

다양한 분야의 소프트웨어 개발을 위한

Qt 프로그래밍

[1판, Ver 1.0]

Qt개발자 커뮤니티, www.qt-dev.com

김대진 지음



Qt Programming

[First Edition, Ver 1.0]

데스크탑, 모바일, 임베디드등 다양한 운영체제 플랫폼에서 동일한
소스코드를 사용해 소프트웨어 개발이 가능한 Qt Framework

인큐빅

www.incubic-corp.com

Qt 프로그래밍

배포일	2019년 8월 2일
배포 버전	1판, Version 1.0 (First Edition, Version 1.0)
홈페이지 URL	www.qt-dev.com
예제 소스 URL	www.qt-dev.com
저자	김대진
저자 이메일	daejin0626@gmail.com

의견을 받습니다.

내용의 장점과 문제점이 무엇인지, 더욱 알차게 꾸밀 수 있는 아이디어가 있으면 저자 이메일로 연락 주시기 바랍니다.

[머리말]

Qt는 MS 윈도우, 리눅스, MacOS 와 같은 Desktop 기반의 운영체제 플랫폼과 다양한 Embedded 디바이스를 환경에서 사용되는 Embedded Linux, UWP, QNX 등과 같은 운영체제 플랫폼 그리고 모바일 디바이스에서 사용하는 Android, iOS 플랫폼에서 동작하는 다양한 소프트웨어를 개발할 수 있는 멀티플랫폼을 지원하는 개발 프레임워크입니다.

전 세계 백만명 이상이 다양한 컴퓨팅 환경에서 Qt를 이용해 소프트웨어를 개발하고 있습니다. 자동차, 국방, 가전, 모바일과 같은 임베디드 환경에 이르기 까지 다양한 분야에서 Qt를 활용하고 있습니다.

요즘과 같이 다양한 운영체제 플랫폼을 사용하는 상황에서 여러 플랫폼을 지원하는 소프트웨어를 개발하는 것은 쉽지 않은 과제입니다. 예를 들어 Embedded Linux에서 동작하는 소프트웨어를 Android, iOS 등과 같은 이 기종의 운영체제 플랫폼에서 동작하는 동일한 소프트웨어를 개발하는 것은 많은 시간과 노력을 투자 해야 합니다.

Qt를 이용해 작성한 소스코드는 다양한 운영체제 플랫폼에서 호환이 가능하므로 개발 시간을 단축할 수 있습니다. 그러므로 Qt는 급변하는 시장에서 기업의 소프트웨어 경쟁력을 확보할 수 있는 적합한 소프트웨어 개발 프레임워크라고 생각합니다.

Qt에 관심 있는 분들께 조금이나마 도움이 되는 마음으로 종이 책으로 출판하지 않고 작성한 내용을 그대로 문서 파일 그대로 독자 분들에게 배포합니다. Qt에 관심 있는 분들께 조금이나마 도움이 되길 바랍니다.

마지막으로 이스라엘 민족에게 축복과 예루살렘의 평안을 가져다 주시옵고 이스라엘 민족이 모두 예수님을 메시아로 믿도록 하여 주시옵소서. 그리고 아직 복음이 전파되지 않은 곳에 하나님의 복음과 사랑이 전파되길 기도합니다. 또한 하나님의 선하신 은혜가 여러분과 함께하길 기도합니다. 아멘.

김대진 드림

Table of Contents

1. Qt 소개와 개발환경 구축.....	1
2. Qt 프로그래밍의 시작	15
3. Qt Basic	32
3.1. Qt GUI Widgets	33
3.2. 레이아웃	74
3.3. Qt 에서 제공하는 데이터 타입과 클래스	80
3.4. Signal 과 Slot	97
3.5. Qt Designer 를 이용한 GUI 설계	102
3.6. 다이얼로그	111
3.7. MDI 기반 윈도우 GUI 구현	123
3.8. 파일 처리와 데이터스트림	128
3.9. Qt Property	137
3.10. QTimer	143
3.11. QThread	146
3.12. Qt Linguist Tool 을 이용한 다국어 처리.....	154
3.13. 라이브러리 제작.....	162
3.14. Model 과 View.....	173
4. QPainter 클래스를 이용한 GUI 2D 그래픽스.....	185
5. Qt Graphics View Framework	208
6. Animation Framework 와 State Machine	219
7. Qt OpenGL 모듈을 이용한 3D 그래픽스	231
8. Qt 3D 모듈을 이용한 3D 그래픽스	243

9.	XML	250
10.	JSON.....	260
11.	Qt Chart.....	265
12.	Qt Data Visualization.....	274
13.	네트워크 프로그래밍	282
14.	Qt WebSockets	317
15.	Qt WebEngine	327
16.	Inter Process Communication.....	334
	16.1. Unix Domain Socket 과 Named pipe.....	335
	16.2. QProcess 를 이용한 외부 프로세스와의 통신	343
	16.3. QSerialPort 를 이용한 시리얼 통신	347
	16.4. Shared Memory 를 이용한 공유 메모리 사용.....	355
17.	데이터베이스	362
18.	멀티미디어	381
	18.1. 오디오.....	383
	18.2. 비디오.....	402
	18.3. 카메라.....	411
19.	Qt Install Framework	418

1. Qt 소개와 개발환경 구축

Qt는 MS윈도우, 리눅스, MacOS 와 같은 데스크탑 기반 운영체제에서 어플리케이션을 개발하기 위해 동일한 개발 프레임워크를 제공하기 때문에 개발에 필요한 시간과 비용을 절약할 수 있다. 그리고 Qt 프레임워크를 이용하면 안드로이드(리눅스와 동일한 커널을 사용)와 iOS 모바일 플랫폼에서 동일한 Qt 프레임워크를 이용해 어플리케이션 개발이 가능하다는 장점이 있다.

Qt는 데스크탑, 모바일 기반의 플랫폼 이외에도 소형 기기와 같은 디바이스에 내장된 임베디드 플랫폼에서도 Qt를 이용해 응용 어플리케이션 개발이 가능하다. Qt 프레임워크는 Embedded Linux, QNX, WinRT 플랫폼에서 Qt개발 프레임워크를 이용해 어플리케이션 개발이 가능하다.

Qt는 C++을 사용한다. 그리고 Qt 프레임워크는 C++외에도 Qt Quick(QML) 을 제공한다. Qt Quick 은 QML을 이라는 인터프리터언어를 사용한다. Qt로 어플리케이션 개발 시, GUI 또는 UX를 QML을 이용해 개발할 수 있다. QML을 사용하면 비즈니스로직과 디자인로직으로 나누어 개발할 수 있다.

여기서 비즈니스로직이란 GUI상에서 어떤 버튼을 클릭했을 때 동작하는 과정을 비즈니스로직이라고 한다. 디자인로직은 GUI를 가리킨다. Qt로 어플리케이션을 개발 할 때 비즈니스로직을 C++로 개발하고 디자인로직을 QML로 개발함으로써 개발 유지보수성을 높일 수 있다.

예를 들어 GUI화면을 QML로 개발하고 사용자 이벤트 및 처리과정을 C++로 개발함으로써 두개의 분리된 로직을 분리함으로써 비즈니스로직의 재 사용성을 높일 수 있다.

또한 C++ 을 이용해 GUI를 개발할 수 있다. GUI를 QML로 개발하는 경우 장단점이 있다. 사용하는 GUI가 터치 스크린을 사용하고 안드로이드나 iOS와 같이 그래픽 Effect 효과를 많이 사용해야 하는 경우 QML이 적합하다.

하지만 데스크탑과 같이 많은 정보를 사용자에 제공하는 경우는 GUI를 C++ 개발하는 것이 더 효율적이다.

그리고 메모리 사용량이 제한된 임베디드 시스템 이거나 CPU(또는GPU) 성능이 낮다면 C++을 사용하는 것이 QML을 사용하는 것보다 성능면에서 유리하다. 따라서 개발하려는 소프트웨어가 어떤 특징이 있는지 먼저 살펴보고 GUI(디자인로직) 를 C++로 개발

할 것인지 QML을 사용할 것인지 결정하는 것이 바람직하다.

- 데스크탑 기반의 플랫폼에서 개발환경 구축하기

Qt는 데스크탑 기반의 플랫폼(또는 운영체제)으로 MS윈도우, 리눅스 그리고 MacOS 플랫폼을 지원한다. Qt 프레임워크 SDK는 Qt 공식 홈페이지에서 플랫폼 별 다운로드가 가능한 웹사이트를 제공한다.

- ✓ Qt개발 프레임워크 다운로드 URL - <http://download.qt.io>

Name	Last modified	Size	Metadata
■ snapshots/	09-Nov-2017 15:39	-	
■ online/	13-Mar-2014 08:45	-	
■ official_releases/	12-Jun-2018 10:20	-	
■ ministro/	20-Feb-2017 10:32	-	
■ linguist_releases/	26-Mar-2019 07:49	-	
■ learning/	22-May-2013 16:20	-	
■ development_releases/	25-Feb-2019 12:06	-	
■ community_releases/	23-Feb-2017 07:29	-	
■ archive/	16-Dec-2014 10:39	-	
■ timestamp.txt	04-Apr-2019 08:00	11	Details

<그림> Qt개발 프레임워크 SDK 다운로드 웹사이트

이전 화면에서 보는 것과 같이 항목 중 “archive” 항목을 클릭한다. 다음으로 항목 중 “qt” 항목을 클릭한다.

Name	Last modified	Size	Metadata
Parent Directory		-	
vsaddin/	13-Feb-2019 16:10	-	
qtcreator/	17-Jan-2019 10:43	-	
qt/	03-Dec-2018 07:37	-	
online_installers/	23-May-2017 15:51	-	

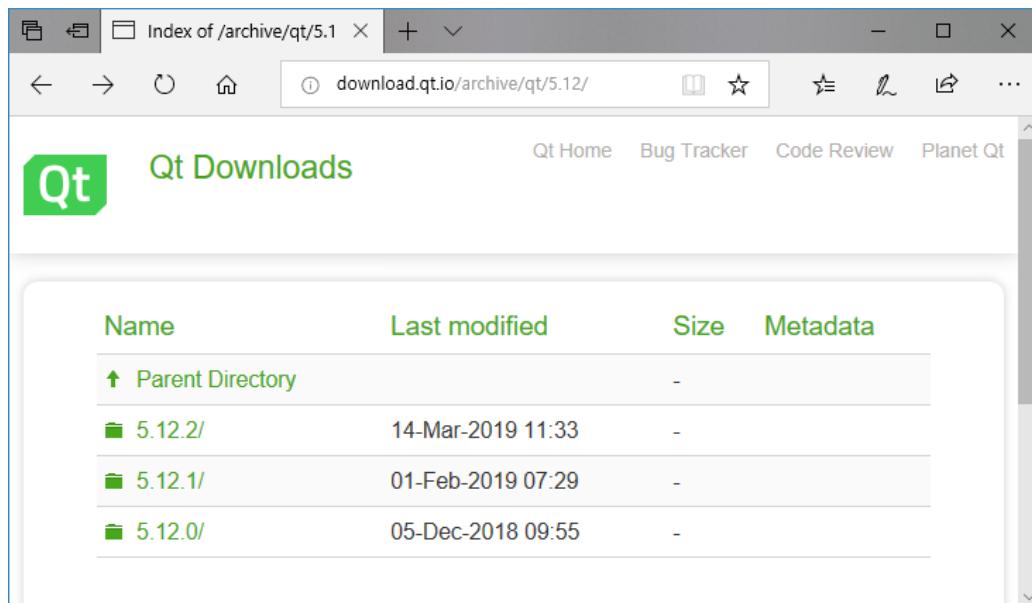
<그림> Qt개발 프레임워크 SDK 다운로드 웹사이트

위의 그림에서 보는 것과 같이 “qt” 항목을 클릭하면 다음 화면에서 보는 것과 같이 버전 별, 원하는 Qt 개발 프레임워크 SDK를 다운로드 받을 수 있다.

Name	Last modified	Size
Parent Directory		-
5.12/	14-Mar-2019 09:19	-
5.11/	03-Dec-2018 07:35	-
5.10/	13-Feb-2018 10:20	-
5.9/	22-Oct-2018 07:44	-
5.8/	23-Jan-2017 08:38	-
5.7/	14-Dec-2016 07:31	-

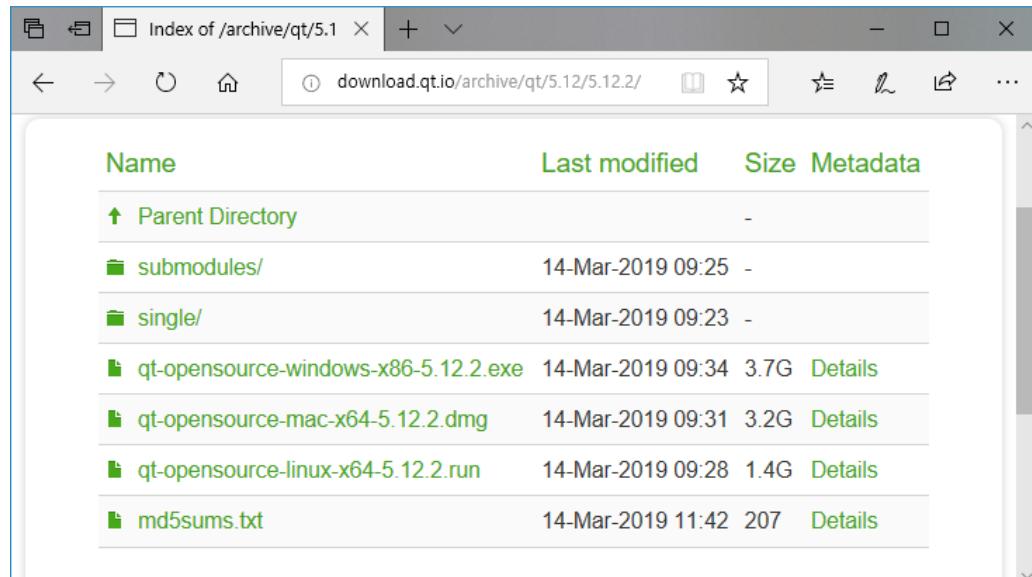
<그림> Qt개발 프레임워크 SDK 다운로드 웹사이트

이전의 그림에서 보는 것과 같이 5.12 버전 항목을 선택한다. 5.12 버전 항목을 선택하면 다음 그림에서 보는 것과 같이 하위 버전을 선택할 수 있다.



<그림> Qt개발 프레임워크 SDK 다운로드 웹사이트

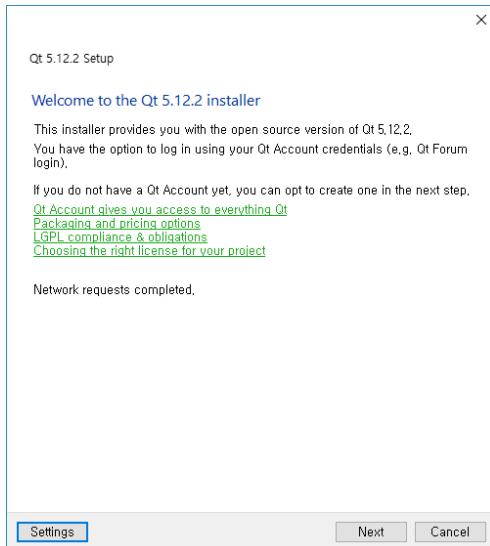
위의 그림에서 보는 것과 같이 하위 버전 중 가장 상단에 있는 5.12.2 버전 항목을 클릭한다. 그럼 아래 화면과 같이 플랫폼(운영체제) 별 Qt개발 프레임워크 SDK를 다운로드 받을 수 있다.



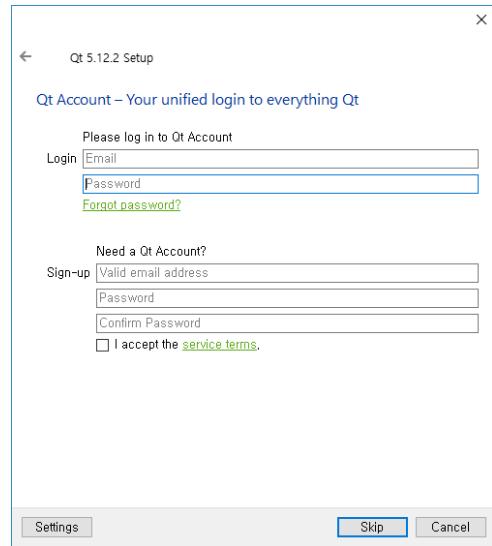
<그림> 플랫폼 별Qt개발 프레임워크 SDK

✓ MS 윈도우에서 개발 환경 구축하기

MS윈도우 플랫폼(운영체제)에서 Qt 개발 프레임워크 SDK를 다운로드 받았다면 다운로드 받은 SDK 설치를 시작해보자. 이전 절에서 설명한 SDK 중 MS윈도우 플랫폼에 설치 파일을 실행한다.

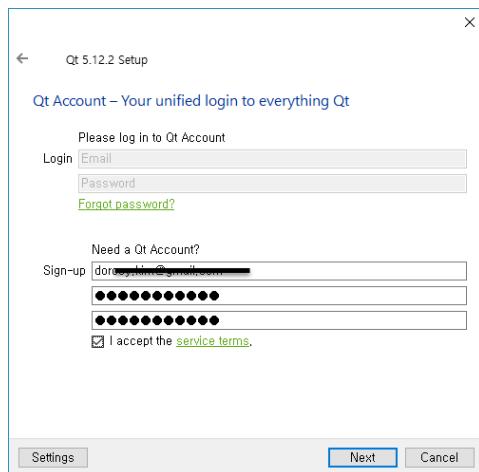


<그림> 첫 번째 설치 화면

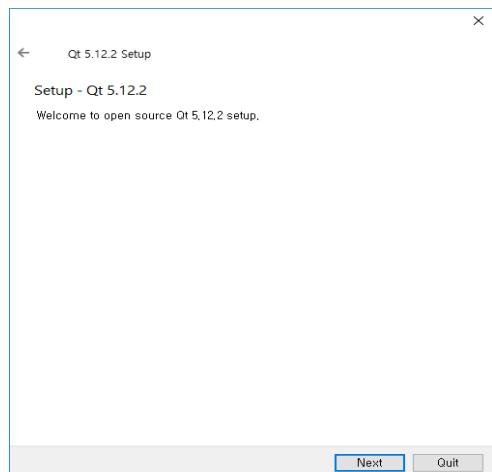


<그림> 두 번째 설치 화면

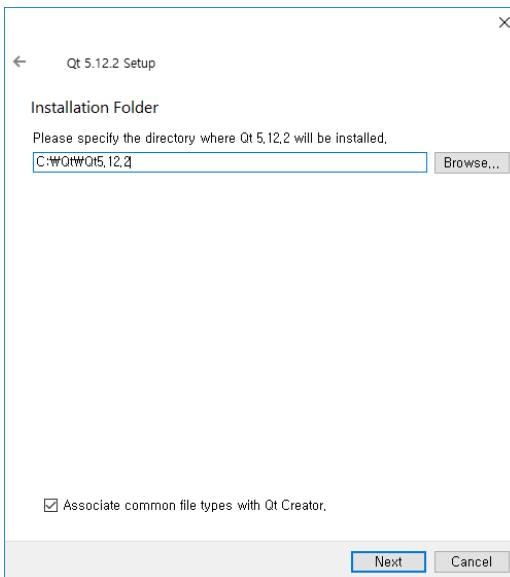
이전 그림 중 두 번째 설치 화면은 계정 아이디와 비밀번호를 입력 한다. 만약 계정이 없다면 두 번째 설치 화면의 하단에서 보는 것과 같이 사용할 새로운 아이디, 비밀번호를 입력하면 [Skip] 버튼이 [Next] 버튼으로 바뀐다. [Next] 버튼을 클릭한다. [Skip] 버튼은 계정 없이 설치할 수 있다



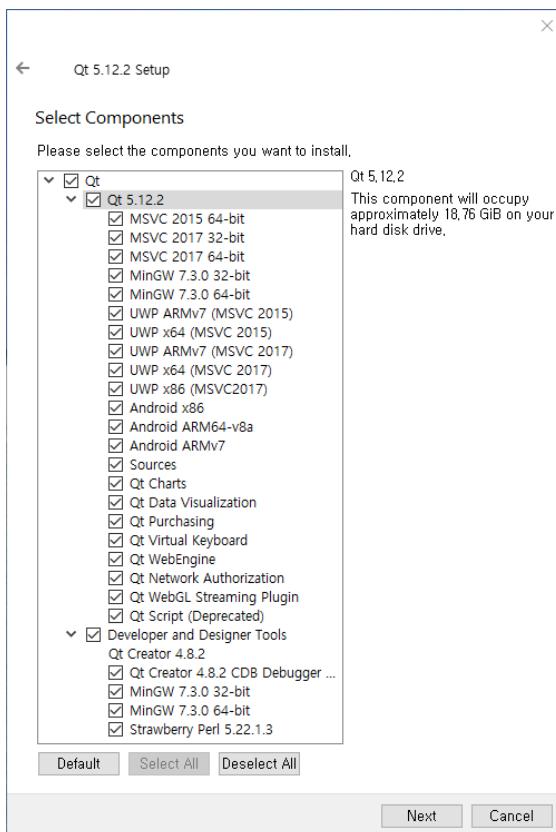
<그림> 새로운 계정 입력



<그림> 설치화면



<그림> 새로운 계정 입력



<그림> Components 선택 화면

좌측 그림은 Qt개발 프레임워크를 설치할 디렉토리를 입력하거나 [Browse] 버튼을 클릭해 디렉토리 다이얼로그 창에서 설치할 디렉토리를 선택할 수 있다.

좌측 화면은 설치할 Component들을 선택하는 화면이다.

Qt > Qt 5.X.X 의 항목들은 어떤 컴파일러를 사용할지 사용자가 선택할 수 있다. 예를 들어 Qt > Qt 5.X.X 항목 중 MSVC 2015 64-bit를 선택하면 Qt로 작성한 어플리케이션을 Microsoft Visual C++ 2015 64-bit 컴파일러로 Qt로 개발한 어플리케이션을 빌드 할 수 있다.

Qt > Qt 5.X.X > MinGW 7.3.0 32-bit 항목은 오픈 GCC 의 32Bit 버전의 컴파일러로 Qt로 개발한 어플리케이션을 빌드 할 수 있다.

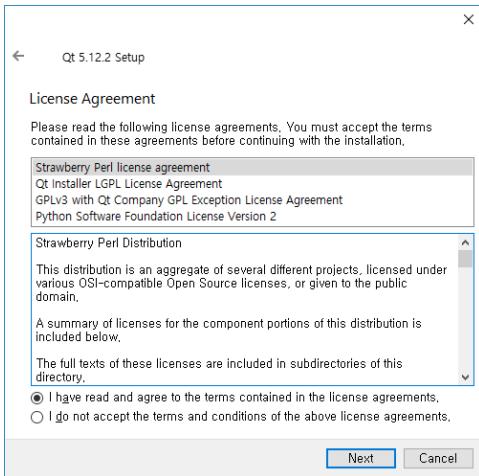
Qt > Qt 5.12.2 > UWP xxx 시작하는 항목 들은 Universal Windows Platform 컴파일러 Qt로 개발한 어플리케이션을 빌드 할 수 있다. 각 항목들은 추후 어플리케이션 개발 시 빌드할 컴파일러를 다중 선택 할 수 있다. 여기서는 모든 항목을 선택하고 [Next] 버튼을 클릭한다.

<표> Component 각 항목별 설명

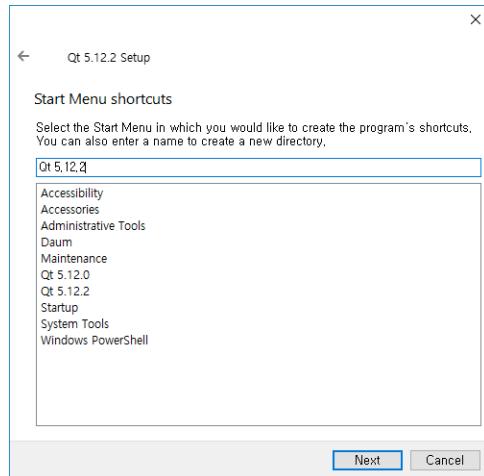
카테고리	Component	설명
Qt X.X.X	MSVC 2015 64-bit	Microsoft Visual C++ 2015 64 Bit 컴파일러
	MSVC 2017 32-bit	Microsoft Visual C++ 2017 32Bit 컴파일러
	MSVC 2017 64-bit	Microsoft Visual C++ 2017 64 Bit 컴파일러
	MinGW 7.3.0 32-bit	오픈소스 GCC/G++ 32 Bit 컴파일러 (Prebuilt)
	MinGW 7.3.0 64-bit	오픈소스 GCC/G++ 64 Bit 컴파일러 (Prebuilt)
	UWP ARMv7 (MSVC 2015)	Universal Windows Platform 컴파일러. (ARMv7 CPU용 Microsoft Visual C++ 2015)
	UWP x64 (MSVC 2015)	Universal Windows Platform 컴파일러. (ARMv7 CPU용 Microsoft Visual C++ 2015 64 bit).
	UWP ARMv7 (MSVC 2017)	Universal Windows Platform 컴파일러. (ARMv7 CPU용 Microsoft Visual C++ 2017)
	UWP x64 (MSVC 2017)	Universal Windows Platform 컴파일러. (인텔 x86 계열 CPU용 Microsoft Visual C++ 2015 64 bit)
	UWP x86 (MSVC 2017)	Universal Windows Platform 컴파일러. (인텔 x86 계열 CPU용 Microsoft Visual C++ 2015)
	Android x86	인텔 x86계열 CPU를 사용하는 모바일 Android 플랫폼에 사용하기 위한 컴파일러.
	Android x64 ARM64-v8a	ARM64-v8a CPU를 사용하는 모바일 Android 플랫폼에 사용하기 위한 64 bit 컴파일러.
	Android ARM7	ARM7 CPU를 사용하는 모바일 Android 플랫폼에 사용하기 위한 64 bit 컴파일러.
	Source	Qt API를 디버깅 하기 위해서 필요. 예를 들어 특정 소스코드 지점을 브레이크 포인트를 이용

		해 Qt API 소스코드를 디버깅 하기 위해 필요.
	Qt Charts	그래프 차트 API 컴포넌트.
	Qt Data Visualization	3차원 형태의 차트 API 컴포넌트.
	Qt Purchasing	Google Play 와 Apple Store 와 같이 인앱(in-app) 구매를 지원하기 위한 API.
	Qt Virtual Keyboard	가상 키보드 컴포넌트 지원
	Qt WebEngine	Qt WebKit 엔진
	Qt Network Authorization	네트워크 인증을 요구하는 기능 구현 시 필요한 네트워크 API.
	Qt WebGL Streaming Plugin	WebGL을 지원하는 웹 브라우저에서 Texture, Buffer, Glyphs 등과 같은 컴포넌트.
	Qt Script (Deprecated)	더 이상 사용되지 않는 항목
Developer and Designer Tools	Qt Creator 4.8.2	Qt IDE 개발 툴이다. 사용자가 선택이 불가능 하다. 필수 설치 항목 이다.
	Qt Creator 4.8.2 CDB Debugger Support	디버깅 용으로 CDB를 지원하기 위한 컴포넌트. MinGW 은 GDB를 사용하지만 MSVC 컴파일러를 사용하는 경우는 CDB로만 디버깅이 가능.
	MinGW 7.3.0 32-bit	MinGW 로 빌드 된 7.3.0 32 bit 툴체인
	MinGW 7.3.0 64-bit	MinGW 로 빌드 된 7.3.0 64 bit 툴체인
	Strawberry Perl 5.22.1.3	마이크로소프트 윈도 플랫폼을 위한 Perl 배포판

다음 설치 화면은 라이선스를 동의한다는 첫 번째 항목을 선택하고 [Next] 버튼을 클릭 한다.

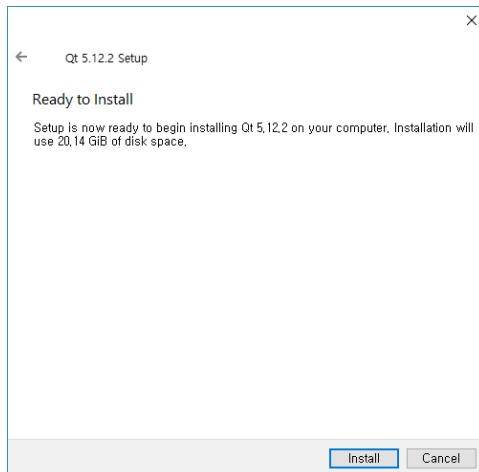


<그림> 라이선스 동의 화면

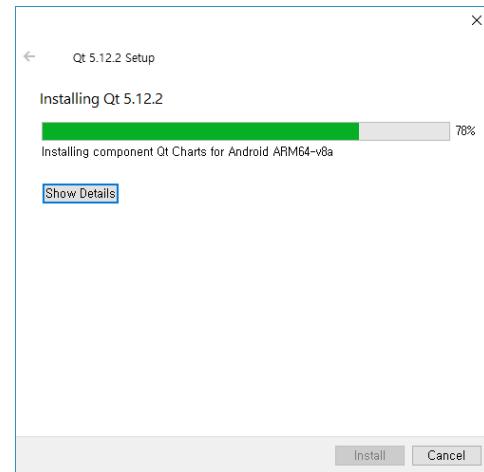


<그림> 시작메뉴에 등록할 이름

시작메뉴에 등록할 이름 화면은 윈도우 시작 메뉴에 등록할 이름 선택 화면에서 [Next] 버튼을 클릭하면 설치할 준비가 끝났다는ダイ얼로그 화면을 확인할 수 있다. 이ダイ얼로그 화면에서 [Next] 버튼을 클릭하면 설치가 진행된다.

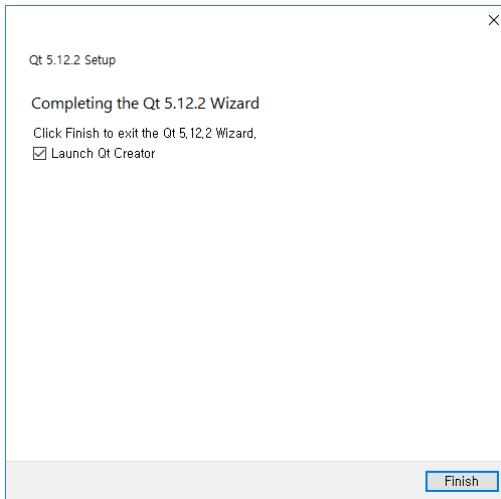


<그림> 설치 준비 화면



<그림> 설치 과정 화면

설치과정이 모두 완료 되면 “Launch Qt Creator” 체크박스 항목이 있다. 이 항목은 Qt 개발 시 사용할 IDE를 시작할 것인가를 물어 보는 체크박스이다. 아래 [Finish] 버튼을 클릭해 설치를 완료한다.



<그림> 설치 완료 화면

그림에서 보는 것과 같이 “Launch Qt Creator” 항목을 체크하고 [Finish] 버튼을 클릭하면 설치 다이얼로그가 종료 후 Qt Creator라는 IDE 개발 툴이 실행된다.

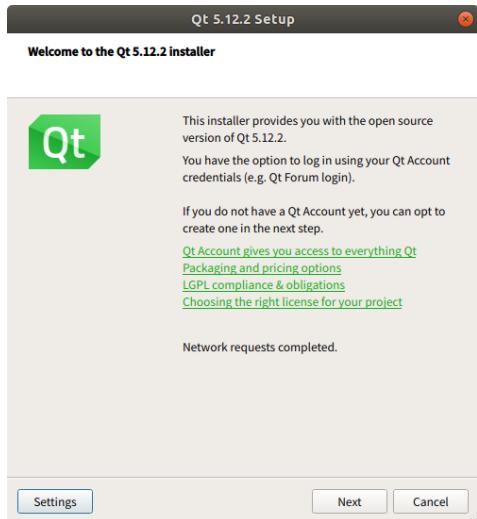
✓ 리눅스에서 개발 환경 구축하기

리눅스 플랫폼(운영체제)은 여러 배포판이 있지만 Qt 프레임워크 SDK를 설치할 플랫폼으로 우분투 18.04 64Bit 버전을 사용한다. 다운로드 받은 리눅스 용 Qt 프레임워크 SDK 설치 파일을 다운로드 받으면 실행할 수 있는 권한을 다음 그림에서 보는 것과 같이 터미널에서 chmod 명령을 이용해 실행 권한을 추가해 줘야 한다.

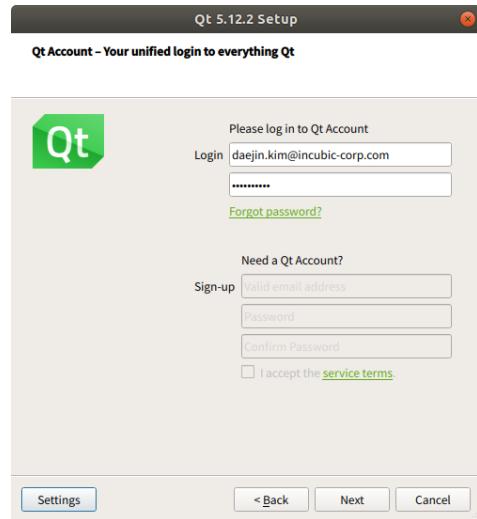
```
incubic@linux1:~/다운로드$ ls -tlr
합계 1422080
-rw-rw-r-- 1 incubic incubic 1456203550 4월  5 13:56 qt-opensource-linux-x64-5.12.2.run
incubic@linux1:~/다운로드$ chmod 755 qt-opensource-linux-x64-5.12.2.run
incubic@linux1:~/다운로드$ ls -ltr
합계 1422080
-rwxr-xr-x 1 incubic incubic 1456203550 4월  5 13:56 qt-opensource-linux-x64-5.12.2.run
incubic@linux1:~/다운로드$
```

<그림> chmod 를 이용한 실행권한 추가

실행 권한 추가를 완료했으면 우분투 리눅스에서 제공하는 파일 브라우저를 이용해 실행파일을 실행 시켜도 된다. 또는 이전 그림에서 보는 것과 같이 터미널에서 직접 실행해도 된다.

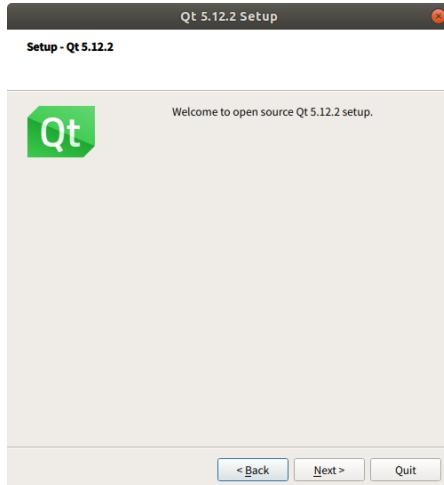


<그림> 설치 시작 화면

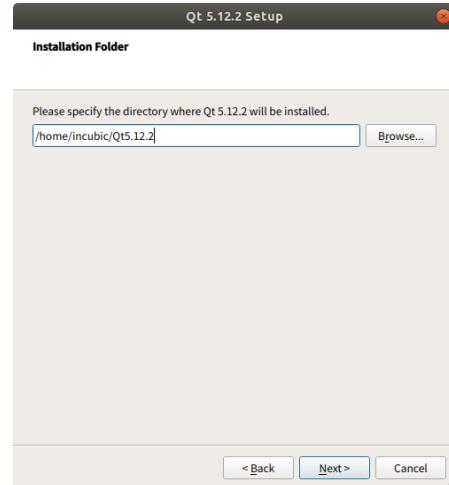


<그림> 계정 입력 환경

이전 그림 중 계정 입력 화면은 계정 아이디와 비밀번호를 입력 한다. 만약 계정이 없다면 계정 입력 다이얼로그 화면의 하단에서 보는 것과 같이 사용할 새로운 아이디, 비밀번호를 입력하고 [Next] 버튼을 클릭한다.

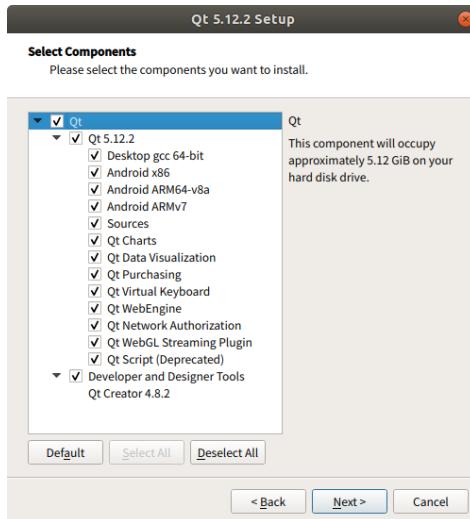


<그림> 설치 화면



<그림> 설치할 디렉토리 설정

Qt 프레임워크 SDK를 설치할 디렉토리 선택한 다음 [Next] 버튼을 클릭한다.



<그림> Components 선택 화면

<표> 리눅스 Qt 프레임워크 SDK 설치 Component 항목별 설명

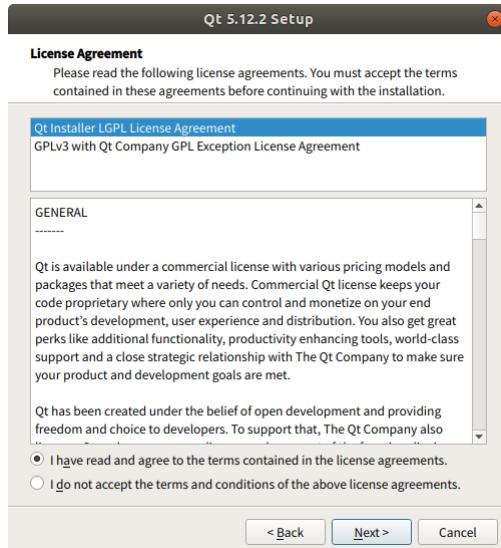
카테고리	Component	설명
Qt X.X.X	Desktop gcc 64-bit	오픈 GCC/G++ 64 bit 컴파일러를 이용해 Qt 어플리케이션을 빌드하기 위한 컴파일러
	Android x86	인텔 x86 CPU를 사용하는 모바일 Android 플랫폼에 사용하기 위한 컴파일러.
	Android x64 ARM64-v8a	ARM64-v8a CPU를 사용하는 Android 플랫폼에 사용하기 위한 64 bit 컴파일러.
	Android x64 ARM7	ARM7 CPU를 사용하는 모바일 Android 플랫폼에 사용하기 위한 64 bit 컴파일러.
	Source	개발한 소스코드 이외에 Qt에서 제공하는 API 내로 디버깅을 이용하기 위해서 필요.
	Qt Charts	그래프 차트 API Component이다.
	Qt Data Visualization	3차원 형태의 차트 API Component이다.
	Qt Purchasing	Google Play 와 Apple Store 와 같이 인앱(in-app) 구매를 지원하기 위한 API이다.
	Qt Virtual Keyboard	가상 키보드
	Qt WebEngine	Qt에서 웹 브라우저 기반의 어플리케이션

좌측의 그림은 설치 할 Component를 선택하는 하면 이다.

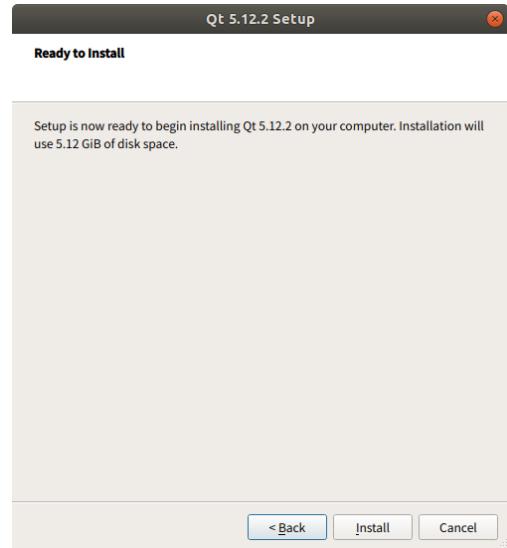
각 Component 설명은 다음 표를 참조 한다. 여기서는 모든 Component를 선택한 다음 [Next] 버튼을 클릭한다.

		개발 시 필요.
	Qt Network Authorization	네트워크에서 인증을 요구하는 기능 구현 시 필요한 네트워크 API
	Qt WebGL Streaming Plugin	WebGL을 지원하는 웹 브라우저 상에서 Texture, Buffer, Glyphs 등과 같은 컴포넌트
	Qt Script (Deprecated)	더 이상 사용되지 않는 항목
Developer and Designer Tools	Qt Creator 4.8.2	Qt IDE 개발 툴이다. 사용자가 선택이 불가능 하다. 필수 설치 항목이다.

이전 그림에서 보는 것과 같이 모든 Component를 선택한 다음 [Next] 버튼을 클릭한다. 라이선스 등의 다이얼로그이다. 첫 번째 항목을 선택한 다음 [Next] 버튼을 클릭한다.

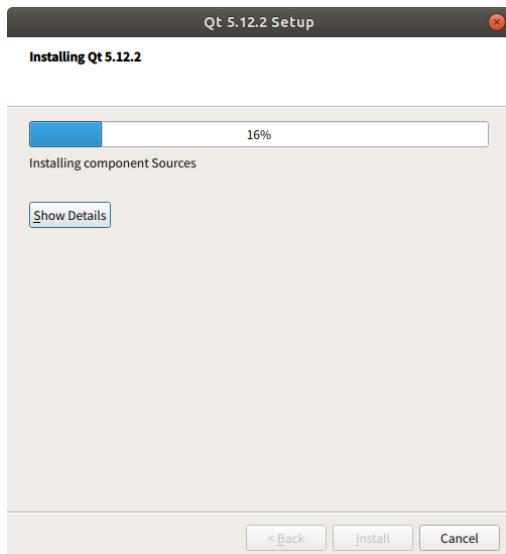


<그림> 라이선스 등의 화면

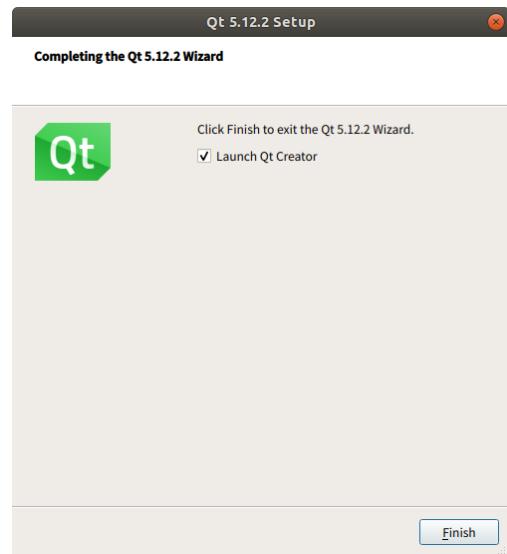


<그림> 설치 시작하기 전 화면

설치 시작하기 전 화면에서 보는 것과 같이 모든 설정이 완료 되었다. [Install] 버튼을 클릭하면 설치를 시작한다.



<그림> 설치 진행 화면



<그림> 설치 완료 화면

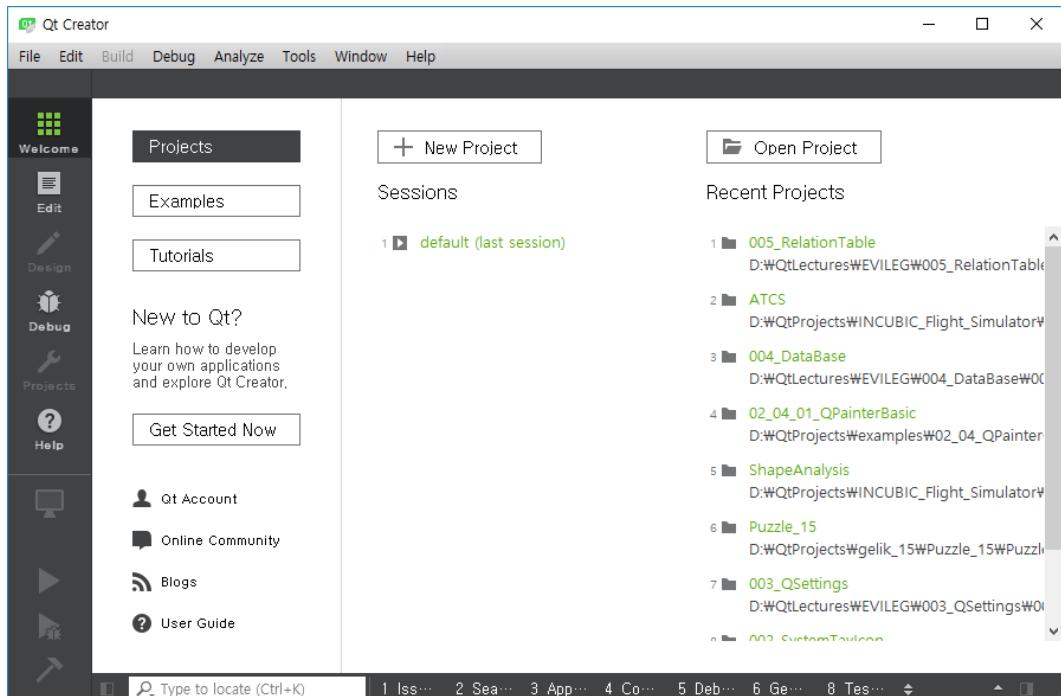
2. Qt 프로그래밍의 시작

이번 장에서는 GUI가 없는 Command 창 또는 터미널 창에 “Hello World”를 출력하는 어플리케이션과 GUI윈도우상에 “Hello World”를 출력하는 어플리케이션을 구현해 보도록 하자.

- Command 또는 터미널에 “Hello World” 출력 예제 프로그래밍.

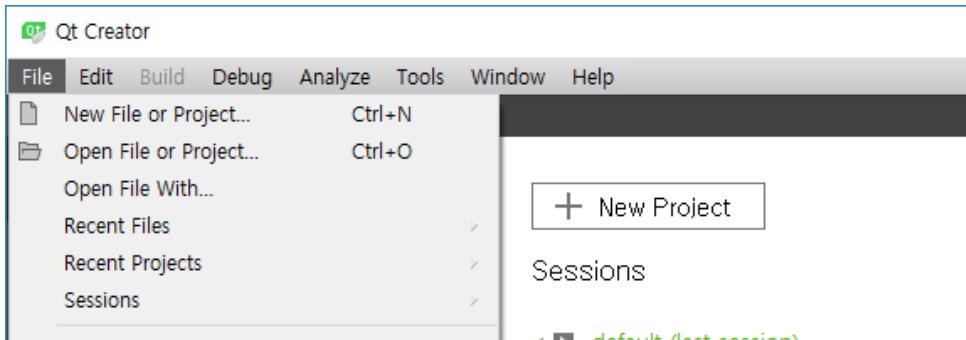
Qt 개발 프레임워크 설치를 완료하였다면 Qt Creator 가 설치되었을 것이다. Qt Creator 는 IDE 툴이다. 개발자에게 어플리케이션 개발 환경을 제공한다. 예를 들어 Visual Studio, Eclipse 등과 같은 개발 툴(IDE) 과같이 Qt에서는 IDE툴로 Qt Creator 개발 툴을 제공한다.

설치된 Qt Creator를 실행하면 다음 그림에서 보는 것과 같이 Qt Creator 가 실행된 것을 확인 할 수 있을 것이다.



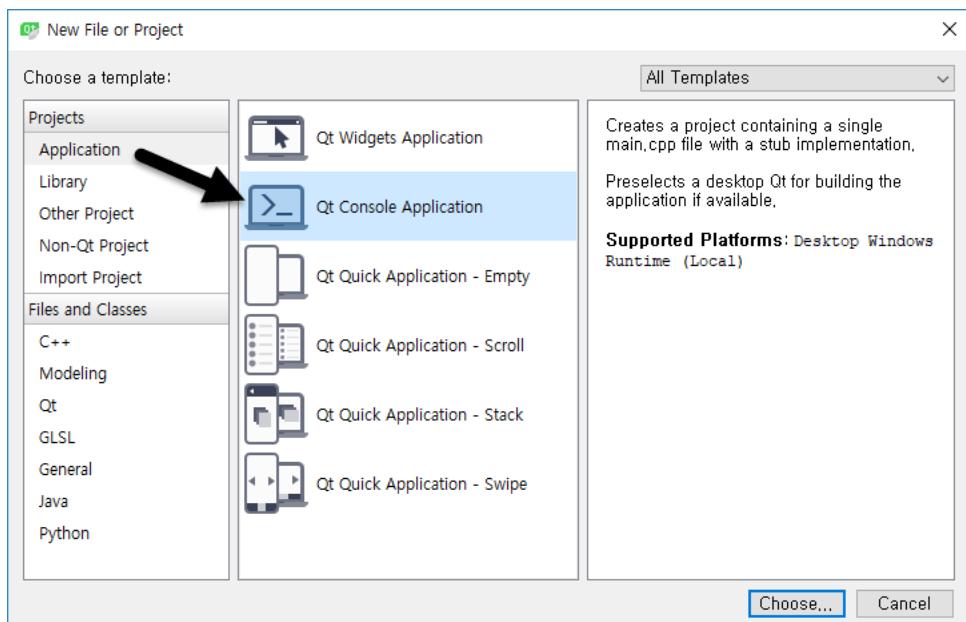
<그림> Qt Creator 실행 화면

Qt Creator 메뉴에서 [File] -> [New File or Project] 메뉴를 클릭한다.



<그림> Qt Creator 의 [File] 메뉴

[New File or Project] 메뉴는 개발하고자 하는 어플리케이션 특성에 따라 프로젝트의 아웃라인(또는 골격) 자동으로 만들어 준다. 예를 들어 내가 개발하고자 하는 특성이 어플리케이션이 아니라 라이브러리라면 라이브러리 개발에 가장 기초적인 구조를 Qt Creator가 자동으로 만들어 준다.

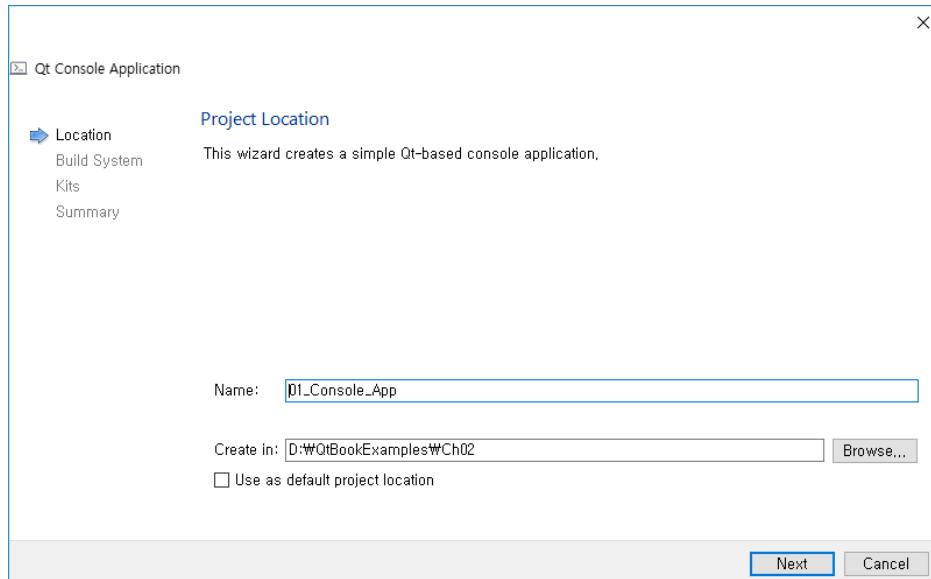


<그림> New File or Project 다이얼로그

Command 창에서 "Hello World"를 출력하는 예제를 작성할 것이므로 위의 그림에서 보는 것과 같이 [Application] -> [Qt Console Application] 을 선택하고 하단의 [Choose] 버튼을 클릭한다.

[Choose] 버튼을 클릭하면 프로젝트가 생성되는 위치를 지정해야 한다. 다음 그림에서 보는 것과 같이 프로젝트 이름과 프로젝트가 위치할 상위 디렉토리를 입력한다.

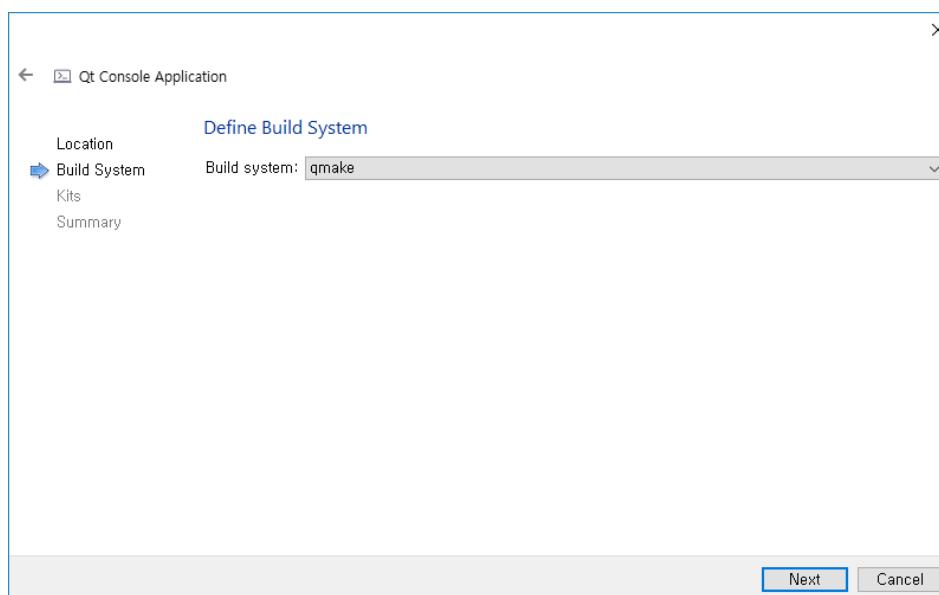
프로젝트의 이름과 위치를 선택할 때, 주의 할 점으로 한글을 사용할 경우 오류가 발생 할 수 있으므로 영문 이름을 입력한다. 그리고 다음 그림에서 보는 것과 같이 프로젝트 가 위치한 디렉토리도 한글이 들어가지 않도록 하자.



<그림> 프로젝트의 이름과 위치를 입력하는ダイ얼로그

그림에서 보는 것과 같이 [Name] 항목은 프로젝트의 이름을 입력한다. [Create In] 항목 은 프로젝트가 위치할 상위 디렉토리를 입력하고 [Next] 버튼을 클릭한다.

다음 그림은 프로젝트를 빌드할 빌드시스템을 선택해야 한다. 다음 그림에서 보는 것과 같이 첫 번째 항목인 [qmake] 를 선택하고 하단의 [Next] 버튼을 클릭한다.

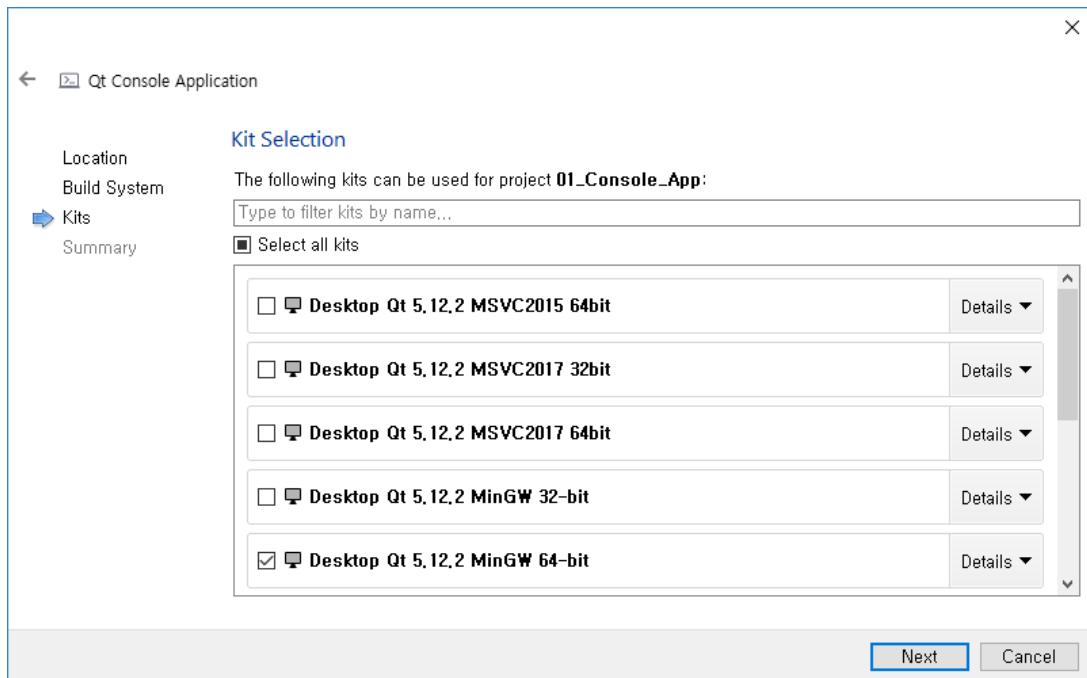


<그림> 빌드 시스템 선택 다이얼로그

다음 화면은 작성하고자 하는 어플리케이션을 컴파일 할 컴파일러를 선택하는 항목이다. 여기서는 MinGW 64-bit 항목을 선택한다. MinGW 는 오픈소스 GCC 컴파일러 이다. MinGW 64-bit 를 선택하거나 MinGW 32-bit 를 선택해도 된다.

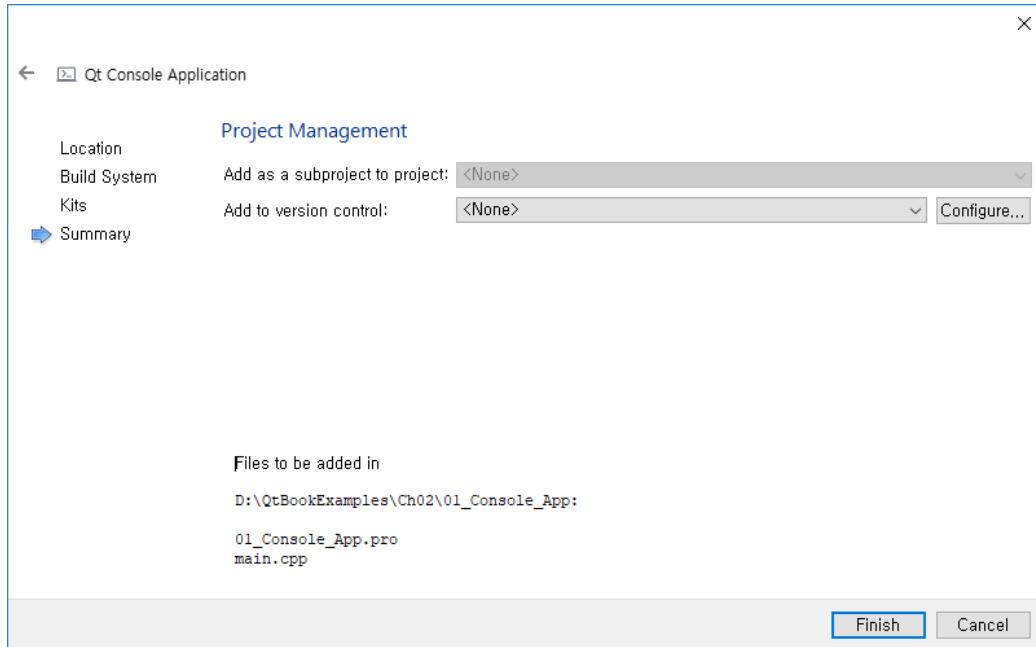
만약 MinGW 이외에 MSVC2017 64bit 버전을 선택하려면 MSVC2017 컴파일러가 먼저 설치되어 있어야 한다. 따라서 PC에 MSVC 2017 이 설치되어 있지 않은 상태에서 아래 그림에서 보는 것과 같이 MSVC 항목 중 하나를 선택하면 에러로 인해 컴파일러를 설치할 수 없다.

하지만 MinGW 를 선택하면 Qt 개발 프레임워크 설치 시 자동으로 MinGW 컴파일러가 설치되므로 별도의 컴파일러를 설치하지 않아도 Qt로 개발한 어플리케이션 빌드가 가능하다.



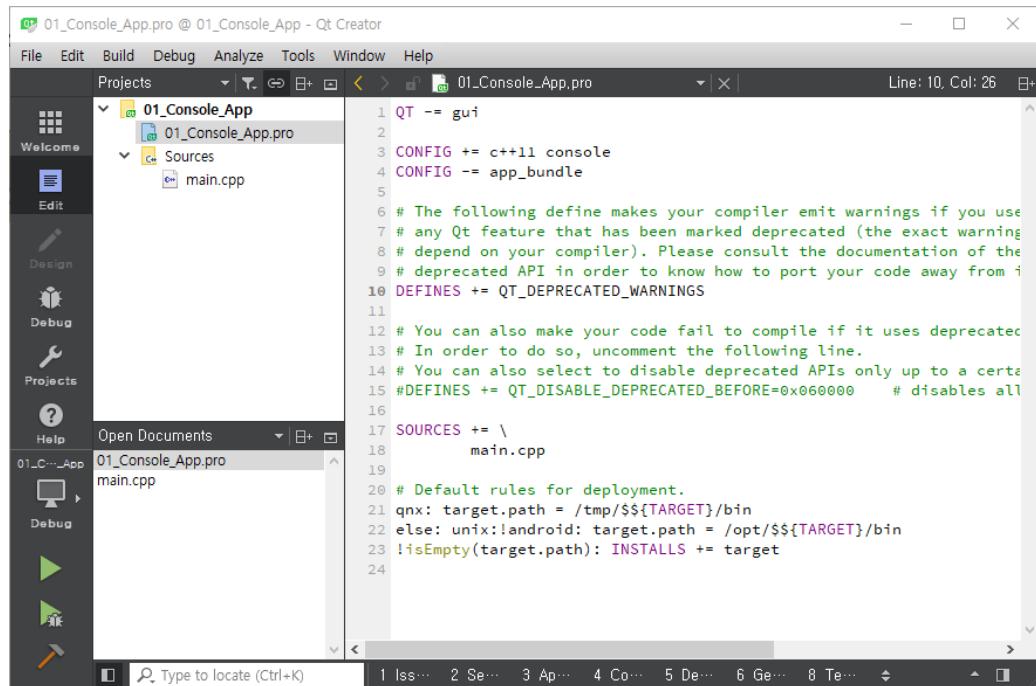
<그림> 빌드할 컴파일러 선택 다이얼로그

위의 그림에서 보는 것과 같이 [Desktop Qt 5.x.x MinGW 64Bit] 항목을 선택한다. 그리고 하단의 [Next] 버튼을 클릭한다.



<그림> 프로젝트 생성 마지막ダイアログ

이전 그림에서 보는 것과 같이 하단 [Finish] 버튼을 클릭하면 모든 설정이 완료되고 콘솔 기반의 어플리케이션 개발, 즉 터미널 또는 Command 창에서 "Hello World" 프로그램을 작성하기 위한 기본적이 소스코드를 Qt Creator 가 자동으로 작성해 준다.



<그림> 프로젝트가 생성된 화면

이전 그림에서 보는 것과 같이 "Hello World" 출력하기 위한 소스코드가 모두 생성된 것을 확인 할 수 있다. 2개의 파일이 생성된 것을 확인할 수 있다. 01_Console_App.pro 파일은 프로젝트의 속성을 지정하는 프로젝트파일이다. 두 번째 생성된 main.cpp 파일은 프로그램이 시작되는 main 함수가 자동으로 작성되어 있다. main.cpp 소스코드 파일에 다음과 같이 소스코드를 추가한다.

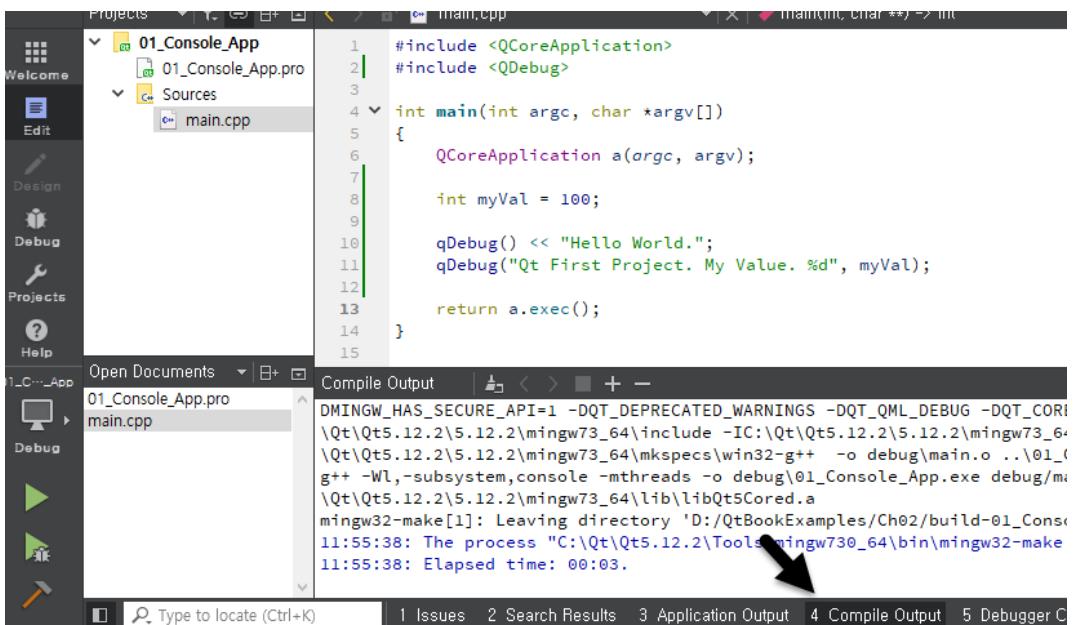
```
#include <QCoreApplication>
#include <QDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    int myVal = 100;
    qDebug() << "Hello World.";
    qDebug("Qt First Project. My Value. %d", myVal);

    return a.exec();
}
```

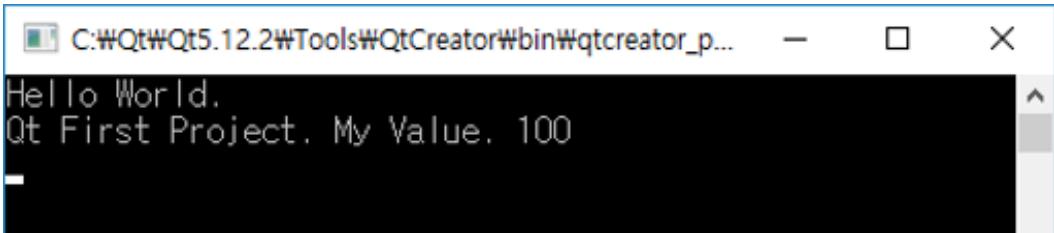
위의 예제 소스코드는 Console에 "Hello World" 를 출력한다. 빌드하기 위해 Qt Creator 툴 원도우 화면 좌측 하단의 망치 모양은 빌드 아이콘 버튼이 있다. 이 버튼을 클릭하면 빌드(컴파일)가 진행된다. 빌드 단축키는 [Ctrl + B] 이다.



<그림> Qt Creator 빌드 로그 출력

빌드하는 과정에서 에러가 발생했는지 그렇지 않고 정상적으로 빌드가 되었는지 확인하기 위해 로그를 통해 빌드 과정을 볼 수 있다. Qt Creator 툴 하단의 [Compile Output]을 클릭하면 빌드 로그를 볼 수 있다.

빌드 후 작성한 프로그램을 실행하기 위해 Qt Creator 의 좌측 하단의 화살표모양을 클릭하면 프로그램이 실행된다. 단축키는 [Ctrl + R] 키이다.



<그림> 프로그램 실행 화면

✓ 프로젝트 파일

프로젝트 파일은 확장자가 .pro 이다. 프로젝트 파일은 프로젝트의 속성을 정의하기 위한 파일이다. 예를 들어 이 프로젝트가 GUI기반 어플리케이션, 콘솔 어플리케이션, 다른 어플리케이션을 위한 라이브러리등 어떤 목적인지를 구분하기 위한 속성을 지정하기 위한 목적으로 사용된다. 다음은 프로젝트파일의 코드 이다.

```
QT -= gui

CONFIG += c++11 console
CONFIG -= app_bundle
DEFINES += QT_DEPRECATED_WARNINGS

SOURCES += \
    main.cpp

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

첫 번째 라인의 QT 매크로는 Qt 프레임워크에서 사용할 API 항목 또는 그룹을 명시해야 한다. 여기서 "gui" 는 Qt에서 제공하는 GUI 모듈을 의미한다. 여기서는 GUI모듈을 사용하지 않는 것을 의미한다.

예를 들어 Qt 에서 제공하는 네트워크 API와 GUI 를 사용하기 위해서는 다음과 같이 QT 매크로로 다음과 같이 추가해야 한다.

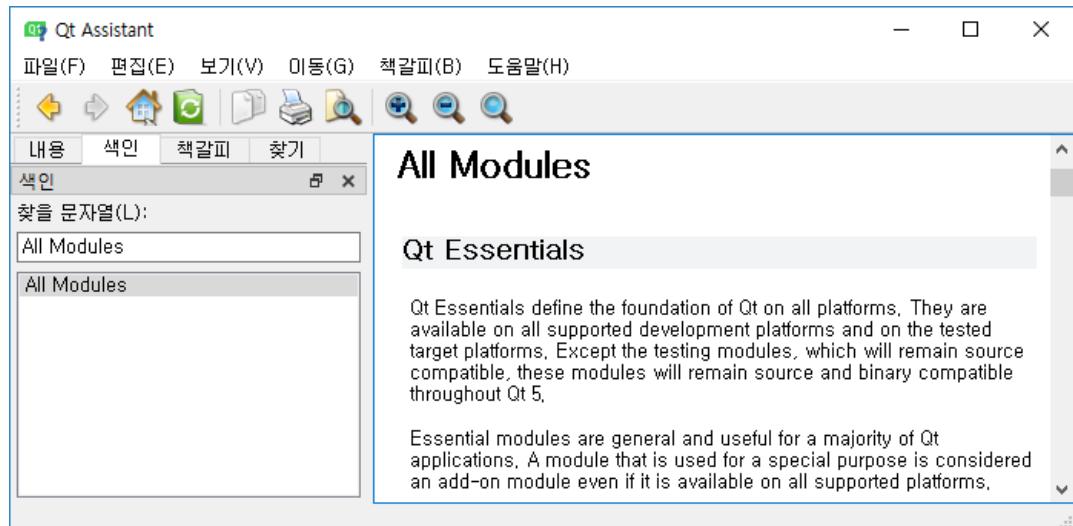
```
QT += network  
QT += gui
```

위에서 언급한 방식으로 하나씩 사용할 수 있지만 여러 모듈을 명시하기 위해서 다음과 같이 사용할 수 도 있다.

```
QT += network gui
```

방금 살펴본 것과 같이 사용하지 않는 모듈 또는 API 를 QT 매크로에서 사용하지 않는 것이 어플리케이션 개발 후 배포 시 실행 프로그램이 사용하는 모듈(API)의 용량을 줄일 수 있다. 예를 들어 network 모듈을 QT 매크로에 명시하면 실행 파일 배포 시, Qt Network 모듈 라이브러리 파일(dll 또는 so 와 같은 파일) 도 함께 배포해야 하기 때문에 사용하지 않는 모듈을 명시 하지 않는 것을 권장한다.

Qt에서 사용 가능한 모듈은 Qt 설치 시 함께 설치 되는 Qt Assistant 라는 도움말을 이용해 검색해 볼 수 있다. 아래 그림에서 보는 것과 같이 좌측 탭에서 [색인]을 선택하고 찾을 문자열에 "All Modules" 입력하면 Qt에서 사용 가능한 모듈을 자세하게 검색해 볼 수 있다.



<그림> Qt Assistant 실행 화면

다음 라인의 CONFIG 는 빌드 설정 이다. c++11 은 빌드 할 C++11 버전을 사용하겠다는 것을 의미한다. console은 이 프로젝트 속성이 console 프로그램인 것을 의미한다.

다음 라인의 app_bundle 은 MacOS 에서 빌드할 때 console 프로그램 속성을 나타낸다. Window에서는 Console 프로그램인 경우 CONFIG 매크로에 console 을 사용하고 MacOS 에서는 app_bundle 을 사용한다.

CONFIG 이외에 프로젝트 파일(.pro) 에서 CONFIG 옵션에 항목을 추가가 위해서 “+=” 을 사용하고 제거하기 위해서는 “-=” 을 사용한다.

DEFINE 매크로는 컴파일에서 옵션을 지정해 컴파일의 특성을 추가할 수 있다. DEFINE 매크로에 “QT_DEPRECATED_WARNINGS” 을 사용하면 Qt 버전 별 누락된 API를 사용했을 때 경고를 출력하라는 의미이다.

SOURCE 매크로는 소스파일을 지정한다. 여기서는 헤더 파일이 없지만, 만약 헤더 파일이 있다면 HAEDER 매크로에 헤더 파일을 지정한다. 그리고 마지막 3라인은 특정 플랫폼일 때 수행되는 예제이다.

qnx: target.path = /tmp/\${TARGET}/bin 은 QNX 라는 플랫폼일 때 수행되는 라인이다. 다음 라인은 QNX 가 아니라 리눅스 플랫폼일 경우 마지막 라인은 QNX 와 리눅스 둘다 아닌 경우를 의미 한다.

프로젝트파일은 다양한 구문과 IF문과 같은 조건 절을 사용해 다양한 옵션을 사용할 수 있다. 예를 들어 32비트 버전인 경우 64비트 버전인 경우에 따라 컴파일 옵션을 다르게 사용할 수 있다. 다음은 main.cpp 소스코드 이다.

```
#include <QCoreApplication>
#include <QDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    int myVal = 100;

    qDebug() << "Hello World.";
    qDebug("Qt First Project. My Value. %d", myVal);

    return a.exec();
}
```

Qt의 시작점은 main() 함수에서 첫 번째 라인은 QCoreApplication 클래스의 오브젝트를 선언한다. 이 오브젝트는 Qt라는 이벤트 루프를 통해 프로그램이 종료되지 않고 사용자 또는 이벤트로부터 요청을 처리한다. main() 함수 마지막 라인에 return 에서 a.exec() 멤버 함수를 사용해야 만 main() 함수 처리가 끝나더라도 프로그램이 종료되지 않는다.

argc 와 argv 인자는 main 함수에서 전달 받은 인자를 넘겨주면 된다. qDebug() 함수

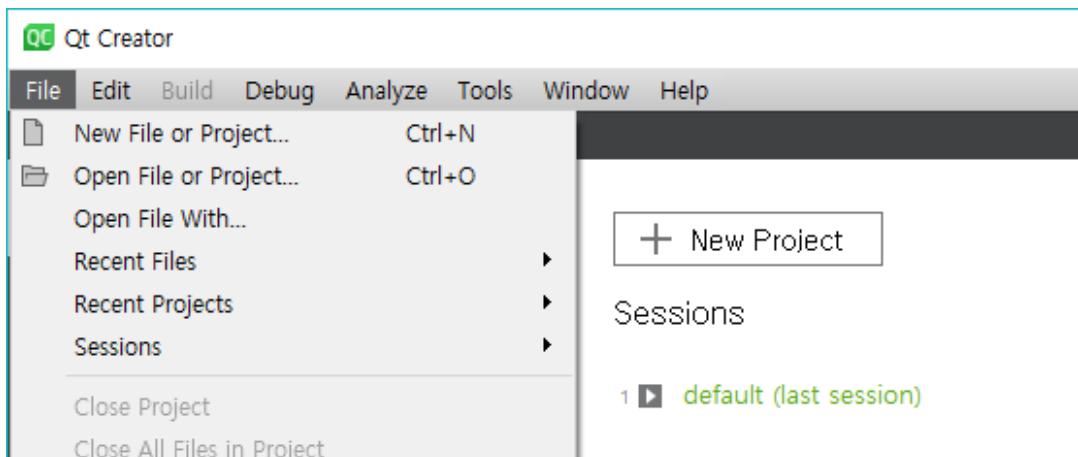
는 C언어에서 사용하는 printf() 함수와 동일 하며 자바에서의 System.out.println() 함수와 동일한 기능을 제공한다.

- GUI 기반의 “Hello World” 출력 예제 프로그래밍.

이번 예제에서는 간단한 GUI기반의 응용 어플리케이션을 구현해 보자. 이 어플리케이션은 GUI상에 버튼 위젯 하나를 배치할 것이다. 버튼의 TEXT로 “Hello World”를 입력한다. 이 버튼을 클릭하면 qDebug() 문을 이용해 “Hello World” 문자열을 디버깅 창에 출력한다.

Qt 는 GUI 기반의 어플리케이션 개발하는 방법으로 Qt Creator IDE에 내장된 디자이너 툴을 이용해 마우스로 드래그 하여 위젯(버튼 등과 같은 요소들)들을 배치할 수 있다.

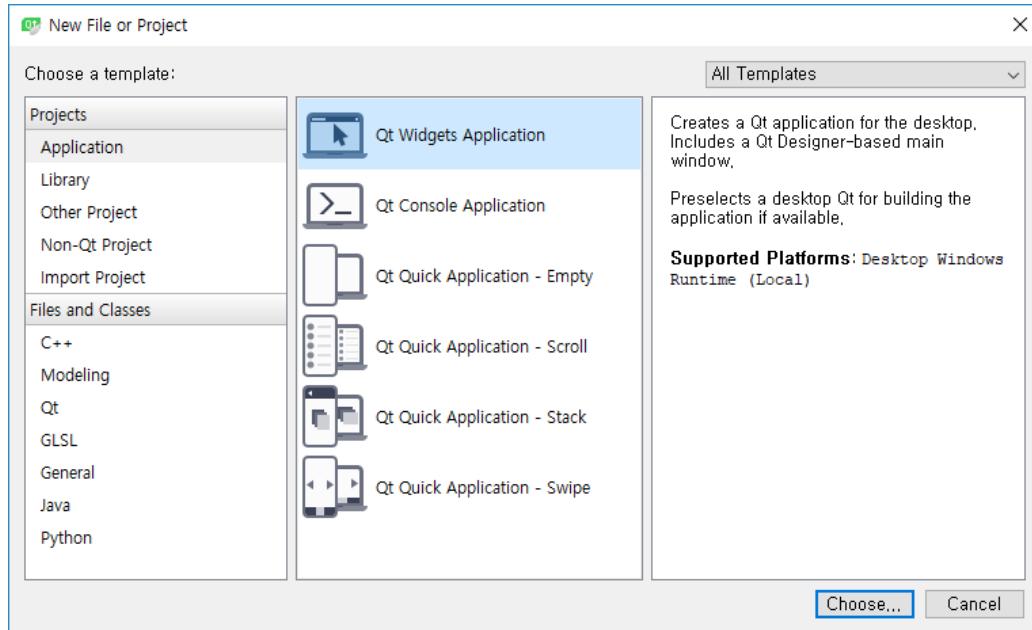
하지만 이 예제에서는 Qt 개발 프레임워크를 어떻게 개발해야 하는지 목적이 있으므로 디자이너 툴을 사용하지 않고 직접 GUI 코드를 C++로 작성해 보도록 하겠다. Qt Creator를 실행한 후 새로운 프로젝트를 생성한다.



<그림> 새로운 프로젝트 생성

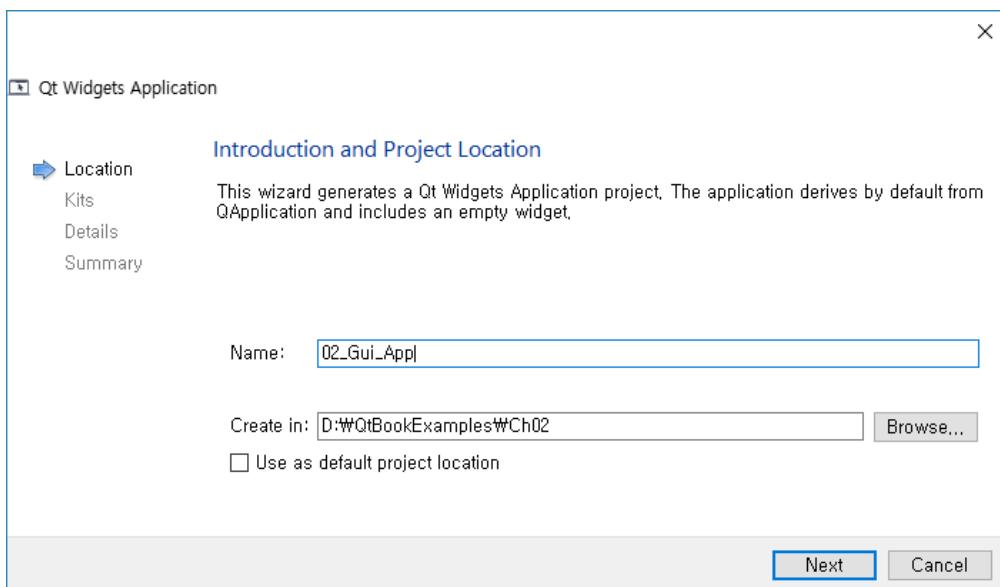
이전 그림에서 보는 것과 같이 [File] -> [New File or Project]를 클릭한다. [New File or Project] 메뉴를 클릭한다.

그러면 다음 그림에서 보는 것과 같이 다이얼로그 창에서 좌측 Projects메뉴 리스트에서 [Application] 을 선택하고 우측의 메뉴에서는 [Qt Widgets Application] 을 선택하고 하단의 [Choose] 버튼을 클릭한다.



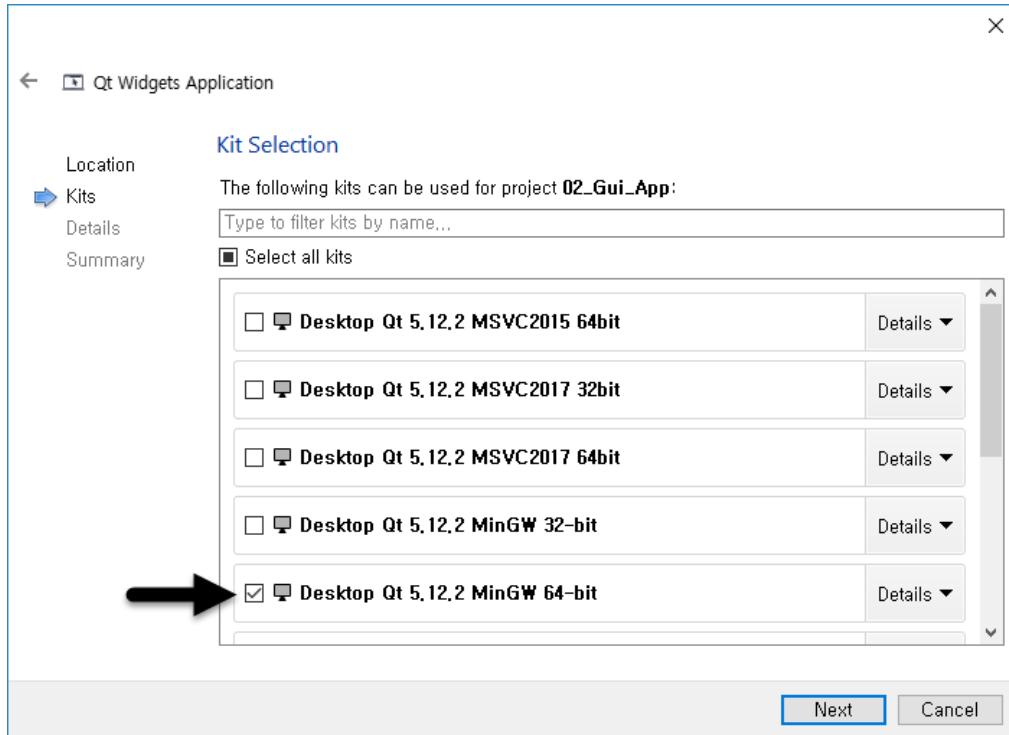
<그림> 템플릿 선택 다이얼로그

다음 다이얼로그 창에서 프로젝트 이름(Name) 과 프로젝트가 생성된 디렉토리의 위치(Create in) 을 입력한 후 하단의 [Next] 버튼을 클릭한다.



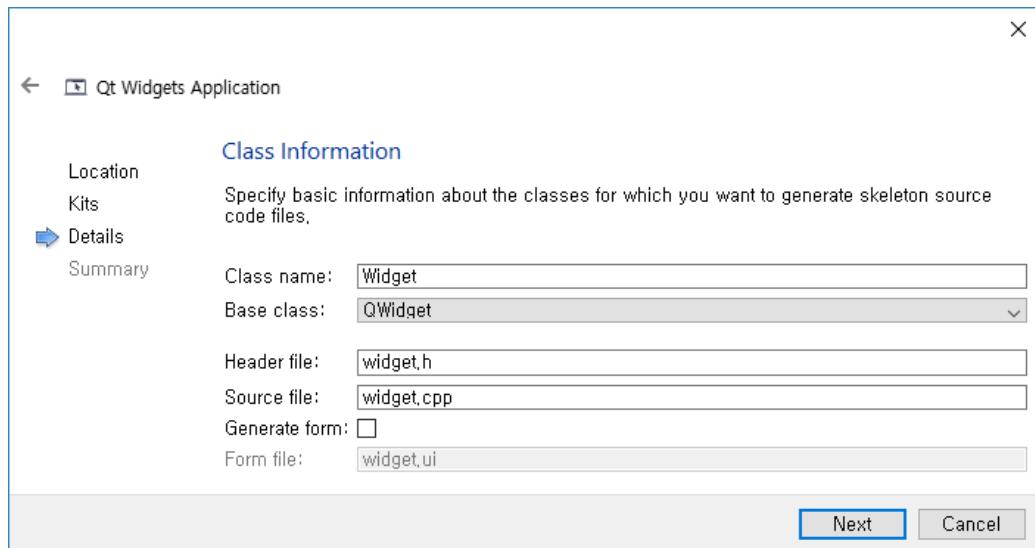
<그림> 프로젝트 이름과 위치 지정 다이얼로그

다음 다이얼로그는 개발할 어플리케이션을 빌드할 컴파일러를 선택하는 다이얼로그 창이다. 모든 체크박스를 해제 하고 다음그림에서 보는 것과 같이 MinGW 64-bit 버전을 선택하고 하단의 [Next] 버튼을 클릭한다.



<그림> 빌드할 컴파일러 선택 다이얼로그

다음은 프로젝트의 메인 클래스(처음 로딩되면 로딩되는 메인 윈도우 위젯 클래스)의 이름, 헤더 파일명, 소스 파일명 그리고 메인 클래스에 디자인 폼을 지정할 것인지 묻는 체크박스 그리고 디자인 폼의 파일명(.ui)을 지정한다.

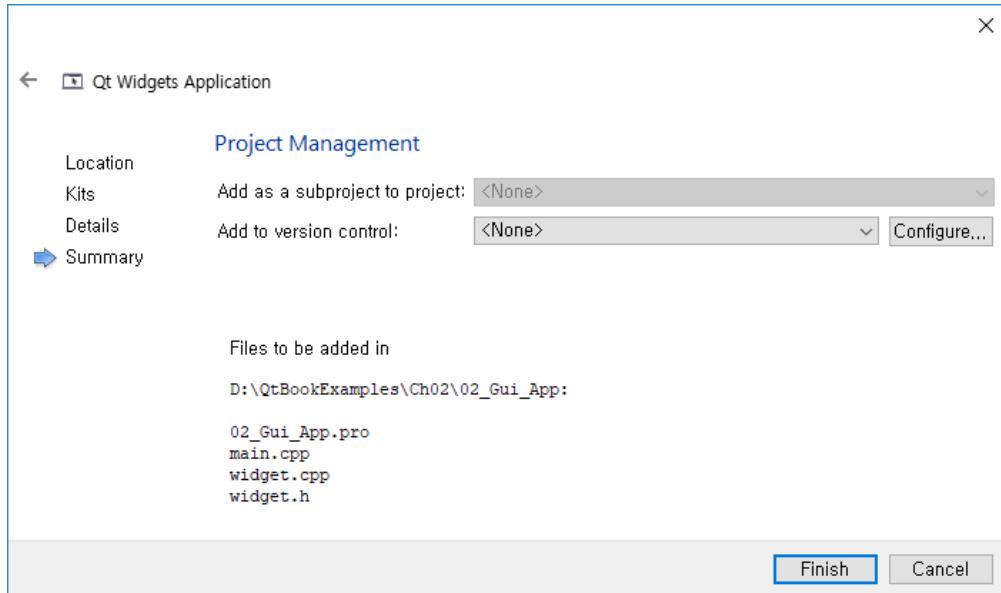


<그림> 클래스 정보 입력 다이얼로그

다이얼로그 창에서 첫 번째 [Class Name] 은 생성되는 메인 클래스의 이름을 지정한다.

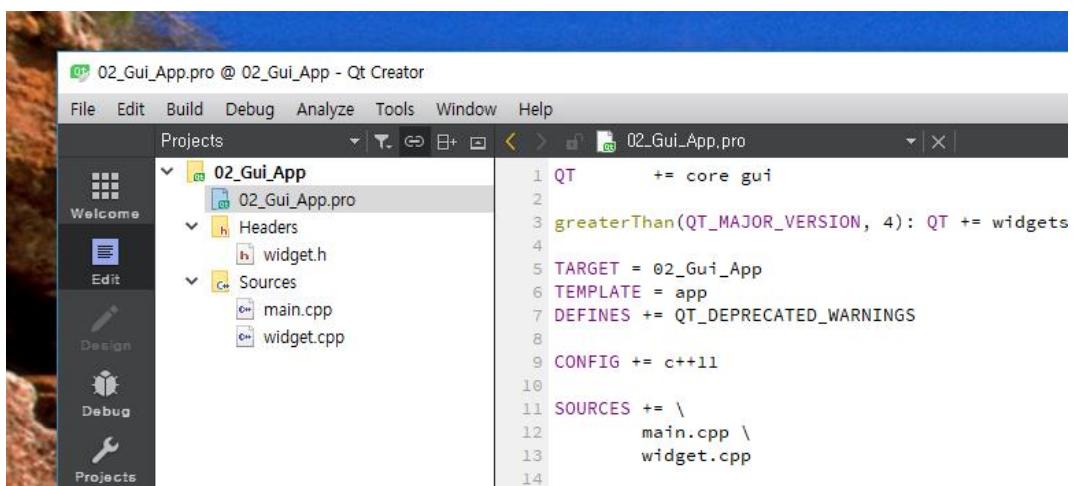
[Base class] 은 메인 클래스가 상속받는 클래스를 선택할 수 있다. 종류는 다음 표에서 보는 것과 같이 3가지 중 하나를 선택할 수 있다.

여기서는 두 번째 QWidget 을 Base Class로 선택한다. 헤더 파일명과 소스 파일명은 widget.h 와 widget.cpp 로 이름을 정하고 [Generate form] 항목은 이전 그림에서 보는 것과 같이 체크박스를 해제한다. 그리고 하단의 [Next] 버튼을 클릭한다.



<그림> Project Management 다이얼로그

이전 그림은 마지막 프로젝트의 마지막 생성 다이얼로그이다. 하단의 [Finish] 버튼을 클릭하면 Qt Creator 가 위젯 생성에 필요한 기본적인 프로젝트파일(.pro) 파일, main.cpp, widget.h 그리고 widget.cpp 파일을 자동으로 생성해준다.



<그림> 프로젝트 생성 완료 후 Qt Creator 화면

다음은 프로젝트파일(.pro) 소스 코드이다.

```
QT      += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = 02_Gui_App
TEMPLATE = app
DEFINES += QT_DEPRECATED_WARNINGS
CONFIG += c++11

SOURCES += \
    main.cpp \
    widget.cpp
HEADERS += \
    widget.h

qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

첫 번째 라인의 QT 매크로(키워드)는 이 프로젝트에서 사용할 모듈을 명시한다. 두 번째 라인은 만약 이 프로젝트가 사용하는 Qt의 메인 상위 버전이 5.x.x 버전이 아닌 4.x.x 버전을 사용할 경우 widgets라는 모듈을 추가한다는 의미이다.

TARGET 매크로는 프로젝트의 이름을 명시한다. TEMPLATE 은 이 프로젝트가 라이브러리 인지 어플리케이션인지와 같은 종류를 선택한다. 이 프로젝트는 어플리케이션 이므로 app 를 명시한다. DEFINE 매크로에 “QT_DEPRECATED_WARNINGS” 을 사용하면 Qt 버전 별 누락된 API를 사용했을 때 경고를 출력한다.

CONFIG 매크로에서 사용할 수 있는 옵션은 많은 옵션을 제공한다. 제사한 옵션에 대한 사항은 Qt Assistant 도움말을 이용하면 어떤 옵션이 있는지 자세히 살펴볼 수 있다. 여기서 사용한 c++11 은 Qt가 사용하는 C++ 버전을 의미한다. SOURCES와 HEADERS 매크로는 이 프로젝트의 소스코드 파일과 헤더 파일을 명시한다. 그리고 마지막 3라인은 특정 플랫폼일 때 수행되는 예제이다.

qnx: target.path = /tmp/\$\${TARGET}/bin 은 QNX 라는 플랫폼일 때 수행되는 라인이다. 다음 라인은 QNX 가 아니라 리눅스 플랫폼일 경우 마지막 라인은 QNX 와 리눅스 둘 다 아닌 경우를 의미 한다.

프로젝트파일은 다양한 구문과 C와 같은 Syntax 구조와 같은 조건 절을 사용해 다양한 옵션을 사용할 수 있다. 예를 들어 32비트 버전인 경우 64비트 버전인 경우에 따라 컴

파일 옵션을 다양하게 사용할 수 있다. 다음은 main.cpp 예제 소스코드 이다.

```
#include "widget.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    Widget mywidget;
    mywidget.show();

    return a.exec();
}
```

이 프로젝트는 GUI 기반이므로 QCoreApplication 대신 QApplication 을 사용한다. 생성된 Widget 클래스의 mywidget 오브젝트이다. 이 클래스는 윈도우 위젯이므로 소스 코드에서 보는 것과 같이 오브젝트를 선언한다. show() 멤버 함수는 GUI 위젯을 화면에 표시하는 기능을 제공한다. 다음 예제는 widget.h 소스코드 이다.

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QPushButton>

class Widget : public QWidget
{
    Q_OBJECT
public:
    Widget(QWidget *parent = 0);
    ~Widget();

private:
    QPushButton *btn;
    QString str;

public slots:
    void slot_btn();
};

#endif // WIDGET_H
```

private 키워드에 명시된 QPushButton 클래스 위젯은 버튼 위젯이다. 소스코드 에서 보

는 것과 같이 btn 오브젝트를 선언하고 QString 은 Qt에서 제공하는 문자열 처리 클래스 변수 이다.

public slots 라는 것은 이벤트 함수 이다. 이 이벤트에서는 SLOT 함수라고 불린다. 이 Slot 함수는 어떤 이벤트가 발생하면 실행되는 함수를 지정할 때 사용한다. 따라서 이 프로젝트에서는 "Hello World" 라는 btn 을 클릭하면 slot_btn() 함수를 호출한다. 다음은 widget.cpp 예제 소스코드 이다.

```
#include "widget.h"
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent)
{
    setFixedSize(300, 200);

    str = QString("Hello World");
    btn = new QPushButton(str, this);
    btn->setGeometry(10, 10, 100, 30);

    connect(btn, &QPushButton::clicked, this, &Widget::slot_btn);
}

void Widget::slot_btn()
{
    qDebug() << "Hello World button clicked.!!";
}

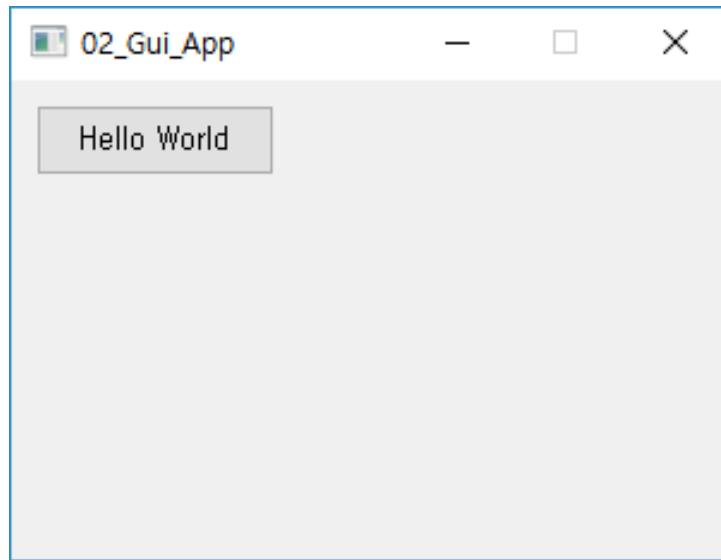
Widget::~Widget()
```

setFixedSize() 함수는 이 위젯의 크기를 지정하기 위한 멤버 함수 이다. 첫 번째 인자는 가로 크기이고 두 번째 인자는 세로 크기이며 단위는 Pixel 이다.

다음 라인의 str 변수는 헤더에 명시했던 QString 클래스 변수이다. 이 클래스 변수에 "Hello World" 문자열을 저장한다. connect() 함수는 btn 버튼의 클릭 이벤트가 발생하면 slot_btn() 함수를 호출한다.

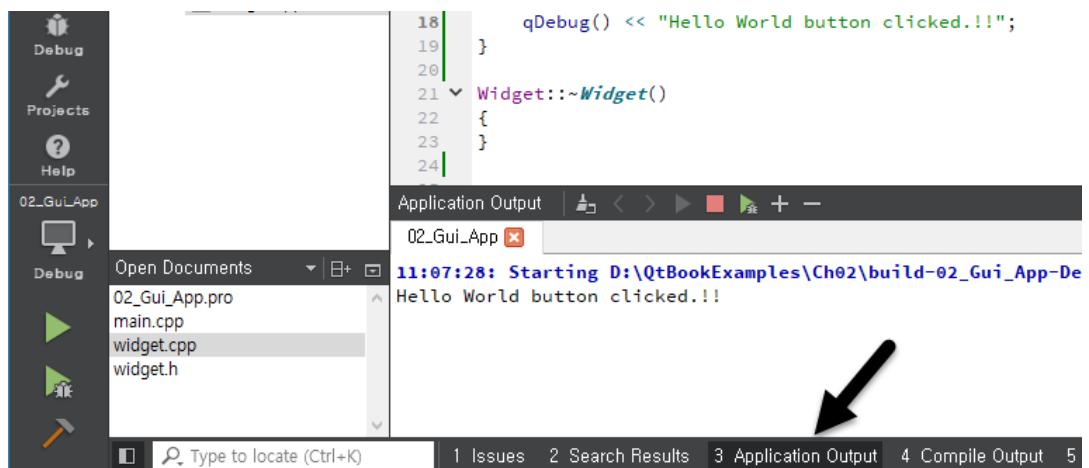
첫 번째 인자는 이벤트를 발생하는 오브젝트, 두 번째는 이벤트의 종류이다. 이벤트의 종류로 clicked, mouse over, released 등이 있다. 세 번째 인자는 이벤트를 받는 클래스, 네 번째 이벤트의 이벤트를 받는 클래스의 헤더파일에서 SLOT키워드에서 정의 함수를

지정한다. Qt 의 이벤트는 Signal 과 Slot 을 사용한다. 다음 장에서는 자세히 다루겠으며 여기서는 기본적인 구조만 살펴보고 넘어가자.



<그림> 어플리케이션 실행 화면

어플리케이션을 빌드하고 실행하면 이전 그림에서 보는 것과 같이 원도우 창에 "Hello World" 버튼을 클릭하면 slot_btn() 함수가 호출된다. 이 함수에는 qDebug() 문을 이용해 "Hello World button clicked.!!" 문을 디버깅 창에 출력한다. qDebug() 함수를 이용해 출력한 문자열은 Qt Creator 하단의 [Application Output] 탭을 클릭하면 출력 로그를 확인할 수 있다.



<그림> 어플리케이션 실행 후 qDebug() 출력 화면

3. Qt Basic

이번 장에서는 Qt 어플리케이션 개발에 필요한 필수적인 요소에 대해 살펴보자. 이번 장에서 다룰 내용은 다음 표에서 보는 것과 같다.

<표> 이번 장에서 다룰 내용의 요약

다룰 내용의 각 제목	설명
Qt GUI Widgets	Qt에서 제공하는 GUI 위젯들의 사용 방법을 설명
레이아웃	GUI의 동적 크기 변환을 위한 레이아웃 사용 방법
Qt에서 제공하는 데이터 탑재와 클래스	데이터 탑재과 클래스를 사용하기 위한 방법.
Signal 과 Slot	Qt에서 제공하는 Signal 과 Slot 을 이용한 이벤트 처리
Qt Designer를 이용한 GUI 설계 및 제작	GUI 인터페이스를 이용해 마우스로 GUI 위젯을 배치할 수 있는 Qt Designer 툴 사용 방법.
다이얼로그	다이얼로그 메시지
MDI 기반 윈도우 GUI 구현	Multi Document Interface 와 Single Document Interface 차이점과 구현 방법
파일 처리와 데이터 스트림	파일 및 대용량의 데이터를 스트림을 이용해 효율적으로 처리하기 위한 방법
Qt Property	Meta-Object System 기반의 객체 간 통신
QTimer	QTimer 를 이용해 반복적인 이벤트를 처리
QThread	쓰레드 구현
Qt Linguist Tools	Qt Linguist 툴을 이용한 다국어 처리
라이브러리 제작	Qt를 이용한 라이브러리 제작
Model 과 View	표와 같은 형태의 위젯에 데이터를 표시하기 위한 방법

3.1. Qt GUI Widgets

이번 절에서는 다음 그림에서 보는 것과 같이 Qt에서 제공하는 GUI 위젯들에 대해 살펴보도록 하자.

- QCheckBox 와 QButtonGroup

QCheckBox 클래스 위젯은 여러 항목을 선택할 수 있는 GUI 인터페이스를 제공 한다. 첫 번째 인자는 체크박스 항목 텍스트이다. 두 번째 인자는 부모 클래스를 지정한다.

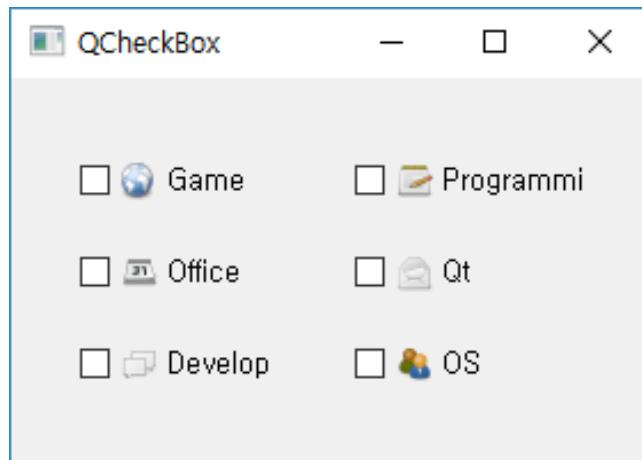
```
QCheckBox *chk = new QCheckBox("P&rogramming", this);
```

위의 "P&rogramming" 문자열에서 사용한 & 문자는 체크 박스의 단축키를 사용할 수 있으며 단축키는 [Alt + R] 키를 누르면 체크박스를 활성화 혹은 비활성화 시킬 수 있다. 체크박스를 그룹으로 분리하기 위해 아래 예제 소스코드와 같이 QButtonGroup을 사용하면 된다.

```
QButtonGroup *group = new QButtonGroup(this);
```

첫 번째 인자는 부모 클래스를 지정한다. QButtonGroup 클래스의 addButton() 멤버 함수를 이용해 체크박스를 그룹에 포함시킬 수 있다.

체크 박스의 특징은 여러 항목 중 다중 선택할 수 있는 특징이 있지만 QButtonGroup 클래스의 setExclusive() 멤버 함수를 사용하면 단일 항목만 선택할 수 있도록 체크 박스의 성질을 변경할 수 있다.



<그림> QCheckBox 예제 실행 화면

```
QString str1[3] = {"Game", "Office", "Develop"};
QString str2[3] = {"Programming", "Qt", "OS"};

int xpos = 30;
int ypos = 30;

chk_group[0] = new QButtonGroup(this);
chk_group[1] = new QButtonGroup(this);

for(int i = 0 ; i < 3 ; i++) {
    exclusive[i] = new QCheckBox(str1[i], this);
    exclusive[i]->setGeometry(xpos, ypos, 100, 30);
    chk_group[0]-> addButton(exclusive[i]);

    non_exclusive[i] = new QCheckBox(str2[i], this);
    non_exclusive[i]->setGeometry(xpos + 120, ypos, 100, 30);
    chk_group[1]-> addButton(non_exclusive[i]);

    connect(exclusive[i], SIGNAL(clicked()), this, SLOT(chkChanged()));
    ypos += 40;
}

chk_group[0]->setExclusive(false);
chk_group[1]->setExclusive(true);
...
```

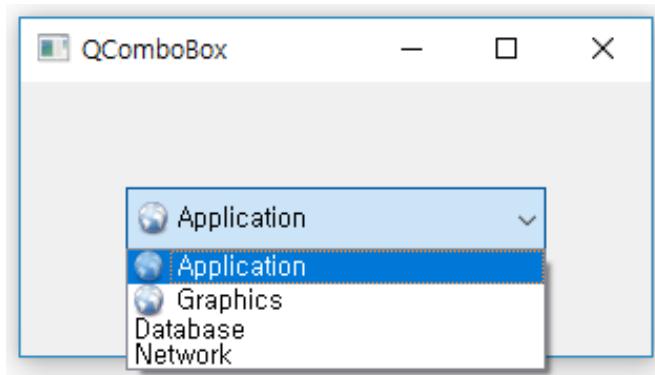
이 예제는 첫 번째 그룹에 등록된 체크 박스들과 두 번째 그룹에 등록된 체크박스들은 그룹으로 분리 하였기 때문에 두 그룹간에 영향을 미치지 않는다. 예제 전체 소스는 Ch03 > 01_BasicWidget > 01_QCheckBox 디렉토리를 참조하면 된다.

● QComboBox

사용자가 위젯을 클릭하면 팝업 메뉴가 나타나고 등록된 항목 중 하나를 선택할 수 있는 GUI를 제공한다. QComboBox 위젯 상에 아이템을 등록하기 위해 텍스트 또는 아이템에 이미지를 사용해 텍스트를 함께 사용할 수 있다.

```
combo = new QComboBox(this);
combo->setGeometry(50, 50, 200, 30);
...
```

```
combo->addItem(QIcon(":/resources/browser.png"), "Application");
combo->addItem(QIcon(":/resources/browser.png"), "Graphics");
combo->addItem("Database");
combo->addItem("Network");
...
```



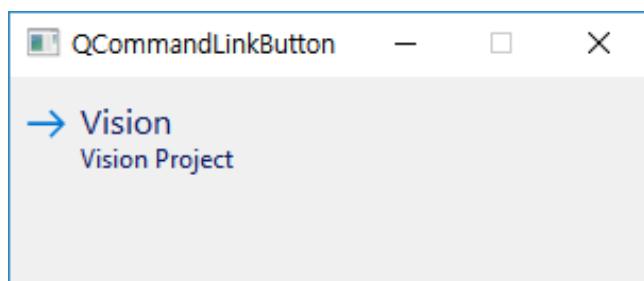
<그림> QComboBox 예제 실행 화면

예제 전체 소스는 Ch03 > 01_BasicWidget > 02_QComboBox 디렉토리를 참조하면 된다.

- QCommandLinkButton

이 위젯은 QPushButton 위젯과 동일한 기능을 제공하는 위젯이다. 특징으로 이 위젯은 MS Windows에서 제공하는 Link Button과 같은 스타일을 제공한다.

