

QT Programming

만든이 : 서영진

valentis@chollian.net

<http://valentis.pe.kr>

생성일자 : 2002 년 1 월 25 일

(KLDP 제 1 회 공개 세미나 교재)

이 문서의 License

이 문서를 교육용으로 사용하는 것에 대해서는 어떠한 조건이나 사용에 제한이 없습니다. 이 문서의 라이선스는 저에게 있습니다. 상업적으로 사용하는 것에 대해서는 미리 연락주시기 바랍니다.

본 문서에서 잘못된 점이나 수정할 점이 있으시면 저에게 메일로 알려 주시면 다른 분들을 위해서 도움이 될 것 같습니다. 알려 주실때에는 정확한 내용(몇쪽 몇번째 줄에 어떤 문제가 있는지)을 알려주시면 감사하겠습니다.

QT Programming

1 장 QT 에 대해서

Qt 는 노르웨이의 Trolltech¹라는 회사에서 개발한 GUI Toolkit 이다.
KDE 의 기본 라이브러리로 유럽쪽에서 많은 개발자를 가지고 있다.
X window 용 Toolkit 의 하나로 화려한 인터페이스 및 객체지향으로 프로그램
의 개발을 쉽게한다.

기존의 X window 개발에 사용되던 모티브(Motif)는 상용이었기 때문에 자유소프트
웨어를 개발하는데는 적합하지 않았다.
그래서 Matthias Ettirch 가 KDE 를 개발하면 Qt 를 선택하게 되었다.
하지만 [기존의 Qt 라이선스에 약간의 문제](#)가 있었기 때문에 라이선스에 반대하는
사람들이 새로운 툴킷인 GTK 를 이용하여 Gnome 프로젝트를 시작되게 되었다.

QT 의 특징

Qt 가 다른 라이브러리와 구분되는 특징은 다음과 같다.

1. 멀티플랫폼 C++ GUI Toolkit
다른 라이브러리에서 프로그램을 개발하는 것과는 다르게 포팅이 자유
롭다. 단지 해당 플랫폼에서 재컴파일하는 것으로 유닉스 / Mac OS X 및
윈도우 환경에서 실행된다.

“Write once, compile anywhere”

Qt Application Source Code			
Qt API			
Qt/Windows	Qt/X11	Qt/Macintosh	Qt/Embedded
GDI	Xlib	Carbon	Embedded Linux
MS-Windows	Unix/Linux	Mac OS X	

Figure 1. Qt 의 구조

2. C++ 기반의 객체지향 클래스 라이브러리
Qt 의 모든 구현이 C++로 클래스로 캡슐화되어 있어 배우기 쉽고 사용
하기 쉽다. 그리고 상속을 이용해서 기능을 확장할 수 있다.

¹ Troll Tech 는 1994 년 Eirik Eng 와 Haavard Nord 에 의해 설립된 소프트웨어 회사로 노르웨이의 수도 오슬러에 위치하고 있다.

1995 년 Qt 를 발표하고 Qt 개발을 중심으로 관련된 라이브러리의 개발, Qt Embedded 를 통한 Embedded 쪽의 개발, 컨설팅과 소프트웨어 개발 하청들을 하고 있다. 코아팀에 의한 Qt 개발은 1992 년부터 이루어졌다고 한다.

3. 국제화 기능 지원(2.0 부터 I18n 기본 제공)
유니코드 지원, 각 나라의 언어로 작성된 메시지 파일만 있으면 해당 언어로 프로그램을 쉽게 사용할 수 있다.
Qt 3.0 부터는 메시지의 번역을 쉽게 하기 위하여 Qt Linguist 를 제공한다.
4. Signal/Slot 메카니즘(C++의 확장)
Qt 에서는 QObject 라고 불리는 클래스에 신호(Unix 에서 말하는 신호는 아니다.)를 내는 기구와 그 신호를 슬롯(멤버 함수)에서 받는 기구가 갖춰져 있다. 객체(Object)의 신호(Signal)를 다른 객체(Object) 슬롯(Slot)에 연결함으로써 프로그램 동작을 결정한다.
기존의 Motif 에서 사용하는 Callback 방식이 아닌 Signal/Slot Mechanism 을 사용한다. 기존의 Events² model 도 제공한다.
5. 표준 GUI 를 구성할 수 있는 풍부한 양의 Widget³을 제공한다.
 - 표준 User Interface(Push button, list, menu, check box..)는 모두 제공
 - 표준 Look 과 Feel 을 제공(Theme 기능 제공)

QT 의 라이선스

기존의 라이선스(1.44)에서는 GPL 을 따를 수 없었지만 2.0 버전부터 QPL 과 GPL 중에서 선택해서 사용할 수 있게 되었다.

2.xx 버전에서는 총 3 가지의 라이선스가 존재한다.(QPL, GPL, 그리고 Qt Commercial License Agreement)

윈도우와 맥용의 경우 자유판은 없고 상용판만이 존재한다. 물론 평가판 (Evaluation Version)은 존재한다.

Unix 의 경우 상용판(Professional Edition)과 무상판(Free Edition) 모두 존재한다. 자유소프트웨어로 개발할 경우 라이선스에 따라서 무상판으로 사용할 수 있다.

QT 의 도구들

Qt 를 설치하면 3 가지가 기본적으로 설치되고 tmake 는 Trolltech 웹사이트⁴에서 구할 수 있다.

1. moc(meta object compiler)
2. Qt designer
3. progen
4. tmake

여기서는 위에서 제공되는 유틸리티에 개략적인 설명을 하고 뒤쪽에서 자세한 설명을 하겠다.

- moc(Meta Object Compiler)
Qt 에서 사용하는 Signal 과 Slot 을 포함한 소스를 C++ 컴파일러에서 컴파일 할 수 있도록 일종의 Meta Object 코드를 생성한다.

² 마우스의 클릭이나 이동/키보드의 키 눌러짐과 같은 행동

³ Windows 에서는 Control 의 개념

⁴ <http://www.trolltech.com/developer/download/tmake.html>

- Qt designer
예전에는 GUI 를 구성하기 위해서 위젯을 모눈종이에 그려넣고 눈금의 위치를 이용하여 위젯의 위치와 크기를 계산했다.
이런 단순작업을 좀 더 쉽게 해주는 것이 Qt designer 이다.
2.xx 까지는 화면만 구성하고 프로그램 코드 사용자가 텍스트 에디터를 이용해서 입력 했었지만 3.0 이상부터는 프로젝트 관리 기능을 제공하고 프로그램의 코드도 직접 입력할 수 있다.

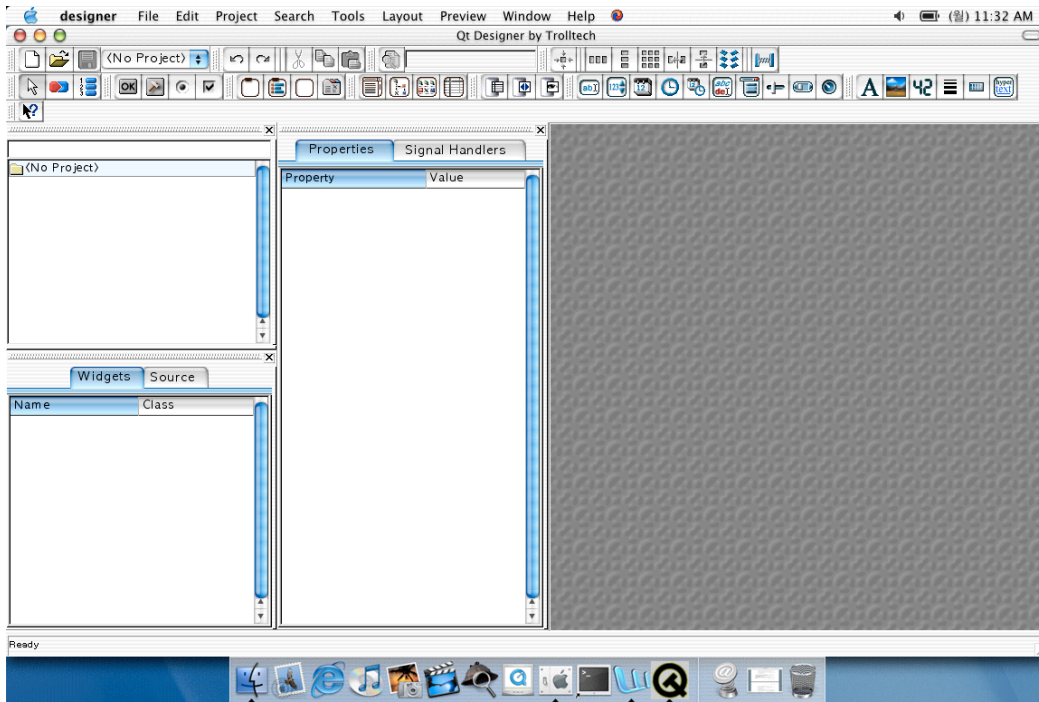


Figure 2. Mac OS X 에서의 Qt designer

- progen
tmake 를 이용하여 makefile 을 만들때 사용하는 프로젝트 파일(.pro) 을 자동 생성하는 유틸리티이다.
- tmake
tmake 는 .pro 파일을 가지고 make 파일에서 사용가능한 makefile 을 만드는 유틸리티이다.

Qt 3.0 에서의 새로운 도구들

Qt 3.0 에는 기존에 있었던 도구들과 함께 두가지의 도구들이 추가되었다.

다국어를 지원하기 위해 도입된 Qt Liguist 와 예전에 예제로 들어있었던 도움말 보기 기능(helpviewer)을 강화시킨 Qt Assistant 가 그것이다.

- Qt Linguist
예전에 다국어 지원을 하기 위해서는 손으로 직접 .ts 파일을 수정했

있다. 이제는 그런 작업을 더욱 쉽게 Qt Linguist 를 이용해서 할 수 있다.

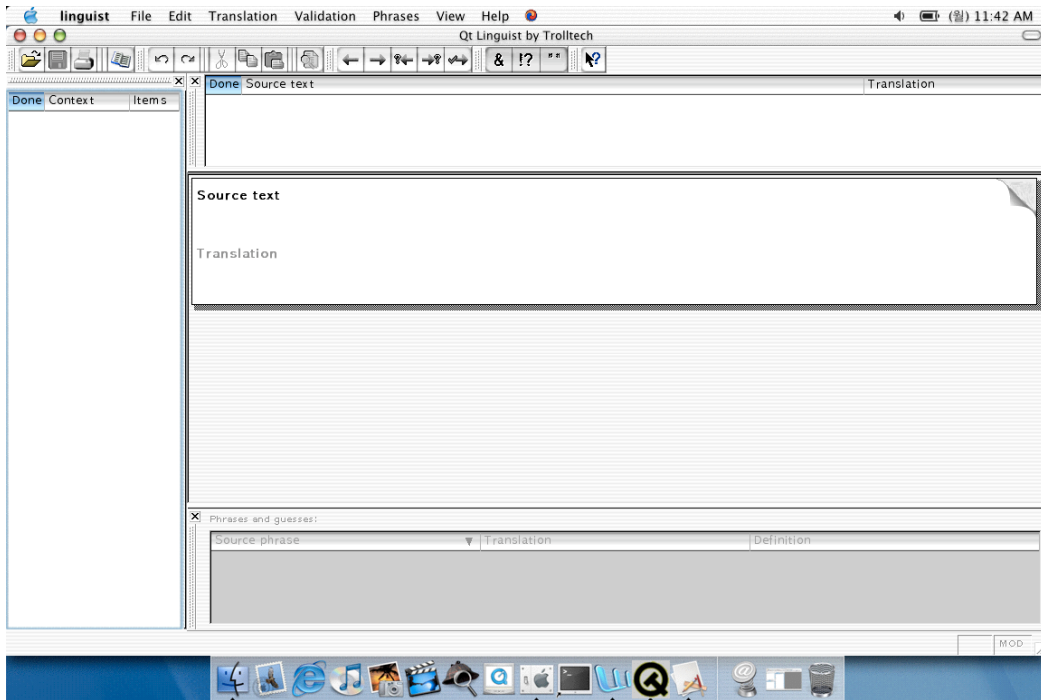


Figure 3. Qt Linguist

- Qt Assistant
MS Windows 에 있는 VisualStudio 의 MSDN 과 같은 기능으로 도움말
말을 찾을 수 있는 기능이다.

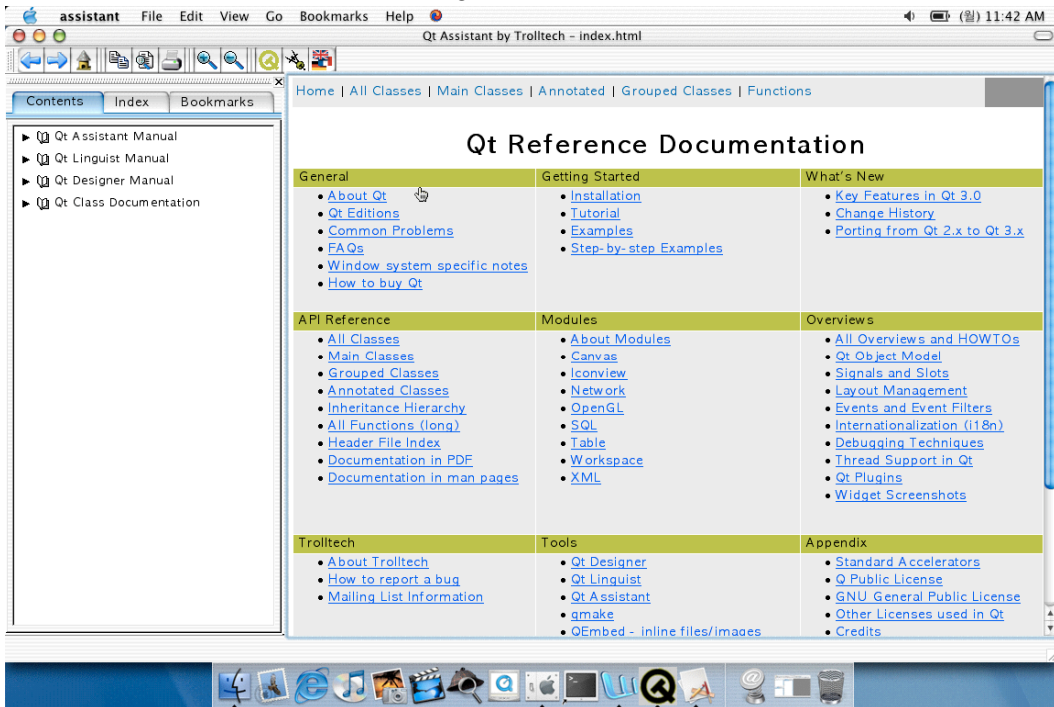


Figure 4. Qt Assistant

예전 2.0 대에서는 \$QTDIR/doc/example 을 컴파일하면 helpviewer 라는 프로그램이 들어있었다. 하지만 검색기능은 없었고 단지 html 문서만 보여주는 기능이었다.

Qt Assistant 를 이용하면 원하는 도움말들을 쉽게 찾을 수 있고 인덱스를 이용해서 보다 편하게 도움을 얻을 수 있다.

기타 Qt 용 프로그램을 개발할 수 있는 프로그램들은 다음과 같은 것들이 있다.

- Qt Architect
- QtEz
- KDE Studio
- Kdevelop

QT 의 설치

리눅스에서의 설치

Qt 의 설치는 <http://www.trolltech.com>에서 Qt 파일을 받아서 설치하는 것으로 시작한다. Redhat 과 같은 배포판에는 기본적으로 Qt 가 설치되지만 설치하지 않았다는 가정에서 설치를 시작한다.

Trolltech 의 홈페이지에 가면

Qt-x11-free-3.0.1.tar.gz(2002 년 1 월 21 일 현재)

을 다운 받을 수 있다.

보통의 경우 tar ball 파일로 설치하면 되고 Redhat 과 같은 배포판의 경우 rpm 파일을 받아서 설치할수도 있다.

먼저 설치를 원하는 디렉토리로 이동(보통의 경우 /usr/lib/qt 에 설치된다.)한 후 위의 파일을

```
$ tar zxvf Qt-x11-free-3.0.1.tar.gz
```

같이 하면 압축을 풀 수 있다.

압축을 해제한 후 압축이 해제된 디렉토리로 이동해서

```
# ./configure  
# make  
# make install
```

을 하게되면 설치가 된다.

설치가 끝난 후에 .bashrc(또는 .cshrc)에 <설치된 경로>/bin 을 등록해 두어야 moc 나 Qt designer 를 사용할 수 있다.

설치가 끝난 후에는 설치가 된 라이브러리들을 시스템에 등록해야 한다.

/etc/ld.so.conf 에 다음 라인을 추가한다.

```
/usr/lib/qt
```

그리고

```
# ldconfig
```

를 실행하면 모든 설치가 끝나게 된다..

OS X 에서의 설치⁵

기본적인 다운로드 방법은 위와 같다.(파일명은 QtMacEval.tgz 로 되어 있다.)

터미널을 실행시켜 설치를 원하는 디렉토리로 이동해서 파일을 풀고 난 후 다음과 같은 작업이 필요하다.

우선 자신이 사용하는 Shell 에 다음과 같은 환경을 추가한다.

```
QMAKESPEC : qmake 를 사용할 경우 compiler 를 설정
QTDIR : Qt 가 설치되어 있는 곳을 정의
PATH : moc 와 같은 Qt 에서 제공하는 유틸리티들에 대한 경로(Path)
DYLD_LIBRARY_PATH : Qt 라이브러리에 대한 경로
```

환경설정은 다음과 같이 하면 된다.

bash, ksh, zsh 또는 sh 을 사용하는 경우, 홈디렉토리의 ~/.profile 에 다음 라인들을 추가한다.

```
QTDIR=<Qt 가 설치된 곳>/QtMacEval
QMAKESPEC=macx-g++
PATH=$QTDIR/bin:$PATH
DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:$QTDIR/lib

Export QTDIR PATH DYLD_LIBRARY_PATH QMAKESPEC
```

Csh 이나 tcsh 를 사용하는 경우, 홈디렉토리의 ~/.login 에 다음과 같은 라인들을 추가한다.

```
setenv QTDIR <Qt 가 설치된 곳>/QtMacEval
setenv QMAKESPEC macx-g++
setenv PATH $QTDIR/bin:$PATH
setenv DYLD_LIBRARY_PATH $DYLD_LIBRARY_PATH:$QTDIR/lib
```

위와 같이 환경변수를 추가한 후 시스템에 변경된 내용(환경)을 적용한다.

```
$ source .login(또는 .profile)
```

이제 마지막으로 설치된 라이브러리를 /usr/lib 밑으로 링크를 시켜야 한다.

링크를 시키기 위해서는 슈퍼유저(root) 권한이 필요하다.
 하지만 Mac OS X 에서 슈퍼유저 권한을 사용하기 위해서는 슈퍼유저를 활성화 되어야 한다.
 슈퍼유저를 활성화 시키기 위해서 NetInfo Manager⁶을 실행시킨다.
 (NetInfo Manager 말고도 슈퍼유저를 활성화 시키는 방법은 있지만 이곳에서는 NetInfo Manager 를 사용하도록 한다. 자세한 것은 다른 문서들을 참조하라.)

⁵ Mac 용 버전에 대한 평가판은 <http://www.trolltech.com/developer/download/maclicense.html>에서 다운 받을 수 있다.

