

The complete software development framework

Qt 6 Programming

Second Edition, Ver 2.0

Qt 개발자 커뮤니티
www.qt-dev.com

Qt는 매우 직관적이고 높은 런타임 성능과 작은 설치 공간으로 가독성이 뛰어나고 유지 관리 및 재사용이 쉬운 코드를 생성하며 크로스 플랫폼을 지원합니다.

예수님은 당신을 사랑합니다.

Qt 6 프로그래밍

배포 버전 Version 2.0, 2024.03.01

홈페이지 URL www.qt-dev.com

머리말

Qt에 관심 있는 분들께 조금이나마 도움이 되는 마음으로 문서 파일 그대로, 무료로 독자 분들께 배포합니다.

한가지 바라는 게 있다면, 아직도 믿음이 없는 분들이 복음을 듣고 하나님 아버지께 돌아오길 기도합니다. 또한 독생자 이신 예수님이 이 땅에 오셔서 우리의 죄를 대신 짊어지셨습니다. 존귀하신 예수님의 보혈로 우리의 죄를 사하여 주셨습니다. 하나님 아버지는 자식을 사랑하는 마음으로 믿지 않는 모든 이들이 하나님 아버지께 돌아오길 기다리고 계십니다.

이 책을 접하신 분들 중 아직도 믿음이 없는 분들이 계시다면 예수님을 구주로 영접하고 믿음의 자녀로 거듭나길 간절한 마음으로 기도하고 축원합니다. 또한 하나님의 선하신 은혜가 여러분과 함께하길 기도합니다. 아멘.

37. 예수께서 그에게 말씀하셨다. '네 마음을 다하고, 네 목숨을 다 하고, 네 뜻을 다하여, 주 너의 하나님을 사랑하여라' 하였으니, 38. 이것이 가장 중요하고 으뜸 가는 계명이다. 39. 둘째 계명도 이것과 같은데, '네 이웃을 네 몸과 같이 사랑하여라' 한 것이다.

[새번역성경] 마태복음 22:37~39

Table of Contents

| | |
|---|-----|
| 1. Qt 6 소개와 설치 | 1 |
| 2. Qt에서 제공하는 유용한 개발 툴 | 16 |
| 3. CMake를 이용한 프로젝트 빌드..... | 20 |
| 3.1. CMake를 이용한 Console 어플리케이션 구현 | 22 |
| 3.2. CMake를 이용한 GUI 어플리케이션 구현 | 30 |
| 4. qmake를 이용한 프로젝트 빌드 | 38 |
| 4.1. qmake를 이용한 Console 어플리케이션 구현 | 39 |
| 4.2. qmake를 이용한 GUI 어플리케이션 구현 | 46 |
| 5. Qt GUI Widgets | 55 |
| 6. Layout..... | 102 |
| 7. Qt에서 제공하는 데이터 타입과 클래스 | 107 |
| 8. Container Classes | 119 |
| 9. Signal and Slot..... | 124 |
| 10. Qt Designer를 이용한 GUI 설계 | 129 |
| 11. 다이얼로그 | 139 |
| 12. QMainWindow를 이용한 GUI 구현 | 151 |
| 13. Stream..... | 156 |
| 14. 파일 입출력 | 160 |
| 15. Qt Property..... | 169 |
| 16. Model and View..... | 175 |
| 17. QPainter 클래스를 이용한 2D 그래픽스 | 187 |

| | |
|---|-----|
| 18. QPainter 를 이용한 Chromakey 영상 처리 구현 | 209 |
| 19. Timer | 216 |
| 20. Thread programming | 220 |
| 21. XML | 228 |
| 22. JSON..... | 238 |
| 23. 라이브러리 제작 (CMake)..... | 248 |
| 23.1. Shared 라이브러리 제작과 사용하기 | 249 |
| 23.2. 라이브러리와 함께 빌드하기..... | 262 |
| 24. 라이브러리 제작 (qmake)..... | 265 |
| 24.1. Shared 라이브러리 제작과 사용하기 | 267 |
| 24.2. 라이브러리와 함께 빌드하기 | 279 |
| 25. D-Pointer..... | 281 |
| 26. Database Programming | 299 |
| 27. Qt for Android | 319 |
| 27.1. MS Windows 에서 Android 개발 환경 구축과 앱 배포..... | 320 |
| 27.2. Linux 에서 Android 개발 환경 구축과 앱 배포..... | 336 |
| 28. Essential Network Programming | 349 |
| 28.1. TCP 프로토콜 기반 서버/클라이언트 구현..... | 352 |
| 28.2. 동기 방식 비 동기 방식 구현..... | 360 |
| 28.3. UDP 프로토콜 기반 네트워크 통신 구현 | 368 |
| 28.4. Broadcast..... | 375 |
| 28.5. Multicast..... | 381 |
| 28.6. 채팅 서버/클라이언트 구현 | 388 |

| | |
|---|-----|
| 29. Qt WebSocket | 403 |
| 30. Qt WebEngine | 415 |
| 31. Qt HTTP Server | 429 |
| 32. Deployment | 441 |
| 33. Qt Graphics View Framework | 452 |
| 34. Animation Framework and State Machine | 462 |
| 35. Qt Chart | 475 |
| 36. Qt Data Visualization | 487 |
| 37. Inter Process Communication | 498 |
| 37.1. Unix Domain Socket and Named pipe | 499 |
| 37.2. QProcess | 510 |
| 37.3. Shared Memory | 516 |
| 37.4. Qt D-Bus | 527 |
| 38. Multimedia | 548 |
| 38.1. Audio | 550 |
| 38.2. Video | 577 |
| 38.3. Camera | 589 |
| 39. Serial Communication | 598 |
| 40. Qt Positioning | 610 |
| 41. Qt PDF | 615 |
| 42. Qt Printer Support | 621 |
| 43. Qt Pprotobuf | 627 |

1. Qt 6 소개와 설치

Qt는 MS윈도우, 리눅스 그리고 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. (Python 도 가능하지만 여기서 Python은 다루지 않음) 그리고 Qt 프레임워크는 C++외에도 Qt Quick(QML) 을 제공한다. Qt Quick 은 QML이라는 인터프리터언어를 사용한다. Qt로 어플리케이션 개발 시, GUI 또는 UX를 QML을 이용해 개발할 수 있다. QML을 사용하면 Function Logic과 Design Logic으로 분리해 개발할 수 있다.



Qt Quick
Architecture

QML
(Design Logic)

C++
(Function Logic)

QWidget 을 사용하는 경우
Architecture

C++
(Design Logic)

C++
(Function Logic)

예수님은 당신을 사랑합니다.

여기서 Design Logic은 GUI를 뜻한다. Function Logic은 GUI상에서 어떤 버튼을 클릭했을 동작하는 과정 또는 기능을 뜻한다.

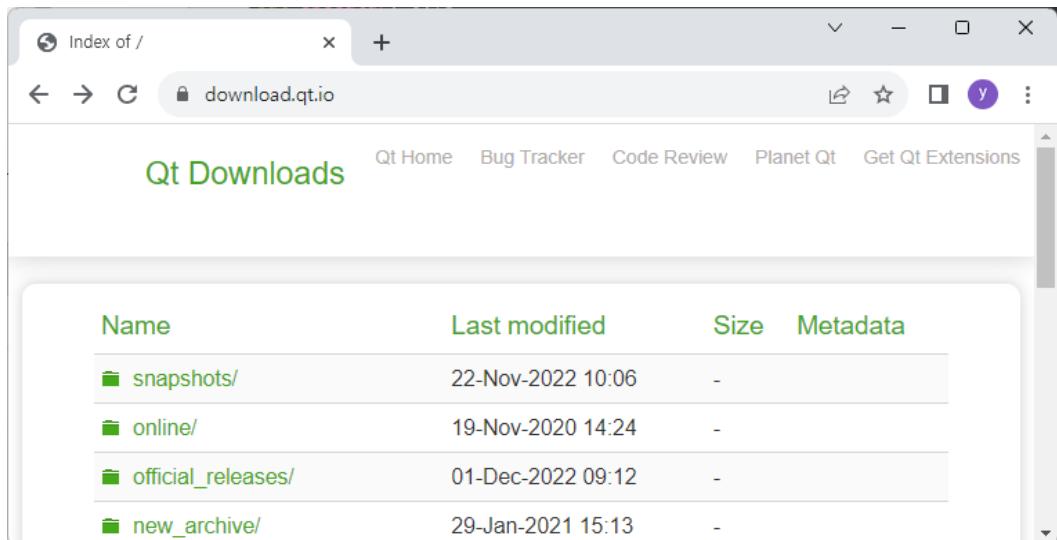
Qt로 어플리케이션을 개발 할 때 Function Logic을 C++로 개발하고 Design Logic을 QML로 개발함으로써 소스코드 재 사용성을 높일 수 있다. 예를 들어 GUI화면을 QML로 개발하고 기능을 C++로 개발함으로써 두개의 Logic을 분리함으로 써 소스코드의 재 사용성을 높일 수 있다.

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 적합한 경우가 더러 있다. (꼭 그런 것만은 아님)

그리고 메모리 사용량이 제한된 임베디드 시스템 이거나 CPU(또는GPU) 성능이 낮다면 C++을 사용하는 것이 QML을 사용하는 것보다 성능 면에서 유리하다. 따라서 개발하려는 소프트웨어가 어떤 특징이 있는지 먼저 살펴보고 GUI(Design Logic)을 C++로 개발할지, QML을 사용할 것인지 결정하는 것이 바람직하다.

● Qt 온라인 설치 파일 다운로드

Qt는 데스크탑 기반의 플랫폼(또는 운영체제)으로 MS윈도우, 리눅스 그리고 MacOS 플랫폼을 지원한다. Qt는 Qt 공식 홈페이지에서 플랫폼 별 다운로드가 가능하다.



| Name | Last modified | Size | Metadata |
|--------------------|-------------------|------|----------|
| snapshots/ | 22-Nov-2022 10:06 | - | |
| online/ | 19-Nov-2020 14:24 | - | |
| official_releases/ | 01-Dec-2022 09:12 | - | |
| new_archive/ | 29-Jan-2021 15:13 | - | |

위의 화면에서 보는 것과 같이 항목 중 “official_releases” 을 클릭한다. 다음으로 항목 중 “online_installers” 항목을 클릭한다.

예수님은 당신을 사랑합니다.

A screenshot of a web browser window titled "Index of /official_releases". The address bar shows "download.qt.io/official_releases/". The page displays a list of directory entries:

| Name | Last modified | Size | Metadata |
|-------------------------|-------------------|------|-------------------------|
| qt/ | 03-Apr-2023 07:37 | - | |
| qt-installer-framework/ | 07-Jun-2023 12:51 | - | |
| qbs/ | 03-Aug-2023 17:32 | - | |
| pyside/ | 30-Nov-2015 13:39 | - | |
| online_installers/ | 07-Jun-2023 13:52 | - | |
| jom/ | 12-Dec-2018 15:13 | - | |
| gdb/ | 17-Nov-2014 13:42 | - | |
| additional_libraries/ | 03-Mar-2021 10:18 | - | |
| QtForPython/ | 12-Apr-2022 15:00 | - | |
| timestamp.txt | 09-Aug-2023 08:00 | 11 | Details |

"online_installers" 를 클릭하면 아래 화면에서 보는 것과 같이 플랫폼 별 온라인 설치 파일을 다운로드 받을 수 있다.

A screenshot of a web browser window titled "Index of /official_releases/online". The address bar shows "download.qt.io/official_releases/online_installers/". The page title is "Qt Downloads". The navigation bar includes links to "Qt Home", "Bug Tracker", "Code Review", "Planet Qt", and "Get Qt Extensions". The main content area displays a table of files:

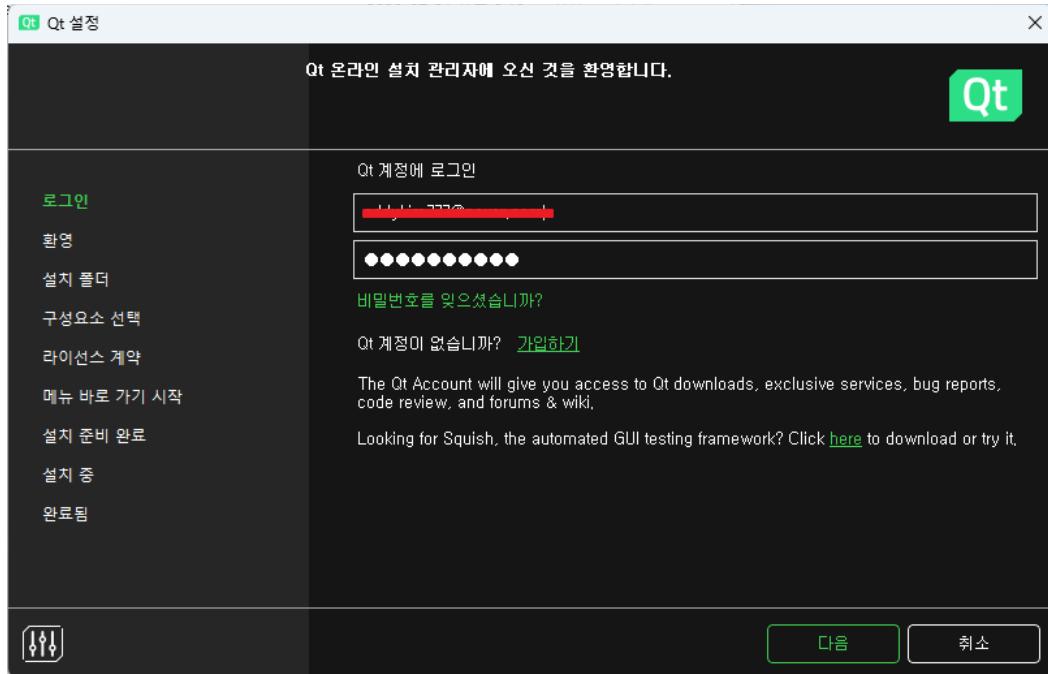
| Name | Last modified | Size | Metadata |
|-----------------------------------|-------------------|------|-------------------------|
| Parent Directory | - | - | |
| qt-unified-windows-x64-online.exe | 07-Jun-2023 12:49 | 39M | Details |
| qt-unified-mac-x64-online.dmg | 07-Jun-2023 12:49 | 15M | Details |
| qt-unified-linux-x64-online.run | 07-Jun-2023 12:49 | 52M | Details |

확장자가 exe 파일은 MS Windows에서 설치하기 위한 온라인 설치 파일이다. 확장자가 dmg는 Mac OS용이다. 확장자가 run 인 파일은 리눅스에서 설치하기 위한 온라인 인스톨러이다.

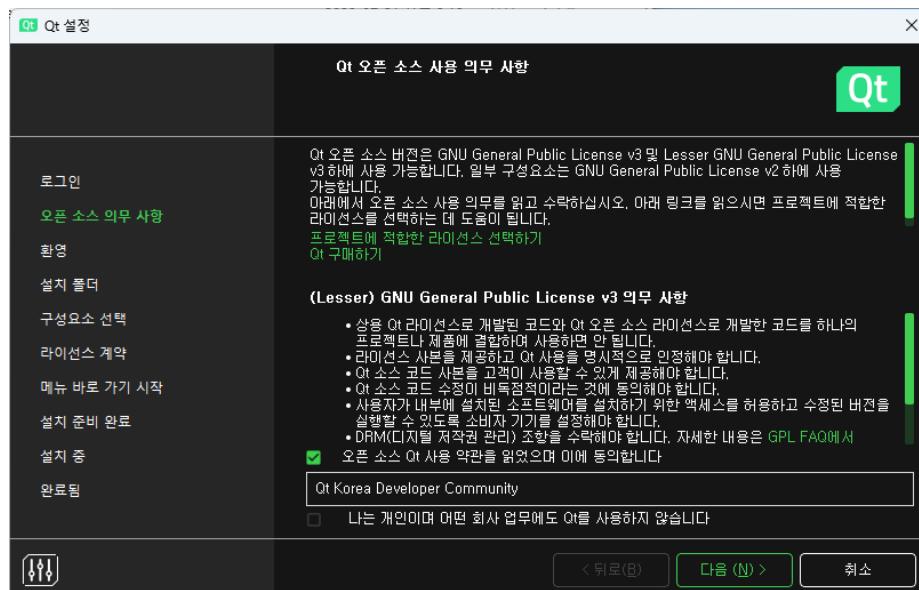
예수님은 당신을 사랑합니다.

- MS Windows 에서 Qt 설치

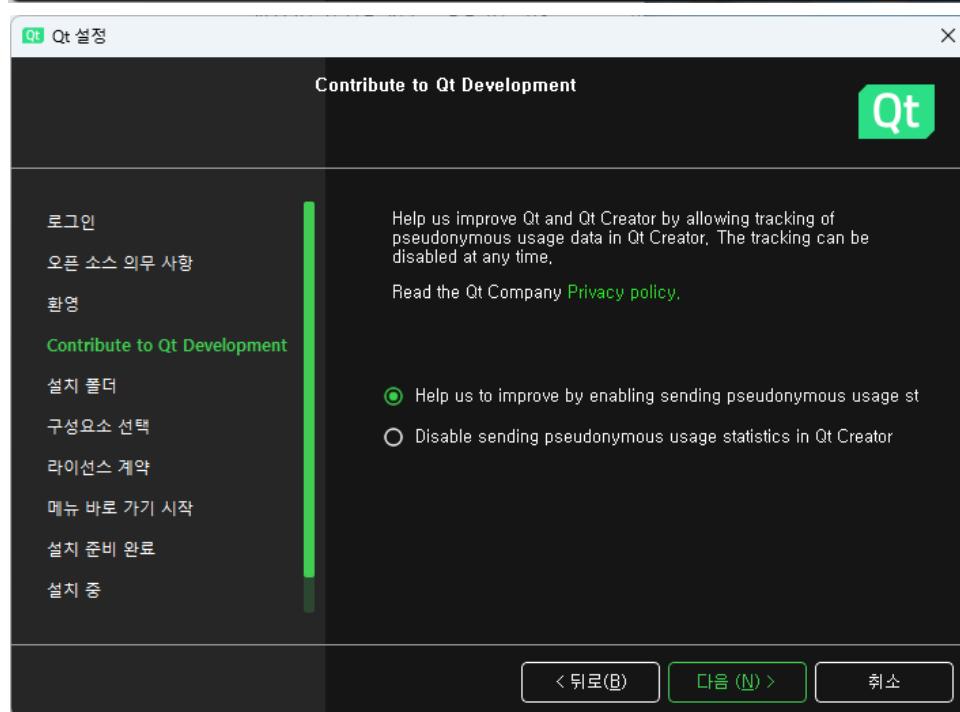
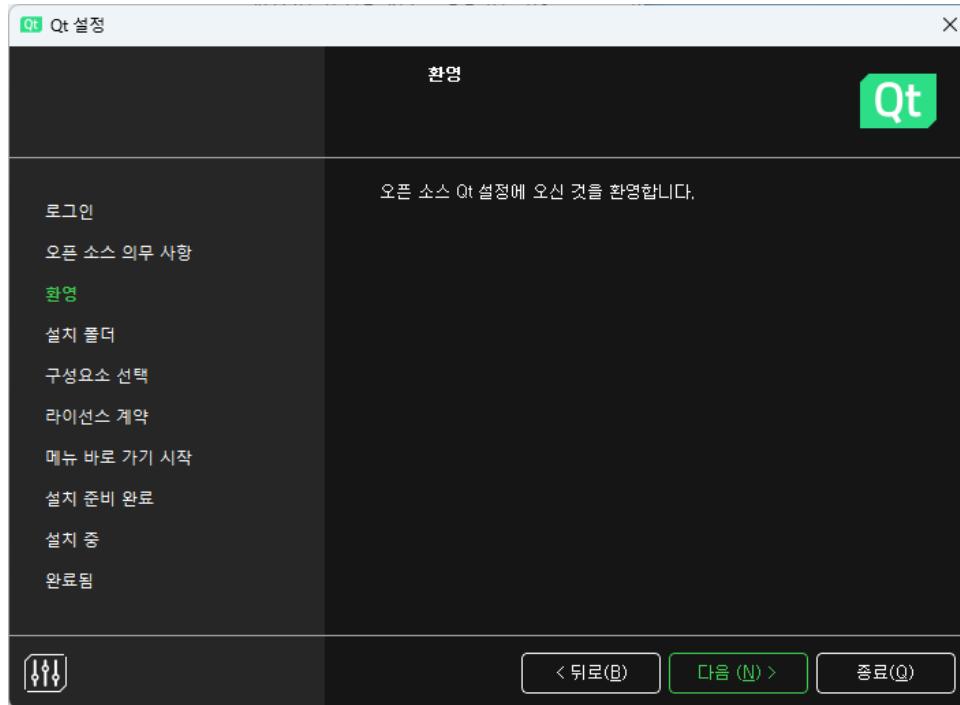
확장자가 exe 파일인 것을 다운로드 받아 실행한다.



위의 다이얼로그에서 보는 것과 같이 아이디와 비밀번호를 입력한다. 만약 아이디가 없다면 위의 다이얼로그 화면에서 [가입하기]를 클릭해 계정을 생성한 후, 계정 정보를 입력하면 된다. 계정 정보(아이디와 비밀번호)를 입력을 완료하였다면 [다음] 버튼을 클릭한다.

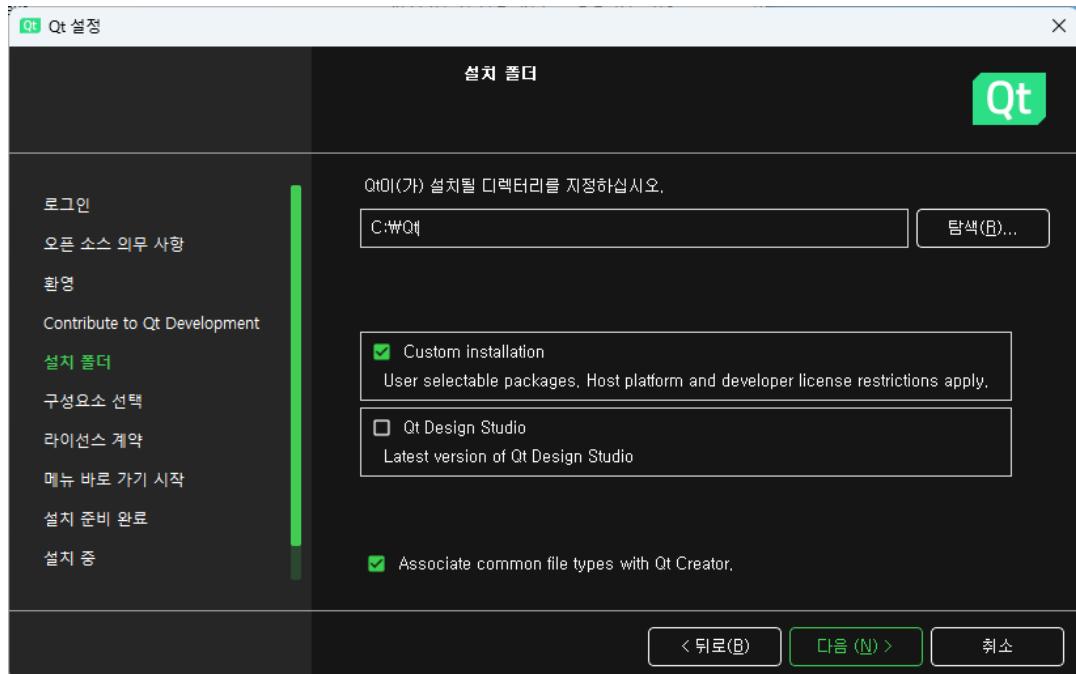


예수님은 당신을 사랑합니다.



위의 그림은 Qt Creator IDE를 사용할 때 발생할 수 있는 오류정보등을 실시간으로 보내어 Qt를 개선하는데 동의할 것인지 그렇지 않을 것인지 물어보는ダイ얼로그이다. 동의하면 위의그림에서 보는 것과 같이 첫 번째 항목을 선택하고, 그렇지 않다면 두 번째 항목을 선택하면 된다.

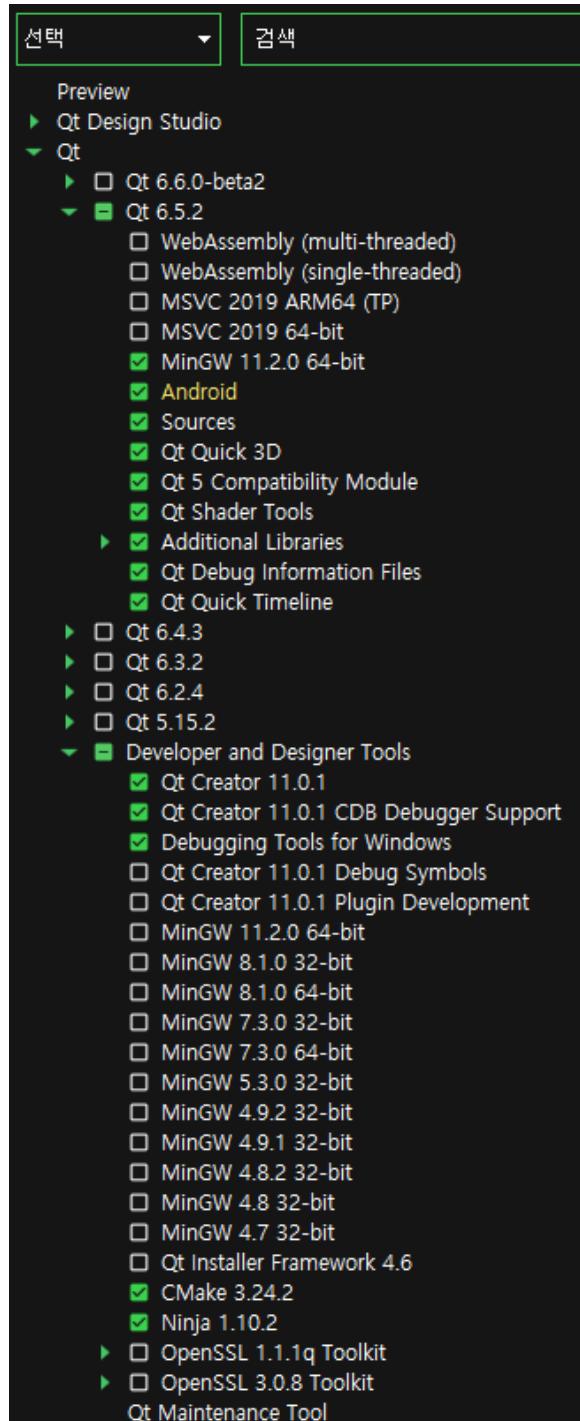
예수님은 당신을 사랑합니다.



Custom Installation 항목을 선택하면 설치 항목(Component)을 직접 선택할 수 있다.



위의 구성요소에서 선택해야 할 항목은 아래 그림에서 보는 것과 같이 선택하면 된다.



버전 중에서 Qt 6.5.2 버전을 설치한다. Qt는 LTS 버전과 그렇지 않은 버전으로 나누어 진다. LTS는 Long Term Service 의 약자이며 LTS 가 아닌 버전에 비해서 업데이트를 지속적으로 제공한다. 하지만 향후 다른 상위 버전에서 LTS를 지정하면 가장 낮은 버전의 LTS 버전은 종료된다. 현재 Qt에서 제공하는 LTS 버전은 6.5 와 5.15 버전이다.

예수님은 당신을 사랑합니다.

여기서는 6.5.2 를 선택한다. 다른 버전은 설치해도 되지만 되도록 6.5.2 버전을 설치하기를 권장한다. 아래 표의 각 Component 중 주요한 항목을 설명한 항목이다.

<표> 주요 Component 설명

| 카테고리 | Component | 설명 |
|----------|----------------------------------|---|
| Qt 6.X.X | WebAssembly (multi-threaded) | Qt를 이용해 WebAssembly 어플리케이션을 개발하기 위해서 필요하다. |
| | WebAssembly (single-threaded) | Qt를 이용해 WebAssembly 어플리케이션을 개발하기 위해서 필요하다. |
| | MSVC 2019 ARM64(TP) | Microsoft Visual C++ 2019 ARM 용 64 Bit 컴파일러이다. 작성한 소스코드를 컴파일 할 때 이 항목으로 컴파일 하기 위해서 MSVC 2019 ARM64 컴파일러를 별도로 설치해야 한다. |
| | MSVC 2019 64-bit | Microsoft Visual C++ 2019 64Bit 컴파일러이다. 작성한 소스코드를 컴파일 할 때 MSVC 2019 64 bit 컴파일러를 별도로 설치해야 한다. |
| | MinGW 11.2.0 64-bit | 오픈소스 GCC/G++ 64 Bit 컴파일러 |
| | Android | 모바일 Android 플랫폼에 사용하기 위한 컴파일러. |
| | Source | Qt API를 디버깅 하기 위해서 필요. 예를 들어 특정 소스코드 지점을 브레이크 포인트를 이용해 Qt API 소스코드를 디버깅 하기 위해 필요. |
| | Qt Quick 3D | QML에서 Qt Quick 3D 모듈을 이용해 3D를 사용하기 위한 모듈이다. |
| | Qt 5 Compatibility Module | Qt 6에서 Qt 5의 호환성을 유지하기 위해서 제공한다. |

예수님은 당신을 사랑합니다.

- 리눅스에서 Qt 설치

리눅스 플랫폼(운영체제)은 여러 배포판이 있다. 예를 들어 우분투, Mint 등 여러 종류가 있지만 여기서는 우분투를 사용한다. 다른 배포판을 사용해도 무방하다.

| Name | Last modified | Size | Metadata |
|--------------------|-------------------|------|----------|
| snapshots/ | 22-Nov-2022 10:06 | - | |
| online/ | 19-Nov-2020 14:24 | - | |
| official_releases/ | 01-Dec-2022 09:12 | - | |
| new_archive/ | 29-Jan-2021 15:13 | - | |
| ministro/ | 20-Feb-2017 10:32 | - | |
| linguist_releases/ | 26-Mar-2019 07:49 | - | |
| learning/ | 24-Feb-2021 15:09 | - | |

Qt 온라인 설치파일을 다운로드 받기 위해서 download.qt.io 사이트에 방문한다. 위의 그림에서 보는 것과 같은 페이지가 나오면 [official_release]를 클릭한다.

| | | |
|-------------------------|-------------------|-----------------|
| qt3dstudio/ | 28-Oct-2020 14:22 | - |
| qt/ | 03-Apr-2023 07:37 | - |
| qt-installer-framework/ | 07-Jun-2023 12:51 | - |
| qbs/ | 03-Aug-2023 17:32 | - |
| pysisde/ | 30-Nov-2015 13:39 | - |
| online_installers/ | 07-Jun-2023 13:52 | - |
| jom/ | 12-Dec-2018 15:13 | - |
| gdb/ | 17-Nov-2014 13:42 | - |
| additional_libraries/ | 03-Mar-2021 10:18 | - |
| QtForPython/ | 12-Apr-2022 15:00 | - |
| timestamp.txt | 11-Aug-2023 08:00 | 11 Details |

[online_installers]를 클릭한다.

예수님은 당신을 사랑합니다.

Qt Downloads

| Name | Last modified | Size | Metadata |
|-----------------------------------|-------------------|------|-------------------------|
| ↑ Parent Directory | | - | |
| qt-unified-windows-x64-online.exe | 07-Jun-2023 12:49 | 39M | Details |
| qt-unified-mac-x64-online.dmg | 07-Jun-2023 12:49 | 15M | Details |
| qt-unified-linux-x64-online.run | 07-Jun-2023 12:49 | 52M | Details |

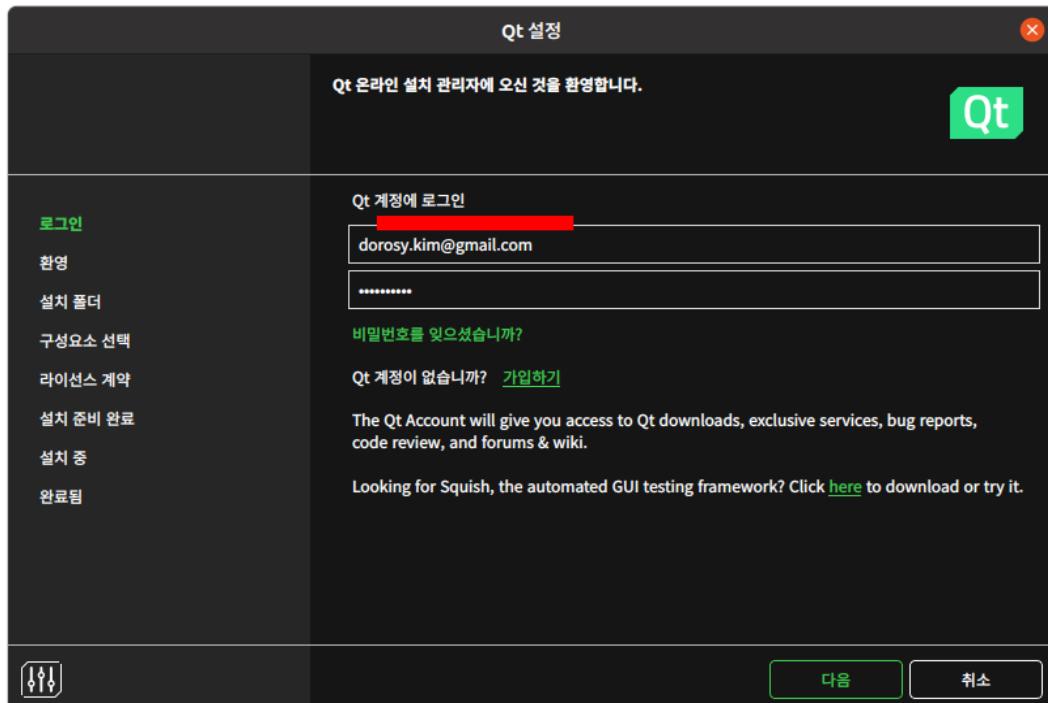
For Qt Downloads, please visit [qt.io/download](#)

확장자가 "run" 인 파일을 다운로드 받는다.

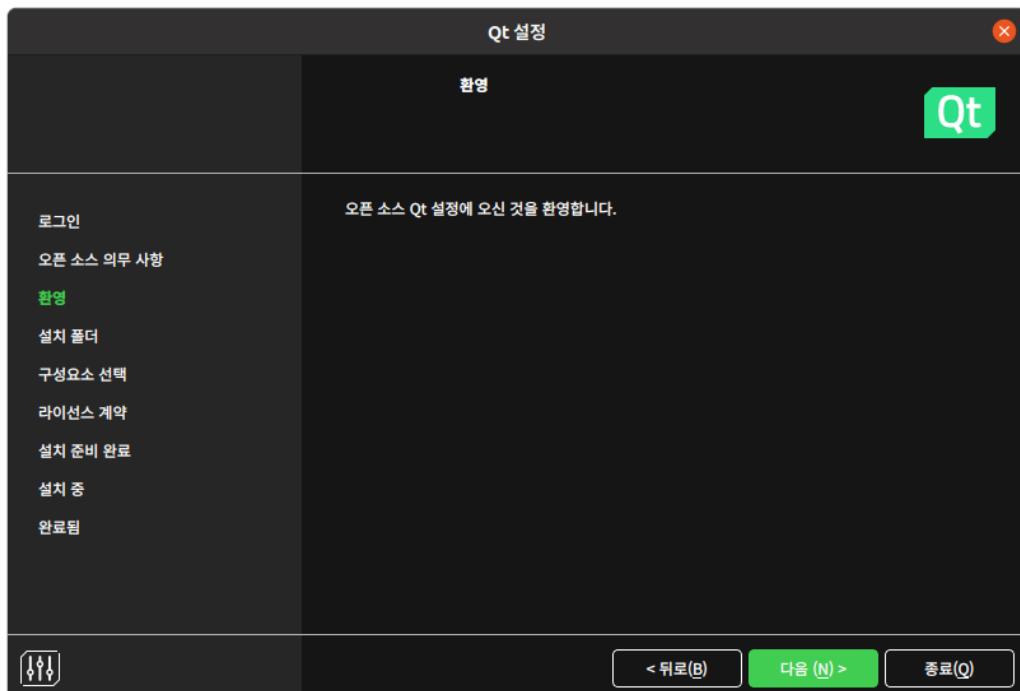
```
dev@ubuntu:~/Downloads$ ls -tlr
합계 53744
-rw-r--r-- 1 dev dev 55032849 8월 11 15:43 qt-unified-linux-x64-4.6.0-online.run
dev@ubuntu:~/Downloads$ chmod 755 qt-unified-linux-x64-4.6.0-online.run
dev@ubuntu:~/Downloads$ ./qt-unified-linux-x64-4.6.0-online.run
```

다운로드가 완료되면 다운로드 디렉토리로 이동한다. 다운로드 파일은 실행 권한이 없으므로 chmod 로 실행 권한을 준다. 그런 다음 Qt 온라인 설치 파일을 실행한다.

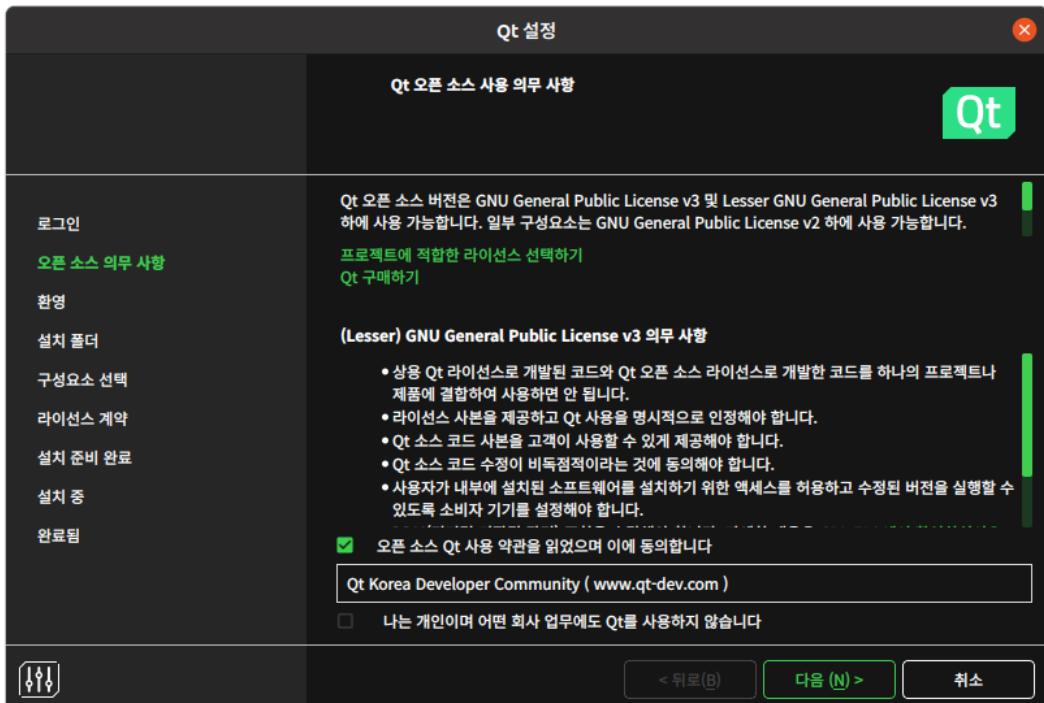
예수님은 당신을 사랑합니다.



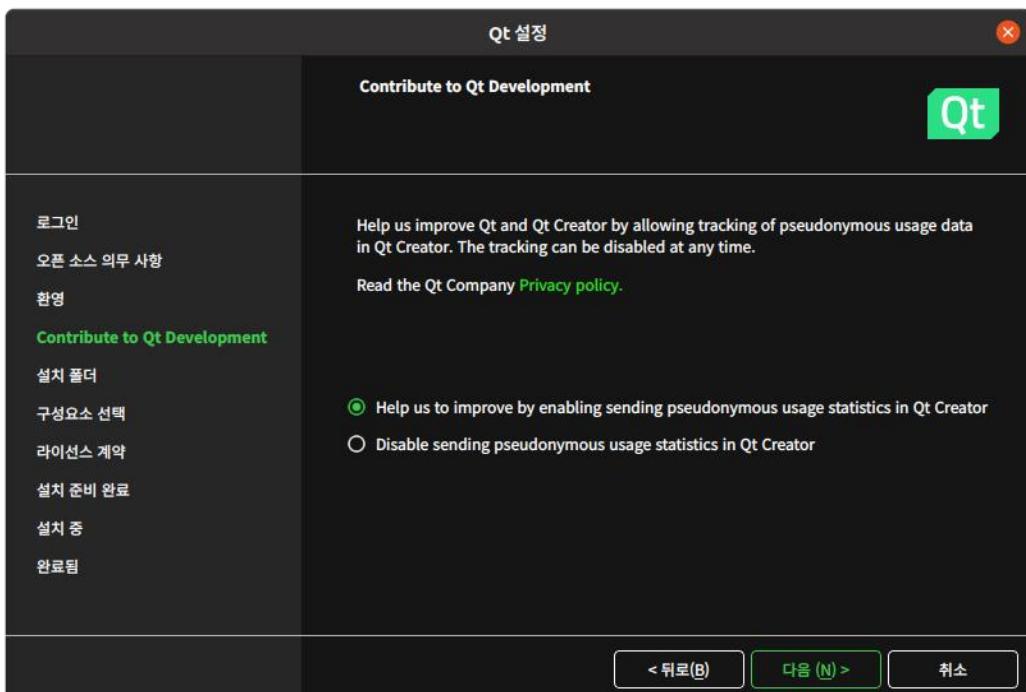
Qt 계정이 없다면 위의 화면에서 보는 것과 같이 [가입하기] 버튼을 클릭하면 회원가입을 할 수 있는 웹 페이지로 이동한다. 회원가입 후, 다시 다이얼로그로 돌아와 아이디와 비밀번호를 입력 후 [다음] 버튼을 클릭한다.



예수님은 당신을 사랑합니다.



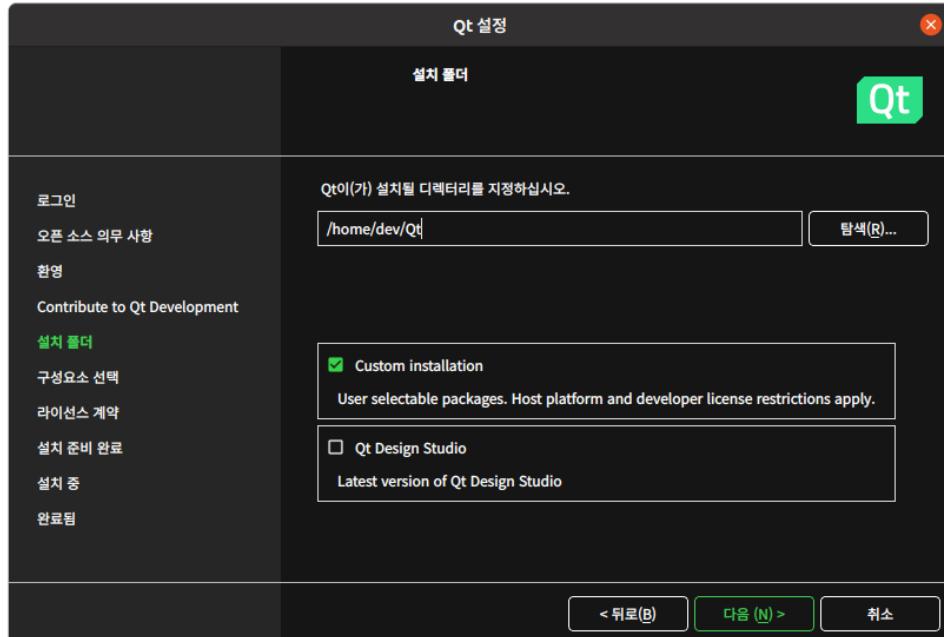
오픈 소스 Qt 사용 약관에 대한 내용을 확인할 수 있다. 위의 그림에서 보는 것과 같이 체크 박스를 활성화하고 [다음] 버튼을 클릭한다.



Qt Creator 와 같은 툴을 사용 중에 오류 등의 정보를 Qt 개발자에게 전송하는 것을

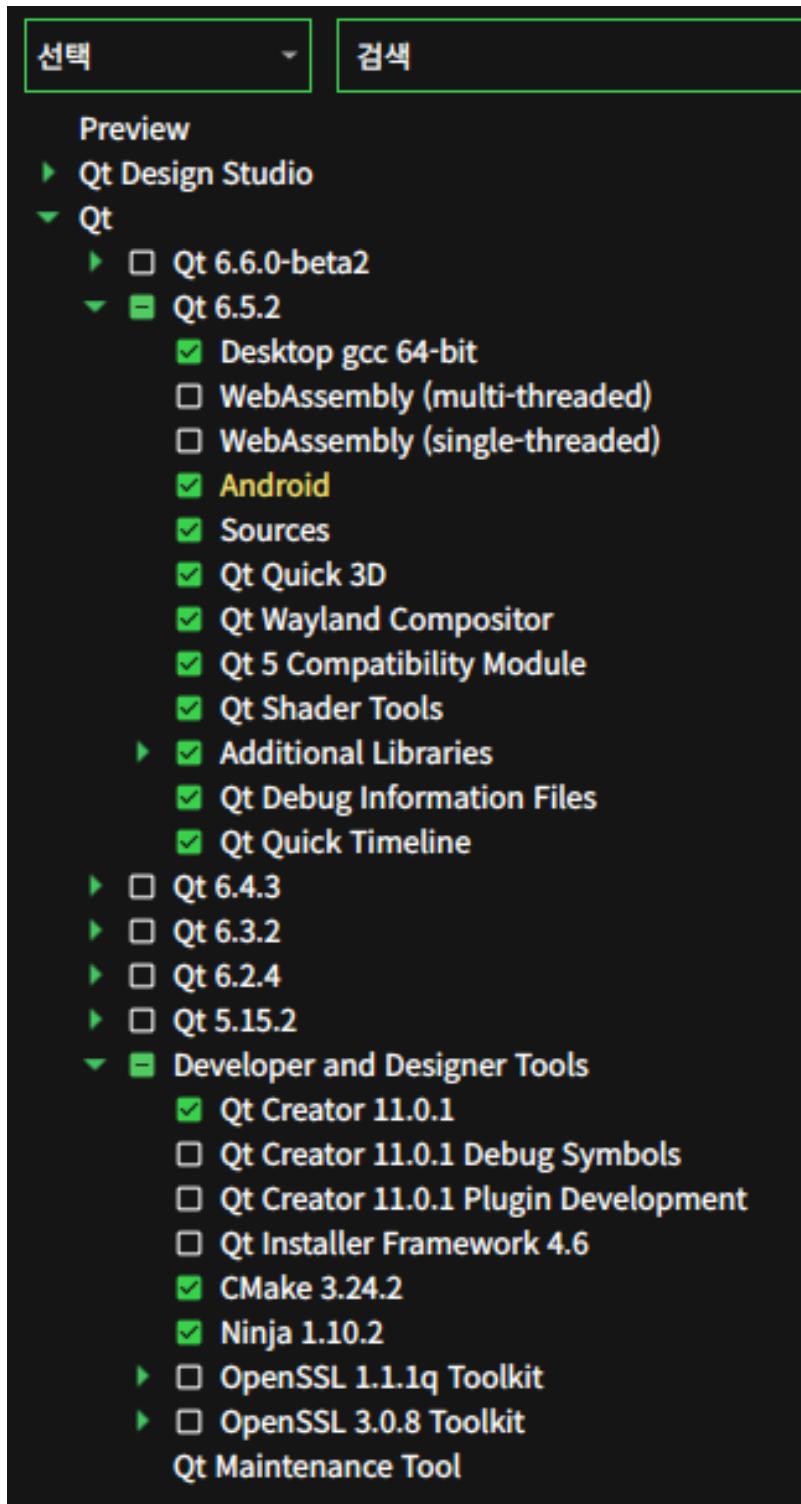
예수님은 당신을 사랑합니다.

동의 하겠냐는ダイ얼로그이다. 허용한다면 첫번째, 그렇지 않다면 두번째 항목을 선택하면 된다.



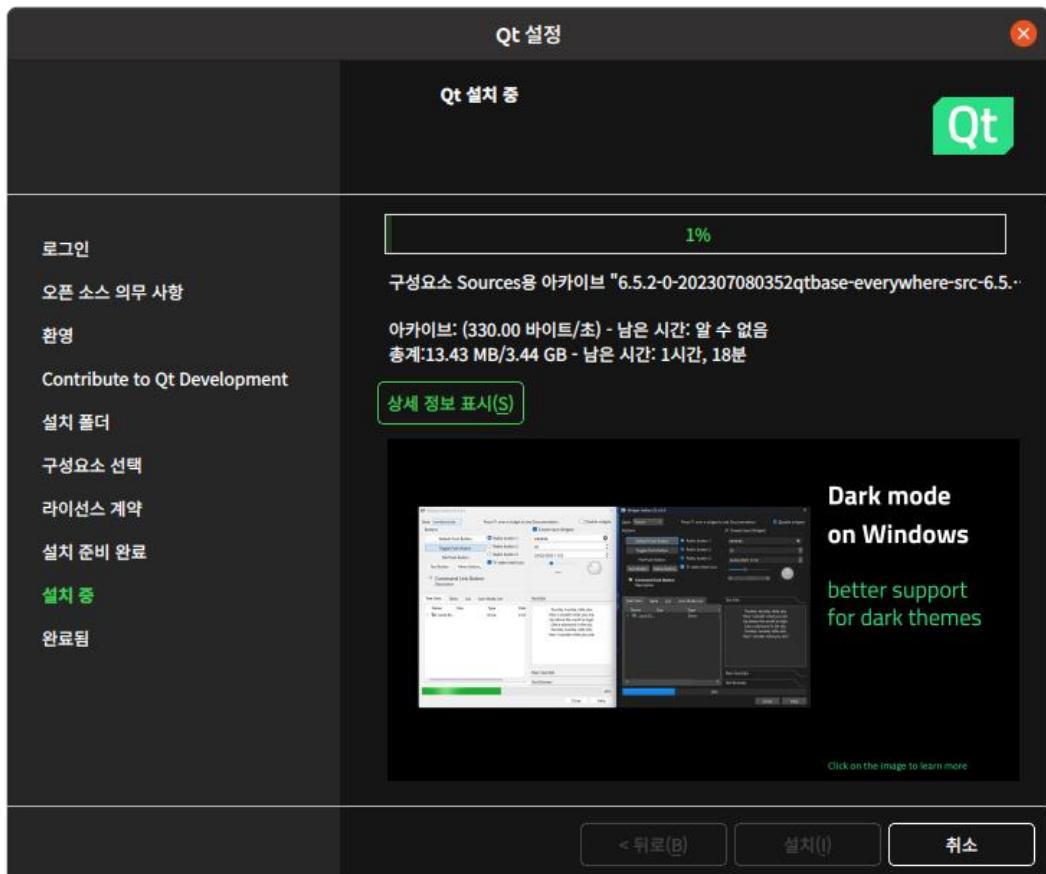
[Custom Install] 항목을 선택한다. 이 항목은 설치할 Component를 선택할 수 있다. 여기서는 위의 그림에서 보는 것과 같이 [Custom Install] 항목을 선택하고 [다음]을 클릭 한다.





위의 그림에서 보는 것과 같이 선택하고 [다음]을 클릭한다. 설치 항목의 설명은 이전에 MS Windows에서 설명한 항목을 참조한다.

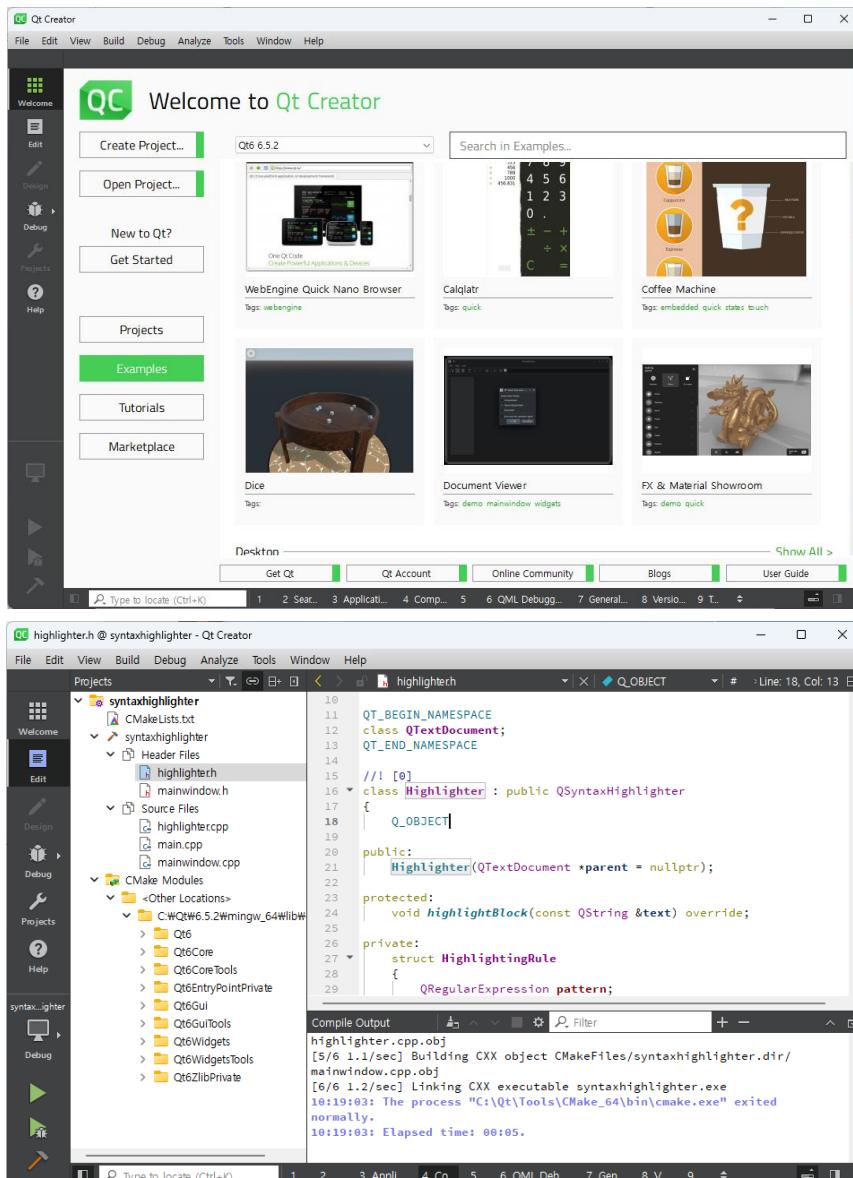
예수님은 당신을 사랑합니다.



2. Qt에서 제공하는 유용한 개발 툴

● Qt Creator IDE 툴

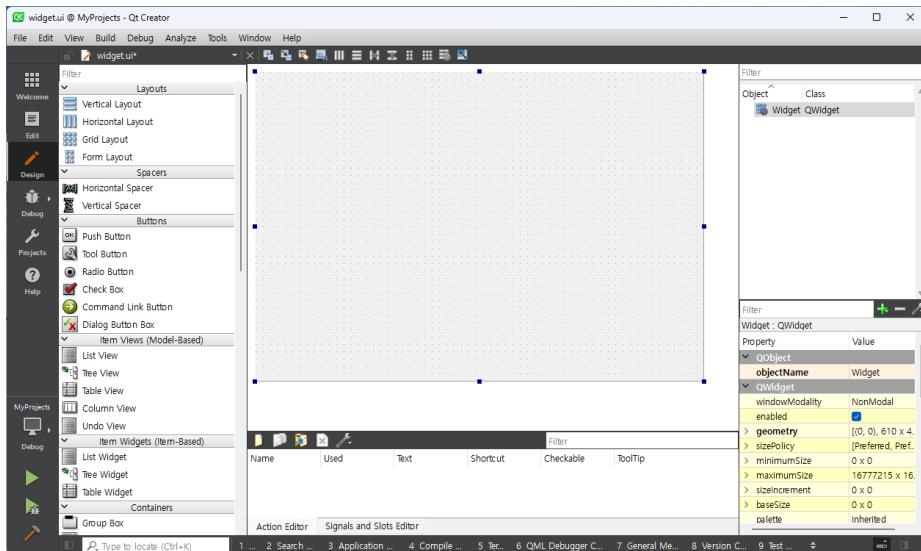
Qt는 소스코드를 작성하기 위한 툴로써 Qt Creator IDE 툴을 제공한다.



Qt Creator IDE 툴은 소스코드 작성 기능 및 디버깅 기능도 제공한다.

예수님은 당신을 사랑합니다.

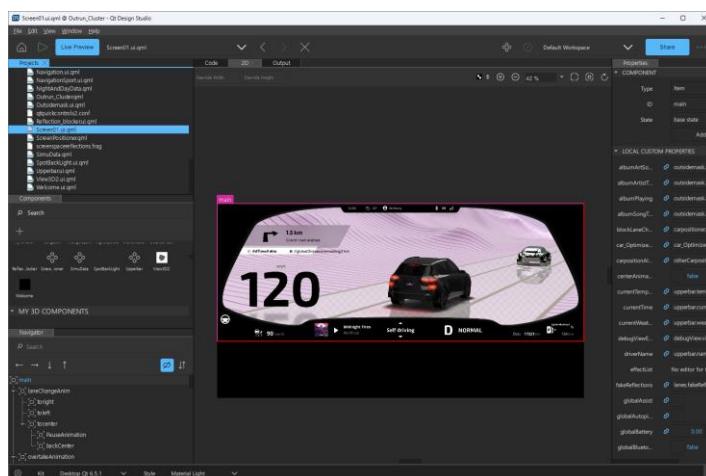
- Qt Designer 툴



Qt Designer 툴은 GUI를 쉽게 구현하기 위해서 제공하는 툴이다. 사용자가 원하는 GUI의 Widget들을 쉽게 배치할 수 있다. 이 툴은 Qt Creator에 통합되어 있다. 하지만 Qt Creator 대신에 Visual Studio를 사용할 수 있다. 따라서 Visual Studio 사용자를 위해서 독립적으로 실행할 수 있다.

- Qt Design Studio

Qt Design Studio 는 QML을 쉽게 작성할 수 있는 툴이다. Qt Designer와 같은 기능을 제공하지만 다른 점은 Qt Designer는 C++ 기반의 GUI를 제작하는 툴이며 Qt Design Studio 는 QML을 작성하는 툴이다.

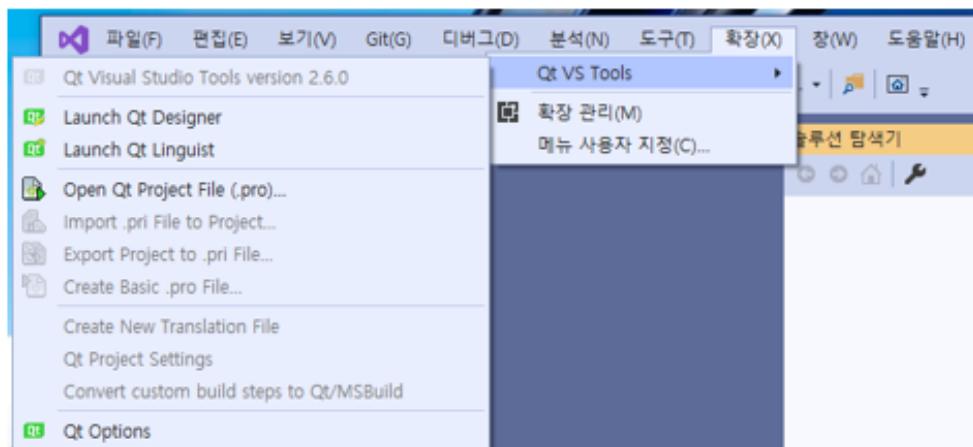
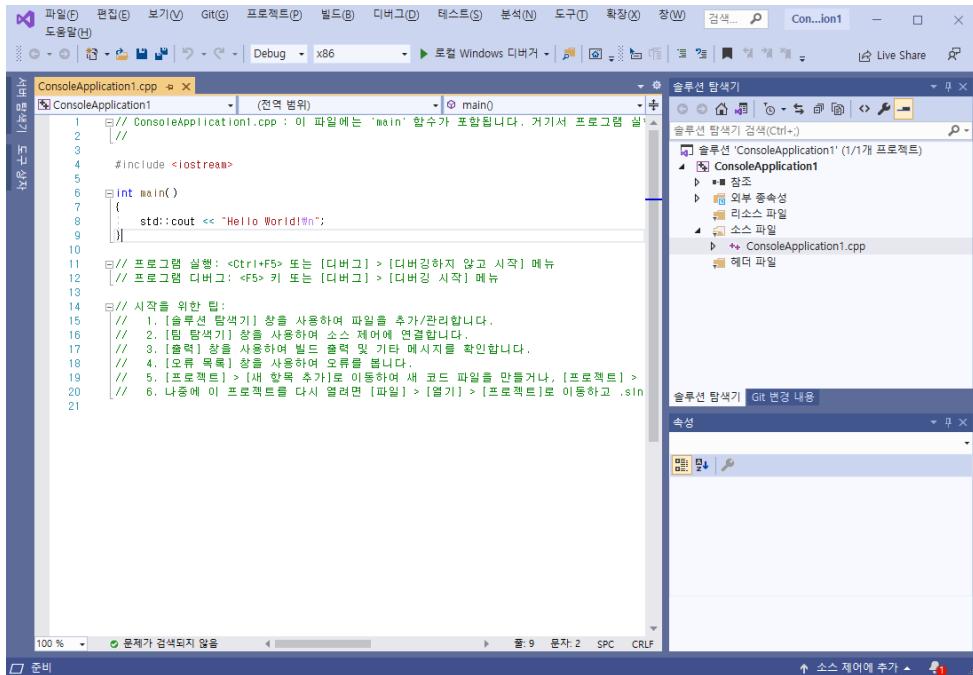


예수님은 당신을 사랑합니다.

● MS Visual Studio에서 Qt 개발

Qt Creator 이외에도 MS Visual Studio를 이용해 어플리케이션을 개발할 수 있다. Qt에서는 Add-in을 설치하면 된다. Add-in은 아래 URL에서 다운로드 받을 수 있다.