```
cmmBtn = new QCommandLinkButton("Vision", "Vision Project", this);
cmmBtn->setFlat(true);
...
```



<그림> QCommandLinkButton 예제 실행 화면

- QDate 클래스와 QDateEdit 위젯 클래스

QDateEdit는 날짜를 표시하거나 변경할 수 있는 GUI를 제공한다. QDateEdit는 QDate 클래스에 설정된 날짜를 표시할 수 있다. QDate 클래스는 년, 월, 일을 지정하거나 현재 날짜를 시스템으로부터 얻어와 QDateEdit 위젯에 연결해 표시할 수 있다.

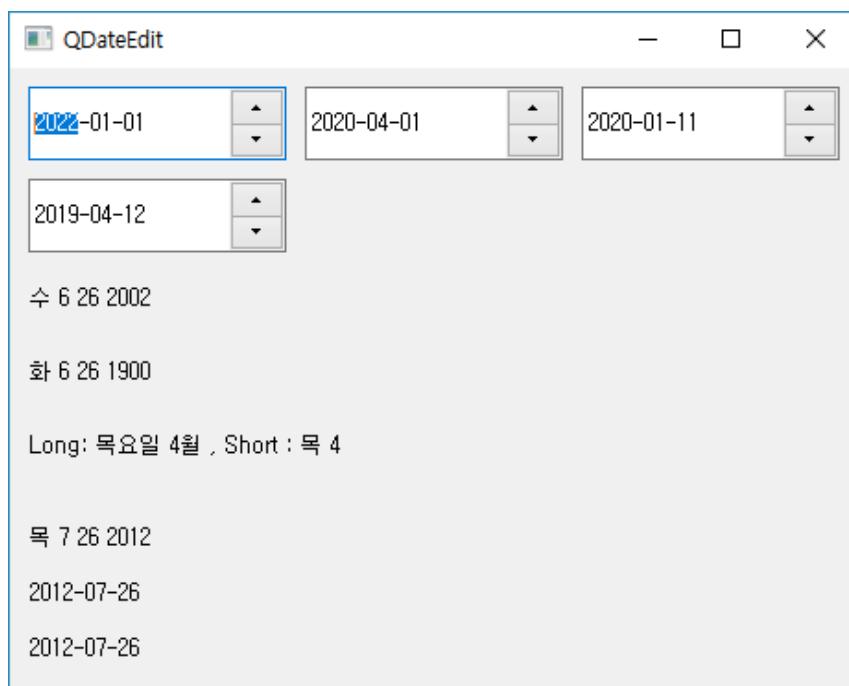
```
QDate dt1 = QDate(2020, 1, 1);
QDate dt2 = QDate::currentDate();

dateEdit[0] = new QDateEdit(dt1.addYears(2), this);
dateEdit[0]->setGeometry(10, 10, 140, 40);

dateEdit[1] = new QDateEdit(dt1.addMonths(3), this);
dateEdit[1]->setGeometry(160, 10, 140, 40);

dateEdit[2] = new QDateEdit(dt1.addDays(10), this);
dateEdit[2]->setGeometry(310, 10, 140, 40);

dateEdit[3] = new QDateEdit(dt2, this);
dateEdit[3]->setGeometry(460, 10, 140, 40);
...
```



<그림> QDate 와 QDateEdit 예제 실행 화면

QDate 클래스는 원하는 포맷으로 날짜를 표시하기 위한 방법으로 QString 데이터 타입을 리턴 하는 멤버 함수를 이용하면 날짜를 다양하게 표현할 수 있다.

```
QDate dt = QDate::currentDate();
QString str = dt.toString("yyyy.MM.dd");
```

<표> 날짜 표시 포맷

표현 문자	표시 형태
d	1~31 표시 (1~31)
dd	1~31 표시하되 2자리로 표시 (01~31)
ddd	3자리 문자로 요일을 표시 (Mon ~ Sun)
dddd	완전한 문자로 요일을 표시 (Monday ~ Sunday)
M	1~12 숫자로 표시 (1~12)
MM	01~12 숫자로 표시하되 2자리로 표시 (01~12)
MMM	월을 3자리 문자로 표시 (Jan ~ Dec)
MMMM	월을 완전한 문자로 표시 (January ~ December)
yy	년을 2자리로 표시 (00~99)
yyyy	년을 4자리로 표시 (2002)

<표> QDate 표시 스타일 변경 가능한 타입

12	Value	설명
Qt::TextDate	0	디폴트 타입 표시
Qt::ISODate	1	ISO8601 확장된 포맷으로 표시
Qt::SystemLocaleDate	2	국가의 표현 방식으로 표시

QDate 클래스는 날짜를 표시하는 스타일을 다음과 같은 타입들로 변경할 수 있다.

```
QDate dt6 = QDate(2012, 7, 26);
lbl[3] = new QLabel(dt6.toString(Qt::TextDate), this);
lbl[3]->setGeometry(10, 240, 250, 30);
```

```

lbl[4] = new QLabel(dt6.toString(Qt::ISODate), this);
lbl[4]->setGeometry(10, 270, 250, 30);

lbl[5] = new QLabel(dt6.toString(Qt::SystemLocaleDate), this);
lbl[5]->setGeometry(10, 300, 250, 30);
...

```

예제 전체 소스는 Ch03 > 01_BasicWidget > 04_QDateEdit 디렉토리를 참조하면된다.

- QTime 클래스와 QTimeEdit 위젯 클래스

QTime 클래스는 현재 시스템 설정 시간과 현재 시간으로부터 특정 시간까지의 경과 시간을 표시하거나 시간을 비교하는 등 어플리케이션 개발에 필요한 다양한 시간 관련 기능을 쉽게 구현할 수 있다. QTimeEdit는 QTime 클래스로부터 얻어온 시간을 GUI 인터페이스 상에 표시할 수 있는 기능을 제공하는 위젯이다.

```

QTime ti = QTime(6, 24, 55, 432); // 시, 분, 초, 밀리세컨드초

QTimeEdit *qte;
qte = new QTimeEdit(ti, this);
qte->setGeometry(10, 30, 150, 30);
...

```

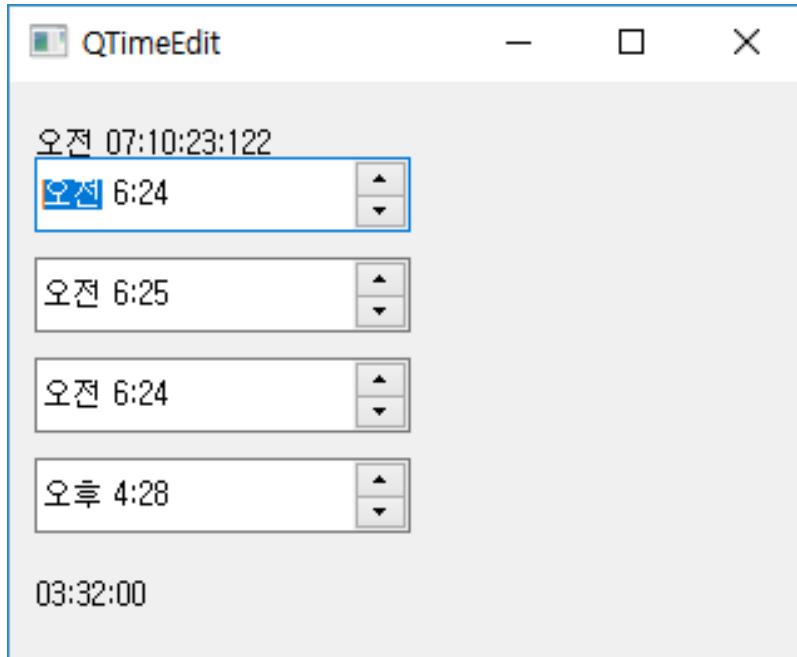
QTime 클래스는 시간 관련 조작을 쉽게 하기 위해 다양한 멤버 함수를 제공한다. 예를 들어 현재 시간에 초와 밀리세컨드초를 지정한 값만큼 더하기 위해서 addSecs() 와 addMSecs() 멤버 함수를 이용하면 해당 시간에 초와 밀리세컨드초를 더해 원하는 결과를 얻을 수 있다.

```

qte[1] = new QTimeEdit(ti1.addMSecs(200), this);
qte[1]->setGeometry(10, 30, 150, 30);

qte[2] = new QTimeEdit(ti1.addSecs(2), this);
qte[2]->setGeometry(10, 30, 150, 30);
...

```



<그림> QTime 과 QTimeEdit 예제 실행 화면

QTime 클래스의 유용한 기능 중 하나는 start() 멤버 함수가 선언된 시점부터 elapsed() 멤버 함수가 호출된 소스 코드 라인까지 처리하는 데 경과된 시간을 얻어오는 함수로 사용할 수 있다.

```
QTime ti3;
ti3.start();

for(int i = 0 ; i < 100000 ; i++)
{
    for(int j = 0 ; j < 10000 ; j++)
}

qDebug("Elapsed Time : %d", ti3.elapsed());
...
```

QTime 클래스는 toString() 멤버 함수를 이용해 다양한 포맷으로 시간을 표시할 수 있다 .

```
QTime ti = QTime(7, 10, 23, 122);
QLabel *lb_str = new QLabel(ti.toString("AP hh:mm:ss:zzz"), this);

lb_str->setGeometry(10, 10, 150, 30);
...
```

<표> 시간 표시 포맷

표현 문자	표시 형태
h	Hour를 0~23 숫자로 표시
hh	Hour를 00~23 숫자로 표시 (항상 2자리로 표시)
m	Minute를 0~59로 표시
mm	Minute를 00~59로 표시
s	Second를 0~59로 표시
ss	Second를 00~59로 표시
z	밀리 초를 표시 0~999로 표시
zzz	밀리 초를 표시 000~999로 표시
AP	오전/오후를 표시하는 AM/PM 문자를 대문자로 표시
ap	오전/오후를 표시하는 AM/PM 문자를 소문자로 표시

예제 전체소스는 Ch03 > 01_BasicWidget > 05_QTimeEdit 디렉토리를 참조하면 된다.

- QDateTime 클래스와 QDateTimeEdit 위젯

QDateTime 클래스는 날짜와 시간을 함께 다룰 수 있는 클래스이며 QDateTimeEdit 클래스는 날짜와 시간을 표시할 수 있는 GUI를 제공한다. QDateTimeEdit 클래스의 setDisplayFormat() 멤버 함수는 포맷에 따라 날짜와 시간을 표시할 수 있다.

```
QDateTimeEdit *qde1;
qde1 = new QDateTimeEdit(QDateTime::currentDateTime(), this);
qde1->setDisplayFormat( "yyyy-MM-dd hh:mm:ss:zzz" );
qde1->setGeometry(10, 30, 250, 50); // x, w, width, height
```

QDateTimeEdit는 위젯에 표시된 날짜와 시간을 스픈 박스 버튼으로 변경할 수 있으며 날짜와 시간을 변경할 때 범위를 지정할 수 있다.

```
QDateTimeEdit *qde[3];

qde[ 0 ] = new QDateTimeEdit(QDate::currentDate(), this);
qde[ 0 ]->setMinimumDate(QDate::currentDate().addYears(-3));
qde[ 0 ]->setMaximumDate(QDate::currentDate().addYears(3));
```

```

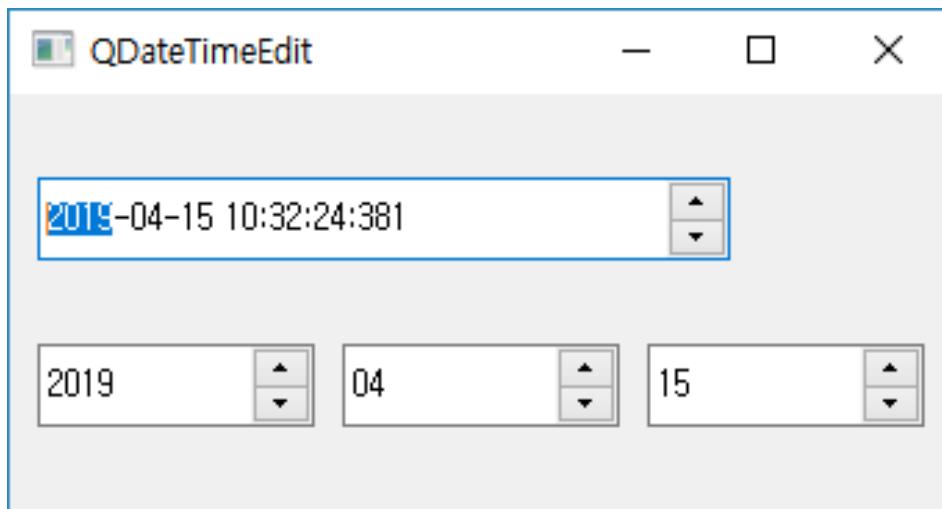
qde[0]->setDisplayFormat("yyyy");
qde[0]->setGeometry(10, 90, 100, 30);

qde[1] = new QDateEdit(QDate::currentDate(), this);
qde[1]->setMinimumDate(QDate::currentDate().addMonths(-2));
qde[1]->setMaximumDate(QDate::currentDate().addMonths(2));
qde[1]->setDisplayFormat("MM");
qde[1]->setGeometry(120, 90, 100, 30);

qde[2] = new QDateEdit(QDate::currentDate(), this);
qde[2]->setMinimumDate(QDate::currentDate().addDays(-20));
qde[2]->setMaximumDate(QDate::currentDate().addDays(20));
qde[2]->setDisplayFormat("dd");
qde[2]->setGeometry(230, 90, 100, 30);
...

```

setMinimumDate()와 setMaximumDate() 멤버 함수는 범위를 지정하여 설정된 범위 내에서만 날짜를 변경할 수 있다.



<그림> QDateTime 과 QDateTimeEdit 클래스를 이용한 예제 실행 화면

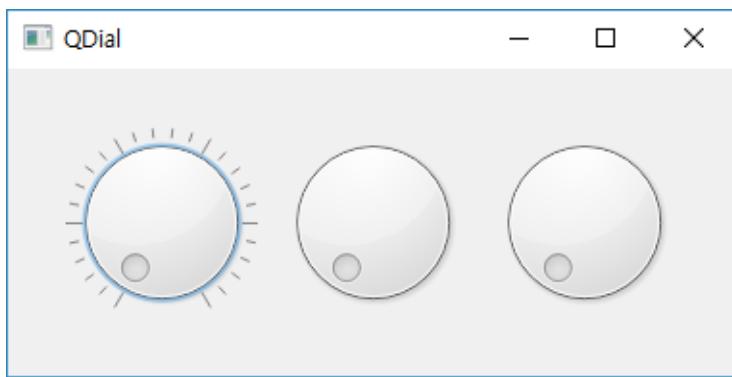
QDateTimeEdit 클래스는 날짜 외에도 setMinimumTime() 와 setMaximumTime() 멤버 함수를 이용해 시간의 최소 값과 최대 값을 지정할 수 있다. 예제 전체 소스는 Ch03 > 01_BasicWidget > 06_QDateTimeEdit 디렉토리를 참조하면 된다.

- QDial

QDial 위젯 클래스는 다이얼과 같은 GUI 인터페이스를 제공한다. 예를 들어 볼륨 조절

시 다이얼을 돌려 조절하는 것과 같은 GUI를 제공한다.

```
for(int i = 0 ; i < 3 ; i++, xpos += 110) {  
    dial[i] = new QDial(this);  
    dial[i]->setRange(0, 100);  
    dial[i]->setGeometry(xpos, 30, 100, 100);  
}  
dial[0]->setNotchesVisible(true);  
connect(dial[0], &QDial::valueChanged, this, &Widget::changedData);  
...
```



<그림> QDial 예제 실행 화면

setRange() 멤버 수는 QDial 위젯의 범위를 지정할 수 있다. setNotchesVisible() 멤버 수는 QDial 위젯에 눈금을 표시할 수 있 기능을 제공한다. QDial 위젯을 마우스로 드래그하면 valueChanged() 시그널을 등록해 변경한 QDial의 현재 값을 얻을 수 있다.

```
Widget::Widget(QWidget *parent) : QWidget(parent)  
{  
    int xpos = 30;  
    for(int i = 0 ; i < 3 ; i++, xpos += 110) {  
        dial[i] = new QDial(this);  
        dial[i]->setRange(0, 100);  
        dial[i]->setGeometry(xpos, 30, 100, 100);  
    }  
    dial[0]->setNotchesVisible(true);  
    connect(dial[0], &QDial::valueChanged, this, &Widget::changedData);  
}  
  
void Widget::changedData()  
{
```

```

    qDebug("QDial 1 value : %d", dial[0]->value());
}
...

```

QDial 의 첫 번째 다이얼로그의 값이 변경되면 changeData() Slot 함수가 호출된다. 아직 Signal / Slot 을 이용한 이벤트 처리를 배우지 않았다. 따라서 여기서는 connect() 함수를 이용해 발생한 이벤트를 처리 하는 함수라는 Slot 함수라는 정도만 이해 하고 넘어가자. 자세한 사항은 Signal 과 Slot 절에서 다루도록 하겠다. 예제 소스는 Ch03 > 01_BasicWidget > 07_QDail 디렉토리를 참조하면 된다.

- QSpinBox 와 QDoubleSpinBox

QSpinBox 클래스는 int형 데이터 타입의 정수 값을 상하 버튼을 이용해 변경할 수 있는 GUI를 제공한다. double 데이터 타입을 사용하기 위해서는 QDoubleSpinBox 위젯을 사용하면 된다.

QSpinBox와 QDoubleSpinBox 위젯 클래스는 사용자가 변경할 수 있는 값의 범위를 제한할 수 있으며 숫자가 표시되는 Prefix와 Suffix 부분에 특정 문자 혹은 단위를 가리키는 문자를 사용할 수 있다. 예를 들어 화폐 기호를 위젯 안에 사용할 수 있다.

```

...
int ypos = 30;
int val[3] = {50, 100, 200};
double double_val[3] = {50.5, 127.32, 171.342};

for(int i = 0 ; i < 3 ; i++)
{
    spin[i] = new QSpinBox(this);
    spin[i]->setMinimum(10);
    spin[i]->setMaximum(300);
    spin[i]->setValue(val[i]);
    spin[i]->setGeometry(10, ypos, 100, 30);

    doublespin[i] = new QDoubleSpinBox(this);
    doublespin[i]->setMinimum(10.0);
    doublespin[i]->setMaximum(300.0);
    doublespin[i]->setValue(double_val[i]);
    doublespin[i]->setGeometry(120, ypos, 100, 30);

    spin[i]->setPrefix("$ ");
}

```

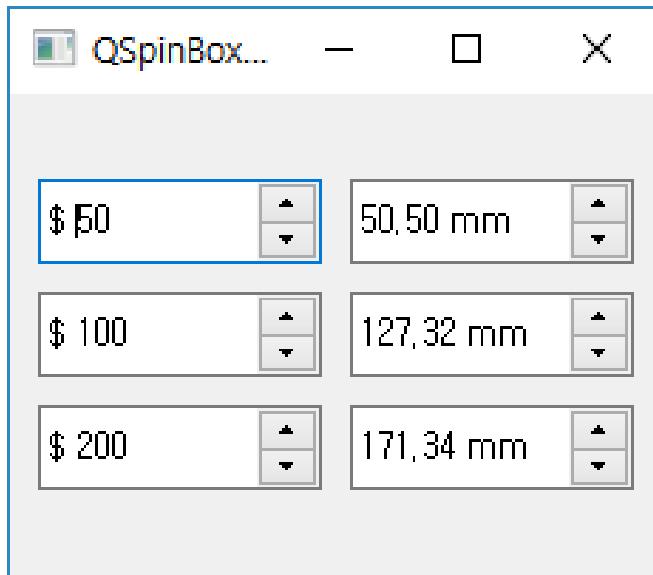
```

doublespin[i]->setSuffix(" mm");

ypos += 40;
}

...

```



<그림> QSpinBox 와 QDoubleSpinBox 예제 실행 화면

예제 전체 소스는 Ch03 > 01_BasicWidget > 08_QSpinBox_QDoubleSpinBox 디렉토리를 참조하면 된다.

- QPushButton 과 QFocusFrame

QPushButton 위젯은 버튼 기능을 제공한다. QFocusFrame은 바깥쪽에 Outer Line을 사용해야 할 경우 QFocusFrame을 사용하면 유용하다. 이 외에도 QFocusFrame은 QStyle (HTML에서 사용하는 Style Sheet) 을 사용 할 수 있다. QPushButton 위젯 바깥쪽에 Outer Line을 그리기 위해 QFocusFrame을 사용하는 방법은 다음과 같다.

```

QPushButton *btn[3];

int ypos = 30;
for(int i = 0 ; i < 3 ; i++) {
    QString str = QString("Frame's button %1").arg(i);
    btn[i] = new QPushButton(str, this);

```

```

btn[i]->setGeometry(10, ypos, 300, 40);
ypos += 50;
}

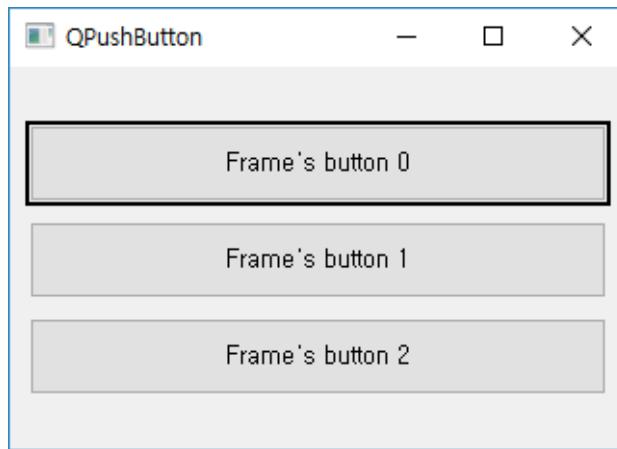
connect(btn[0], &QPushButton::clicked, this, &Widget::btn_click);
connect(btn[0], &QPushButton::pressed, this, &Widget::btn_pressed);
connect(btn[0], &QPushButton::released, this, &Widget::btn_released);

QFocusFrame *btn_frame = new QFocusFrame(this);

btn_frame->setWidget(btn[0]);
btn_frame->setAutoFillBackground(true);
...

```

QPushButton 클래스 생성자의 첫 번째 인자는 버튼에 표시할 텍스트를 입력한다. 두 번째 인자는 QPushButton 클래스의 부모 클래스를 지정한다.



<그림> QPushButton 과 QFocusFrame 예제 화면

예제 전체 소스는 Ch03 > 01_BasicWidget > 09_QPushButton_QFocusFrame 디렉토리를 참조하면 된다.

- QFontComboBox

QFontComboBox 위젯은 GUI상에 폰트를 선택하기 위한 GUI를 제공한다. 이 위젯은 폰트가 알파벳 순서로 나열되고, 폰트의 모양도 확인할 수 있다.

```

QFontComboBox *fontcb[5];
for(int i = 0 ; i < 5 ; i++)
    fontcb[i] = new QFontComboBox(this);

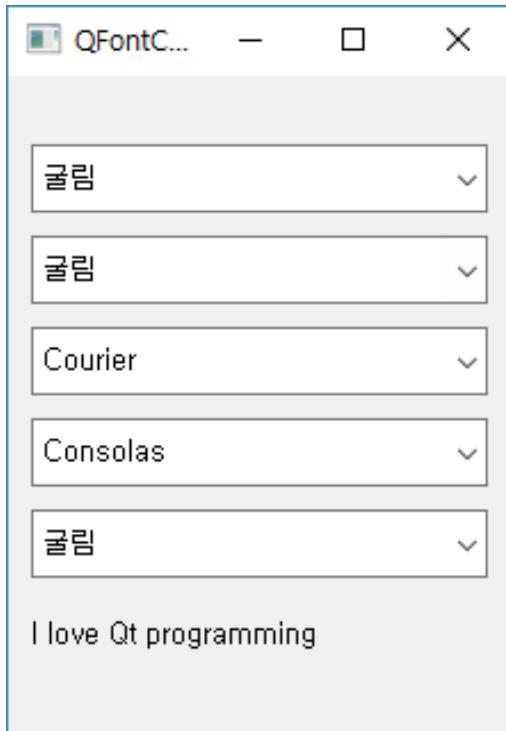
```

```

fontcb[0]->setFontFilters(QFontComboBox::AllFonts);
fontcb[1]->setFontFilters(QFontComboBox::ScalableFonts);
fontcb[2]->setFontFilters(QFontComboBox::NonScalableFonts);
fontcb[3]->setFontFilters(QFontComboBox::MonospacedFonts);
fontcb[4]->setFontFilters(QFontComboBox::ProportionalFonts);

int ypos = 30;
for(int i = 0 ; i < 5 ; i++) {
    fontcombo[i]->setGeometry(10, ypos, 200, 30);
    ypos += 40;
}
...

```



<그림> 예제 실행 화면

<표> setFontFilters() 멤버함수에서 사용 가능한 상수

상수	값	설명
QFontComboBox::AllFonts	0	모든 폰트
QFontComboBox::ScalableFonts	0x1	확대/축소시 동적 자동 변환 가능한 폰트

setFontFilters() 는 QFontComboBox 위젯 상에서 나열할 폰트 목록을 상수를 통해 특정 폰트를 Filtering 기능을 이용해 표시하는 기능을 제공한다.

예제 소스는 Ch03 > 01_BasicWidget > 10_QFontComboBox 디렉토리를 참조하면 된다.

QFontComboBox::NonScalableFonts	0x2	동적 자동 변환이 제공되지 않는 폰트
QFontComboBox::MonospacedFonts	0x3	일정한 문자 넓이 형태를 제공하는 폰트
QFontComboBox::ProportionalFonts	0x4	넓이와 폭의 균형이 잡힌 폰트

- QLabel 과 QLCDNumber

QLabel 위젯은 어플리케이션 상에서 텍스트 또는 이미지를 표시하는 기능을 제공한다. QLCDNumber 위젯은 숫자만 표시할 수 있으며 디지털 시계와 같은 형태로 숫자를 표시할 수 있다. QLCDNumber 위젯은 시간을 표시할 때 사용하는 ":" 문자를 함께 사용할 수 있다.

```
...
QLabel *lbl[3];
lbl[0] = new QLabel("I love Qt programming", this);
lbl[0]->setGeometry(10, 30, 130, 40);

QPixmap pix = QPixmap(":resources/browser.png");
lbl[1] = new QLabel(this);
lbl[1]->setPixmap(pix);
lbl[1]->setGeometry(10, 70, 100, 100);

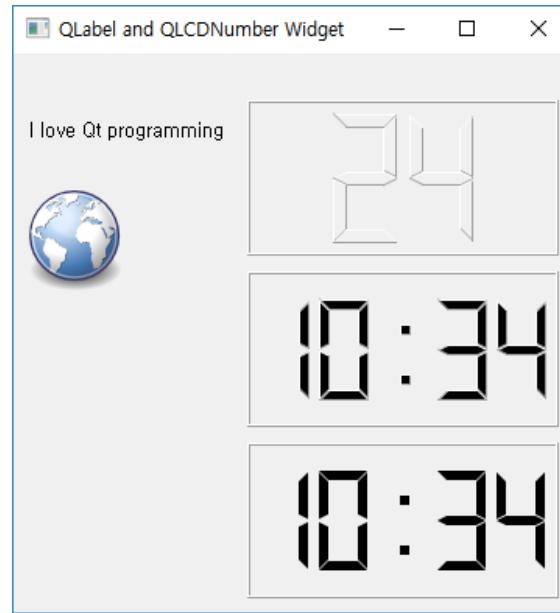
QLCDNumber *lcd[3];
lcd[0] = new QLCDNumber(2, this);
lcd[0]->display(24);
lcd[0]->setGeometry(150, 30, 200, 100);
lcd[0]->setSegmentStyle(QLCDNumber::Outline);

lcd[1] = new QLCDNumber(5, this);
lcd[1]->display("10:34");
lcd[1]->setGeometry(150, 140, 200, 100);
lcd[1]->setSegmentStyle(QLCDNumber::Filled);

lcd[2] = new QLCDNumber(5, this);
lcd[2]->display("10:34");
lcd[2]->setGeometry(150, 250, 200, 100);
lcd[2]->setSegmentStyle(QLCDNumber::Flat);
...
```

QPixmap 클래스는 이미지를 GUI상에 랜더링 하기 위해 제공되는 API이다. QPixmap 클래스를 이용해 이미지를 QLabel 위젯의 setPixmap() 멤버 함수를 이용하면 GUI상에

표시할 수 있다.



<그림> QLabel 과 QLCDNumber 예제 실행 화면

예제 전체 소스는 Ch03 > 01_BasicWidget > 11_QLabel_QLCDNumber 디렉토리를 참조하면 된다.

- QLineEdit

QLineEdit 위젯은 텍스트를 입력 및 수정을 위한 GUI를 제공한다. QLineEdit 클래스 위젯 상에서 복사, 붙여넣기, 자르기 등의 기능을 제공한다.

```
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    edit[0] = new QLineEdit("", this);
    lbl = new QLabel("QlineEdit Text : ", this);

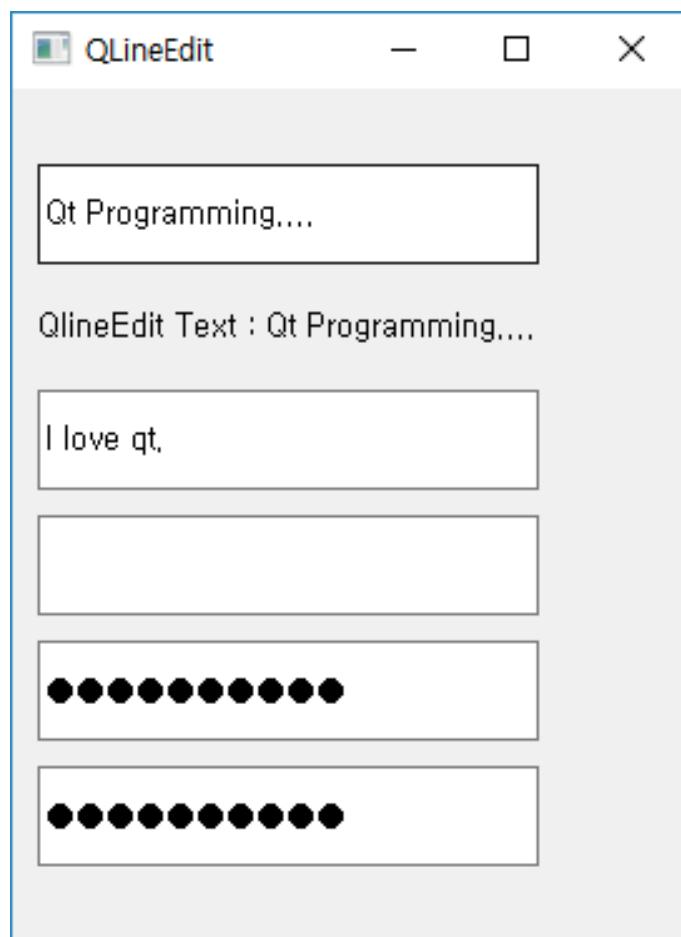
    connect(edit[0], SIGNAL(textChanged(QString)),
            this,      SLOT(textChanged(QString)));

    edit[0]->setGeometry(10, 30, 200, 40);
    lbl->setGeometry(10, 80, 250, 30);

    int ypos = 120;
    for(int i = 1 ; i < 5 ; i++) {
        edit[i] = new QLineEdit("I love qt.", this);
    }
}
```

```
        edit[i]->setGeometry(10, ypos, 200, 40);
        ypos += 50;
    }
    edit[1]->setEchoMode(QLineEdit::Normal);
    edit[2]->setEchoMode(QLineEdit::NoEcho);
    edit[3]->setEchoMode(QLineEdit::Password);
    edit[4]->setEchoMode(QLineEdit::PasswordEchoOnEdit);
}

void Widget::textChanged(QString str)
{
    lbl->setText(QString("QlineEdit Text : %1").arg(str));
}
...
```



<그림> QLineEdit 예제 실행 화면

QLineEdit 는 텍스트를 보이지 않게 하거나 패스워드 처리가 가능하다. 이와 같은 처리를 하기 위해 setEchoMode() 멤버 함수에 인자로 다음과 같은 상수를 사용해야 한다.

<표> setEchoMode() 멤버함수에서 사용 가능한 상수

상수	값	설명
QLineEdit::Normal	0	디폴트와 동일한 스타일
QLineEdit::NoEcho	1	텍스트가 보이지 않으며 커서의 위치도 변경되지 않는 스타일
QLineEdit::Password	2	텍스트가 "*" 문자로 표시
QLineEdit::PasswordEchoOnEdit	3	텍스트가 변경되면 디폴트 스타일과 동일 하지만 포커스가 이동되면 "*" 표시

예제 소스는 Ch03 > 01_BasicWidget > 12_QLineEdit 디렉토리를 참조하면 된다.

- QMenu와 QMenuBar 클래스를 이용한 메뉴

QMenu와 QMenuBar 클래스 위젯은 메뉴 기능을 제공한다. QMenu 위젯은 메뉴를 만들기 위해 제공하는 위젯으로써 addAction()과 addMenu() 멤버 함수를 제공한다. addAction()은 하위 메뉴가 없이 바로 실행할 기능을 연결하기 위해 사용되고, addMenu()는 하위 메뉴를 연결할 때 사용하는 멤버 함수이다. QMenu로 생성한 위젯을 QMenuBar와 연결해 메뉴를 완성할 수 있다.

```
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    menuBar = new QMenuBar(this);

    menu[0] = new QMenu("File");
    menu[0]->addAction("Edit");
    menu[0]->addAction("View");
    menu[0]->addAction("Tools");

    act[0] = new QAction("New", this);
    act[0]->setShortcut(Qt::CTRL | Qt::Key_A);
    act[0]->setStatusTip("This is a New menu.");

    act[1] = new QAction("Open", this);
    act[1]->setCheckable(true);
```

```

menu[1] = new QMenu("Save");
menu[1]->addAction(act[0]);
menu[1]->addAction(act[1]);

menu[2] = new QMenu("Print");
menu[2]->addAction("Page Setup");
menu[2]->addMenu(menu[1]);

menuBar->addMenu(menu[0]);
menuBar->addMenu(menu[2]);

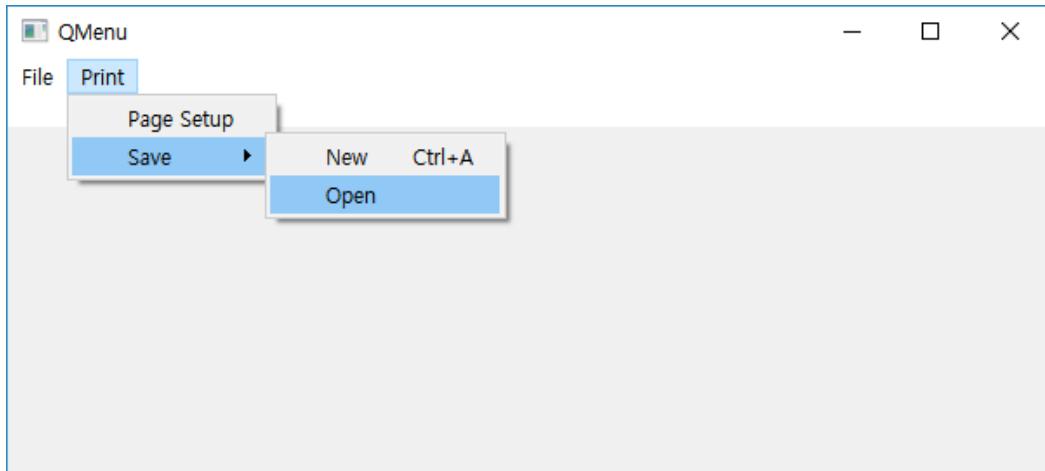
lbl = new QLabel("",this);
connect(menuBar, SIGNAL(triggered(QAction*)),
        this,      SLOT(trigerMenu(QAction*)));

menuBar->setGeometry(0, 0, 600, 40);
lbl->setGeometry(10, 200, 200, 40);
}

void Widget::trigerMenu(QAction *act)
{
    QString str = QString("Selected Menu : %1").arg(act->text());
    lbl->setText(str);
}
...

```

QMenu 클래스 위젯은 생성한 “File”, “Save” 그리고 “Print” 상위 메뉴이다. 그리고 QAction 클래스로 생성한 메뉴는 하위 메뉴로 추가하기 위한 클래스 위젯이다.



<그림> 메뉴 예제 실행 화면

예제 소스는 Ch03 > 01_BasicWidget > 13_QMenu_QMenuBar 디렉토리를 참조하면 된다.

- QProgressBar

진행사항을 표시하기 위한 위젯으로 QProgressBar 위젯을 사용할 수 있으며 위젯의 방향을 가로 또는 세로 방향으로 표시할 수 있다. QProgressBar 위젯은 가로 방향으로 배치할 경우 왼쪽에서 오른쪽으로, 오른쪽에서 왼쪽으로 진행방향을 변경할 수 있다. 반대로 세로 방향으로 배치할 경우 아래에서 위로, 위에서 아래로 진행 방향을 표시할 수 있다.

```
progress[0] = new QProgressBar(this);
progress[0]->setMinimum(0);
progress[0]->setMaximum(100);
progress[0]->setValue(50);
progress[0]->setOrientation(Qt::Horizontal);

progress[1] = new QProgressBar(this);
progress[1]->setMinimum(0);
progress[1]->setMaximum(100);
progress[1]->setValue(70);
progress[1]->setOrientation(Qt::Horizontal);
progress[1]->setInvertedAppearance(true);

progress[2] = new QProgressBar(this);
progress[2]->setMinimum(0);
progress[2]->setMaximum(100);
progress[2]->setValue(50);
progress[2]->setOrientation(Qt::Vertical);

progress[3] = new QProgressBar(this);
progress[3]->setMinimum(0);
progress[3]->setMaximum(100);
progress[3]->setValue(70);
progress[3]->setOrientation(Qt::Vertical);
progress[3]->setInvertedAppearance(true);

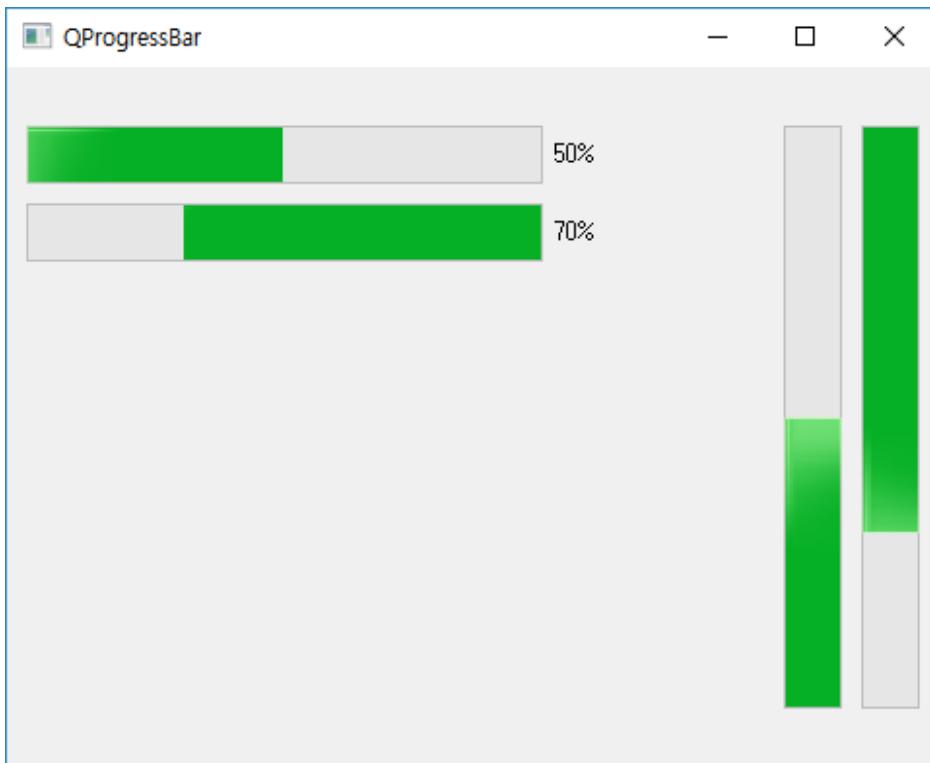
progress[0]->setGeometry(10,30, 300, 30);
progress[1]->setGeometry(10,70, 300, 30);
```

```
progress[2]->setGeometry(400,30, 30, 300);  
progress[3]->setGeometry(440,30, 30, 300);  
...
```

setMinimum() 와 setMaximum() 함수를 이용해 QProgressBar의 최소값과 최대값을 설정할 수 있으며 동일한 기능으로 setRange() 멤버 함수를 제공한다.

setValue() 는 QProgressBar의 최소값과 최대값 사이의 진행 값을 설정하기 위해 사용하는 멤버 함수이며 setOrientation() 함수는 가로 혹은 세로 방향으로 표시할 수 있다.

setInvertedAppearance() 멤버 함수는 진행 방향을 설정하기 위한 멤버 함수이다. 인자로 true 를 설정하면 디폴트 설정의 반대 방향으로 바꿀 수 있다.



<그림> QProgressBar 예제 실행 화면

예제 소스는 Ch03 > 01_BasicWidget > 14_QProgressBar 디렉토리를 참조하면 된다.

- QRadioButton

QRadioButton 위젯은 사용자에게 여러 항목 중 하나를 선택할 수 있는 GUI를 제공한다. 예를 들어 On(checked) 혹은 Off(unchecked)와 같이 둘 중 하나를 선택할 수 있다.

```
QRadioButton *radio1[3];
QRadioButton *radio2[3];

QString str1[3] = {"Computer", "Notebook", "Tablet"};
int ypos = 30;
for(int i = 0 ; i < 3 ; i++)
{
    radio1[i] = new QRadioButton(str1[i], this);
    radio1[i]->setGeometry(10, ypos, 150, 30);
    ypos += 40;
}

QString str2[3] = {"In-Vehicle", "Smart TV", "Mobile"};
ypos = 30;

for(int i = 0 ; i < 3 ; i++)
{
    radio2[i] = new QRadioButton(str2[i], this);

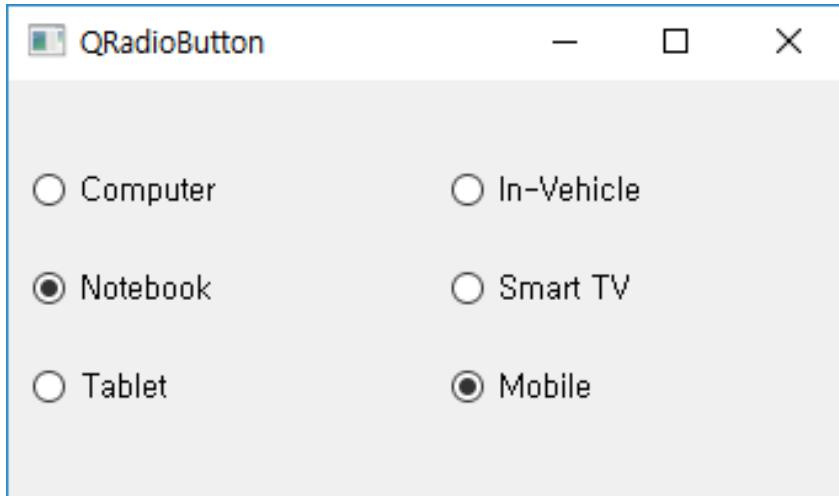
    if(i == 2)
        radio2[i]->setChecked(true);

    radio2[i]->setGeometry(180, ypos, 150, 30);
    ypos += 40;
}

QButtonGroup *group1 = new QButtonGroup(this);
QButtonGroup *group2 = new QButtonGroup(this);

group1-> addButton(radio1[0]);
group1-> addButton(radio1[1]);
group1-> addButton(radio1[2]);

group2-> addButton(radio2[0]);
group2-> addButton(radio2[1]);
group2-> addButton(radio2[2]);
...
```



<그림> QRadioButton 예제 실행 화면

좌측의 예제 실행 화면에서 보는 것과 같이 6개 항목에서 왼쪽 3개 항목을 그룹으로 지정함으로써 오른쪽 QRadioButton들과 분리할 수 있다. 이 프로젝트의 소스는 Ch03 > 01_BasicWidget > 15_QRadioButton 디렉토리를 참조하면 된다.

- QScrollArea

QScrollArea 위젯은 화면 또는 GUI 위젯 상에 모두 표시할 수 없는 경우 스크롤 바를 이용해 가려진 부분으로 이동하는 GUI 기능을 제공한다. 예를 들어 어떤 이미지를 축소하지 않고 화면에 표시하고자 할 때 GUI 위젯의 크기가 부족하게 되면 좌측 또는 하단에 스크롤이 생겨 마우스로 스크롤을 이동하면서 이미지를 볼 수 있는 것과 같은 기능을 제공한다.

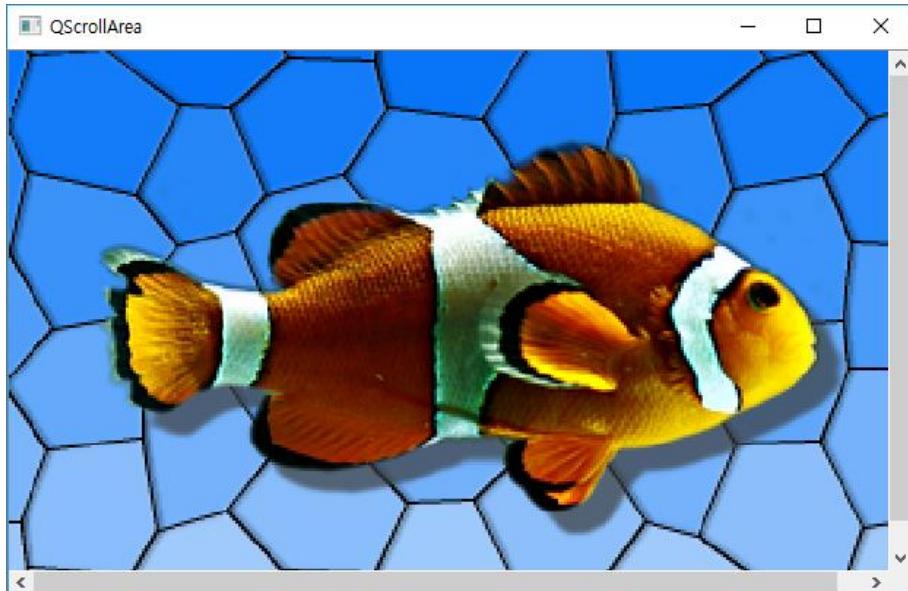
```
QImage image;
QScrollArea *area;

QLabel *lbl = new QLabel(this);
image = QImage(":resources/fish.png");
lbl->setPixmap(QPixmap::fromImage(image));

area = new QScrollArea(this);
area->setWidget(lbl);
area->setBackgroundRole(QPalette::Dark);
area->setGeometry(0, 0, image.width(), image.height());
...
```

QImage 클래스를 이용해 이미지를 QLabel 위젯 상에 setPixmap() 멤버 함수를 이용해

랜더링 한다. QScrollArea 위젯의 setWidget() 멤버 함수를 이용하면 QLabel 위젯을 QScrollArea 위젯 영역에 포함시킬 수 있다. 만약 이미지가 QScrollArea 위젯보다 큰 경우 자동으로 스크롤 바가 활성화 된다.



<그림> QScrollArea 예제 실행 화면

예제 소스는 Ch03 > 01_BasicWidget > 16_QScrollArea 디렉토리를 참조하면 된다.

● QScrollBar

QScrollBar은 슬라이더 위젯의 모양과 비슷하다. QScrollBar 위젯은 좌우 혹은 상하 위치 시킬 때 세로 방향 혹은 가로 방향으로 배치할 수 있는 기능을 제공한다.

```
...
class Widget : public QWidget
{
    Q_OBJECT
public:
    Widget(QWidget *parent = 0);
    ~Widget();
private:
    QScrollBar *vscrollbar[3];
    QScrollBar *hscrollbar[3];
    QLabel     *lbl[3];
private slots:
    void valueChanged1(int value);
```

```

void valueChanged2(int value);
void valueChanged3(int value);
};

...

```

이전 소스코드에서 보는 것과 같이 QScrollBar 와 QLabel 클래스의 오브젝트를 선언한다. 그리고 세로 방향의 QScrollBar 클래스의 오브젝트인 vscrollbar[0] ~ vscrollbar[2] 의 눈금을 드래그 하면 변경된 값을 QLabel 위젯에 표시한다. 또한 vscrollbar[0] ~ vscrollbar[2] 의 값과 각각의 hscrollbar 의 값과 동일하게 변경 한다.

```

...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    int xpos = 10;
    int ypos = 50;
    for(int i = 0 ; i < 3 ; i++)
    {
        vscrollbar[i] = new QScrollBar(Qt::Vertical, this);
        vscrollbar[i]->setRange(0, 100);
        vscrollbar[i]->setGeometry(xpos, 30, 20, 200);

        lbl[i] = new QLabel(QString("%1").arg(vscrollbar[i]->value()), this);
        lbl[i]->setGeometry(xpos + 2, 220, 30, 30);
        xpos += 50;

        hscrollbar[i] = new QScrollBar(Qt::Horizontal, this);
        hscrollbar[i]->setRange(0, 100);
        hscrollbar[i]->setGeometry(150, ypos, 200, 20);
        ypos += 30;
    }
    connect(vscrollbar[0], SIGNAL(valueChanged(int)),
            this,           SLOT(valueChanged1(int)));
    connect(vscrollbar[1], SIGNAL(valueChanged(int)),
            this,           SLOT(valueChanged2(int)));
    connect(vscrollbar[2], SIGNAL(valueChanged(int)),
            this,           SLOT(valueChanged3(int)));
}

void Widget::valueChanged1(int value)
{
    lbl[0]->setText(QString("%1").arg(value));
    hscrollbar[0]->setValue(vscrollbar[0]->value());
}

```

```

}

void Widget::valueChanged2(int value)
{
    lbl[1]->setText(QString("%1").arg(value));
    hscrollbar[1]->setValue(vscrollbar[1]->value());
}

void Widget::valueChanged3(int value)
{
    lbl[2]->setText(QString("%1").arg(value));
    hscrollbar[2]->setValue(vscrollbar[2]->value());
}
...

```

connect() 함수는 vscrollbar[0] ~ vscrollbar[2] 의 값이 변경되면 각각의 Slot 함수가 호출된다. 예를 들어 vscrollbar[1] 번째 눈금이 마우스 드래그로 값이 변경 되면 valueChanged2() Slot함수가 호출된다. 이 Slot 함수는 QLabel 의 값을 Slot 함수의 인자로 받은 값으로 변경한다. 그리고 hscrollbar[1] 의 값을 변경한다.



<그림> QScrollBar 예제 실행 화면

예제 전체 소스는 Ch03 > 01_BasicWidget > 17_QScrollBar 딕렉토리를 참조하면 된다.

- QSizeGrip

QSizeGrip 클래스 위젯은 한정된 크기의 윈도우 영역 안에 위젯의 크기를 조절할 수 있다. 예를 들어 윈도 탐색기와 같이 왼쪽에는 트리 영역, 오른쪽은 파일 및 딕렉토리 속성을 보여주는 GUI 상에서 트리 영역의 경계선을 마우스로 드래그하여 줄이거나 늘릴 수 있는 것처럼 QSizeGrip을 이용하면 Splitter와 같은 GUI를 구현할 수 있다. 다음 소스코드는 widget.h 파일 소스코드이다.

```
...
// SubWindow 클래스 위젯
class SubWindow : public QWidget
{
    Q_OBJECT

public:
    SubWindow(QWidget *parent = nullptr) : QWidget(parent, Qt::SubWindow)
    {
        QSizeGrip *sizegrip = new QSizeGrip(this);
        sizegrip->setFixedSize(sizegrip->sizeHint());

        this->setLayout(new QVBoxLayout);
        this->layout()->setMargin(0);

        layout()->addWidget(new QTextEdit);

        sizegrip->setWindowFlags(Qt::WindowStaysOnTopHint);
        sizegrip->raise();
    }

    QSize sizeHint() const
    {
        return QSize(200, 100);
    }
};

// Widget 클래스 위젯, SubWindow의 부모 클래스 위젯
class Widget : public QWidget
{
    Q_OBJECT
```

```
public:  
    Widget(QWidget *parent = nullptr);  
    ~Widget();  
  
};  
...
```

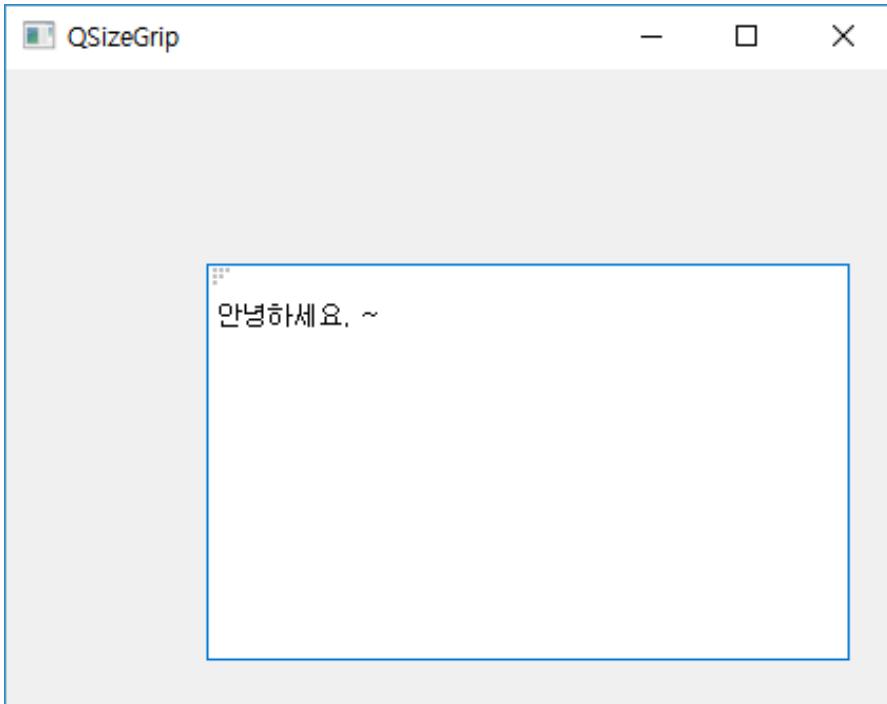
위의 예제 소스코드에서 보는 것과 같이 Widget 클래스는 부모 클래스(윈도우 위젯)로 선언하고 SubWindow 클래스는 Widget 윈도우의 자식(Child) 윈도우 위젯이다. 소스 코드 상에 QVBoxLayout 은 수직 방향의 레이아웃을 이용해 위젯을 배치할 수 있다.

레이아웃은 다음절에 다룬다. 여기서는 Sub Window의 레이아웃으로 선언한다는 정도만 알고 넘어가도록 하자.

QTextEdit 위젯은 메모장과 같이 텍스트를 편집하기 위해 제공되는 위젯이다. QTextEdit 위젯은 SubWindow 영역의 전체 영역 크기로 설정해 Sub 윈도우 크기를 변경할 때 동적으로 크기가 변경할 수 있다. 다음 소스코드는 main.cpp의 소스코드이다.

```
#include < QApplication>  
#include "widget.h"  
  
int main(int argc, char *argv[]){  
    QApplication a(argc, argv);  
  
    Widget w;  
    w.resize(400, 300);  
  
    SubWindow subWindow(&w);  
    subWindow.move(200, 180);  
  
    w.show();  
  
    return a.exec();  
}
```

위의 예제에서 보는 것과 같이 QTextEdit 위젯은 SubWindow 영역의 전체 영역 크기로 설정하였다. Sub 윈도우 크기를 변경할 때 QTextEdit 동적으로 크기가 변경된다.



<그림> QSizeGrip 예제 실행 화면

예제 전체 소스는 Ch03 > 01_BasicWidget > 18_QSizeGrip 디렉토리를 참조하면 된다.

● QSlider

QSlider 위젯은 최소값과 최대값을 지정한 범위 내에서 설정 값을 변경할 수 있는 GUI를 제공한다. QScrollBar와 유사하다. setMinimum()과 setMaximum() 멤버 함수를 사용해 최소값과 최대값을 설정할 수 있다. 최소값과 최대값을 설정하기 위해 setRange() 멤버 함수를 사용하면 QSlider 위젯의 Minimum 값과 Maximum 값을 설정할 수 있다.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    int xpos = 20, ypos = 20;
    for(int i = 0 ; i < 6 ; i++) {
        if(i <= 2) {
            slider[i] = new QSlider(Qt::Vertical, this);
            slider[i]->setGeometry(xpos, 20, 30, 80);
            xpos += 30;
        }
        else if(i >= 3) {
            slider[i] = new QSlider(Qt::Horizontal, this);
            slider[i]->setGeometry(xpos, 20, 30, 80);
            xpos += 30;
        }
    }
}
```

```

        slider[i]->setGeometry(130, ypos, 80, 30);
        ypos += 30;
    }
    slider[i]->setRange(0, 100);
    slider[i]->setValue(50);
}

xpos = 20;
for(int i = 0 ; i < 3 ; i++) {
    QString str = QString("%1").arg(slider[i]->value());
    lbl[i] = new QLabel(str, this);
    lbl[i]->setGeometry(xpos+10, 100, 30, 40);
    xpos += 30;
}
connect(slider[0], SIGNAL(valueChanged(int)),
        this, SLOT(valueChanged1(int)));
connect(slider[1], SIGNAL(valueChanged(int)),
        this, SLOT(valueChanged2(int)));
connect(slider[2], SIGNAL(valueChanged(int)),
        this, SLOT(valueChanged3(int)));
}

void Widget::valueChanged1(int value)
{
    lbl[0]->setText(QString("%1").arg(value));
    slider[3]->setValue(slider[0]->value());
}

void Widget::valueChanged2(int value)
{
    lbl[1]->setText(QString("%1").arg(value));
    slider[4]->setValue(slider[1]->value());
}

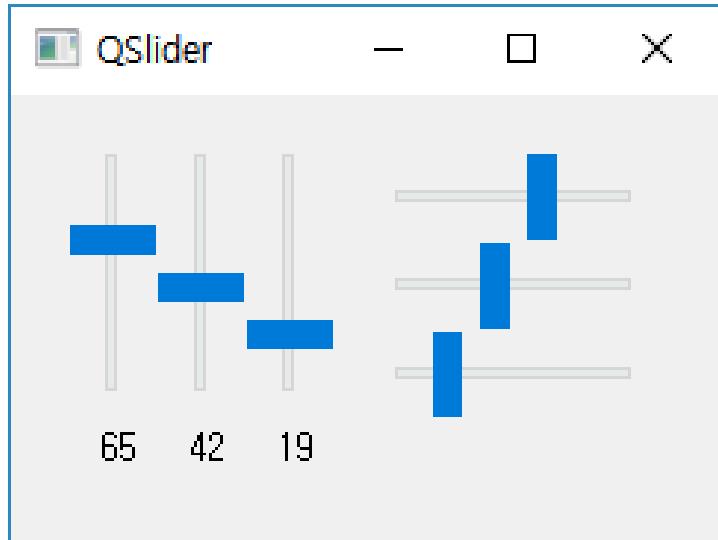
void Widget::valueChanged3(int value)
{
    lbl[2]->setText(QString("%1").arg(value));
    slider[5]->setValue(slider[2]->value());
}
...

```

QSlider 위젯은 수평 방향과 수직 방향의 GUI를 제공하고, QSlider의 생성자 함수의 첫 번째 인자는 Qt::Vertical (수직) 혹은 Qt::Horizontal (수평) 상수를 사용해 위젯의 방향을

변경할 수 있다.

QLabel 위젯은 수직 방향의 QSlider의 눈금이 위치한 값을 표시한다. 그리고 QSlider의 눈금의 위치가 변경될 때 시그널 이벤트를 발생해 QLabel 위젯의 텍스트를 QSlider의 현재 값으로 변경한다.



<그림> QSlider 예제 실행 화면

예제 전체 소스는 Ch03 > 01_BasicWidget > 19_QSlider 딕렉토리를 참조하면 된다.

● QTabWidget

많은 위젯을 배치할 경우 또는 윈도우의 크기가 제한적일 때 탭을 사용하면 유용하다. QTabWidget은 제한된 크기에 모든 탭을 표시할 수 없을 경우 동적으로 페이지를 이동 할 수 있는 기능을 제공한다.

QWidget으로 선언한 위젯을 tab 인스턴스 위젯 상에 배치하기 위해 addTab() 멤버 함수를 사용하면 된다. 각 탭 상에 위젯을 배치하는 방법은 위젯을 배치하기 위해 부모 클래스를 인자로 명시한 것처럼 부모 클래스를 탭 위젯으로 명시하면 그 위젯이 해당 탭 내에 위젯을 배치할 수 있다.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    QTabWidget *tab = new QTabWidget(this);
    QWidget *browser_tab = new QWidget;
    QWidget *users_tab = new QWidget;
```

```

tab->addTab(browser_tab, QIcon(":resources/browser.png"), "Browser");
tab->addTab(users_tab, QIcon(":resources/users.png"), "Users");
tab->setGeometry(20, 20, 300, 250);

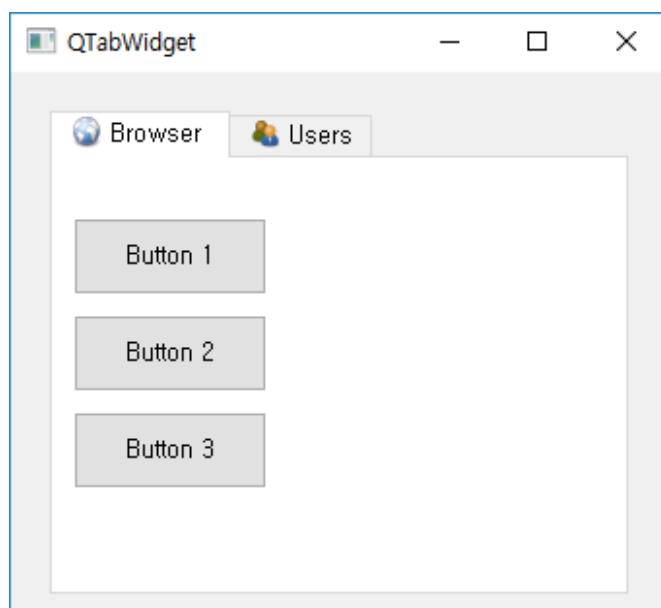
QString btn_str[3] = {"Button 1", "Button 2", "Button 3"};
QPushButton *btn[3];

int ypos = 30;
for(int i = 0 ; i < 3 ; i++) {
    btn[i] = new QPushButton(btn_str[i], browser_tab);
    btn[i]->setGeometry(10, ypos, 100, 40);
    ypos += 50;
}

connect(tab, SIGNAL(currentChanged(int)), this, SLOT(currentTab(int)));
}

void Widget::currentTab(int index)
{
    qDebug("Current Tab : %d", index);
}
...

```



<그림> QTabWidget 예제 실행 화면

예제 전체 소스는 Ch03 > 01_BasicWidget > 20_QTabWidget 디렉토리를 참조하면 된다.

- QToolBar 와 QAction

QToolBar 위젯은 윈도우상에 툴메뉴 바와 같은 GUI를 제공한다. QToolBar 위젯은 addAction() 멤버 함수를 이용해 툴바 메뉴 상에 배치할 수 있다.

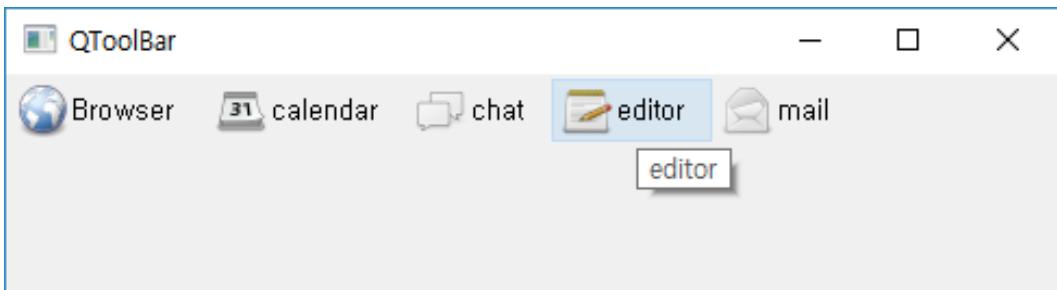
```
QToolBar *toolbar = new QToolBar(this);

QAction *act[5];
act[0] = new QAction(QIcon(":resources/browser.png"), "Browser", this);
act[1] = new QAction(QIcon(":resources/calendar.png"), "calendar", this);
act[2] = new QAction(QIcon(":resources/chat.png"), "chat", this);
act[3] = new QAction(QIcon(":resources/editor.png"), "editor", this);
act[4] = new QAction(QIcon(":resources/mail.png"), "mail", this);

act[0]->setShortcut(Qt::Key_Control | Qt::Key_E);
act[0]->setToolTip("This is a ToolTip.");

toolbar->setToolButtonStyle(Qt::ToolButtonTextBesideIcon);

for(int i = 0 ; i < 5 ; i++)
{
    toolbar->addAction(act[i]);
}
...
```



<그림> QToolBar 예제 실행 화면

이전 그림에서 보는 것과 같이 QAction 메뉴 버튼의 아이콘만 보이게 하거나 아이콘과 버튼 이름을 같이 표시할 수 있도록 다음과 같은 상수 값을 제공한다.

<표> setToolButtonStyle() 멤버 함수에서 사용 가능한 상수

상수	값	설명
Qt::ToolButtonIconOnly	0	아이콘만 표시
Qt::ToolButtonTextOnly	1	버튼 이름만 표시
Qt::ToolButtonTextBesideIcon	2	아이콘을 텍스트 안쪽에 표시
Qt::ToolButtonTextUnderIcon	3	아이콘을 텍스트 아래에 표시

예제 전체 소스는 Ch03 > 01_BasicWidget > 21_QToolBar 디렉토리를 참조하면 된다.

● QWidget

지금까지 살펴본 위젯은 QWidget으로부터 상속받아 구현한 위젯이다. 예를 들어 마우스, 키보드 혹은 윈도우로부터 받은 이벤트를 QPushButton과 같은 위젯에서 사용할 수 있는 것도 QWidget으로부터 상속받았기 때문에 가능하다.

QWidget 클래스는 paintEvent() virtual 함수를 이용해 위젯 영역에 텍스트, 도형(선, 원, 사각형 등), 이미지를 랜더링 할 수 있는 기능을 제공한다. 또한 QWidget 클래스의 update() 멤버 함수를 호출하면 paintEvent() 함수를 호출 할 수 있다. 예를 들어 어떤 버튼을 클릭하면 호출되는 Slot 함수 내에 update() 멤버 함수를 사용 하면 paintEvent() virtual 함수가 호출 된다.

QWidget 클래스는 resizeEvent() virtual 함수를 제공한다. 이 virtual 함수는 QWidget의 크기가 변경되면 호출된다. QWidget 영역 내에 마우스 이벤트를 처리, 키보드 이벤트, 위젯의 영역에 활성화가 Focus 되어 있는지 등의 다양한 virtual 함수를 제공한다.

```
#include <QPainter>
#include <QtEvents>
#include <QLineEdit>

class Widget : public QWidget
{
    Q_OBJECT
public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    QLineEdit *edit;
```

```

protected:
    virtual void paintEvent(QPaintEvent *event);
    virtual void resizeEvent(QResizeEvent *event);

    virtual void mousePressEvent(QMouseEvent *event);
    virtual void mouseReleaseEvent(QMouseEvent *event);
    virtual void mouseDoubleClickEvent(QMouseEvent *event);
    virtual void mouseMoveEvent(QMouseEvent *event);

    virtual void keyPressEvent(QKeyEvent *event);
    virtual void keyReleaseEvent(QKeyEvent *event);
    virtual void focusInEvent(QFocusEvent *event);
    virtual void focusOutEvent(QFocusEvent *event);
};

```

다음 소스코드는 widget.cpp 소스코드이며 각 `virtual` 함수를 구현한 소스코드이다.

```

...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    edit = new QLineEdit("", this);
    edit->setGeometry(120, 20, 100, 30);
}

void Widget::paintEvent(QPaintEvent *event)
{
    Q_UNUSED(event);
    QPainter painter(this);

    QString img_full_name = QString(":resources/browser.png");
    QImage image(img_full_name);
    painter.drawPixmap(0, 0,
                      QPixmap::fromImage(image.scaled(100, 100,
                                                      Qt::IgnoreAspectRatio,
                                                      Qt::SmoothTransformation)));
    painter.end();
}

void Widget::resizeEvent(QResizeEvent *event)
{
    Q_UNUSED(event);
    qDebug("[Resize Event call]");
}

```

```

        qDebug("width: %d, height: %d", this->width(), this->height());
    }

void Widget::mousePressEvent(QMouseEvent *event)
{
    Q_UNUSED(event);

    qDebug("[Mouse Press] x, y : %d, %d", event->x(), event->y());
}

void Widget::mouseReleaseEvent(QMouseEvent *event)
{
    Q_UNUSED(event);
    qDebug("[Mouse Release] x, y : %d, %d", event->x(), event->y());
}

void Widget::mouseDoubleClickEvent(QMouseEvent *event)
{
    Q_UNUSED(event);
    qDebug("[Mouse Double Click] x, y : %d , %d ", event->x(), event->y());
}

void Widget::mouseMoveEvent(QMouseEvent *event)
{
    Q_UNUSED(event);
    qDebug("[Mouse Move] x, y : %d , %d ", event->x(), event->y());
}

void Widget::keyPressEvent(QKeyEvent *event)
{
    Q_UNUSED(event);
    qDebug("Key Press Event.");

    switch(event->key())
    {
    case Qt::Key_A :
        if(event->modifiers())
            qDebug("A");
        else
            qDebug("a");

        qDebug("%x", event->key());
    }
}

```

```

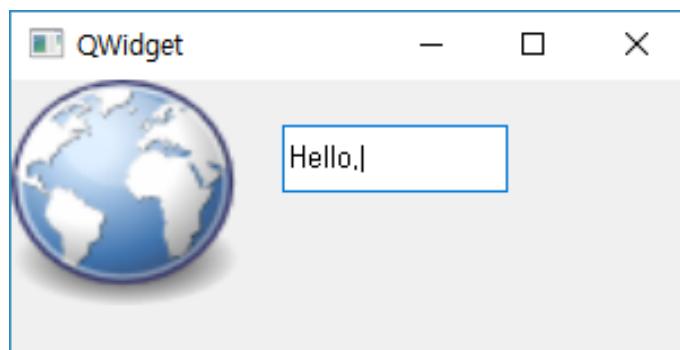
        break;
    default: break;
}
}

void Widget::keyReleaseEvent(QKeyEvent *event)
{
    Q_UNUSED(event);
    qDebug("Key Release Event.");
}

void Widget::focusInEvent(QFocusEvent *event)
{
    Q_UNUSED(event);
    qDebug("focusInEvent Event.");
}

void Widget::focusOutEvent(QFocusEvent *event)
{
    Q_UNUSED(event);
    qDebug("focusOutEvent Event.");
}
...

```



<그림> QWidget에서 제공하는 virtual 함수 예제 실행 화면

예제 전체 소스는 Ch03 > 01_BasicWidget > 22_QWidget 디렉토리를 참조하면 된다.

- QTabBar

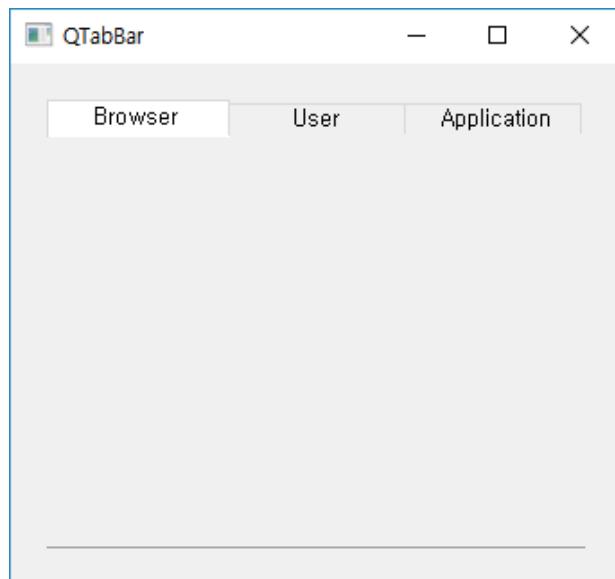
QTabBar 클래스 위젯은 탭 GUI를 제공한다. 이 위젯은 QTabWidget 과 비슷한 기능을

제공한다. 탭에 아이콘을 사용 하기 위해서 setTabIcon() 함수를 사용하면 된다. 각 탭에 표시되는 텍스트가 구별되게 컬러 지정이 가능하다. 탭의 텍스트를 지정하기 위해 setTabTextColor() 함수를 사용하면 된다.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    QTabBar *tab = new QTabBar(this);
    tab->addTab("Browser");
    tab->addTab("User");
    tab->addTab("Application");
    tab->setShape(QTabBar::RoundedNorth);
    tab->setGeometry(20, 20, 300, 250);

    connect(tab, SIGNAL(currentChanged(int)),
            this, SLOT(currentTab(int)));
}

void Widget::currentTab(int index)
{
    qDebug("Current Tab : %d", index);
}
...
```



<그림> QTabBar 예제 실행 화면

QTabBar는 이미 정의된 탭의 모양을 변경할 수 있다. 예를 들어 모서리가 둥근 형태의 탭이나 탭의 왼쪽 모서리만 둥글게 표현한 도형을 이용할 수 있는 기능을 제공하며

`setShape()` 함수를 이용해 지정할 수 있다. 다음 표는 ENUM 타입으로 정의된 템모양 값이다.

<표> `setShape()` 멤버 함수에서 사용 가능한 상수

상수	값	설명
QTabBar::RoundedNorth	0	디폴트 모양
QTabBar::RoundedSouth	1	페이지 아래쪽이 둥근 형태
QTabBar::RoundedWest	2	페이지의 왼쪽 부분이 둥근 형태
QTabBar::RoundedEast	3	페이지의 오른쪽 부분이 둥근 형태
QTabBar::TriangularNorth	4	삼각형 모양
QTabBar::TriangularSouth	5	스프레드시트에서 사용하는 형태와 비슷한 형태
QTabBar::TriangularWest	6	페이지의 왼쪽 부분이 삼각형 모양
QTabBar::TriangularEast	7	페이지의 오른쪽 부분이 삼각형 모양

예제 소스는 Ch03 > 01_BasicWidget > 23_QTabBar 디렉토리를 참조하면 된다.

● QToolBox

`QToolbox` 클래스 위젯은 위젯 아이템들을 새로 방향 템 컬럼 형태로 GUI를 제공한다. 각 템의 이름으로 텍스트와 아이콘을 사용할 수 있다.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    box = new QToolBox(this);
    lay = new QVBoxLayout(this);

    but1 = new QPushButton("DataBase - 1",this);
    but2 = new QPushButton("Network - 2",this);
    but3 = new QPushButton("Graphics - 3",this);

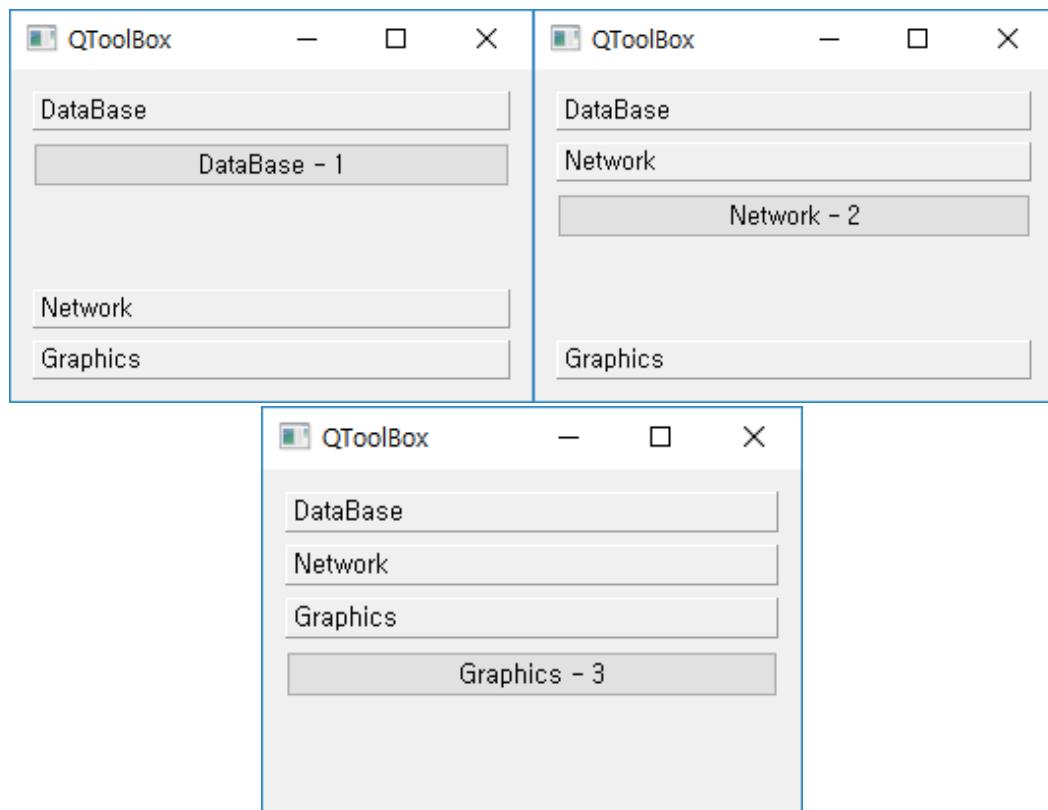
    box->addItem(but1,"DataBase");
    box->addItem(but2,"Network");
    box->addItem(but3,"Graphics");
    lay->addWidget(box);
    setLayout(lay);
```

```

        connect(box, SIGNAL(currentChanged(int)), this, SLOT(changedTab(int)));
    }

void Widget::changedTab(int index)
{
    qDebug("current index : %d", index);
}
...

```



<그림> QToolBox 예제 실행 화면

예제 소스는 Ch03 > 01_BasicWidget > 24_QToolBox 디렉토리를 참조하면 된다.

- QToolButton

QToolButton 위젯은 텍스트 또는 아이콘을 사용해 버튼과 같은 기능을 제공한다. QToolButton의 아이콘은 QIcon 클래스를 이용해 지정할 수 있다. 아이콘은 상태에 따라 활성화 또는 비활성화된 상태로 표시할 수 있으며 비활성화 상태에서는 버튼을 사

용할 수 없다.

setToolButtonStyle() 멤버 함수를 이용하면 스타일을 변경 할 수 있으며 setIconSize() 함수를 이용하면 아이콘의 크기를 지정할 수 있다.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    QToolBar *tool = new QToolBar(this);
    QVBoxLayout *layout = new QVBoxLayout;

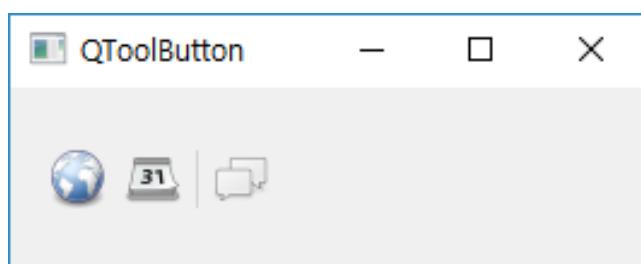
    QToolButton *button = new QToolButton;
    button->setIcon(QIcon(":resources/browser.png"));

    QToolButton *button1 = new QToolButton;
    button1->setIcon(QIcon(":resources/calendar.png"));

    QToolButton *button2 = new QToolButton;
    button2->setIcon(QIcon(":resources/chat.png"));

    tool->addWidget(button);
    tool->addWidget(button1);
    tool->addSeparator();
    tool->addWidget(button2);
    layout->addWidget(tool);

    this->setLayout(layout);
}
...
```

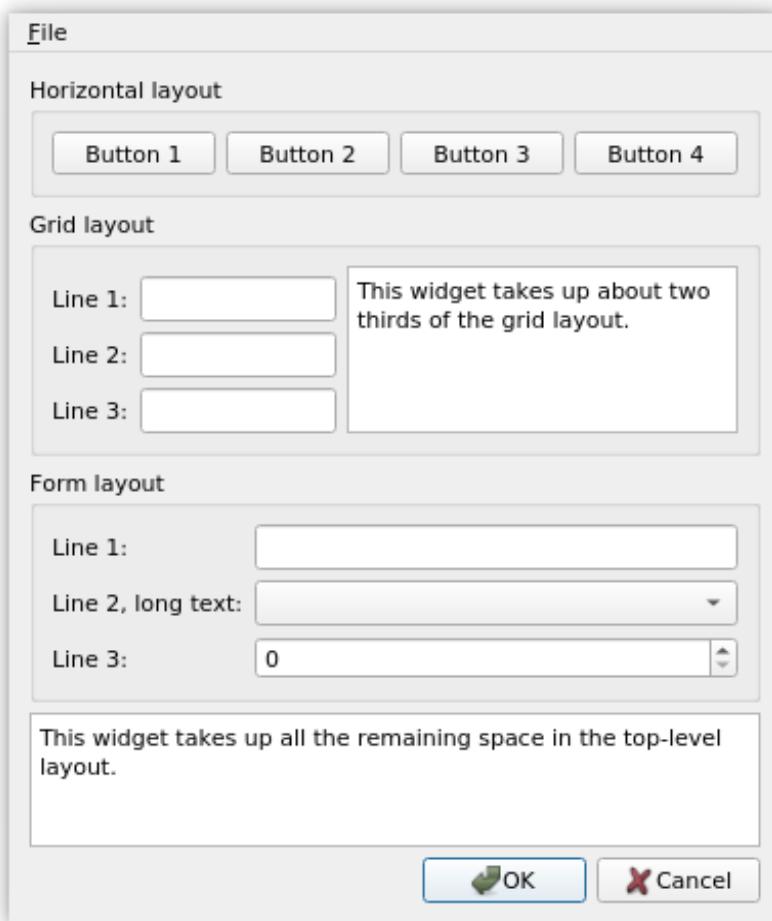


<그림> QToolButton 예제 실행 화면

예제 소스는 Ch03 > 01_BasicWidget > 25_QToolBox 딕레토리를 참조하면 된다.

3.2. 레이아웃

QWidget 클래스의 setGeometry() 멤버 함수를 이용해 GUI 상에서 특정 X, Y 좌표로 위젯을 배치하게 되면 윈도우의 크기가 변경될 때 위젯의 위치가 변경되지 않는다. 하지만 아래 그림에서 보는 것과 같이 레이아웃을 이용해 위젯을 배치하면 위젯들의 크기가 변경되면 GUI 상에 위젯의 위치도 동적으로 변경된다.



<그림> 레이아웃을 이용해 배치한 위젯 화면

윈도우의 크기가 변경되면 레이아웃은 위젯들을 최적의 위치에 정렬되어 일관된 크기의 모양을 유지할 수 있도록 해준다. 다음 표는 Qt에서 주로 사용되는 레이아웃 클래스들이다.

<표> 주로 사용되는 레이아웃

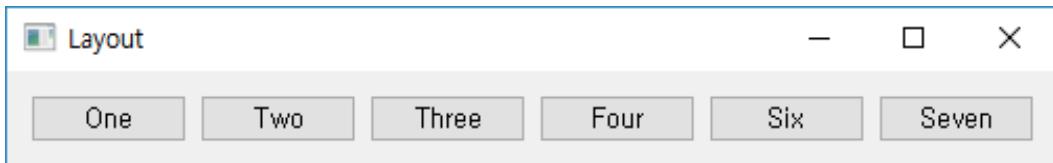
클래스	설명
QHBoxLayout	위젯들을 가로 방향으로 배치
QVBoxLayout	위젯들을 세로 방향으로 배치
QGridLayout	위젯을 그리드(Grid) 또는 바둑판 스타일로 배치
QFormLayout	위젯을 2열로 배치하는 형식.

● QHBoxLayout

QHBoxLayout은 위젯들을 가로 방향으로 배치할 수 있다. 다음 예제에서와 같이 QPushButton 위젯을 addWidget() 멤버 함수를 이용해 추가할 수 있다.

```
QHBoxLayout *hboxLayout = new QHBoxLayout();
QPushButton *btn[6];

QString btnStr[6] = {"One", "Two", "Three", "Four", "Six", "Seven"};
for(int i = 0 ; i < 6 ; i++)
{
    btn[i] = new QPushButton(btnStr[i]);
    hboxLayout->addWidget(btn[i]);
}
```



<그림> QHBoxLayout

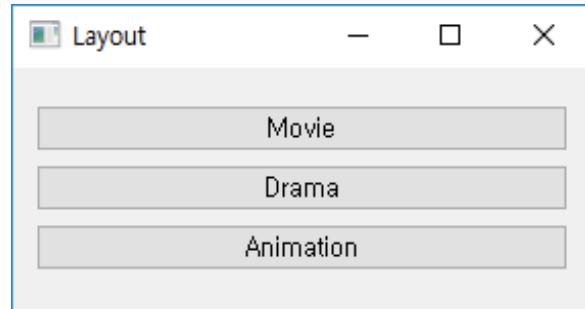
● QVBoxLayout

QVBoxLayout은 위젯을 세로 방향으로 배치할 수 있다.

```
QVBoxLayout *vboxLayout = new QVBoxLayout();
QPushButton *vbtn[6];

QString vbtnStr[3] = {"Movie", "Drama", "Animation"};
for(int i = 0 ; i < 3 ; i++) {
```

```
vbtn[i] = new QPushButton(vbtnStr[i]);
vboxLayout->addWidget(vbtn[i]);
}
```



<그림> QVBoxLayout

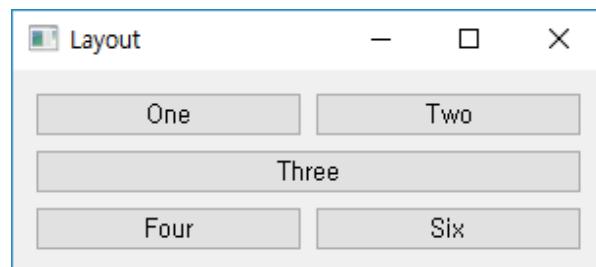
- QVBoxLayout

QGridLayout은 그리드와 같은 스타일로 위젯을 배치할 수 있다. QGridLayout은 특정 행을 하나의 위젯만 배치할 수 있도록 병합하거나 여러 개의 셀로 나눌 수 있다.

```
QGridLayout *gridLayout = new QGridLayout();
QPushButton *gbtn[5];

for(int i = 0 ; i < 5 ; i++)
    gbtn[i] = new QPushButton(btnStr[i]);

gridLayout->addWidget(gbtn[0], 0, 0);
gridLayout->addWidget(gbtn[1], 0, 1);
gridLayout->addWidget(gbtn[2], 1, 0, 1, 2);
gridLayout->addWidget(gbtn[3], 2, 0);
gridLayout->addWidget(gbtn[4], 2, 1);
```



<그림> QGridLayout

- QFormLayout

QFormLayout은 2열 형식으로 위젯을 배치하는 레이아웃 클래스이다. MS윈도우, 리눅스 GNOME에서는 일반적으로 왼쪽 방향 정렬을 사용하며 MacOS은 오른쪽 정렬을 사용한다.

```
nameLabel = new QLabel(tr("&Name:"));
nameLabel->setBuddy(nameLineEdit);

emailLabel = new QLabel(tr("&Name:"));
emailLabel->setBuddy(emailLineEdit);

ageLabel = new QLabel(tr("&Name:"));
ageLabel->setBuddy(ageSpinBox);

QGridLayout *gridLayout = new QGridLayout;
gridLayout->addWidget(nameLabel, 0, 0);
gridLayout->addWidget(nameLineEdit, 0, 1);

gridLayout->addWidget(emailLabel, 1, 0);
gridLayout->addWidget(emailLineEdit, 1, 1);

gridLayout->addWidget(ageLabel, 2, 0);
gridLayout->addWidget(ageSpinBox, 2, 1);
setLayout(gridLayout);
```



<그림> MS윈도우와 GNOME스타일



<그림> MacOS 스타일

- 중첩된 레이아웃을 사용

레이아웃 안에 레이아웃을 배치할 수 있다. 다음 예제는 중첩 구조로 예제이다.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
```

```
QHBoxLayout *hboxLayout = new QHBoxLayout();
QPushButton *btn[6];

QString btnStr[6] = { "One", "Two", "Three", "Four", "Six", "Seven"};
for(int i = 0 ; i < 6 ; i++) {
    btn[i] = new QPushButton(btnStr[i]);
    hboxLayout->addWidget(btn[i]);
}

QVBoxLayout *vboxLayout = new QVBoxLayout();
QPushButton *vbtn[6];
QString vbtnStr[3] = {"Movie", "Drama", "Animation"};

for(int i = 0 ; i < 3 ; i++) {
    vbtn[i] = new QPushButton(vbtnStr[i]);
    vboxLayout->addWidget(vbtn[i]);
}

GridLayout *gridLayout = new GridLayout();
QPushButton *gbtn[5];

for(int i = 0 ; i < 5 ; i++)
    gbtn[i] = new QPushButton(btnStr[i]);

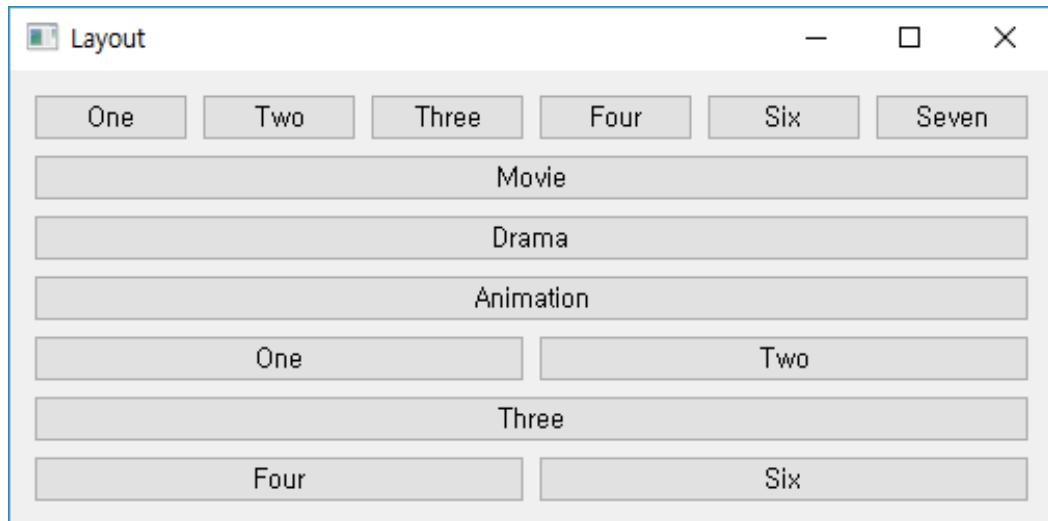
gridLayout->addWidget(gbtn[0], 0, 0);
gridLayout->addWidget(gbtn[1], 0, 1);
gridLayout->addWidget(gbtn[2], 1, 0, 1, 2);
gridLayout->addWidget(gbtn[3], 2, 0);
gridLayout->addWidget(gbtn[4], 2, 1);

QVBoxLayout *defaultLayout = new QVBoxLayout();
defaultLayout->addLayout(hboxLayout);
defaultLayout->addLayout(vboxLayout);
```

```
defaultLayout->addLayout(gridLayout);

setLayout(defaultLayout);

}
```



<그림> 중첩레이아웃 예제 실행 화면

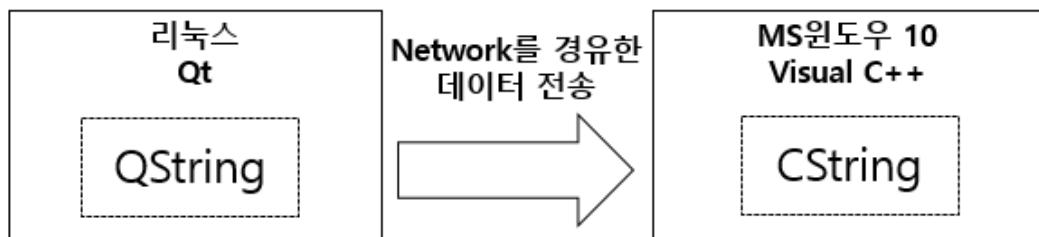
예제 소스는 Ch03 > 02_Layout > 01_Layout 디렉토리를 참조하면 된다.

3.3. Qt에서 제공하는 데이터 타입과 클래스

Qt는 개발자의 편의성을 위해 다양한 데이터타입을 제공한다. 예를 들어 QString과 같은 문자열 내에 특정 패턴을 찾아내기 위해 정규식 표현을 지원하거나 문자열에 특정 문자를 추가하거나 삭제 등 다양한 기능을 제공한다.

Qt에서는 이기 종간의 데이터 교환 시 데이터 타입의 변화로 생기는 문제를 해결하기 위한 데이터 타입도 지원한다. 예를 들어 우분투 리눅스 운영체제상에서 Qt로 개발한 어플리케이션 내에 "Hello World"라는 문자열을 Qt에서 제공하는 QString이라는 문자열을 전송한다.

그리고 이 문자열을 수신 받는 측에서는 MS윈도우 운영체제상에서 Visual C++로 개발하여 "Hello World" 문자열을 CString 문자열을 저장한다면 두 개의 데이터가 다른 데이터가 저장된다.



<그림> 이 기종 간의 데이터 교환 시 문제점

위의 그림에서 보는 것과 같이 이 기종 간의 문제를 해결하기 위해서 QString을 QByteArray로 변환해 전송하면 수신 받는 측에서 Char 형으로 사용할 수 있기 때문에 이기종 간의 데이터 문제를 해결할 수 있다. 또한 Qt는 QVariant 데이터 타입과 같이 void 형 데이터 타입도 지원한다. 다음 표는 Qt에서 주로 사용되는 데이터 타입이다.

<표> 기본 데이터 타입

타입	크기	설명
bool	8 bit	true / false
qint8	8 bit	signed char
qint16	16 bit	signed short
qint32	32 bit	signed int

qint64	64 bit	long long int
quint8	8 bit	unsigned char
quint16	16 bit	unsigned short
quint32	32 bit	unsigned int
quint64	64 bit	unsigned long long int
float	32 bit	IEEE 754 포맷을 사용하는 floating point number
double	64 bit	IEEE 754 포맷을 사용하는 floating point number
const char*	32 bit	문자열 상수를 가리키는 포인터, 마지막에 0 제외

Qt는 데이터 타입의 값을 판단/비교하기 위한 일반 함수와 템플릿 함수를 제공한다. 예를 들어 절대값을 구하기 위해 qAbs(), 최대/최소 사이의 값을 구하기 위해 qBound(), 최소값을 구하기 위한 qMin(), 최대값을 구하기 위한 qMax() 함수 등을 제공한다.

✓ 절대값 구하기

```
int absoluteValue;
int myValue = -4;

absoluteValue = qAbs(myValue); // absoluteValue == 4
```

✓ 최소값과 최대값 사이의 값 구하기

```
int myValue = 10;
int minValue = 2;
int maxValue = 6;

int boundedValue = qBound(minValue, myValue, maxValue);
// boundedValue == 6
```

✓ 오류 메시지를 핸들링 하기

```
int divide(int a, int b)
{
    if (b == 0) // program error
        qFatal("divide: cannot divide by zero");
```

```
    return a / b;  
}
```

✓ 소수점을 비교하기 위한 함수

```
// 0.0과 비교  
qFuzzyCompare(0.0, 1.0e-200); // return false  
qFuzzyCompare(1 + 0.0, 1 + 1.0e-200); // return true
```

✓ 최대값 구하기

```
int myValue = 6;  
int yourValue = 4;  
  
int maxValue = qMax(myValue, yourValue);  
int minValue = qMin(myValue, yourValue);
```

✓ int형 반올림

```
qreal valueA = 2.3;  
qreal valueB = 2.7;  
  
int roundedValueA = qRound(valueA); // roundedValueA = 2  
int roundedValueB = qRound(valueB); // roundedValueB = 3
```

✓ 64bit int형 반올림

```
qreal valueA = 42949672960.3;  
qreal valueB = 42949672960.7;  
  
int roundedA = qRound(valueA); // roundedA = 42949672960  
int roundedB = qRound(valueB); // roundedB = 42949672961
```

● 문자열 데이터 타입 클래스

Qt는 단순한 문자열을 다루는 것 외에도 데이터 스트림, 멀티 바이트 캐릭터 형태의 유니코드 4.0(Unicode Standard Version) 캐릭터를 지원하는 다양한 클래스를 제공한다.

<표> Qt에서 제공하는 다양한 문자열 데이터 타입 클래스

클래스	설명
QByteArray	바이트 단위의 배열을 지원하기 위한 클래스
QByteArrayMatcher	QByteArray로 구현된 바이트 단위의 배열의 index를 이용해 매칭되는 문자열이 있는지 찾기 위해 사용되는 클래스
QChar	16bit 유니코드 Character를 지원하기 위한 클래스
QLatin1Char QLatin1String	US-ASCII/Latin-1 인코딩 문자열을 지원하기 위해 제공되는 클래스
QLocale	숫자 표시 방식 혹은 문자 표시 방식을 다양한 언어의 표현 방식에 맞도록 변환하는 클래스입니다.
QString	유니코드 문자열 캐릭터를 지원하는 클래스
QStringList	문자열 리스트의 집합 클래스
QStringMatcher	문자열 상에 매칭되는 문자열을 찾기 위해 제공되는 클래스
QStringRef	size(), position(), toString()과 같은 서브 스트링 래핑 클래스
QTextStream	Text기반 WRITE/READ를 위한 STREAM 기능 제공
QDataStream	Binary 기반 WRITE/READ를 위한 STREAM 기능 제공

✓ QByteArray

QByteArray 클래스는 바이트(8-bit) 단위의 배열을 제공한다. QByteArray 클래스는 배열을 핸들링하기 위해 append(), prepen(), insert(), replace() 그리고 remove() 멤버 함수를 제공한다.

```

QByteArray x("Q");

x.prepend("I love");      // x == I love Q
x.append("t -^^*");       // x == I love Qt -^^*
x.replace(13, 1, "*");   // x == I love Qt *^^*

QByteArray x("I love Qt -^^*");
x.remove(13, 4); // x == I love Qt

```

✓ QByteArrayMatcher

바이트 배열에서 매칭되는 바이트 배열 패턴을 찾기 위해 제공되는 클래스이다.

```
// 전체 QByteArray
QByteArray x(" hello Qt, nice to meet you.");
QByteArray y("Qt"); // x에서 찾고자 하는 문자열

QByteArrayMatcher matcher(y);

//문자열이 시작되는 위치 index 변수에 저장
int index = matcher.indexIn(x, 0);

qDebug( "index : %d", index);
qDebug( "QByte : %c%c", x.at(index), x.at(index+1));
```

✓ QChar

16 Bit 유니코드를 지원하기 위한 Character 클래스 이다.

```
QLabel *lbl = new QLabel("", this);
QString str = "Hello Qt";
QChar *data = str.data();
QString str_display;

while(!data->isNull())
{
    str_display.append(data->unicode());
    ++data;
}

lbl->setText(str_display); // Hello Qt
```

✓ QLatin1String

US-ASCII/Latin-1 인코딩 문자열을 지원하기 위해 제공되는 클래스 이다.

```
QLatin1String latin("Qt");
QString str = QString("Qt");

if(str == latin)
    qDebug("Equal.");
```

```
else
    qDebug("Not equal.");

bool is_equal = latin.operator==(str);

if(is_equal)
    qDebug("Equal.");
else
    qDebug("Not equal.");
```

✓ QLocale

숫자와 문자를 다양한 언어로 변환하기 위해 사용한다.

```
QLocale egyptian(QLocale::Arabic, QLocale::Egypt);
QString s1 = egyptian.toString(1.571429E+07, 'e');
QString s2 = egyptian.toInt(10);

double d = egyptian.toDouble(s1);
int i = egyptian.toInt(s2);
```

✓ QString

QString 클래스는 유니코드 문자열을 지원하며 16bit QChar를 저장할 수 있는 기능 제공한다.

```
QString str = "Hello";
```

QString 은 const char * 와 같은 문자열 상수를 fromUtf8() 함수를 사용해 대체할 수 있는 기능을 제공한다.

```
static const QChar data[4] = {0x0055, 0x006e, 0x10e3, 0x03a3 };
QString str(data, 4);
```

저장된 문자열의 특정 위치에 QChar 을 저장할 수 있는 기능 제공한다.

```
QString str;
str.resize(4);
str[0] = QChar('U');
str[1] = QChar('n');
str[2] = QChar(0x10e3);
str[3] = QChar(0x03a3);
```

문자열을 비교하기 위해 QString 은 if 문을 이용해 다음과 같은 비교가 가능하다.

```
QString str;

if ( str == "auto" || str == "extern" || str == "static" || str == "register" )
{
    // ...
}
```

찾고자 하는 특정 문자열의 위치를 알고 싶다면 indexOf() 함수를 사용해 찾고자 하는 문자열의 위치를 찾을 수 있다.

```
QString str = "We must be <b>bold</b>, very <b>bold</b>";
int j = 0;

while ((j = str.indexOf("<b>", j)) != -1) {
    qDebug() << "Found <b> tag at index position" << j;
    ++j;
}
```

✓ QStringList

QString 에 저장된 문자열을 QList 와 같이 문자열을 배열 형태로 관리할 수 있는 기능을 제공하며 append(), prepend(), insert() 등의 멤버 함수를 제공한다.

```
QStringList strList;
strList << "Monday" << "Tuesday" << "Wednesday";

QString str;
str = strList.join(",");
// str == " Monday, Tuesday, Wednesday"
```

✓ QByteArrayMatcher

QString 문자열을 비교해 매칭되는 문자를 찾기 위한 기능을 제공한다.

```
QString x("hello World, nice to meet you."); // 전체 QString
QString y("World"); // x에서 찾고자 하는 문자열

QStringMatcher matcher(y);
int index = matcher.indexIn(x, 0);
```

```
//문자열이 시작되는 위치 index변수에 저장  
qDebug("index : %d", index); // index : 6
```

✓ QTextStream

QTextStream 클래스는 대량의 텍스트 데이터를 다루기 위해 STREAM 처리 방식을 제공한다. STREAM 방식을 이용하면 대량의 데이터를 효율적으로 빠르게 접근해 데이터를 READ하거나 WRITE 할 수 있다. 다음 예제는 QFile 클래스를 이용해 파일로부터 READ 한 데이터를 QTextStream 을 사용한 예제이다.

```
QFile file("in.txt");  
if (!file.open(QIODevice::ReadOnly | QIODevice::Text))  
    return;  
  
QTextStream in(&file);  
while (!in.atEnd()) {  
    QString line = in.readLine();  
    ...  
}
```

● Container 클래스

Container 클래스는 특정 유형의 데이터 항목 또는 집합을 저장하는데 사용한다. 예를 들어 QString으로 저장해야 할 항목이 여러 개 있다면 QVector<QString>과 같은 Container 클래스를 사용할 수 있다.

Container 클래스들은 STL에서 제공하는 Container들보다 사용하기 쉽고 안전하다. 또한 경량화 되어 있다. Qt에서 제공하는 Container는 STL에서 제공하는 Container를 대체해 사용할 수 있다.

<표> Qt에서 제공하는 Container 클래스

클래스	설명
QHash<Key, T>	Hash 테이블 기반의 Dictionary를 제공하는 템플릿 클래스
QMap<Key, T>	Binary Search Tree기반 템플릿 클래스
QPair<T1, T2>	Pair(쌍)로 존재하는 아이템 데이터를 처리 클래스.
QList<T>	List 형태의 값을 다루기 위해 제공하는 템플릿 클래스

QLinkedList<T>	링크드리스트를 제공하기 위한 템플릿 클래스
QVector<T>	동적인 QVector 배열을 다루기 위해 제공되는 클래스
QStack<T>	Stack 기반의 push(), pop() 등을 사용하기 위해 제공
QQueue<T>	FIFO 구조의 데이터 조작을 위해 제공
QSet<T>	해시 기반의 빠른 검색을 위해 제공되는 클래스
QMap<Key, T>	Key값에 의해 Mapping 되는 데이터를 하나의 조합으로 연결되어 Dictionary를 제공한다.
QMultiMap<Key, T>	QMap으로부터 상속받은 클래스이며 다중 Mapping 값을 사용할 수 있다.
QMultiHash<Key, T>	QHash로부터 상속받은 클래스이며 다중 Mapping 값을 이용해 해쉬를 사용할 수 있다.

✓ QHash<Key, T>

QHash 클래스는 해시 테이블 기반의 Dictionary를 제공한다. 데이터를 저장하는 방식은 Key, Value 가 Pair(쌍)로 저장 된다. Key 값으로 찾고자 하는 데이터를 빠르게 검색 할 수 있는 기능을 제공한다. QHash는 QMap과 매우 비슷한 기능을 제공하지만 내부 알고리즘은 QMap 보다 빠르다.

```
QHash<QString, int> hash;

hash["one"] = 1;
hash["three"] = 3;
hash["seven"] = 7;
```

QHash에 Key, Value를 쌍으로 저장하기 위한 방법으로 insert() 함수를 사용할 수 있다. 그리고 Value 값을 알기 위해 value() 멤버 함수를 사용할 수 있다.

```
hash.insert("twelve", 12);
int num1 = hash["thirteen"];
int num2 = hash.value("thirteen");
```

✓ QMap<Key, T>

QMap의 사용 방법은 QHash와 유사하다. 다음 예는 QString을 Key로 사용하고 int를

Value로 사용하는 예제 소스코드 이다.

```
QMap<QString, int> map;

map[ "one" ] = 1;
map[ "three" ] = 3;
map[ "seven" ] = 7;

map.insert("twelve", 12);

int num1 = map[ "thirteen" ];
int num2 = map.value("thirteen");
```

다음 예제는 Key 값을 이용해 Value 값을 얻어 오기 위한 방법으로 contains() 함수를 사용할 수 있다.

```
int timeout = 30;
if (map.contains("TIMEOUT"))
    timeout = map.value("TIMEOUT");
```

✓ QPair<T1, T2>

QPair 클래스는 두 개의 아이템을 하나로 구성된 Pair로 저장할 수 있다. 아래 예제에서 보는 것과 같이 QPair 클래스는 다음과 같이 선언하여 사용할 수 있다

```
QPair<QString, double> pair;

pair.first = "pi";
pair.second = 3.14159265358979323846;
```

✓ QList< T >

QList<T>는 빠른 인덱스 기반의 액세스가 가능하며 저장된 데이터 삭제도 매우 빠르다. QList는 인덱스 기반의 클래스이며 QLinkedList의 Iterator 기반보다 사용하기 편리하며 데이터 저장 시 메모리 할당하는 속도에서 QVector 보다 빠르다.

```
QList<int> integerList;
QList<QDate> dateList;
QList<QString> list = { "one", "two", "three" };
```

QList는 비교 연산자를 통해 리턴 값을 아래 예와 같이 사용할 수 있다.

```
if (list[0] == "Bob")
    list[0] = "Robert";
```

QList는 at() 함수를 이용하면 리스트 상에 저장된 위치를 쉽게 검색 할 수 있다.

```
for (int i = 0; i < list.size(); ++i)
{
    if (list.at(i) == "Jane")
        cout << "Found Jane at position " << i << endl;
}
```

✓ QLinkedList<T>

QLinkedList 클래스는 iterator 기반으로, 리스트의 아이템을 저장 및 삭제하는 기능을 제공한다. 다음 예에서 보는 것과 같이 선언할 수 있다.

```
QLinkedList<int> integerList;
QLinkedList<QTime> timeList;
```

리스트 상에 아이템을 추가하기 위해 다음 예와 같이 operator를 사용할 수 있다.

```
QLinkedList<QString> list;

list << one << two << three ;
// list: [ one , two , three ]
```

✓ QVector<T>

QVector는 메모리 위치에 해당 항목을 저장하고 빠른 인덱스 기반의 액세스를 제공한다.