⁶ (Mac OS X 이 설치된 하드 Volume) > Applications > Utilities > NetInfo Manager

도메인 > 보안 > 루트 사용자 활성화
 를 선택한 후 루트사용자를 활성화 시킨다.
 도메인 > 보안 > 루트 암호 변경...
 으로 루트 암호를 넣어주면 슈퍼유저 설정에 대한 것은 끝나게 된다.

```
# ln -s $QTDIR/lib/libqt-mt.dylib.3.0.1 /usr/lib
# ln -s $QTDIR/lib/libqui.dylib.1.0.0 /usr/lib8
# ln -s $QTDIR/lib/libeditor.dylib.1.0.0 /usr/lib
```

2 장 QT 살펴보기

앞 장에서는 Qt 에 대한 개요 및 설치방법에 대해서 살펴보았다.
 이번 장에서는 프로그램의 제작시 필요한 Widget, Signals & Slots 과 기타 Qt 프로그래밍에 필요한 제반 사항에 대해서 살펴보고자 한다.

QT 의 Widget⁹

Qt 에서는 풍부한 양의 위젯(버튼, 스크롤바.....)을 제공한다. Qt 의 Widget 은 유연하고 OOP(Object Oriented Programming)를 지원해서 확장과 사용이 쉽다.

Widget 은 사용자의 화면을 구성하는 보이는(Visual) 요소(Element)이다.
 위젯의 예로는 Buttons, menus, scroll bars, message boxes, application windows 과 같은 것을 들 수 있다.
 Qt 에서 위젯들을 control 과 container 임의로 구분하지 않고 양쪽으로 모두 사용할 수 있다.
 상속을 통해서 사용자 자신의 위젯을 쉽게 생성할 수 있다.

위젯은 QWidget 의 서브클래스로 기본적인 구조는 다음과 같다.

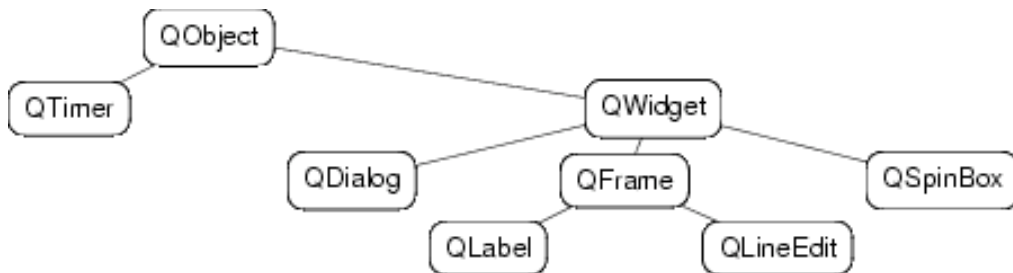


Figure 5. QWidget Class 구조

⁷ \$su - 나 \$su root 한 후 root 로 설정한 암호를 넣으면 된다.
⁸ 아래의 두개는 Qt designer 를 사용하기 위해서 필요하다.
⁹ 윈도우(Window)와 가젯(gadget)이라는 단어가 합쳐진 “위젯(widget)”은 윈도우의 control 가 같은 개념이다. GUI 에서 보이는 사용자 인터페이스도 위젯이고 인터페이스를 구성하는 작은 Element 들도 하나의 위젯이다. Gadget 이란 윈도우를 가지지 않는 widget 을 의미한다.

위젯은 자식 위젯을 가질 수 있다. 자식 위젯은 어미 위젯의 공간속에서 보여진다. 어미 위젯이 없는 위젯을 top-level 위젯이라고 부른다. Qt에서는 모든 위젯이 Top-level 위젯이 될 수 있으며 어떠한 제약도 없다. 어미 위젯에서의 자식 위젯의 위치는 Layout Manager 에 의해서 자동적으로 배치된다. 어미 위젯이 삭제되거나 보여지거나/감춰지거나 하면 자동적으로 자식 위젯도 같은 행동을 하게된다.

QT 의 Slots 와 Signals

GUI 프로그램들은 사용자의 요구(Action)에 반응하여 응답한다. 예를 들면 사용자가 메뉴의 아이템을 클릭하면 프로그램은 프로그램의 코드를 실행한다.

Qt 에서 고수준의 이벤트¹⁰에는 ‘Signals/Slots’, 저수준의 이벤트¹¹에는 가상함수(virtual function)을 이용하여 사용자의 이벤트를 처리한다.

- Signal : 사용자 객체(위젯)에서 어떠한 행위(예 : 마우스를 클릭)를 했을 때 발생하는 신호
- Slot : Signal 이 발생했을 때의 처리방식을 정의

Signals 와 Slots 의 예를 들면 Button Widget 이 있을 때 버튼을 클릭하는 행위는 Signal 이 되고 버튼을 클릭할 때 프로그램을 종료하는 행위는 Slot 이 된다.

Signal 과 Slot 을 하나로 묶기 위해서는 connect()함수를 사용한다. 마찬가지로 묶여있는 Signal 과 Slot 을 분리시키기 위해서는 disconnect()함수를 사용하면 된다.



```
QObject::connect(PushButton, SIGNAL( clicked() ), SLOT( show() ) );
```

Figure 6. Signals & Slots

¹⁰ 사용자가 정확하게(자의적으로) 입력하는 행위로 마우스버튼의 클릭, 키보드 입력, 텍스트 입력 같은 것이 있다.

¹¹ 사용자의 의도가 아닌 마우스가 어느 위젯에 들어간단든지 / 나오는 것과 같은 행위

사용자가 직접 Signal 과 Slot 을 정의하고 사용하기 위해서는 moc(Meta Object Compiler)로 컴파일 하는 절차가 추가로 필요하다.

moc 는 Signal 과 Slot 이 있는 클래스의 선언(보통 헤더파일을 이용)으로 부터 C++ 컴파일러가 사용할 수 있는 소스파일을 자동으로 생성해 준다.

자세한 사용은 나중에 예를 통해 보도록 하겠다.

3 장 QT Programming by Example

앞 장에서는 프로그램의 제작시 필요한 Widget 과 Signals/Slots 에 대해서 살펴 보았다.

이번 장에서는 Qt 로 시작하는 첫 프로그래밍으로 Qt 를 사용한 첫 프로그램 “Hello Qt!”를 만든 후에 Qt designer 를 사용해서 간단한 GUI 만들어 보겠다.

Hello Qt! 프로그래밍¹²

이번 Chapter 에서는 간단한 Qt 프로그램인 “Hello Qt!”를 출력하는 프로그램을 작성하겠다.

아래의 프로그램은 간단한 창에 파랑색으로 Hello Qt!을 써주는 프로그램이다.

소스코드 (hello.cpp)

```
1 #include <qapplication.h>
2 #include <qlabel.h>
3
4 int main( int argc, char **argv )
5 {
6     QApplication app( argc, argv );
7     QLabel *hello = new QLabel( "<font color=blue>Hello <i>Qt</i>"
8                               "</font>", 0 );
9     app.setMainWidget( hello );
10    hello->show();
11    return app.exec();
12 }
```

소스코드 분석

프로그램은 총 12 줄로 구성되어 있다.

1, 2 : 프로그램의 실행에 필요한 헤더파일을 포함(include) 시켰다.

¹² 위의 예제는 <http://www.trolltech.com/products/qt/whitepaper/whitepaper-2-1.html>에서 구할 수 있다.

첫번째 라인에는 Qt 프로그램에서 필요로하는 QApplication 의 Class 의 정의가 들어있는 QApplicaion.h 을 불러온다. 모든 Qt 프로그램에서는 반드시 하나의 QApplication 이 정의되어야 한다.

두번째 라인은 프로그램 내(7, 8 라인의 QLabel)에서 사용하게 될 QLabel.h 를 포함시켰다.

Qt 의 헤더파일은 일반적으로 사용하게 될 “위젯이름.h”의 소문자 형식으로 표현된다.

4 : 프로그램의 시작점인 main 함수를 정의했다. 매개인자로 argc, argv¹³ 를 정의했다.

Qt 에서 main 함수는 프로그램을 시작하고 환경을 설정한 후에 제어권을 Qt 로 넘기는 역할을 한다.

5, 12 : 함수의 시작('{')과 끝('}')을 나타낸다.

6 : QApplication 클래스의 인스턴스 app 를 만든다. QApplication 클래스는 Qt 프로그램에서 공통적으로 필요한 것들을 자동으로 알아서 처리해, 프로그램 개발자는 QApplication 에 대해서 자세한 지식이 없어도 된다. 프로그램 개발에 공통적으로 필요한 부분들이 이 클래스에 포함되어 있다.

X Application 이면 공통적으로 처리해줘야 할 -display, -geometry, -fg, -bg 와 같은 것들을 처리한다.

프로그램 전반에 걸쳐 필요한 공통적인 설정을 할 필요가 있을 때 이 클래스에 필효한 함수를 호출해 프로그램의 특성을 설정하거나 확인할 수 있다.

7, 8 : 'Hello Qt!'라고 Label 을 생성한다.¹⁴

X Window 에서 이렇게 화면에 보이는 Component 를 위젯이라고 하며, Qt 에서는 이러한 위젯을 QWidget 이라는 클래스에서 상속돼 정의된다.(위의 그림 참조) 따라서 위젯들은 대부분의 함수를 상속받기 때문에 공유해서 사용할 수 있다.

QWidget 을 사용하는 경우 클래스들은 부모 위젯(QWidget* parent) 과 인스턴트 이름(const char* name)을 인수할 수 있다. 인스턴트 이름은 지정하지 않으면 NULL 로 할당된다.

¹³ main() 함수에서 사용되는 매개인자들은 프로그램을 실행할 때 명령어 라인으로 넘겨주는 매개인자를 말한다. 예를 들어 우리가 designer 를 실행할 때 다음과 같이 실행한다고 하자.

```
$ designer valentis.ui
```

여기서 매개인자로 valentis.ui 가 넘어가게 된다.

int 형으로 선언되어 있는 argc 는 매개변수의 갯수(여기서는 2개)를 뜻한다. 그리고 char**형으로 선언되어 있는 argv 는 매개변수를 뜻한다. 프로그램 자신도 매개변수에 포함되므로 우리가 생각하는 것에 1 을 더해야 한다.

여기서는 argv[0]는 designer 이고 argv[1]이 valentis.ui 가 된다.

¹⁴ 프로그램을 잘 보면 html 코드로 색을 지정하고 있는 것을 알 수 있다.

부모이름을 지정하지 않거나 0 으로 지정하면 생성되는 위젯은 앞에서 설명한 최상위 레벨(Top Level) 윈도우가 된다.

9 : 앞에서 정의한 hello Label 을 주 위젯으로 할당한다.

주 위젯으로 지정되었을 경우 그 위젯이 없어지면 자동으로 프로그램이 종료된다.

10 : 레이블을 화면에 표시한다.

레이블이 생성되더라도 위의 함수를 실행하지 않으면 화면에 표시되지 않는다. 화면에 표시하기 전에 크기나 위치등을 미리 지정하면 화면이 몇번씩 다시 그려지는 것을 막을 수 있다.

11 : 프로그램을 실행한다.

exec()을 호출하면 사용자와 시스템 이벤트 들을 처리하며, 발생한 이벤트들을 알맞는 위젯에 넘겨준다.

프로그램이 종료될 때까지 내부적으로 로프를 돌면서 필요한 루틴을 실행하게 된다.

Qt 에서 사용되는 기본적인 main()함수는 형태가 비슷하므로 외워두는 것이 좋다.

컴파일하기

gcc¹⁵를 이용해서 다음과 같이 컴파일하면 된다.

```
$ gcc -c hello.cpp -I$QTDIR/include
$ gcc -o hello -L$QTDIR/lib -L/usr/X11R6/lib -lqt -lX11 -lXext
hello.o
```

Mac OS X 에서는

```
$ cc -c hello.cpp -I$QTDIR/include
$ cc -o hello -L$QTDIR/lib -lqt -mt hello.o
```

실행결과



Figure 7. Hello Qt! 실행결과

¹⁵ gcc Option

1. -c : 컴파일시 Object 파일까지만 컴파일한다.
2. -o : 컴파일시 옵션뒤에 있는 이름으로 Application 을 생성한다.
3. -I : 헤더파일이 있는 디렉토리를 뒤에 명시한다.
4. -L : 라이브러리가 있는 디렉토리를 뒤에 명시한다.
5. -l : 컴파일시 포함되어야 할 라이브러리를 명시한다.

Qt 에서 Signal 과 Slot 사용하기¹⁶

이전 Chapter 에서는 간단한 Qt 프로그램을 작성했다.

이번 Chapter 에서는 Qt 프로그램시 반드시 필요한 Signals/Slots 의 개념에 대해서 살펴보도록 하겠다. Qt 프로그래밍시 꼭 필요한 개념이니 주의깊게 살펴 보기 바란다.

소스코드(Calling it Quits)

```

1  #include <qapplication.h>
2  #include <qpushbutton.h>
3  #include <qfont.h>
4
5  int main( int argc, char **argv )
6  {
7      QApplication a( argc, argv );
8
9      QPushButton quit( "Quit", 0 );
10     quit.resize( 75, 30 );
11     quit.setFont( QFont( "Times", 18, QFont::Bold ) );
12
13     QObject::connect( &quit, SIGNAL(clicked()), &a, SLOT(quit()) );
14
15     a.setMainWidget( &quit );
16     quit.show();
17     return a.exec();
18 }
```

위의 프로그램의 기본적인 구조는 앞에서 살펴본 “Hello Qt!”와 같다. 그러므로 다른 점만 살펴보도록 하겠다.

2 : 9 라인에서 사용하는 QPushButton 을 위해서 헤더파일을 포함했다.

3 : 11 라인에서 사용하는 QFont 를 위해서 헤더파일을 포함했다.

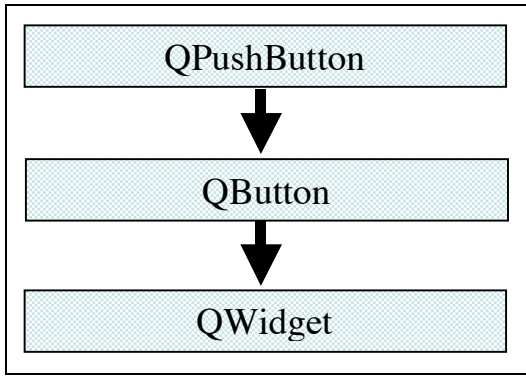
9 : QPushButton 은 사용자의 마우스 입력(마우스 버튼의 클릭/누름/놓기)에 반응하는 위젯이다.

버튼에 나타나는 문자는 “Quit”로 초기화 되었고 버튼을 Top Level 윈도우로 설정하기 위해서 ‘0’으로 초기화 하였다..

10, 11: QPushButton 의 크기와 폰트의 종류와 크기, 그리고 형태에 대해서 정의하였다.

13 : QPushButton Widget 의 Signal 과 Slot 을 연결하였다.

¹⁶ 다음 예제는 <http://doc.trolltech.com/2.3/t2.html>에서 찾을 수 있다.



QPushButton의 상속관계는 다음과 같다. QButton으로 부터 clicked()를 상속받는데 버튼이 클릭이 되면 발생하는 시그널이다. clicked()라는 시그널은 QApplication의 quit()라는 slot에 접속(connect)된다. 그러므로 마우스로 버튼을 클릭하면 프로그램이 끝나게 된다.

실행결과

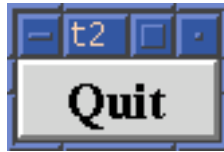


Figure 8. Quit

[부록](#)은 QPushButton에 대한 Trolltech에서 제공하는 레퍼런스이다.(지면상 함수리스트만 열거했다¹⁷.) 레퍼런스를 보면 상속받는 대상과 함수들에 대해서 설명되어 있다.

Qt에서의 Slot 과 Signal

Qt에서 사용되고 있는 Slot 과 Signal에 대해서 살펴보도록 하겠다.

Windows에서의 처리방식

이벤트 처리 방식 프로그래밍(Event-Driven Programming)이라고 해서 사용자가 마우스(클릭하거나 드래그)나 키보드(키를 누르는)를 조작하는 이벤트가 발생했을 때 해당 이벤트에 대해서 응용 프로그램에서 처리해야 하는 코드를 실행하는 것이다. 일반적인 도스에서의 순차적인 실행방식이 아닌 사용자의 행동(Event)에 대한 응답(Response)으로 적절하게 처리하도록 하는 것이다.

사용자가 이벤트를 발생시키면 이벤트의 유형과 파라미터를 메시지 형태로 바꾼 후에 윈도우로 보내 메시지 큐에 저장시킨다. 윈도우 프로그램은 메시지를 디스패칭(dispatching)시켜서 윈도우 프로시저에 전달해서 메시지를 처리하게 한다.

X(Motif)에서의 처리방식

X Server는 클라이언트에게 이벤트를 보냄으로써 클라이언트와 통신한다. 서버는 키보드 키를 누르거나 마우스를 조작하는 사용자 동작(Action)의 직접 또는 간접적인 결과로서 이벤트를 발생시킨다. 또한 서버는 윈도우의 상태 변화를 알리기 위해서도 이벤트를 발생시킨다.

¹⁷ <http://doc.trolltech.com/2.3/qpushbutton.html>를 참조하라

X는 기본적으로 33개의 이벤트를 지원하며 클라이언트로 하여금 추가로 이벤트를 지정할 수 있는 방법을 제공하고 있다.

Event dispatching, Action, Callback 그리고 Event Handler이라는 매커니즘을 제공한다.

모든 Xt는 이벤트 루프를 포함한다. 이 루프에서 응용프로그램은 이벤트 큐(queue)로부터 다음 이벤트를 꺼낸 후, 사건이 일어난 윈도우를 토대로 Widget에게 이벤트를 보낸다.

Action이란 특정사건 혹은 일련의 사건이 발생하였을 때 호출될 함수를 미리 등록하는 것을 말한다.

Callback이란 Widget에서 특정조건이 만족되었을 때에 호출되도록 어떠한 인수를 받아들여야하는 C 함수이자 Widget에 등록된 함수이다. 모든 모티브 위젯은 등록되어야 하는 함수를 위해 확실한 Callback형을 알고 있다.

하지만 데이터형 안정성(type-safety)에 대한 보장을 할 수 없다. 만약 잘못된 형태의 인수를 받아드리도록 함수를 등록한다면 작성한 프로그램은 예기치 못한 문제가 발생할 수 있다. 왜냐하면 컴파일러는 이러한 문제를 해결할 수 있는 능력을 갖고 있지 않기 때문이다.

만약 C++을 사용해 객체지향 프로그램으로 구현된다면 Callback은 this 포인터의 구분이 명확하지 못하기 때문에 구현하기 매우 까다롭다. Event handler는 X에 어떤 이벤트가 발생했을 때 호출되도록 Xt에 등록된 함수를 가르킨다.

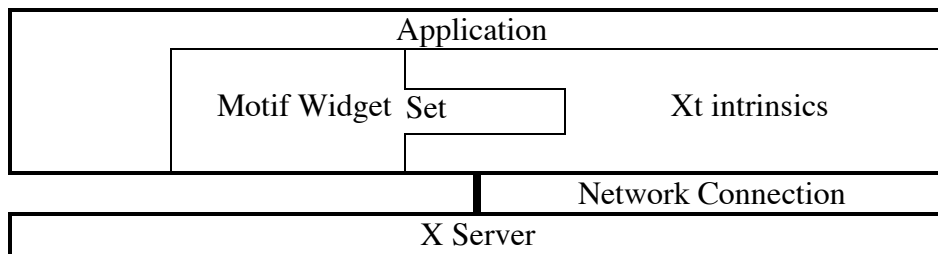


Figure 9. X(Motif)에서의 Server 와 Client

Signals and Slots

Signal과 Slot은 객체사이에 통신을 하기 위해서 사용된다. Signal/slot 방식은 Qt의 기본적인 중요 방식이며 다른 툴킷과의 두드러진 차이점이다.

