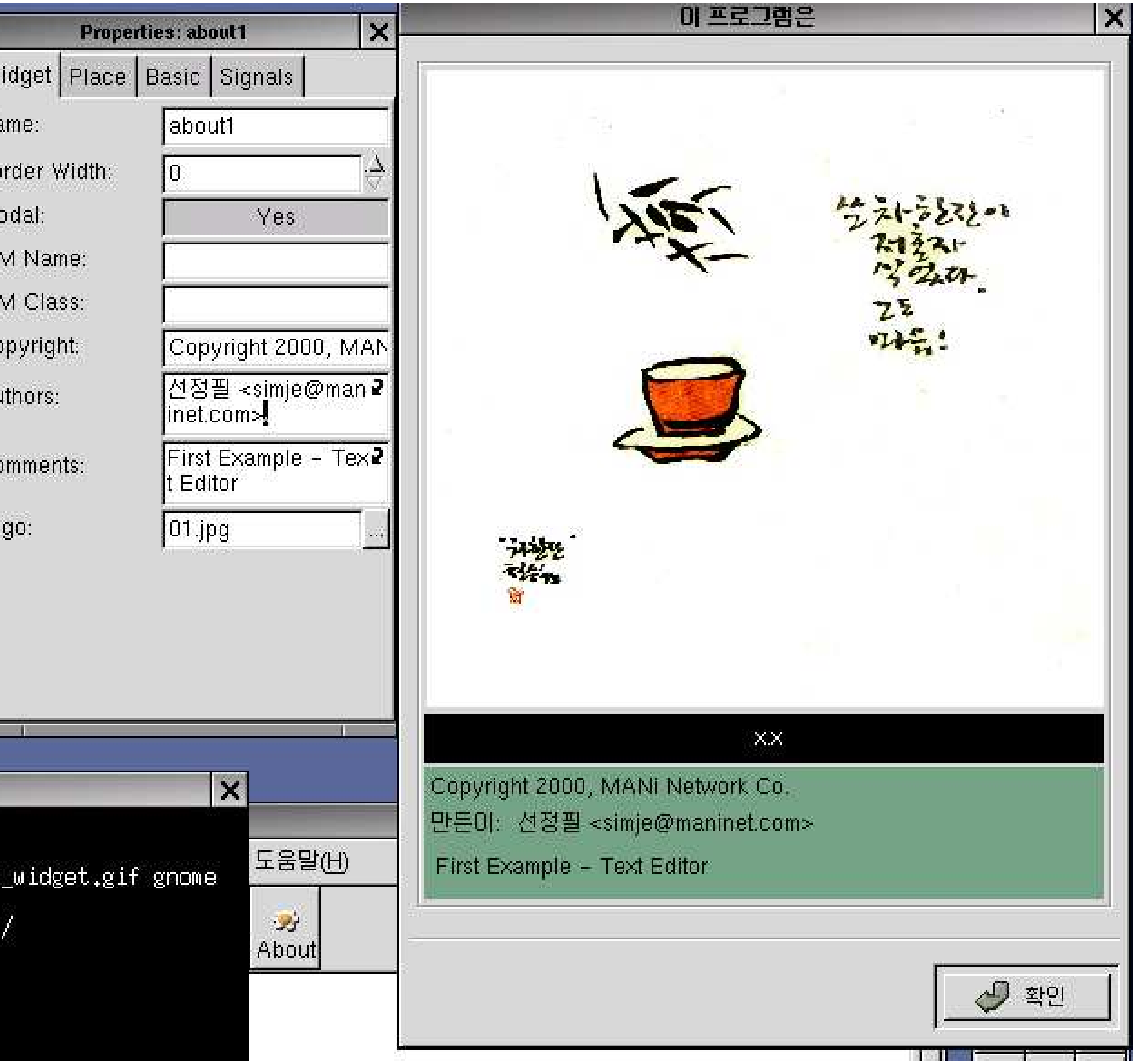
Logo:

Copyright 2000, MANi Network Co.

만든이: 선정필 <simje@maninet.com>

First Example - Text Editor

그림 38: GNOME의 About 대화상자 손질후의 모습



ame: about1
rder Width: 0
odal: Yes
M Name:
M Class:
opyright: Copyright 2000, MAN
uthors: 선정필 <simje@man inet.com>
omments: First Example - Tex t Editor
go: 01.jpg

```
_widget.gif gnome
/
```

이 프로그램은

X.X

Copyright 2000, MANi Network Co.
만든이: 선정필 <simje@maninet.com>
First Example - Text Editor

그림 39: GNOME의 About 대화상자에 로고를 넣은 모습

자 이제 프로젝트를 저장한다.

about 대화상자 - 대화상자 띄우기

프로젝트를 저장한 뒤 소스를 만들어 다시 `make`를 해도 우리는 이상한 결과를 만난다. 즉 바탕 윈도우와 조금 전에 만든 about 대화상자가 동시에 튀어 나온다는 것이다. 어떻게 하면 되겠는가? 어디엔가에서 우리가 원치 않는 이런 액션을 수행하는 코드가 있을 것이다. 다름 아닌 `main.c`의 `main()` 함수 내이다. 그렇다면 이곳을 보고 다음 세 라인을 없애 보자(지워도 되고 C comment처럼 막아도 된다.)

```
GtkWidget *about1;
.......
about1 = create_about1 ();
gtk_widget_show (about1);
```

원하는 대로 about 대화상자가 실행 파일을 실행하자 마자 뜨는 일은 없어졌다. 그래도 문제는 남는다. 어떻게 원할 경우 즉, 사용자가 메뉴나 툴바를 통해서 about를 선택할 경우 우리의 about 대화상자를 띄우는 방법은 무엇인가? 다음과 같은 코드를 비어 있는 메뉴 *About*에 대한 핸들러 `on_About_activate`(`callbacks.c`에 존재) 안에 넣어 주면 된다.

```
GtkWidget *about1;

about1 = create_about1_dialog ();
gtk_widget_show (about1);
```

위에서 지웠던 코드이다. 이것을 원하는 곳에 옮긴거나 다름없다. 자 프로젝트를 저장하고 `make`한 후 실행 파일을 만들어 확인해보자. About 버튼을 눌렀을 경우에도 이와 동일한 효과를 얻으려면 바탕 윈도우의 툴바에서 About 버튼을 선택한 뒤 프로퍼티 윈도우의 시그널 탭에서 위와 같이 시그널 핸들러를 등록하자(다음 그림 참조). 그리고 나서 이 핸들러 안에 메뉴와 동일한 코드를 넣어 주면 된다(아니면 공통 함수를 만들어 이를 호출해도 되고).

그런데 이 것이 제대로 실행된다 하더라도 이상한 점이 있다. 즉 우리가 넣었던 로고가 제대로 출력되지 않는다는 것이다. 왜 그럴까? **GLADE**는 만들어지는 어플리케이션의 pixmap 디렉토리의 디폴트를 "/usr/share/pixmaps" 아래 어플리케이션 이름으로 설정한다. 정확하게 말하면 `gnome-config --datadir`의 결과값을 얻어서[2] 여기에 "pixmaps" 와 어플리케이션 이름을 덧붙인 것이 로고 등 그림 파일이 저장되는 위치가 된다.

그러므로 사용자가 복사를 하든지 아니면 루트 권한으로서 `make install` 하면 로고 등 그림 파일들이 위의 디레토리로 복사한다. 그러면 여기에 있는 파일이 사용되어 About 대화상자의 그림이 보이게 될 것이다.

about 대화상자 - 대화상자 없애기

이렇게 해서 만들어진 about 대화상자는 사용자가 메뉴나 툴바를 통해서 띄울 수 있다. 그리고 이 대화상자의 확인 버튼을 누르면 언제든지 닫을 수 있다.