```
QVector<int> integerVector;
QVector<QString> stringVector;
```

저장할 데이터 개수의 크기를 미리 지정할 수 있으며 Vector 크기는 선언 시 할당한다.

예를 들어 200개의 저장 공간을 가지는 Vector를 다음과 같이 선언할 수 있다. 그리고 미리 선언한 200개의 저장 공간에 Default 값을 설정할 수 있다.

```
QVector<QString> vector(200);
QVector<QString> vector(200, "Pass" );
```

✓ QStack<T>

QStack은 Stack 알고리즘을 제공하기 위한 클래스이다. 나중에 삽입된 데이터가 먼저 나오는 구조(LIFO) 이다.

```
QStack<int> stack;  
stack.push(1);  
stack.push(2);  
stack.push(3);  
  
while (!stack.isEmpty())  
    cout << stack.pop() << endl;
```

✓ QQueue<T>

QQueue는 Queue 알고리즘을 제공하기 위한 클래스로, 먼저 삽입된 데이터가 먼저 나오는 구조(FIFO) 이다.

```
queue.enqueue(1);  
queue.enqueue(2);  
queue.enqueue(3);  
  
while (!queue.isEmpty())  
    cout << queue.dequeue() << endl;
```

✓ QSet<T>

QSet은 검색 시 속도가 매우 빠르다. QSet의 내부구조는 QHash로 구현되었다. 다음 예는 선언과 사용하는 방법이다.

```
QSet<QString> set;  
  
set.insert("one");  
set.insert("three");  
set.insert("seven");  
  
set << "twelve" << "fifteen" << "nineteen";
```

✓ QMultiMap<Key, T>

QMap 클래스로부터 상속받은 클래스이며 다중 Mapping 값을 사용할 수 있는 기능을

제공하며 QMap을 확장해 사용할 수 있다.

```
QMultiMap<QString, int> map1, map2, map3;
map1.insert("plenty", 2000); // map1.size() == 2
map2.insert("plenty", 5000); // map2.size() == 1
map3 = map1 + map2; // map3.size() == 3
```

✓ QMultiHash<Key, T>

QMultiHash 클래스는 다중의 Hash 값을 저장하기 위한 용도로 적합하다. QHash는 다중의 동일한 값을 허용하지 않지만 QMultiHash는 동일한 다중의 값을 허용하는 특징이 있다.

```
QMultiHash<QString, int> hash1, hash2, hash3;
hash1.insert("plenty", 100);
hash1.insert("plenty", 2000); // hash1.size() == 2
hash2.insert("plenty", 5000); // hash2.size() == 1

hash3 = hash1 + hash2; // hash3.size() == 3
```

● 데이터 클래스

Qt에서 제공하는 기본 데이터 타입과 더불어 유연한 데이터 조작을 위해 다양한 클래스를 제공한다. 예를 들어 Bit 단위의 데이터를 다루기 위한 QBitArray, Byte 단위의 데이터를 배열 형태로 다루기 위해 QByteArray 등과 같이 데이터 조작에 편리한 클래스를 제공한다. 여기서는 데이터를 다루는데 유용한 클래스에 대해 알아보도록 하자.

<표> Qt에서 제공하는 데이터 클래스

클래스	설명
QBitArray	비트 연산(AND, OR, XOR, NOT)을 제공하기 위한 bit Array
QMargins	left, top, right, bottom 사각형의 각 Margin을 처리하기 위한 클래스
QPoint	X, Y, Z 값을 처리하기 위해 제공하는 클래스
QQuaternion	벡터 및 스칼라로 구성된 Quaternion을 처리하기 위한 클래스
QRect	사각형의 left (qint32), top (qint32), right (qint32), bottom (qint32)
QRegExp	정규화 표현식을 처리하기 위한 클래스

QRegion	Painter 상에서 클립보드 영역을 정의하기 위해 사용
QSize	넓이와 높이를 저장 할 수 있는 클래스
QVariant	void 타입으로 저장 할 수 있는 union 형태의 클래스
QVector2D	2차원 공간에서 Vector 또는 vertex를 나타내기 위한 클래스
QVector3D	x, y, z 3개의 좌표를 사용할 수 있는 클래스
QVector4D	4차원 공간에서 Vector 또는 vertex를 나타내기 위해 사용.

✓ QBitArray

QBitArray 클래스는 Bit 단위를 데이터로 관리할 수 있는 기능을 제공한다. AND, OR, XOR 그리고 NOT 연산을 통해 Bit 연산 기능을 제공한다.

```
QBitArray ba(200);

QBitArray ba;
ba.resize(3);
ba[0] = true;
ba[1] = false;
ba[2] = true;

QBitArray x(5);
x.setBit(3, true); // x: [ 0, 0, 0, 1, 0 ]
QBitArray y(5);
y.setBit(4, true); // y: [ 0, 0, 0, 0, 1 ]
x |= y; // x: [ 0, 0, 0, 1, 1 ]
```

✓ QMargins

QMargins 클래스는 Left, Top, Right, Bottom을 저장할 수 있으며 setLeft(), setRight(), setTop() 그리고 setBottom() 멤버 함수를 이용해 값을 저장 또는 수정할 수 있다.

```
QMargins margin;

margin.setLeft(10);
margin.setTop(12);
margin.setRight(14);
margin.setBottom(16);
```

```
qDebug() << "QMargins Value : " << margin;  
// QMargins Value : QMargins(10, 12, 14, 16)
```

✓ QPoint

QPoint 클래스는 좌표 지점을 표시하기 위해 사용되는 기능을 제공한다.

```
QPoint p;  
p.setX(p.x() + 1);  
p += QPoint(1, 0);  
p.rx()++;
```

QPoint 는 operator를 통해 아래와 같이 사용할 수 있다.

```
QPoint p( 3, 7);  
QPoint q(-1, 4);  
  
p += q; // p becomes (2, 11)
```

✓ QQuaternion

QQuaternion은 벡터 및 스칼라로 구성된 Quaternion을 사용할 수 있다. Quaternion은 3D 공간에서 SCALAR, X, Y 그리고 Z 좌표와 회전과 같은 각도를 3D에 사용되는 데이터를 관리하는 기능을 제공한다.

```
QQuaternion yRot;  
yRot = QQuaternion::fromAxisAndAngle(0.0f, 1.0f, 0.0f, horiAng * radiDeg);
```

✓ QRect 와 QRectF

QRect 클래스는 정수(Integer), QRectF는 실수(Float)를 사용해 사각형의 좌표 값을 저장하기 위한 용도로 사용할 수 있다. 사각형의 X, Y, Width, Height 값을 저장 할 수 있다.

```
QRect r1(100, 200, 11, 16);  
QRect r2(QPoint(100, 200), QSize(11, 16));  
  
QRectF rf1(100.0, 200.1, 11.2, 16.3);  
QRectF rf2(QPointF(100.0, 200.1), QSizeF(11.2, 16.3));
```

✓ QRegExp

QRegExp 클래스는 정규식 표현 기능을 제공한다.

```
QRegExp rx("^\\d\\d?"); // 0~99 의 정수값인 경우  
  
rx.indexIn("123"); // return -1  
rx.indexIn("-6"); // return -1  
rx.indexIn("6"); // return 0
```

✓ QRegion

QRegion은 QPainter::setClipRegion() 함수와 함께 사용한다. Painter 상에서 사각형 내부의 특정 영역을 사각형으로 클립보드로 사용하기 위해 다음과 같이 사용할 수 있다.

```
void MyWidget::paintEvent(QPaintEvent *)  
{  
    QRegion r1(QRect(100, 100, 200, 80), QRegion::Ellipse);  
    QRegion r2(QRect(100, 120, 90, 30));  
    QRegion r3 = r1.intersected(r2);  
  
    QPainter painter(this);  
    painter.setClipRegion(r3);  
    ...
```

✓ QSize

QSize 클래스는 int 값을 이용해 width, height 를 저장하기 위해 주로 사용된다.

```
QSize size(100, 10);  
size.setHeight() += 5;  
// size (100,15)
```

✓ QVariant

QVariant 클래스는 데이터 타입이 정해지지 않은 유형의 데이터 타입을 지정할 수 있는 기능을 제공한다. 예를 들어 void * 와 같이 지정할 수 있다.

```
QDataStream out(...);  
QVariant v(123); // int 형을 저장  
int x = v.toInt(); // x = 123
```

```

out << v;
v = QVariant("hello");           // QByteArray 타입의 문자열을 저장
v = QVariant(tr("hello")); // QString 타입의 문자열을 저장

int y = v.toInt();
QString s = v.toString();
out << v;
...

QDataStream in(...);
in >> v;
int z = v.toInt();
qDebug("Type is %s", v.typeName());

v = v.toInt() + 100;
v = QVariant(QStringList());

```

✓ QVector2D, QVector3D 그리고 QVector4D

QVector2D는 2D 좌표에서 Vector와 Vertex를 다루기 위해 사용된다. QVector3D는 X, Y에 Z가 추가된 클래스이다. 그리고 QVector4D 클래스는 W가 추가된 기능을 제공한다.

```

...
void GeometryEngine::initCubeGeometry()
{
    VertexData vertices[] = {
        // Vertex data for face 0
        {QVector3D(-1.0f, -1.0f, 1.0f), QVector2D(0.0f, 0.0f)},
        {QVector3D( 1.0f, -1.0f, 1.0f), QVector2D(0.33f, 0.0f)},
        {QVector3D(-1.0f, 1.0f, 1.0f), QVector2D(0.0f, 0.5f)},
        {QVector3D( 1.0f, 1.0f, 1.0f), QVector2D(0.33f, 0.5f)},
        // Vertex data for face 1
        {QVector3D( 1.0f, -1.0f, 1.0f), QVector2D( 0.0f, 0.5f)},
        {QVector3D( 1.0f, -1.0f, -1.0f), QVector2D(0.33f, 0.5f)},
        {QVector3D( 1.0f, 1.0f, 1.0f), QVector2D(0.0f, 1.0f)},
        {QVector3D( 1.0f, 1.0f, -1.0f), QVector2D(0.33f, 1.0f)},
    ...
}
```

3.4. Signal 과 Slot

Qt는 이벤트를 처리하기 위한 메커니즘으로 시그널(Signal)과 슬롯(Slot)을 사용한다. 예로 어떤 버튼이 클릭했다는 행위는 Qt에서 시그널(Signal)이라고 한다. 그리고 시그널이 발생하면 호출하는 함수를 슬롯(Slot) 함수라고 한다. 시그널이라는 이벤트가 발생하면 시그널과 연결된 슬롯 함수가 호출된다.

시그널이라는 것은 어떠한 상황에 발생하는 이벤트이다. Qt의 모든 이벤트 처리는 시그널과 슬롯이라는 메커니즘을 사용한다.

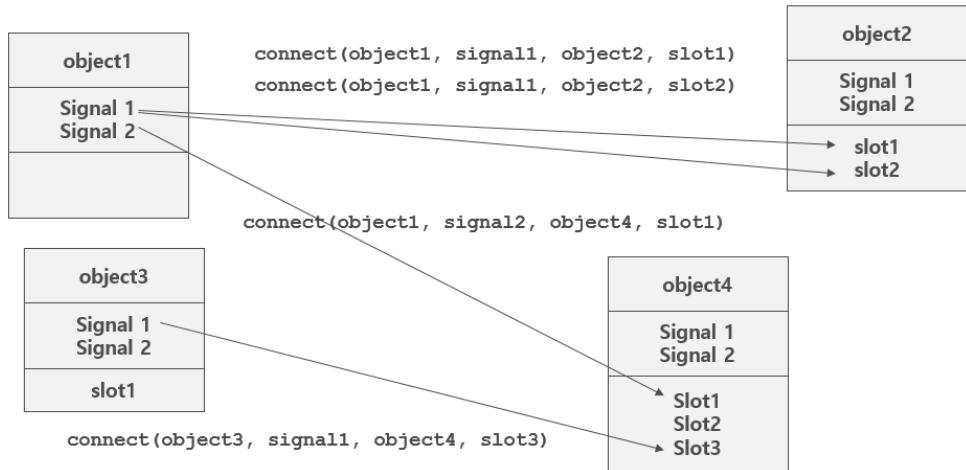
예를 들어 Qt로 채팅 프로그램에서 A라는 사용자가 B라는 사용자에게 메시지를 보낸다고 가정해보자 B의 입장에서 A에게로부터 메시지를 받은 행위는 시그널(Signal)이라고 정의할 수 있다.

그리고 메시지를 받은 시그널과 연결된 슬롯(Slot) 함수를 호출한다. Qt는 모든 이벤트 처리를 시그널과 슬롯을 사용한다. 방금 예로든 GUI 와 네트워크모듈뿐만 아니라 모든 API에서 시그널과 슬롯이라는 이벤트 메커니즘을 사용한다. 시그널과 슬롯은 프로그램 소스코드를 단순화 시켜주기 때문에 개발기간을 단축 시킬 수 있으며 복잡한 프로그램 구조를 단순화 할 수 있다.

네트워크 채팅 프로그램을 Qt의 시그널 슬롯을 사용하지 않고 채팅 프로그램을 개발한다고 가정해보자. 여러 개의 쓰레드(Thread) 구조의 프로그램을 개발해야 한다. 예를 들어 특정 사용자가 보내는 메시지를 처리하는 쓰레드, 깃속말 처리 쓰레드, 새로운 사용자가 등록되면 알려주는 쓰레드 등 여러 개의 쓰레드를 사용해야 하기 때문에 프로그램이 복잡해 질 수 있다. 하지만 Qt에서 시그널과 슬롯을 사용하면 쓰레드가 필요 없어 지기 때문에 매우 간단하게 구현할 수 있다.

Qt에 제공하는 모든 GUI 위젯은 미리 정해진 다양한 시그널을 가지고 있다. 예를 들어 QPushButton의 click, double click, mouse over 등과 같이 다양한 시그널이 정의되어 있다.

시그널과 슬롯은 하나의 파이프라인과 같이 생각하면 된다. 하나의 시그널이 여러 개의 슬롯 함수를 호출할 수 있다. 또한 여러 개의 시그널이 하나의 슬롯을 호출할 수 있다.



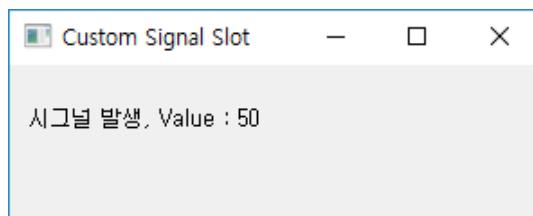
<그림> Signal 과 Slot 의 Architecture

시그널과 슬롯 함수를 연결하기 위한 함수는 QObject 클래스의 connect() 함수를 이용해 Signal 과 Slot을 연결할 수 있다. connect() 멤버 함수의 첫 번째 인자는 이벤트가 발생한 오브젝트(클래스의 인스턴스), 두 번째 인자는 오브젝트의 시그널(이벤트)을 입력한다.

예를 들어 A라는 버튼이 있으면 A라는 버튼의 오브젝트 명이 첫 번째 인자이고 두 번째 인자는 A버튼의 클릭 또는 더블클릭이 시그널이 될 수 있다. 따라서 클릭 이벤트를 두번째 인자로 명시한다. 세 번째 인자는 시그널과 호출할 슬롯 함수 있는 오브젝트의 이름을 명시한다. 네 번째 인자는 발생한 시그널 발생 시 호출할 슬롯 함수를 명시한다. 지금까지 Signal 과 Slot 에 대한 개념에 대해 살펴보았다. 이번에는 직접 Signal 과 Slot을 이용해 예제 프로그램 작성해 보도록 하자.

● Signal 과 Slot 예제

이 예제 소스코드는 시그널이 발생하면 윈도우 상에 배치된 라벨의 텍스트를 출력하는 예제이다.



<그림> Signal 과 Slot 예제 실행 화면

이 예제는 SignalSlot 이라는 클래스와 Widget 이라는 두개의 클래스가 존재한다. 다음 예제 소스코드는 SignalSlot 이라는 클래스이다.

```
...
class SignalSlot : public QObject
{
    Q_OBJECT

public:
    void setValue(int val) {
        emit valueChanged(val);
    }

signals:
    void valueChanged(int newValue);

private:
    int m_value;
};
```

위의 예제에서 보는 것과 같이 signals: 키워드 하단에 valueChanged() 라는 함수가 있다. 이 함수가 시그널 함수이다.

Signal 함수는 구현 부는 없으며 소스코드에서 보는 것과 같이 헤더에 정의 부만 구현하면 된다. valueChanged() Signal 함수는 int 인자를 명시하였다. 이 인자는 시그널을 발생할 때 값을 Slot함수에게 인자로 전달할 수 있다. 여기서는 하나의 값을 사용하였다. 필요에 따라 인자를 사용하지 않거나 여러 개를 인자로 사용할 수 있다.

위의 예제 소스코드의 SignalSlot 이라는 클래스의 public 키워드에 setValue() 멤버 함수이다. 이 멤버 함수가 호출되면 함수 내에 emit 이라는 키워드를 사용한 소스코드를 확인할 수 있다. emit 키워드는 시그널 이벤트를 발생한다. 다음 예제 소스코드는 Widget 클래스의 헤더 부분이다.

```
...
class Widget : public QWidget
{
    Q_OBJECT
public:
    Widget(QWidget *parent = nullptr);
    ~Widget();
private:
    QLabel *lbl;
```

```
public slots:  
    void setValue(int val);  
};
```

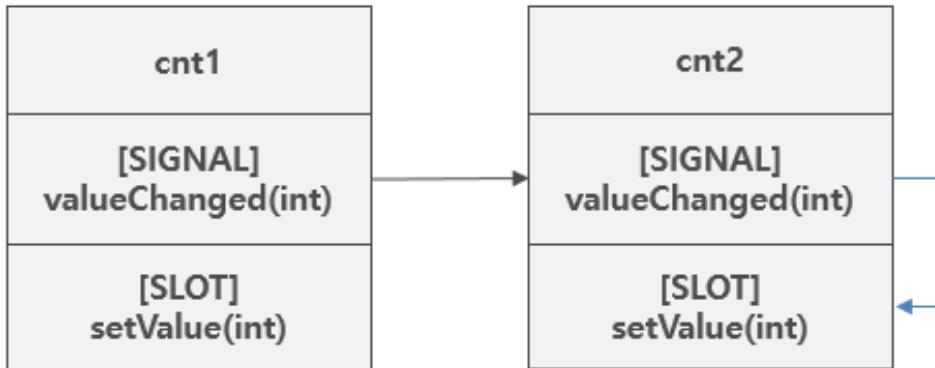
Widget 클래스의 하단에 보면 public 키워드에 slots 키워드를 함께 사용한 것을 확인 할 수 있을 것이다. slots 키워드는 private 또는 public 키워드와 함께 사용할 수 있다. slots 라는 키워드를 사용하여 명시한 멤버 함수는 Slot 함수를 정의할 수 있다. 다음 예제 소스코드는 Widget 클래스의 구현 부분 소스코드이다.

```
#include "widget.h"  
  
Widget::Widget(QWidget *parent) : QWidget(parent)  
{  
    lbl = new QLabel("", this);  
    lbl->setGeometry(10, 10, 250, 40);  
  
    SignalSlot myObject;  
  
    // New Style  
    connect(&myObject, &SignalSlot::valueChanged, this, &Widget::setValue);  
  
    /* Old Style  
    connect(&myObject, SIGNAL(valueChanged(int)), this, SLOT(setValue(int)));  
    */  
  
    myObject.setValue(50);  
}  
  
void Widget::setValue(int val)  
{  
    QString text = QString("시그널발생, Value:%1").arg(val);  
    lbl->setText(labelText);  
}
```

connect() 멤버 함수를 이용해 Signal 과 Slot 을 연결 할 수 있다. Signal 과 Slot 을 연결하는 방법에는 두 가지 방법을 제공한다. 주석으로 New Style 이라고 되어 있는 방식은 Qt 5.5 이상 버전에서 추가된 Signal과 Slot을 연결하는 방식이다. New Syntax 과 Old Style 모두 사용할 수 있다. Qt 5.5 이하버전에서는 Old Style 방식만을 사용할 수 있다. New Style 스타일은 여러 개의 인자를 사용할 수 없다. 그리고 New Style 은 Slot 함수 뿐만 아니라 일반 멤버 함수도 Slot 함수와 같이 connect() 함수에서 4번째 인자

로 사용할 수 있다.

살펴본 예제와 같이 Signal 과 Slot함수는 다른 클래스에만 구현되어 있어야 되는 것은 아니다. 동일한 클래스에 존재하는 Signal 이 Slot 함수를 호출할 수 있다. 그리고 다음 그림에서 보는 것과 같이 Signal 은 Slot 함수 외에도 Signal을 호출할 수 있다.



<그림> Signal 이 Signal 을 호출

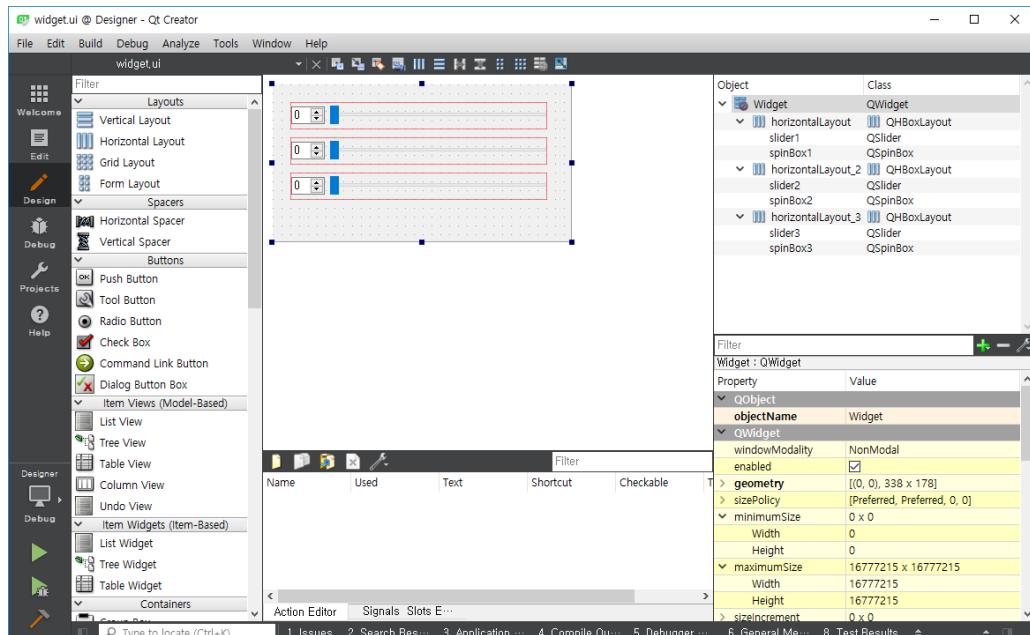
```
cnt1 = new Counter("counter 1");
cnt2 = new Counter("counter 2");

connect(cnt1, SIGNAL(valueChaged(int)), cnt2, SIGNAL(valueChaged(int)));
connect(cnt2, SIGNAL(valueChaged(int)), cnt2, SLOT(setValue(int)));
```

Signal 과 Slot을 사용하는 클래스 상단에 보면 Q_OBJECT 라는 키워드를 사용한 것을 확인할 수 있다. 이 키워드는 Qt에서 Signal 과 Slot을 사용할 때 반드시 필수로 Q_OBJECT를 클래스 헤더에 보는 것과 같이 명시 해야한다. Q_OBJECT를 선언하지 않고 Signal 과 Slot을 사용하는 경우가 있는데, 이럴 경우 에러가 발생하므로 주의해야 한다. 이번 예제의 전체 소스는 ch03 > 04_SignalSlot > 01_CustomSignalSlot 디렉토리를 참조하면 된다.

3.5. Qt Designer 를 이용한 GUI 설계

Qt 는 원하는 GUI 를 쉽게 빠르게 구현할 수 있도록 Qt Designer 를 제공한다. Qt Designer 는 사용자가 GUI 상에 배치할 위젯을 마우스로 드래그 하면서 위젯을 배치할 수 있다. Qt Creator IDE 툴이 제공되지 않은 시절에는 Qt Designer 라는 툴이 독립적으로 제공하였다. 하지만 지금은 Qt Creator IDE 툴에 통합되었다. Qt Designer 툴은 아직도 독립적인 실행 프로그램으로 제공한다. 그 이유는 아직도 Visual Studio 와 같은 IDE 툴을 이용하는 개발자를 위해서이다.



<그림> Qt Creator 내에 포함된 Designer 화면

확장자가 ui 파일인 GUI 파일은 아래 소스코드와 같이 XML 포맷으로 되어 있다. 이 파일은 사용자가 마우스를 이용해 위젯을 배치하면 Qt Creator 의 Designer 툴이 자동으로 XML 로 작성한다. 그리고 빌드를 하게 되면 XML 로 되어 있는 GUI 파일을 C++ 소스코드로 변환 후 빌드 한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>Widget</class>
<widget class="QWidget" name="Widget">
<property name="geometry">
```

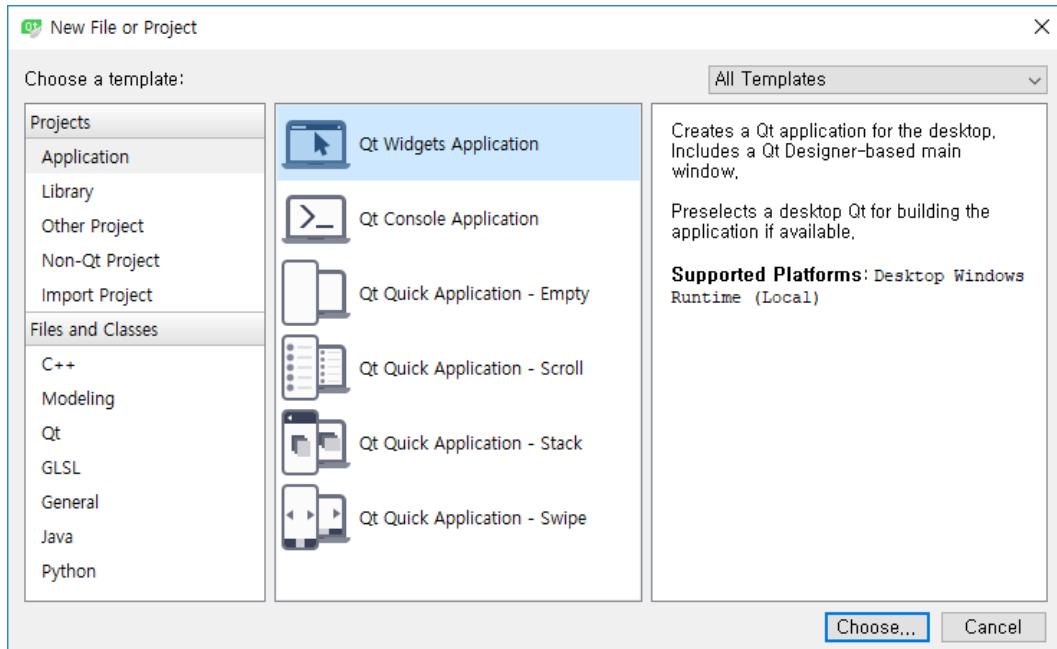
```

<rect>
<x>0</x>
<y>0</y>
<width>338</width>
<height>178</height>
</rect>
</property>
<property name="windowTitle">
<string>Widget</string>
</property>
<widget class="QWidget" name="horizontalLayoutWidget">
...

```

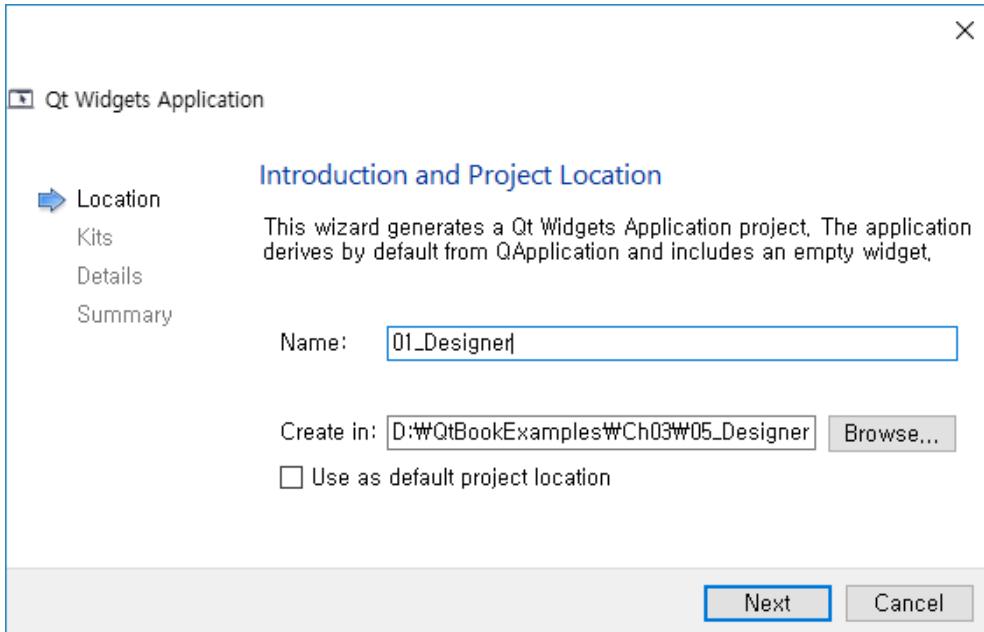
● Qt Designer 를 이용한 예제

다음 그림에서 보는 것과 같이 Qt Creator 를 실행한 후 메뉴에서 [File] > [New file or Project...] 메뉴를 클릭한다. 다이얼로그가 로딩되면 다이얼로그에서 좌측의 [Projects] 항목 중에서 [Application] 항목을 선택한다.



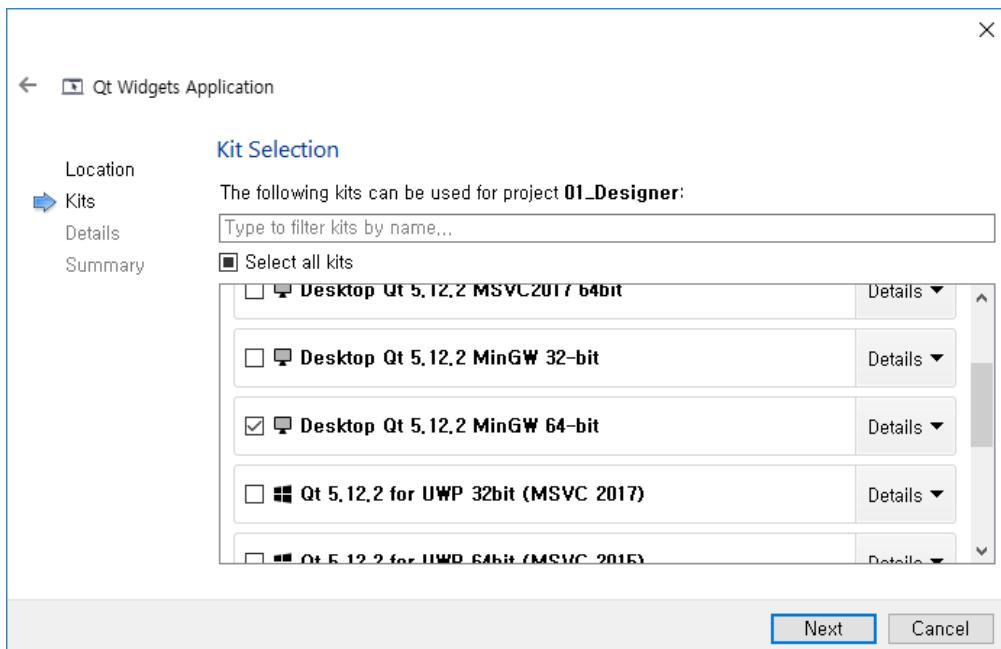
<그림> New File or Project 다이얼로그 화면

그런 다음 중앙에 항목 중 [Qt Widgets Application] 을 선택하고 하단의 [Choose] 버튼을 클릭한다. 다음 화면은 프로젝트 이름과 생성되는 위치를 입력하고 [Next] 버튼을 클릭한다.



<그림> 프로젝트의 이름과 위치 지정 다이얼로그

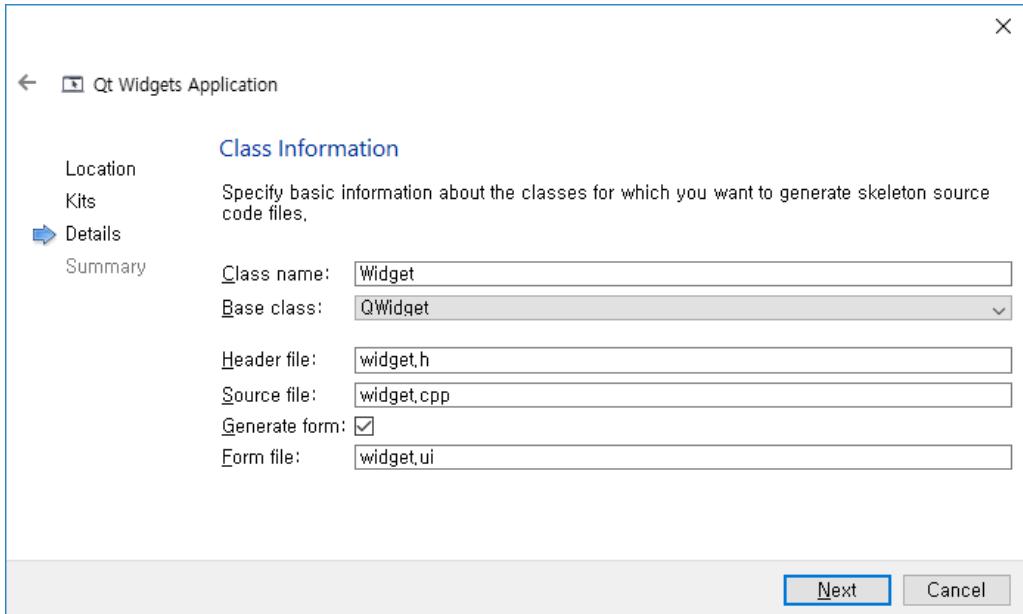
다음은 어플리케이션을 빌드할 컴파일러를 선택한다. MinGW 32 bit 또는 64 bit 버전 중 하나를 선택한다. 선택을 완료한 후 하단의 [Next] 버튼을 클릭한다.



<그림> 빌드할 컴파일러 선택 화면

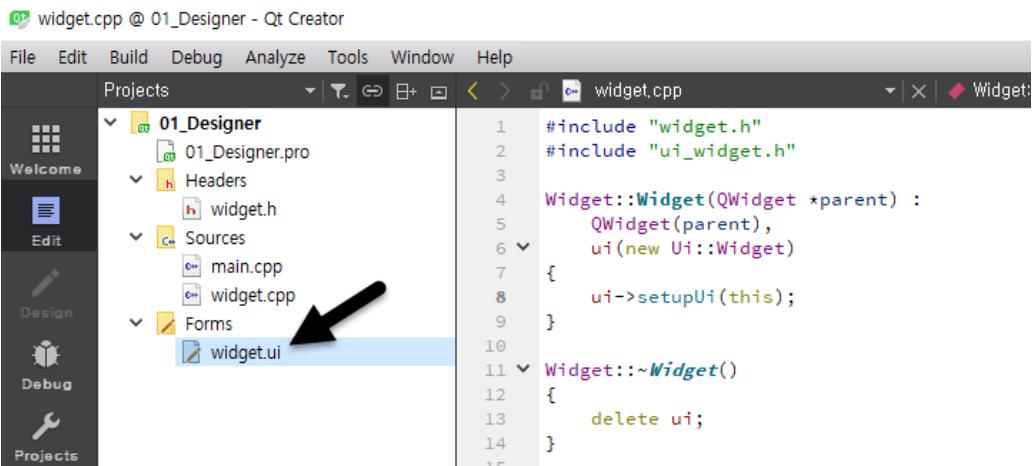
다음 화면은 클래스를 정의하기 위한 다이얼로그이다. Base class 항목에서 QWidget 을 선택한다. 그리고 Generate form 체크박스를 활성화(체크)한다.

이 항목은 이번 절에서 설명한 Designer 를 사용해 화면을 디자인하기 위한 기능을 사용하기 위한 항목이다. widget.ui 는 widget 클래스의 UI 파일을 자동생성 해 준다. 다음 그림에서 보는 것과 같이 선택 후 하단의 [Next] 버튼을 클릭한다.

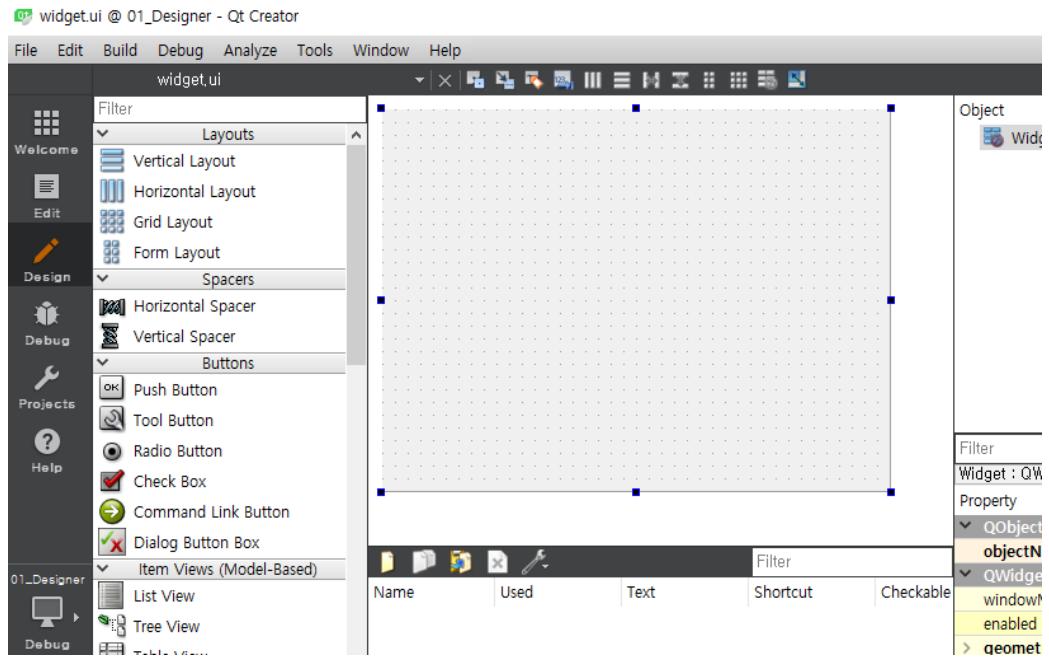


<그림> Class Information 다이얼로그 화면

다음 다이얼로그 창에서는 하단의 [Finish] 버튼을 클릭하면 프로젝트 생성이 완료된다. 프로젝트 생성이 완료되면 다음 그림에서 보는 것과 같이 Qt Creator 좌측에 프로젝트 창에서 widget.ui 소스코드를 더블 클릭하면 Designer 화면으로 전환된다.

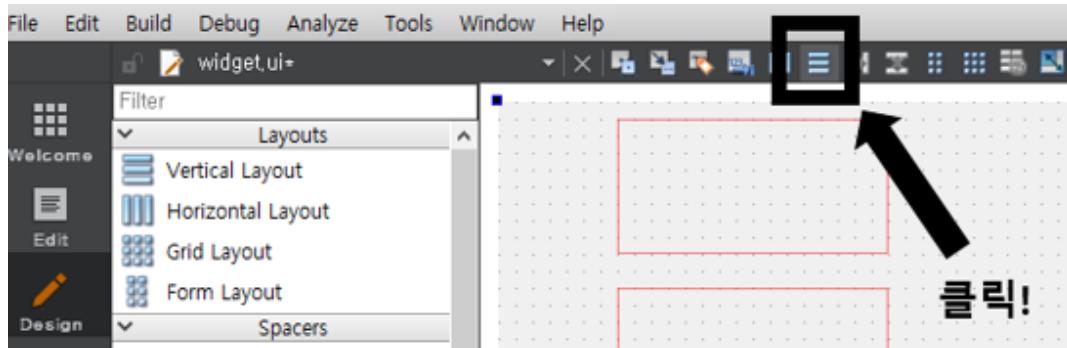


<그림> Qt Creator 화면



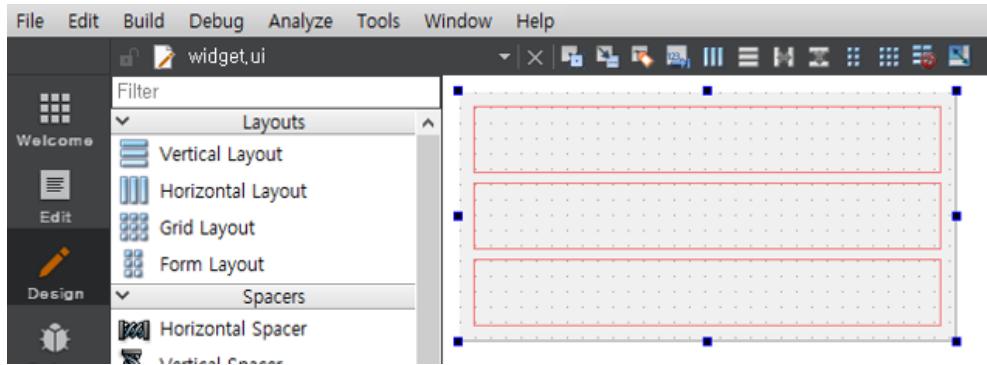
<그림> Designer로 전환된 화면

이전 그림에서 보는 것과 같이 Designer 툴로 전환한 화면에서 위젯들을 배치해 보자. 첫 번째로 좌측 [Layout] 탭에서 3 개의 Horizontal Layout 을 마우스로 드래그 해 GUI 위젯에 배치한다. 그리고 다음 그림에서 보는 것과 같이 상단의 툴바에서 세로모양으로 직사각형 3 개가 쌓여 있는 아이콘 버튼을 클릭한다.



<그림> 세로 방향으로 배치하기 위한 아이콘 클릭 화면

이전 그림에서 보는 것과 같이 아이콘을 클릭하면 윈도우 상에 Horizontal Layout 이 윈도우 크기가 변경되면 Horizontal Layout 이 자동으로 크기를 조절한다.



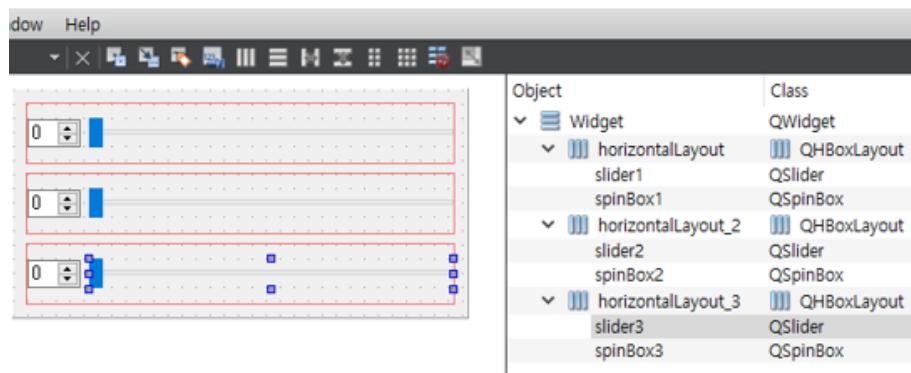
<그림> 세로방향으로 Horizontal Layout 을 배치한 화면

이전 그림에서 보는 것과 같이 화면 배치를 완료하였으면 각각의 Horizontal Layout 에 QSpinBox 와 QSlider 를 배치하자.

<표> 배치할 위젯들

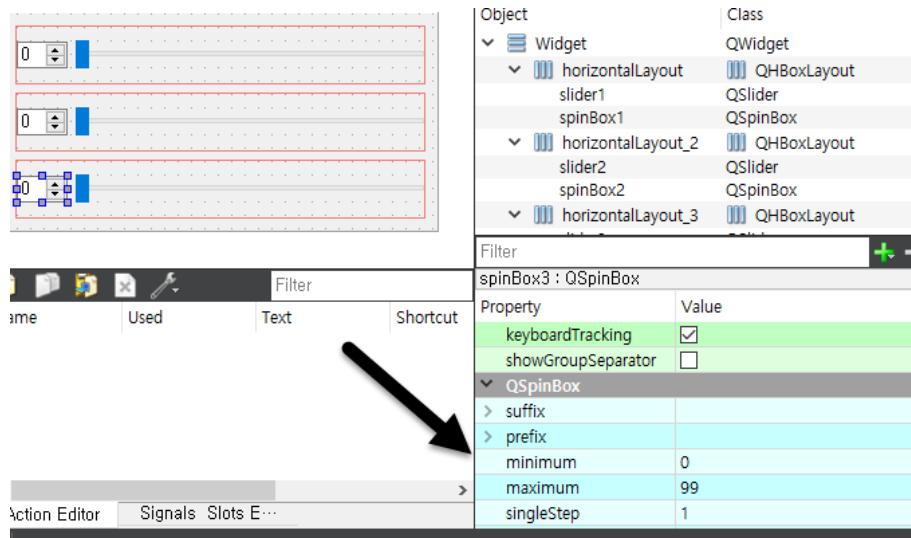
Layout	Class	Object Name
첫 번째 Horizontal Layout	QSpinBox	spinBox1
	QSlider	slider1
첫 번째 Horizontal Layout	QSpinBox	spinBox2
	QSlider	slider2
첫 번째 Horizontal Layout	QSpinBox	spinBox3
	QSlider	slider3

각 위젯을 마우스로 드래그해 배치한 표에서 보는 것과 같이 위젯들의 Object Name 을 변경한다. Object Name 변경은 우측 하단의 항목 중 objectName 항목을 더블클릭하면 변경할 수 있다.



<그림> objectName 변경 화면

아래 그림에서 보는 것과 같이 Object Name 을 변경 후 각 위젯의 minimum 값을 0 으로 변경하고 maximum 값을 99 로 변경한다.



<그림> 위젯 세부 속성 창 화면

GUI 상에 위젯들을 모두 배치 하였다면 어플리케이션을 빌드 후 실행해 보자. 어플리케이션이 구동 되면 마우스로 위젯의 크기를 변경해 보고 자동으로 위젯들의 크기가 변경되는지 확인해 보자.



<그림> 어플리케이션 실행 화면

Designer 툴에서 배치한 각 QSlider 위젯의 눈금을 마우스로 위치를 변경하면 QSlider 의 값이 변경된다. 이 값이 변경되면 변경된 값을 QSpinBox 의 값으로 설정하는 예제이다. 아래 소스코드는 widget.h 헤더 파일이다.

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>

namespace Ui { class Widget; }
```

```

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();
private:
    Ui::Widget *ui; // 윈도우상에 배치한 위젯을 접근하기 위한 오브젝트
private slots:
    void slider1_valueChanged(int value);
    void slider2_valueChanged(int value);
    void slider3_valueChanged(int value);
};

#endif // WIDGET_H

```

widget.h 헤더파일에서 ui 를 선언한 Object 명이 있다. 이 오브젝트 명이 Designer 툴에서 배치한 위젯을 접근할 수 있는 접근자이다. 예를 들어 Designer 툴에서 배치한 slider2 라는 오브젝트를 접근하기 위해서 소스코드에서 ui 오브젝트를 사용하면 된다.

아래 예제 소스코드 하단의 Slot 함수는 Designer에서 배치한 QSlider 의 값이 변경되면 호출되는 Slot 함수이다. 다음은 widget.cpp 소스 코드이다.

```

#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->slider1, SIGNAL(valueChanged(int)),
            this,           SLOT(slider1_valueChanged(int)));
    connect(ui->slider2, SIGNAL(valueChanged(int)),
            this,           SLOT(slider2_valueChanged(int)));
    connect(ui->slider3, SIGNAL(valueChanged(int)),
            this,           SLOT(slider3_valueChanged(int)));
}

Widget::~Widget() {
    delete ui;
}

void Widget::slider1_valueChanged(int value)

```

```

{
    ui->spinBox1->setValue(value);
}

void Widget::slider2_valueChanged(int value)
{
    ui->spinBox2->setValue(value);
}

void Widget::slider3_valueChanged(int value)
{
    ui->spinBox3->setValue(value);
}

```

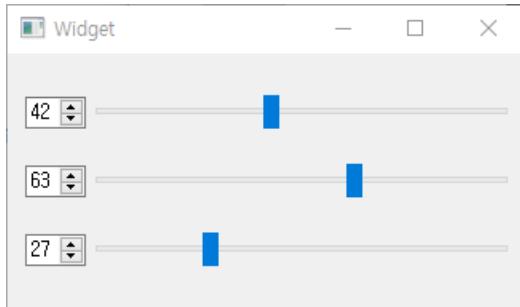
위의 예제에서 한가지 주의해야 할 점은 connect() 함수의 인자를 Old Style 형태로 사용했다는 점이다. 만약 New Style 을 사용하면 “no matching member function for call to ‘connect’” 라는 에러가 발생한다. 이런 에러가 발생하는 이유는 QSpinBox 에서 제공하는 valueChanged() 멤버 함수는 Overloaded 된 멤버 함수로 int 형과 QString 형인 두가지 멤버 함수를 제공하기 때문이다.

```

void valueChanged(int i)
void valueChanged(const QString &text)

```

위의 소스코드에서 보는 것과 같이 Overloaded 된 시그널이 있다면 Old Style 을 사용해야 한다.



<그림> 어플리케이션 실행 화면

좌측의 예제 전체 소스코드는 Ch03 > 05_Designer > 01_Designer 디렉토리 참조하면 된다.

3.6. 다이얼로그

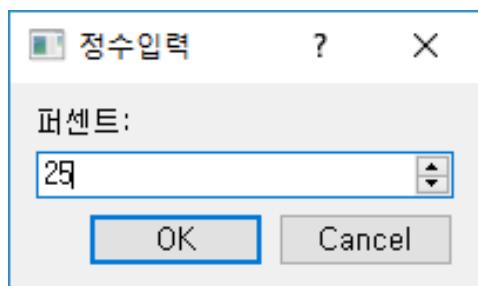
다이얼로그는 어플리케이션이 동작 중에 이벤트가 발생했을 때 사용자에게 메시지를 전달하기 위한 목적으로 사용된다. 그리고 사용자로부터 입력 값을 받거나 여러 개 중 하나를 선택할 수 있는 GUI를 제공한다. 다음 표는 Qt에서 제공하는 다이얼로그 중 자주 사용되는 다이얼로그이다.

종류	설명
QInputDialog	사용자로부터 값을 입력 받을 수 있는 다이얼로그
QColorDialog	특정 컬러를 선택할 수 있는 다이얼로그
QFileDialog	파일 또는 디렉토리를 선택하는 GUI 인터페이스를 제공.
QFontDialog	폰트를 선택하기 위한 다이얼로그
QProgressDialog	퍼센트와 같은 진행사항을 보여주기 위한 다이얼로그
QMessageBox	모달 방식의 다이얼로그

- QInputDialog

QInputDialog 클래스는 사용자로부터 값을 입력 받을 수 있다.

```
bool retVal;
int i = QInputDialog::getInt(this, "정수입력", "퍼센트:",
                             25, 0, 100, 1, &retVal);
if (retVal)
    qDebug("true %d", i);
```



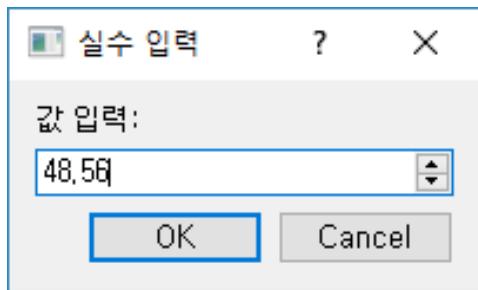
QInputDialog 클래스의 getInt() 멤버 함수는 사용자로부터 정수 값을 입력 받을 수 있는 다이얼로그를 제공한다.

QInputDialog 클래스의 getInt() 멤버 함수는 사용자로부터 정수 값을 입력 받을 수 있는 다이얼로그를 제공한다.

<그림> 정수 값 입력ダイアログ

첫 번째 인자는 부모 클래스, 두 번째 인자는 윈도우의 타이틀 바에 표시할 제목, 세 번째 인자는 입력 값 위젯 항목의 좌측에 표시할 항목 이름이다. 네 번째 인자는 디폴트 설정 값, 다섯 번째와 여섯 번째는 사용자가 입력할 수 있는 값의 범위이고 다음 인자는ダイ얼로그의 스피ن 박스의 증가 값이다. 마지막 인자는ダイ얼로그에서 [OK] 또는 [Cancel] 버튼을 클릭했는지 확인할 수 있는 값을 저장한다.

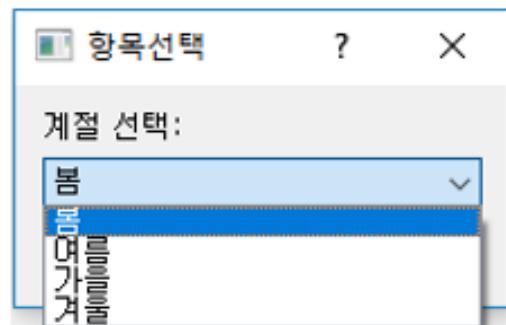
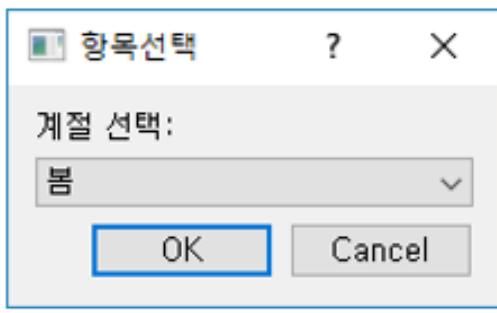
```
bool retValue;  
double dVal = QInputDialog::getDouble(this, "실수 입력", "값 입력:",  
48.56, -100, 100, 2, &retValue);
```



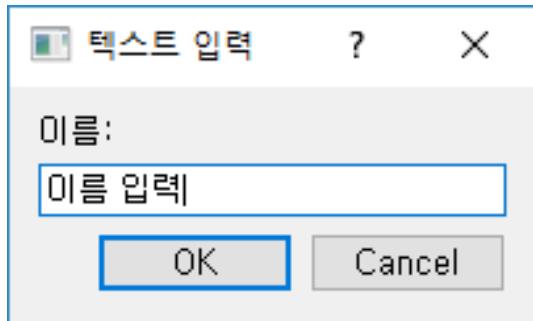
QInputDialog 클래스의 getDouble() 멤버 함수는 실수 값을 입력 받을 수 있다. getItem() 멤버 함수는 문자열(또는 단어) 중에 하나를 선택할 수 있는 기능을 제공한다.

<그림> Double 값 입력ダイアル로그

```
QStringList items;  
items << "봄" << "여름" << "가을" << "겨울";  
bool ok;  
QString item = QInputDialog::getItem(this, "항목선택", "계절 선택:",  
items, 0, false, &ok);
```



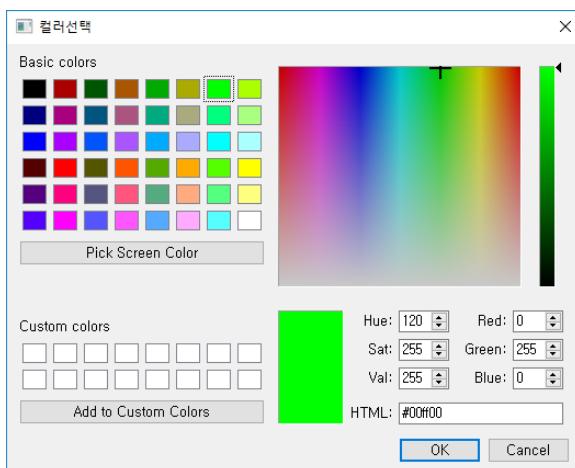
<그림> QInputDialog 클래스의 항목 선택ダイアル로그



<그림> TEXT 입력 다이얼로그

```
bool ok;
QString text = QInputDialog::getText(this, "텍스트 입력", "이름:",
                                     QLineEdit::Normal, "이름 입력", &ok);
```

- QColorDialog



QColorDialog 클래스는 사용자가 색상표를 보고 원하는 색상을 선택하기 위한 기능을 제공한다.

<그림> QColorDialog 실행 화면

```
QColor color;
color = QColorDialog::getColor(Qt::green, this, "컬러선택",
                               QColorDialog::DontUseNativeDialog);
if (color.isValid())
    qDebug() << Q_FUNC_INFO << "유효한 색상.;"
```

- QFileDialog

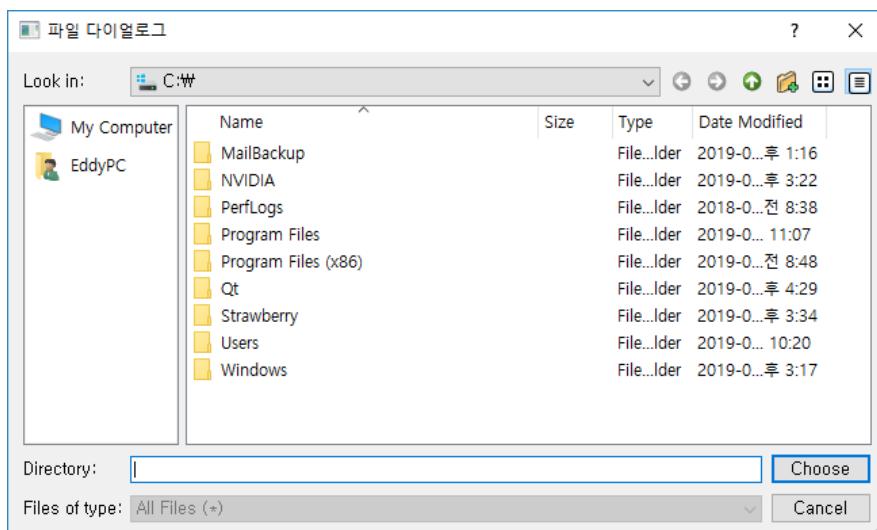
QFileDialog 클래스는 사용자로부터 파일을 선택할 수 있는 다이얼로그를 제공한다. 특정 확장자 또는 특정 파일을 필터링하여 사용자에게 보여줄 수 있다.

QFileDialog 클래스의 `getOpenFileNames()` 멤버 함수는 디렉토리 내에 존재하는 파일을 다중 선택할 수 있는 기능을 제공한다. `getExistingDirectory()` 멤버 함수는 사용자가 디렉토리를 선택할 수 있다. 그리고 `getSaveFileName()` 멤버 함수는 사용자가 저장할 파일을 지정할 수 있다.

```
QFileDialog::Options options;
options = QFileDialog::DontResolveSymlinks | QFileDialog::ShowDirsOnly;
options |= QFileDialog::DontUseNativeDialog;

QString directory = QFileDialog::getExistingDirectory(this,
    "파일 다이얼로그", "C:", options);
```

`getExistingDirectory()` 멤버 함수는 사용자로부터 디렉토리를 선택할 수 있는 기능을 제공한다. 첫 번째 인자는 부모 클래스, 두 번째 인자는 다이얼로그의 타이틀바 제목, 세 번째 인자는 지정한 디렉토리가 디폴트로 지정하기 위해 사용하는 인자이다. 마지막 인자는 파일 다이얼로그의 상수 값을 이용해 필터링하기 위한 옵션이다.



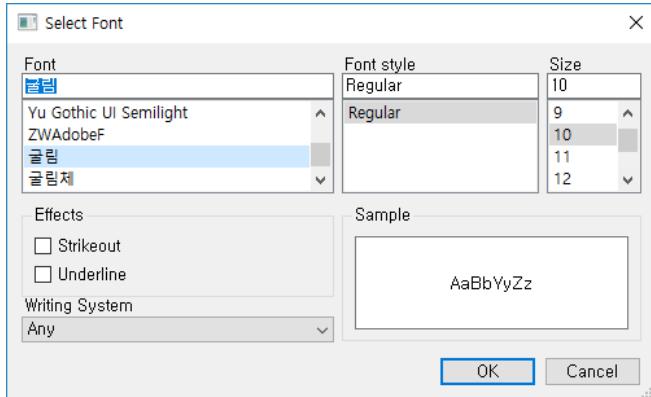
<그림> QFileDialog 실행 화면

<표> QFileDialog::Options

상수	설명
<code>QFileDialog::ShowDirsOnly</code>	디렉토리만 표시
<code>QFileDialog::DontResolveSymlinks</code>	심볼릭 링크를 표시하지 않기 위해 사용
<code>QFileDialog::DontConfirmOverwrite</code>	덮어쓰기 할 때 경고 메시지를 표시하지 않기

QFileDialog::DontUseNativeDialog	시스템 기본 파일ダイアル로그를 사용하지 않기 위해 사용
QFileDialog::ReadOnly	읽기 모드로 파일ダイアル로그를 사용
QFileDialog::HideNameFilterDetails	필터를 이용해 파일을 감추기 위해 사용

- QFontDialog



<그림> QFontDialog 실행 화면

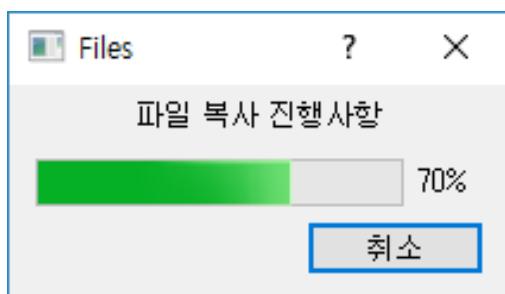
QFontDialog는 사용자로부터 폰트를 선택할 수 있는ダイアル로그를 제공한다.

첫 번째 인자는 사용자가ダイアル로그 하단에 위치해 있는 [OK] 버튼과 [CANCEL] 버튼 중 어떤 버튼을 클릭했는지 확인하기 위한 변수를 지정 한다.

두 번째 인자는 폰트ダイアル로그 상에서 디폴트 선택으로 지정할 수 있는 폰트를 지정 한다.

```
bool ok;
QFont font;
font = QFontDialog::getFont(&ok, QFont( Courier 10 Pitch ), this);
```

- QProgressDialog



QProgressDialog 클래스는 사용자에게 현재 진행 사항을 보여주기 위한 목적으로 사용한다. 예를 들어 대용량 파일 복사와 같이 시간이 다소 걸리는 사항에 대해서ダイアル로그창에 진행사항을 보여 줄 수 있다.

<그림> QProgressDialog

다음은 QProgressDialog 클래스를 이용해 사용자에게 진행사항을 보여주기 위한 다이

얼로그 예제 소스코드이다. QWidget 클래스를 Base 클래스로 프로젝트를 생성한 후 widget.h 헤더 파일을 다음과 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QProgressDialog>
#include <QTimer>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;
    QProgressDialog *pd;
    QTimer *timer;
    int steps;

public slots:
    void perform();
    void cancel();
};

#endif // WIDGET_H
```

위의 예제에서 QTimer 는 1초에 한번씩 타이머 이벤트를 호출해 QProgressDialog 의 진행사항 값을 1%씩 증가하기 위해 사용하였다. 다음 소스코드는 widget.cpp 소스코드이다.

```
#include "widget.h"
#include "ui_widget.h"
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
```

```

steps = 0;
pd = new QProgressDialog("파일 복사 진행사항", "취소", 0, 100);
connect(pd, SIGNAL(canceled()), this, SLOT(cancel()));

timer = new QTimer(this);
connect(timer, SIGNAL(timeout()), this, SLOT(perform()));
timer->start(1000);

}

Widget::~Widget()
{
    delete ui;
}

void Widget::perform()
{
    pd->setValue(steps);
    steps++;

    if (steps > pd->maximum())
        timer->stop();
}

void Widget::cancel()
{
    timer->stop();
}

```

- QMessageBox

QMessageBox 클래스는 모달(Modal) 방식의ダイアル로그를 제공한다. 사용자에게 정보를 전달할 수 있으며 QMessageBox 클래스가ダイアル로그 창에서 사용 가능하도록 제공하는 버튼들은 다음 표에서 보는 것과 같이 다양한 버튼을 사용할 수 있도록 제공한다.

<표> QMessageBox에서 사용 가능한 버튼

상수	값	설명
QMessageBox::Ok	0x00000400	OK 버튼
QMessageBox::Open	0x00002000	파일 열기 버튼
QMessageBox::Save	0x00000800	저장 버튼

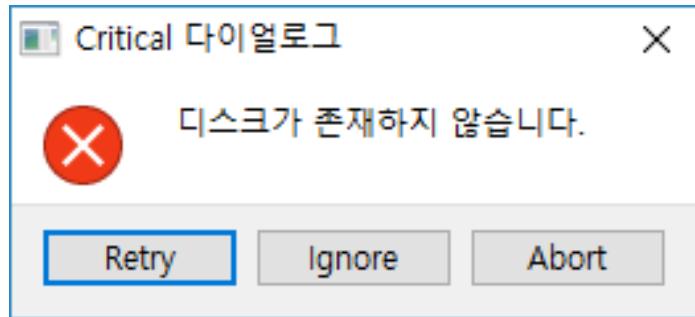
QMessageBox::Cancel	0x00400000	취소 버튼
QMessageBox::Close	0x00200000	닫기 버튼
QMessageBox::Discard	0x00800000	저장하지 않고 버리기 버튼
QMessageBox::Apply	0x02000000	APPLY 버튼
QMessageBox::Reset	0x04000000	RESET 버튼
QMessageBox::RestoreDefaults	0x08000000	다시 저장하기 버튼
QMessageBox::Help	0x01000000	도움말 버튼
QMessageBox::SaveAll	0x00001000	모두 저장하기 버튼
QMessageBox::Yes	0x00004000	YES 버튼
QMessageBox::YesToAll	0x00008000	모두 YES 적용하기 버튼
QMessageBox::No	0x00010000	NO 버튼
QMessageBox::NoToAll	0x00020000	모두 NO 적용하기 버튼
QMessageBox::Abort	0x00040000	중지 버튼
QMessageBox::Retry	0x00080000	재시도 버튼
QMessageBox::Ignore	0x00100000	Ignore(무시) 버튼
QMessageBox::NoButton	0x00000000	An invalid button

다음 예제는 QMessageBox 클래스를 이용해 [Abort] 버튼, [Retry] 버튼, [Ignore] 버튼을 사용하기 위한 예제이다.

```
QMessageBox::StandardButton reply;

reply = QMessageBox::critical(this,
    "Critical 다이얼로그",
    "디스크가 존재하지 않습니다.",
    QMessageBox::Abort |
    QMessageBox::Retry |
    QMessageBox::Ignore);

if (reply == QMessageBox::Abort)
    ...
else if (reply == QMessageBox::Retry)
    ...
    ...
```

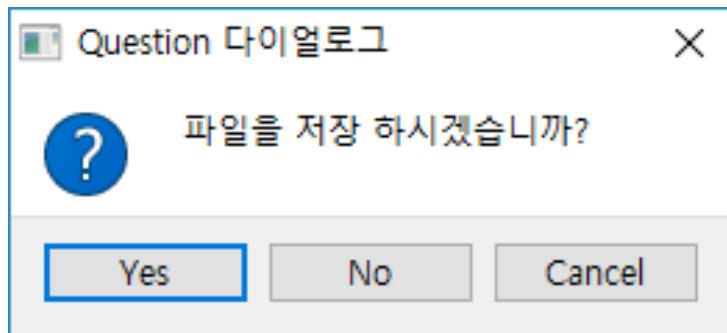


<그림> Critical 대이얼로그

QMessageBox 클래스는 information(), question(), warning() 멤버 함수를 제공한다. 다음은 question() 멤버 함수 사용 예제이다.

```
QMessageBox::StandardButton reply;
reply = QMessageBox::question(this, "Question 대이얼로그",
    "파일을 저장 하시겠습니까?",
    QMessageBox::Yes | QMessageBox::No | QMessageBox::Cancel);

if (reply == QMessageBox::Abort)
    ...
else if (reply == QMessageBox::Retry)
    ...
```



<그림> Question 대이얼로그

- 사용자 정의 대이얼로그 구현 예제

이번 예제는 QDialog 클래스를 상속받아 사용자가 원하는 스타일의 대이얼로그를 구현해 보도록 하자. 다음 예제소스코드는 dialog.h 헤더 파일이다.

```

#ifndef DIALOG_H
#define DIALOG_H
#include <QDialog>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QVBoxLayout>
#include <QHBoxLayout>

class Dialog : public QDialog
{
    Q_OBJECT
public:
    Dialog(QWidget *parent = 0);
    ~Dialog();

private:
    QLabel      *lbl;
    QLineEdit   *edit;
    QPushButton *okbtn;
    QPushButton *cancelbtn;

private slots:
    void slot_okbtn();
    void slot_cancelbtn();
};

#endif // DIALOG_H

```

dialog.h 헤더 파일은 QDialog 클래스를 상속받은 클래스를 구현하기 위한 헤더파일이다. 이 다이얼로그는 이름을 입력 받을 수 있으며 버튼 두개를 배치 하였다. 다음 예제 소스코드는 dialog.cpp 전체 소스코드이다.

```

#include "dialog.h"

Dialog::Dialog(QWidget *parent) : QDialog(parent)
{
    setWindowTitle("Custom Dialog");

    lbl = new QLabel("이름");
    edit = new QLineEdit("");
    okbtn = new QPushButton("확인");
    cancelbtn = new QPushButton("취소");

```

```

QHBoxLayout *hlay1 = new QHBoxLayout();
hlay1->addWidget(lbl);
hlay1->addWidget(edit);
QHBoxLayout *hlay2 = new QHBoxLayout();
hlay2->addWidget(okbtn);
hlay2->addWidget(cancelbtn);

QVBoxLayout *vlay = new QVBoxLayout();
vlay->addLayout(hlay1);
vlay->addLayout(hlay2);
setLayout(vlay);
}

void Dialog::slot_okbtn()
{
    emit accepted();
}

void Dialog::slot_cancelbtn()
{
    emit rejected();
}

Dialog::~Dialog()
{
}

```

slot_okbtn() Slot 함수는 다이얼로그의 [확인] 버튼을 클릭 시 호출된다. slot_cancelbtn() Slot 함수는 [취소] 버튼을 클릭하면 호출된다. 다음 예제 소스코드 main.cpp 소스코드이다.

```

#include <QApplication>
#include <QDebug>
#include "dialog.h"

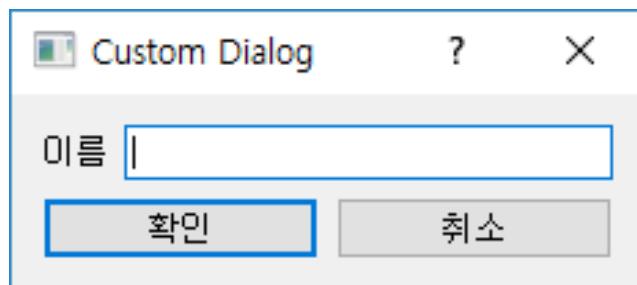
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    Dialog dlg;
    int retVal = dlg.exec();

```

```
if(retval == QDialog::Accepted)
{
    qDebug() << Q_FUNC_INFO << "QDialog::Accepted";
}
else if(retval == QDialog::Rejected)
{
    qDebug() << Q_FUNC_INFO << "QDialog::Rejected";
}

return a.exec();
}
```



<그림> Custom Dialog 예제 실행 화면

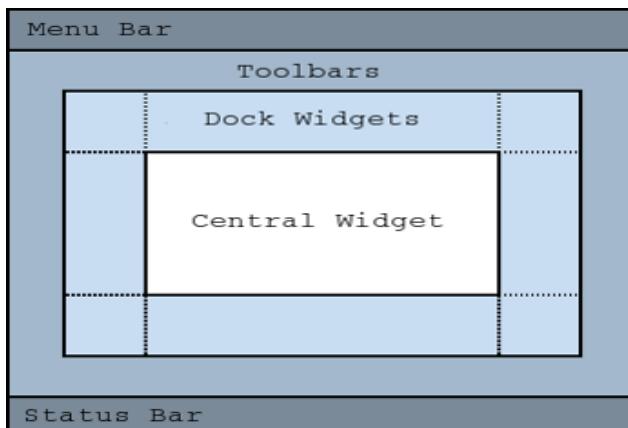
위의 예제에서 보는 것과 같이 Dialog 클래스의 dlg 오브젝트는 사용자가 [확인] 버튼 클릭 시 int 값 1을 리턴 한다. [취소] 버튼 클릭 시 0을 리턴 한다. 예제의 전체 소스는 Ch03 > 06_Dialog > 01_CustomDialog 디렉토리를 참조하면 된다.

3.7. MDI 기반 원도우 GUI 구현

지금까지 Qt를 이용한 GUI 기반의 위젯을 원도우 상에 배치하는 예제를 다루어 보았다. 지금까지 다룬 방식은 SDI (Single Document Interface) 방식이다. SDI 방식은 한 개의 원도우 화면만 존재하는 방식으로 단순한 GUI 구성에 적합하다.

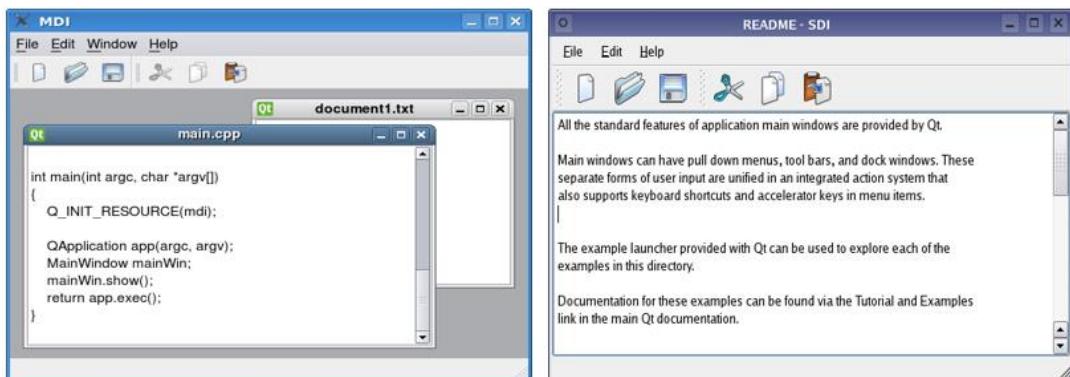
하지만 기능이 복잡하고 사용자에게 많은 기능을 제공해야 하는 경우 GUI 구현 시 SDI 방식 보다 MDI (Multi Document Interface) 방식을 이용해 GUI를 구현하는 것이 사용자에게 직관적인 GUI를 제공할 수 있을 것이다.

MDI 방식은 다양한 영역으로 구성할 수 있다. 예를 들어 Menu Bar, Toolbars, Status Bar, Dock Widget, Central Widget 등으로 위젯들을 특정 영역에 배치할 수 있다.



MDI 는 SDI 방식과 차이점으로 SDI 는 하나의 원도우 영역을 가질 수 있지만 MDI 방식은 QMdiArea 클래스를 이용해 메인 원도우 영역 안에 여러 개의 원도우를 가질 수 있다. 따라서 복잡한 원도우 GUI를 구현해야 한다면 MDI 방식을 이용하는 것을 권장한다.

<그림> MDI 방식의 원도우 배치



<그림> MDI 방식과 SDI 방식의 원도우

- QMdiArea 클래스를 이용한 MDI 기반의 GUI 예제

이번 예제에서는 QMdiArea 클래스를 이용해 MDI 기반의 GUI를 구현해 보도록 하자. 이 예제는 Qt를 설치하면 제공되는 예제이다. Examples 의 MDI Example 예제에 전체 소스코드 참조하면 된다. 이 예제 소스코드는 2개의 클래스로 구현되어 있다.

MainWindow 클래스는 QMainWindow 클래스를 상속받아 구현 메인 윈도우 GUI 이다. 이 클래스에는 Menu Bar, Tool Bar, Central Widget 영역 등이 구현되어 있다.

MDIMainWindow 클래스는 QTextEdit 위젯 클래스를 상속받는 위젯 클래스이다. 이 클래스는 MainWindow 중앙 배치할 다중 에디트 위젯으로 사용한다. 즉 울트라 에디터, 노트패드 등과 같이 여러 개의 텍스트 파일을 하나의 윈도우 영역 안에서 편집할 수 있는 기능을 제공하기 위해 구현된 클래스이다. 다음 예제 소스코드는 MainWindow 클래스의 헤더 소스코드이다.

```
#include <QMainWindow>
#include "MDIMainwindow.h"

class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
private slots:
    void newFile();
    void open();
};

...
```

newFile() Slot 함수는 메뉴 바에서 [New] 메뉴를 클릭했을 때 호출된다. open() 함수는 [Open] 메뉴를 클릭하면 호출된다. 다음 예제는 mainwindow.cpp 소스코드이다.

```
#include "mainwindow.h"
#include <QMenu>
#include <QAction>
#include <QMenuBar>
#include <QToolBar>
#include <QDockWidget>
#include <QListWidget>
#include <QStatusBar>
#include <QDebug>
```

```

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent)
{
    QMenu *fileMenu;
    QAction *newAct;
    QAction *openAct;

    newAct = new QAction(QIcon(":/images/new.png"), tr("&New"), this);
    newAct->setShortcuts(QKeySequence::New);
    newAct->setStatusTip(tr("Create a new file"));
    connect(newAct, SIGNAL(triggered()), this, SLOT(newFile()));

    openAct = new QAction(QIcon(":/images/open.png"), tr("&Open"), this);
    openAct->setShortcuts(QKeySequence::Open);
    openAct->setStatusTip(tr("Open an existing file"));
    connect(openAct, SIGNAL(triggered()), this, SLOT(open()));

    fileMenu = menuBar()->addMenu(tr("&File"));
    fileMenu->addAction(newAct);
    fileMenu->addAction(openAct);

    QToolBar *fileToolBar;
    fileToolBar = addToolBar(tr("File"));
    fileToolBar->addAction(newAct);
    fileToolBar->addAction(openAct);

    QDockWidget *dock = new QDockWidget(tr("Target"), this);
    dock->setAllowedAreas(Qt::LeftDockWidgetArea | Qt::RightDockWidgetArea);

    QListWidget *customerList = new QListWidget(dock);
    customerList->addItems(QStringList()
        << "One" << "Two" << "Three" << "Four" << "Five");

    dock->setWidget(customerList);
    addDockWidget(Qt::RightDockWidgetArea, dock);
    setCentralWidget(new MDIMainWindow());
    statusBar()->showMessage(tr("Ready"));
}

MainWindow::~MainWindow()
{
}

```

```

void MainWindow::newFile()
{
    qDebug() << Q_FUNC_INFO;
}

void MainWindow::open()
{
    qDebug() << Q_FUNC_INFO;
}

```

MainWindow 클래스의 생성자에서는 MDI 윈도우 GUI상에서 메뉴와 툴바에 배치할 메뉴 항목을 정의한다. 그리고 각 메뉴의 클릭 시 Signal 이벤트 발생시 Slot 함수와 연결한다.

QDockWidget 은 MDI 윈도우 좌측에 위치하고 사용자가 GUI상에서 새로운 창으로 분리할 수 있는 GUI를 제공한다. MDIMainWindow 클래스는 MDI 윈도우에서 Child 윈도우로 사용한다. 즉 GUI 내에 여러 개의 Child 윈도우를 제공하는 것과 같은 기능을 제공한다. 다음은 MDIMainWindow 클래스의 헤더 소스코드이다.

```

#include < QMainWindow>
#include <QObject>

class MDIMainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MDIMainWindow(QWidget *parent = nullptr);
};

```

이 클래스 생성자에서는 QMdiArea 클래스와 QMdiSubWindow 클래스를 이용해 MDIMainWindow 하위에 서브 윈도우로 등록할 윈도우를 생성한다. 다음 예제 소스코드는 MDIMainwindow.cpp 소스코드이다.

```

#include "MDIMainwindow.h"
#include <QMdiArea>
#include <QMdiSubWindow>
#include <QPushButton>

MDIMainWindow::MDIMainWindow(QWidget *parent) : QMainWindow(parent)
{

```

```

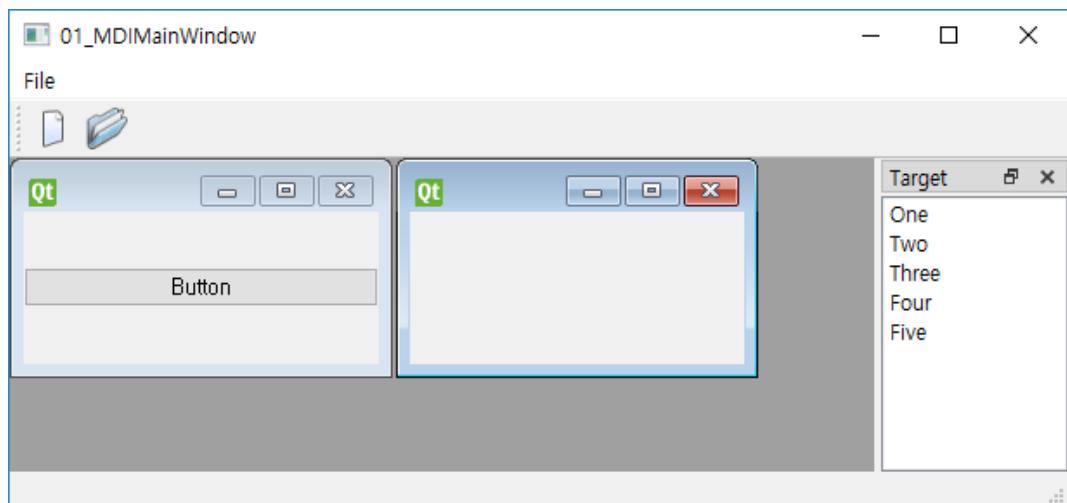
setWindowTitle(QString::fromUtf8("My MDI"));

QMdiArea* area = new QMdiArea();
area->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);

QMdiSubWindow* subWindow1 = new QMdiSubWindow();
subWindow1->resize(300, 200);
QPushButton *btn = new QPushButton(QString("Button"));
subWindow1->setWidget(btn);

QMdiSubWindow* subWindow2 = new QMdiSubWindow();
subWindow2->resize(300, 200);
area->addSubWindow(subWindow1);
area->addSubWindow(subWindow2);
setCentralWidget(area);
}

```



<그림> MDI 예제 실행 화면

예제 소스는 Ch03 > 07_MDI > 01_MDIMainWindow 디렉토리를 참조하면 된다.

3.8. 파일 처리와 데이터스트림

Qt는 파일을 처리하기 위해 QFile 클래스를 제공한다. 그리고 대량의 데이터를 효율적으로 READ/WRITE 하기 위한 목적으로 QTextStream 과 QDataStream 클래스를 제공한다. 예를 들어 파일 크기가 작은 파일을 READ / WRITE 하는데 QFile에서 제공하는 멤버 함수만 이용해도 문제가 없지만 파일 용량이 큰 파일을 처리하는데 QTextStream 클래스와 QDataStream 클래스를 이용하면 대용량 파일을 핸들링 하는데 효율적으로 파일을 READ/WRITE 할 수 있다.

다음 예제는 데이터 스트림을 사용하지 않고 QFile 클래스만 사용해 파일로부터 데이터를 READ 하는 예제이다.

- ✓ QFile 클래스를 이용해 파일로부터 데이터 읽어 오기

```
QFile file("in.txt");
if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
    return;

while (!file.atEnd()) {
    QByteArray line = file.readLine();
    ...
}
```

QFile 클래스의 open() 멤버 함수는 파일을 Open 하는 기능을 제공한다. Open 시 파일을 읽어 들일 때 파일을 읽기 모드로 OPEN 할지 READ모드, WRITE모드 또는 READ/WRITE를 함께 사용할 수 있는 모드로 OPEN할지 open() 함수 인자로 지정할 수 있다.

QIODevice::ReadOnly 옵션을 사용하면 파일을 READ 만 가능하다. QIODevice::Text 옵션은 파일을 TEXT 모드로 오픈하기 위한 목적으로 사용할 수 있다.

<표> 파일 모드

Constant	값	설명
QIODevice::NotOpen	0x0000	파일을 열지 않을 때 사용
QIODevice::ReadOnly	0x0001	읽기 전용 모드로 사용
QIODevice::WriteOnly	0x0002	쓰기 전용 모드로 사용

QIODevice::ReadWrite	Read Write	읽기 / 쓰기 모드를 함께 사용
QIODevice::Append	0x0004	파일 끝에 추가하기 위해 사용
QIODevice::Truncate	0x0008	만약 이전에 Open된 파일이 있다면 새로운 연결을 위해 이전에 사용된 파일을 삭제.
QIODevice::Text	0x0010	TEXT모드로 읽을 때 마지막에 '\n'을 사용. MS윈도우인에서는 '\r\n'을 마지막에 사용.
QIODevice::Unbuffered	0x0020	버퍼를 사용하지 않고 디바이스를 바로 사용

위의 예제소스코드에서 while 문에서 조건으로 사용한 atEnd() 멤버 함수는 파일의 마지막일 때 까지 반복한다. readLine() 멤버 함수는 '\n' 문자를 만날 때 까지 읽어 들이는 기능을 제공한다.

QFile 클래스는 데이터를 STREAM 을 처리하는 QTextStream 과 QDataStream 클래스를 이용해 파일을 READ/WRITE를 할 수 있다. 다음 예제는 QFile 과 QTextStream 클래스를 사용해 파일로부터 데이터를 읽어 들이기 위한 방법이다.

```
#include <QCoreApplication>
#include <QFile>
#include <QString>
#include <QDebug>
#include <QTextStream>

void write(QString filename)
{
    QFile file(filename);
    if(!file.open(QFile::WriteOnly | QFile::Text)) {
        qDebug() << "Open fail.";
        return;
    }

    QTextStream out(&file);
    out << "Write Test";

    file.flush();
    file.close();
}

void read(QString filename)
{
```

```

QFile file(filename);
if(!file.open(QFile::ReadOnly | QFile::Text)) {
    qDebug() << " Open fail.";
    return;
}

QTextStream in(&file);
qDebug() << in.readAll();

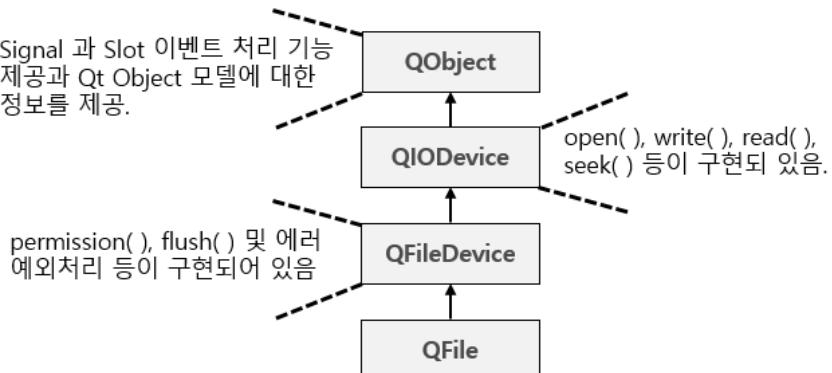
file.close();
}

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QString filename = "C:/Qt/MyFile.txt";
    write(filename);
    read(filename);
    return a.exec();
}

```

QFile 클래스는 파일 처리를 위해 Open, Exists, Link, Remove, Rename 등과 같은 기능을 제공한다. QFile 클래스는 QFileDevice 클래스로부터 상속 받아 구현되었다. QFileDevice 클래스는 파일의 Permission, Flush, Error 처리와 같은 예외 처리 기능 등이 구현 되어 있다.

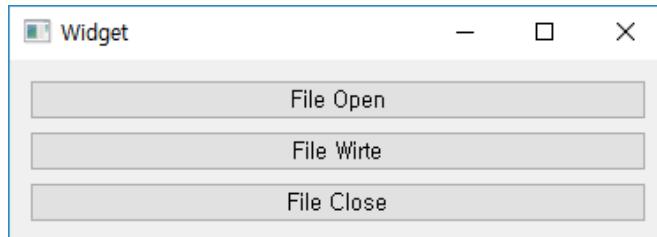
QIODevice 클래스는 QIODevice 클래스를 상속받았다. QIODevice 클래스는 파일을 Open, Write, Read, Seek 등의 기능이 구현되어 있다. QIODevice 는 QFile 외에도 여러 클래스로부터 상속받아 사용된다. 예를 들어 멀티미디어에서 음원 데이터 즉, MP3파일로부터 미디어를 읽어 오기 용도로 사용되기도 한다.



<그림> QFile 클래스의 상속 관계

여기서 QFile 클래스의 상속 관계를 설명한 이유는 만약 새로운 디바이스로부터 데이터를 READ/WRITE 해야하는 API를 구현해야 한다면 QFile 과 같은 상속 관계로 구현한다면 이미 구현된 Open, Write, Read, Seek 와 같은 기능을 사용할 수 있을 것이다.

- QFile 클래스를 이용한 예제



<그림> 예제 실행 화면

이번에 다룰 예제는 파일 Open, Write 그리고 Close 기능을 구현한 예제이다. 위의 그림에서 보는 것과 같이 [File Open] 버튼을 클릭하면 파일 다이얼로그에서 파일 중 하나를 선택한다. 그러면 사용자가 선택한 파일을 Open 하고 QTextStream 클래스를 이용해 파일로부터 데이터를 Read한다. read한 데이터를 라인 단위로 읽어와 qDebug()를 이용해 Console 상에 출력한다.

[File Write] 버튼을 클릭하면 QFile 클래스의 seek() 멤버 함수를 이용해 파일의 마지막으로 이동한 후 "₩n Hello World₩n" 문자열을 파일에 Write 한다. [File Close] 버튼을 클릭하면 파일을 Close 한다. 다음 예제는 widget.h 소스코드이다.

```
#include <QWidget>
#include <QFile>

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();
}
```

```

private:
    Ui::Widget *ui;
    QFile *mFile;

private slots:
    void slotPbtOpenPress();
    void slotPbtWritePress();
    void slotPbtClosePress();
    void slotAboutToClose();
};

```

이 클래스 생성자에서는 mFile 오브젝트를 초기화 한다. 그리고 파일이 Close 시그널이 발생하면 이 Signal을 slotAboutToClose() Slot 함수를 호출한다. 다음 예제 소스코드는 widget.cpp 소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"
#include <QDebug>
#include <QTextStream>
#include <QFileDialog>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pbtOpen, SIGNAL(pressed()), this, SLOT(slotPbtOpenPress()));
    connect(ui->pbtWrite, SIGNAL(pressed()), this, SLOT(slotPbtWritePress()));
    connect(ui->pbtClose, SIGNAL(pressed()), this, SLOT(slotPbtClosePress()));

    mFile = new QFile();
    connect(mFile, SIGNAL(aboutToClose()), this, SLOT(slotAboutToClose()));
}

Widget::~Widget()
{
    delete ui;
}

void Widget::slotPbtOpenPress()
{
    QString fileName;
    fileName = QFileDialog::getOpenFileName(this, "Open File",
                                           QDir::currentPath(), "Files (*.txt)");
}

```

```

mFile->setFileName(fileName);
if(!mFile->open(QIODevice::ReadWrite | QIODevice::Text)) {
    qDebug() << "File open fail.";
    return;
}

QTextStream in(mFile);
while(in.atEnd())
    qDebug() << in.readLine();
}

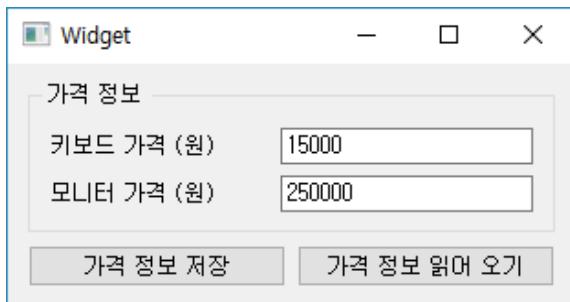
void Widget::slotPbtWritePress()
{
    QTextStream in(mFile);
    mFile->seek(mFile->size());
    in << "파일 마지막 메세지 추가.\n";
}

void Widget::slotPbtClosePress()
{
    if(mFile->isOpen())
        mFile->close();
}

void Widget::slotAboutToClose()
{
    qDebug() << Q_FUNC_INFO;
}

```

- QFile 클래스와 QDataStream 을 이용한 예제



<그림> 예제 실행 화면

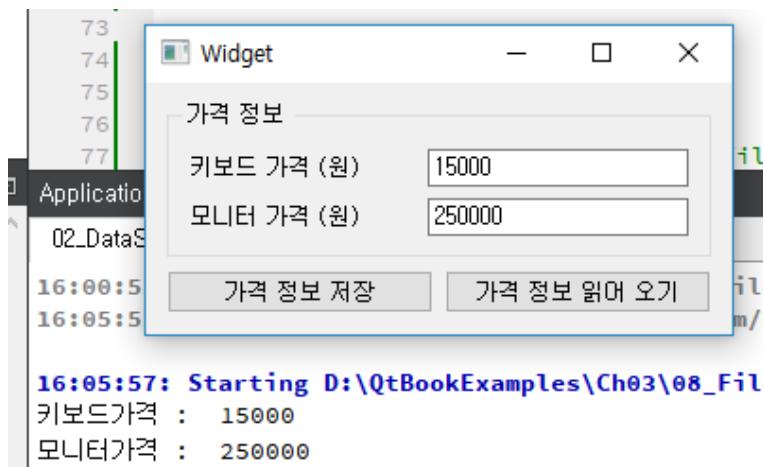
이번 예제는 QDataStream 클래스를 이용한 예제이다. 좌측 그림의 예제 실행 화면에서 보는 것과 같이 키보드 가격과 모니터 가격을 [가격 정보 저장] 버튼을 클릭하면 값을 파일에 저장한다.

파일 저장 시 이전 예제와 같이 TEXT 기반으로 저장하게 되면 예제 실행에서 보는 것과 같이 “15000”을 문자로 저장하면 5 Bytes 가 필요하고 모니터 가격 “250000”을 저장하려면 6 Bytes 가 필요하다. 따라서 문자열로 저장하게 되면 총 11 Bytes 가 된다.

하지만 키보드 가격과 모니터 가격의 문자열을 숫자로 변환 한다면 각각의 값을 저장하는데 총 8 Bytes 이면 충분하다. 왜냐하면 int형 32bit(4Byte) 변수 2개를 사용하면 저장이 가능하기 때문이다.

이런 방법과 같이 이기종 간의 데이터 통신 시 정해진 프로토콜을 이진 데이터 형식으로 데이터 송수신을 하는 경우가 많다. 이 경우에 장점은 필요한 만큼의 저장 공간만 사용하기 때문에 불필요한 데이터 낭비를 최소화 할 수 있다.

예를 들어 UART, I2S 등 임베디드 환경에서 프로세스간 통신 또는 이 기종 간의 원격의 디바이스 간의 데이터 통신 시 많이 사용된다. 그리고 GUI상에서 [가격 정보 읽어 오기] 버튼을 클릭하면 파일로부터 데이터를 QDataStream 을 이용해 READ 한다. 그런 다음 읽어온 데이터를 qDebug() 함수를 이용해 Console 에 출력한다.



<그림> Console 창에 출력 화면

예제 전체 소스코드는 Ch03 > 08_File_Stream > 02_DataStream 디렉토리를 참조하면 된다. 다음 예제 소스코드는 widget.h 헤더 파일 소스코드이다.

```
#include <QWidget>
#include <QFile>

class Widget : public QWidget
{
    Q_OBJECT
public:
```

```

explicit Widget(QWidget *parent = nullptr);
~Widget();

private:
    Ui::Widget *ui;
    QFile *mFile;

private slots:
    void slotPbtFileSave();
    void slotPbtFileRead();
};

```

[가격 정보 저장] 버튼을 클릭하면 slotPbtFileSave() Slot함수가 호출된다. [가격 정보 읽어 오기] 버튼 클릭하면 slotPbtFileRead() Slot 함수가 호출된다. 다음은 widget.cpp 소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"
#include <QDebug>
#include <QDataStream>
#include <QMMessageBox>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pbtSave, SIGNAL(pressed()), this, SLOT(slotPbtFileSave()));
    connect(ui->pbtFileRead, SIGNAL(pressed()), this, SLOT(slotPbtFileRead()));
    mFile = new QFile();
}

Widget::~Widget() {
    delete ui;
}

void Widget::slotPbtFileSave()
{
    QString fileName;
    fileName = QString("c:/price.data");

    mFile->setFileName(fileName);
    if(!mFile->open(QIODevice::WriteOnly | QIODevice::Truncate)) {
        qDebug() << "File open fail.";
    }
}

```

```

        return;
    } else {
        qint32 keyboardPrice = ui->leKeyboard->text().toInt();
        qint32 monitoryPrice = ui->leMonitor->text().toInt();

        QDataStream out(mFile);
        out << keyboardPrice;
        out << monitoryPrice;

        mFile->close();
    }
}

void Widget::slotPbtFileRead()
{
    if(!mFile->open(QIODevice::ReadOnly)) {
        qDebug() << "File open fail.";
        return;
    } else {
        qint32 keyboardPrice;
        qint32 monitoryPrice;

        QDataStream in(mFile);
        in >> keyboardPrice;
        in >> monitoryPrice;
        mFile->close();

        qDebug() << "키보드가격 : " << keyboardPrice;
        qDebug() << "모니터가격 : " << monitoryPrice;
    }
}

```

3.9. Qt Property

Qt에서 제공하는 Property System은 C++에서 제공하는 Property System과 비슷한 기능을 제공한다. Property는 객체에서 값을 객체에 설정하고 가져오는 경우에 사용된다. 예를 들어 어떤 특정 클래스에서 다음 예제 소스코드와 같이 값을 설정하거나 가져오는 경우를 살펴보자.

```
class Person : public QObject
{
    Q_OBJECT
public:
    explicit Person(QObject *parent = nullptr);

    QString getName() const {
        return m_name;
    }

    void setName(const QString &n) {
        m_name = n;
    }
private:
    QString m_name;
};
```

위의 예제 소스코드에서 보는 것과 같이 Person이라는 클래스는 Public 접근자로 선언한 getName()과 setName() 멤버 함수를 제공한다. getName() 멤버 함수를 m_name 변수 값을 리턴 한다. setName() 멤버 함수는 m_name 값을 설정하는 함수이다. Person이라는 클래스의 오브젝트를 선언하고 다음과 같이 사용해보자.

```
Person goodman;

goodman.setName("Kim Dae Jin");
qDebug() << "Goodman name : " << goodman.getName();
```

Person 클래스 오브젝트를 선언하고 setName() 멤버 함수를 이용해 변수 값을 설정하였다. 그리고 setName() 멤버 함수로 선언한 값을 getName() 멤버 함수로 값을 얻어와 Console 창에 출력하는 예제 소스코드이다. 예제소스코드 살펴본 방법을 Qt에서 제공하는 Property System으로 접근해 보도록 하자. 다음 소스와 같이 Person 클래스

에 Q_PROPERTY 매크로를 추가해 보자.

```
...
class Person : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString name READ getName WRITE setName)

public:
    explicit Person(QObject *parent = nullptr);
    QString getName() const {
        return m_name;
    }

    void setName(const QString &n) {
        m_name = n;
    }

private:
    QString m_name;
};

...
```

위의 예제 소스코드에서 보는것과 같이 getName() 과 setName() 멤버 함수를 Q_PROPERTY 매크로에 등록하였다. 그리고 다음과 같이 Q_PROPERTY 매크로에 등록한 멤버 함수들을 다음과 같이 사용할 수 있다.

```
Person goodman;

goodman.setProperty("name", "Kim Dae Jin");
qDebug() << "Goodman name : " << goodman.getName();

QVariant myName = goodman.property("name");
qDebug() << "My name is " << myName.toString();

// [ 결과 ]
// Goodman name : "Kim Dae Jin"
// My name is "Kim Dae Jin"
```

getName() 와 setName() 멤버 함수를 접근해 name 변수 값을 얻어오거나 설정할 수 있지만 Q_PROPERTY 매크로를 사용하면 setProperty() 와 property() 를 사용해 값을 얻어오거나 설정할 수 있다.

클래스에서 Q_PROPERTY 매크로 그다지 큰 유용성이 없어 보인다. 하지만 QML을 사용하는 경우 매우 중요하다. QML로 UI를 개발하고 기능을 C++로 개발한다고 했을 때 QML과 C++ 간의 데이터를 주고 받아야 하는 경우가 빈번하게 발생한다. 이 때 QML에서 C++의 변수의 값을 얻어오거나 변경 하하고 할 때 반드시 Q_PROPERTY 매크로를 사용해야 한다. Q_PROPERTY 매크로는 예제소스코드에 다루었던 방식 이외에도 다음과 같이 다양한 옵션을 사용할 수 있다.

```
Q_PROPERTY(type name  
          (READ getFunction [WRITE setFunction] |  
           MEMBER memberName [(READ getFunction |  
                                WRITE setFunction)] )  
          [RESET resetFunction]  
          [NOTIFY notifySignal]  
          [REVISION int]  
          [DESIGNABLE bool]  
          [SCRIPTABLE bool]  
          [STORED bool]  
          [USER bool]  
          [CONSTANT]  
          [FINAL])
```

MEMBER 키워드는 READ 키워드로부터 읽어 들일 값을 지정할 수 있다. 예를 들어 위의 예제 소스코드에서 private 접근 제한자에서 선언한 m_name 변수를 다음과 같이 지정할 수 있다.

```
class Person : public QObject  
{  
    Q_OBJECT  
    Q_PROPERTY(QString name MEMBER m_name READ getName WRITE setName)  
    ...
```

Q_PROPERTY 매크로가 제공하는 키워드 중 NOTIFY 키워드는 시그널 지정할 수 있다. 다음 예제에서 보는 것과 같이 시그널을 사용할 수 있다.

```
class Person : public QObject  
{  
    Q_OBJECT  
    Q_PROPERTY(QString name MEMBER m_name READ getName WRITE setName  
               NOTIFY nameChanged)  
public:  
    explicit Person(QObject *parent = nullptr);
```

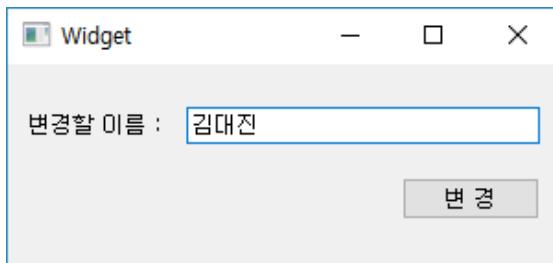
```

QString getName() const {
    return m_name;
}
void setName(const QString &n) {
    m_name = n;
    emit nameChanged(n);
}
private:
    QString m_name;

signals:
    void nameChanged(const QString &n);
...

```

- Q_PROPERTY를 이용한 예제



<그림> 예제 실행 화면

이번에 다룰 예제는 아래 그림에서 보는 것과 같이 [변경] 버튼을 클릭하면 Person 클래스의 setName() 멤버 함수를 QObject 클래스에서 제공하는 setProperty() 멤버 함수를 이용해 값을 변경 한다.

그리고 setName() 멤버 함수를 호출되면 Q_PROPERTY 매크로에서 NOTIFY 키워드로 명시한 시그널 이벤트를 발생할 것이다. 다음 예제는 widget.h 의 헤더 소스코드이다.

```

...
class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

public slots:
    void buttonPressed();
    void nameChanged(const QString &n);

private:

```

```
Ui::Widget *ui;
Person *goodman;
};

...
```

[변경] 버튼 클릭하면 buttonPressed() Slot 함수가 호출된다. 그리고 nameChanged() Slot 함수는 Person 클래스의 nameChanged() 시그널이 발생하면 호출된다. 다음 예제 소스코드는 widget.cpp 예제 소스코드이다.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pushButton, &QPushButton::pressed, this, &Widget::buttonPressed);

    goodman = new Person();
    connect(goodman, &Person::nameChanged, this, &Widget::nameChanged);
}

void Widget::buttonPressed()
{
    QString name = ui->leName->text();
    goodman->setProperty("name", name);
}

void Widget::nameChanged(const QString &n)
{
    qDebug() << Q_FUNC_INFO << "Name Changed : " << n;
    QVariant myName = goodman->property("name");
    qDebug() << "My name is " << myName.toString();
}

Widget::~Widget()
{
    delete ui;
}
...
```

nameChanged() Slot 함수가 호출되면 Person 클래스에서 제공하는 getName() 멤버 함수를 QObject 클래스에서 제공하는 property() 함수를 이용해 값을 얻어온다. 다음 예제 소스코드는 Person 클래스의 헤더파일이다.

```

...
class Person : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString name MEMBER m_name READ getName
               WRITE setName NOTIFY nameChanged)
public:
    explicit Person(QObject *parent = nullptr);
    QString getName() const {
        return m_name;
    }
    void setName(const QString &n) {
        m_name = n;
        emit nameChanged(n);
    }
private:
    QString m_name;
signals:
    void nameChanged(const QString &n);
};

...

```

Person 클래스에서 Q_PROPERTY 매크로를 이용해 getName() 와 setName() 멤버 함수를 접근할 수 있다. 그리고 nameChanged() 시그널을 Q_PROPERTY 매크로의 NOTIFY 키워드로 지정하였다.

따라서 setName() 멤버 함수에서 emit 을 이용해 시그널 이벤트가 발생하면 Widget 클래스의 연결된 Slot 함수가 호출 된다. 예제의 전체 소스코드는 Ch03 > 09_Property > 01_Simple_Property 디렉토리를 참조하면 된다.

3.10. QTimer

QTimer 클래스는 원하는 함수를 지정한 시간을 기준으로 반복해 호출할 수 있다. 다음 예제 소스코드는 QTimer를 이용한 예제이다.

```
QTimer *timer = new QTimer(this);
connect(timer, SIGNAL(timeout()), this, SLOT(update()));

timer->start(1000);
```

위의 예제 소스코드에서 보는 것과 같이 QTimer 클래스의 오브젝트를 선언한다. 그리고 지정한 시간을 반복해 호출하기 위해서 connect() 함수를 이용해 Signal과 Slot 함수를 연결해야 한다.

connect() 함수에서 첫 번째 인자는 QTimer 오브젝트를 명시한다. 두 번째는 Signal을 명시한다. timeout() Signal은 지정한 시간이 경과되면 4번째 인자에 지정한 update() Slot 함수가 호출된다.

마지막 라인의 QTimer 클래스의 start() 멤버 함수는 첫 번째 인자에 지정한 시간만큼 경과되면 connect() 함수에서 명시한 Slot 함수가 반복해 계속 호출된다. 첫 번째 인자의 단위 밀리 세컨드(milliseconds) 이다.

QTimer 클래스는 stop() 멤버 함수를 이용해 QTimer를 정지할 수 있는 기능을 제공한다. 만약 QTimer 클래스를 반복해서 실행하지 않고 단 한번만 호출되도록 하기 위해 singleShot() 멤버 함수를 제공한다. singleShot() 멤버 함수는 다음 예제 소스코드에서 보는 것과 같이 사용할 수 있다.

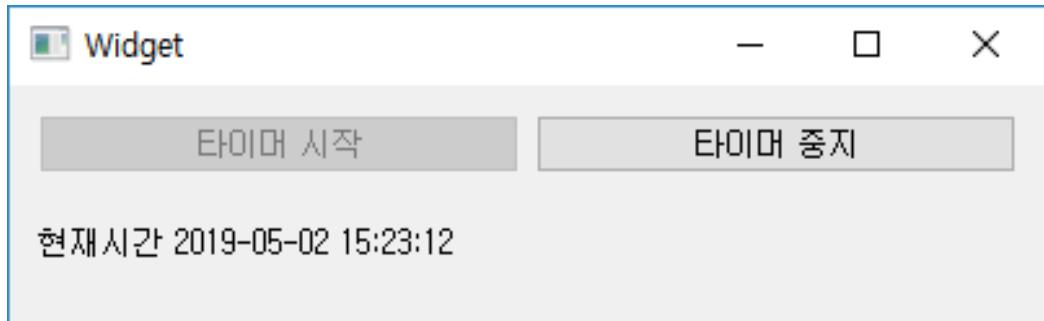
```
QTimer::singleShot(200, this, SLOT(updateCaption()));
```

singleShot() 멤버 함수의 첫 번째 인자는 경과 되는 시간이며 단위는 밀리 세컨드이다. 그리고 세 번째 인자는 첫 번째 인자의 시간이 경과된 후 호출될 Slot 함수를 지정하면 된다.

- QTimer 클래스를 이용한 예제

다음 예제는 1초 간격으로 Slot 함수가 호출된다. 다음 그림에서 보는 것과 같이 [타이머 시작] 버튼을 클릭하면 타이머가 시작된다. [타이머 중지] 버튼을 클릭하면 타이머가

중지된다.



<그림> QTimer 예제 실행 화면

QTimer 예제 실행 화면에서 하단의 현재 시간은 QTimer에서 지정한 1000 밀리 세컨드(1초)를 주기로 현재 시간을 시스템으로부터 얻어와 QLabel 위젯에 출력하는 예제이다. 다음 예제 소스코드는 widget.h 소스코드이다.