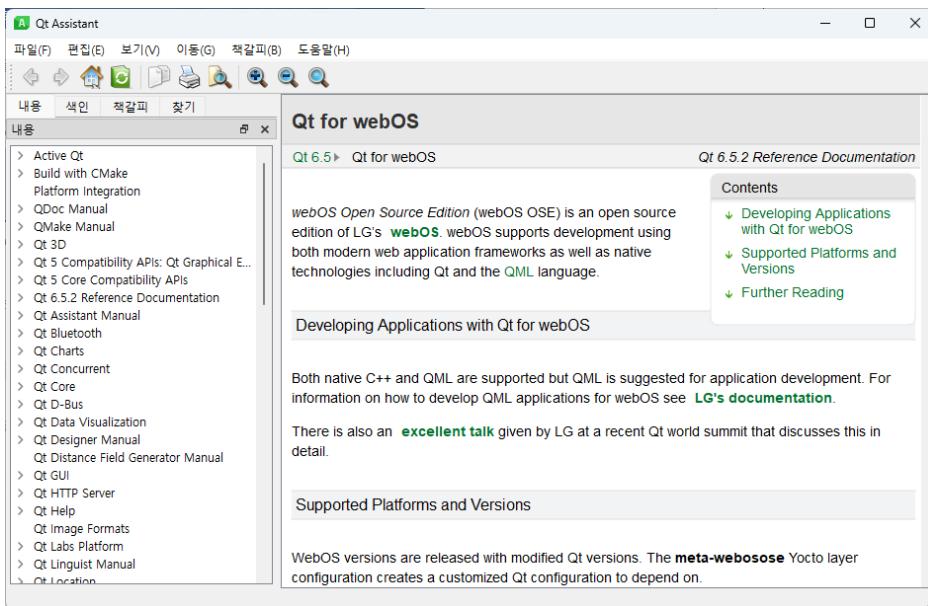
- ✓ 다운로드 주소: https://download.qt.io/official_releases/vsaddin/



● Qt Assistant 툴

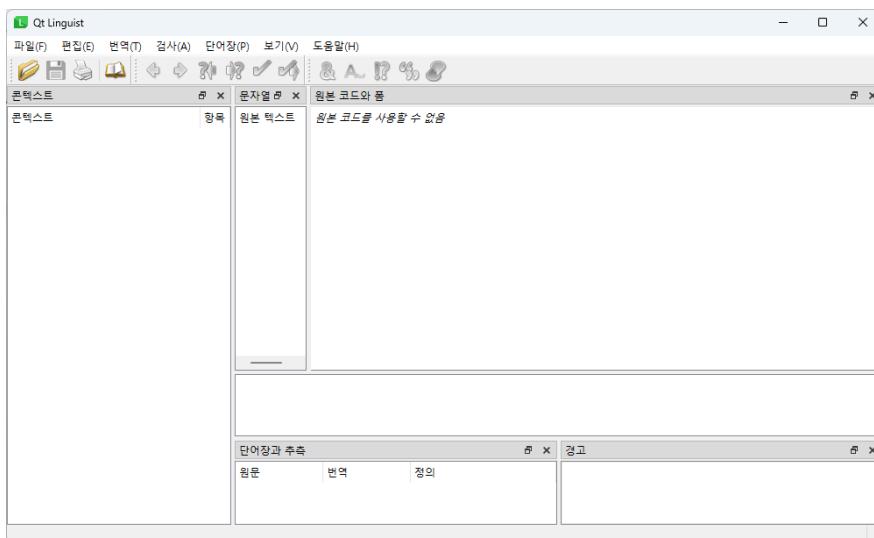
이 툴은 도움말을 제공하는 툴이다. Qt를 이용해 어플리케이션을 개발하다 보면 모르는 클래스 또는 QML을 검색해 볼 수 있다.

예수님은 당신을 사랑합니다.



● Qt Linguist 툴

이 툴은 다국어를 지원하기 위한 툴로써 어플리케이션에서 각 국가별로 사용하는 언어에 따라 GUI 다양한 언어를 표시할 수 있다.

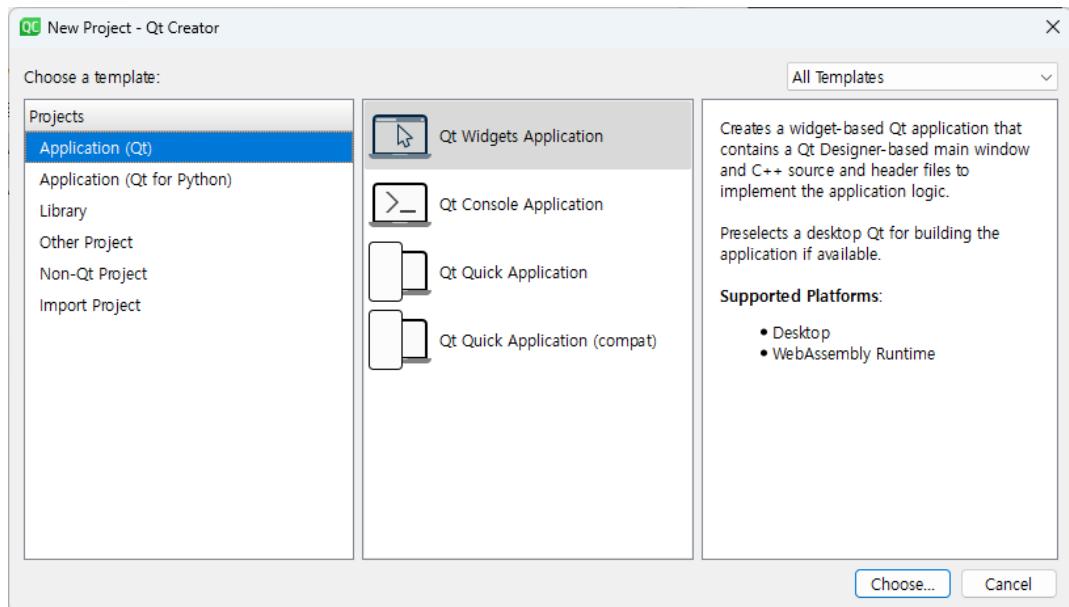


3. CMake 를 이용한 프로젝트 빌드

CMake 는 프로젝트를 쉽게 빌드하기 위해서 제공하는 툴이다 구현한 소스코드를 쉽게 빌드해 실행파일을 만드는 기능을 제공한다. CMake 는 빌드 도구이며, Qt 뿐만 아니라 Visual Studio 등 많은 개발 프레임워크에서 CMake 를 빌드 도구로 사용한다.

Qt 는 CMake 와 qmake 중 하나를 선택할 수 있다. Qt는 아직까지 qmake 를 많이 사용되고 있지만 점차 CMake 를 많이 사용하고 있는 추세이다.

Qt Creator 는 프로젝트 생성 시 자동으로 프로젝트 파일을 만들어 주는 기능이 포함되어 있다. 프로젝트 생성 시 CMake, qmake 그리고 qbs 중 하나를 선택할 수 있다.



위의 그림에서 보는 것과 같이 Qt Widget Application, Qt Console Application 응용 어플리케이션 기반 프로젝트 생성 시, 여러분은 CMake, qmake 그리고 qbs 중 하나를 선택할 수 있다.

하지만 Qt Creator IDE 툴의 버전이 10.x.x 버전부터 Qt Quick Application 을 프로젝트 선택 시, 자동으로 프로젝트 파일을 만들어 주는 기능은 CMake 만 지원된다.

그렇다고 해서 CMake 만 사용할 수 있는 것은 아니다. 프로젝트 생성 시, 프로젝트 파일이 자동 생성만 안될 뿐 프로젝트 파일을 수동으로 만들어 사용할 수 있다. 따라서 Qt Creator IDE 10.x.x 부터 자동 생성만 안될 뿐 qmake 를 여전히 사용할 수 있다.

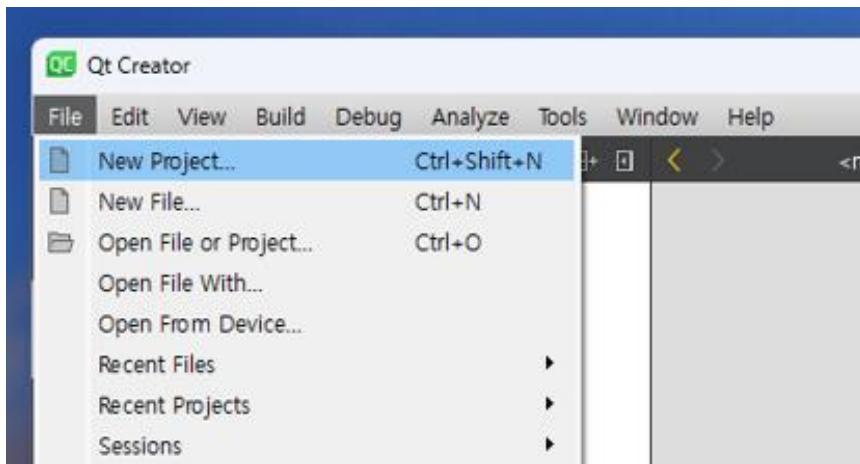
예수님은 당신을 사랑합니다.

이번 장에서 CMake를 이용해 Console 응용 어플리케이션을 빌드하는 방법과 GUI 어플리케이션을 빌드하는 방법에 대해서 알아보도록 하자.

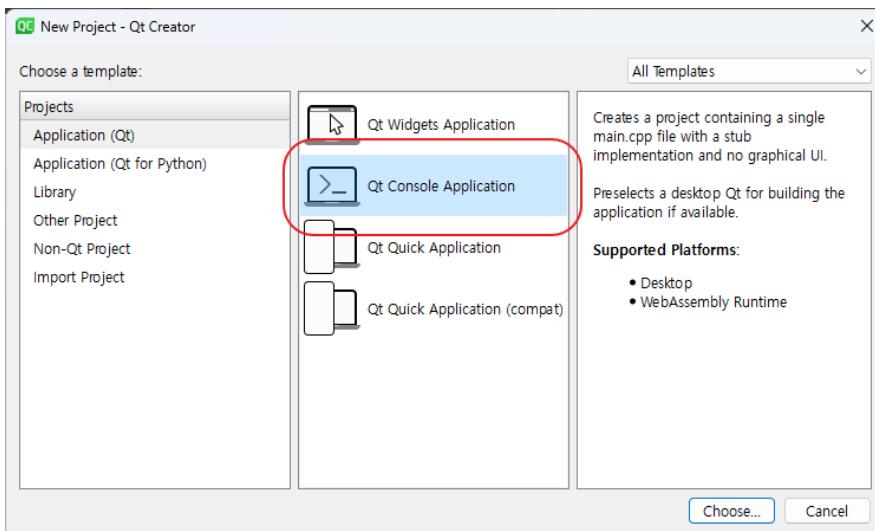
3.1. CMake를 이용한 Console 어플리케이션 구현

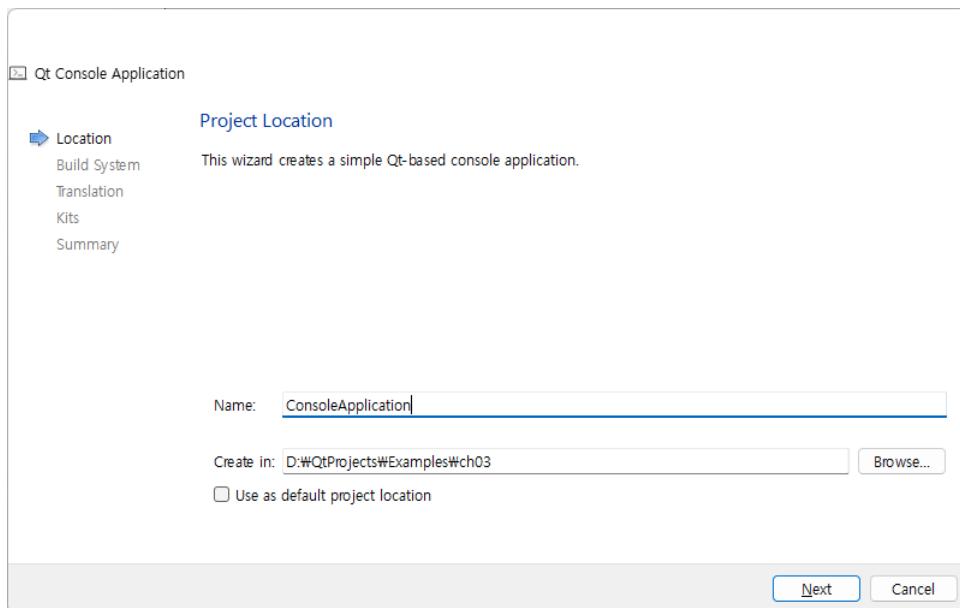
이번 장에서는 CMake를 이용해 Console 어플리케이션을 구현해 보도록 하자. 여기서 Console 어플리케이션은 GUI가 없는 어플리케이션을 의미한다. 예를 들어 Server Side 또는 Backend 와 비슷하다.

아래 그림에서 보는 것과 같이 프로젝트를 생성한다. 프로젝트를 생성하기 위해서는 Qt Creator 메뉴에서 [File] -> [New Project]를 선택한다. 단축키는 [Ctrl + Shift + N] 키를 누르면 된다.



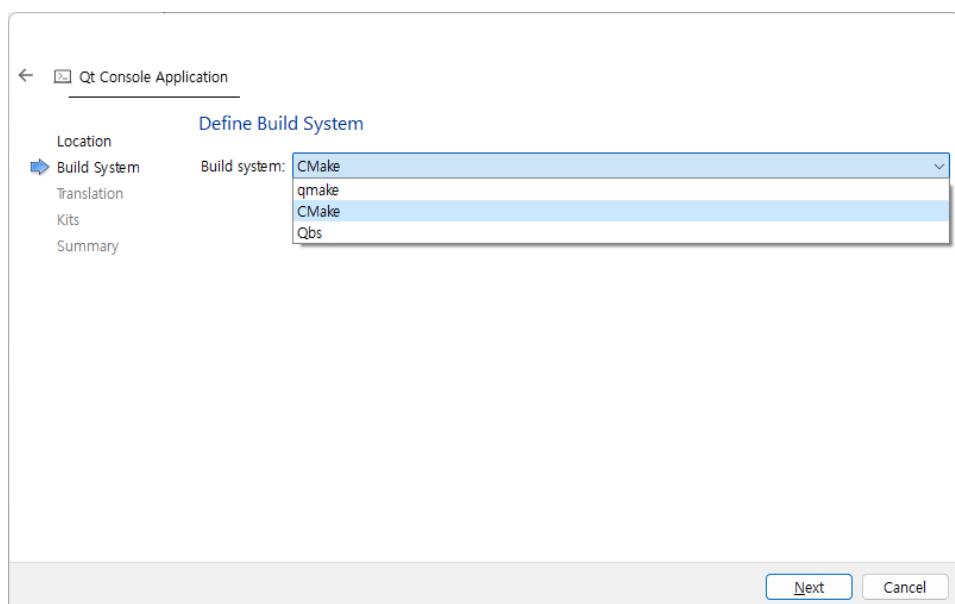
그리고 아래 그림에서 보는 것과 같이 다이얼로그 창에서 [Qt Console Application]를 선택한다.



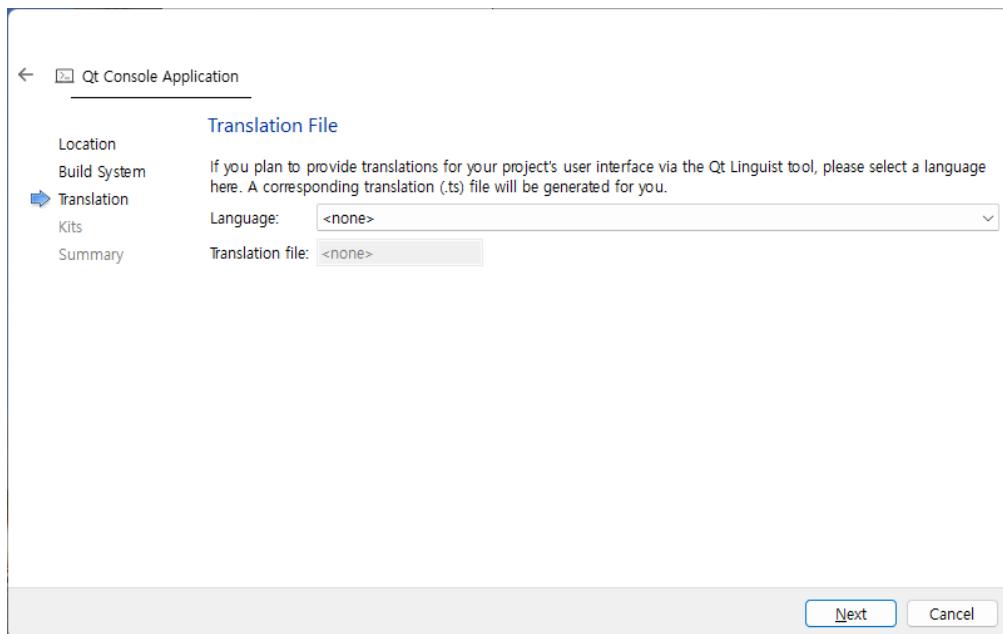


Name 항목에는 프로젝트 이름을 입력한다. Create In 항목은 프로젝트가 생성되는 위치를 말한다.

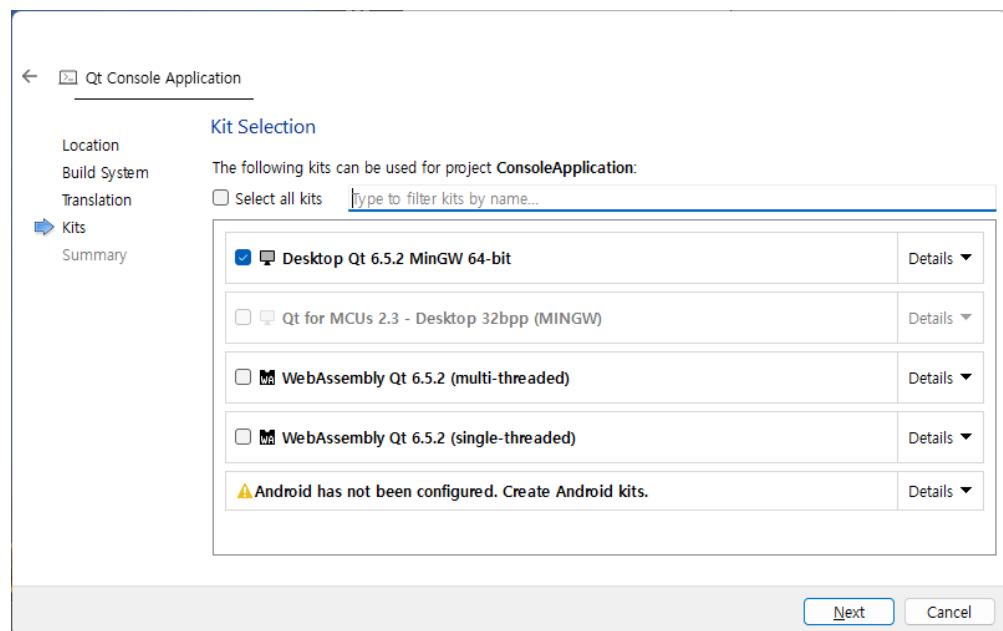
위의 그림에서 보는 것과 같이 프로젝트 이름과 프로젝트가 위치할 디렉토리를 지정한 다음 [Next] 버튼을 클릭한다.



다음으로 프로젝트 빌드 도구를 선택하는 다이얼로그이다. 위의 그림에서 보는 것과 같이 CMake를 선택하고 [Next] 버튼을 클릭한다.

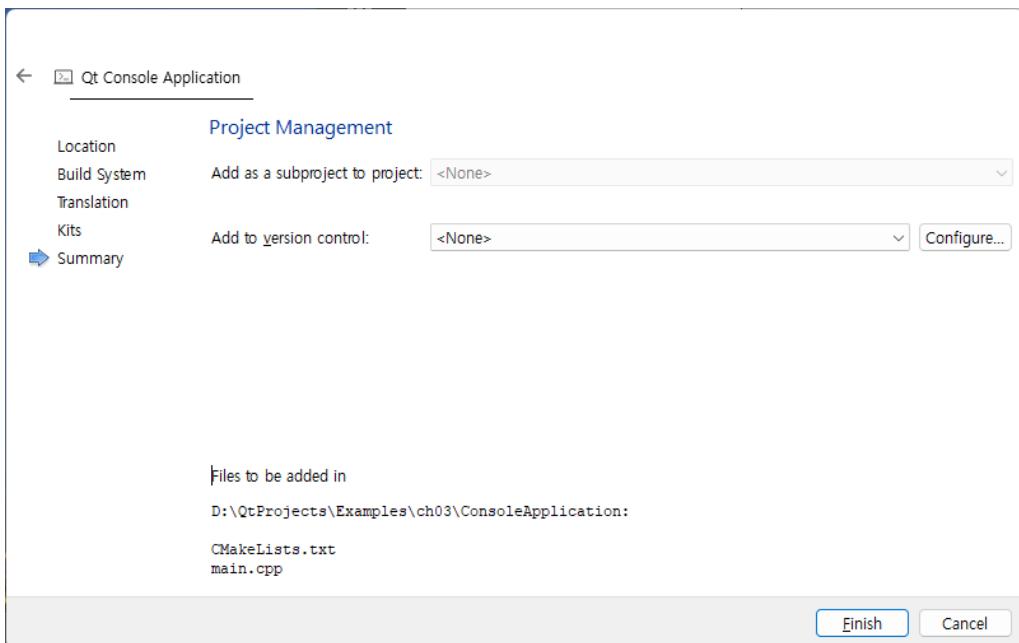


다국어 언어 지원을 위한 언어를 선택하는ダイ얼로그이다. 여기서는 위의 그림에서 보는 것과 같이 아무런 변경 없이 [Next] 버튼을 클릭한다.

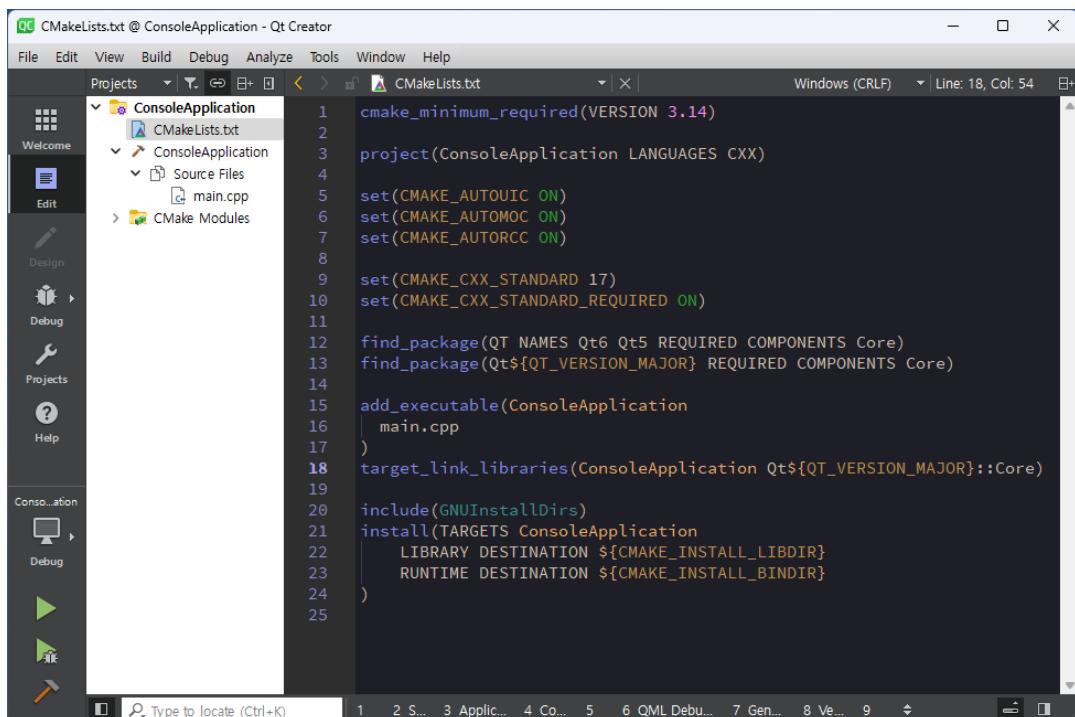


다음ダイ얼로그는 컴파일러를 선택하는ダイ얼로그이다. 위의 그림에서 보는 것과 같이 MinGW 컴파일러를 선택한 다음 [Next] 버튼을 클릭한다.

예수님은 당신을 사랑합니다.



다음은 소스코드 버전 관리 시스템을 선택하는ダイ얼로그이다. 디폴트로 두고 [Finish] 버튼을 클릭한다.



위의 그림에서 보는 것과 같이 프로젝트 생성이 완료되었으면 왼쪽의 프로젝트 탐색 창에서 CMakeList.txt 파일을 더블 클릭한다.

예수님은 당신을 사랑합니다.

CMakeList.txt 는 이 프로젝트의 빌드하기 위한 프로젝트 파일이다. CMakeList.txt 에 대해서 자세히 알아보도록 하자.

```
cmake_minimum_required(VERSION 3.14)

project(ConsoleApplication LANGUAGES CXX)

set(CMAKE_AUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt NAMES Qt6 Qt5 REQUIRED COMPONENTS Core)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Core)

add_executable(ConsoleApplication
    main.cpp
)
target_link_libraries(ConsoleApplication Qt${QT_VERSION_MAJOR}::Core)

include(GNUInstallDirs)
install(TARGETS ConsoleApplication
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

위의 내용은 CMakeList.txt 파일의 내용이다. 첫 번째 라인은 CMake에서 버전이 존재한다. 여기서는 CMake를 명시한다. 여기서는 최소 3.14 버전 이상을 요구한다.

```
project(ConsoleApplication LANGUAGES CXX)
```

프로젝트의 이름과 사용된 언어가 C++ 언어임을 명시한다.

```
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
```

첫 번째 라인은 C++ 17 버전을 이상을 사용하겠다는 뜻이다. 두 번째 라인은 만약 C++ 버전이 너무 오래된 컴파일이면 오류를 출력하라는 뜻이다.

```
find_package(Qt NAMES Qt6 Qt5 REQUIRED COMPONENTS Core)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Core)
```

예수님은 당신을 사랑합니다.

```
target_link_libraries(ConsoleApplication Qt${QT_VERSION_MAJOR}::Core)
```

위의 소스코드는 Qt에서 사용할 모듈이 명시한다. Qt는 다양한 모듈로 구성되어 있다. 예를 들어 프로젝트에서 Network 모듈(또는 API)를 사용하기 위해서는 아래와 같이 사용하면 된다.

```
find_package(Qt6 REQUIRED COMPONENTS Network)
target_link_libraries(mytarget PRIVATE Qt6::Network)
```

그리고 아래와 같이 add_executable() 함수는 프로젝트에서 추가할 소스코드 파일을 명시하면 된다.

```
add_executable(ConsoleApplication
    main.cpp
)
```

- main.cpp 소스코드 작성 및 빌드 후 실행하기

다음으로 Qt Creator의 왼쪽 프로젝트 창에서 main.cpp 소스코드를 선택하고 아래와 같이 소스코드를 추가한다.

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

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

    qDebug("Hello world.");

    return a.exec();
}
```

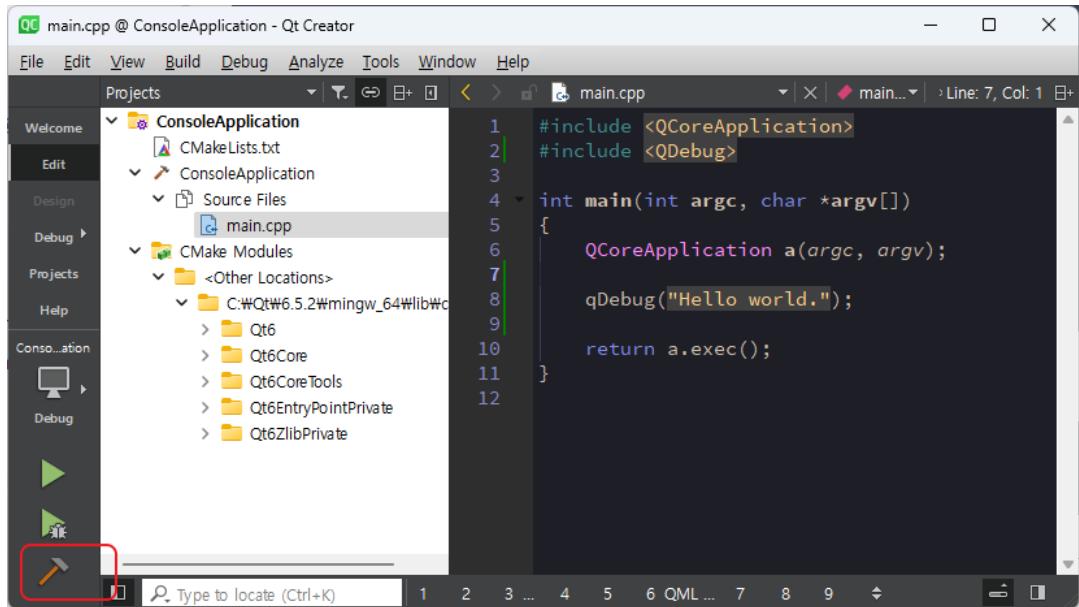
첫 번째 QCoreApplication a(argc, argv) 는 필수로 main() 함수의 인자인 argc, argv 를 넘겨줘야 한다. 지금 실행하는 응용 어플리케이션이 Console 이므로 QCoreApplication 이지만 만약, GUI 기반의 어플리케이션인 경우 QGuiApplication를 사용해야 한다.

qDebug() 는 메시지를 출력한다. 따라서 Console 창에 Hello world.를 출력한다. 마지막 라인은 프로그램이 종료되지 않도록 해주는 역할을 한다.

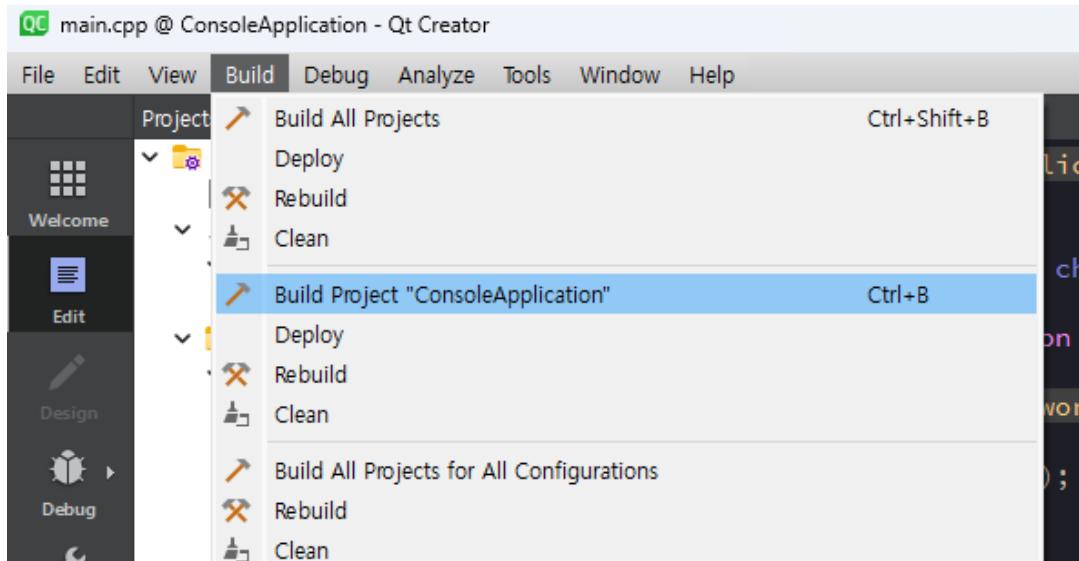
위와 같이 소스코드 작성을 완료하였으면 프로젝트를 빌드하고 실행해보도록 하자. 빌드하기 위해서는 단축키 Ctrl + B 키를 누르면 된다. 또는 Qt Creator에서 망치 모양의

예수님은 당신을 사랑합니다.

아이콘을 클릭하면 된다.



또는 메뉴에서 [Build] -> [Build Project "프로젝트명"] 을 클릭하면 된다.



빌드가 완료되면 단축키 Ctrl + R 또는 Qt Creator에서 빌드 아이콘 위에 위치한 화살표 모양의 아이콘을 실행하면 된다. 또는 메뉴에서 [Build] -> [Run] 을 실행하면 된다. 실행하면 아래 그림에서 보는 것과 같이 실행되는 것을 확인할 수 있다.

아래 그림에서와 같이 Console 창에서 메시지를 확인하기 위해서는 Qt Creator 하단의 [3 Application] 을 클릭하면 Console 창에 출력되는 메시지를 확인할 수 있다.

The screenshot shows the Qt Creator IDE interface. The top menu bar includes 'File', 'Edit', 'Debug', 'Analyze', 'Tools', 'Window', and 'Help'. The left sidebar displays project files: 'roleApplication', 'MakeLists.txt', 'onsoleApplication', 'Source Files' (with 'main.cpp' selected), 'Make Modules', and 'Qt6' (under '<Other Locations>'). The main code editor window shows the following C++ code:

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

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

    qDebug("Hello world.");

    return a.exec();
}
```

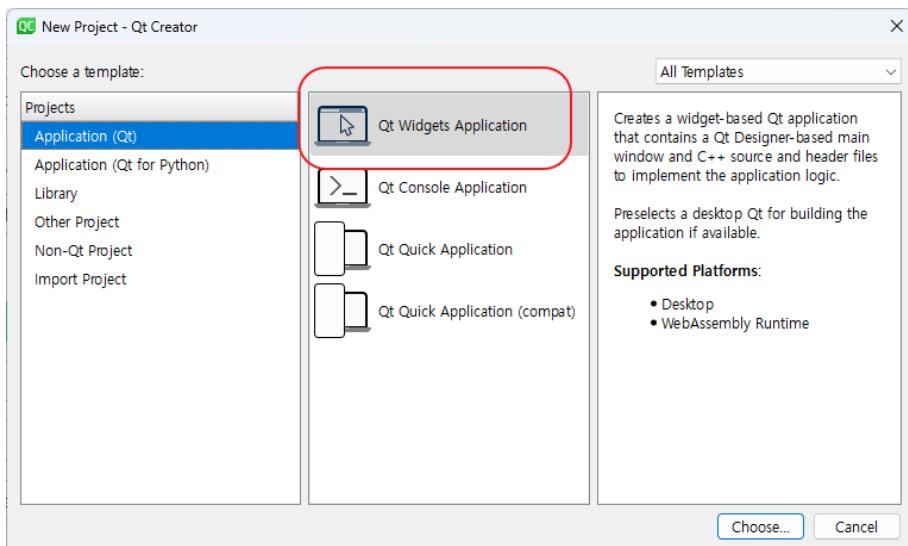
The bottom pane shows the 'Application Output' tab, which displays the terminal output of the application's execution. The output reads:

```
14:10:01: Starting D:\QtProjects\Examples\ch03\bu
Desktop_Qt_6_5_2_MinGW_64_bit-Debug\ConsoleApplic
Hello world.|
```

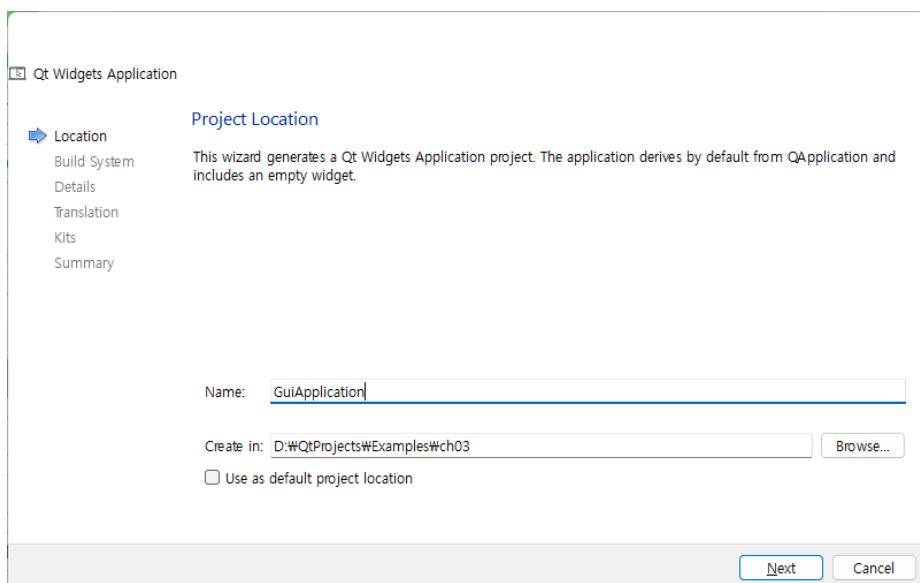
A red circle highlights the 'Hello world.' text in the output, and a red arrow points from this circle to the '3 Application...' tab in the bottom navigation bar, which is also highlighted with a red border.

3.2. CMake 를 이용한 GUI 어플리케이션 구현

이번에는 프로젝트 생성 시 GUI 기반의 어플리케이션을 생성해 보도록 하자. 프로젝트 생성 다이얼로그가 실행 되면 아래 그림에서 보는 것과 같이 Qt Widget Application 항목을 선택하고 아래에 [Chose] 버튼을 클릭한다.

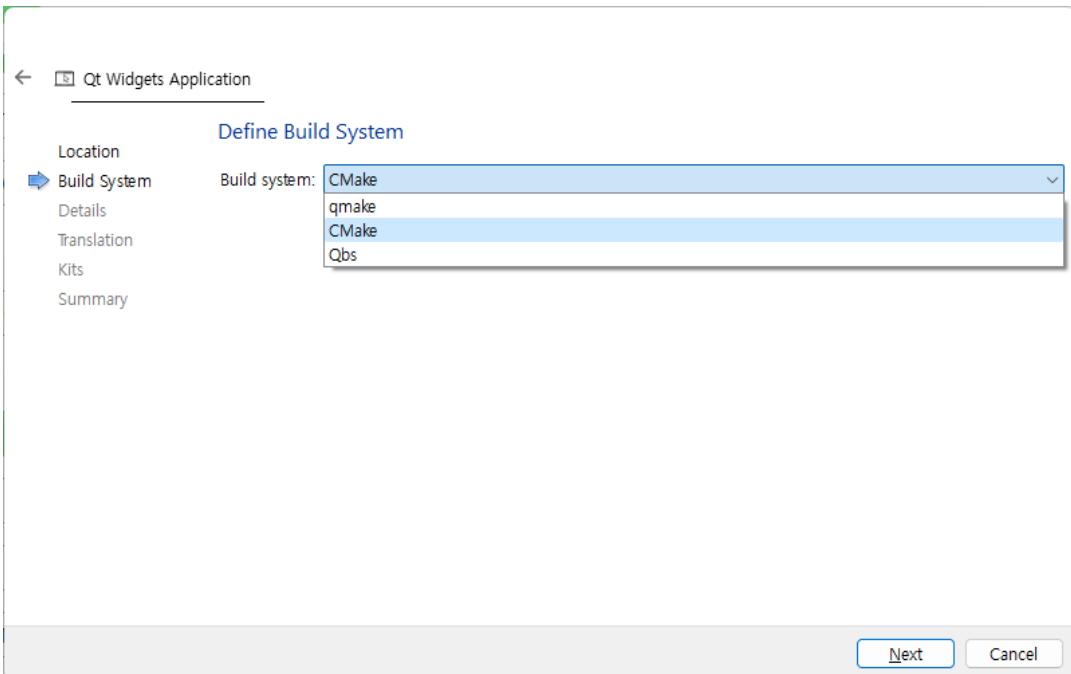


다음으로 프로젝트 이름과 프로젝트가 위치할 디렉토리를 지정한다.

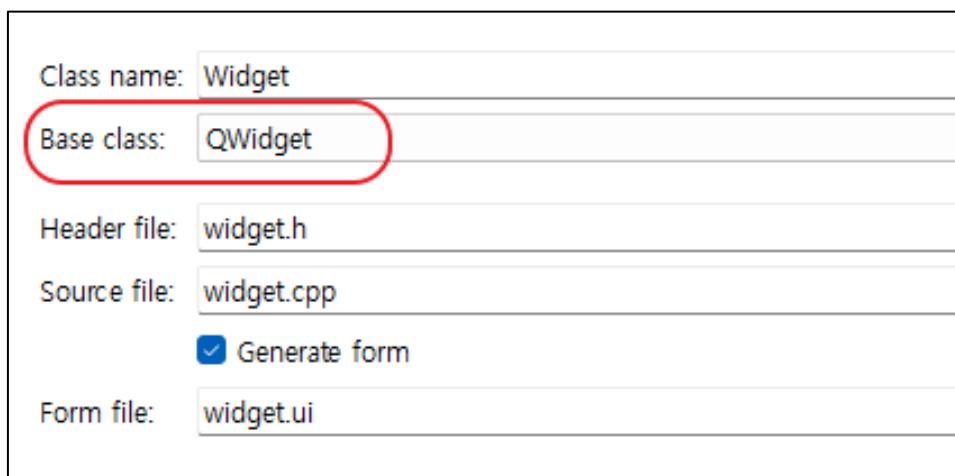


다음에는 프로젝트 빌드 도구로 CMake를 선택한다.

예수님은 당신을 사랑합니다.

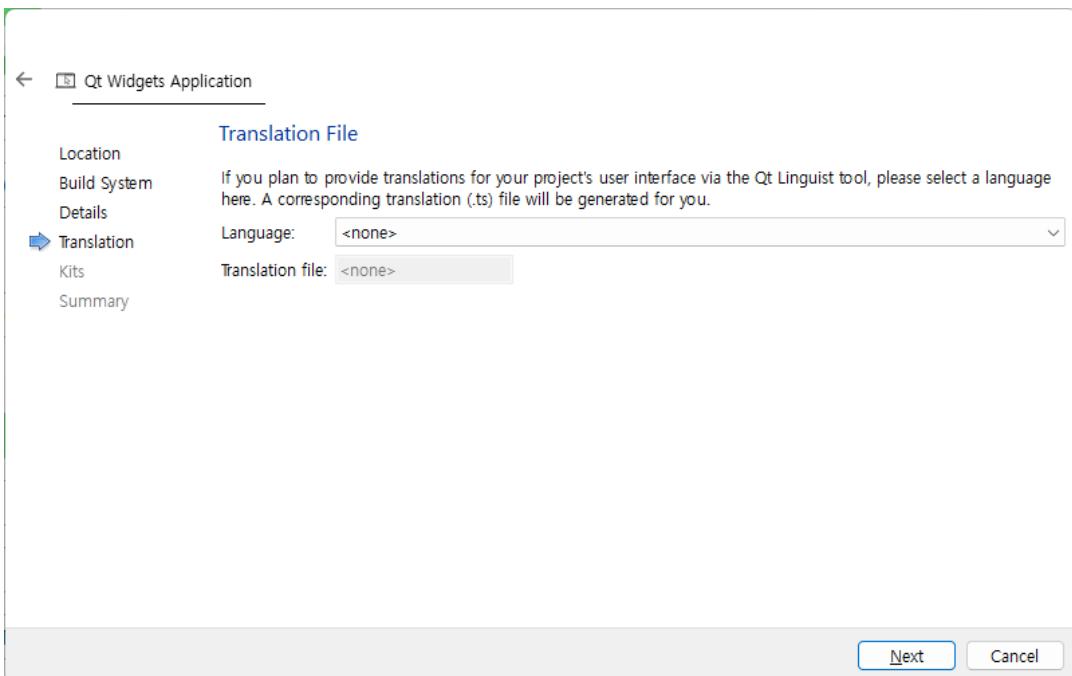


다음으로 아래 그림에서 보는 것과 같이 Base Class 로 QWidget 을 선택하고 [Next] 버튼을 클릭한다.

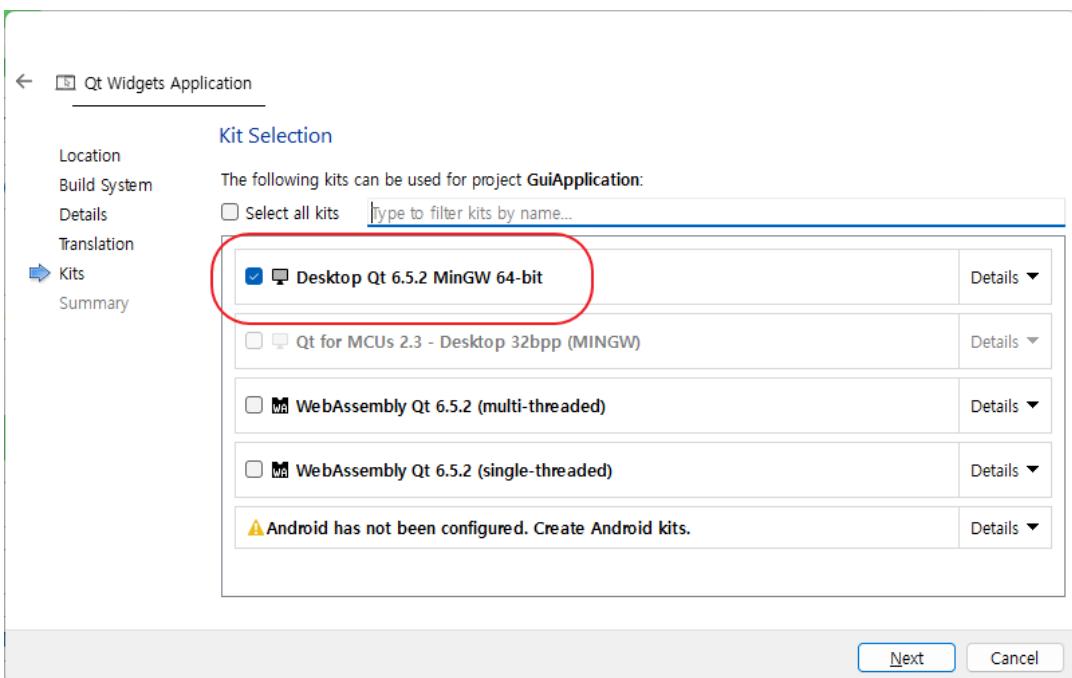


다음으로 다국어를 지원하는 언어를 선택하는ダイ얼로그이 이다. 여기서는 디폴트로 놓고 [Next] 버튼을 클릭한다.

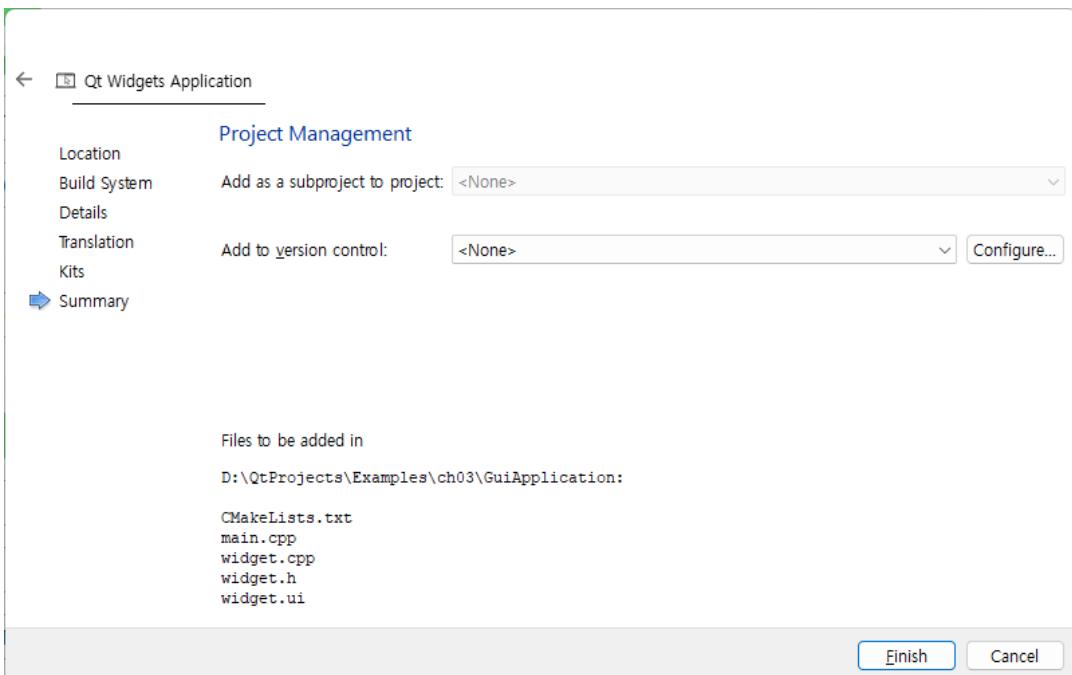
예수님은 당신을 사랑합니다.



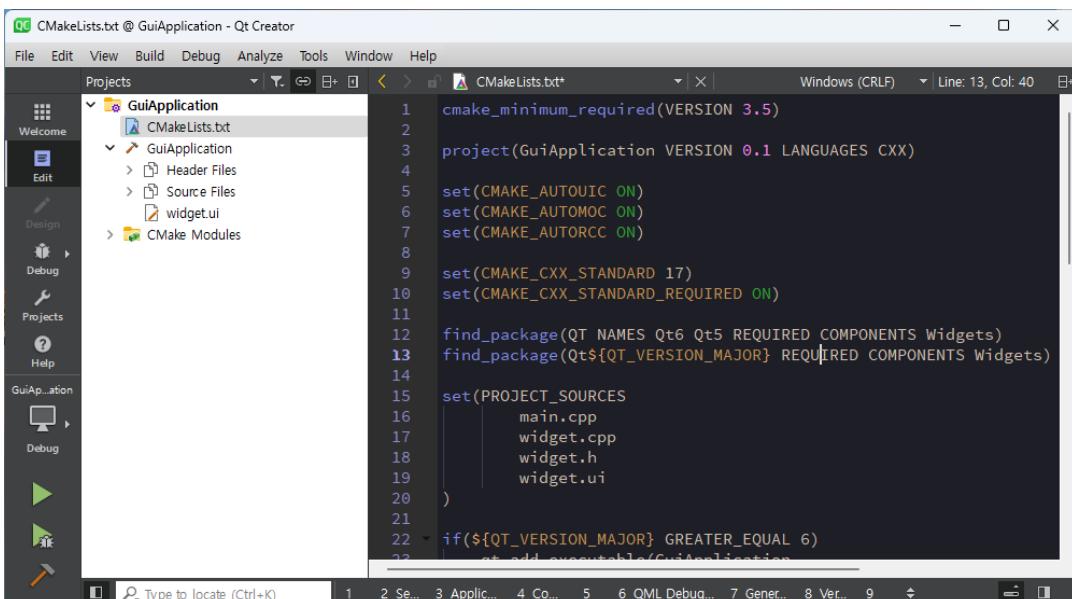
다음 다이얼로그에서는 컴파일러를 선택하는 화면이다. 컴파일러를 선택하는 화면에서는 MinGW 컴파일러를 선택하고 [Next] 버튼을 클릭한다.



예수님은 당신을 사랑합니다.



위의 그림에서 보는 것과 같이 <None> 으로 놓은 상태에서 아래 [Finish] 버튼을 클릭 한다. [Finish] 버튼을 클릭하면 프로젝트가 생성되며 Qt Creator 의 왼쪽 탐색 창에서 CmakeList.txt 파일을 클릭한다.



CMakeList.txt 파일이 이전의 Console 보다 조금 복잡해 보인다. 한 줄씩 어떤 명령인지 살펴보도록 하자.

```
project(GuiApplication VERSION 0.1 LANGUAGES CXX)
```

예수님은 당신을 사랑합니다.

Console 에서는 버전 정보가 없지만 위에서 보는 것과 같이 버전 정보를 직접 명시할 수 있다.

```
set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)
```

CMAKE_AUTOUIC 는 Qt 의 uic(Qt User Interface Compiler) 파일을 자동으로 생성하라는 문구이다.

예를 들어 이 문구가 없으면 Qt 가 자동으로 생성해주는 파일이 있는데 그 중에서 확장자가 .ui 인 XML 포맷의 UI 파일의 소스코드를 자동의 생성해 주지 않는다.

여기서는 widget.ui 라는 uic 를 사용하였다. 따라서 빌드하게 되면 ui_widget.h 파일이 자동으로 생성되는데, CMAKE_AUTOUIC 가 없으면 ui_widget.h 파일이 자동으로 생성되지 않는다.

CMAKE_AUTOMOC 는 moc(Meta Object Compiler) 파일을 자동으로 생성해준다. 그리고 CMAKE_AUTORCC 는 리소스파일을 자동으로 생성해준다.

```
if(${QT_VERSION_MAJOR} GREATER_EQUAL 6)
    qt_add_executable(GuiApplication
        MANUAL_FINALIZATION
        ${PROJECT_SOURCES}
    )
else()
    if(ANDROID)
        add_library(GuiApplication SHARED
            ${PROJECT_SOURCES}
        )
    else()
        add_executable(GuiApplication
            ${PROJECT_SOURCES}
        )
    endif()
endif()
```

CMake 에서는 if 문을 사용할 수 있다. 위의 소스코드 는 Qt 버전이 6 이상인 경우 괄호 안에 내용을 실행하라는 문구이며 그렇지 않고, 만약 플랫폼이 Android 이면 괄호 안에 내용을 실행한다. 그렇지 않으면 else() 의 괄호 안에 내용을 실행하라는 뜻이다.

add_library() 는 라이브러리를 만들기 위해서 사용한다.

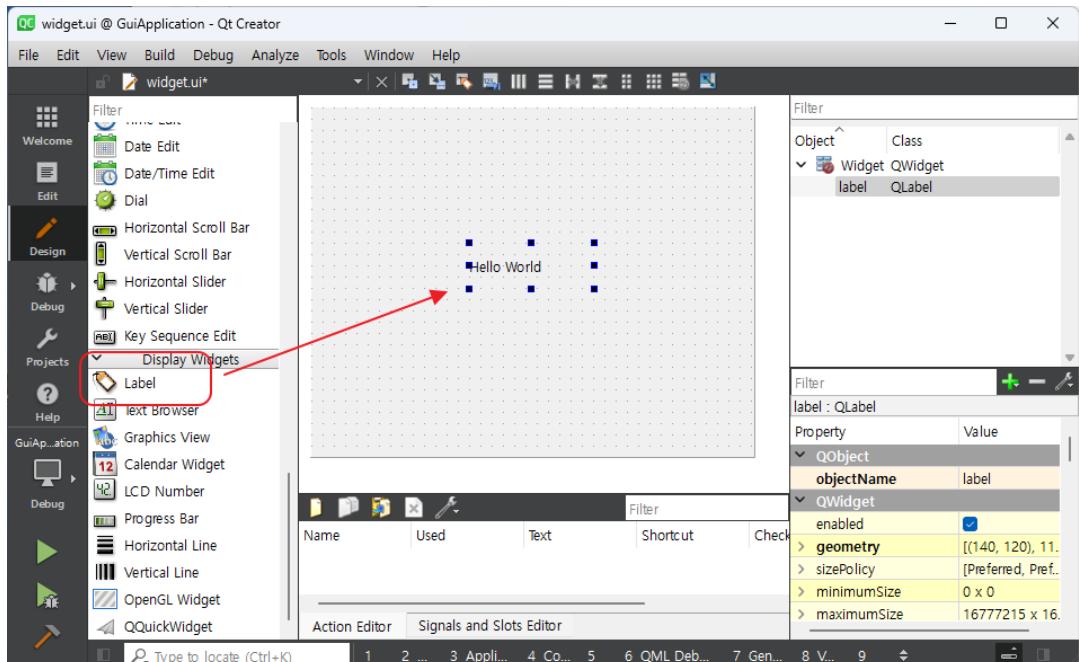
예수님은 당신을 사랑합니다.

예를 들어 프로젝트를 생성할 때 실행할 수 있는 응용 어플리케이션 인지, 라이브러리 인지 정의 할 수 있다. 여기서 `add_library()` 는 라이브러리 프로젝트를 만들 때 사용한다. 그리고 `add_executable()` 함수는 실행 가능한 어플리케이션을 만들 때 사용한다.

● ui 파일을 이용한 GUI 위젯 배치

프로젝트 탐색 창에 보면 확장자가 .ui 인 파일이 생성된 것을 확인 할 수 있을 것이다. 이 파일은 GUI 위젯을 마우스로 원하는 위치를 쉽게 배치할 수 있는 기능을 배치 할 수 있다. 예제로 QLabel 위젯을 배치하고 빌드 후 실행 보도록 하자.

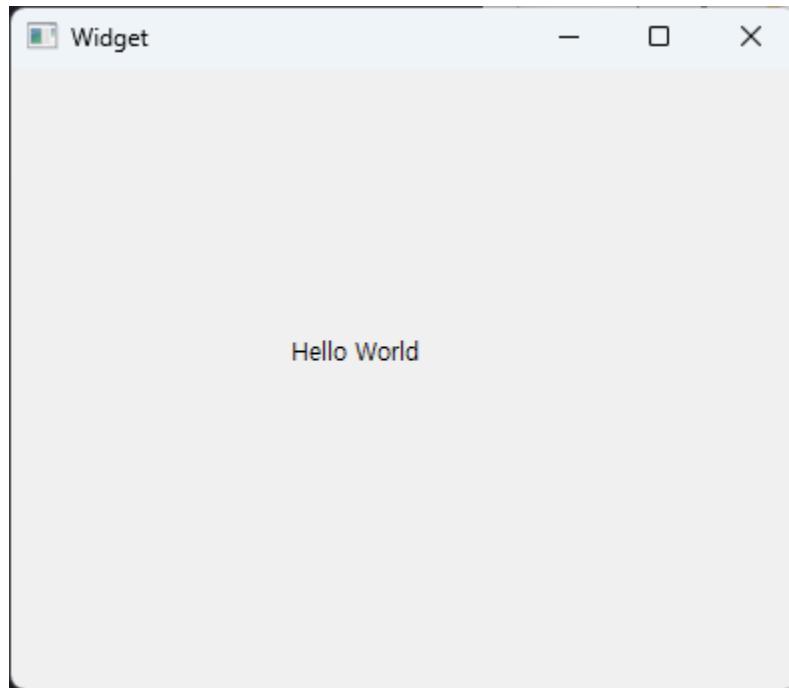
widget.ui 파일을 더블 클릭하면 아래 그림에서 보는 것과 같은 화면을 볼 수 있다. 아래 화면에서 QLabel 을 배치하고 그 위젯에 내용을 Hello World 로 바꾼 후 실행해 보도록 하자.



다시 소스코드로 돌아 가기 위해서는 <Esc> 키를 누르면 GUI 위젯을 배치할 수 있는 Designer 가 닫힌다. 이 방법 이외에도 Qt Creator 좌측에 [Edit] 아이콘을 누르면 소스 코드 창으로 돌아 갈 수 있다.

위와 같이 배치했으면 빌드 실행해 보도록 하자. 실행하면 아래 그림에서 보는 것과 같이 실행되는 것을 확인할 수 있다.

예수님은 당신을 사랑합니다.



지금까지 GUI 를 UI Designer를 이용했다. 확장자가 ui 인 파일을 사용할 수도 있지만 ui 파일을 사용하지 않고도 직접 소스코드를 작성하는 방식으로 GUI 코드를 작성할 수 있다.

- Widget 클래스

main.cpp 에서 Widget 클래스 인스턴스를 생성하고 실행한다.

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

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Widget w;
    w.show();
    return a.exec();
}
```

Widget 클래스의 헤더 파일에 보면 private 키워드에 ui 인스턴스가 있는 것을 확인 할 수 있을 것이다.

```
#ifndef WIDGET_H
```

예수님은 당신을 사랑합니다.

```
#define WIDGET_H

#include <QWidget>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
};

#endif // WIDGET_H
```

ui_widget.h 는 uic 가 확장자가 .ui 인 파일을 기반으로 자동으로 소스코드로 변환해 ui_widget.h 파일을 만들어 준다.

여기서 단순히 QLabel 을 사용했지만 나중에 복잡한 GUI 를 만들고 해당 이벤트를 처리해주면 된다.

예를 들어 GUI 상에서 버튼을 배치하고 버튼을 처리하는 함수를 Widget 클래스 상에서 처리하는 함수를 만들면 된다.

4. qmake 를 이용한 프로젝트 빌드

Qt 에서 제공하는 qmake 는 프로젝트 빌드하기 위해서 제공하는 툴이다. 프로젝트에는 소스코드 외에도 이미지 파일, 설정파일, uic, moc 등 파일들이 있다. 이러한 파일들을 이용해 쉽게 빌드할 수 있는 툴이라고 보면 된다.

그리고 Debug 모드로 빌드 할 때 또는 Release 모드로 빌드할 때 어떤 특징을 조건으로 설정할 수 있다.

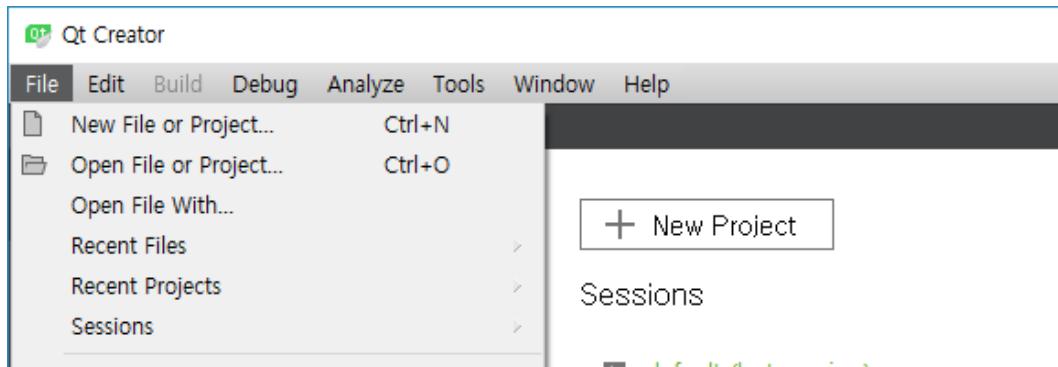
qmake 는 확장자가 .pro 이며 파일명은 프로젝트 이름을 사용한다. CMake 의 프로젝트 파일명은 CMakeList.txt 이지만 qmake 의 프로젝트 파일 이름은 [프로젝트 명.pro] 이 된다. 예를 들어 프로젝트명이 MyApp 이면 프로젝트 명은 MyApp.pro 가 된다.

qmake 는 지금까지 가장 많이 사용된 빌드 도구 이다. 최근 들어 CMake 를 사용하는 추세이기는 하지만 아직도 많은 프로젝트에서 qmake를 주로 사용되고 있으며, 최근에도 많이 사용되는 빌드 도구이다.

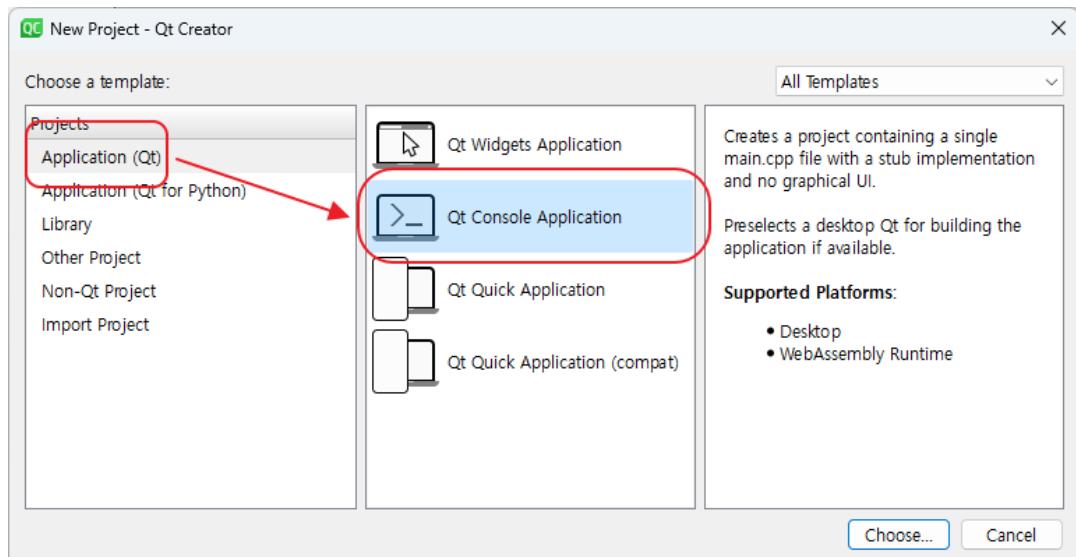
이번 장에서 qmake를 이용해 Console 어플리케이션과 GUI 어플리케이션을 개발하는 방법에 대해서 알아보도록 하자.