3.3 개발 세번째 단계 - file open 대화상자

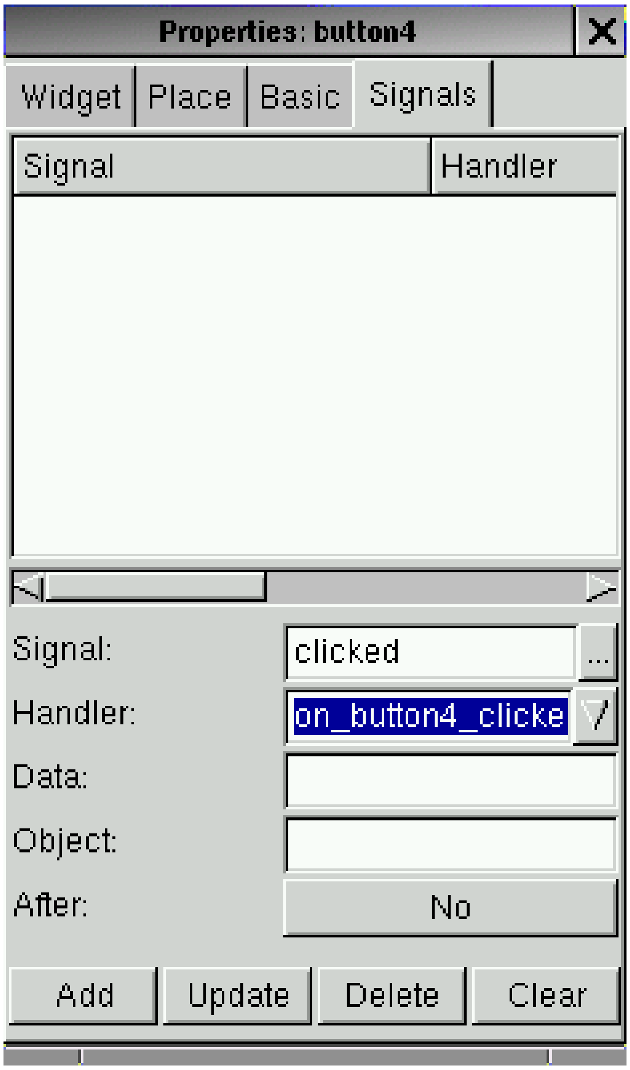
먼저 다음 41 (파일 선택 대화상자 위젯)을 "위젯 윈도우"-"GTK+ Basic 패널" 에서 클릭하면 프로젝트에 추가된다.

이렇게 하면 자동으로 "src/interface.c"에 `create_fileselection1` 라는 함수가 생기고 여기에 `ok_button1`와 `cancel_button1` 버튼들이 만들어져 붙여진다.

이 대화상자가 적절한 사용자 요구에 대해서 화면에 디스플레이되도록 설정하려면 다음과 같이 하자.

---

[2] 이것은 대개 "/usr/share" 이다



Properties: button4

Signal: clicked

Handler: on_button4_clicke

Data:

Object:

After: No

그림 40: About 버튼의 시그널

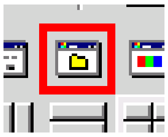


그림 41: 파일 선택 대화상자 컨틀롤

메뉴와 연결하기

바탕 윈도우 폼에서 메뉴를 선택하자. 그러면 프로퍼티 윈도우의 *Widget* 탭에 *Edit Menus...* 라는 버튼이 보일 것이다. 이것을 클릭하여 *Menu Editor* 윈도우를 부른 뒤 File 메뉴의 Open 서브 메뉴에 대한 핸들러 이름을 확인한다. 대개 *on_Open_activate* 일 것이다. 우리는 이 핸들러[3] 안에 file open 대화상자를 부르는 코드를 넣어야 한다. 다음과 같이 하자.

```
void
on_Open_activate  ( GtkMenuItem   *menuitem,
                    gpointer      user_data)
{
    /* 추 가  시 작 */
    GtkWidget *open_file;
    open_file = create_fileselection1();
    gtk_widget_show (open_file);
    /* 추 가  끝   */
}
```

툴바 버튼과 연결하기

메인 윈도우의 Open 버튼을 클릭하여 선택한 후 속성 윈도우의 Signals 탭에서 clicked 핸들러를 등록한다. 그리고 Build 버튼을 누른다. 그러면 callbacks.c 에 해당 핸들러 함수의 바디만 추가될 것이다. 대개 Open 버튼은 툴바에서 두번째이므로 on_button2_clicked()[4] 이 될 것이다. 여기에도 위의 항목과 같은 세 라인을 추가하자.

그러면 Open 버튼을 클릭할 때마다 다음과 같은 파일 선택 대화상자가 나올 것이다.

자 이제는 이 대화 상자와 인터페이스를 해야 한다. 사용자가 확인 버튼을 누르면 입력된 파일을 가져오고 대화상자를 종료하며, 취소 버튼을 누르면 그냥 대화상자를 종료하자.

자 이제 각 버튼 시그널 핸들러들을 등록해보자.