```
...
class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;
    QTimer *m_timer;

public slots:
    void startPressed();
    void stopPressed();
    void elapsedTime();
};

...
...
```

startPressed() 는 [타이머 시작] 버튼을 클릭하면 호출되는 Slot 함수이다. stopPressed() 는 [타이머 중지] 버튼을 클릭하면 호출되는 Slot 함수이다. elapsedTime() 는 QTimer에서 지정한 시간이 경과되면 호출된다. 다음 예제는 widget.cpp 소스코드이다.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
```

```

ui->setupUi(this);
connect(ui->pbtStart, &QPushButton::pressed, this, &Widget::startPressed);
connect(ui->pbtStop, &QPushButton::pressed, this, &Widget::stopPressed);

ui->pbtStart->setEnabled(true);
ui->pbtStop->setEnabled(false);

m_timer = new QTimer();
connect(m_timer, &QTimer::timeout, this, &Widget::elapsedTime);
}

Widget::~Widget()
{
    delete ui;
}

void Widget::startPressed()
{
    ui->pbtStart->setEnabled(false);
    ui->pbtStop->setEnabled(true);
    m_timer->start(1000);
}

void Widget::stopPressed()
{
    ui->pbtStart->setEnabled(true);
    ui->pbtStop->setEnabled(false);
    m_timer->stop();
}

void Widget::elapsedTime()
{
    QDateTime curr = QDateTime::currentDateTime();
    QString timeStr = curr.toString("현재시간 yyyy-MM-dd hh:mm:ss");
    ui->leCurrentTime->setText(timeStr);
}
...

```

이 예제의 전체 소스코드는 Ch03 > 10_QTimer > 01_Timer 디렉토리를 참조하면 된다.

3.11. QThread

Qt에서는 Thread를 지원하기 위해서 QThread 클래스를 제공한다. QThread 클래스를 이용해 Thread 를 사용하는 경우 특정 상황에서 동기화가 필요한 경우가 있다. 동기화는 Thread 환경에서 다중 사용자가 참조하는 변수를 특정 사용자가 변수 값을 변경할 때 다른 사용자에게 잘못된 값을 참조할 수 있다.

따라서 Thread 로 동작되는 특정 함수의 소스코드의 시작 시점부터 종료 시점까지 다른 사용자가 변수를 참조하거나 변경하지 못하게 함으로써 변수가 변경되기 이전의 잘못된 값을 참조하지 못하도록 하기 위한 기능을 제공한다.

이러한 기능을 동기화 기능이라고 하는데 Qt에서는 동기화를 지원하기 위해 QMutex 클래스를 제공한다. 다음 예제 소스코드는 QThread 클래스를 상속받아 구현한 클래스의 일부이다.

```
#include <QThread>
#include <QMutex>
#include <QDateTime>

class MyThread : public QThread
{
    Q_OBJECT
public:
    void run() override {
        while(!m_threadStop)
        {
            m_mutex.lock();
            ...
            m_mutex.unlock();
            ...
            sleep(1);
        }
    }
public:
    MyThread(int n);

private:
    bool m_threadStop;
    QMutex m_mutex;
```

```
};  
...
```

위의 예제 소스코드에서 보는 것과 같이 QThread 를 이용해 MyThread 클래스를 구현한 예제이다. run() 함수는 QThread 에서 상속받은 Virtual 멤버 함수이며 Thread 에서 동작 하고자 하는 기능을 이 함수에서 구현하면 된다.

run() 함수는 내부에서 구현한 함수이기 때문에 클래스 외부에서 QThread 에서 제공하는 start() 함수를 호출 하면 자동으로 run() 함수가 호출된다.

```
MyThread *thread = new MyThread(this);  
thread->start();
```

간혹 run() 함수를 Public 으로 선언하고 외부에서 run() 함수를 외부에서 호출하는 경우가 있다. 이 경우 run() 함수가 Thread 에서와 같이 동작하지 않는다. 즉 Thread가 아닌 일반 함수와 같이 동작하므로 run() 함수를 바로 호출하는 경우가 없도록 주의해야 한다. run() 함수에서 사용한 QMutex 클래스는 동기화를 위한 기능이다.

위의 예제 소스코드에서 보는 것과 같이 QMutex 클래스에서 제공하는 lock() 멤버 함수는 동기화를 시작되도록 선언하고 unlock() 멤버 함수를 통해 동기화의 마지막 구간임을 선언하면 이 구간 내에 선언된 변수는 외부 클래스가 unlock() 멤버 함수 호출될 때 까지 접근할 수 없다.

예를 들어 네트워크 어플리케이션에서 현재 접속자가 10명이라고 가정해 보자. 그리고 현재 접속자가 10명인 경우 값을 users 라는 int 공용 변수에 저장했다고 가정해 보자. 만약 새로운 접속 자로 인해 11명으로 증가 하는 경우, 이 때 users 변수 값을 1증가하는 시점에 동일한 시간 때 현재 접속자를 확인하는 프로세스가 있다면 잘못된 현재 접속자 정보를 다른 접속자에게 잘못된 정보를 줄 수 있다.

따라서 이럴 때 users 변수 값을 증가하는 소스코드 구간을 lock() 과 unlock() 구간에는 users 변수 값을 1 증가 하도록 동기화를 사용하면 users 변수는 변경하거나 참조하지 못하도록 대기 열에서 기다리도록 처리한다.

동기화가 필요한 경우 QMutex 클래스를 사용하면 유용하지만 너무 자주 사용하게 되면 프로그램이 Blocking 되어 멈춤 현상이 발생할 수 있으므로 필요한 경우에만 사용하는 것을 권장한다.

- Reentrancy 와 Thread-Safety 를 만족하는 Thread 예제

Reentrancy 는 두 개 이상의 Thread가 동작되고 있을 때 Thread 가 수행되는 순서에

상관없이 하나의 Thread 가 수행되고 난 후, 다음 Thread 가 수행된 것처럼 수행되는 것을 의미한다.

Thread-Safety 는 두 개 이상의 쓰레드가 동작하는 상황에서 Static 또는 Heap 메모리 영역과 같이 공유되는 메모리 영역을 접근할 때 안정성을 보장하기 위해 Mutex 와 같은 메커니즘을 사용하는 것을 의미한다. 다음 소스코드 두 개의 쓰레드를 생성하고 Reentrancy 와 Thread-Safety 를 고려하지 않은 쓰레드를 구현한 예제이다.

```
static QMutex mutex;
static QWaitCondition incNumber;
static int numUsed;

class Producer : public QThread
{
    Q_OBJECT
public:
    void run() override {
        for(int i = 0 ; i < 10 ; i++) {
            sleep(1);
            ++numUsed;
            qDebug("[Producer] numUsed : %d", numUsed);
        }
    }
};

class Consumer : public QThread
{
    Q_OBJECT
public:
    void run() override {
        for(int i = 0 ; i < 10 ; i++) {
            qDebug("[Consumer] numUsed : %d", numUsed);
        }
    }
};
```

위의 예제 소스코드에서 보는 것과 같이 Producer 와 Consumer 클래스를 구현하고 main.cpp 에서 다음과 같이 작성해 보도록 하자.

```

#include <QCoreApplication>
#include "mythread.h"

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    Producer producer;
    Consumer consumer;
    producer.start();
    consumer.start();

    return a.exec();
}

```

위의 예제소스코드에서 2개의 구현한 Thread 클래스 오브젝트를 선언하고 QThread에서 제공하는 start() 멤버 함수를 호출하면 다음 그림에서 보는 것과 같이 Thread를 시작한다.

```

C:\Qt\Qt5.12.2\Tools\QtCreator\bin\qtcreator_process_stub.exe
[Consumer] numUsed : 0
[Producer] numUsed : 1
[Producer] numUsed : 2
[Producer] numUsed : 3
[Producer] numUsed : 4
[Producer] numUsed : 5
[Producer] numUsed : 6
[Producer] numUsed : 7
[Producer] numUsed : 8
[Producer] numUsed : 9
[Producer] numUsed : 10

```

<그림> 예제 실행 화면

예제 실행 화면에서 보는 것과 같이 Consumer 클래스의 run() 함수가 먼저 모두 수행된 후 Producer 클래스의 run() 함수가 수행될 것이다.

다음 예제는 Reentrancy 와 Thread-Safety 를 만족하는 Thread를 구현한 예제이다. 이전 예제에서 구현한 Consumer 클래스와 Producer 클래스를 다음과 같이 수정해 보도

록 하자.

```
#ifndef MYTHREAD_H
#define MYTHREAD_H
#include <QWaitCondition>
#include <QMutex>
#include <QThread>

static QMutex mutex;
static QWaitCondition incNumber;
static int numUsed;

class Producer : public QThread
{
    Q_OBJECT
public:
    void run() override {
        for(int i = 0 ; i < 10 ; i++) {
            sleep(1);
            ++numUsed;
            qDebug("[Producer] numUsed : %d", numUsed);

            mutex.lock();
            incNumber.wakeAll();
            mutex.unlock();
        }
    }

public:
    Producer() {}
};

class Consumer : public QThread
{
    Q_OBJECT
public:
    void run() override {
        for(int i = 0 ; i < 10 ; i++) {
            mutex.lock();
            incNumber.wait(&mutex);
            mutex.unlock();

            qDebug("[Consumer] numUsed : %d", numUsed);
        }
    }
}
```

```
public:  
    Consumer() { }  
};  
#endif // MYTHREAD_H
```

이전의 예제 소스코드에 작성했던 Producer 와 Consumer 클래스에서는 두 Thread 간의 실행과 관계 없이 Thread 가 동작한다. 하지만 수정한 Producer 와 Consumer 클래스는 Reentrancy 와 Thread-Safety 를 만족하는 Thread 클래스를 구현한 예제이다.

Producer 와 Consumer Thread 는 numUsed 값이 변경될 때에 동작한다. 즉 Consumer 는 QWaitCondition 클래스의 wait() 멤버 함수에 의해 Waiting 되어 있는 상태이다. Producer 클래스가 numUsed 값을 1 증가시키고 QWaitCondition 클래스의 wakeAll() 멤버 함수를 이용해 Consumer 클래스를 Wakeup 한다.

이러한 방식으로 두 개 이상의 Thread 에서 Reentrancy 와 Thread-Safety 를 만족하는 Thread 를 쉽게 구현할 수 있다.

```
[Producer] numUsed : 1  
[Consumer] numUsed : 1  
[Producer] numUsed : 2  
[Consumer] numUsed : 2  
[Producer] numUsed : 3  
[Consumer] numUsed : 3  
[Producer] numUsed : 4  
[Consumer] numUsed : 4  
[Producer] numUsed : 5  
[Consumer] numUsed : 5  
[Producer] numUsed : 6  
[Consumer] numUsed : 6  
[Producer] numUsed : 7  
[Consumer] numUsed : 7  
[Producer] numUsed : 8  
[Consumer] numUsed : 8  
[Producer] numUsed : 9  
[Consumer] numUsed : 9  
[Producer] numUsed : 10  
[Consumer] numUsed : 10
```

<그림> 수정한 예제 실행 화면

예제 소스는 Ch03 > 11_Thread > 02_WaitCondition 디렉토리를 참조하면 된다.

- QtConcurrent 클래스를 이용한 Thread 구현

Qt에서는 QThread 를 사용해 Thread 를 구현하는 방법보다 간단하게 Multi Thread를 구현할 수 있는 QtConcurrent 클래스를 제공한다. QtConcurrent 클래스는 쓰레드 클래스를 작성하지 않고 특정 함수를 Thread 와 같이 동작 시킬 수 있다. 다음 예제 소스 코드는 QtConcurrent 를 이용한 예제 소스코드 이다.

```
#include <QThread>
#include <QtConcurrent/QtConcurrent>
#include <QtConcurrent/QtConcurrentRun>

void hello(QString name)
{
    qDebug() << "Hello" << name << "from" << QThread::currentThread();
    for(int i = 0 ; i < 10 ; i++)
    {
        QThread::sleep(1);
        qDebug("[%s] i = %d", name.toLocal8Bit().data(), i);
    }
}

void world(QString name)
{
    qDebug() << "World" << name << "from" << QThread::currentThread();
    for(int i = 0 ; i < 3 ; i++)
    {
        QThread::sleep(1);
        qDebug("[%s] i = %d", name.toLocal8Bit().data(), i);
    }
}

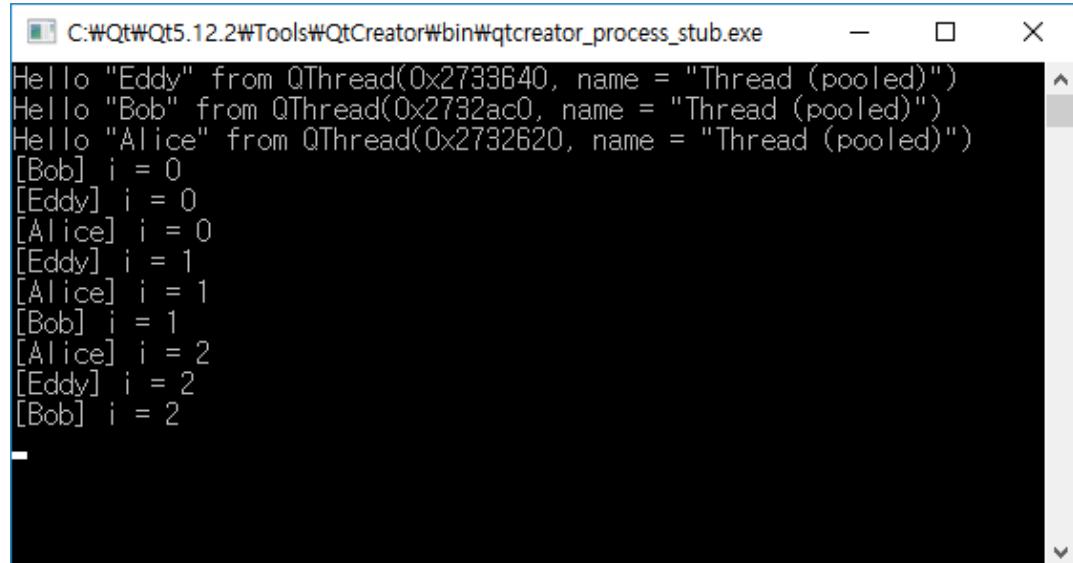
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    QFuture<void> f1 = QtConcurrent::run(hello, QString("Alice"));
    QFuture<void> f2 = QtConcurrent::run(hello, QString("Bob"));
    QFuture<void> f3 = QtConcurrent::run(hello, QString("Eddy"));

    f1.waitForFinished();
    f2.waitForFinished();
    f3.waitForFinished();
```

```
    return a.exec();
}
```

위의 예제 소스코드에서 보는 것과 같이 QtConcurrent 클래스의 run() 멤버 함수의 첫 번째 인자에 함수 명을 지정한다. 그러면 Thread 와 같이 함수를 동작시킬 수 있다.



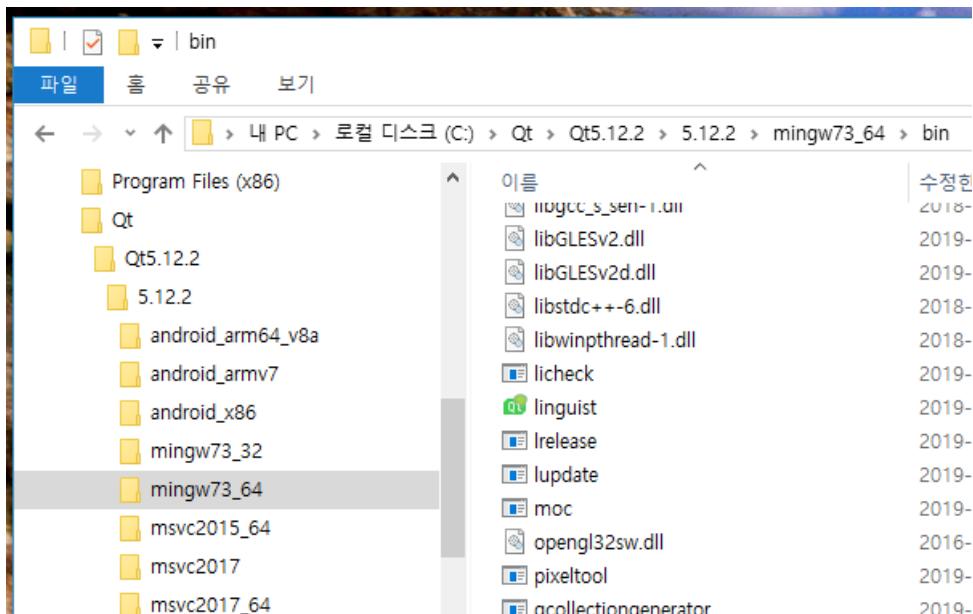
```
C:\Qt\Qt5.12.2\Tools\QtCreator\bin\qtcreator_process_stub.exe
Hello "Eddy" from QThread(0x2733640, name = "Thread (pooled)")
Hello "Bob" from QThread(0x2732ac0, name = "Thread (pooled)")
Hello "Alice" from QThread(0x2732620, name = "Thread (pooled)")
[Bob] i = 0
[Eddy] i = 0
[Alice] i = 0
[Eddy] i = 1
[Alice] i = 1
[Bob] i = 1
[Alice] i = 2
[Eddy] i = 2
[Bob] i = 2
```

<그림> QtConcurrent 예제 실행 화면

예제 전체 소스는 Ch03 > 11_Thread > 03_WaitCondition 디렉토리를 참조하면 된다.

3.12. Qt Linguist Tool 을 이용한 다국어 처리

Qt Linguist 툴은 다국어를 지원하기 위해 제공하는 툴이며 Qt 설치 시 함께 설치 된다. Qt Linguist 는 Qt 설치 시 각 컴파일러로 컴파일 된 Qt Linguist를 제공한다. 예를 들어 MinGW 64Bit 버전을 사용해 구현하였다면 Qt Linguist 툴도 MinGW 64 Bit 버전을 사용해야 한다.



<그림> 컴파일러 별 Qt Linguist 툴 위치

다음 예제 소스코드에서 보는 것과 같이 tr() 함수를 사용한 예를 볼 수 있다. QWidget 클래스의 생성자에서 사용한 tr() 멤버 함수는 다국어 처리를 위한 QObject 클래스의 멤버 함수이다. tr() 함수의 첫 번째 인자에 "Button" 을 입력하였다.

```
Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    ui->pushButton->setText(tr("Button"));
    ...
}
```

tr() 멤버 함수에 등록된 단어가 Qt Linguist 툴로 등록한 다국어 파일 내에 단어가 매칭 되는 단어가 있다면 단어로 자동으로 바꾸어주는 역할을 한다.

그렇기 때문에 다른 언어로 변경하고자 하는 경우 tr() 함수를 사용해야 한다. tr() 함수

를 이용해 다국어를 지원하는 예제를 다루어 보도록 하자. 위의 예제 소스코드에서 보는 것과 같이 Widget 클래스를 Base 클래스로 하는 프로젝트를 생성하고 버튼을 배치한다. 그리고 위의 소스코드에서 보는 것과 같이 QPushButton 클래스의 setText() 멤버 함수에서 보는 것과 같이 tr() 함수를 사용한다.

위의 예제 소스코드에서 보는 것과 같이 소스코드를 추가했다면 프로젝트 파일에 다음과 같이 다국어 지원을 위한 파일을 추가한다. 프로젝트 파일에 다국어를 추가하기 위해서 TRANSLATIONS라는 키워드를 사용하며

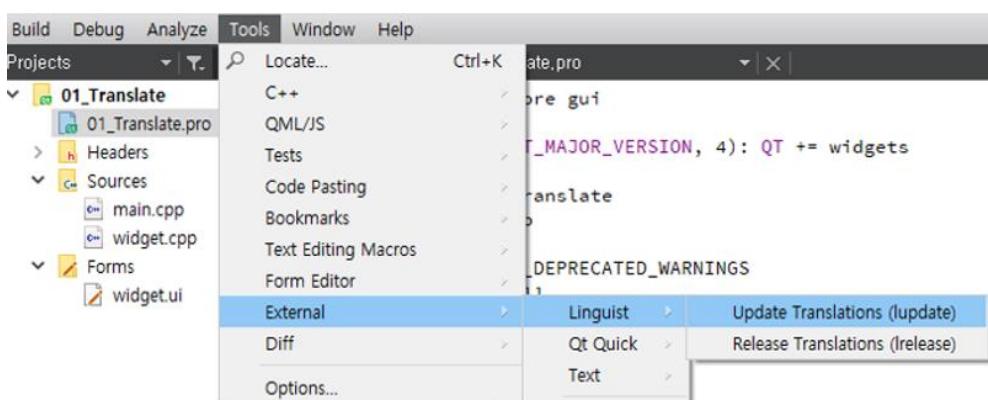
다음 예제 소스코드에서 보는 것과 같이 다국어 파일을 추가한다. 다국어 파일의 파일명은 원하는 파일명을 사용해도 된다. 하지만 파일의 확장자는 ".ts"를 사용해야 한다.

```
QT      += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = 01_Translate
TEMPLATE = app
DEFINES += QT_DEPRECATED_WARNINGS
CONFIG += c++11

TRANSLATIONS = widget.ts
...
```

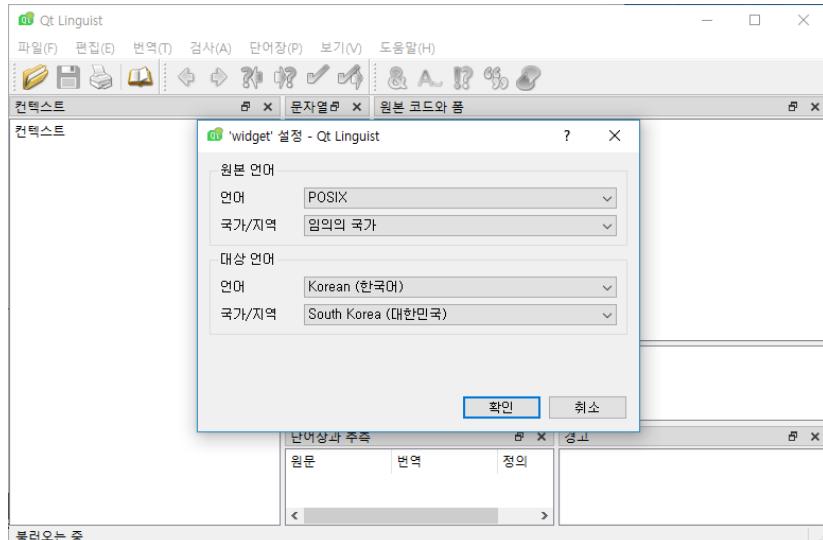
위의 프로젝트 파일 예제에서 보는 것과 같이 TRANSLATIONS 키워드에 원하는 다국어 파일명을 입력한다. 그리고 lupdate 툴을 이용해 프로젝트 파일에 명시한 widget.ts 파일을 생성해야 한다. 다음 그림에서 보는 것과 같이 Qt Creator 툴의 파일 메뉴에서 lupdate 툴을 실행한다. lupdate 툴을 실행하기 위해서 [Tools] > [External] > [Linguist] > [Update Translations (lupdate)] 메뉴를 실행한다.



<그림> lupdate 툴 실행 화면

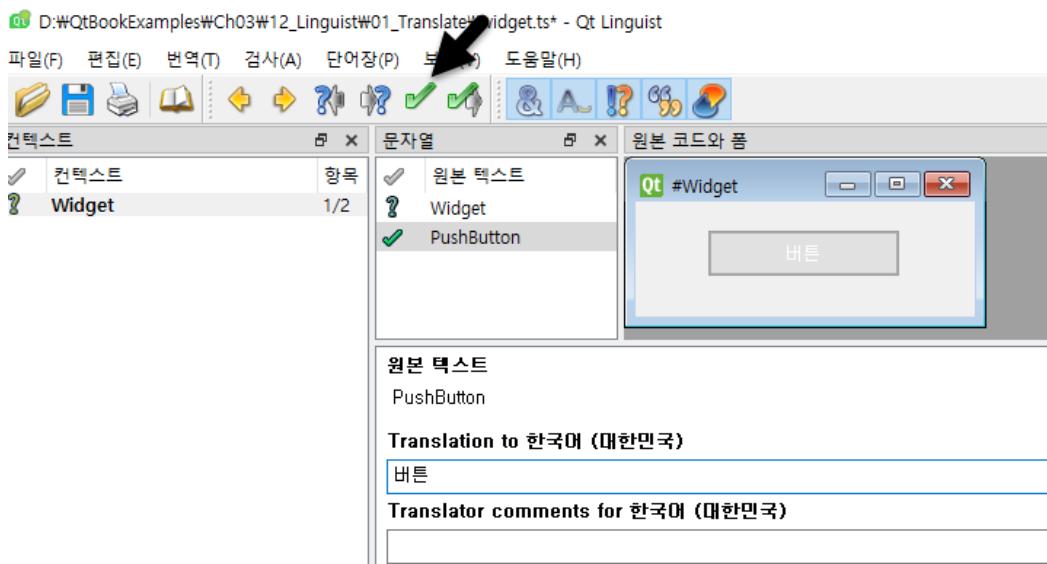
위의 그림에서 보는 것과 같이 lupdate 툴을 실행하면 프로젝트 소스코드가 위치한 디렉토리에 widget.ts 파일이 생성된다.

생성한 widget.ts 파일을 Qt Linguist 툴을 이용해 파일을 Open 한다. 파일의 오픈하면 다음과 같은 다이얼로그 창을 확인할 수 있다. 다음 그림에서 보는 것과 같이 다이얼로그 창 하단의 [확인] 버튼을 클릭한다.



<그림> Qt Linguist Tool 화면

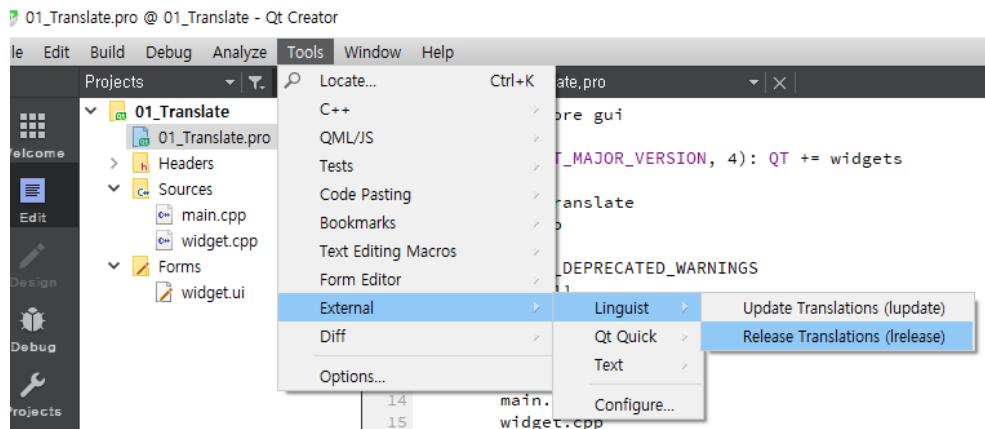
[확인] 버튼을 클릭하면 다음 그림에서 보는 것과 같이 프로젝트에서 생성한 Widget 을 다음 그림에서 보는 것과 같이 확인할 수 있다.



<그림> Qt Linguist Tool 화면

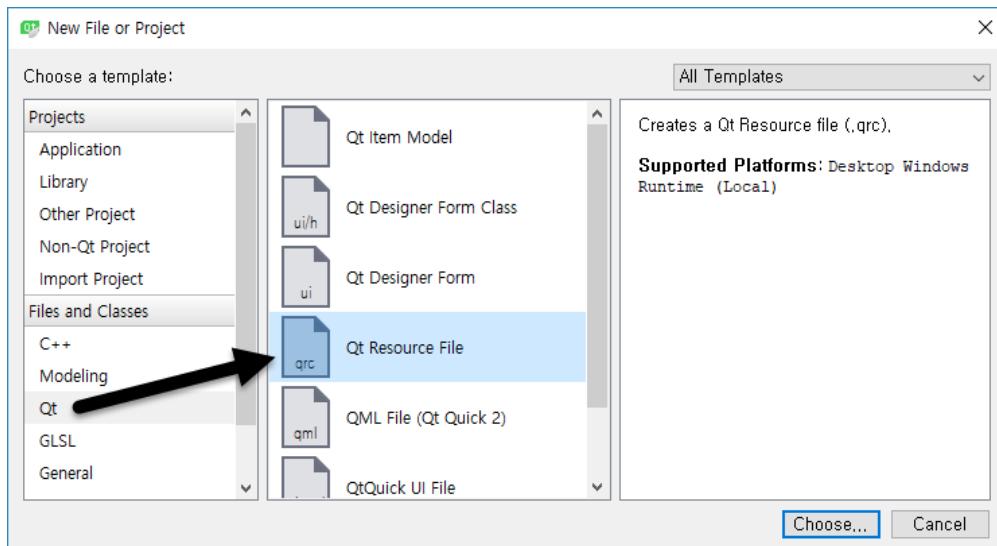
문자열 화면에서 “PushButton” 을 선택하고 하단 창에 한글로 “버튼”을 입력한다. 그리고 상단의 툴바에서 체크 모양의 아이콘을 클릭한다.

이전 그림에서 보는 것과 같이 체크 모양의 아이콘을 클릭한 다음 Qt Linguist 툴에 편집한 파일을 저장한다. 저장이 완료되면 Qt Creator 창으로 돌아와 Irelease 툴을 실행 한다. Irelease 는 Qt Creator 의 [Tools] > [External] > [Linguist] > [Release Translations (Irelease)] 메뉴를 실행 한다.



<그림> Qt Linguist Tool 화면

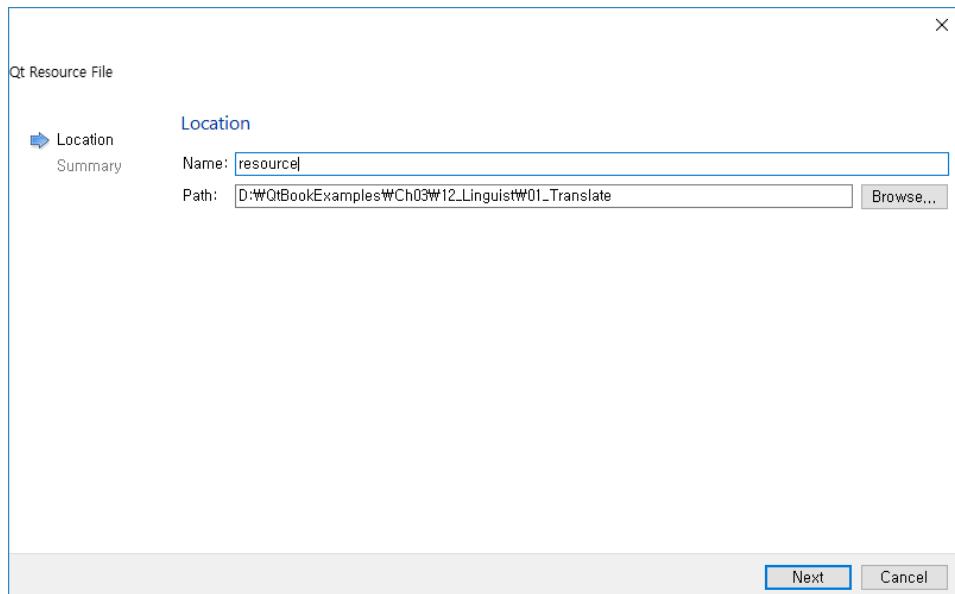
위의 그림에서 보는 것과 같이 Irelease 툴을 실행하면 widget.ts 가 위치한 디렉토리에 widget.qm 파일이 생성된 것을 확인할 수 있다. 이 파일은 widget.ts 파일을 이용해 생성된 실제 사용할 다국어 파일이다.



<그림> Qt Resource File 선택 화면

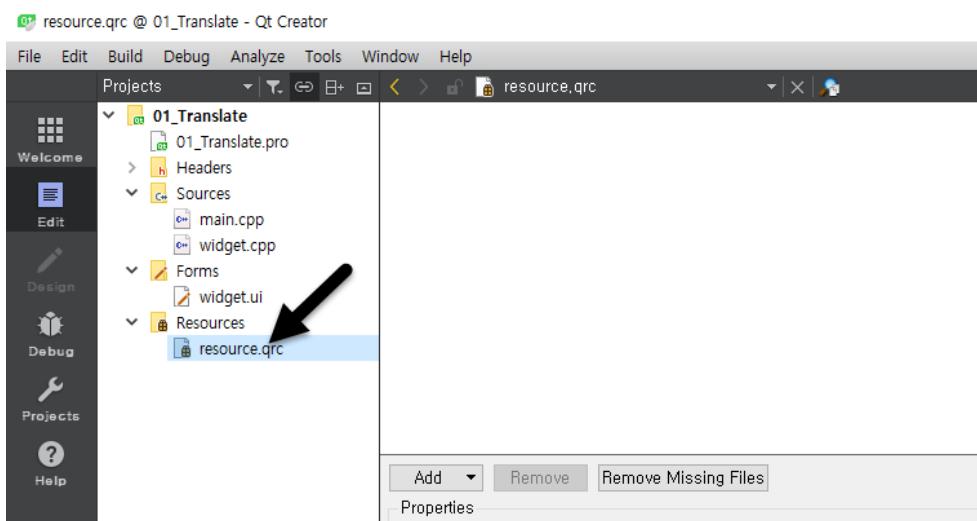
최종 생성된 파일인 widget.qm 파일을 어플리케이션 빌드 시 함께 배포되게 하기 위해 페이지 157

해서 리소스파일에 등록한다. 리소스파일에 등록하기 위해 [Ctrl + N] 단축키 또는 다음 그림에서 보는 것과 같이 Qt Creator 메뉴에서 [File] > [New File or Project] 메뉴를 클릭한다. 다이얼로그 창에서 Qt Resource File 을 선택하고 하단의 [Choose] 버튼을 클릭 한다. 다음 다이얼로그 화면에서는 다음과 같이 Name 항목에 “resource” 를 입력하고 하단의 [Next] 버튼을 클릭한다.



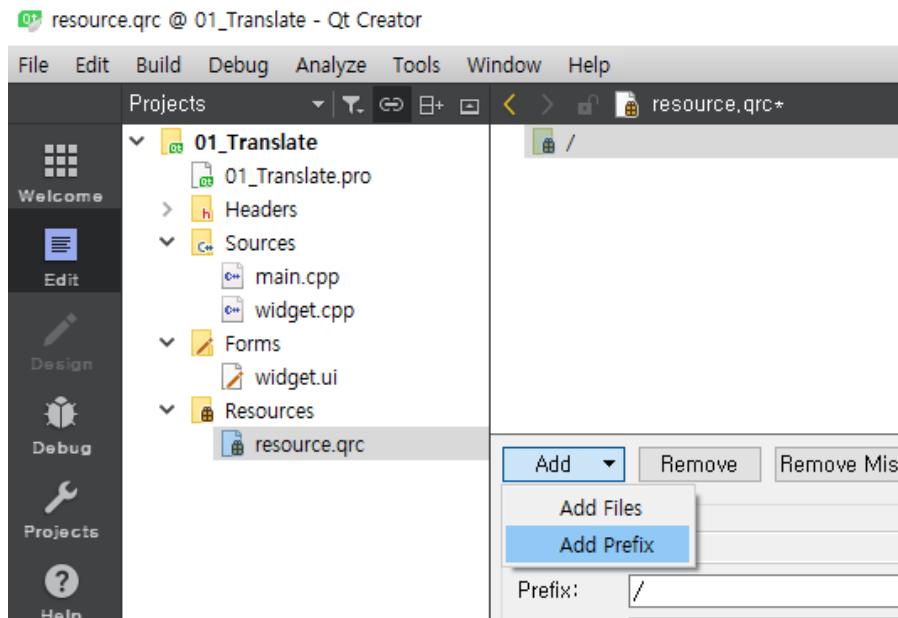
<그림> 리소스 등록 화면

다음 그림에서 보는 것과 같이 Qt Creator 프로젝트 탐색 창에서 resource.qrc 파일을 더블 클릭하면 resource.qrc 에디트 화면이 로딩된다.



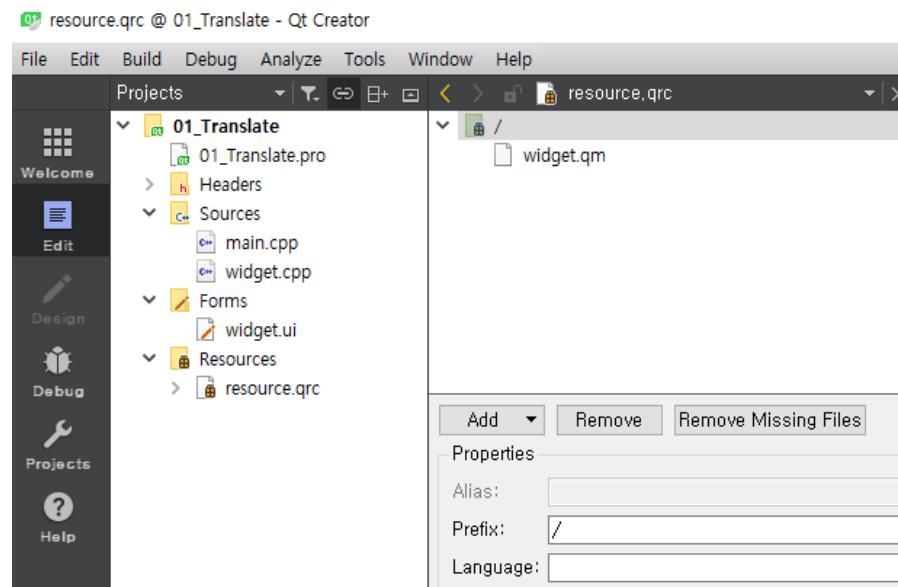
<그림> 리소스 에디트 창

리소스 에디트 창의 [Add] 버튼을 눌러 생성한 widget.qm 파일을 등록한다. widget.qm 파일을 등록하기 전에 다음 그림에서 보는 것과 같이 먼저 [Add Prefix]를 먼저 등록해야 한다.



<그림> Add Prefix 등록 화면

그림에서 보는 것과 같이 [Add Prefix]를 클릭하고 리소스 에디트 창 하단의 Prefix 항목에 "/" 입력한다. 그리고 [Add Files] 메뉴를 클릭해 widget.qm 파일을 등록한다.



<그림> widget.qm 등록 화면

그리고 main.cpp 파일에 다음과 같이 소스코드를 추가한다.

```
#include "widget.h"
#include <QApplication>
#include <QTTranslator>

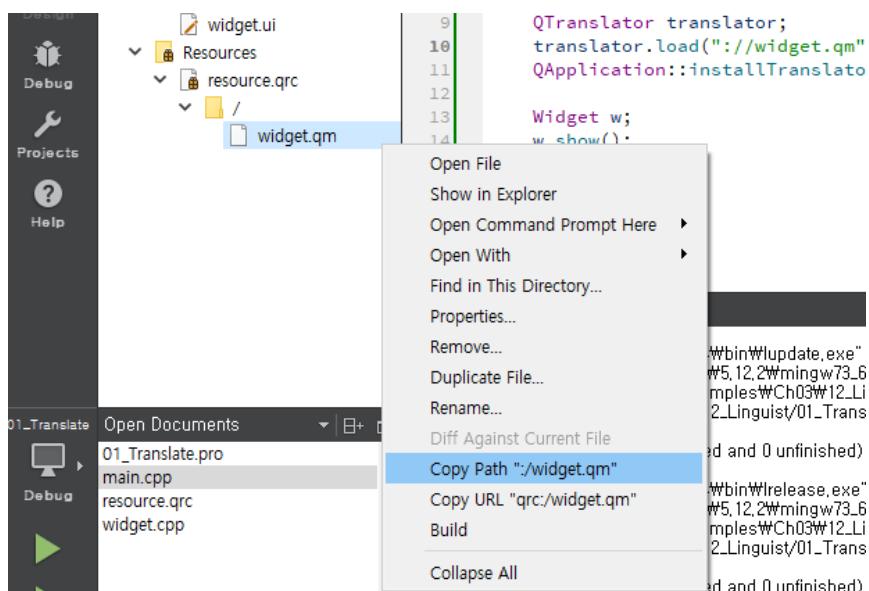
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    QTTranslator translator;
    translator.load(":/widget.qm");
    QApplication::installTranslator(&translator);

    Widget w;
    w.show();

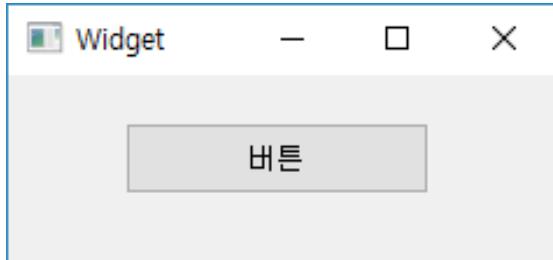
    return a.exec();
}
```

main.cpp 소스코드 파일에서 translator.load() 함수의 첫 번째 인자는 경로는 등록한 리소스파일에 마우스 오른쪽 버튼을 클릭하면 Context 메뉴에서 [Copy Path] 또는 [Copy URL] 을 클릭하면 widget.qm 파일의 경로 및 파일명이 복사된다. 그리고 translator.load() 함수의 첫 번째 인자에 붙여 넣기를 소스코드에서 보는 것과 같이 붙여 넣는다.



<그림> Copy Path 선택 화면

그리고 변경한 소스코드를 빌드하고 실행하면 위젯 상에 QPushButton 의 Text 가 "버튼"으로 변경된 것을 확인할 수 있다.



<그림> 예제 실행 화면

main.cpp 소스코드 상에 추가한 코드는 다국어를 사용할 위젯을 이전에 먼저 선언해야 한다.

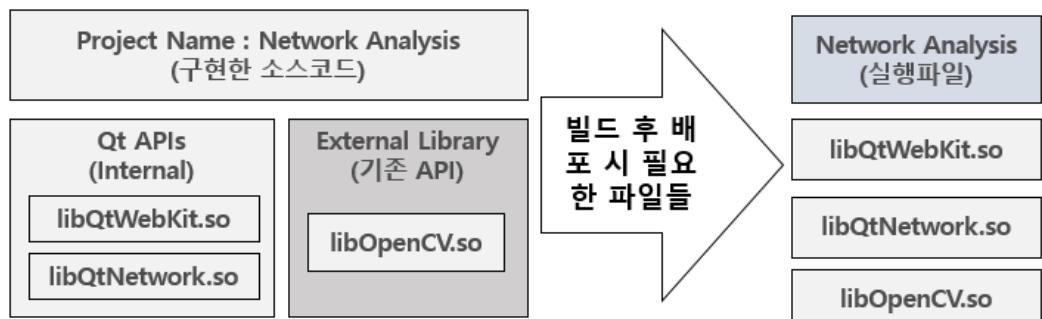
만약 Widget 오브젝트를 생성 한 다음 추가한 다국어 소스코드를 추가하면 Widget 상에 적용되지 않는다.

따라서 Widget 오브젝트를 생성하기 이전에 먼저 선언해야 함을 주의하도록 하자. 지금까지 다룬 예제 소스코드는 Ch03 > 12_Linguist > 01_Translate 딕토리를 참조하면 된다.

3.13. 라이브러리 제작

Qt 는 C/C++로 구현된 외부 라이브러리를 포팅해 소프트웨어를 개발할 수 있다. 예를 들어 OpenCV, CUDA, OpenCL 등과 같은 다양한 외부 라이브러리를 사용할 수 있다.

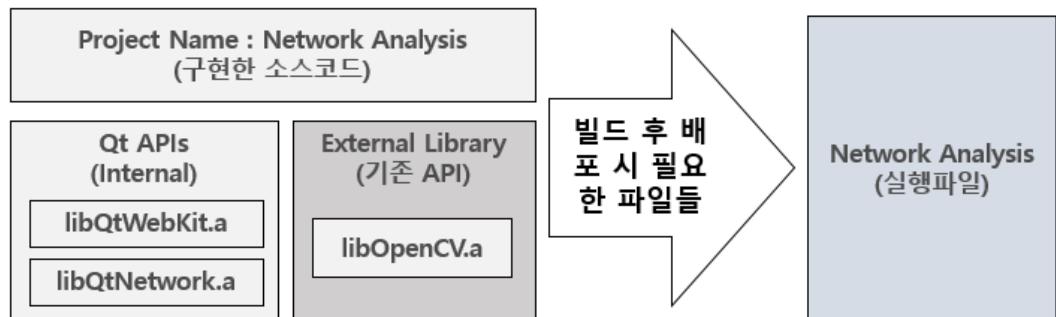
Qt 는 Shared 라이브러리 방식의 라이브러리와 Static 라이브러리 방식 모두 사용 가능하다. Shared 라이브러리 방식은 구현하고자 하는 소프트웨어를 외부라이브러리를 사용해 구현하였다면 빌드한 실행 파일과 Shared 라이브러리 도 함께 배포해야만 소프트웨어가 동작한다.



<그림> Shared 라이브러리 방식

위의 그림에서 보는 것과 같이 리눅스 운영체제에서 확장자가 so 인 파일은 Shared 방식의 라이브러리 파일이다.

Static 라이브러리 방식은 빌드 시 라이브러리들이 실행 파일과 하나의 파일로 통합된다. 따라서 Static 방식으로 빌드하였다면 배포 시 실행 파일만 배포하면 된다.



<그림> Static 라이브러리 방식

위의 그림에서 보는 것과 같이 라이브러리 확장자가 a 인 것은 리눅스 운영체제에서 Static 라이브러리 파일의 확장자로 사용한다. Qt에서 외부 라이브러리를 사용할 때 주

의해야 할 점으로 외부 라이브러리를 컴파일한 컴파일러와 동일한 컴파일러를 사용해 빌드해야 한다.

예를 들어 Qt로 개발한 소프트웨어를 MinGW 기반 컴파일러로 빌드 하였다면 외부 라이브러리 또한 MinGW 기반의 컴파일러로 빌드한 라이브러리를 사용해야 한다.

또한 MinGW 는 32Bit 와 64Bit 두 가지가 있는데 구현하고자 하는 소프트웨어와 라이브러리도 동일한 Bit로 컴파일한 라이브러리를 사용해야 한다.

예를 들어 구현하는 소프트웨어 컴파일러가 MinGW 32Bit 컴파일러를 사용해 빌드 했다면 외부 라이브러리도 MinGW 32Bit 컴파일러로 컴파일한 라이브러리를 사용해야 한다. 다음은 Qt에서 외부 라이브러리를 사용하기 위해 프로젝트 파일에 다음과 같이 라이브러리를 명시하면 된다.

```
INCLUDEPATH += "C:/Intel/OpenCL/sdk/include"  
LIBS += -L"C:/Intel/OpenCL/sdk/lib/x64"  
LIBS += -lOpenCL
```

위의 예제에서 보는 것과 같이 INCLUDEPATH 키워드는 외부 라이브러리의 헤더 파일이 있는 위치 디렉토리를 명시하면 된다.

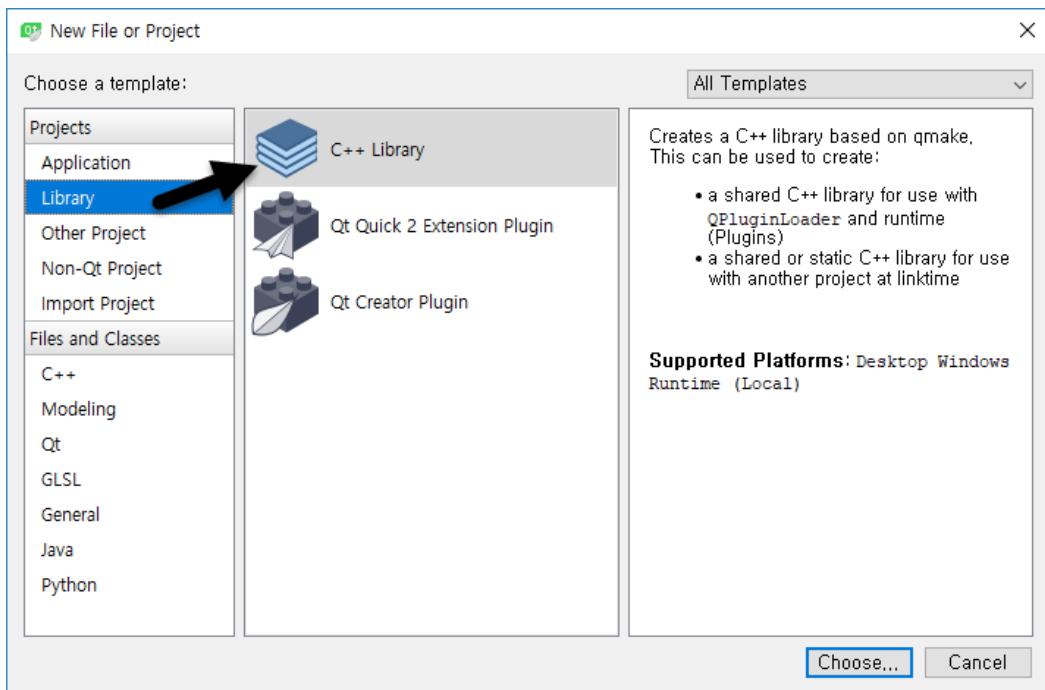
LIBS 키워드는 so 파일 또는 a 파일과 같이 라이브러리 파일이 위치한 디렉토리를 명시하면 된다. 디렉토리 경로 앞에 "-L"은 라이브러리를 위치한 디렉토리이다.

그리고 세 번째 라인의 -lOpenCL은 실제 사용할 Shared 라이브러리 파일을 명시하면 된다. 라이브러리명 앞에 "-l" 옵션은 라이브러리 파일을 지정하겠다는 의미이다. 라이브러리 파일에 위치한 디렉토리에 보면 확장자가 리눅스인 경우 so이며 MS윈도우 인 경우 dll 이거나 lib이다.

● Qt 라이브러리 구현

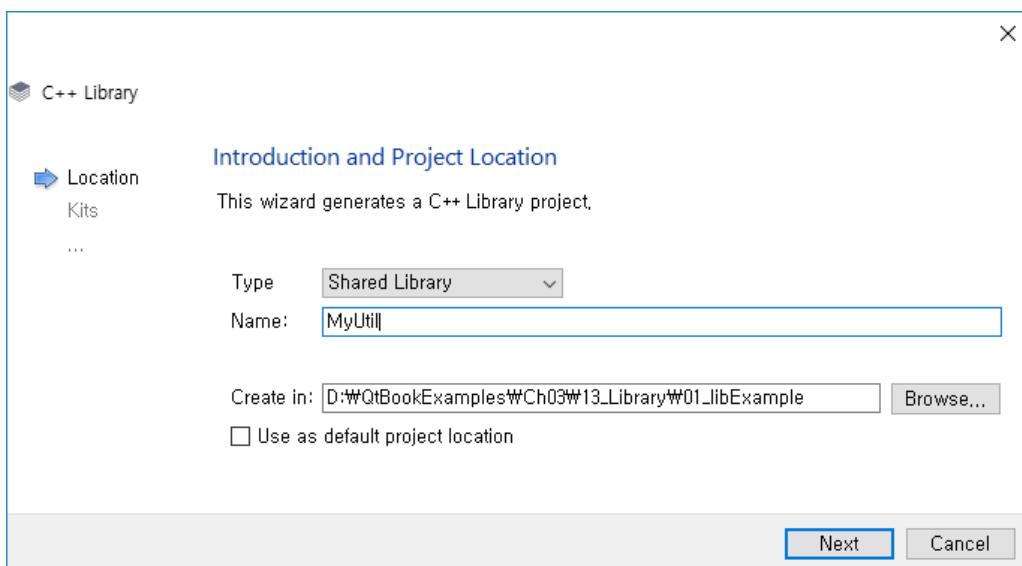
이번 예제에서는 2개의 Qt 프로젝트를 생성할 것이다. 첫 번째 프로젝트는 라이브러리 구현을 위한 프로젝트이다. 두 번째 프로젝트는 첫 번째 프로젝트에서 구현한 라이브러리를 사용할 것이다.

Qt Creator Tool에서 새로운 프로젝트를 생성한다. 프로젝트 생성 다이얼로그에서 다음 그림에서 보는 것과 같이 좌측 상단 [Projects] 탭에서 Library 항목을 선택하고 중앙에 리스트 중 C++ Library 항목을 선택한다.



<그림> 첫 번째 라이브러리 프로젝트 생성 화면

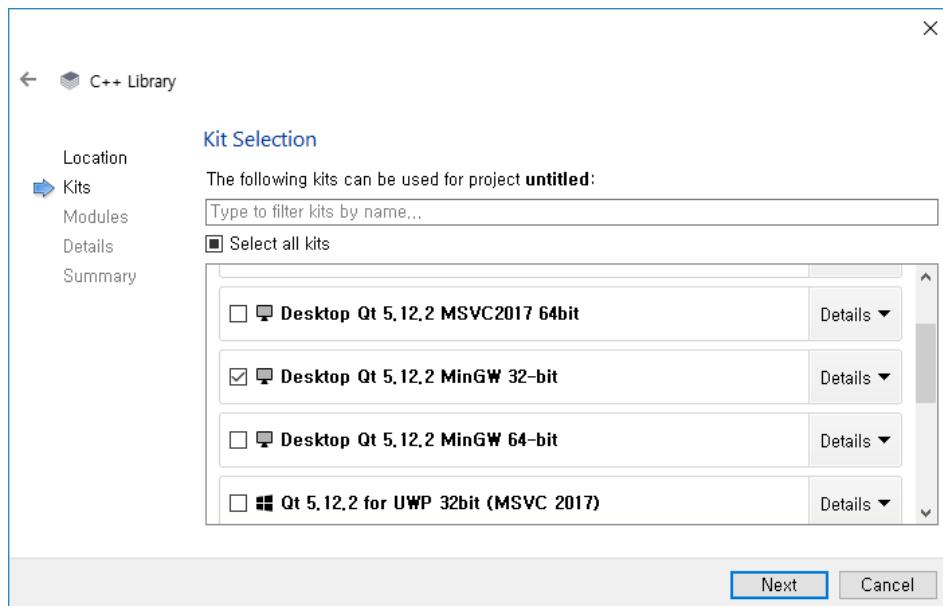
다음 다이얼로그 화면은 라이브러리 타입, 이름 그리고 위치를 선택한다. Type 항목은 Shared Library 항목을 선택한다. Name 항목에서 MyUtil 을 입력한다. Create In 항목은 라이브러리 소스코드 또는 라이브러리 프로젝트가 위치할 상위 디렉토리를 지정한다.



<그림> 두 번째 프로젝트 생성 화면

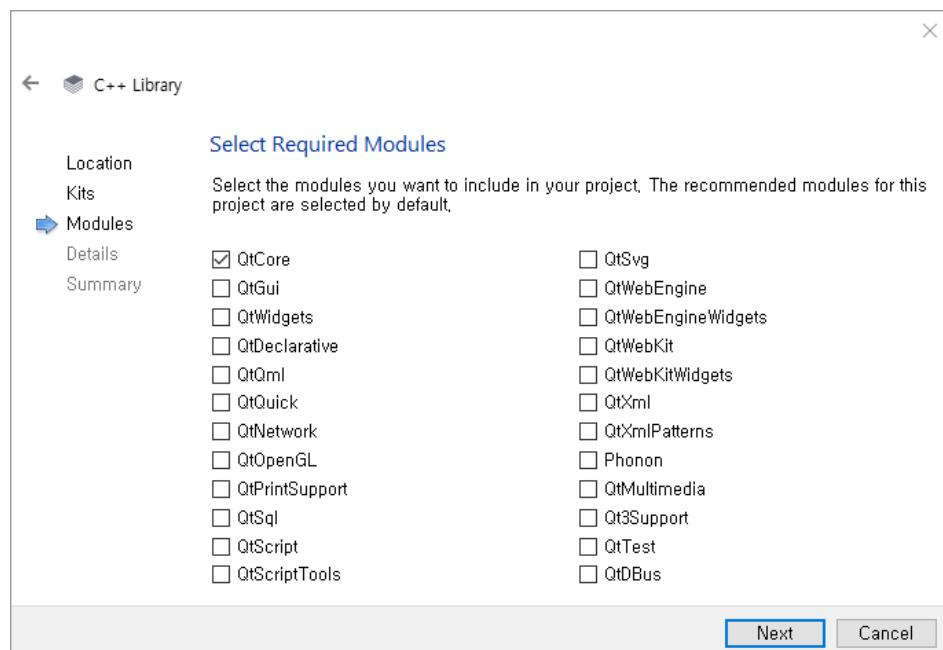
다음은 빌드할 컴파일러를 선택한다. 다음 그림에서 보는 것과 같이 MinGW 32-bit 버

전을 선택한다.



<그림> 컴파일러 선택 화면

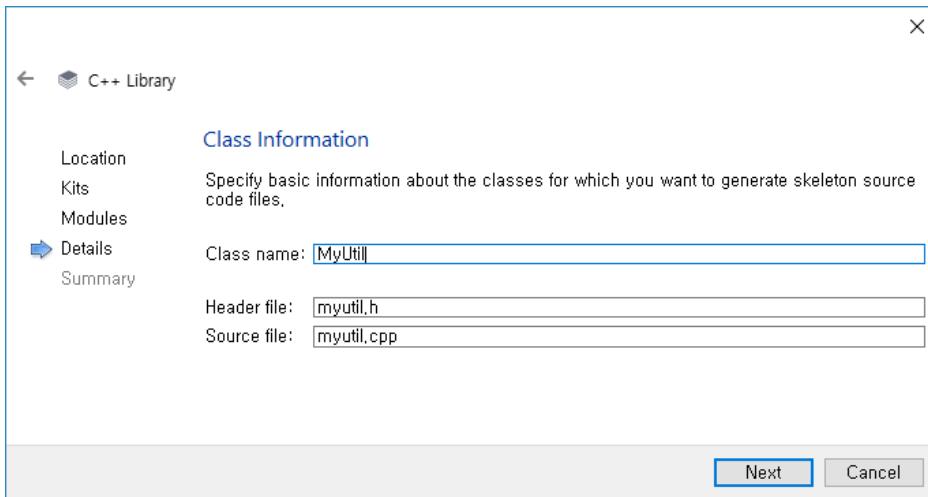
다음 다이얼로그 화면은 라이브러리에서 사용할 Qt 모듈을 선택하는 화면이다. 이 프로젝트에서는 QtCore 항목만 사용 할 것이다.



<그림> 사용할 Qt 라이브러리 항목 선택 화면

다음은 생성할 라이브러리 클래스를 정보를 입력한다. Class name 은 MyUtil 을 입력하

고 헤더 파일은 myutil.h, 소스코드 파일은 myutil.cpp를 입력한다.



<그림> Class Information 화면

다음 다이얼로그는 하단에 [Finish] 버튼을 누르면 라이브러리 프로젝트 생성이 완료된다. 다음 예제 소스코드에서 보는 것과 같이 프로젝트를 살펴보자.

```
QT      -= gui

TARGET = MyUtil
TEMPLATE = lib

DEFINES += MYUTIL_LIBRARY
DEFINES += QT_DEPRECATED_WARNINGS
SOURCES += myutil.cpp

HEADERS += myutil.h myutil_global.h

unix {
    target.path = /usr/lib
    INSTALLS += target
}
```

일반 어플리케이션을 프로젝트와 달리 라이브러리 프로젝트는 TEMPLATE 키워드가 다르다. 일반 어플리케이션은 TEMPLATE 키워드에 app 가 명시되지만 라이브러리는 lib 가 명시된다.

이 라이브러리는 두개의 인자 값을 받는다. 그리고 두 개의 값을 더한 값을 리턴 하는 기능을 제공하는 라이브러리를 구현할 것이다. 생성된 myutil.h 헤더파일상에 다음과

같이 소스코드를 작성한다.

```
#ifndef MYUTIL_H
#define MYUTIL_H

#include "myutil_global.h"

class MYUTILSHARED_EXPORT MyUtil
{
public:
    MyUtil();
    qint32 getSumValue(qint32 a, qint32 b);
};

#endif // MYUTIL_H
```

다음은 myutil.cpp 소스코드 파일에 다음과 같이 소스코드를 추가한다.

```
#include "myutil.h"

MyUtil::MyUtil()
{
}

qint32 MyUtil::getSumValue(qint32 a, qint32 b)
{
    return a + b;
}
```

위의 예제 소스코드와 같이 소스코드를 추가하고 빌드한다. 에러가 발생하지 않고 정상적으로 빌드 되었다면 라이브러리가 빌드 되었을 것이다.

Ch03 > 13_Library > 01_libExample > build-MyUtil-Desktop_Qt_5_12_2_MinGW_32_bit-Debug > debug

이름	수정한 날짜	유형	크기
libMyUtil.a	2019-05-10 오후...	A 파일	8KB
moc_myutil.cpp	2019-05-10 오후...	C++ Source file	3KB
moc_myutil.o	2019-05-10 오후...	O 파일	321KB
moc_prelude.h	2019-05-10 오후...	C++ Header file	15KB
MyUtil.dll	2019-05-10 오후...	응용 프로그램 확장	620KB
myutil.o	2019-05-10 오후...	O 파일	313KB

<그림> 빌드 된 디렉토리

이전 그림에서 보는 것과 같이 빌드 된 디렉토리 하위에 libMyUtil.a 와 MyUtil.dll 파일

이 생성된 것을 확인할 수 있을 것이다. 이번에는 빌드한 라이브러리를 사용할 프로젝트를 생성한다. 생성할 프로젝트는 GUI 기반의 위젯을 생성한다.

프로젝트 생성 시, 이전에 말한 것과 같이 라이브러리를 빌드한 동일한 컴파일러를 사용해야 한다. 빌드 할 프로젝트를 MinGW 32 Bit 버전을 선택한다. 그리고 프로젝트 파일에 사용할 라이브러리를 다음 예제에서 보는 것과 같이 추가한다.

```
QT      += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = AppWithLibrary
TEMPLATE = app

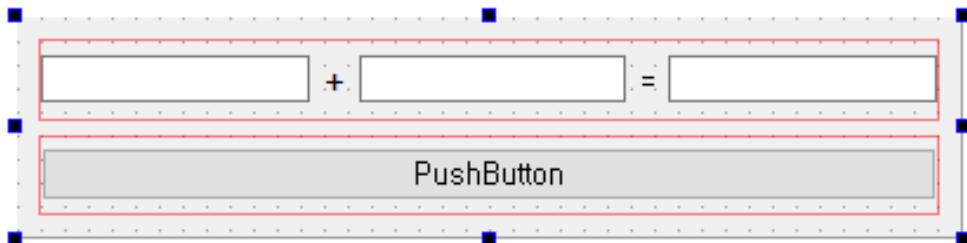
SOURCES += main.cpp \
           widget.cpp

HEADERS += widget.h

FORMS    += widget.ui

LIBS += -L$$PWD/../../[빌드된 디렉토리 명]/debug/ -lMyUtil
INCLUDEPATH += $$PWD/../../MyUtil
DEPENDPATH += $$PWD/../../MyUtil
```

아래 그림에서 보는것과 같이 GUI 상에 위젯을 배치한다.



<그림> GUI 위젯 배치 화면

위의 그림에서 보는 것과 같이 [PushButton] 버튼을 클릭하면 두 개의 입력 받은 값을 더해 결과 QLineEdit 위젯에 출력할 것이다. 아래 widget.h 헤더 파일과 같이 소스코드를 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H
```

```

#include <QWidget>
#include "myutil.h"

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    MyUtil *myUtil;

    Ui::Widget *ui;

private slots:
    void slotBtnClick();

};

#endif // WIDGET_H

```

다음은 widget.cpp 소스코드를 작성한다.

```

#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);

    myUtil = new MyUtil();
    connect(ui->pbtSum, SIGNAL(pressed()), this, SLOT(slotBtnClick()));
}

void Widget::slotBtnClick()
{
    qint32 arg1 = ui->leArg1->text().toInt();
}

```

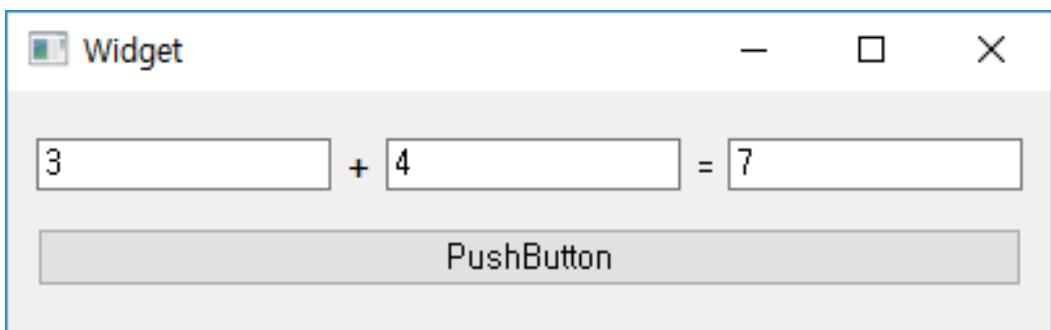
```

qint32 arg2 = ui->leArg2->text().toInt();

qint32 sumValue = myUtil->getSumValue(arg1, arg2);
ui->leSum->setText(QString("%1").arg(sumValue));
}

Widget::~Widget()
{
    delete ui;
}

```



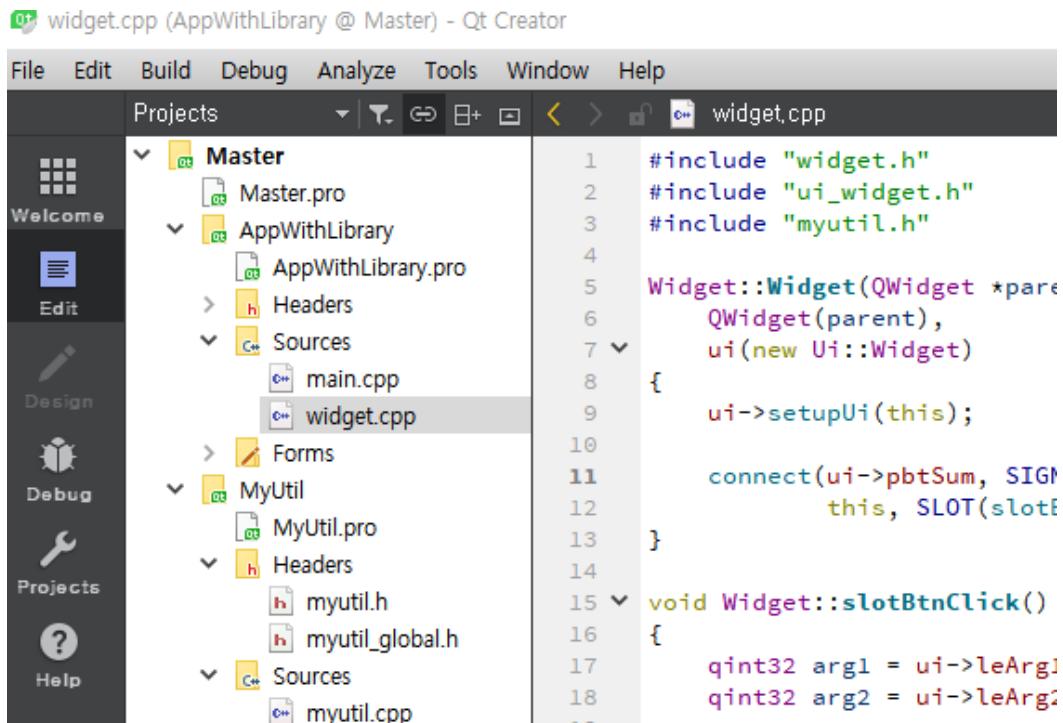
<그림> 예제 실행 화면

위의 그림에서 보는 것과 같이 두개의 정수 값을 입력 후 [PushButton] 버튼을 클릭하면 우측에 결과 값을 출력한다. 살펴본 예제 소스코드는 Ch03 > 13_Library > 01_libExample 디렉토리에 전체 소스코드가 있다.

● 라이브러리와 함께 빌드하기

라이브러리 프로젝트와 어플리케이션 프로젝트를 별도의 Qt Creator 툴을 이용해 빌드하였다. 만약 라이브러리를 수정해야 하는 일이 발생한다고 가정해 보자 라이브러리 프로젝트를 Open 한 Qt Creator 툴에서 라이브러리를 수정한 다음 빌드하고 어플리케이션 프로젝트를 오픈한 Qt Creator 창으로 돌아와 수정한 라이브러리를 적용하여야 하는 불편함이 있을 것이다.

하지만 Qt 에서는 여러 프로젝트를 한 개의 Qt Creator 창에서 수정할 수 있다. 이렇게 하면 장점으로 프로젝트 상에서 라이브러리 디렉토리 위치만 정확 다면 동일한 Qt Creator 툴에서 라이브러리와 어플리케이션 소스코드를 동시에 수정하고 적용할 수 있을 것이다. 다음 그림은 두개의 프로젝트를 한 개의 Qt Creator 툴을 Open 한 예이다.



<그림> 여러 개의 프로젝트를 Open한 화면

위의 그림에서 보는 것과 같이 여러 개의 프로젝트를 Qt Creator에서 Open 하기 위해서 다음과 같이 프로젝트 파일을 작성한다.

```
TEMPLATE      = subdirs

SUBDIRS       = MyUtil \
                  AppWithLibrary

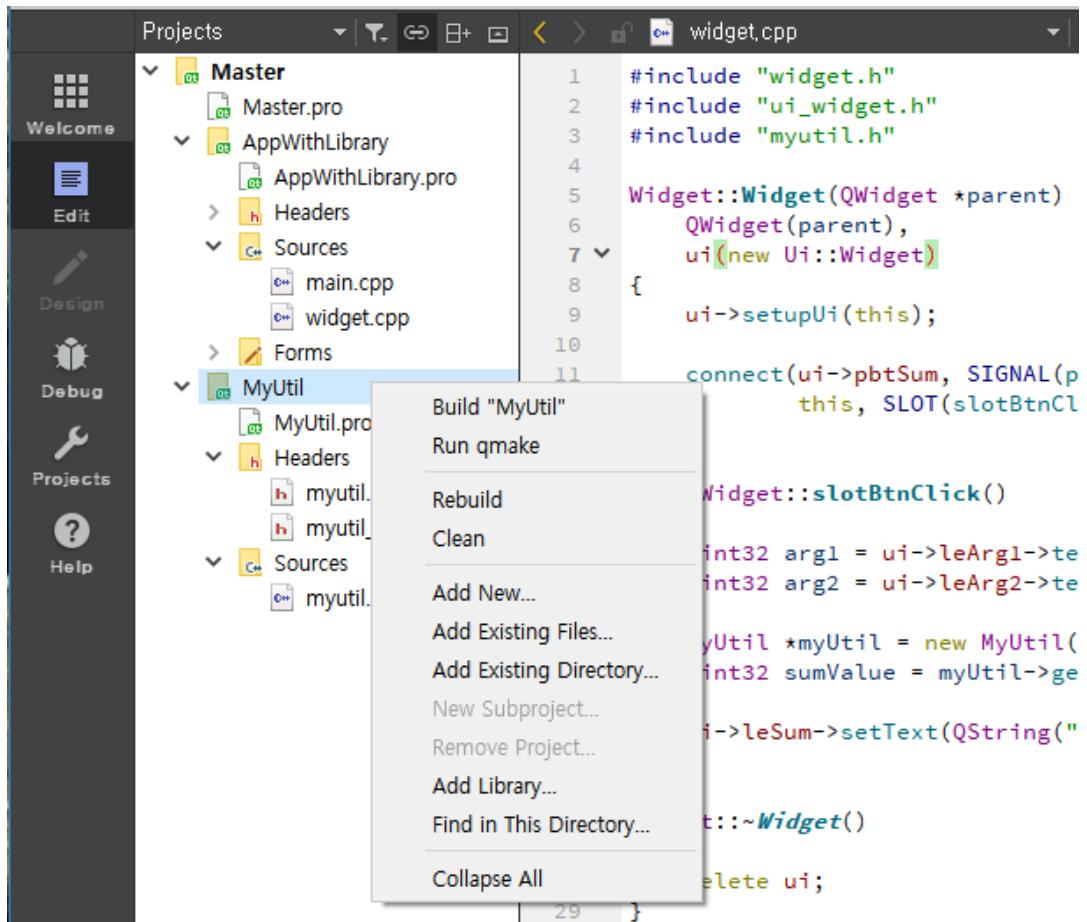
MyUtil.file   = MyUtil/MyUtil.pro
AppWithLibrary.file = AppWithLibrary/AppWithLibrary.pro
```

위의 예제에서 보는 것과 같이 프로젝트파일(.pro) 파일을 작성한다. 이 프로젝트 파일에서는 두 개의 프로젝트 파일을 각각 명시함으로써 한 개의 Qt Creator 툴에서 프로젝트를 수정하고 빌드 할 수 있기 때문에 매우 편리하게 빌드할 수 있다.

예를 들어 라이브러리 소스코드를 변경하게 되면 변경된 프로젝트를 자동으로 Qt Creator 가 빌드 해주기 때문에 라이브러리를 빌드를 따로 해줄 필요가 없이 편하게 빌드할 수 있다.

하지만 최초 라이브러리를 참조해야 하기 때문에 라이브러리를 먼저 따로 빌드해야한다. 라이브러리를 따로 빌드하기 위해서는 라이브러리 프로젝트에 마우스를 클릭하고

다시 마우스 오른쪽 버튼을 누르면 아래 그림과 같이 메뉴가 생성된다.

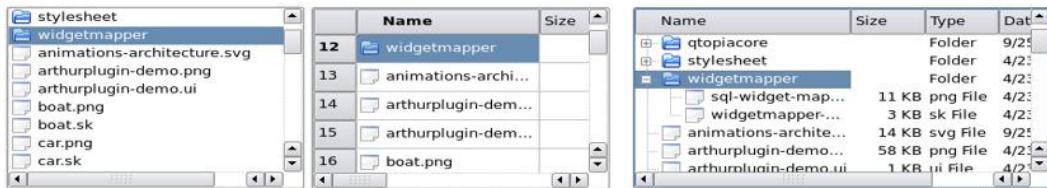


<그림> 라이브러리 빌드 화면

위의 그림에서 보는 것과 같이 마우스 오른쪽 버튼을 클릭하면 라이브러리 프로젝트를 먼저 빌드할 수 있다. 전체 소스코드는 Ch03 > 13_Library > 02_libExample 디렉토리를 참조하면 된다.

3.14. Model 과 View

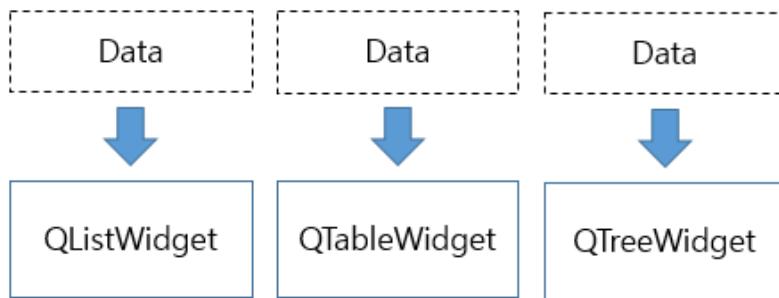
Qt에서는 다음 그림에서 보는 것과 같이 표와 같은 위젯에 데이터를 표시하기 위한 위젯으로 QListWidget, QTableWidget, QTreeWidget, QListView, QTableView, QTreeView, QColumnView 등 다양한 클래스를 제공한다.



<그림> 표와 같은 위젯을 표시하기 위한 위젯들

QListWidget 은 QListView 동일한 형태이다. 한 개의 Column 의 데이터를 여러 라인에 표시하기 위한 위젯 이다. 하지만 QListWidget 과 QListView 는 데이터를 삽입/수정/삭제하는데 차이가 있다.

클래스 마지막에 Widget 이라는 단어를 사용한 클래스들은 아래 그림에서 보는 것과 같이 데이터를 직접 삽입/수정/삭제 할 수 있는 멤버 함수를 제공한다.



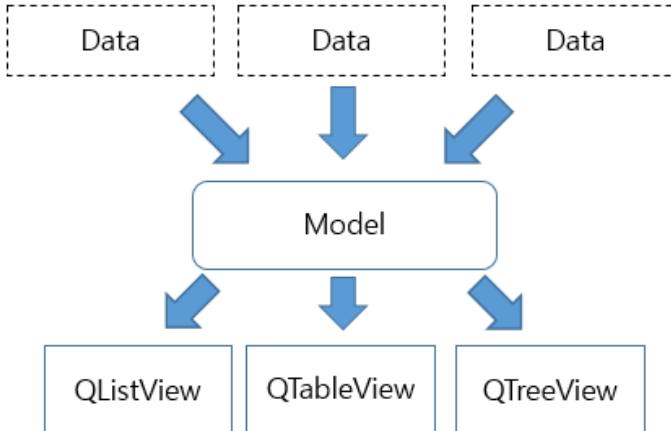
<그림> QListWidget, QTableWidget 그리고 QTreeWidget

위의 그림에서 보는 것과 같이 클래스 이름 마지막에 Widget 이라는 단어가 쓰인 위젯들은 모두 직접 데이터 삽입/수정/삭제 가 가능한 멤버 함수를 제공한다. 예를 들어 QListWidget 위젯 클래스는 insertItem() 을 제공한다.

QTableWidget 클래스는 데이터를 삽입하기 위해 다른 멤버 함수를 제공한다. 즉 QListWidget 에서 데이터 삽입하는 방법과 QTableWidget 의 사용 방법이 다르기 때문에 각각의 클래스에서 제공하는 데이터 삽입하는 방법을 익혀야 한다는 단점이 있다.

그리고 QListView, QTableView, QTreeView 클래스와 같이 마지막에 View 라는 단어를

사용하는 위젯 클래스들은 각각의 멤버 함수를 사용해 데이터를 삽입/수정/삭제 하지 않고 Model 클래스라는 매개체를 이용해 데이터를 삽입/수정/삭제 할 수 있다.

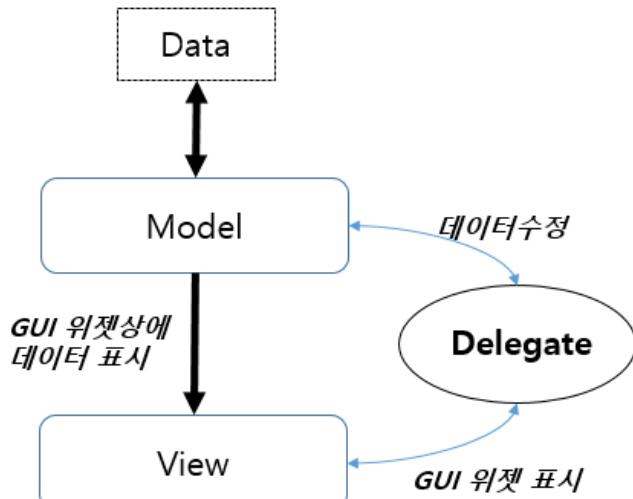


<그림> Model / View 아키텍처

위의 그림에서 보는 것과 같이 QListView, QTableView, QTreeView 클래스들은 위젯에 데이터를 삽입/수정/삭제 하기 위해 각 클래스에서 제공하는 멤버 함수를 사용하지 않고 Model 클래스를 사용하다.

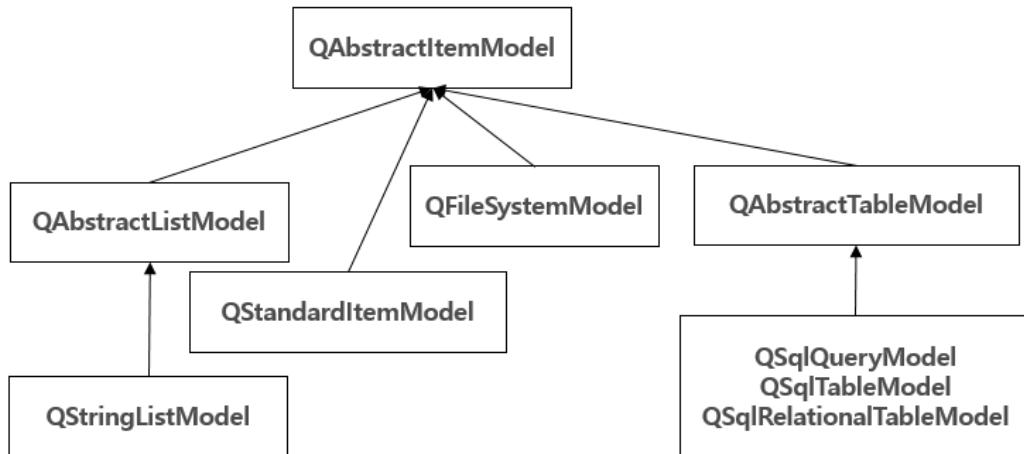
이 방식을 사용할 경우 QListView 를 사용하든지 QTableView 를 사용하든지 동일한 Model을 사용할 수 있다는 장점을 가지고 있다.

Qt에서 제공하는 Model/View 는 다음 그림에서 보는 것과 같이 Delegate를 이용해 데이터를 제어할 수 있다. 예를 들어 View 위젯 상에 표시된 데이터 항목 중 특정 항목을 마우스로 더블 클릭해 데이터를 수정하기 위해서 이벤트를 발생해야 하는데 Qt에서는 이러한 이벤트를 처리하기 위해 Delegate를 사용할 수 있다.



<그림> Delegate 를 도식화

Model 클래스는 데이터를 관리(삽입/수정/삭제) 하기 위한 기능을 제공한다. 예를 들어 QSqlQueryModel 클래스를 이용하면 SQL 문을 직접 쿼리(QUERY) 할 수 있는 멤버 함수를 제공한다. 따라서 별도의 데이터베이스 쿼리로 가져온 데이터를 편집 후에 Model 을 이용해도 되지만 QSqlQueryModel 클래스를 이용하면 직접 데이터를 삽입할 수 있다. 이외에도 다음 그림에서 보는 것과 같이 다양한 Model 클래스를 제공한다.



<그림> Model 클래스들과 상속 관계

Model 클래스는 단순히 한 개의 컬럼으로 구성된 데이터 리스트를 관리하기 위한 Model 클래스부터 복잡한 Model 클래스까지 다양한 Model 클래스를 제공한다.

QStringListModel 클래스는 QString 데이터 타입의 단순한 데이터 리스트를 관리할 있는 기능을 제공한다.

```
QStringListModel *model = new QStringListModel();  
  
QStringList list;  
list << "Hello World" << "Qt Programming" << "Model is Good";  
  
model->setStringList(list);  
...
```

QStandardItemModel 클래스는 테이블 형태 또는 트리 형태와 같이 데이터를 관리할 수 있는 기능을 제공한다.

```
QStandardItemModel model(4, 4);  
  
for (int row = 0; row < 4; ++row)
```

```

{
    for (int column = 0; column < 4; ++column) {
        QString data = QString("row %0, column %1").arg(row).arg(column);
        QStandardItem *item = new QStandardItem(data);
        model.setItem(row, column, item);
    }
}
...

```

QFileSystemModel 클래스는 파일 시스템으로부터 일어온 파일과 디렉토리를 관리할 수 있는 기능을 제공한다.

```

QFileSystemModel *model = new QFileSystemModel;
model->setRootPath(QDir::currentPath());

QTreeView *tree = new QTreeView(this);
tree->setModel(model);
...

```

QSqlQueryModel 클래스는 SQL 쿼리(QUERY) 문을 이용해 데이터베이스 테이블로부터 데이터를 액세스 할 수 있는 기능을 제공하며 QSqlTableModel 클래스는 데이터베이스 테이블로부터 데이터를 가져올 때 특정 테이블 명을 인자로 전달하면 Model에 데이터를 가져올 수 있는 기능을 제공한다.

예를 들어 setTable() 멤버 함수에 첫 번째 인자로 테이블 명을 입력하면 데이터베이스 테이블로부터 데이터를 가져와 저장한다.

```

QSqlQueryModel *model = new QSqlQueryModel;

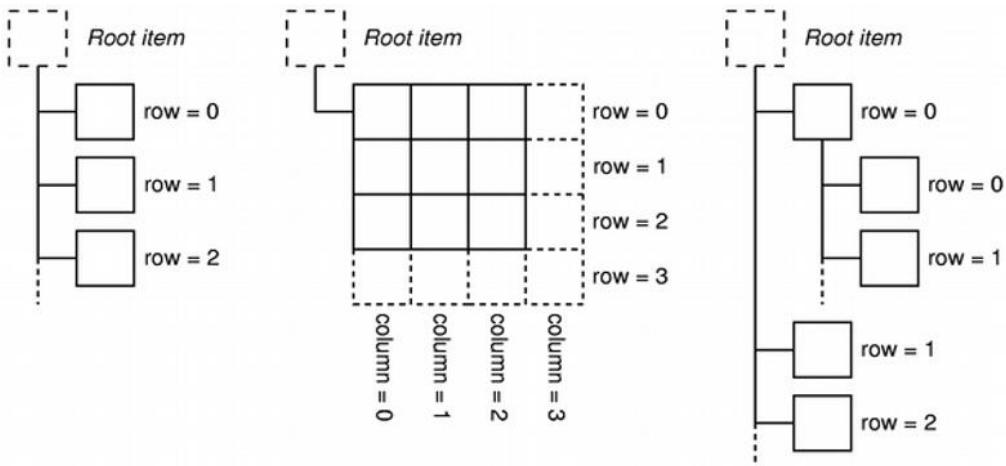
model->setQuery("SELECT name, salary FROM employee");

model->setHeaderData(0, Qt::Horizontal, tr("Name"));
model->setHeaderData(1, Qt::Horizontal, tr("Salary"));

QTableView *view = new QTableView;
view->setModel(model);
view->show();
...

```

Model / View 를 이용해 많은 양의 데이터를 표시할 때 다음 그림에서 보는 것과 같이 3가지 종류로 구분할 수 있다.

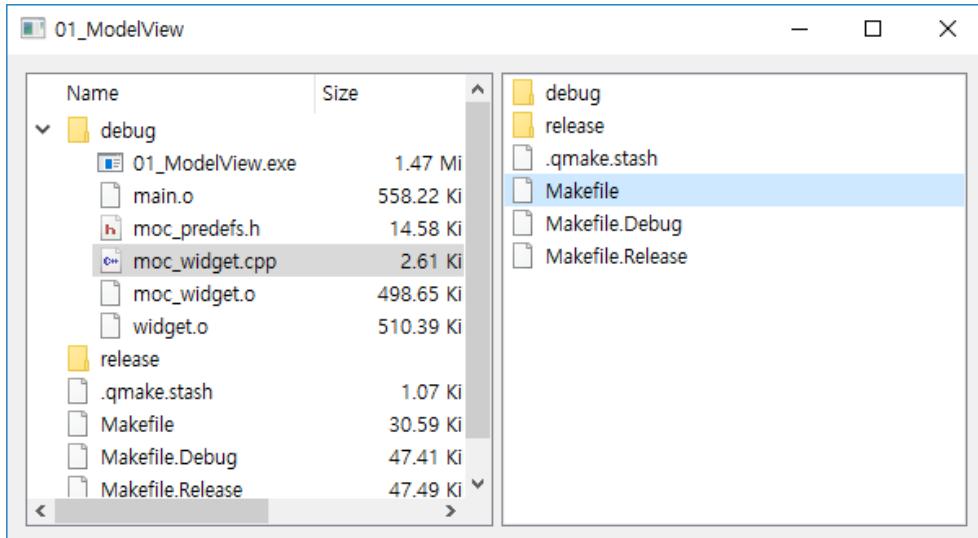


<그림> View 의 데이터 표시 형태의 종류

가장 좌측에 있는 데이터 형태라면 QListWidget 클래스를 사용하는 것이 적합하다. 표와 같이 표시해야 한다면 QTableView 를 사용하는 것이 적합하다. 그리고 트리 형태로 표시해야 한다면 QTreeView를 사용하는 것이 적합하다.

- QTreeView 와 QListWidget 클래스를 이용한 예제

이 예제는 파일로부터 데이터를 읽어와 Model 에 저장한 다음 Model 을 View 과 연결 해 파일 시스템을 표시하는 예제이다.



<그림> QTreeView 와 QListWidget 예제 실행 화면

좌측은 QTreeView 클래스를 이용해 파일시스템으로부터 가져온 데이터를 표시하였다.

우측의 QListview 위젯은 현재 디렉토리에 존재하는 파일과 디렉토리를 표시하였다.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    resize(600, 300);
    QSplitter *splitter = new QSplitter(this);

    QFileSystemModel *model = new QFileSystemModel;
    model->setRootPath(QDir::currentPath());

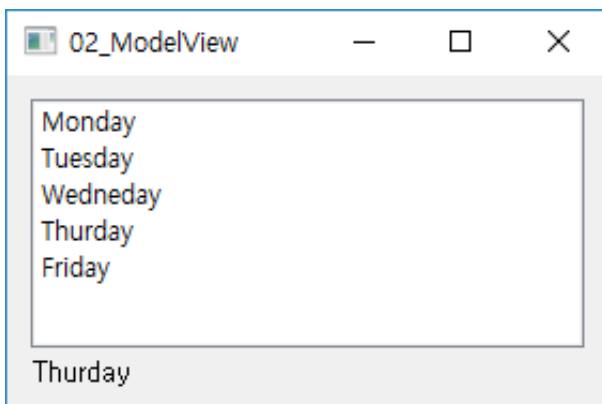
    QTreeView *tree = new QTreeView(splitter);
    tree->setModel(model);
    tree->setRootIndex(model->index(QDir::currentPath()));

    QListView *list = new QListView(splitter);
    list->setModel(model);
    list->setRootIndex(model->index(QDir::currentPath()));

    QVBoxLayout *layout = new QVBoxLayout();
    layout->addWidget(splitter);
    setLayout(layout);
}
...
```

전체 소스는 Ch03 > 14_ModelView > 01_ModelView 디렉토리를 참조하면 된다.

- QListView 클래스를 이용한 예제



좌측의 그림은 QListView 클래스와 QAbstractItemModel 클래스를 이용한 예제 실행 화면이다. QListView 는 한 개의 컬럼과 여러 줄로 표시하는데 적합한 위젯이다.

<그림> QListView 예제 실행 화면

```

...
resize(600, 300);
QStringList strList;

strList << "Monday" << "Tuesday" << "Wednesday" << "Thursday" << "Friday";
QAbstractItemModel *model = new QStringListModel(strList);

QListView *view = new QListView();
view->setModel(model);

QModelIndex index = model->index(3, 0);
QString text = model->data(index, Qt::DisplayRole).toString();

QLabel *lbl = new QLabel("");
lbl->setText(text);

QVBoxLayout *lay = new QVBoxLayout();
lay->addWidget(view);
lay->addWidget(lbl);

setLayout(lay);
...

```

전체 소스는 Ch03 > 14_ModelView > 02_ModelView 디렉토리를 참조하면 된다.

- `QTableView` 클래스를 이용한 예제

`QTableView` 클래스는 아래 그림에서 보는 것과 같이 표 형태의 데이터를 표시하는데 적합한 위젯이다.

	Subject	Description	Date
Col 1	Monitor	LCD	2030-10-04 오전 12:00
Col 2	CPU	Samsung	2030-12-05 오전 12:00

<그림> `QTableView` 예제 실행 화면

```

...
QStandardItemModel *model = new QStandardItemModel(0, 3);

model->setHeaderData(0, Qt::Horizontal, QObject::tr("Subject"));
model->setHeaderData(1, Qt::Horizontal, QObject::tr("Description"));
model->setHeaderData(2, Qt::Horizontal, QObject::tr("Date"));

model->setVerticalHeaderItem(0, new QStandardItem("Col 1"));
model->setVerticalHeaderItem(1, new QStandardItem("Col 2"));

model->setData(model->index(0, 0), "Monitor");
model->setData(model->index(0, 1), "LCD");
model->setData(model->index(0, 2),
QDateTime(QDate(2030, 10, 4)));

model->setData(model->index(1, 0), "CPU");
model->setData(model->index(1, 1), "Samsung");
model->setData(model->index(1, 2),
QDateTime(QDate(2030, 12, 5)));

QTableView *table = new QTableView();
table->setModel(model);

QVBoxLayout *lay = new QVBoxLayout();
lay->addWidget(table);

setLayout(lay);
...

```

전체 소스코드는 Ch03 > 14_ModelView > 03_TableModel 디렉토리를 참조하면 된다.

● QTableWidget 예제

이번 예제는 Model 을 사용하지 않고 QTableWidget 을 이용해 데이터를 삽입하는 예제를 다루어 볼 것이다. 또한 이 예제는 첫 번째 Column 에 QCheckBox 위젯을 삽입하는 방법도 다루어 보도록 하겠다.

<그림> QTableWidget 예제 실행 화면

위의 그림에서 보는 것과 같이 QTableWidget 헤더에 체크 박스가 있다. 이 체크박스를 변경하면 모든 라인의 컬럼에의 값이 헤더에 있는 체크 박스의 값과 동일하게 변경이 가능하다. 그리고 각각의 체크 박스의 값을 사용자가 변경할 수 있다.

Qt는 QTableWidget 의 헤더(Header)에 원하는 모양 또는 특정 위젯이 추가된 헤더를 커스터마이징 하기 위한 방법을 제공 한다.

QHeaderView 클래스를 상속받아 커스터마이징한 헤더(Header)를 QTableWidget 클래스의 setHorizontalHeader() 멤버 함수를 이용해 커스터마이징한 헤더를 추가 할 수 있다. 다음 예제 소스코드는 widget.cpp 소스코드이다.

```

...
#include "checkboxheader.h"

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    ui->tableWidget->setRowCount(5);
    ui->tableWidget->setColumnCount(2);

    CheckBoxHeader* header;
    header = new CheckBoxHeader(Qt::Horizontal, ui->tableWidget);

    ui->tableWidget->setHorizontalHeader(header);
    connect(header, &CheckBoxHeader::checkBoxClicked,
            this, &Widget::checkBoxClicked);
}

```

```

QStringList nameList;
nameList << "노트북" << "모바일" << "데스크탑" << "키보드" << "모니터";
for(int i = 0; i < 5 ; i++)
{
    ui->tableView->insertRow(i);
    QTableWidgetItem *dateItem = new QTableWidgetItem("2021.05.07");
    dateItem->setCheckState(Qt::Checked);

    ui->tableView->setItem(i, 0, dateItem);
    ui->tableView->setItem(i, 1, new QTableWidgetItem(nameList.at(i)));
}
}

void Widget::checkBoxClicked(bool state)
{
    for(int i = 0 ; i < 5 ; i++) {
        QTableWidgetItem *item = ui->tableView->item(i, 0);
        if(state)
            item->setCheckState(Qt::Checked);
        else
            item->setCheckState(Qt::Unchecked);
    }
}
...

```

위의 예제에서 CheckBoxHeader 클래스는 QHeaderView 클래스를 상속받아 구현한 클래스이다. 이 클래스를 헤더로 사용하기 위해서 setHorizontalHeader() 멤버 함수를 사용하면 된다.

그리고 checkBoxClicked() 는 헤더의 체크박스가 변경되면 발생한 이벤트와 연결된 Slot 함수이다. 이 Slot 함수에서는 헤더의 체크 박스의 값에 따라 첫 번째 컬럼의 체크 박스의 값을 동일하게 변경한다. 다음 예제 소스코드는 CheckBoxHeader 클래스의 헤더파일이다.

```

...
class CheckBoxHeader : public QHeaderView
{
    Q_OBJECT
public:
    CheckBoxHeader(Qt::Orientation orientation, QWidget* parent = nullptr);
    bool isChecked() const { return isChecked_; }

```

```

void setIsChecked(bool val);

signals:
    void checkBoxClicked(bool state);

protected:
    void paintSection(QPainter* painter, const QRect& rect,
                      int logicalIndex) const;

    void mousePressEvent(QMouseEvent* event);

private:
    bool isChecked_;
    void redrawCheckBox();
};

...

```

paintSection() 는 Virtual 함수이다. 헤더 영역의 마우스가 클릭 되면 이 함수를 호출해 체크 박스의 값에 따라 체크박스 상태를 변경하는 기능을 제공한다. 다음 예제는 checkboxheader.cpp 소스코드이다.

```

#include "checkboxheader.h"

CheckBoxHeader::CheckBoxHeader(Qt::Orientation orientation, QWidget* parent)
    : QHeaderView(orientation, parent)
{
    isChecked_ = true;
}

void CheckBoxHeader::paintSection(QPainter* painter, const QRect& rect,
                                  int logicalIndex) const
{
    painter->save();
    QHeaderView::paintSection(painter, rect, logicalIndex);
    painter->restore();

    if (logicalIndex == 0) {
        QStyleOptionButton option;
        option.rect = QRect(1,3,20,20);
        option.state = QStyle::State_Enabled | QStyle::State_Active;

        if (isChecked_)

```

```

        option.state |= QStyle::State_On;
    else
        option.state |= QStyle::State_Off;

    option.state |= QStyle::State_Off;
    style()->drawPrimitive(QStyle::PE_IndicatorCheckBox, &option, painter);
}
}

void CheckBoxHeader::mousePressEvent(QMouseEvent* event)
{
    Q_UNUSED(event)
    setIsChecked(!isChecked());

    emit checkBoxClicked(isChecked());
}

void CheckBoxHeader::redrawCheckBox()
{
    viewport()->update();
}

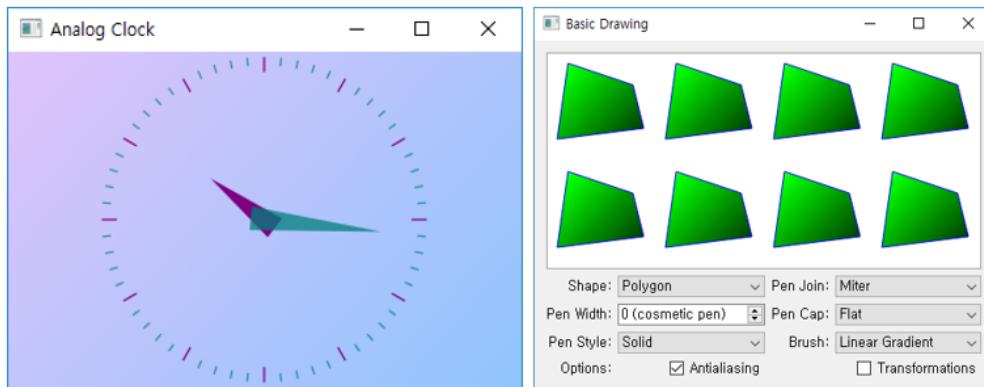
void CheckBoxHeader::setIsChecked(bool val)
{
    if (isChecked_ != val) {
        isChecked_ = val;
        redrawCheckBox();
    }
}

```

mousePressEvent() 는 헤더 위젯 영역에 마우스가 클릭 이벤트가 발생되면 호출 되는 Virtual 함수이다. 전체 예제 소스는 ch03 > 14_ModelView > 04_TableWidget 디렉토리 를 참조하면 된다.

4. QPainter 클래스를 이용한 GUI 2D 그래픽스

Qt는 GUI 위젯 영역에 QPainter 클래스를 이용해 텍스트, 선(Line), 도형을 표시할 수 있다. 기본적인 드로잉 기능 이외에 QImage, QPixmap, QPicture 클래스를 이용해 이미지파일을 위젯 영역에 표시할 수 있다. 그리고 Gradients, Transformation, Composition 등과 같은 효과를 이용해 QPainter 영역에 적용할 수 있는 기능도 제공한다.



<그림> QPainter 를 이용한 드로잉 예제

위젯 영역 내에서 QPainter 클래스를 사용하기 위해서 다음 예제에서 보는 것과 같이 QWidget 클래스의 paintEvent() Virtual 함수를 사용할 수 있다.

```
...
#include <QPainter>

class Widget : public QWidget
{
    Q_OBJECT
public:
    Widget(QWidget *parent = 0);
    ~Widget();

protected:
    virtual void paintEvent(QPaintEvent *event);
};

...
```

위의 예에서 보는 것과 같이 QWidget 클래스를 상속받아 구현 시 paintEvent() 함수를 사용할 수 있다. paintEvent() 함수는 위젯이 가려진 상태에서 모니터에 보여지게 되는

현상, 위젯의 이동, 위젯의 확대/축소 등이 발생할 때 자동으로 호출된다.

그리고 만약 위젯 영역을 다시 드로잉 해야 하는 이벤트를 발생하고자 한다면 update() 함수를 사용하면 paintEvent() 함수를 호출할 수 있다. 다음 예제는 paintEvent() 함수에서 도형을 드로잉한 예제 소스코드이다.

```
...
void Widget::paintEvent(QPaintEvent *event)
{
    QPainter painter;
    painter.begin(this);

    painter.setPen(Qt::blue);
    painter.drawLine(10, 10, 100, 40); // 선
    painter.drawRect(120, 10, 80, 80); // 사각형

    QRectF rect(230.0, 10.0, 80.0, 80.0);
    painter.drawRoundedRect(rect, 20, 20); // 둥근 사각형

    QPointF p1[3]={ QPointF(10.0, 110.0),
                    QPointF(110.0, 110.0),
                    QPointF(110.0, 190.0)};

    painter.drawPolyline(p1, 3); // 포인트 지점을 선으로 그리기

    QPointF p2[3]={ QPointF(120.0, 110.0),
                    QPointF(220.0, 110.0),
                    QPointF(220.0, 190.0)};
    painter.drawPolygon(p2, 3); // 포인트 지점으로 도형 그리기

    painter.setFont(QFont("Arial", 20)); // 폰트지정
    painter.setPen(Qt::black);
    QRect fontRect(10, 150, 220, 180); // 텍스트를 표시할 영역
    painter.drawText(fontRect, Qt::AlignCenter,
                     "Qt 개발자 커뮤니티(http://www.qt-dev.com)");
    painter.end();
}
...
```

위의 예제 소스코드에서 QPainter 클래스의 begin() 멤버 함수와 end() 함수는 실제 드로잉한 결과를 바로 그리지 않는다. 가상의 영역에 그리기 시작 한다. 즉 begin() 멤버 함수는 가상의 메모리 영역에 그리기 시작 후 end() 멤버 함수가 호출되면 begin()

과 end() 함수 사이에 드로잉한 결과를 실제 드로잉 영역에 적용한다.

예를 들어 begin() 과 end() 함수를 사용하지 않는다면 각 드로잉 요소를 그리는 함수를 호출할 때마다 실제 드로잉 영역에 적용된다. 만약 이런 단순하게 드로잉 하는 결과물인 경우는 그리는 것이 눈에 보이지 않는다.

하지만 컴퓨터 시스템이 복잡한 연산을 수행하고 있다고 하면 순차적으로 드로잉 하는 것이 보일 수 있다. 따라서 이러한 문제를 방지하기 위해 begin() 과 end() 를 사용하면 드로잉을 가상의 메모리 영역에 드로잉한 다음 end() 함수가 호출 되면 전체를 실제 그려지는 메모리 영역에 복사하므로 드로잉 되는 과정이 보이지 않게 된다.



<그림> 예제 실행 화면

QPainter 는 드로잉할 때 선과 도형 윤곽의 컬러, 두께, 스타일을 다양하게 표현할 수 있는 기능을 제공한다.

```
...
QPainter painter;
painter.begin(this);

QPen pen(Qt::blue);
pen.setWidth(4);
painter.setPen(pen);
QRect rect1(10.0, 20.0, 80.0, 50);
painter.drawEllipse(rect1);

pen.setStyle(Qt::DashLine);
painter.setPen(pen);
QRect rect2(110.0, 20.0, 80.0, 50.0);
painter.drawEllipse(rect2);
...
```



<그림> 예제 실행 화면

QPainter 클래스는 선을 그릴 때 선의 모서리 스타일을 QPen 클래스의 setJoinStyle() 멤버 함수를 이용해 지정할 수 있다.

```
...
QPen pen(Qt::blue);
pen.setWidth(20);

QPointF p1[3] = {QPointF(30.0, 80.0),
QPointF(20.0, 40.0),
QPointF(80.0, 60.0) };

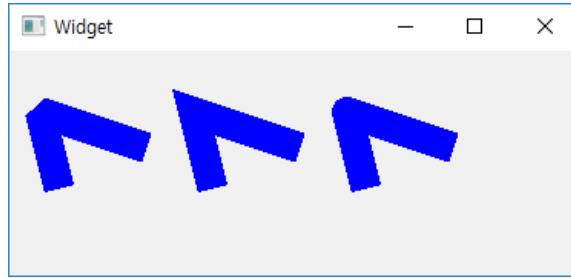
pen.setJoinStyle(Qt::BevelJoin);
painter.setPen(pen);
painter.drawPolyline(p1, 3);

QPointF p2[3] = {QPointF(130.0, 80.0),
QPointF(120.0, 40.0),
QPointF(180.0, 60.0) };

pen.setJoinStyle(Qt::MiterJoin);
painter.setPen(pen);
painter.drawPolyline(p2, 3);

QPointF p3[3] = {QPointF(230.0, 80.0),
QPointF(220.0, 40.0),
QPointF(280.0, 60.0) };

pen.setJoinStyle(Qt::RoundJoin);
painter.setPen(pen);
painter.drawPolyline(p3, 3);
...
```



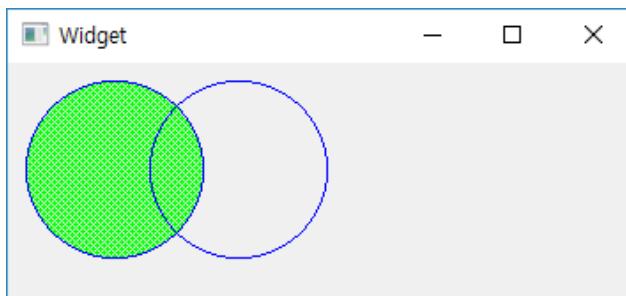
<그림> 예제 실행 화면

QPainter 클래스에서 제공하는 setBrush() 멤버 함수를 이용하면 도형의 내부에 특정 색으로 채울 수 있다.

```
QPen pen(Qt::blue);

painter.setBrush(QBrush(Qt::green, Qt::Dense3Pattern));
painter.setPen(Qt::blue);
painter.drawEllipse(10, 10, 100,100);

painter.setBrush(Qt::NoBrush);
painter.setPen(Qt::blue);
painter.drawEllipse(80, 10, 100, 100);
...
```



<그림> 예제 실행 화면

QBrush를 특정 컬러로 채우는 것 이외에도 이미지를 호출해 내부를 채울 수 있다.

```
QPixmap pixmap(":resources/qtblog.png");
int w = pixmap.width();
int h = pixmap.height();
pixmap.scaled(w, h, Qt::IgnoreAspectRatio, Qt::SmoothTransformation);
```

```

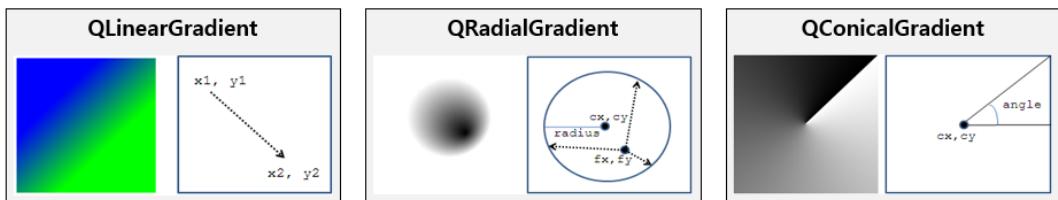
QBrush brush(pixmap);
painter.setBrush(brush);
painter.setPen(Qt::blue);
painter.drawRect(0, 0, w, h);
...