4.1. qmake를 이용한 Console 어플리케이션 구현

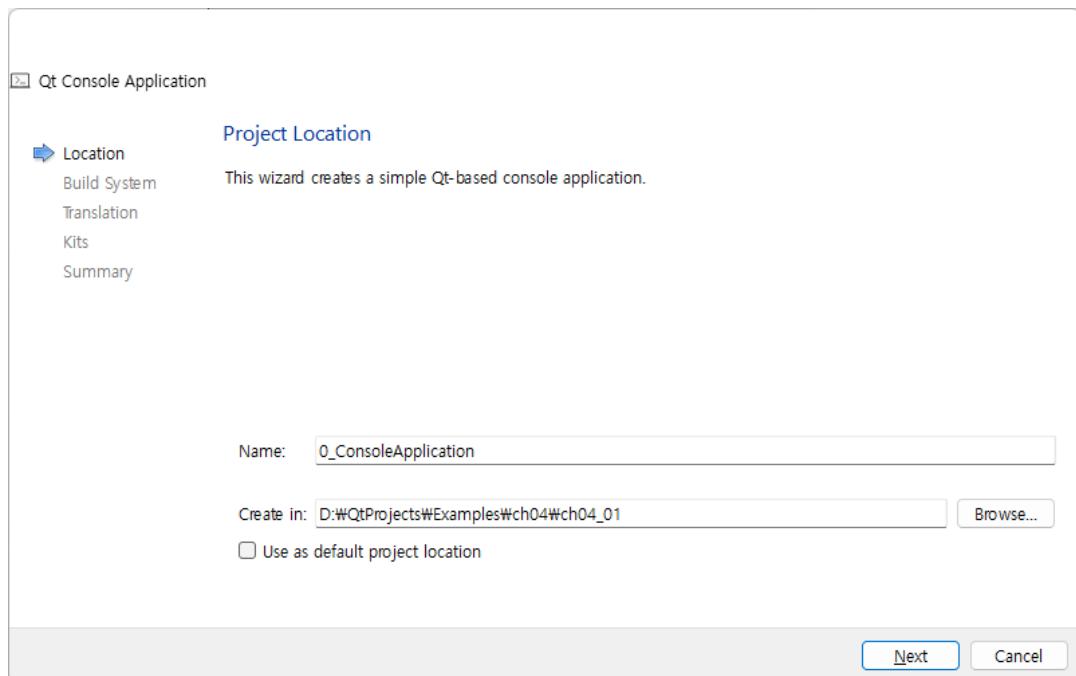
이번 장에서는 qmake를 이용해 GUI가 없는 Console 창 또는 Application Output 창에 “Hello World” 를 출력하는 어플리케이션 만들어 보도록 하자. Qt Creator 메뉴에서 [File] -> [New File or Project] 메뉴를 클릭한다.



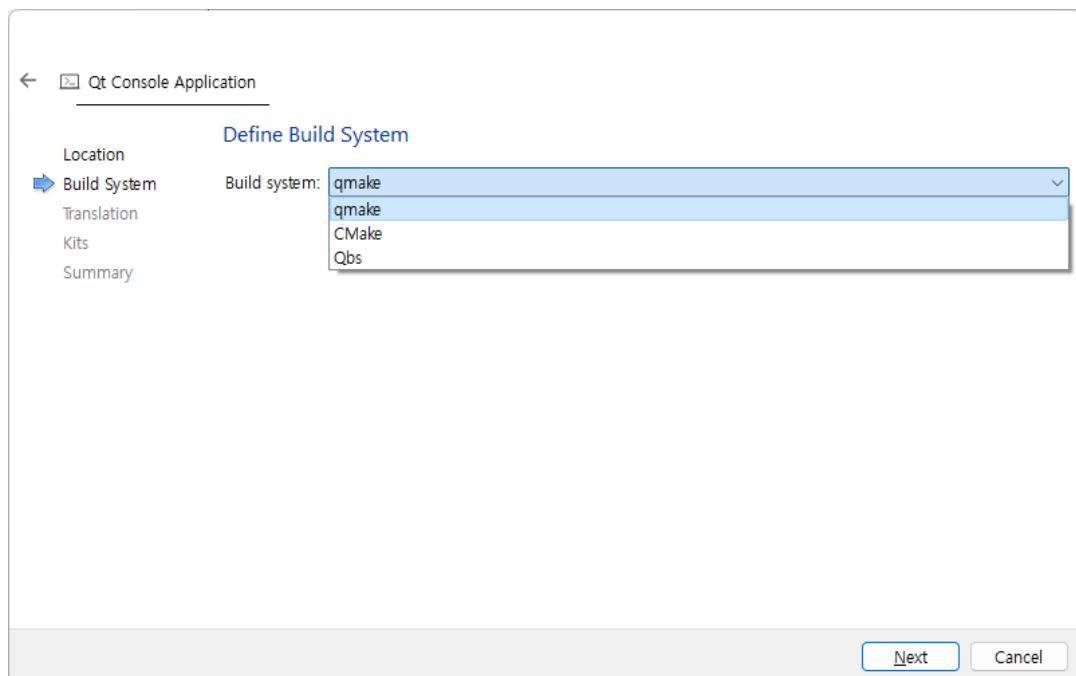
[New File or Project] 메뉴를 클릭하면 아래 그림에서 보는 것과 같이ダイ얼로그 창이로딩된다.



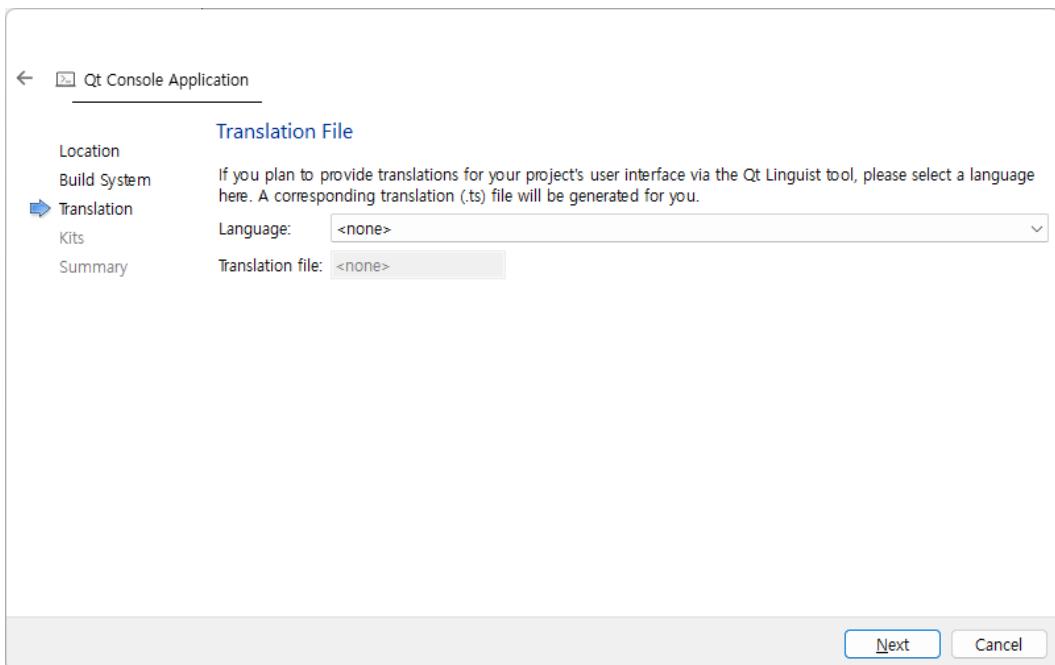
위의 그림에서 보는 것과 같이 [Qt Console Application] 항목을 선택하고 하단에 [Choose] 버튼을 클릭한다.



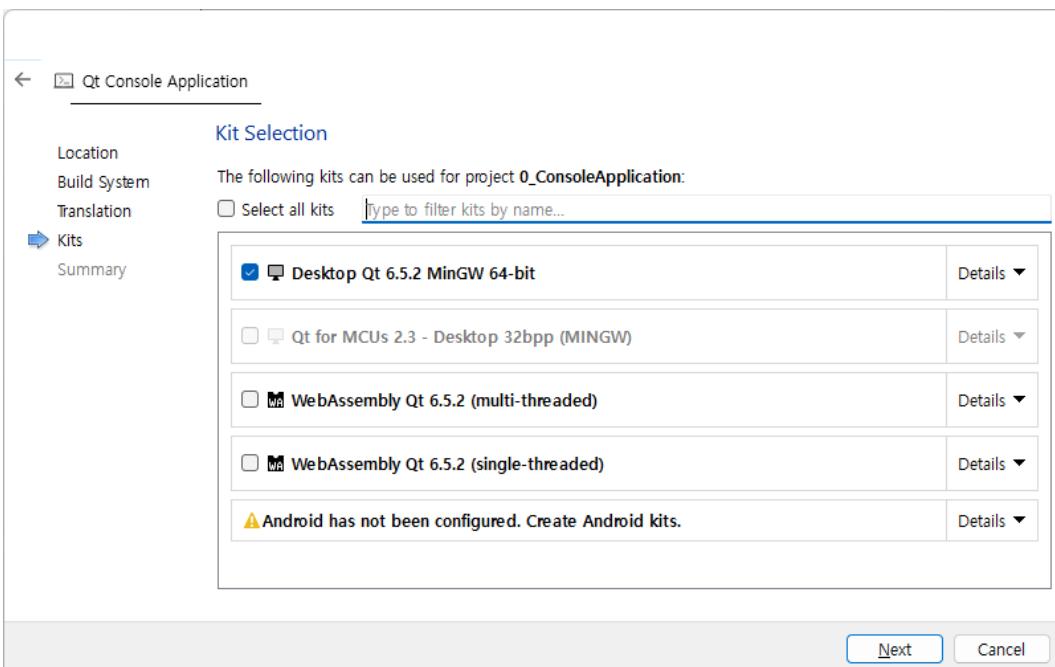
[Name] 항목에는 프로젝트 이름, [Create In] 항목에는 프로젝트가 위치할 디렉토리 이름을 지정한다.



빌드 도구로 qmake를 선택한다.



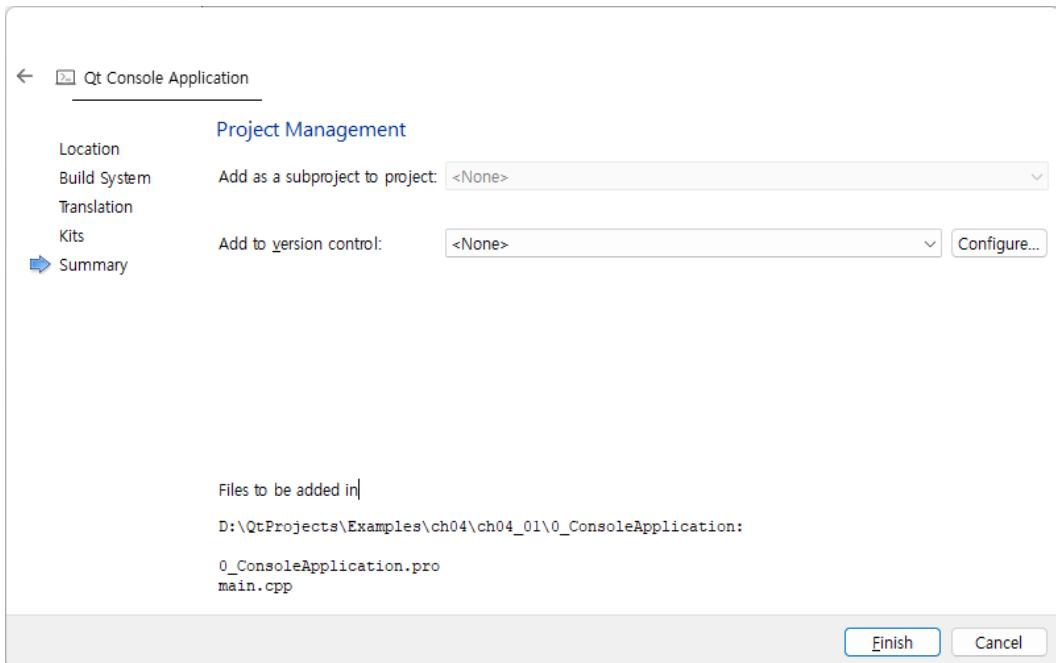
위의 다이얼로그 화면은 다국어 지원을 위한 다이얼로그이다. 이 다이얼로그 창에서는 아무것도 변경하지 않고 하단의 [Next] 버튼을 클릭한다.



위의 다이얼로그는 컴파일할 다이얼로그를 선택하는 창이다. 위의 그림에서 보는 것과 같이 MinGW 컴파일러를 선택한다. 여기서 MSVC 컴파일러가 보이지 않는 이유는 Qt 설치 시 MSVC 컴파일러를 선택하지 않았기 때문이다. 만약 컴파일러로 MSVC를 선택

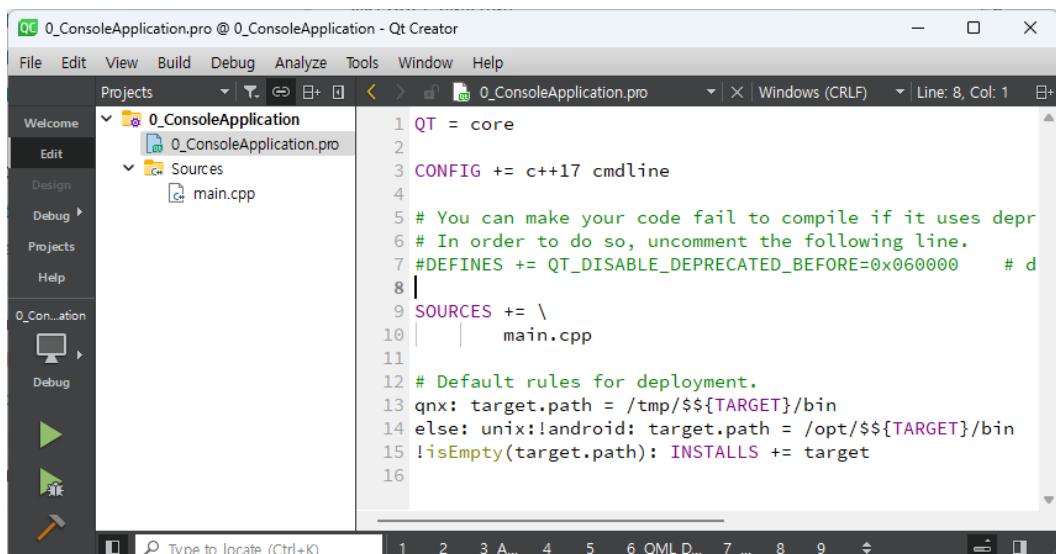
예수님은 당신을 사랑합니다.

하고 싶다면 Qt 설치 시 MSVC Component를 선택하고 설치한다. 그리고 MSVC 컴파일러를 설치해야 한다. MSVC를 먼저 설치해도 상관없다.



버전 관리 시스템(Git 과 같은)을 선택하는ダイ얼로그이다. 이ダイ얼로그에서는 아무 것도 변경하지 않고 하단의 [Finish] 버튼을 클릭해 프로젝트 생성ダイ얼로그를 완료한다.

프로젝트 생성이 완료되면 아래 그림에서 보는 것과 같이 Qt Creator 창에 생성된 프로젝트가 로딩될 것이다. 로딩이 완료되면 프로젝트파일(확장자가 .pro)을 더블 클릭한다.



예수님은 당신을 사랑합니다.

위의 그림에서 보는 것과 같이 프로젝트파일이 상당히 단순하다. 전체 코드를 자세히 살펴보도록 하자. 아래 소스코드는 프로젝트 파일의 소스코드이다.

```
QT = core

CONFIG += c++17 cmdline

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000
#           disables all the APIs deprecated before Qt 6.0.0

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
```

QT 키워드로 시작하는 부분은 이 프로젝트 Console 기반의 어플리케이션만 사용할 경우 core 모듈만 프로젝트에 포함시키라는 뜻이다. Qt 는 제공하는 라이브러리가 모듈 별로 나눠져 있다.

예를 들어 네트워크 API 또는 라이브러리를 사용하기 위해서는 Qt 키워드에 network 모듈을 아래와 같이 추가해야 한다.

```
QT = core
QT += network
```

CONFIG 키워드에서 “C++17” 은 사용할 C++ 버전이 17버전을 사용하겠다는 뜻이다. “cmdLine” 은 Console 어플리케이션일 경우를 뜻한다. 이 옵션은 모든 플랫폼에서 사용할 수 있다.

이 방법 이외에 MS윈도우에서는 CONFIG += console 을 사용할 수 있고, macOS 에서는 CONFIG -= app_bundle 을 사용할 수 있다.

그리고 CONFIG 는 미리 정의된 변수들 이외에도 사용자 직접 변수처럼 지정해 사용할 수 있다. 예를 들어 아래와 같이 opengl 이라는 변수를 사용자가 선언해 사용할 경우 아래와 같이 사용할 수 있다.

```
CONFIG += opengl
```

예수님은 당신을 사랑합니다.

위와 같이 opengl 이라는 변수를 CONFIG 에 등록하고 아래와 같이 특정 조건일 경우 사용할 수 있다.

```
opengl {  
    TARGET = application-gl  
} else {  
    TARGET = application  
}
```

SOURCE 키워드는 소스코드를 명시한다. 여기서는 소스코드 main.cpp 만 존재하기 때문에 SOURCE 키워드에는 main.cpp 만 있다. 예를 들어 MyCode.h 와 MyCode.cpp 소스코드가 있다면 아래와 같이 SOURCE 와 HEADER 코드에 MyCode.h 와 MyCode.cpp 소스코드 파일명이 추가 된다.

```
SOURCES += \  
    MyCode.cpp \  
    main.cpp  
  
HEADERS += \  
    MyCode.h
```

위의 소스에서 Back Slash (" \ ") 문자는 줄 바꿈(New Line) 을 하라는 뜻이다.

아래 소스코드에서 "qnx" 키워드는 플랫폼이 QNX 인 경우를 뜻한다. target.path 는 빌드 된 실행파일이 위치할 디렉토리이다.

그리고 프로젝트파일에서는 if 문과 같은 분기 문을 사용할 수 있다. else 는 만약 QNX 플랫폼이 아니라면 이라는 뜻이고 unix:!android: 는 플랫폼이 리눅스이고 안드로이드 플랫폼이 아닌 경우 빌드 된 실행 파일 위치를 뜻한다. TARGET 변수는 프로젝트의 실행파일명을 뜻한다.

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

여기까지 프로젝트 파일에 대해서 알아보았다. main.cpp 의 내용은 이전 장에서 설명하였으므로 이번 장에서는 따로 설명하지 않겠다.

그러면 main.cpp 상에 아래와 같이 "Hello world"를 출력하는 예제를 작성해 보도록 하자.

예수님은 당신을 사랑합니다.

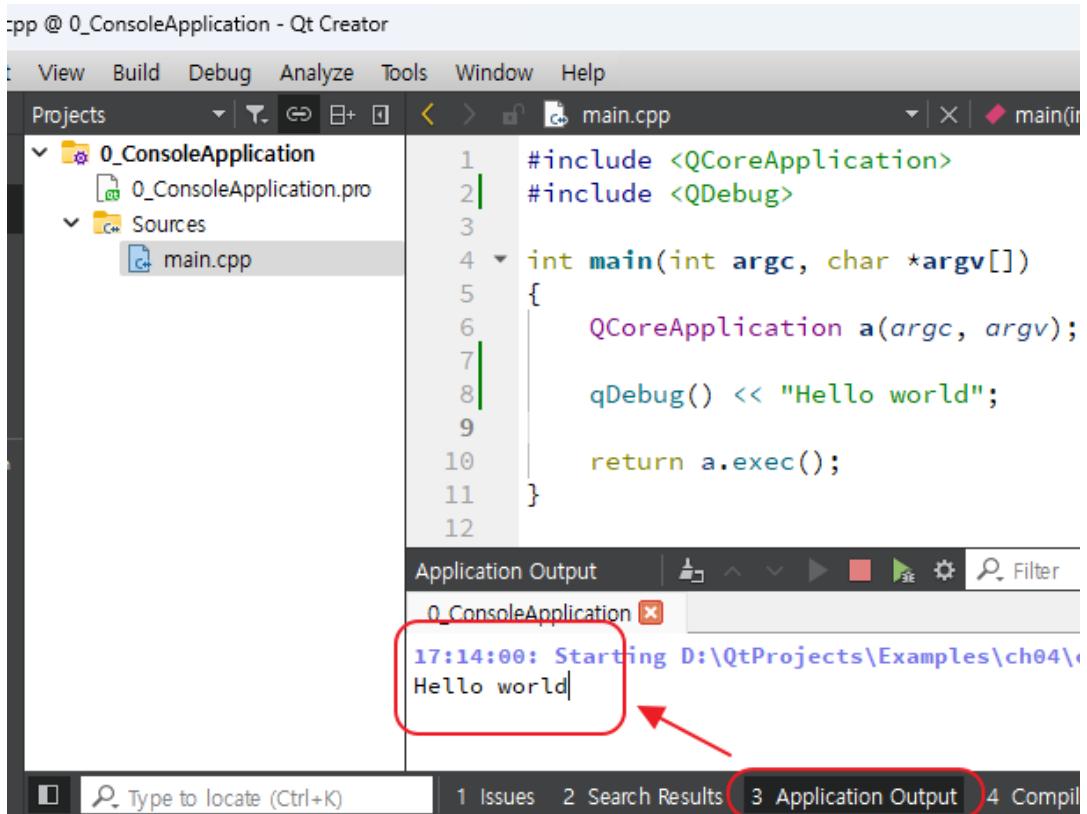
```
#include <QCoreApplication>
#include <QDebug>

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

    qDebug() << "Hello world";

    return a.exec();
}
```

위와 같이 소스코드를 작성하고 빌드 후 실행 해 보도록 하자.

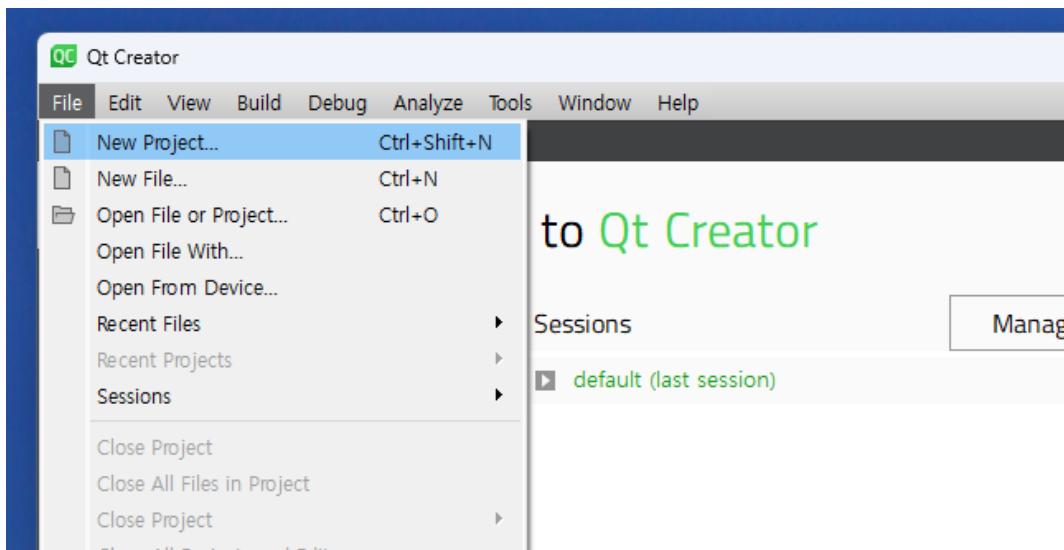


위의 그림에서 보는 것과 같이 Qt Creator 창 하단에 [Application Output] 탭을 클릭하면 "Hello world"를 출력하는 것을 확인할 수 있을 것이다.

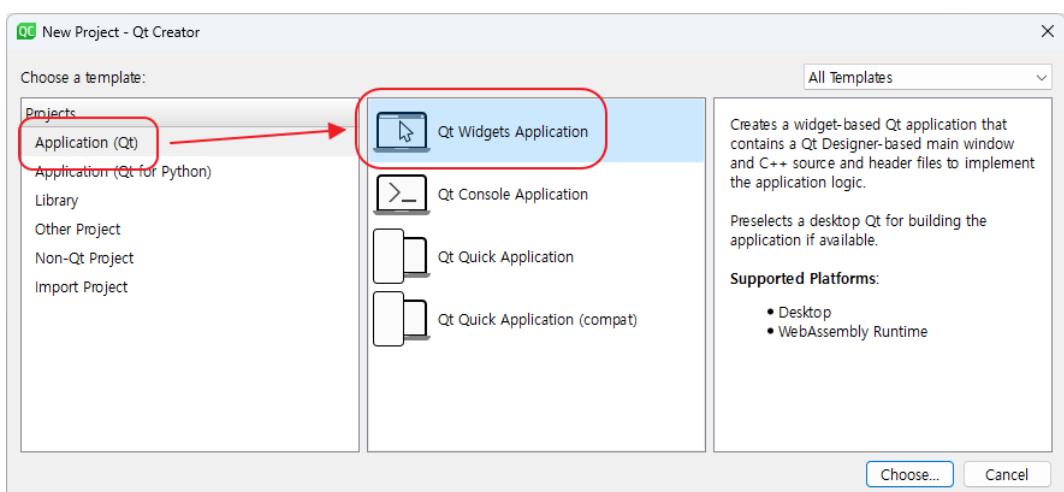
4.2. qmake 를 이용한 GUI 어플리케이션 구현

이번 장에서는 GUI상에 버튼을 하나 배치하고 이 버튼을 클릭하면 "Hello world"를 출력하는 예제를 구현해 보도록 하자.

이벤트에 대해서는 나중에 자세히 다루기로 하고 여기서는 간단히 버튼 이벤트만 구현해 보도록 하자. 아래와 같이 Qt Creator 툴을 실행한다. 그리고 새로운 프로젝트를 생성한다.



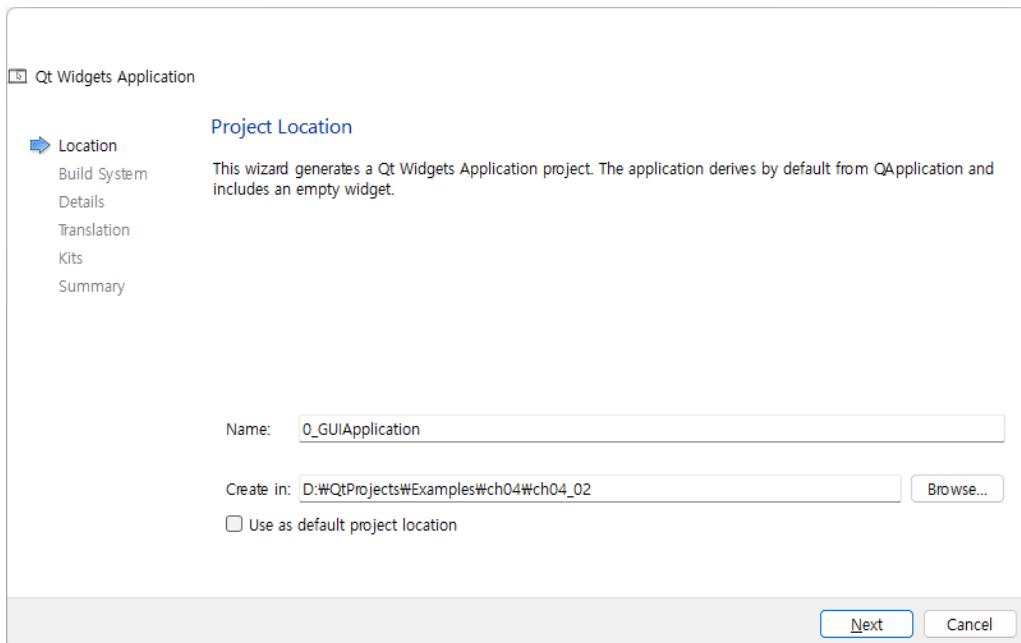
단축키는 **Ctrl + Shift + N** 을 클릭하면 된다.



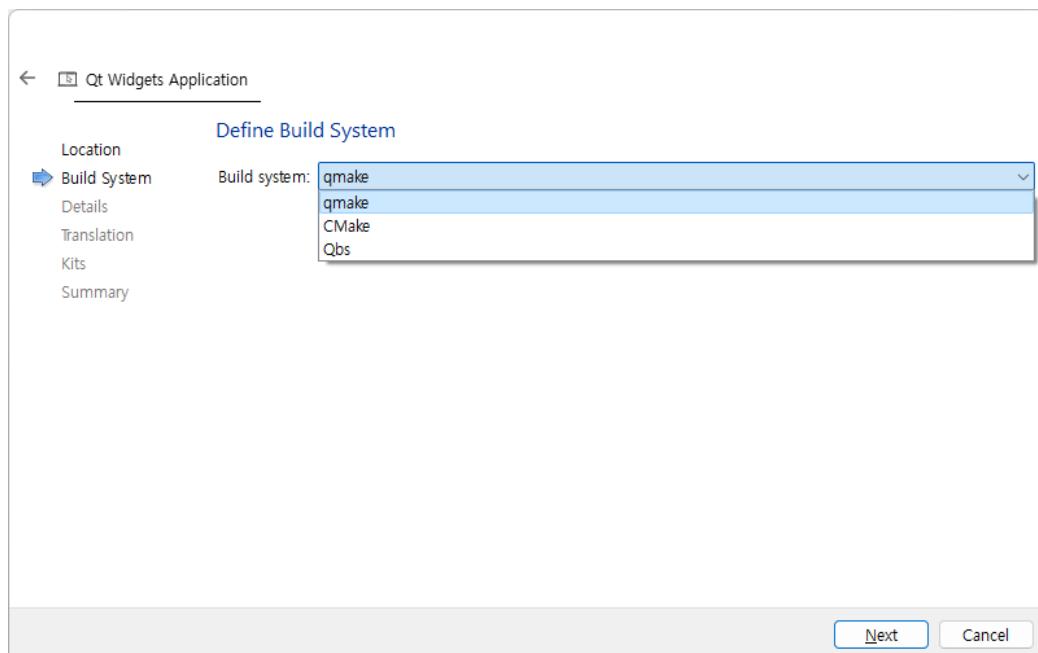
다이얼로그의 왼쪽 리스트에서 [Application Qt] 항목을 선택하고 중간에 [Qt Widget

예수님은 당신을 사랑합니다.

Application] 항목을 선택한다. 그리고 하단에 [choose] 버튼을 클릭한다. 다음은 Name 항목에서 프로젝트명을 입력하다. Create in 항목은 프로젝트가 위치할 디렉토리(폴더)를 선택하고 [Next] 버튼을 클릭한다.



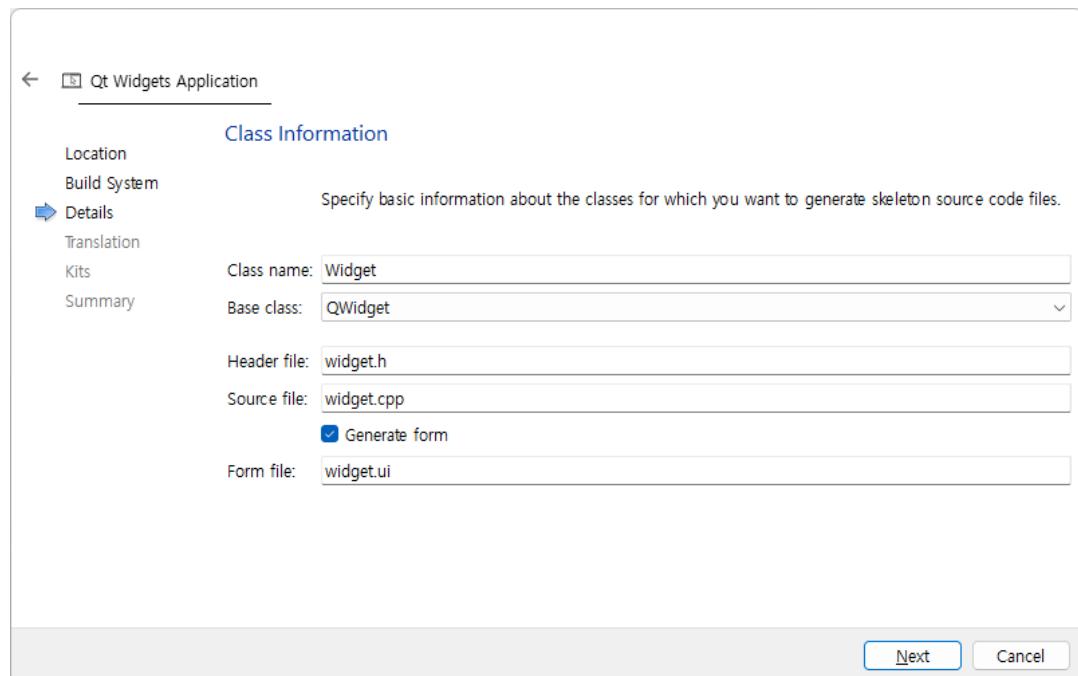
다음은 빌드 도구를 선택하는 다이얼로그이다. qmake 항목을 선택하고 [Next] 버튼을 클릭한다.



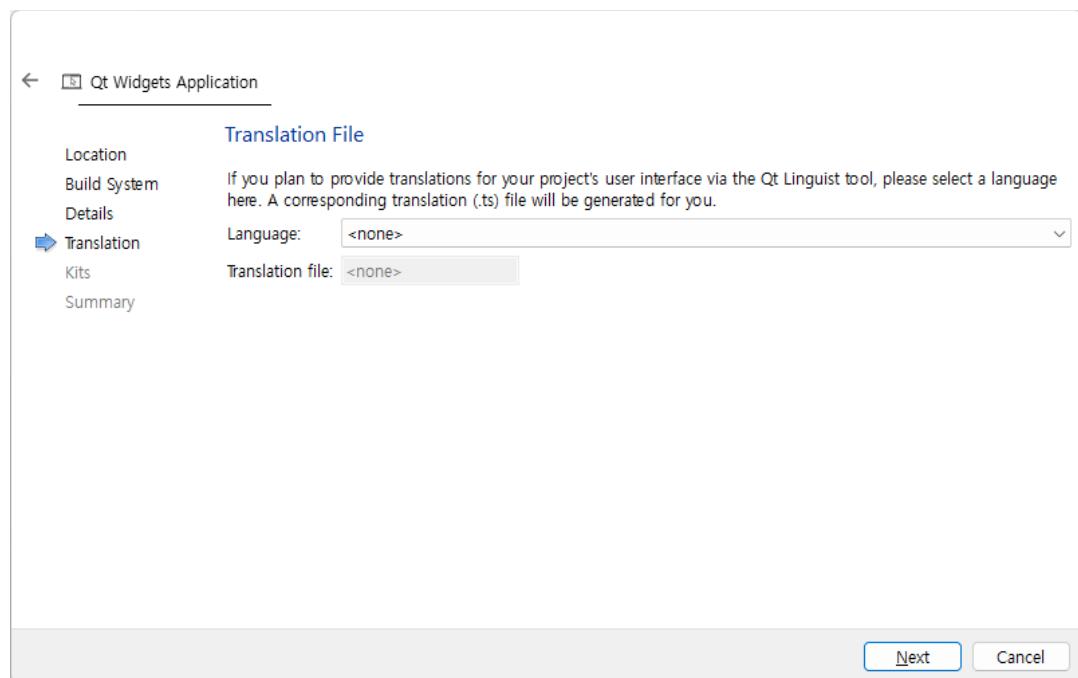
다음은 생성할 Class 정보를 입력하는 창이다. 이 창에서는 자동으로 생성할 GUI 클래

예수님은 당신을 사랑합니다.

스를 명시한다. Base class 항목의 콤보박스에서 [QWidget] 을 선택하고 [Next] 버튼을 클릭한다.

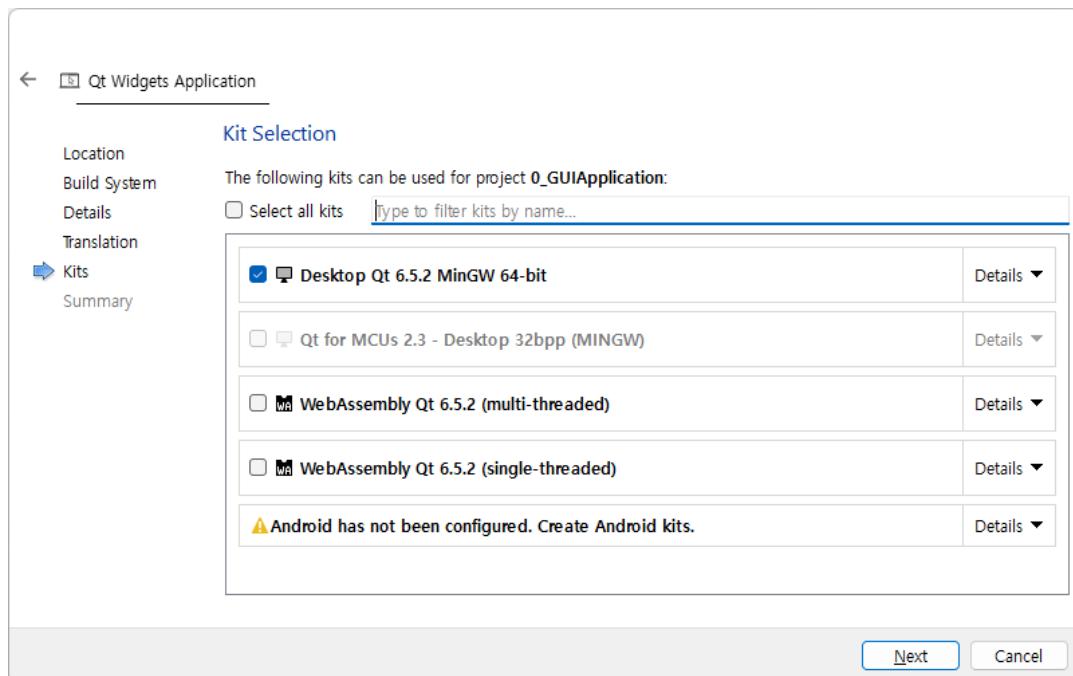


다음으로 다국어 지원을 위한 다이얼로그 창이다. 이 창에서는 아무것도 바꾸지 않고 [Next] 버튼을 클릭한다.

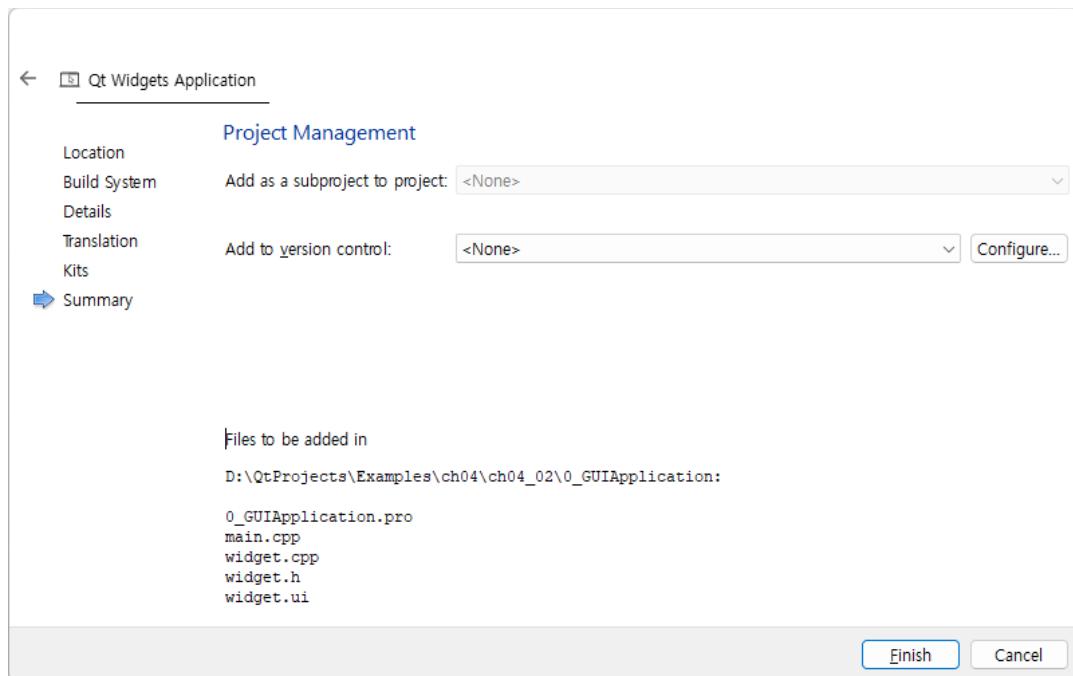


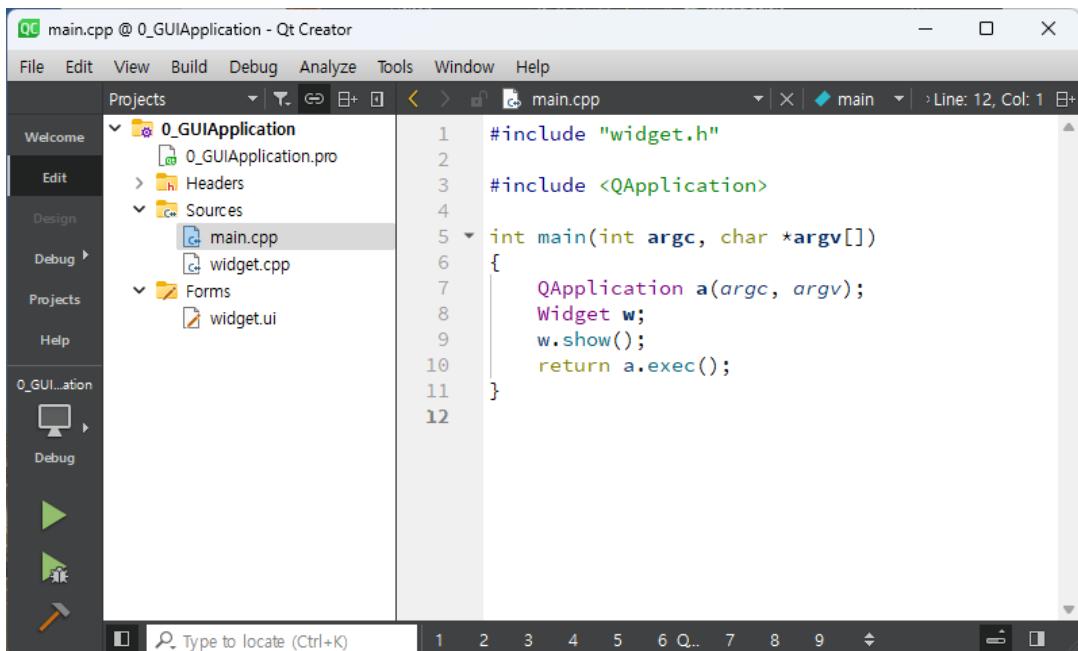
예수님은 당신을 사랑합니다.

다음은 어플리케이션을 빌드할 컴파일러를 선택하는 화면이다. 여기서는 MinGW 컴파일러를 선택한다.



다음으로 버전 관리 시스템을 선택하는 화면이다. 여기서는 아무것도 선택하지 않고 하단의 [Finish] 버튼을 클릭해 프로젝트 생성을 완료한다.





The screenshot shows the Qt Creator interface. The left sidebar has a 'Welcome' section with 'Edit', 'Design', 'Debug', 'Projects', and 'Help' buttons. Below that is a project tree for '0_GUIApplication' containing '0_GUIApplication.pro', 'Headers', 'Sources' (which contains 'main.cpp' and 'widget.cpp'), and 'Forms' (which contains 'widget.ui'). The right pane shows the code editor with the following content:

```
#include "widget.h"  
  
#include <QApplication>  
  
int main(int argc, char *argv[]){  
    QApplication a(argc, argv);  
    Widget w;  
    w.show();  
    return a.exec();  
}
```

프로젝트 생성을 완료하면 위와 같이 소스코드들이 자동으로 생성될 것이다. main.cpp 는 프로그램이 시작되는 소스코드이다. main.cpp 소스코드에서 Widget 이라는 클래스 인스턴스를 생성한다. 그리고 이 인스턴스의 GUI를 보이기 위해서 show() 함수를 호출 한다. Widget 클래스는 GUI를 구현하기 위한 클래스를 Qt Creator 에서 자동으로 생성 해준 클래스이다.

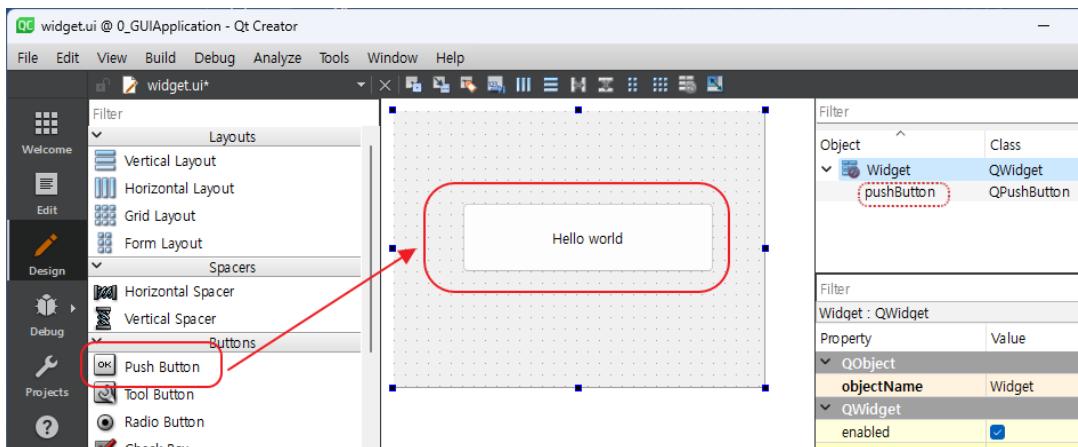
그리고 widget.ui 파일도 생성된 것을 확인할 수 있을 것이다. 이 파일은 Qt 에서 제공 하는 Qt Designer 툴로 쉽게 마우스로 GUI 위젯들을 드래그해 GUI상에 배치할 수 있다. Qt Designer 툴로 배치한 Widget들의 위치 및 특성 정보를 widget.ui 파일에 저장한다. 파일의 형식은 XML이다.

이 파일은 나중에 컴파일 시, Qt가 자동으로 C++ 소스코드로 변환해 준다. widget.ui 는 Widget 클래스와 맵핑 되어 있으며 여기서는 클래스 이름과 ui 파일의 이름은 동일하게 저장되어 있지만, 동일하지 않아도 된다.

또한 여기서는 1개의 ui 파일만 존재하지만 2개 이상의 GUI를 가질 수 도 있다. 예를 들어 한 개의 클래스는 여러 개의 ui 파일과 맵핑 할 수 있다.

여기서 widget.ui 파일을 더블 클릭하면 Qt Designer 가 Qt Creator 창에서 자동으로 실행된다. 아래 그림에서 보는 것과 같이 widget.ui 파일을 더블 클릭해 Qt Designer를 실행한다.

예수님은 당신을 사랑합니다.



위의 그림에서 보는 것과 왼쪽 Button 탭에서 Push Button을 마우스로 드래그해 화면에 배치한다. 그리고 버튼의 이름을 위의 그림에서 보는 것과 같이 "Hello world"로 변경한다.

그리고 위의 버튼의 고유한 이름을 변경할 수 있다. 이 인스턴스의 고유한 이름은 우측 상단에 표시된다. 여기서는 이름을 "pushButton"으로 하고 ui 파일을 저장(Ctrl + S) 한다. 그리고 Qt Creator 창의 왼쪽에 [Edit] 아이콘을 클릭해 소스코드 창으로 화면을 전환한다.

아래 소스코드는 widget.h이다. 이 소스코드 버튼을 클릭하면 호출되는 Slot 함수(이벤트)를 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
```

예수님은 당신을 사랑합니다.

```
Ui::Widget *ui;

private slots:
    void slot_clicked();

};

#endif // WIDGET_H
```

위의 소스코드에서 보는 것과 같이 slot_clicked() 함수를 작성하고 widget.cpp 에서 이 Slot 함수의 구현 부를 아래와 같이 작성한다.

이 함수에서는 “Hello world” 를 출력하기 위해서 Qt 에서 제공하는 함수를 사용해 보자. widget.cpp 파일 상단에 QDebug 헤더를 include 한다. 그리고 slot_clicked() 함수 상에 “Hello world”를 출력하는 소스코드를 작성해 보자.

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

...

void Widget::slot_clicked()
{
    qDebug() << "Hello world";
}
```

마지막으로 버튼을 클릭했을 때 slot_clicked() 함수를 호출하기 위해서는 connect() 함수를 이용해 시그널(이벤트)과 Slot 함수인 slot_clicked() 함수를 연결한다.

```
...
Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pushButton, SIGNAL(clicked()), this, SLOT(slot_clicked()));

}
...
```

위의 connect() 함수의 첫 번째 인자는 시그널을 발생하는 오브젝트이다. 이 오브젝트의 고유한 이름은 Qt Designer 에서 정의하였다.

예수님은 당신을 사랑합니다.

두 번째 인자는 시그널을 발생하는 종류이다. 클릭, 더블 클릭, 클릭 후 해제 했을 때 등 여러 개의 시그널이 존재한다.

여기서는 이 버튼을 클릭했을 때 발생하는 시그널을 명시한다. 세 번째 인자는 이 시그널과 연결할 Slot 함수가 있는 인스턴스명을 입력한다. 여기서는 자기 자신을 명시하기 때문에 this 을 입력하였다. 마지막인자는 호출 할 Slot 함수를 명시한다. 여기서는 slot_clicked() 함수를 명시한다.

여기까지 작성하였으면 모든 소스코드 작성이 완료되었다. 아래 소스코드는 widget.cpp 전체 소스코드이다.

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

#include <QDebug>

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

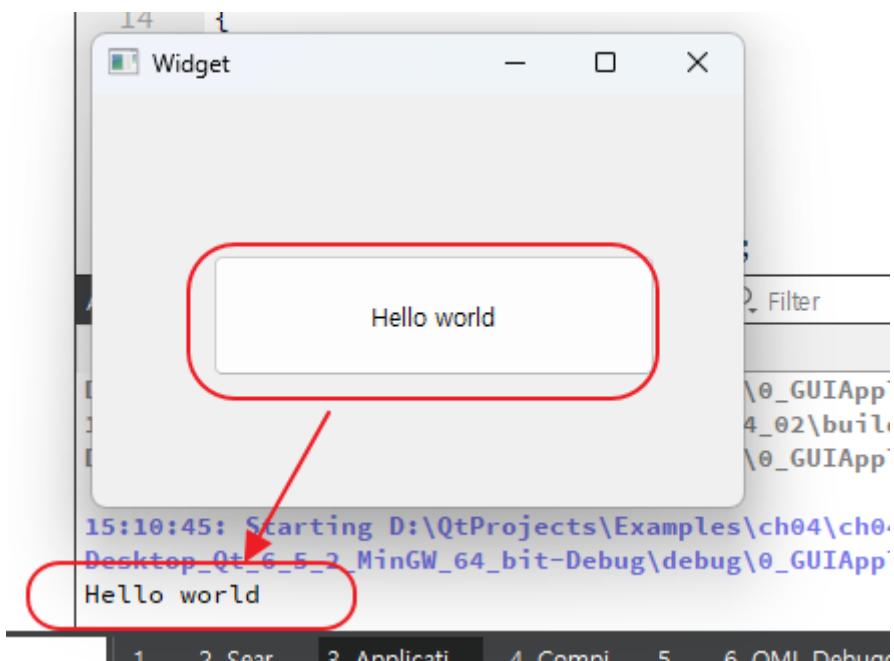
    connect(ui->pushButton, SIGNAL(clicked()), this, SLOT(slot_clicked()));
}

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

void Widget::slot_clicked()
{
    qDebug() << "Hello world";
}
```

다음으로 어플리케이션을 빌드한 다음 실행해 보고 아래와 같이 실행하는지, 확인해 보도록 하자.

예수님은 당신을 사랑합니다.

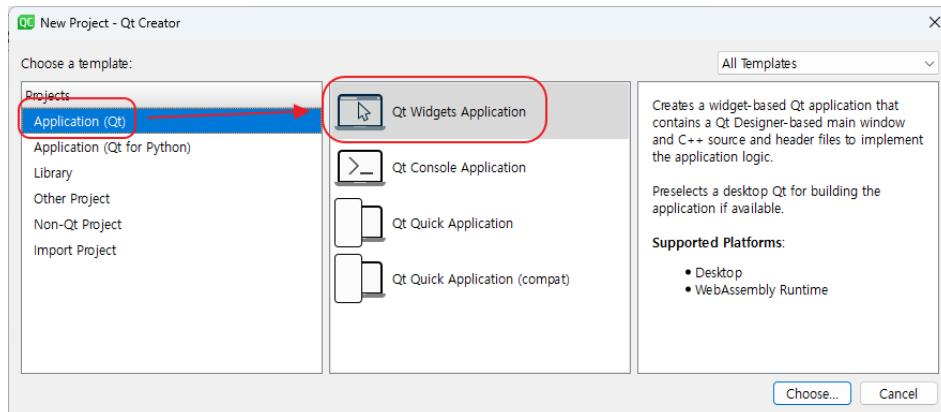


5. Qt GUI Widgets

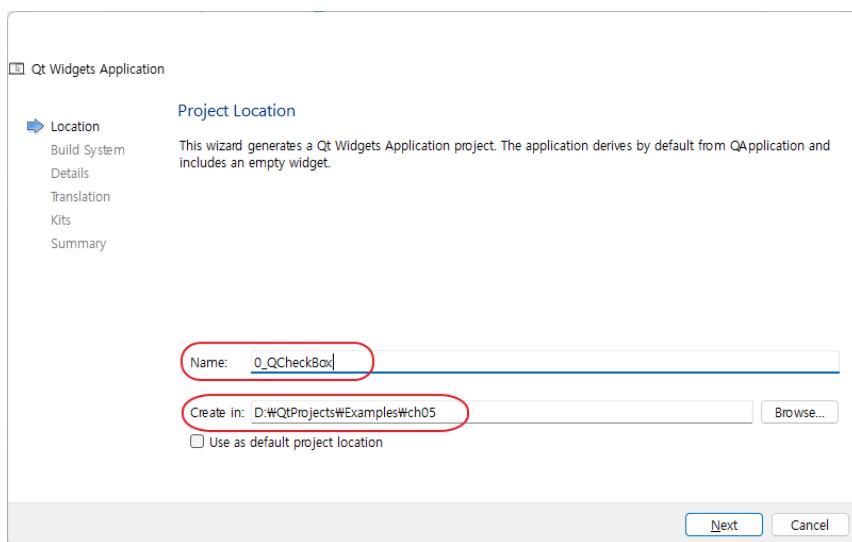
Qt에서 Widget은 버튼, 체크박스, 라디오 버튼 등을 Widgets이라고 한다. 따라서 이번 장에서는 Qt에서 제공하는 Widget들에 대해서 다루어 보도록 하자.

- ✓ QCheckBox 와 QButtonGroup

QCheckBox와 QButtonGroup를 이용한 예제를 작성해 보도록 하자. 이번 예제에서는 QCheckBox에서 이미지 리소스를 사용하는 방법도 함께 알아보도록 하자. 아래와 같이 프로젝트를 생성할 때 [QWidget Application]를 선택한다.

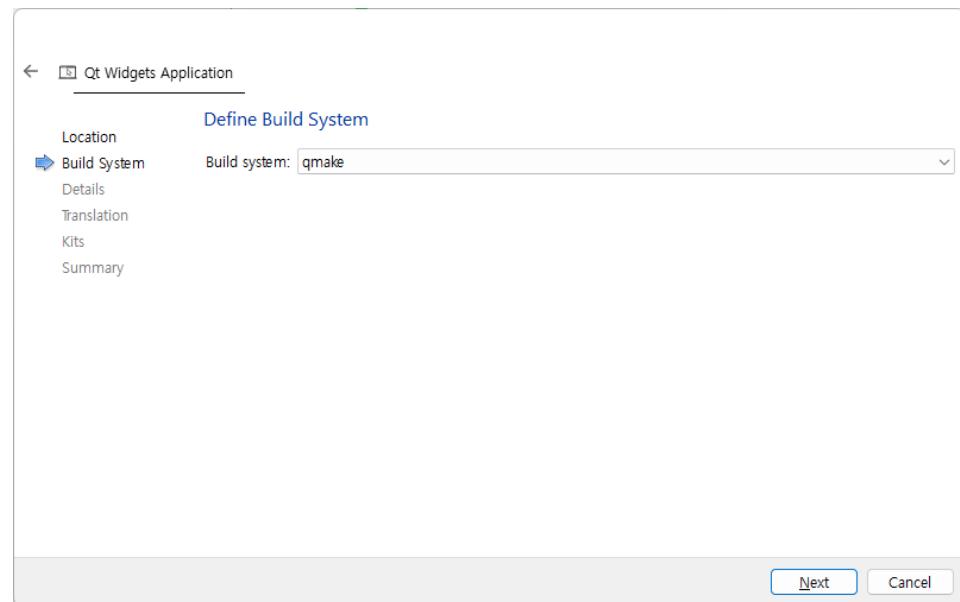


다음으로 프로젝트명과 프로젝트가 위치할 디렉토리를 선택 후 [Next] 버튼을 클릭한다.



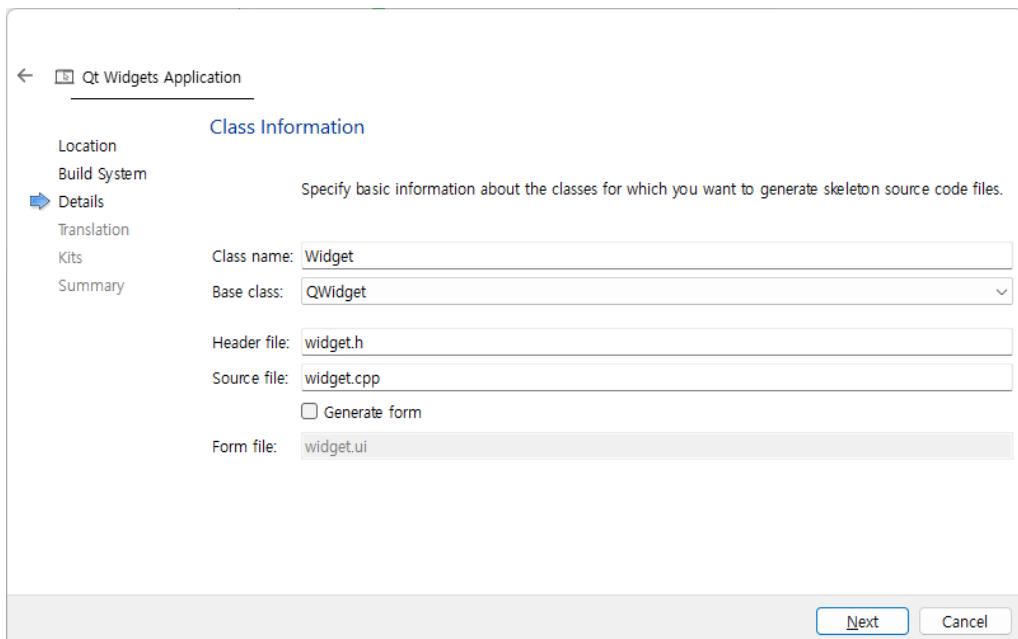
예수님은 당신을 사랑합니다.

프로젝트 빌드 도구를 선택하는 창에서 qmake를 선택해 보도록 하자.



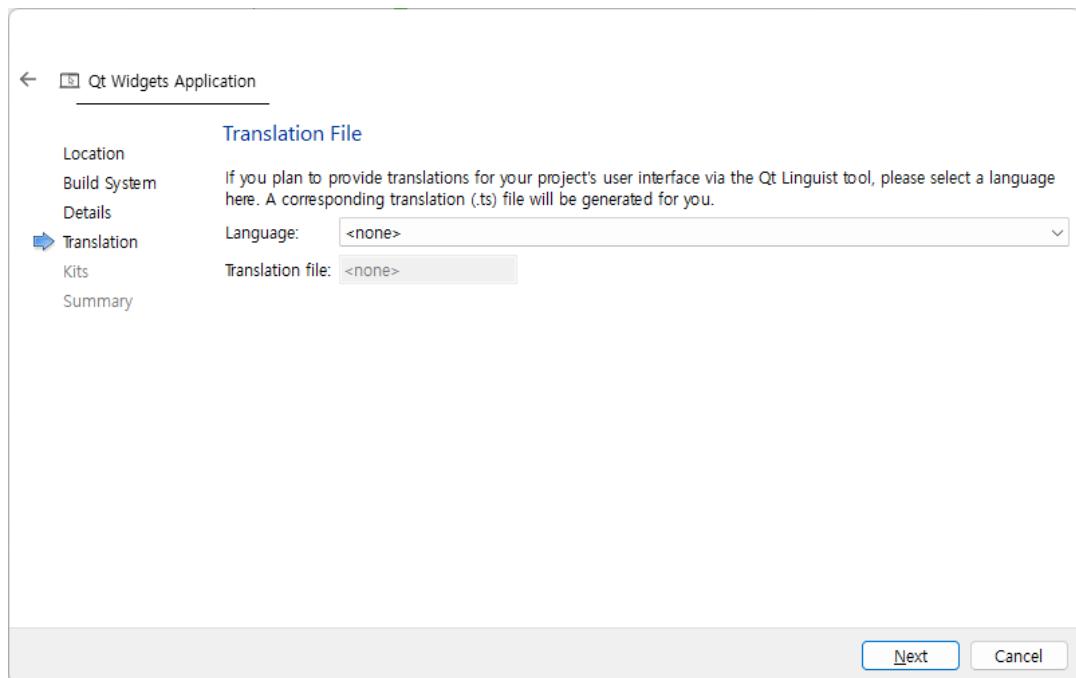
다음으로 프로젝트 생성 시 생성할 Class 정보를 입력하는 창이다. Class name 은 Class의 이름을 입력한다. Base class 는 QWidget 을 선택한다. QWidget 을 선택하면 생성하는 Class 가 QWidget 을 상속받도록 Class를 만들어 준다.

여기서는 Qt Designer를 사용하지 않고 직접 GUI Widget 을 소스코드로 작성해 보도록 하자. 아래 다이얼로그 창에서 보는 것과 같이 Generate form 항목에 체크를 해제한다.

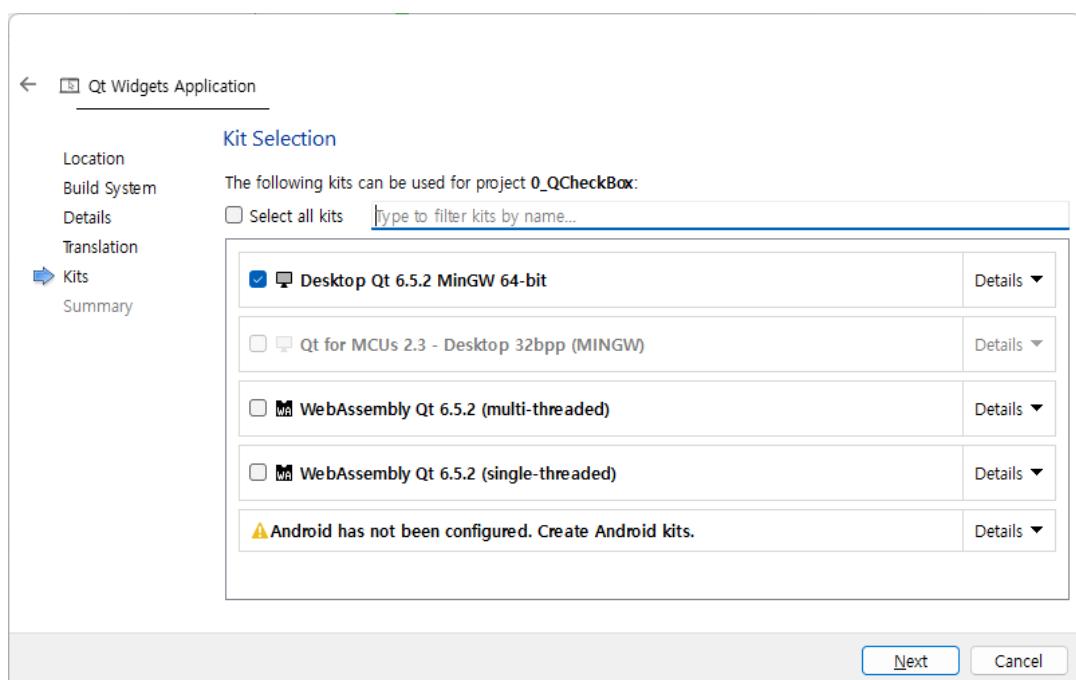


예수님은 당신을 사랑합니다.

다음으로 아무것도 변경하지 않고 [Next]버튼을 클릭한다.



컴파일러는 MinGW를 선택한다.



다음 창에서는 버전 관리 시스템을 선택하는 화면이다. 아무것도 변경하지 않고 [Finish] 버튼을 클릭한다.

예수님은 당신을 사랑합니다.

프로젝트 생성이 완료 되었다면 widget.h 파일에 다음과 같이 소스코드를 작성해 보도록 하자.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QCheckBox>
#include <QButtonGroup>

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    QButtonGroup      *m_chk_group[2];
    QCheckBox        *m_exclusive[3];
    QCheckBox        *m_non_exclusive[3];

private slots:
    void slot_chkChanged();

};

#endif // WIDGET_H
```

위에서 보는 것과 같이 QButtonGroup 과 QCheckBox 오브젝트를 선언한다. 그리고 시그널(이벤트)을 처리할 slot_chkChanged() 함수를 선언한다.