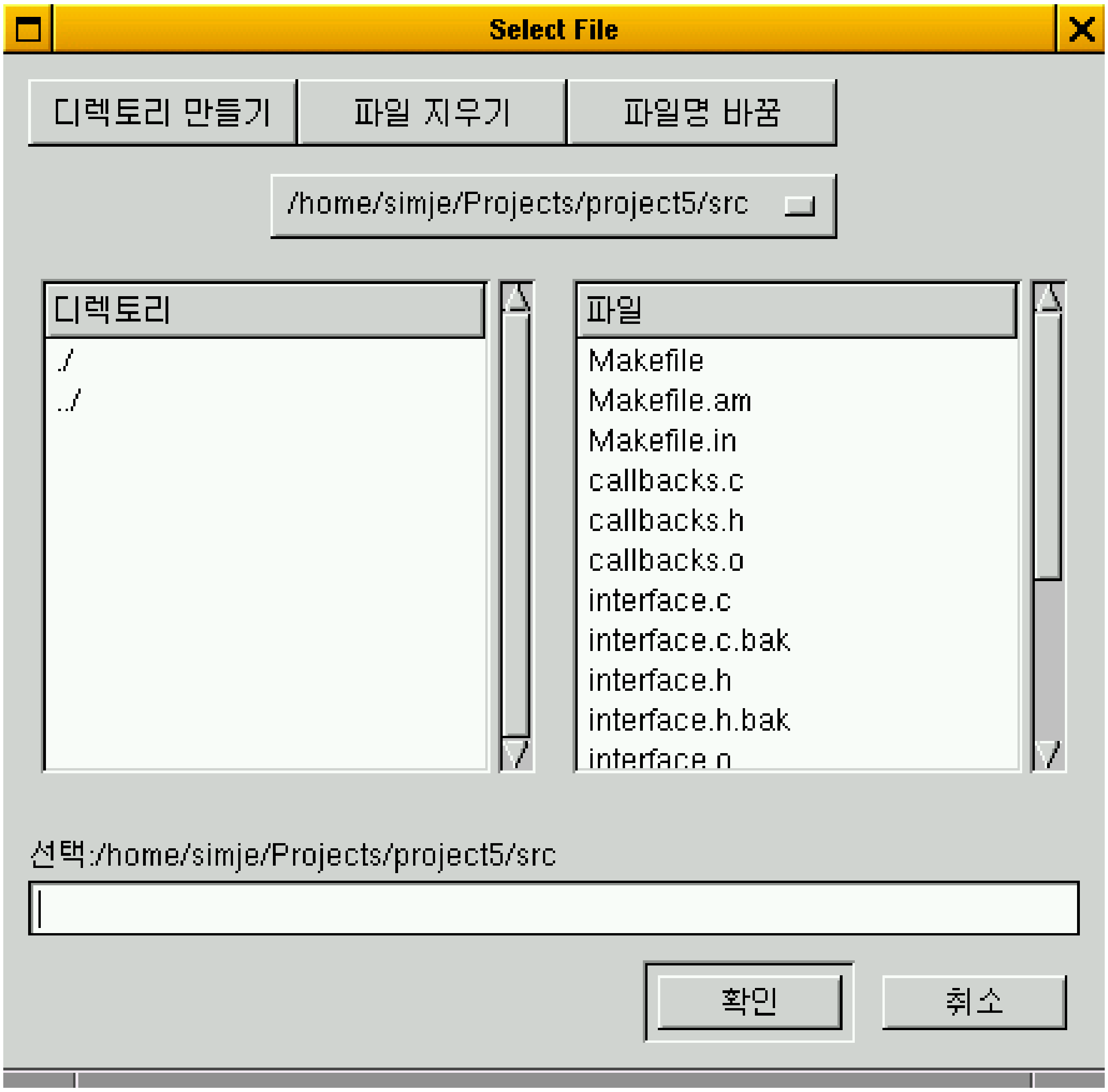
문제 및 사전 지식

1. 바탕 윈도우에서 새로운 대화상자를 위의 코드처럼 생성하였는 데, 그 결과 위젯에 대한 정보, 즉 위젯 포인터를 따로 저장하지 않았다. 이 것을 알아야 바탕 윈도우나 핸들러에서 대화상자를 닫거나 선택된 파일에 대한 정보를 얻어 올 수 있다. 결과적으로 대화상자에 대한 조작을 처리할 때, 전역변수 다른 방법을 통해서 대화상자 포인터를 알아내야 한다.
2. 선택된 파일은 `gtk_file_selection_get_filename (GTK_FILE_SELECTION(`*widget_pointer*`)))` 의 리턴값인 문자열이다.
3. gtk_signal_connect() 함수로 시그널과 시그널 핸들러를 연결할 때 마지막 파라미터로 `void *`를 말하는 gpointer 형 값을 줄 수 있고 받는 쪽, 즉 시그널 핸들러 쪽에서도 gpointer 형의 값을 받는다.
4. 시그널과 시그널 핸들러를 연결할 때 `gtk_signal_connect_object` 를 쓸 수도 있다. 이 함수는 다른 객체의 콜백 함수를 시그널 핸들러로 연결할 때 사용된다. 그리고 이 함수는 표준 함수(예: gtk_widget_show 등)를 시그널 핸들러로 연결할 때 유용하다. 이 함수의 네번째 파라미터는 연결되는 시그널 핸들러의 첫번째 파라미터로 전달된다.
5. GLADE 는 대화상자의 버튼에 대한 핸들러를 개발자가 추가할 때 Data 또는 Object 를 쓰지 않으면 gtk_signal_connect() 라는 함수만을 써서, 마지막 파라미터로는 무조건 NULL 을 주고서 등록한다. 그러나 Data 를 사용하면 마지막 파라미터로써 지정된 값을 사용하여 생성하며 Object

---

[3] callbacks.c 에 있다

[4] 이것도 callbacks.c 에 있다



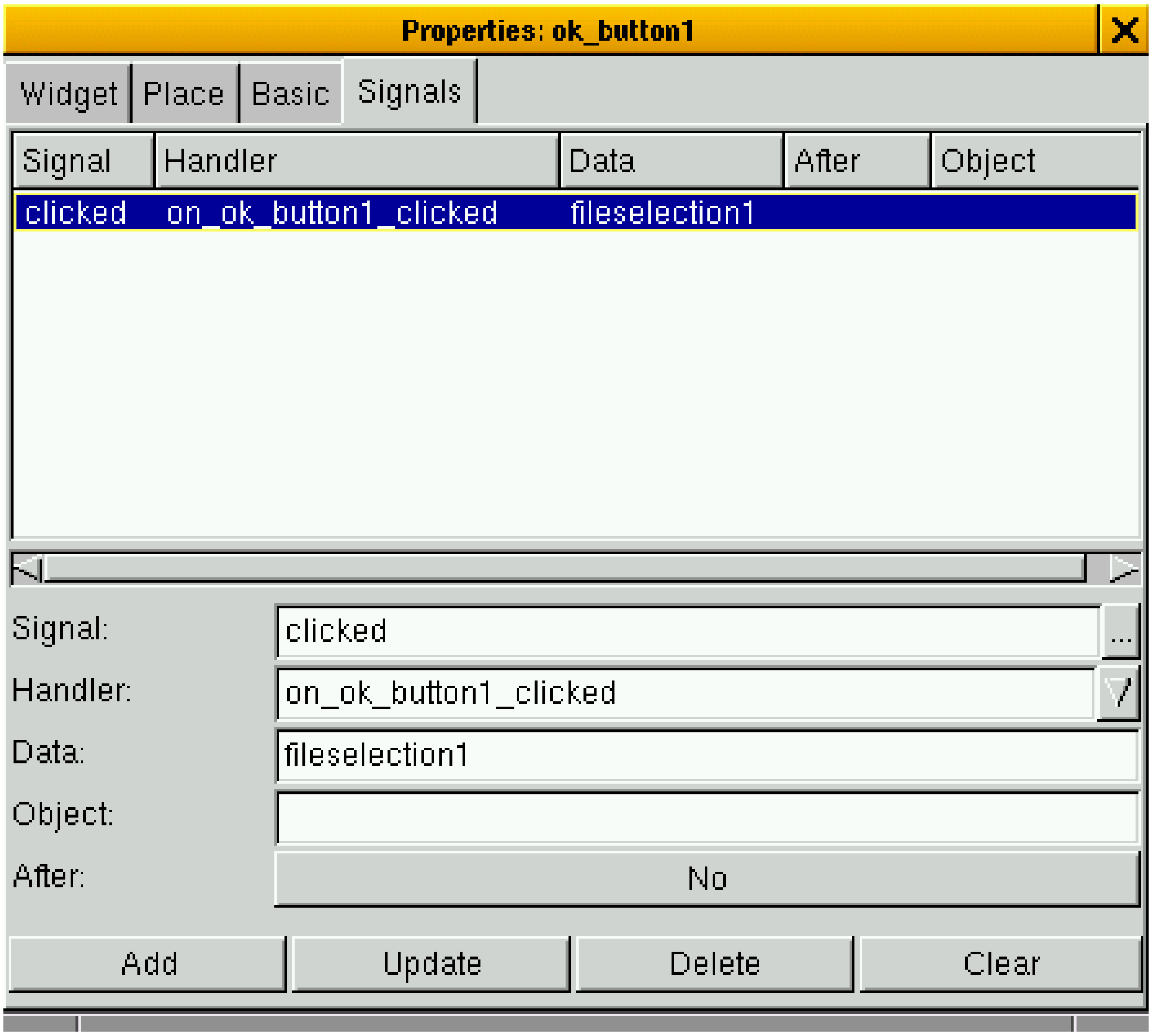
/home/simje/Projects/project5/src

./
../

Makefile
Makefile.am
Makefile.in
callbacks.c
callbacks.h
callbacks.o
interface.c
interface.c.bak
interface.h
interface.h.bak
interface.o

선택:/home/simje/Projects/project5/src

그림 42: 실행후 파일 선택 대화상자



Signal	Handler	Data	After	Object
clicked	on_ok_button1_clicked	fileselection1		

Signal: clicked
Handler: on_ok_button1_clicked
Data: fileselection1
Object:
After: No

그림 43: 확인 버튼 시그널 핸들러 등록

를 사용하면 gtk_signal_connect_object() 함수를 쓰며 마지막 파라미터로 Object 에서 지정한 것을 쓴다. 그리고 After 버튼을 클릭하면 gtk_signal_connect_after() 를 사용하여 시그널을 등록한다.