```



<그림> 예제 실행 화면

Gradients 효과를 사용하기 위해서 다음 그림에서 보는 것과 같은 Gradients 효과를 사용할 수 있다.



<그림> Gradient의 종류

다음 예제 소스코드는 QLinearGradient 클래스를 이용해 Gradients 효과를 사용한 예제이다.

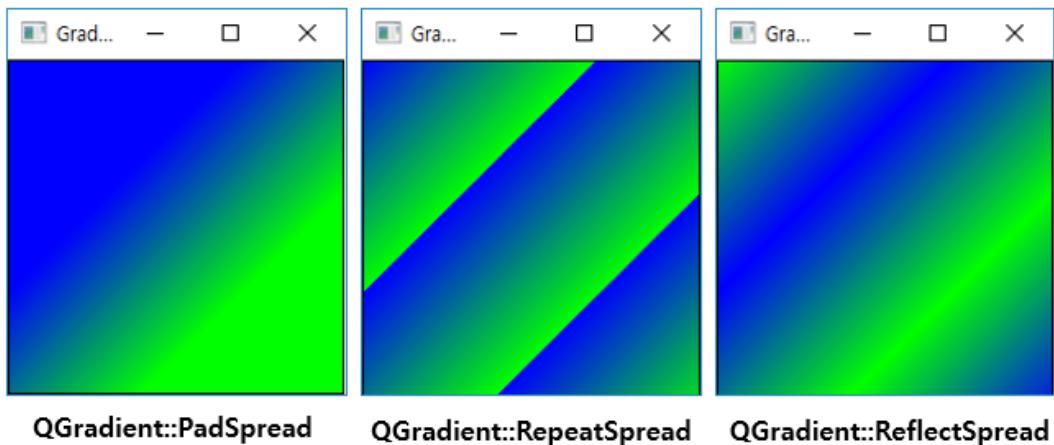
```

QLinearGradient ling(QPointF(70, 70), QPoint( 140, 140 ) );
ling.setColorAt(0, Qt::blue);
ling.setColorAt(1, Qt::green);

ling.setSpread( QGradient::PadSpread );
// ling.setSpread( QGradient::RepeatSpread );
// ling.setSpread( QGradient::ReflectSpread );

QBrush brush(ling);
painter.setBrush(brush);
painter.drawRect(0, 0, 200, 200);
...

```



QGradient::RepeatSpread

QGradient::ReflectSpread

<그림> 예제 실행 화면

Qt는 QTransform 클래스를 이용해 Scaling(확대/축소), Rotation(회전), Perspective(원근 표현) 기법을 사용할 수 있다.



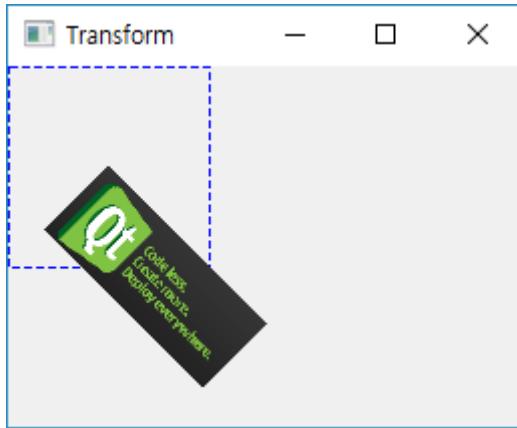
<그림> QTransform 클래스 사용 예

```
...
QImage image(":/resources/qtblog.png");

QPainter painter(this);
painter.setPen(QPen(Qt::blue, 1, Qt::DashLine));
painter.drawRect(0, 0, 100, 100);

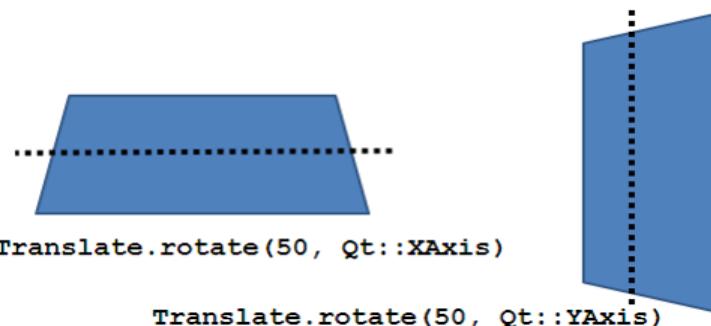
QTransform transform;
transform.translate(50, 50);
transform.rotate(45);
transform.scale(0.5, 0.5);

painter.setTransform(transform);
painter.drawImage(0, 0, image);
...
```



<그림> 예제 실행 화면

다음 예제는 Perspective 기법을 적용한 예이다. Perspective 기법을 적용하기 위한 방법으로 X축, Y축 또는 Z축으로 이동할 수 있는 기능을 제공한다.

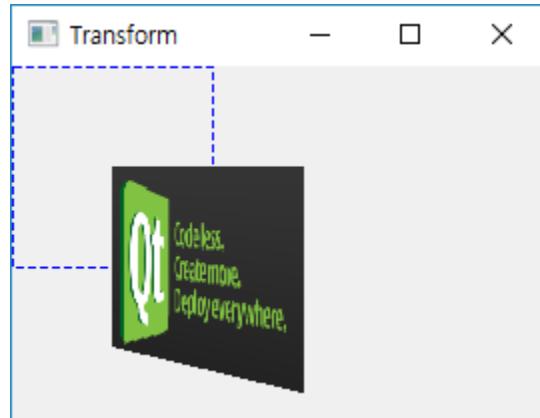


<그림> Perspective 기법

```
QPainter painter(this);
painter.setPen(QPen(Qt::blue, 1, Qt::DashLine));
painter.drawRect(0, 0, 100, 100);

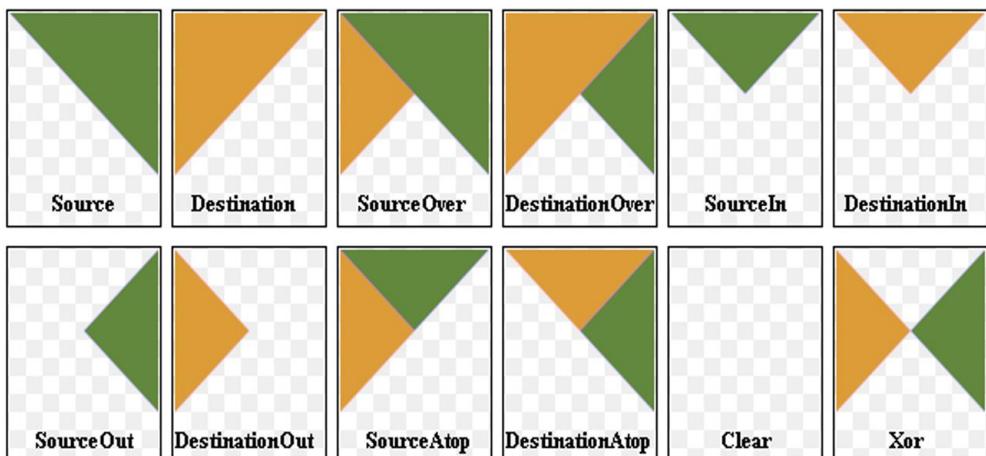
QTransform transform;
transform.translate(50, 50);
transform.rotate(70, Qt::YAxis);

painter.setTransform(transform);
painter.drawImage(0, 0, image);
...
```



<그림> 예제 실행 화면

QImage를 통해서 중첩된 영역을 Composition 기법을 사용해 다음 그림에서 보는 것과 같은 패턴을 적용할 수 있다.



<그림> Composition 종류

```
...
painter.drawImage(0, 0, destinationImage);
painter.setCompositionMode(QPainter::CompositionMode_DestinationOver);
painter.drawImage(0, 0, sourceImage);
...
```

● 커스터마이징 버튼 구현

이번 예제에서는 QPainter 와 QWidget 클래스를 이용해 QPushButton 위젯과 같은 버튼을 구현해 보도록 하자. QPainter 클래스와 사용자 입력 이벤트를 처리하기 위한 클래스를 주로 사용해 커스터마이징 위젯을 구현할 수 있다.

따라서 이번 예제에서는 QPainter 클래스와 사용자 입력을 처리하는 마우스 이벤트를 이용해 버튼을 구현해 보도록 하자. 버튼을 구현하기 위해서 다음과 같은 이미지가 필요하다. 이미지 예제 소스코드에 첨부하였으니 참조하기 바란다.



<그림> 이미지 리소스

다음 예제 소스코드에서 보는 것과 같이 커스터마이징 버튼을 구현하기 위해 ImageButton 클래스 이름으로 헤더 파일을 작성한다.

```
#ifndef IMAGEBUTTON_H
#define IMAGEBUTTON_H
#include <QWidget>
#include <QPainter>

class ImageButton : public QWidget
{
    Q_OBJECT
public:
    explicit ImageButton(QWidget *parent = 0);
    void setDisabled(bool val);

private:
    QString imgFileName;
    qint32 behaviour;
    bool disabled;

signals:
    void clicked();

protected:
    void paintEvent(QPaintEvent *event);
    virtual void enterEvent(QEvent* event);
    virtual void leaveEvent(QEvent* event);
    virtual void mousePressEvent(QMouseEvent* event);
    virtual void mouseReleaseEvent(QMouseEvent* event);
    virtual void mouseDoubleClickEvent(QMouseEvent *event);
};

#endif // IMAGEBUTTON_H
```

위의 예제에서 setDisabled() 멤버 함수는 버튼을 Disable 처리하기 위한 기능을 제공한다. disable 변수는 setDisabled에 설정된 값을 저장한다. paintEvent() 함수는 마우스의 이벤트에 따라 마우스의 이미지를 변경한다.

예를 들어 마우스가 버튼 위젯 영역에 위치하면 버튼의 이미지를 위의 그림에서 보는 것과 같이 두 번째 이미지로 변경한다. 그리고 로 위젯 영역 내에서 마우스를 클릭하면 clicked() 시그널을 발생한다. 다음 예제는 ImageButton 클래스의 구현부 소스코드이다.

```
#include "imagebutton.h"

#define BEHAVIOUR_NOMAL      0
#define BEHAVIOUR_ENTER      1
#define BEHAVIOUR_LEAVE      2
#define BEHAVIOUR_PRESS      3
#define BEHAVIOUR_RELEASE    4
#define BEHAVIOUR_DISABLE    5

ImageButton::ImageButton(QWidget *parent) :
    QWidget(parent),
    disabled(false)
{
    behaviour = BEHAVIOUR_NOMAL;

    QImage image(":/resources/normal.png");
    this->setFixedWidth(image.width());
    this->setFixedHeight(image.height());
}

void ImageButton::setDisabled(bool val)
{
    disabled = val;
    update();
}

void ImageButton::paintEvent(QPaintEvent *event)
{
    Q_UNUSED(event)

    QPainter painter;
    painter.begin(this);
```

```

    if(disabled == true) {
        imgFileName = QString(":/resources/disable.png");
    }
else
{
    if(this->behaviour == BEHAVIOUR_NOMAL)
        imgFileName = QString(":/resources/normal.png");
    else if(this->behaviour == BEHAVIOUR_ENTER)
        imgFileName = QString(":/resources/enter.png");
    else if(this->behaviour == BEHAVIOUR_LEAVE)
        imgFileName = QString(":/resources/normal.png");
    else if(this->behaviour == BEHAVIOUR_PRESS)
        imgFileName = QString(":/resources/press.png");
}

QImage image(imgFileName);
painter.drawImage(0, 0, image);
painter.end();
}

void ImageButton::enterEvent(QEvent *event)
{
    Q_UNUSED(event);
    this->behaviour = BEHAVIOUR_ENTER;
    update();
}

void ImageButton::leaveEvent(QEvent *event)
{
    Q_UNUSED(event);
    this->behaviour = BEHAVIOUR_NOMAL;
    update();
}

void ImageButton::mousePressEvent(QMouseEvent *event)
{
    Q_UNUSED(event);
    this->behaviour = BEHAVIOUR_PRESS;
    update();

    emit clicked();
}

```

```

void ImageButton::mouseReleaseEvent(QMouseEvent *event)
{
    Q_UNUSED(event);

    this->behaviour = BEHAVIOUR_ENTER;
    update();
}

void ImageButton::mouseDoubleClickEvent(QMouseEvent *event)
{
    Q_UNUSED(event)
}

```

enterEvent() 는 마우스가 위젯 영역에 위치했을 때 호출된다. leaveEvent() 는 마우스가 위젯 영역 안에서 밖으로 나아갈 때 발생한다. mousePressEvent() 는 위젯 영역 안에서 마우스를 클릭했을 때 발생하며 mouseReleaseEvent() 는 마우스 버튼을 클릭 후 해제 했을 때 발생한다. 그리고 mouseDoubleClickEvent() 는 마우스 버튼을 더블 클릭 했을 때 발생한다.

Qt에서 제공하는 마우스 이벤트 Virtual 함수에 보면 update() 함수를 사용했다. 이 함수를 사용하면 paintEvent() 함수를 호출한다. 위의 헤더 파일과 소스 파일 데로 ImageButton 클래스를 구현 하였다면 구현한 ImageButton 클래스의 오브젝트를 선언하고 사용해 보도록 하자. 다음 예제 소스코드에서 보는 것과 같이 Widget 클래스의 헤더 파일을 작성해 보도록 하자.

```

#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include "imagebutton.h"

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:

```

```

explicit Widget(QWidget *parent = 0);
~Widget();

private:
    Ui::Widget *ui;

public slots:
    void clicked();

};

#endif // WIDGET_H

```

위의 예제에서 보는 것과 같이 `clicked()` Slot 함수는 ImageButton 클래스의 시그널 이벤트가 발생했을 때 호출하는 함수이다. 다음은 Widget 클래스의 소스코드 파일을 다음과 같이 작성해 보도록 하자.

```

#include "widget.h"
#include "ui_widget.h"
#include <QHBoxLayout>
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);

    ImageButton *imgBtn1 = new ImageButton(this);
    ImageButton *imgBtn2 = new ImageButton(this);

    QHBoxLayout *hLay = new QHBoxLayout(this);
    hLay->addWidget(imgBtn1);
    hLay->addWidget(imgBtn2);

    setLayout(hLay);
    connect(imgBtn1, &ImageButton::clicked, this, &Widget::clicked);
    imgBtn2->setDisabled(true);
}

Widget::~Widget()
{
    delete ui;
}

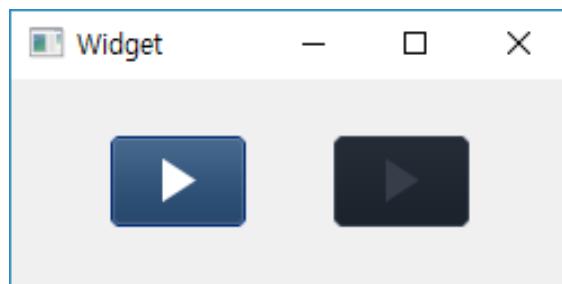
```

```

void Widget::clicked()
{
    qDebug() << Q_FUNC_INFO;
}

```

connect() 함수를 이용해 imgBtn1 오브젝트의 이벤트 발생시 Slot 함수와 연결한다. 그리고 두 번째 imgBtn2 는 Disable 상태로 만들기 위해 ImageButton 클래스에서 구현한 setDisabled() 함수를 사용하였다. 아래 그림에서 보는 것과 같이 작성한 예제를 빌드하고 실행 보도록 하자. 전체 예제 소스코드는 Ch04 > 06_CustomButton 디렉토리를 참조하면 된다.



<그림> 예제 실행 화면

● 이미지 스케일 비율 유지 기능 예제 구현

이번 예제에서는 PNG 포맷의 이미지 사진 파일을 위젯 영역에 랜더링(표시)하는 예제이다. 이 예제에서는 랜더링 하고 이미지의 크기가 위젯의 크기가 변경됨에 따라 이미지 원본의 유지하며 이미지가 확대 되거나 축소되는 이미지이다.

예를 들어 이미지의 크기의 비율이 위젯의 영역 크기에 비례해서 이미지가 확대되거나 축소된다. 그리고 이미지 크기 비율에 따라 이미지가 표시되지 않은 영역은 아래 그림에서 보는 것과 같이 검은색으로 칠하고 이미지는 중앙에 배치한다.



<그림> 예제 실행 화면

위의 그림에서 보는 것과 같이 위젯이 크기가 커질 때 이미지가 비율을 계산해 위젯의 크기에 비례해 랜더링 하는 예제이다. 다음 예제 소스코드는 paintEvent() 함수이다. 일전에 설명했던 것과 같이 윈도우 크기가 변경되면 paintEvent() 함수가 호출되기 때문에 paintEvent() 함수를 호출하지 않아도 자동으로 호출된다.

```
...
void Widget::paintEvent(QPaintEvent *event)
{
    Q_UNUSED(event)

    QPainter painter;
    painter.begin(this);

    int w = this->window()->width();
    int h = this->window()->height();

    painter.setPen(QColor(0, 0, 0));
    painter.fillRect(0, 0, w, h, Qt::black);

    QPixmap imgPixmap = QPixmap(":/images/picture.png")
                           .scaled(w, h, Qt::KeepAspectRatio);

    int imgWidth = imgPixmap.width();
    int imgHeight = imgPixmap.height();

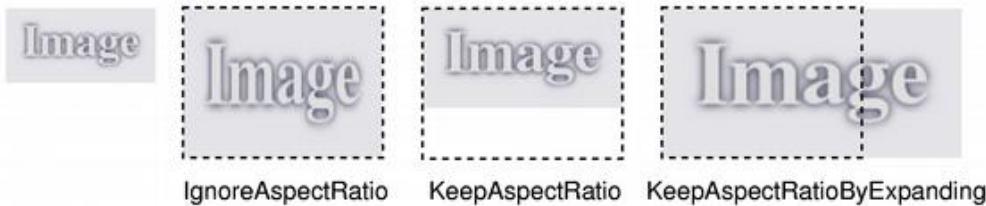
    int xPos = 0;
    int yPos = 0;

    if(w > imgPixmap.width())
        xPos = (w - imgWidth) / 2;
    else if( h > imgPixmap.height())
        yPos = (h - imgHeight) / 2;

    painter.drawPixmap(xPos, yPos, imgPixmap);
    painter.end();
}
```

QPixmap 은 이미지 파일을 디코딩하는 기능을 제공한다. QPixmap 클래스의 scaled() 함수는 첫 번째 인자(가로), 두 번째 인자(세로) 크기를 지정한다. 그리고 세 번째 인자는 이미지를 랜더링 시 어떤 방식을 사용할지 결정하는 방식이다. 세 번째 인자에 사용

할 수 있는 방식은 다음 표에서 보는 것과 같이 하나를 선택할 수 있다.



<그림> 이미지 비율 표시 방식

Constant	Value	설명
Qt::IgnoreAspectRatio	0	이미지 크기 비율에 상관 없이 가로와 세로 크기에 가득 차게 표시
Qt::KeepAspectRatio	1	가로와 세로 크기에 상관 없이 이미지 비율 유지
Qt::KeepAspectRatioByExpanding	2	가로 세로 크기 와 이미지 크기 비율에 상관 없이 이미지 원본 크기에 맞게 표시

QPainter 클래스의 drawPixmap() 멤버 함수는 QPixmap 클래스로 랜더링한 이미지를 QPainter 영역에 표시하는 기능을 제공한다. 첫 번째 인자와 두 번째 인자는 X,Y 시작 좌표이다. 마지막 세번째 인자는 QPixmap 클래스의 오브젝트를 명시하면 된다. 이 예제의 전체 소스는 Ch04 > 07_ScaledImageRender 디렉토리를 참조하면 된다.

● 크로마키(Chromakey) 영상처리 기법을 이용한 이미지 합성 구현

이번 예제는 크로마키 (Chromakey) 영상처리 기법을 이용해 처리하는 방법에 대해서 구현해 보도록 하자. 여기서 말하는 크로마키란 배경의 색을 필터링해 배경을 다른 이미지로 대체하는 방법을 말한다. 예를 들어 날씨 뉴스 방송의 배경을 다른 이미지나 영상으로 대체하는 영상을 많이 본적이 있을 것이다. 이번 예에서는 사진 이미지 포맷의 배경을 다른 이미지로 변경해 보도록 하자.

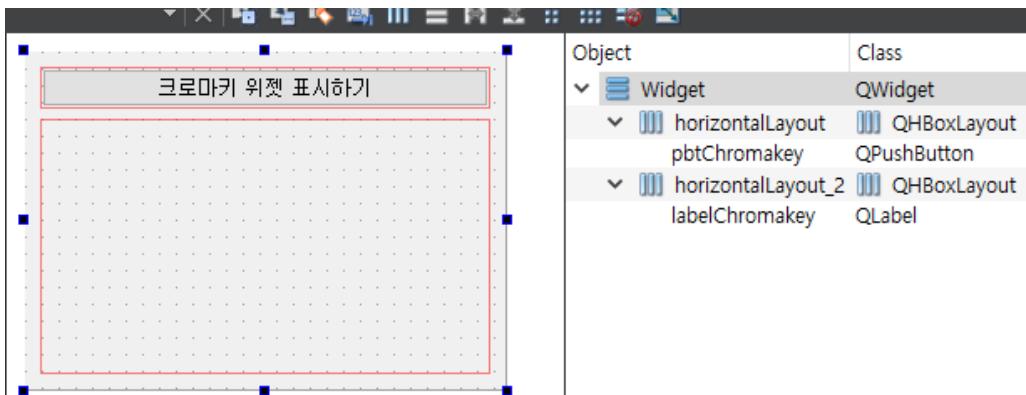


<그림> 크로마키에 상용할 이미지와 결과 이미지

위의 그림에서 SOURCE 이미지상에서 배경은 녹색이다. SOURCE 이미지 사진에서 배경색을 TARGET 이미지로 대체한다. 그러기 위해서는 SOURCE 이미지의 사람 부분과 배경을 분리할 수 있는 방법이 필요하다. SOURCE 이미지의 배경은 녹색이다.

그렇기 때문에 사람과 배경을 분리하기 위해서 SOURCE 이미지의 배경색인 녹색의 Threshold 값을 찾으면 사람과 녹색인 배경을 분리할 수 있다. 그리고 분리한 배경을 TARGET 이미지로 변경하면 RESULT 이미지와 같이 된다.

QWidget 기반의 새로운 프로젝트를 생성하고 다음과 같이 GUI상에 위젯을 배치한다.



<그림> 위젯 배치

위의 그림에서 보는 것과 같이 QPushButton 과 QLabel 을 배치한다. 그리고 다음 예제에서 보는 것과 같이 ImageProcessing 클래스를 프로젝트에 추가한 다음 헤더 파일을 다음과 같이 작성한다.

```
#ifndef IMAGEPROCESSING_H
#define IMAGEPROCESSING_H

#include <QImage>
#include <QColor>

class ImageProcessing
{
public:
    ImageProcessing(int width, int height, int dataSize);
    void chromakeyProcess(QImage& sourceImage,
                          QImage& targetImage,
                          QImage& resultImage);
private:
    int imageWidth;
    int imageHeight;
```

```

    int imageDataSize;
};

#endif // IMAGEPROCESSING_H

```

ImageProcessing 클래스의 생성자 첫 번째 와 두 번째 인자는 크로마키 처리를 할 크기를 인자로 전달한다. 그리고 세 번째 인자는 이미지 사이즈를 넘겨준다. 이미지 사이즈 크기를 세 번째 인자에 넘겨줄 때 가로 크기와 세로 크기를 곱한 값에 4를 곱해야 한다.

왜냐하면 여기서는 RGB32를 사용한다. 즉 RGBA 는 각 4개의 값을 사용하며 1개의 값은 1Byte 크기 이므로 전체 크기는 다음과 같이 계산하면 된다.

가로 크기 x 세로 크기 x RGBA(4) = 이미지 크기

이미지 크기의 단위는 Pixel 이다. 크로마키 기법을 사용할 때 주의해야 하는 것 중 하나는 SOURCE 이미지의 크기, TARGET 이미지의 크기 그리고 RESULT 이미지 크기가 동일해야 한다.

ImageProcessing 클래스에서 구현할 chromakeyProcess() 멤버 함수는 SOURCE 이미지와 TARGET 이미지를 크로마키 처리하여 결과를 세 번째 인자에 넘겨준다. 다음 예제 소스코드는 ImageProcessing 클래스 헤더의 구현 소스코드이다.

```

#include "imageprocessing.h"
#include <QDebug>

ImageProcessing::ImageProcessing(int width, int height, int dataSize)
{
    this->imageWidth = width;
    this->imageHeight = height;
    this->imageDataSize = dataSize;
}

void ImageProcessing::chromakeyProcess(QImage& sourceImage,
                                         QImage& targetImage,
                                         QImage& resultImage)
{
    uchar *pSourceData = sourceImage.bits();
    uchar *pTargetData = targetImage.bits();
    uchar *pResultData = resultImage.bits();

    QColor maskColor = QColor::fromRgb(sourceImage.pixel(1,1));

```

```

int kred    = maskColor.red();
int kgreen  = maskColor.green();
int kblue   = maskColor.blue();

int sPixRed, sPixGreen, sPixBlue;

for (int inc = 0; inc < this->imageDataSize ; inc += 4)
{
    sPixRed    = pSourceData[inc+2];
    sPixGreen = pSourceData[inc+1];
    sPixBlue  = pSourceData[inc];

    if((abs(kred - sPixRed) + abs(kgreen - sPixGreen) +
       abs(kblue - sPixBlue)) / 5 < 22 )
    {
        pResultData[inc+2] = pTargetData[inc+2];
        pResultData[inc+1] = pTargetData[inc+1];
        pResultData[inc]   = pTargetData[inc+0];
    }
    else
    {
        pResultData[inc+2] = pSourceData[inc+2];
        pResultData[inc+1] = pSourceData[inc+1];
        pResultData[inc]   = pSourceData[inc+0];
    }
}
}

```

chromakeyProcess() 멤버 함수의 인자로 QImage 를 사용하였다. QPixmap 과 같은 다양한 클래스가 있음에도 불구하고 QImage 클래스를 사용한 이유는 이미지의 각 픽셀의 RGBA 값을 바로 접근할 수 있기 때문이다.

각 픽셀의 RGBA 에서 4개의 값이 의미 하는 값은 R은 Red, G는 Green, B는 Blue 그리고 A는 Alpha 로 투명 값을 의미한다. QImage 클래스는 RGBA 외에도 다양한 포맷을 제공한다. 또한 QImage 클래스에서 사용하는 RGBA 의 포맷에 따라 BGRA, RGBA, ABGR 등의 순서로 변경될 수 있기 때문에 각 포맷에 따른 RGBA 의 순서에 따라 RGBA 의 순서를 정확히 확인해야 한다.

더불어 QImage 는 Alpha 값을 사용하지 않는 포맷도 제공하며 RGB값의 한 개의 값만 제공할 수 있는 포맷도 제공한다. 다음은 Widget 클래스의 헤더파일 소스이다.

```

#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include "imageprocessing.h"

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();

private:
    ImageProcessing *imgProcess;

    QImage sourceQImage;
    QImage targetQImage;
    QImage resultQImage;

    int sourceQImageWidth;
    int sourceQImageHeight;
    int sourceQImageDataSize;

private slots:
    void slotChromakey();

private:
    Ui::Widget *ui;
};

#endif // WIDGET_H

```

QImage 클래스의 sourceQImage 오브젝트에는 SOURCE 이미지, targetQImage 는 TARGET 이미지 그리고 resultQImage 는 크로마키 처리를 완료한 이미지 결과를 ImageProcessing 클래스로부터 결과를 저장할 오브젝트이다. 다음 예제 소스코드는 Widget 클래스의 구현 소스코드이다.

```

...
#define IMGSOURCE(":/images/jana_480p.png"
#define IMGTARGET(":/images/target_480p.png"

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pbtChromakey, SIGNAL(clicked()), this, SLOT(slotChromakey()));

    sourceQImage = QImage(IMGSOURCE);
    targetQImage = QImage(IMGTARGET);
    resultQImage = QImage(targetQImage.width(),
                          targetQImage.height(),
                          QImage::Format_RGB32);

    sourceQImageWidth = targetQImage.width();
    sourceQImageHeight = targetQImage.height();
    sourceQImageDataSize = targetQImage.width() * targetQImage.height() * 4;

    imgProcess = new ImageProcessing(sourceQImageWidth,
                                    sourceQImageHeight,
                                    sourceQImageDataSize);
}

void Widget::slotChromakey()
{
    imgProcess->chromakeyProcess(sourceQImage, targetQImage, resultQImage);

    int width = ui->labelChromakey->width();
    int height = ui->labelChromakey->height();
    QPixmap drawPixmap = QPixmap::fromImage(resultQImage).scaled(width, height);

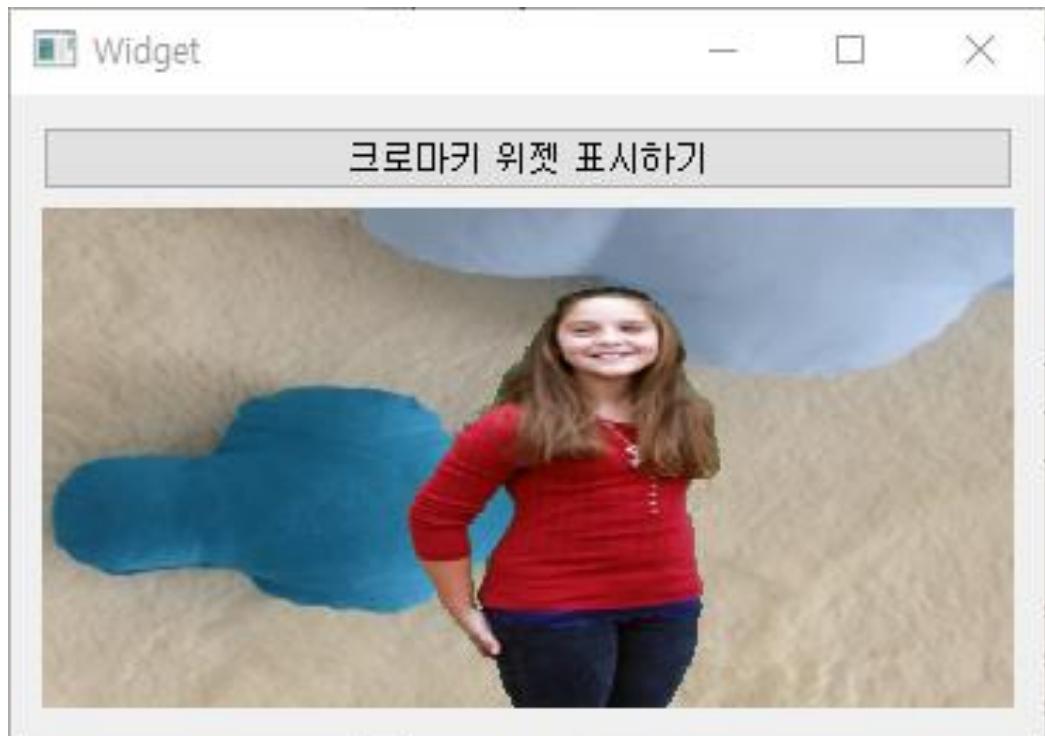
    ui->labelChromakey->setPixmap(drawPixmap);
}

Widget::~Widget()
{
    delete ui;
}
...

```

이전에 설명한 QImage 클래스의 세 번째 인자는 QImage 가 사용할 RGB 포맷의 종류

를 지정한다. 여기서는 QImage::Format_RGB32 를 사용하였다. 이 포맷은 각 픽셀 당 RGBA 값을 사용하는 포맷을 의미한다. QImage 클래스에서 제공하는 포맷의 종류는 Assistant 도움말을 참조하면 QImage 에서 사용 가능한 다양한 포맷을 확인할 수 있다. 위의 예제에서 보는 것과 같이 소스코드 작성은 완료하고 빌드 후 실행해보도록 하자. 작성한 예제를 실행하고 [크로마키 위젯 표시하기] 버튼을 클릭해 결과를 확인해 보도록 하자.



<그림> 예제 실행 화면

위의 그림에서 보는 것과 같이 QLabel 위젯 영역에 Result 이미지를 표시하였다. 예제의 전체 소스코드와 리소스 이미지 파일은 Ch04 > 08_Chromakey 디렉토리를 참조하면 된다.

5. Qt Graphics View Framework

1920 x 1080 픽셀을 표시할 수 있는 1080p 해상도 모니터를 사용한다고 가정해보자. 그런데 우리가 개발해야 하는 어플리케이션은 네비게이션과 같은 맵(Map) 상에 건물 등과 같은 물체 수백 개를 표시하고 맵의 크기는 모니터 해상도의 2배인 3840 x 2160 픽셀 크기의 맵이라고 해보자.

3840 x 2160 픽셀 크기의 맵의 크기를 1080p 해상도 모니터에 표시하는 것은 그리 어렵지 않다. 맵의 현재 이미지를 1080p 해상도로 축소하여 QPainter 와 같은 영역에 표시하면 된다.

하지만 맵이 GPS 에서 사용하는 표준 GPS좌표계(WGS84)와 맵의 각 픽셀 좌표가 매핑되었 있다면 어떻게 축소를 하겠는가?

이는 매우 어렵게 구현해야 할 것이다. 또한 맵 상에 표시되어 있는 각 물체의 좌표가 GSP 좌표계와 동일하다면 어떻게 축소 및 확대를 구현 하겠는가? 시간을 들여 충분히 구현이 가능하지만 며칠 만에 쉽게 구현할 사항은 아니다.

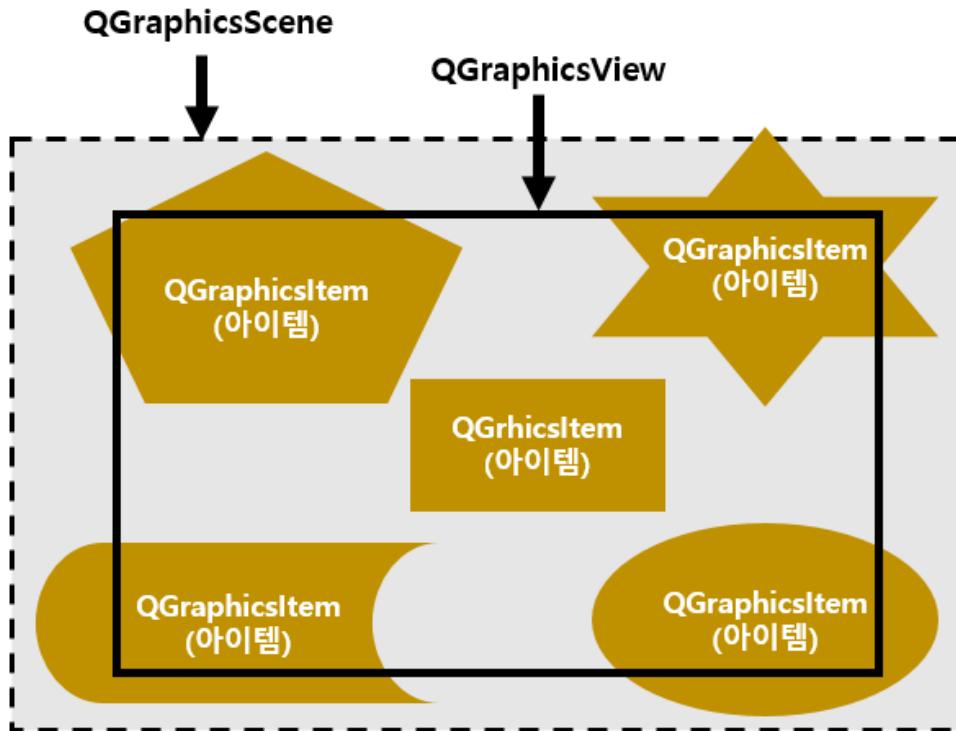
네비게이션 맵을 예로 들었지만 이 외에도 게임을 예로 들어보자. 맵 상에 주인공을 게임에서 사용하는 좌표에 표시하고 주인공 이외에 다른 오브젝트(예로 몬스터?) 들을 표시해야 한다.

그리고 확대 및 축소가 가능해야 한다면 어떻게 구현하겠는가? 여러가지 방법이 있겠지만 쉽게 구현하기가 까다로울 것이다. Qt는 이러한 상황에서 사용할 수 있도록 Graphics View Framework를 제공한다. Graphics View Framework 는 요소로 다음과 같이 3가지를 제공한다.

<표> Qt Graphics View Framework 의 3가지 요소

요소	설명
QGrphicsView	시작적으로 표시가 가능한 영역. 예를 들어 GUI에서 특정 영역. 이 클래스는 GUI 상에서 실제 크기의 영역이다.
QGraphcisScene	QGraphcisScene 클래스는 논리 적인 영역 이다. QGrphicsView 영역 안에 표시된다.
QGraphicsItem	QGraphcisScene 클래스 오브젝트 상에 표시된다. 예를 들어 맵 상에 표시되는 오브젝트와 같은 개념으로 사용된다.

다음 그림은 Qt Graphics View Framework 의 3가지 요소를 도식화한 구조이다.



<그림> Graphics View Framework 의 구조

위의 그림에서 보는 것과 같이 Graphics View Framework 는 QGraphicsView, QGraphicsScene 그리고 QGraphicsItem 을 사용한다. 예를 들어 어플리케이션에서 실제 보여지는 영역은 QGraphicsView 영역이다. GUI 위젯인 QGraphicsView 는 위젯의 영역이라고 봐도 무방하다.

QGraphicsScene 은 가상의 영역이다. 위의 그림에서 보는것과 같이 QGraphicsScene 은 QGraphicsView 의 영역보다 크거나 작을 수 있다.

QGraphicsItem 은 QGraphicsScene 상에 위치한다. 예를 들어 맵 상에서 물체(건물 등) 맵에 위치 하듯이 생성한 QGraphicsItem 들은 모두 QGraphicsScene 상에 등록 해야 한다. QGraphicsItem 클래스는 QPainter 와 같이 QGraphicsItem 영역에 텍스트, 이미지 그리고 도형 등을 표시할 수 있는 paint() Virtual함수를 제공한다. 다음은 QGraphicsItem 을 상속받아 구현한 예제 소스 코드 이다.

```
...
class SimpleItem : public QGraphicsItem
{
public:
```

```

QRectF boundingRect() const override
{
    qreal penWidth = 1;
    return QRectF(-10 - penWidth / 2, -10 - penWidth / 2,
                  20 + penWidth, 20 + penWidth);
}

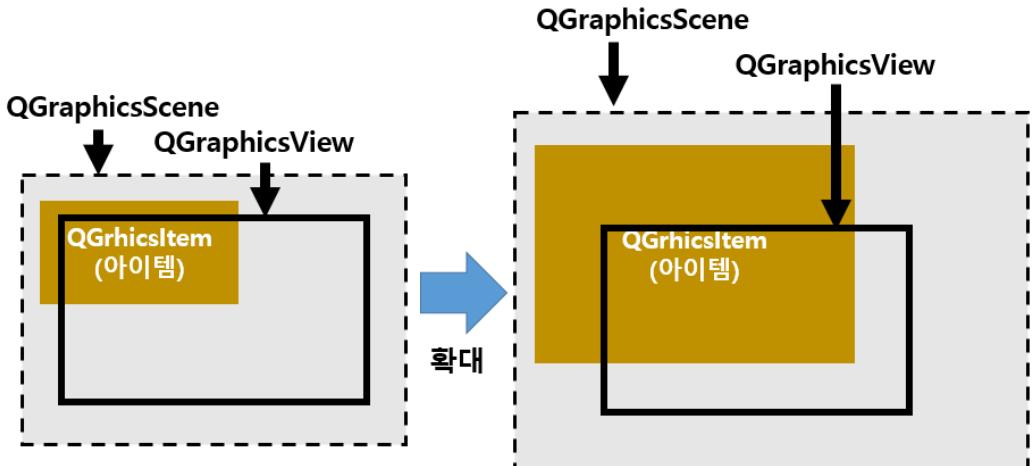
void paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
           QWidget *widget) override
{
    painter->drawRoundedRect(-10, -10, 20, 20, 5, 5);
}
};

...

```

위의 예제 소스코드에서 paint() 함수는 QWidget의 paintEvent() 함수와 동일한 기능을 제공한다. QGraphicsItem 클래스 외에도 QGraphicsRectItem, QGraphicsTextItemmm, QGraphicsPixmapItem 등 다양한 형태의 아이템 클래스를 제공한다.

Graphics View Framework는 QGraphicsScene을 QGraphicsView 영역에 맞게 확대/축소할 수 있는 기능을 제공한다.

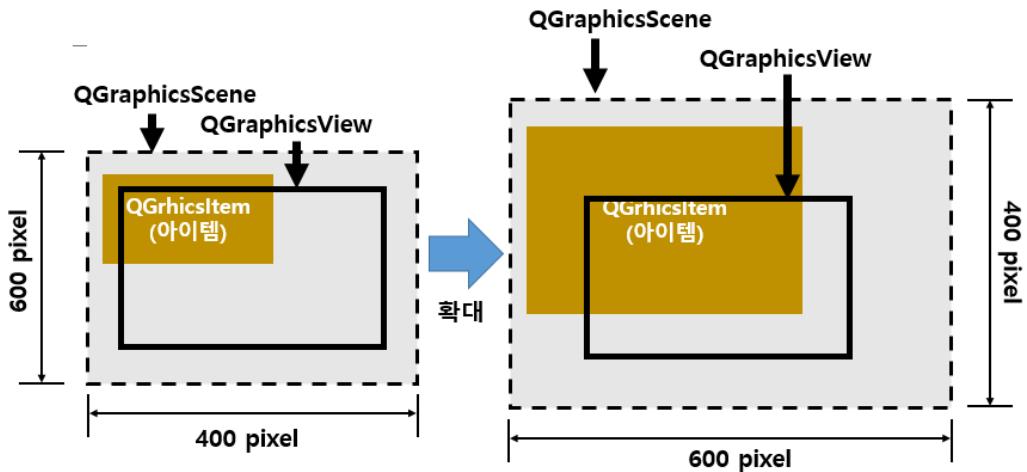


<그림> 확대 했을 때의 도식화한 그림

위의 그림과 같이 Qt Graphics View Framework는 확대 시 QGraphicsView의 크기는 변경되지 않는 상태에서 QGraphicsScene의 크기가 논리적으로 커진다.

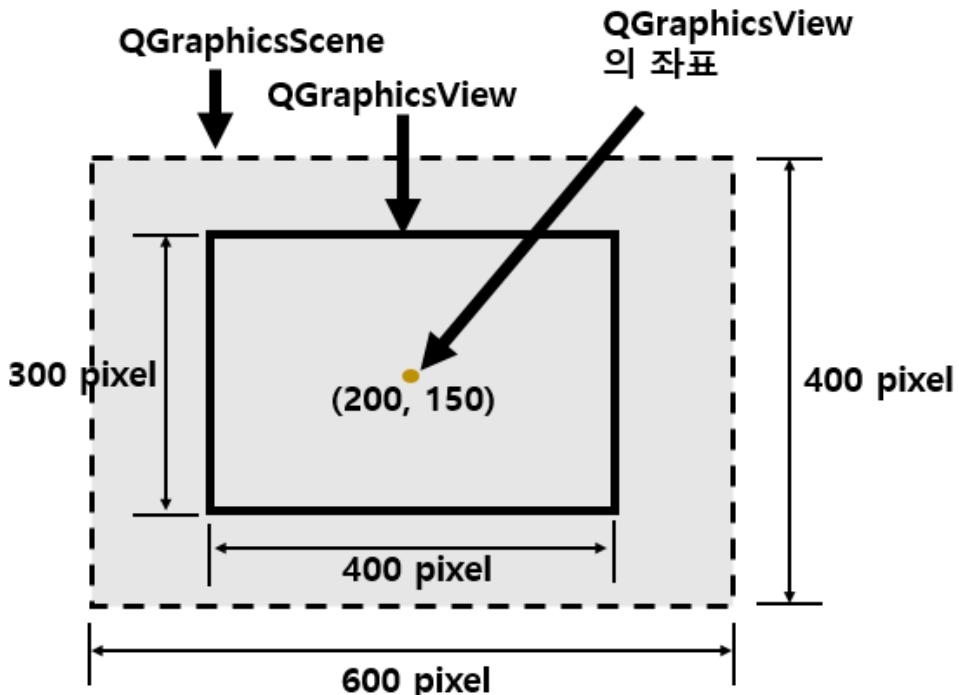
하지만 그렇다고 해서 실제 QGraphicsScene의 가로 와 세로 크기가 커지지는 않는다. 또한 QGraphicsScene에 등록된 QGraphicsItem도 마찬가지이다. 다음 그림에서 보는것과 같이 확대 시 QGraphicsScene의 원래 크기는 변경되지 않으며 축소 시에도

마찬가지다.



<그림> 확대 시 QGraphicsScene 의 크기

위의 그림에서 보는 것과 같이 확대 시 실제 QGraphicsScene 의 크기가 변경되지 않는다. 이는 축소 시에도 마찬가지이다. 그리고 화면에서 보는 QGraphicsView 와 가상의 QGraphicsScene 의 좌표계가 서로 다를 수 있다. 예를 들어 다음 그림에서처럼 QGraphicsView 의 X, Y 좌표가 200, 150 이라고 가정해 보자.



<그림> QGraphicsView 의 좌표계와 QGraphicsScene 의 좌표 관계

위의 그림에서 보는 것과 같이 QGraphicsView 의 가로와 세로 크기는 400, 300 이며 QGraphicsView 중심 좌표에 있는 좌표가 200, 150 이라고 가정해 보자. 위의 그림에서 보는 것과 같이 200, 150 좌표는 QGraphicsView 의 좌표일 뿐 QGraphicsScene 의 좌표가 아니다. 위의 그림에서 예로 QGraphicsView 의 200, 150 의 좌표 점은 QGraphicsScene 에서 대략 300, 200 일 것이다.

이와 같이 Graphics View Framework 에서 좌표를 변환 하는 함수를 제공한다. 위의 그림에서 보는 것과 같이 다양한 QGraphicsView 의 좌표를 QGraphicsScene 좌표로 바꾸거나 반대로 QGraphicsScene 의 좌표를 QGraphicsView 의 좌표를 변경하는 함수를 제공한다.

예를 들어 mapFromScene() 멤버 함수와 mapToScene() 등과 같은 다양한 함수를 제공한다. 또한 QGraphicsScene 에 등록된 아이템의 좌표를 QGraphicsView 의 좌표로 변경하는 기능도 제공한다. 다음 예제 소스코드 에서 보는 것과 같이 QGraphicsView 를 QGraphicsScene 과 매핑하기 위해서 다음과 같이 사용하면 된다.

```
...
QGraphicsScene scene;
myPopulateScene(&scene);

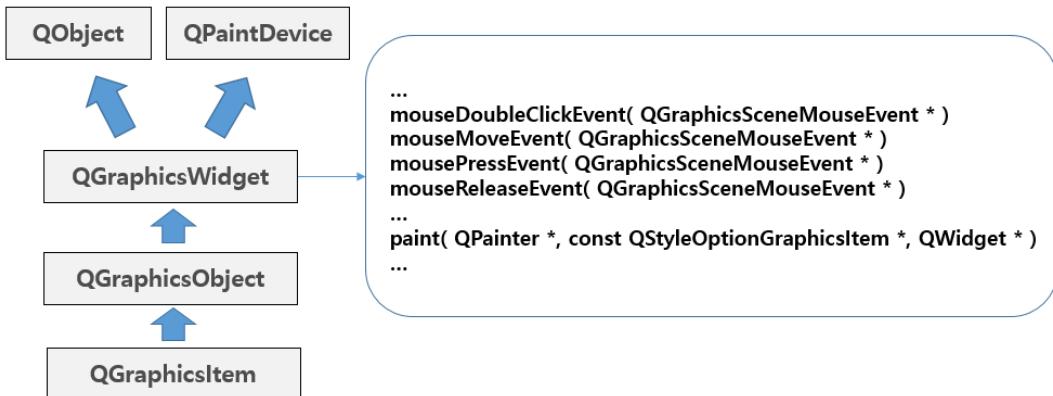
QGraphicsItem *item;
scene.addItem(&item);

QGraphicsView view(&scene);
view.show();
...
```

QGraphicisScene 상에 QGraphicsItem 을 등록하기 위해서 QGraphicsScene 클래스에서 제공하는 addItem() 멤버 함수를 사용하면 된다.

QGraphicsItem 클래스는 일전에 설명한 것 과 같이 paint() Virtual 함수를 이용해 영역에 2D 그래픽스 요소를 사용해 도형을 드로잉 하거나 이미지를 표시할 수 있다.

또한 QGraphicsItem 은 QGraphicsWidget 을 상속받아 구현되었기 때문에 다음 그림에서 보는 것과 같이 QGraphicsItem 영역에서 사용자 입력과 같은 이벤트를 처리할 수 있는 Virtual 함수를 사용 할 수 있다.



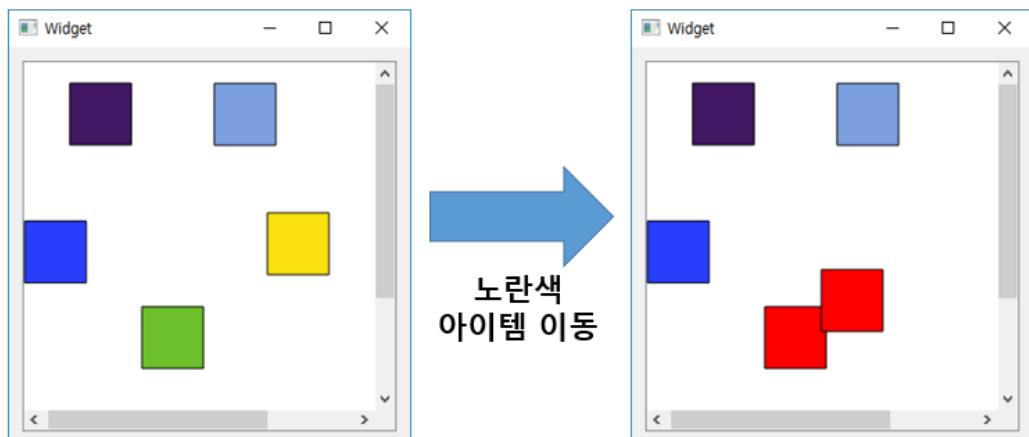
<그림> QGraphicsItem 클래스의 상속 관계

Graphics View Framework 는 확대/축소, 회전, 수많은 아이템을 빠르게 상호작용 할 수 있도록 내부적으로 BSP(Binary Space Partitioning) 알고리즘을 사용 한다. 따라서 수십 만개 이상의 아이템도 사용가능하다.

- QGraphicsItem 의 충돌 감지 구현 예제

이번에 다룰 예제는 QGraphicsItem 클래스로 생성한 Shape 아이템이 충돌하는 경우 감지하는 기능을 구현한 예제이다. QGraphicsScene 에 등록된 Shape 아이템을 마우스로 드래그 할 수 있다.

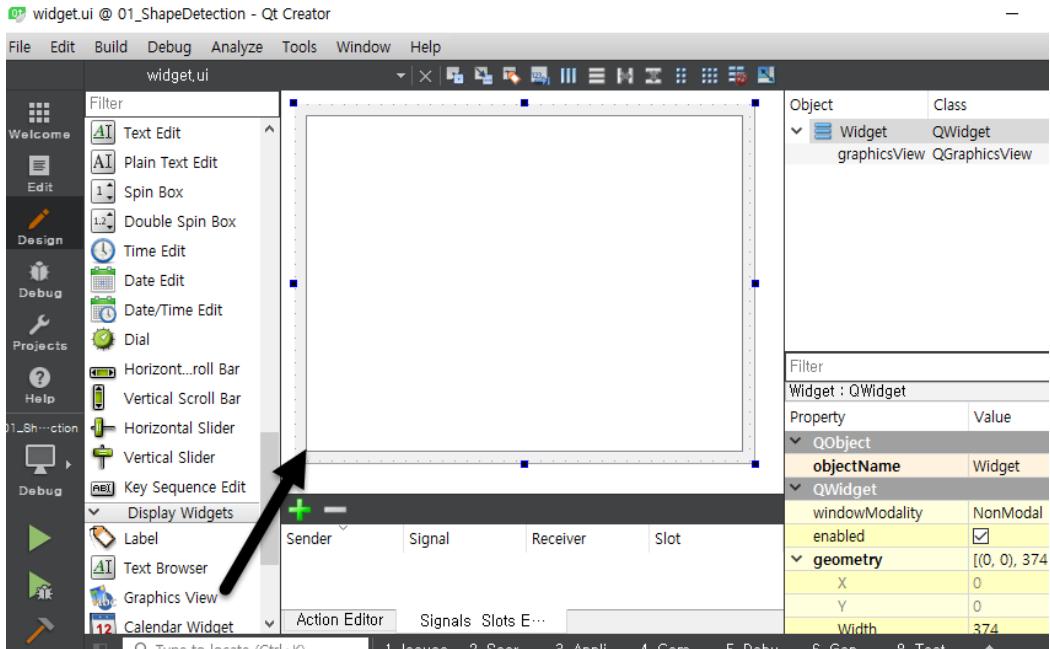
드래그 한 Shape 아이템이 다른 Shape 아이템과 충돌하면 충돌된 아이템의 내부 컬러를 빨간색으로 변경한다. 다음 그림은 예제를 실행한 화면이다.



<그림> 예제 실행 화면

위의 그림에서 보는 것과 같이 Shape 아이템 중 하나를 마우스로 드래그 하여 다른 아이템과 겹쳐지면 충돌이 발생한다. 그리고 충돌한 두개의 아이템의 색을 빨간색으로

변경한다. QWidget 기반의 새로운 프로젝트를 생성한다. 그리고 GUI 폼에 다음과 같이 위젯을 배치한다.



<그림> GUI 폼 화면

위의 그림에서 보는 것과 같이 좌측 위젯 리스트에서 [Graphics View] 아이템을 GUI 상에 배치한다. 그리고 Shape라는 새로운 클래스를 프로젝트에 생성한다. 다음 예제에서 보는 것과 같이 Shape 클래스의 헤더 파일을 작성한다.

```
#ifndef SHAPE_H
#define SHAPE_H

#include <QObject>
#include <QGraphicsItem>

class Shape : public QGraphicsItem
{
public:
    Shape();
    QRectF boundingRect() const override;
    QPainterPath shape() const override;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
               QWidget *widget) override;
protected:
    void advance(int step) override;
```

```

void mouseMoveEvent(
    QGraphicsSceneMouseEvent *event) override;

private:
    QColor color;
};

#endif // SHAPE_H

```

Shape 클래스는 QGraphicsItem 클래스를 상속받아 구현한다. boundingRect() 재정의 함수는 QGraphicsScene 내부에서 사용되는 함수이다. 이 함수에서는 이 아이템의 시작 점 x, y, width, height 값을 넘겨주면 된다.

shape() 재 정의 함수는 충돌을 감지하는 함수이다. 이 함수에서는 충돌이 감지되는 영역을 지정하면 된다. QGraphicsScene 내에서 등록된 아이템이 충돌이 일어나면 이 함수에 의해 등록된 영역을 감지해 충돌을 감지해 준다. paint() 함수는 QGraphicsItem 영역에 도형이나 원하는 요소를 드로잉 할 수 있다. QWidget 클래스의 paintEvent() 함수와 동일한 기능을 제공한다.

advance 함수는 Widget 클래스에서 타이머를 사용하였다. 타이머가 호출되면 이 함수가 자동으로 호출된다. 그리고 mouseMoveEvent() 함수는 QGraphicsScene 내에서 특정 QGraphicsItem을 마우스로 드래그 하기 위한 기능을 제공한다. 다음 예제 소스코드는 Shape 클래스의 함수 구현 부이다.

```

#include "shape.h"

#include <QGraphicsScene>
#include <QPainter>
#include <QRandomeGenerator>
#include <QStyleOption>
#include <QGraphicsSceneMouseEvent>
#include <QDebug>

Shape::Shape()
{
    setFlags(QGraphicsItem::ItemIsSelectable | QGraphicsItem::ItemIsMovable);
    color = QColor(QRandomGenerator::global()->bounded(256),
                  QRandomGenerator::global()->bounded(256),
                  QRandomGenerator::global()->bounded(256));
}

```

```

QRectF Shape::boundingRect() const
{
    return QRectF(0, 0, 50, 50);
}

QPainterPath Shape::shape() const
{
    QPainterPath path;
    path.addRect(0, 0, 50, 50);
    return path;
}

void Shape::paint(QPainter *painter, const QStyleOptionGraphicsItem *,
                  QWidget *)
{
    if(scene()->collidingItems(this).isEmpty()) {
        painter->setBrush(color);
    } else {
        painter->setBrush(QColor(Qt::red));
    }

    painter->drawRect(0, 0, 50, 50);
}

void Shape::advance(int step)
{
    if (!step)
        return;
    update();
}

void Shape::mouseMoveEvent(QGraphicsSceneMouseEvent *event)
{
    QPointF eventpos = event->pos();
    QPointF shapePos = this->pos();

    QPointF wPos(eventpos.x() + shapePos.x() - 25,
                 eventpos.y() + shapePos.y() - 25);
    setPos(wPos);
    update();
    QGraphicsItem::mouseMoveEvent(event);
}

```

생성자에서는 setFlags() 함수의 지자는 아이템을 움직일 수 있도록 하기 위한 값을 설정할 수 있다. color 변수는 Shape 내의 Brush 의 컬러로 사용된다.

paint() 함수에서 scene()->collidingItems(this).isEampty() 함수는 아이템이 현재 충돌했는지 알 수 있다. 따라서 충돌하지 않은 상태라면 생성자에서 color 변수 값을 사용한다. 그렇지 않고 충돌이 감지 되면 Qt::red 상수로 지정된 빨간색을 Brush 로 사용한다.

다음 예제 소스코드는 Shape 클래스를 사용하기 위한 Widget 클래스이다. 다음 예제에서 보는 것과 같이 헤더 파일을 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QGraphicsScene>
#include <QTimer>
#include "shape.h"

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:

    explicit Widget(QWidget *parent = nullptr);
    ~Widget();
    void timerStart();

private:
    Ui::Widget *ui;
    QGraphicsScene *m_scene;
    Shape      *m_shape[5];
    QTimer     m_timer;
};

#endif // WIDGET_H
```

위의 클래스 생성자에서 QGraphicsScene 을 생성한다. 생성한 QGraphicsScene 상에서 QGraphicsItem 을 등록한다. 그리고 QGraphicsView 에 QGraphicsScene을 setScene() 함수를 이용해 등록한다. 다음 예제 소스코드는 main.cpp 이다. 여기서는 QTimer 클래스의 오브젝트를 start() 멤버 함수를 호출 한다.

```
#include "widget.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    Widget w;
    w.show();
    w.timerStart();

    return a.exec();
}
```

지금까지 작성한 예제를 실행한 다음 마우스로 QGraphicsItem 을 드래그 해보도록 하자. 만약 충돌이 일어나면 충돌한 Shape는 내부 색이 빨간색으로 변경된다. 충돌이 해제되면 원래의 색으로 변경된다. 전체 소스는 Ch05 > 01_ShapeDetection 디렉토리를 참조하면 된다.

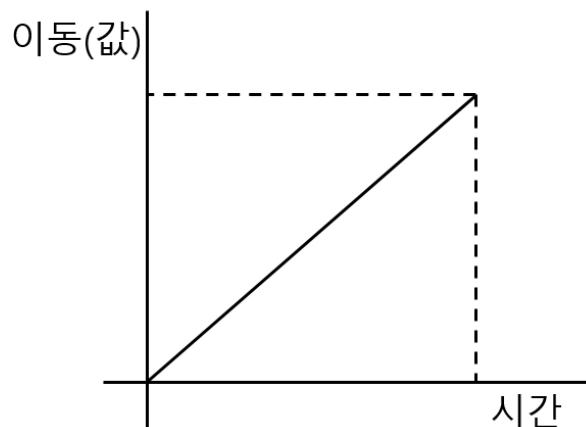
6. Animation Framework 와 State Machine

GUI상에서 화면 전환 시 부드러운 화면 전환과 같은 애니메이션 요소를 사용하기 위해서 Qt에서는 Animation Framework를 제공한다. 윈도우 화면에서 애니메이션 요소를 사용 가능하며 GUI 상에 배치된 위젯들도 각각의 애니메이션 요소를 사용할 수 있다.

애니메이션 요소로 투명, 이동, 확대/축소 등과 같은 애니메이션 요소를 사용할 수 있다. 예를 들어 어플리케이션 실행 시 화면 가로와 세로 크기가 600 x 400 픽셀 크기의 윈도우가 실행되어야 한다면 100 x 100 크기에서 지정한 시간 동안 600 x 400 크기로 윈도우 창의 크기가 커지는 것을 애니메이션 요소로 사용할 수 있다.

또한 동시에 투명(Opacity) 를 사용해 0.0부터 1.0으로 투명을 지정된 시간 동안 변경되도록 애니메이션 효과를 사용할 수 있다. 0.0 은 완전한 투명 상태이고 1.0 은 투명이 모두 사라진 상태이다.

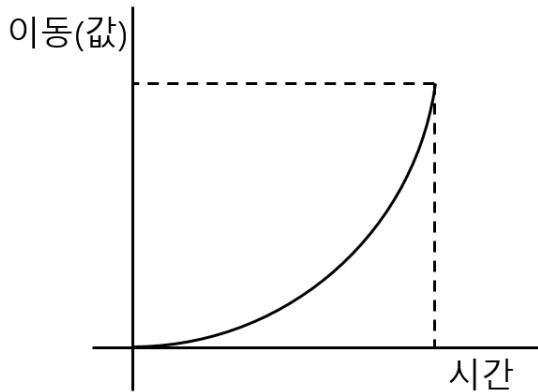
또한 이외에도 애니메이션의 시간 값에 Easing Curves를 사용할 수 있다. 예를 들어 1.0 초 동안 GUI 버튼의 x, y 위치가 100, 100 의 위치에서 200, 200 의 좌표로 이동하고 이동하는 시간을 1.0 초로 지정했다고 가정해보자. 이때 x, y 시작 좌표부터 목적지 좌표까지 이동하는데 일정한 속도로 이동할 것이다.



<그림> 이동과 시간 값 그래프

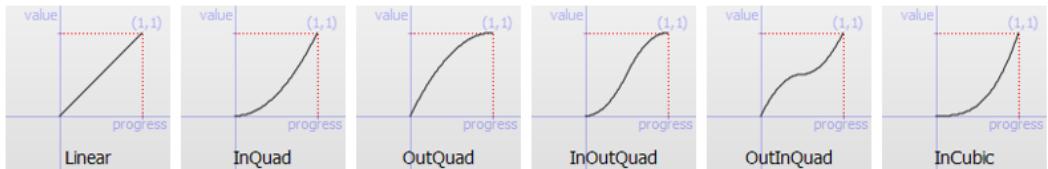
위의 그림에서 보는 것과 같이 이동(값)이라는 Y축은 GUI 버튼의 x, y 위치 좌표가 100, 100 에서 200, 200 값으로 이동하는 데 1.0 초 시간이 일정하게 소요되는 것을 나타내는 그래프이다. 즉 이동 값에 따라 시간이 일정하게(Linear) 증가한다. 하지만 시간

에 Easing Curves를 사용하면 이동 값에 시간의 값을 다음 그림에서 보는 것과 같이 변경할 수 있다.



<그림> Easing Curve 적용 시 그래프

최종 x, y 좌표까지 이동하는 시간의 값을 위의 그림에서 보는 것과 같이 적용이 가능하다. 이외에도 Easing Curve 는 약 46가지의 다양한 Easing Curve를 적용할 수 있다.



<그림> 다양한 Easing Curve 적용

그리고 Qt 는 Animation 요소에 State Machine 이라는 기법을 사용할 수 있다. State Machine 은 예로 ON/OFF 스위치를 예를 들 수 있다. 한가지 값을 변경하기 위해 ON/OFF 를 사용할 수 있을 뿐만 아니라 여러가지 옵션을 함께 사용할 수 있다. 예를 들어 집 현관의 형광등과 거실의 형광등을 함께 켜지게 하고자 할 때 State Machine 이라는 기법을 사용해 함께 켜지거나 꺼지게 할 수 있다. 즉 어떤 상황이 되면 그 상황에 따라 여러가지 값이 동시에 적용되는 기법을 State Machine 이라는 기법을 사용할 수 있다.

GUI 상에 A버튼, B버튼 그리고 C버튼이 있다고 가정해 보자. A버튼을 누르면 GUI 상에 C버튼의 위치를 100, 100 의 위치에서 200, 200으로 이동하고 투명을 1.0 에서 0.0으로 동시에 변경하고자 한다면 이때 State Machine 을 사용할 수 있다. 물론 Animation 에서 병렬 실행하는 기능을 제공하지만 만약 수십 개의 상태를 변경해야 한다면 State Machine 기법을 사용하는 것이 더 효과적일 수 있다. 지금까지 이번 장에서 다룬 내용을 간략하게 요약해 보았다. 이번 장에서는 크게 Animation, Easing Curve 그리고 State

Machine 을 예제를 통해 다루어보도록 하겠다.

- GUI 상의 버튼을 이동 시키기 위해 Animation 을 사용해 구현

이번 예제에서는 GUI 상에 QPushButton 을 배치한다. 배치한 QPushButton을 특정 위치로 이동하는데 Animation 요소를 사용하는 예제이다. 버튼을 클릭하면 x, y 위치가 10, 10 의 위치에서 200, 150 의 위치로 버튼의 x, y 좌표가 이동한다. 이동하는데 시간은 3000 밀리세컨드이다. 다음은 widget.h 소스코드이다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include &ltQPushButton>
#include &ltQPropertyAnimation>

class Widget : public QWidget
{
    Q_OBJECT
public:
    Widget(QWidget *parent = 0);
    ~Widget();

private:
    QPropertyAnimation *animation;

public slots:
    void btnClicked();
};

#endif // WIDGET_H
```

위의 예제 소스코드에서 보는 것과 같이 QPropertyAnimation 클래스의 오브젝트를 선언한다. btnClicked() Slot 함수는 버튼을 클릭하면 호출된다. 다음은 widget.cpp 소스코드이다.

```
#include "widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent)
{
    this->resize(350, 200);
```

```

QPushButton *btn = new QPushButton("Button", this);
connect(btn, &QPushButton::pressed, this, &Widget::btnClicked);
btn->setGeometry(10, 10, 100, 30);

animation = new QPropertyAnimation(btn, "geometry", this);

animation->setDuration(3000); // 3초(단위 밀리세컨드)
animation->setStartValue(QRect(10, 10, 100, 30)); // 시작 좌표
animation->setEndValue(QRect(200, 150, 100, 30)); // 끝나는 좌표
}

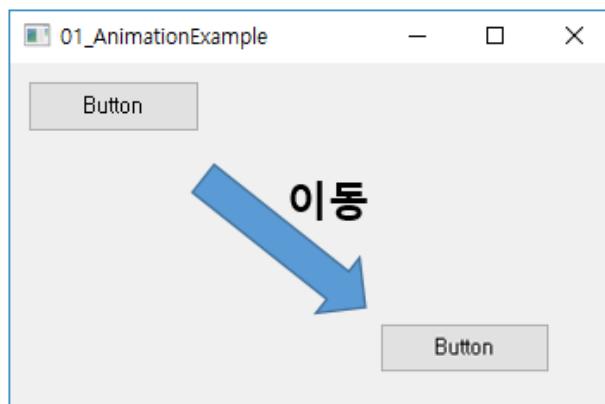
void Widget::btnClicked()
{
    animation->start();
}

Widget::~Widget()
{
}

```

QPropertyAnimation 클래스를 new 연산자로 생성자 함수를 실행할 때 첫 번째 인자에 사용한 btn 은 QPushButton 의 오브젝트를 명시한다. 그리고 두 번째 “geometry” 는 이동을 의미하는 요소이다.

setDuration() 는 이동하는데 소요되는 총 시간을 지정 할 수 있다. setStartValue() 는 시작 좌표의 x, y, width, height 의 값을 지정한다. setEndValue() 는 이동하는 위치와 가로, 세로 크기를 지정한다. 그리고 btnClicked() 함수를 클릭 하면 QPropertyAnimation 이 실행된다.



<그림> 예제 실행 화면

위의 그림에서 보는 것과 같이 버튼을 클릭하면 x, y 좌표가 200, 150로 3초간 이동한다. 다음은 이 예제에 다음과 같이 Easing Curve를 사용해 보도록 하자. 다음 예제에서 보는 것과 같이 Widget 생성자 마지막에 다음 소스코드를 추가하자.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    this->resize(350, 200);
    QPushButton *btn = new QPushButton("Button", this);
    ...
    // 아래 소스코드 추가
    animation->setEasingCurve(QEasingCurve::OutInQuart);
}
...
```

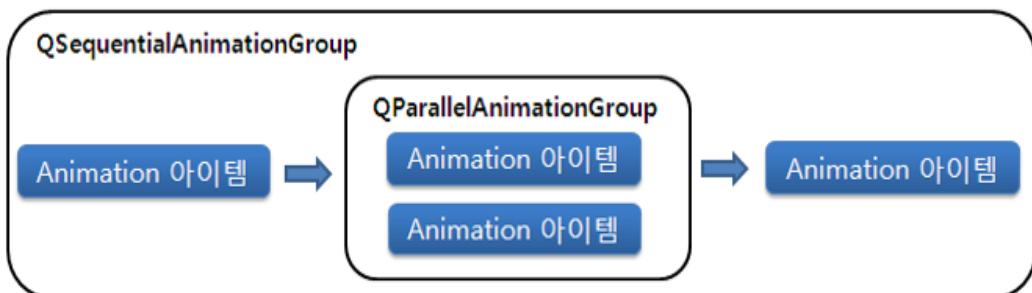
위의 예제 소스코드에서 보는 것과 같이 setEasingCurve() 함수를 추가한 후 빌드 후 실행해 보도록 하자. 버튼을 클릭하면 Easing Curve 가 적용된 것을 확인할 수 있다.

Easing Curve 는 시작좌표부터 마지막 좌표까지 이동하는 경로의 진행 속도를 보간하여 움직임의 패턴을 변경할 수 있는 기능을 제공한다. 이동 좌표 외에도 투명, 확대/축소에 사용할 수 있다.

지금 다룬 이 예제의 전체소스코드는 Ch06 > 01_AnimationExample 디렉토리를 참조하면 된다.

● Animation 그룹을 사용한 예제

이번에 다룬 예제는 그룹화를 사용해 Animation 요소를 동시에 또는 순차적 실행하기 위한 기능 구현을 살펴보도록 하자. Animation Framework에서는 사용하고자 하는 애니메이션 요소가 많은 경우 다음 그림에서 보는 것과 같이 그룹화해 Animation 을 사용할 수 있다.



<그림> Animation 그룹을 예

위의 그림과 같이 애니메이션을 그룹화 하기 위해 QSequentialAnimationGroup 클래스와 QParallelAnimationGroup 클래스를 사용할 수 있다. QSequentialAnimationGroup 은 애니메이션을 순차적으로 실행할 수 있으며 QParallelAnimationGroup 클래스는 애니메이션을 동시(병렬)에 수행 시킬 수 있다.

또한 그림에서 보는 것과 같이 QSequentialAnimationGroup 내에 애니메이션 요소를 내부 그룹화 하기 위해 QParallelAnimationGroup 을 사용할 수 있다.

반대로 QParallelAnimationGroup 내에 QSequentialAnimationGroup를 사용할 수 있다. 다음 예제 소스코드는 widget.h 소스코드이다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QPushButton>
#include <QPropertyAnimation>
#include <QSequentialAnimationGroup>
#include <QParallelAnimationGroup>

class Widget : public QWidget
{
    Q_OBJECT
public:
    Widget(QWidget *parent = 0);
    ~Widget();
private slots:
    void btn_clicked();
private:
    QPropertyAnimation *anim1;
    QPropertyAnimation *anim2;
    QSequentialAnimationGroup *sGroup;
};

#endif // WIDGET_H
```

위의 예제 소스코드에서 보는 것과 같이 QSequentialAnimationGroup 클래스는 애니메이션 요소를 수행시킬 수 있는 기능을 제공한다. 다음 예제 소스코드는 widget.cpp이다.

```
#include "widget.h"
```

```

Widget::Widget(QWidget *parent) : QWidget(parent)
{
    resize(320, 270);
    QPushButton *btn1 = new QPushButton("First", this);
    btn1->setGeometry(10, 10, 100, 30);

    QPushButton *btn2 = new QPushButton("Second", this);
    btn2->setGeometry(10, 45, 100, 30);

    anim1 = new QPropertyAnimation(btn1, "geometry");
    anim1->setDuration(2000); // 2초(단위 밀리세컨드)
    anim1->setStartValue(QRect(10, 10, 100, 30)); // 시작 좌표
    anim1->setEndValue(QRect(200, 150, 100, 30)); // 끝 좌표

    anim2 = new QPropertyAnimation(btn2, "geometry");
    anim2->setDuration(2000); // 2초(단위 밀리세컨드)
    anim2->setStartValue(QRect(10, 45, 100, 30)); // 시작 좌표
    anim2->setEndValue(QRect(200, 195, 100, 30)); // 끝 좌표

    sGroup = new QSequentialAnimationGroup;
    sGroup->addAnimation(anim1);
    sGroup->addAnimation(anim2);

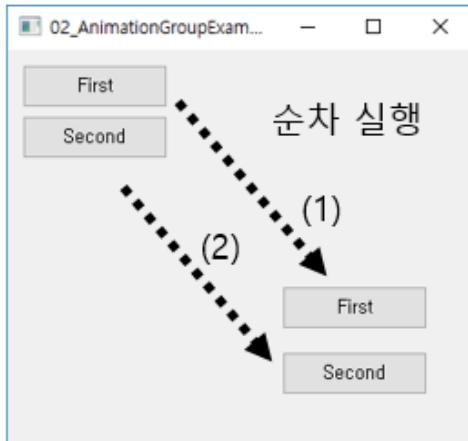
    connect(btn1, SIGNAL(clicked()), this, SLOT(btn_clicked()));
    connect(btn2, SIGNAL(clicked()), this, SLOT(btn_clicked()));
}

void Widget::btn_clicked()
{
    sGroup->start(QPropertyAnimation::DeleteWhenStopped);
}

Widget::~Widget()
{
}

```

위의 예제 소스코드는 두개의 버튼을 배치한다. 애니메이션이 시작되면 [First] 버튼을 먼저 이동한 후 [Second] 버튼을 이동한다. 다음 그림은 예제 실행 화면이다.



<그림> 예제 실행 화면

좌측 그림에서 보는 것과 같이 2개의 버튼의 애니메이션이 순차적으로 실행된다.

다음 예는 지금까지 작성한 소스코드를 수정해 병렬로 변경해 보도록 한다.

widget.h 소스코드상에서 다음과 같이 주석처리 후 하단에 소스코드를 추가해 보도록 한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QPushButton>
#include <QPropertyAnimation>
#include <QSequentialAnimationGroup>
#include <QParallelAnimationGroup>

class Widget : public QWidget
{
    Q_OBJECT
public:
    Widget(QWidget *parent = 0);
    ~Widget();

private slots:
    void btn_clicked();

private:
    QPropertyAnimation *anim1;
    QPropertyAnimation *anim2;
    //QSequentialAnimationGroup *sGroup;
    QParallelAnimationGroup *pGroup;
};

#endif // WIDGET_H
```

다음은 widget.cpp에서 QSequentialAnimationGroup 클래스를 선언한 부분과 Slot 함

수를 아래 예제 소스코드에서 보는 것과 같이 병렬로 애니메이션이 동작할 수 있도록 다음과 같이 수정 및 추가해 보도록 하자.

```
#include "widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent)
{
    resize(320, 270);
    QPushButton *btn1 = new QPushButton("First", this);
    btn1->setGeometry(10, 10, 100, 30);

    QPushButton *btn2 = new QPushButton("Second", this);
    btn2->setGeometry(10, 45, 100, 30);

    anim1 = new QPropertyAnimation(btn1, "geometry");
    anim1->setDuration(2000); // 2초(단위 밀리세컨드)
    anim1->setStartValue(QRect(10, 10, 100, 30)); // 시작 좌표
    anim1->setEndValue(QRect(200, 150, 100, 30)); // 끝 좌표

    anim2 = new QPropertyAnimation(btn2, "geometry");
    anim2->setDuration(2000); // 2초(단위 밀리세컨드)
    anim2->setStartValue(QRect(10, 45, 100, 30)); // 시작 좌표
    anim2->setEndValue(QRect(200, 195, 100, 30)); // 끝 좌표

    pGroup = new QParallelAnimationGroup; // Parallel Group
    pGroup->addAnimation(anim1);
    pGroup->addAnimation(anim2);

    connect(btn1, SIGNAL(clicked()), this, SLOT(btn_clicked()));
    connect(btn2, SIGNAL(clicked()), this, SLOT(btn_clicked()));
}

void Widget::btn_clicked()
{
//    sGroup->start(QPropertyAnimation::DeleteWhenStopped);
    pGroup->start(QPropertyAnimation::DeleteWhenStopped);
}

Widget::~Widget()
{
```

예제를 실행하면 [First] 버튼과 [Second] 버튼의 애니메이션이 동시에 실행되는 것을

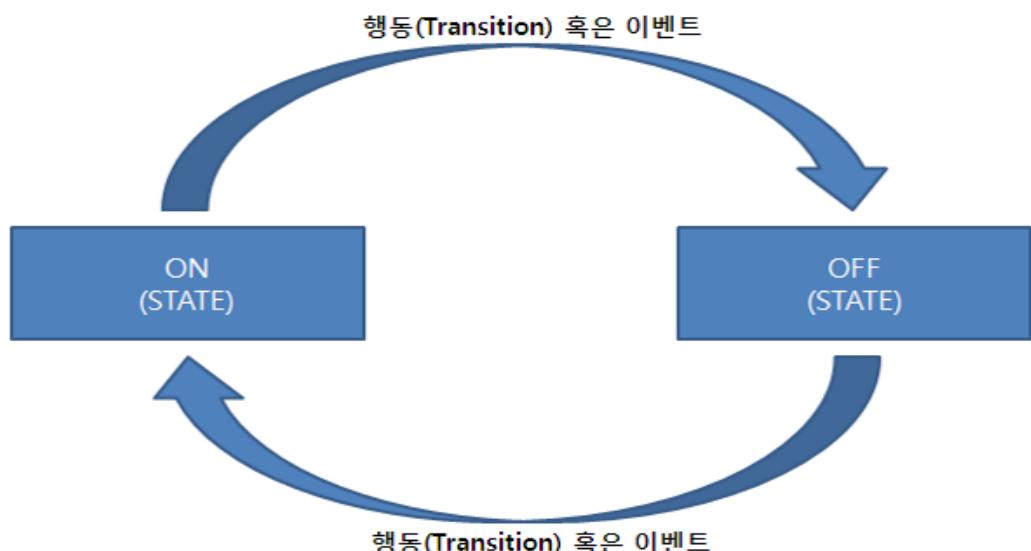
확인할 수 있을 것이다.



<그림> 예제 실행 화면

- Animation 과 State Machine 을 사용한 예제

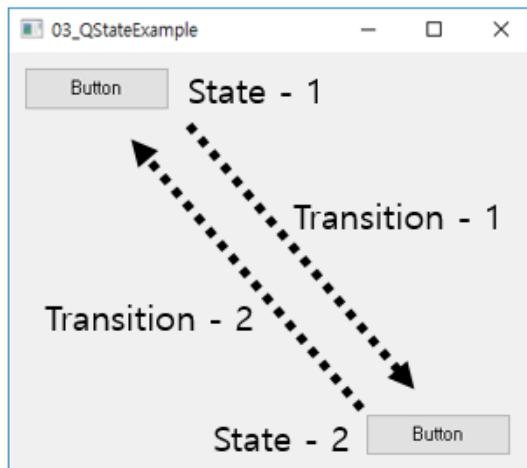
State Machine 을 ON/OFF 에 비유해 설명 했다. ON 과 OFF 를 상태, State 로 ON 에서 OFF 로 State 가 바뀌는 행동 또는 행위를 Transition 으로 구분할 수 있다. 즉 State 는 ON 또는 OFF 인지 상태를 나타낸다. Transition은 ON 에서 OFF 로 바뀌는 행동으로 볼 수 있다.



<그림> State 와 Transition

State Machine 기법을 Animation에서 사용하는 이유는 수 많은 Animation 요소를 사용할 경우 동시에 애니메이션을 요소를 수행하기 위한 목적으로 사용할 수 있다. 예를 들어 수십 개의 애니메이션 요소를 수행한다고 가정해보자 그룹화를 시킬 수 있으나 소스코드가 복잡해 진다. 하지만 State Machine을 적용하면 쉽게 구현이 가능하다.

Qt에서 State Machine 기법은 위의 그림에서 보는 것과 같이 State와 Transition으로 구분된다. State를 사용하기 위해서 QState 클래스를 제공한다. 그리고 Transition을 적용하기 위해서 QTransition 클래스를 제공한다.



<그림> 예제 실행 화면

좌측의 그림은 State Machine 예제의 실행 화면이다. State - 1은 버튼의 x, y 좌표가 10, 10인 상태가 State - 1이다.

그리고 x, y 좌표가 250, 250인 상태가 State - 2이다.

State - 1에서 State - 2로 이동하는 것을 Transition - 1이고 State - 2에서 State - 1 상태로 이동하는 것을 Transition - 2로 지정하였다.

따라서 State - 1 상태에서 버튼을 클릭하면 Transition - 2이 수행되어 State - 2 상태가 변경된다. 반대로 State - 2에서 버튼을 클릭하면 Transition - 2가 수행되어 State - 1 상태가 된다. 다음은 widget.cpp 소스코드의 일부이다.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    resize(360, 290);
    QPushButton *button = new QPushButton("Button", this);
    button->setGeometry(10, 10, 100, 30);

    QStateMachine *machine = new QStateMachine;
    QState *state1 = new QState(machine); // state-1
    state1->assignProperty(button, "geometry", QRect(10, 10, 100, 30));
    machine->setInitialState(state1);

    QState *state2 = new QState(machine); // state-2
    state2->assignProperty(button, "geometry", QRect(250, 250, 100, 30));
```

```
// transition-1
QSignalTransition *transition1;
Transition1 = state1->addTransition(button, SIGNAL(clicked()), state2);
transition1->addAnimation(new QPropertyAnimation(button, "geometry"));

// transition-2
QSignalTransition *transition2;
transition2 = state2->addTransition(button, SIGNAL(clicked()), state1);
transition2->addAnimation(new QPropertyAnimation(button, "geometry"));
machine->start();
}

...
```

QState 클래스로 state1 과 state2 오브젝트를 생성한다. 생성한 오브젝트의 상태를 위치 좌표로 선언 하였다. 그리고 state1 과 state2 상태를 서로 변경하기 위해 QSignalTransition을 사용하였다. 그리고 마지막에 start() 함수를 수행하였다.

start() 함수는 바로 실행되지 않고 버튼이 클릭 되면 State 에 따라 Transition 이 수행된다. 전체 소스는 Ch06 > 03_QStateExample 디렉토리를 참조하면 된다.

7. Qt OpenGL 모듈을 이용한 3D 그래픽스

Qt에서는 OpenGL 을 이용해 3D 그래픽스 프로그래밍을 좀더 사용하기 쉽게 제공하기 위해 OpenGL을 래핑(Wrapping) 한 Qt OpenGL 모듈을 제공한다. Qt OpenGL 모듈은 OpenGL과 동일하다. 우리가 GUI 프로그래밍을 위해 QWidget 을 사용하는 것처럼 OpenGL 을 이용해 3D 를 랜더링하기 위해서 Qt OpenGL 모듈에서는 QOpenGLWidget 과 QGLWidget 클래스를 제공한다.

OpenGL 은 실리콘 그래픽스에서 개발한 3D 그래픽스 표준 API이며 약 250개의 함수를 제공한다. 기하도형에서 복잡한 3D 그래픽스를 표현할 수 있는 기능을 제공하며 OpenGL 과 OpenGL ES 로 분류된다. OpenGL 은 데스크탑 PC와 같은 환경에서 사용되는 API이다.

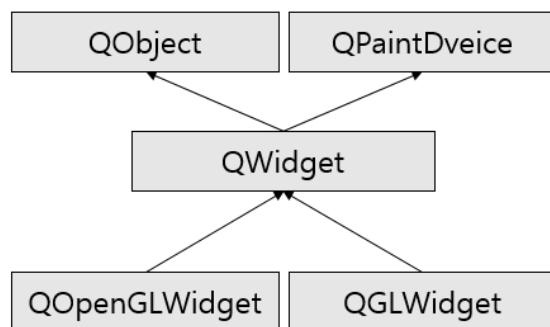
OpenGL ES는 임베디드 시스템에서와 같이 자원이 제한된 환경에서 3D 그래픽스를 위해 사용되며 Qt 에서도 OpenGL 과 OpenGL ES 모두 지원한다. Qt에서 OpenGL 을 사용하기 위해서 프로젝트 파일에 다음과 같이 opengl 을 명시해야 한다.

```
QT + opengl
```

그리고 헤더에 다음과 같이 include를 사용해야 한다.

```
#include <QtOpenGL>
```

QOpenGLWidget 과 QGLWidget 은 3D 그래픽 요소를 렌더링 하기 위한 동일한 기능을 제공한다. 두개의 클래스 중 어느 것을 사용해도 무방하지만 Qt 5.4 버전부터는 QOpenGLWidget 을 사용하는 것을 권장한다. QWidget 처럼 OpenGL 을 랜더링 하기 위해서 QGLWidget 을 사용할 수 있다.



<그림> QOpenGLWidget 과 QGLWidget 의 상속 관계

QOpenGLWidget 은 일반적으로 OpenGL 을 이용하여 구현했던 것과 같이 쉽게 구현 할 수 있도록 Qt에서는 다음과 같은 3개의 Virtual 함수를 제공한다.

- ✓ paintGL() – OpenGL 을 렌더링(표시) 하기 위한 Virtual 함수이다. QGLWidget 위젯 을 업데이트하면 이 함수가 호출된다.
- ✓ resizeGL() – OpenGL의 viewport, project 등을 설정한다. 위젯의 크기가 변경 될 때 자동으로 호출된다. QGLWidget 이 처음 실행될 때 resize 이벤트가 자동으로 호출 된다.
- ✓ initializeGL() – OpenGL 렌더링 context를 설정한다. paintGL(), resizeGL() 함수가 호출되기 전 초기화를 위해 호출된다.

다음은 QOpenGLWidget을 사용한 예제 소스코드 이다.

```
...
class MyGLDrawer : public QOpenGLWidget
{
    Q_OBJECT // signals/slot 을 사용하기 위해서는 선언해야 함.
public:
    MyGLDrawer(QWidget *parent)
        : QGLWidget(parent) {}

protected:
    void initializeGL() override {
        ...
        glClearColor(0.0, 0.0, 0.0, 0.0);
        glEnable(GL_DEPTH_TEST);
        ...
    }

    void resizeGL(int w, int h) override {
        glViewport(0, 0, (GLint)w, (GLint)h);
        ...
        glFrustum(...);
        ...
    }
}
```

```

void paintGL() override {
    // draw the scene:
    glRotatef(...);
    glMaterialfv(...);
    glBegin(GL_QUADS);
    glVertex3f(...);
    glVertex3f(...);
    ...
    glEnd();
    ...
}
};

...

```

QWidget 위젯 영역을 업데이트 하기 위해서 update() 함수를 사용한 것과 같이 QGLWidget 을 업데이트 하기 위해서 updateGL() 함수를 사용할 수 있다. 이 함수를 호출하면 paintGL() 함수가 호출된다.

그리고 OpenGL 을 지원하는 시스템 이라면 QGLWidget 은 시스템 상에서 오버레이를 사용하기 위해서 다음과 같은 3개의 오버레이 함수를 지원한다.

- ✓ paintOverlayGL() – paintGL() 함수와 동일하다. 위젯의 main context 를 대신해 위젯의 오버레이 연산을 위해 사용하는 재정의(Virtual) 함수이다.
- ✓ resizeOverlayGL() – resizeGL() 함수와 기능은 비슷하다. Widget 의 Main context 대신 Widget 의 오버레이 Context를 위해 사용하는 재정의 함수이다.
- ✓ initializeOverlayGL() – initializeGL() 함수와 동일하다. 위젯의 main context를 대신해 위젯의 오버레이 연산을 사용하는 재정의(Virtual) 함수이다. 이 함수는 paintOverlayGL() 혹은 resizeOverlayGL() 함수를 호출하기 전 한번만 호출한다. 이 함수는 OpenGL의 context 랜더링 플래그를 설정하기 위해 사용

QOpenGLWidget 과 QGLWidget 은 QWidget 상속 받기 때문에 QWidget 클래스에서 사용하는 2D 그래픽 렌더링을 위한 paintEvent() Virtual 함수를 사용할 수 있다.

```

...
class GLWidget : public QOpenGLWidget
{
    Q_OBJECT

public:
    GLWidget(Helper *helper, QWidget *parent);

public slots:
    void animate();

protected:
    void paintEvent(QPaintEvent *event) override;

private:
    Helper *helper;
    int elapsed;
};

...

```

또한 QOpenGLWidget 과 QGLWidget 에서 사용하는 paintGL() Virtual 함수는 OpenGL 을 이용해 3D 그래픽 요소를 렌더링 하는 것과 같이 2D 그래픽 요소를 paintEvent() 함수를 사용해 Over Painting 이 가능하다.

- Triangle 3D 렌더링 예제

이 예제의 목적은 3D 그래픽 요소를 어떤 방식으로 렌더링 하는지 설명하기 위한 간단한 예제이다.

즉 QOpenGLWidget 클래스를 이용해 어떤 방식으로 initializeGL(), resizeGL(), paintGL() 을 사용하는 방법을 간단히 살펴보기 위한 예제이다. 또한 이번 예제에서는 GLSL(GL Shader Language) 을 사용하기 위한 QGLShader 클래스 사용 방법에 대해서도 살펴보도록 하자.

다음 예제 소스코드는 GLSL로 작성된 소스코드이며 파일명은 vertexShader.vsh 이며 다음과 같이 소스코드를 작성한다.

```

uniform mat4 mvpMatrix;
in vec4 vertex;

void main(void)

```

```
{  
    gl_Position = mvpMatrix * vertex;  
}
```

다음 예제 소스코드는 fragmentShader.fsh 소스코드이다.

```
#version 130  
  
uniform vec4 color;  
out vec4 fragColor;  
  
void main(void)  
{  
    fragColor = color;  
}
```

위의 예와 같이 2개의 GLSL 파일을 작성한다. 그리고 다음과 같이 QOpenGLWidget 클래스를 상속받아 glwidget.h 와 glwidget.cpp 소스코드를 작성한다. 다음 예제 소스코드는 glwidget.h 헤더파일 소스코드 이다.

```
#ifndef GLWIDGET_H  
#define GLWIDGET_H  
  
#include <QtOpenGL>  
#include <QOpenGLWidget>  
#include <QGLShaderProgram>  
  
class GlWidget : public QOpenGLWidget  
{  
    Q_OBJECT  
public:  
    GlWidget(QWidget *parent = 0);  
    ~GlWidget();  
    QSize sizeHint() const;  
  
protected:  
    void initializeGL();  
    void resizeGL(int width, int height);  
    void paintGL();  
  
private:  
    QMatrix4x4 pMatrix;  
    QGLShaderProgram shaderProgram;
```

```

    QVector<QVector3D> vertices;
};

#endif // GLWIDGET_H

```

위의 예제에서 보는 것과 같이 GlWidget 클래스는 QOpenGLWidget 클래스로부터 상속받아 구현 한다. 다음 예제 소스코드는 glwidget.cpp 소스코드 이다.

```

#include "glwidget.h"

GlWidget::GlWidget(QWidget *parent) : QOpenGLWidget(parent)
{
}

GlWidget::~GlWidget()
{
}

QSize GlWidget::sizeHint() const
{
    return QSize(640, 480);
}

void GlWidget::initializeGL()
{
    QOpenGLFunctions *f = QOpenGLContext::currentContext()->functions();

    f->glEnable(GL_DEPTH_TEST);
    f->glEnable(GL_CULL_FACE);
    f->glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

    shaderProgram.addShaderFromSourceFile(QGLShader::Vertex,
                                        (":/vertexShader.vsh"));
    shaderProgram.addShaderFromSourceFile(QGLShader::Fragment,
                                        (":/fragmentShader.fsh"));
    shaderProgram.link();

    vertices << QVector3D(1, 0, -2) << QVector3D(0, 1, -2)
        << QVector3D(-1, 0, -2);
}

void GlWidget::resizeGL(int width, int height)
{
}

```

```

if (height == 0)
    height = 1;

pMatrix.setToIdentity();
pMatrix.perspective(60.0, (float)width / (float)height, 0.001, 1000);

QOpenGLFunctions *f = QOpenGLContext::currentContext()->functions();
f->glViewport(0, 0, width, height);
}

void GlWidget::paintGL()
{
    QOpenGLFunctions *f =
    QOpenGLContext::currentContext()->functions();

    f->glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    QMatrix4x4 mMatrix;
    QMatrix4x4 vMatrix;

    shaderProgram.bind();
    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
    shaderProgram.setUniformValue("color", Color(Qt::white));
    shaderProgram.setAttributeArray("vertex", vertices.constData());
    shaderProgram.enableAttributeArray("vertex");

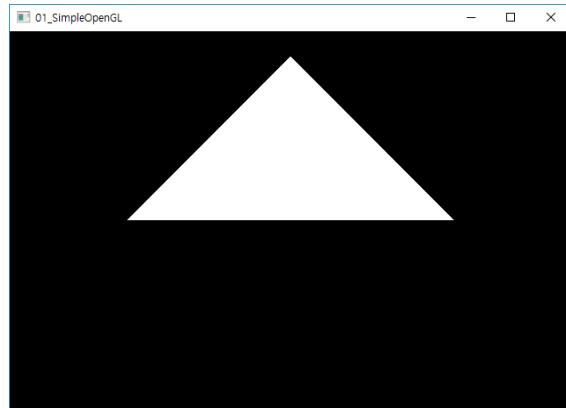
    f->glDrawArrays(GL_TRIANGLES, 0, vertices.size());
    shaderProgram.disableAttributeArray("vertex");

    shaderProgram.release();
}

```

헤더파일에서 선언한 QGLShaderProgram 클래스의 shaderProgram 오브젝트는 GLSL로 작성한 소스코드를 읽어와 Qt에서 사용할 수 있는 기능을 제공한다.

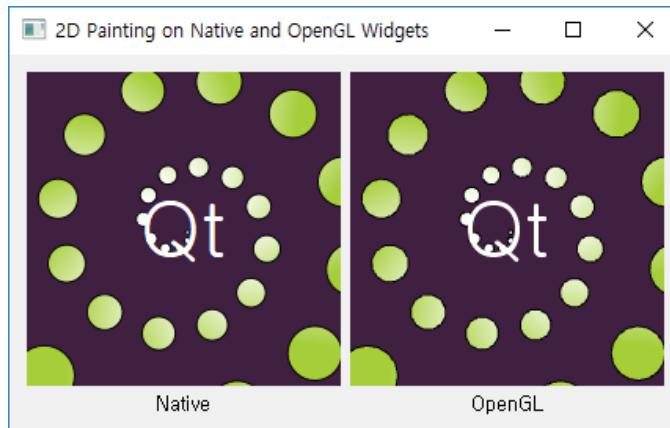
paintGL() Virtual 함수 내에서 QGLShaderProgram 클래스의 setUniformValue(), setAttributeArray(), enableAttributeArray() 등 함수를 이용해 GLSL을 사용할 수 있다. 이 예제의 전체 소스코드는 Ch07 > 01_SimpleOpenGL 디렉토리를 참조하면 된다.



<그림> 예제 실행 화면

- QOpenGLWidget에서 paintEvent() Virtual 함수를 사용하기 위한 예제

QWidget에서 사용하는 2D 그래픽 요소를 렌더링 하기 위한 방법으로 paintEvent()를 사용했던 것과 같이 QOpenGLWidget에서도 paintEvent() 함수를 사용할 수 있다. 이번 예제에서는 QOpenGLWidget에서 paintEvent() 함수를 사용하는 방법에 대해서 알아보도록 하자. 이 예제 소스코드는 Qt를 설치하면 함께 제공되는 Example에서 제공하는 예제이다. 또한 Ch07 > 2dpainting 디렉토리에 동일한 소스코드를 제공한다.



<그림> 예제 실행 화면

위의 그림과 같이 왼쪽의 위젯은 QWidget 클래스를 상속받아 paintEvent()를 사용하였다. 그리고 오른쪽 위젯은 QOpenGLWidget 클래스를 상속받아 paintEvent()를 사용한 예제이다. 이 예제 소스코드는 4개의 클래스로 구성된다.

Window 클래스는 Widget 클래스 위젯과 GLWidget 클래스 위젯을 위의 그림에서 보는 것과 같이 Windows의 GUI 상에 배치하기 위한 클래스이다. 그리고 Helper 클래스

는 Widget 클래스와 GLWidget 클래스의 paintEvent()함수에서 위의 그림에 보는 것과 같이 2D 그래픽요소를 랜더링 하는 기능이 구현 되어 있다. 다음 예제 소스코드는 Helper 클래스의 헤더 소스코드이다.

```
...
class Helper
{
public:
    Helper();
public:
    void paint(QPainter *painter, QPaintEvent *event, int elapsed);

private:
    QBrush background;
    QBrush circleBrush;
    QFont textFont;
    QPen circlePen;
    QPen textPen;
};
```

이 클래스의 paint() 함수의 첫 번째 인자는 Widget 클래스와 GLWidget 클래스에서 사용하는 QPainter 의 포인터를 넘겨준다. 따라서 이 함수에서 QPainter 위의 예제 실행화면에서 보는 것과 같이 2D 그래픽요소를 드로잉 하는 기능을 제공하는 클래스이다. 다음 예제 소스 코드는 Helper 클래스의 함수 구현 부 소스코드이다.

```
...
Helper::Helper()
{
    QLinearGradient gradient(QPointF(50, -20), QPointF(80, 20));
    gradient.setColorAt(0.0, Qt::white);
    gradient.setColorAt(1.0, QColor(0xa6, 0xce, 0x39));

    background = QBrush(QColor(64, 32, 64));
    circleBrush = QBrush(gradient);
    circlePen = QPen(Qt::black);
    circlePen.setWidth(1);
    textPen = QPen(Qt::white);
    textFont.setPixelSize(50);
}

void Helper::paint( QPainter *painter, QPaintEvent *event, int elapsed )
{
```

```

painter->fillRect(event->rect(), background);
painter->translate(100, 100);

painter->save();
painter->setBrush(circleBrush);
painter->setPen(circlePen);
painter->rotate(elapsed * 0.030);

qreal r = elapsed / 1000.0;
int n = 30;
for (int i = 0; i < n; ++i) {
    painter->rotate(30);
    qreal factor = (i + r) / n;
    qreal radius = 0 + 120.0 * factor;
    qreal circleRadius = 1 + factor * 20;
    painter->drawEllipse(QRectF(radius, -circleRadius,
                                circleRadius * 2, circleRadius * 2));
}

painter->restore();
painter->setPen(textPen);
painter->setFont(textFont);
painter->drawText(QRect(-50, -50, 100, 100),
                  Qt::AlignCenter, QStringLiteral("Qt"));
}
...

```

위의 예제에서 보는 것과 같이 paint() 멤버 함수는 Widget 클래스와 GLWidget 클래스에서 사용된다. 다음 예제 소스코드는 GLWidget 클래스의 헤더소스코드이다.

```

#include <QOpenGLWidget>

class Helper;
class GLWidget : public QOpenGLWidget
{
    Q_OBJECT
public:
    GLWidget(Helper *helper, QWidget *parent);

public slots:
    void animate();

protected:
    void paintEvent(QPaintEvent *event) override;

```

```

private:
    Helper *helper;
    int elapsed;
};

```

helper 오브젝트는 paintEvent() 함수에서 사용한다. 다음 예제는 이 클래스의 함수 구현 소스코드이다.

```

GLWidget::GLWidget(Helper *helper, QWidget *parent)
    : QOpenGLWidget(parent), helper(helper)
{
    elapsed = 0;
    setFixedSize(200, 200);
    setAutoFillBackground(false);
}

void GLWidget::animate()
{
    elapsed = (elapsed + qobject_cast<QTimer*>(sender())->interval()) % 1000;
    update();
}

void GLWidget::paintEvent(QPaintEvent *event)
{
    QPainter painter;
    painter.begin(this);
    painter.setRenderHint(QPainter::Antialiasing);
    helper->paint(&painter, event, elapsed);
    painter.end();
}

```

위의 예제 소스코드에서 보는 것과 같이 실제 2D 그래픽 요소를 랜더링 하는 기능을 Helper 클래스에서 제공한다. QWidget 클래스를 상속받아 구현한 Widget 클래스의 paintEvent() 함수에서도 위와 동일한 방식으로 Helper 클래스를 사용한다.

위의 예제 소스코드에서 animate() 함수는 QGLWidget 을 배치한 부모 윈도우인 Window 클래스에서 사용한 QTimer 로 50ms(밀리세컨드) 단위로 호출된다. 따라서 50ms 주기로 paintEvent() 함수가 호출된다. 다음 예제 소스코드는 window 클래스의 함수 구현 부 소스코드 일부이다.

```

...
Window::Window()

```

```

{
    setWindowTitle(tr("2D Painting on Native and OpenGL Widgets"));
    Widget *native = new Widget(&helper, this);
    GLWidget *openGL = new GLWidget(&helper, this);

    QLabel *nativeLabel = new QLabel(tr("Native"));
    nativeLabel->setAlignment(Qt::AlignHCenter);

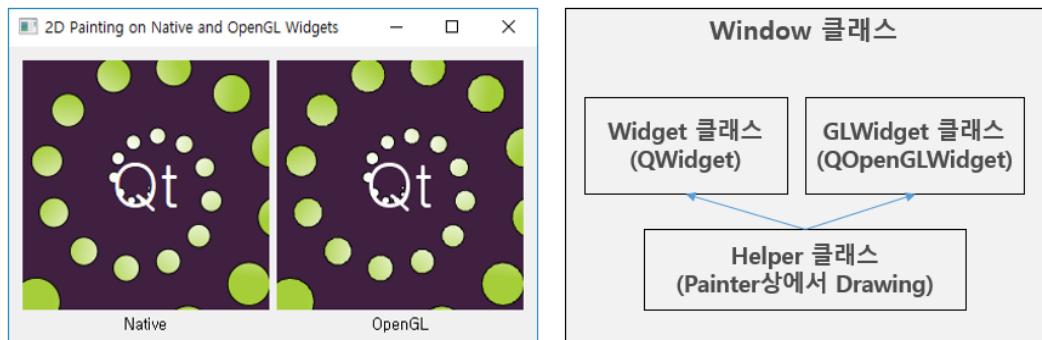
    QLabel *openGLLabel = new QLabel(tr("OpenGL"));
    openGLLabel->setAlignment(Qt::AlignHCenter);

    QGridLayout *layout = new QGridLayout;
    layout->addWidget(native, 0, 0);
    layout->addWidget(openGL, 0, 1);
    layout->addWidget(nativeLabel, 1, 0);
    layout->addWidget(openGLLabel, 1, 1);
    setLayout(layout);

    QTimer *timer = new QTimer(this);
    connect(timer, &QTimer::timeout, native, &Widget::animate);
    connect(timer, &QTimer::timeout, openGL, &GLWidget::animate);
    timer->start(50);
}
...

```

위의 예제에서 보는 것과 같이 QTimer로 50ms를 주기로 Widget 클래스와 GLWidget 클래스의 paintEvent() 함수가 계속 호출된다.



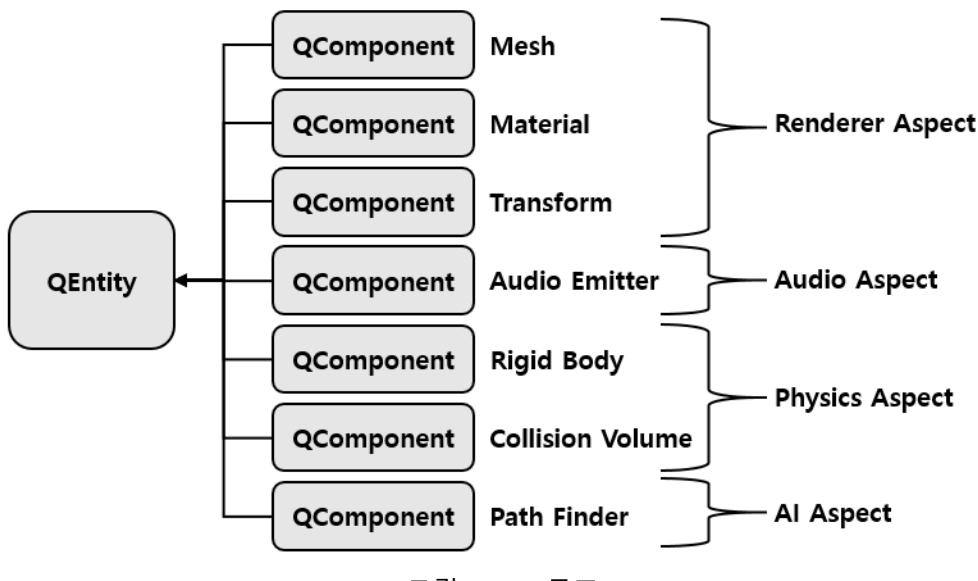
<그림> 예제 실행 화면과 클래스의 관계

8. Qt 3D 모듈을 이용한 3D 그래픽스

Qt 3D 모듈은 3D 그래픽 렌더링을 지원하는 모듈이다. OpenGL 보다 쉽고 직관적인 API를 제공한다. Qt 3D 모듈은 3D 그래픽 렌더링 이외에 물리, 오디오, 충돌 감지, 인공지능이 포함되어 있다.

Qt 3D 모듈은 Qt 5.5 버전에서 처음 소개 되었으며 Qt 5.7 버전에서 정식 버전으로 Release 되었다. OpenGL 과 같이 다양한 플랫폼(운영체제)에서 그래픽 성능을 보장하기 위해 GPU를 사용한다. Qt 3D 모듈은 3D 그래픽 렌더링과 더불어 2D 그래픽 렌더링을 지원한다. Qt 3D 는 OpenGL 버전에서 다른 렌더링 접근법을 총족 시키며 여러 렌더링 패스를 가능하게 한다. 또한 C++ 과 QML에서 사용이 가능하다. Qt 3D 모듈은 GLSL Shader를 쉽게 매핑할 수 있다.

Qt 3D 는 OpenGL 과 같이 Vertex, Tessellation control, Tessellation evaluation, geometry 그리고 Fragment shaders를 모두 지원한다. Qt 3D 모듈은 ECS(Entity Component System) 을 사용한다.