이 Slot 함수는 QCheckBox에서 발생한 시그널을 처리하는 함수이다. Slot 함수를 선언하기 위해서는 접근 제한자(private 또는 public) 과 “slots”라는 키워드를 선언해야 한다.

다음으로 widget.cpp를 아래에서 보는 것과 같이 작성해 보도록 하자.

```
#include "widget.h"

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

```
QString str1[3] = {"Game", "Office", "Develop"};
QString str2[3] = {"P&rogramming", "Q&t", "O&S"};

int xpos = 30;
int ypos = 30;

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

for(int i = 0 ; i < 3 ; i++)
{
    m_exclusive[i] = new QCheckBox(str1[i], this);
    m_exclusive[i]->setGeometry(xpos, ypos, 120, 30);
    m_chk_group[0]->addButton(m_exclusive[i]);

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

    connect(m_exclusive[i], SIGNAL(clicked()),
            this,           SLOT(slot_chkChanged ()));
}

ypos += 40;
}

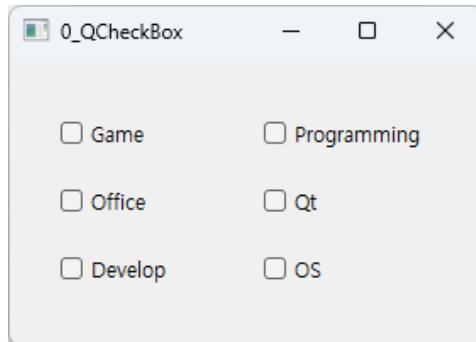
m_chk_group[0]->setExclusive(false);
m_chk_group[1]->setExclusive(true);
}

void Widget::slot_chkChanged()
{
    for(int i = 0 ; i < 3 ; i++) {
        if(m_exclusive[i]->checkState())
        {
            qDebug("checkbox %d selected ", i+1);
        }
    }
}

Widget::~Widget()
{
}
```

예수님은 당신을 사랑합니다.

위의 소스코드를 작성하고 빌드 후 실행 해 보도록 하자.



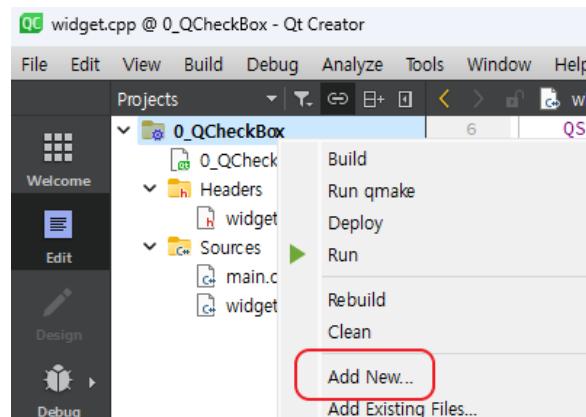
왼쪽의 Game, Office, Develop 항목과 오른쪽의 Programming, Qt, OS 은 QButtonGroup 으로 분리하였다. 왼쪽의 QCheckBox 항목은 다중 선택이 가능하지만 오른쪽 항목은 다중 선택이 불가능 하도록 구현 하였다. 다중 선택이 불 가능하게 하기 위해서는 QButtonGroup 클래스에서 제공하는 setExclusive() 멤버함수를 사용하면 된다.

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

위에서 보는 것과 같이 setExclusive() 함수에서 첫 번째 인자를 넘겨주면 다중 선택의 가능하거나 불가능하게 처리 할 수 있다. false를 사용하면 다중 선택이 가능하며 true 를 사용하면 체크박스임에도 불구하고 다중 선택이 불가능하다.

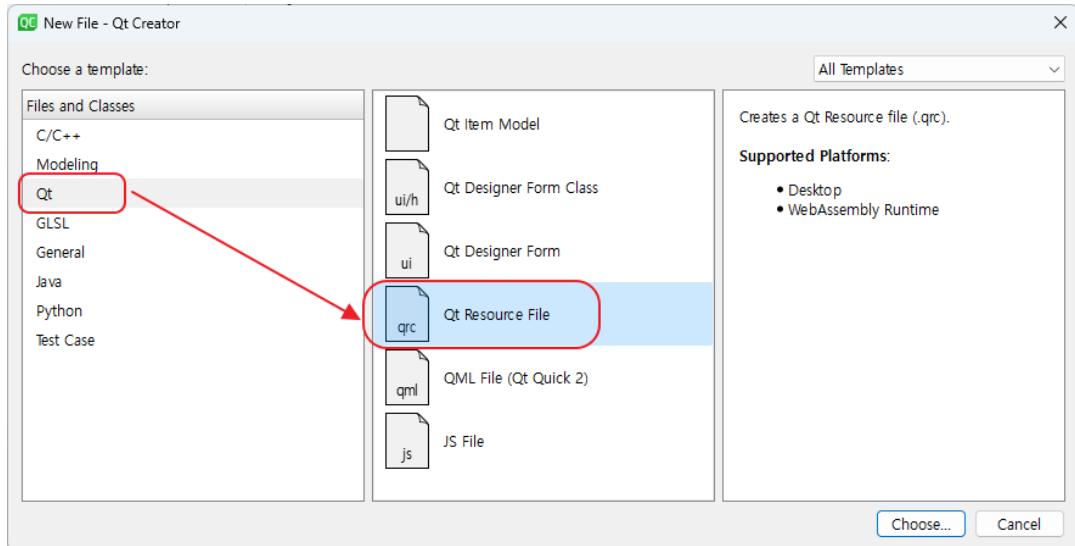
이번에는 Qt에서 이미지 리소스를 이용해 QCheckBox 에서 사용해 보도록 하자. 프로젝트 디렉토리에 resource 디렉토리를 만들고 사용할 이미지를 복사한다.

Qt Creator 창에서 프로젝트의 이름 상에 마우스를 위치해 놓은 다음 마우스 오른쪽 버튼을 클릭한다. 그러면 메뉴가 로딩된다.

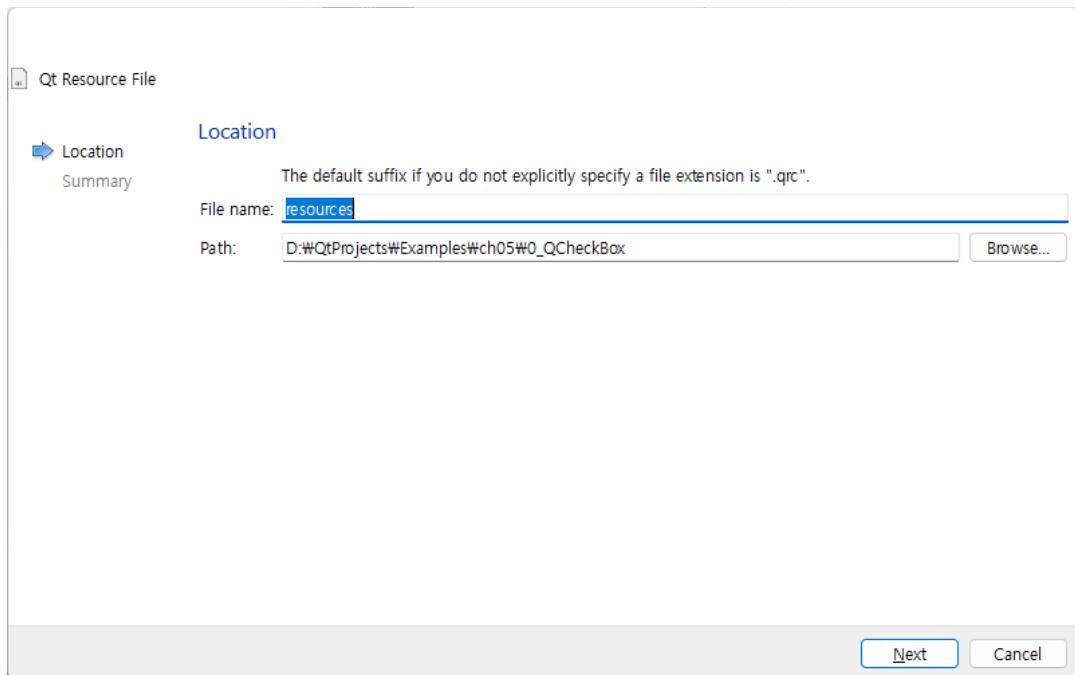


예수님은 당신을 사랑합니다.

[Add New] 메뉴를 클릭한다. 그런 다음 다이얼로그 창에서 왼쪽 항목에서 Qt 항목을 선택한 다음 중간 탭에서 [Qt Resource File] 항목을 클릭한다.



다음 다이얼로그 창에서는 리소스 파일명을 입력한다. 리소스 이름은 자유롭게 이름을 명시해도 된다. 여기서는 "resources"를 입력하자.

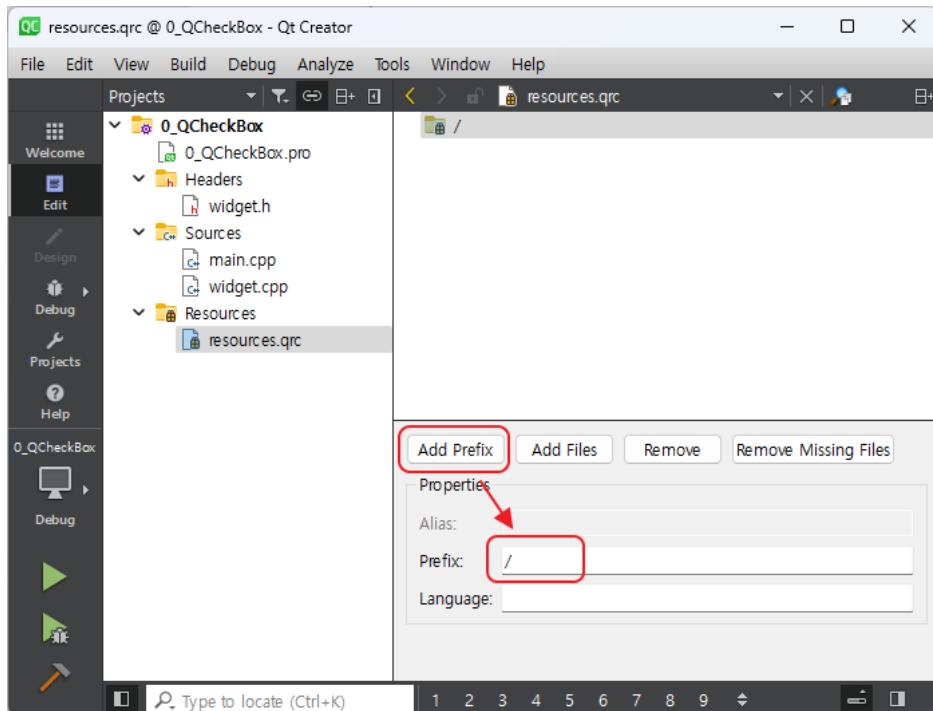


다음 다이얼로그 창에서는 아무것도 변경하지 않고 하단에 [Finish] 버튼을 클릭한다.

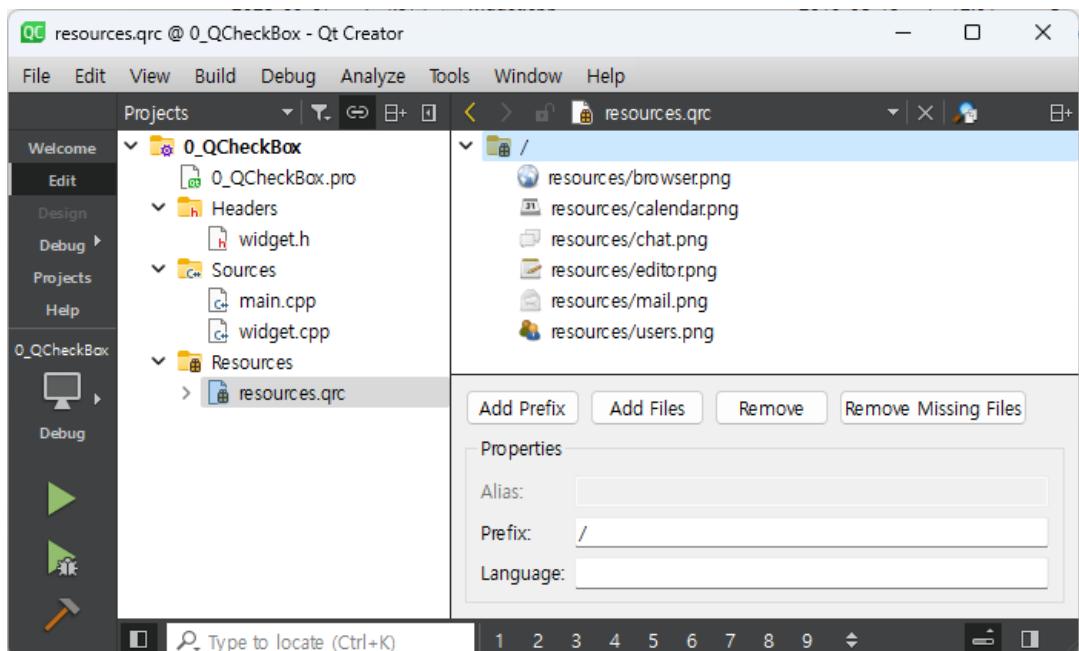
그러면 아래 그림에서 보는 것과 같이 resources.qrc 파일이 생성된 것을 확인할 수 있을 것이다.

예수님은 당신을 사랑합니다.

다음으로 [Add Prefix]를 누른다. 그리고 하단에 Prefix 항목에 "/"를 입력한다.



다음으로 [Add Files] 버튼을 클릭한 후 이미지 파일을 등록한다.



위의 그림에서 보는 것과 같이 이미지 파일들을 등록한다. 이미지 등록을 완료한 후에 widget.cpp 소스코드의 생성자 함수에 다음과 추가해 준다.

```

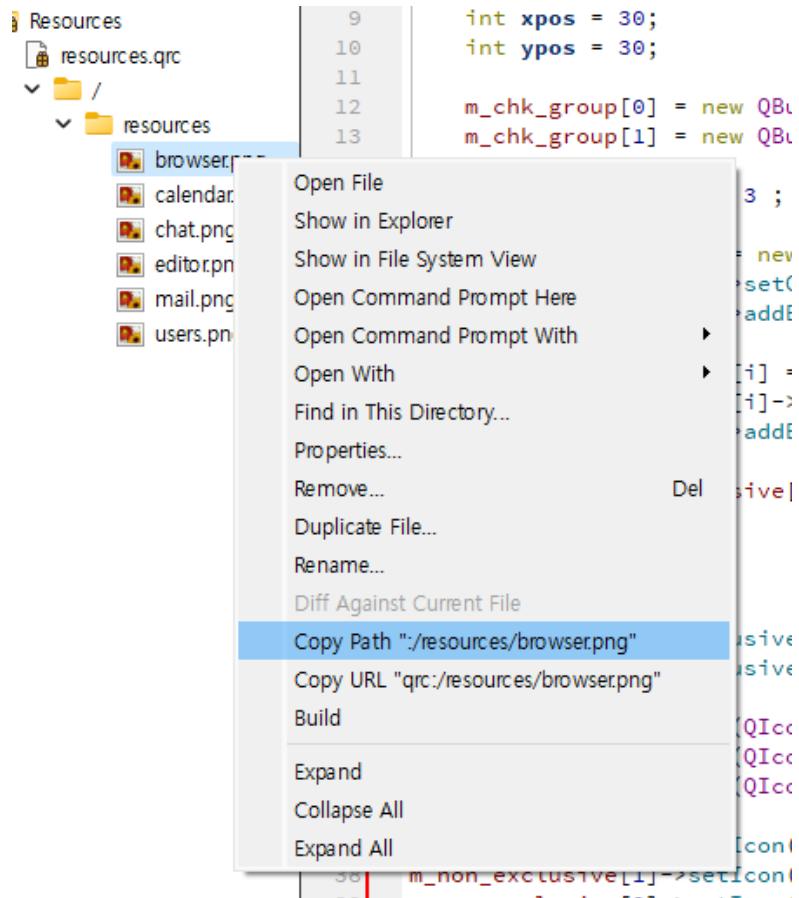
...
m_chk_group[0]->setExclusive(false);
m_chk_group[1]->setExclusive(true);

m_exclusive[0]->setIcon(QIcon(":resources/browser.png"));
m_exclusive[1]->setIcon(QIcon(":resources/calendar.png"));
m_exclusive[2]->setIcon(QIcon(":resources/chat.png"));

m_non_exclusive[0]->setIcon(QIcon(":resources/editor.png"));
m_non_exclusive[1]->setIcon(QIcon(":resources/mail.png"));
m_non_exclusive[2]->setIcon(QIcon(":resources/users.png"));
...

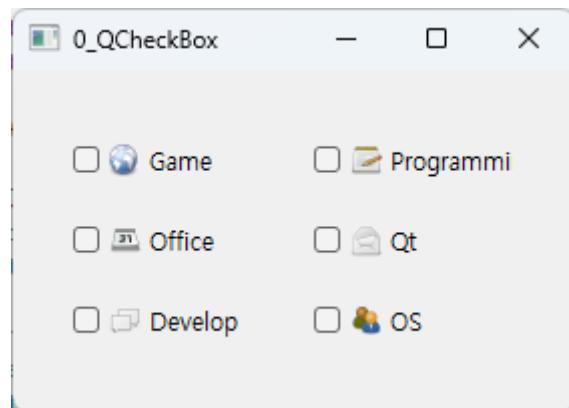
```

각 이미지의 리소스 파일의 위치를 알기 위해서는 Qt Creator 의 왼쪽 프로젝트 창에서 이미지에 마우스를 올려 놓은 다음에 마우스 오른쪽 버튼을 클릭하면 메뉴가 나타난다. 그러면 메뉴에서 Copy Path 또는 Copy URL를 누르면 해당 이미지의 리소스 명을 메모리에 복사된다. 그런 다음 소스코드 붙여 넣기 하면 된다.



예수님은 당신을 사랑합니다.

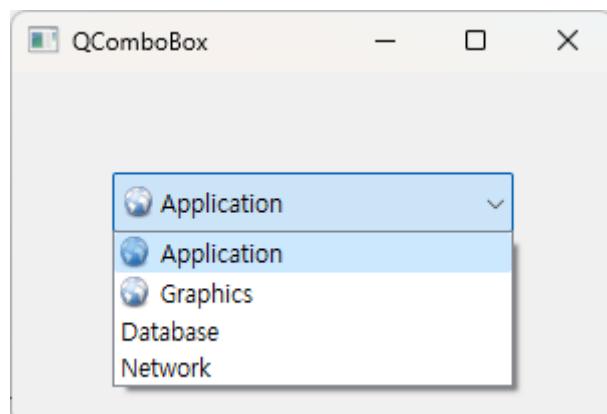
여기까지 작성은 완료하였다면 빌드 후 실행해 보도록 하자.



✓ 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/mail.png"), "Graphics");
combo->addItem("Database");
combo->addItem("Network");
...
```

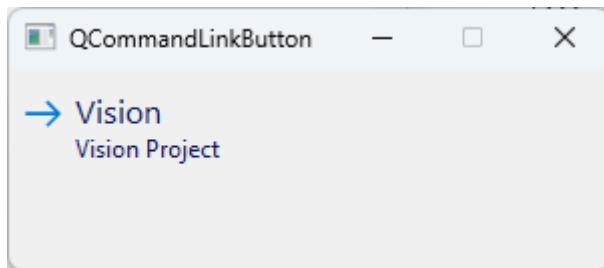


예제 전체 소스는 Ch05 > 01_QComboBox 에 있다.

✓ 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);
...
```



✓ 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.

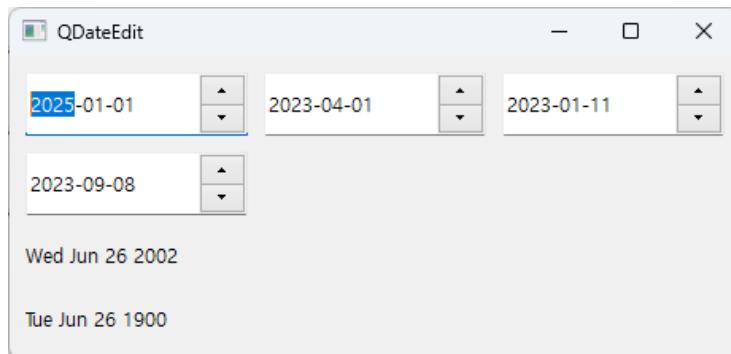
```
QDate dt1 = QDate(2020, 1, 1);
QDate dt2 = QDate::currentDate();

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(10, 60, 140, 40);
...
```



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");
```

| 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) |

예제 전체 소스는 03_QDateEdit 디렉토리에 있다.

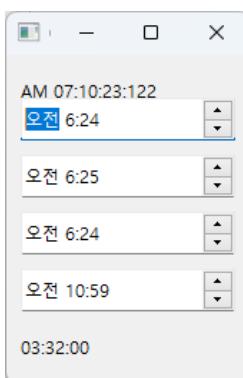
✓ QTime 클래스와 QTimeEdit 위젯 클래스

QTime 클래스는 시간을 표시하거나 특정 조건에 비교 등 어플리케이션 개발에 필요한 시간 관련 기능을 쉽게 구현할 수 있다. QTimeEdit은 QTime 클래스로부터 얻어온 시간을 GUI 인터페이스 상에 표시할 수 있는 기능을 제공한다.

```
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);  
...
```



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

예수님은 당신을 사랑합니다.

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 전체 소스는 04_QTimeEdit 디렉토리를 참조하면 된다.

✓ QDateTime 클래스와 QDateTimeEdit 위젯

QDateTime 클래스는 날짜와 시간을 함께 다룰 수 있는 클래스이며 QDateTimeEdit 클래스는 날짜와 시간을 표시할 수 있다. 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" );
```

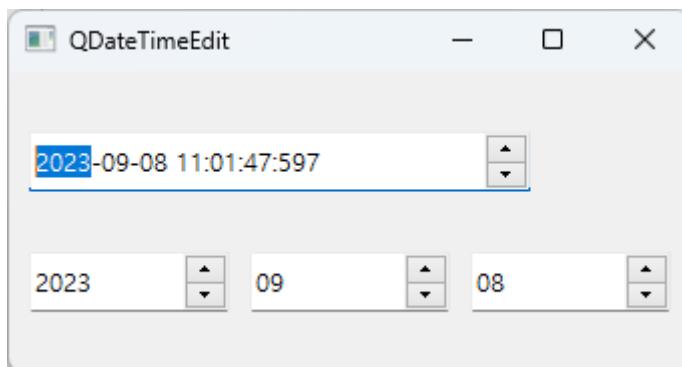
예수님은 당신을 사랑합니다.

```
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 QDateTimeEdit(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 QDateTimeEdit(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.

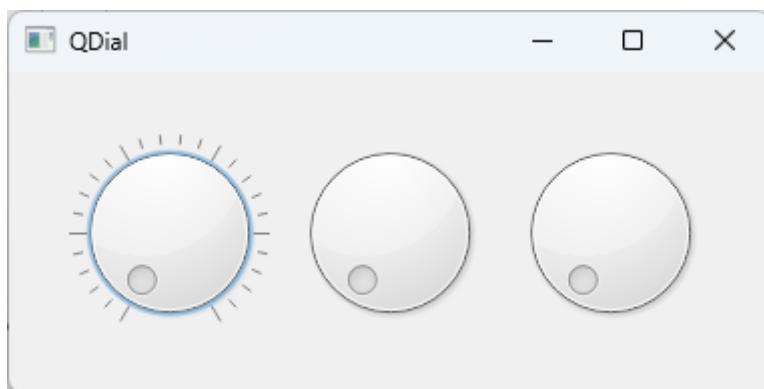


QDateTimeEdit 클래스는 날짜 외에도 setMinimumTime() 와 setMaximumTime() 멤버 함수를 이용해 시간의 최소 값과 최대 값을 지정할 수 있다. 예제 전체 소스는 05_QDateTimeEdit 디렉토리를 참조하면 된다.

✓ 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);  
...
```



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 함수가 Slot 함수라는 정도만 이해 하고 넘어가자. 자세한 사항은 Signal 과 Slot 절에서 다루도록 하겠다. 예제 소스는 06_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);

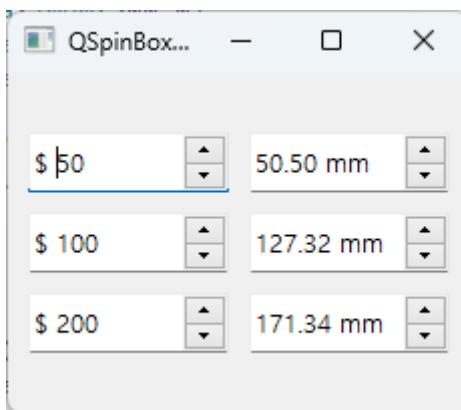
    doublespin[i] = new QDoubleSpinBox(this);
    doublespin[i]->setMinimum(10.0);
```

예수님은 당신을 사랑합니다.

```
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;
}
...
```



예제 전체 소스는 7_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.

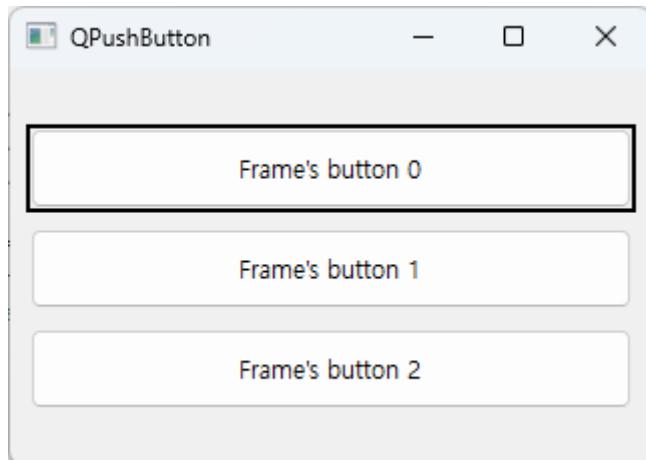
```
QPushButton *btn[3];

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);
```

예수님은 당신을 사랑합니다.

```
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.



예제 전체 소스는 08_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);
```

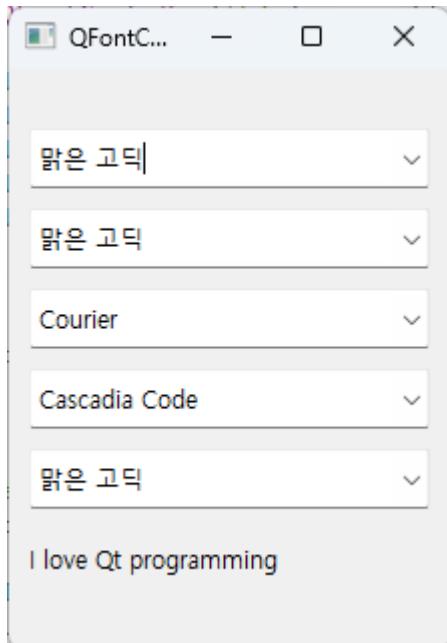
예수님은 당신을 사랑합니다.

```
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;
}
...
```

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

예제 소스는 09_QFontComboBox 디렉토리를 참조하면 된다.



아래 표는 setFontFilters() 멤버함수에서 사용 가능한 상수이다.

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

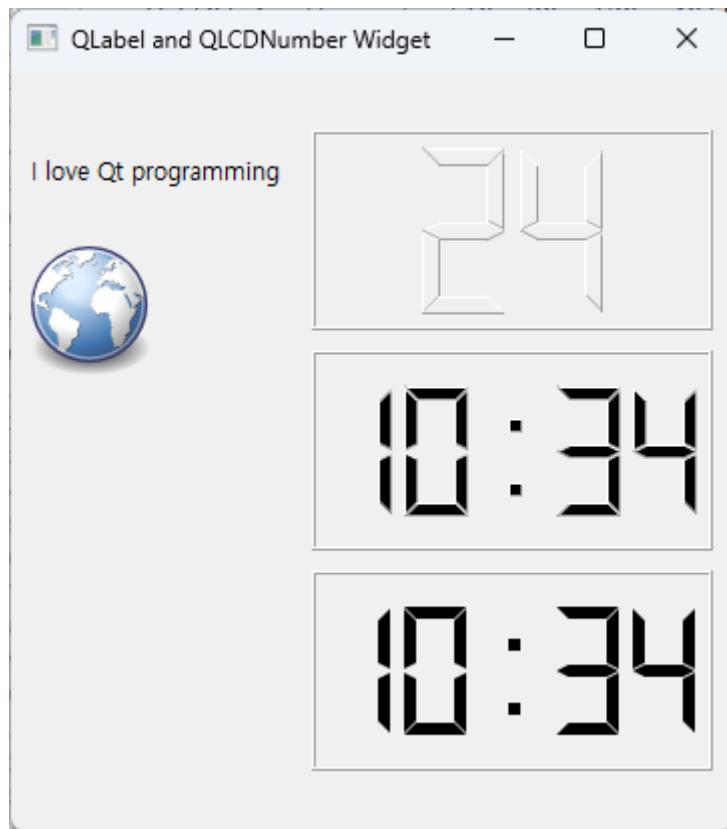
lcd[1] = new QLCDNumber(5, this);
lcd[1]->display("10:34");
lcd[1]->setGeometry(150, 140, 200, 100);
lcd[1]->setSegmentStyle(QLCDNumber::Filled);

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

예수님은 당신을 사랑합니다.

표시할 수 있다.



예제 전체 소스는 10_QLabel_QLCDNumber 디렉토리를 참조하면 된다.

✓ QLineEdit

QLineEdit 위젯은 텍스트를 입력 및 수정을 위한 GUI를 제공한다. QLineEdit 클래스 위젯에서 복사, 붙여 넣기, 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)));

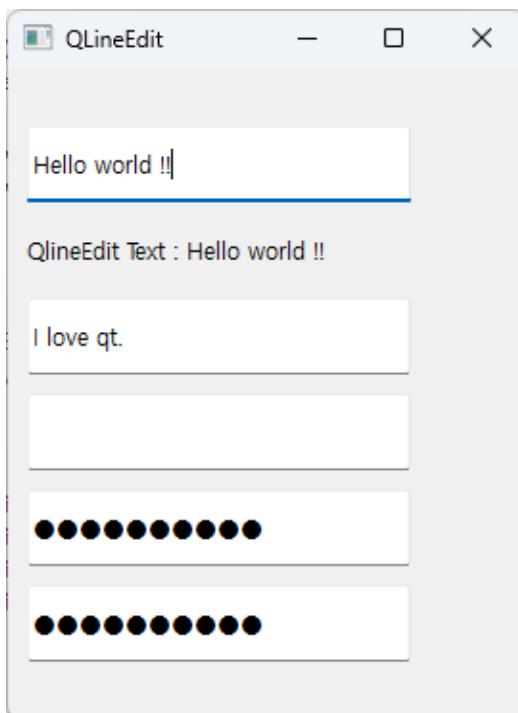
    edit[0]->setGeometry(10, 30, 200, 40);
```

예수님은 당신을 사랑합니다.

```
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));
}
...
```



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

| 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 동일하지만 포커스가 이동되면 "*" 표시 |

예제 소스는 11_QLineEdit 디렉토리를 참조하면 된다.

✓ QMenu 와 QMenuBar 클래스를 이용한 메뉴

QMenu 와 QMenuBar 클래스 위젯은 메뉴 기능을 제공한다. QMenu 위젯은 메뉴를 만들기 위해 제공하는 위젯으로써 addAction()과 addMenu() 멤버 함수를 제공한다. 아래는 이 함수들을 이용해 작성한 소스코드이다.

```
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");
```

예수님은 당신을 사랑합니다.

```
menu[2]->addAction("Page Setup");
menu[2]->addMenu(menu[1]);

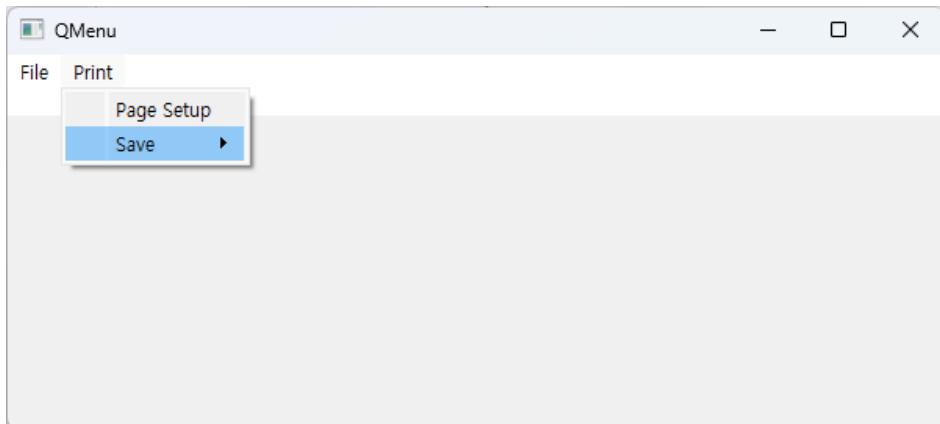
menuBar->addMenu(menu[0]);
menuBar->addMenu(menu[2]);

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.



예제 소스는 12_QMenu_QMenuBar 디렉토리를 참조하면 된다.

✓ QProgressBar

진행사항을 표시하기 위한 위젯으로 QProgressBar 위젯을 사용할 수 있으며 위젯의 배

예수님은 당신을 사랑합니다.

치 방향을 가로 또는 세로로 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);

...
```

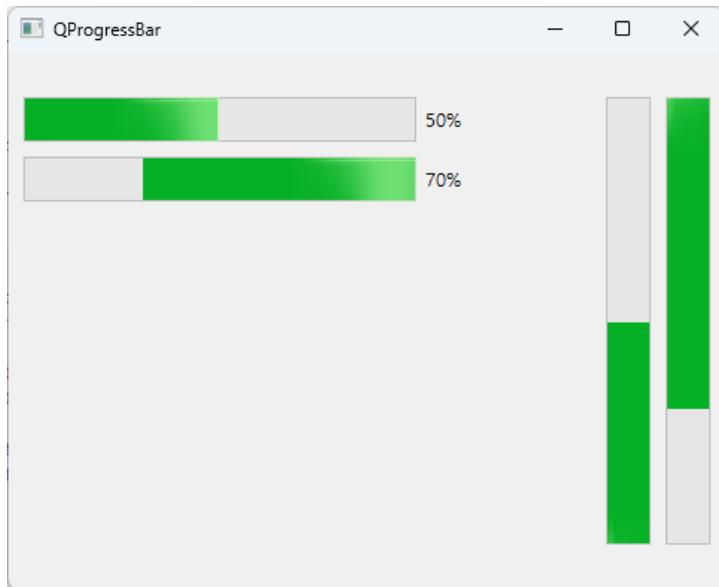
setMinimum() 와 setMaximum() 함수를 이용해 QProgressBar의 최소값과 최대값을 설정할 수 있으며 동일한 기능으로 setRange() 멤버 함수를 제공한다.

setValue() 는 QProgressBar의 최소값과 최대값 사이의 현재의 진행 값을 설정하기 위

예수님은 당신을 사랑합니다.

해 사용하는 멤버 함수이며 setOrientation() 함수는 QProgressBar를 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.



예제 소스는 13_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;
}
```

예수님은 당신을 사랑합니다.

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

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

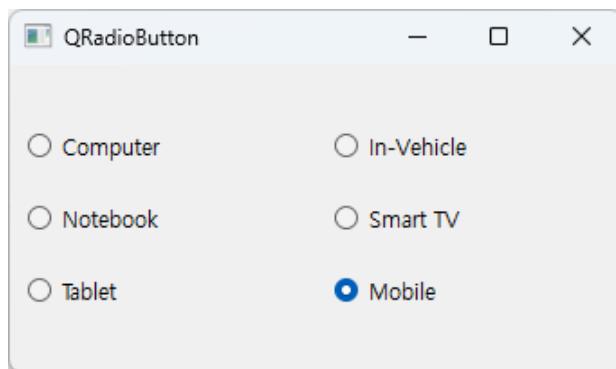
    if(i == 2)
        radio2[i]->setChecked(true);

    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]);
...
```



예제 실행 화면에서 보는 것과 같이 6개 항목에서 왼쪽 3개 항목을 그룹으로 지정함으로써 오른쪽 QRadioButton 들과 분리할 수 있다. 이 예제 소스는 14_QRadioButton 디렉토리를 참조하면 된다.

예수님은 당신을 사랑합니다.

✓ QScrollArea

QScrollArea 위젯은 GUI가 표시되는 윈도우 상에서 위젯들을 모두 표시할 수 없는 경우 스크롤 바를 이용해 가려진 부분으로 이동하는 방식으로 GUI를 모두 표시할 수 있는 기능을 제공한다.

예를 들어 어떤 이미지를 축소하지 않고 화면에 표시 하고자 할 때 GUI 위젯의 크기가 부족하게 되면 좌측 또는 하단에 스크롤이 생겨 마우스로 스크롤을 이동하면서 이미지를 볼 수 있는 것과 같은 기능을 제공한다.

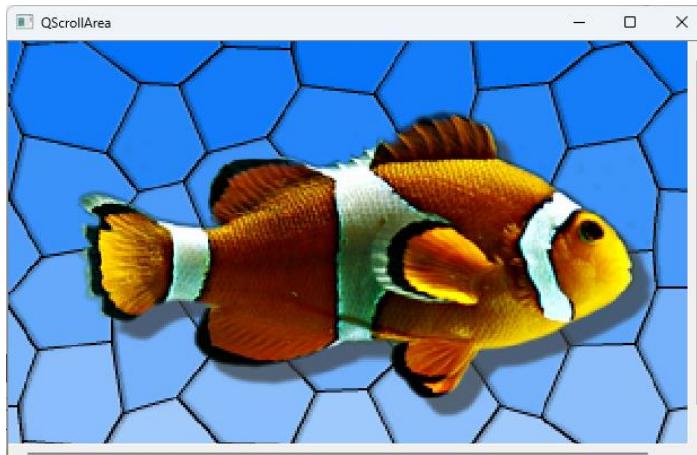
```
QImage image;
QScrollArea *area;

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 클래스의 image 오브젝트는 이미지를 랜더링(표시) 한다. 랜더링한 이미지를 QLabel 클래스의 lbl 오브젝트에 보여지도록 한다.

그리고 QScrollArea 영역 안에 lbl 오브젝트의 이미지가 보여지도록 하기 위해서 QScrollArea에서 제공하는 setWidget() 멤버함수를 이용한다. 만약 이미지가 QScrollArea 위젯보다 큰 경우 자동으로 스크롤 바가 활성화 된다.



예수님은 당신을 사랑합니다.

예제 소스는 15_QScrollArea 디렉토리를 참조하면 된다.

✓ QScrollBar

QScrollBar 위젯 클래스는 슬라이더 위젯의 모양과 비슷하다. QScrollBar 위젯은 좌우 혹은 상하 위치 시킬 때 세로 방향 혹은 가로 방향으로 직접 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);
```

예수님은 당신을 사랑합니다.

```
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());
}

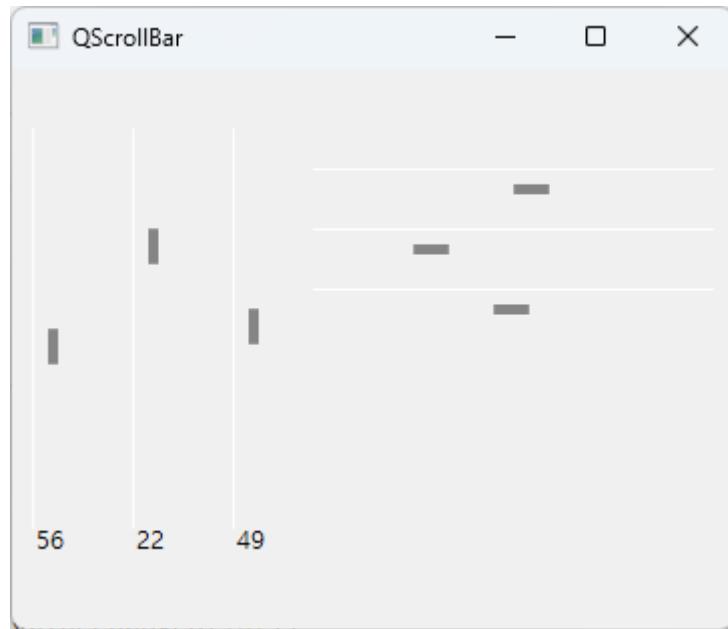
void Widget::valueChanged2(int value)
{
    lbl[1]->setText(QString("%1").arg(value));
    hscrollbar[1]->setValue(vscrollbar[1]->value());
}

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 인

예수님은 당신을 사랑합니다.

자로 받은 값으로 변경한다. 그리고 hscrollbar[1] 의 값을 변경한다.



예제 전체 소스는 16_QScrollBar 디렉토리를 참조하면 된다.

✓ QSizeGrip

QSizeGrip 클래스 위젯은 한정된 크기의 윈도우 영역 안에 위젯의 크기를 조절할 수 있다. 예를 들어 MS Windows의 탐색기와 같이 왼쪽에는 트리 영역, 오른쪽은 파일 및 디렉토리 속성을 보여주는 GUI 상에서 트리 영역의 경계선을 마우스로 드래그하여 줄 이거나 늘릴 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);

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();

};

...
```

Widget 클래스는 부모 클래스(윈도우 위젯)로 선언하고 SubWindow 클래스는 Widget 윈도우의 자식(Child) 윈도우 위젯이다. 소스 코드 상에서 QVBoxLayout 은 수직 방향의 레이아웃을 이용해 위젯을 배치할 수 있다.

레이아웃은 다음 장에서 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[])
{
```

예수님은 당신을 사랑합니다.

```
QApplication a(argc, argv);

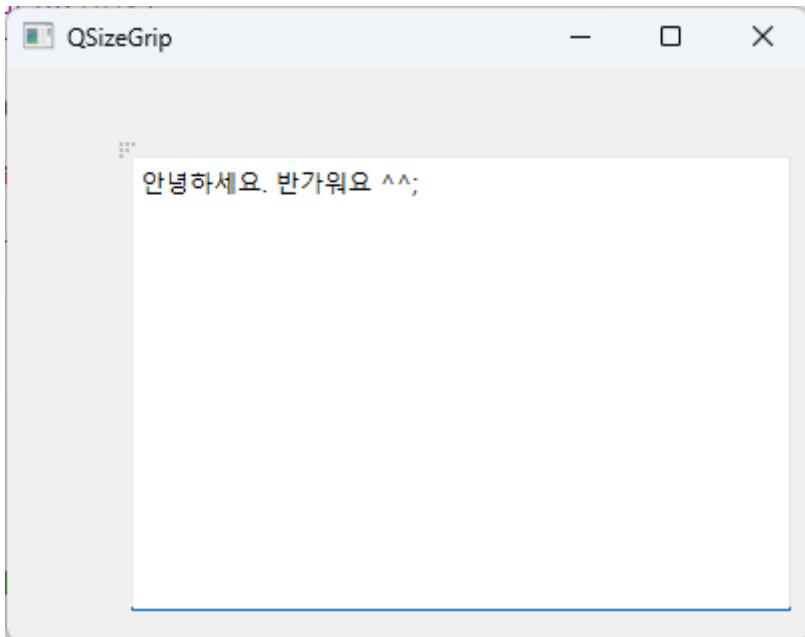
Widget w;
w.resize(400, 300);

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.



예제 소스는 17_QSizeGrip 디렉토리를 참조하면 된다.

✓ QSlider

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

예수님은 당신을 사랑합니다.

setRange() 멤버 함수를 사용할 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(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)));
}

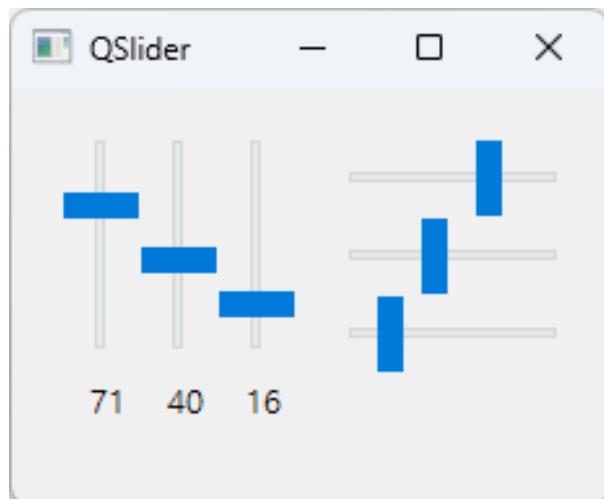
void Widget::valueChanged1(int value)
{
    lbl[0]->setText(QString("%1").arg(value));
    slider[3]->setValue(slider[0]->value());
}

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.



예제 소스는 18_QSlider 디렉토리를 참조하면 된다.

✓ QTabWidget

많은 위젯을 배치할 경우 또는 윈도우의 크기가 제한적일 때 이 위젯을 사용하면 유용하다. 이 위젯은 제한된 크기에 모든 탭을 표시할 수 없을 경우 동적으로 페이지를 이

예수님은 당신을 사랑합니다.

동할 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);

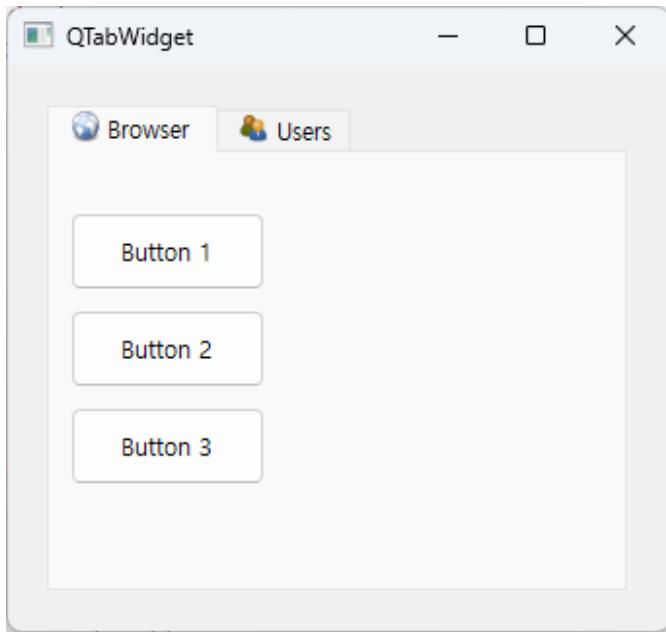
    QStringList btn_str[3] = {"Button 1", "Button 2", "Button 3"};
    QPushButton *btn[3];

    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);
}
...
```

예수님은 당신을 사랑합니다.



예제 전체 소스는 19_QTabWidget 디렉토리를 참조하면 된다.

✓ QToolBar 와 QAction

QToolBar 위젯은 윈도우 상에 툴 메뉴 바와 같은 GUI를 제공한다. QToolBar 위젯은 addAction() 멤버 함수를 이용해 툴바 상에 메뉴를 추가할 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);

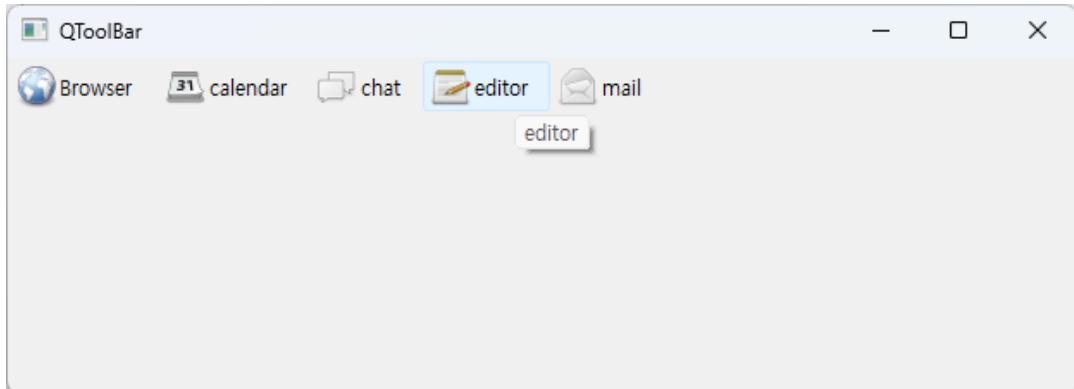
act[0]->setShortcut(Qt::Key_Control | Qt::Key_E);
act[0]->setToolTip("This is a ToolTip.");

toolbar->setToolButtonStyle(Qt::ToolButtonTextBesideIcon);

for(int i = 0 ; i < 5 ; i++)
{
```

예수님은 당신을 사랑합니다.

```
    toolbar->addAction(act[i]);  
}  
...
```



이전 그림에서 보는 것과 같이 QAction 메뉴 버튼의 아이콘만 보이게 하거나 아이콘과 버튼 이름을 같이 표시할 수 있도록 다음과 같은 상수 값을 제공한다. 아래 상수 값을 설정하기 위해서 setToolButtonStyle() 멤버 함수를 사용하면 된다.

| 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 소스는 20_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

예수님은 당신을 사랑합니다.

버튼을 클릭하면 호출되는 Slot 함수 내에 update() 멤버 함수를 사용 하면 paintEvent() virtual 함수가 호출 된다. 따라서 QWidget 의 Paint 영역을 다시 Drawing 한다.

QWidget 클래스는 resizeEvent() virtual 함수를 제공한다. 이 virtual 함수는 QWidget 의 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 소스코드 이다.

```
#include "widget.h"

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

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

void Widget::paintEvent(QPaintEvent *event)
{
    Q_UNUSED(event);
    QString img_full_name;

    QPainter painter(this);

    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)
{
    qDebug() << "[Mouse Press] x, y : " << event->pos();
}

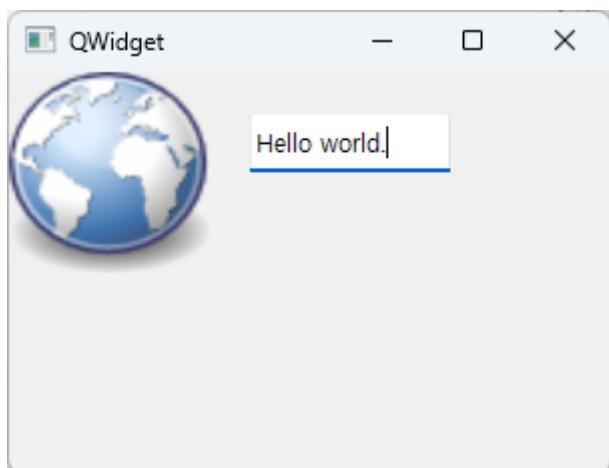
void Widget::mouseReleaseEvent(QMouseEvent *event)
{
    qDebug() << "[Mouse Release] x, y : " << event->pos();
}

void Widget::mouseDoubleClickEvent(QMouseEvent *event)
```

```
{  
    qDebug() << "[Mouse Double clicked] x, y : " << event->pos();  
}  
  
void Widget::mouseMoveEvent(QMouseEvent *event)  
{  
    qDebug() << "[Mouse Move] x, y : " << event->pos();  
}  
  
void Widget::keyPressEvent(QKeyEvent *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.");
}

Widget::~Widget()
{
```



예제 소스는 21_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.

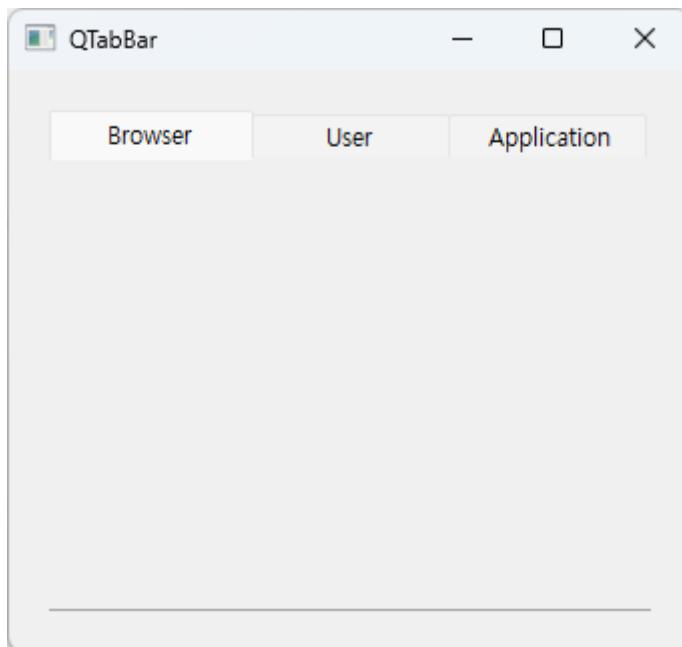
```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    QTabBar *tab = new QTabBar(this);
    tab->addTab("Browser");
    tab->addTab("User");
```

예수님은 당신을 사랑합니다.

```
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);
}
...
```



QTabBar는 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 제공하는 탭 모양의 값들이다.

| 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 22_QTabBar 디렉토리를 참조하면 된다.

✓ QToolBox

QToolbox 클래스 위젯은 위젯 아이템들을 새로 방향 텁 컬럼 형태로 GUI를 제공한다. 각 텁의 이름으로 텍스트와 아이콘을 사용할 수 있다.

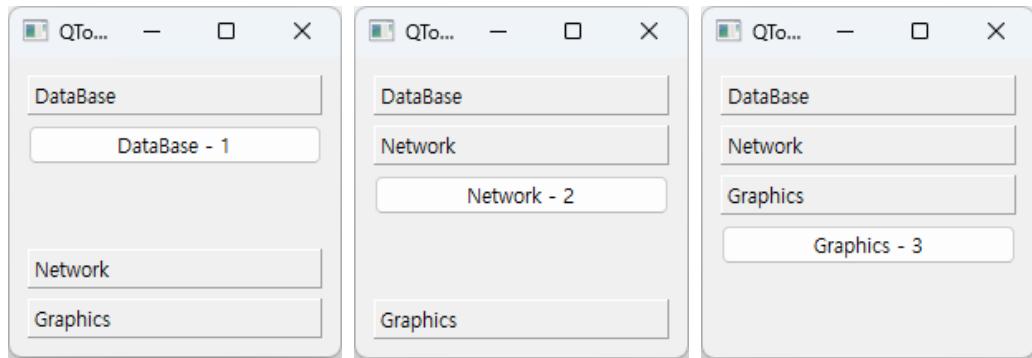
```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    box = new QToolBox(this);
    lay = new QHBoxLayout(this);

    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);
}
...
```



예제 소스는 23_QToolBar 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 사용할 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"));

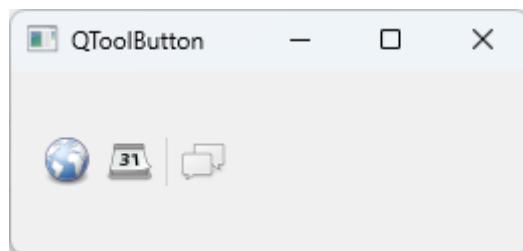
    tool->addWidget(button);
    tool->addWidget(button1);
    tool->addSeparator();
```

예수님은 당신을 사랑합니다.

```
tool->addWidget(button2);
layout->addWidget(tool);

this->setLayout(layout);
}

...
```

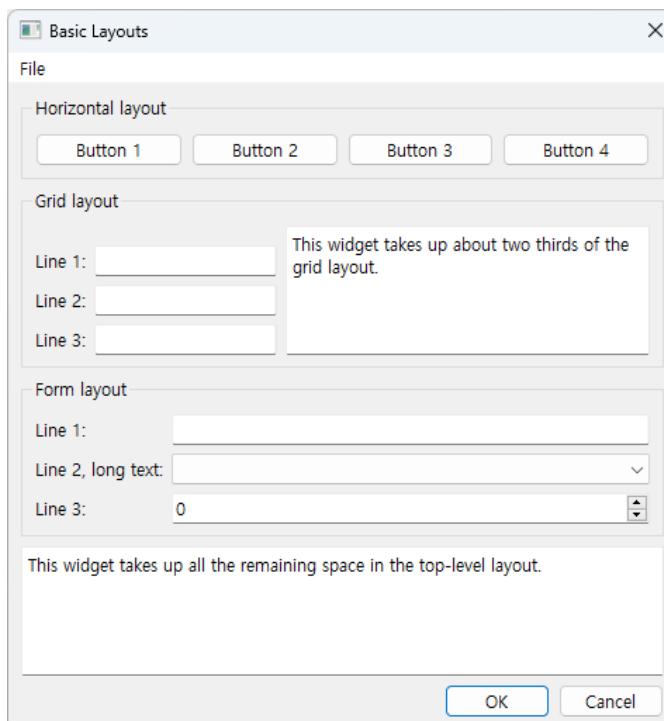


예제 소스는 24_QToolBox 디렉토리를 참조하면 된다.

6. Layout

QWidget 클래스의 setGeometry() 멤버 함수를 이용해 GUI 상에서 특정 X, Y 좌표로 위젯을 배치하게 되면 윈도우의 크기가 변경될 때 위젯의 위치가 변경되지 않는다.

하지만 레이아웃을 이용하면 윈도의 크기가 될 때마다 동적으로 GUI 상에 위젯들의 크기도 동적으로 변경된다.



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

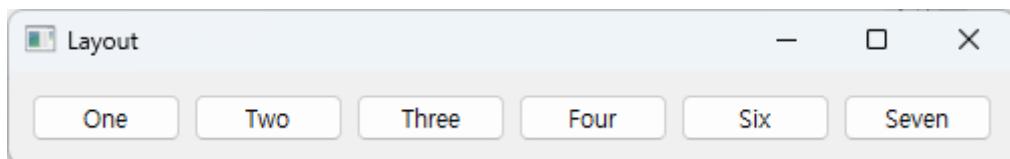
위젯을 2열로 배치하는 형식.

✓ 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]);
}
```



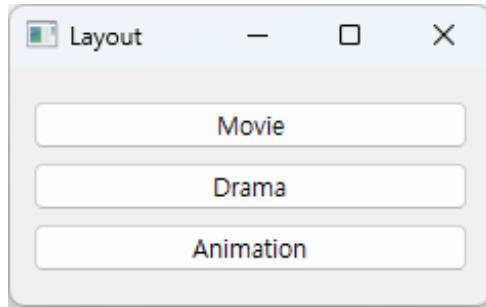
✓ QVBoxLayout

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]);
}
```

예수님은 당신을 사랑합니다.



✓ QGridLayout

QGridLayout은 바둑판 모양과 같은 스타일로 위젯을 배치할 수 있다. QGridLayout 은 특정 행을 하나의 위젯만 배치할 수 있도록 병합 하거나 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);
```



✓ 중첩된 레이아웃을 사용

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

```
...
```

```
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    QBoxLayout *hboxLayout = new QHBoxLayout();
    QPushButton *btn[6];

    QStringList 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];
    QStringList vbtnStr[3] = {"Movie", "Drama", "Animation"};

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

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

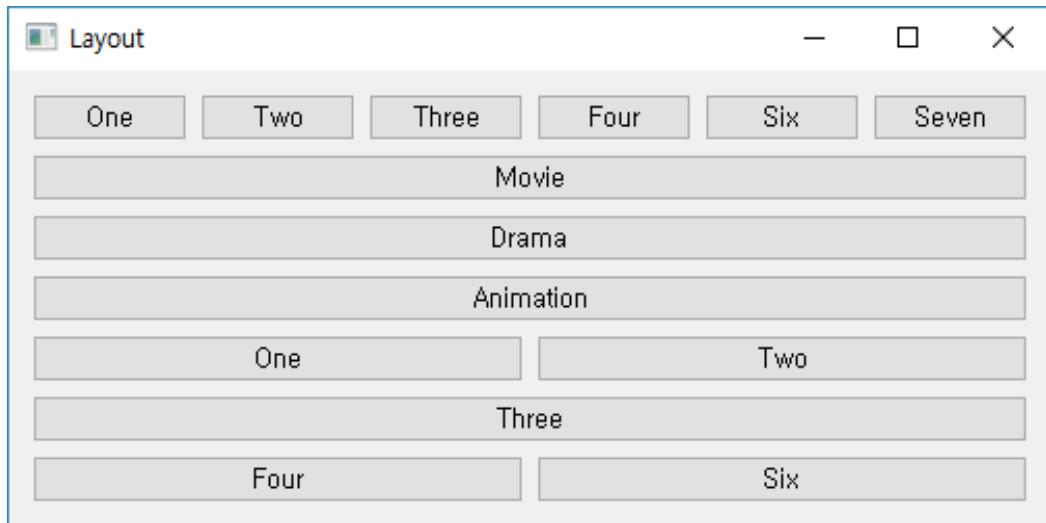
    for(int i = 0 ; i < 5 ; i++)
        gbtn[i] = new QPushButton(btnStr[i]);

    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);

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

    setLayout(defaultLayout);
}
```

예수님은 당신을 사랑합니다.



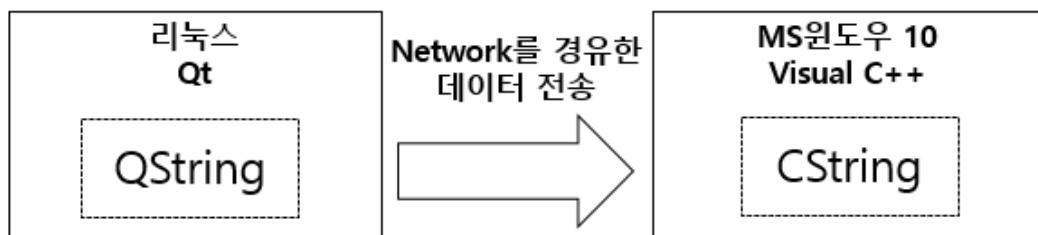
예제 소스는 00_Layout 디렉토리를 참조하면 된다.

7. Qt에서 제공하는 데이터 타입과 클래스

Qt는 개발자의 편의성을 위해 다양한 데이터타입을 제공한다. 예를 들어 QString 과 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() 함수를 제공하며 최대/최소 사이의 값을 구하기 위해 qBound() 함수를 제공한다. 최소값을 구하기 위한 qMin()을 제공하며 최대 값을 구하기 위한 qMax() 함수를 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
```

✓ 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 클래스 이외에도 데이터 스트림, 멀티 바이트 캐릭터 형태의 유니코드 4.0(Unicode Standard Version) 캐릭터를 지원하는 다양한 클래스를 제공한다.

| 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 지원 하는 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/READ 를 위한 STREAM 기능 제공 |
| QDataStream | Binary 기반 WRITE/READ 를 위한 STREAM 기능 제공 |

✓ QByteArray

QByteArray 클래스는 바이트(8-bit) 단위의 배열을 제공한다. QByteArray 클래스는 배열을 핸들링 하기 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);
```

예수님은 당신을 사랑합니다.