다른 툴킷에서 사용하는 Callback 방식은 각 행동에 대한 함수의 포인터인 Callback을 이용하여 처리하는 방식이다.

Callback 방식은 발생한 이벤트를 중앙에서 어떠한 위젯이 필요한 지를 알아서 배분해 주는 것이다. 그러나 Signals/Slots 방식은 이벤트(Signal)가 발생하면 1:1로 되어있는 연결통로(Slot)를 이용하여 위젯에 이벤트가 넘겨지는 방식이다.

Signal/Slot 방식에서는 매개변수의 갯수와 형에 상관없이 사용할 수 있다. Callback과는 다르게 형(type)에 안정하다.

QObject로부터 상속받거나 상속받은 클래스(Subclass)로부터 상속받는 경우 Signal과 Slot을 가지고 있다.

Signal은 객체가 의미있는 상태로 상태를 변화했을 때 발생하며 Slot은 이것을 받아서 처리한다.

Signal은 어떤 객체의 Slot에 접속이 되는지 모르고 Slot도 어떤 객체의 Signal이 접속되는지 알 수 없다. 이것을 정보은닉(Information encapsulation)이라고 부른다.

복수개의 Signal을 하나의 Slot에 접속시킬 수 있고 하나의 Signal을 복수개의 Slot에 접속시킬 수 있다.

Qt designer 를 이용하여 주소록 만들기

이전 Chapter에서는 간단한 Qt 프로그램을 작성했다.

이번 Chapter은 심화학습의 장으로 Trolltech에서 기본으로 제공해주는 Qt designer의 사용법과 Signal과 Slot에 대한 학습을 시작하겠다.

Qt designer는 Qt 2.xx 버전과 Qt 3.0 버전과는 차이가 있지만 현재 많은 사람들이 사용하고 있는 Qt 2.xx 버전으로 강의를 진행하겠다.

Qt 3.0 버전은 KDE 3.0이 대중화되는 시기를 맞춰서 널리 사용될 것 같다.

Qt designer 에 대한 간단한 설명(2.xx 에서)

Qt Designer는 사용자가 만들고 싶은 GUI를 쉽게 에디팅하게 해주는 프로그램으로 .ui의 파일포맷을 가지있다.

.ui파일은 XML 형식으로 되어 있고 uic를 이용하여 C++에서 인식할 수 있는 소스코드로 변환한다.

Qt designer 사용하기

Qt가 올바르게 설치가 되어 있다면 다음과 같은 방법으로 실행시킬 수 있다.

```
$ designer18
```

Mac OS X을 사용하고 있다면 <Qt가 설치된 디렉토리>/bin로 가면 Qt designer를 찾을 수 있다.

이제 Qt designer를 실행하도록 하겠다.

```
$ designer Addressbook.ui  
(Qt designer의 확장자는 ui이다.)
```

라고 실행시켜 보자.

물론 처음에 디자이너만 실행시키고 저장시에 파일명을 입력해도 된다.

¹⁸ 실행이 되지 않는다면 Qt가 설치되어 있지 않거나 Path에 잡혀있지 않는 것이다.

\$ which designer(또는 whereis designer)

로 designer가 나타나지 않는다면 위의 것을 의심하라.

설치가 되어 있는데 Qt designer가 실행되지 않는다면 Qt 1.44 이하 버전일수도 있다.

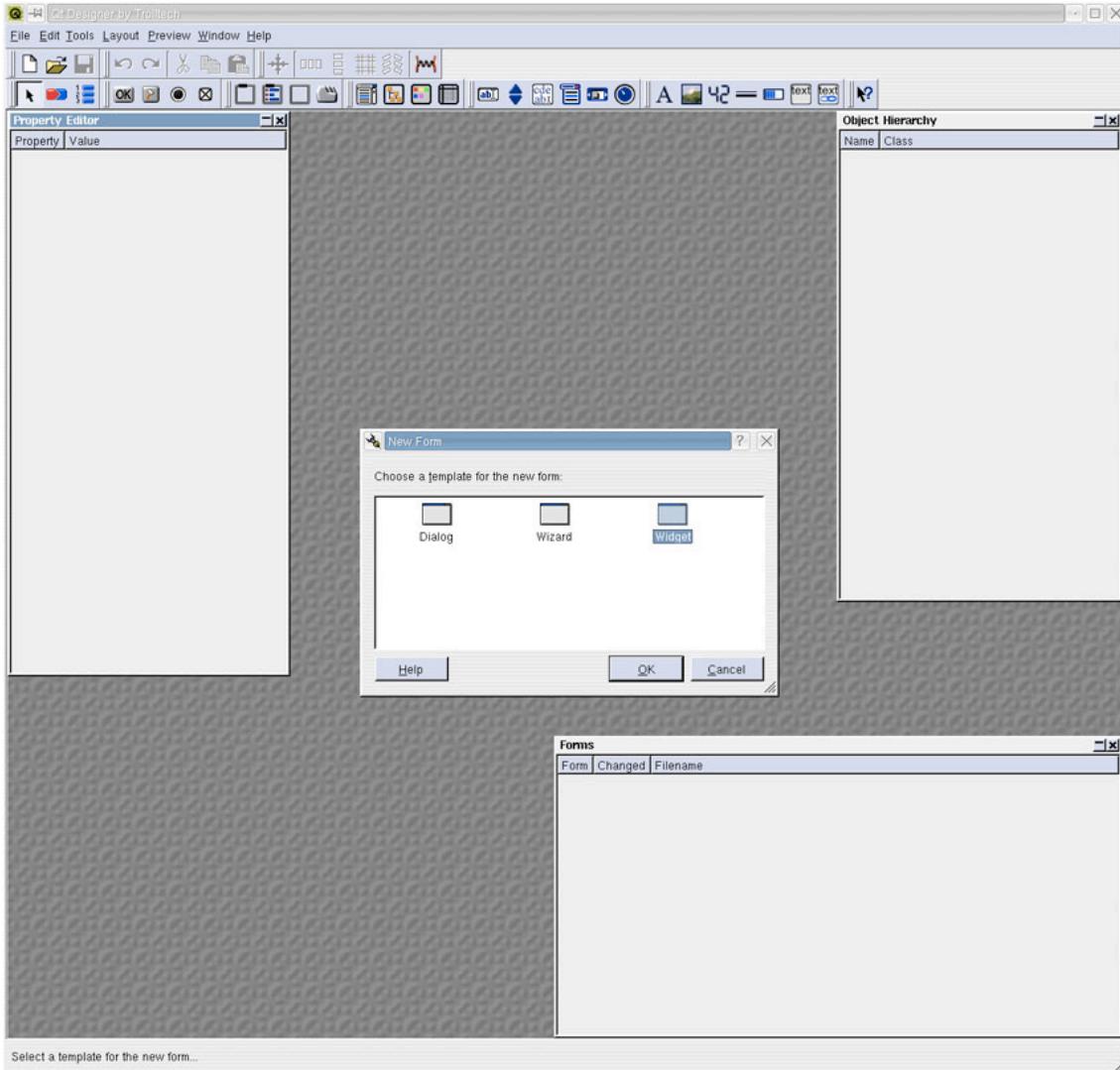


Figure 10. Qt designer 실행화면 - Addressbook.ui

프로그램을 시작하면 다음과 같은 화면이 나온다.
이제부터 본격적으로 프로그래밍을 시작해보자.

Qt designer 를 실행시키면 3 가지의 선택사항이 나온다.

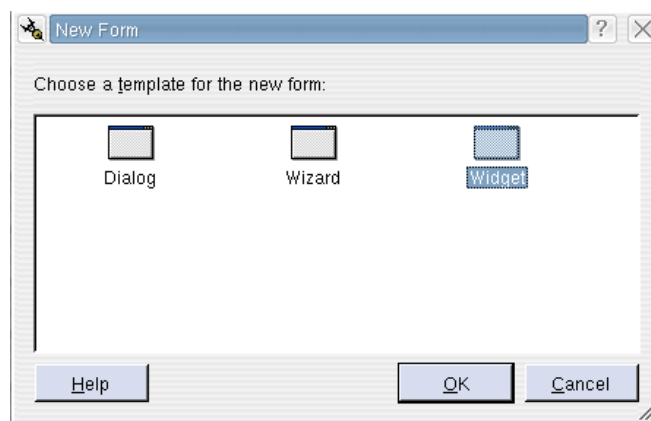


Figure 11. 생성할 폼의 형태를 결정

1. Dialog
2. Wizard
3. Widget

위에서 Wizard 는 마법사와 같은 GUI 를 만들때 사용되는 것이다.
GUI 프로그래밍을 해보신 분들은 알고 있겠지만 dialog 는 그 자신의 창을 가지고 있지만 Widget 의 경우 보통 다른 dialog 나 widget 에 포함되어 있다.

우리는 여기서 Widget 을 선택하고 OK 버튼을 누르면 된다.
OK 버튼을 클릭하면 아무것도 없는 비어있는 폼이 하나 생성된다.

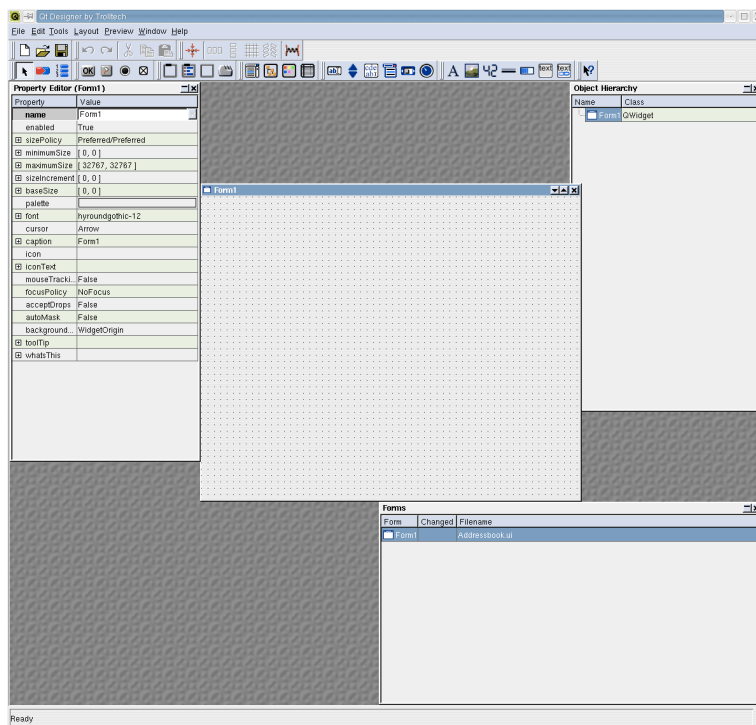


Figure 12. 비어있는 폼(Empty Form)

제일 먼저해야 할 작업은 생성할 Class 의 이름과 정보를 입력하는 것이다.(물론 나중에 해줘도 되지만 처음 생성할 때 해주는 것이 좋다.)

메뉴에서 edit>>Form Settings 를 선택하면 다음과 같은 다이얼로그가 열린다.

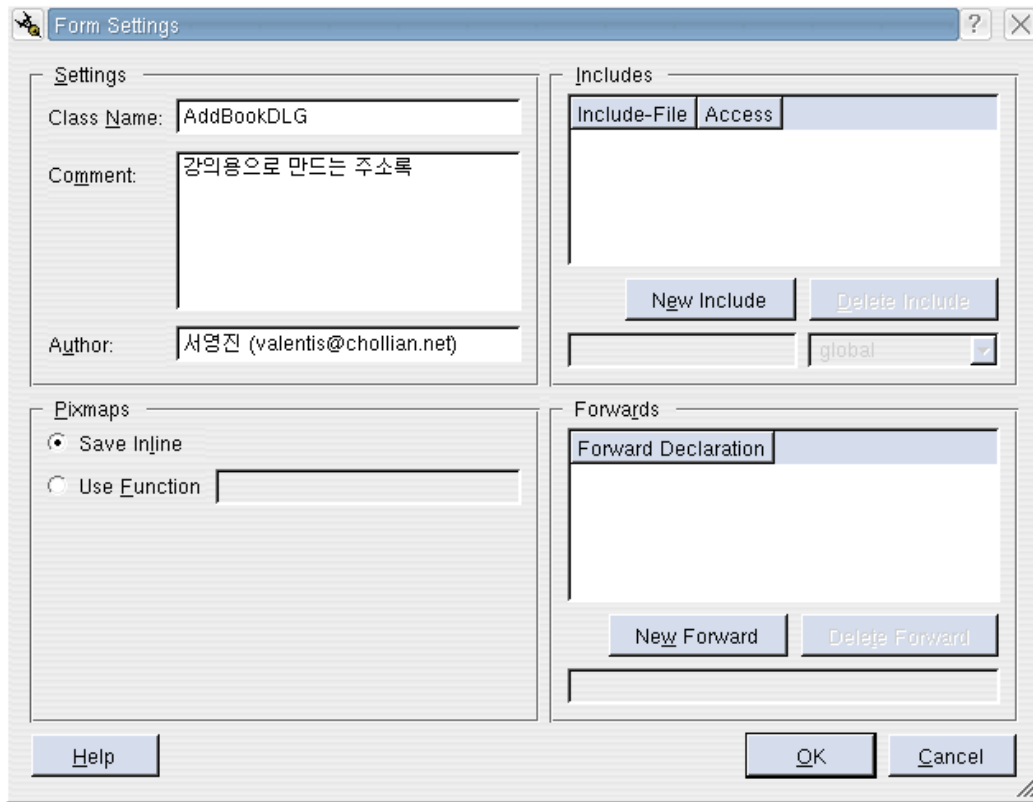


Figure 13. 폼 세팅하기


Setting 이라는 그룹박스 안의 Class Name 원하는 Class 명을 넣는다. 그리고 Comment 와 만든이의 정보를 넣고 OK 버튼을 클릭하면 기본적인 폼 세팅은 끝나게 된다. 보다 자세한 사항은 troltech 사의 홈페이지에서 Qt designer Manual¹⁹을 참조한다.

이제 본격적으로 GUI 인터페이스를 구성해 보겠다.

우선 비어있는 폼에 어떤 위젯을 삽입하고 싶을 때에는

















에서 원하는 위젯을 선택(아이콘을 클릭)하고 비어있는 폼에 마우스를 대고 드래그²⁰하면 원하는 크기로 원하는 위치에 위젯을 위치시킬 수 있다.











	위젯	설 명
	PushButton	사용자가 ‘눌러서’ 어떤 동작(Action 또는 command) 이 발생하게 하는 위젯을 말한다. PushButton 에 문자나 그림을 이용해서 어떤 역할을 하는지 시각적인 정보를 제공할 수 있다.

¹⁹ 2.xx 버전의 Qt designer 는 <http://doc.troltech.com/2.3/designer/book1.html> 을 참고하면 된다.

²⁰ 마우스의 왼쪽 버튼을 누른 상태에서 마우스를 움직이는 것, 원하는 위치에서 마우스 왼쪽 버튼을 놓으면 된다.

	Tool Button	특별한 명령이나 옵션을 빠르게 접근하는 것을 제공하는 버튼으로 보통 QToolBar 내부에서 사용된다.
	Radio Button ²¹	동그란 박스와 설명을 위한 레이블을 가지고 있다. 보통 두개 이상을 버튼그룹박스로 그룹을 이룬 후에 사용하는데 여러개의 선택사항 중 하나만을 선택할 때 사용한다.
	Checkbox	Radio Button 과 기본적인 사용은 같지만 동그란 박스가 아닌 네모난박스를 가지고 있다. 사용자가 네모박스를 클릭해서 체크표시를 하거나 없앨 수 있다. 특정 값을 표시할 때 TRUE/FALSE 의 값을 리턴한다.
	Groupbox	서로 관련된 위젯을 묶는데 사용되며 제목을 붙여서 시각적인 효과를 제공한다.
	ButtonGroup	보통 Push Button 이나 Radio Button 을 묶기 위해서 사용된다.
	Frame	제목은 갖지 않고 서로 관련된 위젯을 시각적으로 묶는데 사용된다.
	Tab Dialog	Tab 다이얼로그를 구현할 때 사용한다.복수의 폼을 단일 위젯의 하나의 폼에 구성하려고 할 때 이용된다.
	Listbox	문자열로 구성된 정보를 표시, 각 항목을 마우스를 이용하여 선택이 가능하다.
	List View	Listbox 가 문자열로 구성된 항목만을 나열하는데에 비해서 ListView 는 여러 컬럼으로 구성된 세부적인 정보와 다양한 기능을 제공한다.
	Icon View	아이콘과 같은 Pixmap 을 나열하기 위해서 사용된다.
	Table View	항목을 스프레드시트에서 사용하는 테이블 형식으로 나열한다.
	Line Edit	사용자에게 어떤 문자를 입력하거나 수정하게 할 때 사용된다.
	Spin Box	Label 위젯 또는 다른 위젯에 표시되는 숫자값을 증가시키거나 감소시키는데 사용된다.
	MultiLine Edit	여러 줄로 입력되는 값을 받거나 수정하고자 할 때 사용된다.

²¹ Radio Button 은 Radio 를 생각하면 어떤 결과를 하는지 쉽게 알 수 있다. 우리가 Radio 에서 어떤 방송국을 선택하면 다른 방송국은 선택할 수 없다(선택해지가 된다.)Radio 한 대로 한순간에 한개의 방송국만 선택할 수 있다.

	Combo Box ²²	Dropdown List Box 라고도 불리우는데 에디터를 할 수 있는 영역과 리스트 영역(여러개의 값중 하나를 선택할 수 있는 영역)으로 구성되어 있다. 사용자에게 어떤 값을 선택하게 하고 싶을 때 사용된다.
	Slider	슬라이더와 틱(Tick)마크를 포함하며, 사용자가 마우스를 이용하여 슬라이더를 끌거나 좌우측으로 이동할 수 있다.
	Dial	스핀박스과 기본적인 기능은 같지만 보다 가식적인 형태로 입력값과 출력값을 표시한다.
	Text Label	사용자에게 정보를 제공하는 단순한 텍스트를 나타내기 위해서 사용된다. 단지 읽기를 위한 용도이지만 프로그램 내부적으로 나타나는 문자를 변경하게 할 수 있다.
	Pixmap Label	사용자에게 이미지로 된 정보를 나타내는데 사용된다. 기본적인 기능은 Text Label 과 같다.
	LCD Number	전자시계에서 사용하는 LCD Display 같은 형태로 숫자를 표시한다.
	Line	화면에 선을 나타내기 위해 사용된다. 항목간의 구분이나 시각적인 효과를 위해서 사용한다.
	Progress bar	박스 안의 Bar 의 크기를 조절하여 시각적인 양의 %를 나타낸다. 보통 작업의 진행사항을 표시할 때 사용한다.
	TextView	한 줄의 텍스트가 아닌 여러 줄의 텍스트를 표시하기 위해서 사용된다.
	Text Browser	TextView 에 간단한 탐색기능을 추가한 문자 표시기이다.

Qt designer 에서 아이콘으로 제공하는 위젯이 Qt 에서 제공하는 모든 위젯은 아니다. 그러므로 필요한 위젯은 나중에 프로그래머가 코딩을 통해서 추가해야 한다.