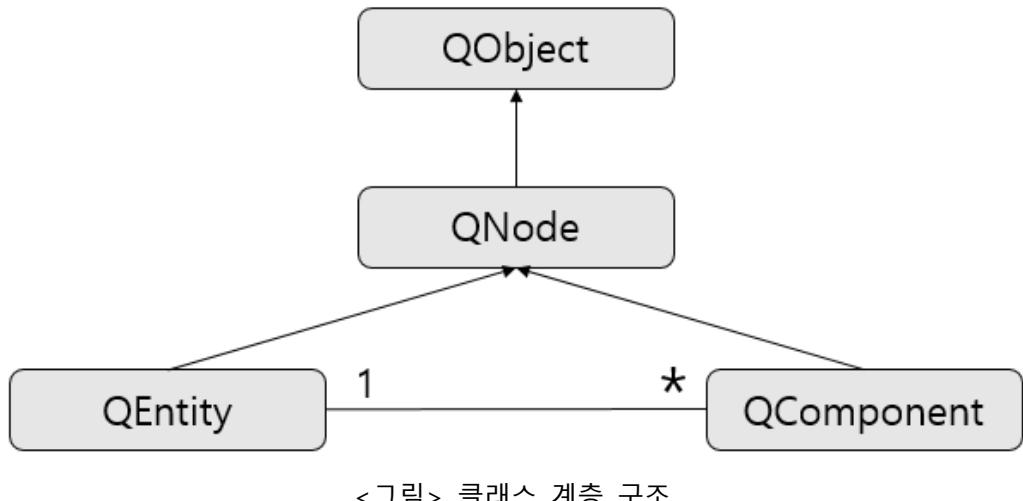


<그림> ECS 구조

ECS 자체는 어떠한 3D 자체를 나타내지 않지만 Entity가 하나 이상의 Component를 포함시킨다. 즉 ECS는 시뮬레이션 된 개체를 나타내지만 그 자체로는 특정 동작이나 특성이 없다. 다음 그림에서 보는 것과 같이 하나의 Entity가 하나 이상의 Component를 추가 함으로써 하나의 Entity는 여러 개의 Component로 구성할 수 있다.

위의 그림에서 보는 것과 같이 Qt 3D 모듈은 ECS를 간단한 클래스 계층으로 구현되었다. QEntity 와 QComponent 클래스는 QNode 클래스를 상속받아 구현되었으며 QNode 는 QObject를 상속받아 구현되었다.

Qt3DCore::QEntity는 다수의 Qt3DCore::QComponent 를 개체로 포함할 수 있다.



Qt 3D 는 3D 그래픽 외에도 2D 그래픽 렌더링도 지원하며 Qt 3D 가 지원하는 다양한 클래스를 지원한다. 상당히 많은 클래스를 지원하며 그 클래스들은 다음과 같이 7개로 분류할 수 있다.

<표> Qt 3D 모듈에서 제공하는 C++/QML 분류

분류	클래스 모듈	설명
C++	Qt3DCore	실시간 시뮬레이션 시스템을 지원하는 기능이 포함.
	Qt3DInput	사용자 입력을 처리하기 위한 클래스
	Qt3DLogic	Qt3D Backend와 프레임 동기화를 지원하기 위한 클래스.
	Qt3DRender	3D 와 2D 렌더링을 지원하기 위한 클래스.
	Qt3DAnimation	애니메이션을 제공하기 위한 클래스.
	Qt3DExtras	미리 만들어진 3D 요소를 제공.
	Qt3DScene2D	QML 컨텐츠를 Qt 3D 텍스처로 렌더링 제공

QML	Qt3D.Core	실시간 시뮬레이션 시스템을 지원하는 기능이 포함.
	Qt3D.Input	사용자 입력을 처리하기 위한 QML 모듈
	Qt3D.Logic	Qt3D Backend와 프레임 동기화를 지원하기 위한 모듈
	Qt3D.Render	3D 와 2D 렌더링을 지원하기 위한 클래스.
	Qt3D.Animation	애니메이션을 제공하기 위한 QML 모듈
	Qt3D.Extras	미리 만들어진 3D 요소를 제공.
	QtQuick.Scene2D	Scened2d 모듈을 Qt 3D QML 태입으로 제공
	QtQuick.Scene3D	Scened3d 모듈을 Qt 3D QML 태입으로 제공

위의 표에서 보는 것과 같이 Qt3DAnimation, Qt3DExtras 그리고 Qt3DRender 클래스 항목은 Preview로 제공 되지만 아직 개발 중인 항목이다.

QML에서는 Qt3D.Animation, Qt3D.Extras, QtQuick.Scene2D, QtQuick.Scene3D 모듈은 아직 개발 중인 항목이다. Qt 3D 모듈을 사용하기 위해서 프로젝트파일(.pro) 파일에 다음과 같이 추가해야한다.

```
QT += 3dcore 3drender 3dinput 3dlogic 3dextras 3danimation
```

그리고 다음과 같이 include 를 포함해야 Qt 3D 모듈을 사용할 수 있다.

```
#include <Qt3DCore>
#include <Qt3DRender>
#include <Qt3DInput>
#include <Qt3DLogic>
#include <Qt3DExtras>
#include <Qt3DAnimation>
```

QML에서 Qt3D모듈을 사용하기 위해 프로젝트파일에 다음과 같이 추가해야 한다.

```
QT += 3dcore 3drender 3dinput 3dlogic 3dextras
QT += qml quick 3dquick 3danimation
```

Qt 3D 모듈은 MS Windows, Linux(X11기반), macOS, Android, Embedded Linux 를 지원하며 WinRT는 아직 지원하지 않는다.

- 3D 구현 예제



<그림> 예제 실행 화면

이번 예제는 TEXT 랜더링 API를 이용해 3D 상에서 랜더링 하는 예제이다. 위의 그림에서 보는 것과 같이 오른쪽에 QSlider 위젯들은 좌측의 3D 영역의 카메라 위치를 실시간 변경할 수 있는 위젯이다. 각 QSlider 위젯은 카메라의 x, y, z 위치를 변경할 수 있다.

QSlider 의 X, Y, Z 값을 변경하면 Qt3DRender::QCamera 클래스에서 제공하는 setPosition() 함수를 이용해 x, y, z 값을 변경할 수 있다. 전체 예제 소스는 Ch08 > 01_Simple3D 디렉토리를 참조하면 된다. 다음은 widget.h 헤더파일 소스코드이다.

```
#ifndef WIDGET_H
#define WIDGET_H

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;
    Qt3DEExtras::Qt3DWindow *m_view;
    Qt3DCore::QEntity *m_rootEntity;
    Qt3DRender::QCamera *m_camera;
    Qt3DEExtras::QOrbitCameraController *camController;
    Qt3DCore::QEntity *m_textEntity;
```

```

void createRootEntry();

QVector3D m_camPos3D;
QVector3D m_camUpVector3D;
QVector3D m_camViewCenter3D;

public slots:
    void setCamPosX_ValueChanged();
    void setCamPosY_ValueChanged();
    void setCamPosZ_ValueChanged();
};

#endif // WIDGET_H

```

Qt3DEextras::Qt3DWindow 클래스는 Qt 3D 의 위젯이다. 3D 그래픽 요소를 랜더링 할 VIEW 영역 위젯이다. 위의 헤더파일 소스코드에서 m_view 오브젝트는 실제 영역 위젯이다. m_rootEntry 는 가장 상위의 QEntity 이다. 다음 은 widget.cpp 소스코드 이다.

```

...
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);

    m_camPos3D      = QVector3D(0, 0, 40.0);
    m_camUpVector3D = QVector3D(0, 1, 0);
    m_camViewCenter3D = QVector3D(0, 0, 0);

    m_view = new Qt3DEextras::Qt3DWindow();
    m_view->defaultFrameGraph()->setClearColor(QColor(77, 77, 77));

    m_rootEntity = new Qt3DCore::QEntity;
    m_view->setRootEntity(m_rootEntity);

    m_camera = m_view->camera();
    m_camera->lens()->setPerspectiveProjection(45.0f, 2.0f, 0.1f, 2000.0f);

    m_camera->setPosition(m_camPos3D);
    m_camera->setUpVector(m_camUpVector3D);
    m_camera->setViewCenter(m_camViewCenter3D);

```

```

Qt3DEngine::QOrbitCameraController *camController;
camController = new Qt3DEngine::QOrbitCameraController(m_rootEntity);

camController->setLinearSpeed( 50.0f );
camController->setLookSpeed( 180.0f );
camController->setCamera(m_camera);

createRootEntry();
QWidget *container = QWidget::createWindowContainer(m_view);
ui->horizontalLayout->addWidget(container);

connect(ui->xSlider, &QSlider::valueChanged,
        this,           &Widget::setCamPosX_ValueChanged);
connect(ui->ySlider, &QSlider::valueChanged,
        this,           &Widget::setCamPosY_ValueChanged);
connect(ui->zSlider, &QSlider::valueChanged,
        this,           &Widget::setCamPosZ_ValueChanged);
}

Widget::~Widget()
{
    delete ui;
}

void Widget::setCamPosX_ValueChanged()
{
    m_camPos3D.setX(ui->xSlider->value());
    m_camera->setPosition(m_camPos3D);
}

void Widget::setCamPosY_ValueChanged()
{
    m_camPos3D.setY(ui->ySlider->value());
    m_camera->setPosition(m_camPos3D);
}

void Widget::setCamPosZ_ValueChanged()
{
    m_camPos3D.setZ(ui->zSlider->value());
    m_camera->setPosition(m_camPos3D);
}

```

```

void Widget::createRootEntry()
{
    // Root entity
    Qt3DEExtras::QExtrudedTextMesh *urlTextMesh;
    urlTextMesh = new Qt3DEExtras::QExtrudedTextMesh();

    urlTextMesh->setText("www.qt-dev.com");

    Qt3DEExtras::QPhongMaterial *urlTextMaterial;
    urlTextMaterial = new Qt3DEExtras::QPhongMaterial();
    urlTextMaterial->setDiffuse(QColor(Qt::green));

    Qt3DCore::QTransform *urlTextTransform = new Qt3DCore::QTransform();
    urlTextTransform->setScale(2.0f);
    urlTextTransform->setTranslation(QVector3D(-10.0f, -4.0f, 15.0f));

    m_textEntity = new Qt3DCore::QEntity(m_rootEntity);
    m_textEntity->addComponent(urlTextMesh);
    m_textEntity->addComponent(urlTextMaterial);
    m_textEntity->addComponent(urlTextTransform);
}
...

```

`m_camera` 는 `Qt3DRender::QCamera` 클래스의 오브젝트이다. 위의 소스코드에서 보는 것과 같이 이 클래스에서 제공하는 `setPerspectiveProjection()` 함수는 3D 영역의 카메라 Perspective 를 표시하기 위한 기능을 제공한다.

첫 번째 인자는 수직 시야 각 (Y축), 두 번째 인자는 수평 시야 각 (X축), 세 번째 인자는 전방 절단면 (Near Clipping Plane), 네 번째 인자는 후방 절단 면 (Far Clipping Plane) 을 표시한다.

`Qt3DRender::QCamera` 클래스의 `setPosition()` 멤버 함수는 3D 공간에서의 카메라 위치를 설정할 수 있다. `setUpVector()` 는 카메라의 Up Vector, `setViewCenter()` 는 카메라의 Center View 위치를 설정할 수 있다.

`createRootEntry()` 는 가장 상위 `QEntity` 인 `m_rootEntity` 에 텍스트 오브젝트를 추가하였다. 그리고 Slot 함수는 GUI 상에서 `QSlider` 의 값이 변경되면 호출되는 함수이다.

9. XML

Qt 는 XML(extensible Markup Language) 을 다룰 수 있는 API를 제공한다. Qt에서 제공하는 XML 모듈을 사용하기 위해서 프로젝트 파일에 다음과 같이 추가한다.

```
QT += xml
```

Qt에서는 XML을 사용하기 위한 방식으로 DOM(Document object model) 방식과 SAX(Simple API for XML) 방식을 제공한다.

DOM 은 XML 내용을 모두 메모리에 트리 형태로 저장한 다음 읽어 들인다. DOM 방식이 메모리에 가지고 있기 때문에 수정, 삭제와 같은 편집이 쉽다

SAX 방식은 DOM 방식에 비해 구현이 쉽다. 그리고 읽어 들인 데이터 또는 분석이 끝난 데이터를 메모리에 저장하지 않기 때문에 수정 및 삭제가 어렵다는 특징이 있다. DOM 방식과 SAX 방식 모두 Qt에서 사용 가능한 다양한 API를 제공한다.

Qt에서 제공하는 QXmlStreamReader 클래스는 XML 포맷의 데이터를 READ하는 기능을 제공하고 QXmlStreamWriter 클래스는 XML 포맷으로 WRITE하는 기능을 제공한다.

QXmlStreamReader 와 QXmlStreamWriter 클래스는 Qt에서 제공하는 QFile 의 오브젝트를 연결해 사용할 수 있다.

다음 예에서 보는것과 같이 QFile 오브젝트를 QXmlStreamReader 클래스에서 제공하는 setDevice() 멤버 함수에 첫 번째 인자로 QFile 오브젝트를 지정하면 된다. 다음은 XML 포맷 데이터 파일의 예이다.

```
<?xml version="1.0" encoding="utf-8"?>
<students>
    <student>
        <firstName>김</firstName>
        <lastName>진아</lastName>
        <grade>3</grade>
    </student>
    <student>
        <firstName>최</firstName>
        <lastName>인수</lastName>
        <grade>4</grade>
    </student>
    ...

```

```
</students>
```

위의 예제와 같이 XML 포맷으로 저장된 파일을 READ하기 위해서 QXmlStreamReader 클래스를 사용할 수 있다. 다음은 QXmlStreamReader 클래스를 이용해 위의 XML 포맷으로 저장된 파일로부터 XML을 읽어오는 예제 소스코드의 일부이다.

```
...
QFile file(QFileDialog::getOpenFileName(this, "Open"));

QXmlStreamReader xmlReader;
xmlReader.setDevice(&file);

QList<Student> students;
xmlReader.readNext();

// XML 파일의 끝까지 읽어 들이기
while (!xmlReader.isEndDocument())
{
    if (xmlReader.isStartElement())
    {
        QString name = xmlReader.name().toString();
        if (name == "firstName" || name == "lastName" || name == "grade")
        {
            QMessageBox::information(this, name, xmlReader.readElementText());
        }
    }
    else if (xmlReader.isEndElement())
    {
        xmlReader.readNext();
    }
}
...
...
```

QXmlStreamReader 클래스의 setDevice() 멤버 함수에 첫 번째 인자는 파일 디이얼로 그로 선택한 파일을 읽어온다. 그리고 읽어온 데이터를 students라는 QList에 저장하기 위한 예이다.

위의 예제에서 보는 것과 같이 Qt에서는 파일에 저장된 XML 포맷 데이터를 읽어올 수 있다. 또한 QXmlStreamWriter 클래스를 이용해 파일에 XML 포맷 데이터를 WRITE 할 수 있다.

```

QXmlStreamWriter stream(&output);

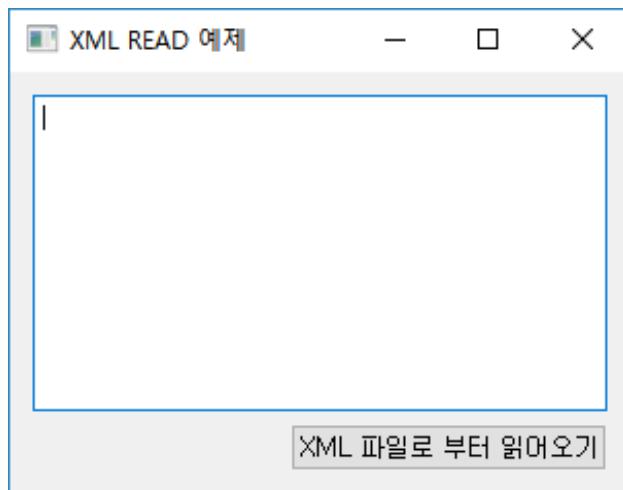
stream.setAutoFormatting(true);
stream.writeStartDocument();
...
stream.writeStartElement("bookmark");
stream.writeAttribute("href", "http://www.qt-dev.com");
stream.writeTextElement("title", "Qt 개발자 커뮤니티");
stream.writeEndElement(); // bookmark
...

```

위의 예제에서 보는 것과 같이 쉽게 QXmlStreamWriter 클래스를 사용할 수 있다. 다음은 실제 예제를 통해 XML을 다루는 방법에 대해 살펴보도록 하자.

- QXmlStreamReader 를 이용해 XML 파일 READ 예제

이번 예제는 파일에 저장된 XML포맷 데이터를 파일로부터 읽어와 QTextEdit 상에 출력하는 예제이며 예제 실행 화면은 다음과 같다.



<그림> 예제 실행 화면

위의 그림에서 보는 것과 같이 [XML 파일로부터 읽어 오기] 버튼을 클릭하면 파일 다이얼로그에서 파일을 선택한다. 선택한 파일은 XML 파일이며 sample.xml 파일은 이 예제 디렉토리에 sample.xml 포 제공되며 XML파일의 내용은 다음과 같다.

```

<?xml version="1.0" encoding="UTF-8"?>
<xbel version="1.0">
    <folder folded="yes">
        <title>Programming</title>

```

```

<bookmark href="http://qt-dev.com">
<title>Qt 개발자 커뮤니티</title>
</bookmark>

<bookmark href="http://www.google.com">
<title>Google</title>
</bookmark>
</folder>
</xbel>

```

위에서 보는 것과 같이 sample.xml 파일에 저장된 XML포맷 데이터를 읽어와 QTextEdit 위젯에 출력하는 예제이며 예제 헤더 파일은 다음과 같다.

```

#include <QWidget>
#include <QFile>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget      *ui;
    QFile           *mReadFile;

public slots:
    void readButtonClicked();

};


```

mReadFile 오브젝트는 XML 파일을 읽어올 QFile 클래스의 오브젝트이다. 그리고 readButtonClicked() 함수는 예제 실행 화면에서 보는 것과 같이 [XML 파일로부터 읽어 오기] 버튼을 클릭하면 호출 되는 Slot 함수이다. 다음 예제 소스코드는 widget.cpp 소스코드 이다.

```

#include "widget.h"
#include "ui_widget.h"

```

```

#include <QFileDialog>
#include <QXmlStreamReader>
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pushButton, &QPushButton::pressed,
            this,           &Widget::readButtonClicked);

    mReadFile = new QFile();
}

Widget::~Widget()
{
    delete ui;
}

void Widget::readButtonClicked()
{
    QString fName = QFileDialog::getOpenFileName(this, "Open XML File",
                                                QDir::currentPath(),
                                                "XML Files (*.xml)");
    mReadFile->setFileName(fName);

    if(!QFile::exists(fName)) {
        ui->textEdit->setText("파일이 존재하지 않습니다. ");
        return;
    }

    if(!mReadFile->open(QIODevice::ReadOnly)) {
        ui->textEdit->setText("파일 Open 실패.");
        return;
    }

    QXmlStreamReader reader(mReadFile);
    QString inputData;
    while(!reader.atEnd())
    {
        reader.readNext();
        if(!reader.text().isEmpty()) {
            QString data = reader.text().toString();

```

```

        data.replace('\n', "");
        data.replace('\t', "");

        if(data.length() > 0)
            inputData.append(data).append("<br>");
    }

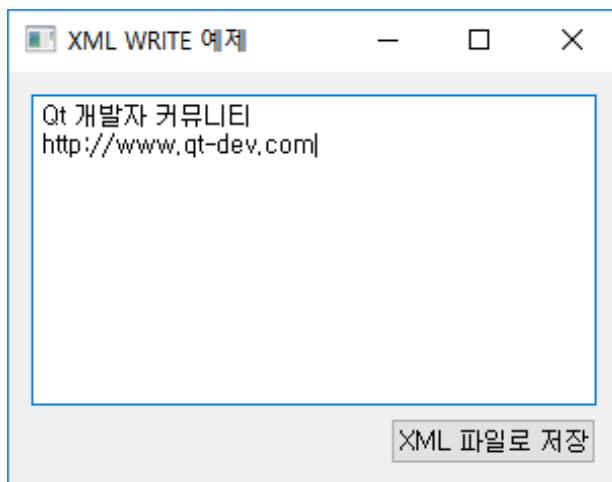
    ui->textEdit->setText(inputData);
}

```

위의 예제 소스코드에서 보는 것과 같이 `readButtonClicked()` Slot 함수가 호출되면 파일 다이얼로그가 로딩된다.

파일 다이얼로그에서 XML파일을 선택하면 `QXmlStreamReader` 클래스를 이용해 XML데이터를 읽어온다. `QXmlStreamReader` 클래스의 `readNext()` 멤버 함수는 XML 태그를 차례대로 읽어오는 기능을 제공하며 `text()` 멤버 함수는 실제 태그의 텍스트를 읽어온다. 그리고 읽어온 텍스트를 `QTextEdit`에 출력하기 위해 `QString`에 저장한다. 저장 후 XML 파일 READ가 끝난 후 저장된 `QString`에 저장된 텍스트를 `QTextEdit`에 출력한다. 예제 전체 소스는 Ch09 > 01_ReaderExample 디렉토리를 참조하면 된다.

- `QXmlStreamWriter` 를 이용해 XML 파일에 저장하는 예제 구현



<그림> 예제 실행 화면

이번 예제는 XML포맷 데이터를 파일에 저장하는 예제이다. 아래 그림에서 보는 것과 같이 QTextEdit 상에 출력된 데이터를 읽어와 XML 포맷 데이터로 변경 후 파일에 저장하는 예제이다.

위의 예제 실행 화면에서 보는 것과 같이 [XML 파일로 저장] 버튼을 클릭하면 "C:/output.xml" 파일에 XML을 저장한다. 다음은 widget.h 소스코드 파일이다.

```

#include <QWidget>
#include <QFile>
namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;
    QFile *mWriteFile;
    QList<QString> mOriData;

public slots:
    void writeButtonClicked();
};

```

mWriteFile 오브젝트는 QFile 의 오브젝트이다. mOriData 는 QTextEdit 에 출력된 데이터가 저장된 변수이다. 다음 예제 소스코드는 widget.cpp 소스코드 이다.

```

...
#include <QFileDialog>
#include <QXmlStreamReader>
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pushButton, &QPushButton::pressed,
            this,           &Widget::writeButtonClicked);

    mWriteFile = new QFile();
    mOriData.append("Qt 개발자 커뮤니티");
    mOriData.append("http://www.qt-dev.com");

    for(int i = 0 ; i < mOriData.count() ; i++)
        ui->textEdit->append(mOriData.at(i));
}

Widget::~Widget()

```

```

{
    delete ui;
}

void Widget::writeButtonClicked()
{
    QFile file("C:/output.xml");
    file.open(QIODevice::WriteOnly);

    QXmlStreamWriter xmlWriter(&file);
    xmlWriter.setAutoFormatting(true);
    xmlWriter.writeStartDocument();
    xmlWriter.writeStartElement("Qt");
    xmlWriter.writeStartElement("Info");
    xmlWriter.writeTextElement("Name", mOriData.at(0));
    xmlWriter.writeTextElement("URL", mOriData.at(1));
    xmlWriter.writeEndElement(); // Info End 테그
    xmlWriter.writeEndElement(); // Qt End 테그
    file.close();
}
...

```

`writeButtonClicked()` 함수에서 `QXmlStreamWriter` 클래스의 `writeStartElement()` 함수는 XML 시작 태그이다. 그리고 `writeTextElement()` 함수는 테그에 저장할 텍스를 입력하면 된다. 마지막으로 `writeEndElement()` 함수는 테그를 마지막 테그를 실제 파일에 저장한다. 위와 같이 작성 후 파일을 닫으면 “`C:/output.xml`” 파일에 XML데이터가 저장되며 결과는 다음과 같다.

```

<?xml version="1.0" encoding="UTF-8"?>
<Qt>
    <Info>
        <Name>Qt 개발자 커뮤니티</Name>
        <URL>http://www.qt-dev.com</URL>
    </Info>
</Qt>

```

작성한 예제 전체 소스코드는 Ch09 > 02_WriteExample 디렉토리를 참조하면 된다.

- DOM 방식의 예제 구현

이번 예제는 DOM(Document object model) 방식을 이용해 파일로부터 XML포맷 데이

터를 읽어오는 예제이다. 아래 그림에서 보는 것과 같이 dom.xml 파일명으로 저장한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<Building>
    <Korea ID="0" Name="Korea Building 0">
        <Korea ID="0" Name="Room 0"/>
        <Korea ID="0" Name="Room 1"/>
        <Korea ID="0" Name="Room 2"/>
    </Korea>
    <Korea ID="1" Name="Korea Building 1">
        <Korea ID="1" Name="Sub Room 0"/>
        <Korea ID="1" Name="Sub Room 1"/>
        <Korea ID="1" Name="Sub Room 2"/>
    </Korea>
</Building>
```

위의 XML 데이터를 파일로부터 읽어 오기 위해 QDomDocument 클래스를 사용할 수 있으며 소스코드는 다음과 같이 작성한다.

```
#include <QApplication>
#include <QtXml>
#include <QDebug>

void retrievElements(QDomElement root, QString tag, QString att)
{
    QDomNodeList nodes = root.elementsByTagName(tag);

    for(int i = 0; i < nodes.count(); i++)
    {
        QDomNode elm = nodes.at(i);
        if(elm.isElement()){
            QDomElement e = elm.toElement();
            qDebug() << "Attribute : " << e.attribute(att);
        }
    }
}

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    QDomDocument document;
    QFile file("c:/dom.xml");
```

```

if(!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
    qDebug() << "Failed to open file.";
    return -1;
} else {
    if(!document.setContent(&file)) {
        qDebug() << "Failed to reading.";
        return -1;
    }
    file.close();
}

QDomElement root = document.firstChildElement();
retrievElements(root, "Korea", "Name");

return a.exec();
}

```

`retrievElements()` 함수의 첫 번째 인자는 QDomElement 클래스의 오브젝트이다. 두 번째 인자는 XML 데이터에서 “Korea” 태그, 세 번째 인자는 “Attr” 속성(Attribute)을 지정한다. 지정한 태그와 속성을 `qDebug()` 를 이용해 Console 출력한다.

```

노드의 개수 = 8
속성 : "Korea Building 0"
속성 : "Room 0"
속성 : "Room 1"
속성 : "Room 2"
속성 : "Korea Building 1"
속성 : "Sub Room 0"
속성 : "Sub Room 1"
속성 : "Sub Room 2"

```

예제 전체 소스코드는 Ch09 > 03_DomExample 디렉토리를 참조하면 된다.

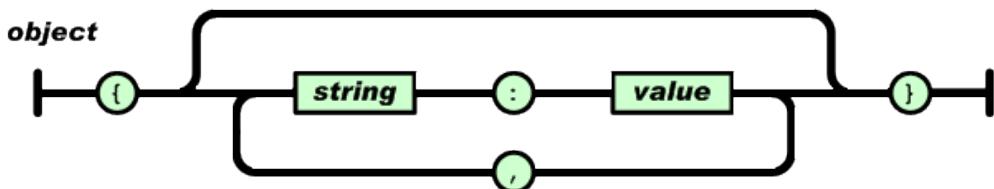
10. JSON

JSON (JavaScript Object Notation)의 사용 목적은 XML과 동일하다. 차이점은 경량의 DATA 교환 방식으로 XML보다 데이터 사용 용량이 작다는 장점을 가지고 있다. 예를 들어 XML은 태그에 필요한 단어를 저장하기 위한 많은 데이터가 낭비된다. 하지만 JSON은 태그 대신에 " 와 [] 기호를 사용하기 때문에 XML에 비해 데이터를 저장하는 데 공간을 절약 할 수 있다. 다음은 XML과 JSON 데이터 포맷을 비교한 내용이다.

XML	JSON
<pre><?xml version="1.0" encoding="UTF-8"?> <note> <to>Tove</to> <from>Jani</from> <heading>Reminder</heading> </note> <note> <to>Tove</to> <from>Jani</from> <heading>Reminder</heading> </note></pre>	<pre>{ "FirstName": "John", "Age": 43, "Address": { "Street": "Downing Street 10", "Country": "Great Britain" }, "Phone numbers": ["+44 1234567", "+44 2345678"] }</pre>

<그림> XML 과 JSON

위의 예제에서 보는 것과 같이 JSON은 XML과 비교해 이 기종 간의 데이터 통신 시 JSON을 이용하면 XML보다 데이터를 적게 사용할 수 있다는 잠정이 있다. JSON은 string/value 형태의 쌍(Pair)으로 저장된다.



<그림> JSON 구조

string과 value를 구분하기 위해 ":"을 사용하며 하나의 쌍은 ","로 구분한다. 다음 예제 소스코드는 JSON 포맷 데이터이다.

```
{
    "FirstName": "John", "LastName": "Doe", "Age": 43,
    "Address": {
        "Street": "Downing Street 10", "City": "Seoul", "Country": "Korea"
    }
}
```

```
},
"Phone numbers": [ "+44 1234567", "+44 2345678" ]
}
```

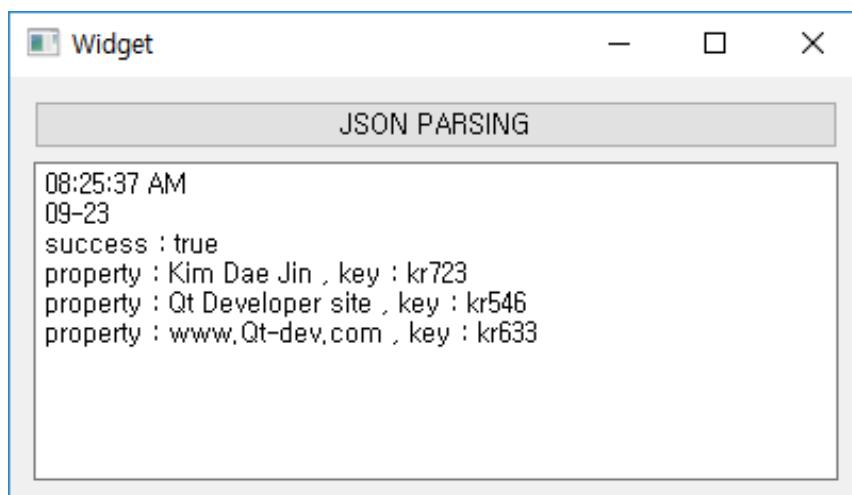
위의 예제에서 보는 것과 같이 Qt에서 제공하는 JSON 모듈은 Bool, Double, String, Array, Object, Null 과 같은 6가지의 데이터 타입을 사용할 수 있다. { (좌 중괄호)로 시작하고 } (우 중괄호)로 끝내어 표현할 수 있다.

그리고 [(대괄호) 와] (대괄호) 를 시작과 끝으로 사용할 수 있다. 또한 중괄호와 대괄호는 중첩된 구조로 사용할 수 있다. 실제 예제를 통해서 JSON 의 사용 방법을 살펴보도록 하자.

● JSON Parsing 예제

이번 예제는 Sample.json 파일로부터 JSON 데이터를 가져와 GUI에서 QPlainTextEdit 위젯에 출력하는 예제이다. 예제 Sample.json 파일의 내용은 다음과 같다.

```
{
    "time": "08:25:37 AM", "date": "09-23", "success": true,
    "properties": [
        { "ID": 1001, "PropertyName": "Kim Dae Jin", "key": "kr723" },
        { "ID": 1002, "PropertyName": "Qt Developer site", "key": "kr546" },
        { "ID": 1003, "PropertyName": "www.Qt-dev.com", "key": "kr633" }
    ]
}
```



<그림> 예제 실행 화면

위의 그림에서와 같이 [JSON PARSING]버튼을 클릭하면 GUI 상에서 QPlainTextEdit 위젯에 결과를 출력한다. 다음은 이 예제의 widget.h 소스코드이다.

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();

private:
    Ui::Widget *ui;
    void parseJSON(const QString &data);
    void addText(const QString &addLine);

private slots:
    void slotPbtJSONParser();
};

#endif // WIDGET_H
```

위의 예제에서 slotPbtJSONParser() Slot 함수는 [JSON PARSING] 버튼을 클릭하면 호출되는 Slot 함수이다. parseJSON() 함수는 JSON 파일에서 읽어온 데이터를 Parsing 하는 함수이다. addText() 함수는 parseJSON() 함수에 의해 추출한 데이터를 위젯에 출력하기 위한 함수이다. 다음은 widget.cpp 소스코드이다.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pbtJSONParser, SIGNAL(pressed()),
            this,                      SLOT(slotPbtJSONParser()));
}

Widget::~Widget()
```

```

{
    delete ui;
}

void Widget::slotPbtJSONParser()
{
    QString inputFilePath;
    InputFilePath = QFileDialog::getOpenFileName(this,
                                                tr("Open File"),
                                                QDir::currentPath(),
                                                tr("JSON Files (*.json)"));

    QFile file(inputFilePath);
    if(!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        qDebug() << "Failed to open file.";
        return;
    }

    QString data = file.readAll();
    file.close();

    ui->textEdit->clear();
    parseJSON(data);
}

void Widget::parseJSON(const QString &data)
{
    QJsonDocument jsonResponse = QJsonDocument::fromJson(data.toLocal8Bit());
    QJsonObject jsonObj = jsonResponse.object();

    addText(jsonObj[ "time" ].toString().append("\n"));
    addText(jsonObj[ "date" ].toString().append("\n"));

    if(jsonObj[ "success" ].toBool() == true)
        addText(QString("success : true \n"));
    else
        addText(QString("success : false \n"));

    QJsonArray jsonArray = jsonObj[ "properties" ].toArray();
    foreach (const QJsonValue & value, jsonArray) {
        QJsonObject obj = value.toObject();
        QString property = obj[ "PropertyName" ].toString();

```

```

QString key      = obj[ "key" ].toString();

QString arrayData; = QString("property : %1 , key : %2 \n")
                    .arg(property).arg(key);
addText(arrayData);
}

void Widget::addText(const QString &addLine)
{
    ui->textEdit->insertPlainText(addLine);
}

```

parseJSON() 함수의 첫 번째 인자는 JSON 파일로부터 읽어온 원본 JSON 데이터이다. 읽어온 원본 데이터는 QJsonDocument 의 jsonResopnse 오브젝트에서 사용한다.

QJsonDocument::fromJson() 함수에서 첫 번째 인자는 QByteArray 이다. 따라서 QString 원본데이터를 QByteArray로 바꾸기 위해 QByteArray 클래스의 toLocal8Bit() 멤버 함수를 사용하였다.

QJsonDocument 클래스의 jsonResopnse 오브젝트는 fromJson() 함수로 가공한 JSON 원본데이터에 대한 정보를 저장한다. 저장된 데이터를 QJsonObject 클래스를 이용해 데이터를 추출할 수 있도록 QJsonDocument 클래스의 object() 멤버 함수를 이용해 QJsonObject 타입을 넘겨준다.

그리고 실제 데이터에 접근하기 위해서는 QJsonObject 클래스 타입을 다시 QJsonArray 로 넘겨준다. 그리고 각각의 데이터를 접근하기 위해 foreach 문에서 사용한 것과 같이 QJsonValue 클래스를 사용하면 된다. 전체 예제 소스코드와 Sample.json 파일은 Ch09 > 01_JSON_Parser 디렉토리를 참조하면 된다.

11. Qt Chart

Qt는 데이터를 그래프 차트로 표현할 수 있는 그래픽 API를 제공한다. Area chart, Pie chart, Line chart, Bar Chart, Spline Chart, Scatter chart 등 다양한 종류의 차트를 제공한다.



<그림> Chart 예제 실행 화면

Qt에서 제공하는 Chart API를 사용하기 위해서는 프로젝트파일(.pro)에 Qt Chart 모듈을 다음과 같이 추가해야 한다.

```
QT += charts
```

Qt Chart 모듈은 사용하기 API로 구성되어 있으며 C++과 QML에서 사용 가능하다. 예를 들어 QLineSeries 클래스를 이용해 다음과 같이 소스코드를 작성할 수 있다.

```
QLineSeries* series = new QLineSeries();
series->add(0, 6);
series->add(2, 4);
...
chartView->chart()->addSeries(series);
chartView->chart()->createDefaultAxes();
```

X, Y 데이터를 추가하기 위해서 QLineSerise 클래스의 add() 멤버 함수를 사용해 데이터를 추가 할 수 있다. add() 멤버 함수의 첫 번째 인자는 X값, 두 번째 인자는 Y축의

값이다. 다음은 QML로 Pie chart를 사용한 예이다.

```
import QtQuick 2.0
import QtCharts 2.0

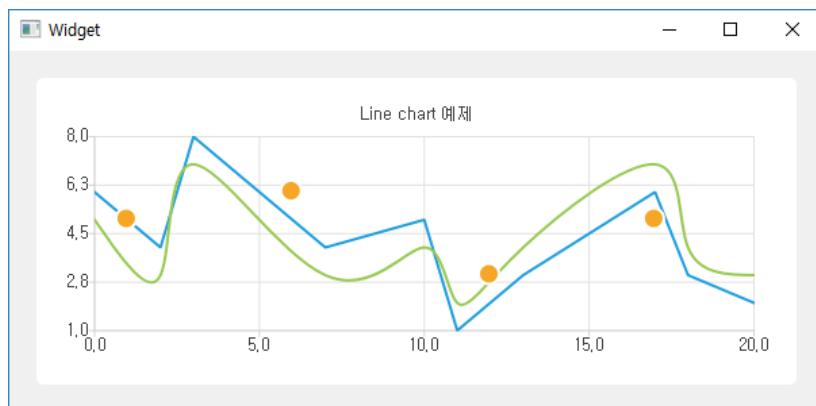
ChartView {
    width: 400
    height: 300
    theme: ChartView.ChartThemeBrownSand
    antialiasing: true

    PieSeries {
        id: pieSeries
        PieSlice { label: "eaten"; value: 94.9 }
        PieSlice { label: "not yet eaten"; value: 5.1 }
    }
}
```

위의 예제 소스코드는 Pie chart를 QML로 작성한 예제 소스코드이다. C++과 QML 모두 사용 가능하다. Qt Chart 모듈은 차트를 표시할 때 애니메이션 효과를 사용할 수 있다. Qt Chart 모듈은 다양한 종류의 그래프 Chart를 지원한다. 따라서 예제를 통해 살펴보도록 하자.

● Line chart

이번 예제에서는 X, Y 축으로 Chart를 표시하는 예제이다. 3가지 종류의 데이터를 표시하였다. 첫 번째는 직선형 그래프, 두 번째는 곡선형 그래프 그리고 Scatter 방식의 그래프로 표시한 방법을 사용하였다.



<그림> 예제 실행 화면

위의 그림에서 보는 것과 같이 직선형은 QLineSeries 클래스를 이용해 표시할 수 있다. 두 번째 곡선형은 QSplineSeries 클래스를 사용하면 된다.

마지막으로 Scatter 방식은 QScatterSeries 클래스를 사용하면 된다. 다음은 예제 소스 코드이다. 전체 예제 소스는 Ch11 > 01_LineChart 디렉토리를 참조하면 된다.

```
...
#include <QHBoxLayout>
#include <QtCharts>
#include <QGraphicsView>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);

    QLineSeries* series = new QLineSeries();
    series->append(0, 6);
    series->append(2, 4);
    series->append(3, 8);
    series->append(7, 4);
    series->append(10, 5);
    *series << QPointF(11, 1) << QPointF(13, 3) << QPointF(17, 6)
        << QPointF(18, 3) << QPointF(20, 2);

    QSplineSeries* series1 = new QSplineSeries();
    series1->append(0, 5);
    series1->append(2, 3);
    series1->append(3, 7);
    series1->append(7, 3);
    series1->append(10, 4);
    *series1 << QPointF(11, 2) << QPointF(13, 4) << QPointF(17, 7)
        << QPointF(18, 4) << QPointF(20, 3);

    QScatterSeries* series2 = new QScatterSeries();
    *series2 << QPointF(1,5) << QPointF(6,6) << QPointF(12,3)
        << QPointF(17,5);

    QChart *chart = new QChart();
    chart->legend()->hide(); // 범례 숨김
    chart->addSeries(series); // series 추가
    chart->addSeries(series1);
    chart->addSeries(series2);
```

```

chart->createDefaultAxes(); // 기본 축 생성
chart->setTitle("Line chart 예제");

QChartView *chartView = new QChartView(chart);
chartView->setRenderHint(QPainter::Antialiasing);

QHBoxLayout *hLay = new QHBoxLayout();
hLay->addWidget(chartView);

setLayout(hLay);
}
...

```

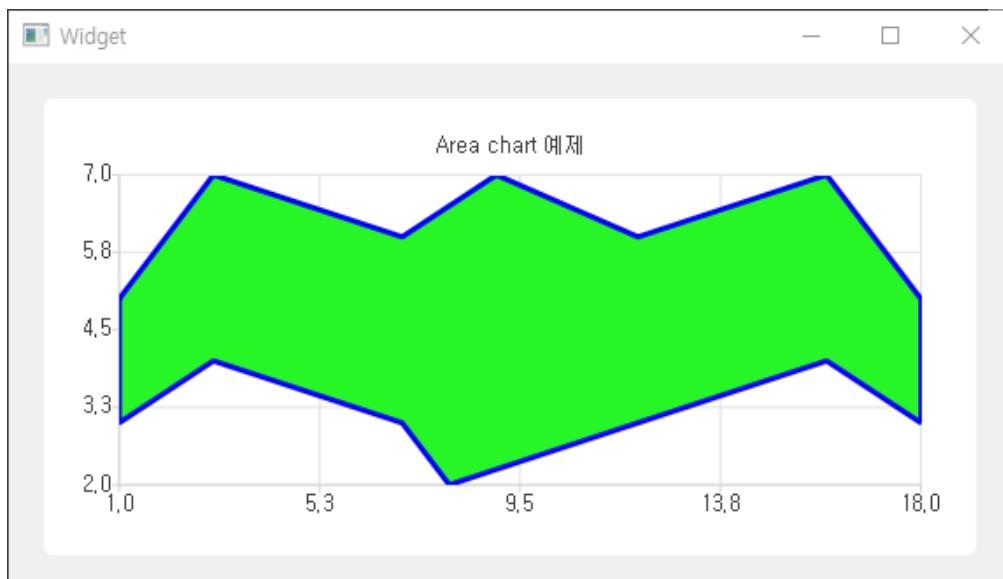
QLineSeries, QSplineSeries 그리고 QScatterSeries 클래스의 오브젝트에 데이터를 삽입하고 QChart 클래스의 addSeries() 멤버 함수를 이용해 각 그래프를 추가하면 된다.

그리고 QChartView 클래스 선언 시 생성자의 첫 번째 인자로 QChart 클래스의 오브젝트를 선언하면 된다.

QChartView 클래스의 setRenderHint() 함수에 사용한 QPainter::Antialiasing 옵션은 안티알리아싱을 사용하기 위한 옵션이다.

● Area chart

Area chart는 X, Y 축의 그래프 범위에서 어떤 특정 범위를 표시하는데 사용할 수 있으며 Qt에서 제공하는 QAreaSeries 클래스를 사용하면 된다.



<그림> 예제 실행 화면

QAreaSeries 클래스는 X, Y Chart 상에 Area를 표시할 수 있다. 그래프 내부에 Color 설정 이외에 그라디언트를 사용할 수 있다. 다음은 Area Chart 의 예제소스코드이다. 예제 전체 소스코드는 Ch11 > 02_AreaChart를 참고하면 된다.

```
...
#include <QHBoxLayout>
#include <QtCharts>
#include <QGraphicsView>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    QLineSeries *series1 = new QLineSeries();
    QLineSeries *series2 = new QLineSeries();

    *series1 << QPointF(1, 5) << QPointF(3, 7) << QPointF(7, 6)
        << QPointF(9, 7) << QPointF(12, 6) << QPointF(16, 7)
        << QPointF(18, 5);

    *series2 << QPointF(1, 3) << QPointF(3, 4) << QPointF(7, 3)
        << QPointF(8, 2) << QPointF(12, 3) << QPointF(16, 4)
        << QPointF(18, 3);

    QAreaSeries *series = new QAreaSeries(series1, series2);
    series->setName("Area Data");
    QPen pen(Qt::blue);
    pen.setWidth(3);
    series->setPen(pen);

    QLinearGradient gradient(QPointF(0, 0), QPointF(0, 1));
    gradient.setColorAt(0.0, 0x3cc63c);
    gradient.setColorAt(1.0, 0x26f626);
    series->setBrush(gradient);

    QChart *chart = new QChart();
    chart->legend()->hide();
    chart->addSeries(series);

    chart->createDefaultAxes();
    chart->setTitle("Area chart 예제");
```

```

QChartView *chartView = new QChartView(chart);
chartView->setRenderHint(QPainter::Antialiasing);

QHBoxLayout *hLay = new QHBoxLayout();
hLay->addWidget(chartView);

setLayout(hLay);
}

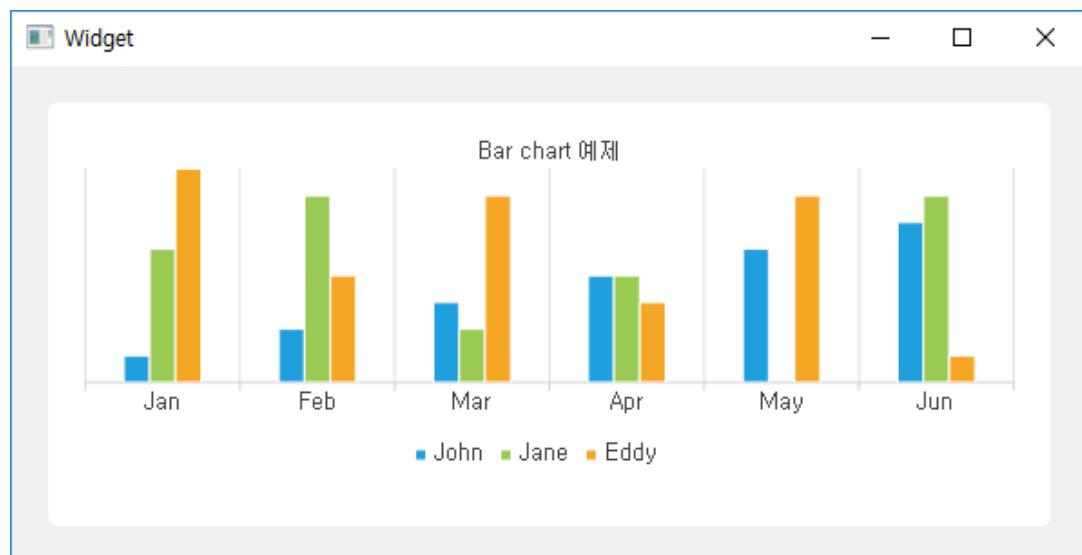
...

```

QAreaSeries 클래스의 setBrush() 멤버 함수의 첫 번째 인자로 QLinearGradient 클래스 오브젝트를 사용하면 그라디언트를 사용할 수 있다.

- Bar chart

Bar chart 는 QBarSeries 클래스를 이용해 X, Y 축으로 막대 그래프를 표시할 수 있는 기능을 제공한다.



<그림> 예제 실행 화면

위의 그림에서 보는 것과 같이 막대 그래프를 사용하기 위해서 QBarSeries 클래스를 사용하면 된다. 그리고 QBarSeries 클래스에 데이터를 추가하기 위해서 QBarSet 클래스를 사용하면 된다. 다음은 widget.cpp 예제 소스코드 일부이다. 전체 소스코드는 Ch11 > 03_Barchart 를 참고하면 된다.

```
...
```

```

#include <QHBoxLayout>
#include <QtCharts>
#include <QGraphicsView>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    QBarSet *set1 = new QBarSet("John");
    QBarSet *set2 = new QBarSet("Jane");
    QBarSet *set3 = new QBarSet("Eddy");

    *set1 << 1 << 2 << 3 << 4 << 5 << 6;
    *set2 << 5 << 7 << 2 << 4 << 0 << 7;
    *set3 << 8 << 4 << 7 << 3 << 7 << 1;

    QBarSeries *series = new QBarSeries();
    series->append(set1);
    series->append(set2);
    series->append(set3);

    QStringList categories;
    categories << "Jan" << "Feb" << "Mar" << "Apr" << "May" << "Jun";

    QBarCategoryAxis *axis = new QBarCategoryAxis();
    axis->append(categories);

    QChart *chart = new QChart();
    chart->addSeries(series);
    chart->setTitle("Bar chart 예제");
    chart->legend()->setVisible(true);
    chart->legend()->setAlignment(Qt::AlignBottom);

    chart->addAxis(axis, Qt::AlignBottom);
    series->attachAxis(axis);

    QChartView *chartView = new QChartView(chart);
    chartView->setRenderHint(QPainter::Antialiasing);

    QHBoxLayout *hLay = new QHBoxLayout();
    hLay->addWidget(chartView);

    setLayout(hLay);
}

```

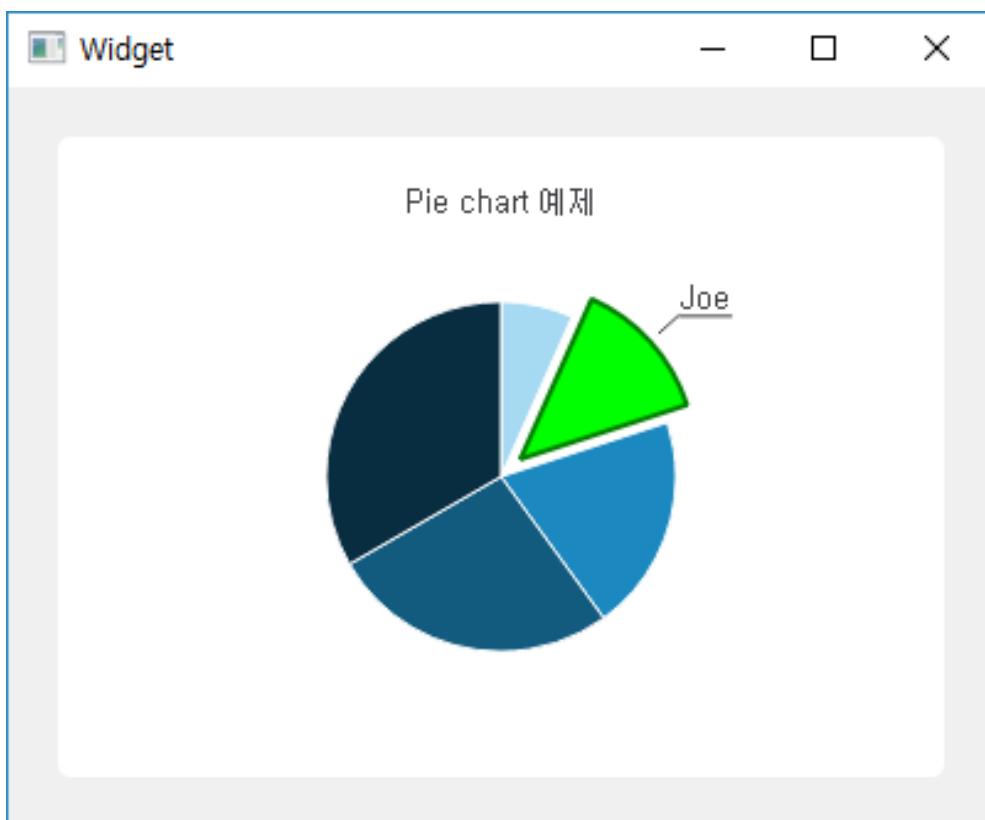
```
}
```

```
...
```

Chart 상에 X, Y축에 Jan, Feb, Mar 등 카테고리를 사용하기 위해서 QBarCategoryAxis를 사용할 수 있다. QBarCategoryAxis 클래스의 append() 멤버 함수의 첫 번째 인자로 데이터를 전달하면 위의 그림에서 보는 것과 같이 카테고리상에 데이터를 표시할 수 있다.

- Pie chart

Pie chart 는 QPieSeries 클래스를 이용해 Pie chart를 그래프로 표시할 수 있다. 다음 그림에서 보는 것과 같이 Pie 형태로 데이터를 표시하기 위해서 QPieSeries 클래스를 이용하면 된다. 데이터를 삽입하기 위해서 append() 멤버 함수를 사용하면 된다. append() 함수에서 첫 번째 인자는 각 데이터의 이름을 표시할 수 있다.



<그림> 예제 실행 화면

그리고 두 번째 인자는 데이터의 크기이다. 다음은 예제 소스코드이다. 전체 소스코드

는 Ch11 > 04_PieChart 디렉토리를 참조하면 된다.

```
...
#include <QHBoxLayout>
#include <QtCharts>
#include <QGraphicsView>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);

    QPieSeries *series = new QPieSeries();
    series->append("Jane", 1); // 이름과 비율을 함께 삽입
    series->append("Joe", 2);
    series->append("Andy", 3);
    series->append("Barbara", 4);
    series->append("Axel", 2);

    // 두 번째 아이템 선택
    QPieSlice *slice = series->slices().at(1);

    slice->setExploded(); // 아이템 분리
    slice->setLabelVisible(); // 아이템 네임 visible

    slice->setPen(QPen(Qt::darkGreen, 2));
    slice->setBrush(Qt::green);

    QChart *chart = new QChart();
    chart->legend()->hide(); // 범례 숨김
    chart->addSeries(series); // series 추가
    chart->createDefaultAxes(); // 기본 축 생성
    chart->setTitle("Pie chart 예제");

    QChartView *chartView = new QChartView(chart);
    chartView->setRenderHint(QPainter::Antialiasing);

    QHBoxLayout *hLay = new QHBoxLayout();
    hLay->addWidget(chartView);
    setLayout(hLay);
}

...
```

12. Qt Data Visualization

Qt Data Visualization 모듈은 데이터를 3D 기반으로 데이터를 시각화하는 기능을 제공한다. 막대 그래프, 분산 그래프 등과 같이 3D로 데이터를 시각화 할 수 있다. Qt Data Visualization 모듈은 OpenGL 을 기반으로 하드웨어 가속을 사용하기 때문에 랜더링 속도가 빠르다.



<그림> Qt Data Visualization 모듈을 이용한 예제 실행 화면

Qt Data Visualization 모듈은 데이터를 3D로 표시하는 형태로 막대 그래프, Scatter 그래프, Surface (예를 들어 육지와 바다의 해수면 표시) 그래프로 3D 데이터를 시각화 할 수 있다.

Qt Data Visualization 모듈은 Camera 의 개념을 사용해 사용자가 그래프의 방향을 상/하/좌/우로 이동 할 수 있고 확대/축소 할 수 있으며 특정 데이터를 마우스로 클릭해 시각화 할 수 있는 기능을 제공한다.

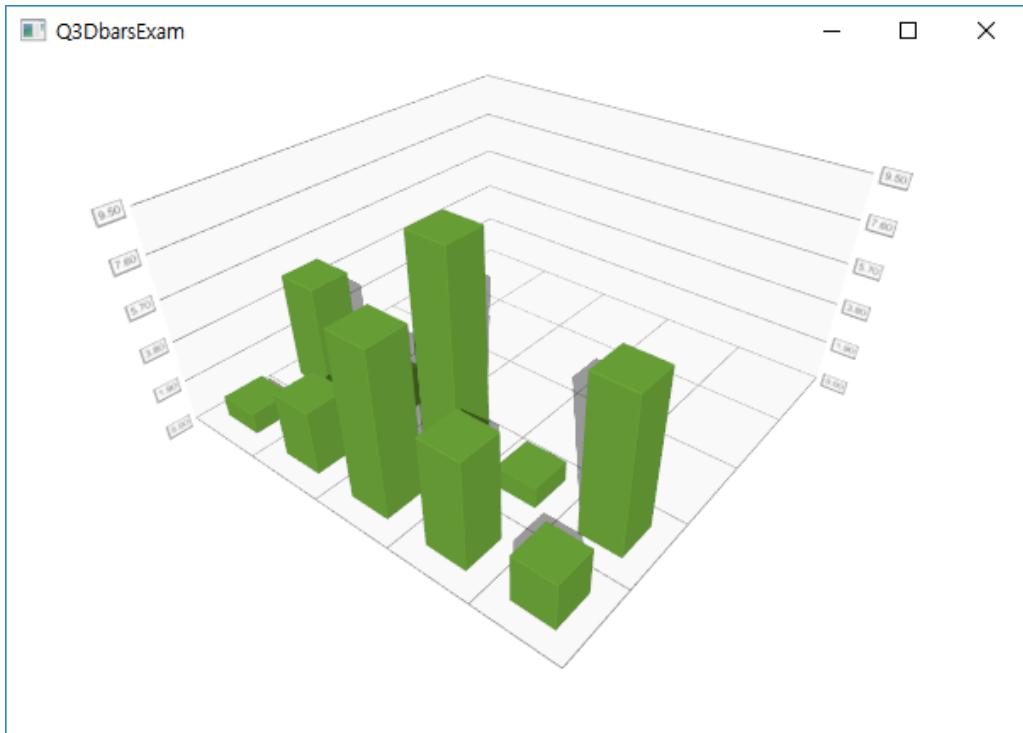
예를 들어 특정 범례의 데이터를 컬러를 변경하는 것과 같은 기능을 제공한다. Qt Data Visualization 모듈을 사용하기 위해서는 프로젝트 파일에 다음과 같이 추가해야 한다.

```
QT += datavisualization
```

Qt Data Visualization 모듈에서 제공하는 막대 그래프, Scatter 그래프, Surface 그래프를 실제 예제를 통해 살펴보도록 하자.

- Bar 3D 예제

Q3DBar 클래스를 이용하면 3D 막대 그래프를 표시할 수 있다. Q3DBar 클래스는 화면을 상/하/좌/우 및 확대/축소를 자유롭게 할 수 있다. 다음은 예제 실행 화면이다.



<그림> 예제 실행 화면

Q3DBars 클래스는 행(Row)의 범위를 설정하기 위해 `rowAxis()->setRange()` 멤버 함수를 사용할 수 있다. 컬럼의 범위를 설정하기 위해서 `columnAxis()->setRange()` 멤버 함수를 사용할 수 있다. 다음 예제는 Bar 3D 구현 소스코드이다. 전체 소스 코드는 Ch12 > 01_BarsExample 디렉토리를 참조하면 된다.

```
#include <QApplication>
#include <QtDataVisualization>

using namespace QtDataVisualization;

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
```

```

QBarDataRow *data1 = new QBarDataRow;
*data1 << 1.0f << 3.0f << 7.5f << 5.0f << 2.2f;

QBarDataRow *data2 = new QBarDataRow;
*data2 << 5.0f << 2.0f << 9.5f << 1.0f << 7.2f;

QBar3DSeries *series = new QBar3DSeries;
series->dataProxy()->addRow(data1);
series->dataProxy()->addRow(data2);

Q3DBars *bars = new Q3DBars;
bars->setFlags(bars->flags() ^ Qt::FramelessWindowHint);

bars->scene()->activeCamera()->setCameraPreset(
    Q3DCamera::CameraPresetFront);

bars->rowAxis()->setRange(0,4);
bars->columnAxis()->setRange(0,4);

bars->addSeries(series);
bars->setFloorLevel(0);

bars->setGeometry(50,50,600,400);
bars->show();

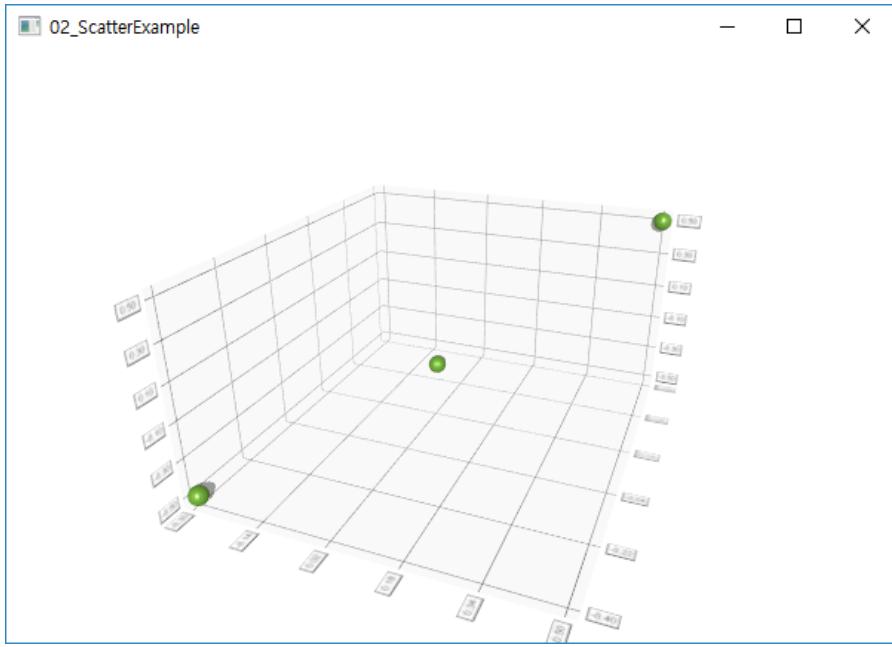
return a.exec();
}

```

3D Bar 에 데이터를 추가 하기 위해서 QBarDataRow 클래스와 QBar3DSeries를 사용하면 된다. 데이터를 추가한 QBarDataRow 클래스의 오브젝트를 QBar3DSeries 클래스 오브젝트 추가 해야 한다. 그리고 Q3DBars 클래스의 addSeries() 멤버 함수의 인자에 QBar3DSeries 클래스 오브젝트를 넘겨주면 된다.

● Scatter 3D 예제

Scatter 그래프는 특정 X, Y 좌표에 점을 표시하는 것과 같이 분산 형태의 그래프 종류이다. Qt 에서는 Scatter 그래프를 3D 로 표현하기 위해서 Q3DScatter 클래스를 제공한다. 다음 그림은 예제 실행 화면 이다.



<그림> 예제 실행 화면

Q3DScatter 클래스에 QScatter3DSeries 와 QScatterdataArray 클래스를 이용해 데이터를 추가할 수 있다. 다음 예제는 Scatter 예제 소스코드이며 전체 소스코드는 Ch12 > 02_ScatterExample 디렉토리를 참조하면 된다.

```
#include <QApplication>
#include <QtDataVisualization>
using namespace QtDataVisualization;

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    QScatter3DSeries *series = new QScatter3DSeries;
    QScatterdataArray data;
    data << QVector3D(0.5f, 0.5f, 0.5f) << QVector3D(-0.3f, -0.5f, -0.4f)
        << QVector3D(0.0f, -0.3f, 0.2f);

    Q3DScatter *scatter = new Q3DScatter;
    scatter->setFlags(scatter->flags() ^ Qt::FramelessWindowHint);
    scatter->scene()->activeCamera()->setCameraPreset(
        Q3DCamera::CameraPresetFront);

    series->dataProxy()->addItems(data);
}
```

```

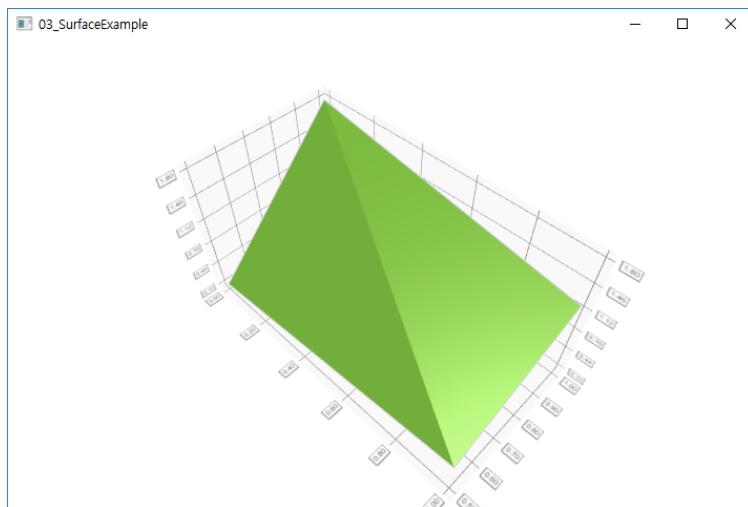
scatter->addSeries(series);
scatter->setGeometry(50,50,600,400);
scatter->show();

return a.exec();
}

```

- Surface 3D 예제

Surface 그래프는 단면을 X, Y 축으로 단면을 표시할 수 있다. 예를 들어 육지의 표면을 표시하는데 사용할 수 있다. 다음 그림은 예제 실행 화면이다.



<그림> 예제 실행 화면

Surface 형태의 그래프를 표시하기 위해서 Q3DSurface 그래프를 사용할 수 있다. 그리고 데이터를 추가하기 위해 QSurfacedataArray 와 QSurfaceDataRow 클래스를 제공한다. 다음은 예제 소스코드이며 전체 소스는 Ch12 > 03_SurfaceExample 디렉토리를 참조하면 된다.

```

#include <QApplication>
#include <QtDataVisualization>

using namespace QtDataVisualization;

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

```

```

Q3DSurface surface;
surface.setFlags(surface.flags() ^ Qt::FramelessWindowHint);

QSurfacedataArray *data = new QSurfacedataArray;
QSurfaceDataRow *dataRow1 = new QSurfaceDataRow;
QSurfaceDataRow *dataRow2 = new QSurfaceDataRow;

*dataRow1 << QVector3D(0.0f, 0.1f, 0.5f) << QVector3D(1.0f, 0.5f, 0.5f);
*dataRow2 << QVector3D(0.0f, 1.8f, 1.0f) << QVector3D(1.0f, 1.2f, 1.0f);
*data << dataRow1 << dataRow2;

QSurface3DSeries *series = new QSurface3DSeries;
series->dataProxy()->resetArray(data);
surface.addSeries(series);
surface.show();
return a.exec();
}

```

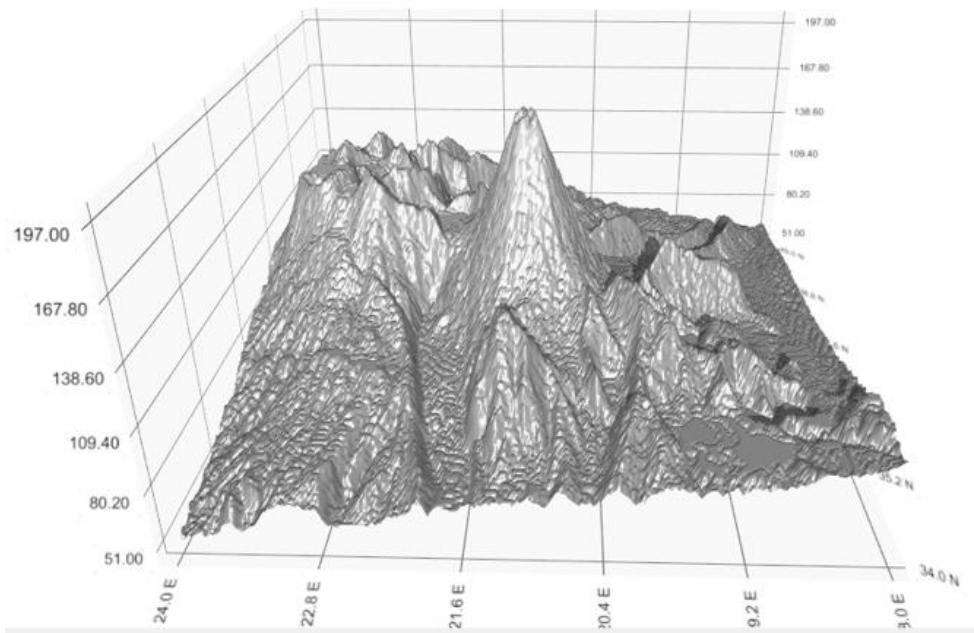
- 등고선 추출 이미지를 이용한 등고선 3D Surface 표시 예제



<그림> 등고선 추출을 위한 이미지

이번 예제는 아래 그림에서 보는 것과 같이 등고선 추출을 위해 촬영한 이미지를 이용해 3D Surface를 표시하기 위한 예제이다.

좌측의 그림은 등고선 추출을 위해 촬영된 이미지이다. 흰색이 높을수록 높이가 높다. 즉 높이가 높다는 것은 고도가 높아짐을 의미한다. 이와 같은 사진을 Q3DSurface를 이용해 3D 그래프로 시각화 해보도록 하자. 다음 그림은 예제 실행 화면이다.



<그림> 예제 실행 화면

위의 그림에서 보는 것과 같이 원본이미지를 QImage 클래스 오브젝트에 이미지를 로딩 한다. 그리고 QImage 클래스 오브젝트를 QHeightMapSurfaceDataProxy 클래스 오브젝트에 넘겨준다.

이 클래스는 이미지를 Surface로 시각화하기 위해서 이미지의 색상을 X, Y, Z 축으로 값을 변환하기 위한 기능을 제공한다. 즉 이 클래스는 QImage 클래스 데이터를 시각화 데이터로 변환하기 위해 RGB 데이터를 추출하여 X, Y, Z의 값으로 변환하는 기능을 제공한다. 그리고 QHeightMapSurfaceDataProxy 클래스 오브젝트에 저장된 데이터를 QSurface3DSeries에 인자로 전달하면 Q3DSurface가 사용 가능한 데이터로 변환 할 수 있다.

QSurface3DSeries 클래스의 오브젝트를 Q3DSurface 클래스의 addSeries() 멤버 함수의 첫 번째 인자로 전달하면 위의 그림에서 보는 것과 같이 3D 표시 할 수 있다. 다음은 예제 소스코드이다. 전체 소스코드와 이미지는 Ch12 > 04_ImageHeightMap 디렉토리를 참조하면 된다.

```
#include <QApplication>
#include <QtDataVisualization>
using namespace QtDataVisualization;
```

```

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    Q3DSurface *surface = new Q3DSurface;
    surface->setFlags(surface->flags() ^ Qt::FramelessWindowHint);

    surface->scene()->activeCamera()->setCameraPreset(
        Q3DCamera::CameraPresetFront);

    QImage heightMapImage(":/mountain.png");
    QHeightMapSurfaceDataProxy *heightMapProxy;
    heightMapProxy = new QHeightMapSurfaceDataProxy(heightMapImage);

    QSurface3DSeries *series = new QSurface3DSeries(heightMapProxy);
    series->setItemLabelFormat(QStringLiteral("(@xLabel, @zLabel): @yLabel"));

    heightMapProxy->setValueRanges(34.0f, 40.0f, 18.0f, 24.0f);

    surface->axisX()->setLabelFormat("0.1f N");
    surface->axisZ()->setLabelFormat("0.1f E");
    surface->axisX()->setRange(34.0f, 40.0f);
    surface->axisY()->setAutoAdjustRange(true);
    surface->axisZ()->setRange(18.0f, 24.0f);

    surface->axisX()->setTitle(QStringLiteral("Latitude"));
    surface->axisY()->setTitle(QStringLiteral("Height"));
    surface->axisZ()->setTitle(QStringLiteral("Longitude"));

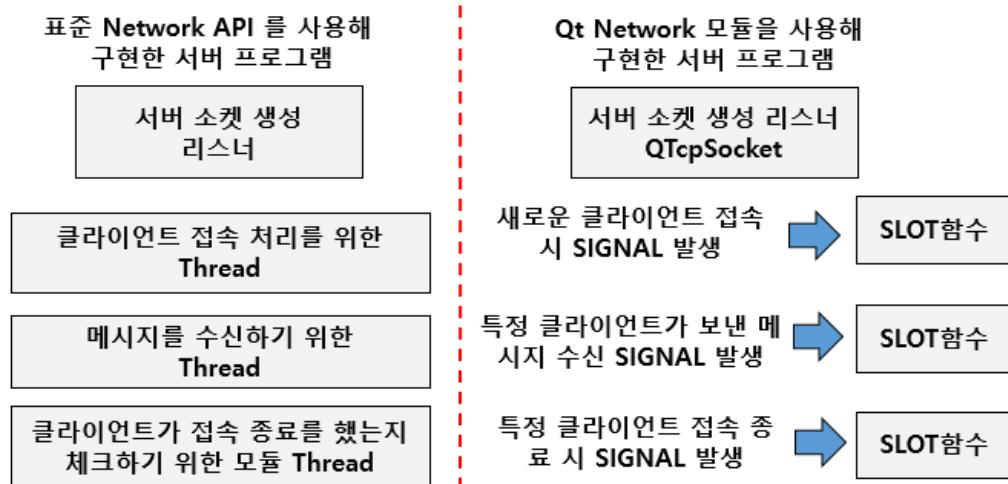
    surface->addSeries(series);
    surface->setGeometry(50,50,600,400);
    surface->show();

    return a.exec();
}

```

13. 네트워크 프로그래밍

Qt는 표준 네트워크 라이브러리를 사용하는 것보다 쉽게 빠르게 네트워크 기반의 클라이언트 또는 서버 소프트웨어를 개발할 수 있다. Qt에서는 네트워크 프로그래밍 구현 시 Signal과 Slot 개념을 사용하기 때문에 개발 쉽게 네트워크 응용 어플리케이션을 구현할 수 있다. 다음 그림은 표준 Network API를 이용해 개발하는 방식과 Qt에서 제공하는 Network 모듈을 이용해 개발하는 방법을 비교한 그림이다.



<그림> 표준 Network API 방식과 Qt Network 모듈 방식

위의 그림에서 외쪽은 표준 Network API를 이용해 구현하는 방식이다. 이 방식은 서버 프로그램의 소켓 리스너(Listener)를 생성하고 새로운 클라이언트가 접속하는지 계속 검사하는 Thread가 필요하다. 그리고 특정 클라이언트가 보내는 메시지가 있는지 검사하고 메시지가 있다면 접속한 클라이언트에게 전달하는 Thread가 필요하다. 그리고 접속한 클라이언트가 종료했는지 검사하는 Thread가 필요하다.

예로 3개의 Thread를 예로 들었지만 실제 서버 프로그램에서는 이것보다 더 많은 Thread 구현이 필요하다. Thread 가 많이 사용 될수록 프로그램 소스코드의 복잡성도 높아지며 유지보수도 어려워 진다.

하지만 Qt에서 제공하는 Network 모듈을 사용해 서버 프로그램을 개발한다고 가정해 보자. 새로운 접속자를 처리하기 위해 표준 네트워크 API를 사용하는 방식은 새로운 클라이언트를 처리하는 Thread 가 필요하지만 Qt에서는 새로운 클라이언트를 처리하기 위한 Connect() 함수로 Signal과 Slot을 연결하면 된다. 예를 들어 새로운 클라이언트

접속 시그널 발생 시 Slot 함수로 연결하면 된다. 다음 예제 소스코드 Qt에서 제공하는 Network 모듈을 이용해 Signal과 Slot으로 구현한 예제이다.

```
QHttpSocket::QHttpSocket(QObject *parent) : QObject(parent)
{
    QTcpSocket *socket = new QTcpSocket(this);
    connect(socket, SIGNAL(connected()), this, SLOT(slotConnected()));
    ...
}

void QHttpSocket::slotConnected()
{
    qDebug("[%s] CONNECTED", Q_FUNC_INFO);
    socket->write("HEAD / HTTP/1.0\r\n\r\n\r\n\r\n\r\n");
}
...
```

위의 예제 소스코드에서 보는 것과 QTcpSocket 클래스의 오브젝트를 선언하고 connect() 함수를 이용해 새로운 접속자 이벤트 connected() 시그널을 제공한다. 이 Signal을 slotConnected() 함수와 연결한다.

따라서 Signal과 Slot을 연결하면 새로운 접속자 시그널이 발생하면 slotConnected() 함수가 호출된다. 즉 Qt에서는 새로운 접속자를 처리하기 위한 Thread 구현 없이 Slot 함수만 구현하면 되기 때문에 매우 간단하게 네트워크 프로그램을 쉽게 구현할 수 있다.

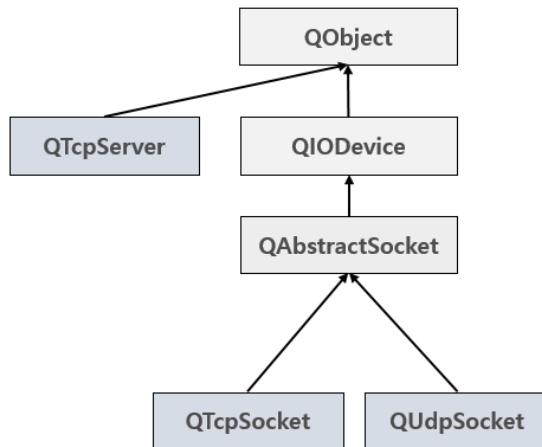
Qt 네트워크 모듈은 TCP/UDP 프로토콜 기반 서버 또는 클라이언트 구현을 위한 다양한 API, 네트워크 상태를 나타내는 API 그리고 SSL(Secure Sockets Layer)를 지원하는 API를 제공한다.

TCP/UDP 프로토콜 기반 응용 서버 및 클라이언트 어플리케이션을 개발하기 위한 주요 클래스로 QTcpSocket, QTcpServer 그리고 QUdpSocket 클래스를 제공한다.

✓ LOW 레벨 네트워크 클래스들

- QTcpSocket – TCP 프로토콜 기반의 네트워크 클래스
- QTcpServer – TCP 프로토콜 기반의 서버 구현에 적합한 클래스
- QUdpSocket – UCP 프로토콜 기반의 네트워크 클래스

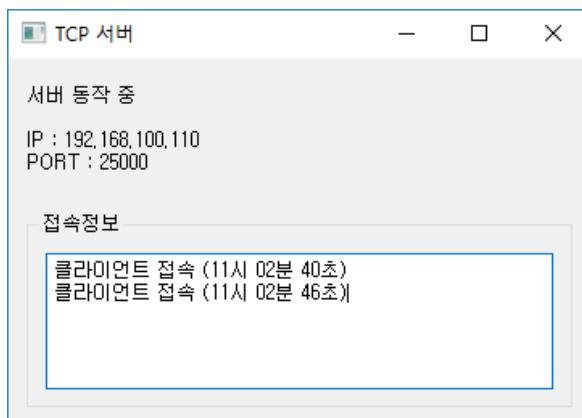
LOW 레벨 클래스들은 개발자가 세부적인 네트워크 기능을 구현에 적합하다.



`QTcpSocket` 클래스와 `QUdpSocket` 클래스는 `QAbstractSocket` 클래스로부터 상속받아 구현 되었다. TCP/UDP 프로토콜의 기반이 아닌 새로운 네트워크 프로토콜을 구현하기 위해서 `QAbstractSocket` 클래스를 상속 받아 구현한다면 많은 시간을 절약할 수 있다. `QTcpServer`는 `QTcpSocket` 과 동일한 TCP 프로토콜 기반의 클래스이다. 차이점은 `QTcpServer` 클래스는 서버 기반의 응용 어플리케이션 구현에 편리한 기능을 제공한다. Qt에서 제공하는 네트워크 모듈을 사용하기 위해서는 프로젝트 파일(.pro)에 다음과 같이 추가해야 한다.

```
QT += network
```

- `QTcpServer` 와 `QTcpSocket` 클래스를 이용한 서버/클라이언트 구현



<그림> 서버 예제 실행 화면

위의 그림에서 보는 것과 같이 GUI 상에 상단에 QLabel 위젯을 배치하고 하단의 그룹박스에는 클라이언트가 접속한 시간을 출력하는 텍스트로 출력할 수 있는 QTextEdit를 배치하고 Widget 클래스의 헤더 소스코드를 다음과 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QtNetwork>

class QTcpServer;
class QNetworkSession;

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;
    void initialize();
    QTcpServer *tcpServer;

private slots:
    void newConnection();
};

#endif // WIDGET_H
```

위의 소스코드에서 initialize() 함수는 QTcpServer 클래스의 오브젝트를 초기화를 위한 함수이며 newConnection() Slot 함수는 클라이언트가 접속 Signal 이 발생하면 호출되는 함수이다. 다음은 widget.cpp 소스코드이다.

```
#include "widget.h"
#include "ui_widget.h"
#include <QtWidgets>
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
```

```

{
    ui->setupUi(this);
    initialize();
}

void Widget::initialize()
{
    QHostAddress hostAddress;
    QList<QHostAddress> ipAddressesList = QNetworkInterface::allAddresses();

    // localhost(127.0.0.1) 가 아닌 것을 사용
    for (int i = 0; i < ipAddressesList.size(); ++i)
    {
        if(ipAddressesList.at(i) != QHostAddress::LocalHost &&
           ipAddressesList.at(i).toIPv4Address())
        {
            hostAddress = ipAddressesList.at(i);
            break;
        }
    }

    if (hostAddress.toString().isEmpty())
        hostAddress = QHostAddress(QHostAddress::LocalHost);

    tcpServer = new QTcpServer(this);
    if (!tcpServer->listen(hostAddress, 25000))
    {
        QMessageBox::critical(this, tr("TCP Server"),
                             tr("서버를 시작할 수 없습니다. 에러메세지 : %1.")
                             .arg(tcpServer->errorString()));
        close();
    }

    return;
}

ui->labelStatus->setText(tr("서버 동작 중 \n\n"
                            "IP : %1\n"
                            "PORT : %2\n")
                           .arg(hostAddress.toString())
                           .arg(tcpServer->serverPort()));

connect(tcpServer, SIGNAL(newConnection()), this, SLOT(newConnection()));

```

```

        ui->connMsgEdit->clear();
    }

void Widget::newConnection()
{
    QTcpSocket *clientConnection = tcpServer->nextPendingConnection();

    connect(clientConnection, SIGNAL(disconnected()),
            clientConnection, SLOT(deleteLater()));

    QString currTime = QTime::currentTime().toString("hh시 mm분 ss초");
    QString text = QString("클라이언트 접속 (%1)").arg(currTime);

    ui->connMsgEdit->append(text);
    QByteArray message = QByteArray("Hello Client ~ ");
    clientConnection->write(message);
    clientConnection->disconnectFromHost();
}

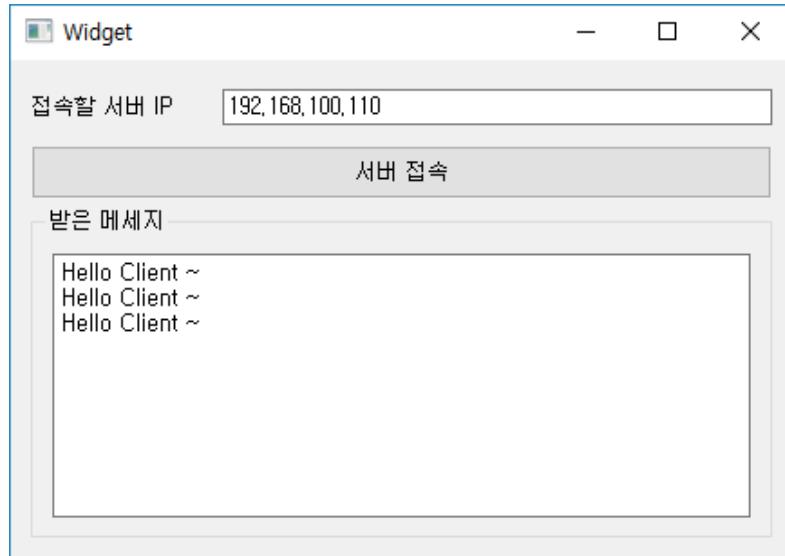
Widget::~Widget()
{
    delete ui;
}

```

생성자에서 호출되는 `initialize()` 함수에서는 클라이언트가 접속할 서버의 IP를 시스템으로부터 얻어온다. 그리고 `QTcpServer` 클래스의 `tcpServer` 오브젝트 초기화하고 `listen()` 멤버 함수를 이용해 클라이언트 접속 요청 대기 상태가 될 수 있도록 한다. `listen()` 함수의 첫 번째 인자는 IP주소이며 두 번째 인자는 서버 Port 번호이다.

그리고 `initialize()` 함수 하단의 `connect()` 함수는 클라이언트가 접속하게 되면 호출되는 Signal 과 Slot 함수를 연결하였다. 따라서 새로운 클라이언트가 접속하게 되면 `newConnection()` 함수가 호출된다.

`newConnection()` 함수는 GUI 상에 `QTextEdit` 위젯에 클라이언트가 접속한 시간을 출력하고 접속한 클라이언트에게 “Hello Client ~ ” 메시지를 전송한다. 다음은 클라이언트 예제이다. `QWidget` 을 상속받는 프로젝트를 생성하고 다음 그림에서 보는 것과 같이 GUI상에 위젯을 배치한다. 클라이언트의 전체 예제 소스코드는 Ch13 > 01_TcpClient 딕토리를 참조하면 된다.



<그림> 클라이언트 예제 실행 화면

위의 그림에서 보는 것과 같이 접속할 서버의 IP와 PORT를 입력할 수 있도록 QLineEdit 위젯을 배치한다. 그리고 [접속] 버튼과 하단의 QTextEdit를 배치한다.

[접속] 버튼을 클릭하면 서버와 연결을 시도한다. 서버와 연결이 완료되면 서버로부터 받은 메시지를 QTextEdit 위젯에 출력한다. 아래 예제 소스코드와 같이 widget.h 헤더 소스코드를 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QTcpSocket>

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
```

```

Ui::Widget *ui;
void initialize();
QTcpSocket *tcpSocket;

private slots:
void connectButton();
void readMessage(); // 서버로부터 메세지를 받을 때 호출됨
void disconnected();
};

#endif // WIDGET_H

```

connectButton() Slot 함수는 [접속] 버튼 클릭 시 호출된다. readMessage() Slot 함수는 서버로부터 메시지를 받는 Signal 이 발생하면 호출되는 함수이다.

그리고 disconnected() Slot 함수는 서버로부터 접속이 종료된 Signal 이 발생하면 호출되는 함수이다. 다음은 widget.cpp 예제소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"

#include <QHostAddress>
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->connectButton, SIGNAL(clicked()), this, SLOT(connectButton()));

    initialize();
}

void Widget::initialize()
{
    tcpSocket = new QTcpSocket(this);
    connect(tcpSocket, SIGNAL(readyRead()), this, SLOT(readMessage()));
    connect(tcpSocket, SIGNAL(disconnected()), this, SLOT(disconnected()));
}

void Widget::connectButton()
{
    QString serverip = ui->serverIP->text().trimmed();
    QHostAddress serverAddress(serverip);
}

```

```

        tcpSocket->connectToHost(serverAddress, 25000);
    }

void Widget::readMessage()
{
    if(tcpSocket->bytesAvailable() >= 0)      {
        QByteArray readData = tcpSocket->readAll();
        ui->textEdit->append(readData);
    }
}

void Widget::disconnected()
{
    qDebug() << Q_FUNC_INFO << "서버 접속 종료.";
}

Widget::~Widget()
{
    delete ui;
}

```

initialize() 함수는 QTcpSocket 클래스의 tcpSocket 오브젝트를 선언하고 서버로부터 메시지를 받을 발생한 Signal 을 readMessage() Slot 함수와 연결한다. 그리고 서버와 연결 종료 Signal을 받으면 disconnected() Slot 함수와 연결한다. 따라서 서버로부터 메시지를 받으면 readMessage() 함수를 호출하고 서버와 연결이 종료되면 disconnect() Slot 함수가 호출된다.

readMessage() Slot 함수에서 tcpSocket->bytesAvailable() 함수는 서버가 보내온 메시지의 Bytes 수를 구할 수 있다. 그리고 readAll() 멤버 함수는 서버가 보내온 메시지를 읽어올 수 있는 기능을 제공한다.

- QTcpSocket 클래스 동기 방식과 비동기 방식의 구현과 차이점 비교

이전 예제에서는 서버와 클라이언트를 구현해 보았다. 이전 예제에서 서버 구현 시 클라이언트가 언제 접속할 수 없기 때문에 클라이언트를 접속 시 처리하기 위해 Signal / Slot 을 이용하였다. 하지만 네트워크 어플리케이션을 구현하다 보면 동기 방식으로 구현해야 할 경우가 있다.

여기서 말하는 동기 방식은 특정 순서대로 진행되어야 하는 경우를 말한다. 예를 들어 클라이언트가 메시지를 서버에게 보내면 어떤 메시지를 받기 전까지 다른 처리를 하지

않고 기다려야 하는 경우가 있다. 즉 서버로부터 요청하면 응답을 받기까지 기다려야 하는 처리과정을 구현해야 한다면 이전에 다룬 비동기 방식이 아닌 동기 방식으로 구현해야 한다.

이번 예제에서는 qt-dev.com 웹 서버에 접속 시 동기 방식과 비 동기 방식을 구현해 보고 차이점을 비교해 보도록 하자.

다음 예제는 GUI를 포함하지 않는다. 따라서 콘솔 기반의 프로젝트를 생성하고 다음과 같이 QObject 클래스를 상속받는 QHttpSocket라는 이름의 클래스를 생성하자. 다음 예제 소스코드는 QHttpSocket의 헤더파일이다. 전체 예제 소스코드는 Ch13 > 02_QTcpSocket_Sync 디렉토리를 참조하면 된다.

```
#ifndef QHTTPSOCKET_H
#define QHTTPSOCKET_H
#include <QObject>
#include <QTcpSocket>

class QHttpSocket : public QObject
{
    Q_OBJECT
public:
    explicit QHttpSocket(QObject *parent = 0);
    ~QHttpSocket();

    void httpConnect();

public slots:
    void httpDisconnected();

private:
    QTcpSocket *socket;
};

#endif // QHTTPSOCKET_H
```

httpConnect() 함수에서는 qt-dev.com 웹 서버에 접속하고 메시지를 송/수신하는 기능을 제공한다. 다음 예제는 QHttpSocket 클래스의 소스코드이다.

```
#include "qhttpsocket.h"
#include <QDebug>
```

```

QHttpSocket::QHttpSocket(QObject *parent) : QObject(parent)
{
    socket = new QTcpSocket(this);
    connect(socket, SIGNAL(disconnected()), this, SLOT(httpDisconnected()));
}

QHttpSocket::~QHttpSocket()
{
}

void QHttpSocket::httpConnect()
{
    socket->connectToHost("qt-dev.com", 80);
    if(socket->waitForConnected(5000))
    {
        int writeBytes = socket->write("Hello server\r\n\r\n");
        socket->waitForBytesWritten(1000);
        qDebug() << "write bytes : " << writeBytes;

        socket->waitForReadyRead(3000);
        qDebug() << "Reading: " << socket->bytesAvailable();
        qDebug() << socket->readAll();
    }
    else
    {
        qDebug() << "Not connected!";
    }
}

void QHttpSocket::httpDisconnected()
{
    qDebug() << Q_FUNC_INFO;
}

```

httpConnect() 함수에서 qt-dev.com 웹 서버에 연결이 완료 될 때 까지 5초간 기다린다. 5초간 연결이 될 때 까지 기다리기 위해서 waitForConnected() 함수를 사용한다. waitForConnected() 함수 인자의 단위는 Millisecond 이다.

연결이 완료 되면 qt-dev.com 웹 서버에 연결하고 write() 함수를 이용해 qt-dev.com 웹 서버로 메시지를 전송한다. 그리고 전송이 완료 될 때까지 기다린다. write() 함수로 웹 서버 전송한 메시지가 전송될 때 까지 기다리기 위해서 waitForBytesWritten() 함수를 사용하면 된다.

그리고 write() 함수의 리턴 값은 전송된 Bytes 수를 리턴 한다. 만약 에러가 발생하게 되면 0을 리턴 한다.

waitForReadyRead() 함수는 서버로부터 메시지를 수신할 때 까지 기다리며 이 함수의 인자는 기다리는 시간을 지정할 수 있다. 따라서 3초간 서버로부터 수신 받는 메시지를 3초간 기다린다. 다음 예제 소스코드는 QHttpSocket 클래스 오브젝트를 생성하고 수행하는 main.cpp 소스코드 예제이다.

```
#include <QCoreApplication>
#include "qhttpsocket.h"

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    QHttpSocket *httpSocket = new QHttpSocket();
    httpSocket->httpConnect();

    return a.exec();
}
```

지금까지 동기 방식의 예제를 다루어 보았다. 이 예제와 동일한 방식으로 비 동기 방식으로 구현해 보도록 하자. 동기 방식과 같이 QHttpSocket 클래스를 아래와 같이 구현해 보도록 하자. 다음 예제 소스코드는 QHttpSocket 클래스의 헤더파일이다. 이 예제의 전체 소스코드는 Ch13 > 02_QTcpSocket_Async 디렉토리를 참조하면 된다.

```
#ifndef QHTTPSOCKET_H
#define QHTTPSOCKET_H
#include <QObject>
#include <QTcpSocket>

class QHttpSocket : public QObject
{
    Q_OBJECT

public:
    explicit QHttpSocket(QObject *parent = 0);
    ~QHttpSocket();
    void httpConnect();

public slots:
    void slotConnected();
```

```

void slotDisconnected();
void slotBytesWritten(qint64 bytes);
void slotReadPandingDatagram();

private:
    QTcpSocket *socket;
};

#endif // QHTTPSOCKET_H

```

위의 예제는 비 동기 방식인 QHttpSocket 클래스의 헤더 소스이다. slotConnected() 함수는 웹 서버와 연결이 완료되면 호출되는 Slot 함수이다. slotDisconnected() 함수는 웹 서버와 연결이 종료되면 종료되는 Slot 함수이다.

slotBytesWritten() 함수는 서버에 전송을 완료하면 호출되는 Slot 함수이며 slotReadPandingDatagram() 함수는 서버로부터 메시지를 수신 받으면 호출 되는 Slot 함수이다. 다음 예제 소스코드는 QHttpSocket 클래스의 구현 소스코드이다.

```

#include "qhttpsocket.h"
#include <QDebug>

QHttpSocket::QHttpSocket(QObject *parent) : QObject(parent)
{
    socket = new QTcpSocket(this);

    connect(socket, SIGNAL.connected(), this, SLOT(slotConnected()));
    connect(socket, SIGNAL(disconnected()), this, SLOT(slotDisconnected()));
    connect(socket, SIGNAL(bytesWritten(qint64)),
            this, SLOT(slotBytesWritten(qint64)));
    connect(socket, SIGNAL(readyRead()), this, SLOT(slotReadPandingDatagram()));
}

QHttpSocket::~QHttpSocket()
{
}

void QHttpSocket::httpConnect()
{
    socket->connectToHost("qt-dev.com", 80);
    if(!socket->waitForConnected(3000)) {
        qDebug() << "Socket Error : " << socket->errorString();
    }
}

```

```

}

void QHttpSocket::slotConnected()
{
    qDebug("\n [%s] CONNECTED", Q_FUNC_INFO);
    socket->write("HEAD / HTTP/1.0\r\n\r\n\r\n\r\n\r\n");
}

void QHttpSocket::slotDisconnected()
{
    qDebug("\n [%s] DISCONNECTED", Q_FUNC_INFO);
}

void QHttpSocket::slotBytesWritten(qint64 bytes)
{
    qDebug("\n [%s] Bytes Written [size :%d]", Q_FUNC_INFO, bytes);
}

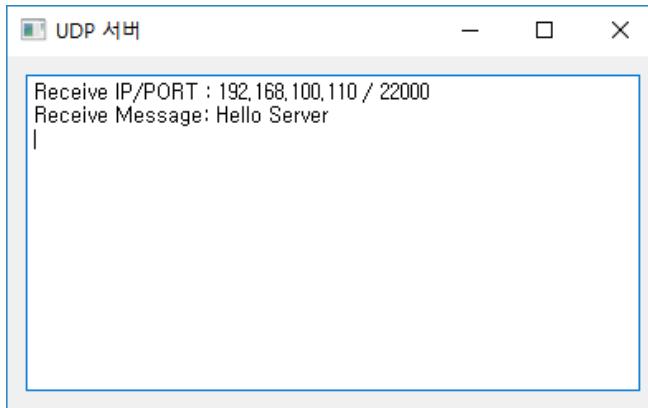
void QHttpSocket::slotReadPendingDatagram()
{
    qDebug() << socket->readAll();
}

```

생성자에서는 QTcpSocket 클래스 오브젝트를 선언하고 서버와 연결, 종료, 메시지 전송 완료 시 그리고 웹 서버로부터 메시지 수신 시 Signal 을 Slot 함수와 연결한다. httpConnect() 함수를 실행하면 웹 서버와 연결을 시도한다. 이와 같이 이전 동기 방식과 다르게 이번 예제에서는 Signal 과 Slot을 이용해 비 동기 방식을 사용할 수 있다.

- QUdpSocket 클래스를 이용한 서버 / 클라이언트 구현

UDP 프로토콜은 비 신뢰성 이므로 TCP 와 같이 메시지를 송/수신 하기 전에 서버와 클라이언트가 연결을 맺어야 하지만 UDP 는 Connection을 맺지 않고 양자간 메시지를 송/수신 할 수 있다. 즉 UDP는 보내고자 하는 IP와 PORT 주소로 바로 메시지를 송신하거나 수신할 수 있다.



<그림> UDP 서버 실행 화면

이 위젯은 클라이언트로부터 메시지를 받으면 메시지를 보낸 클라이언트의 IP, PORT 번호 그리고 메시지를 출력하는 예제이다. 프로젝트 생성 시 QWidget 클래스 기반으로 생성한다. `widget.h` 헤더 파일을 다음과 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QUdpSocket>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;
    QUdpSocket *udpSocket;

private slots:
    void readPendingDatagram();
};

#endif // WIDGET_H
```

`readPendingDatagram()` Slot 함수는 클라이언트로부터 메시지를 받았을 때 호출되는

이번 예제는 비 신뢰성 UDP 프로토콜 기반 메시지를 송/수신 하는 예제를 살펴보도록 하자. 전체 소스코드는 Ch13 > 03_UDP_Server 디렉토리를 참조한다.

위의 그림에서 보는 것과 같이 QTextEdit 위젯을 배치한다.

Slot 함수이다. 다음은 widget.cpp 소스코드이다.

```
#include "widget.h"
#include "ui_widget.h"
#include <QDebug>
#define SERVER_PORT 21000

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);

    udpSocket = new QDhcpSocket(this);
    udpSocket->bind(QHostAddress("192.168.100.110"), SERVER_PORT);
    connect(udpSocket, SIGNAL(readyRead()), this, SLOT(readPendingDatagram()));
}

void Widget::readPendingDatagram()
{
    QByteArray buffer;
    buffer.resize(udpSocket->pendingDatagramSize());

    QHostAddress sender;
    quint16 senderPort;
    udpSocket->readDatagram(buffer.data(), buffer.size(),
                            &sender, &senderPort);

    QString msg = QString("Receive IP/PORT : %1 / %2 <br>"
                          "Receive Message: %3 <br>")
                  .arg(sender.toString())
                  .arg(senderPort)
                  .arg(buffer.data());

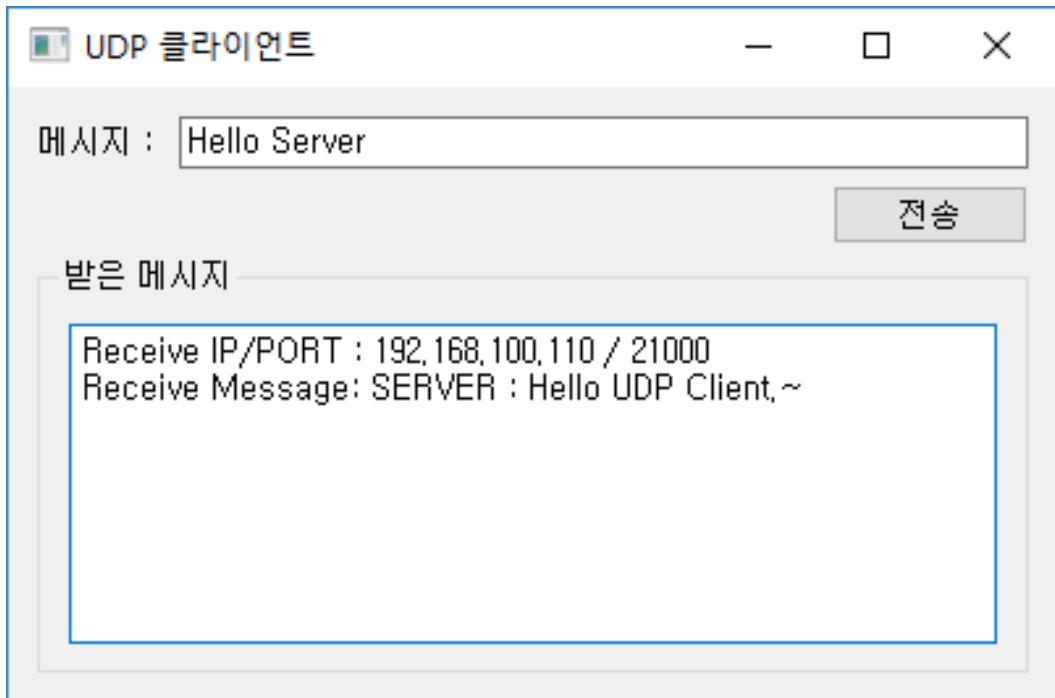
    ui->textEdit->append(msg);

    QByteArray writeData;
    writeData.append("SERVER : Hello UDP Client.~ ");
    udpSocket->writeDatagram(writeData, sender, senderPort);
}

Widget::~Widget()
{
    delete ui;
}
```

`readPendingDatagram()` Slot 함수에서 `pendingDatagramSize()` 멤버 함수는 수신 메시지의 크기를 리턴 한다. 그리고 수신 받은 메시지를 QTextEdit 위젯에 출력하고 수신 받은 클라이언트에게 다시 메시지를 전송한다.

다음 예제는 클라이언트 구현 예제이다. 클라이언트의 전체 소스 코드는 Ch13 > 03_UDP_Client 디렉토리를 참조하면 된다.



<그림> UDP 클라이언트 예제 실행 화면

위의 그림에서와 같이 위젯을 배치한다. 그리고 아래와 같이 widget.h 헤더 파일의 소스코드를 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QUdpSocket>

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
```

```

Q_OBJECT

public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;
    QUdpSocket *udpSocket;

public slots:
    void sendButton();
    void readPendingDatagram();
};

#endif // WIDGET_H

```

위의 예제에서 보는 것과 같이 sendButton() Slot 함수는 [전송] 버튼을 클릭하면 호출된다. 그리고 readPendingDatagram() Slot 함수는 메시지를 수신 받으면 호출된다. 다음 예제는 widget.cpp 소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"
#include <QDebug>

#define SERVER_PORT 21000
#define CLIENT_PORT 22000

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->sendButton, &QPushButton::pressed, this, &Widget::sendButton);

    udpSocket = new QUdpSocket(this);
    udpSocket->bind(QHostAddress("192.168.100.110"), CLIENT_PORT);
    connect(udpSocket, SIGNAL(readyRead()), this, SLOT(readPendingDatagram()));
}

void Widget::sendButton()
{
    QByteArray msg;
    msg = ui->sendMsg->text().toLocal8Bit();
}

```

```

QHostAddress sender("192.168.100.110");
    udpSocket->writeDatagram(msg, sender, SERVER_PORT);
}

void Widget::readPendingDatagram()
{
    QByteArray buffer;
    buffer.resize(udpSocket->pendingDatagramSize());

    QHostAddress sender;
    quint16 senderPort;
    udpSocket->readDatagram(buffer.data(), buffer.size(), &sender, &senderPort);

    QString msg = QString("Receive IP/PORT : %1 / %2 <br>"
                           "Receive Message: %3 <br>")
                           .arg(sender.toString())
                           .arg(senderPort)
                           .arg(buffer.data());
    ui->textEdit->append(msg);
}

Widget::~Widget()
{
    delete ui;
}

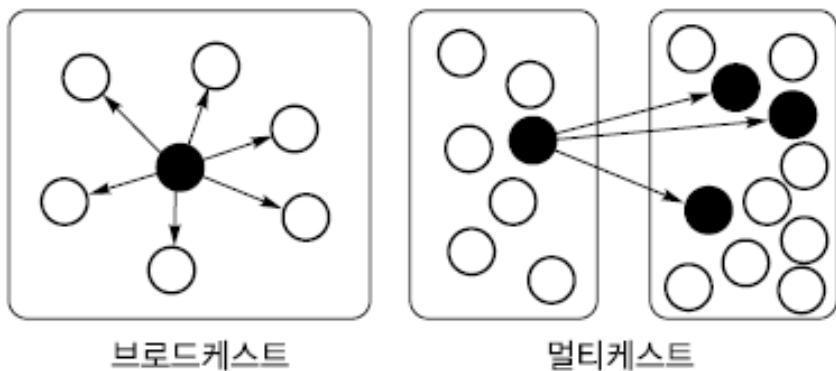
```

readPendingDatagram() Slot 함수는 서버로부터 메시지를 수신 받으면 QTextEdit 위젯에 수신 받은 메시지를 출력한다.

- QUdpSocket 클래스를 이용한 브로드캐스트 구현

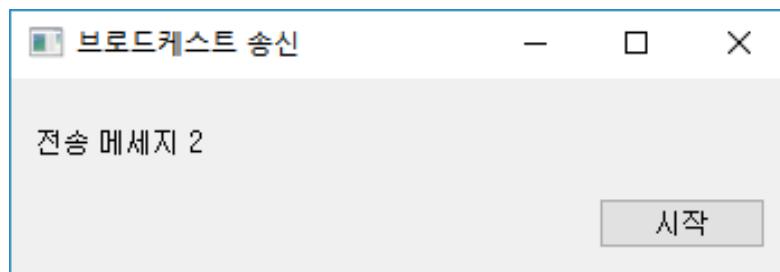
컴퓨터 네트워크 상에서 UDP 기반 프로토콜을 이용해 송신자와 수신자 관점에서 나누어보면 브로드캐스트, 멀티캐스트로 구분할 수 있다.

브로드캐스트 방식은 하나의 송신자가 모든 수신자에게 메시지를 전달할 수 있는 방식이며 멀티캐스트 방식은 하나이상의 사용자가 하나이상의 송신자에게 데이터를 전송하는 방식이다.



<그림> 브로드캐스트와 멀티캐스트 방식

QWidget 클래스를 상속받는 프로젝트를 생성하고 다음 그림에서 보는 것과 같이 GUI를 배치한다. 전체 소스코드는 Ch13 > 04_Broadcast_Sender 디렉토리를 참조하면 된다.



<그림> 브로드캐스트 송신 예제 실행 화면

다음 예제 소스코드는 widget.h 소스코드이다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QUdpSocket>
#include <QTimer>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
```

```

Ui::Widget *ui;
Q_udpSocket *udpSocket;
QTimer *timer;
int msgNumber;

private slots:
    void startButton();
    void broadcastSend();
};

#endif // WIDGET_H

```

QTimer 클래스의 timer 오브젝트는 지정한 시간을 반복해 브로드캐스트 메시지를 전송 한다. 다음은 widget.cpp 소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    timer = new QTimer(this);
    udpSocket = new Q_udpSocket(this);
    msgNumber = 1;

    connect( ui->startButton, SIGNAL(clicked()), this, SLOT(startButton()));
    connect( timer, SIGNAL(timeout()), this, SLOT(broadcastSend()));
}

void Widget::startButton()
{
    if(!timer->isActive())
        timer->start(1000);
}

void Widget::broadcastSend()
{
    ui->sendMsg->setText(QString("전송 메세지 %1").arg(msgNumber));
    QByteArray datagram = "브로드캐스트 번호 " + QByteArray::number(msgNumber);

    udpSocket->writeDatagram(datagram.data(), datagram.size(),
                               QHostAddress::Broadcast, 35000);
    msgNumber++;
}

```

```

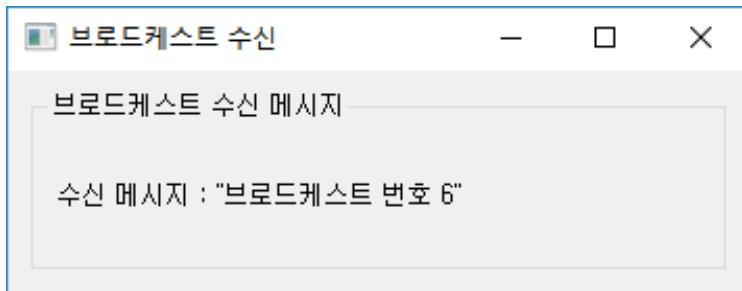
}

Widget::~Widget()
{
    delete ui;
}

```

[시작] 버튼을 클릭하면 QTimer 클래스의 timer 오브젝트가 1초에 한번씩 주기적으로 실행된다. 그리고 1초마다 broadcastSend() 함수를 실행한다. broadcastSend() 함수에서 writeDatagram() 함수의 3번째 인자는 브로드캐스트 방식으로 데이터를 전송 시 사용할 수 있다.

QHostAddress::Broadcast 값을 3번째 인자로 사용하면 네트워크 그룹 (255.255.255.255) 내에 모든 컴퓨터에게 메시지를 전송한다. 다음 예제는 브로드캐스트 메시지를 수신 받는 예제이다.



<그림> 브로드캐스트 수신 예제 실행 화면

위의 그림에서 보는 것과 같이 브로드캐스트 메시지를 수신 받으면 위젯 상에 메시지를 출력한다. 다음은 widget.cpp 소스이다. 전체 소스는 Ch13 > 04_Broadcast_Receiver 디렉토리를 참조하면 된다.