```
//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 클래스이다.

```
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 클래스이다.

```
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

이 클래스는 다양한 언어 Character set으로 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

예수님은 당신을 사랑합니다.

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"
```

✓ **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();
    ...
}
```

✓ 데이터 클래스

Qt에서 제공하는 기본 데이터 타입과 더불어 유연한 데이터 조작을 위해 다양한 클래스를 제공한다. 예를 들어 Bit 단위의 데이터를 다루기 위한 `QByteArray`, 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 |
|--------------------------|---|
| <code>QByteArray</code> | 비트 연산(AND, OR, XOR, NOT)을 제공하기 위한 bit Array |
| <code>QMargins</code> | left, top, right, bottom 사각형의 각 Margin을 처리하기 위한 클래스 |
| <code>QPoint</code> | X, Y, Z 값을 처리하기 위해 제공하는 클래스 |
| <code>QQuaternion</code> | 벡터 및 스칼라로 구성된 Quaternion을 처리하기 위한 클래스 |
| <code>QRect</code> | 사각형의 left (qint32), top (qint32), right (qint32), bottom (qint32) |
| <code>QRegExp</code> | 정규화 표현식을 처리하기 위한 클래스 |
| <code>QRegion</code> | Painter 상에서 클립보드 영역을 정의하기 위해 사용 |
| <code>QSize</code> | 넓이와 높이를 저장 할 수 있는 클래스 |
| <code>QVariant</code> | void 타입으로 저장 할 수 있는 union 형태의 클래스 |
| <code> QVector2D</code> | 2차원 공간에서 Vector 또는 vertex를 나타내기 위한 클래스 |
| <code> QVector3D</code> | x, y, z 3개의 좌표를 사용할 수 있는 클래스 |
| <code> QVector4D</code> | 4차원 공간에서 Vector 또는 vertex를 나타내기 위해 사용. |

✓ `QByteArray`

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

```
QByteArray ba(200);

QByteArray 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 아래과 같이 사용할 수 있다.

```
QPoint p( 3, 7);
QPoint q(-1, 4);

p += q; // p becomes (2, 11)
```

✓ QQuaternion

QQuaternion는 벡터 및 스칼라로 구성된 Quaternion을 사용할 수 있다. Quaternion은
페이지 115 7. Qt에서 제공하는 데이터 타입과 클래스

예수님은 당신을 사랑합니다.

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) },
    ...
}
```

8. Container Classes

Container 클래스는 특정 유형의 데이터를 집합 또는 배열 형태로 저장하는데 사용한다. 예를 들어 QString으로 저장해야 할 항목이 여러 개 있다면 QList<QString>과 같은 Container 클래스를 사용할 수 있다.

Container 클래스들은 STL에서 제공하는 Container들보다 사용하기 쉽고 안전하다. 또한 경량화 되어 있다. 따라서 Qt에서 제공하는 Container는 C++의 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 Qt 6에서는 QList로 통합되었다. |
| 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 소스코드이다.

```
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;
```

예수님은 당신을 사랑합니다.

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

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

✓ 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
```

9. Signal and 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.

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

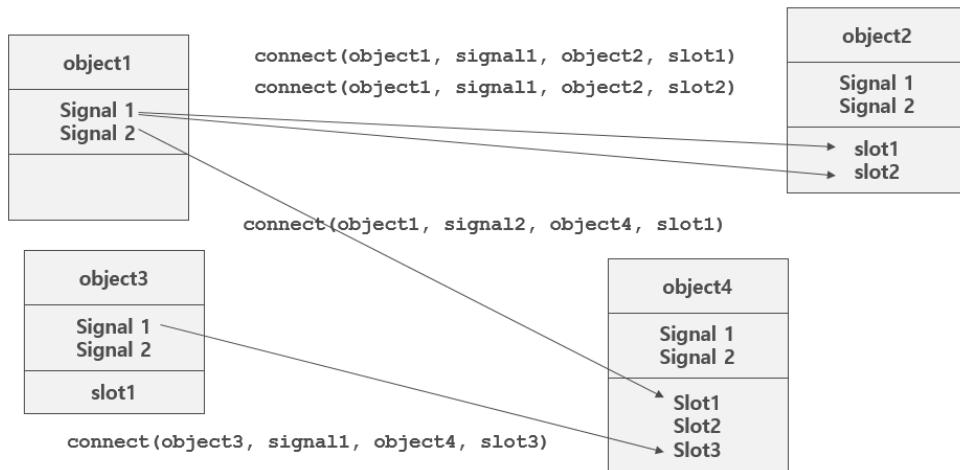
예를 들어 Qt로 채팅 프로그램에서 A라는 사용자가 B라는 사용자에게 메시지를 보낸다고 가정해보자 B의 입장에서 A에게로부터 메시지를 받은 행위는 시그널(Signal)이라고 정의할 수 있다.

그리고 메시지를 받은 시그널과 연결된 슬롯(Slot) 함수를 호출한다. Qt는 모든 이벤트 처리를 시그널과 슬롯을 사용한다. 방금 예로든 GUI 및 네트워크모듈뿐만 아니라 Qt에서 제공하는 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, 귀속말 처리 쓰레드, 새로운 사용자가 등록되면 알려주는 쓰레드 등 여러 개의 쓰레드를 사용해야 하기 때문에 프로그램이 복잡해 질 수 있다. 하지만 Qt에서 시그널과 슬롯을 사용하면 쓰레드를 사용하지 않고도 간단하게 채팅 프로그램을 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.



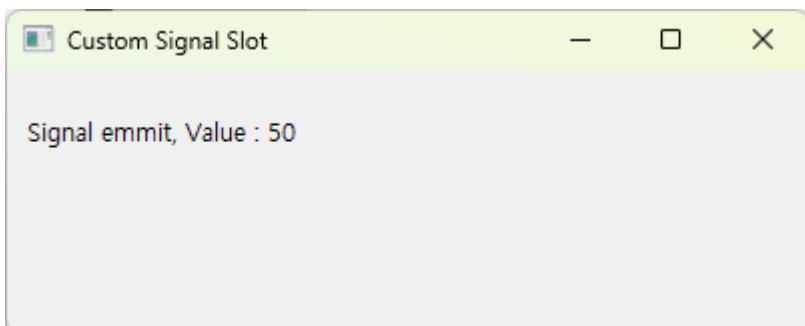
<그림> Signal 과 Slot 의 Architecture

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.



예수님은 당신을 사랑합니다.

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 emit이라는 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);  
};
```

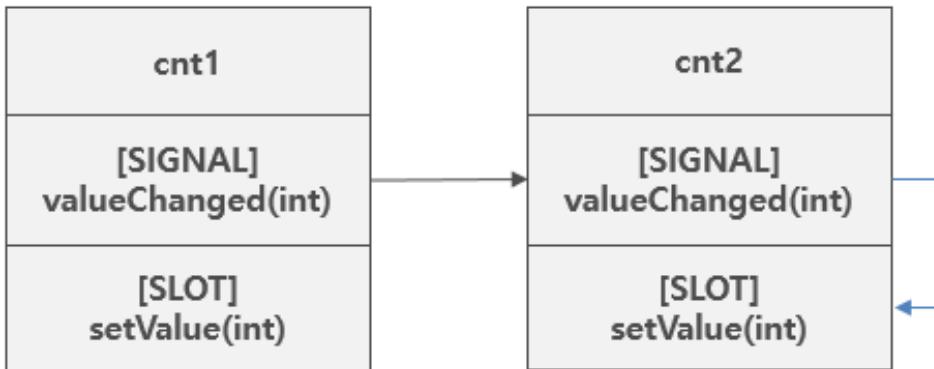
Widget 클래스의 하단에 보면 public 키워드에 slots 키워드를 함께 사용한 것을 확인 할 수 있을 것이다. slots 키워드는 private 또는 public 키워드와 함께 사용할 수 있다. slots 라는 키워드를 사용하여 명시한 멤버 함수는 Slot 함수를 정의할 수 있다. 다음 예제 소스코드는 widget.cpp 소스코드이다.

```
#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("Signal emmit, 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.



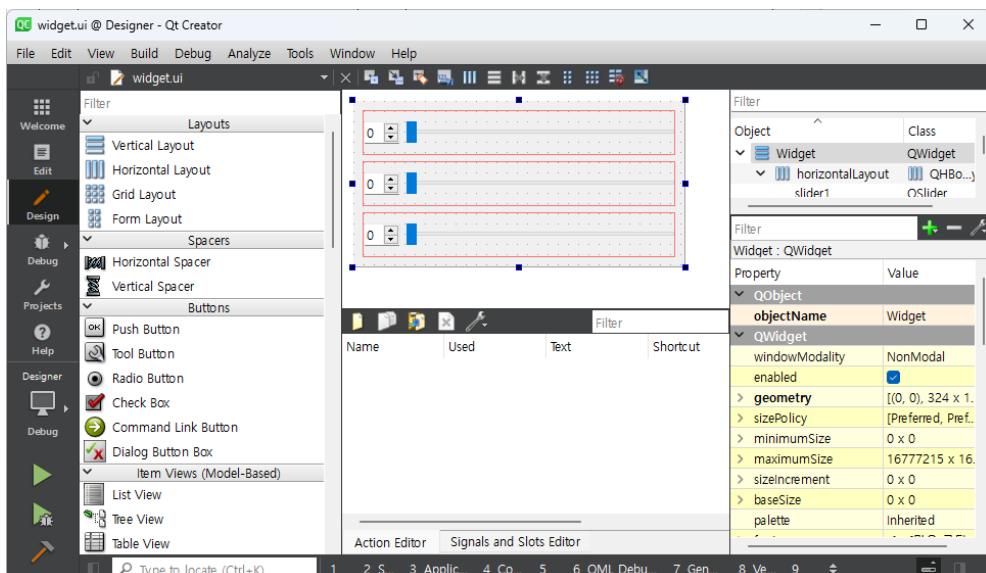
```
cnt1 = new Counter("counter 1");
cnt2 = new Counter("counter 2");

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 00_CustomSignalSlot 디렉토리를 참조하면 된다.

10. 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.



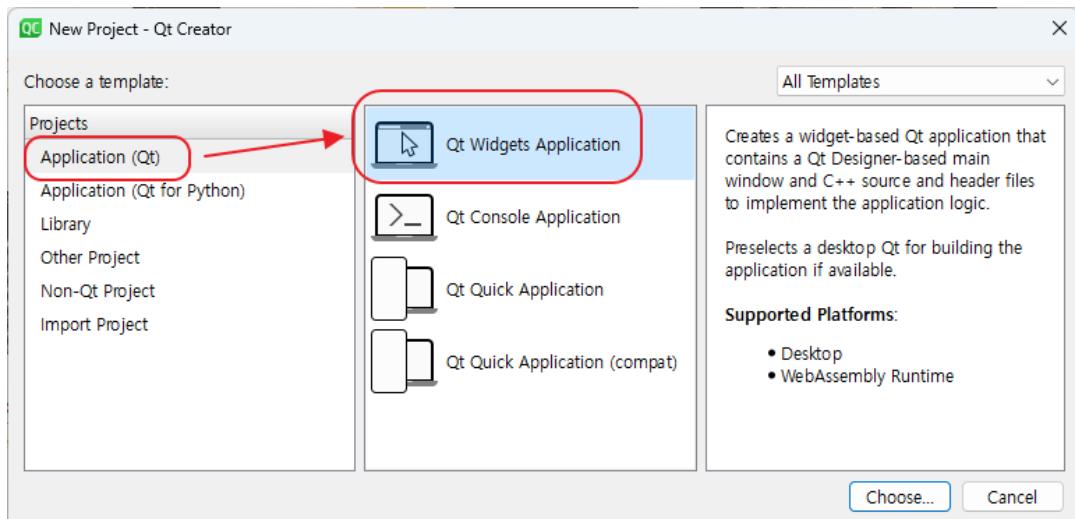
확장자가 ui 파일인 GUI 파일은 아래 소스코드와 같이 XML 포맷으로 되어 있다. 이 파일은 사용자가 마우스를 이용해 위젯을 배치하면 Qt Creator 의 Designer 툴이 자동으로 XML 로 작성한다. 그리고 빌드를 하게 되면 XML 로 되어 있는 GUI 파일을 C++ 소스코드로 변환 후 빌드 한다.

```
<?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">
...
...
```

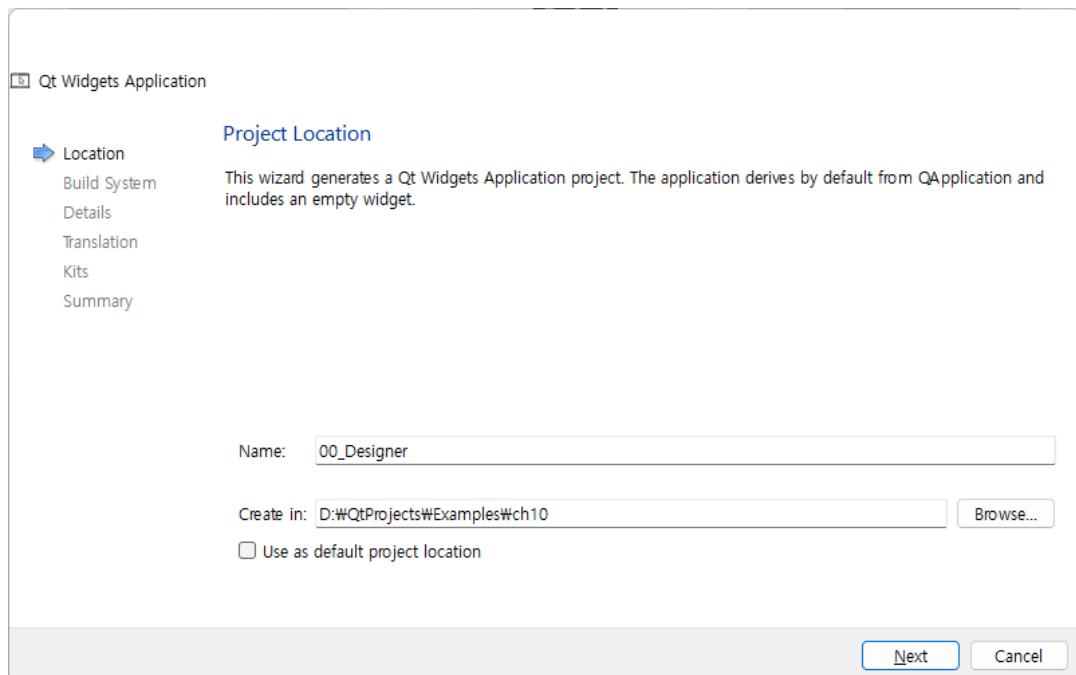
✓ Qt Designer 를 이용한 예제

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.

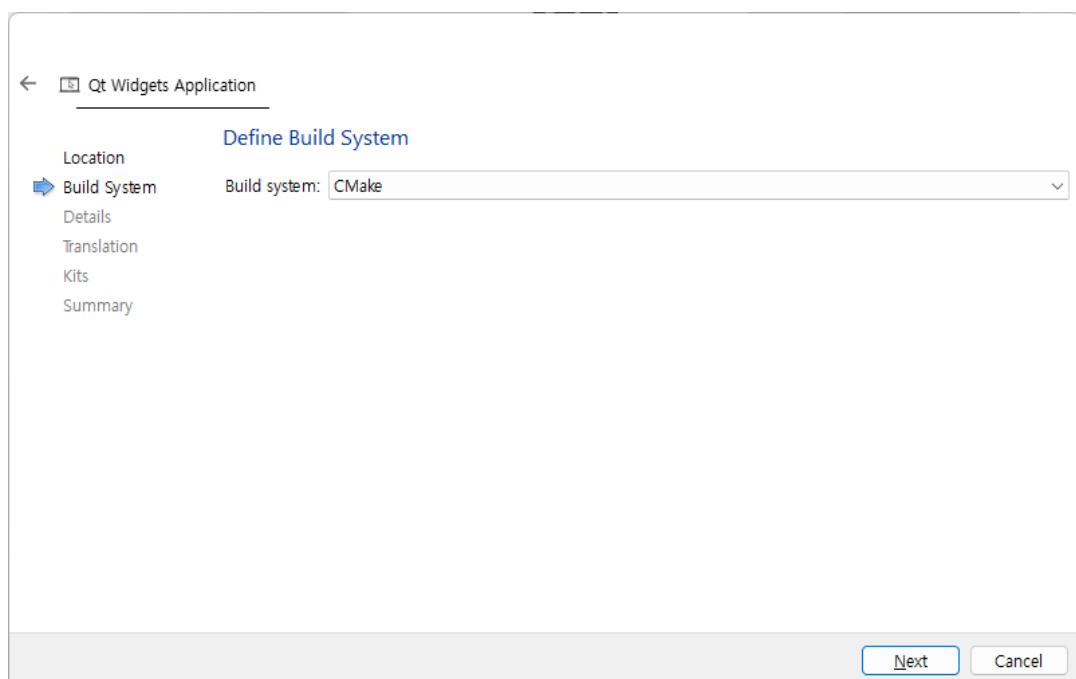


그런 다음 중앙에 항목 중 [Qt Widgets Application] 을 선택하고 하단의 [Choose] 버튼을 클릭한다. 다음 화면은 프로젝트 이름과 생성되는 위치를 입력하고 [Next] 버튼을 클릭한다.

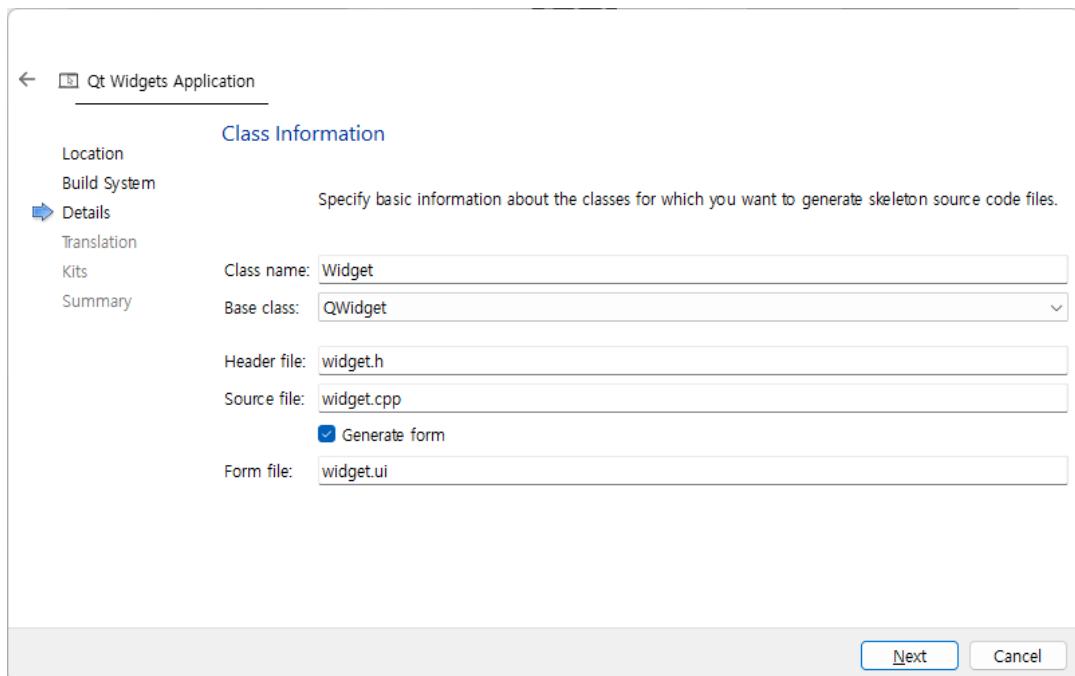
예수님은 당신을 사랑합니다.



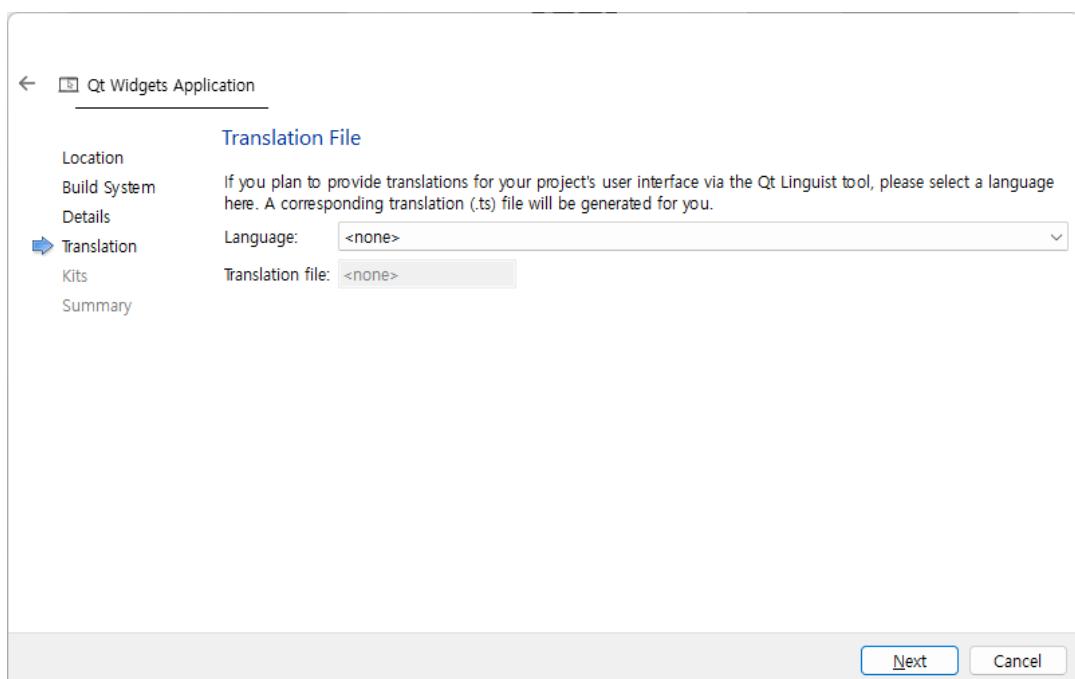
빌드 도구로 qmake 를 선택해도 되지만 여기서는 CMake 를 선택한다.



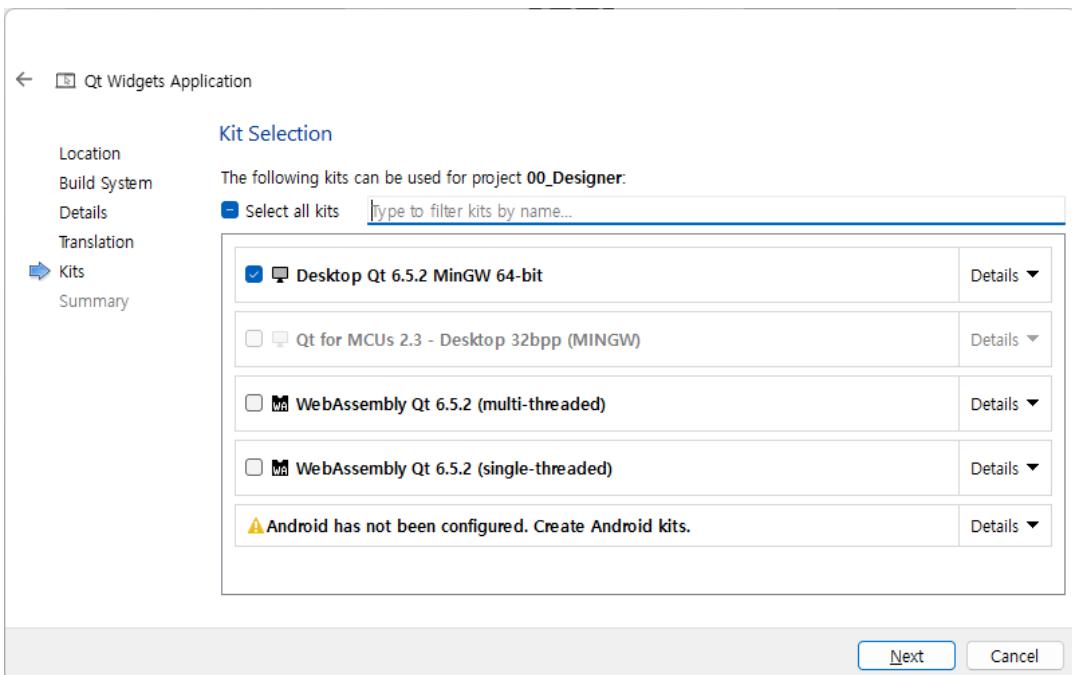
다음으로 아래와 같이 Class Information 을 입력하고 하단의 [Next] 버튼을 클릭한다.



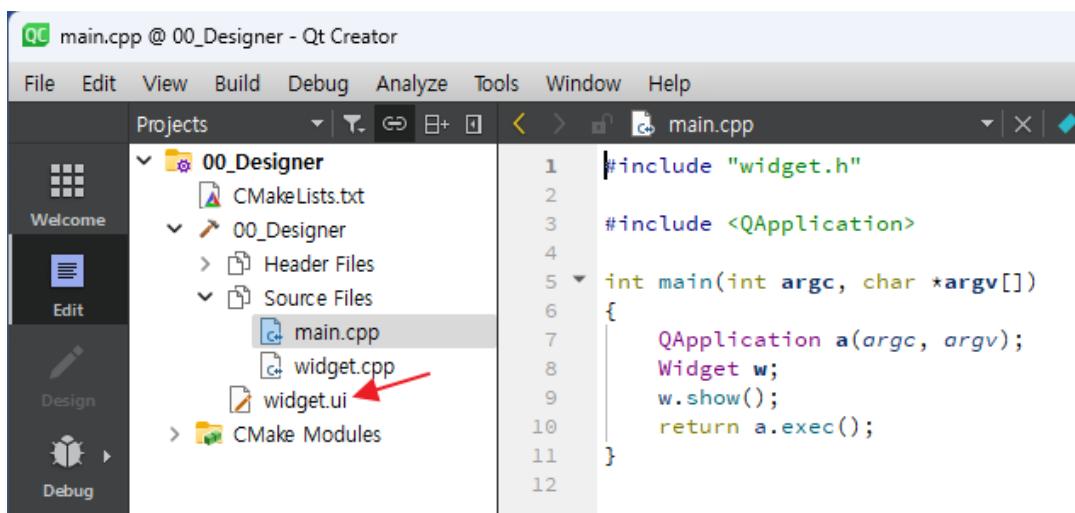
다음 화면에서는 아무것도 선택하지 않고 [Next]버튼을 클릭한다.

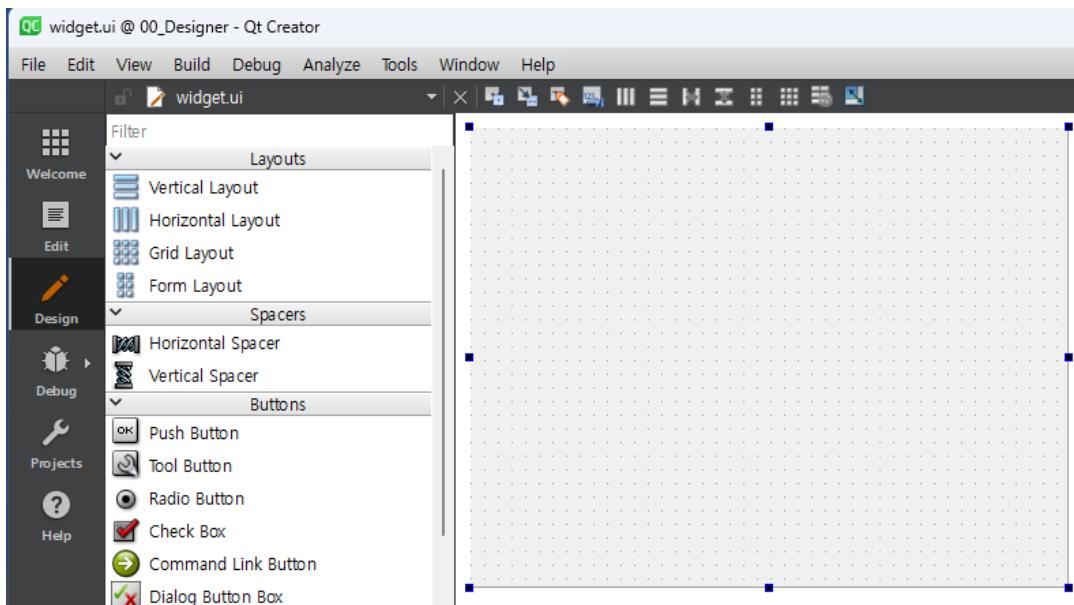


다음은 어플리케이션을 빌드할 컴파일러를 선택한다. MinGW 컴파일러를 선택한다. 하나를 선택한다. 선택을 완료한 후 하단의 [Next] 버튼을 클릭한다.

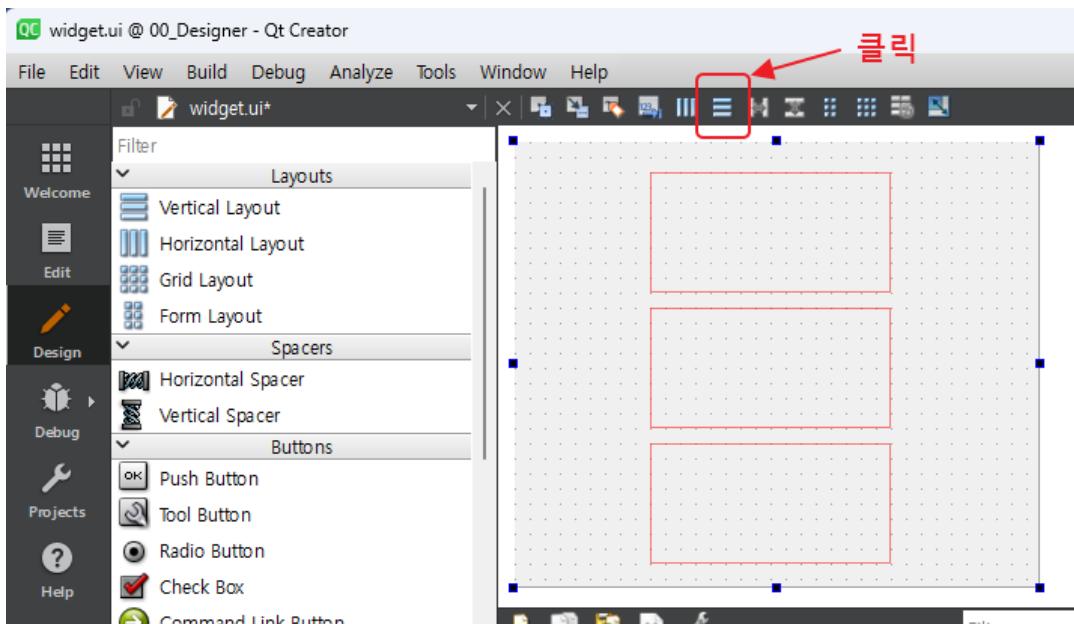


다음 다이얼로그 창에서는 하단의 [Finish] 버튼을 클릭하면 프로젝트 생성이 완료된다. 프로젝트 생성이 완료되면 다음 그림에서 보는 것과 같이 Qt Creator 좌측에 프로젝트 창에서 widget.ui 소스코드를 더블 클릭하면 Designer 화면으로 전환된다.

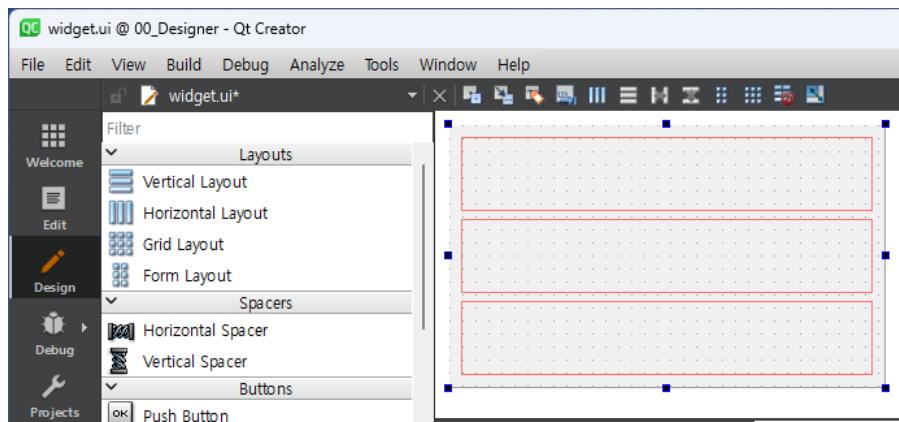




위와 것과 같이 Designer 툴로 전환한 화면에서 위젯들을 배치해 보자. 첫 번째로 좌측 [Layout] 탭에서 3 개의 Horizontal Layout 을 마우스로 드래그 해 GUI 위젯에 배치한다. 그리고 다음 그림에서 보는 것과 같이 상단의 툴 바에서 세로모양으로 직사각형 3 개가 쌓여 있는 아이콘 버튼을 클릭한다.



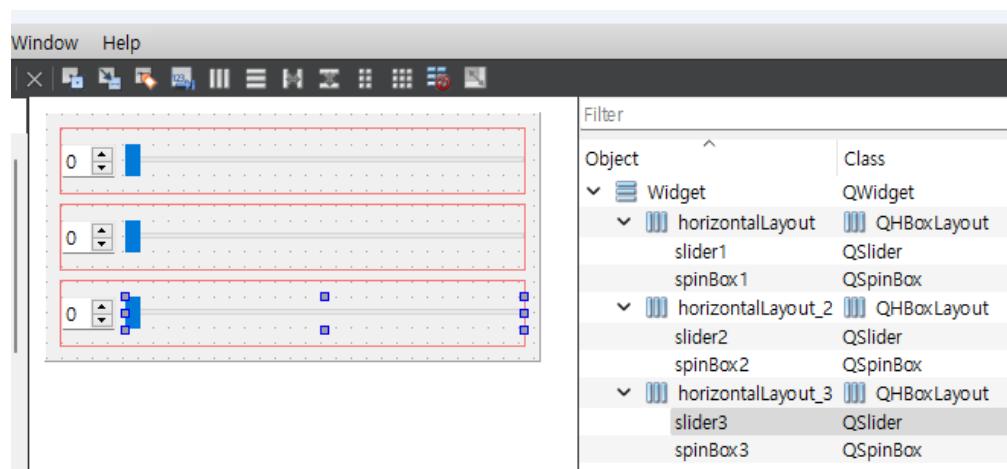
위의 그림에서 보는 것과 같이 아이콘을 클릭하면 윈도우 상에 Horizontal Layout 이 윈도우 크기가 변경되면 Horizontal Layout 이 자동으로 크기를 조절한다.



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

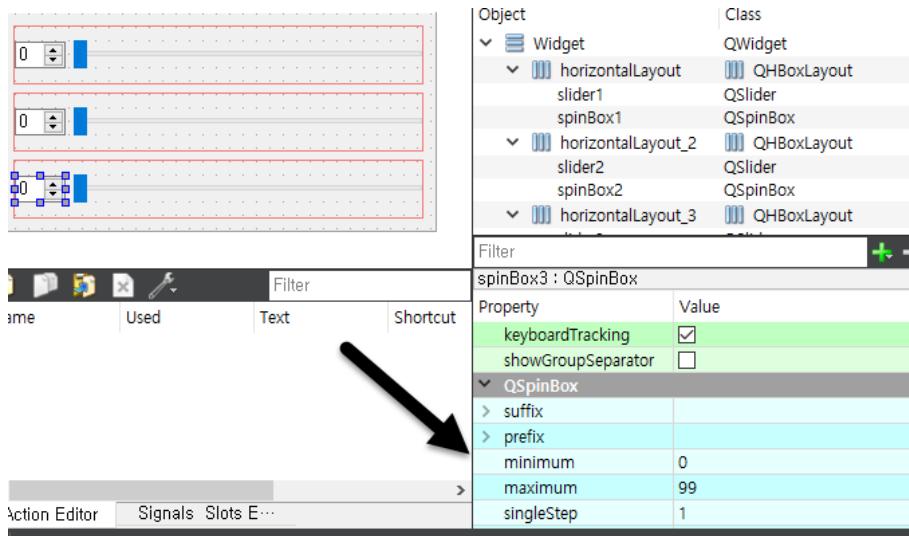
| Layout | Class | Object Name |
|------------------------|----------|-------------|
| 첫 번째 Horizontal Layout | QSpinBox | spinBox1 |
| | QSlider | slider1 |
| 두 번째 Horizontal Layout | QSpinBox | spinBox2 |
| | QSlider | slider2 |
| 세 번째 Horizontal Layout | QSpinBox | spinBox3 |
| | QSlider | slider3 |

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



예수님은 당신을 사랑합니다.

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



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



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;  
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 Designer 에서 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);  
}
```

```
}

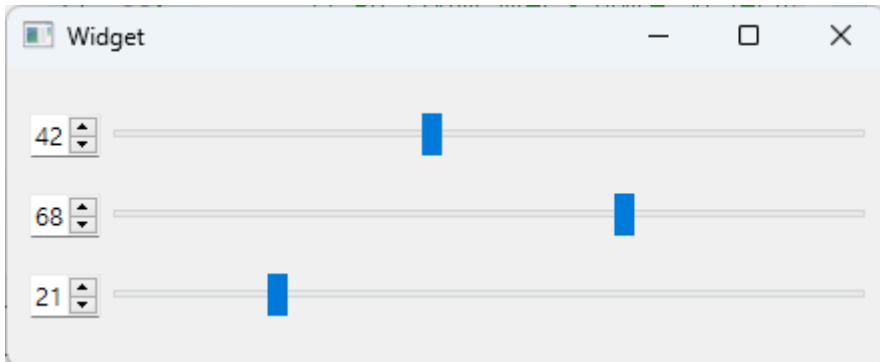
void Widget::slider2_valueChanged(int value)
{
    ui->spinBox2->setValue(value);
}

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. 다음은 지금까지 작성한 예제를 실행한 화면이다.



예제 전체 소스코드는 00_Designer 디렉토리 참조하면 된다.

11. 다이얼로그

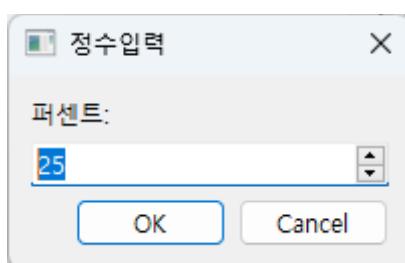
다이얼로그는 어플리케이션이 동작 중에 이벤트가 발생했을 때 사용자에게 메시지를 전달하기 위한 목적으로 사용된다. 그리고 사용자로부터 입력 값을 받거나 여러 개 중 하나를 선택할 수 있는 GUI를 제공한다. 다음 표는 Qt에서 제공하는 다이얼로그 중 자주 사용되는 다이얼로그이다.

| 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);
```

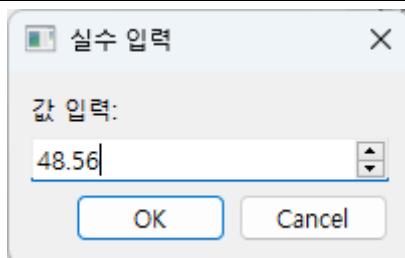


예수님은 당신을 사랑합니다.

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.

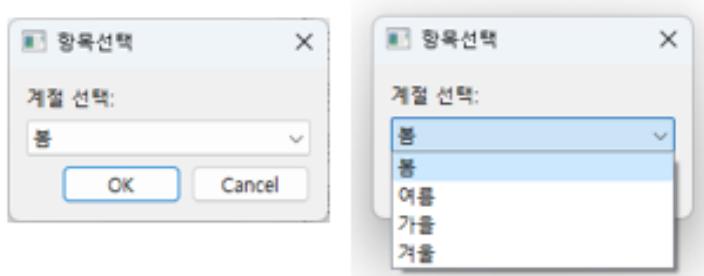
첫 번째 인자는 부모 클래스, 두 번째 인자는 윈도우의 타이틀 바에 표시할 제목, 세 번째 인자는 입력 값 위젯 항목의 좌측에 표시할 항목 이름이다. 네 번째 인자는 디폴트 설정 값, 다섯 번째와 여섯 번째는 사용자가 입력할 수 있는 값의 범위이고 다음 인자는 다이얼로그의 스피너 박스의 증가 값이다. 마지막 인자는 다이얼로그에서 [OK] 또는 [Cancel] 버튼을 클릭했는지 확인할 수 있는 값을 저장한다.

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



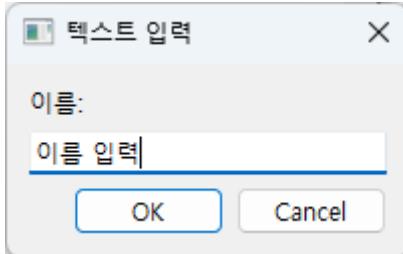
QInputDialog 클래스의 getDouble() 멤버 함수는 실수 값을 입력 받을 수 있다. getItem() 멤버 함수는 문자열(또는 단어) 중에 하나를 선택할 수 있는 기능을 제공한다.

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



예수님은 당신을 사랑합니다.

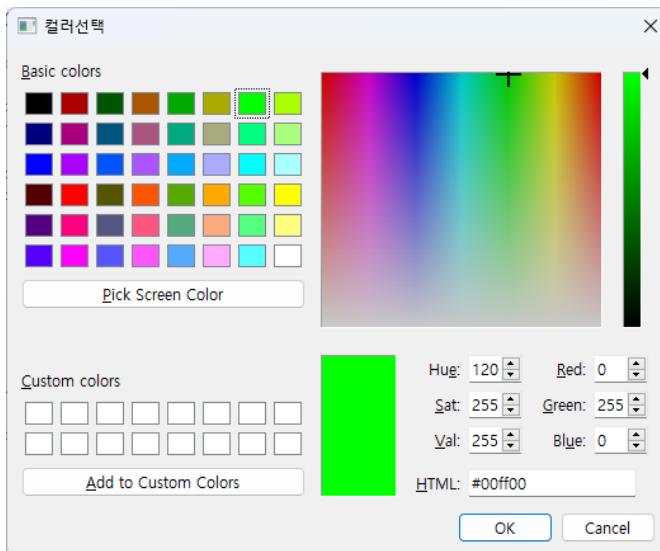
QInputDialog 클래스의 getText() 멤버 함수는 사용자로부터 텍스트를 입력 받을 수 있는 다이얼로그를 제공한다.



```
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.



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

✓ QFileDialog

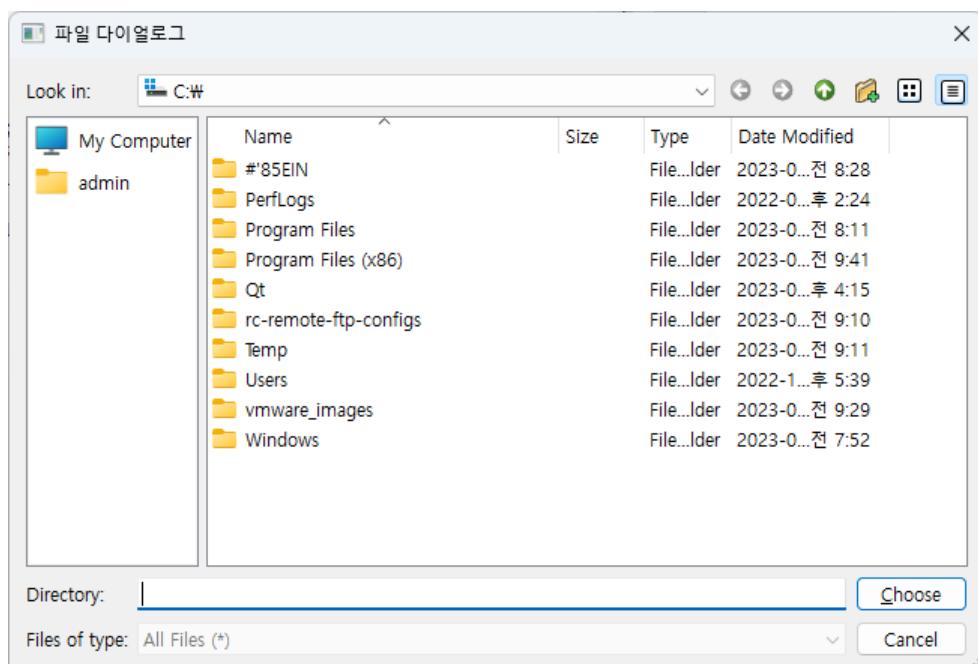
QFileDialog 클래스는 사용자로부터 파일을 선택할 수 있는ダイ얼로그를 제공한다. 특정 확장자 또는 특정 파일을 필터링하여 사용자에게 보여줄 수 있다.

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);
```

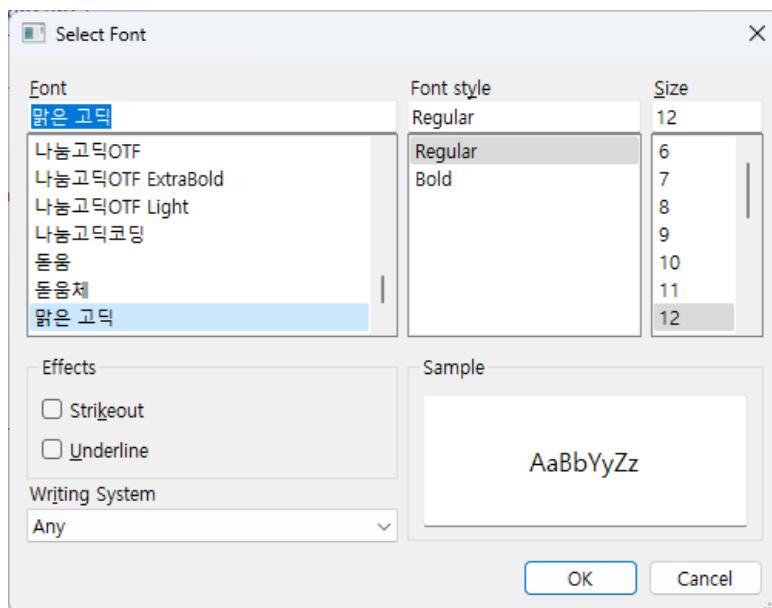
`getExistingDirectory()` 멤버 함수는 사용자로부터 디렉토리를 선택할 수 있는 기능을 제공한다. 첫 번째 인자는 부모 클래스, 두 번째 인자는 다이얼로그의 타이틀 바 제목, 세 번째 인자는 지정한 디렉토리가 디폴트로 지정하기 위해 사용하는 인자이다. 마지막 인자는 파일 다이얼로그의 상수 값을 이용해 필터링하기 위한 옵션이다.



| 상수 | 설명 |
|------------------------------------|--------------------------------|
| QFileDialog::ShowDirsOnly | 디렉토리만 표시 |
| QFileDialog::DontResolveSymlinks | 심볼릭 링크를 표시하지 않기 위해 사용 |
| QFileDialog::DontConfirmOverwrite | 덮어쓰기 할 때 경고 메시지를 표시하지 않기 |
| QFileDialog::DontUseNativeDialog | 시스템 기본 파일 다이얼로그를 사용하지 않기 위해 사용 |
| QFileDialog::ReadOnly | 읽기 모드로 파일 다이얼로그를 사용 |
| QFileDialog::HideNameFilterDetails | 필터를 이용해 파일을 감추기 위해 사용 |

✓ QFontDialog

QFontDialog는 사용자로부터 폰트를 선택할 수 있는ダイ얼로그를 제공한다. 첫 번째 인자는 사용자가 다이얼로그 하단에 위치해 있는 [OK] 버튼과 [CANCEL] 버튼 중 어떤 버튼을 클릭했는지 확인하기 위한 변수를 지정 한다.



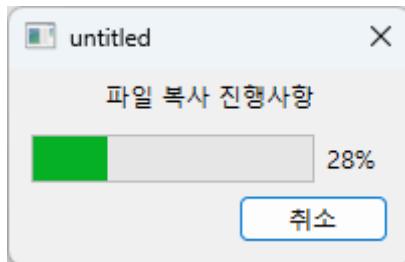
QFontDialog는 사용자로부터 폰트를 선택할 수 있는ダイ얼로그를 제공한다. 첫 번째 인자는 사용자가 다이얼로그 하단에 위치해 있는 [OK] 버튼과 [CANCEL] 버튼 중 어떤 버튼을 클릭했는지 확인하기 위한 변수를 지정 한다.

두 번째 인자는 폰트 다이얼로그 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 사용 한다. 예를 들어 대용량 파일 복사와 같이 시간이 다소 걸리는 사항에 대해서ダイ얼로그 창에 진행사항을 보여 줄 수 있다.



다음은 QProgressDialog 클래스를 이용해 사용자에게 진행사항을 보여주기 위한 디얼로그 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얼로그 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 |

예수님은 당신을 사랑합니다.

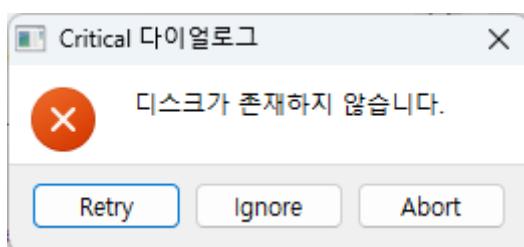
| | | |
|-----------------------|------------|-------------------|
| QMessageBox::Ignore | 0x00100000 | Ignore(무시) 버튼 |
| QMessageBox::NoButton | 0x00000000 | An invalid button |

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

```
QMessageBox::StandardButton reply;

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

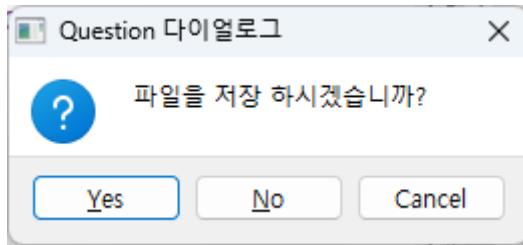
if (reply == QMessageBox::Abort)
    ...
else if (reply == QMessageBox::Retry)
    ...
```



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

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

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



✓ 사용자 정의ダイアル로그 구현 예제

이번 예제는 QDialog 클래스를 상속받아 사용자가 원하는 스타일의ダイ얼로그를 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 헤더파일이

예수님은 당신을 사랑합니다.

다. 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("Name");
    edit = new QLineEdit("");
    okbtn = new QPushButton("Confirm");
    cancelbtn = new QPushButton("Cancel");

    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 소스코드

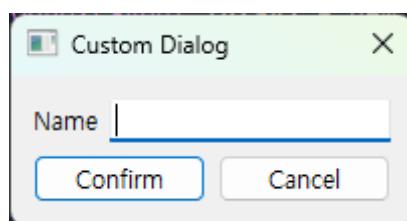
이다.

```
#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();
}
```



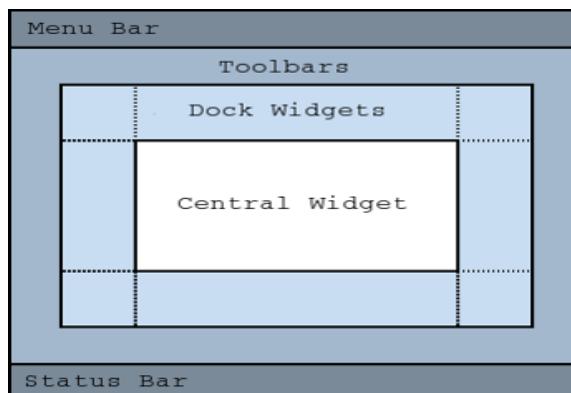
위의 예제에서 보는 것과 같이 Dialog 클래스의 dlg 오브젝트는 사용자가 [확인] 버튼 클릭 시 int 값 1을 리턴 한다. [취소] 버튼 클릭 시 0을 리턴 한다. 예제의 전체 소스는 00_CustomDialog 디렉토리를 참조하면 된다.

12. QMainWindow 를 이용한 GUI 구현

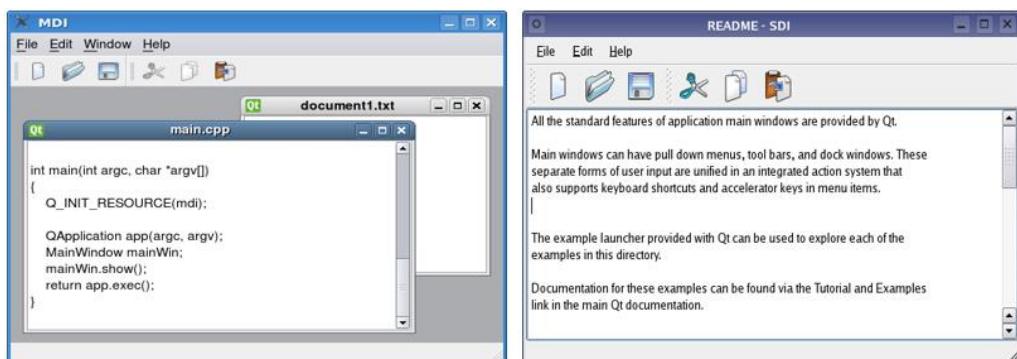
지금까지 Qt를 이용한 GUI 기반의 위젯을 윈도우 상에 배치하는 예제를 다루어 보았다. 지금까지 다룬 방식은 QWidget 을 이용해 한 개의 윈도우 화면만 존재하는 방식으로 GUI를 구성하였다.

하지만 기능이 복잡하고 사용자에게 많은 기능을 제공해야 하는 경우 GUI 구현 시 QWidget 보다는 QMainWindow 를 이용해 GUI를 구현하는 것이 사용자에게 직관적인 GUI를 제공할 수 있을 것이다.

예를 들어 Menu Bar, Toolbars, Status Bar, Dock Widget, Central Widget 등으로 위젯들을 특정 영역에 배치할 수 있다.



그리고 Qt는 MDI(Multi Document Interface) 방식을 구현할 수 있다. MDI 방식은 QMdiArea 클래스를 이용해 구현할 수 있다. 복잡한 윈도우 GUI를 구현해야 한다면 QMainWindow 와 더불어 QMdiArea 클래스를 함께 사용하는 것을 권장한다.



- ✓ QMdiArea 클래스를 이용한 MDI 기반의 GUI 예제

예수님은 당신을 사랑합니다.

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()
```

예수님은 당신을 사랑합니다.

```
{  
    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.

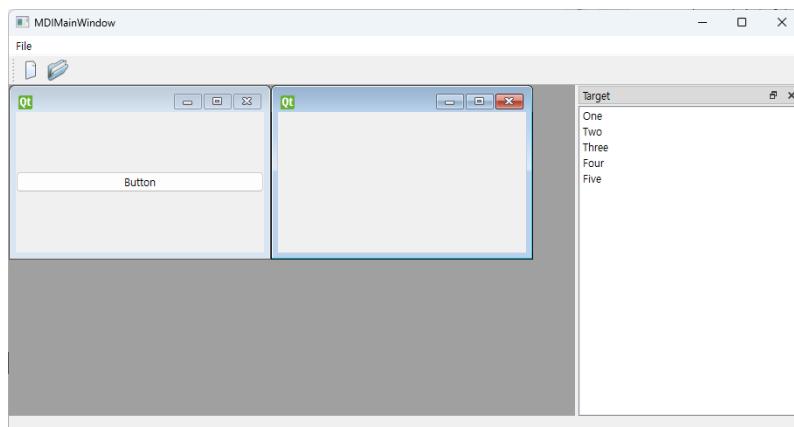
```
#include "MDIMainwindow.h"  
#include <QMdiArea>  
#include <QMdiSubWindow>  
#include <QPushButton>  
  
MDIMainWindow::MDIMainWindow(QWidget *parent) : QMainWindow(parent)  
{  
    setWindowTitle(QString::fromUtf8("My MDI"));
```

예수님은 당신을 사랑합니다.

```
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);
}
```



예제 소스는 00_MDIMainWindow 디렉토리를 참조하면 된다.

13. Stream

여기에서 Stream 이란 데이터를 특정 변수에 Write/Read 를 쉽게 하기 위한 방법을 말한다. 예를 들어 quint32 타입(4 Bytes)의 변수 데이터를 QByteArray 에 Write/Read 해야 하는 경우 Qt 에서 제공하는 QDataStream 또는 QTextStream 을 사용하면 쉽게 핸들링 할 수 있다.

QDataStream 은 Binary 데이터를 Write/Read 하는데 사용하며 QTextStream 은 Text 기반의 데이터를 Write/Read 하는데 사용된다.

이번 장에서는 예제를 이용해 QDataStream 과 QTextStream 사용하는 방법에 대해서 살펴보도록 하자.

✓ QDataStream 을 이용한 예제

이번 예제에서는 QDataStream 상에 데이터를 Write/Read 하는 예제를 다루어 보도록 하자. encoding() 함수에서는 quint32, quint8, quint32 타입의 데이터를 QDataStream 을 이용해 QByteArray 에 저장할 것이다. 각 변수에는 123, 124, 125가 저장할 것이다.

그리고 decoding() 함수에서는 QByteArray를 저장된 데이터를 차례대로 읽어오는 예제를 구현할 것이다. 아래는 예제 소스코드이다.

```
#include <QCoreApplication>
#include <QIODevice>
#include <QDataStream>
#include <QDebug>

QByteArray encoding()
{
    quint32 value1 = 123;
    quint8  value2 = 124;
    quint32 value3 = 125;

    QByteArray outData;
    QDataStream outStream(&outData, QIODevice::WriteOnly);

    outStream << value1;
```

```
outStream << value2;
outStream << value3;

qDebug() << "outData size : " << outData.size() << " Bytes";

return outData;
}

void decoding(QByteArray _data)
{
    QByteArray inData = _data;

    quint32 inValue1 = 0;
    quint8 inValue2 = 0;
    quint32 inValue3 = 0;

    QDataStream inStream(&inData, QIODevice::ReadOnly);

    inStream >> inValue1; // 123
    inStream >> inValue2; // 124
    inStream >> inValue3; // 125

    qDebug("[First : %d] [Second : %d] [Third : %d]"
           , inValue1, inValue2, inValue3 );
}

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

    QByteArray encData = encoding();
    decoding(encData);

    return a.exec();
}
```

encoding() 함수에서 quint32, quint8, quint32 변수 타입에 저장된 데이터 123, 124, 125 를 QDataStream 를 이용해 QByteArray 에 저장한다.

그리고 QByteArray 에 저장한 값을 encData 에 저장 한다. 그리고 encData 를 decoding() 함수에 첫 번째 인자로 넘겨 준다. decoding() 함수에서는 첫 번째 인자로 넘겨 받은 QByteArray 값을 QDataStream을 이용해 Read 한다. 따라서 decoding() 함

예수님은 당신을 사랑합니다.

수의 마지막 라인의 qDebug() 를 이용해 값을 출력한다. 이번 예제 소스코드는 00_DataStream 디렉토리를 참조하면 된다.

✓ QTextStream 을 이용한 예제

이번 예제에서는 이 전 예제와 비슷한 방법으로 작성된 예제이다. writeData() 함수에서는 QByteArray 에 저장된 값을 QTextStream 을 이용해 Write 할 것이다.

그리고 readData() 함수에서는 반대로 QByteArray 에 저장된 값을 Read 할 것이다. 다음 예제는 이번 예제의 전체 소스코드이다.

```
#include <QCoreApplication>

#include <QIODevice>
#include <QTextStream>
#include <QDebug>

QByteArray writeData(QByteArray _data)
{
    QByteArray temp = _data;

    QByteArray outData;
    QTextStream outStream(&outData, QIODevice::WriteOnly);

    for(qsizetype i = 0 ; i < temp.size() ; i++) {
        outStream << temp.at(i);
    }
    outStream.flush();

    return outData;
}

void readData(QByteArray _data)
{
    QTextStream outStream(&_data, QIODevice::ReadOnly);

    QByteArray inData;
    for(qsizetype i = 0 ; i < _data.size() ; i++)
    {
        char data;
        outStream >> data;
    }
}
```

```
    inData.append(data);
}

qDebug("READ DATA : [%s]", inData.data());
}

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

    QByteArray retData = writeData("Hello world."); // 12 Bytes
    qDebug() << "WRITE DATA size : " << retData.size() << " Bytes";

    readData(retData);

    return a.exec();
}
```

writeData 함수에서는 첫 번째 인자로 "Hello world" 를 인자로 넘겨 줄 것이다. 이 문자열은 QByteArray 에 저장된다. 저장 된 이 데이터를 QTextStream 을 이용해 QByteArray 에 저장한다.

그리고 readData() 함수에는 "Hello world." 가 저장되어 있는 QByteArray 를 QTextStream 을 이용해 데이터를 읽어 올 것이다. 따라서 readData() 함수의 마지막 라인에서 "Hello world"를 출력할 것이다.

이번 예제의 소스코드는 01_QTextStream 을 참조하면 된다.

14. 파일 입출력

Qt에서 파일 입출력 처리를 위해서 QFile 클래스를 제공한다. 그리고 대량의 데이터를 효율적으로 READ/WRITE 하기 위한 목적으로 QTextStream 과 QDataStream 클래스를 함께 사용할 수 있다. 파일 크기가 작은 파일을 READ / WRITE 하는데 QFile 에서 제공하는 멤버 함수만 이용해도 문제가 없지만 파일 용량이 큰 파일을 처리하는데 QTextStream 클래스와 QDataStream 클래스를 이용하면 대용량 파일을 핸들링 하는데 효율적으로 파일을 핸들링 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 사용할 수 있다.

| 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 예제 소스코드에서 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 데이터 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();

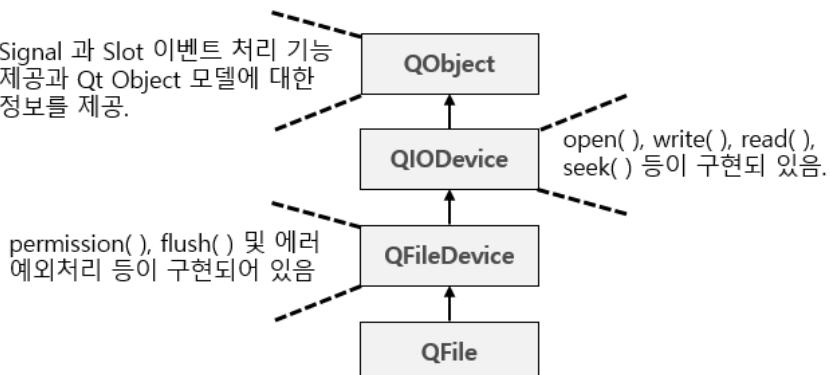
file.close();
}

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

```

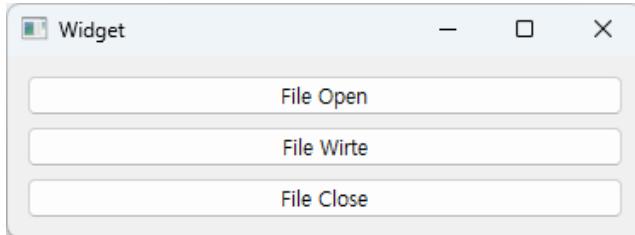
QFile 클래스는 파일 처리를 위해 Open, Exists, Link, Remove, Rename 등과 같은 기능을 제공한다. QFile 클래스는 QFileDevice 클래스로부터 상속 받아 구현 되었다. QFileDevice 클래스는 파일의 Permission, Flush, Error 처리와 같은 예외 처리 기능 등이 구현 되어 있다.

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.