우리가 프로그램에서 사용할 위젯은 Text Label, ListView, Group Box, Push button 와 Line Editor 이다. 그리고 여기에는 나타나있지 않지만 MessageBox 를 이용할 것이다.

이제 다시 Qt designer 를 가지고 GUI Interface 를 구성해보도록 하자.

다음에 할 작업은 메인 Form 에 위젯의 이름을 넣고 타이틀(Caption)을 넣는 것이다.

²² Radio Button 과 ComboBox 의 일반적인 용도는 비슷하다. 여러 개의 값중 하나의 값을 선택받는다. 하지만 Combo 박스의 경우 Radio Button 보다 더 다양한 기능을 제공한다. 공간적인 제약이나 새로운 값을 추가할 수 있어 Radio Button 에 비해서 보다 자유롭게 사용할 수 있다.

아래와 같이 빈 Form 을 선택하면 해당하는 Form 에 맞게 Property Editor 가 바뀐다. QPushButton 을 선택하면 QPushButton 에 맞게 RadioButton 을 클릭하면 RadioButton 에 맞는 Property 가 나타난다.

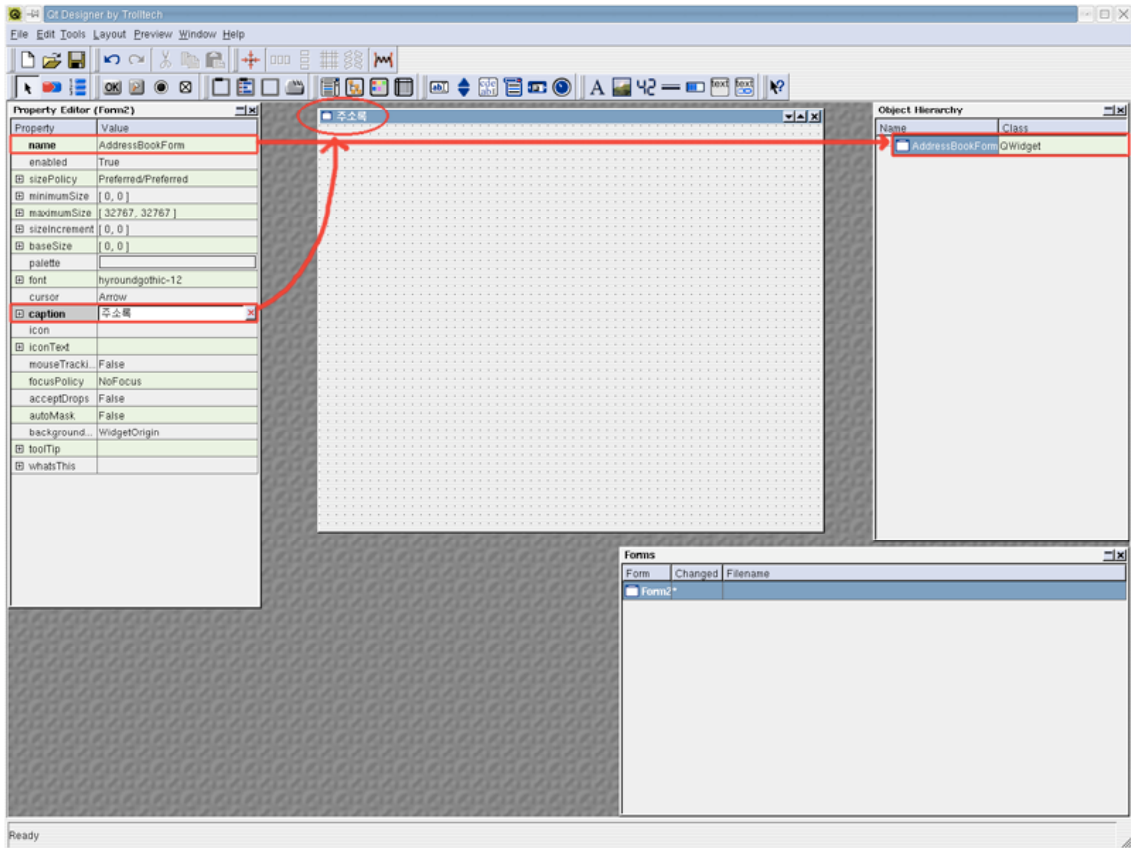


Figure 14. Form Setting 하기

우선 위에 그림에서 빨간표시가 되어있는 name 과 caption 에 값을 넣도록 해 보자.

- Name : 위젯의 이름
C++ Code 에서 사용되는 객체의 이름을 적는 곳
- Caption : Title bar 에 나타나는 이름을 적는 곳

이제 기본적인 폼을 사용할 수 있는 준비를 끝냈다. 나머지 Property 에 대해서는 Trolltech 의 문서들을 참조하라.(widget 부분을 참조)

화면의 오른쪽 Object Hierarchy 를 보면 Class 가 우리가 처음 선택한 QWidget 인 것을 알 수 있다.

이제 화면에 ListView 를 추가해보자.

주소록은 개인이 여러개의 항목을 가지고 있기 때문에 Listbox 나 Table 보다 는 Listview 가 적절하다. 우리는 주소록의 항목으로 “성명”, “전화번호”, “주소”, 그리고 “비고”를 넣을 것이다.

아래 그림과 같이 적절한 위치에 Listview 를 배치하면 된다.

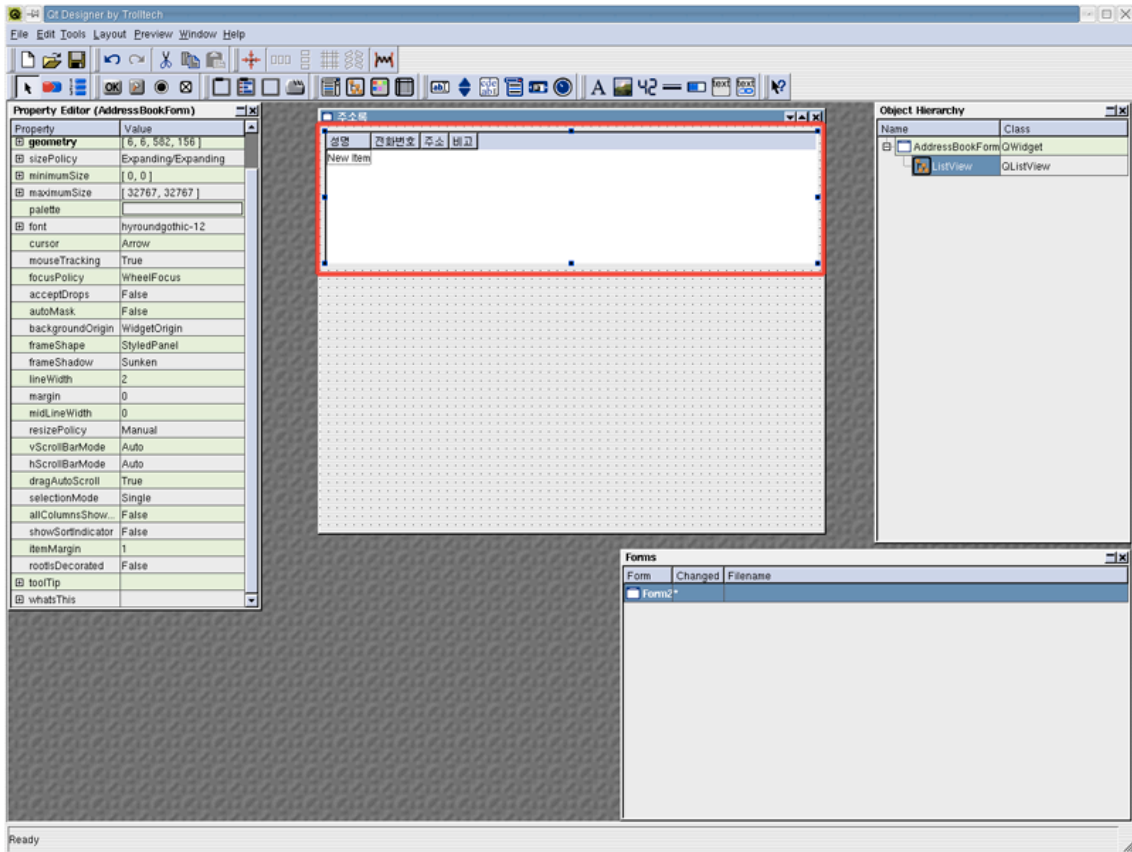


Figure 15. Listview

Listview 에 항목이나 Column 을 추가하려면 화면의 Listview 를 더블클릭하자. 그러면 다음과 같은 다이얼로그 박스가 나온다.

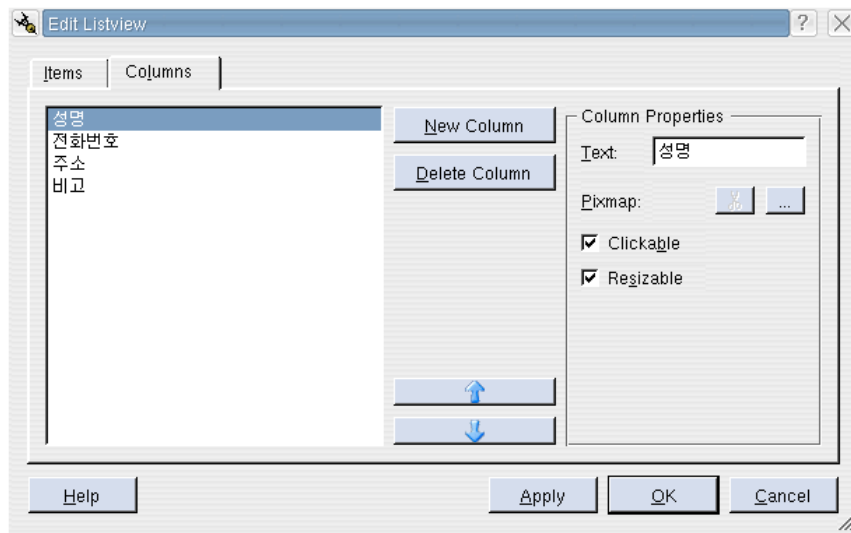


Figure 16. Listview 항목 추가하기

항목을 추가해서 원하는 형태로 Listview 를 세팅하고 난 후에 우리는 사용자 정보를 입력받을 수 있도록 입력폼을 만들 것이다. 사용자 입력으로 사용되는 위젯은 서로 관계가 있기 때문에 GroupBox 를 사용해서 묶고 그 안에 위젯들을 배치할 것이다.

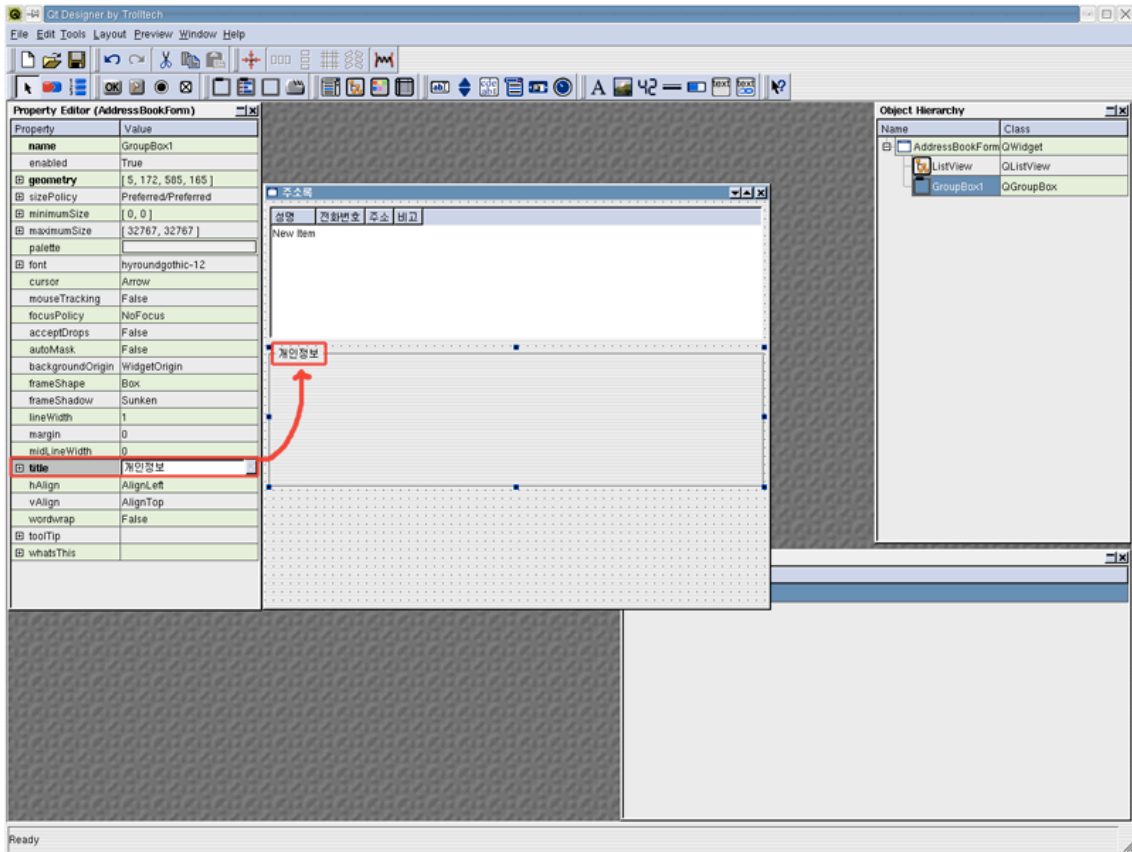


Figure 17. GroupBox Setting

GroupBox 의 설정도 폼과 마찬가지로 name 과 title 을 입력하면 된다.

이제 GroupBox 안에 Label 을 위치시키고 값을 입력하자.

레이블도 다른 위젯과 마찬가지로 Text Label 아이콘을 클릭하고 원하는 위치에 드래그해서 놓으면 된다.

Text Label 도 Property Editor 의 text 항목에 문자를 입력하면 된다.

Text Label 에는 또 다른 입력방법이 있는데 Label 을 더블클릭하면 text 를 입력하는 창이 따로 뜨는데 이창을 이용해서 내용을 입력해도 된다.

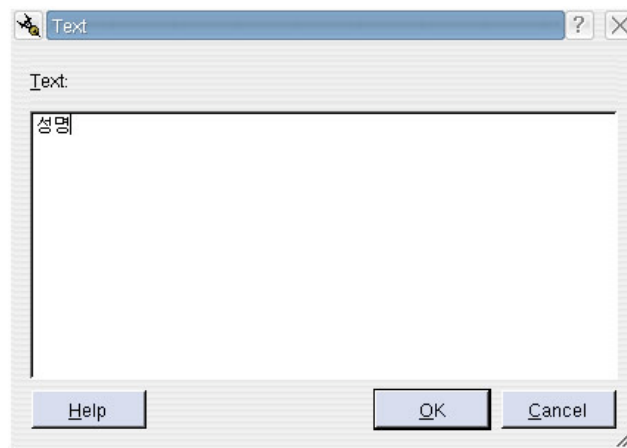


Figure 18. Text Label 에 Text 입력하기

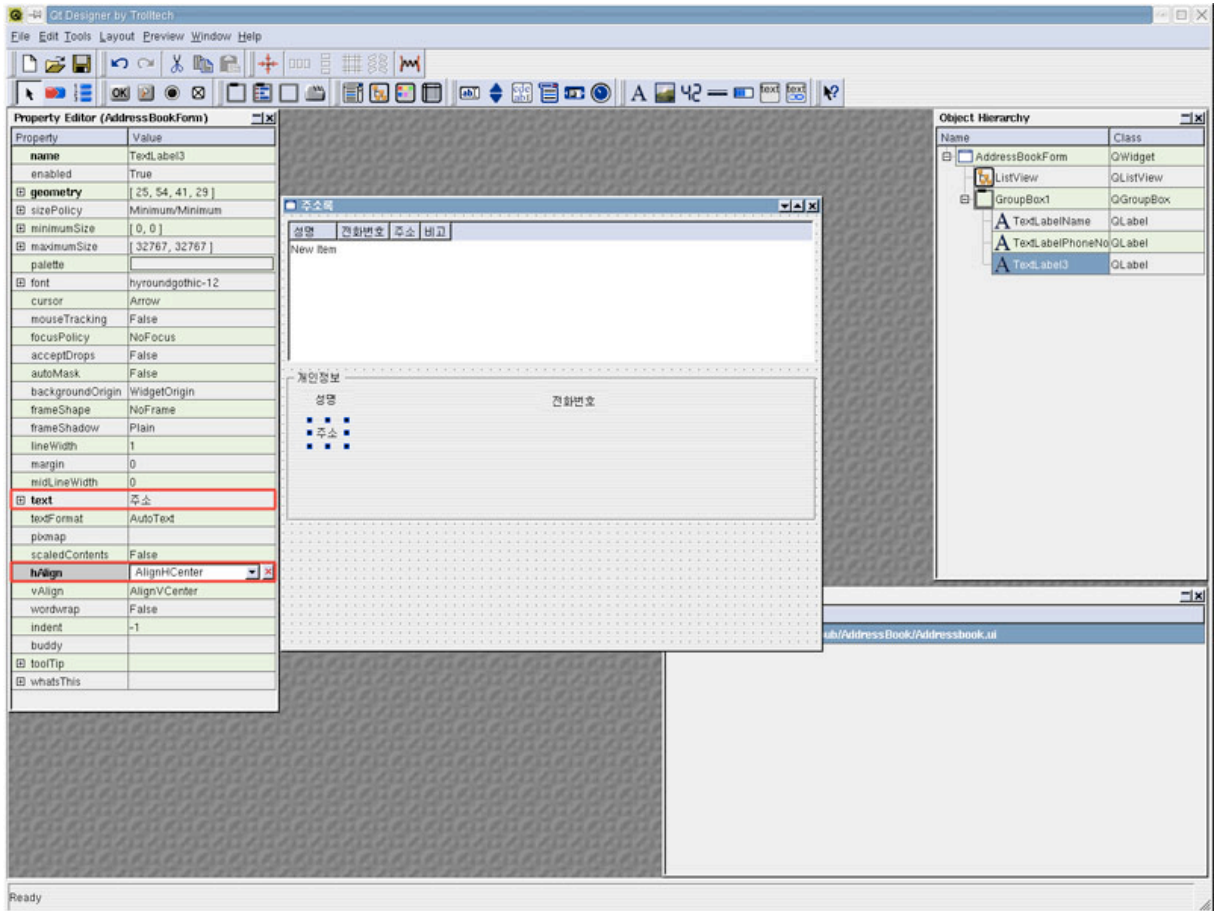


Figure 19. Label

위와 같이 적절한 위치에 Text Label 위젯을 위치시키면 된다.

Text Label 을 위치시킬 때의 몇가지의 주의점이 있다.

1. 생각하는 크기보다 좀 더 크게 공간을 배치하라.
 사용하는 배포판이나 윈도우 매니저에 따라서 기본적으로 사용하는 폰트의 종류가 다르다. 폰트마다 같은 크기에서 나타나는 글자의 크기가 다르므로 적당한 공간을 두지 않으면 다른 시스템에서는 글자가 잘리는 현상이 발생한다.
2. 정렬(Align)에 신경을 써라
 예제의 경우는 가운데 정렬로 배치했다. 위와 같은(주의점 1 번의) 이유로 다른 시스템에서는 보이는 위치가 다를 수 있다.