```

#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);

    udpSocket = new QUdpSocket(this);
    udpSocket->bind(35000, QUdpSocket::ShareAddress);
    connect(udpSocket, SIGNAL(readyRead()), this, SLOT(readDatagrams()));
}

```

```

void Widget::readDatagrams()
{
    while (udpSocket->hasPendingDatagrams()){
        QByteArray datagram;
        datagram.resize(udpSocket->pendingDatagramSize());
        udpSocket->readDatagram(datagram.data(), datagram.size());
        ui->recvMsg->setText(tr("수신 메시지 : \"%1\"").arg(datagram.data()));
    }
}

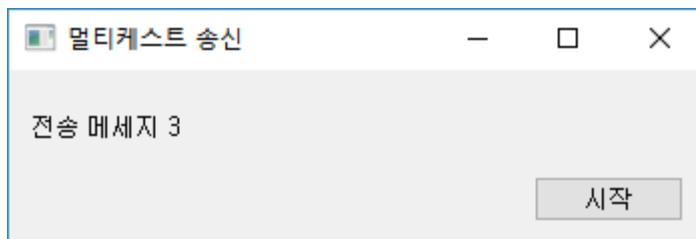
Widget::~Widget() {
    delete ui;
}

```

- QUdpSocket 클래스를 이용한 멀티캐스트 구현

이번 예제에서는 멀티캐스트 방식을 이용해 어플리케이션을 구현 하는 방법에 대해서 알아보도록 하자. 멀티캐스트 방식은 하나이상의 사용자가 하나이상의 송신자에게 데이터를 전송하는 방식이다.

QWidget 클래스를 상속받는 프로젝트를 생성하고 다음 그림에서 보는 것과 같이 GUI 상에 위젯을 배치한다. 전체 소스코드는 Ch13 > 05_Multicast_Sender 디렉토리를 참조하면 된다.



<그림> 멀티캐스트 송신 예제 실행 화면

다음 예제 소스코드는 widget.h 소스코드이다.

```

#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QUdpSocket>
#include <QTimer>

namespace Ui { class Widget; }

```

```

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();
private:
    Ui::Widget *ui;
    QUdpSocket udpSocket4;
    QUdpSocket udpSocket6;
    QHostAddress groupAddress4;
    QHostAddress groupAddress6;

    QTimer      *timer;
    int         msgNumber;

private slots:
    void startButton();
    void broadcastSend();
};

#endif // WIDGET_H

```

QTimer 클래스의 timer 오브젝트는 지정한 시간을 반복해 브로드캐스트 메시지를 전송 한다. 다음은 widget.cpp 소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    groupAddress4 = QHostAddress(QStringLiteral("239.255.43.21"));
    groupAddress6 = QHostAddress(QStringLiteral("ff12::2115"));

    timer = new QTimer(this);
    udpSocket4.bind(QHostAddress(QHostAddress::AnyIPv4), 0);
    udpSocket6.bind(QHostAddress(QHostAddress::AnyIPv6),
                    udpSocket4.localPort());
    msgNumber = 1;

    connect(ui->startButton, SIGNAL(clicked()), this, SLOT(startButton()));
    connect(timer, SIGNAL(timeout()), this, SLOT(multicastSend()));
}

```

```

}

void Widget::startButton()
{
    if(!timer->isActive())
        timer->start(1000);
}

void Widget::multicastSend()
{
    ui->sendMsg->setText(QString("전송 메세지 %1").arg(msgNumber));
    QByteArray datagram = "멀티캐스트 번호 " + QByteArray::number(msgNumber);
    udpSocket4.writeDatagram(datagram, groupAddress4, 45000);

    if (udpSocket6.state() == QAbstractSocket::BoundState)
        udpSocket6.writeDatagram(datagram, groupAddress6, 45000);

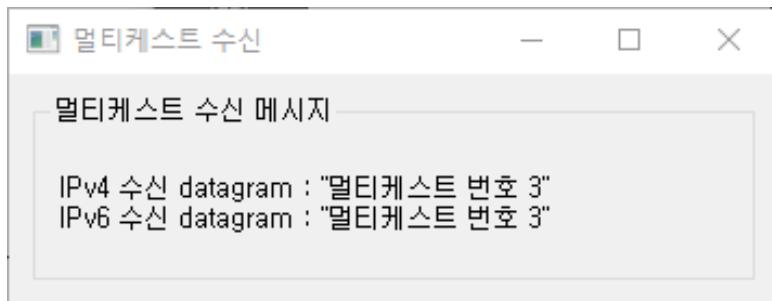
    msgNumber++;
}

Widget::~Widget()
{
    delete ui;
}

```

[시작] 버튼을 클릭하면 QTimer 클래스의 timer 오브젝트가 1초에 한번씩 주기적으로 실행된다. 그리고 1초마다 multicastSend() 함수를 실행한다.

multicastSend() 함수에서 writeDatagram() 함수의 첫 번째 인자는 보낼 메시지, 두 번째 메시지는 네트워크의 특정 그룹 그리고 세 번째 인자는 보낼 PORT 번호이다. 다음 예제는 멀티캐스트 메시지를 수신 받는 예제이다.



<그림> 브로드캐스트 수신 예제 실행 화면

위의 그림에서 보는 것과 같이 멀티캐스트 메시지를 수신 받으면 위젯 상에 메시지를

출력한다. 다음은 widget.cpp 소스코드이다. 전체 소스는 Ch13 > 05_Multicast_Receiver 디렉토리를 참조하면 된다.

```
#include "widget.h"
#include "ui_widget.h"
#include <QtWidgets>
#include <QtNetwork>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    groupAddress4 = QHostAddress(QStringLiteral("239.255.43.21"));
    groupAddress6 = QHostAddress(QStringLiteral("ff12::2115"));

    udpSocket4.bind(QHostAddress::AnyIPv4, 45000, QUdpSocket::ShareAddress);
    udpSocket4.joinMulticastGroup(groupAddress4);

    if (!udpSocket6.bind(QHostAddress::AnyIPv6, 45000,
        QUdpSocket::ShareAddress) ||
        !udpSocket6.joinMulticastGroup(groupAddress6))
    {
        qDebug() << Q_FUNC_INFO << "IPv4 멀티캐스트만 사용 가능.";
    }

    // 이전 Signal Slot 스타일
    connect(&udpSocket4, SIGNAL(readyRead()), this, SLOT(readDatagrams()));
    // 새로운 Signal Slot 스타일
    connect(&udpSocket6, &QUdpSocket::readyRead, this, &Widget::readDatagrams);
}

void Widget::readDatagrams()
{
    QByteArray datagram;
    while (udpSocket4.hasPendingDatagrams())
    {
        datagram.resize(int(udpSocket4.pendingDatagramSize()));
        udpSocket4.readDatagram(datagram.data(), datagram.size());
        ui->recvMsg->setText(tr("IPv4 수신 datagram : \"%1\"")
            .arg(datagram.constData()));
    }

    // using QUdpSocket::receiveDatagram (API since Qt 5.8)
}
```

```

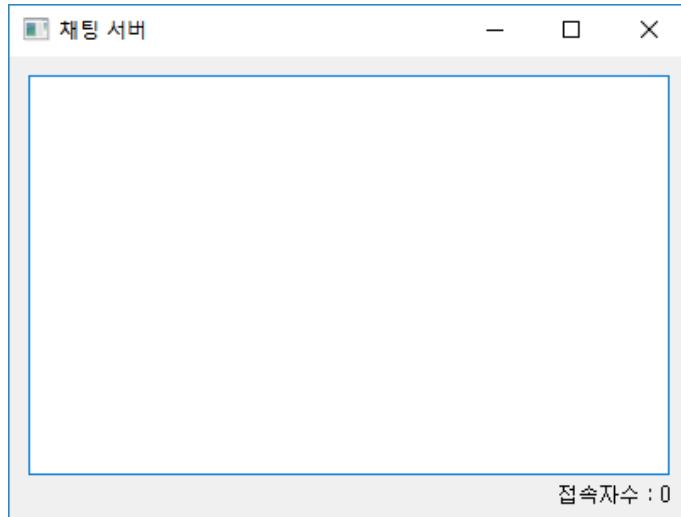
while (udpSocket6.hasPendingDatagrams())
{
    QNetworkDatagram dgram = udpSocket6.receiveDatagram();
    ui->recvMsg->setText(ui->recvMsg->text() +
                           tr("\nIPv6 수신 datagram : \"%1\"");
                           .arg(dgram.data().constData()));
}
}

Widget::~Widget()
{
    delete ui;
}

```

- 채팅 서버와 클라이언트 구현

이번 예제는 채팅 서버와 채팅 클라이언트를 구현해 보도록 하자. 먼저 채팅 서버 구현을 살펴보도록 하자. 아래와 같이 GUI 상에는 접속한 사용자 정보와 서버에 접속해 있는 접속자를 표시한다.



<그림> 채팅 예제 실행 화면

위의 그림에서 보는 것과 같이 GUI 상에 위젯을 배치한다. 중앙에 배치한 위젯은 QTextEdit 위젯이고 하단에 접속자 수를 표시하는 위젯은 QLabel 위젯이다. 프로젝트 생성 시 QWidget 클래스를 Base로 하는 Widget 클래스를 생성한다.

그리고 QTcpServer 클래스를 상속 받는 ChatServer 클래스를 생성한다. 전체 예제 소스

코드는 Ch13 > 06_ChatServer 디렉토리를 참조하면 된다. 다음은 ChatServer 클래스의 헤더파일이다.

```
#ifndef CHATSERVER_H
#define CHATSERVER_H
#include <QtNetwork/QTcpServer>
#include <QtNetwork/QTcpSocket>

class ChatServer : public QTcpServer
{
    Q_OBJECT
public:
    ChatServer(QObject *parent=0);
private slots:
    void readyRead();
    void disconnected();
    void sendUserList();
signals:
    void clients_signal(int users);
    void message_signal(QString msg);
protected:
    void incomingConnection(int socketfd);
private:
    QSet<QTcpSocket*> clients;
    QMap<QTcpSocket*,QString> users;
};

#endif // CHATSERVER_H
```

위의 헤더에서 protected 접근 제한자에서 선언한 incomingConnection() 함수는 새로운 클라이언트가 접속하게 되면 호출되는 함수이다. clients_signal() 시그널은 이 함수에서 발생한다. 이 시그널의 인자로 현재 서버에 접속한 사용자 수를 인자로 넘겨준다. 그리고 클라이언트에 대한 메시지를 수신하면 readyRead() 함수와 연결한다.

따라서 클라이언트가 메시지를 보내오면 readyRead() 함수가 호출된다. 그리고 disconnected() 시그널은 클라이언트 접속을 종료하면 시그널이 발생하고 이 시그널과 연결된 disconnected() 함수 호출된다. 다음 예제 소스코드는 ChatServer 클래스의 구현 소스코드이다.

```
#include <QtWidgets>
#include <QRegExp>
#include "chatserver.h"
#include <QDebug>
```

```

ChatServer::ChatServer(QObject *parent) : QTcpServer(parent)
{
}

void ChatServer::incomingConnection(int socketfd)
{
    QTcpSocket *client = new QTcpSocket(this);
    client->setSocketDescriptor(socketfd);
    clients.insert(client);
    emit clients_signal(clients.count());

    QString str;
    str = QString("새로운 접속자: %1").arg(client->peerAddress().toString());
    emit message_signal(str);

    connect(client, SIGNAL(readyRead()), this, SLOT(readyRead()));
    connect(client, SIGNAL(disconnected()), this, SLOT(disconnected()));
}

void ChatServer::readyRead()
{
    QTcpSocket *client = (QTcpSocket*)sender();
    while(client->canReadLine())
    {
        QString line = QString::fromUtf8(client->readLine()).trimmed();
        QString str = QString("Read line: %1").arg(line);
        emit message_signal(str);

        QRegExp meRegex("^/me:(.*)$");

        if(meRegex.indexIn(line) != -1) {
            QString user = meRegex.cap(1);
            users[client] = user;
            foreach(QTcpSocket *client, clients)
            {
                client->write(QString("서버: %1 접속").arg(user).toUtf8());
            }
            sendUserList();
        }
        else if(users.contains(client)) {
            QString message = line;
        }
    }
}

```

```

        QString user = users[client];

        QString str;
        str = QString("유저명: %1, 메시지: %2").arg(user).arg(message);
        emit message_signal(str);

        foreach(QTcpSocket *otherClient, clients)
            otherClient->write(QString(user+": "+message+"\n").toUtf8());
    }
}

void ChatServer::disconnected()
{
    QTcpSocket *client = (QTcpSocket*)sender();
    QString str = QString("접속자 연결 종료 :: %1")
                  .arg(client->peerAddress().toString());
    emit message_signal(str);

    clients.remove(client);
    emit clients_signal(clients.count());

    QString user = users[client];
    users.remove(client);

    sendUserList();
    foreach(QTcpSocket *client, clients)
        client->write(QString("서버: %1 접속종료").arg(user).toUtf8());
}

void ChatServer::sendUserList()
{
    QStringList userList;
    foreach(QString user, users.values())
        userList << user;

    foreach(QTcpSocket *client, clients)
        client->write(QString("/유저:" + userList.join(",") + "\n").toUtf8());
}

```

incomingConnection() 함수에서 QTcpSocket 을 새로 만드는 것은 새로운 클라이언트의 소켓 오브젝트를 생성하기 위함이다. 따라서 이 함수에서 생성된 QTcpSocket 클래스

스의 오브젝트는 users 라는 클라이언트 QMap 컨테이너에 저장된다.

readyRead() 함수는 서버에 접속한 클라이언트가 메시지를 보내면 호출되는 Slot 함수이다. 이 함수에서는 클라이언트가 보낸 메시지를 서버에 접속한 모든 클라이언트에게 전송한다.

disconnected() 함수는 클라이언트가 접속을 종료하면 호출되는 Slot 함수이다. 이 함수에서는 users 라는 QMap 컨테이너에서 접속을 종료한 클라이언트의 QTcpSocket 클래스의 오브젝트를 제거한다. 다음은 widget.h 소스코드이다.

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include "chatserver.h"

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
private:
    Ui::Widget *ui;
    ChatServer *server;
private slots:
    void slot_clients(int users);
    void slot_message(QString msg);
};

#endif // WIDGET_H
```

Widget 클래스에서 slot_clients() Slot 함수는 새로운 클라이언트 접속하거나 접속을 종료 했을 때 ChatServer 클래스에서 발생하는 시그널이며 최종 클라이언트의 접속자를 보내준다. 그리고 slot_message() 함수는 ChatServer 클래스에서 클라이언트가 메시지를 보내거나 접속을 종료하면 호출되는 Slot 함수이다. 다음 예제 소스는 widget.cpp 소스 코드이다.

```
#include "widget.h"
#include "chatserver.h"
#include "ui_widget.h"
```

```

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);

    server = new ChatServer();
    connect(server, SIGNAL(clients_signal(int)),
            this, SLOT(slot_clients(int)));
    connect(server, SIGNAL(message_signal(QString)),
            this, SLOT(slot_message(QString)));

    server->listen(QHostAddress::Any, 35000);
}

void Widget::slot_clients(int users)
{
    QString str = QString("전속자수 : %1").arg(users);
    ui->label->setText(str);
}

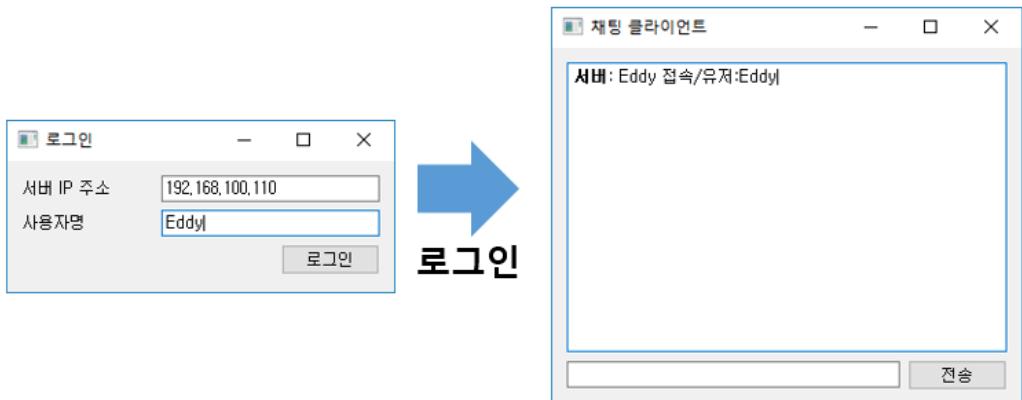
void Widget::slot_message(QString msg)
{
    ui->textEdit->append(msg);
}

Widget::~Widget()
{
    delete ui;
}

```

`slot_clients()` Slot 함수가 호출되면 GUI 상에서 접속자의 숫자를 업데이트 한다.
`slot_message()` Slot 함수는 QTextEdit 창에 메시지를 출력한다. 지금까지 채팅 서버에 대해 알아보았다. 다음은 채팅 서버와 메시지 송수신을 하는 클라이언트를 구현해 보도록 하자.

채팅 클라이언트는 화면이 2개로 구성되어 있다. 첫 번째는 아래 그림에서 보는 것과 같이 로그인 화면이다. 로그인 위젯에서 보는 것과 같이 [로그인] 버튼을 클릭하면 서버 IP주소로 채팅 서버에 접속이 완료 되고 로그인 위젯은 Hide 되고 채팅 클라이언트 위젯이 활성화(Show) 된다.



<그림> 채팅 클라이언트 예제 실행 화면

채팅 클라이언트 위젯에서 중간에 위치한 QTextEdit 위젯은 서버로부터 수신한 메시지를 출력한다. 하단의 QLineEdit 는 서버로 보낼 메시지를 입력한다. 그리고 [전송] 버튼을 클릭하면 메시지를 채팅 서버로 전송한다. 다음 예제 소스코드 widget.h 소스코드이다. 전체 예제 소스코드 Ch13 > 06_ChatClient 디렉토리를 참조하면 된다.

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QtNetwork/QTcpSocket>
#include "loginwidget.h"

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;
    LoginWidget *loginWidget;
    QTcpSocket *socket;
    QString     ipAddr;
    QString     userName;

private slots:
    void loginInfo(QString addr, QString name);
}
```

```

void sayButton_clicked();
void connected();
void readyRead();
};

```

loginInfo() Slot 함수는 LoginWidget 클래스에서 [로그인] 버튼을 클릭하면 로그인창에서 입력한 서버 IP주소와 사용자명을 전달하기 위한 시그널이 발생하며 이 시그널이 발생하면 호출되는 Slot 함수이다. 첫 번째 인자는 서버 IP주소이며 두 번째 인자는 사용자 명이다. 다음 예제는 widget.cpp 소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    loginWidget = new LoginWidget();

    connect(loginWidget, SIGNAL(loginInfo(QString, QString)),
            this, SLOT(loginInfo(QString, QString)));
    connect(ui->sayButton, &QPushButton::pressed,
            this, &Widget::sayButton_clicked);

    loginWidget->show();

    socket = new QTcpSocket(this);
    connect(socket, SIGNAL(readyRead()), this, SLOT(readyRead()));
    connect(socket, SIGNAL.connected(), this, SLOT(connected()));
}

void Widget::loginInfo(QString addr, QString name)
{
    ipAddr = addr;
    userName = name;
    socket->connectToHost(ipAddr, 35000);
}

void Widget::sayButton_clicked()
{
    QString message = ui->sayLineEdit->text().trimmed();
    if(!message.isEmpty()) {

```

```

        socket->write(QString(message + "\n").toUtf8());
    }
    ui->sayLineEdit->clear();
    ui->sayLineEdit->setFocus();
}

void Widget::connected()
{
    loginWidget->hide();
    this->window()->show();
    socket->write(QString("/me:" + userName + "\n").toUtf8());
}

void Widget::readyRead()
{
    while( socket->canReadLine() ) {
        QString line = QString::fromUtf8(socket->readLine()).trimmed();
        QRegExp messageRegex("^([^\:]+)(\.*$)");

        if(messageRegex.indexIn(line) != -1) {
            QString user = messageRegex.cap(1);
            QString message = messageRegex.cap(2);
            ui->roomTextEdit->append("<b>" + user + "</b>:" + message);
        }
    }
}

Widget::~Widget()
{
    delete ui;
}

```

sayButton_clicked() Slot 함수는 [전송] 버튼을 클릭하면 호출되는 함수이다. connected() Slot 함수는 채팅 서버와 연결이 완료되면 호출된다. readyRead()는 채팅 서버가 메시지를 전송하면 호출되는 Slot 함수이다. 이 Slot 함수에서 메시지를 QTextEdit 위젯에 출력한다.

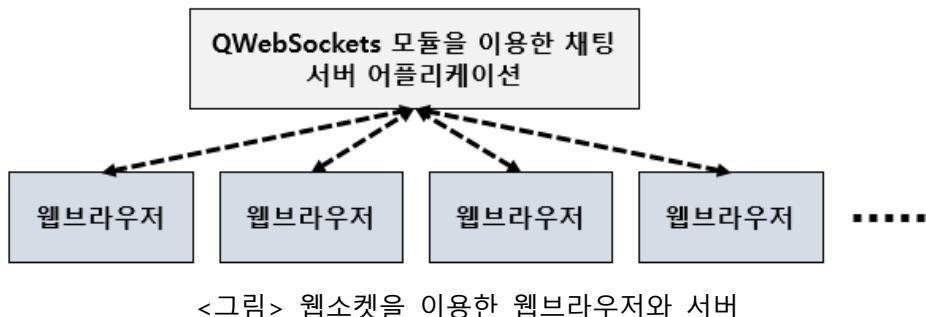
14. Qt WebSockets

웹브라우저를 이용해 포털 사이트(웹서버)에 접속해 원하는 정보를 얻을 수 있다. 이때 사용자가 웹브라우져에서 웹서버에게 정보를 요청하면 HTTP 프로토콜을 기반으로 TCP Connection 을 맺는다. 그리고 Connection 이 완료되면 웹브라우저는 원하는 정보를 웹서버에게 요청한다. 웹서버는 사용자가 원하는 정보를 전달하면 웹서버는 웹브라우저와 연결된 TCP Connection을 종료한다.

웹서버는 웹브라우져에게 요청한 정보를 전달하면 TCP Connection 을 종료하기 때문에 다시 정보를 얻기 위해서는 TCP Connection을 다시 맺어야 한다.

이러한 단점을 없애기 위해서 이전 장에서 배운 QTcpSocket 을 이용해 실시간 통신했던 것과 같이 새로고침 없이 현재 웹브라우저에서 보고 있는 페이지 정보 중 특정 부분의 데이터가 업데이트 되었을 때 웹서버가 실시간으로 정보를 갱신하기 위해서 웹소켓을 사용할 수 있다.

기존의 HTTP 프로토콜은 웹서버가 원하는 정보를 웹브라우저에게 정보를 전달하면 TCP Connection이 종료되지만 웹소켓은 TCP Connection 이 종료되지 않고 계속 연결된다. 웹소켓은 AJAX(Aynchronous JavaScript and XML)와 사용 목적과 동일하다.



위의 그림에서 보는 것과 같이 Qt WebSockets 모듈을 이용하면 웹브라우져와 통신할 수 있는 어플리케이션을 쉽게 구현할 수 있다.

Qt WebSockets 모듈을 이용하면 Node.js 를 이용해 서버 Side 어플리케이션을 쉽게 개발할 수 있는 것과 같이 Qt WebSockets 모듈을 이용해 쉽게 서버 Side 어플리케이션을 구현할 수 있다.

또한 Node.js 는 서버에서 필요한 멀티 Thread를 사용할 수 없다는 단점이 있다. 하지만 Qt WebSockets 은 멀티 Thread를 사용할 수 있다는 장점이 있다. Qt WebSockets

모듈을 사용하기 위해서는 프로젝트파일에 다음과 같이 추가 해야한다.

```
QT += websockets
```

Qt WebSockets 는 쉽게 개발할 수 있도록 QWebSocket 클래스를 제공한다. 그리고 서버 프로그래밍 시 QWebSocket 을 사용할 수 있지만 서버 어플리케이션을 쉽게 개발 할 수 있도록 QWebSocketServer 클래스를 제공한다.

QWebSocket 클래스는 네트워크 프로그래밍 장에서 다른 QTcpSocket과 거의 동일한 방법으로 사용할 수 있으며 복잡한 Thread 구현 없이 쉽게 구현할 수 있도록 Signal 과 Slot 을 이용해 이벤트를 처리할 수 있다.

예를 들어 QWebSocket 으로 구현한 서버 어플리케이션에 연결을 하고 TCP Connection 연결이 완료되면 QWebSocket 에서 제공하는 connected() 시그널이 발생 한다. 이 시그널이 발생하면 연결된 Slot 함수를 호출하기 때문에 Thread 없이 쉽게 구현할 수 있다. 다음은 Signal 과 Slot 을 연결한 예제 소스코드 일부이다.

```
EchoClient::EchoClient(const QUrl &url, bool debug, QObject *parent)
: QObject(parent), m_url(url)
{
    connect(&m_webSocket, &QWebSocket::connected,
            this,           &EchoClient::onConnected);
    connect(&m_webSocket, &QWebSocket::disconnected,
            this,           &EchoClient::closed);

    m_webSocket.open(QUrl(url));
}

void EchoClient::onConnected()
{
    connect(&m_webSocket, &QWebSocket::textMessageReceived,
            this,           &EchoClient::onTextMessageReceived);

    m_webSocket.sendTextMessage(QStringLiteral("Hello, world!"));
}

void EchoClient::onTextMessageReceived(QString message)
{
    qDebug() << "Message received:" << message;
}
```

위의 예는 QWebSocket 클래스를 이용해 구현한 예제 소스코드의 일부이다. 서버와 연

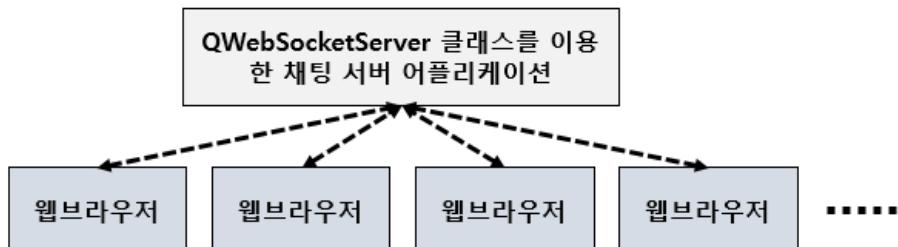
결이 완료되면 connected() 시그널이 발생하고, 이 시그널과 연결된 onConnected() Slot 함수가 호출된다. 그리고 onConnected() 함수에서 보는 것과 같이 서버가 보내온 메시지를 수신하면 시그널이 발생하고 이 시그널과 연결된 onTextMessageReceived() Slot 함수가 실행된다.

서버와 연결이 끊어지면(Close) disconnected() 시그널이 발생하고 closed()라는 Slot 함수가 호출된다. 위의 예제에서 보는 것과 같이 Qt WebSockets 모듈을 이용하면 웹브라우저와 통신할 수 있는 어플리케이션을 쉽게 구현할 수 있다. 다음은 웹브라우저와 Qt WebSockets 모듈을 이용해 통신하는 예제를 구현해 보도록 한다.

● 웹브라우저와 통신을 위한 서버 어플리케이션 구현

이번 예제에서는 웹브라우저와 통신을 하기 위해 Qt WebSockets 모듈을 이용해 서버 어플리케이션을 구현해 보도록 하자. 서버 어플리케이션을 구현하기 위해서 QWebSocketServer 클래스를 이용해 구현해 보도록 하자.

이 예제에서 구현할 서버와 통신하기 위해서 웹브라우저는 구글 크롬(Chrome), 파이어폭스, Safari 등을 사용하자. 다른 웹브라우저를 이용해도 된다. 하지만 이용할 웹브라우저가 HTML5를 지원하는지 확인해야 한다.



<그림> 구현할 채팅 서버와 통신할 웹브라우저간의 구조

위의 그림에서 보는 것과 같이 구현하기 위해서 웹브라우저 HTML 소스코드를 먼저 작성해 보도록 하자. 다음 예제 소스코드 HTML 소스코드이다. 프로젝트에 대한 전체 소스코드는 Ch14 > 01_Chat_Example 디렉토리를 참조하면 되다. 또한 예제 HTML 소스코드 파일도 이 디렉토리에 보면 chatclient.html 파일이 있으니 참조하면 된다.

```
<html>
  <head><title>웹소켓 채팅 클라이언트</title></head>
  <body>
    <h1>웹소켓 채팅 클라이언트</h1>
    <p>
```

```

<button onClick="initWebSocket();">서버연결</button>
<button onClick="stopWebSocket();">연결종료</button>
<button onClick="checkSocket();">상태</button>
</p>
<p>
    <textarea id="debugTextArea" style="width:400px;height:100px;">
    </textarea>
</p>
<p>
    <input type="text" id="inputNick" value="nickname" />
    <input type="text" id="inputText"
        onkeydown="if(event.keyCode==13)sendMessage();"/>
    <button onClick="sendMessage();">Send</button>
</p>

<script type="text/javascript">
    var debugTextArea = document.getElementById("debugTextArea");

    function debug(message) {
        debugTextArea.value += message + "\n";
        debugTextArea.scrollTop = debugTextArea.scrollHeight;
    }

    function sendMessage()
    {
        var nickname = document.getElementById("inputNick").value;
        var msg = document.getElementById("inputText").value;
        var strToSend = nickname + ": " + msg;

        if ( websocket != null ) {
            document.getElementById("inputText").value = "";
            websocket.send( strToSend );

            console.log( "string sent :", "'"+strToSend+"' " );
            debug(strToSend);
        }
    }

    var wsUri = "ws://localhost:1234";
    var websocket = null;

    function initWebSocket() {

```

```

try {
    if (typeof MozWebSocket == 'function')
        WebSocket = MozWebSocket;

    if ( websocket && websocket.readyState == 1 )
        websocket.close();

    websocket = new WebSocket( wsUri );

    websocket.onopen = function (evt) {
        debug("CONNECTED");
    };

    websocket.onclose = function (evt) {
        debug("DISCONNECTED");
    };

    websocket.onmessage = function (evt) {
        console.log("Message received: ", evt.data );
        debug( evt.data );
    };

    websocket.onerror = function (evt) {
        debug('ERROR: ' + evt.data);
    };
}
catch (exception) {
    debug('ERROR: ' + exception);
}

function stopWebSocket() {
    if (websocket)
        websocket.close();
}

function checkSocket() {
    if (websocket != null) {
        var stateStr;
        switch (websocket.readyState) {
            case 0: { stateStr = "CONNECTING"; break; }
            case 1: { stateStr = "OPEN"; break; }
        }
    }
}

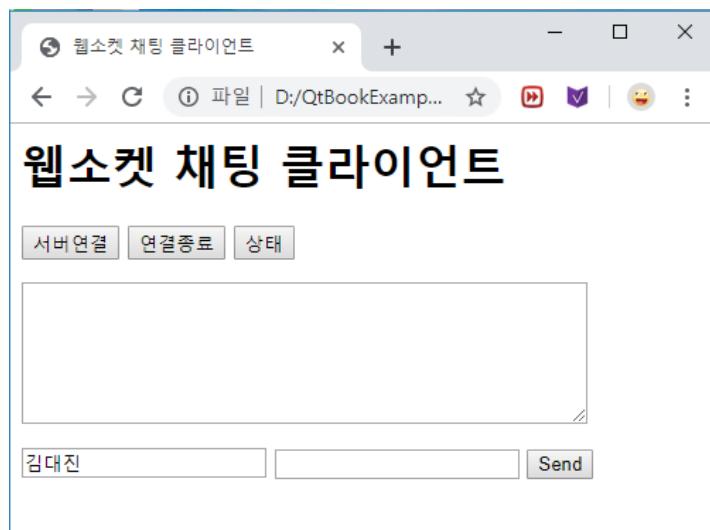
```

```

        case 2: { stateStr = "CLOSING"; break; }
        case 3: { stateStr = "CLOSED"; break; }
        default: { stateStr = "UNKNOW"; break; }
    }
    debug("WebSocket state = " +
        websocket.readyState + " (" + stateStr + ")");
} else {
    debug("WebSocket is null");
}
}
</script>
</body>
</html>

```

위의 HTML 소스코드로 작성한 웹소켓 기반의 채팅 클라이언트를 아래 그림에서 보는 것과 같이 웹브라우저로 실행시켜 보도록 하자.



<그림> 작성한 HTML 파일을 실행한 웹브라우저 화면

위의 그림에서 보는 것과 같이 작성한 HTML 소스코드를 웹브라우저에서 실행한다. 이번에는 웹브라우저와 통신할 서버 어플리케이션을 작성해 보도록 하자. Qt Creator에서 GUI가 없는 콘솔 기반의 프로젝트를 생성한다. ChatServer 클래스를 생성하고 헤더 파일 소스코드를 다음과 같이 소스코드를 작성해 보자.

```

#ifndef CHATSERVER_H
#define CHATSERVER_H

#include <QtCore/QObject>

```

```

#include <QtCore/QString>
#include <QtCore/QObject>

QT_FORWARD_DECLARE_CLASS(QWebSocketServer)
QT_FORWARD_DECLARE_CLASS(QWebSocket)

class ChatServer : public QObject
{
    Q_OBJECT
public:
    explicit ChatServer(quint16 port,
    QObject *parent = Q_NULLPTR);

    virtual ~ChatServer();

private Q_SLOTS:
    void onNewConnection();
    void processMessage(QString message);
    void socketDisconnected();

private:
    QWebSocketServer *m_pWebSocketServer;
    QList<QWebSocket *> m_clients;
};

#endif //CHATSERVER_H

```

이 예제 소스코드에서는 웹서버를 구현하기 위해서 QWebSocketServer 클래스를 사용하였다. onNewConnection() Slot 함수는 새로운 접속자 Signal 이 발생하면 호출된다. processMessage() Slot 함수는 클라이언트가 보내온 메시지를 서버에 접속한 모든 클라이언트(웹브라우저)에게 메시지를 전달한다.

socketDisconnected() Slot 함수는 특정 클라이언트 접속을 종료하면 접속한 클라이언트와 통신하기 위한 QWebSocket 클래스 오브젝트를 제거하는 기능이 구현되어 있다. 다음 예제 소스코드는 ChatServer 클래스의 구현 부 소스코드이다.

```

#include "chatserver.h"
#include "QtWebSockets/QWebSocketServer"
#include "QtWebSockets/QWebSocket"
#include <QtCore/QDebug>

QT_USE_NAMESPACE

```

```

ChatServer::ChatServer(quint16 port, QObject *parent) :
    QObject(parent), m_pWebSocketServer(Q_NULLPTR), m_clients()
{
    m_pWebSocketServer = new QWebSocketServer(QStringLiteral("Chat Server"),
                                              QWebSocketServer::NonSecureMode,
                                              this);

    if (m_pWebSocketServer->listen(QHostAddress::Any, port))
    {
        qDebug() << "Chat Server listening on port" << port;
        connect(m_pWebSocketServer, &QWebSocketServer::newConnection,
                this,             &ChatServer::onNewConnection);
    }
}

void ChatServer::onNewConnection()
{
    QWebSocket *pSocket = m_pWebSocketServer->nextPendingConnection();
    connect(pSocket, &QWebSocket::textMessageReceived,
            this,     &ChatServer::processMessage);
    connect(pSocket, &QWebSocket::disconnected,
            this,     &ChatServer::socketDisconnected);

    m_clients << pSocket;
}

void ChatServer::processMessage(QString message)
{
    QWebSocket *pSender = qobject_cast<QWebSocket *>(sender());
    Q_FOREACH (QWebSocket *pClient, m_clients) {
        if (pClient != pSender)
            pClient->sendTextMessage(message);
    }
    qDebug() << "Send Message : " << message;
}

void ChatServer::socketDisconnected()
{
    QWebSocket *pClient = qobject_cast<QWebSocket *>(sender());
    if (pClient) {
        m_clients.removeAll(pClient);
        pClient->deleteLater();
    }
}

```

```

    }

ChatServer::~ChatServer()
{
    m_pWebSocketServer->close();
    qDeleteAll(m_clients.begin(), m_clients.end());
}

```

위와 같이 작성한 ChatServer 클래스를 main.cpp에서 다음과 같이 수행한다. 다음 예제 소스코드는 main.cpp 소스코드이다.

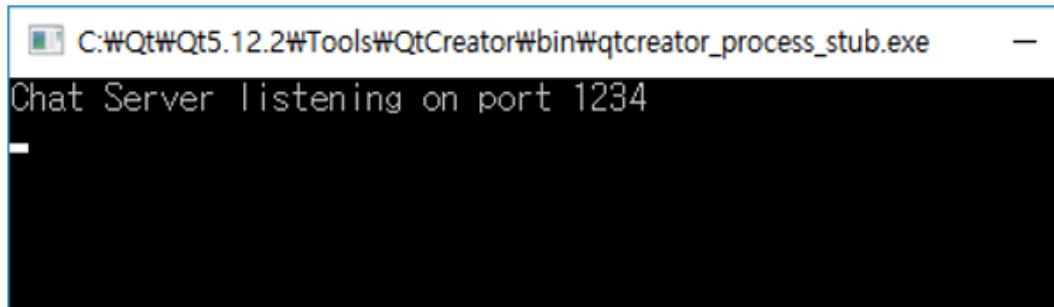
```

#include <QtCore/QCoreApplication>
#include "chatserver.h"

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    ChatServer server(1234);
    return a.exec();
}

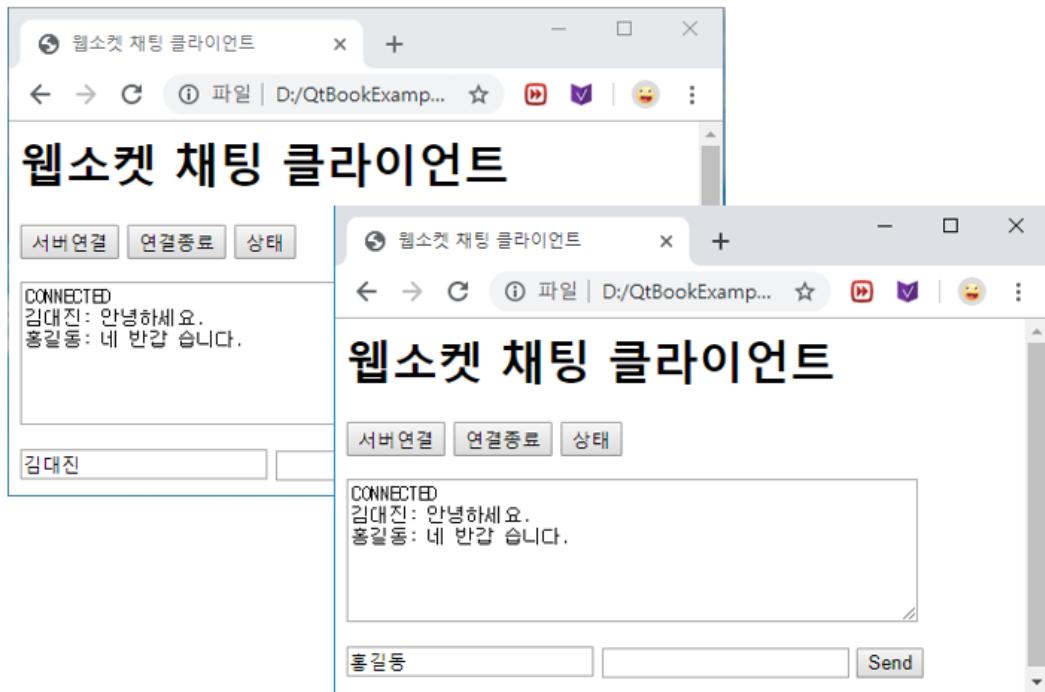
```

위의 예제 소스코드에서 ChatServer 오브젝트의 첫 번째 인자는 포트번호이다. 위와 같이 작성하고 어플리케이션을 실행해보자. 실행하면 다음과 같은 콘솔 창이 실행될 것이다.



<그림> 서버 실행 화면

다음은 작성한 HTML 코드를 실행한 2개의 웹브라우저를 아래 그림에서 보는것과 같이 실행한다. 그리고 [서버연결] 버튼을 클릭하고 메시지를 입력 후 [Send] 버튼을 클릭해보자.



<그림> 웹브라우저 실행 화면

그리고 서버를 실행한 콘솔 창에서 다음과 같은 로그를 확인할 수 있다.

```
C:\Qt\Qt5.12.2\Tools\QtCreator\bin\qtcreator_process_stub.exe
Chat Server listening on port 1234
New Connection
New Connection
Send Message : "김대진: 안녕하세요."
Send Message : "홍길동: 네 반갑습니다."
```

<그림> 서버 실행 화면

15. Qt WebEngine

Qt WebEngine 모듈은 웹브라우저 엔진을 이용해 응용 어플리케이션을 개발할 수 있는 기능을 제공한다. WebEngine 모듈은 구글 크롬(Chrome) 기반의 오픈소스 웹브라우저 엔진을 사용한다.

Qt 5.5 버전 이하에서는 Qt WebEngine 모듈 대신 Qt WebKit 모듈을 사용하였다. Qt WebKit 모듈은 WebKit 오픈소스 기반이며 Qt WebEngine 모듈은 Google Chrome 웹 엔진 기반이다.

MS윈도우 운영체제에서는 MinGW 컴파일러는 Qt WebEngine 모듈을 지원하지 않는다.

그러므로 MS윈도우에서 Qt WebEngine 을 이용해 응용 어플리케이션을 구현하기 위해서는 MSVC 컴파일러 사용해야 한다. Qt WebEngine 모듈을 사용하기 위해서는 프로젝트파일 다음과 같이 사용해야 한다.

```
QT += webenginewidgets
```

- QWebView 클래스를 이용한 간단한 웹브라우저 구현

이번 예제에서는 QWebView 클래스를 이용한 간단한 웹브라우저 기능을 구현해 보도록 하자. 다음 그림은 예제를 실행한 화면이다.

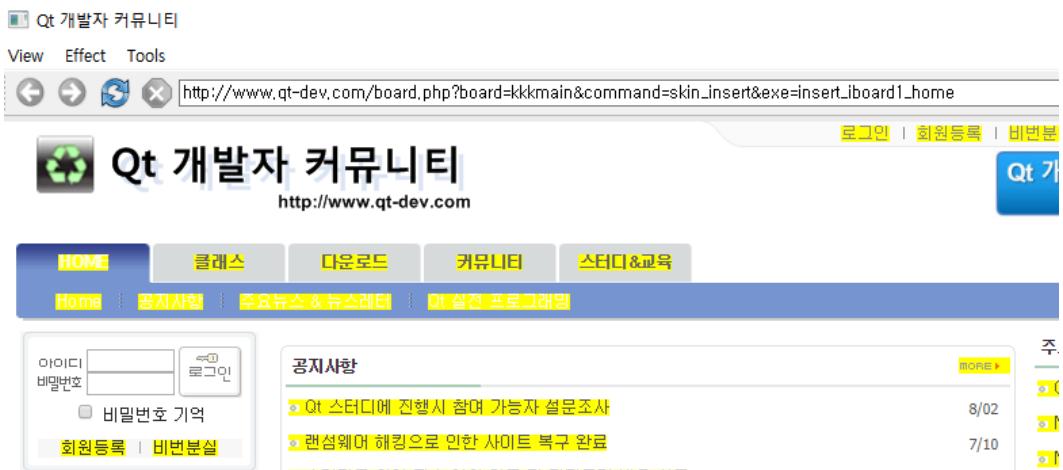


<그림> 예제 실행 화면

위의 그림에서 보는 것과 같이 예제를 실행 하면 Qt 개발자 커뮤니티 사이트 처음 페이지를 로딩하는 예제이다. 메뉴에서 첫 번째 View 메뉴에는 [Page Source] 메뉴이다. 이 메뉴를 클릭하면 현재 웹 페이지의 소스코드를 볼 수 있는 창이 로딩된다.



<그림> View 메뉴 실행 시 소스코드 보기 창 로딩



<그림> Effect 메뉴에서 [Highlight all links] 메뉴 클릭 시 화면

위의 메뉴에서 [Effect] 메뉴 > [Highlight all links] 메뉴를 클릭 위의 그림에서 보는 것과 같이 웹페이지에서 하이퍼링크 HTML 태그의 색을 노란색으로 변경한다.

Tools 메뉴에서 [Remove GIF images] 를 클릭하면 웹페이지에서 사용한 GIF 이미지를 모두 HTML 태그에서 제거한다. 메뉴 하단의 툴바 메뉴에서 첫 번째 아이콘은 현재 웹페이지에서 이전 페이지로 돌아가기 기능을 제공한다. 두 번째 아이콘은 다음 페이지,

세 번째 아이콘은 현재 페이지를 새로고침 하는 기능을 제공한다. 네 번째 아이콘은 현재 로딩 중일 때 중지하는 기능을 제공한다. 그리고 우측의 주소창은 현재 웹페이지 URL 주소 창이다.

프로젝트 생성 시 QMainWindow 클래스를 Base 클래스로 프로젝트를 생성하며 MainWindow 클래스의 헤더를 다음과 같이 작성한다. 전체 예제소스코드 Ch15 > 01_Simgle_Browser 디렉토리를 참조 하면 된다.

```
#include <QtWidgets>

QT_BEGIN_NAMESPACE
class QWebEngineView;
class QLineEdit;
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(const QUrl& url);

protected slots:
    void adjustLocation();
    void changeLocation();
    void adjustTitle();
    void setProgress(int p);
    void finishLoading(bool);
    void viewSource();
    void highlightAllLinks();
    void removeGifImages();

private:
    QString jQuery;
    QWebEngineView *view;
    QLineEdit *locationEdit;
    int progress;
};
```

adjustLocation() Slot 함수는 주소창에 현재 웹페이지 주소를 표시하는 기능을 제공한다. 웹페이지 로딩이 완료되면 QWebEngineView 클래스의 loadFinished() Signal이 발생하고 이 시그널 발생 시 adjustLocation() Slot 함수가 호출된다.

changeLocation() Slot 함수는 주소 창에서 사용자가 엔터를 클릭하면 주소창의 주소를 가져와 페이지를 로딩하기 위해 주소창에 입력한 웹서버로 웹 페이지를 요청한다.

adjustTitle() Slot 함수는 웹 페이지의 <title> 태그의 문자열을 QMainWindow 위젯 타이틀 바에 표시한다. 그리고 finishLoading() 함수는 웹페이지 가져오기가 완료되면 호출된다.

viewSource() Slot 함수는 View 메뉴에서 [Page Source] 를 클릭하면 호출되는 함수이다. highlightAllLinks() Slot 함수는 Effect 메뉴에서 [Highlight all links] 버튼을 클릭하면 호출 된다. 그리고 Tools 메뉴에서 [Remove GIF images] 를 클릭하면 removeGifImages() 함수를 호출 한다.

이 프로젝트에서 jQuery 를 사용하였다. jQuery 는 JavaScript 를 좀더 쉽게 사용하기 위해 오픈소스 자바스크립트이다. 다음 예제 소스코드 MainWindow 함수 구현 소스코드 이다.

```
#include <QtWidgets>
#include <QtWebEngineWidgets>
#include "mainwindow.h"

MainWindow::MainWindow(const QUrl& url)
{
   setAttribute(Qt::WA_DeleteOnClose, true);
progress = 0;

QFile file;
file.setFileName(":/jquery.min.js");
file.open(QIODevice::ReadOnly);
jQuery = file.readAll();
jQuery.append("\nvar qt={'jQuery': jQuery.noConflict(true)};");

file.close();

view = new QWebEngineView(this);
view->load(url);
connect(view, &QWebEngineView::loadFinished,
        this, &MainWindow::adjustLocation);
connect(view, &QWebEngineView::titleChanged,
        this, &MainWindow::adjustTitle);
connect(view, &QWebEngineView::loadProgress,
        this, &MainWindow::setProgress);
```

```

connect(view, &QWebEngineView::loadFinished,
        this, &MainWindow::finishLoading);

locationEdit = new QLineEdit(this);
locationEdit->setSizePolicy(QSizePolicy::Expanding,
                             locationEdit->sizePolicy().verticalPolicy());

connect(locationEdit, &QLineEdit::returnPressed,
        this, &MainWindow::changeLocation);

QToolBar *toolBar = addToolBar(tr("Navigation"));

toolBar->addAction(view->pageAction(QWebEnginePage::Back));
toolBar->addAction(view->pageAction(QWebEnginePage::Forward));
toolBar->addAction(view->pageAction(QWebEnginePage::Reload));
toolBar->addAction(view->pageAction(QWebEnginePage::Stop));
toolBar->addWidget(locationEdit);

QMenu *viewMenu = menuBar()->addMenu(tr("&View"));
 QAction *viewSourceAction = new QAction(Page Source, this);

connect(viewSourceAction, &QAction::triggered,
        this, &MainWindow::viewSource);

viewMenu->addAction(viewSourceAction);
QMenu *effectMenu = menuBar()->addMenu(tr("&Effect"));

effectMenu->addAction("Highlight all links",
                      this,
                      &MainWindow::highlightAllLinks);

QMenu *toolsMenu = menuBar()->addMenu(tr("&Tools"));
toolsMenu->addAction("Remove GIF images",
                      this,
                      &MainWindow::removeGifImages);
setCentralWidget(view);
}

void MainWindow::viewSource()
{
    QTextEdit *TextEdit = new QTextEdit(nullptr);
    TextEdit->setAttribute(Qt::WA_DeleteOnClose);
}

```

```

        textEdit->adjustSize();
        textEdit->move(this->geometry().center() - textEdit->rect().center());
    }

    textEdit->show();
    view->page()->toHtml([TextEdit](const QString &html) {
        textEdit->setPlainText(html);
    });
}

void MainWindow::adjustLocation()
{
    locationEdit->setText(view->url().toString());
}

void MainWindow::changeLocation()
{
    QUrl url = QUrl::fromUserInput(locationEdit->text());
    view->load(url);
    view->setFocus();
}

void MainWindow::adjustTitle()
{
    if (progress <= 0 || progress >= 100)
        setWindowTitle(view->title());
    else
        setWindowTitle(QStringLiteral("%1 (%2%)")
                      .arg(view->title()).arg(progress));
}

void MainWindow::setProgress(int p)
{
    progress = p;
    adjustTitle();
}

void MainWindow::finishLoading(bool)
{
    progress = 100;
    adjustTitle();
    view->page()->runJavaScript(jQuery);
}

```

```
void MainWindow::highlightAllLinks()
{
    QString code = QStringLiteral(
        "qt.jQuery('a').each( function () "
        "{ qt.jQuery(this).css('background-color',"
        " 'yellow') } )"
    );

    view->page()->runJavaScript(code);
}

void MainWindow::removeGifImages()
{
    QString code;
    code = QStringLiteral("qt.jQuery('[src*=gif]').remove()");
    view->page()->runJavaScript(code);
}
```

16. Inter Process Communication

IPC (Inter Process Communication)는 같은 시스템 내에 존재하는 프로세스간의 통신을 의미한다. Qt에서는 IPC(Inter Process Communication)를 지원하기 위해 다음과 같은 방법을 제공한다.

① Unix Domain Socket 과 Named pipe

리눅스에서는 UDS(Unix Domain Socket), MS Windows에서는 Named pipe 이름을 사용하는 프로세스간 통신.

② QProcess 클래스를 이용한 외부 프로세스와의 통신.

외부 프로세스를 실행 또는 결과를 얻어 오기 위해 사용

③ QSerialPort 클래스를 이용한 시리얼(Serial) 통신.

UART (Universal Asynchronous Receiver/Transmitter) 를 이용한 Serial 통신

④ Shared Memory를 이용해 외부 프로그램에서 메모리를 공유.

QSharedMemory 클래스를 이용해 외부 프로그램 간 공유 메모리를 사용.

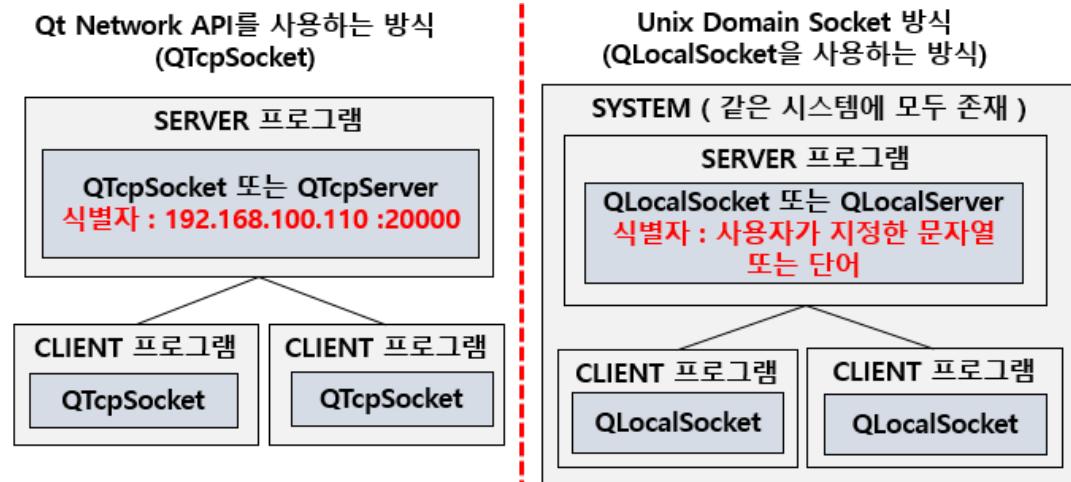
이번 장에서는 위에서 언급한 같은 시스템내에서 프로세스간 통신하는 방법에 대해서 살펴 보도록 하자.

16.1. Unix Domain Socket 과 Named pipe

UDS (Unix Domain Socket) 란 이름은 리눅스 플랫폼 사용하는 프로세스간 통신 방법의 이름이다. MS Windows 플랫폼에서는 “Named pipe”라는 이름으로 사용된다 둘 다 사용 목적은 동일하나 명칭은 다르다.

Qt 에서는 리눅스를 사용하든지 MS Windows를 사용하든지 상관없이 QLocalSocket 과 QLocalServer 클래스를 사용하면 된다. Unix Domain Socket 방식은 상대방을 식별 할 수 있는 방법으로 프로세스의 이름을 사용한다.

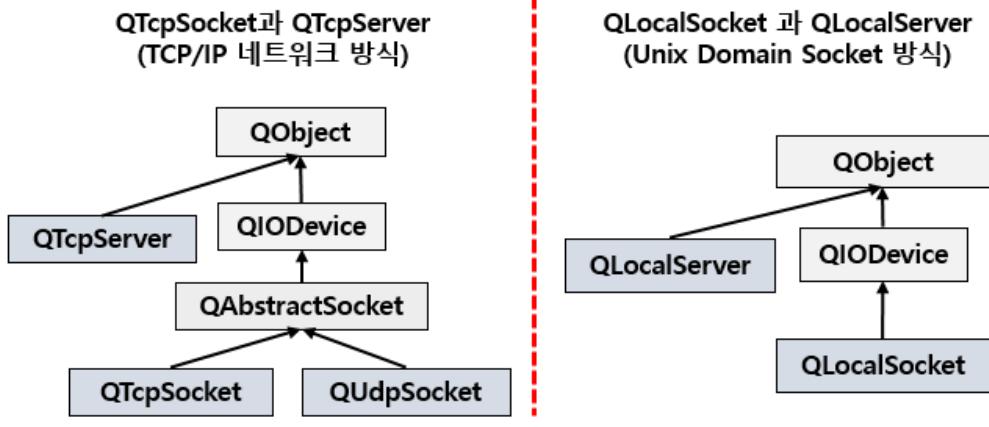
예를 들어 네트워크에서 특정 서버와 통신을 위해서는 식별자로 IP와 PORT 를 사용하는 것과 같이 Unix Domain Socket은 프로세스 이름 (또는 어플리케이션의 이름)을 사용자가 정의한 문자열(단어)을 식별자로 사용할 수 있다.



<그림> Network 통신 방식과 Unix Domain Socket 방식의 차이점

QLocalSocket 과 QLocalServer 클래스는 네트워크에서 사용했던 QTcpSocket 과 QTcpServer 클래스와 사용 방법이 매우 비슷하며 Signal 과 Slot 을 이용해 쉽게 구현할 수 있다.

QLocalSocket 과 QLocalServer 클래스는 동일하지만 QLocalServer 서버 사이드 프로그램 작성에 적합하며 QLocalSocket 클라이언트 프로그램 작성시 사용하는 것이 적합하다. 예를 들어 서버 사이드에서 클라이언트 접속을 하기 위해서 listen 하기 위한 기능 등이 QLocalServer 클래스에서는 제공된다.



<그림> 클래스 상속 관계

QLocalSocket 클래스를 이용해 클라이언트를 접속을 기다리는 listen 하기 위한 기능 구현이 가능하나 QLocalServer 클래스를 사용하는 것을 권장한다.

TCP/IP 네트워크 프로그래밍에서 서버가 클라이언트 접속을 대기하기 위해 QTcpServer 클래스가 제공하는 listen() 멤버 함수를 사용한 것과 같이 QLocalServer 클래스도 클라이언트를 접속을 기다리기 위해 listen() 멤버 함수를 제공하며 다음과 같이 사용 할 수 있다.

```

server = new QLocalServer(this);

if (!server->listen("MyServer")) {
    qDebug() << "Server error : " << server->errorString();
    ...
}

connect(server, SIGNAL(newConnection()),
        this,   SLOT(clientConnection()));
...

```

TCP/IP 프로토콜 네트워크에서 클라이언트가 서버에 접속 하기 위해서 IP와 PORT를 사용하지만 Unix Domain Socket 에서는 사용자가 원하는 이름을 사용할 수 있다.

connect() 멤버 함수는 새로운 클라이언트가 접속하면 newConnection() 시그널을 clientConnection() Slot 함수와 연결 하였다. 다음 예제 소스코드는 QLocalSocket 클래스를 이용해 클라이언트가 서버에 접속하기 위한 예제 소스코드의 일부이다.

```

socket = new QLocalSocket(this);

connect(socket, SIGNAL(readyRead()), this, SLOT(readData()));

```

```

connect(socket, SIGNAL(error(QLocalSocket::LocalSocketError)),
        this, SLOT(sockError(QLocalSocket::LocalSocketError)));

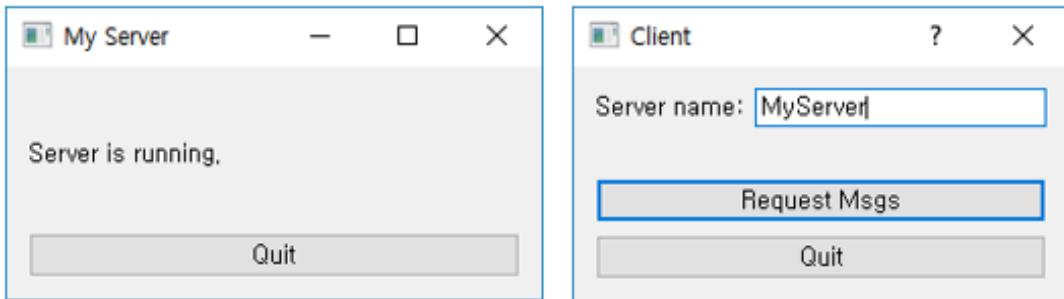
socket->connectToServer("MyServer");
...

```

connect() 함수에서 readyRead() 시그널은 클라이언트가 메시지를 수신하면 발생하는 시그널이다. 여기서는 메시지 시그널을 처리하기 위해 readData() Slot 함수와 연결 했다.

그리고 하단의 QLocalSocket 클래스의 connectToServer() 멤버 함수는 서버에 접속하기 위한 기능을 제공한다. connectToServer() 멤버 함수의 첫 번째 인자는 서버의 식별자 이름이다.

- QLocalServer 와 QLocalSocket 클래스를 이용한 프로세스간 통신 구현



<그림> 서버와 클라이언트 예제 실행 하면

좌측의 예제 실행 화면은 서버 프로그램이고 우측은 클라이언트 프로그램이다. 클라이언트 프로그램이 실행 되면 GUI 상에 서버 이름을 입력하고 [Request Msgs] 버튼을 클릭하면 서버 프로그램에 접속한다. 서버와 클라이언트 중 서버 프로그램을 먼저 살펴보도록 하자. 다음 예제 소스코드는 서버에 구현된 Server 클래스의 헤더 소스이다.

```

#ifndef SERVER_H
#define SERVER_H

#include <QWidget>

QT_BEGIN_NAMESPACE
class QLabel;
class QPushButton;
class QLocalServer;
QT_END_NAMESPACE

```

```

class Server : public QWidget
{
    Q_OBJECT

public:
    Server(QWidget *parent = 0);

private slots:
    void clientConnection();

private:
    QLabel *statusLabel;
    QPushButton *quitButton;
    QLocalServer *server;
    QStringList sendMsgs;
    bool msgKind;
};

#endif

```

clientConnect() Slot 함수는 새로운 클라이언트가 연결을 요청하면 호출되는 Slot 함수이다. 다음 예제는 Server 클래스의 구현 부 소스코드이다.

```

#include <QtWidgets>
#include <QtNetwork>
#include "server.h"
#include <qlocalserver.h>
#include <qlocalsocket.h>

Server::Server(QWidget *parent) : QWidget(parent)
{
    statusLabel = new QLabel;
    statusLabel->setWordWrap(true);
    quitButton = new QPushButton(tr("Quit"));
    quitButton->setAutoDefault(false);

    server = new QLocalServer(this);

    if (!server->listen("MyServer"))
    {
        qDebug() << "Server error : " << server->errorString();
        close();
        return;
    }
}

```

```

    }

    connect(quitButton, SIGNAL(clicked()), this, SLOT(close()));
    connect(server, SIGNAL(newConnection()), this, SLOT(clientConnection()));

    statusLabel->setText(tr("Server is running."));
    QVBoxLayout *mainLayout = new QVBoxLayout;
    mainLayout->addWidget(statusLabel);
    mainLayout->addWidget(quitButton);
    setLayout(mainLayout);

    setWindowTitle(tr("My Server"));
}

void Server::clientConnection()
{
    QByteArray writeData;

    if(msgKind) {
        writeData.append("Welcome to my server.");
        msgKind = false;
    } else {
        writeData.append("Who are you");
        msgKind = true;
    }

    QLocalSocket *clientConnection;
    clientConnection = server->nextPendingConnection();

    connect(clientConnection, SIGNAL(disconnected()),
            clientConnection, SLOT(deleteLater()));

    clientConnection->write(writeData, writeData.size());
    clientConnection->flush();
    clientConnection->disconnectFromServer();
}

```

위의 예제에서 생성자 함수에서는 QLocalServer 클래스의 오브젝트를 선언하고 클라이언트 접속을 대기하기 위해서 listen() 함수를 사용하였다. listen() 함수의 인자는 클라이언트가 서버에 접속할 고유한 식별자이다.

connect() 함수에서는 새로운 클라이언트가 접속하면 newConnection() 시그널 이벤트

가 발생하고 clientConnection() Slot 함수가 호출된다. 이 Slot 함수에서는 접속한 클라이언트에게 메시지를 송신한 후 연결을 종료한다. 다음은 클라이언트 프로그램에서 구현한 Client 클래스의 헤더 소스코드이다.

```
#ifndef CLIENT_H
#define CLIENT_H

#include <QDialog>
#include <QLocalSocket>
#include <qlocalsocket.h>

QT_BEGIN_NAMESPACE
class QDialogButtonBox;
class QLabel;
class QLineEdit;
class QPushButton;
class QLocalSocket;
QT_END_NAMESPACE

class Client : public QDialog
{
    Q_OBJECT
public:
    Client(QWidget *parent = 0);
private slots:
    void requestNewMsg();
    void readData();
    void sockError(QLocalSocket::LocalSocketError socketError);
private:
    QLabel *hostLabel;
    QLineEdit *hostLineEdit;
    QLabel *statusLabel;
    QPushButton *reqButton;
    QPushButton *quitButton;
    QLocalSocket *socket;
    quint16 blockSize;
};

#endif
```

위의 예에서 보는 것과 같이 requestNewMsg() Slot 함수는 [Request Msgs] 버튼을 클릭하면 호출되는 Slot 함수이다. readData() Slot 함수는 서버가 보낸 메시지를 수신할 때 호출되며 sockError() Slot 함수는 에러가 발생하면 호출되는 Slot 함수이다. 다음은

Client 클래스의 구현 부 소스코드이다.

```
#include <QtWidgets>
#include <QtNetwork>
#include "client.h"

Client::Client(QWidget *parent) : QDialog(parent)
{
    hostLabel      = new QLabel(tr("Server name:"));
    hostLineEdit   = new QLineEdit("MyServer");
    statusLabel    = new QLabel(tr(""));
    reqButton      = new QPushButton(tr("Request Msgs"));
    quitButton     = new QPushButton(tr("Quit"));

    socket = new QLocalSocket(this);
    connect(reqButton, SIGNAL(clicked()), this, SLOT(requestNewMsg()));
    connect(quitButton, SIGNAL(clicked()), this, SLOT(close()));
    connect(socket, SIGNAL(readyRead()), this, SLOT(readData()));
    connect(socket, SIGNAL(error(QLocalSocket::LocalSocketError)),
            this, SLOT(sockError(QLocalSocket::LocalSocketError)));

    QGridLayout *mainLayout = new QGridLayout;

    mainLayout->addWidget(hostLabel, 0, 0);
    mainLayout->addWidget(hostLineEdit, 0, 1);
    mainLayout->addWidget(statusLabel, 2, 0, 1, 2);
    mainLayout->addWidget(reqButton, 3, 0, 1, 2);
    mainLayout->addWidget(quitButton, 4, 0, 1, 2);
    setLayout(mainLayout);

    setWindowTitle(tr("Client"));
    hostLineEdit->setFocus();
}

void Client::requestNewMsg()
{
    reqButton->setEnabled(false);
    socket->connectToServer(hostLineEdit->text());
}

void Client::readData()
{
    statusLabel->setText(socket->readAll());
```

```

    reqButton->setEnabled(true);
}

void Client::sockError(QLocalSocket::LocalSocketError socketError)
{
    switch (socketError)
    {
    case QLocalSocket::ServerNotFoundError:
        qDebug() << "Server Not Found Error";
        break;
    case QLocalSocket::ConnectionRefusedError:
        qDebug() << "Connection Refused Error";
        break;
    case QLocalSocket::PeerClosedError:
        qDebug() << "Peer Closed Error";
        break;
    default:
        qDebug() << "Error : " << socket->errorString();
    }

    reqButton->setEnabled(true);
}

```

Client 클래스 생성자 함수는 GUI 위젯을 배치하고 QLocalSocket 클래스의 오브젝트를 선언한다. connect() 함수에서 reqButton 오브젝트는 [Request Msgs] 버튼 클릭 시 requestNewMsg() Slot 함수가 호출된다.

이 Slot 함수에서는 QLineEdit 에 입력한 서버 식별자를 이용해 서버 프로그램에 접속 한다. 그리고 클라이언트의 QLocalSocket 에서 에러가 발생하면 sockError() Slot 함수 가 호출된다. 이 함수에서는 어떤 에러가 발생했는지 확인할 수 있는 기능을 제공 한다. 서버 프로그램의 전체 소스는 Ch16 > 01_LocalServer 디렉토리를 참조하면 된다. 그리고 클라이언트 프로그램 전체 소스는 Ch16 > 01_LocalClient 디렉토리를 참조하면 된다.

16.2. QProcess 를 이용한 외부 프로세스와의 통신

QProcess 클래스는 구현한 응용 프로그램 내에서 외부 프로그램을 실행하거나 실행한 결과를 가져올 수 있는 기능을 제공한다. 예를 들어 리눅스에서 “/bin/ls” 는 디렉토리와 파일 정보를 출력한다. “/bin/ls” 를 이용해 현재 디렉토리가 아닌 “/usr” 디렉토리안에 디렉토리 및 파일 정보를 자세히 출력하기 위해 다음과 같이 Arguments 를 사용할 수 있다.

```
/bin/ls -ltr /usr
```

위의 예제에서 보는 것과 같이 명령을 리눅스 터미널에서 /usr 디렉토리 안에 있는 디렉토리 및 파일 정보를 출력하기 위해서 QProcess를 사용할 수 있다.

```
QString program = "/bin/ls";
QStringList arguments;
arguments << "-ltr" << "/usr";

QProcess *myProcess = new QProcess(parent);
myProcess->start(program, arguments);
```

위의 예제에서 보는 것과 같이 QProcess 클래스의 start() 멤버 함수를 실행하면 외부 프로그램을 실행 하며 첫 번째 인자는 실행할 프로그램 명(또는 파일명)을 입력 한다. 두 번째 arguments 는 프로그램 뒤에 사용할 수 있는 옵션을 사용할 수 있으며 만약 옵션이 없으면 첫 번째 인자만 사용해도 된다. QProcess 에 의해 실행되는 결과는 Signal 과 Slot 에 의해 결과를 얻어 올 수 있으며 다음 예제를 통해 자세히 알아보도록 하자.

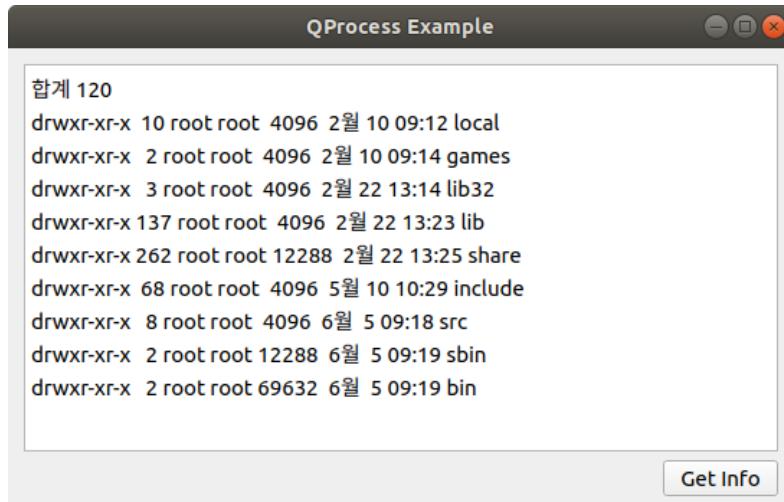
- QProcess 클래스를 이용한 외부 프로그램 실행 결과 가져오기 예제

이번 예제는 QProcess 클래스를 이용해 외부 프로그램을 실행 후 결과를 얻어와 GUI 위젯에 결과를 출력해 보도록 하자. 리눅스에서 “/bin/ls -ltr /usr” 명령을 실행하면 다음과 같은 결과를 출력한다.

```
qtdev@linux1:~$ ls -ltr /usr
합계 120
drwxr-xr-x 10 root root 4096 2월 10 09:12 local
drwxr-xr-x  2 root root 4096 2월 10 09:14 games
```

```
drwxr-xr-x 3 root root 4096 2월 22 13:14 lib32
drwxr-xr-x 137 root root 4096 2월 22 13:23 lib
...
```

위에서 보는 것과 같이 “/bin/ls” 명령어와 함께 사용한 “-ltr” 옵션은 파일 및 디렉토리 정보를 자세히 출력하는 옵션이며 /usr 옵션은 출력할 디렉토리 명이다. 다음은 예제를 실행한 화면이다.



<그림> 예제 실행 화면

위의 그림에 보는 것과 같이 하단의 [Get Info] 버튼을 클릭하면 외부 명령을 실행하고 결과를 얻어와 QTextEdit 창에 결과를 출력한다. 프로젝트 생성 시 QWidget 을 상속받는 클래스를 생성하고 widget.h 파일을 다음과 같이 작성한다. 예제 전체 소스는 Ch16 > 02_QProcessExample 디렉토리를 참조하면 된다.

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QProcess>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();
```

```

private:
    Ui::Widget *ui;
    QProcess *m_process;
private slots:
    void getInfoButton();
    void finished(int exitCode, QProcess::ExitStatus exitStatus);
    void readyReadStandardError();
    void readyReadStandardOutput();
    void started();
};

#endif // WIDGET_H

```

finished() Slot 함수는 QProcess 클래스를 이용해 외부 프로그램 실을 완료하였을 때 호출된다. readyReadStandardError() Slot 함수는 QProcess 클래스를 이용해 외부 프로그램 실행 후 에러가 발생하면 호출된다.

readyReadStandardOutput() Slot 함수는 실행 결과를 얻어오며 started() Slot 함수는 QProcess 에 의해 외부 프로그램이 실행되면 호출된다. 다음은 widget.cpp 의 소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->getInfoButton, &QPushButton::pressed,
            this,           &Widget::getInfoButton);

    m_process = new QProcess();
    connect(m_process, SIGNAL(finished(int, QProcess::ExitStatus)),
            this,           SLOT(finished(int, QProcess::ExitStatus)));
    connect(m_process, SIGNAL(readyReadStandardError()),
            this,           SLOT(readyReadStandardError()));
    connect(m_process, SIGNAL(readyReadStandardOutput()),
            this,           SLOT(readyReadStandardOutput()));
    connect(m_process, SIGNAL(started()), this, SLOT(started()));
}

Widget::~Widget()
{

```

```

    delete ui;
}

void Widget::getInfoButton()
{
    QString program = "/bin/ls";
    QStringList arguments;
    arguments << "-ltr" << "/usr";
    m_process->start(program, arguments);
}

void Widget::finished(int exitCode, QProcess::ExitStatus exitStatus)
{
    qDebug() << "Exit Code :" << exitCode;
    qDebug() << "Exit Status :" << exitStatus;
}

void Widget::readyReadStandardError()
{
    qDebug() << Q_FUNC_INFO << "ReadyError";
}

void Widget::readyReadStandardOutput()
{
    QByteArray buf = m_process->readAllStandardOutput();
    ui->textEdit->setText(buf);
}

void Widget::started()
{
    qDebug() << "Proc Started";
}

```

위의 예제에서 보는 것과 같이 클래스 생성자에서 QProcess 클래스의 오브젝트를 선언하고 QProcess 에서 제공하는 Signal 과 Slot 을 연결한다. GUI 상에서 [Get Info] 버튼을 클릭하면 getInfoButton() Slot 함수가 호출되며 QProcess 클래스의 start() 멤버 함수를 이용해 외부 프로그램을 실행한다. readyReadStandardOutput() 함수에서 QProcess 클래스의 readAllStandardOutput() 멤버 함수는 결과를 가져오고 가져온 결과를 buf 에 저장하고 QTextEdit 위젯에 출력한다.

16.3. QSerialPort 를 이용한 시리얼 통신

Qt에서 제공하는 QSerialPort 클래스는 시리얼(Serial)을 이용해 1:1 데이터 송/수신을 위한 기능을 제공한다. 현재 네트워크 속도에 비하면 느리다. 하지만 시리얼은 통신은 이기종 간 또는 물리적으로 떨어져 있는 임베디드 디바이스 칩 간의 통신으로 많이 사용된다.

QSerialPort 클래스의 가장 큰 장점은 리눅스 플랫폼에 시리얼 통신을 하든지 MS윈도우에서 시리얼 통신을 하든지 QSerialPort 클래스를 사용하면 운영체제 플랫폼에 상관 없이 QSerialPort 클래스에서 제공하는 기능을 이용해 시리얼 통신을 할 수 있다.

예를 들어 리눅스 플랫폼에 QSerialPort 클래스를 이용해 작성한 소스코드가 MS윈도우에서도 동일하게 사용할 수 있다. 리눅스에서는 시리얼 포트를 인식하는 디바이스 드라이버 명을 /dev/ttyUSB0 등과 같은 이름을 사용하며 MS 윈도우에서는 COM1 ~ COM4 등과 같이 디바이스 이름을 사용한다는 것 외에는 데이터 송수신을 위한 모든 기능은 표준을 따르기 때문에 운영체제 플랫폼과 상관없이 QSerialPort를 사용할 수 있다.

Qt 에서 제공하는 QSerialPort 클래스와 같은 시리얼을 이용하기 위해서 프로젝트파일에 다음과 같이 추가해야 한다.

```
QT += serialport
```

QSerialPort 에서도 Signal 과 Slot 을 사용한다. 따라서 QTcpSocket 또는 QUdpSocket 과 유사한 Signal 과 Slot 구조로 되어 있다. 다음은 QSerialPort 클래스를 이용해 디바이스를 연결하는 예이다.

```
QSerialPort *m_serial = new QSerialPort(this);

connect(m_serial, &QSerialPort::errorOccurred, this, &MainWindow::handleError);
connect(m_serial, &QSerialPort::readyRead, this, &MainWindow::readData);

QString name      = QString("COM1");
qint32 baudRate = QSerialPort::Baud115200;
QSerialPort::DataBits dataBits  = QSerialPort::Data8;
QSerialPort::Parity parity     = QSerialPort::NoParity;
QSerialPort::StopBits stopBits = QSerialPort::OneStop;

QSerialPort::FlowControl flowControl = QSerialPort::NoFlowControl;
```

```

m_serial->setPortName(name);
m_serial->setBaudRate(baudRate);
m_serial->setDataBits(dataBits);
m_serial->setParity(parity);
m_serial->setStopBits(stopBits);
m_serial->setFlowControl(flowControl);

m_serial->open(QIODevice::ReadWrite);

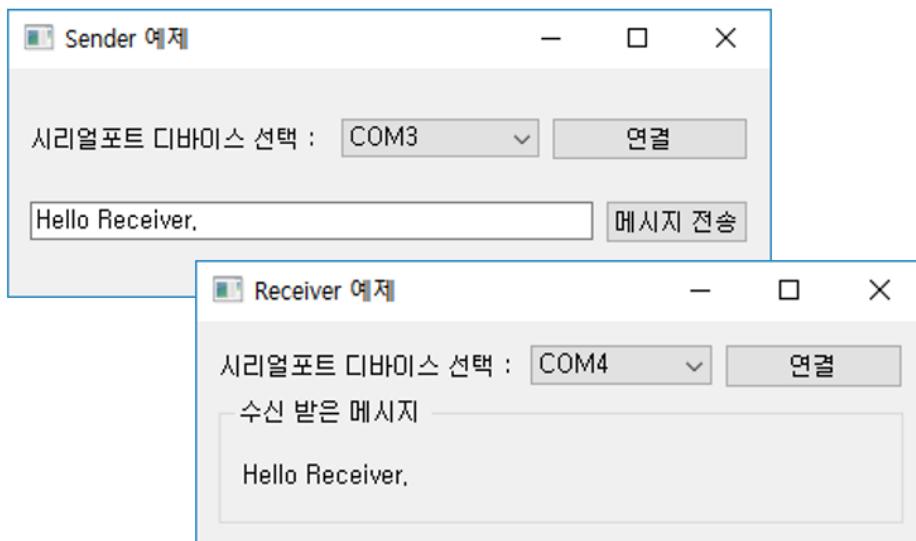
```

QSerialPort 클래스는 위의 예제 소스코드에서 보는 것과 같이 에러가 발생하면 errorOccurred() 시그널이 발생한다. 에러 발생시 Slot 함수를 호출 할 수 있다. 그리고 readyRead() 시그널을 이용해 상대방이 전송하는 데이터를 수신 받을 수 있다.

시리얼은 표준으로 상대방 측과 시리얼로 연결하기 위해서는 동일하게 설정 맞추어야 한다. setPortName() 멤버 함수는 사용할 시리얼 디바이스 이름을 입력한다. setBaudRate 는 속도를 지정할 수 있다. setDataBits() 함수는 character 당 사용할 데이터 비트 수, setParity() 는 사용할 parity 설정 등을 설정한다.

위와 같이 설정을 완료하고 시리얼 통신을 사용하기 위해서는 QSerialPort 클래스를 제공하는 open() 함수를 이용해 데이터 송/수신을 위한 디바이스를 OPEN한다. 다음은 예제를 통해 시리얼 통신을 구현해 보도록 하자.

- QSerialPort 클래스를 이용한 데이터 송수신 예제 구현



<그림> Sender 와 Receiver 예제 실행 화면

이번에 다룰 예제는 두개의 어플리케이션을 이용해 데이터를 송/수신 하는 예제이다. 첫 번째 예제 어플리케이션은 시리얼을 이용해 데이터를 송신한다. 두 번째 어플리케이션은 시리얼포트로 수신 받은 데이터를 GUI상에 출력하는 예제이다.

위의 그림에서 좌측은 Sender 예제이다. GUI상에서 사용할 디바이스를 선택하고 [연결] 버튼을 클릭한다. 그리고 우측의 Receiver 예제도 마찬가지로 GUI상에서 사용할 선택하고 [연결] 버튼을 클릭하면 두 어플리케이션 간 데이터를 송/수신 할 준비가 모두 완료된다.

첫 번째 Sender 어플리케이션의 하단의 QLineEdit 위젯에 전송할 메시지를 입력 후 [메시지 전송] 버튼을 클릭하면 위의 그림에서 보는 것과 같이 Receiver에게 메시지를 송신한다.

Receiver는 Sender가 보낸 메시지를 GUI 상에 출력한다. 두 개의 어플리케이션 중 Sender 예제의 소스코드를 먼저 살펴본 후 Receiver 예제를 살펴보도록 하자. 프로젝트 전체 소스코드는 Ch16 > 03_Sender 와 03_Receiver 디렉토리를 참조하면 된다. 다음 예제 소스코드는 Sender 예제의 widget.h 소스코드이다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QSerialPort>
#include <QSerialPortInfo>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();
private:
    Ui::Widget *ui;

    struct SerialSettings {
        QString portName;
        qint32 baudRate;
        QSerialPort::DataBits dataBits;
        QSerialPort::Parity parity;
    };
}
```

```

    QSerialPort::StopBits stopBits;
    QSerialPort::FlowControl flowControl;
};

SerialSettings m_serialSettings;
QSerialPort *m_serial = nullptr;

private slots:
    void connectButton();
    void sendButton();
};

#endif // WIDGET_H

```

SerialSettings 구조체에는 QSerialPort 클래스의 오브젝트를 설정할 값을 저장하기 위한 구조체이다. connectButton()은 Sender 어플리케이션의 GUI 상에서 [연결] 버튼을 클릭하면 호출되는 Slot 함수이다. sendButton()은 [메시지 전송] 버튼 클릭 시 호출되는 Slot 함수이다. 다음은 widget.cpp 예제 소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->connectionButton, SIGNAL(pressed()),
            this, SLOT(connectButton()));

    connect(ui->sendButton, SIGNAL(pressed()), this, SLOT(sendButton()));

    m_serialSettings.baudRate      = 115200;
    m_serialSettings.dataBits     = QSerialPort::Data8;
    m_serialSettings.parity       = QSerialPort::NoParity;
    m_serialSettings.stopBits     = QSerialPort::OneStop;
    m_serialSettings.flowControl  = QSerialPort::NoFlowControl;

    const auto infos = QSerialPortInfo::availablePorts();

    for (const QSerialPortInfo &info : infos)
        ui->comboBox->addItem(info.portName());
}

```

```

    m_serial = new QSerialPort(this);
}

void Widget::connectButton()
{
    if(m_serial->isOpen()) return;

    QString devName = ui->comboBox->currentText().trimmed();
    m_serialSettings.portName = devName;

    m_serial->setPortName(m_serialSettings.portName);
    m_serial->setBaudRate(m_serialSettings.baudRate);
    m_serial->setDataBits(m_serialSettings.dataBits);
    m_serial->setParity(m_serialSettings.parity);
    m_serial->setStopBits(m_serialSettings.stopBits);
    m_serial->setFlowControl(m_serialSettings.flowControl);

    if (!m_serial->open(QIODevice::ReadWrite)) {
        qDebug() << "에러메시지 :" << m_serial->errorString();
    }
}

void Widget::sendButton()
{
    if(!m_serial->isOpen()) return;

    QString sendMsg = ui->sendLineEdit->text().trimmed();
    QByteArray msg = sendMsg.toLocal8Bit();

    m_serial->write(msg);
}

Widget::~Widget()
{
    delete ui;
}

```

클래스 생성자에서 QSerialPortInfo::availablePorts() 멤버 함수는 현재 동작하는 플랫폼에서 활용 가능한 모든 시리얼 디바이스의 정보를 얻어오는 기능을 제공한다. 활용 가능한 시리얼 디바이스 정보를 얻어와 GUI 상에서 콤보박스에 등록한다.

connectButton() Slot 함수가 호출되면 클래스 생성자에서 저장한 시리얼 연결을 위한

옵션 값을 설정하고 시리얼포트 디바이스를 연결한다. sendButton() Slot 함수에서는 QLineEdit에서 입력한 문자 변수 값을 QByteArray로 변환 후 QSerialPort 클래스의 write() 멤버 함수를 이용해 메시지를 Receiver에게 전송한다. 다음 예제 소스코드는 Receiver 예제의 widget.h 소스코드이다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QSerialPort>
#include <QSerialPortInfo>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;

    struct SerialSettings {
        QString portName;
        qint32 baudRate;
        QSerialPort::DataBits dataBits;
        QSerialPort::Parity parity;
        QSerialPort::StopBits stopBits;
        QSerialPort::FlowControl flowControl;
    };
    SerialSettings m_serialSettings;
    QSerialPort *m_serial;

private slots:
    void connectButton();
    void readData();
    void error(QSerialPort::SerialPortError err);
};

#endif // WIDGET_H
```

위의 예제 소스코드는 Receiver 예제의 Widget 클래스의 헤더파일 소스코드이다. Sender 와 차이점으로 readData() 와 error() Slot 함수가 추가 되었다. readData() 함수는 시리얼포트로 수신 받을 때 호출된다.

그리고 error() Slot 함수는 시리얼포트에 에러가 발생하면 호출된다. 다음은 예제 소스코드는 widget.cpp 이다.

```
#include "widget.h"
#include "ui_widget.h"
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->connectionButton, SIGNAL(pressed()),
            this,                      SLOT(connectButton()));

    m_serialSettings.baudRate     = 115200;
    m_serialSettings.dataBits     = QSerialPort::Data8;
    m_serialSettings.parity       = QSerialPort::NoParity;
    m_serialSettings.stopBits     = QSerialPort::OneStop;
    m_serialSettings.flowControl = QSerialPort::NoFlowControl;

    const auto infos = QSerialPortInfo::availablePorts();

    for (const QSerialPortInfo &info : infos)
        ui->comboBox->addItem(info.portName());

    m_serial = new QSerialPort(this);
    connect(m_serial, SIGNAL(error(QSerialPort::SerialPortError)),
            this,      SLOT(error(QSerialPort::SerialPortError)));
    connect(m_serial, SIGNAL(readyRead()),
            this,      SLOT(readData()));
}

void Widget::connectButton()
{
    if(m_serial->isOpen())
        return;

    QString devName = ui->comboBox->currentText().trimmed();
    m_serialSettings.portName = devName;
```

```
m_serial->setPortName(m_serialSettings.portName);
m_serial->setBaudRate(m_serialSettings.baudRate);
m_serial->setDataBits(m_serialSettings.dataBits);
m_serial->setParity(m_serialSettings.parity);
m_serial->setStopBits(m_serialSettings.stopBits);
m_serial->setFlowControl(m_serialSettings.flowControl);

if (!m_serial->open(QIODevice::ReadWrite)) {
    qDebug() << "에러메시지 : " << m_serial->errorString();
}
}

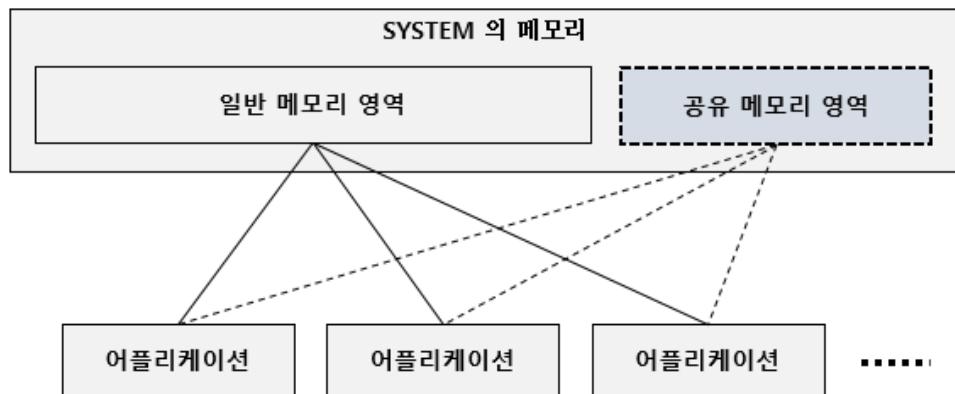
void Widget::error(QSerialPort::SerialPortError err)
{
    if (err == QSerialPort::ResourceError) {
        qDebug()<<"Critical Error: " << m_serial->errorString();
        m_serial->close();
    }
}

void Widget::readData()
{
    const QByteArray data = m_serial->readAll();
    ui->receiveMsgLabel->setText(data);
}

Widget::~Widget()
{
    delete ui;
}
```

16.4. Shared Memory 를 이용한 공유 메모리 사용

Shared Memory 를 이용한 공유 메모리를 사용하는 방식은 같은 시스템 내에 프로그램 간의 데이터를 교환하기 위한 목적으로 메모리 영역을 공유해 사용하는 방식을 말한다. Qt에서는 독립된 어플리케이션간 데이터 공유를 위해 QSharedMemory 클래스를 제공한다.



<그림> 독립된 어플리케이션이 공유 메모리 영역을 공유

위의 그림과 같이 독립된 어플리케이션에서 공유 메모리 영역을 READ/WRITE 하기 위해서는 특정 메모리 영역을 접근할 수 있는 매개체를 이용한다.

Qt에서는 고유한 KEY값을 이용해 공유 메모리 영역에 데이터를 READ하거나 WRITE 할 수 있다. 다음 예제 소스코드는 QSharedMemory 클래스를 이용해 KEY값을 설정하기 위한 방법이다.

```
QString key = QString("qt-dev.com");
QSharedMemory *m_sharedMemory = new QSharedMemory(key);
```

위의 예에서 보는 것과 같이 QSharedMemory 클래스 오브젝트 선언 시 인자로 공유한 KEY 값을 설정할 수 있다.

그리고 QSharedMemory 클래스가 제공하는 setKey() 멤버 함수에 인자로 KEY 값을 전달하면 고유한 KEY 값을 설정할 수 있다. 설정된 KEY 값을 참조하기 위해서는 key() 멤버 함수를 사용하면 된다.

QSharedMemory 클래스를 이용해 공유 메모리 영역에 데이터를 WRITE 하기 전에 위의 예에서 보는 것과 같이 KEY값을 항상 먼저 설정해야 한다. 그리고 데이터를 쓰기

위해서 다음의 예제에서 보는 것과 같이 사용할 수 있다.

```
QString key = QString("qt-dev.com");
QSharedMemory *m_sharedMemory = new QSharedMemory(key);

QBuffer buffer;
...
m_sharedMemory->lock();

char *to = (char*)m_sharedMemory->data();
const char *from = buffer.data().data();
memcpy(to, from, qMin(m_sharedMemory->size(), size));

m_sharedMemory->unlock();
```

위의 예제에서 보는 것과 같이 공유 메모리 영역에 데이터를 쓰기 전에 lock()을 이용해 외부에서 참조하기 못하도록 할 수 있다. 그리고 data() 함수를 이용해 공유메모리 영역 주소 번지를 얻어와 memcpy()를 이용해 데이터를 WRITE 할 수 있다.

위의 예제 소스코드에서 보는 것과 같이 공유 메모리 영역에 데이터를 읽어 오기 위해서는 constData() 멤버 함수를 이용해 공유 메모리 영역에서 데이터를 읽어 올 수 있다. 다음 예제는 공유 메모리 영역영역부터 WRITE 한 데이터를 READ 하기 위한 예제이다.

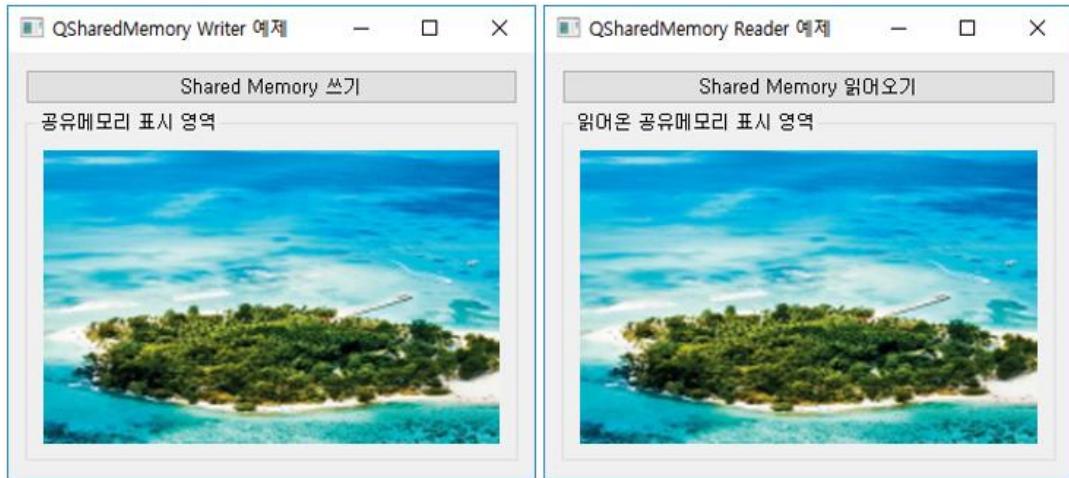
```
QBuffer buffer;
...
m_sharedMemory->lock();
buffer.setData((char*)m_sharedMemory->constData(),
m_sharedMemory->size());
...
m_sharedMemory->unlock();
m_sharedMemory->detach();
```

QSharedMemory 클래스는 위에서 보는 것과 같이 매우 사용하기 쉽다. 다음은 공유 메모리 영역에 데이터 WRITE/READ 하는 실제 예제 어플리케이션을 구현해 보도록 하자.

- QSharedMemory 클래스를 이용한 공유 메모리 WRITE/READ 구현

두 개의 독립된 어플리케이션을 구현할 것이다. 첫 번째 어플리케이션은 KEY 값을 "qt-dev.com"으로 설정하고 이미지파일을 읽어와 GUI상에 표시한다. 그리고 이미지파일의 바이너리데이터를 공유 메모리 영역에 WRITE 한다.

두 번째 어플리케이션에서는 첫 번째 어플리케이션에서 WRTIE 한 데이터를 READ 한다. READ 한 이미지 바이너리 데이터를 QImage로 저장한 다음 GUI에 이미지로 출력하는 예제이다.



<그림> 공유메모리 공유 Writer예제 와 Reader예제 실행 화면

위의 그림에서 보는 것과 같이 좌측의 어플리케이션은 공유 메모리 영역에 이미지의 데이터를 공유 메모리 영역에 WRITE 한다. 공유 데이터 영역에 데이터를 WRITE 하기 위해서 이미지 파일을 읽어온다. 즉 좌측의 GUI 상에서 [Shared Memory 쓰기] 버튼을 클릭하면 파일 다이얼로그가 생성되고 이미지 파일을 선택한다.

그리고 선택된 이미 파일의 바이너리 데이터를 공유 메모리 영역에 WRITE 하고 GUI에 이미지를 QLabel 위젯에 출력한다. 우측의 예제는 공유 메모리 영역에서 데이터를 읽어와 QImage 로 변환한 다음 GUI상에서 보는 것과 같이 QLabel 위젯에 출력한다.

위의 두개 예제의 전체소스코드는 Ch16 > 04_QSharedMemory_Write 디렉토리는 좌측의 어플리케이션은 공유 메모리 영역에 WRITE 하기 위한 전체 소스코드이며 04_QSharedMemory_Reader 디렉토리는 우측의 예제 어플리케이션 전체 소스코드이다.

위의 2개의 예제 어플리케이션 중 좌측의 WRITE하는 예제를 먼저 살펴보도록 하자. 다음 예제는 widget.h 소스코드이다.

```
#include <QWidget>
#include <QSharedMemory>

namespace Ui { class Widget; }

class Widget : public QWidget
{
```

```

_Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;
    QSharedMemory *m_sharedMemory;

private slots:
    void writeButton();
};

```

이 클래스의 생성자에서는 QSharedMemory 클래스 오브젝트를 선언하고 공유메모리에서 사용할 키 값이 정의 되어 있다. 키 값은 고유한 키 값을 사용하며 타입은 QString을 사용한다. 다음 예제는 widget.cpp 예제 소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"
#include <QFileDialog>
#include <QBuffer>
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->writeButton, &QPushButton::pressed,
            this,           &Widget::writeButton);

    QString key = QString("qt-dev.com");
    m_sharedMemory = new QSharedMemory(key);
}

void Widget::writeButton()
{
    if (m_sharedMemory->isAttached()) {
        if (!m_sharedMemory->detach())
            ui->label->setText("Shared memory detach 실패.");
        return;
    }

    QString fileName;

```

```

fileName = QFileDialog::getOpenFileName(0, QString(), QString(),
                                         tr("Images (*.png *.xpm *.jpg)"));

QImage image;
if (!image.load(fileName)) {
    ui->label->setText(tr("이미지 파일을 선택해 주세요."));
    return;
}
ui->label->setPixmap(QPixmap::fromImage(image));

QBuffer buffer;
buffer.open(QBuffer::ReadWrite);
QDataStream out(&buffer);
out << image;
int size = buffer.size();

if (!m_sharedMemory->create(size)) {
    ui->label->setText("공유 메모리 Segment 생성 실패");
    return;
}

m_sharedMemory->lock();

char *to = (char*)m_sharedMemory->data();
const char *from = buffer.data().data();
memcpy(to, from, qMin(m_sharedMemory->size(), size));

m_sharedMemory->unlock();
}

Widget::~Widget()
{
    delete ui;
}

```

writeButton() Slot 함수는 [Shared Memory 쓰기] 버튼 클릭 시 호출된다. 이 Slot 함수에서는 파일 중 하나를 선택할 수 있는 다이얼로그가 선택한다. 파일 중 이미지를 선택하면 선택한 이미지를 GUI에 출력한다. 그리고 이미지파일의 바이너리 데이터를 읽어와 공유 메모리 영역에 WRITE 한다.

다음은 우측의 예제 어플리케이션 소스코드이며 공유 메모리 영역에서 WRITE 한 데이터를 READ하기 위한 예제이다. 다음은 widget.h 소스코드이다.

```

#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QSharedMemory>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;
    QSharedMemory *m_sharedMemory;

private slots:
    void readButton();
};

#endif // WIDGET_H

```

readButton() Slot 함수는 [Shared Memory 읽어오기] 버튼을 클릭하면 호출된다. 이 Slot 함수에서는 공유 메모리 영역에서 데이터를 읽어와 QImage에 저장하고 GUI상에 이미지를 QLabel에 출력한다. 다음은 widget.cpp 소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"
#include <QBuffer>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->readButton, &QPushButton::pressed,
            this,           &Widget::readButton);

    QString key = QString("qt-dev.com");
    m_sharedMemory = new QSharedMemory(key);
}

```

```
void Widget::readButton()
{
    if (!m_sharedMemory->attach()) {
        ui->label->setText("공유메모리 영역으로부터 데이터 읽기 실패.");
        return;
    }

    QBuffer buffer;
    QDataStream in(&buffer);
    QImage image;

    m_sharedMemory->lock();
    buffer.setData((char*)m_sharedMemory->constData(), m_sharedMemory->size());
    buffer.open(QBuffer::ReadOnly);
    in >> image;
    m_sharedMemory->unlock();

    m_sharedMemory->detach();
    ui->label->setPixmap(QPixmap::fromImage(image));
}

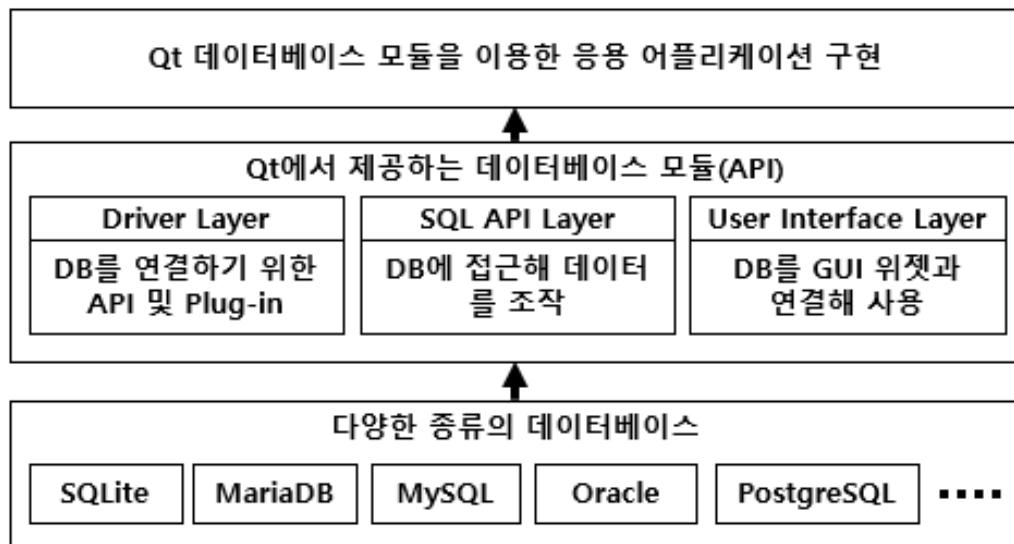
Widget::~Widget()
{
    delete ui;
}
```

17. 데이터베이스

Qt에서 제공하는 데이터베이스 모듈(API)은 다른 개발 프레임워크에 비해 사용하기 쉽다. Qt에서 제공하는 데이터베이스 모듈을 이용하면 통일된 API를 사용해 다양한 데이터베이스에 접근할 수 있다. 예를 들어 MariaDB 데이터베이스에 저장된 데이터를 SQL문을 이용해 QUERY하기 위해서 Qt에서 제공하는 QSqlQuery 클래스를 사용할 수 있다.

그리고 SQLite 데이터베이스에 저장된 데이터를 SQL문을 이용해 QUERY하기 위해서 MariaDB 데이터베이스에 접근했던 동일한 QSqlQuery 클래스를 이용할 수 있다.

즉, 각 데이터베이스에 접근해 데이터를 이용하기 위해서 제공하는 API를 사용하지 않고 Qt에서 제공하는 공통된 데이터베이스 모듈을 이용해 Qt 응용 어플리케이션을 개발할 수 있다.



<그림> Qt 데이터베이스 모듈 아키텍처 및 레이어

위의 그림에서 보는 것과 같이 SQLite를 사용하든지 MariaDB를 사용하든지 Qt에서 제공하는 데이터베이스 모듈을 사용하면 수 있다.

즉 Qt는 다양한 데이터베이스에 접근하기 위해서 공통된 데이터베이스 모듈(API)을 사용할 수 있다. Qt에서 제공하는 데이터베이스 모듈은 다음과 같이 3가지로 분류 할 수 있다.

① Driver Layer

첫 번째는 Driver Layer 이다. 이 Layer는 특정 데이터베이스를 연결하기 위한 Low Level Bridge 역할을 담당한다. 즉 데이터베이스와 연결을 위한 드라이버 역할을 담당한다.

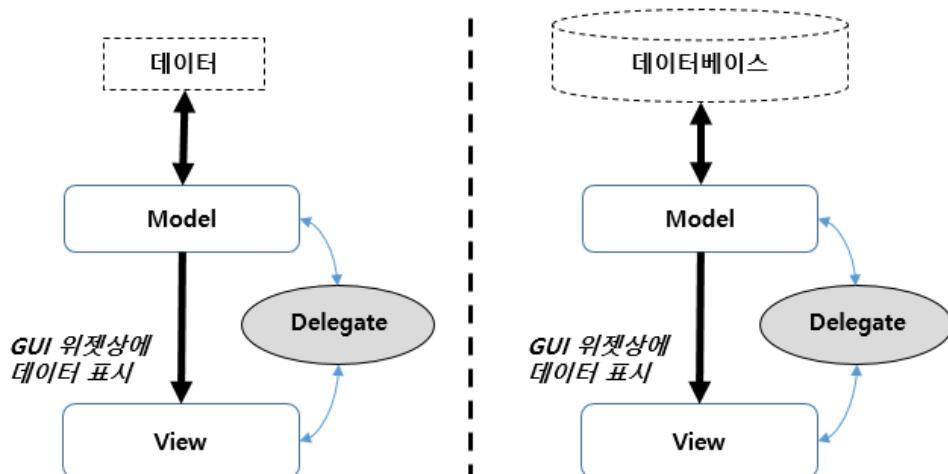
② SQL API Layer

데이터베이스에 연결하기 위한 클래스를 제공하며 데이터베이스 테이블에 데이터를 활용할 수 있는 데이터베이스를 제공한다. 예를 들어 QSqlDatabase 클래스를 이용해 데이터베이스에 연결하고 QUERY하기 위해서 QSqlQuery 클래스를 사용할 수 있다.

③ User Interface Layer

이 계층은 일전에 우리가 데이터를 다룰 때 Model/View를 다루었다. Model 은 데이터가 저장된 Container 와 같은 역할을 한다. View 는 QTableView 와 같은 GUI 위젯을 View 또는 View 위젯이라고 한다. View 위젯에 Model을 연결 했던 것과 같이 Qt에서는 데이터베이스 Model을 제공한다.

예를 들어 QSqlQueryModel은 데이터베이스에 저장된 테이블의 데이터를 QUERY 해 데이터를 Model 에 저장한다. 따라서 QSqlQueryModel 을 View 위젯과 연결할 수 있다.



<그림> 데이터베이스 Model / View

위의 그림에서 보는 것과 같이 좌측은 일반 Model/View 개념이고 우측은 데이터베이스 Model 클래스와 View 의 개념이다. 이와 같이 Qt에서 제공하는 데이터베이스 Model 클래스들을 이용해 View 위젯과 연결할 수 있다. Qt는 주로 사용하는 Model 클래스로 QSqlQueryModel, QSqlTableModel, QSqlRelationalTableModel 클래스를 제공한다. Qt에서 제공하는 데이터베이스 모듈을 사용하기 위해서는 프로젝트 파일에 다음과 같이 "sql" 키워드를 사용해야 한다.

```
Qt += sql
```

✓ 데이터베이스 연결

TCP에서 상대방과 데이터를 송/수신 하기 위해서 Connection을 맺어야 하는 것과 같이 Qt에서 데이터베이스를 이용하기 위해서 데이터베이스와 먼저 Connection을 맺어야 한다.

```
QSqlDatabase db1 = QSqlDatabase::addDatabase("QMYSQL");
QSqlDatabase db2 = QSqlDatabase::addDatabase("QSQLITE");
QSqlDatabase db3 = QSqlDatabase::addDatabase("QPSQL");
```

QSqlDatabase 클래스는 데이터베이스에 연결하기 위한 기능을 제공한다. 또한 고유한 사용자 계정을 사용해 연결 기능도 지원한다. 다음은 데이터베이스에 접근하기 위한 예제 소스코드이다.

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");

db.setHostName("bigblue");           // IP 또는 DNS Host name
db.setDatabaseName("flightdb");     // DB명
db.setUserName("acarlson");        // 계정 명
db.setPassword("1uTbSbAs");        // 계정 Password

bool ok = db.open();
```

위의 예제 소스코드에 보는 것과 같이 addDatabase() 함수의 첫 번째 인자로 사용하고자 하는 데이터베이스 드라이버명을 명시해야 한다. 다음 표는 Qt에서 제공하는 데이터베이스 드라이버 명이다.

<표> 드라이버 명

드라이버 명	설명
QDB2	IBM DB2 7.1 이상 버전

QIBASE	Borland InterBase
QMYSQL	MariaDB 또는 MySQL
QOCI	Oracle Call Interface Driver
QODBC	Open Database Connectivity
QPSQL	PostgreSQL 7.3 이상 버전
QSQLITE2	SQLite Version 2
QSQLITE	SQLite Version 3
QTDS	Sybase Adaptive Server

✓ SQL 문을 이용한 데이터베이스 QUERY

데이터베이스 테이블의 데이터를 QUERY하기 위해서 다음 예제에서 보는 것과 같이 QSqlQuery 클래스를 사용할 수 있다.