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



이번에 다룰 예제는 파일 Open, Write 그리고 Close 기능을 구현한 예제이다. [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 "Hello World" 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();
}

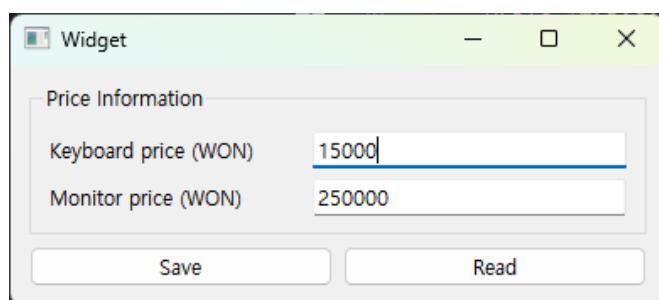
void Widget::slotPbtWritePress()
{
    QTextStream in(mFile);
    mFile->seek(mFile->size());
    in << "End.\n";
}

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

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

- ✓ QFile 클래스와 QDataStream 을 이용한 예제

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

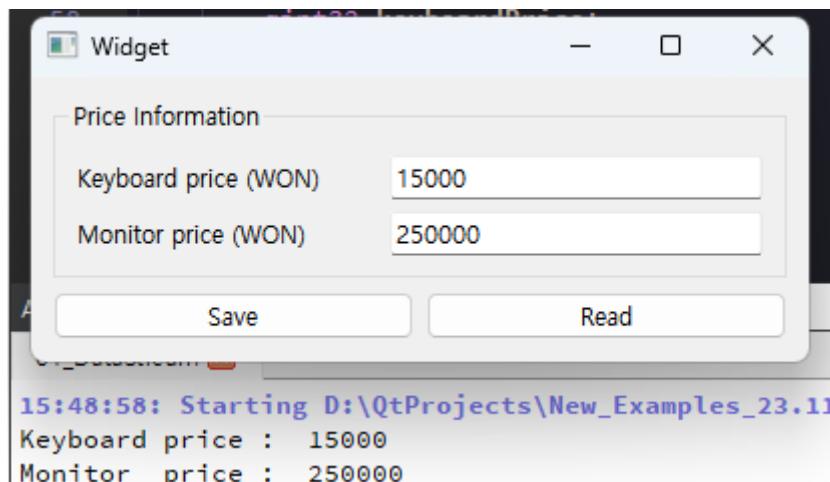


예수님은 당신을 사랑합니다.

파일에 저장 시 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.



예제 전체 소스코드는 01_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();
};
```

[Save] 버튼을 클릭하면 slotPbtFileSave() Slot함수가 호출된다. [Read] 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("d:/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 monitorPrice = ui->leMonitor->text().toInt();

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

    mFile->close();
}
}

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

    QDataStream in(mFile);
    in >> keyboardPrice;
    in >> monitorPrice;

    mFile->close();

    qDebug() << "Keyboard price : " << keyboardPrice;
    qDebug() << "Monitor price : " << monitorPrice;
}
}
```

15. Qt Property

Qt에서 제공하는 Property System은 C++에서 제공하는 Property System과 비슷하다. Property는 객체에서 값을 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 함수는 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 출력하는 예제 소스코드이다.

예수님은 당신을 사랑합니다.

예제 소스코드에서 살펴본 방법을 Qt에서 제공하는 Property System으로 접근해 보도록 하자.

```
...
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();

// [ Result ]
// 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

예수님은 당신을 사랑합니다.

얻어오거나 설정할 수 있다.

클래스에서 Q_PROPERTY 매크로 그다지 큰 유용성이 없어 보인다. 하지만 QML을 사용하는 경우 매우 요긴하게 사용할 수 있다.

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 Q_PROPERTY 매크로를 사용할 수 있다.

Q_PROPERTY 매크로는 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 private에서 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

이번에 다룰 예제는 아래 그림에서 보는 것과 같이 [Change] 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;  
}  
...
```

예수님은 당신을 사랑합니다.

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);
};

...
```

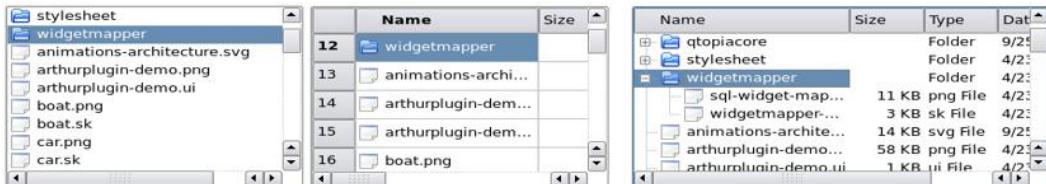
Person 클래스에서 Q_PROPERTY 매크로를 이용해 getName() 와 setName() 멤버 함수를 접근할 수 있다. 그리고 nameChanged() 시그널을 Q_PROPERTY 매크로의 NOTIFY 키워드로 지정하였다.

따라서 setName() 멤버 함수에서 emit 을 이용해 시그널 이벤트가 발생하면 Widget 클래스의 연결된 Slot 함수가 호출 된다.

예제의 전체 소스코드는 00_Simple_Property 디렉토리를 참조하면 된다.

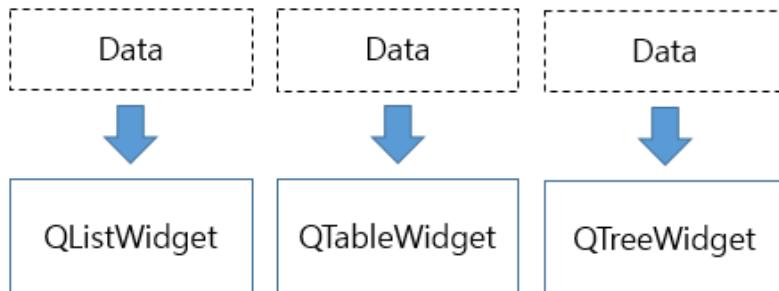
16. Model and View

Qt에서는 다음 그림에서 보는 것과 같이 표와 같은 위젯에 데이터를 표시하기 위한 위젯으로 QListWidget, QTableWidget, QTreeWidget, QListView, QTableView, QTreeView, QColumnView 등 다양한 클래스를 제공한다.



QListWidget 은 QListView와 UI가 동일하다. 하지만 QListWidget 과 QListView 는 데이터를 삽입/수정/삭제 하는데 차이가 있다.

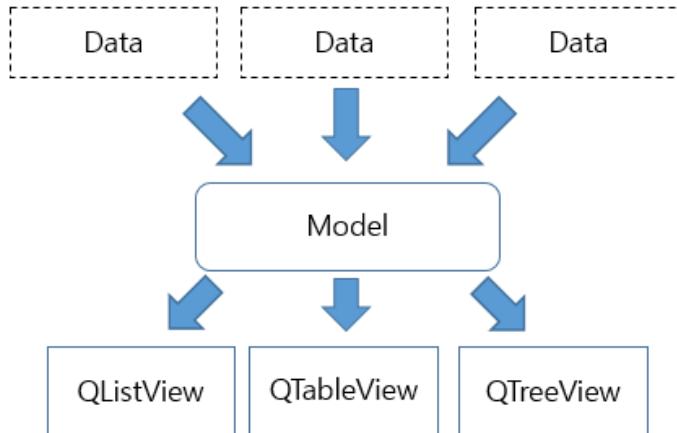
클래스 이름의 마지막에 View 대신, Widget 이라는 단어를 사용한 클래스들은 아래 그림에서 보는 것과 같이 데이터를 직접 삽입/수정/삭제 할 수 있는 멤버 함수를 제공한다.



클래스 이름 마지막에 Widget 이라는 단어가 쓰인 위젯 클래스들은 모두 직접 데이터를 삽입/수정/삭제 가 가능한 멤버 함수를 제공한다. 예를 들어 QListWidget 위젯 클래스는 insertItem() 함수를 이용해 데이터를 삽입할 수 있다.

하지만 각 위젯 클래스들은 사용방법이 다소 차이가 있으므로 각각의 위젯 클래스들의 사용 방법을 익혀야 한다.

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.

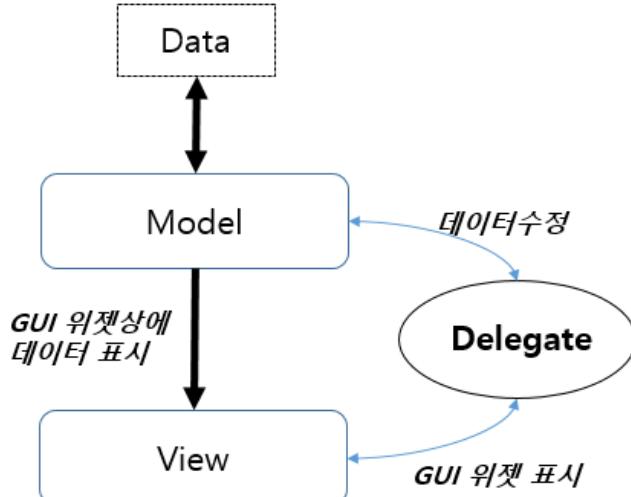


위의 그림에서 보는 것과 같이 QListView, QTableView, QTreeView 클래스들은 위젯에 데이터를 삽입/수정/삭제 하기 위해 각 클래스에서 제공하는 멤버 함수를 사용하지 않고 Model 클래스를 사용한다.

이 방식을 사용할 경우 QListView 를 사용하든지 QTableView 를 사용하든지 동일한 Model을 사용할 수 있다는 장점을 가지고 있다.

Qt 에서 제공하는 Model/View 는 다음 그림에서 보는 것과 같이 Delegate를 이용해 데이터를 핸들링 할 수 있다.

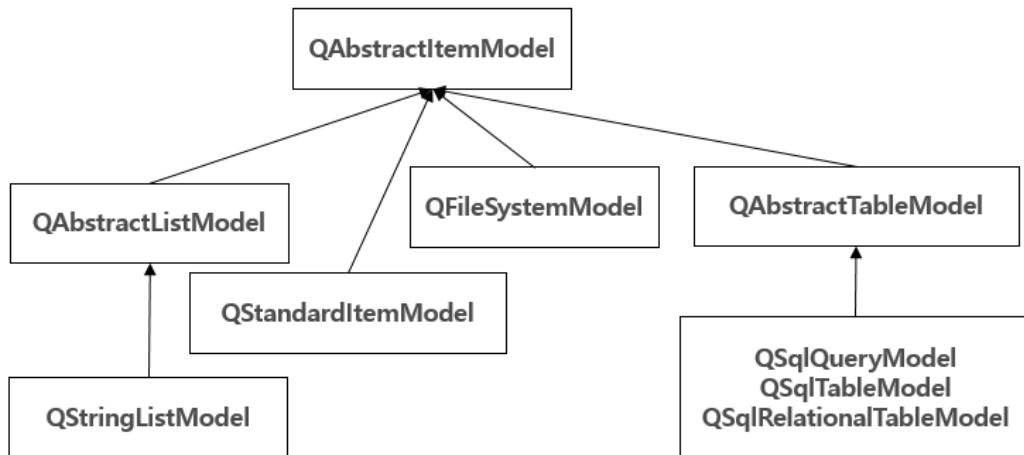
예를 들어 View 위젯 상에 표시된 데이터 항목 중 특정 항목을 마우스로 더블 클릭해 데이터를 수정하기 위해서 이벤트를 발생해야 하는데 Model/View 에서는 이러한 이벤트를 처리하기 위해 Delegate를 사용할 수 있다.



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 함

예수님은 당신을 사랑합니다.

수를 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.



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);  
    }  
}
```

예수님은 당신을 사랑합니다.

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 문을 이용해 데이터베이스 테이블로부터 데이터를 액세스 할 수 있는 기능을 제공하며 QSqlTableModel 클래스는 데이터베이스로부터 데이터를 가져올 때 특정 테이블 명을 인자로 전달하면 Model에 데이터를 가져올 수 있다.

예를 들어 setTable() 멤버 함수의 첫 번째 인자로 테이블 명을 입력하면 데이터베이스 테이블로부터 데이터를 가져올 수 있다.

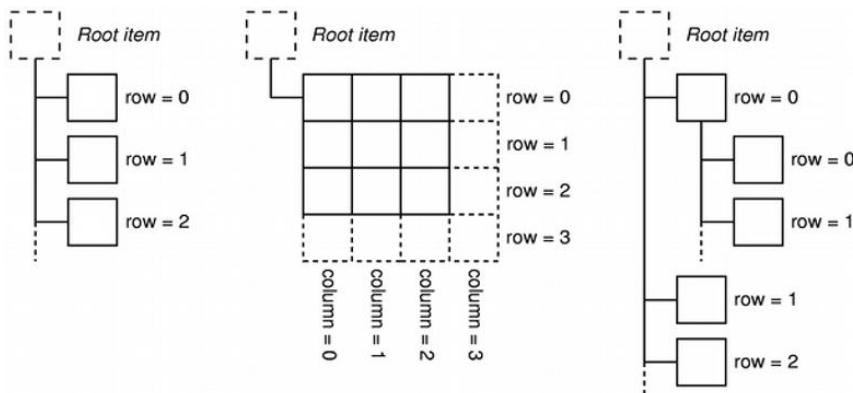
```
QSqlQueryModel *model = new QSqlQueryModel;

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.

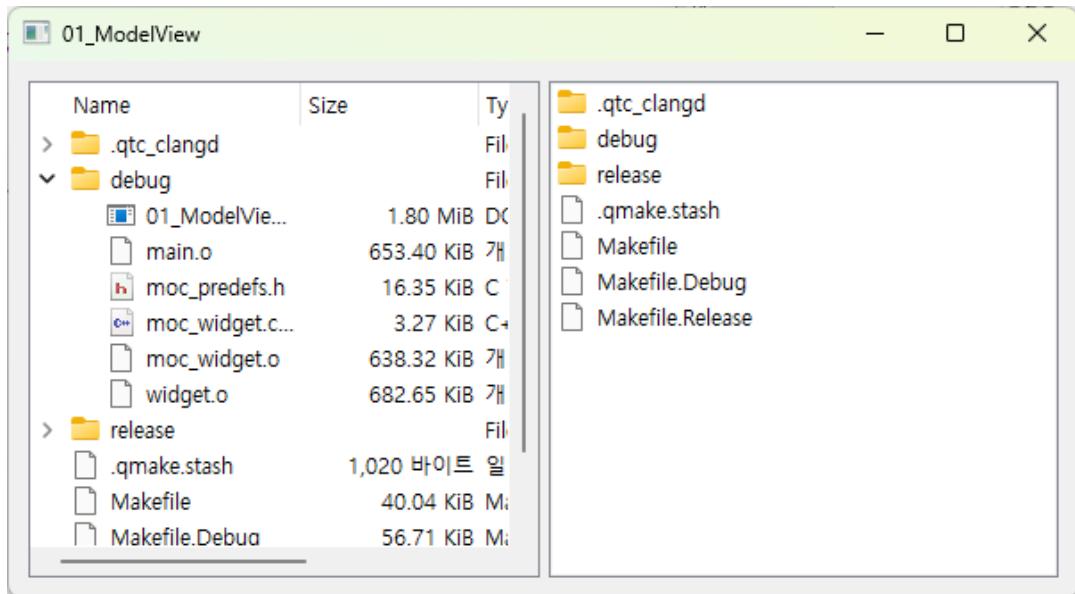


예수님은 당신을 사랑합니다.

가장 좌측에 있는 데이터 형태라면 QListview 클래스를 사용하는 것이 적합하다. 표와 같이 표시해야 한다면 QTableview 를 사용하는 것이 적합하다. 그리고 트리 형태로 표시해야 한다면 QTreeview를 사용하는 것이 적합하다.

- ✓ QTreeview 와 QListview 클래스를 이용한 예제

이 예제는 파일로부터 데이터를 읽어와 Model 에 저장한 다음 Model 을 View 와 연결해 파일 시스템을 표시하는 예제이다.



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()));
```

예수님은 당신을 사랑합니다.

```
QListView *list = new QListView(splitter);
list->setModel(model);
list->setRootIndex(model->index(QDir::currentPath()));

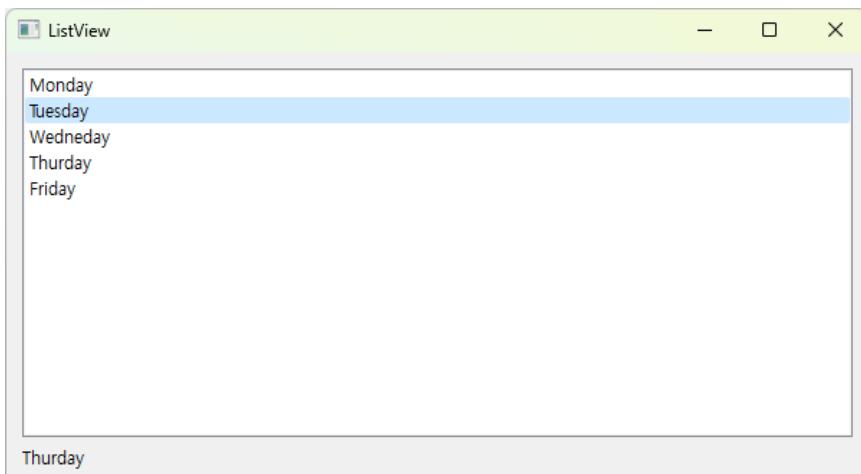
QVBoxLayout *layout = new QVBoxLayout();
layout->addWidget(splitter);
setLayout(layout);

}

...
```

00_ModelView 디렉토리를 참조하면 된다.

✓ QListView 클래스를 이용한 예제



위의 그림은 QListView 클래스와 QAbstractItemModel 클래스를 이용한 예제 실행 화면이다. QListView 는 한 개의 컬럼과 여러 줄로 표시하는데 적합한 위젯이다.

```
...
resize(600, 300);
QStringList strList;

strList << "Monday" << "Tuesday" << "Wednesday" << "Thurday" << "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 01_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 |
| Col 2 | CPU | Samsung | 2030-10-04 |

```
...
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");
```

예수님은 당신을 사랑합니다.

```
model->setData(model->index(0, 2), QDate(2030, 10, 4));  
  
model->setData(model->index(1, 0), "CPU");  
model->setData(model->index(1, 1), "Samsung");  
model->setData(model->index(1, 2), QDate(2030, 12, 5));  
  
QTableView *table = new QTableView();  
table->setModel(model);  
  
QVBoxLayout *lay = new QVBoxLayout();  
lay->addWidget(table);  
  
setLayout(lay);  
...
```

전체 소스코드는 02_TableModel 디렉토리를 참조하면 된다.

✓ QTableWidget 예제

이번 예제는 Model 을 사용하지 않고 QTableWidget 을 이용해 데이터를 삽입하는 예제를 다루어 볼 것이다. 그리고 첫 번째 Column 에 QCheckBox 위젯을 삽입하는 방법에 대해서 알아보도록 하자.

| 1 | 2 |
|---|------------|
| 1 | 2025.05.07 |
| 2 | 2025.05.07 |
| 3 | 2025.05.07 |
| 4 | 2025.05.07 |
| 5 | 2025.05.07 |

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.

예수님은 당신을 사랑합니다.

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 << "Notebook" << "Mobile" << "Desktop" << "Keyboard" << "Monitor";

    for(int i = 0; i < 5 ; i++)
    {
        ui->tableWidget->insertRow(i);
        QTableWidgetItem *dateItem = new QTableWidgetItem("2021.05.07");
        dateItem->setCheckState(Qt::Checked);

        ui->tableWidget->setItem(i,0, dateItem);
        ui->tableWidget->setItem(i,1, new QTableWidgetItem(nameList.at(i)));
    }
}

void Widget::checkBoxClicked(bool state)
{
    for(int i = 0 ; i < 5 ; i++) {
        QTableWidgetItem *item = ui->tableWidget->item(i, 0);
        if(state)
            item->setCheckState(Qt::Checked);
        else
            item->setCheckState(Qt::Unchecked);
```

예수님은 당신을 사랑합니다.

```
}
```

```
}
```

```
...
```

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 소스코드 이다.

```
#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();
}
```

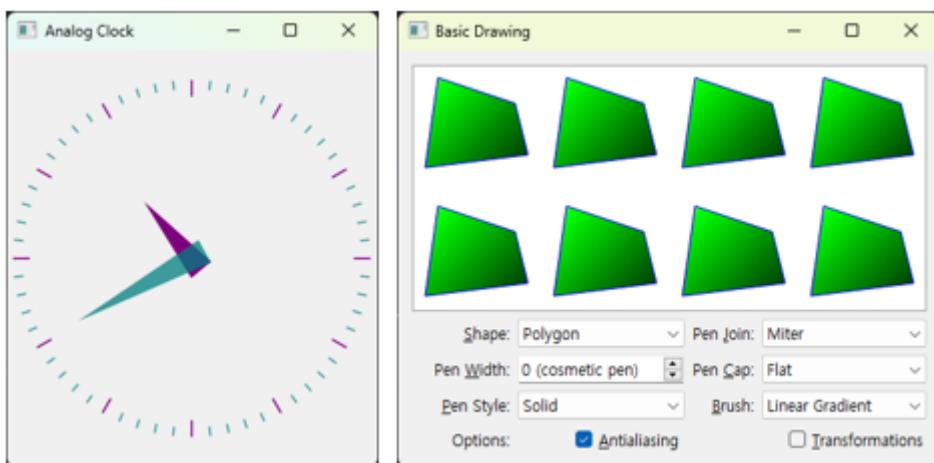
예수님은 당신을 사랑합니다.

```
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 03_TableWidget 딕토리를 참조하면 된다.

17. QPainter 클래스를 이용한 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 QPainter 영역에 적용할 수 있다.



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();

    void paintEvent(QPaintEvent *event) override;
};

...
```

위의 예제 소스코드에서 보는 것과 같이 QWidget 클래스를 상속받아 구현 시 paintEvent() 함수를 사용할 수 있다. paintEvent() 함수는 위젯이 가려진 상태에서 모니

예수님은 당신을 사랑합니다.

터에 보여지게 되는 현상이 발생하거나 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() 함수가 자동으로 호출할 수 있다. 여기서 주의할 사항으로 paintEvent() 함수를 직접 호출하면 안된다. paintEvent() 함수를 호출해야 할 경우 update() 함수를 호출해야 한다.

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)); // 폰트 지정
    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();
}
```

예수님은 당신을 사랑합니다.

위의 예제 소스코드에서 QPainter 클래스의 begin() 멤버 함수와 end() 함수는 실제로 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.



QPainter 는 영역 내에서 드로잉을 할 때에 선과 도형 윤곽의 컬러, 두께, 스타일을 다양하게 표현할 수 있는 기능을 제공한다.

```
...
QPainter painter;
painter.begin(this);

QPen pen(Qt::blue, 3, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin);

painter.setPen(pen);
QRect rect1(10.0, 20.0, 80.0, 50);
painter.drawEllipse(rect1);

pen.setStyle(Qt::DashLine);
```

예수님은 당신을 사랑합니다.

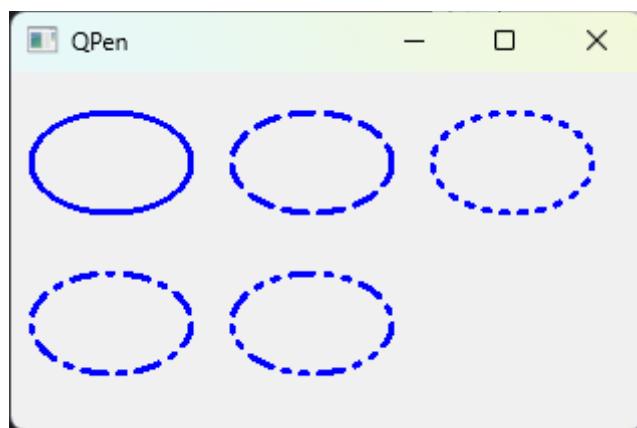
```
painter.setPen(pen);
QRect rect2(110.0, 20.0, 80.0, 50.0);
painter.drawEllipse(rect2);

pen.setStyle(Qt::DotLine);
painter.setPen(pen);
QRect rect3(210.0, 20.0, 80.0, 50.0);
painter.drawEllipse(rect3);

pen.setStyle(Qt::DashDotLine);
painter.setPen(pen);
QRect rect4(10.0, 100.0, 80.0, 50.0);
painter.drawEllipse(rect4);

pen.setStyle(Qt::DashDotDotLine);
painter.setPen(pen);
QRect rect5(110.0, 100.0, 80.0, 50.0);
painter.drawEllipse(rect5);

pen.setStyle(Qt::CustomDashLine);
painter.setPen(pen);
QRect rect6(210.0, 100.0, 80.0, 50.0);
painter.drawEllipse(rect6);
...
```



QPainter 클래스는 선을 그릴 때 선의 모서리 스타일을 QPen 클래스의 setJoinStyle() 멤버 함수를 이용해 지정할 수 있다.

```
...
QPen pen(Qt::black);
pen.setWidth(20);

QPointF p1[3] = {QPointF(30.0, 80.0),
QPointF(20.0, 40.0),
QPointF(80.0, 60.0)};

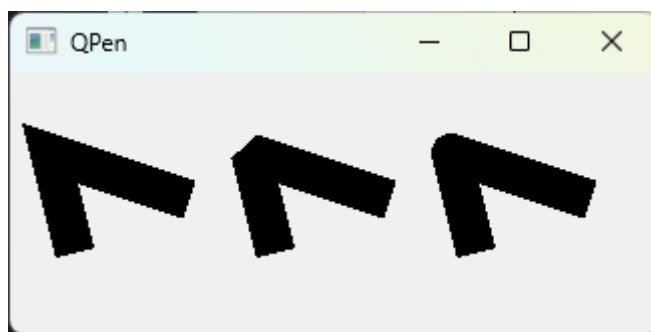
pen.setJoinStyle(Qt::BevelJoin);
painter.setPen(pen);
painter.drawPolyline(p1, 3);

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);
...
```



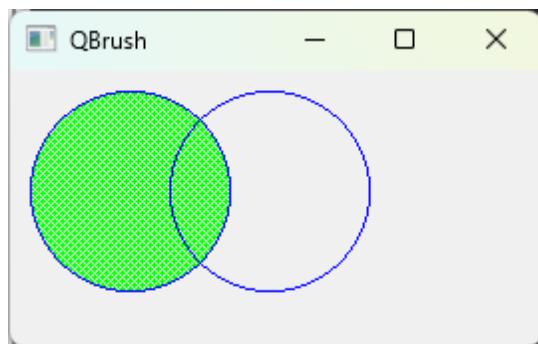
QPainter 클래스에서 제공하는 setBrush() 멤버 함수를 이용하면 도형의 내부에 특정 색으로 채울 수 있다.

예수님은 당신을 사랑합니다.

```
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);
...
```



QBrush는 특정 컬러로 내부를 채울 수 있다. 또한 이미지를 호출해 내부를 채울 수 있다.

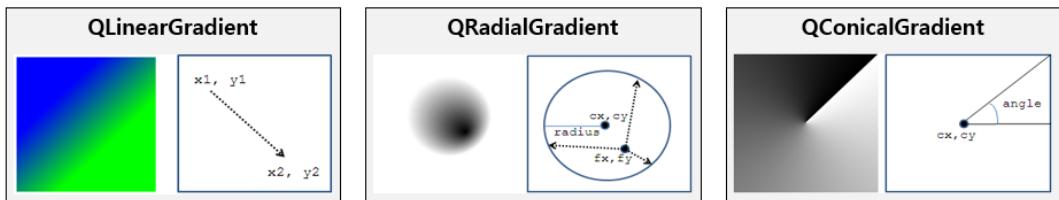
```
...
QPixmap pixmap(":resources/qtblog.png");
int w = pixmap.width();
int h = pixmap.height();
pixmap.scaled(w, h, Qt::IgnoreAspectRatio, Qt::SmoothTransformation);

QBrush brush(pixmap);
painter.setBrush(brush);
painter.setPen(Qt::blue);
painter.drawRect(0, 0, w, h);
...
```

예수님은 당신을 사랑합니다.



Gradients 효과를 사용하기 위해서 다음 그림에서 보는 것과 같은 Gradients 효과를 사용할 수 있다.



<그림> Gradient 의 종류

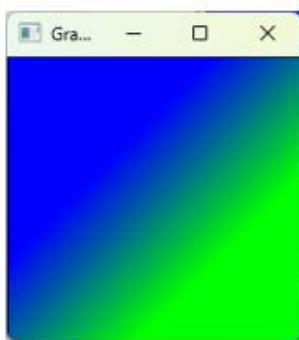
다음 예제 소스코드는 QLinearGradient 클래스를 이용해 Gradients 효과를 사용한 예제이다.

```
QLinearGradient ling(QPointF(70, 70), QPoint( 140, 140 ) );
ling.setColorAt(0, Qt::blue);
ling.setColorAt(1, Qt::green);

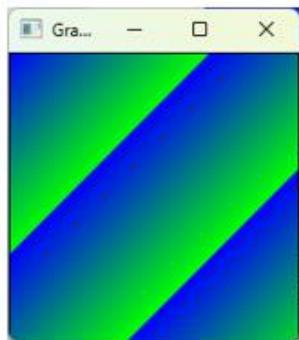
ling.setSpread( QGradient::PadSpread );
// ling.setSpread( QGradient::RepeatSpread );
// ling.setSpread( QGradient::ReflectSpread );

QBrush brush(ling);
painter.setBrush(brush);
painter.drawRect(0, 0, 200, 200);
...
```

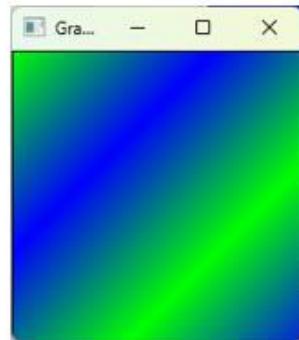
예수님은 당신을 사랑합니다.



QGradient::PadSpread



QGradient::RepeatSpread



QGradient::ReflectSpread

Qt는 `QTransform` 클래스를 이용해 Scaling(확대/축소), Rotation(회전), Perspective(원근 표현) 기법을 사용할 수 있다.



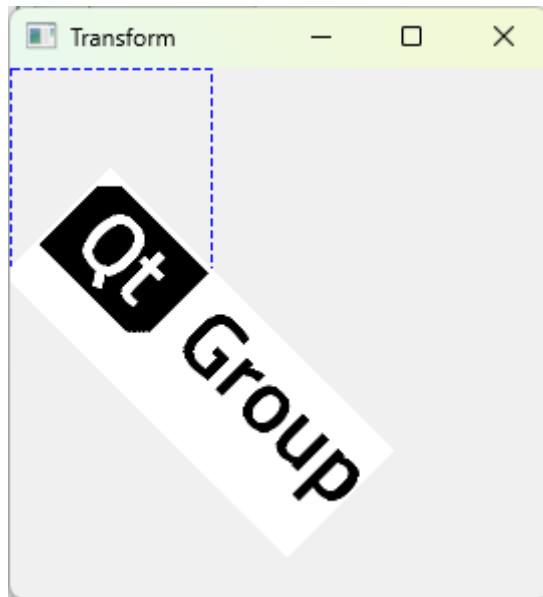
```
...
QImage image(":/resources/qtblog.png");

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);

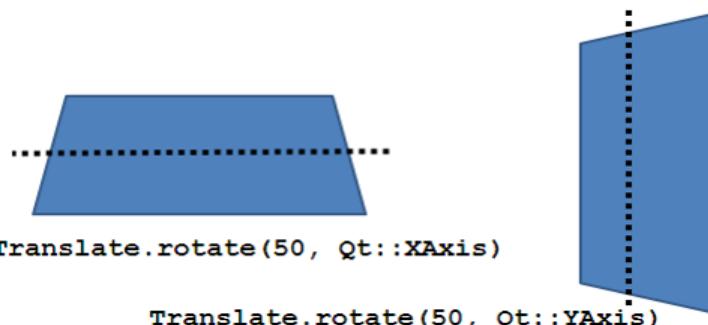
painter.setTransform(transform);
painter.drawImage(0, 0, image);
...
```

예수님은 당신을 사랑합니다.



<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

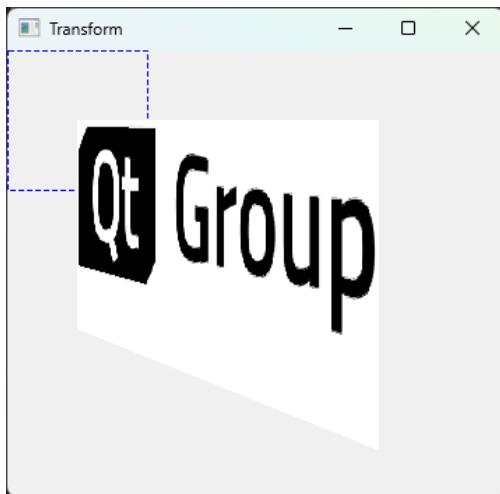
```
QPainter painter(this);
painter.setPen(QPen(Qt::blue, 1, Qt::DashLine));
painter.drawRect(0, 0, 100, 100);

QTransform transform;
transform.translate(50, 50);
transform.rotate(70, Qt::YAxis);

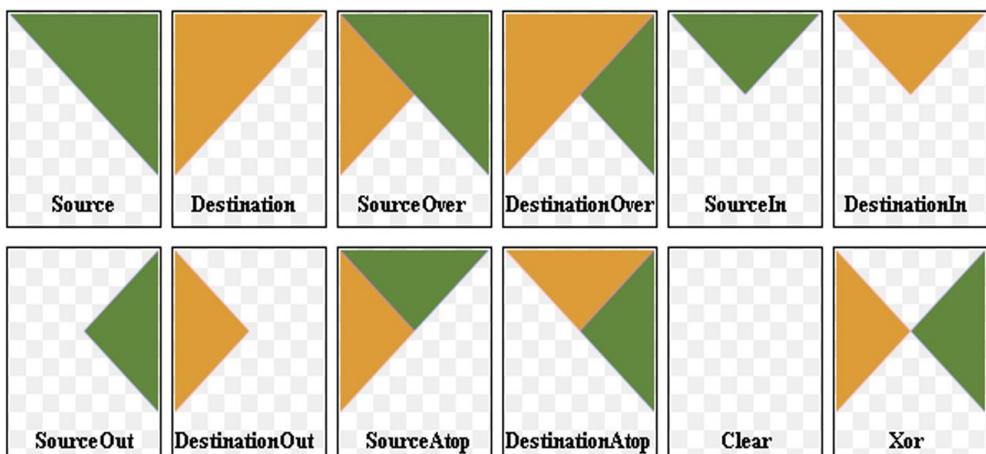
painter.setTransform(transform);
```

예수님은 당신을 사랑합니다.

```
painter.drawImage(0, 0, image);  
...
```



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.



다음 예제에서 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);
    void paintEvent(QPaintEvent *event) override;

private:
    QString imgFileName;
    qint32 behaviour;
    bool disabled;

signals:
    void clicked();

protected:
    virtual void enterEvent(QEnterEvent* event);
    virtual void leaveEvent(QEvent* event);
    virtual void mousePressEvent(QMouseEvent* event);
    virtual void mouseReleaseEvent(QMouseEvent* event);
}
```

예수님은 당신을 사랑합니다.

```
virtual void mouseDoubleClickEvent(QMouseEvent *event);  
};  
  
#endif // IMAGEBUTTON_H
```

위의 예제에서 setDisabled() 멤버 함수는 버튼을 Disable 처리하기 위한 기능을 제공한다. disable 변수는 setDisabled()에 설정된 값을 저장한다. paintEvent() 함수는 마우스의 이벤트에 따라 마우스의 이미지를 변경한다.

예를 들어 마우스가 버튼 위젯 영역에 위치하면 버튼 이미지를 위의 그림에서 보는 것과 같이 두 번째 이미지로 변경한다. 그리고 위젯 영역 내에서 마우스를 클릭하면 clicked() 시그널을 발생한다. 다음 예제는 ImageButton.cpp 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(QEnterEvent *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() 함수는 마우스가 위젯 영역 안에서 밖으로 나아갈 때 발생한다. mousePressEvent() 는 위젯 영역 안에서 마우스를 클릭했을 때 발생한다. mouseReleaseEvent() 는 마우스 버튼을 클릭 후 해제 했을 때 발생한다. 그리고 mouseDoubleClickEvent() 는 마우스 버튼을 더블 클릭했을 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>

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>
#include "imagebutton.h"

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

    ImageButton *imgBtn1 = new ImageButton(this);
    ImageButton *imgBtn2 = new ImageButton(this);

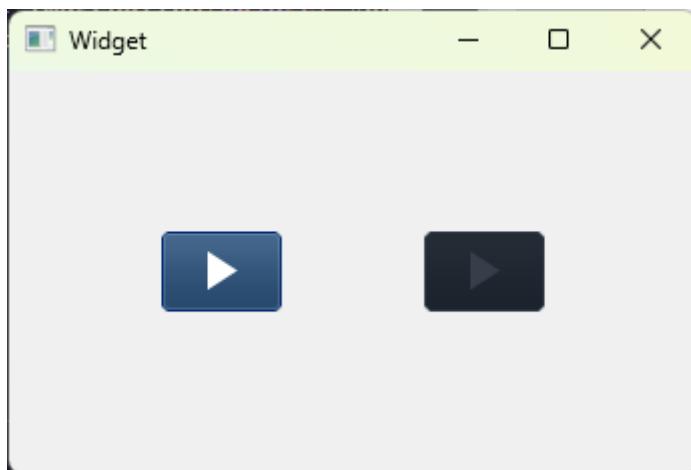
    QHBoxLayout *hLay = new QHBoxLayout(this);
    hLay->addWidget(imgBtn1);
    hLay->addWidget(imgBtn2);

    setLayout(hLay);
    connect(imgBtn1, &ImageButton::clicked, this, &Widget::clicked);
    imgBtn2->setDisabled(true);
}
```

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

void Widget::clicked()
{
    qDebug() << Q_FUNC_INFO;
}
```

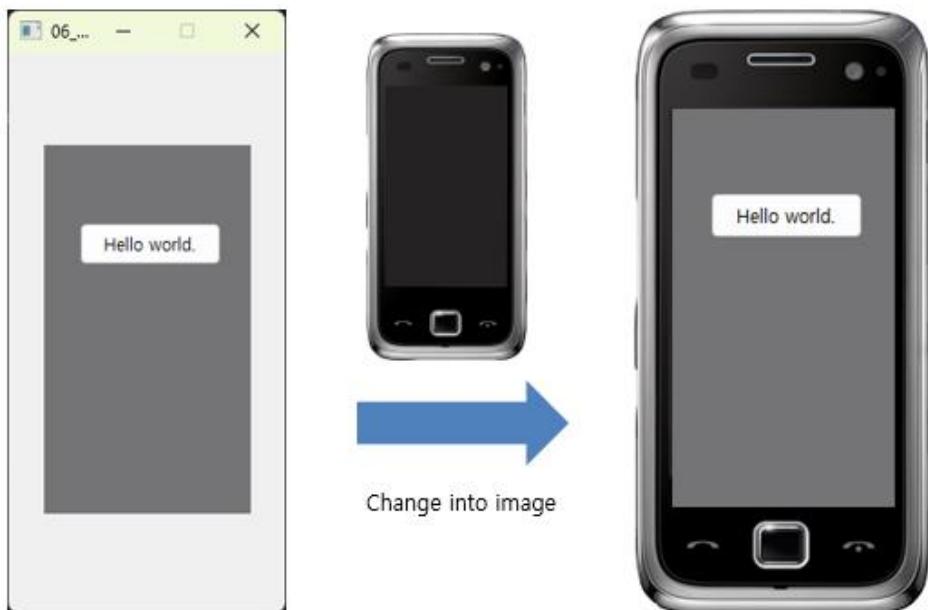
connect() 함수를 이용해 imgBtn1 오브젝트의 이벤트 발생시 Slot 함수와 연결한다. 그리고 두 번째 imgBtn2 는 Disable 상태로 만들기 위해 ImageButton 클래스에서 구현한 setDisabled() 함수를 사용하였다. 아래 그림에서 보는 것과 같이 작성한 예제를 빌드하고 실행 보도록 하자. 전체 예제 소스코드는 Ch04 > 06_CustomButton 디렉토리를 참조하면 된다.



✓ 원하는 윈도우 모양을 바꾸기 위한 예제 구현

이번 예제에서는 윈도우의 모양을 바꿔보고 윈도우 바탕(Background)을 특정 이미지 변경하는 방법에 대해서 살펴보는 예제이다.

예를 들어 직 사각형의 윈도우가 아니라 타원 형태의 윈도우 Form이라든지, 모서리를 Round 처리하는 것과 같이 윈도우의 형태를 변경하는 방법에 대해서 예제를 통해서 구현해 보도록 하자.



위의 그림에서 보는것과 같이 핸드폰 이미지를 윈도우의 Background 이미지로 사용한다. 그리고 Background로 사용하는 스마트폰 이미지의 모서리가 Round 처리가 되어 있다. 따라서 위의 이미지를 QWidget의 모양으로 사용하였다.

프로젝트 생성 시 QWidget 기반 어플리케이션을 생성한다. 프로젝트 생성 후 스마트폰 이미지를 사용할 수 있도록 프로젝트에 RESOURCE 파일을 아래와 같이 추가한다.

Background로 사용할 이미지는 예제 디렉토리인 06_CustomBackgroundWidget 디렉토리 하위에 resources 디렉토리에 보면 background.png 파일이 있다. 이 파일을 사용하면 된다.

```

Projects           CMakeLists.txt
└─ 06_CustomBackgroundWidget
    ├── CMakeLists.txt
    └── 06_CustomBackgroundWidget
        ├── Header Files
        ├── Source Files
        │   ├── main.cpp
        │   └── widget.cpp
        └── resources.qrc
            └── /resources
                └── background.png
    > CMake Modules

```

```

7 set(CMAKE_AUTORCC ON)
8
9 set(CMAKE_CXX_STANDARD 17)
10 set(CMAKE_CXX_STANDARD_REQUIRED ON)
11
12 find_package(Qt NAMES Qt6 Qt5 REQUIRED COMPONENTS
13 find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPON
14
15 set(PROJECT_SOURCES
16     main.cpp
17     widget.cpp
18     widget.h
19 )
20
21 if(${QT_VERSION_MAJOR} GREATER_EQUAL 6)
22     qt_add_executable(06_CustomBackgroundWidget
23         MANUAL_FINALIZATION
24         ${PROJECT_SOURCES}
25         resources.qrc
26     )

```

예수님은 당신을 사랑합니다.

위의 그림에서 보는 것과 같이 리소스 등록이 완료 하였으면 widget.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QPushButton>

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    QPixmap m_bgImage;
    QPushButton *pbtHello;

private slots:
    void slot_pbtHello();

protected:
    virtual void paintEvent(QPaintEvent* event);
};

#endif // WIDGET_H
```

m_bgImage 에는 Background 로 사용할 이미지를 저장한다. pbtHello 는 윈도우 상에 QPushButton 을 배치하는 용도로 사용할 것이다.

그리고 paintEvent() 함수에서는 Background 이미지를 윈도우 상에 랜더링 하고 중간에 배경을 회색으로 변경한다. 다음으로 widget.cpp 소스코드 파일을 아래와 같이 작성 한다.

```
#include "widget.h"
#include <QPainter>
#include <QPaintEvent>
```

```
Widget::Widget(QWidget *parent)
    : QWidget(parent, Qt::FramelessWindowHint)
{
    m_bgImage = QPixmap(":/resources/background.png", "png");

    QBitmap bitmap = m_bgImage.createHeuristicMask();
    setFixedSize(m_bgImage.size());
    setMask(bitmap);

    pbtHello = new QPushButton("Hello world.", this);
    connect(pbtHello, SIGNAL(pressed()), this, SLOT(slot_pbtHello()));

    pbtHello->setGeometry(50, 120, 100, 30);
}

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

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

    painter.drawPixmap(event->rect(), m_bgImage);

    painter.setPen(QColor(116, 116, 118));
    painter.setBrush(QColor(116, 116, 118));
    painter.drawRect(25, 65, 145, 260);
}

Widget::~Widget()
{
```

상단의 Qt::FramelessWindowHint 는 윈도우 Title 바를 없애기 위해서 추가하였다. 그리고 생성자에서 Background 이미지를 불러와 저장한다. 그리고 윈도우 크기에 맞게 Mask를 입히는 작업을 한다. 다음으로 QPushButton 을 Widget 상에 배치한다.

paintEvent() 함수에서는 Background 이미지를 Widget 상에 Rendering 한다. 그리고 회색을 윈도우의 바탕 색으로 지정하였다. 다음으로 빌드 후 실행해 보도록 하자.

예수님은 당신을 사랑합니다.



위의 예제는 06_CustomBackgroundWidget 디렉토리를 참조하면 된다.

✓ 이미지 스케일 비율 유지 기능 예제 구현

이번 예제에서는 이미지 파일을 위젯 영역에 랜더링(표시)하는 예제이다. 이 예제에서는 랜더링 하고 이미지의 크기가 위젯의 크기가 변경됨에 따라 이미지가 확대 되거나 축소된다.

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.



위의 그림에서 보는 것과 같이 위젯이 크기가 변경 될 때 이미지의 비율을 계산해 위젯의 크기에 비례해 랜더링 하는 예제이다.

예수님은 당신을 사랑합니다.

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;

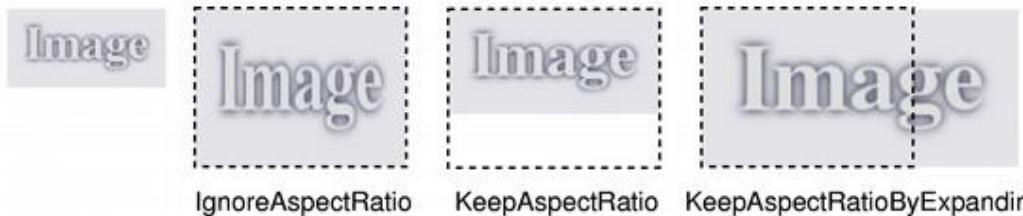
    if(w > imgPixmap.width())
        xPos = (w - imgWidth) / 2;
    else if( h > imgPixmap.height())
        yPos = (h - imgHeight) / 2;

    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. Scaled()함수의 세 번째 인자에서 사용할 수 있는 방식은 다음 표에서 보는 것과 같이 하나를 선택할 수

예수님은 당신을 사랑합니다.

있다.



<그림> 이미지 비율 표시 방식

| Constant | Value | Description |
|--------------------------------|-------|--|
| 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 07_ScaledImageRender 디렉토리를 참조하면 된다.

18. QPainter를 이용한 Chromakey 영상 처리 구현

이번 장에서는 이전에서 다루었던 QPainter를 이용해 간단한 Chromakey 응용 어플리케이션을 구현해 보도록 하자.

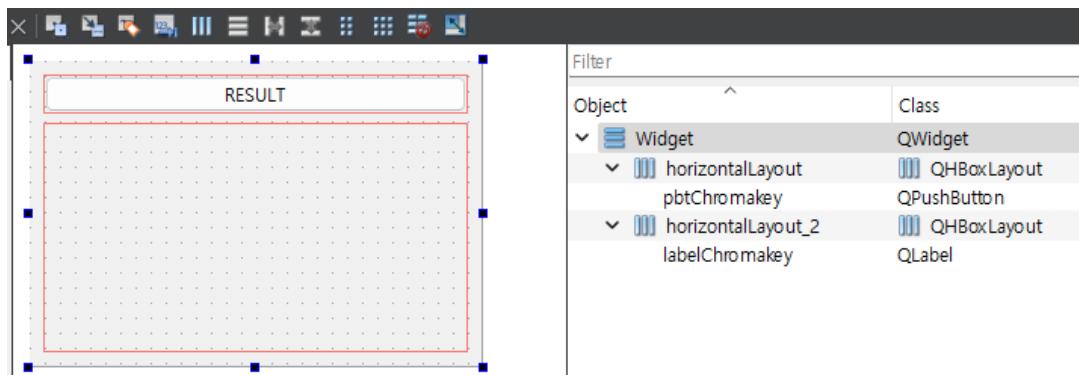
여기서 말하는 Chromakey란 특정 색을 필터링해 다른 이미지의 픽셀로 대체하는 방법을 말한다. 예를 들어 날씨 뉴스 방송의 배경을 다른 이미지나 영상으로 대체하는 영상을 많이 본적이 있을 것이다. 이번 예에서는 사진 이미지 포맷의 배경을 다른 이미지로 변경해 보도록 하자.



위의 그림에서 SOURCE 이미지상에서 배경은 녹색이다. SOURCE 이미지에서 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.



예수님은 당신을 사랑합니다.

위의 그림에서 보는 것과 같이 QPushButton 과 QLabel 을 배치한다. 그리고 다음 예제 소스코드에서 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 모두 동일해야 한다.

ImageProcessing 클래스에서 구현할 chromakeyProcess() 멤버 함수는 SOURCE 이미지 와 TARGET 이미지를 크로마키 처리하여 결과를 세 번째 인자에 넘겨준다. 다음 예제

예수님은 당신을 사랑합니다.

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
        {
```

예수님은 당신을 사랑합니다.

```
    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 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 RGB를 표시하기 위해서 1 Byte 만 사용하는 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;
```

예수님은 당신을 사랑합니다.

```
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 ImageProcessing 클래스로부터 결과를 저장할 이미지 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);
```

```
}

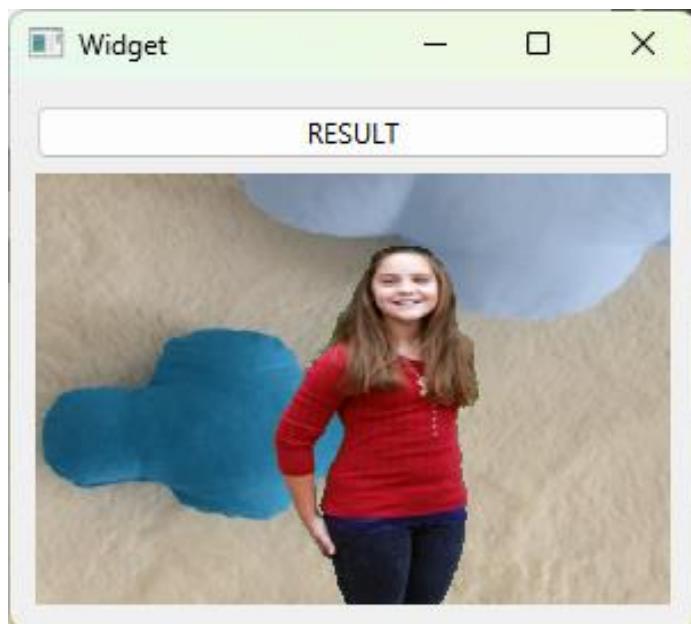
void Widget::slotChromakey()
{
    imgProcess->chromakeyProcess(sourceQImage, targetQImage);

    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;
}
...
```

QImage 클래스의 세 번째 인자는 QImage 가 사용할 RGB 포맷의 종류를 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. 작성한 예제를 실행하고 [RESULT] 버튼을 클릭해 결과를 확인해 보도록 하자.



예수님은 당신을 사랑합니다.

위의 그림에서 보는 것과 같이 QLabel 위젯 영역에 Result 이미지를 표시하였다. 예제의 전체 소스코드와 리소스는 00_Chromakey 디렉토리를 참조하면 된다.

19. Timer

QTimer 클래스는 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() Signal 은 지정한 시간이 경과되면 4번째 인자에 지정한 update() Slot 함수가 호출된다.

마지막 라인의 QTimer 클래스의 start() 멤버 함수는 첫 번째 인자로 지정한 시간만큼 경과되면 connect() 함수에서 명시한 Slot 함수가 반복해 호출된다. 첫 번째 인자의 단위는 millisecond 이다.

QTimer 클래스는 stop() 멤버 함수를 이용해 타이머를 정지할 수 있다. 만약 타이머를 반복해서 실행하지 않고 단 한번만 호출되도록 하기 위해 singleShot() 멤버 함수를 사용하면 된다. singleShot() 멤버 함수는 다음 예제 소스코드에서 보는 것과 같이 사용할 수 있다.

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

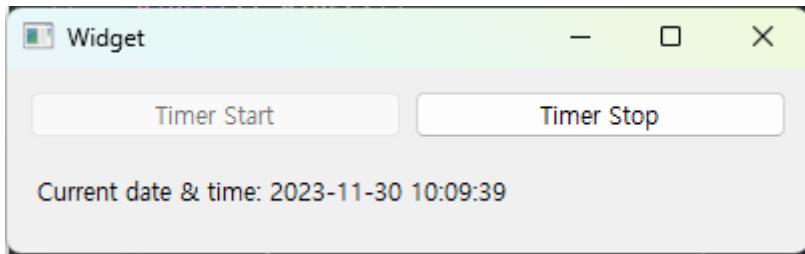
singleShot() 멤버 함수의 첫 번째 인자는 경과 되는 시간이며 단위는 밀리 세컨드이다. 그리고 세 번째 인자는 첫 번째 인자의 시간이 경과된 후 호출될 Slot 함수를 지정하면 된다.

- ✓ 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

예수님은 당신을 사랑합니다.

중지된다.



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.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QTimer>

namespace Ui {
class Widget;
}

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();
};

#endif // WIDGET_H
```

예수님은 당신을 사랑합니다.

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.

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

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;
    curr = QDateTime::currentDateTime();

    QString timeStr = QString(" Current date & time: %1 ")
                    .arg(curr.toString("yyyy-MM-dd hh:mm:ss"));

    ui->leCurrentTime->setText(timeStr);
}
```

이 예제의 전체 소스코드는 00_Timer 디렉토리를 참조하면 된다.

20. Thread programming

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.

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;
};

...
```

예수님은 당신을 사랑합니다.

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 구현 한 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.

예수님은 당신을 사랑합니다.

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 Thread를 생성하고 Reentrancy 와 Thread-Safety 를 고려하지 않은 Thread를 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;
        }
    }
};

public:
    Producer() {}
};

class Consumer : public QThread
{
    Q_OBJECT
public:
    void run() override {
        for(int i = 0 ; i < 10 ; i++) {
            qDebug("[Consumer] numUsed : %d", numUsed);
        }
    }
};

public:
    Consumer() { }
};
```

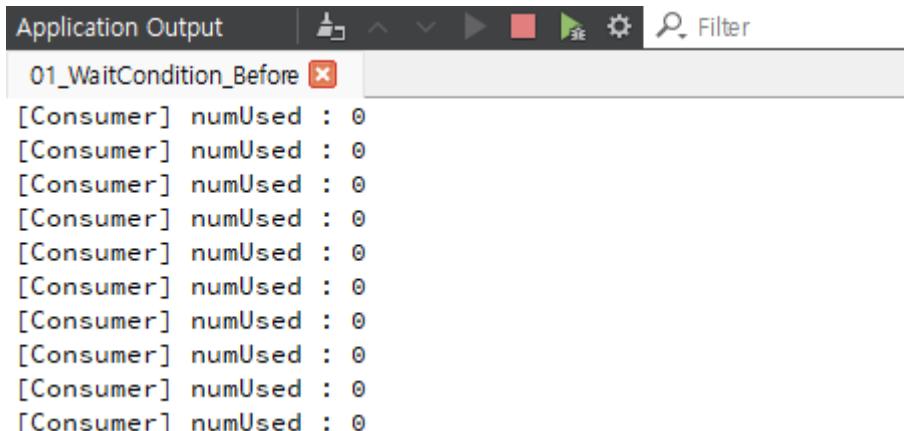
위의 예제 소스코드에서 보는 것과 같이 Producer 와 Consumer 클래스를 구현하고 main.cpp 에서 다음과 같이 작성해 보도록 하자.

```
#include <QCoreApplication>
#include "mythread.h"

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

```
{  
    QApplication 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.



The screenshot shows the 'Application Output' window with the title '01_WaitCondition_Before'. The window contains several lines of text output from the application. The text consists of multiple entries from the 'Consumer' class, each followed by a colon and the value '0'. The entries are: [Consumer] numUsed : 0, and [Consumer] numUsed : 0.

```
[Consumer] numUsed : 0  
[Consumer] numUsed : 0
```

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 보도록 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() { }
};

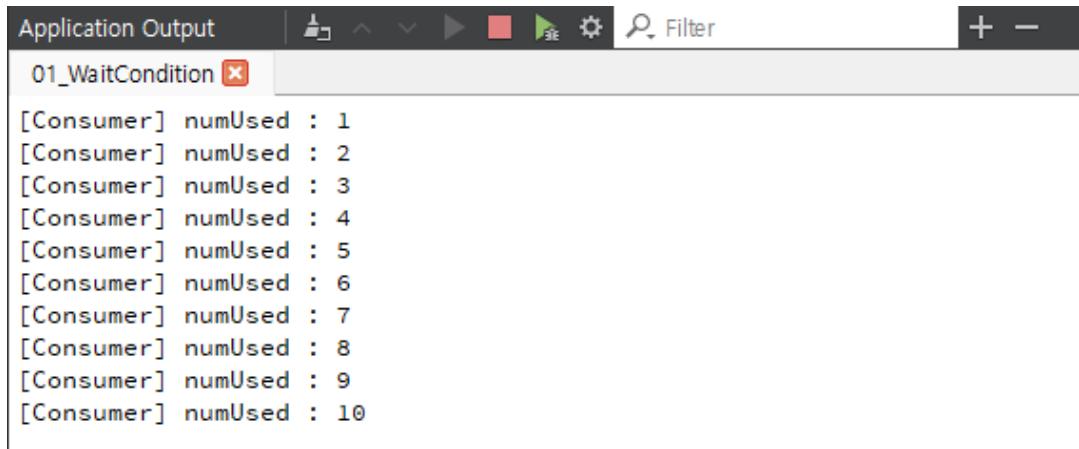
#endif // MYTHREAD_H
```

이전의 예제 소스코드에 작성했던 Producer 와 Consumer 클래스에서는 두 Thread 간의 실행과 관계 없이 Thread 가 동작한다. 하지만 수정한 Producer 와 Consumer 클래스

스는 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.



The screenshot shows the 'Application Output' tab in the Qt Creator interface. A single entry labeled '01_WaitCondition' is displayed. The output text is as follows:

```
[Consumer] numUsed : 1
[Consumer] numUsed : 2
[Consumer] numUsed : 3
[Consumer] numUsed : 4
[Consumer] numUsed : 5
[Consumer] numUsed : 6
[Consumer] numUsed : 7
[Consumer] numUsed : 8
[Consumer] numUsed : 9
[Consumer] numUsed : 10
```

예제 소스코드는 01_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 Thread 클래스를 작성하지 않고 특정 함수를 Thread 와 같이 동작 시킬 수 있다. 다음 예제 소스 코드는 QtConcurrent 를 이용한 예제 소스코드 이다.

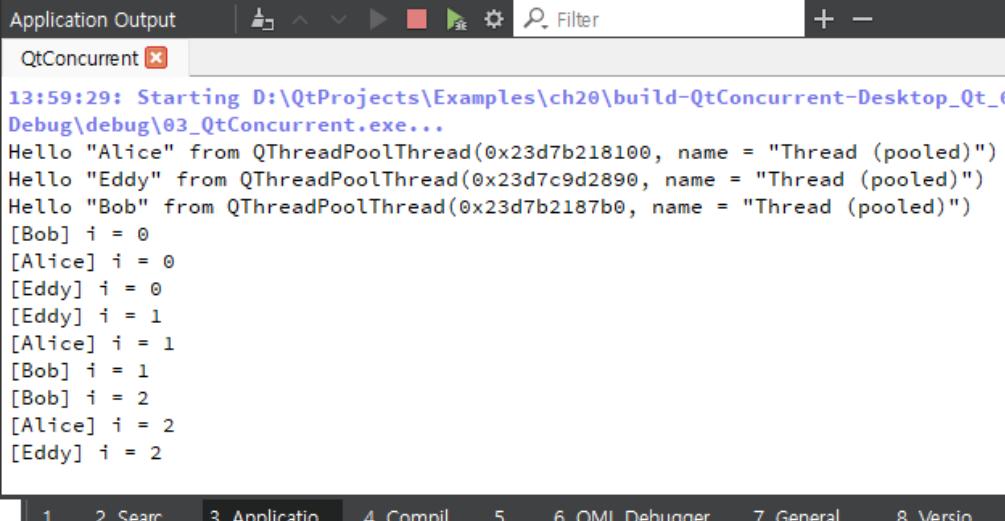
```
#include <QThread>
#include <QtConcurrent/QtConcurrent>
#include <QtConcurrent/QtConcurrentRun>

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.

예수님은 당신을 사랑합니다.



The screenshot shows the 'Application Output' tab in the Qt Creator interface. The title bar says 'Application Output'. Below it is a toolbar with icons for file operations, search, and filter. A dropdown menu is open, showing 'QtConcurrent' as the current tab. The main area displays the following text:

```
13:59:29: Starting D:\QtProjects\Examples\ch20\build-QtConcurrent-Desktop_Qt_6
Debug\debug\03_QtConcurrent.exe...
Hello "Alice" from QThreadPoolThread(0x23d7b218100, name = "Thread (pooled)")
Hello "Eddy" from QThreadPoolThread(0x23d7c9d2890, name = "Thread (pooled)")
Hello "Bob" from QThreadPoolThread(0x23d7b2187b0, name = "Thread (pooled)")
[Bob] i = 0
[Alice] i = 0
[Eddy] i = 0
[Eddy] i = 1
[Alice] i = 1
[Bob] i = 1
[Bob] i = 2
[Alice] i = 2
[Eddy] i = 2
```

At the bottom of the window, there is a navigation bar with tabs labeled 1, 2, Search..., 3 Application..., 4 Compilation..., 5 ..., 6 QML Debugger..., 7 General..., and 8 Version...

예제 전체 소스는 02_QtConcurrent 디렉토리를 참조하면 된다.

21. XML

Qt 는 XML(extensible Markup Language) 을 다룰 수 있는 API를 제공한다. Qt에서 제공하는 XML 모듈을 사용하기 위해서 프로젝트 파일에 다음과 같이 추가한다. 만약 빌드 도구로 qmake를 사용한다면 아래와 같이 추가한다.

```
QT += xml
```

만약 CMake를 사용한다면 아래와 같이 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS Xml)
target_link_libraries(mytarget PRIVATE Qt6::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 가능하며 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 디렉토리로 그로 선택한 파일을 읽어온다. 그리고 읽어온 데이터를 students라는 QList에 저장하

예수님은 당신을 사랑합니다.

기 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);

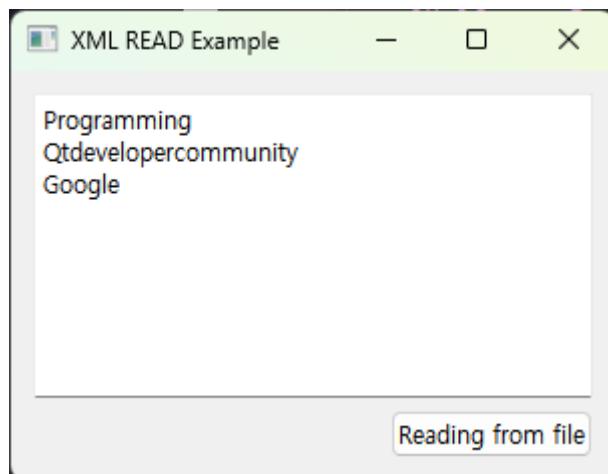
stream.setAutoFormatting(true);
stream.writeStartDocument();

...
stream.writeStartElement("bookmark");
stream.writeAttribute("href", "https://www.qt-dev.com");
stream.writeTextElement("title", "Qt developer community");
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.