이제는 사용자 정보를 입력받을 수 있는 LineEdit 를 배치하자.

Text Label 을 배치하는 것과 마찬가지로 LineEdit 를 배치하면 된다. 입력받기를 원하는 글자의 수에 맞춰서 LineEdit 의 크기를 조정하면 된다.

LineEdit 에는 EchoMode 라는 매소드가 존재하는데 3 가지의 형태로 입력을 처리하는 형태를 결정할 수 있다.

- Normal Mode : 입력받은 문자를 그래도 LineEdit 창에 표시한다.
- NoEcho Mode : 입력받은 문자를 LineEdit 창에 표시하지 않는다.

- Password Mode : 입력받은 문자를 글자의 갯수만큼 '*'로 바꿔서 LineEdit 창에 표시한다. 로그인시 패스워드를 입력받는 부분으로 생각해보면 쉽게 이해가 될 것이다.

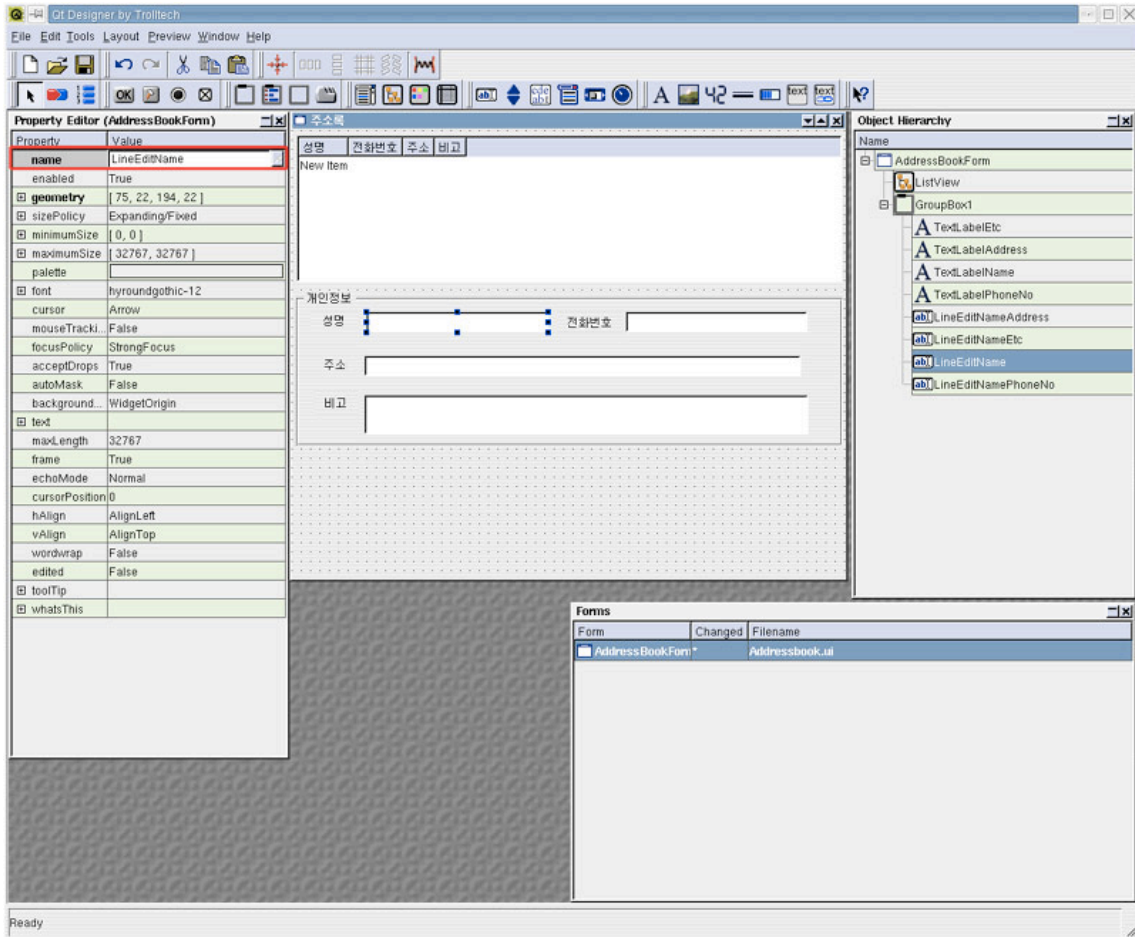


Figure 20. LineEdit

LineEdit 사용시 꼭 잊지 말아야 할 사항이 name 을 잘 입력하는 것이다. LineEdit 는 사용자에게 정보를 입력받는 곳이기 때문에 프로그램 중에 사용하는 빈도의 수가 많다. 처음부터 가독성이 좋은 이름으로 해놓으면 프로그램 작성시 일일이 기억하지 못해서 소스코드에서 위젯의 이름을 찾아 헤매는 수고를 덜 수 있다.

이제 프로그램의 입력값을 저장/삭제할 수 있는 PushButton 과 Help 를 화면에 표시할 수 있는 PushButton 을 배치할 차례이다. 이것도 아이콘을 클릭해서 적절한 위치에 위젯을 배치하면 된다.

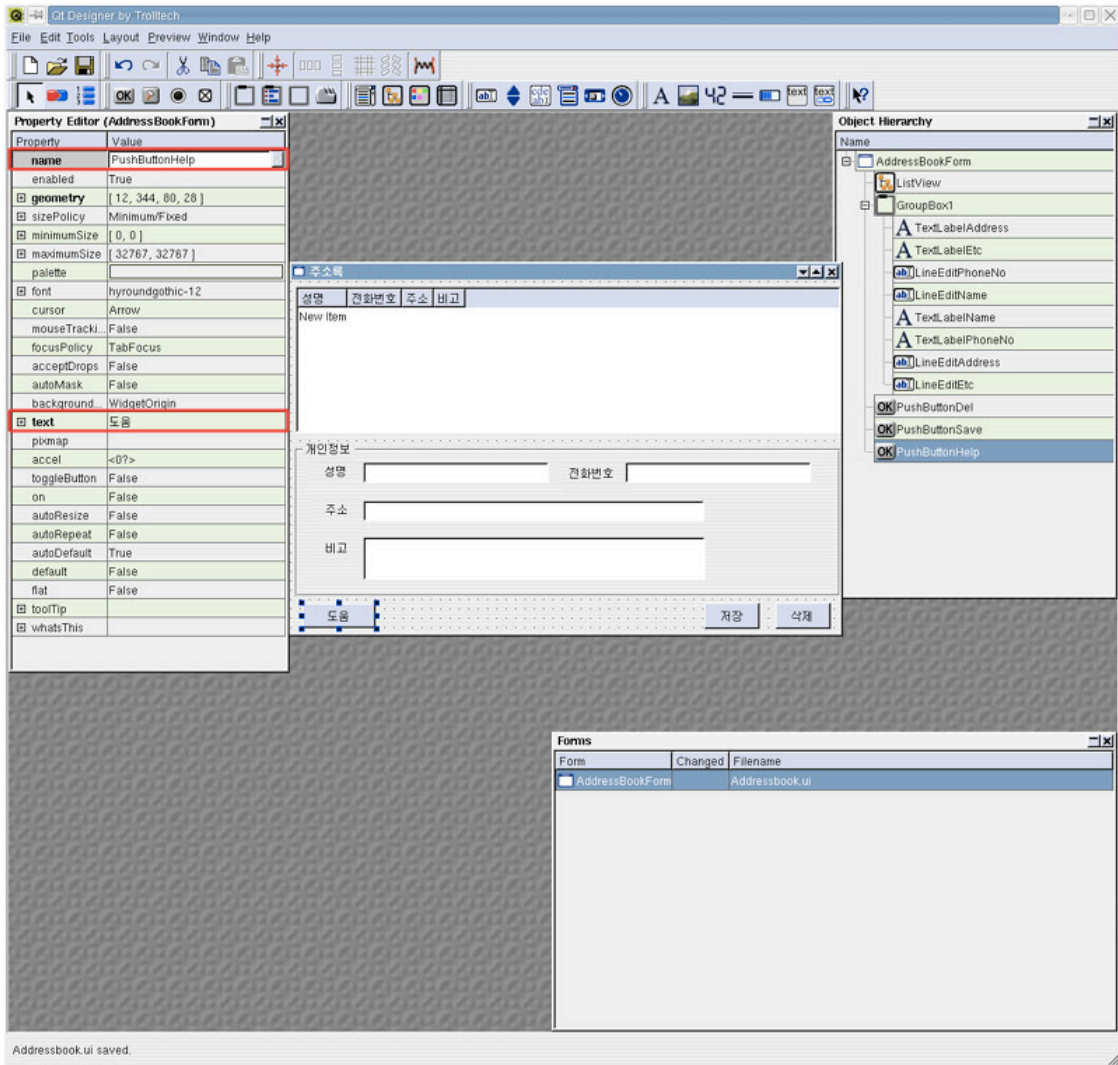


Figure 21. PushButton

버튼에 표시되는 텍스트는 Text Label 과 같이 입력하면 된다.

원하는 위젯의 세팅이 다 되었으면 위젯의 크기를 알맞게 조정하고 Object Hierarchy 를 통해서 위젯의 이름과 Class 의 타입이 제대로 들어가고 있는지 확인하고 저장을 하자.

프로그램을 종료하기전에 마지막으로 해야할 것은 Tab 키를 눌렀을 때의 순서를 정하고 프로그램에서 사용할 기본적인 Signal 과 Slot 을 정의하는 것이다.

우선 화면에서 세번째의 숫자가 있는 아이콘을 클릭을 하든지 F4 키를 누르면 화면이 다음과 같이 바뀐다.

마우스를 이용해서 Tab 키를 눌렀을 때 작동되길 원하는 순서대로 위젯을 클릭해주면 번호가 바뀐다.

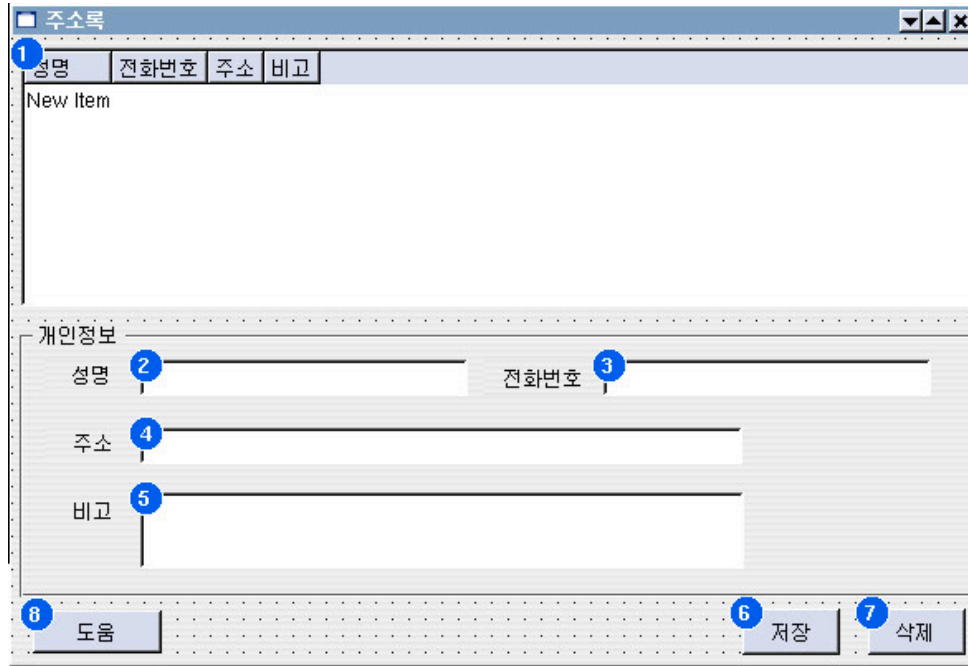


Figure 22. Tab Order

이제 Slot/Signal 을 연결해보자.

Slot 과 Signal 은 Connect Signal/Slot 아이콘이나 F3 키를 누른 후에 원하는 위젯을 연결해주면(화면상에는 가상의 선으로 연결된다.) 다이얼로그 창이 띄게 된다.

예로서 간단한 PushButton 과 LineEdit 를 선택해 보았다.(실제 프로그램에서는 사용하지 않는다.)

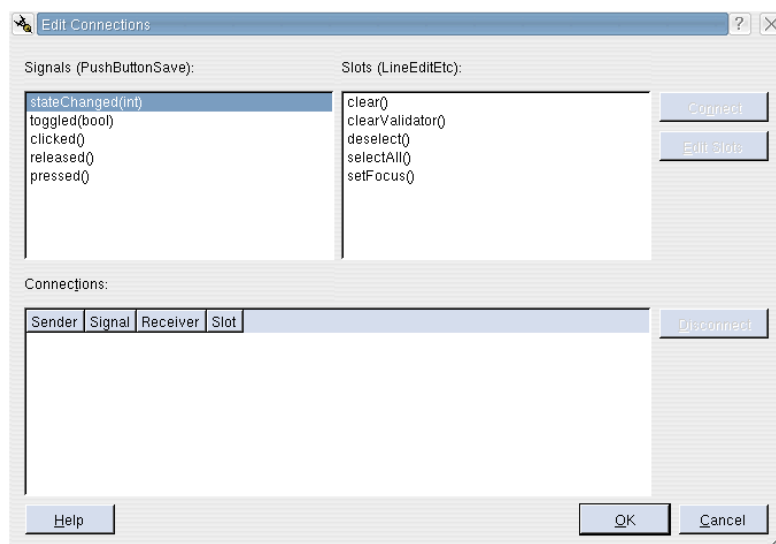


Figure 23. Connect Signal/Slot

왼쪽창에는 PushButton 에서 사용할 수 있는 Signal 의 종류가 나와있고 오른쪽 창에는 LineEdit 에서 사용할 수 있는 Slot 의 종류가 나와있다. 위의 창에서 알맞은 값을 정하고 오른쪽의 Connect 버튼을 클릭하면 된다.

삭제시에는 아래의 Connection 에서 삭제하기를 원하는 항목을 선택하고 disconnect 버튼을 누르면 된다.

이제 Ctrl+T 를 눌러보자. 그러면 Preview 모드로 들어가서 만든 위젯을 미리 테스트해 볼 수 있다. Preview 모드를 통해서 위젯이 화면에 나타나는 모양이나 기본적인 동작을 테스트해 볼 있다.

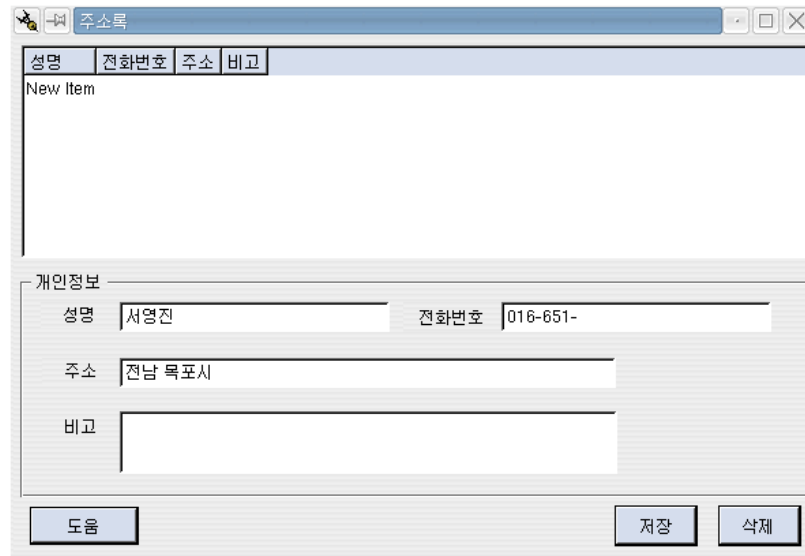


Figure 24. Preview Mode

테스트가 끝났으면 저장을 하고 이제 본격적으로 코딩에 들어가보도록 하자. 우선 Qt designer 를 사용해서 만든 파일(.ui)을 C++코드로 변경해야 한다.

이때 사용되는 것이 **uic**(UI Compiler)라는 것인데 사용방법은 다음과 같다. 우선 .ui 파일을 이용해서 .h 파일을 만든후에 .h 파일과 .ui 파일을 이용해서 .cpp 파일을 만들면 된다.

uic 에 대한 자세한 내용은 Trolltech 홈페이지를 참조하라.

```
$ uic Addressbook.ui > Addressbook.h
$ uic -impl Addressbook.h Addressbook.ui > Addressbook.cpp
```

위와 같이하면 Qt designer 에서 만든 GUI 에 대한 기본적인 소스파일이 완성 된다.