6. GLADE 는 취소 버튼을 디폴트로 만든다.

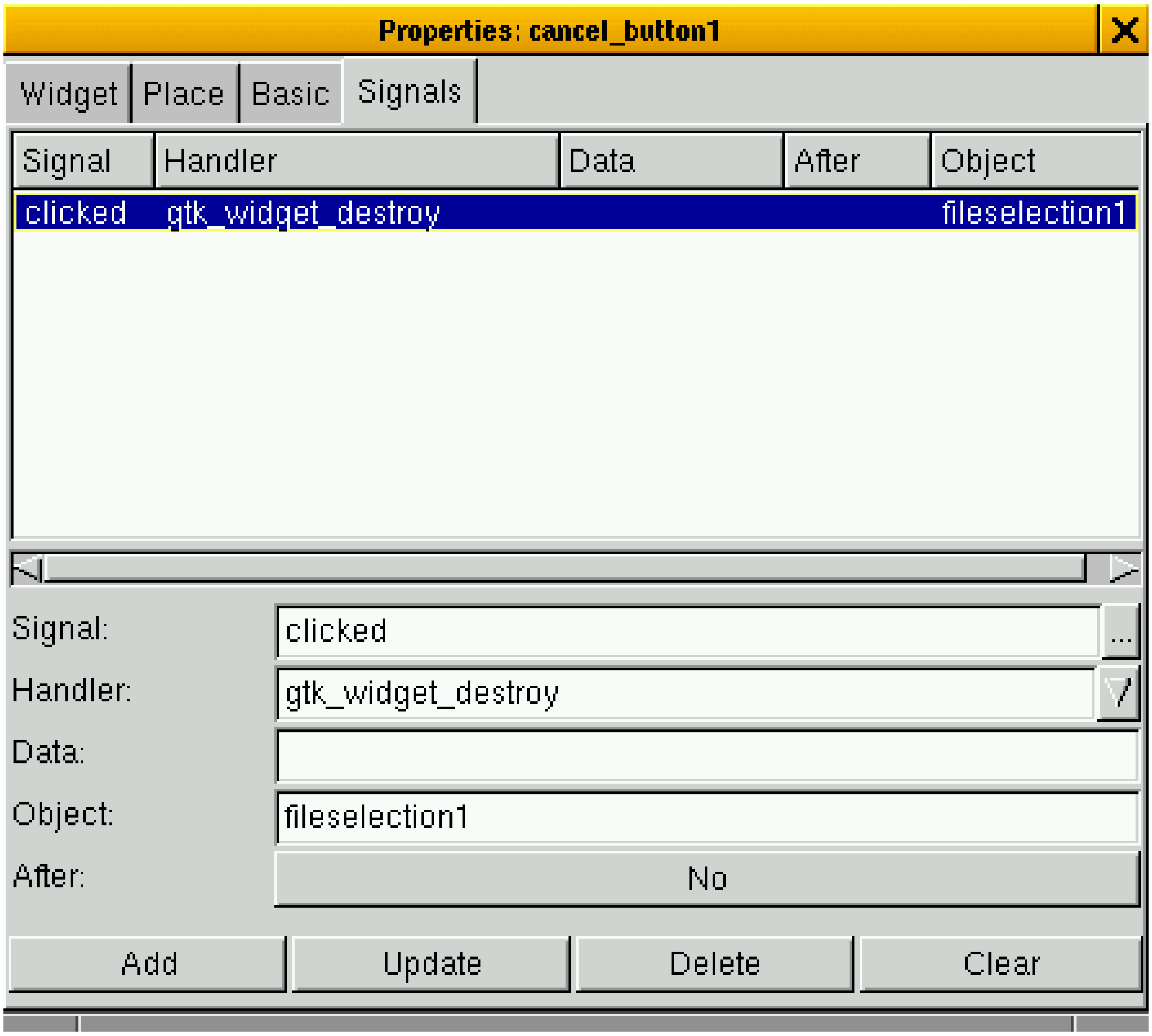
확인 버튼 시그널 핸들러

먼저 대화상자(여기서는 fileselection1) 템플릿이 화면에 떠있지 않으면 GLADE 메인 윈도우에서 해당 대화상자를 더블클릭하여 화면에 띄운다. 그리고 프로퍼티 윈도우를 띄운뒤, 대화상자의 확인 버튼을 누르면 이 버튼에 대한 프로퍼티를 보여줄 것이다. 이 프로퍼티 윈도우의 시그널 탭에서 다음 그림처럼 핸들러를 추가하자. 그리고 Data 부분에 fileselection1 을 넣도록 하자(다음 그림 참조). 그러면 gtk_signal_connect() 함수 네번째 파라미터에 이 값을 GLADE 가 넣을 것[5]이다.

이후 interface.c 에 다음과 같은 라인이 추가됨을 확인할 수 있다.

```
gtk_signal_connect (GTK_OBJECT (ok_button1), "clicked",
                    GTK_SIGNAL_FUNC(on_ok_button1_clicked),
                    fileselection1);
```

[5] 물론 GLADE 메인 윈도우에서 Build 를 클릭해야 한다



Signal	Handler	Data	After	Object
clicked	gtk_widget_destroy			fileselection1

Signal: clicked
Handler: gtk_widget_destroy
Data:
Object: fileselection1
After: No

그림 44: 취소 버튼에 대한 핸들러 등록

취소 버튼 시그널 핸들러

취소 버튼 시그널 핸들러는 확인 버튼 시그널 핸들러와 비슷하게 등록하되 프로퍼티 윈도우/시그널 탭의 시그널 핸들러 등록 콤보(다음 그림 참조)를 클릭하여 gtk_widget_destroy() 라는 스톡(표준, 내장) 핸들러를 선택하여 등록하자. 그리고 Object 부분에는 fileselection1 을 넣도록 하자(다음 그림 참조). 이렇게 하면 gtk_signal_connect() 함수 대신에 gtk_signal_connect_object() 라는 함수를 사용하며 이 함수의 네번째 파라미터에 NULL 아닌 조금 전에 넣어준 값을 GLADE 가[6] 넣는다.

위의 그림은 내용이 다 보이라고 디폴트로 나오는 것을 조금 늘린 것이다.

그러면 interface.c 에 다음과 같은 라인이 추가됨을 확인할 수 있다.

```
gtk_signal_connect_object (GTK_OBJECT (cancel_button1), "clicked",
                           GTK_SIGNAL_FUNC (gtk_widget_destroy),
                           fileselection1);
```

취소 버튼의 경우는 그대로 놔두어도 무방하지만 확인 버튼의 경우 시그널 핸들러에 조금 코드를 추가하여야 사용자가 무엇을 선택했는지를 파악할 수 있다. 다음과 같이 추가하면 된다:

---

[6] 물론 GLADE 메인 윈도우에서 Build 를 클릭해야 한다

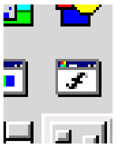


그림 45: 폰트 선택 대화상자 컨틀롤

```
void
on_ok_button1_clicked (GtkButton *button,
                       gpointer user_data )
{
    /* 추가 시작 */
    GtkWidget *open_file = (GtkWidget *)user_data;
    g_print ("%s\n", gtk_file_selection_get_filename (GTK_FILE_SELECTION
             (open_file)));
    gtk_widget_destroy( (GtkWidget *)open_file );
    /* 추가 끝   */
}
```

우선은 선택된 파일을 텍스트로 터미널에 출력하는 것으로 이 섹션의 얘기를 끝내도록 하자. 나중에 이 값을 활용하기로 하고.