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="https://qt-dev.com">
            <title>Qt developer community </title>
            </bookmark>

            <bookmark href="https://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.

```
#ifndef WIDGET_H
#define WIDGET_H

#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();
};

#endif // WIDGET_H
```

예수님은 당신을 사랑합니다.

mReadFile 오브젝트는 XML 파일을 읽어 올 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)) {
        return;
    }

    if(!mReadFile->open(QIODevice::ReadOnly)) {
        ui->textEdit->setText("File open failed.");
    }
}
```

```
    return;
}

QXmlStreamReader reader(mReadFile);

QString inputData;
while(!reader.atEnd())
{
    reader.readNext();
    if(!reader.text().isEmpty()) {
        QString data = reader.text().toString();
        data.replace(" ", "");
        data.replace('\n', "");
        data.replace('\t', "");

        if(data.length() > 0) {
            inputData.append(data).append("<br>");
        }
    }
}

ui->textEdit->setText(inputData);
}
```

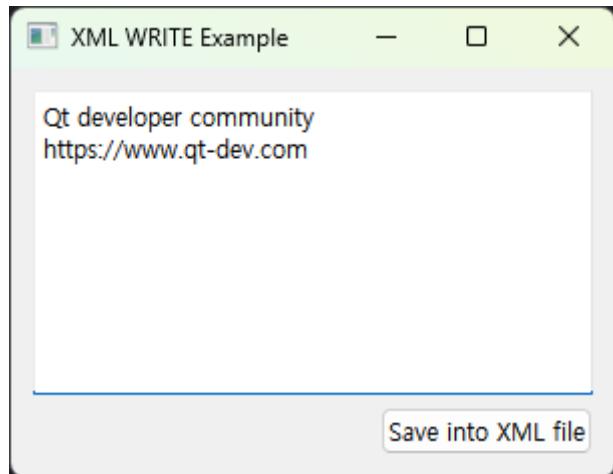
위의 예제 소스코드에서 보는 것과 같이 readButtonClicked() Slot 함수가 호출되면 파일 디아일로그가 로딩된다.

파일 디아일로그에서 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 QTextEdit에 출력하기 위해 QString에 저장한다. 저장 후 XML 파일 READ가 끝난 후 저장된 QString에 저장된 텍스트를 QTextEdit에 출력한다. 예제 전체 소스는 00_ReaderExample 디렉토리를 참조하면 된다.

- ✓ QXmlStreamWriter 를 이용해 XML 파일에 저장하는 예제 구현

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.

예수님은 당신을 사랑합니다.



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 developer community");
    mOriData.append("https://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("db:/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 tag
    xmlWriter.writeEndElement(); // Qt End tag
    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 장성 후 파일을 닫으면 "D:/output.xml" 파일에 XML데이터가 저장되며

결과는 다음과 같다.

```
<?xml version="1.0" encoding="UTF-8"?>
<Qt>
  <Info>
    <Name>Qt developer community</Name>
    <URL>https://www.qt-dev.com</URL>
  </Info>
</Qt>
```

작성한 예제 전체 소스코드는 01_WriteExample 디렉토리를 참조하면 된다.

✓ DOM 방식의 예제 구현

이번 예제는 DOM(Document object model) 방식을 이용해 파일로부터 XML포맷 데이터를 읽어오는 예제이다. 아래 그림에서 보는 것과 같이 dom.xml 파일명으로 저장한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<Building>
  <KR Name="Trade Center"></KR>
  <KR Name="Lotte Tower"></KR>
  <KR Name="Namsan Tower"></KR>
</Building>
```

위의 XML 데이터를 파일로부터 읽어 오기 위해 QDomDocument 클래스를 사용할 수 있으며 소스코드는 다음과 같이 작성한다.

```
#include <QApplication>
#include <QtXml>
#include <QtDebug>

void retrievElements(QDomElement root, QString tag, QString att)
{
    QDomNodeList nodes = root.elementsByTagName(tag);

    qDebug() << "Node counts = " << nodes.count();
    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(":/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, "KR", "Name");
    qDebug() << "Reading finished";

    return a.exec();
}
```

`retrievElements()` 함수의 첫 번째 인자는 `QDomElement` 클래스의 오브젝트이다. 두 번째 인자는 XML 데이터에서 “KR” 태그, 세 번째 인자는 “Name” 속성(Attribute)을 지정한다. 지정한 태그와 속성을 `qDebug()` 를 이용해 Console 출력한다.

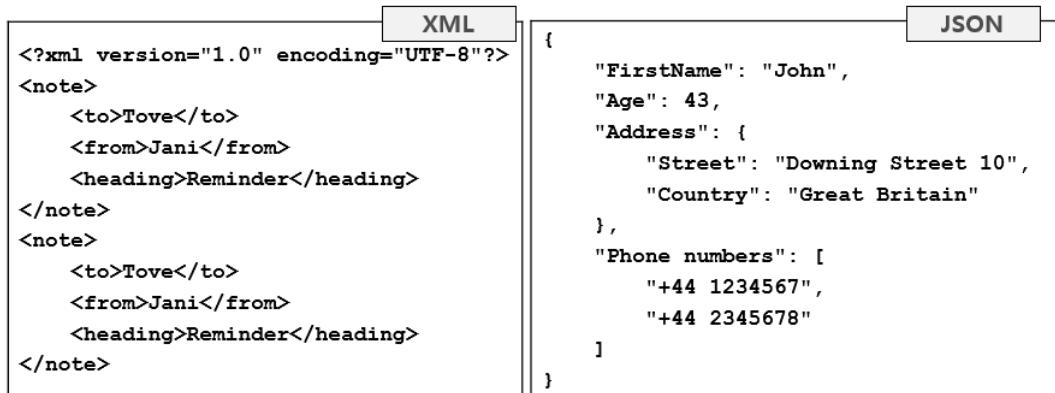
```
Node counts = 3
Attribute : "Trade Center"
Attribute : "Lotte Tower"
Attribute : "Namsan Tower"
Reading finished
```

예제 전체 소스코드는 02_DomExample 디렉토리를 참조하면 된다.

22. JSON

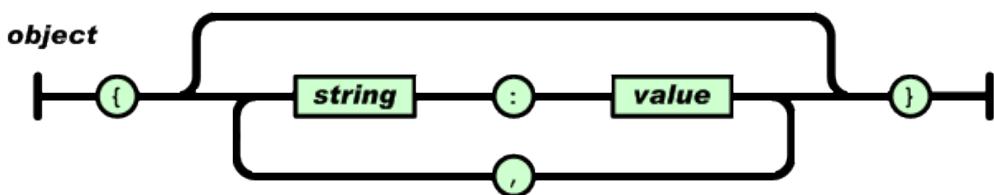
JSON (JavaScript Object Notation)의 사용 목적은 XML과 동일하다. 차이점은 경량의 DATA 교환 방식으로 XML보다 데이터 사용 용량이 작다는 장점을 가지고 있다.

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.



<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"
    }
}
```

예수님은 당신을 사랑합니다.

```
},
"Phone numbers": [ "+44 1234567", "+44 2345678" ]
}
```

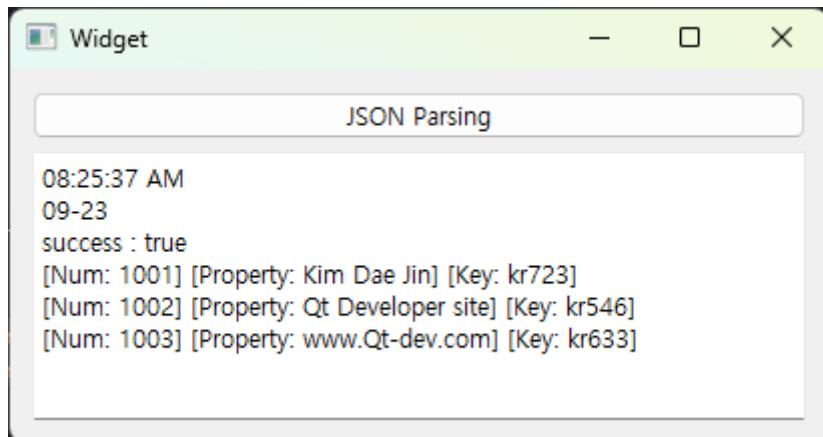
위의 예제에서 보는 것과 같이 Qt에서 제공하는 JSON 모듈은 Bool, Double, String, Array, Object, Null 과 같은 6가지의 데이터 타입을 사용할 수 있다. { (좌 중괄호)로 시작하고 } (우 중괄호)로 끝내어 표현할 수 있다.

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" }
    ]
}
```



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();
    }
}
```

예수님은 당신을 사랑합니다.

```
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. 예제 소스코드와 JSON 샘플 파일은 00_JSON_Parser 디렉토리를 참조하면 된다.

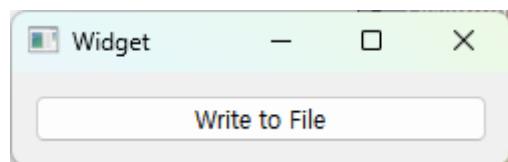
✓ JSON 포맷의 파일을 작성하는 예제

이번 예제에서는 아래에서 보는 것과 같이 JSON 포맷의 내용을 파일에 저장하는 예제를 작성해 보도록 하자.

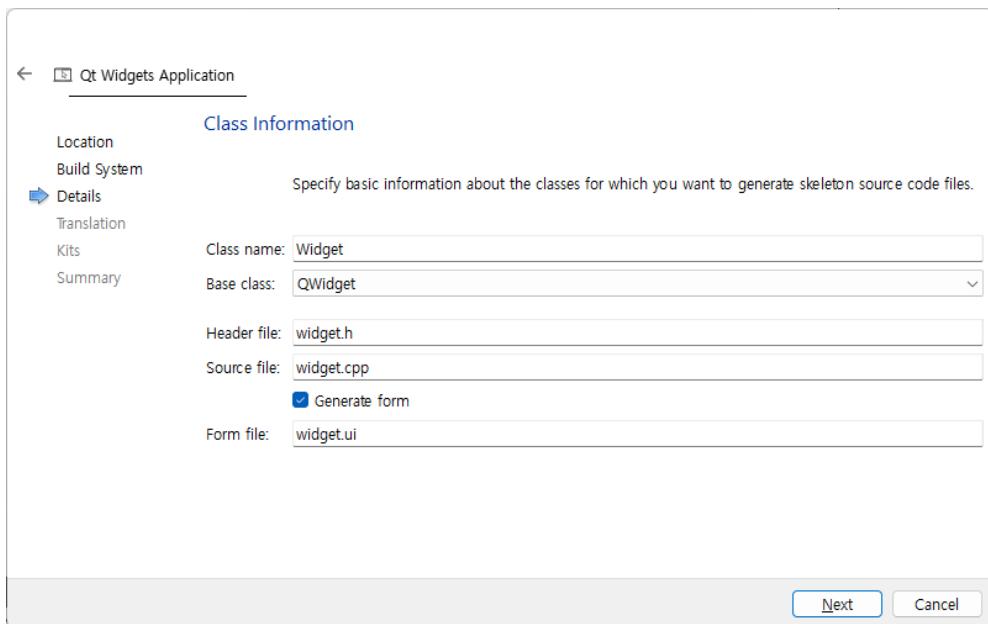
```
[
{
    "Address": {
        "City": "Yongin",
        "Nation": "South Korea"
    }
}
```

```
},
  "FirstName": "Dae Jin",
  "LastName": "Kim",
  "School": {
    "Grade": 3,
    "Major": "Computer science"
  }
},
{
  "Address": {
    "City": "Suwon",
    "Nation": "South Korea"
  },
  "FirstName": "Gil Dong",
  "LastName": "Hong",
  "School": {
    "Grade": 2,
    "Major": "Math"
  }
}
]
```

위에서 보는 것과 같은 JSON 포맷의 내용을 파일에 저장할 것이다. 작성할 어플리케이션은 아래 실행 화면에서 [Write to File] 버튼을 위의 JSON 포맷의 내용을 파일에 저장할 것이다.



프로젝트 생성 시 Class Information 을 입력하는 디아일로그에서 Form file 을 생성한다. [Generate form] 의 체크박스에 체크를 하면 된다.



프로젝트 생성 완료 후 widget.h 파일에 다음과 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

    typedef struct _tMember {
        QString firstName;
        QString lastName;
    
```

```
QString addr_nation;
QString addr_city;

int school_grade;
QString school_major;

} tMember;

QList<tMember> m_memList;

void writeJSON();

private slots:
    void slot_pbtWriteFile();

};

#endif // WIDGET_H
```

tMember 구조체는 JSON에 저장할 데이터를 임시 저장할 구조체이다. writeJSON() 멤버 함수는 JSON 포맷의 내용을 저장하는 기능이 구현된 함수이다. Slot_pbtWriteFile() 함수는 [Write to File] 버튼을 클릭 시 호출되는 Slot 함수이다. 아래 소스 코드는 widget.cpp 이다.

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

#include <QJsonDocument>
#include <QJsonArray>
#include <QJsonObject>
#include <QFile>
#include <QDebug>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pbtWirteFile, &QPushButton::clicked,
            this, &Widget::slot_pbtWriteFile);
}

Widget::~Widget()
```

```
{  
    delete ui;  
}  
  
void Widget::writeJSON()  
{  
    qDebug() << Q_FUNC_INFO;  
  
    QFile file("d:/write_sample.json");  
  
    if(!file.open(QIODevice::ReadWrite)) {  
        qDebug() << "File open error";  
        return;  
    }  
  
    file.resize(0);  
  
    tMember member1;  
    member1.firstName = QString("Dae Jin");  
    member1.lastName = QString("Kim");  
    member1.addr_nation = QString("South Korea");  
    member1.addr_city = QString("Yongin");  
    member1.school_grade = 3;  
    member1.school_major = QString("Computer science");  
  
    tMember member2;  
    member2.firstName = QString("Gil Dong");  
    member2.lastName = QString("Hong");  
    member2.addr_nation = QString("South Korea");  
    member2.addr_city = QString("Suwon");  
    member2.school_grade = 2;  
    member2.school_major = QString("Math");  
  
    m_memList.append(member1);  
    m_memList.append(member2);  
  
    QJsonArray jsonArray;  
  
    for(qsizetype i = 0 ; i < m_memList.size() ; i++)  
    {  
        QJsonObject jsonObject;
```

```
jsonObject.insert("FirstName", m_memList.at(i).firstName);
jsonObject.insert("LastName", m_memList.at(i).lastName);

QJsonObject jsonAddrObject;
jsonAddrObject.insert("Nation", m_memList.at(i).addr_nation);
jsonAddrObject.insert("City", m_memList.at(i).addr_city);

QJsonObject jsonSchoolObject;
jsonSchoolObject.insert("Grade", m_memList.at(i).school_grade);
jsonSchoolObject.insert("Major", m_memList.at(i).school_major);

jsonObject.insert("School", jsonSchoolObject);
jsonObject.insert("Address", jsonAddrObject);

JSONArray.append(jsonObject);
}

QJsonDocument jsonDocment;

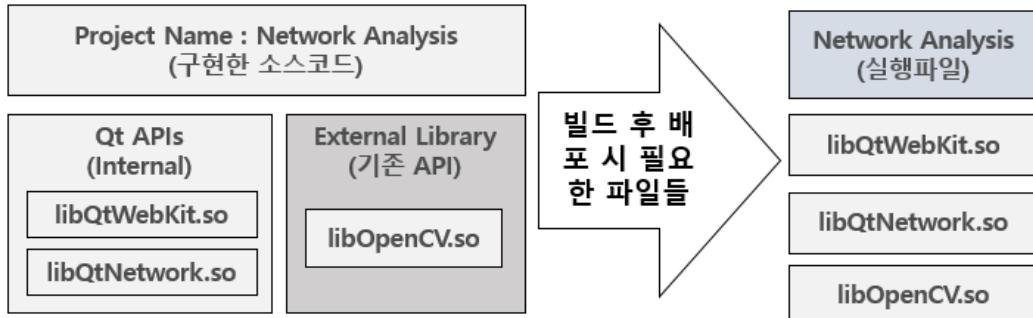
jsonDocment.setArray(JSONArray);
file.write(jsonDocment.toJson());
file.close();

}

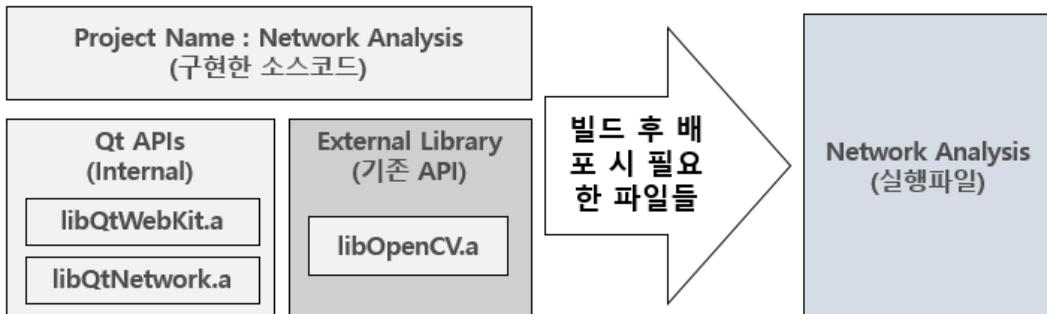
void Widget::slot_pbtWriteFile()
{
    writeJSON();
}
```

23. 라이브러리 제작 (CMake)

Qt 는 Shared 라이브러리 방식의 라이브러리와 Static 라이브러리를 사용할 수 있다. Shared 라이브러리는 아래 그림에서 보는 것과 같이 실행 파일과 라이브러리 파일이 분리된 형태이다.



Static 라이브러리 방식은 빌드 시 라이브러리들이 실행 파일과 하나의 파일로 통합된다. 따라서 Static 방식으로 빌드하였다면 배포 시 실행 파일만 배포하면 된다.



Qt에서 외부 라이브러리를 사용할 때 주의해야 할 점으로 외부 라이브러리를 컴파일한 컴파일러와 동일한 컴파일러를 사용해 빌드해야 한다.

예를 들어 응용 어플리케이션을 MinGW 기반 컴파일러로 빌드 하였다면 외부 라이브러리 또한 MinGW 기반의 컴파일러로 빌드한 라이브러리를 사용해야 한다.

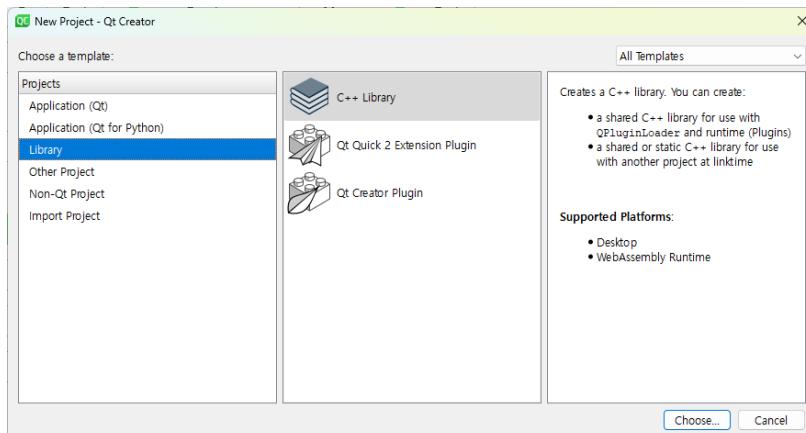
이번 장에서는 직접 Shared 라이브러리를 만들어본 다음에 구현한 라이브러리를 사용하는 예제를 다루어 보도록 하자.

23.1. Shared 라이브러리 제작과 사용하기

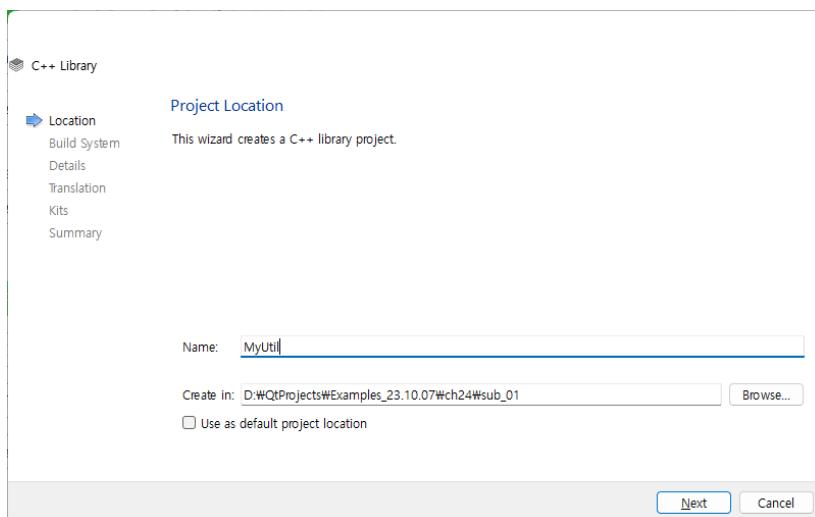
이번 장에서는 두 개의 예제 프로젝트를 작성할 것이다. 첫 번째는 라이브러리 프로젝트이다. 라이브러리 프로젝트에서는 라이브러리를 구현할 것이며 두 번째 프로젝트에서는 구현한 라이브러리를 사용하는 프로젝트이다.

✓ 라이브러리 구현 예제

Qt Creator에서 프로젝트를 생성한다. 프로젝트 생성ダイ얼로그가 생성되면 다음으로 그 창의 왼쪽 탭에서 [Library]를 선택하고 중간 탭에서 [C++ Library]를 선택한다.

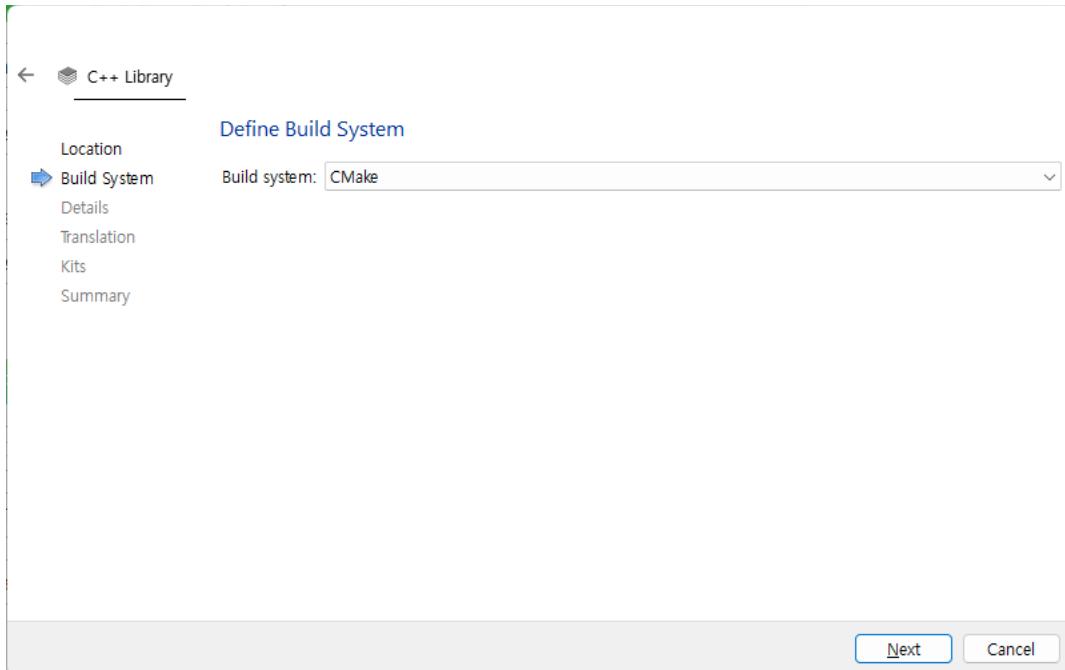


다음ダイ얼로그에서 프로젝트 Name으로 "MyUtil"을 입력하고 하단의 [Next] 버튼을 클릭한다.

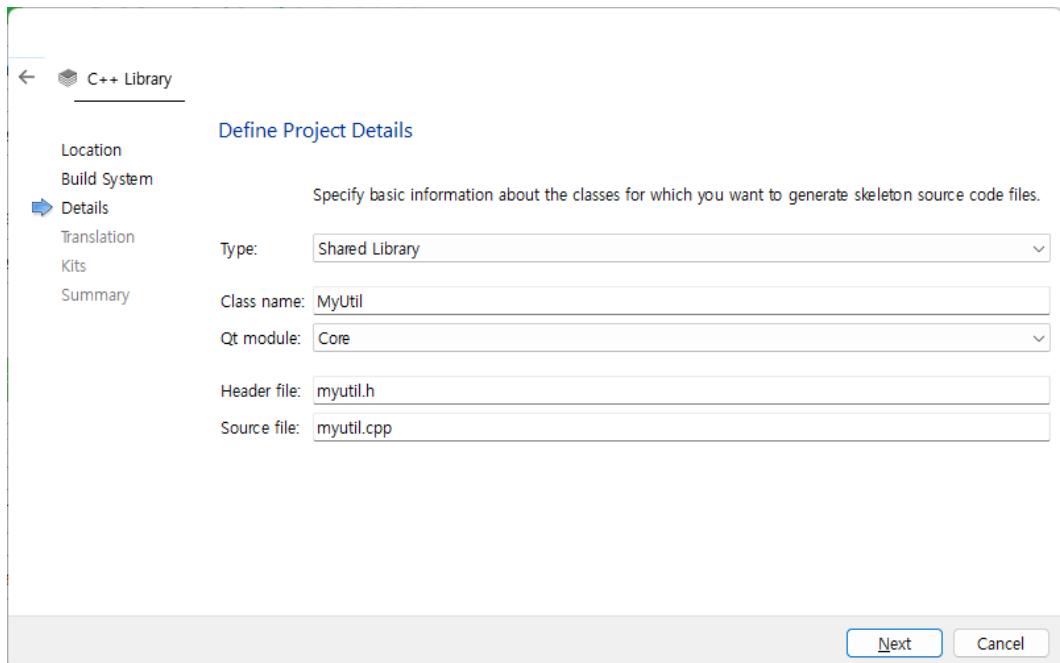


예수님은 당신을 사랑합니다.

Build system 으로 CMake를 선택한다.

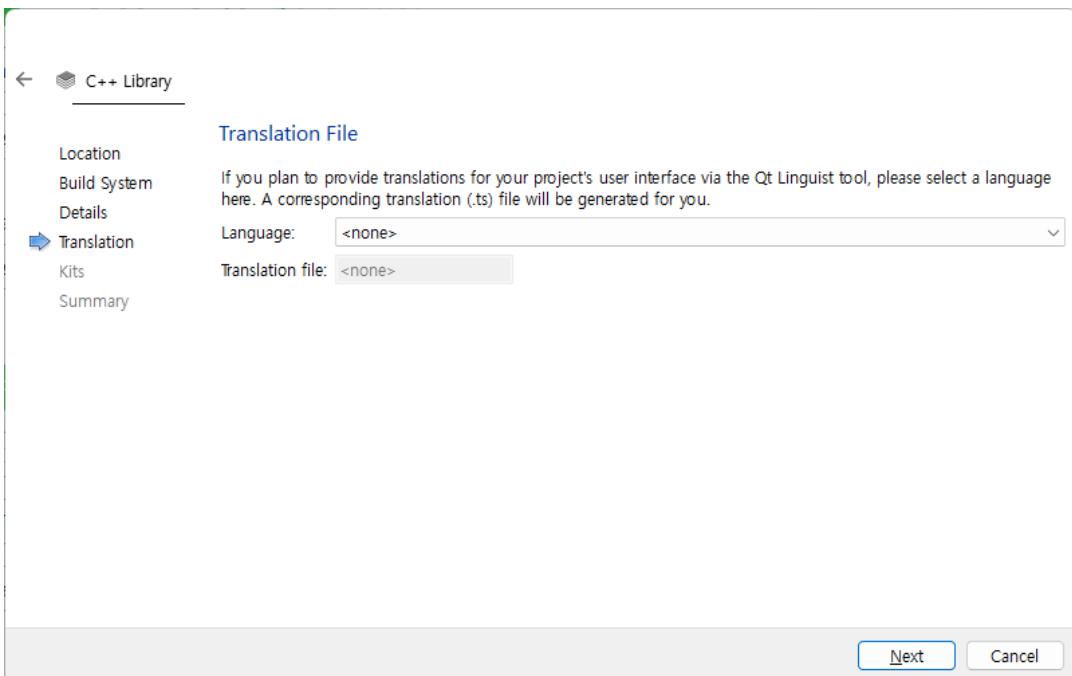


Type 으로 Shared Library를 선택한다. 그리고 Class 이름은 "MyUtil" 을 입력한다. Qt Module 은 "Core" 를 선택한다. Header file 명은 "myutil.h" 을 입력하고 Source file 명은 "myutil.cpp" 를 입력한다.

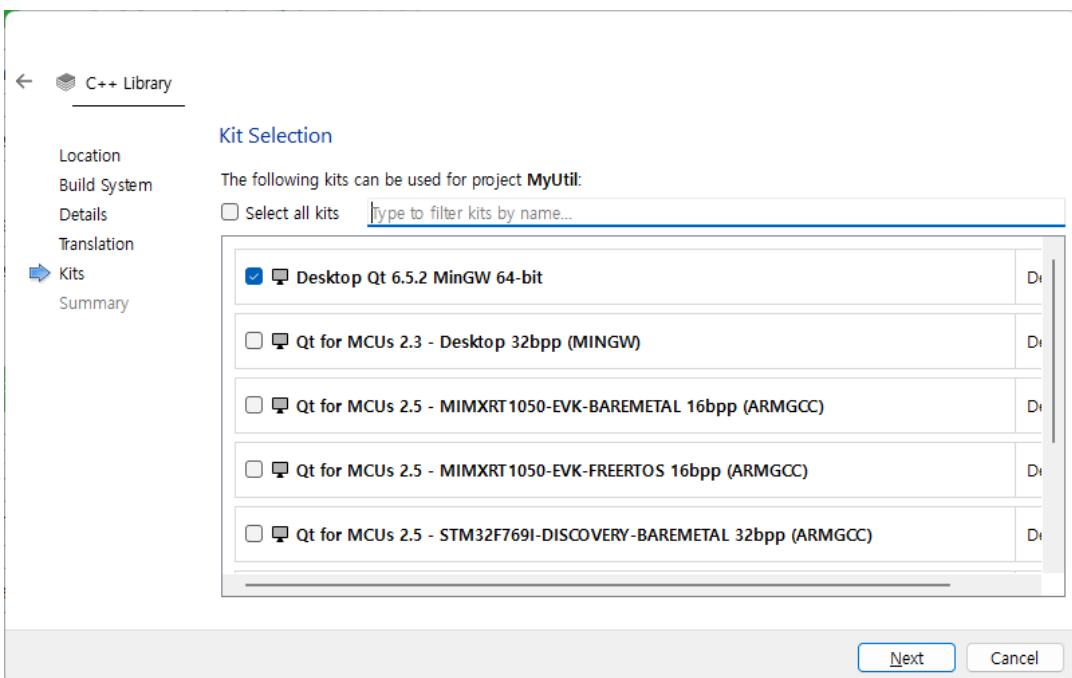


다음 다이얼로그에서는 디플트 상태로 둔 상태에서 [Next] 버튼을 클릭한다.

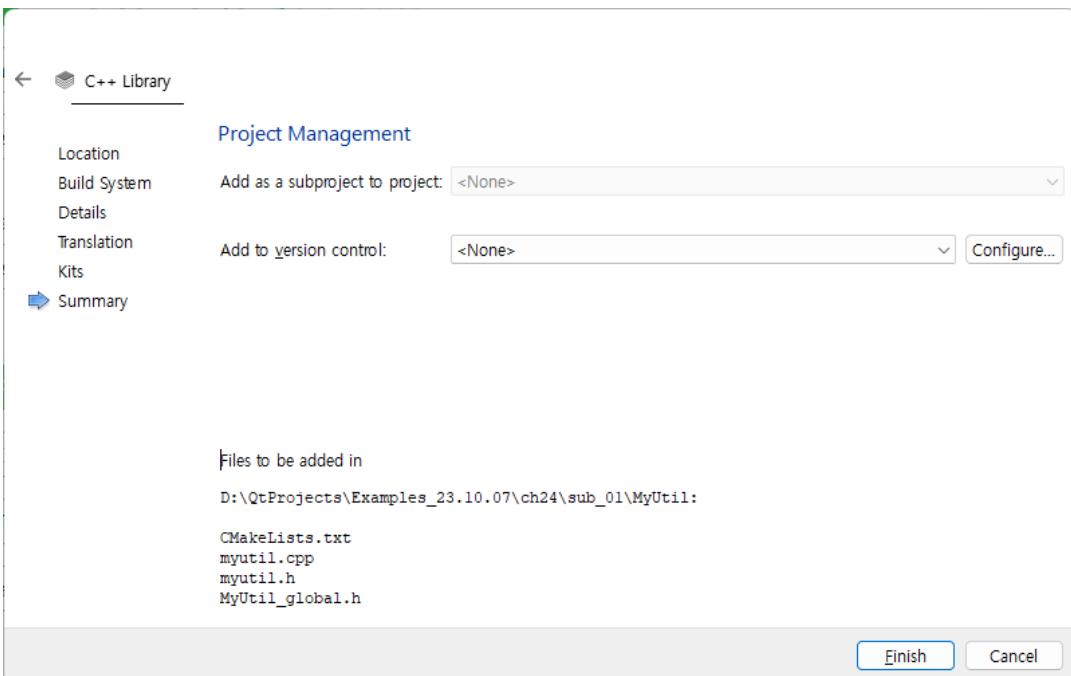
예수님은 당신을 사랑합니다.



Kit Selection 다이얼로그에서는 MinGW 를 선택하고 하단의 [Next] 버튼을 클릭한다.



다음 다이얼로그에서는 소스코드 버전 관리 시스템을 선택하는 다이얼로그이다. 아래 그림에서 보는 것과 같이 디폴트 상태로 두고 [Finish] 버튼을 클릭한다.



다음으로 myutil.h 헤더파일에서 아래와 같이 소스코드를 작성한다.

```
#ifndef MYUTIL_H
#define MYUTIL_H

#include <QObject>
#include "MyUtil_global.h"

class MYUTIL_EXPORT MyUtil : public QObject
{
public:
    explicit MyUtil(QObject *parent = nullptr);
    int getSumValue(int a, int b);

};

#endif // MYUTIL_H
```

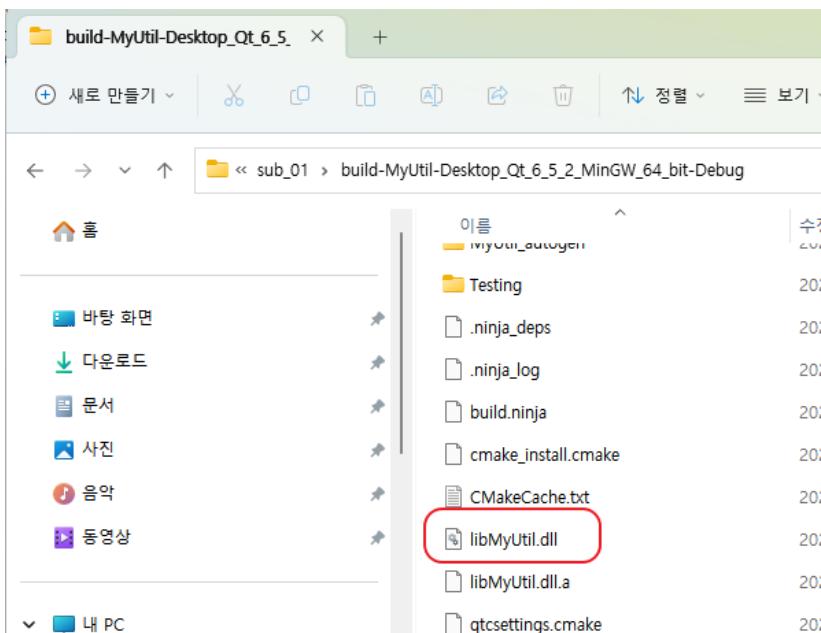
그리고 myutil.cpp 를 열어서 아래와 같이 소스코드를 작성한다.

```
#include "myutil.h"
```

```
MyUtil::MyUtil(QObject *parent) : QObject(parent)
{
}

int MyUtil::getSumValue(int a, int b)
{
    return a + b;
}
```

MyUtil 클래스는 간단하게 두 개의 값을 함수 인자로 넘겨주면 결과 값으로 두 개의 값을 더한 값을 넘겨준다. 여기까지 작성하였다면 빌드한 다음, 빌드 된 디렉토리에 가면 라이브러리 파일이 생성된 것을 확인할 수 있다.



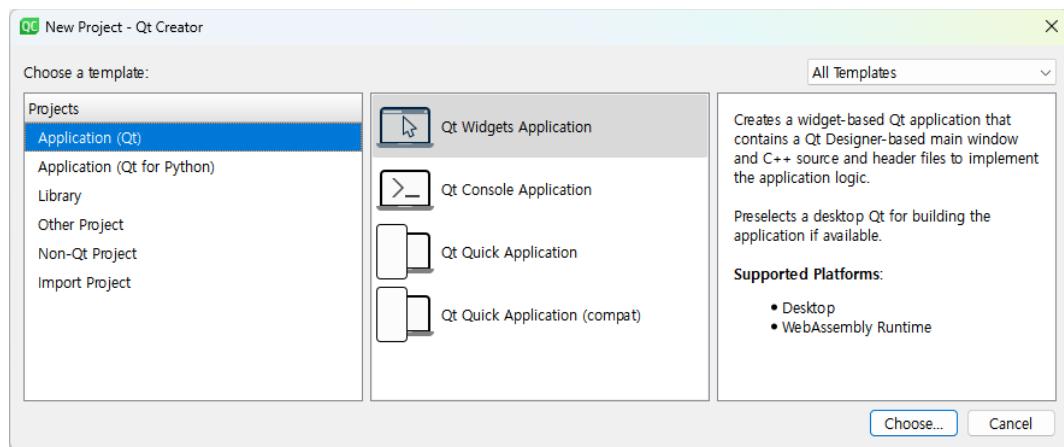
위의 그림에서 보는 것과 같이 정상적으로 라이브러리가 생성되었다면 이 라이브러리 사용하는 예제를 작성해 보도록 하자.

✓ 작성한 라이브러리를 사용하는 예제

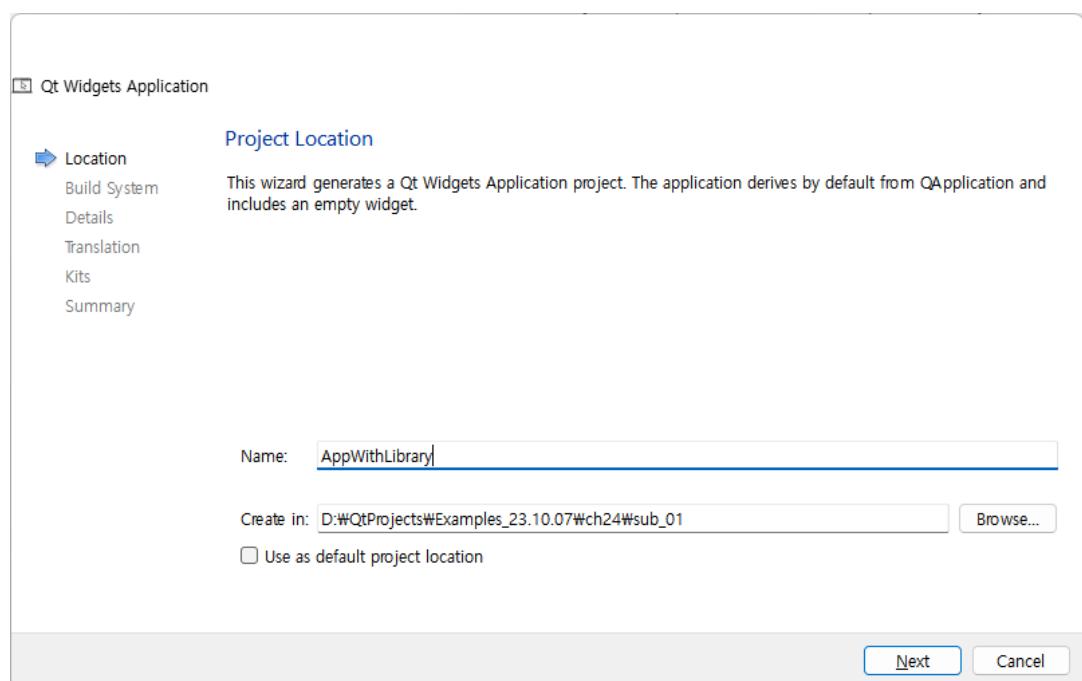
아래 그림에서 보는 것과 같이 2개의 값을 입력한 다음 [Result] 버튼을 누르면 라이브러리에서 작성한 MyUtil 클래스의 getSumValue() 함수를 이용해 결과 값을 리턴 받는 예제를 작성해 보도록 하자. 프로젝트 생성 시 아래 그림에서 보는 것과 같이 [Qt

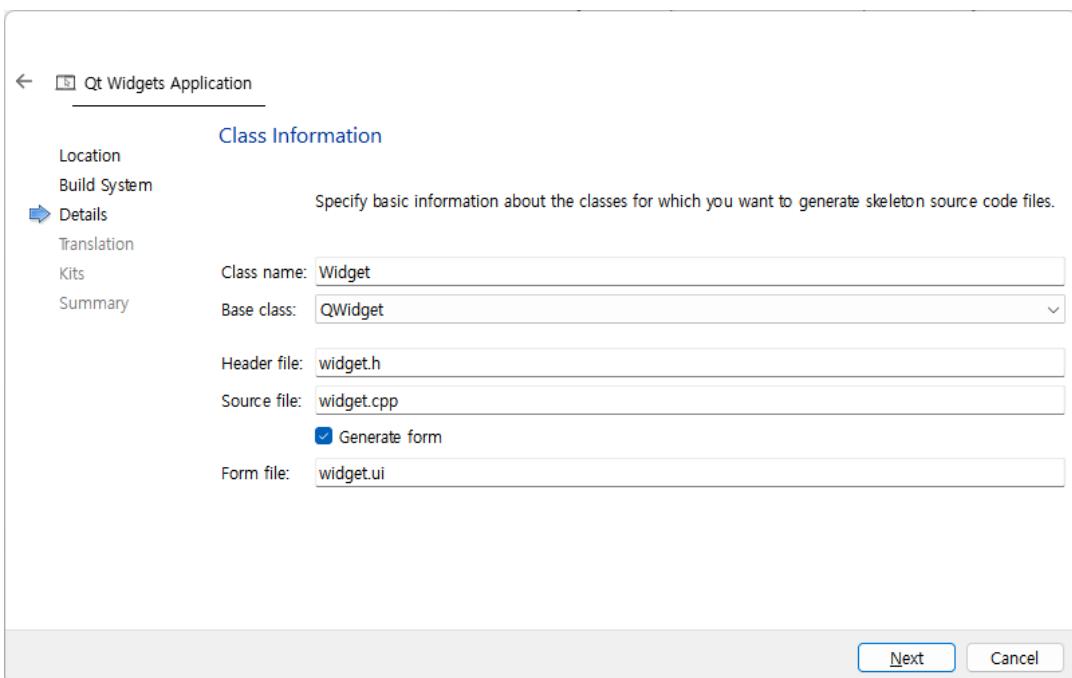
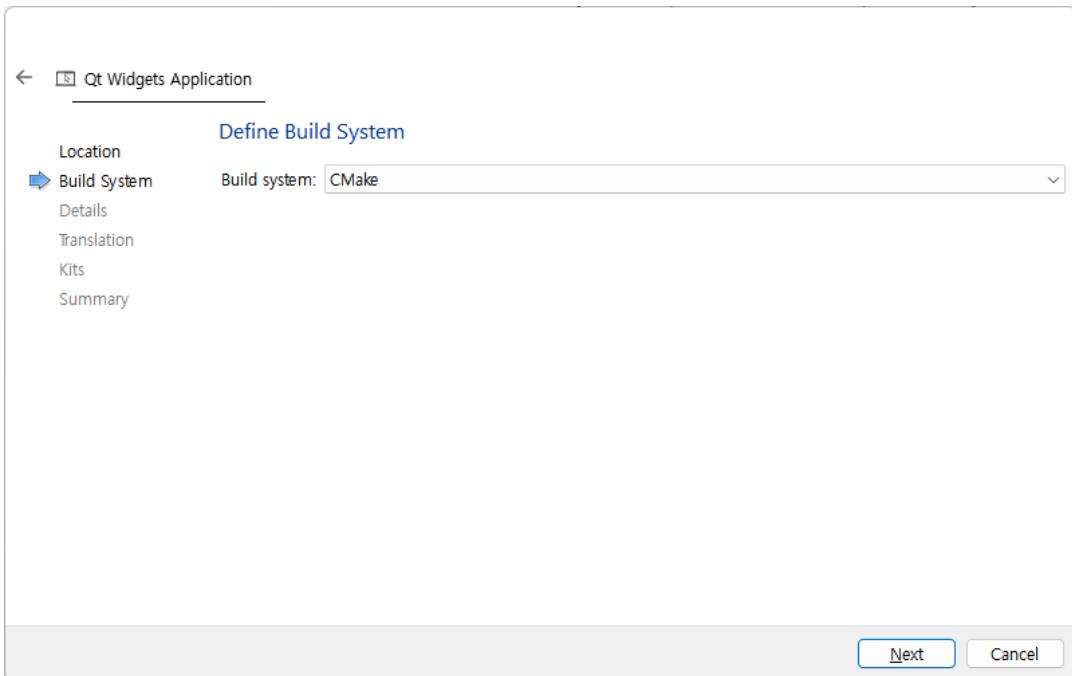
예수님은 당신을 사랑합니다.

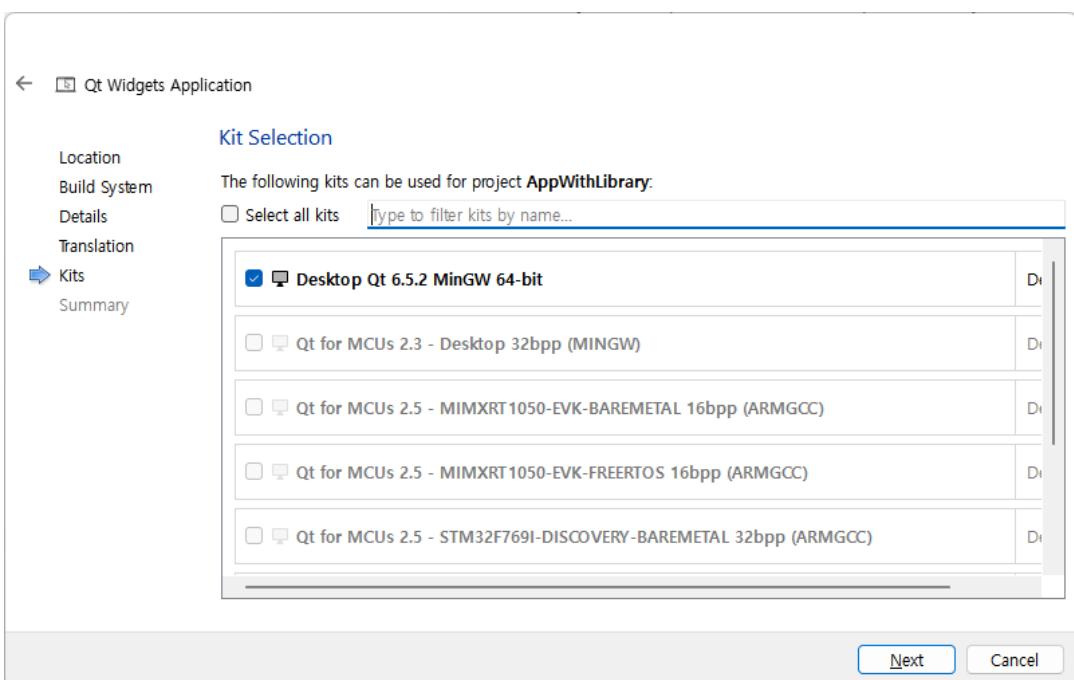
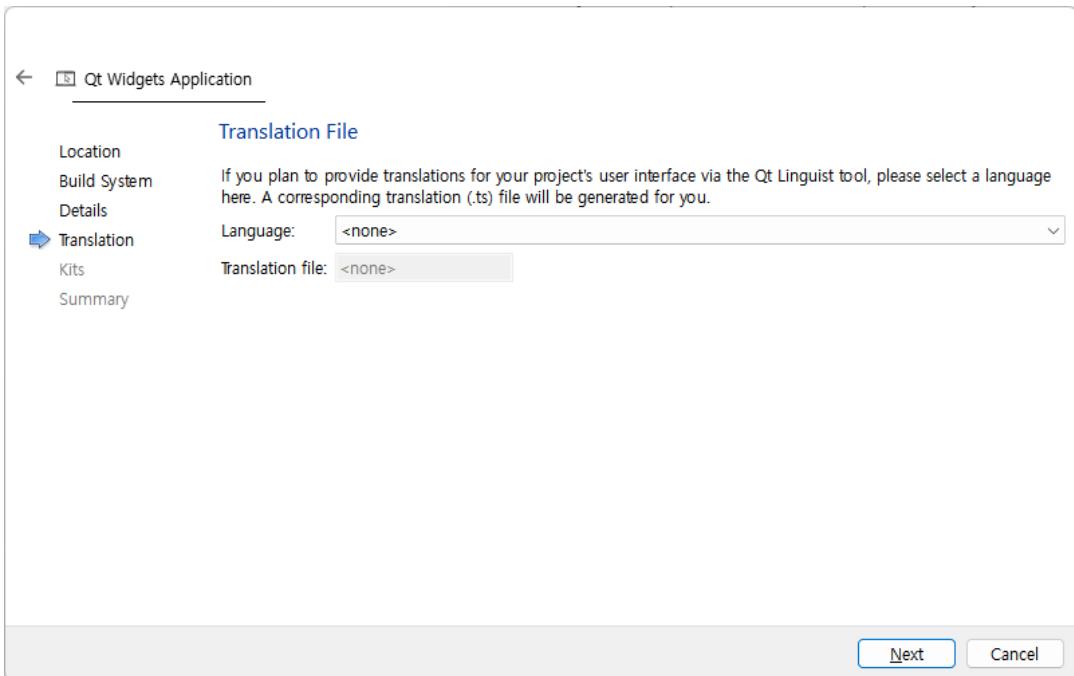
Widget Application] 을 선택한다.

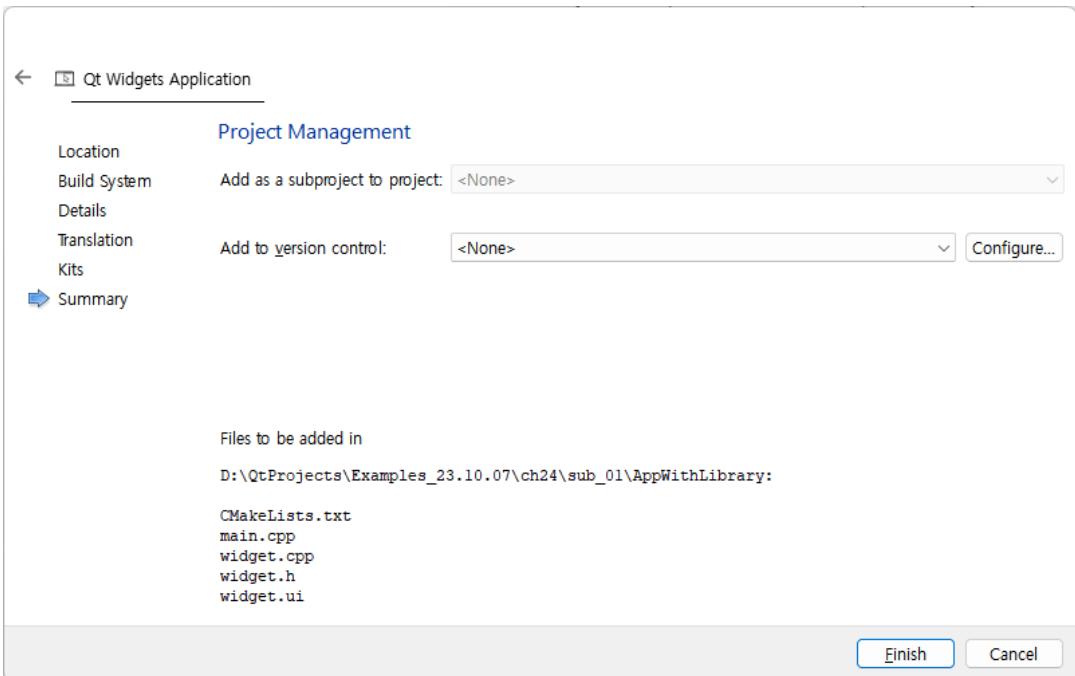


프로젝트 생성 시, 라이브러리파일과 헤더 파일의 위치를 명시해야 하기 때문에 MyUtil 프로젝트가 생성된 동일한 위치에 프로젝트를 생성한다.









프로젝트 생성이 완료되면 CMakeList.txt 파일을 열어서 아래와 같이 추가한다.

```
cmake_minimum_required(VERSION 3.5)

project(AppWithLibrary VERSION 0.1 LANGUAGES CXX)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt NAMES Qt6 Qt5 REQUIRED COMPONENTS Widgets)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Widgets)

set(PROJECT_SOURCES
    main.cpp
    widget.cpp
    widget.h
    widget.ui)
```

```
)  
  
include_directories(../MyUtil)  
link_directories(../build-MyUtil-Desktop_Qt_6_5_2_MinGW_64_bit-Debug)  
  
if(${QT_VERSION_MAJOR} GREATER_EQUAL 6)  
    qt_add_executable(AppWithLibrary  
        MANUAL_FINALIZATION  
        ${PROJECT_SOURCES}  
    )  
else()  
    if(APPLE)  
        add_library(AppWithLibrary SHARED  
            ${PROJECT_SOURCES}  
        )  
    else()  
        add_executable(AppWithLibrary  
            ${PROJECT_SOURCES}  
        )  
    endif()  
endif()  
  
target_link_libraries(AppWithLibrary MyUtil)  
  
target_link_libraries(AppWithLibrary Qt${QT_VERSION_MAJOR}::Widgets)  
  
if(${QT_VERSION} VERSION_LESS 6.1.0)  
    set(BUNDLE_ID_OPTION MACOSX_BUNDLE_GUI_IDENTIFIER com.example.AppWithLibrary)  
endif()  
set_target_properties(AppWithLibrary PROPERTIES  
    ${BUNDLE_ID_OPTION}  
    MACOSX_BUNDLE_BUNDLE_VERSION ${PROJECT_VERSION}  
    MACOSX_BUNDLE_SHORT_VERSION_STRING  
    ${PROJECT_VERSION_MAJOR}.${PROJECT_VERSION_MINOR}  
    MACOSX_BUNDLE TRUE  
    WIN32_EXECUTABLE TRUE
```

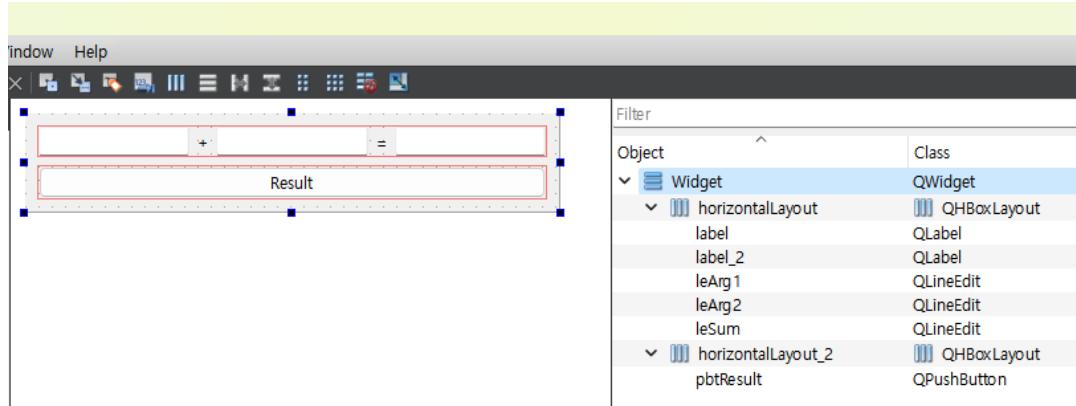
```
)  
  
include(GNUInstallDirs)  
install(TARGETS AppWithLibrary  
        BUNDLE DESTINATION .  
        LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}  
        RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}  
)  
  
if(QT_VERSION_MAJOR EQUAL 6)  
    qt_finalize_executable(AppWithLibrary)  
endif()
```

set() 다음에 include_directories() 를 추가 한다. 이 함수는 라이브러리 함수의 헤더의 위치를 가리킨다.

다음으로 link_directories() 함수를 추가한다. 이 함수는 라이브러리 파일이 있는 위치를 가리킨다.

마지막으로 target_link_libraries() 함수를 추가한다. 이 함수는 라이브러리 명을 명시한다.

위와 같이 라이브러리를 추가했다면 아래 그림에서 보는 것과 같이 widget.ui 파일을 열어서 UI를 작성한다.



위와 같이 UI 를 작성한다. 그리고 [Result] 버튼을 클릭하면 라이브러리의 MyUtil 클래스에서 작성한 getSumValue() 함수를 호출해 결과를 받아 온 다음 leSum 오브젝트에 결과를 표시하도록 소스코드를 작성해 보도록 하자.

먼저 widget.h 소스코드 파일을 열어서 아래와 같이 소스코드를 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include "myutil.h"

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
    MyUtil *m_myUtil;

private slots:
    void slot_pbtResult();

};

#endif // WIDGET_H
```

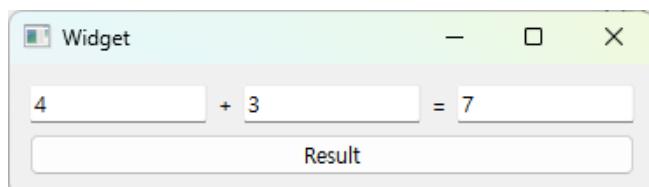
다음으로 widget.cpp 소스코드 파일을 열어서 아래와 같이 작성해 보도록 하자.

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

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
```

```
{  
    ui->setupUi(this);  
  
    m_myUtil = new MyUtil();  
  
    connect(ui->pbtResult, &QPushButton::clicked,  
            this,           &Widget::slot_pbtResult);  
}  
  
Widget::~Widget()  
{  
    delete ui;  
}  
  
void Widget::slot_pbtResult()  
{  
    qDebug() << Q_FUNC_INFO;  
  
    qint32 arg1 = ui->leArg1->text().toInt();  
    qint32 arg2 = ui->leArg2->text().toInt();  
  
    //qint32 sumValue = 0;  
    qint32 sumValue = m_myUtil->getSumValue(arg1, arg2);  
    ui->leSum->setText(QString("%1").arg(sumValue));  
}
```

위와 같이 작성했다면 결과를 확인해 보도록 하자.



위의 예제는 sub_01 디렉토리를 참조하면 된다.

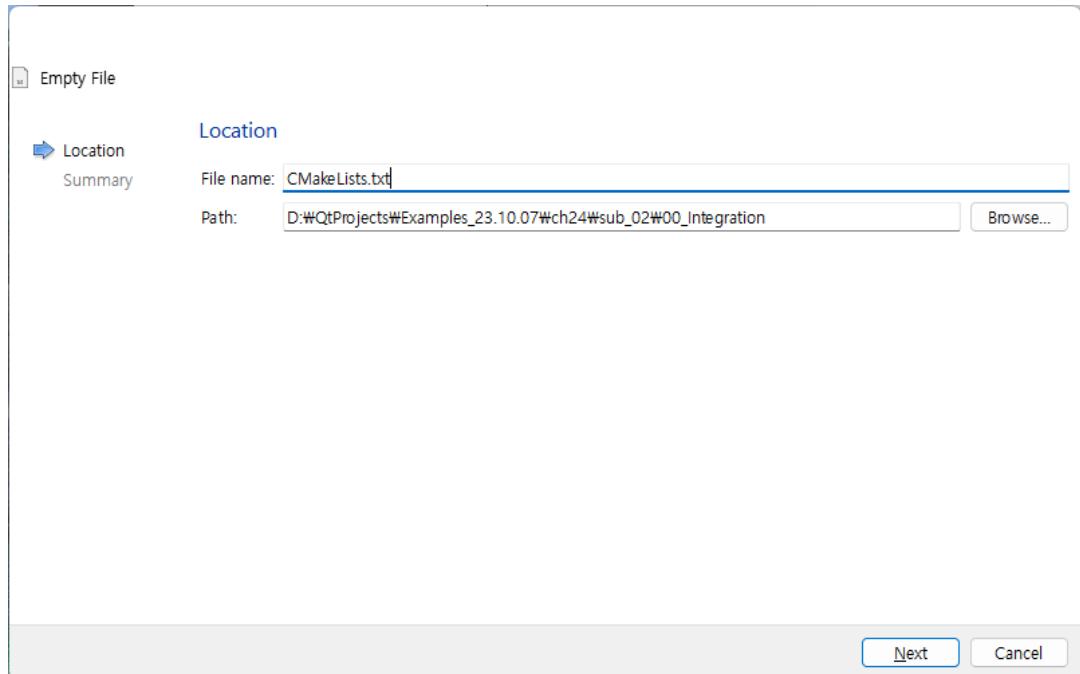
23.2. 라이브러리와 함께 빌드하기

Qt에서는 두개의 프로젝트를 한번에 일괄 빌드하기 위한 방법을 제공한다. 이번 장에서는 두개의 프로젝트를 한 개의 프로젝트로 묶어서 빌드하는 방법에 대해서 알아보도록 하자.

첫 번째로 CMakeList.txt 파일을 Qt Creator 툴을 이용해 생성한다. CMakeLists.txt 파일 생성 시 메모장과 같은 툴을 이용해도 무방하다. 여기서는 Qt Creator 툴을 이용해 CMakeLists.txt 파일을 생성해 보도록 하겠다.

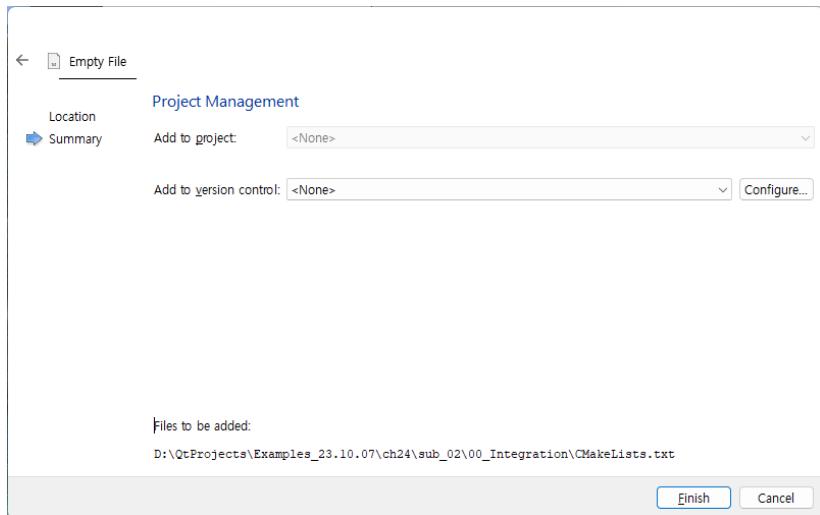
Qt Creator의 [File] 메뉴에서 [New File]을 클릭한다. 그리고 아래 그림에서와 같이 왼쪽 탭에서 [General]을 선택하고 중간 탭에서 [Empty File]을 선택한다.

그리고 아래 그림에서 보는 것과 같이 File name 항목에 "CMakeLists.txt"를 입력하고 하단의 [Next] 버튼을 클릭한다.

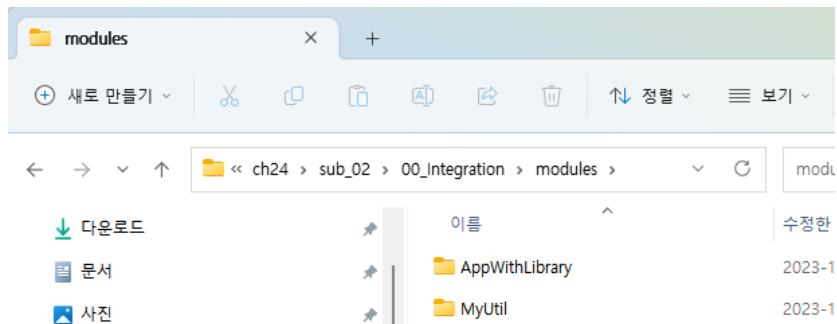


그리고 다음 다이얼로그 창에서 디폴트로 둔 상태에서 [Finish] 버튼을 클릭한다.