소스파일(Addressbook.h)

```

/*****
** Form interface generated from reading ui file 'Addressbook.ui'
**
** Created: Wed Jan 23 21:19:01 2002
** by: The User Interface Compiler (uic)
**
** WARNING! All changes made in this file will be lost!
*****/
#ifndef ADDRESSBOOKFORM_H
#define ADDRESSBOOKFORM_H

#include <qvariant.h>

```

```
#include <qwidget.h>
class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;
class QGroupBox;
class QLabel;
class QLineEdit;
class QListView;
class QListViewItem;
class QPushButton;

class AddBookDLG : public QWidget
{
    Q_OBJECT

public:
    AddBookDLG( QWidget* parent = 0, const char* name = 0, WFlags fl = 0 );
    ~AddBookDLG();

    QListView* ListView;
    QGroupBox* GroupBox1;
    QLabel* TextLabelAddress;
    QLabel* TextLabelEtc;
    QLineEdit* LineEditPhoneNo;
    QLineEdit* LineEditName;
    QLabel* TextLabelName;
    QLabel* TextLabelPhoneNo;
    QLineEdit* LineEditAddress;
    QLineEdit* LineEditEtc;
    QPushButton* PushButtonDel;
    QPushButton* PushButtonSave;
    QPushButton* PushButtonHelp;

};

#endif // ADDRESSBOOKFORM_H
```

소스파일(Addressbook.cpp)

```
/*
*****
** Form implementation generated from reading ui file 'Addressbook.ui'
**
** Created: Wed Jan 23 21:19:19 2002
** by: The User Interface Compiler (uic)
**
** WARNING! All changes made in this file will be lost!
*****
#include "Addressbook.h"

#include <qgroupbox.h>
#include <qheader.h>
#include <qlabel.h>
#include <qlineedit.h>
#include <qlistview.h>
#include <qpushbutton.h>
```

```

#include <qlayout.h>
#include <qvariant.h>
#include <qtooltip.h>
#include <qwhatsthis.h>
#include <qimage.h>
#include <qpixmap.h>

/*
 * Constructs a AddBookDLG which is a child of 'parent', with the
 * name 'name' and widget flags set to 'f'
 */
AddBookDLG::AddBookDLG( QWidget* parent, const char* name, WFlags fl )
    : QWidget( parent, name, fl )
{
    if ( !name )
        setName( "AddressBookForm" );
    resize( 584, 378 );
    setCaption( tr( QString::fromUtf8( "二晝 Ūā 濡 ù" ) ) );

    ListView = new QListView( this, "ListView" );
    ListView->addColumn( tr( QString::fromUtf8( "īÑ ħTMÖ" ) ) );
    ListView->addColumn( tr( QString::fromUtf8( "ī † Nīôî 踰 àì*" ) ) );
    ListView->addColumn( tr( QString::fromUtf8( "二晝 Ūā" ) ) );
    ListView->addColumn( tr( QString::fromUtf8( "諭 Ñ 怨 †" ) ) );
    QListWidgetItem * item = new QListWidgetItem( ListView, 0 );
    item->setText( 0, tr( "New Item" ) );

    ListView->setGeometry( QRect( 6, 6, 582, 156 ) );

    GroupBox1 = new QGroupBox( this, "GroupBox1" );
    GroupBox1->setGeometry( QRect( 5, 172, 582, 165 ) );
    GroupBox1->setTitle( tr( QString::fromUtf8( "媛 úù 兠 †ī 蹂*" ) ) );

    TextLabelAddress = new QLabel( GroupBox1, "TextLabelAddress" );
    TextLabelAddress->setGeometry( QRect( 16, 63, 58, 19 ) );
    TextLabelAddress->setText( tr( QString::fromUtf8( "二晝 Ūā" ) ) );
    TextLabelAddress->setAlignment( int( QLabel::AlignCenter ) );

    TextLabelEtc = new QLabel( GroupBox1, "TextLabelEtc" );
    TextLabelEtc->setGeometry( QRect( 16, 104, 58, 19 ) );
    TextLabelEtc->setText( tr( QString::fromUtf8( "諭 Ñ 怨 †" ) ) );
    TextLabelEtc->setAlignment( int( QLabel::AlignCenter ) );

    LineEditPhoneNo = new QLineEdit( GroupBox1, "LineEditPhoneNo" );
    LineEditPhoneNo->setGeometry( QRect( 353, 22, 198, 22 ) );

    LineEditName = new QLineEdit( GroupBox1, "LineEditName" );
    LineEditName->setGeometry( QRect( 73, 22, 198, 22 ) );

    TextLabelName = new QLabel( GroupBox1, "TextLabelName" );
    TextLabelName->setGeometry( QRect( 15, 22, 58, 19 ) );
    TextLabelName->setText( tr( QString::fromUtf8( "īÑ ħTMÖ" ) ) );
    TextLabelName->setAlignment( int( QLabel::AlignCenter ) );

```

```

TextLabelPhoneNo = new QLabel( groupBox1, "TextLabelPhoneNo" );
TextLabelPhoneNo->setGeometry( QRect( 281, 22, 72, 22 ) );
TextLabelPhoneNo->setText( tr( QString::fromUtf8( "☎️" ) ) );
TextLabelPhoneNo->setAlignment( int( QLabel::AlignCenter ) );

LineEditAddress = new QLineEdit( groupBox1, "LineEditAddress" );
LineEditAddress->setGeometry( QRect( 73, 63, 364, 22 ) );

LineEditEtc = new QLineEdit( groupBox1, "LineEditEtc" );
LineEditEtc->setGeometry( QRect( 73, 102, 365, 46 ) );

PushButtonDel = new QPushButton( this, "PushButtonDel" );
PushButtonDel->setGeometry( QRect( 517, 343, 61, 30 ) );
PushButtonDel->setText( tr( QString::fromUtf8( "✖️" ) ) );

PushButtonSave = new QPushButton( this, "PushButtonSave" );
PushButtonSave->setGeometry( QRect( 441, 343, 61, 30 ) );
PushButtonSave->setText( tr( QString::fromUtf8( "💾️" ) ) );

PushButtonHelp = new QPushButton( this, "PushButtonHelp" );
PushButtonHelp->setGeometry( QRect( 12, 344, 80, 28 ) );
PushButtonHelp->setText( tr( QString::fromUtf8( "🆘️" ) ) );

// tab order
setTabOrder( listView, LineEditName );
setTabOrder( LineEditName, LineEditPhoneNo );
setTabOrder( LineEditPhoneNo, LineEditAddress );
setTabOrder( LineEditAddress, LineEditEtc );
setTabOrder( LineEditEtc, PushButtonSave );
setTabOrder( PushButtonSave, PushButtonDel );
setTabOrder( PushButtonDel, PushButtonHelp );
}

/*
 * Destroys the object and frees any allocated resources
 */
AddBookDLG::~AddBookDLG()
{
    // no need to delete child widgets, Qt does it all for us
}

```

이제 `progen` 과 `tmake` 를 이용해서 `project` 파일과 `Make` 파일을 생성해보자.

`progen` 의 사용법은 어렵지 않다.

```
$ prigen -o Addressbook.pro Addressbook.cpp Addressbook.h
```

`progen` 은 `-o` 옵션뒤에 원하는 프로젝트 명을 쓰고 프로젝트에 들어가는 파일을 나열하면 된다. 그러면 `Addressbook.pro` 라는 파일이 생성된다.

프로젝트파일(Addressbook.pro)

```

TEMPLATE       = app
CONFIG         = qt warn_on_release
HEADERS       = Addressbook.h
SOURCES       = Addressbook.cpp
INTERFACES    =

```


tmake 프로그램도 progen 과 마찬가지로 -o 옵션을 이용해서 Makefile 을 생성 하면 된다.

```
$ tmake -o Makefile Addressbook.pro
```

Makefile 파일

```
#####  
# Makefile for building Addressbook  
# Generated by tmake at 21:00, 2002/01/23  
#   Project: Addressbook  
#   Template: app  
#####  
  
##### Compiler, tools and options  
  
CC      =      gcc  
CXX     =      g++  
CFLAGS  =      -pipe -Wall -W -O2 -DNO_DEBUG  
CXXFLAGS = -pipe -Wall -W -O2 -DNO_DEBUG  
INCPATH =      -I$(QTDIR)/include  
LINK    =      g++  
LFLAGS  =  
LIBS    =      $(SUBLIBS) -L$(QTDIR)/lib -L/usr/X11R6/lib -lqt -lXext -lX11 -lm  
MOC     =      $(QTDIR)/bin/moc  
UIC     =      $(QTDIR)/bin/uic  
  
TAR     =      tar -cf  
GZIP    =      gzip -9f  
  
##### Files  
  
HEADERS =      Addressbook.h  
SOURCES =      Addressbook.cpp  
OBJECTS =      Addressbook.o  
INTERFACES =  
UICDECLS =  
UICIMPLS =  
SRCMOC   =      moc_Addressbook.cpp  
OBJMOC   =      moc_Addressbook.o  
DIST     =  
TARGET   =      Addressbook  
INTERFACE_DECL_PATH = .  
  
##### Implicit rules  
  
.SUFFIXES: .cpp .cxx .cc .C .c  
  
.cpp.o:  
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $@ $<  
  
.cxx.o:  
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $@ $<  
  
.cc.o:
```

```

$(CXX) -c $(CXXFLAGS) $(INCPATH) -o $$@ $$<

.C.o:
$(CXX) -c $(CXXFLAGS) $(INCPATH) -o $$@ $$<

.c.o:
$(CC) -c $(CFLAGS) $(INCPATH) -o $$@ $$<

##### Build rules

all: $(TARGET)

$(TARGET): $(UICDECLS) $(OBJECTS) $(OBJMOC)
$(LINK) $(LFLAGS) -o $(TARGET) $(OBJECTS) $(OBJMOC) $(LIBS)

moc: $(SRCMOC)

tmake: Makefile

Makefile: Addressbook.pro
tmake Addressbook.pro -o Makefile

dist:
$(TAR) Addressbook.tar Addressbook.pro $(SOURCES) $(HEADERS) $(INTERFACES)
$(DIST)
$(GZIP) Addressbook.tar

clean:
-rm -f $(OBJECTS) $(OBJMOC) $(SRCMOC) $(UICIMPLS) $(UICDECLS) $(TARGET)
-rm -f *~ core

##### Sub-libraries

##### Combined headers

##### Compile

Addressbook.o: Addressbook.cpp \
Addressbook.h

moc_Addressbook.o: moc_Addressbook.cpp \
Addressbook.h

moc_Addressbook.cpp: Addressbook.h
$(MOC) Addressbook.h -o moc_Addressbook.cpp

```

이제 **Makefile** 까지 완성되었다.

이제 컴파일을 해보자. 컴파일은 그냥

\$ make

라고 하면 된다. 컴파일이 성공하면 아무런 메시지가 남지않고 컴파일 과정이 종료되고 문제가 있으면 에러 메시지를 표시하고 종료한다.

컴파일했더니 다음과 같은 메시지를 출력되었다.

```
g++ -o Addressbook Addressbook.o moc_Addressbook.o -L/usr/lib/qt-2.3.1/lib -L/usr/X11R6/lib -lqt -lXext -lX11 -lm
/usr/lib/gcc-lib/i386-redhat-linux/2.96/../../../../crt1.o: In function `_start':
/usr/lib/gcc-lib/i386-redhat-linux/2.96/../../../../crt1.o(.text+0x18):
undefined reference to `main'
collect2: ld returned 1 exit status
make: *** [Addressbook] 오류 1
```

왜일까? 만들어진 소스파일을 보면서 이유를 생각해 보자.

소스 코드를 보면 알겠지만 소스 코드에는 `main()` 함수가 선언되어 있지 않다. 그래서 “undefined reference to `main'”라는 에러 메시지가 나온 것이다.

이제부터는 손으로 직접 하나씩 코딩을 해야 한다.

Qt designer(2.xx 버전에서)가 해주는 것은 여기까지이다.

우선 `main` 함수를 추가해 보자. `main` 함수의 형태는 예전에 배운 [“Hello, Qt!” 프로그램](#)과 같다.

프로그램의 처음부분에 다음의 라인을 추가한다.

```
#include <qapplication.h>
```

그리고 프로그램의 제일 끝에 다음의 라인을 추가한다.

```
int main(int argc, char** argv)
{
    QApplication QApp(argc, argv);

    AddBookDLG AddDlg;
    QApp.setMainWidget(&AddDlg);
    AddDlg.show();

    return QApp.exec();
}
```

`make` 를 실행하면 컴파일이 된다.

이제 프로그램을 실행해 보자.

프로그램을 실행시키면 다음과 같은 화면이 나온다.

그런데 무언가 이상하지 않는가?

화면을 보면 한글로 작성한 부분이 모두 빠져있다.

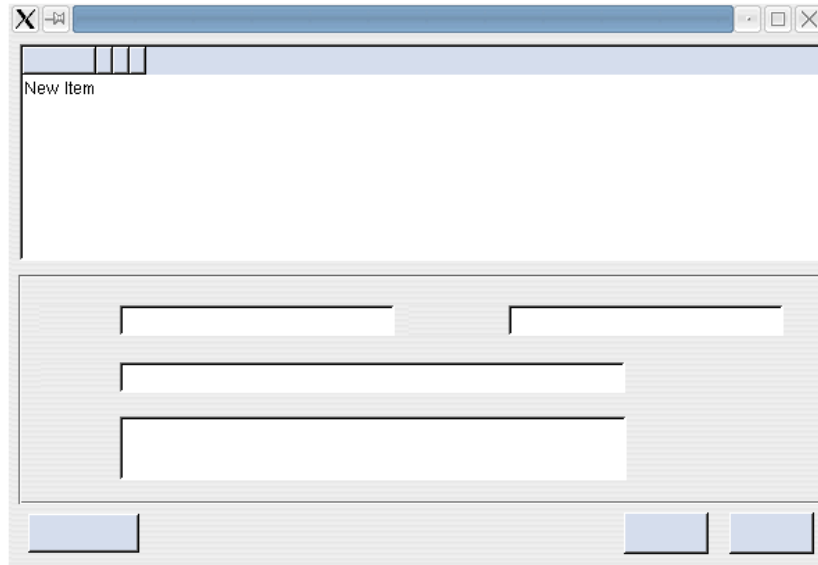


Figure 25. Addressbook Beta1

Qt 에서 한글 사용하기

몇년 전까지 X window 에서 한글을 보기 위해서는 HanX 라는 것을 따로 실행했었고 한글을 입력하는 것은 매우 힘들었다. 각 프로그램마다 한글판이 따로 나와서 “한글 패치된...”이라는 말을 쉽게 볼 수 있었다. 하지만 근래에 들어서는 I18n 이 보편화 되면서 한글의 입력과 출력이 보다 쉽게 되었다.

이제 Qt 를 이용해서 한글을 입력하고 출력하는 방법에 대해서 알아보도록 하자.

한글을 출력하기 위해서

```
QString::fromLocal8Bit(“출력을 원하는 문자”)
```

로 하게 되면 한글을 쉽게 출력할 수 있다.

보통

```
#define kor(str) QString::fromLocal8Bit(str)
```

로 미리 정의해 놓고 사용하기도 한다.

Qt 에서 한글을 입력받기 위해서는 출력하는 것에 비해서 약간의 절차가 더 필요하다.

우선 입력받는 것은 보통 입력하는 방법과 같다.

예를 들어 QLineEdit* LineEdit 라고 선언된 한 줄의 문자를 입력받는 것을 가정해보자.

일반적으로 QLineEdit 에 한글을 입력한 후 결과값²³을 보면 NULL 이거나 우리가 원하는 값으로 들어가 있지 않는 결과가 많다.

이럴 경우 한번의 변환 과정을 거쳐야 하는데

```
LineEdit->text().local8Bit()
```

로 변환해 주어야 정확한 값으로 인식하게 된다.

²³ QLineEdit->text().ascii()

Qt 는 2.xx 大부터 유니코드를 지원하므로 유니코드를 이용해서 표현해도 된다.

그럼 우선 필요한 한글을 볼 수 있게 해보자.

Qt designer 에서는 일반적으로 한글을 입력하고 uic 를 이용해서 소스코드를 만들어보면 소스코드 상에서는 한글로 나타내는 것이 아니라 다른 문자셋으로 변경한다. 앞의 소스코드를 보면 우리가 입력한 “성명”은 “?깁아_”와 같은 문자로 변경되었다.

화면에 한글이 제대로 나오게 하려면

```
ListView->addColumn( tr( QString::fromUtf8( "?깁아_" ) ) );
```

위의 코드에서 tr()²⁴만 제거하면 한글이 제대로 보이게 된다.

```
ListView->addColumn( QString::fromUtf8( "?깁아_" ) );
```

한글로 입력해서 보이게 하려면 앞에서 설명한 방법을 이용하면 된다.

```
ListView->addColumn( QString::fromLocal8Bit( "성명" ) );
```

소스코드를 수정하고 다시 컴파일해보면 한글이 제대로 보일 것이다.

이제 마지막 작업을 하겠다.

소스코드에 우리가 원하는 행동들을 추가할 차례이다. 저장 버튼을 누르면 저장이 되고, Listview 에서 항목을 선택한 후 삭제 버튼을 누르면 삭제할 수 있는 그런 기능을 추가할 것이다. 그리고 간단한 About 수준의 도움말도 제공할 것이다.

우선 Addressbook.h 파일에 다음과 같은 것을 Class 의 아랫 부분에 선언하자.

```
public slots:
    void pressSave();
    void pressDel();
    void pressHelp();
```

Addressbook.h 파일에 Slot 을 public 의 형태로 추가하였다.

3 개의 Slot 은 3 개의 버튼(PushButtonSave, PushButtonDel, PushButtonHelp)을 위한 Slot 이다.

이제 Addressbook.cpp 파일에 함수들을 추가해보자.

```
void AddBookDLG::pressSave()
{
    return;
}

void AddBookDLG::pressDel()
{
    return;
}

void AddBookDLG::pressHelp()
```

²⁴ Qt 의 tr()은 I18N 을 위해서 사용된다. 자세한 설명은 Trolltech 홈페이지를 참조하자.

```
{
    return;
}
```

3 개의 함수가 추가되었다.

컴파일해 보면 아무런 에러가 없이 컴파일되는 것을 볼 수 있다. 프로그램을 실행시켜서 버튼을 클릭해보자. 어떤 결과가 나타나는가? 아무런 결과도 나타나지 않는다.

이제 버튼을 누르면 원하는 결과가 나타나도록 각 함수에 내용을 채워보자.

첫번째의 함수인 Save 버튼을 클릭했을 때의 행동을 정의해보자. “저장 버튼을 누르면 4 개의 QLineEdit 의 값을 읽어서 ListView 에 항목을 추가한다.”

우선 버튼을 누르면 pressSave() 함수가 동작하도록 해보자. Signal/Slot 을 연결하려면 어떤 함수가 쓰여야 하는가? 앞의 예에서 connect() 를 사용한 것을 알고 있을 것이다. 마우스의 클릭에 함수가 동작하도록 다음과 같은 것을 생성자(AddBookDLG::AddBookDLG() 함수) 의 아랫부분에 추가하자.

```
QObject::connect(PushButtonSave, SIGNAL( clicked() ), SLOT( pressSave() ));
```

이제 함수의 내부에 필요한 코드(내용)을 채울 시간이다. QLineEdit 의 내용(문자)를 읽어오는 방법은 text() 메소드를 사용하면 된다. QLineEditName->text()의 방법을 이용하면 QLineEditName 의 값을 가져올 수 있다. 가져오는 값에 한글을 사용하기 위해서 QLineEditName->text().local8Bit()을 이용하면 한글로 값을 읽어올 수 있다.

이제 ListView 에 새로운 항목(Item)을 추가해보자. ListView 에 항목을 추가하기 위해서는 QListWidgetItem 을 사용한다. QListWidgetItem 을 ListView 에 새로 생성하는 방법으로 항목을 추가하면 된다.

- 항목을 추가하는 방법으로는 다음과 같은 두가지 방법이 있다.
- 처음 방법은 처음부터 값을 입력하면서 항목을 생성하는 방법
 - 두번째 방법은 항목을 먼저 만들고 나중에 값을 추가하는 방법

첫번째 방법	<pre>QListWidgetItem* Item = new QListWidgetItem(ListView, kor(LineEditName->text().local8Bit()), kor(LineEditAddress->text().local8Bit()), kor(LineEditPhoneNo->text().local8Bit()), kor(LineEditEtc->text().local8Bit()));</pre>
두번째 방법	<pre>QListWidgetItem* Item = new QListWidgetItem(ListView, 0); Item->setText(0, kor(LineEditName->text().local8Bit())); Item->setText(1, kor(LineEditAddress->text().local8Bit())); Item->setText(2, kor(LineEditPhoneNo->text().local8Bit())); Item->setText(3, kor(LineEditEtc->text().local8Bit()));</pre>

우리는 여기서 첫번째의 방법을 사용하도록 하겠다.

두번째로 Delete 버튼을 클릭했을 때의 행동을 정의해보자.
 “삭제버튼을 누르면 ListView 에서 어떤 항목(Item)이 선택되었는지를 알아온 후 그 항목을 제거한다.”

처음과 마찬가지로 삭제버튼을 눌렀을 때 pressDel()함수가 작동하게 해보자.
 다음을 라인을 생성자의 아랫부분에 추가하자.

```
QObject::connect(PushButtonDel, SIGNAL( clicked() ), SLOT( pressDel() ));
```

이제 함수의 내부를 구성할 순서이다.
 Listview 에서 현재 선택된 항목을 알아오기 위해서는 다음의 메소드를 사용하면 된다. currentItem()을 사용하면 현재의 ListView 에 어떤 항목(Item)이 선택되어 있는지 알 수 있다.
 항목을 삭제하기 위해서는 Save 에서 사용한 new 의 반대인 delete 를 사용하면 된다.

```
delete(ListView->currentItem());
```

마지막으로 간단한 도움말을 만들어보자.
 도움말 버튼을 눌렀을 때 pressHelp()함수를 작동시키려면 마찬가지로 connect()함수를 이용하면 된다.

```
QObject::connect(PushButtonHelp, SIGNAL( clicked() ), SLOT( pressHelp() ));
```

도움말을 어떤 식으로 구성하면 좋을까?
 우리는 여기서 MessageBox 라는 것을 이용해 간단한 도움만을 만들어 보도록 하겠다. MessageBox 는 간단한 버튼이 하나 있는 정보를 나타내는 창 (Information), 버튼이 2 개 있는 사용자의 의향을 물어보는(보통 ‘Yes’/’No’) 창(warning)으로 사용할 수 있다.(MessageBox 에는 두가지 말고도 다른 메소드도 있다.)