3.4 개발 네번째 단계 - font selection 대화상자

먼저 다음 45 (폰트 선택 대화상자 위젯)을 "위젯 윈도우"-"GTK+ Basic 패널" 에서 클릭하면 프로젝트에 추가된다.

위에서 본 파일 선택 대화상자와 거의 모든 면에서 동일하기 때문에 많은 부분을 생략한다. 그러나 사용자가 확인이나 적용을 눌렀을 경우 사용자가 선택한 내용을 얻어오는 부분은 여기서 다루기로 한다. 이런 역할을 하는 함수는 파일선택과는 달리 1개가 아니라 3개정도 있다.

1. gtk_font_selection_dialog_get_font()
2. gtk_font_selection_dialog_get_font_name()
3. gtk_font_selection_dialog_get_preview_text()

세번째는 특수한 경우에(예를 들면 사용자가 그림에 넣을 텍스트를 선택하는 경우 등) 쓰일 것으로 보인다. 첫번째와 두번째는 함수 이름들이 가르켜주는 바와 같다. 다만 첫번째는 GdkFont 포인터를 리턴하고 두번째는 이름만 리턴한다.

확인 버튼에 대해서 위의 섹션에서 한 것과 비슷하게 하되 다음과 같이 선택 대상 정보를 얻는 부분을 추가하자.

```
void
on_ok_button2_clicked (GtkButton *button,
                               gpointer  user_data)
{
    /* 추가 시작 */
    GtkWidget *font_select = (GtkWidget *)user_data;
    GdkFont *gdk_font = NULL;
    gdk_font = gtk_font_selection_dialog_get_font(
```

```
                   GTK_FONT_SELECTION_DIALOG(font_select));
    g_print ("type : %s\n", gdk_font->type ==
             GDK_FONT_FONT ? "font" : ( gdk_font->type ==
             GDK_FONT_FONTSET ?  "fontset" : "unkonwn") );
    g_print ("ascent : %d\n", gdk_font->ascent );
    g_print ("descent : %d\n", gdk_font->descent);
    g_print ( "name = %s\n",
        gtk_font_selection_dialog_get_font_name(
            GTK_FONT_SELECTION_DIALOG(font_select)) );
    gtk_widget_destroy( (GtkWidget *)font_select );
    /* 추 가 끝  */
}
```

이 폰트 선택 대화상자를 띄우는 메뉴를 하나 추가하여 이 대화상자를 구동시켜보자(2.2.2 (메뉴 에디터)를 활용하자).

실행후 결과는 다음과 같다(오른쪽 터미널에 찍힌 내용을 주목하자).

3.5 개발 다섯번째 단계 - text area 제어

자 이제 텍스트 영역을 제어해보자. 기본적으로 텍스트 영역 위젯 자체가 텍스트 입력 기능, 입력된 텍스트의 선택 기능, 방향키 및 백스페이스 키 기능 등을 지원한다. 그리고 Del 키는 커서 뒤 문자 하나를 지우며 마우스 선택 후 Del 키는 선택된 영역을 모두 지운다. 이것은 GtkText 라는 위젯으로 지원된다. 그리고 이 위젯은 GtkEditable 위젯으로부터 승계된 것이다.

그러면 여기에 복사/자르기/붙이기 기능을 넣어보도록 하자.

복사

우리는 `void gtk_editable_copy_clipboard(GtkEditable *editable);` 함수를 사용하여 선택된 영역을 클립보드로 복사할 수 있다. 그래서 편집/복사 메뉴를 선택했을 때 구동되는 시그널 핸들러 on_Copy_activate() 안에 다음과 같은 내용을 넣어보자.

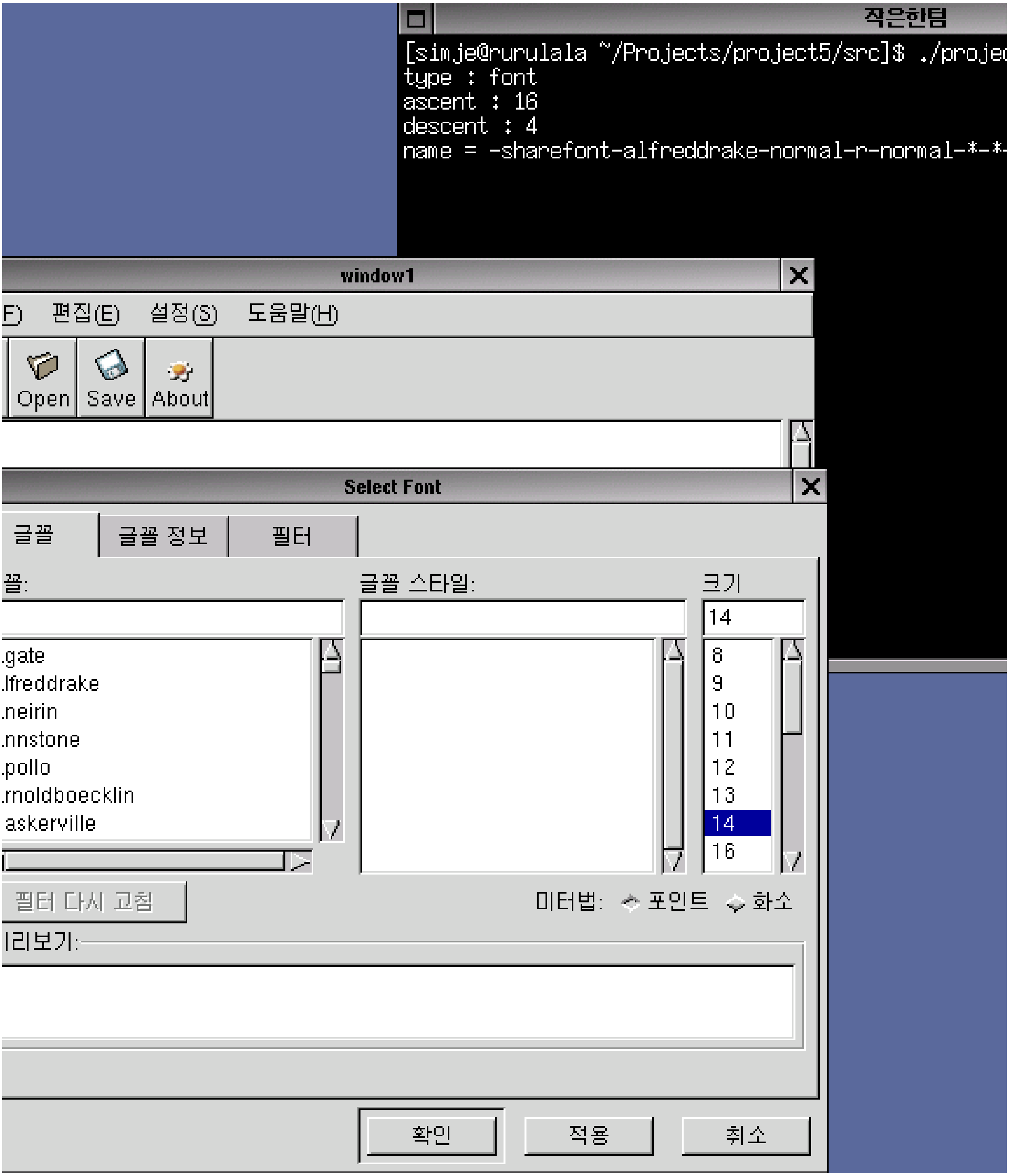
```
gtk_editable_copy_clipboard( (GtkEditable *)gTextArea );
```