```
QSqlQuery query;

QString qry;
qry = "SELECT name, salary FROM employee WHERE salary > 500";
query.exec(qry);
```

그리고 테이블의 다음 데이터를 차례대로 검색하고 원하는 필드의 데이터를 얻기 위해서 다음과 같이 사용할 수 있다.

```
while (query.next())
{
    QString name = query.value(0).toString();
    int salary = query.value(1).toInt();
}
```

테이블에 데이터를 삽입하기 위해서는 위에서 다음과 같이 사용할 수 있다.

```
QSqlQuery query;
query.exec("INSERT INTO employee (id, name, salary) "
          "VALUES (1001, 'Thad Beaumont', 65000);")
```

만약 많은 양의 데이터를 SQL 문을 사용시 Placeholder 방식과 Positioning 방식을 사용할 수 있다. 성능 측면에서는 Placeholder 방식을 권장한다. 다음은 Placeholder 방식

의 예제 소스코드이다.

```
QSqlQuery query;
query.prepare("INSERT INTO employee (id, name, salary) "
             "VALUES (:id, :name, :salary)");
for(...)
{
    query.bindValue(":id", 1001);
    query.bindValue(":name", "Thad Beaumont");
    query.bindValue(":salary", 65000);

    query.exec();
}
```

다음은 Positioning 방식의 예이다.

```
QSqlQuery query;
query.prepare( "INSERT INTO employee(id, name, salary) VALUES (?, ?, ?)");

for(...) {
    query.addBindValue(1001);
    query.addBindValue("Thad Beaumont");
    query.addBindValue(65000);

    query.exec();
}
```

✓ Transaction 처리

Qt는 QSqlDatabase 클래스에서 제공하는 멤버 함수를 이용하면 Transaction 처리가 가능하다. 다음 예제는 transaction() 과 commit() 멤버 함수를 사용한 예제이다.

```
QSqlDatabase::database().transaction();
QSqlQuery query;
query.exec( "SELECT id FROM employee WHERE name = 'Torild Halvorsen'");

if (query.next()) {
    int employeeId = query.value(0).toInt();
    query.exec("INSERT INTO project (id, name, ownerid) "
              "VALUES (201, 'Manhattan Project', "
              + QString::number(employeeId) + ')');
}
```

```
QSqlDatabase::database().commit();
```

✓ 데이터베이스 Model 클래스

Qt에서 제공하는 데이터베이스 모델 클래스 중에서 가장 사용 빈도가 높은 클래스는 다음과 같은 클래스들을 제공한다.

<표> 주요 데이터베이스 모델 클래스

모델 명	설명
QSqlQueryModel	SQL 을 이용해 데이터를 READ 하기 위한 방식
QSqlTableModel	테이블의 데이터를 READ/WRITE 하기 위해 적합.
QSqlRelationalTableModel	Foreign Key를 이용하는 방식

다음 예제 소스코드는 QSqlQueryModel 클래스를 사용해 QUERY 한 데이터를 Model에 저장하는 예제 소스코드이다.

```
QSqlQueryModel model;
model.setQuery("SELECT * FROM employee");

for (int i = 0; i < model.rowCount(); ++i)
{
    int id = model.record(i).value("id").toInt();
    QString name = model.record(i).value("name").toString();
    qDebug() << id << name;
}
```

QSqlQueryModel 클래스의 setQuery() 멤버 함수를 이용해 SQL질의를 설정하고 record(int) 멤버 함수를 호출하면 데이터베이스에 저장된 테이블의 레코드를 읽어올 수 있다. 다음 예제는 QSqlTableModel 클래스를 사용한 예제 소스코드이다.

```
QSqlTableModel model;
model.setTable("employee");
...

for (int i = 0; i < model.rowCount(); ++i) {
    QSqlRecord record = model.record(i);
    double salary = record.value("salary").toInt();
    salary *= 1.1;
    record.setValue("salary", salary);
```

```

    model.setRecord(i, record);
}

model.submitAll();

```

테이블에 저장된 레코드를 record() 멤버 함수를 통해 데이터를 읽어올 수 있으며 setRecord() 멤버 함수를 사용해 각 행의 레코드를 수정 할 수 있다. 다음은 setData() 멤버 함수를 이용해 특정 row(행) 와 column(열) 을 다음과 같이 수정할 수 있다.

```

QSqlTableModel model;
model.setTable("employee");

model.setData(model.index(row, column), 75000);
model.submitAll();

```

그리고 QSqlTableModel 클래스의 removeRows() 멤버 함수를 이용해 데이터를 일괄 삭제할 수 있다.

```

model.removeRows(row, 5);
model.submitAll();

```

removeRows() 멤버 함수의 첫 번째 인자는 삭제할 시작 행(row)의 번호 이다. 두 번째 인자는 삭제할 레코드의 개수이다. 다음은 QSqlRelationalTableModel 클래스를 이용하는 방식에 대해서 알아보도록 하자. 이 클래스는 Foreign Key를 이용해 테이블을 검색 할 수 있는 Model 을 제공한다. 아래와 같이 2개의 테이블을 예로 들어 보자.

<표> MainTable 테이블

필드 명	타입
id	INTERGER
Date	DATE
Time	TIME
Hostname	INTERGER
IP	INTERGER

<표> DeviceTable 테이블

필드 명	타입
id	INTERGER

Hostname	VARCHAR(255)
IP	VARCHAR(16)

위의 표에서 보는 것과 같이 Main Table 과 Device Table 이 있다. Device Table 에 레코드를 먼저삽입한다. 그리고 그 후에 Main Table 의 데이터를 삽입 시 Device Table 에 있는 Hostname 과 IP 를 삽입하고자 한다면 다음과 같이 사용할 수 있다.

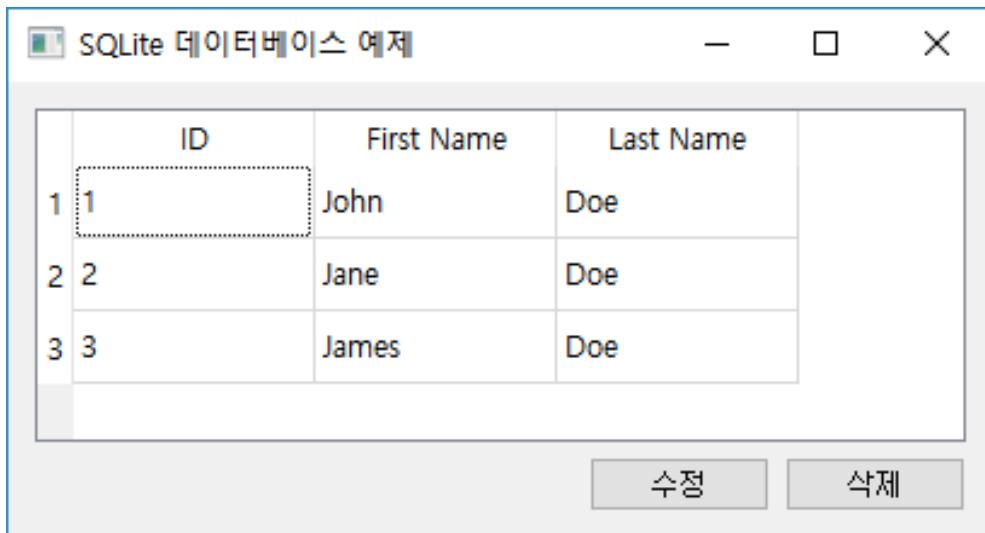
```
QSqlRelationalTableModel *modelMain;
QSqlRelationalTableModel *modelDevice;

modelMain->setRelation(3, QSqlRelation("DeviceTable", "id", "Hostname"));
modelMain->setRelation(4, QSqlRelation("DeviceTable", "id", "IP"));
```

위의 예제와 같이 QSqlRelationalTableModel 클래스에서 제공하는 setRelation() 멤버 함수를 사용해 “DeviceTable” 테이블의 Hostname 과 IP 레코드 필드 데이터를 maodelMain 오브젝트에 동일하게 삽입된다. 지금까지 우리는 Qt에서 제공하는 데이터베이스 모듈에 대해 살펴보았다. 다음은 예제를 직접 작성해 보도록 하자.

- SQLite 데이터베이스를 이용한 예제

이번 예제는 SQLite 데이터베이스를 이용한 예제를 작성해 보도록 하자. 다음은 예제를 실행한 화면이다.



<그림> 예제 실행 화면

위의 그림에서 보는 것과 같이 프로그램을 실행하면 SQLite 데이터베이스를 생성한다.

데이터베이스가 생성되면 name라는 테이블이 생성된다. 그리고 name 테이블에 데이터를 삽입하고 위의 그림에서 보는것과 같이 QTableView 위젯에 데이터를 출력하는 예제이다.

[수정] 버튼을 클릭하면 id 가 1번인 레코드의 First Name 을 'Eddy'로 변경한다. 그리고 Last Name 을 'Kim'으로 변경한다. [삭제] 버튼을 클릭하면 id 가 1번인 데이터를 삭제하는 예이다.

QWidget 기반의 프로젝트를 생성하고 다음과 같이 widget.h 헤더 소스코드를 작성한다. 전체 소스코드는 Ch17 > 01_SQLite_Example 디렉토리를 참조하면 된다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QDebug>
#include <QSqlDatabase>
#include <QSqlError>
#include <QSqlRecord>
#include <QSqlQuery>
#include <QSqlTableModel>
#include <QTableView>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
private:
    Ui::Widget *ui;
    QSqlDatabase db;
    QSqlTableModel *model;

    void initializeDataBase();
    void creationTable();
    void insertDataToTable();
    void initializeModel();

private slots:
```

```

    void slotPbtUpdate();
    void slotPbtDelete();
};

#endif // WIDGET_H

```

initializeDataBase() 함수는 SQLite 데이터베이스 파일을 생성한다. creationTable() 함수는 생성한 데이터베이스 내에 테이블을 생성한다. insertDataToTable() 은 생성한 테이블에 데이터를 삽입한다.

initializeModel() 함수는 QSqlTableModel 클래스 오브젝트를 생성하고 names 테이블을 model 데이터로 설정한다. 그리고 헤더를 설정한다. 다음 예제는 widget.cpp 소스 코드 이다.

```

#include "widget.h"
#include "ui_widget.h"
#include <QTableView>
#include <QFile>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);

    initializeDataBase();
    creationTable();
    insertDataToTable();
    initializeModel();

    ui->tableView->setModel(model);
    connect(ui->pbtUpdate, SIGNAL(pressed()), this, SLOT(slotPbtUpdate()));
    connect(ui->pbtDelete, SIGNAL(pressed()), this, SLOT(slotPbtDelete()));
}

Widget::~Widget()
{
    delete ui;
}

void Widget::initializeDataBase()
{
    QFile::remove("./testdatabase.db");
    db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("./testdatabase.db");
}

```

```

if( !db.open() )
    qDebug() << db.lastError();
}

void Widget::creationTable()
{
    QSqlQuery qry;
    qry.prepare("CREATE TABLE IF NOT EXISTS names "
               "("
               "    id INTEGER UNIQUE PRIMARY KEY, "
               "    firstname VARCHAR(30), "
               "    lastname  VARCHAR(30)"
               ")");

    if( !qry.exec() )
        qDebug() << qry.lastError();
}

void Widget::insertDataToTable()
{
    QSqlQuery qry;
    qry.prepare("INSERT INTO names (id, firstname, lastname) "
               "VALUES (1, 'John', 'Doe')");

    if( !qry.exec() )
        qDebug() << qry.lastError();

    qry.prepare("INSERT INTO names (id, firstname, lastname) "
               "VALUES (2, 'Jane', 'Doe')");
    if( !qry.exec() )
        qDebug() << qry.lastError();

    qry.prepare("INSERT INTO names (id, firstname, lastname) "
               "VALUES (3, 'James', 'Doe')");
    if( !qry.exec() )
        qDebug() << qry.lastError();
}

void Widget::initializeModel()
{
    model = new QSqlTableModel(this, db);
}

```

```

model->setTable("names");
model->setEditStrategy(QSqlTableModel::OnManualSubmit);
model->select();

model->setHeaderData(0, Qt::Horizontal, tr("ID"));
model->setHeaderData(1, Qt::Horizontal, "First Name");
model->setHeaderData(2, Qt::Horizontal, "Last Name");
}

void Widget::slotPbtUpdate()
{
    QSqlQuery qry;
    qry.prepare("UPDATE names SET firstname = 'Eddy' , "
               "lastname = 'Kim' WHERE id = 1" );

    if( !qry.exec() )
        qDebug() << qry.lastError();

    model->setTable("names");
    model->select();
    ui->tableView->setModel(model);
}

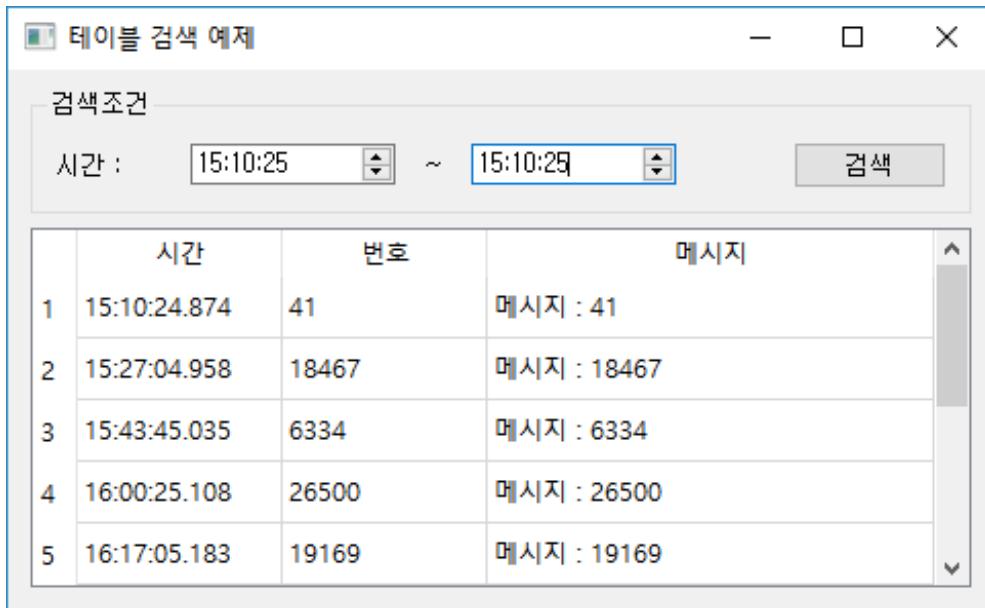
void Widget::slotPbtDelete()
{
    QSqlQuery qry;
    qry.prepare( "DELETE FROM names WHERE id = 1" );
    if( !qry.exec() )
        qDebug() << qry.lastError();

    model->setTable("names");
    model->select();
    ui->tableView->setModel(model);
}

```

- 데이터베이스 테이블 검색 예제

이번 예제에서는 데이터베이스 테이블에 저장된 데이터를 QSqlTableModel로 가져와 QTableView 위젯에 출력한다. 그리고 시간을 기준으로 검색하는 기능을 구현해 보도록 하자.



<그림> 예제 실행 화면

위의 그림에서 보는 것과 같이 시간을 기준으로 [검색] 버튼을 클릭하면 아래 데이터 중에서 시간 조건과 일치하는 데이터만을 화면에 표시한다. 데이터를 검색하기 위해서 QSqlTableModel 클래스에서 제공하는 setFilter() 멤버 함수를 사용하면 된다.

프로젝트 생성 시 QWidget 클래스를 상속받는 Widget 클래스를 상속받는다. 그리고 추가로 DatabaseHandler 클래스를 생성한다. 다음 예제 소스코드는 DatabaseHandler 클래스의 헤더 소스코드이다. 전체 소스코드는 Ch17 > 02_Search_Example 디렉토리를 참조하면 된다.

```
#include <QObject>
#include <QSql>
#include <QSqlQuery>
#include <QSqlError>
#include <QSqlDatabase>
#include <QFile>
#include <QDate>
#include <QDebug>
```

```

#define DATABASE_HOSTNAME    "QtDevDataBase"
#define DATABASE_NAME        "qtdev.db"

class DatabaseHandler : public QObject
{
    Q_OBJECT
public:
    explicit DatabaseHandler(QObject *parent = nullptr);
    ~DatabaseHandler();
    void connectToDataBase();
    bool insertIntoTable(const QVariantList &data);

private:
    QSqlDatabase db;

private:
    bool openDataBase();
    bool restoreDataBase();
    void closeDataBase();
    bool createTable();
};

```

openDataBase() 함수는 데이터베이스 파일을 생성한다. 그리고 QSqlDatabase 클래스의 오브젝트를 선언한다. restoreDataBase() 함수는 openDataBase() 함수를 호출한다.

그리고 이 함수에서 true를 리턴 하면 createTable() 함수를 호출한다. 이 함수에서는 RECEIVE_MAIL 이름으로 Define된 테이블을 생성한다. 다음 예제는 DatabaseHandler 클래스의 함수 구현 부 소스코드이다.

```

#include "databasehandler.h"
#include <QDir>
#include <QDebug>

DatabaseHandler::DatabaseHandler(QObject *parent) : QObject(parent)
{
}

void DatabaseHandler::connectToDataBase()
{
    QString dirPath = QDir::currentPath();
    dirPath.append( "/" DATABASE_NAME );

```

```

QFile(dirPath).remove(dirPath);
this->restoreDataBase();
}

bool DatabaseHandler::restoreDataBase()
{
    if(this->openDataBase()){
        if(!this->createTable()){
            return false;
        } else {
            return true;
        }
    } else {
        qDebug() << "데이터베이스 연결 실패.";
        return false;
    }
}

bool DatabaseHandler::openDataBase()
{
    QString dirPath = QDir::currentPath();
    dirPath.append="/" DATABASE_NAME);

    db = QSqlDatabase::addDatabase("QSQLITE");
    db.setHostName(DATABASE_HOSTNAME);
    db.setDatabaseName(dirPath);

    if(db.open()){
        return true;
    } else {
        return false;
    }
}

void DatabaseHandler::closeDataBase()
{
    db.close();
}

bool DatabaseHandler::createTable()
{
    QSqlQuery query;

```

```

if(!query.exec( "CREATE TABLE RECEIVE_MAIL ("
    "id      INTEGER PRIMARY KEY AUTOINCREMENT, "
    "TIME    TIME        NOT NULL,"
    "MESSAGE INTEGER     NOT NULL,"
    "RANDOM  VARCHAR(255) NOT NULL"
    " )"
    )){

qDebug() << "테이블 생성 에러 : " << query.lastError().text();
return false;
} else {
    return true;
}
}

bool DatabaseHandler::insertIntoTable(const QVariantList &data)
{
    QSqlQuery query;
    query.prepare("INSERT INTO RECEIVE_MAIL (TIME, RANDOM, MESSAGE) "
                  "VALUES (:TIME, :RANDOM, :MESSAGE)");

    query.bindValue(":TIME",   data[0].toTime());
    query.bindValue(":MESSAGE", data[1].toInt());
    query.bindValue(":RANDOM",  data[2].toString());

    if(!query.exec()){
        qDebug() << "데이터 삽입 에러 - " << query.lastError().text();
        return false;
    } else {
        return true;
    }
}

DatabaseHandler::~DatabaseHandler()
{
}

```

위의 함수 중 insertIntoTable() 함수는 생성한 테이블에 레코드를 삽입한다. 테이블에 레코드 삽입 시 Placeholder 방식을 사용해 레코드를 삽입한다. 다음 예제 소스코드는 widget.h 헤더 소스코드이다.

```
#include <QWidget>
#include <QSqlTableModel>
```

```

#include "databasehandler.h"

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;
    DatabaseHandler *dbHandler;
    QSqlTableModel *model;

    void setupModel(const QString &tableName, const QStringList &headers);
    void createUserInterface();

private slots:
    void onPushButton();
};


```

위의 함수 중 `onPushButton()` 은 [검색] 버튼을 클릭하면 호출되는 Slot 함수이다. `setupModel()` 함수는 `QSqlTableModel` 클래스 오브젝트를 선언하고 헤더를 선언한다. 그리고 오름차순으로 정렬 시킨다.

`createUserInterface()` 함수에서는 GUI 상에 배치한 `QTableView` 위젯을 설정한다. 그리고 GUI상에서 보는 것과 같이 시간을 검색하기 위해 `QTimeEdit`의 시간을 현재 시간으로 설정한다. 다음 예제는 `widget.cpp` 의 소스코드 이다.

```

#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    ui->timeFROM->setDisplayFormat("hh:mm:ss");
    ui->timeTO->setDisplayFormat("hh:mm:ss");

    dbHandler = new DatabaseHandler();
    dbHandler->connectToDataBase();
}


```

```

        for(int i = 0; i < 10; i++) {
            QVariantList data;
            QTime currTime = QTime::currentTime();
            currTime = currTime.addSecs(i*1000);

            int random = qrand();
            data.append(currTime);
            data.append(random);
            data.append("메시지 :" + QString::number(random));
            dbHandler->insertIntoTable(data);
        }

        this->setupModel("RECEIVE_MAIL",
                           QStringList() << "ID" << "시간" << "번호" << "메시지");

        this->createUserInterface();
    }

void Widget::setupModel(const QString &tableName, const QStringList &headers)
{
    model = new QSqlTableModel(this);
    model->setTable(tableName);
    for(int i = 0, j = 0; i < model->columnCount(); i++, j++)
    {
        model->setHeaderData(i, Qt::Horizontal, headers[j]);
    }

    model->setSort(0,Qt::AscendingOrder);
}

void Widget::createUserInterface()
{
    ui->tableView->setModel(model);
    ui->tableView->setColumnHidden(0, true);
    ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
    ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection);
    ui->tableView->resizeColumnsToContents();
    ui->tableView->horizontalHeader()->setStretchLastSection(true);

    for(int i = 0 ; i < 4 ; i++)
        ui->tableView->setColumnWidth(i, 100);
}

```

```
model->select(); // 테이블로부터 데이터를 Fetch
ui->timeFROM->setTime(QTime::currentTime());
ui->timeTO->setTime(QTime::currentTime());

connect(ui->pbtSearch, SIGNAL(pressed()), this, SLOT(onPushButton()));
}

void Widget::onPushButton()
{
    model->setFilter(QString( "TIME between '%1' and '%2'"))
        .arg(
            ui->timeFROM->time().toString("hh:mm:ss"),
            ui->timeTO->time().toString("hh:mm:ss"))
        );
    model->select(); // 테이블로부터 데이터를 Fetch
}

Widget::~Widget()
{
    delete ui;
}
```

18. 멀티미디어

Qt는 오디오, 비디오, 카메라를 이용한 응용 어플리케이션 개발을 위한 멀티미디어 모듈을 제공한다. 간단한 오디오 재생 기능 외에도 Bit rate와 Sample rate 를 추출해 특정 시간 구간의 음원을 추출해 분석하거나 네트워크로 전송하는 기능 등 멀티미디어 응용 어플리케이션 구현에 필요한 다양한 API를 제공한다. Qt에서 멀티미디어 모듈을 사용하기 위해서는 프로젝트파일에 다음과 같이 추가해야 한다.

```
QT += multimedia multimediawidgets
```

Qt에서 제공하는 멀티미디어 모듈은 이벤트를 처리하기 위해서 Signal 과 Slot 을 사용한다. 이는 멀티미디어를 다루는데 있어서 개발자에게 매우 복잡한 기능을 쉽게 구현할 수 있도록 해준다.

예를 들어 60초 음원이 있다면, 60 초 분량의 음원 데이터를 100 Millisecond 초 단위로 쪼개어 재생하게 된다. 내부적으로 데이터를 쪼개서 재생하지 않고 재생하는 API를 사용한다면 쪼개어 재생하는 구현을 신경 쓰지 않아도 되지만 만약 미디어 음원이 재생되지 않는 경우를 가정해 보자.

100 Millisecond 로 쪼개어 지진 않지만 100 Millisecond 로 쪼개어 재생하는 것을 구현한다고 가정해 보자 100 Millisecond 가 끝날 때마다 다음 음원 데이터를 재생해야 한다. 이때에 100 Millisecond 가 끝났다는 시그널을 발생해 준다면 쉽게 다음 음원을 데이터를 가져와 재생하는 기능을 Slot 함수로 구현하면 매우 복잡한 구현을 쉽게 구현할 수 있을 것이다.

또 다른 예로 인터넷 라디오 방송국을 예로 들어보자. 순간마다 음원을 재생해야 하는 인터넷 라디오 방송과 같이 마이크로부터 입력 받은 음원을 네트워크로 전송하고 수신 받은 측 어플리케이션에서 재생해야 한다면 단순히 MP3를 재생하는 것으로 구현할 수 없다. 방금 설명한 것과 같이 쪼개어 네트워크로 전송된 실시간 음원을 재생해야 한다. 이는 단순히 MP3를 재생하는 간단한 API를 사용하지 않고 좀더 Low Level 오디오 API 를 이용해 음원을 제어해야 하는데 Signal 과 Slot 을 사용하면 복잡한 구현을 쉽게 구현할 수 있는 장점이 있다.

또한 Qt는 멀티 플랫폼을 지원한다. 따라서 리눅스에서 구현한 멀티미디어 응용 어플리케이션을 MS윈도우에서 사용할 수 있는 장점이 있다. 지금까지 간략히 Qt 멀티미디

어 모듈의 특징 및 장점을 살펴보았으며 이번 장에서 다룰 주용 내용은 다음과 같다.

① 오디오

MP3와 같이 인코딩(압축된 상태)된 음원을 디코딩(복원 또는 압축해제 후 재생) 하는 기능 구현에 대해 살펴본다. 그리고 마이크로부터 입력 받은 데이터를 다루는 방법과 더불어 Low Level 음원을 다루는 방법에 대해서 살펴볼 것이다.

② 비디오

동영상 파일을 재생하는 방법에 대해서 살펴보도록 하자. 그리고 Qt에서 제공하는 비디오 관련 API외에 동영상 디코딩을 지원하는 오픈소스 API에 대해서도 살펴보도록 하자.

③ 카메라

시스템에서 사용 가능한 카메라 디바이스를 검색하고 카메라로부터 받은 영상을 GUI에 출력하는 방법에 대해서 살펴보도록 하자.

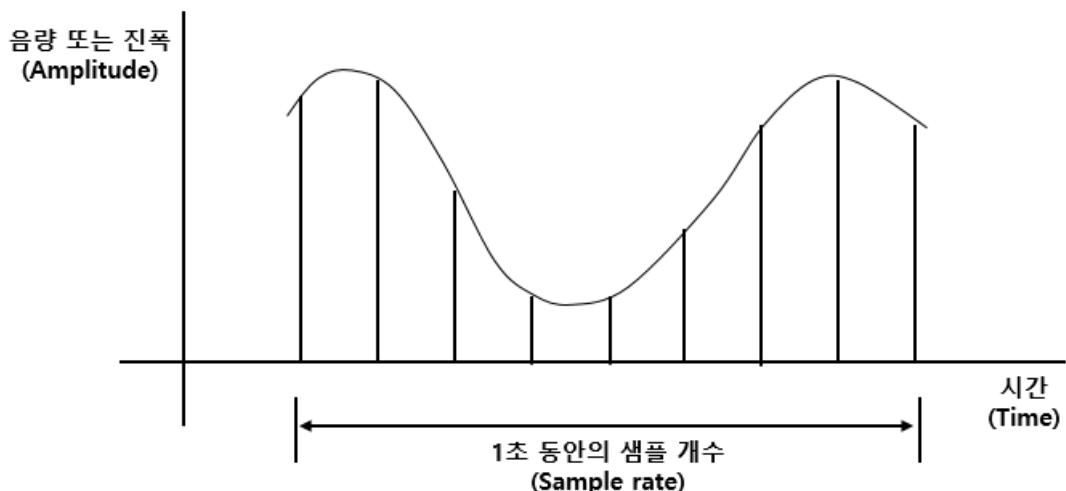
18.1. 오디오

Qt에서 제공하는 오디오 기능 구현을 살펴보기 전에 오디오를 다룰 때 필요한 필수적인 기초 개념에 대해서 먼저 살펴보도록 하자.

사람이 말할 때 음파가 아날로그 파형 형태로 다른 사람의 귀에 전달되지만 컴퓨터에서 음원을 디지털 기기에서 재생하기 위해 음원과 같은 아날로그 파형 신호를 2진수 형태로 변환 해야 한다. 2진수로 변환한 음원을 제어하기 위해서 Sample rate와 Bit rate의 개념을 먼저 이해해야 한다.

✓ Sample rate

Sample rate는 아날로그 신호와 같이 연속된 파형을 2진수 디지털로 변환해 추출한 특정 값을 의미한다. 즉 Sample rate는 1초당 추출한 샘플의 개수 또는 샘플의 빈도 수를 의미한다. 오디오에서 사용하는 용어로 52.3 KHz (52,300 Hz) 또는 22.4 KHz (22,400 Hz) 단위로 표현한다. 이는 1초당 반복되는 샘플의 개수를 의미한다.

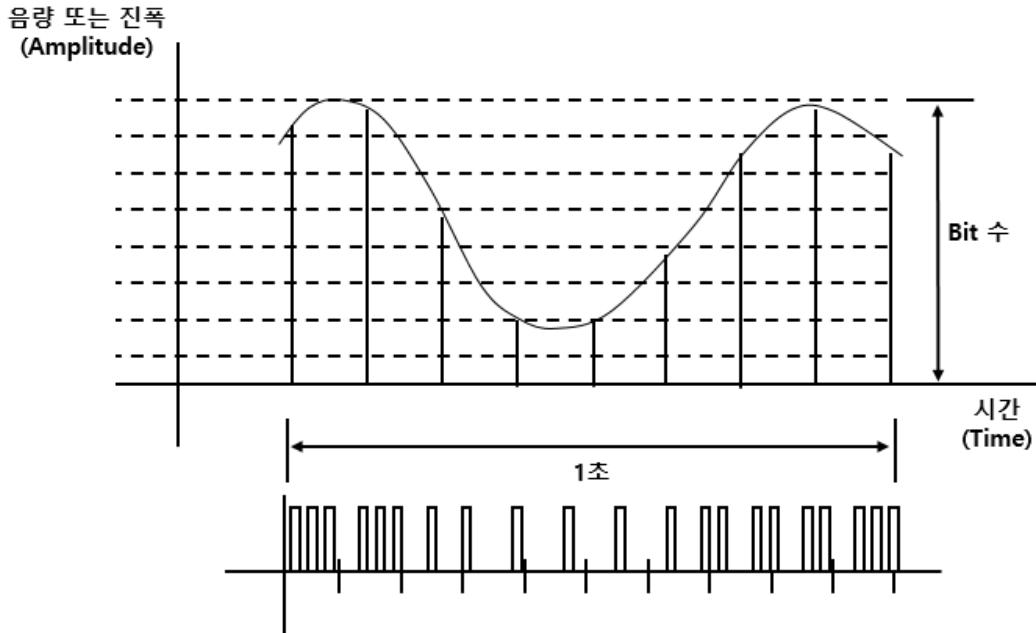


<그림> Sample rate

예를 들어 52.1 KHz 는 1초 동안에 52,100개의 샘플을 추출한다는 뜻이다. 샘플 개수가 더 커질 수록 더욱 세밀해진다는 것을 알 수 있다. 우리가 CD음질로 사용하는 음질은 44.1Khz를 사용한다.

✓ Bit rate

Bit rate 는 초당 Bit 수를 의미한다. 즉 1초동안 사용한 Bit 수를 의미하며 bps 단위를 사용한다. 우리가 음원의 품질을 말할 때 96 Kbps, 128 Kbps, 192 Kbps 등과 같이 bps 를 말한다.



<그림> Bit rate

우리가 사용하는 핸드폰과 같은 전화 상의 음질은 8 Kbps, AM 라디오는 32Kbps, MP3 는 56Kbps, FM라디오는 96 Kbps, CD 음질은 192 Kbps, FLAC 와 같은 무 손실 음원은 500 Kbps ~ 1 Mbps 를 사용한다. 즉 bps 가 높을수록 음질은 좋아진다.

위의 그림에서 보는 것과 같이 아날로그를 디지털로 변환 신호를 1과 0로 표현하는데 이 두 가지 상태만 표시가 가능하다. 그리고 아날로그를 1과 0으로 디지털화 하는 것을 PCM(Pulse Code Modulation) 이라 한다.

지금까지 살펴본 Sample rate 와 Bit Rate 를 구하면 MP3 파일을 디코딩 후 전체 음원의 Bytes 를 계산할 수 있다. 예를 들어 sample.mp3 파일은 재생 시간이 299초(4분 59 초)이고 Bit Rate 는 56 Kbps, Sample rate 수가 22,050 Hz 라면 재생 시간 * Bit rate / 8 를 계산한 값이 sample.mp3 파일의 Bytes 수이다.

그리고 PCM 을 다음과 같이 구할 수 있다. PCM 파일 크기는 PCM 초당 데이터 처리량 * 재생 시간 / 8을 하면 PCM 파일 크기를 계산할 수 있다.

파일명: sample.mp3

재생시간: 299.277초 (4분 59초)

Bit rate: 56Kbps (CBR)

Channel: 2 ch

비트: 16 bit

Sample rate: 22050 Hz

크기: 2,094,939 byte

파일명: stop.pcm

크기: 26,394,624 byte

MP3 파일의 크기 계산

파일크기 = 재생시간 * Bit rate / 8

$$299.277 * 56,000(\text{Hz}) / 8 = 2,093,000 \text{ Bytes}$$

실제 파일의 크기와 계산한 결과의 차이가 있는 것은 1초 이하의 시간은 고려하지 않았기 때문에 차이가 있음.

PCM 초당 데이터량 = Sample rate * Channel * Bit
22,050(Hz) * 2 * 16 (bit) = 705,600 Bit (88,200 Kbytes)

초당 데이터량은 705 Kbps 가 됨. 예를 들어 보통 MP3의 오디오 품질이 56Kbps 이므로 Kbps에 비해 약 12배가 차이가 남

PCM 파일 크기 = PCM 초당 데이터 처리량 * 재생시간 / 8
705,600 * 299.277 / 8 = 26,394,624 Bytes

실제 파일의 크기와 계산한 결과의 차이가 있는 것은 1초 이하의 시간은 고려하지 않았기 때문에 약간 상이

<그림> MP3 파일 크기와 PCM 파일 크기 계산

오디오에서 채널이란 한 방향으로 소리를 전달하는 것을 Mono라고 하며 두 개의 채널, 즉 2두 개의 방향으로 소리를 전달하는 것을 스테레오 채널이라고 한다. 단순히 2개의 스피커에서 똑같은 소리가 나오는 것을 스테레오 채널이라고 하지 않는다.

예를 들어 녹음 과정에서 몇 개의 방향으로 소리를 나누어 구분하는 것인지를 의미한다. 여기에서 MP3 파일 계산 시 참조한 Channel은 방금 말한 채널을 의미한다. 그리고 여기서 비트(Bit)는 샘플 당 비트 수를 의미한다. 비트 수가 클수록 더 작게 아날로그 파형을 쪼갤 수 있다.

지금까지 우리가 오디오를 구현하기 이전에 필요한 기초적인 개념을 알아보았다. 지금 까지 살펴본 Sample rate, Bit rate, 비트 수, Channel 은 실제 오디오 포맷을 설정하는데 사용된다.

이런 개념을 정확히 알지 못한다고 해도 MP3 파일을 재생하는 기능을 구현하는 데 문제가 되지 않는다. 단순히 MP3를 재생하기 위해서 QMediaPlayer 클래스를 이용해 쉽게 재생하는 기능을 구현할 수 있다.

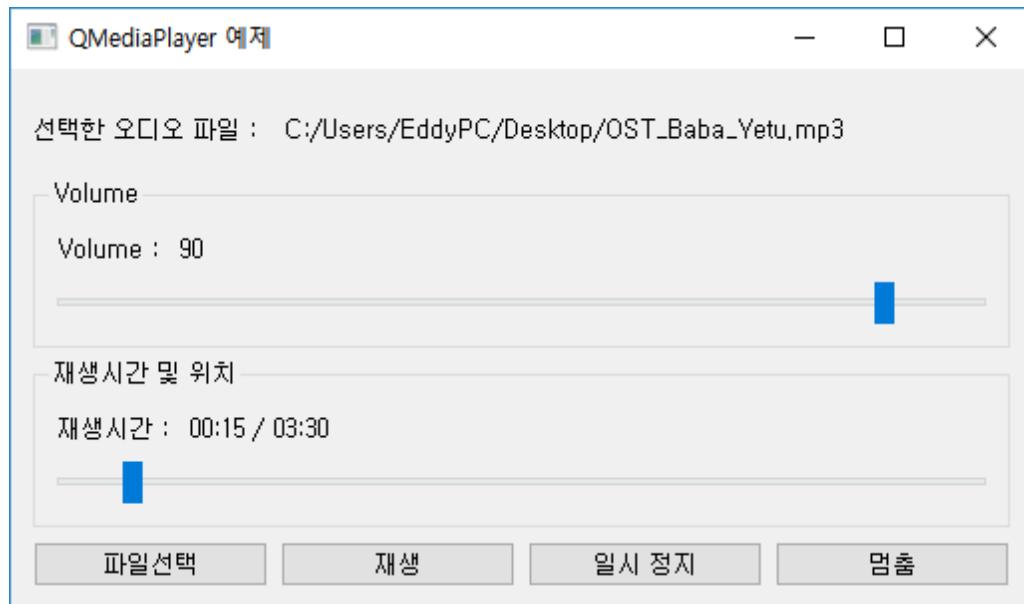
```
player = new QMediaPlayer;  
...  
player->setMedia(QUrl::fromLocalFile("/My/Music/song.mp3"));  
player->setVolume(50);  
player->play();
```

QMediaPlayer 클래스는 시스템의 파일시스템에 저장된 로컬 파일을 재생할 수 있는 기능 외에도 URL을 통해 외부에 위치한 파일을 HTTP 프로토콜을 이용해 재생할 수 있는 기능을 제공한다.

```
player = new QMediaPlayer;  
playlist = new QMediaPlaylist(player);  
  
playlist->addMedia(QUrl("http://example.com/myfile1.mp3"));  
playlist->addMedia(QUrl("http://example.com/myfile2.mp3"));  
...  
playlist->setcurrentIndex(1);  
player->play();
```

MP3 파일을 재생하는 방법에 대해 간단히 살펴 보았다. 다음으로 실제 예제를 통해 Qt에서 제공하는 오디오 기능에 대해 자세히 알아보도록 하자.

- QMediaPlayer 클래스를 이용한 오디오 재생 예제



<그림> 예제 실행 화면

위의 그림은 이번 예제의 실행 화면이다. 이번 예제는 QMediaPlayer 클래스를 이용해 오디오 MP3 파일을 재생하는 기능을 구현해 보자. 위의 예제 실행 화면에서 보는 것과 같이 [파일 선택] 버튼을 클릭하면 파일 다이얼로그에서 MP3 파일을 선택한다. [재생] 버튼은 선택한 음악 파일을 재생한다.

GUI상에서 첫 번째 QSlider 는 Volume 을 조절한다. 두 번째 QSlider 는 현재 재생되고 있는 위치가 표시되며 사용자가 QSlider를 이용해 재생 위치를 변경할 수 있다. 다음의 예제 소스코드는 widget.h 헤더 소스파일이다. 이 프로젝트의 전체 소스는 Ch18 > 01_QMediaPlayer_Example 디렉토리를 참조하면 된다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QMediaPlayer>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;
    QString          m_fName;
    QMediaPlayer    *m_player;
    qint64           m_duration;

private slots:
    void onOpenBtn();
    void onPlayBtn();
    void onPauseBtn();
    void onStopBtn();

    void sliderValueChange(int val);
    void durationChanged(qint64 duration);
    void positionChanged(qint64 progress);
```

```

    void seek(int seconds);
};

#endif // WIDGET_H

```

`m_fName` 변수에는 파일 다이얼로그에서 선택한 파일의 경로 및 파일명을 저장한다. `m_duration` 은 음악파일의 총 재생 시간을 초로 저장한다. `sliderValueChange()` Slot 함수는 현재 재생하는 `QMediaPlayer` 의 볼륨을 조절하는 기능을 제공한다.

`durationChanged()` 함수는 [파일 선택] 버튼을 클릭하고 재생할 음악 파일의 총 재생 시간이 변경되면 호출 되는 Slot 함수이다. `positionChanged()` 함수는 재생 하는 동안에 현재 재생 위치가 변경되면 호출 되는 Slot 함수이다.

`seek()` Slot 함수는 GUI상에서 현재 위치를 가리키는 `QSlider` 위젯을 사용자가 Drag 하여 위치가 변경되면 재생되는 위치도 변경되어 다시 재생된다. 다음 예제는 `widget.cpp`의 소스코드 이다.

```

#include "widget.h"
#include "ui_widget.h"
#include <QFileDialog>
#include <QTime>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    m_duration = 0;
    ui->setupUi(this);
    connect(ui->volSlider, SIGNAL(valueChanged(int)),
            this,           SLOT(sliderValueChange(int)));

    ui->sliderPosition->setEnabled(false);
    connect(ui->pbtOpen,  SIGNAL(clicked()), this, SLOT(onOpenBtn()));
    connect(ui->pbtPlay,  SIGNAL(clicked()), this, SLOT(onPlayBtn()));
    connect(ui->pbtPause, SIGNAL(clicked()), this, SLOT(onPauseBtn()));
    connect(ui->pbtStop,  SIGNAL(clicked()), this, SLOT(onStopBtn()));

    m_player = new QMediaPlayer();
    connect(m_player, &QMediaPlayer::durationChanged,
            this,           &Widget::durationChanged);
    connect(m_player, &QMediaPlayer::positionChanged,
            this,           &Widget::positionChanged);
    connect(ui->sliderPosition, &QSlider::sliderMoved,
            this,           &Widget::seek);
}

```

```

void Widget::sliderValueChange(int val)
{
    ui->labelVolume->setText(QString("%1").arg(val));
    m_player->setVolume(val);
}

void Widget::onOpenBtn()
{
    m_fName = QFileDialog::getOpenFileName(this,
                                            "Open File",
                                            QDir::homePath(),
                                            "MP3 files (*.mp3);");

    if(!m_fName.isNull())
        ui->lblFileName->setText(m_fName);
}

void Widget::onPlayBtn()
{
    if(!m_fName.isNull())
    {
        m_player->setMedia(QUrl::fromLocalFile(m_fName));
        ui->sliderPosition->setEnabled(true);
        m_player->play();
    }
}

void Widget::onPauseBtn()
{
    int state = m_player->state();
    if(state == QMediaPlayer::PausedState)
    {
        m_player->play();
    }
    else if(state == QMediaPlayer::PlayingState)
    {
        m_player->pause();
    }
}

void Widget::onStopBtn()
{
}

```

```

int state = m_player->state();
if(state == QMediaPlayer::PlayingState)
{
    m_player->stop();
}
}

void Widget::durationChanged(qint64 duration)
{
    m_duration = duration / 1000;
    ui->sliderPosition->setMaximum(m_duration);
}

void Widget::positionChanged(qint64 progress)
{
    if (!ui->sliderPosition->isSliderDown())
        ui->sliderPosition->setValue(progress / 1000);

    qint64 currentInfo = progress / 1000;

    QString playTimeStr;
    if (currentInfo || m_duration) {
        QTime currentTime((currentInfo / 3600) % 60, (currentInfo / 60) % 60,
                           currentInfo % 60, (currentInfo * 1000) % 1000);

        QTime totalTime((m_duration / 3600) % 60, (m_duration / 60) % 60,
                        m_duration % 60, (m_duration * 1000) % 1000);

        QString format = "mm:ss";

        if (m_duration > 3600)
            format = "hh:mm:ss";

        playTimeStr = currentTime.toString(format)
                     + " / " + totalTime.toString(format);
    }

    ui->labelPlayTime->setText(playTimeStr);
}

void Widget::seek(int seconds)
{

```

```

    m_player->setPosition(seconds * 1000);
}

Widget::~Widget()
{
    delete ui;
}

```

onPauseBtn() Slot 함수는 [일시 정지] 버튼을 클릭하면 호출되는 Slot 함수이다. 이 Slot 함수에서는 현재 Play 되고 있다면 일시정지 시킨다. 반대로 일시 정지된 상태라면 다시 재생한다.

durationChanged() Slot 함수는 MP3파일을 선택했을 때 호출한다. 여기서 MP3파일이 호출될 때 단위는 Millisecond 이다. 따라서 초로 저장하기 위해 1000으로 나누었다. positionChanged() Slot 함수는 재생 중에 재생되는 위치가 계속 바뀌므로 일정 시간을 주기로 계속 호출된다. 이 Slot 함수에서는 현재 재생되고 있는 시간과 총 재생 시간을 GUI상에 표시한다.

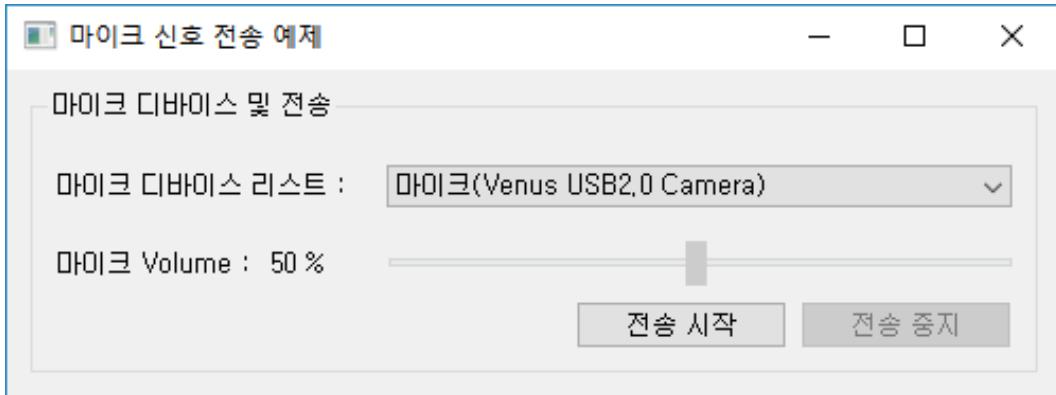
- 마이크 입력 신호 네트워크 송/수신 구현

이번에 다룰 예제는 QAudioInput 클래스를 이용해 마이크로 입력 받은 신호를 네트워크 UDP로 전송하는 어플리케이션을 구현한다. 그리고 네트워크 UDP 프로토콜로 전송한 데이터를 수신 받은 어플리케이션에서 스피커로 출력을 내보낸다.



<그림> 예제 어플리케이션의 역할

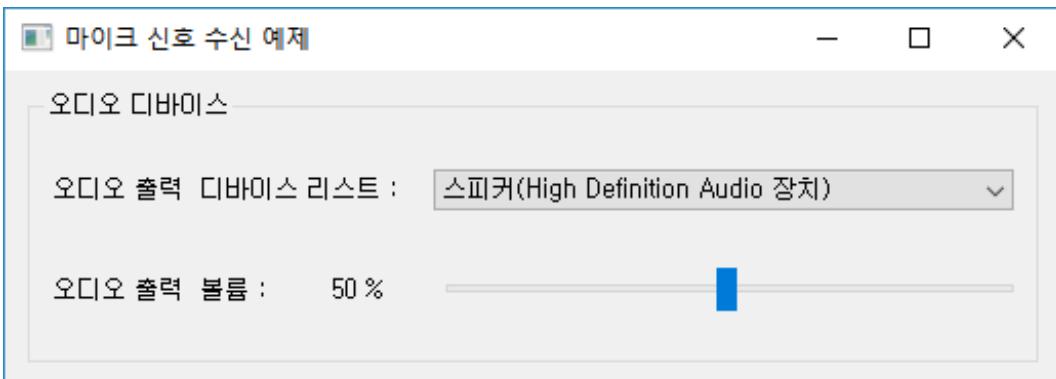
전송 어플리케이션은 QAudioInput 클래스를 이용해 마이크로부터 신호 데이터를 얻어와 데이터를 네트워크 UDP 프로토콜을 이용해 전송한다. 마이크 신호 데이터를 UDP로 수신 받은 어플리케이션은 QAudioOutput 클래스를 이용해 스피커로 출력한다.



<그림> 마이크 신호 전송 예제 실행 화면

위의 그림은 마이크 신호 전송 예제이다. 마이크 디바이스 리스트는 현재 시스템에서 인식한 마이크 디바이스를 출력한다.

그리고 아래에 [전송 시작] 버튼을 클릭하면 사용자가 선택한 마이크 디바이스로부터 입력되는 마이크 신호 데이터를 추출해 UDP 프로토콜을 경유해 수신 어플리케이션으로 전송한다.



<그림> 마이크 신호 수신 예제 실행 화면

위의 그림은 마이크 신호 수신 어플리케이션 실행 화면이다. 어플리케이션이 실행 되면 현재 콤보박스에서 선택된 오디오 출력 디바이스로 수신 받은 마이크 신호 데이터를 출력한다. 위의 GUI상에서 출력 디바이스 리스트 콤보박스 중 다른 디바이스를 선택하면 선택한 디바이스로 출력을 변경한다.

지금까지 살펴본 예제 중 송신하는 예제부터 살펴보도록 하자. 전체 예제 소스코드 Ch18 디렉토리 하위에 있으며 송신 어플리케이션 전체 소스코드는 01_AudioSender 디렉토리를 참조하면 된다. 다음 예제 소스는 송신 어플리케이션의 widget.h 소스코드이다. 예제 소스는 01_AudioReceiver 디렉토리를 참조하면 된다.

```

#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QAudioInput>
#include <QUdpSocket>
#include <QHostAddress>
#include <QEapsedTimer>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;

    QList<QAudioDeviceInfo> devList;

    QAudioFormat      mFormat;
    QAudioDeviceInfo  mDevInfo;
    QAudioInput       *mAudioInput;
    QIODevice        *mInput;

    QByteArray        mBuffer;
    QEapsedTimer     mElapsedTimer;
    int               mInputVolume;

    void audioInitialize();
    QUdpSocket        mUdpSocket;

private slots:
    void volSliderChanged(int val);
    void sendStartBtn();
    void sendStopBtn();

    void notified();

```

```

void stateChanged(QAudio::State state);
void readMore();
};

#endif // WIDGET_H

```

위의 예제 소스에서 QAudioFormat 은 오디오의 Sample rate, Channel, Sample size, 등 오디오 포맷을 설정하기 위한 기능을 제공한다. QAudioInput 클래스는 마이크 디바이스로부터 신호 데이터를 추출할 수 있는 기능을 제공한다.

QIODevice 는 QAudioInput 클래스로부터 추출한 마이크 신호를 QByteArray 로 변환하는 기능 구현 시 사용된다. 따라서 mBuffer 변수에는 마이크 장치(QIODevice) 신호로부터 얻어온 데이터를 저장한다. 그리고 이 데이터를 네트워크로 전송한다. 다음은 송신 예제의 widget.cpp 소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"
#include <QIODevice>
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->volSlider,      &QSlider::valueChanged,
            this,                  &Widget::volSliderChanged);
    connect(ui->pbtSendStart,   &QPushButton::clicked,
            this,                  &Widget::sendStartBtn);
    connect(ui->pbtSendStop,    &QPushButton::clicked,
            this,                  &Widget::sendStopBtn);

    ui->volSlider->setEnabled(false);
    ui->pbtSendStop->setEnabled(false);
    audioInitialize();
}

void Widget::audioInitialize()
{
    devList = QAudioDeviceInfo::availableDevices(QAudio::AudioInput);

    for(int i = 0; i < devList.size(); ++i)
        ui->comboDevList->addItem(devList.at(i).deviceName());
}

```

```

mDeviceInfo = devList.at(ui->comboDevList->currentIndex());

mFormat.setSampleRate(8000);
mFormat.setChannelCount(1);
mFormat.setSampleSize(16);
mFormat.setSampleType(QAudioFormat::SignedInt);
mFormat.setByteOrder(QAudioFormat::LittleEndian);
mFormat.setCodec("audio/pcm");

mAudioInput = new QAudioInput(mDeviceInfo, mFormat, this);
mInputVolume = ui->volSlider->value();

ui->volLabel->setText(QString("%1 %").arg(mInputVolume));

mBuffer = QByteArray(14096, 0);
mElapsedTimer.start();
}

void Widget::volSliderChanged(int val)
{
    mInputVolume = val;
    ui->volLabel->setText(QString("%1 %").arg(mInputVolume));
    mAudioInput->setVolume(mInputVolume);
}

void Widget::sendStartBtn()
{
    mDeviceInfo = devList.at(ui->comboDevList->currentIndex());
    mAudioInput = new QAudioInput(mDeviceInfo, mFormat, this);
    connect(mAudioInput, SIGNAL(notify()),
            this, SLOT(notified()));
    connect(mAudioInput, SIGNAL(stateChanged(QAudio::State)),
            this, SLOT(stateChanged(QAudio::State)));

    mAudioInput->setVolume(qreal(mInputVolume) / 100);

    mInput = mAudioInput->start();
    connect(mInput, SIGNAL(readyRead()), this, SLOT(readMore()));
}

void Widget::sendStopBtn()
{

```

```

if(mInput != nullptr) {
    disconnect(mInput, nullptr, this, nullptr);
    mInput = nullptr;
}
mAudioInput->stop();
mAudioInput->disconnect(this);
delete mAudioInput;
}

void Widget::notified()
{
    qDebug() << "elapsedUSecs=" << mAudioInput->elapsedUSecs()
           << "processUSecs=" << mAudioInput->processedUSecs();
}

void Widget::stateChanged(QAudio::State state)
{
    qDebug() << "state = " << state;
    if(state == QAudio::IdleState || state == QAudio::ActiveState)      {
        ui->volSlider->setEnabled(true);
        ui->pbtSendStart->setEnabled(false);
        ui->pbtSendStop->setEnabled(true);
    }
    else if(state == QAudio::StoppedState)
    {
        ui->volSlider->setEnabled(false);
        ui->pbtSendStart->setEnabled(true);
        ui->pbtSendStop->setEnabled(false);
    }
}

void Widget::readMore()
{
    if(!mAudioInput) return;

    qint64 len = mAudioInput->bytesReady();
    if(len > 4096)
        len = 4096;

    qint64 readLen = mInput->read(mBuffer.data(), len);
    if(readLen > 0) {
        mUdpSocket.writeDatagram(mBuffer.data(), len, QHostAddress::LocalHost,

```

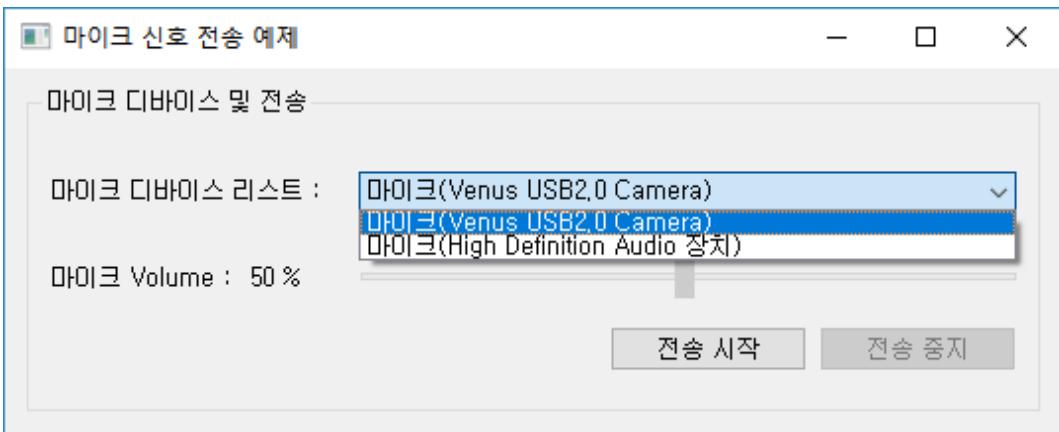
```

        15000);
    }
}

Widget::~Widget()
{
    delete ui;
}

```

위의 예제 소스에서 보는 것과 같이 클래스 생성자 함수에서 GUI 위젯의 Signal 과 Slot 함수를 연결한다. 그리고 audioInitialize() 함수를 호출한다. audioInitialize() 함수에서는 QAudioDeviceInfo 클래스를 이용해 마이크와 같은 입력 디바이스를 시스템으로부터 얻어와 콤보박스에 등록한다.



<그림> 오디오 입력 디바이스

그리고 오디오 포맷을 설정하고 QAudioInput 클래스를 선언한다. QAudioInput 클래스 오브젝트 선언 시 생성자의 첫 번째 인자는 GUI상에서 마이크 디바이스 리스트 콤보박스 항목 중 선택한 항목의 디바이스이다. 두 번째 인자는 QAudioFormat 클래스 오브젝트를 넘겨준다.

[전송 시작] 버튼을 클릭하면 QAudioOutput 클래스의 start() 함수를 이용해 마이크 신호 추출을 시작한다. 마이크 신호가 추출되면 readMore() Slot 함수가 호출된다. 이 Slot 함수에서는 추출한 데이터를 QByteArray로 저장한 다음 UDP로 전송한다. 다음은 수신 어플리케이션의 widget.h 소스코드이다.

```

#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

```

```

#include <QtMultimedia>
#include <QtNetwork>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;

    QList<QAudioDeviceInfo> devList;
    QAudioDeviceInfo mDeviceInfo;
    int mOutputVolume;
    QAudioFormat mFormat;
    QAudioOutput *mAudioOutput;
    QIODevice *mOutput;

    void audioInitialize();
    void audioOutputProcess(const QByteArray &ba);

    QUdpSocket *mUdpSocket;
    void networkInitialize();

private slots:
    void devListIndexChanged(int index);
    void volSliderChanged(int val);
    void readUdpData();
};

#endif // WIDGET_H

```

QAudioDeviceInfo 클래스는 시스템에서 출력 가능한 오디오 디바이스의 오브젝트를 선언한다.

네트워크로 UDP프로토콜을 경유해 수신 받은 데이터를 QAudioOutput 클래스를 이용해 선택한 출력 디바이스로 내보내는 기능을 제공한다. 다음은 widget.cpp 예제 소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"
#include <QDebug>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    audioInitialize();
    networkInitialize();

    connect(ui->comboDevList, SIGNAL(currentIndexChanged(int)),
            this,           SLOT(devListIndexChanged(int)));
    connect(ui->volSlider,   &QSlider::valueChanged,
            this,           &Widget::volSliderChanged);
}

void Widget::audioInitialize()
{
    devList = QAudioDeviceInfo::availableDevices(QAudio::AudioOutput);
    for(int i = 0; i < devList.size(); ++i) {
        ui->comboDevList->addItem(devList.at(i).deviceName());

    mDeviceInfo = devList.at(ui->comboDevList->currentIndex());

    mFormat.setSampleRate(8000);
    mFormat.setChannelCount(1);
    mFormat.setSampleSize(16);
    mFormat.setSampleType(QAudioFormat::SignedInt);
    mFormat.setByteOrder(QAudioFormat::LittleEndian);
    mFormat.setCodec("audio/pcm");

    mAudioOutput = new QAudioOutput(mDeviceInfo, mFormat, this);
    mOutput = mAudioOutput->start();

    mOutputVolume = ui->volSlider->value();
    ui->volLabel->setText(QString("%1 %").arg(mOutputVolume));
    mAudioOutput->setVolume(mOutputVolume);
}

void Widget::networkInitialize()
{
    mUdpSocket = new QUdpSocket(this);
}

```

```

mUdpSocket->bind(QHostAddress::LocalHost, 15000);
connect(mUdpSocket, SIGNAL(readyRead()), this, SLOT(readUpdData()));
}

void Widget::devListIndexChanged(int index)
{
    mDevInfo = devList.at(index);
    if(mOutput != nullptr) {
        disconnect(mOutput, nullptr, this, nullptr);
        mOutput = nullptr;
    }

    mAudioOutput->stop();
    mAudioOutput->disconnect(this);
    delete mAudioOutput;

    mAudioOutput = new QAudioOutput(mDevInfo, mFormat, this);
    mOutput = mAudioOutput->start();
}

void Widget::volSliderChanged(int val)
{
    mOutputVolume = val;
    ui->volLabel->setText(QString("%1 %").arg(mOutputVolume));
    mAudioOutput->setVolume(mOutputVolume);
}

void Widget::readUpdData()
{
    while (mUdpSocket->hasPendingDatagrams())
    {
        QNetworkDatagram datagram = mUdpSocket->receiveDatagram();

        QByteArray audioData = datagram.data();
        audioOutputProcess(audioData);
    }
}

void Widget::audioOutputProcess(const QByteArray &ba)
{
    qint64 len = ba.size();
    if(len > 0)

```

```
    mOutput->write(ba.data(), ba.size());  
}  
  
Widget::~Widget()  
{  
    delete ui;  
}
```

QAudioOutput 클래스의 start() 멤버 함수는 출력을 시작한다. 이 함수는 리턴 값은 출력 디바이스의 QIODevice 의 포인터 주소를 넘겨준다.

따라서 QIODevice 클래스의 mOutput 에 start() 함수가 넘겨준 포인터를 저장하며 실제 마이크로부터 수신 받은 데이터를 mOutput 오브젝트에 Write 하면 사용자가 선택한 출력 오디오 디바이스의 스피커 장치에 출력하게 된다.

그리고 생성자에서 호출한 networkInitialize() 함수는 네트워크 UDP 프로토콜을 경유해 수신 받을 QUdpSocket 클래스의 오브젝트를 선언 및 초기화 한다.

여기서는 Localhost 와 15000 번 포트로 수신 받기 위한 설정과 수신 받은 데이터 Signal 을 받으면 readUdpData() Slot 함수가 호출되게 Connect() 함수를 이용해 Signal 과 Slot을 연결해 준다.

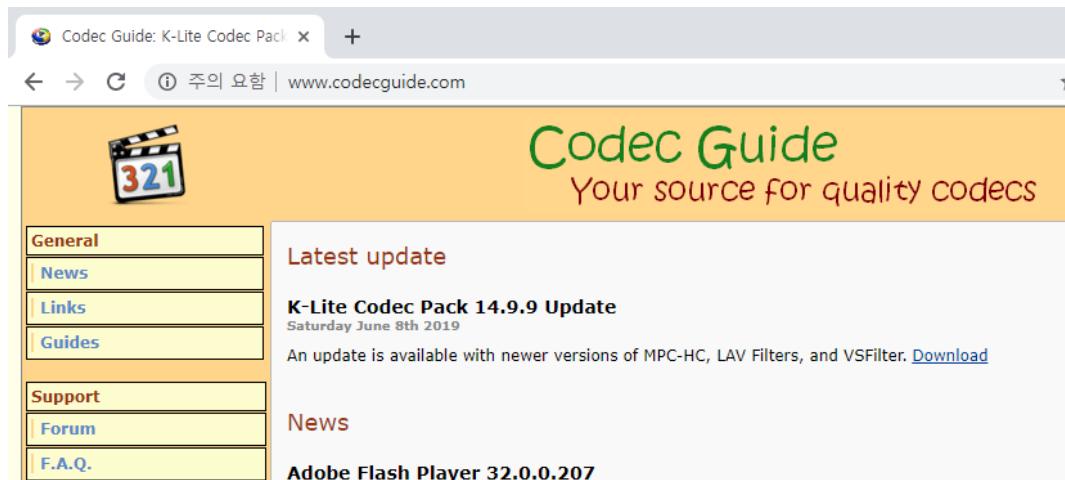
이 Slot 함수가 호출되면 수신 받은 데이터를 QByteArray 로 변환 후 QIODevice 클래스의 오브젝트에 Write 한다. 그러면 현재 GUI 에서 선택한 출력 디바이스로 마이크 신호 데이터를 스피커로 출력한다.

18.2. 비디오

동영상을 재생하기 위해서는 동영상을 인코딩한 영상 파일을 디코딩하기 위해서 압축 Codec을 제공 한다. 압축 Codec으로는 H.264, MOV 등 다양한 압축 Codec이 있다.

예를 들어 H.264 Codec 으로 인코딩된 동영상을 디코딩(재생)하기 위해서는 H.264 를 지원하는 Codec 이 시스템에 설치되어 있어야 한다. Qt는 다양한 동영상 Codec을 라이선스 문제로 인해 Qt 와 함께 제공하지 않는다.

그 이유는 각 Codec 에는 개발 라이선스가 연관되어 Qt에서 함께 제공할 수 없다. 따라서 필요한 Codec을 다운로드 받아 설치해야 한다. 쉽게 구할 수 있는 통합 Codec으로 K-Lite Codec Pack을 사용해 보도록 하자. www.codecguide.com 사이트에 접속하면 K-Lite Codec Packet 중 여러가지 종류가 있다. 이 중에서 Basic, Standard, Full 등을 제공하는데 여기서는 Full 버전을 다운로드 받아 설치한다.



<그림> K-Lite Codec Pack 공식 홈페이지

Qt 멀티미디어 모듈을 이용해 동영상 재생 어플리케이션을 구현 하는데 문제가 되지 않지만 재생하고자 하는 동영상이 사용한 Codec 이 설치되어 있지않으면 재생되지 않는다. 따라서 위의 그림에서 보는 것과 같이 Codec 을 꼭 설치한다.

Qt에서 제공한 멀티미디어 모듈을 이용해 동영상 플레이어를 구현한다고 가정해 보자. Qt에서 제공하는 동영상 재생기를 구현하기 위해서는 다음과 같이 2가지의 클래스를 이용해야 한다. 첫 번째는 QMediaPlayer 이다. 음악파일과 같은 MP3파일은 동영상의 영상을 출력할 필요가 없다. 그래서 QMediaPlayer 클래스를 이용해 재생 가능하다. 하

지만 동영상은 음원과 함께 동영상이 포함되어 있다. 그렇기 때문에 동영상을 표시해야 하는데 Qt 멀티미디어에서는 QVideoWidget 클래스를 사용해야 한다.

즉 QMediaPlayer 클래스로 동영상의 음원과 영상을 디코딩할 수 있다. 동영상의 화면에 출력할 GUI 위젯이 필요한데 Qt 멀티미디어 모듈에서 제공하는 QVideoWidget 을 함께 사용해 동영상 재생기를 구현할 수 있다. 다음 예제 소스는 QMediaPlayer 와 QVideoWidget 클래스를 이용한 예제이다.

```
player = new QMediaPlayer;
videoWidget = new QVideoWidget;
player->setVideoOutput(videoWidget);

player->setMedia(QUrl("http://example.com/movie1.mp4"));
//player->setMedia(QUrl::fromLocalFile("/My/ movie2.mp4"));

videoWidget->show();
player->play();
```

위의 예제 소스코드에 보는 것과 같이 동영상 출력을 위한 위젯을 설정하기 위해서 QMediaPlayer 클래스에서 제공하는 setVideoOutput() 멤버 함수를 이용해 동영상 표시 위젯을 설정하면 된다.

QMediaPlayer 클래스는 동영상을 출력할 위젯으로 QVideoWidget 클래스 외에도 QGraphicsVideoItem 클래스를 사용할 수 있다. 이 클래스는 일전에 다루었던 Graphics View Framework 에서 아이템 중 하나로써 QGraphicsView 영역에 동영상을 재생할 수 있는 아이템을 사용할 수 있다.

```
player = new QMediaPlayer(this);

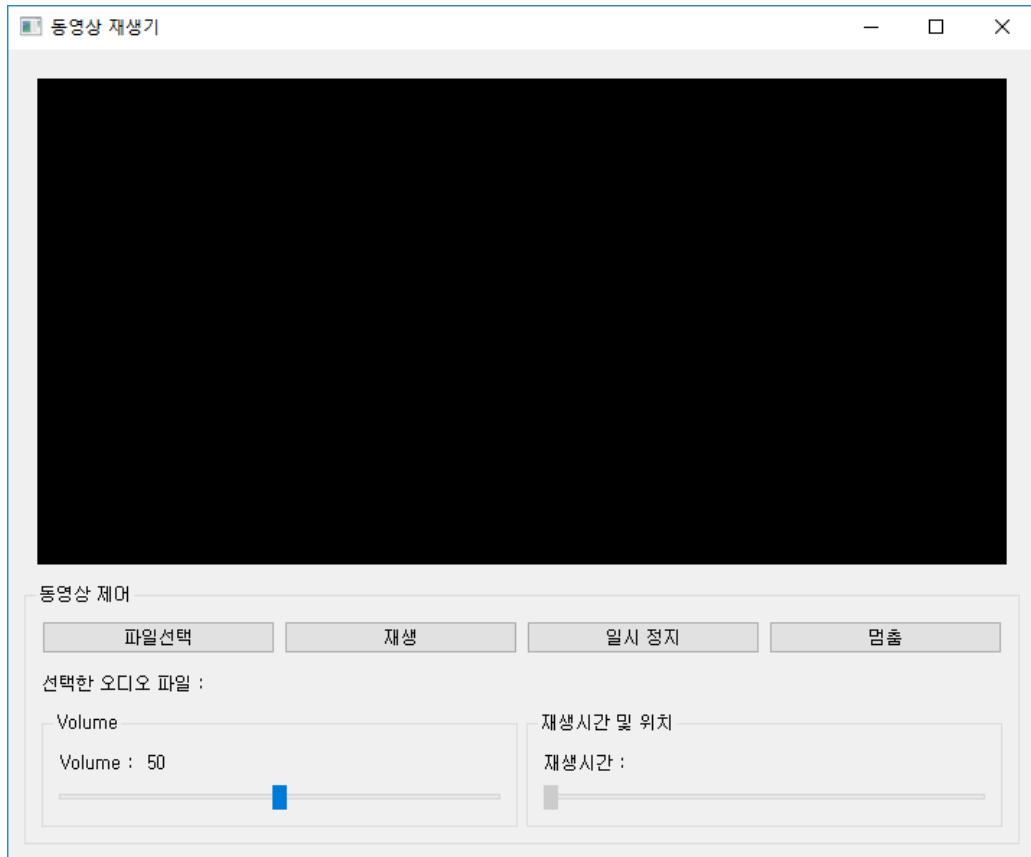
QGraphicsVideoItem *item = new QGraphicsVideoItem;
player->setVideoOutput(item);
graphicsView->scene()->addItem(item);
graphicsView->show();

player->setMedia(QUrl("http://example.com/my.mp4"));
player->play();
```

다음은 QMediaPlayer 클래스와 QVideoWidget 클래스를 이용해 동영상 재생기를 구현해 보도록 하자.

- 동영상 재생기 구현

이번 예제는 QMediaPlayer 클래스와 QVideoWidget 클래스를 이용한 동영상 재생기이다. 다음은 예제 실행 화면이다.



<그림> 예제 실행 화면

위의 그림에서 보는 것과 GUI 상에 동영상을 출력하기 위해서 QVideoWidget 을 사용하였다. 그리고 동영상이 출력되는 부분을 더블 클릭하면 동영상이 출력되는 영역이 전체화면으로 변경된다.

그리고 전체화면 상태에서 ESC키를 클릭하면 원래 크기 화면으로 돌아온다. 이와 같은 기능을 구현하기 위해서 QVideoWidget 클래스를 상속 받는 DisplayWidget 클래스를 구현하였다.

다음 예제 소스코드는 DisplayWidget 클래스의 헤더소스코드 이다. 전체 소스코드는 Ch18 > 02_VideoPlayer 디렉토리를 참조하면 된다.

```
#ifndef DISPLAYWIDGET_H  
#define DISPLAYWIDGET_H
```

```

#include <QVideoWidget>

class DisplayWidget : public QVideoWidget
{
    Q_OBJECT
public:
    explicit DisplayWidget(QWidget *parent = nullptr);
protected:
    void keyPressEvent(QKeyEvent *event) override;
    void mouseDoubleClickEvent(QMouseEvent *event) override;
    void mousePressEvent(QMouseEvent *event) override;
};
#endif // DISPLAYWIDGET_H

```

protected 키워드에 사용된 함수는 Virtual 함수이다. mouseDoubleClickEvent() Virtual 함수는 마우스 더블클릭 시 호출된다. 이 함수에서는 마우스 더블 클릭 시 동영상 랜더링 위젯이 전체 화면으로 전환된다. 다음 예제 소스코드는 DisplayWidget 클래스 소스 코드이다.

```

#include "displaywidget.h"
#include <QKeyEvent>
#include <QMouseEvent>

DisplayWidget::DisplayWidget(QWidget *parent) : QVideoWidget(parent)
{
    setSizePolicy(QSizePolicy::Ignored, QSizePolicy::Ignored);

    QPalette p = palette();
    p.setColor(QPalette::Window, Qt::black);
    setPalette(p);

    setAttribute(Qt::WA_OpaquePaintEvent);
}

void DisplayWidget::keyPressEvent(QKeyEvent *event)
{
    if (event->key() == Qt::Key_Escape && isFullScreen())
    {
        setFullScreen(false);
        event->accept();
    }
}

```

```

        else if (event->key() == Qt::Key_Enter && event->modifiers() & Qt::Key_Alt)
    {
        setFullScreen(!isFullScreen());
        event->accept();
    }
    else {
        QVideoWidget::keyPressEvent(event);
    }
}

void DisplayWidget::mouseDoubleClickEvent(QMouseEvent *event)
{
    setFullScreen(!isFullScreen());
    event->accept();
}

void DisplayWidget::mousePressEvent(QMouseEvent *event)
{
    QVideoWidget::mousePressEvent(event);
}

```

keyPressEvent() 는 키보드 이벤트가 발생하면 호출되며 ESC(Escape) 키 또는 ALT + Enter 키가 클릭 하면 원래 화면 크기로 전환된다.

다음 예제 소스코드는 DisplayWidget 클래스를 이용해 을 사용한 Widget 클래스를 살펴보도록 하자. 다음 예제는 widget.h 소스코드 이다.

```

#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QMediaPlayer>
#include "displaywidget.h"

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

```

```

private:
    Ui::Widget *ui;
    QString      m_fName;
    QMediaPlayer *m_player;
    DisplayWidget *m_displayWidget;
    qint64       m_duration;

private slots:
    void onOpenBtn();
    void onPlayBtn();
    void onPauseBtn();
    void onStopBtn();

    void sliderValueChange(int val);
    void durationChanged(qint64 duration);
    void positionChanged(qint64 progress);
    void seek(int seconds);
};

#endif // WIDGET_H

```

onOpenBtn() 은 [파일선택] 버튼을 클릭하면 호출되는 Slot 함수이다. 이 함수에서는 QFileDialog 클래스를 이용해 파일 선택ダイ얼로그 중 동영상 파일을 선택할 수 있다. onPlayBtn() 함수는 동영상 파일을 재생한다. QMediaPlayer 클래스의 setMedia() 함수를 이용해 재생할 파일을 설정한다.

그리고 play() 함수를 이용해 동영상 파일을 재생한다. onPauseBtn() 은 동영상 재생 일시정지 하는 기능을 제공하며 onStopBtn() 은 동영상을 중지 하는 기능을 제공한다. 다음 예제 소스코드는 widget.cpp 소스코드이다.

```

#include "widget.h"
#include "ui_widget.h"
#include <QFileDialog>
#include <QTime>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    m_duration = 0;

    ui->setupUi(this);
    connect(ui->volSlider, SIGNAL(valueChanged(int)),
            this,           SLOT(sliderValueChange(int)));

```

```

ui->sliderPosition->setEnabled(false);
connect(ui->pbtOpen, SIGNAL(clicked()), this, SLOT(onOpenBtn()));
connect(ui->pbtPlay, SIGNAL(clicked()), this, SLOT(onPlayBtn()));
connect(ui->pbtPause, SIGNAL(clicked()), this, SLOT(onPauseBtn()));
connect(ui->pbtStop, SIGNAL(clicked()), this, SLOT(onStopBtn()));

m_displayWidget = new DisplayWidget();

QVBoxLayout *vLay = new QVBoxLayout();
vLay->addWidget(m_displayWidget);
ui->containerWidget->setLayout(vLay);

m_player = new QMediaPlayer();
m_player->setVideoOutput(m_displayWidget);
connect(m_player, &QMediaPlayer::durationChanged,
        this, &Widget::durationChanged);
connect(m_player, &QMediaPlayer::positionChanged,
        this, &Widget::positionChanged);
connect(ui->sliderPosition, &QSlider::sliderMoved,
        this, &Widget::seek);
}

void Widget::sliderValueChange(int val)
{
    ui->labelVolume->setText(QString("%1").arg(val));
    m_player->setVolume(val);
}

void Widget::onOpenBtn()
{
    m_fName = QFileDialog::getOpenFileName(this,
                                           tr("Open File"), QDir::homePath());
    if(!m_fName.isNull())
        ui->lblFileName->setText(m_fName);
}

void Widget::onPlayBtn()
{
    if(!m_fName.isNull())
    {
        m_player->setMedia(QUrl::fromLocalFile(m_fName));
        ui->sliderPosition->setEnabled(true);
    }
}

```

```

        m_player->play();
    }
}

void Widget::onPauseBtn()
{
    int state = m_player->state();
    if(state == QMediaPlayer::PausedState) {
        m_player->play();
    }
    else if(state == QMediaPlayer::PlayingState) {
        m_player->pause();
    }
}
void Widget::onStopBtn()
{
    int state = m_player->state();
    if(state == QMediaPlayer::PlayingState) {
        m_player->stop();
    }
}

void Widget::durationChanged(qint64 duration)
{
    m_duration = duration / 1000;
    ui->sliderPosition->setMaximum(m_duration);
}

void Widget::positionChanged(qint64 progress)
{
    if (!ui->sliderPosition->isSliderDown())
        ui->sliderPosition->setValue(progress / 1000);

    qint64 currentInfo = progress / 1000;
    QString playTimeStr;
    if (currentInfo || m_duration)
    {
        QTime currentTime((currentInfo / 3600) % 60, (currentInfo / 60) % 60,
                          currentInfo % 60, (currentInfo * 1000) % 1000);

        QTime totalTime((m_duration / 3600) % 60, (m_duration / 60) % 60,
                       m_duration % 60, (m_duration * 1000) % 1000);

```

```

QString format = "mm:ss";
if (_duration > 3600) format = "hh:mm:ss";

playTimeStr = currentTime.toString(format) + " / "
                + totalTime.toString(format);
}

ui->labelPlayTime->setText(playTimeStr);
}

void Widget::seek(int seconds)
{
    m_player->setPosition(seconds * 1000);
}

Widget::~Widget()
{
    delete ui;
}

```

클래스 생성자에서 DisplayWidget 을 GUI 상에 배치하고 QMediaPlayer 클래스의 setVideoOutput() 함수의 첫 번째 인자로 DisplayWidget 클래스의 오브젝트를 넘겨주면 QMediaPlayer 클래스가 동영상을 재생한다. QMediaPlayer가 동영상을 재생하면 이 코딩한 화면을 DisplayWidget 클래스 위젯에 동영상 화면을 출력한다.

sliderValueChange() 는 GUI 상에서 볼륨 QSlider 위젯의 값이 변경되면 호출된다. 이 Slot 함수에서는 QMediaPlayer 클래스의 setVolume() 함수를 이용해 동영상 재생 시 볼륨을 조절할 수 있다.

durationChanged() 함수는 동영상 파일을 선택하고 정상적으로 QMediaPlayer 에 의해 재생되면 한번 호출된다. 이 함수에서는 재생 시간을 Millisecond 단위로 재생 시간을 알려준다. positionChanged() 는 현재 재생 되고 있는 위치를 알려주는 기능을 제공한다. 이 Slot 함수에서는 전체 시간을 “시:분:초” 로 변경한 후 GUI 위젯에 출력한다.

18.3. 카메라

Qt 멀티미디어 모듈에서 카메라를 이용해 응용 어플리케이션 개발할 수 있는 다양한 카메라 API를 제공한다. 다양한 클래스 중 가장 많이 사용하는 QCamera 클래스는 물리적인 카메라 하드웨어 장치를 이용해 응용 어플리케이션으로 개발할 수 있다.

카메라 장치로부터 얻어온 영상 이미지를 QCamera 클래스가 얻어온다. 그런 후 GUI에 출력하기 위해서 QCameraViewfinder 클래스를 이용해 GUI 상에 출력할 수 있다. 다음 예제 소스코드는 QCamera 클래스와 QCameraViewfinder를 이용해 카메라 장치를 사용하기 위한 방법이다.

```
QMcamea *camera = new QCamera;  
viewfinder = new QCameraViewfinder;  
  
camera->setViewfinder(viewfinder);  
viewfinder->show();  
  
camera->start();
```

QCamera 가 카메라 장치로부터 얻어온 영상을 QCameraViewfinder 클래스 위젯 영역에 출력한다. 위의 예제 소스코드에서 보는 것과 같이 QCamera 클래스에서 제공하는 setViewfinder() 멤버 함수의 첫 번째 인자로 QCameraViewfinder 클래스 오브젝트를 지정하면 된다.

그리고 QCamera 클래스의 start() 멤버 함수를 이용해 실제 카메라 장치로부터 영상을 얻어온다. 이 외에도 QCamera 클래스와 QCameraImageCapture 클래스를 이용하면 사용자가 원하는 카메라 영상을 이미지로 캡처 할 수 있다. 다음 예제는 캡처 기능을 구현하기 위한 예제이다.

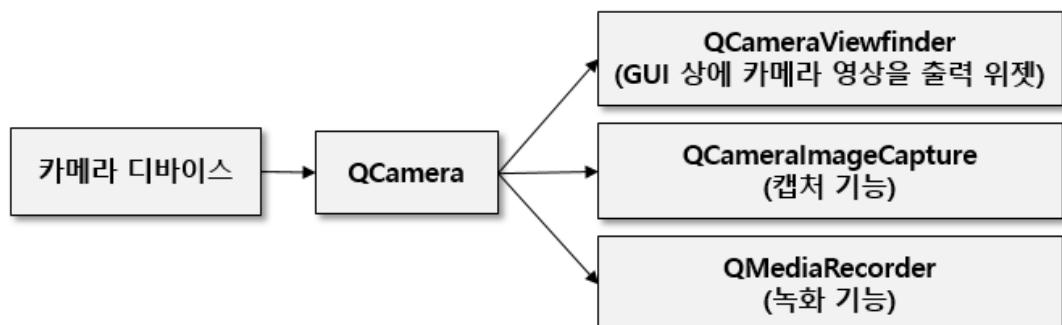
```
imageCapture = new QCameraImageCapture(camera);  
camera->setCaptureMode(QCamera::CaptureStillImage);  
camera->start();  
...  
camera->searchAndLock();  
imageCapture->capture();  
camera->unlock();  
...
```

또한 QCamera 클래스는 카메라로부터 영상을 녹화하기 위해 QMediaRecorder 클래스를 제공한다.

QMediaRecorder 클래스도 QCameralImageCapture 클래스와 마찬가지로 카메라 장치로부터 바로 캡처 영상 이미지를 가져오지 못하고 QCamera 클래스를 경유해 가져 오는 것과 같이 QMediaRecorder 클래스도 QCamera 클래스를 인터페이스로 사용해 카메라로부터 얻어온 영상을 녹화 할 수 있다.

```
camera = new QCamera;  
recorder = new QMediaRecorder(camera);  
  
camera->setCaptureMode(QCamera::CaptureVideo);  
camera->start();  
  
recorder->record();  
...  
recorder->stop();
```

지금까지 살펴본 카메라 영상을 얻어와 GUI 상에 표시하기 위한 방법, 카메라 영상 캡처 방법 그리고 녹화 하는 방법에 대해서 살펴보았다. 따라서 QCamera 클래스는 물리적인 카메라 하드웨어 장치로부터 영상을 다루기 위한 인터페이스를 제공한다.



<그림> QCamera 클래스 관계

카메라의 포커스를 다루기 위해 QCameraFocus, 카메라의 Exposure 설정을 위해서 QCameraExposure 클래스 등을 제공한다. 지금까지 카메라를 다루기 위한 방법에 대해 살펴보았다. 다음은 실제 카메라 장치로부터 영상을 다루는 예제를 구현해 보도록 하자.

● 카메라 장치를 이용한 예제 구현

이번 예제는 USB 카메라 장치로부터 카메라 영상을 GUI상에 출력하는 예제이다. 추가 기능으로 카메라 영상을 캡처하는 기능을 부가 기능으로 구현해 보도록 하자. 다음 그

림은 예제 실행 화면이다.



<그림> 예제 실행 화면

위의 그림에서 보는 것과 같이 GUI 상에 QComboBox 위젯은 시스템에서 검색된 카메라 장치 디바이스를 모두 등록한다. [Start] 버튼을 클릭하면 카메라로부터 영상을 GUI 상에 출력한다. [Stop] 버튼은 카메라 영상을 가져오는 것을 중지한다.

그리고 [Capture] 버튼은 현재 출력되는 영상을 이미지로 캡처한다. 다음은 예제의 widget.h 헤더이다. 전체 소스코드는 Ch18 > 03_CameraCapture 디렉토리를 참조하면 된다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QCamera>
```

```

#include <QCameraViewfinder>
#include <QCameraImageCapture>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
    void initCamera();
    void cameraDevicesSearch();

private:
    Ui::Widget *ui;
    QCamera           *mCamera;
    QCameraViewfinder *mViewfinder;
    QCameraImageCapture *mCapture;
    QList<QByteArray>   camDevNameLists;

private slots:
    void onStartBtn();
    void onStopBtn();
    void onCaptureBtn();
    void camError(QCamera::Error error);
    void imageCaptured(int pId, QImage pPreview);
};

#endif // WIDGET_H

```

위의 예제에서 initCamera() 함수는 QCamera 와 QCameralImageCapture 클래스를 초기화한다. 그리고 GUI 상에 QCameralImageCapture 클래스 위젯을 배치한다.

cameraDevicesSearch() 함수는 시스템에서 검색된 모든 카메라 장치를 QComboBox 클래스의 위젯에 등록한다.

onStartBtn() 함수는 [Start] 버튼 클릭 시 호출된다. onStopBtn() 함수는 [중지] 버튼 클릭 히 호출되며 onCaptureBtn() 함수는 [Capture] 버튼 클릭 시 호출 된다. camError() 함수는 카메라로부터 에러 Signal이 발생하면 호출된다.

그리고 마지막 imageCaptured() 함수는 카메라 캡처 기능을 수행하면 호출된다. 다음 예제는 widget.cpp 소스코드 이다.

```

#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    this->initCamera();
    this->cameraDevicesSearch();

    connect(ui->pbtStart, SIGNAL(clicked()), this, SLOT(onStartBtn()));
    connect(ui->pbtStop, SIGNAL(clicked()), this, SLOT(onStopBtn()));
    connect(ui->pbtCapture, SIGNAL(clicked()), this, SLOT(onCaptureBtn()));
}

Widget::~Widget()
{
    delete ui;
    delete mCapture;
    delete mCamera;
}

void Widget::initCamera()
{
    mCamera      = new QCamera;
    mCapture     = new QCameraImageCapture(mCamera);
    mViewfinder  = new QCameraViewfinder();

    QVBoxLayout *vLay = new QVBoxLayout();
    vLay->addWidget(mViewfinder);
    ui->camWidget->setLayout(vLay);
}

void Widget::cameraDevicesSearch()
{
    camDevNameLists.clear();
    ui->comboBox->clear();

    camDevNameLists.clear();
    ui->comboBox->clear();

    foreach (const QByteArray &deviceName, QCamera::availableDevices())
    {

```

```

        QString description;
        description = mCamera->deviceDescription(deviceName);
        camDevNameLists.append(deviceName);
        ui->comboBox->addItem(description);
    }
}

void Widget::onStartBtn()
{
    delete mCapture;
    delete mCamera;

    if(camDevNameLists.count() < 1) {
        mCamera = new QCamera;
    }else{
        int curIndex = ui->comboBox->currentIndex();
        mCamera = new QCamera(camDevNameLists.at(curIndex));
    }

    connect(mCamera, SIGNAL(error(QCamera::Error)),
            this,      SLOT(camError(QCamera::Error)));

    mCamera->setViewfinder(mViewfinder);
    mCamera->setCaptureMode(QCamera::CaptureVideo);
    mCapture = new QCameraImageCapture(mCamera);

    //imageCapture->setCaptureDestination(
    //           QCameraImageCapture::CaptureToBuffer);

    mCapture->setCaptureDestination(
        QCameraImageCapture::CaptureToBuffer |
        QCameraImageCapture::CaptureToFile);

    mCapture->setBufferFormat(QVideoFrame::Format_RGB32);
    connect(mCapture, SIGNAL(imageCaptured(int,QImage)),
            this,      SLOT(imageCaptured(int,QImage)));

    mCamera->start();
}

void Widget::onStopBtn()
{

```

```

    mCamera->stop();
}

void Widget::onCaptureBtn()
{
    mCapture->capture("c:/CaptuerImage.jpg");
}

void Widget::camError(QCamera::Error error)
{
    qDebug() << Q_FUNC_INFO << "Error : " << error;
}

void Widget::imageCaptured(int pId, QImage pPreview)
{
    Q_UNUSED(pId);

    qDebug() << "IMAGE CAPTUE SIZE (WIDTH X HEIGHT) : "
           << pPreview.byteCount();
}

```

만약 시스템에 여러 대의 카메라 장치가 있다고 가정해보자. 여러 대의 카메라가 있다면 GUI 상의 콤보박스상에 여러 대의 카메라 장치가 등록될 것이다. 사용할 카메라 장치를 콤보박스에서 선택하고 [Start] 버튼을 누르면 onStartBtn() Slot 함수가 호출된다. onStartBtn() Slot 함수에서 QCamera 클래스와 QCameralImageCapture 클래스를 초기화 및 선언 하였다.

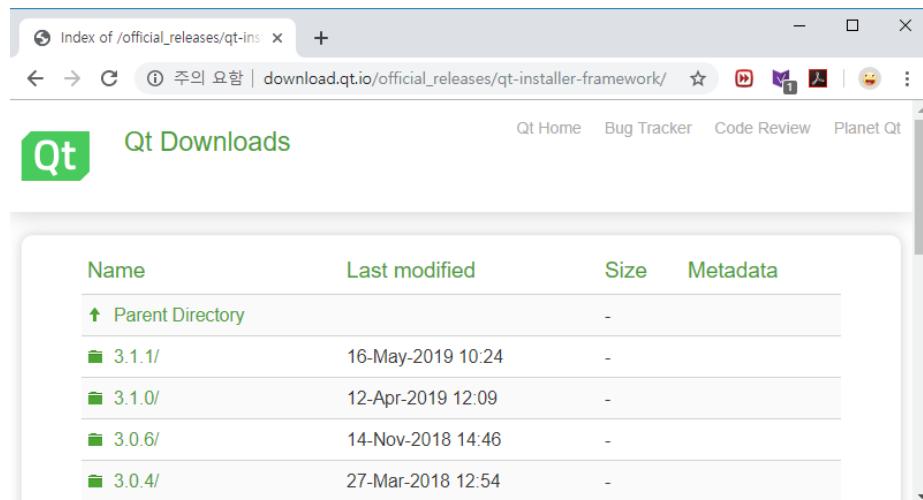
19. Qt Install Framework

Qt Install Framework 는 사용자가 구현한 응용 어플리케이션을 사용자가 설치할 수 있도록 설치 SDK 또는 설치 배포판을 만들 수 있는 기능을 제공한다. 즉 설치 파일과 더불어 설치 파일이 사용하는 라이브러리, 각종 리소스 파일 등을 하나의 설치 배포판으로 제작해 사용자에게 제공할 수 있다.

예를 들어 MS윈도우에서 Install Shield를 이용해 구현한 어플리케이션을 설치 배포판을 제공하는 것과 같이 Qt Install Framework를 사용할 수 있다. Qt Install Framework는 Qt와 마찬가지로 멀티 플랫폼을 지원한다. Qt Install Framework는 리눅스를 사용하든, MS윈도우를 사용하든 또는 MacOS를 사용하든 동일한 Qt Install Framework 사용 방법을 통해 설치 파일을 만들 수 있다.

✓ Qt Install framework 다운로드 및 설치

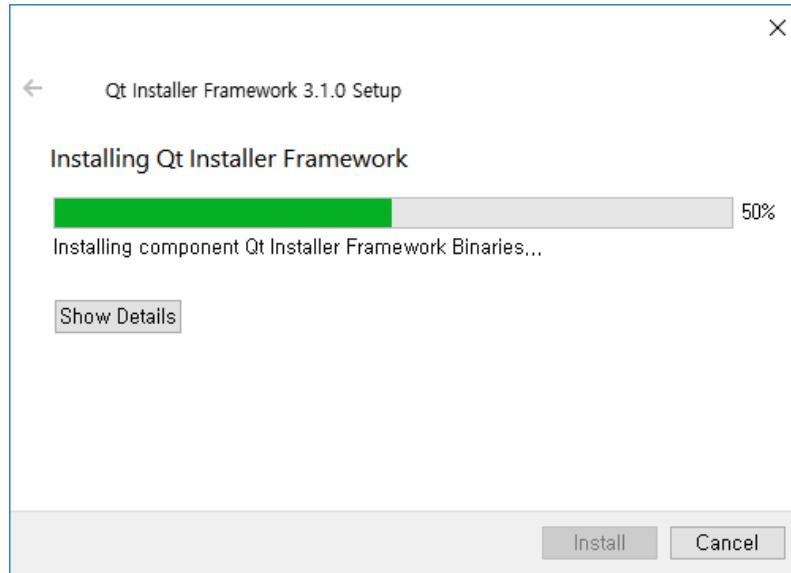
웹브라우저를 이용해 http://download.qt.io/official_releases/qt-installer-framework 사이트에 접속하면 Qt Install Framework를 다운로드 받을 수 있다.



<그림> Qt Install Framework 다운로드 URL

위의 그림에서 보는 것과 같이 각 버전 디렉토리 별로 제공한다. 원하는 버전을 클릭하면 운영체제 플랫폼 별 사용할 수 있는 Qt Install Framework를 제공한다. 여기서는 MS Window 버전을 사용해 보도록 하자.

다음 그림에서 보는 것과 같이 QtInstallerFramework-win-x86.exe 버전을 다운로드 받은 후 설치해 보도록 하자.



<그림> Qt Install Framework 설치 화면

✓ Qt Install framework 배포 방식

Qt Install Framework 는 Online 방식과 Offline 방식을 제공한다. Online 방식은 특정 웹 서버 Repository에 접속하여 자동 다운로드 및 설치하는 방식이다. Offline 방식은 설치 파일을 직접 다운로드 받아 설치하는 방식이다. 그리고 어플리케이션을 배포하기 위한 방법으로 다이얼로그 형태의 Setup Wizard 형식으로 배포판을 제작할 수 있다.



<그림> Qt Install Framework 로 설치 Wizard Setup 다이얼로그 예

위의 그림에서 보는 것과 같이 Workflow 형태의 템플릿 형태로 Qt Install Framework를 이용해 설치파일을 만들 수 있다. 다음 예제를 통해 설치 배포판을 만들어 보도록 하자.

- Qt Install Framework를 이용해 Offline 방식의 설치 배포판 만들기

아래 예제 소스코드에서 보는 것과 같이 01_Installer_Example라는 디렉토리를 생성한다. 그리고 생성한 디렉토리에 01_Installer_Example.pro 파일을 다음과 같이 작성한다.

```
TEMPLATE = aux

INSTALLER = 01_Installer_Example
INPUT = $$PWD/config/config.xml $$PWD/packages

myexam.input = INPUT
myexam.output = $$INSTALLER

myexam.commands = C:/Qt/QtIFW-3.1.0/bin/binarycreator \
                  -c $$PWD/config/config.xml \
                  -p $$PWD/packages ${QMAKE_FILE_OUT}

myexam.CONFIG += target_predeps no_link combine

QMAKE_EXTRA_COMPILERS += myexam
```

TEMPLATE 키워드는 설치 파일이 라이브러리 인지 어플리케이션인지 구분하기 위해서 명시한다. 만약 라이브러리 lib를 입력한다. 만약 어플리케이션이라면 aux를 입력한다. 이번 예제는 응용 어플리케이션을 배포하는 것으로 위의 예제에서 보는 것과 같이 aux를 입력한다.

INSTALLER 키워드는 설치 파일의 이름을 명시한다. 나중에 빌드하면 생성된 설치 파일 이름이 INSTALLER 키워드로 지정한 이름이 설치파일명으로 생성된다.

다음의 INPUT 키워드는 두가지 항목을 명시해야 한다. 첫 번째는 config.xml 파일을 입력한다. 두 번째 항목은 packages 디렉토리를 명시한다.

config.xml 파일은 프로젝트의 이름, 버전, 공급자, MS윈도우인 경우 시작 메뉴의 이름 등을 명시한다.

packages 디렉토리에는 설치된 실행파일, 즉 개발한 응용어플리케이션 실행 파일 및 라이브러리와 설치 시 라이선스 정보, 파일 정보 및 설치 스크립트등이 위치한 디렉토리다. myexam.input 키워드는 INPUT 키워드를 명시한다. 두 번째 myexam.output 은

INSTALLER 키워드에서 명시한 설치 파일명을 명시한다. myexam.commands 는 설치 파일을 만들기 위해서 Qt Install Framework 에서 제공하는 binarycreator 를 입력하고 -c 옵션에는 config.xml 파일의 위치와 이름을 지정한다. 그리고 -p 는 패키지를 입력한다. 위와 같이 프로젝트 파일을 작성 완료하였다면 config 디렉토리를 만든다. 그리고 config 디렉토리 안에 다음 예제와 같이 config.xml을 작성한다.

```
<?xml version="1.0" encoding="UTF-8"?>

<Installer>

    <Name>Example Software</Name>
    <Version>1.0.0</Version>
    <Title>Example Software</Title>

    <Publisher>Qt 개발자</Publisher>
    <StartMenuDir>Qt 개발자 메뉴</StartMenuDir>
    <TargetDir>C:/Example_Software</TargetDir>

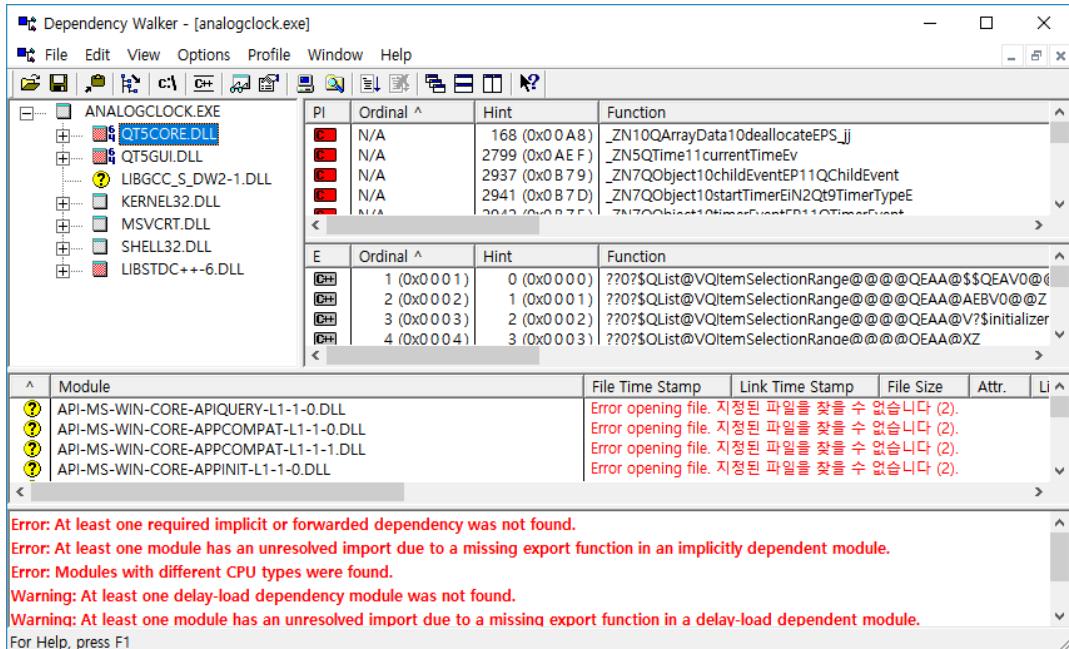
</Installer>
```

프로젝트의 가장 상위 디렉토리에 packages 디렉토리를 생성하고 packages 디렉토리 안에 com.vendor.product 디렉토리를 생성한다. 그리고 com.vendor.product 디렉토리 안에 data 와 meta 디렉토리를 생성한다. data 디렉토리는 설치할 응용 프로그램의 실행 파일, 실행파일이 사용하는 라이브러리, 그리고 실행파일이 사용하는 리소스 파일 등을 이 디렉토리에 복사한다.

여기서는 Qt 예제 중 아날로그 예제 실행 파일을 MinGW 32 Bit Release 모드로 빌드한 후 data 디렉토리에 넣을 것이다. 이 예제 외에 다른 실행파일 및 라이브러리를 복사해 넣어도 된다.

지금까지 Qt Creator IDE 툴에서 빌드하고 실행하였지만 이번에는 빌드한 실행파일을 직접 실행 보자. 그러면 실행 파일이 사용하는 라이브러리가 없다는 에러가 나올 것이다. Qt Creator IDE 툴에서는 자동으로 라이브러리 위치를 찾아 주지만 직접 실행할 경우 라이브러리도 실행파일이 있는 위치에 같이 존재해야 한다.

만약 어떤 라이브러리가 필요한지 궁금하다면 MS 윈도우에서는 아래 그림과 같이 Dependency Walker를 이용해 실행 파일이 참조하는 라이브러리 목록을 볼 수 있다. Dependency Walker 는 인터넷에서 무료로 다운로드 받을 수 있다.



<그림> Dependency Walker 실행 화면

만약 리눅스라면 ldd 명으로 실행파일이 참조하는 라이브러리를 확인할 수 있다.

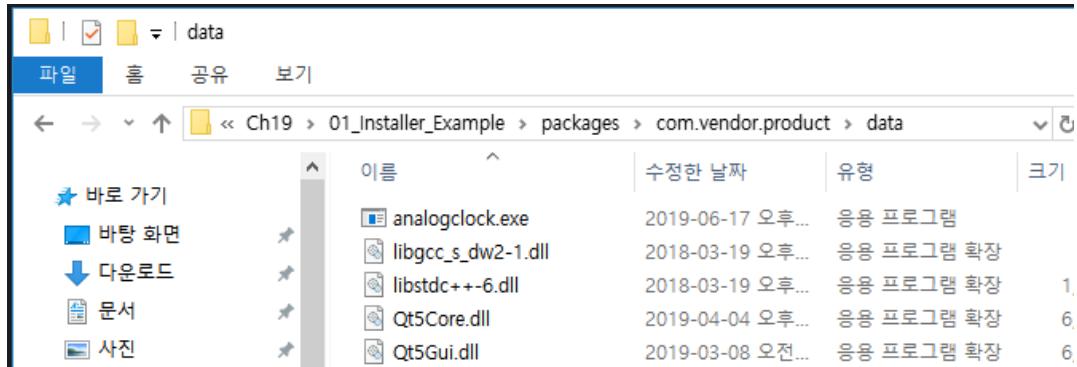
```
# ldd /usr/sbin/qtApplication
    linux-vdso.so.1 => (0x00007fff85bff000)
    libpcre.so.3 => /lib/libpcre.so.3 (0x00007ff366142000)
    libapr-1.so.0 => /usr/lib/libapr-1.so.0 (0x00007ff30)
    libpthread.so.0 => /lib/libpthread.so.0 (0x00007ff00)
    libexpat.so.1 => /lib/libexpat.so.1 (0x00007ff360000)
...
```

만약 사용하는 운영체제가 MacOS 라면 otool 명령어를 이용해 다음과 같이 배포 어플리케이션의 참조 라이브러리를 확인할 수 있다.

```
otool -L MyApp.app/Contents/MacOS/MyApp
```

Qt에서 제공하는 analogclock 예제를 MinGW 32 Bit 버전 컴파일러로 빌드했기 때문에 Qt 설치 디렉토리 하위에 각 컴파일러 별 디렉토리에 bin 디렉토리에 보면 필요한 파일들이 있을 것이다.

해당 컴파일러 하위 bin 디렉토리에 analogclock 예제 실행 파일이 필요한 라이브러리를 같은 위치에 복사한다.



<그림> 필요한 라이브러리 및 실행파일

그리고 com.vendor.product 디렉토리 아래에 meta 디렉토리를 만들 후 다음과 같이 파일을 작성한다.

<표> 설치 배포판 제작에 필요한 파일들

파일명	설명
package.xml	패키지 설정 파일
license.txt	라이선스 정보 파일
installscript.qs	설치 스크립트

package.xml 파일은 표시할 이름, Description, 설치 디아일로그에서 표시할 라이선스정보가 기록된 파일명 그리고 설치 시 필요한 스크립트 파일을 명시한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package>
    <DisplayName>Example 소프트웨어</DisplayName>
    <Description>Example 아날로그 시계 예제입니다.</Description>
    <Version>1.0.0-1</Version>
    <ReleaseDate>2030-05-18</ReleaseDate>
    <Licenses>
        <License name="My License Agreement" file="license.txt"/>
    </Licenses>
    <Default>script</Default>
    <Script>installscript.qs</Script>
</Package>
```

Installscript.qs 는 설치할 패키지의 Short Cut 메뉴, Start 메뉴의 lnk 파일, 실행 아이콘의 위치를 명시한다. Installscript.qs 는 아래와 같이 작성 한다.

```
function Component()
```

```

{
    // default constructor
}

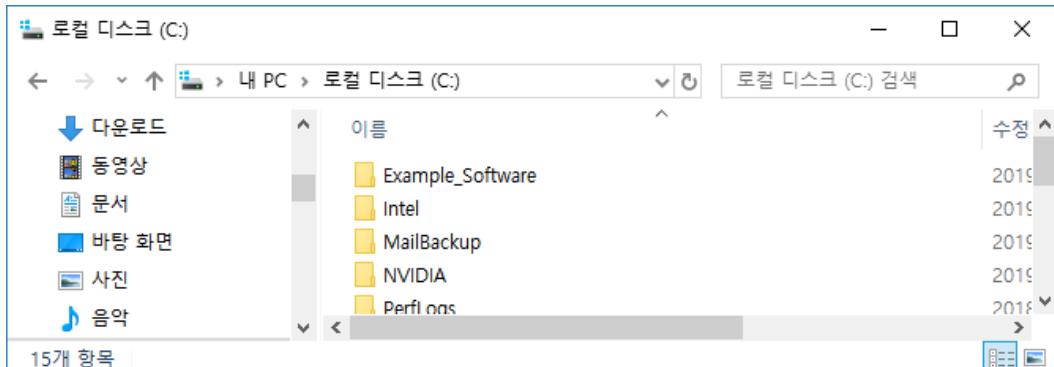
Component.prototype.createOperations = function()
{
    component.createOperations();

    if (systemInfo.productType === "windows")
    {
        component.addOperation(
            "CreateShortcut",
            "@TargetDir@/analogclock",
            "@StartMenuDir@/analogclock.lnk",
            "workingDirectory=@TargetDir@",
            "iconPath=@TargetDir@/main_icon.ico");
    }
}

```

위와 같이 작성하였으면 설치 배포판을 만들 모든 준비가 끝났다. Qt Creator 툴에서 빌드하면 빌드 디렉토리에 01_Installer_Example.exe 설치 배포판 파일이 생성된 것을 확인 할 수 있다.

01_Installer_Example.exe 설치 배포판을 설치 후 config.xml 파일에서 <TargetDir> 태그에서 명시한 디렉토리가 생성되고 필요한 파일이 있는지 확인해 보자.



<그림> Example_Software 디렉토리

그리고 아래 그림과 같이 시작 메뉴에 설치한 프로그램이 등록되었는지 확인해 보도록 하자.



<그림> 시작 메뉴의 analogclock 메뉴

그리고 아래 그림에서 보는 것과 같이 config.xml 파일에 명시한 데로 시작 메뉴 그룹이 등록된 것을 확인할 수 있다.



<그림> 시작 메뉴 그룹

지금까지 다룬 Qt Install Framework 예제는 Ch19 > 01_Installer_Example 디렉토리를 참조하면 된다.