우리는 간단한 도움말을 나타내기 위해서 다음과 같은 코드를 사용하겠다.

```
QMessageBox::information(this, kor("만든사람"), kor("전화번호부를 이용해 주셔서 감사합니다."));
```

도움말은 다음과 같은 창으로 표시된다.

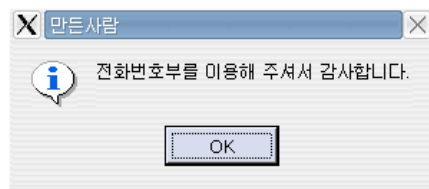


Figure 26. Help

위의 코드를 사용하기 위해서는 프로그램 윗쪽에 QMessageBox 에 대한 Header(.h)파일을 포함하고 한글을 사용하기 위해 kor()을 정의해야 한다.

```
#include <qmessagebox.h>
#define kor(str) QString::fromLocal8Bit(str)
```

다시 컴파일한 후에 프로그램을 실행시켜 보면 항목이 추가/삭제되는 것을 볼 수 있을 것이다.

하지만 약간의 불편한 점이 있다. 항목이 추가/삭제 후 QLineEdit 글자가 그대로 남아있어서 사용자가 직접 글자들을 지워야 하는 점이다.

이제 QLineEdit의 글자를 자동으로 지우게 해보자. QLineEdit의 내용을 지우기 위해서는 clear() 메소드를 사용하면 된다.

다음과 같은 항목을 함수에 추가해 주면 된다.

```
LineEditName->clear();
LineEditAddress->clear();
LineEditPhoneNo->clear();
LineEditEtc->clear();
```

이제 프로그램이 완성되었다.

소스코드를 다시 컴파일해서 프로그램을 실행시키면 된다.

전체 소스파일의 내용은 다음과 같다.

소스파일(Addressbook.h)

```

/*****
** Form interface generated from reading ui file 'Addressbook.ui'
**
** Created: Wed Jan 23 21:19:04 2002
**    by: The User Interface Compiler (uic)
**
** WARNING! All changes made in this file will be lost!
*****/
#ifndef ADDRESSBOOKFORM_H
#define ADDRESSBOOKFORM_H

#include <qvariant.h>
#include <qwidget.h>
class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;
class QGroupBox;
class QLabel;
class QLineEdit;
class QListView;
class QListViewItem;
class QPushButton;

class AddBookDLG : public QWidget
{
    Q_OBJECT

public:
    AddBookDLG( QWidget* parent = 0, const char* name = 0, WFlags fl = 0 );
    ~AddBookDLG();

    QListView* ListView;
    QGroupBox* GroupBox1;
    QLabel* TextLabelAddress;

```



```

QLabel* TextLabelEtc;
QLineEdit* LineEditPhoneNo;
QLineEdit* LineEditName;
QLabel* TextLabelName;
QLabel* TextLabelPhoneNo;
QLineEdit* LineEditAddress;
QLineEdit* LineEditEtc;
QPushButton* PushButtonDel;
QPushButton* PushButtonSave;
QPushButton* PushButtonHelp;

public slots:
    void pressSave();
    void pressDel();
    void pressHelp();
};
#endif // ADDRESSBOOKFORM_H

```

소스파일(Addressbook.cpp)

```

/*****
** Form implementation generated from reading ui file 'Addressbook.ui'
**
** Created: Wed Jan 23 21:19:14 2002
**    by: The User Interface Compiler (uic)
**
** WARNING! All changes made in this file will be lost!
*****/
#include "Addressbook.h"

#include <qapplication.h>
#include <qgroupbox.h>
#include <qheader.h>
#include <qlabel.h>
#include <qlineedit.h>
#include <qlistview.h>
#include <qpushbutton.h>
#include <qlayout.h>
#include <qvariant.h>
#include <qtooltip.h>
#include <qwhatsthis.h>
#include <qimage.h>
#include <qpixmap.h>
#include <qmessagebox.h>

#define kor(str)      QString::fromLocal8Bit(str)

/*
 * Constructs a AddBookDLG which is a child of 'parent', with the
 * name 'name' and widget flags set to 'f'
 */
AddBookDLG::AddBookDLG( QWidget* parent, const char* name, WFlags fl )
    : QWidget( parent, name, fl )

```

```

{
    if ( !name )
        setName( "AddressBookForm" );
    resize( 584, 378 );
    setCaption( QString::fromUtf8( "二솟?濡? ) );

    ListView = new QListView( this, "ListView" );
    ListView->addColumn( QString::fromUtf8( "?깁아_" ) );
    ListView->addColumn( QString::fromUtf8( "?__???? ) );
    ListView->addColumn( QString::fromUtf8( "二솟?" ) );
    ListView->addColumn( QString::fromUtf8( "籛__怨?" ) );
    QListViewItem* item = new QListViewItem( ListView, 0 );
    item->setText( 0, tr( "New Item" ) );

    ListView->setGeometry( QRect( 6, 6, 582, 156 ) );

    GroupBox1 = new QGroupBox( this, "GroupBox1" );
    GroupBox1->setGeometry( QRect( 5, 172, 582, 165 ) );
    GroupBox1->setTitle( QString::fromUtf8( "媛????낫" ) );

    TextLabelAddress = new QLabel( GroupBox1, "TextLabelAddress" );
    TextLabelAddress->setGeometry( QRect( 16, 63, 58, 19 ) );
    TextLabelAddress->setText( QString::fromUtf8( "二솟?" ) );
    TextLabelAddress->setAlignment( int( QLabel::AlignCenter ) );

    TextLabelEtc = new QLabel( GroupBox1, "TextLabelEtc" );
    TextLabelEtc->setGeometry( QRect( 16, 104, 58, 19 ) );
    TextLabelEtc->setText( QString::fromUtf8( "籛__怨?" ) );
    TextLabelEtc->setAlignment( int( QLabel::AlignCenter ) );

    LineEditPhoneNo = new QLineEdit( GroupBox1, "LineEditPhoneNo" );
    LineEditPhoneNo->setGeometry( QRect( 353, 22, 198, 22 ) );

    LineEditName = new QLineEdit( GroupBox1, "LineEditName" );
    LineEditName->setGeometry( QRect( 73, 22, 198, 22 ) );

    TextLabelName = new QLabel( GroupBox1, "TextLabelName" );
    TextLabelName->setGeometry( QRect( 15, 22, 58, 19 ) );
    TextLabelName->setText( QString::fromUtf8( "?깁아_" ) );
    TextLabelName->setAlignment( int( QLabel::AlignCenter ) );

    TextLabelPhoneNo = new QLabel( GroupBox1, "TextLabelPhoneNo" );
    TextLabelPhoneNo->setGeometry( QRect( 281, 22, 72, 22 ) );
    TextLabelPhoneNo->setText( QString::fromUtf8( "?__???? ) );
    TextLabelPhoneNo->setAlignment( int( QLabel::AlignCenter ) );

    LineEditAddress = new QLineEdit( GroupBox1, "LineEditAddress" );
    LineEditAddress->setGeometry( QRect( 73, 63, 364, 22 ) );

    LineEditEtc = new QLineEdit( GroupBox1, "LineEditEtc" );
    LineEditEtc->setGeometry( QRect( 73, 102, 365, 46 ) );

    PushButtonDel = new QPushButton( this, "PushButtonDel" );
    PushButtonDel->setGeometry( QRect( 517, 343, 61, 30 ) );

```

```

PushButtonDel->setText( QString::fromUtf8( "???" ) );

PushButtonSave = new QPushButton( this, "PushButtonSave" );
PushButtonSave->setGeometry( QRect( 441, 343, 61, 30 ) );
PushButtonSave->setText( QString::fromUtf8( "?☑?" ) );

PushButtonHelp = new QPushButton( this, "PushButtonHelp" );
PushButtonHelp->setGeometry( QRect( 12, 344, 80, 28 ) );
PushButtonHelp->setText( QString::fromUtf8( "?__?☑" ) );

// tab order
setTabOrder( ListView, LineEditName );
setTabOrder( LineEditName, LineEditPhoneNo );
setTabOrder( LineEditPhoneNo, LineEditAddress );
setTabOrder( LineEditAddress, LineEditEtc );
setTabOrder( LineEditEtc, PushButtonSave );
setTabOrder( PushButtonSave, PushButtonDel );
setTabOrder( PushButtonDel, PushButtonHelp );

QObject::connect( PushButtonSave, SIGNAL( clicked() ), SLOT( pressSave() ) );
QObject::connect( PushButtonDel, SIGNAL( clicked() ), SLOT( pressDel() ) );
QObject::connect( PushButtonHelp, SIGNAL( clicked() ), SLOT( pressHelp() ) );
}

/*
 * Destroys the object and frees any allocated resources
 */
AddBookDLG::~AddBookDLG()
{
    // no need to delete child widgets, Qt does it all for us
}

void AddBookDLG::pressSave()
{
    QListViewItem* Item = new QListViewItem( ListView,
        kor(LineEditName->text().local8Bit() ),
        kor(LineEditAddress->text().local8Bit() ),
        kor(LineEditPhoneNo->text().local8Bit() ),
        kor(LineEditEtc->text().local8Bit() ) );

    LineEditName->clear();
    LineEditAddress->clear();
    LineEditPhoneNo->clear();
    LineEditEtc->clear();

    return;
}

void AddBookDLG::pressDel()
{
    delete(ListView->currentItem());

    LineEditName->clear();
    LineEditAddress->clear();
}

```

```

        LineEditPhoneNo->clear();

        return;
    }

void AddBookDLG::pressHelp()
{
    QMessageBox::information(this, kor("만든사람"), kor("전화번호부를 이용해 주셔서 감사합니다."));

    return;
}

int main(int argc, char** argv)
{
    QApplication QApp(argc, argv);

    AddBookDLG AddDlg;
    QApp.setMainWidget(&AddDlg);
    AddDlg.show();

    return QApp.exec();
}

```

4 장 Linux 에서의 멀티미디어 프로그래밍

앞 장에서는 Qt 를 가지고 작은 Application 을 개발해 보았다. 이번 장에서는 리눅스에서의 멀티미디어 환경에 대해서 알아보고 간단한 프로그래밍을 해보자.

Linux 용 Multimedia 환경

- 사운드(Sound)

기본적으로 리눅스에서는 커널차원에서 사운드 드라이버를 지원한다. 하지만 커널에서 모든 사운드 카드의 드라이버를 지원하지 못하고 질적/기능적으로 떨어지기 때문에 다른 사운드 드라이버들이 존재하고 있다. 사운드 드라이버를 구분하면 크게 다음과 같이 3 가지로 구분할 수 있다.

1. Linux Kernel 과 함께 배포된 것(OSS/Free)
2. 상업용 드라이버 OSS
3. 무료로 배포되고 있는 ALSA

커널 표준 사운드 드라이버

커널 표준 드라이버는 OSS/Free 라고도 불리우며 4Front Technologies 사에서 OSS 개발을 하고 있는 Hanu Savolainen 씨가 대부분을 개발했다. 지금도 OSS 에서 무상 드라이버를 기부하여 이에 대응하는 사운드카드는 점점 늘어가고 있다.

OSS(Open Sound System)

미국의 4Front Technologies 라는 회사가 개발하여 배포하고 있는 사운드 드라이버로 자유롭게 사용할 수 있는 것과 상업용이 있다. 예전에 커널에서 표준으로 지원하는 카드의 종류가 적었을 때 상업용 배포판 회사들 중에는 OSS 드라이버를 번들하기도 하였다. 최근에는 커널 표준 드라이버에서도 많은 사운드카드를 지원하므로 OSS 드라이버를 번들하는 경우가 적어졌지만 다양한 기능을 필요로 하는 경우에 사용되고 있다.

OSS 드라이버는 커널 표준 드라이버와 호환성이 있어 커널 드라이버를 사용하도록 설계된 어플리케이션들은 OSS 드라이버에서도 작동할 수 있게 되어 있다.

설치 작업은 간단해서, 배포 패키지 자체가 Linux 실행파일(ELF 포맷)로 되어 있어서 이를 직접 실행하여 화면상의 표시에 따르면 된다.

ALSA(Advanced Linux Sound Architecture)²⁵

Jaroslav Kysela(체코 거주)가 메인 프로그래머로서 활동하고 있는 ALSA Project 에서 제작, 배포하고 있는 자유 Linux 용 사운드 드라이버이다. 상업용인 OSS 와 달리 GPL 에 의한 재배포가 가능하다.

커널 표준 드라이버나 OSS 드라이버와의 호환성이 있어, 이들 드라이버용으로 작성된 프로그램의 대부분이 ALSA 드라이버에서 그대로 사용할 수 있다.

인스톨 과정이 좀 복잡하지만 다른 사운드 드라이버에 비해 빠른 지원과 발전을 보여주고 있으므로 앞으로의 기대가 크다.(현재 안정화 0.5.xx / 개발자 0.9.xx 베타)

● 화상(Video)

리눅스에서 지원하는 화상캡처 장치들로는 TV 수신카드와 영상 Capture 장치들 그리고 USB 화상 CAM 으로 나눌 수 있다.

Video4Linux

- Kernel 에서 지원하는 기본 모듈
- TV 수신카드를 지원하기 위해 등장

Video4Linux2

Video4Linux 1.0 Version 이 TV 수신카드를 위해 등장했기 때문에 화상 Cam 에는 맞지 않기 때문에 USB Web Cam 을 위해서 등장

- 화면의 확대/축소 기능 지원

Linux 용 Sound Tester

이번 Chapter 에서는 리눅스에서 기본적으로 사용되는 OSS(Open Sound System) API(Application Programming Interface)²⁶를 이용하여 실시간으로 사운드의 입력과 출력을 동시에 할 수 있는 프로그램을 작성할 것이다.

²⁵ ALSA 드라이버는 디폴트상태(처음 부팅한 상태)에서 사운드카드의 모든 음량이 0(Mute)인 상태에 있다.

²⁶ 위의 API 에 관한 매뉴얼은 4Front Technologies 의 사이트에서 구할 수 있다.

기본적인 화면의 구성은 Qt designer 를 사용할 것이고 사운드의 입출력을 위해서는 입력을 처리하고 입력받은 사운드를 출력을 위해서 2 개의 thread 를 사용한다.

Thread 는 Qt 에서 제공하는 Thread 나 pthread 를 사용해도 되지만 우리는 프로그램을 좀 더 쉽게 작성하기 위해서 Qt 에서 제공해주는 Timer 를 사용하도록 하겠다.

화면의 구성

앞장에서 배운 Qt designer 를 이용해서 다음과 같이 구성한다.

기본적인 Class 명과 Type 은 다음과 같다.

Name	Type	Text
SoundPlayer	QWidget	
PlayLabel	QLabel	OFF
RecLabel	QLabel	OFF
StartButton	QPushButton	Start
CloseButton	QPushButton	Close

그림에서 Object Hierarchy 를 보면 알 수 있다.

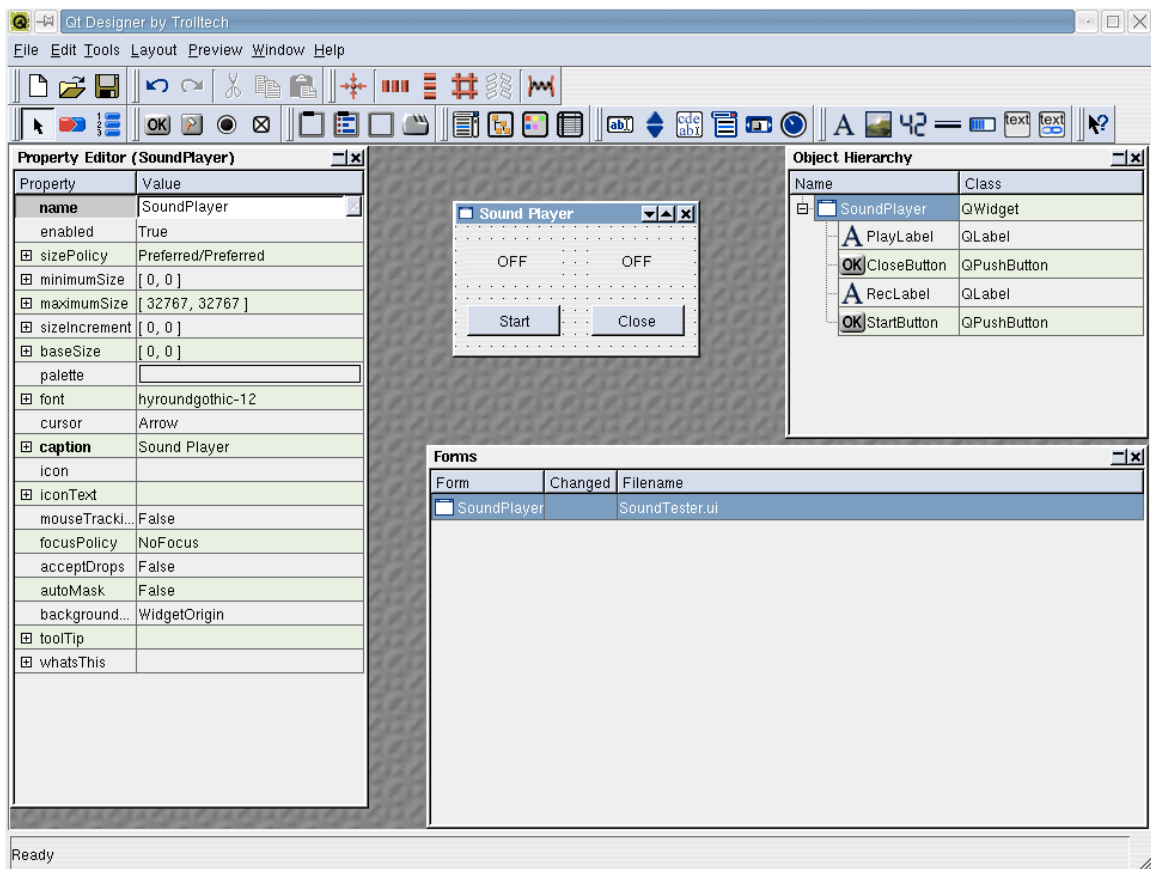


Figure 27. SoundTester UI

기본적인 Form의 입력은 다음과 같이 하면 된다.