그러나 먼저 GtkText 포인터형 전역 변수 gTextArea 가 선언되고 우리의 바탕 윈도우에 있는 텍스트 영역을 가리키도록 정의되어 있어야 한다. 그러므로 callbacks.c 에 `GtkText *gTextArea` 라고 전역 변수 선언을 한다. 그리고 callbacks.c 에 focus_in_event 시그널에 대한 핸들러를 만들고 이 안에 다음과 같은 코드를 추가하자[7]:

```
gboolean
on_text1_focus_in_event(GtkWidget *widget,
                         GdkEventFocus *event,
                         gpointer user_data)
{
```

---

[7] interface.c 에 위의 전역변수를 선언하거나 이 전역변수 초기화를 넣는다면 문제가 날 수 있다. 왜냐면 GLADE 는 interface.c 는 대개 함수 몸체 코드가 들어 있는 `create_`*xxx* 과 같은 함수들을 넣고 이는 개발자가 GLADE 메인 윈도우의 Build 를 누를 때마 새로이 갱신되어 개발자 커스터마이즈 코드가 유실되기 때문이다. 대신 callbacks.c 에 있는 내용들은 주로 시그널 핸들러들이며 interface.c 에 비해서 유실되는 것이 거의 없다. 그래서 추후 GLADE 의 Build 에 의해서 소스가 새로 갱신될 때, 영향을 받지 않는 것을 필요로 한다면 callbacks.c 에 넣는 것이 좋겠다. 그러나 쉽게 하는 방법인 전역 변수를 써야 한다다면 Build 하지 않는 방향으로 해도 좋다.



```
[simje@rurulala ~/Projects/project5/src]$ ./proje
type : font
ascent : 16
descent : 4
name = -sharefont-alfreddrake-normal-r-normal-*-*-
```

그림 46: 폰트 선택 대화상자 실행 화면

```
        /* 추가 시작 */
    g_print( "on_text1_focus_in_event() start\n" );
    if ( gTextArea == NULL )
        gTextArea = (GtkText *)user_data;
        /* 추가 끝   */
    return FALSE;
}
```

GTK 내부적으로 텍스트 영역의 선택 영역 복사에 대해서 Ctrl-C 를 단축키로 할당해 놓고 있다.

자르기

우리는 `void gtk_editable_cut_clipboard(GtkEditable *editable);` 함수를 사용해서 선택된 영역을 클립보드로 잘라 넣을 수 있다. 다음과 같이 편집/자르기 메뉴를 선택했을 구동되는 시그널 핸들러 on_Cut_activate() 안에 다음과 같은 내용을 넣어보자.

```
gtk_editable_cut_clipboard( (GtkEditable *)gTextArea );
```

GTK 내부적으로 텍스트 영역의 선택 영역 자르기에 대해서 Ctrl-X 를 단축키로 할당해 놓고 있다.

붙이기

우리는 `void gtk_editable_paste_clipboard(GtkEditable *editable);` 함수를 사용해서 클립보드에 있는 내용을 현재 커서 위치로 넣을 수 있다. 다음과 같이 편집/붙이기 메뉴를 선택했을 구동되는 시그널 핸들러 on_Paste_activate() 안에 다음과 같은 내용을 넣어보자.

```
gtk_editable_paste_clipboard( (GtkEditable *)gTextArea );
```

GTK 내부적으로 텍스트 영역의 선택 영역 붙이기에 대해서 Ctrl-P 를 단축키로 할당해 놓고 있다.

이제는 파일을 읽어서 텍스트 영역으로 읽어들이고, 파일에 저장해보도록 하자.

파일 읽기

위의 파일 선택 대화 상자 섹션에서 선택된 파일을 읽어서 우리의 어플리케이션 텍스트 영역에다 디스플레이해보자.

다음과 같이 파일 Open 대화상자를 구동한 후 사용자가 확인 버튼을 눌렀을 때 작동되는 시그널 핸들러 on_ok_button1_clicked() 에다 파일을 읽어서 텍스트 영역에 넣는 코드를 추가해보자.

```
void
on_ok_button1_clicked (GtkButton *button,
                       gpointer user_data )
{
    /* 추가 시작 */
    GtkWidget *open_file = (GtkWidget *)user_data;
    gchar *file_name =  gtk_file_selection_get_filename
        (GTK_FILE_SELECTION(open_file));
    FILE *infile = NULL;

    g_print ("%s\n", file_name );
    /* 텍스트 영역 락킹 */
    gtk_text_freeze (GTK_TEXT (text));
```

```
    infile = fopen(file_name, "r");
    if (infile) {
        char buffer[1024];
        int nchars;

        while (1) {
            nchars = fread(buffer, 1, 1024, infile);
            gtk_text_insert (GTK_TEXT (gTextArea), NULL, NULL,
                NULL, buffer, nchars);

            if (nchars < 1024)
                break;
        }
        fclose (infile);
     }

     /* 텍스트 영역 락킹 해제 */
     gtk_text_thaw (GTK_TEXT (text));

     gtk_widget_destroy( (GtkWidget *)open_file );
     /* 추가 끝   */
}
```

이 코드는 현재 삽입 포인트(current insert point) 뒤에다 텍스트를 추가하는 것이므로 파일 내용을 넣기전에 텍스트 영역에 어떤 내용이 있었다면 그 이후에 파일 내용이 들어갈 것이다.

위에서 새로 사용된 GTK 함수는 다음과 같다.

1. gtk_text_insert() - 텍스트 영역의 현재 커서 이후에 주어진 버퍼의 내용을 채운다.
2. gtk_text_freeze(), gtk_text_thaw() - 텍스트 영역을 락킹하고 언락킹한다.

파일 저장

이제는 현재 텍스트 영역의 내용을 파일로 써보자. 그럴려면 텍스트 영역 전체를 추출해야 한다. 텍스트 영역 일부를 추출하는 함수는 `gtk_editable_get_chars( GtkText *editable, gint start_pos, gint end_pos);` 이다. 여기서 start_pos, end_pos 는 0부터 시작하는 위치 정보이다. 그리고 보조 함수로써 텍스트 영역의 전체 길이를 구하는 함수 `gtk_text_get_length( GtkText *gtktext );` 가 있다. 이들을 이용해서 파일에 저장해보기로 하자.