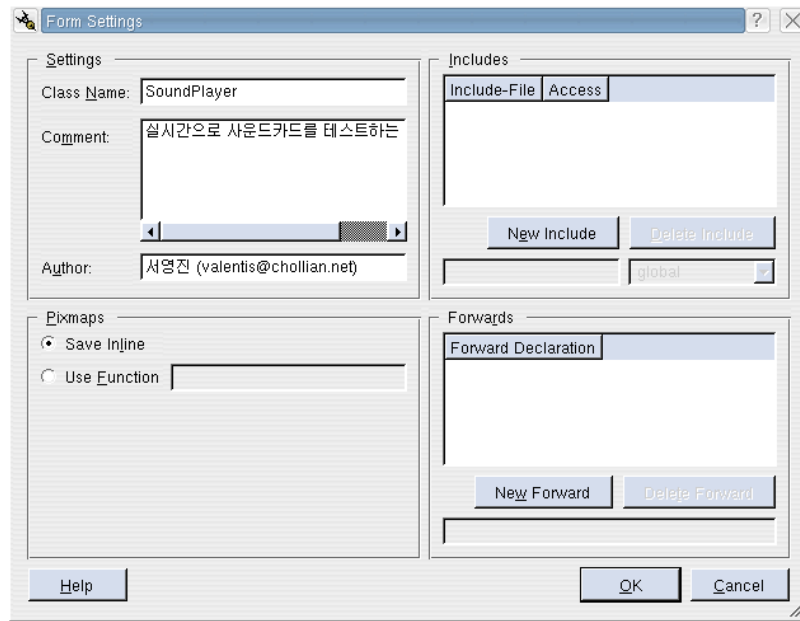


Figure 28. SoundTester Form

화면이 구성되었으면 저장을 하고 앞 장에서 사용한 uic를 이용해서 C++ 소스코드로 변경한다.(변경하는 방법은 3-3 절 참조)

그리고 나서 uic를 이용해서 만들어진 코드에 볼드체로 된 다음의 코드를 추가한다.

소스파일(SoundTester.h)

```

/*****
** Form interface generated from reading ui file 'SoundTester.ui'
**
** Created: Fri Nov 9 16:38:36 2001
**   by: The User Interface Compiler (uic)
**
** WARNING! All changes made in this file will be lost!
*****/
#ifndef SOUNDPLAYER_H
#define SOUNDPLAYER_H

#include <qvariant.h>
#include <qwidget.h>
#include <qtimer.h>

class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;
class QLabel;
class QPushButton;

class SoundPlayer : public QWidget

```

```

{
    Q_OBJECT

public:
    SoundPlayer( QWidget* parent = 0, const char* name = 0, WFlags fl = 0 );
    ~SoundPlayer();

    QLabel* RecLabel;
    QLabel* PlayLabel;
    QPushButton* StartButton;
    QPushButton* CloseButton;

    QTimer* PlayTimer;
    QTimer* RecordTimer;

    int fd;
    int caps;

    unsigned char* Buf;
    int count;
    bool mode;

public slots:
    virtual int Init();
    virtual int Record();
    virtual int Play();
};

#endif // SOUNDPLAYER_H

```

소스파일(SoundTester.cpp)

```

/*****
** Form implementation generated from reading ui file 'SoundTester.ui'
**
** Created: Fri Nov 9 16:38:48 2001
**      by: The User Interface Compiler (uic)
**
** WARNING! All changes made in this file will be lost!
*****/
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/soundcard.h>
#include <sys/types.h>
#include <stdlib.h>

#include "SoundTester.h"

#include <qapplication.h>

```



```

#include <qlabel.h>
#include <qpushbutton.h>
#include <qlayout.h>
#include <qvariant.h>
#include <qtooltip.h>
#include <qwhatsthis.h>
/*
 * Constructs a SoundPlayer which is a child of 'parent', with the
 * name 'name' and widget flags set to 'f'
 */
SoundPlayer::SoundPlayer( QWidget* parent, const char* name, WFlags fl )
    : QWidget( parent, name, fl )
{
    if ( !name )
        setName( "SoundPlayer" );
    resize( 207, 108 );
    setCaption( tr( "Sound Player" ) );

    PlayLabel = new QLabel( this, "PlayLabel" );
    PlayLabel->setGeometry( QRect( 112, 15, 82, 29 ) );
    PlayLabel->setText( tr( "OFF" ) );
    PlayLabel->setAlignment( int( QLabel::AlignCenter ) );

    CloseButton = new QPushButton( this, "CloseButton" );
    CloseButton->setGeometry( QRect( 113, 65, 80, 28 ) );
    CloseButton->setText( tr( "Close" ) );

    Reclabel = new QLabel( this, "Reclabel" );
    Reclabel->setGeometry( QRect( 8, 15, 82, 29 ) );
    Reclabel->setText( tr( "OFF" ) );
    Reclabel->setAlignment( int( QLabel::AlignCenter ) );

    StartButton = new QPushButton( this, "StartButton" );
    StartButton->setGeometry( QRect( 9, 65, 80, 28 ) );
    StartButton->setText( tr( "Start" ) );

    if( (fd = open("/dev/dsp", O_RDWR, 0)) == -1)
    {
        perror("OSS: error opening Device\n");
        return;
    }

    QObject::connect( StartButton, SIGNAL( clicked() ), SLOT( Init() ) );
    QObject::connect( CloseButton, SIGNAL( clicked() ), this, SLOT( close() ) );

    //////////////////////////////////// Initialize Here !!!
    Buf = (unsigned char*)malloc(sizeof(unsigned char *) * 128 );
    count = 512;
}

/*
 * Destroys the object and frees any allocated resources

```

```

*/
SoundPlayer::~SoundPlayer()
{
    // no need to delete child widgets, Qt does it all for us
    if(Buf) free(Buf);

    close(fd);

    return;
}

int SoundPlayer::Init()
{
    int format, rate, stereo;

    if(ioctl(fd, SNDCTL_DSP_SETDUPLEX, 0) == -1)
    {
        perror("SOUND_PCM_SETDUPLEX");
        return -1;
    }

    format = AFMT_S16_LE;
    if(ioctl(fd, SNDCTL_DSP_SETFMT, &format) == -1)
    {
        perror("SOUND_PCM_SETFMT");
        return -1;
    };

    stereo = 2;
    if(ioctl(fd, SNDCTL_DSP_CHANNELS, &stereo) == -1)
    {
        perror("SOUND_PCM_CHANNELS");
        return -1;
    };

    rate = 8000;
    if(ioctl(fd, SNDCTL_DSP_SPEED, &rate) == -1)
    {
        perror("SOUND_PCM_SPEED");
        return -1;
    };

    RecordTimer = new QTimer(this);
    RecordTimer->start(0);
    QObject::connect(RecordTimer, SIGNAL(timeout()), SLOT(Record()));

    PlayTimer = new QTimer(this);
    PlayTimer->start(0);
    QObject::connect(PlayTimer, SIGNAL(timeout()), SLOT(Play()));

    return 0;
}

```

```
int SoundPlayer::Record()
{
    int status;

    mode = FALSE;
    if((status = read(fd, Buf, count)) == -1)
    {
        perror("SOUND_READ_ERROR");
        return FALSE;
    }
    mode = TRUE;

    if(Buf)
        RecLabel->setText( "ON" );
    else
        RecLabel->setText( "OFF" );

    return status;
}

int SoundPlayer::Play()
{
    int status;

    if(!mode) return -1;

    if(Buf)
    {
        PlayLabel->setText( "ON" );
    } else {
        PlayLabel->setText( "OFF" );
        return -1;
    }

    if((status = write(fd, Buf, count)) == -1)
    {
        perror("SOUND_write_ERROR");
        return FALSE;
    }

    return status;
}

int main( int argc, char **argv )
{
    QApplication a( argc, argv );
    SoundPlayer* SoundTester = new SoundPlayer;

    a.setMainWidget( SoundTester );
    SoundTester->show();
    int result = a.exec();
}
```

```

return result;
}

```

사운드 프로그래밍

리눅스에서의 사운드 프로그래밍은 커널에서 제공해주는 API 를 이용해서 만든다.

리눅스 사운드에서 Digital Audio(“/dev/dsp”), mixer(“/dev/mixer”), MIDI, raw Music 그리고 synthesizer 와 같은 장치들을 사용한다.

기본적으로 사용되는 Device 장치들은 다음과 같다.

- /dev/dsp(Digital voice device)
마이크로 사운드를 녹음하고 스피커로 사운드를 출력하기 위해서 사용된다.
- /dev/mixer(mixer)
사운드 볼륨을 조절할 때 사용되며 마이크의 입력 방향 설정(마이크, Line 입력, CD 입력)을 바꿀 수도 있다.
- /dev/sequencer(Synthesizer)
사운드로 효과를 만드는데 사용된다. 미리 녹음된 소리(악기소리나 기타소리)를 합성해서 새로운 소리를 만든다.
- /dev/music
기본적으로는 “/dev/dsp”와 같다. 위의 차이는 알고리즘적으로 “/dev/dsp”가 8bit unsigned 선형(Linear) Encoding 을 사용하지만 “/dev/music”는 mu-law Encoding 을 사용한다.
- /dev/midi
MIDI 인터페이스를 이용해서 MIDI 장치들(Keyboards, synthesizers, stage props)을 연결하는데 사용된다.

우리는 여기서 /dev/dsp 를 이용해서 사운드를 입력 받아서 출력하는 Application 을 작성한다.

OSS API Basic

OSS API 들은 <soundcard.h>라는 헤더파일에 정의되어 있다.

위의 헤더파일은 보통 “/usr/include/sys”에 위치하고 있다.

OSS 에서 장치의 값을 설정하고 읽어오는 방법은 ioctl()함수를 이용한다.

우리는 “/dev/dsp”를 이용해서 프로그래밍 할 것이므로 “/dev/mixer”나 기타 다른 장치는 여기서 다루지 않는다. 위의 장치들은 OSS API 문서를 참조하라.

Audio Programming

디지털 오디오(Digital Audio)는 컴퓨터 외부로 소리를 표현하는 일반적인 방법이다.

오디오 프로그래밍은 일반적으로 다음과 같은 방법으로 한다.

1. 오디오 장치를 `open()`한다.
2. 환경을 설정한다.
3. 입출력을 위한 메모리(버퍼)를 설정한다.
4. Read/Write 작업
5. 사용이 끝난 오디오 장치를 `close()`한다.

우선 “/dev/dsp”를 사용하기 위해서는 `open()`²⁷을 이용해서 장치를 열어야 한다.

사운드장치 열기(Opening the Sound Device)

`open()`을 사용할 때 flag 의 값은 `O_RDONLY`, `O_WRONLY` 그리고 `O_RDWR` 세 종류를 사용할 수 있다.

- `O_RDONLY`: 사운드 장치(보통 마이크)로부터 입력을 받을 때 사용한다.
- `O_WRONLY`: 사운드 장치(보통 스피커)로 사운드를 출력할 때 사용한다.
- `O_RDWR`: 사운드 장치²⁸에서 사운드를 입력과 출력을 동시에 할 경우 사용된다.

```
if( (fd = open("/dev/dsp", O_RDWR, 0)) == -1)
{
    perror("OSS: error opening Device\n");
    return;
}
```

사운드 장치를 사용할 수 없을 경우 `-1` 을 반환한다. 만약 다른 프로그램에서 사운드 장치를 사용하고 있을 경우 “EBUSY”라고 에러값을 세팅한다.²⁹

샘플링 환경 세팅하기(Setting Sampling parameter)

사운드 장치를 `open` 한 후에는 사운드 입출력하기 전에 기본적인 세팅이 필요하다. 사운드 음질에 영향을 미치는 다음의 기본적인 세가지 Parameter 가 존재한다.

1. Sample format(number of bits)
2. Channel 의 수(mono 또는 stereo)
3. Sampling rate(speed)

위의 설정은 순서대로 해야 한다. 아래의 설정을 위의 설정보다 먼저하면 설정이 되지 않는다.

²⁷ `open()`, `close()`, `read()`, `write()`에 대해서 모르는 분들은 Linux System 프로그래밍에 관련된 책을 먼저 공부하라. 개인적으로 Stevens 의 “Advanced Programming in the UNIX Environment”를 추천한다.

²⁸ 리눅스에서 Full Duplex(동시에 입출력이 가능)를 지원하는 장치의 경우에만 사용가능하다.

²⁹ 하지만 실제 프로그래밍 해보면 위의 경우 사운드 장치가 사용가능할 때까지 (사운드 장치를 사용하는 프로그램의 종료될 때까지) 프로그램의 동작을 멈추는 경우가 많다. 이러한 경우 스레드를 이용해서 몇 초 동안 동작이 멈춰있으면 프로그램의 동작을 결정하는 것이 좋다.

오디오 포맷 설정하기(Selecting Audio Format)

Sample format 은 음질에 영향을 미치는 중요한 변수(parameter)이다. Sample format 에는 여러가지가 있지만 우리는 여기서 AFMT_S16_LE 를 사용하겠다. AFMT_S16_LE 는 Signed 16Bit Little Endian 을 뜻한다. 이 포맷은 일반적인 PC 의 사운드카드에서 사용되고 있다.

사운드카드를 세팅하기 위해서는 앞에서 설명한 ioctl()에 매개변수로 SNDCTL_DSP_SETFMT 를 사용한다.

```
format = AFMT_S16_LE;
if(ioctl(fd, SNDCTL_DSP_SETFMT, &format) == -1)
{
    perror("SOUND_PCM_SETFMT");
    return -1;
};
```

채널의 수 설정하기(Selecting the Number of Channels)

현재 사용되는 대부분의 오디오 장치들은 Stereo 모드를 지원한다. 기본적으로 설정되는 모드는 Mono 이다. ioctl()에 SNDCTL_DSP_CHANNELS 를 사용해서 채널의 수를 선택할 수 있다.

Mono 의 경우 값이 1 이고 Stereo 는 2 이다.

```
stereo = 2; /* 1=mono, 2=stereo */
if(ioctl(fd, SNDCTL_DSP_CHANNELS, &stereo) == -1)
{
    perror("SOUND_PCM_CHANNELS");
    return -1;
};
```

Sampling Rate 설정하기(Selecting Sampling Rate)

Sampling Rate 는 소리의 질을 결정하는 파라미터이다. OSS API 는 1Hz 에서 2GHz 의 대역을 지원한다. 기본적으로 지원되는 Sampling rate 는 8kHz 이다. 전화기(음성)에서 기본적으로 사용하는 것이 8kHz 이다. CD 의 경우 44.1kHz, DVD 의 경우 96kHz 의 값을 가지고 있다.

우리는 음성을 이용해서 녹음/재생을 할 것이므로 8kHz 를 이용하겠다. 물론 이 부분은 사용자 마음대로 아무런 바뀌도 문제없다.

ioctl()에 SNDCTL_DSP_SPEED 를 이용하면 Sampling Rate 를 선택할 수 있다.

```
Rate = 8000;
if(ioctl(fd, SNDCTL_DSP_SPEED, &rate) == -1)
{
    perror("SOUND_PCM_SPEED");
    return -1;
};
```

Full Duplex 설정하기

많은 장치들이 half duplex 이다. Half duplex 의 경우 Recording 이나 Playing 이 동시에 되지 않는다. 오직 한 번에 한가지의 일을 할 수 있다.

물론 다중채널이 지원되는 경우 Recording 과 Playing 을 각각 다른 채널을 이용해서 하면 된다.

단일 채널에 Full Duplex 를 지원하기 위해서는 사운드 장치를 설치하기에 앞서서 Full Duplex 로 설정해야 한다.

설정은 다음과 같이 하면 된다.

```
if(ioctl(fd, SNDCTL_DSP_SETDUPLEX, 0) == -1)
{
    perror("SOUND_PCM_SETDUPLEX");
    return -1;
}
```

사운드 입/출력

일반적으로 리눅스에서는 장치를 파일의 개념으로 사용한다.

사운드를 입력받기 위해서는 파일을 읽는 것과 마찬가지로 read()를 사용한다.

```
if((status = read(fd, Buf, count)) == -1)
{
    perror("SOUND_READ_ERROR");
    return FALSE;
}
```

마찬가지로 사운드를 출력하기 위해서는 write()를 사용한다.

```
if((status = write(fd, Buf, count)) == -1)
{
    perror("SOUND_write_ERROR");
    return FALSE;
}
```

사운드에 관련된 기본적인 동작은 거의 알아보았다. 더 자세한 사항을 알고 싶은 분들은 OSS API 문서를 참조하라.

동시에 여러가지 일하기

여러 개의 일을 동시에 실행시키기 위해서는 thread 나 멀티 process 를 사용해야 한다. 하지만 이 것들은 세팅은 복잡하고 초보자들이 쉽게 이해하기 어렵다. 그래서 여기서는 Qt 에서 기본적으로 제공하는 QTimer 를 사용하도록 한다.

QTimer 는 주기적으로 어떤 일을 반복적으로 시키기 위해서 사용된다.

주기는 milliseconds 로 세팅하는데 시작할 때 start()의 매개변수로 이용해서 사용하면 된다. Timer 의 시작은 start()를 사용하고 종료는 stop()를 사용한다.

Timer 와 시킬 일(function)과의 연결은 connect()를 사용하면 된다.

```
RecordTimer = new QTimer(this);
RecordTimer->start(0);
QObject::connect(RecordTimer, SIGNAL(timeout()), SLOT(Record()));

PlayTimer = new QTimer(this);
PlayTimer->start(0);
QObject::connect(PlayTimer, SIGNAL(timeout()), SLOT(Play()));
```

이 프로그램은 연습용으로 프로그래밍했기 때문에 음질개선이나 소리 지연 같은 문제를 해결하지 않았다.

참고서적 및 문헌

The X Window Programming And Application With Xt (Douglas A. Young, Prentice Hall)

Tcl and The Tk Toolkit(John K. Ousterhout)

Practical Programming in Tcl and Tk 2nd Ed.(Brent B. Welch, Prentice Hall)

Professional Linux Programming(Neil Matthew and Richard Stones, Wrox)

Programming with Qt(Matthias Kalle Dalheimer, O'Reilly)

Qt Programming in 24 Hours(Daniel Solin, Sams)

Qt 리눅스 프로그래밍 (송호중, DreamBook)

프로그래머세계 2000년 7월 특집 리눅스 어플리케이션 개발의 모든 것

Qt 3.0 Whitepaper(<http://doc.trolltech.com>)

트롤테크 홈페이지 (<http://www.trolltech.com>)

Open Sound System(<http://www.opensound.com>)

QPushButton Class Reference

The QPushButton widget provides a command button. [More...](#)

```
#include <qpushbutton.h>
```

Inherits [QPushButton](#).

[List of all member functions.](#)

Public Members

[QPushButton](#) (QWidget * parent, const char * name=0)
[QPushButton](#) (const QString & text, QWidget * parent, const char * name=0)
[QPushButton](#) (const QIconSet & icon, const QString & text, QWidget * parent, const char * name=0)
[~QPushButton](#) ()
virtual void [setToggleButton](#) (bool)
bool [autoDefault](#) () const
virtual void [setAutoDefault](#) (bool autoDef)
bool [isDefault](#) () const
virtual void [setDefault](#) (bool def)
virtual void [setIsMenuButton](#) (bool)(*obsolete*)
bool [isMenuButton](#) () const(*obsolete*)
void [setPopup](#) (QPopupMenu * popup)
QPopupMenu* [popup](#) () const
Void [setIconSet](#) (const QIconSet &)
QIconSet* [iconSet](#) () const
Void [setFlat](#) (bool)
bool [isFlat](#) () const

Public Slots

virtual void [setOn](#) (bool)
void [toggle](#) ()

Important Inherited Members

QString [text](#) () const
virtual void [setText](#) (const QString & text)
QString [text](#) () const
const QPixmap* [pixmap](#) () const
virtual void [setPixmap](#) (const QPixmap & pixmap)
int [accel](#) () const
virtual void [setAccel](#) (int key)
bool [isToggleButton](#) () const
virtual void [setDown](#) (bool enable)
bool [isDown](#) () const
bool [isOn](#) () const
bool [autoRepeat](#) () const
bool [isExclusiveToggle](#) () const
QButtonGroup* [group](#) () const
virtual void [setAutoRepeat](#) (bool enable)
void [toggle](#) ()
void [pressed](#) ()
void [released](#) ()
void [clicked](#) ()
void [toggled](#) (bool on)
void [stateChanged](#) ([int] state)