먼저 Save 대화 상자를 위의 파일 선택 대화상자에서 했던 것처럼 추가하자. 그리고 사용자가 확인 버튼을 누르면 구동되는 시그널 핸들러인 on_ok_button3_clicked() 함수 안에 다음과 같은 코드를 넣어보자:

```
    GtkWidget *save_file = (GtkWidget *)user_data;
    gchar *file_name =  gtk_file_selection_get_filename
        (GTK_FILE_SELECTION(save_file));
    FILE *outfile = NULL;
    int ntotal = gtk_text_get_length( gTextArea );
    int nchars;
    gchar *buffer = NULL;
    gint nEnd = ntotal > 1024 ? 1024 : ntotal;
    gint nStart = 0;
```

```
g_print ("save file name : %s\n", file_name );
g_print ("ntotal =  %d\n", ntotal );

/* 텍스트 영역 락킹 */
gtk_text_freeze (GTK_TEXT (gTextArea));
outfile = fopen(file_name, "w+");
if (outfile) {
    buffer = gtk_editable_get_chars( (GtkEditable *)gTextArea,
            0, nEnd );
    nchars = fwrite(buffer, 1, nEnd, outfile);
    g_free( buffer );
    while (nEnd < ntotal) {
        nStart = nEnd;
        nEnd = (nEnd + 1024) < ntotal ? nEnd + 1024 : ntotal;
        buffer = gtk_editable_get_chars( (GtkEditable *)gTextArea,
            nStart, nEnd );
        g_print( "when iter : nStart=%d:nEnd=%d:buffer=%s\n",
                nStart, nEnd, buffer );
        nchars = fwrite(buffer, 1, nEnd-nStart, outfile);
        g_free( buffer );
    }
    fclose (outfile);
 }
 /* 텍스트 영역 락킹 해제 */
 gtk_text_thaw (GTK_TEXT (gTextArea));

 gtk_widget_destroy( (GtkWidget *)save_file );
```

여기서 한가지 주의할 것은 GTK 계열의 함수들이 한글을 모두 2-byte 문자가 아닌 1-byte 문자로 처리한다는 것이다. 반면에 fwrite 는 2-byte 문자를 2-byte 그대로 처리한다. 그래서 이런 오차로 한글을 파일로 저장하면 유실이 발생한다.

그리고 g_free 함수는 gtk_editable_get_chars() 함수를 호출하여 얻은 gchar 포인터에 대해서 호출하여 리소스를 반환하는 함수로써 반드시 이렇게 리소스를 반환하여야 한다고 한다(GTK 매뉴얼 참조).

마지막으로 폰트를 바꾸어 디스플레이 해보자.

폰트 변경 후 디스플레이

폰트를 변경해서 디스플레이할려면 먼저 폰트 선택 대화상자에서 확인을 눌렀을 경우 이에 대하여 구동되는 시그널 핸들러에서 시작해야 한다. 대략적으로 설명하면 이 핸들러에서 해야 할 일은, 전체 텍스트를 얻고 텍스트 영역을 모두 지운 다음 사용자가 선택한 폰트를 이용해서 gtk_text_insert() 함수를 호출하는 것이다. 이 함수의 폰트 파라미터에다 NULL을 주면 디폴트 폰트를 사용한다. 다음 소스를 시그널 핸들러(예: on_ok_button2_clicked())의 예의 소스(위의 섹션 참조)마지막에 추가하도록 하자:

```
    ntotal = gtk_text_get_length( (GtkText *)gTextArea );
/* 텍스트 획득 */
text_value = gtk_editable_get_chars(
        (GtkEditable *)gTextArea, 0, ntotal );
/* 텍스트 모두 지우기 */
gtk_editable_delete_text( (GtkEditable *)gTextArea, 0, ntotal );
```

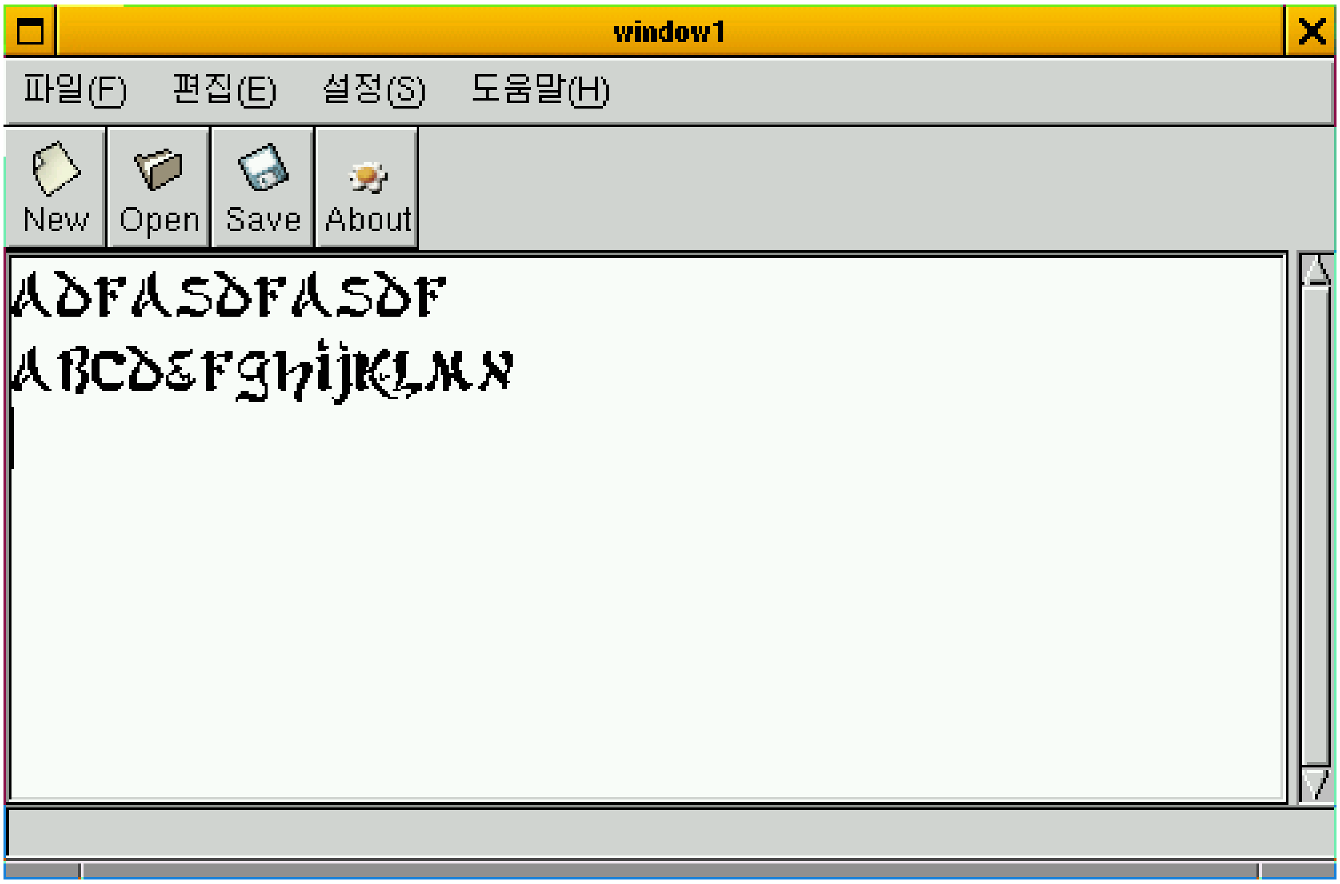


그림 47: 폰트 변경 후의 실행화면

```
        gtk_text_insert( (GtkText *)gTextArea, gdk_font, NULL, NULL,
                text_value, ntotal );
        g_free( text_value );
```

폰트 변경후의 실행화면은 다음과 같다.

이후에 입력되는 모든 텍스트가 지정된 폰트로 디스플레이된다. 단 한가지 문제는 정확한 threshold 값은 모르겠지만 커다란 폰트(예: 72pt)로 디스플레이가 안된다.

3.6 개발 여섯번째 단계 - 상태바 제어

상태바는 GLADE 로 붙이면 GtkStatusbar 라는 위젯이 사용된다. 여기에 메시지를 출력하려면 `guint gtk_statusbar_push(GtkStatusbar *statusbar, guint context_id, const gchar *text);` 라는 함수를 사용하면 된다. 예를 들어서 마우스가 툴바 버튼 중 Open 버튼 위에 있을 때 상태바에 ”파일 읽기” 라는 메시지를 출력해보자. 그리고 마우스가 이 버튼을 떠날 때 상태바에서 메시지를 지우도록 하자.

우선 해야 할일이 몇가지 있다. 첫번재 위의 push 함수의 첫번째 파라미터를 전역변수로 관리해야 한다. 그럴려면 다음과 같이 interface.c 에다 전역변수를 다음과 같이 선언하고,

```
GtkStatusBar *gStatusBar = NULL;
```

create_window1() 함수내에서 상태바 생성 후 생성된 것에다 이 전역변수를 정의하자[8].

```
gStatusBar = statusbar1;
```

두번째로 해야 할 일은 마우스 이동을 주시해서 Open 버튼 위로 왔을 때 발생되는 enter 시그널을 잡아 처리하는 시그널 핸들러를 등록해야 한다. 이것은 위에서 많이 해 보았으므로 생략한다. 그리고 마우스가 이 버튼을 떠날 때도 시그널을 잡아 시그널 핸들러를 등록해야 한다. 이것도 역시 생략한다.

두가지 일이 끝났다면 먼저 callbacks.c 에 위의 전역변수를 extern 으로 다음과 같이 선언하자.

```
extern GtkStatusbar gStatusBar;
```

그리고 마우스 enter 시그널 핸들러에다 다음과 같은 코드를 넣자.

```
gtk_statusbar_push( gStatusbar, 1, "파일 오픈" );
```

그리고 leave 시그널 핸들러에다 다음과 같은 코드를 넣자.

```
gtk_statusbar_pop( gStatusbar, 1 );
```

실행 후 상태바에 메시지가 뜨는 모습은 다음과 같다:

제 4 절 마무리

비록 단순하나마 우리는 **GLADE**를 가지고서 텍스트 에디터를 만들어 보았다. 어떤가? 윈도우즈의 비주얼 스튜디오만큼 쉽게 만들 수 있다. 그러나 다소 미진한 부분들이 있어서 여기에 그것을 정리하고자 한다.

4.1 GLADE에 어떤 기능이 추가되어야 할 것인가?

GLADE는 유기적인 생명체처럼 진화하고 있는 개체이다. 생명체이므로 계속되는 신진대사가 있어야 하는 것 아니겠는가? 나는 그런 의미에서 다음과 같이 **GLADE**가 가졌으면 하는 기능들을 뽑아 보았다. 이 문서를 보시는 분들 중에 **GLADE**에 이런 기능들 중의 일부를 제공할 수 있는 기회가 되시는 분들이 많았으면 한다.

데이터 베이스 연동 기능

이것은 리눅스가 상업적으로 활용되기 위해서 가장 필요한 기능이다. 다른 *RAD* 툴에서 제공하는 것처럼 *ODBC*, *JDBC* 등과 같은 연동 채널이 공급되어야 하며, 이를 바탕으로 한 다양한 데이터 베이스 핸들링 위젯들이 공급되어야 할 것이다.

그래픽 기능

동영상이나 정지 영상을 처리 및 보여주는 위젯이 추가되었으면 한다.

통신 및 인터넷 기능

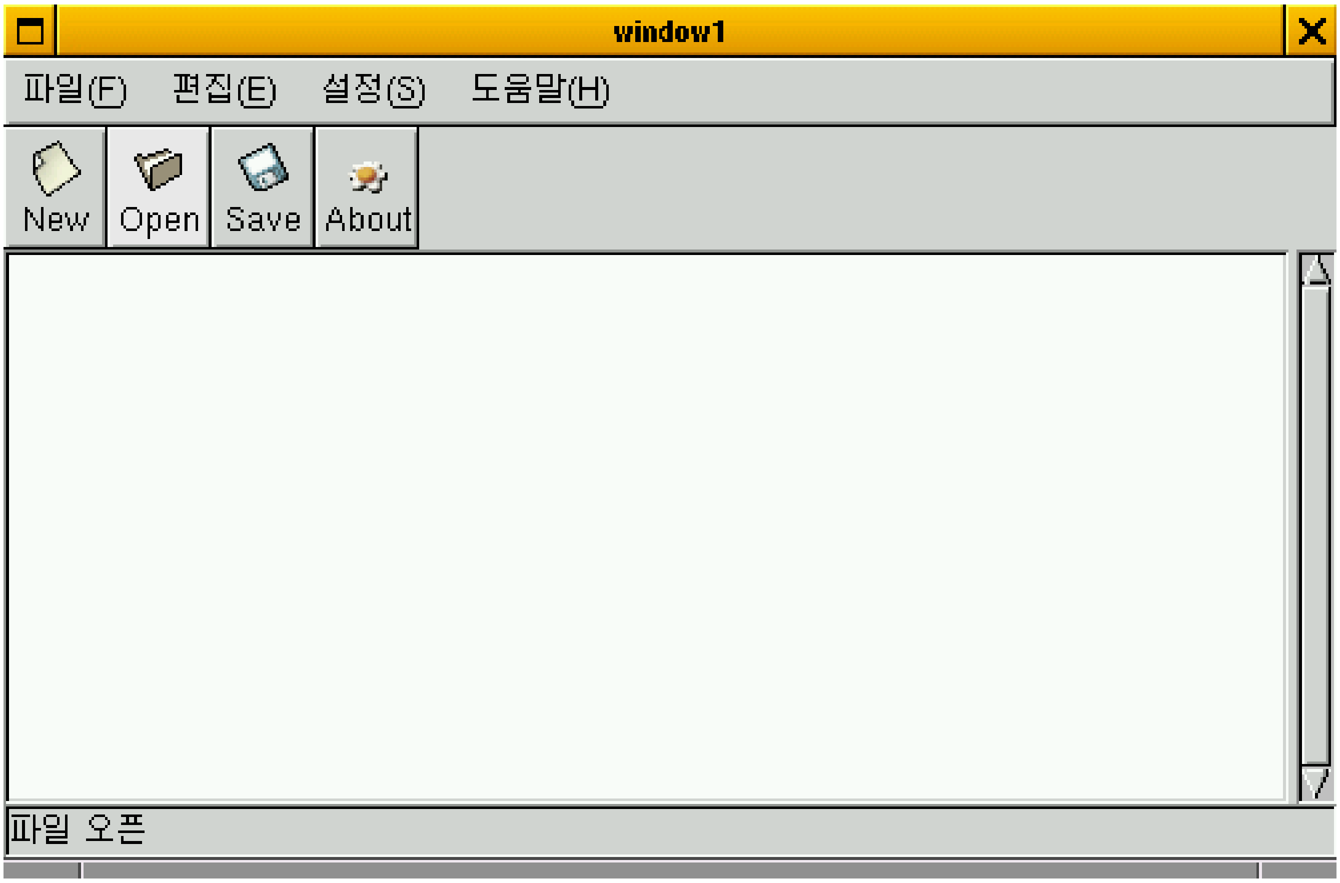
시리얼[9] 그리고 더 나아가 인터넷까지 이들을 처리하고 보여주는 위젯을 추가되었으면 한다.

논외의 얘기지만 GTK 의 아킬레스건은 아직 한글화라고 본다. 모든 GTK 라이브러리 함수들안에서 한글화가 조속히 이루어지기를 바라고 나도 거기에 동참할 수 있는 기회가 주어진다면 참 좋겠다.

---

[8] 솔직히 얘기하면 전역변수를 안쓰고 상태바의 포인터를 획드하는 방법을 찾아야 했으나 게을러서 나중으로 미룬다. 다른 GNOME 기반 소스들을 보면 대개 GLADE를 이용하지 않은 듯 보이고 이들은 전역변수가 아니면 gnome 상태바를 그냥 쓰는 경우가 많았다

[9] 물론 socket 은 지원한다



파일 오픈

그림 48: 실행후 상태바 모습