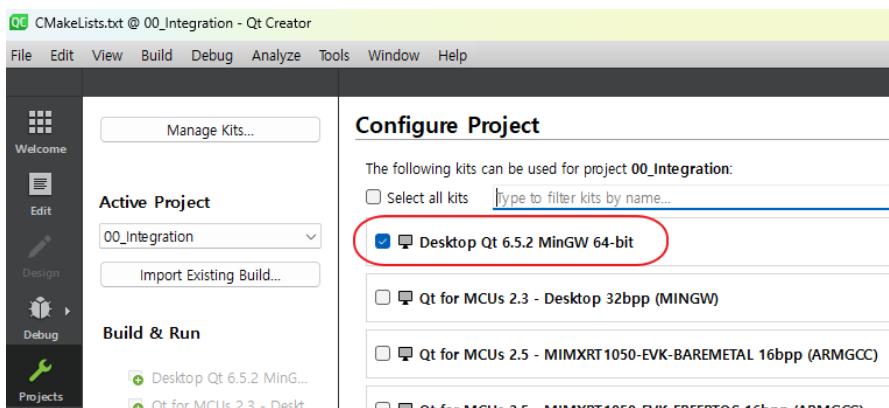
예수님은 당신을 사랑합니다.



다음으로 CMakeLists.txt 파일이 생성된 위치에 modules라는 디렉토리를 만들고 그 디렉토리안에 이 전장에서 작성한 예제 2개를 복사한다.



그런 다음 CMakeLists.txt 파일을 Qt Creator에서 Open 한다. 이 파일을 Open하게 되면 프로젝트를 빌드할 컴파일러를 선택해야 한다. 아래 그림에서 보는 것과 같이 MinGW를 선택한다.



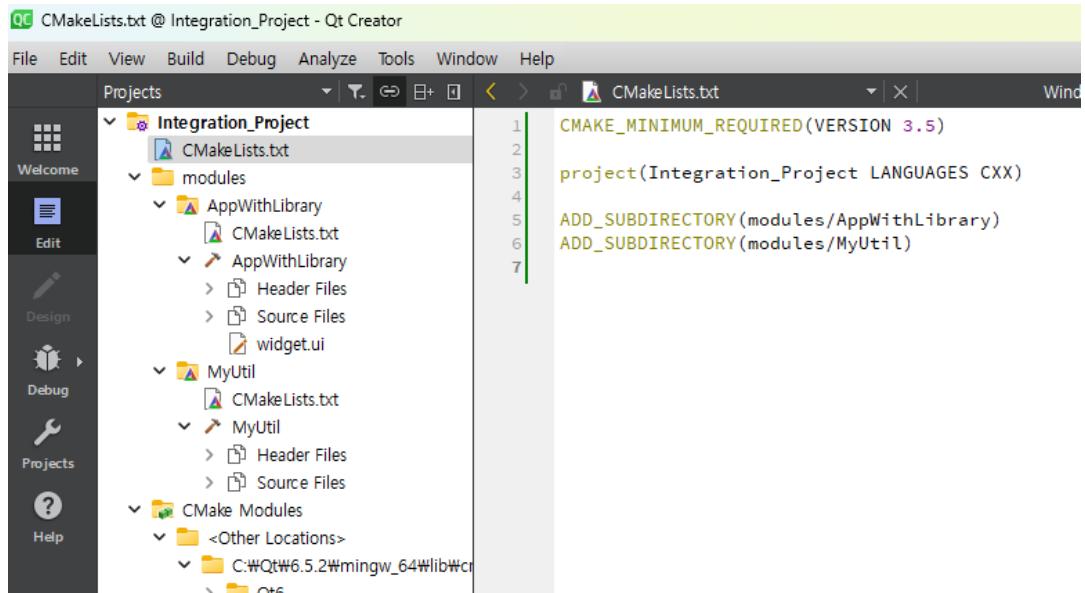
빈 프로젝트가 생성되면 2개의 프로젝트를 추가하기 위해서 CMakeLists.txt 파일을 열어서 아래와 같이 작성한다.

```
CMAKE_MINIMUM_REQUIRED(VERSION 3.5)

project(Integration_Project LANGUAGES CXX)

ADD_SUBDIRECTORY(modules/AppWithLibrary)
ADD_SUBDIRECTORY(modules/MyUtil)
```

위와 같이 작성하고 빌드하면 2개의 프로젝트가 일괄 빌드 될 것이다. 이렇게 하면 장점으로 수정 사항이 있을 경우, 특정 프로젝트를 수정 한 다음 빌드하게 되면 일괄 빌드를 하기 때문에 각 프로젝트를 Qt Creator에서 별도로 빌드하지 않아도 되는 장점이 있다.

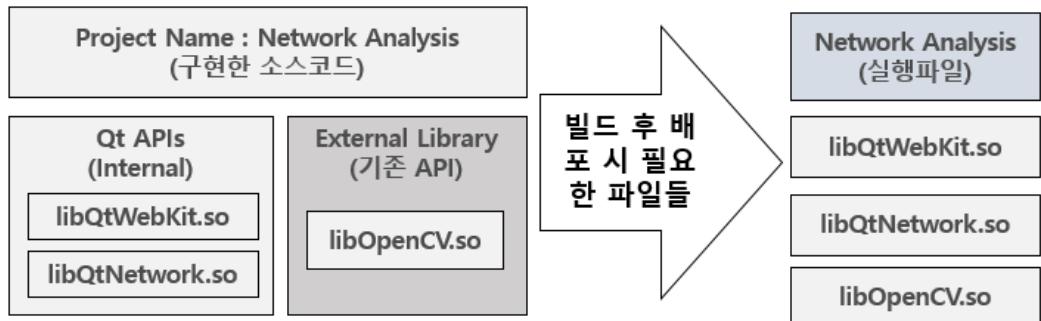


이 프로젝트의 예제는 sub_02 > 00_Integration 디렉토리를 참조하면 된다.

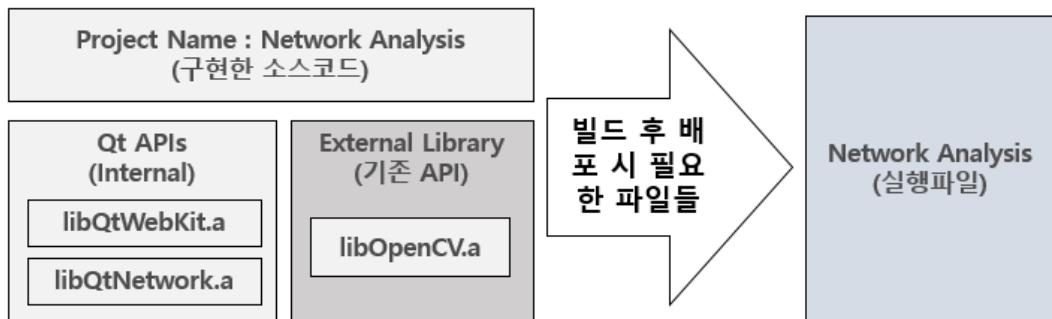
24. 라이브러리 제작 (qmake)

이번 장에서는 이전 장에서 다루었던 내용과 동일하다. 하지만 이전 장에서는 CMake를 사용했다면 이번 장에서는 qmake를 이용해 라이브러리를 구현하는 방법과 사용하는 방법에 대해서 살펴보도록 하겠다.

Qt는 Shared 라이브러리 방식의 라이브러리와 Static 라이브러리를 사용할 수 있다. Shared 라이브러리는 아래 그림에서 보는 것과 같이 실행 파일과 라이브러리 파일이 분리된 형태이다.



Static 라이브러리 방식은 빌드 시 라이브러리들이 실행 파일과 하나의 파일로 통합된다. 따라서 Static 방식으로 빌드하였다면 배포 시 실행 파일만 배포하면 된다.



Qt에서 외부 라이브러리를 사용할 때 주의해야 할 점으로 외부 라이브러리를 컴파일한 컴파일러와 동일한 컴파일러를 사용해 빌드해야 한다.

예를 들어 응용 어플리케이션을 MinGW 기반 컴파일러로 빌드 하였다면 외부 라이브러리 또한 MinGW 기반의 컴파일러로 빌드한 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.

"-L" 은 라이브러리를 위치한 디렉토리 이다. 그리고 세 번째 라인의 -lOpenCL 은 실제 사용할 Shared 라이브러리 파일을 명시하면 된다. 라이브러리명 앞에 "-l" 옵션은 라이브러리 파일이라는 뜻이다. 라이브러리 파일에 위치한 디렉토리에 보면 확장자가 리눅스인 경우 so이며 MS윈도우 인 경우 dll 이거나 lib 이다.

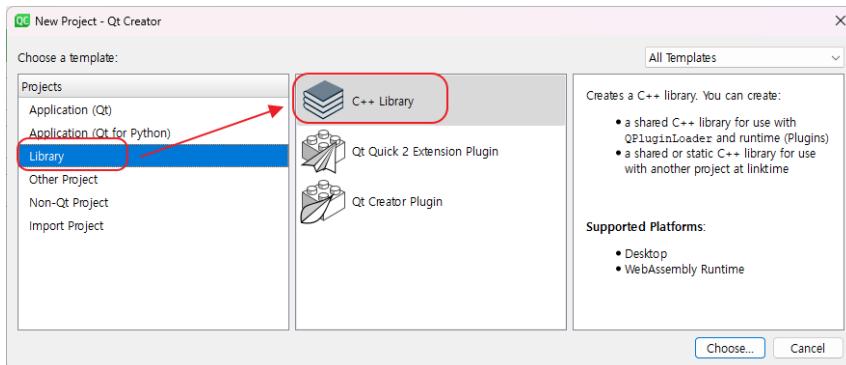
우리는 이번 장에서 라이브러리를 구현하는 방법과 통합 빌드하는 방법을 소 단원으로 나누어 살펴보도록 하겠다.

24.1. Shared 라이브러리 제작과 사용하기

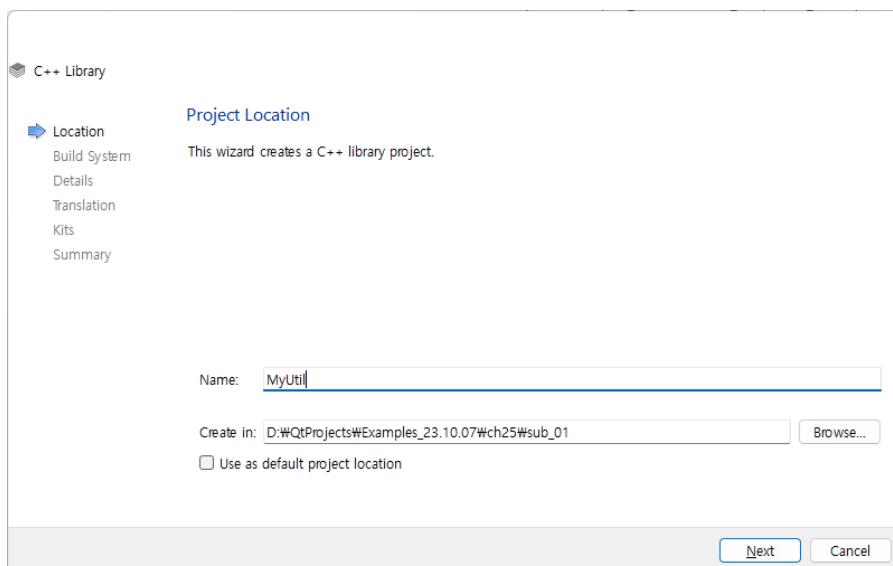
이번 예제에서는 2개의 Qt 프로젝트를 생성할 것이다. 첫 번째 프로젝트는 라이브러리 구현을 위한 프로젝트이다. 두 번째 프로젝트는 첫 번째 프로젝트에서 구현한 라이브러리를 사용하는 예제이다.

✓ 라이브러리 구현 예제

Qt Creator에서 새로운 프로젝트를 생성한다. 프로젝트 생성 다이얼로그에서 다음 그림에서 보는 것과 같이 좌측 상단 [Projects] 탭에서 Library를 선택하고 가운데 리스트에서 C++ Library 항목을 선택한다.

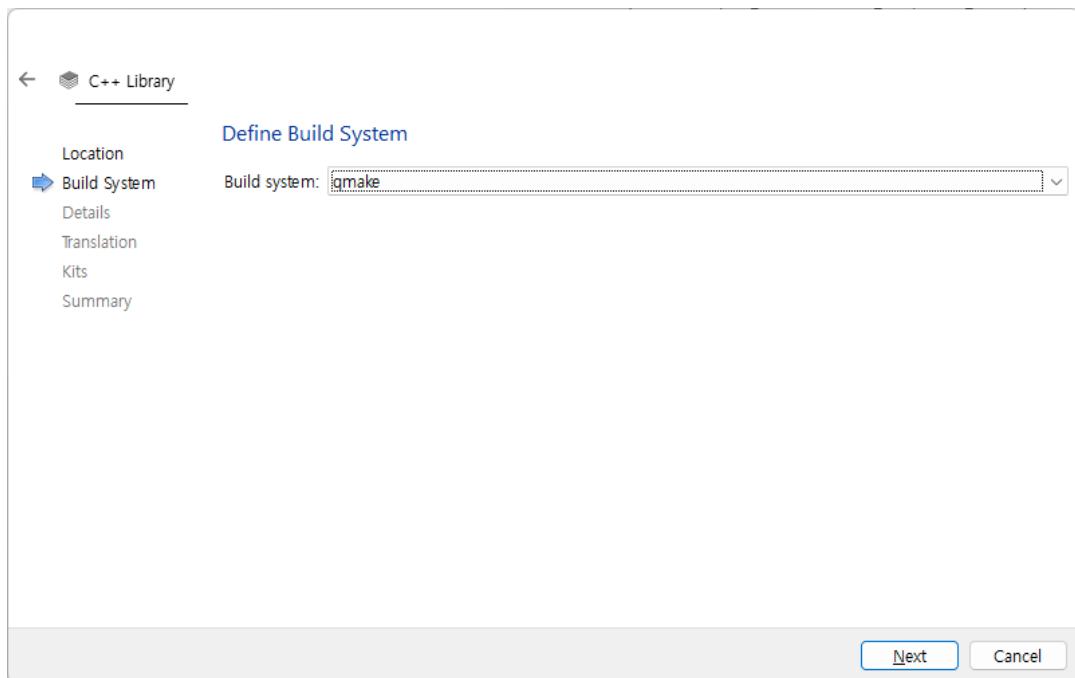


다음 다이얼로그 화면에서 Name 항목에 “MyUtil”을 입력한다.

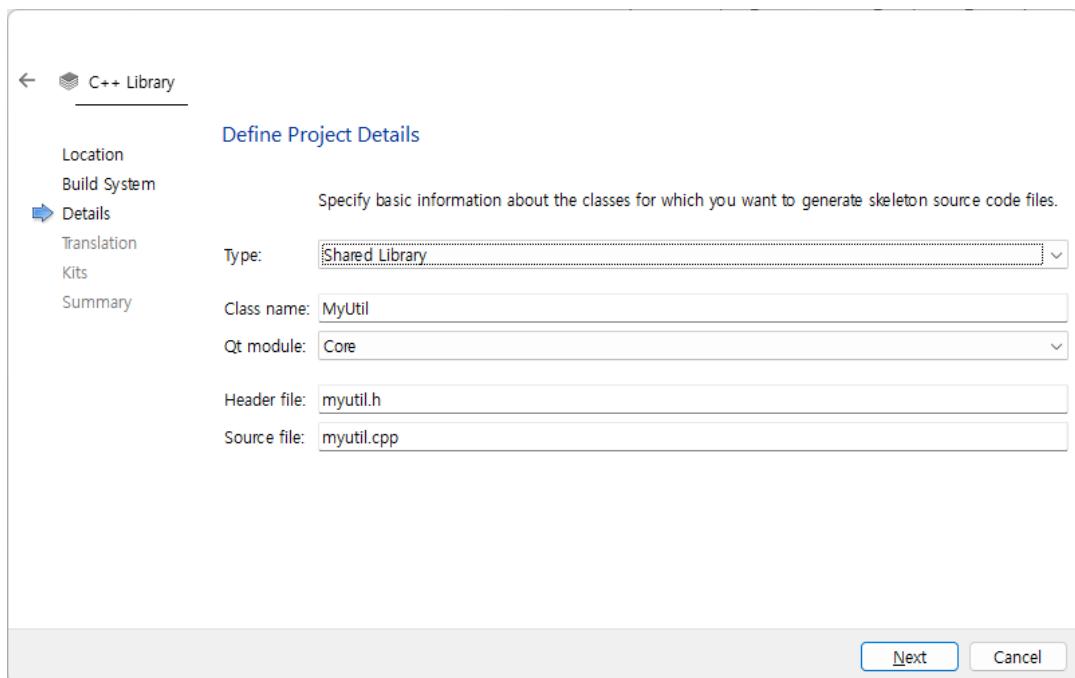


예수님은 당신을 사랑합니다.

빌드 도구로 qmake 를 선택한다.

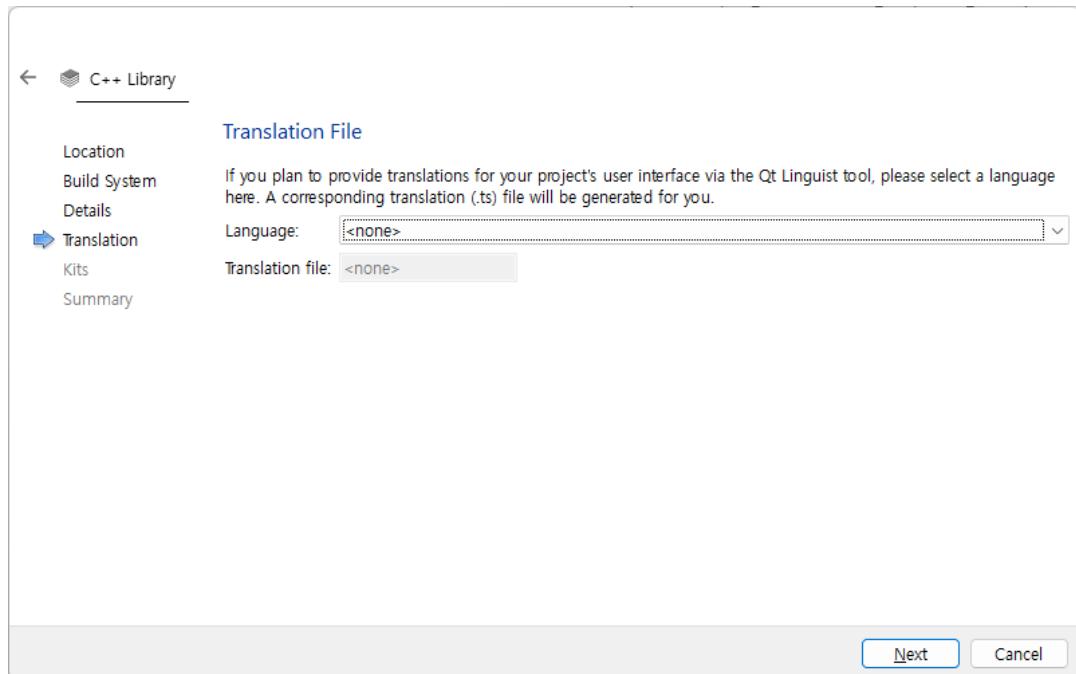


아래와 같이 Type 에서 Shared Library 를 선택한다. Class name 에서는 MyUtil 을 입력 한다. Qt module 은 Core 를 입력한다. Heder file 은 myutil.h 를 입력하고 Source file 항목에는 myutil.cpp 를 입력한다.

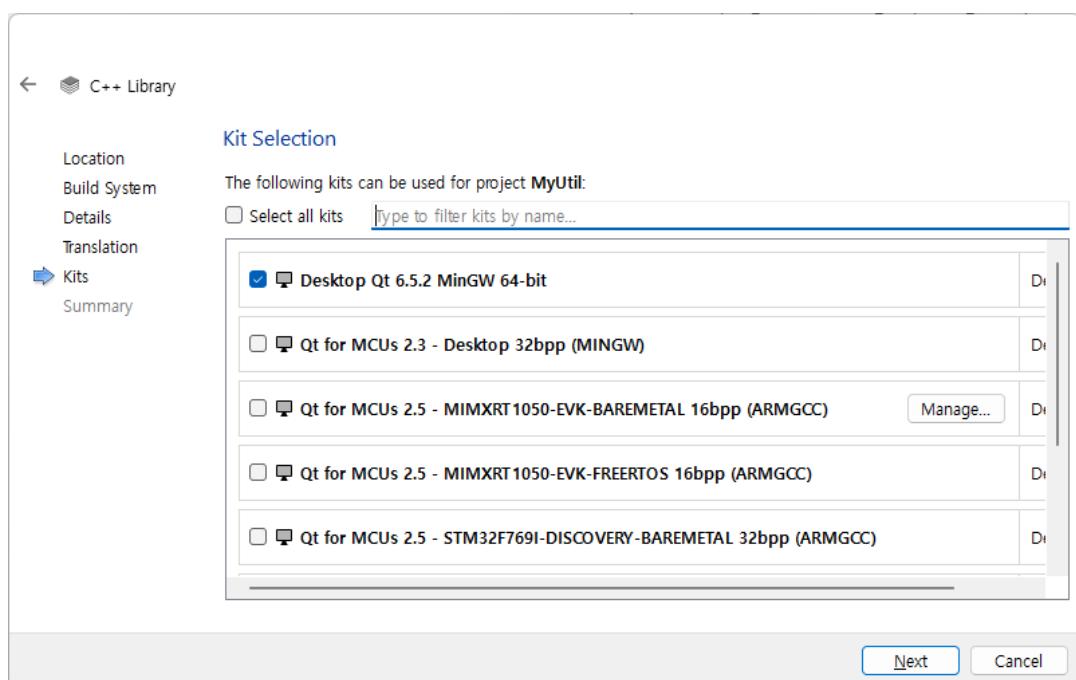


예수님은 당신을 사랑합니다.

다국어를 위한 설정이다. 여기서는 디폴트 상태로 두고 [Next]버튼을 클릭한다.

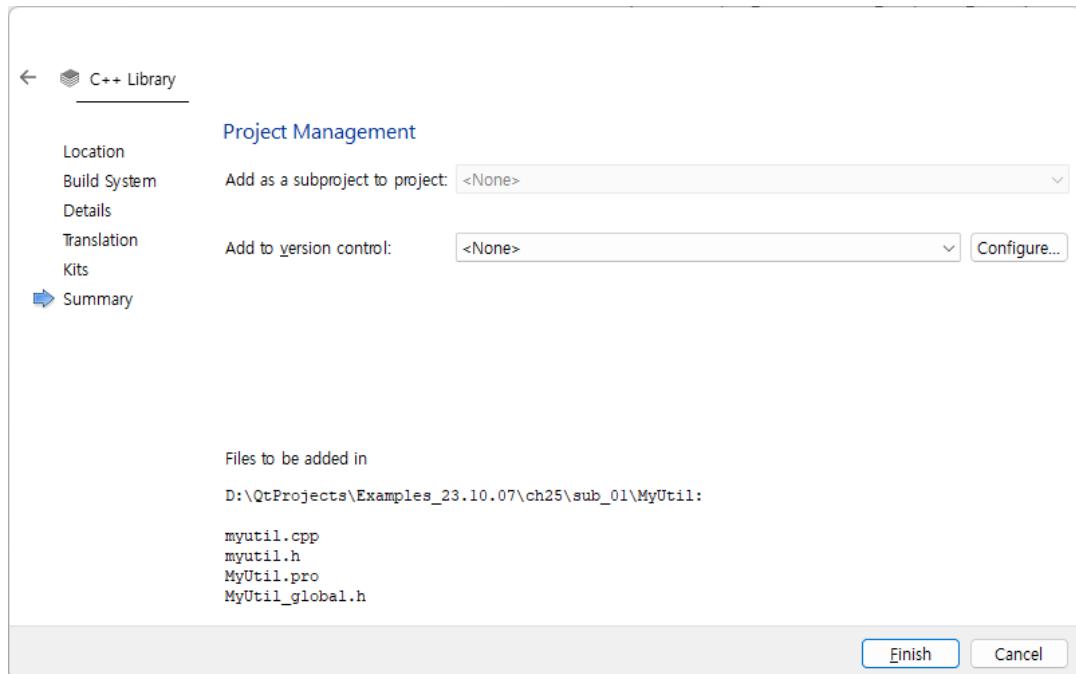


컴파일러 선택ダイアログ에서 MinGW 컴파일러를 선택한다.



예수님은 당신을 사랑합니다.

아래 다이얼로그에서 보는 것과 같이 하단의 [Finish] 버튼을 클릭하면 프로젝트 생성이 완료된다.



다음으로 프로젝트 생성이 완료된 후 프로젝트 파일을 살펴보자.

```
QT -= gui

TEMPLATE = lib
DEFINES += MYUTIL_LIBRARY

CONFIG += c++17

SOURCES += \
    myutil.cpp

HEADERS += \
    MyUtil_global.h \
    myutil.h

# Default rules for deployment.
unix {
    target.path = /usr/lib
}
!isEmpty(target.path): INSTALLS += target
```

예수님은 당신을 사랑합니다.

일반 어플리케이션을 프로젝트와 달리 라이브러리 프로젝트는 TEMPLATE 키워드가 다르다. 일반 어플리케이션은 TEMPLATE 키워드에 app 가 명시되지만 라이브러리는 lib 가 명시된다.

다음으로 myutil.h 을 아래와 같이 소스코드를 작성한다.

```
#ifndef MYUTIL_H
#define MYUTIL_H

#include <QObject>
#include "MyUtil_global.h"

class MYUTIL_EXPORT MyUtil : public QObject
{
    Q_OBJECT
public:
    explicit MyUtil(QObject *parent = nullptr);
    int getSumValue(int a, int b);

};

#endif // MYUTIL_H
```

그리고 myutil.cpp 를 열어서 아래와 같이 소스코드를 작성한다.

```
#include "myutil.h"

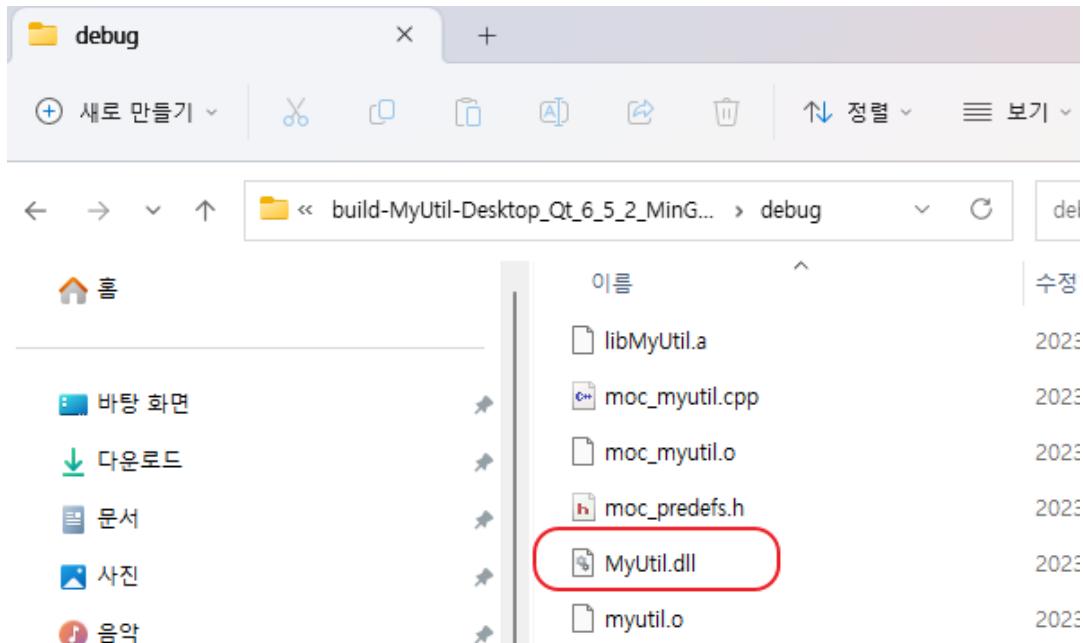
MyUtil::MyUtil(QObject *parent) : QObject(parent)
{
}

int MyUtil::getSumValue(int a, int b)
{
    return a + b;
}
```

MyUtil 클래스는 간단하게 두 개의 값을 함수 인자로 넘겨주면 결과 값으로 두 개의 값을 더한 값을 넘겨준다. 여기까지 작성하였다면 빌드한 다음, 빌드 된 디렉토리에 가

예수님은 당신을 사랑합니다.

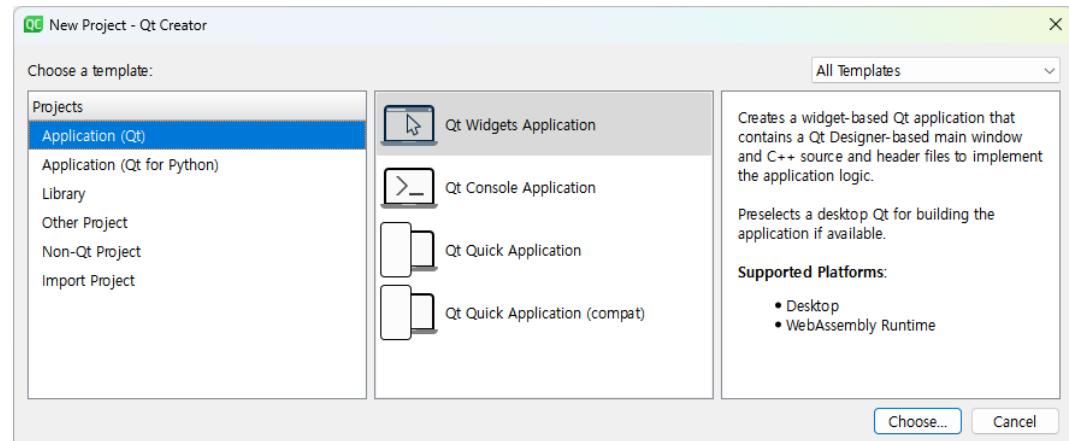
면 라이브러리 파일이 생성된 것을 확인할 수 있다.



위의 그림에서 보는 것과 같이 정상적으로 라이브러리 파일이 생성되었다면 이 라이브러리를 사용하는 예제를 작성해 보도록 하자.

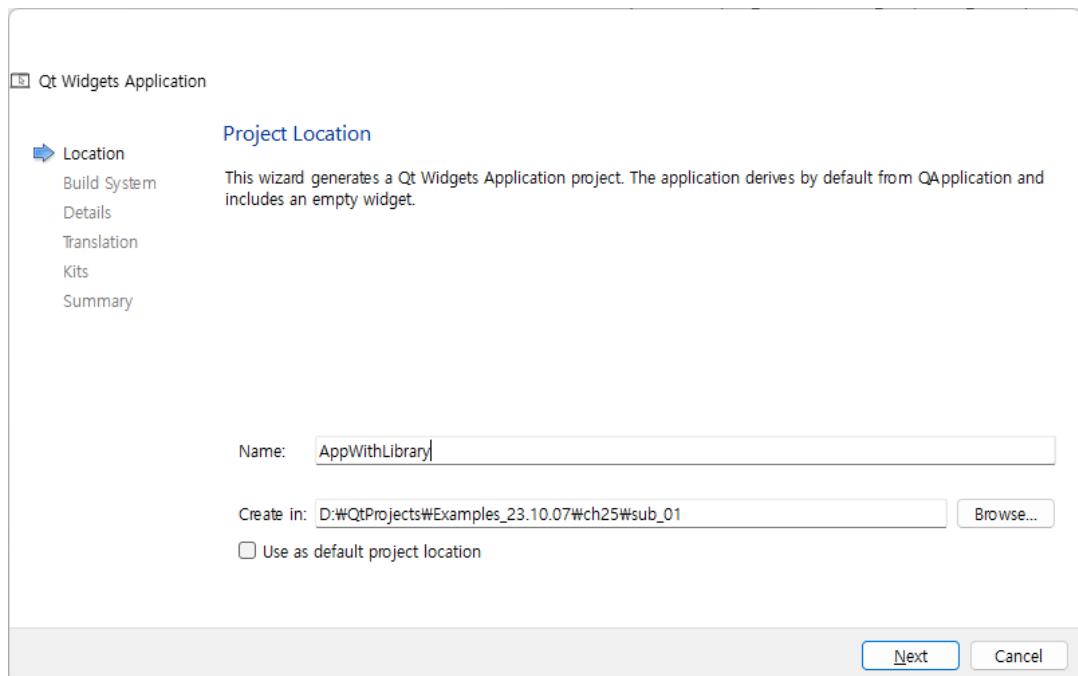
✓ 구현한 라이브러리를 사용하는 예제

프로젝트 생성 시 아래 그림에서 보는 것과 같이 [Qt Widget Application] 을 선택한다.

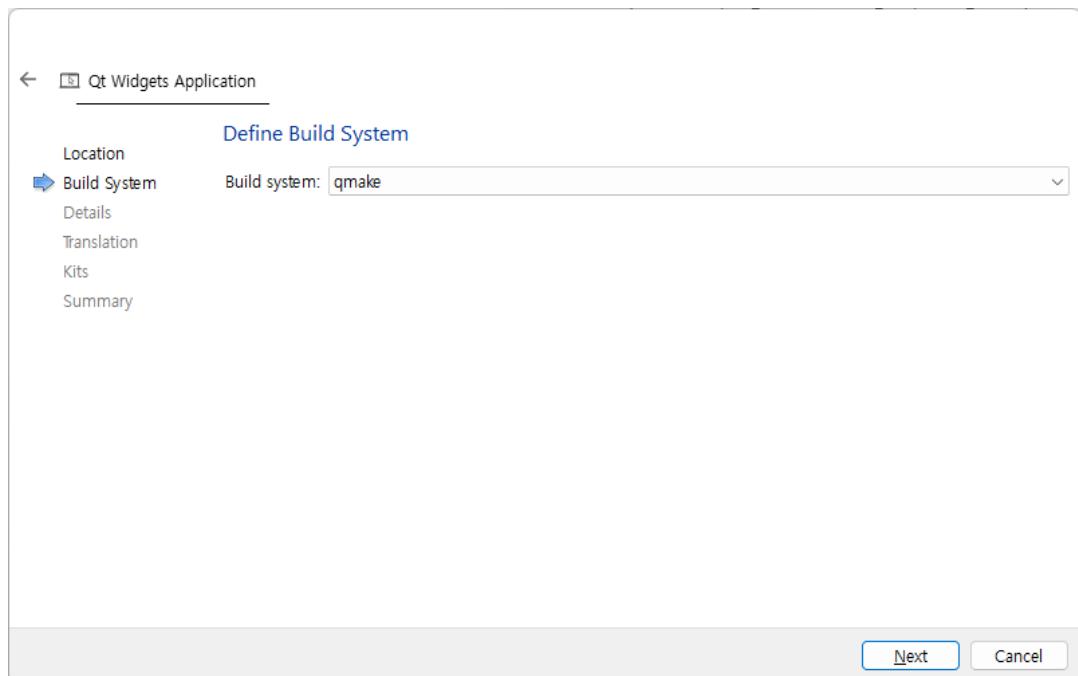


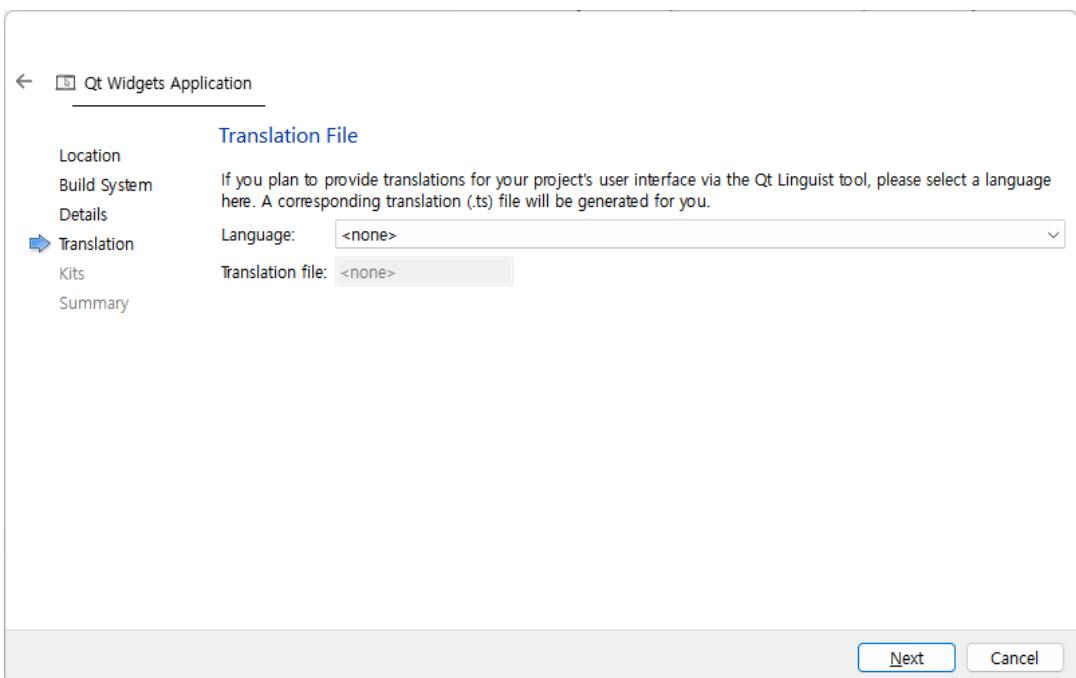
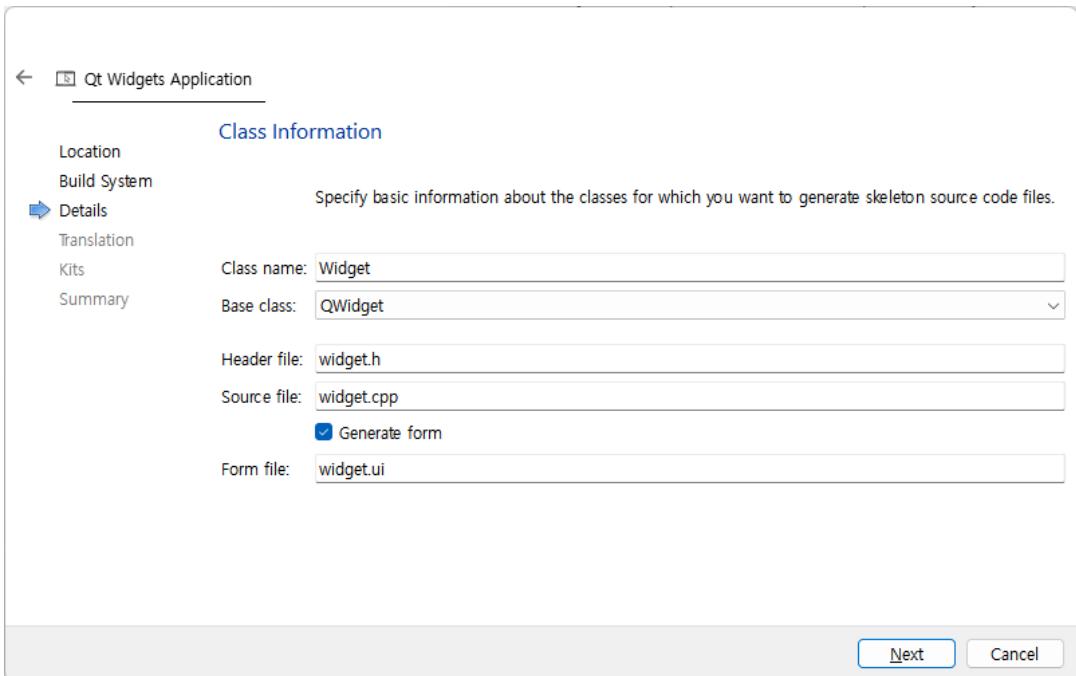
프로젝트 생성 시, 라이브러리파일과 헤더 파일의 위치를 명시해야 하기 때문에 MyUtil 프로젝트가 생성된 동일한 위치에 프로젝트를 생성한다.

예수님은 당신을 사랑합니다.

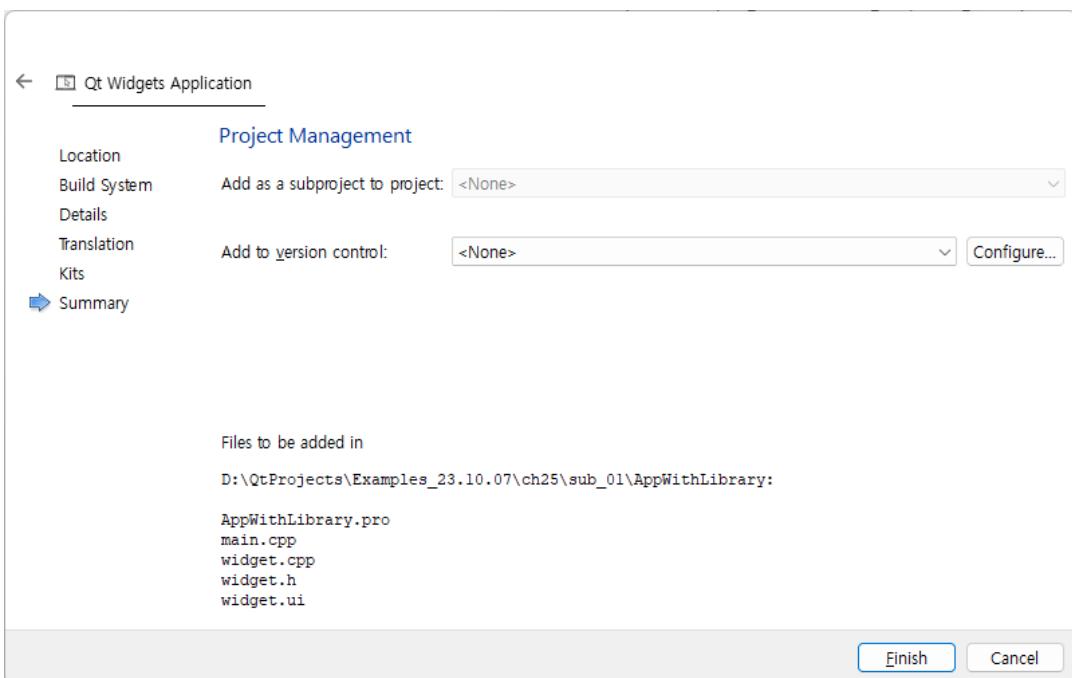
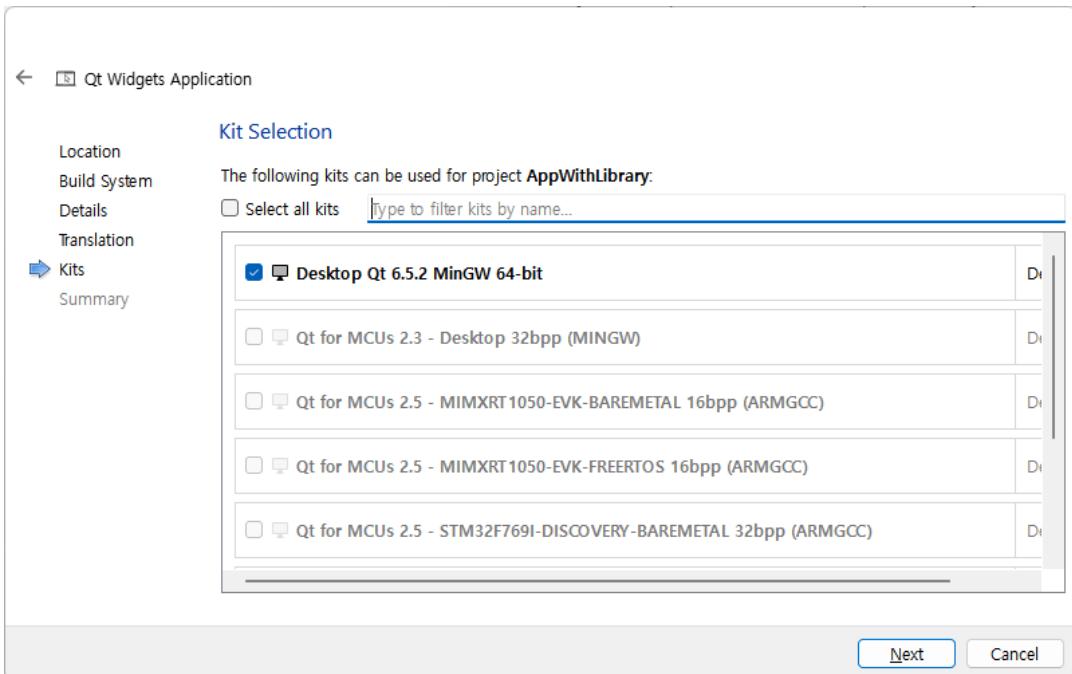


빌드 시스템으로 qmake 를 선택한다.





예수님은 당신을 사랑합니다.



프로젝트 생성이 완료되면 프로젝트파일을 열어 아래와 같이 수정한다.

```
QT      += core gui
```

```

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++17

SOURCES += \
    main.cpp \
    widget.cpp

HEADERS += \
    widget.h

FORMS += \
    widget.ui

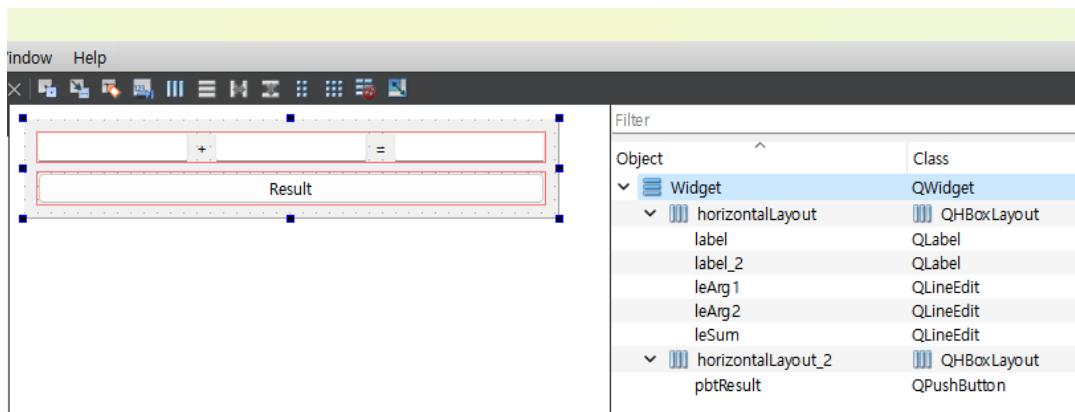
# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

LIBS += -L$$PWD/../../[빌드된 디렉토리 명]/debug/ -lMyUtil

INCLUDEPATH += $$PWD/../../MyUtil
DEPENDPATH += $$PWD/../../MyUtil

```

위와 같이 라이브러리를 추가했다면 아래 그림에서 보는 것과 같이 widget.ui 파일을 열어서 UI를 작성한다.



위와 같이 UI 를 작성한다. 그리고 [Result] 버튼을 클릭하면 라이브러리의 MyUtil 클래스

예수님은 당신을 사랑합니다.

스에서 작성한 getSumValue() 함수를 호출해 결과를 받아 온 다음 leSum 오브젝트에 결과를 표시하도록 소스코드를 작성해 보도록 하자.

먼저 widget.h 소스코드 파일을 열어서 아래와 같이 소스코드를 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include "myutil.h"

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
    MyUtil *m_myUtil;

private slots:
    void slot_pbtResult();

};

#endif // WIDGET_H
```

다음으로 widget.cpp 소스코드 파일을 열어서 아래와 같이 작성해 보도록 하자.

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

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

    m_myUtil = new MyUtil();

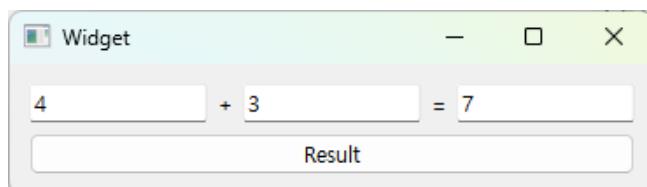
    connect(ui->pbtResult, &QPushButton::clicked,
            this,           &Widget::slot_pbtResult);
}

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

void Widget::slot_pbtResult()
{
    qint32 arg1 = ui->leArg1->text().toInt();
    qint32 arg2 = ui->leArg2->text().toInt();

    qint32 sumValue = m_myUtil->getSumValue(arg1, arg2);
    ui->leSum->setText(QString("%1").arg(sumValue));
}
```

위와 같이 작성했다면 결과를 확인해 보도록 하자.

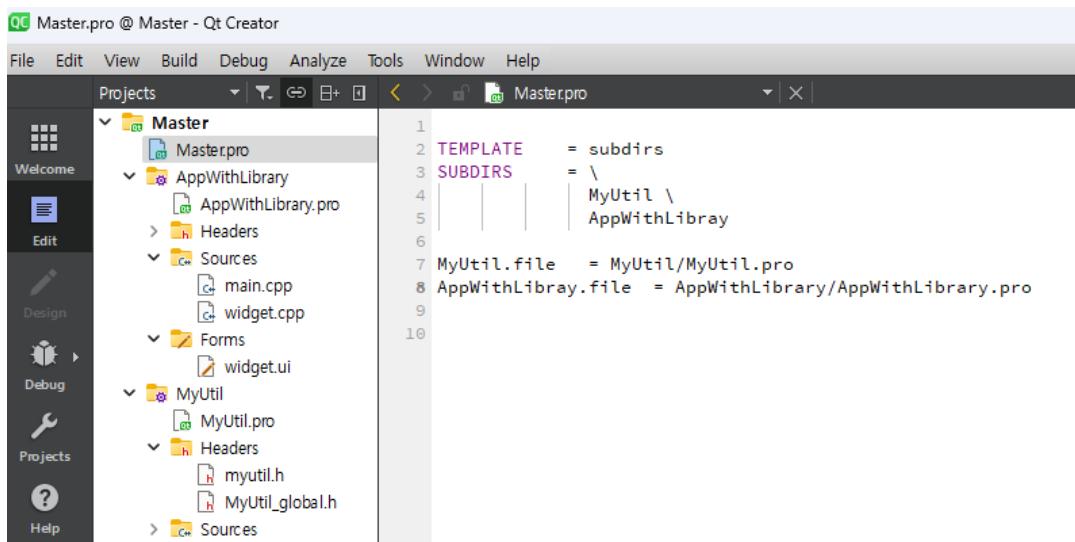


예제는 sub_01 디렉토리를 참조하면 된다.

24.2. 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 정확다면 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.



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

```

예수님은 당신을 사랑합니다.

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.

이 예제 소스코드는 sub_02 디렉토리를 참조하면 된다.

25. D-Pointer

C/C++ 에서는 버전 호환성과 소스코드의 비밀을 유지하기 위한 목적으로 Opaque Pointer(또는 Opaque Type)를 사용한다. Qt에서도 Opaque Pointer를 사용할 수 있다 하지만 좀더 Qt에 맞게끔 Opaque Pointer를 개선한 것이 D-Pointer 이다.

그리고 Qt에서 제공하는 Internal 모듈(라이브러리)의 헤더 코드를 보면 아래와 같이 D-Pointer 의 형태로 구현된 것을 확인할 수 있다.

```
#ifndef QCOREAPPLICATION_H
#define QCOREAPPLICATION_H

#include <QtCore/qglobal.h>
...
class QCoreApplicationPrivate;
...
class Q_CORE_EXPORT QCoreApplication
{

```

위의 예제 소스코드는 Qt에서 제공하는 QCoreApplication 클래스의 헤더파일이다. 이 클래스를 보면 QCoreApplicationPrivate 이라는 Private 클래스를 추가로 선언되어 있다. 즉 클래스 이름 뒤에 Private 이라는 단어를 붙여서 사용하는 것을 알 수 있다. 이러한 형태가 D-Pointer라고 한다.

이번 장에서는 D-Pointer의 장점 및 사용 방법을 예제 소스코드를 통해서 알아보도록 하자.

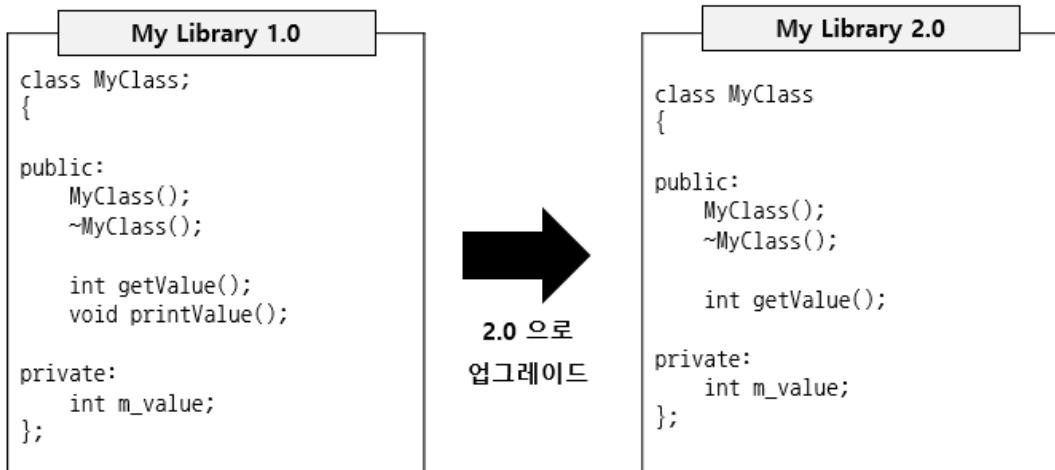
✓ D-Pointer 의 장점 및 특징

D-Pointer를 쓰는 이유는 크게 두가지로 살펴볼 수 있다. 첫 번째는 라이브러리의 버전 별 호환성을 유지 하는 목적으로 사용한다.

라이브러리 개발자는 각 라이브러리 버전의 호환성을 유지해야한다. 라이브러리 버전이 올라감에 따라 라이브러리상에서 수정되는 부분으로 인하여 라이브러리 버전 별 호환성이 잘 될 수 있을지 고민할 것이다.

예를 들어 라이브러리 상에서 특정 클래스의 멤버 함수가 추가되거나 없어 질 수 있다.

추가되는 것은 크게 문제가 되지 않으나 특정 멤버 함수가 삭제 되는 것은 버전 호환성에 큰 문제가 된다.



위의 예제에서 보는 것과 같이 My Library 1.0 에서 2.0 버전으로 업그레이드 되면서 printValue() 함수가 삭제 된 것을 확인 할 수 있다.

만약 이 라이브러리를 사용하는 End User 개발자가 2.0 버전으로 업그레이드 된 라이브러리를 사용한다고 가정해 보자. 그런데 printValue() 함수를 사용한다면 어떻게 될 것인가?. printValue() 함수가 없다고 에러가 날 것이다.

따라서 이러한 문제점을 해결하는데 도움이 될 수 있는 방법으로 Opaque Pointer를 사용한다.

D-Pointer를 사용한다고 해서 버전 별 호환성 문제를 완벽히 해결 하는 것은 불가능 하지만 D-Pointer를 사용함으로써 버전의 호환성을 유지할 수 있다.

다시 D-Pointer 로 돌아와 살펴보자.

Qt에서 제공하는 클래스들의 소스코드를 살펴보면 D-Pointer를 활용한 것을 볼 수 있다. QApplication 클래스를 보면, 이 클래스 이름 뒤에 Private 이라는 단어가 붙어서 QApplicationPrivate 이라는 추가 클래스를 제공하는 형태가 바로 D-Pointer 클래스이다.

라이브러리를 구현할 때 버전이 업데이트 되어도 주로 변화가 없을 것 같은 내용은 주로 QApplication 에 구현을 하고 변화가 많이 주로 있을 것 같은 내용은 주로 Private 클래스에 구현 함으로써 버전 별 호환성을 유지 할 수 있다.

예를 들어 위의 예제 소스코드에서 printValue()를 삭제할 때 MyClass 에 동일한 이름의 멤버 함수를 만들고 이 함수를 사용할 때, 이 함수는 없어졌다는 정보를 리턴 한다

예수님은 당신을 사랑합니다.

면 라이브러리를 사용하는 개발자는 삭제된 멤버함수를 사용함으로써 발생한 문제를 좀더 쉽게 해결 할 수 있을 것이다.

두 번째로 D-Pointer를 사용하는 이유는 소스코드의 비밀을 유지하기 위한 목적으로 사용할 수 있다.

예를 들어 라이브러리를 배포할 때 헤더 파일을 함께 배포해야 한다. 하지만 Private 클래스의 헤더 파일은 공개하지 않아도 된다.

예를 들어 QCoreApplication 클래스를 사용하는 개발자는 QCoreApplication.h 헤더 파일과 더불어 QCoreApplicationPrivate.h 중에서 QCoreApplicationPrivate.h는 배포하지 않아도 된다. (D-Pointer 에서는 Private 클래스 이름은 클래스 이름 뒤에 _p 가 붙는다.) 따라서 소스코드의 비밀을 유지하고 싶은 소스코드를 QCoreApplicationPrivate 클래스에서 구현하면 된다.

그렇게 되면 최종 라이브러리를 사용하는 개발자에게 QCoreApplicationPrivate.h 헤더 파일은 배포하지 않아도 라이브러리를 사용할 수 있기 때문이다.

✓ D-Pointer 의 규칙

D-Pointer를 사용하기 위해서는 규칙이 있다. 예를 들어 라이브러리를 구현 할 때 MyClass 라는 클래스에 D-Pointer를 사용하는 예를 들어보겠다.

클래스명: **MyClass**
(**myclass.h**)

클래스명: **MyClass**Private****
(**myclass_p.h**)

위의 예에서 보는 것과 같이 MyClass 클래스를 만들고 Private 클래스는 MyClass 이름 뒤에 Private 이라는 단어를 붙여서 구현해야 한다. 그리고 클래스이름은 클래스 이름 뒤에 "_p" 를 붙여야 한다.

따라서 MyClass 의 Private 클래스 이름은 **MyClass**Private**** 이라는 이름으로 헤더 파일을 생성해야 한다.

그리고 myclass.h 가 클래스 헤더 파일 이름이라면 Private 클래스의 헤더 파일의 이름은 **myclass_p.h** 가 되어야 한다.

예수님은 당신을 사랑합니다.

- ✓ D-Pointer에서 클래스 간 멤버 함수 호출

MyClass와 MyClassPrivate 클래스 간에 멤버 함수를 호출해야 하는 경우가 발생한다.

예를 들어서 MyClass에서 MyClassPrivate 클래스의 함수를 호출할 때 Q_D()를 사용한다. 반대로 MyClassPrivate에서 MyClass 클래스의 함수를 이용해야 하는 경우 Q_Q()를 사용해야 한다.

아래 예제 소스코드는 MyClass에서 MyClassPrivate 클래스의 printValue() 멤버 함수를 호출하기 위해서 Q_D() 함수를 사용한 예제이다.

```
#include "myclass.h"
#include "myclass_p.h"

MyClass::MyClass()
{
    d_ptr = new MyClassPrivate(this);
    Q_D(MyClass);

    m_value = 100;
    d->setValue(200);
}

MyClass::~MyClass()
{
}

int MyClass::getValue()
{
    return m_value;
}

void MyClass::printValue()
{
    Q_D(MyClass);
    d->printValue();
}
```

다음 예제는 MyClassPrivate 클래스에서 MyClass 클래스의 getValue() 멤버함수를 호출하기 위해서 Q_Q() 함수를 사용한 예이다.

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

```
MyClassPrivate::MyClassPrivate(MyClass *q)
{
    q_ptr = q;
}

MyClassPrivate::~MyClassPrivate()
{
}

void MyClassPrivate::setValue(int val)
{
    m_value = val;
}

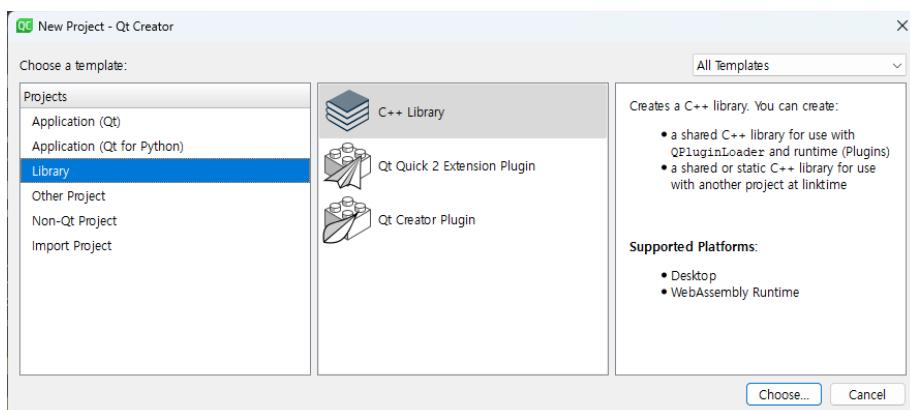
void MyClassPrivate::printValue()
{
    Q_Q(MyClass);

    qDebug() << " MyClass 의 m_value : " << q->getValue();
    qDebug() << " MyClassPrivate 의 m_value : " << m_value;
}
```

위의 예제에서 보는 것과 MyClass 와 MyClassPrivate 간에 멤버함수를 호출하기 위해서 Q_D() 와 Q_Q() 함수를 사용해야 한다.

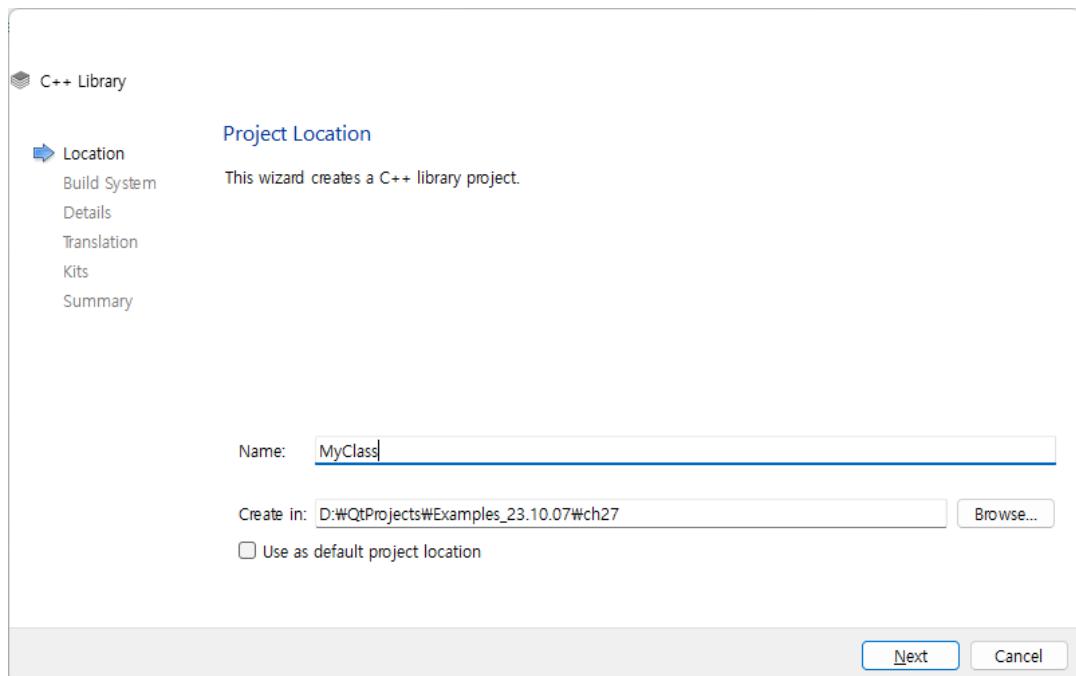
✓ D-Pointer를 이용한 라이브러리 구현

이번에는 D-Pointer를 이용하는 라이브러리를 구현해 보도록 하자. 프로젝트 생성 시 아래 그림에서 보는 것과 같이 프로젝트를 생성한다.

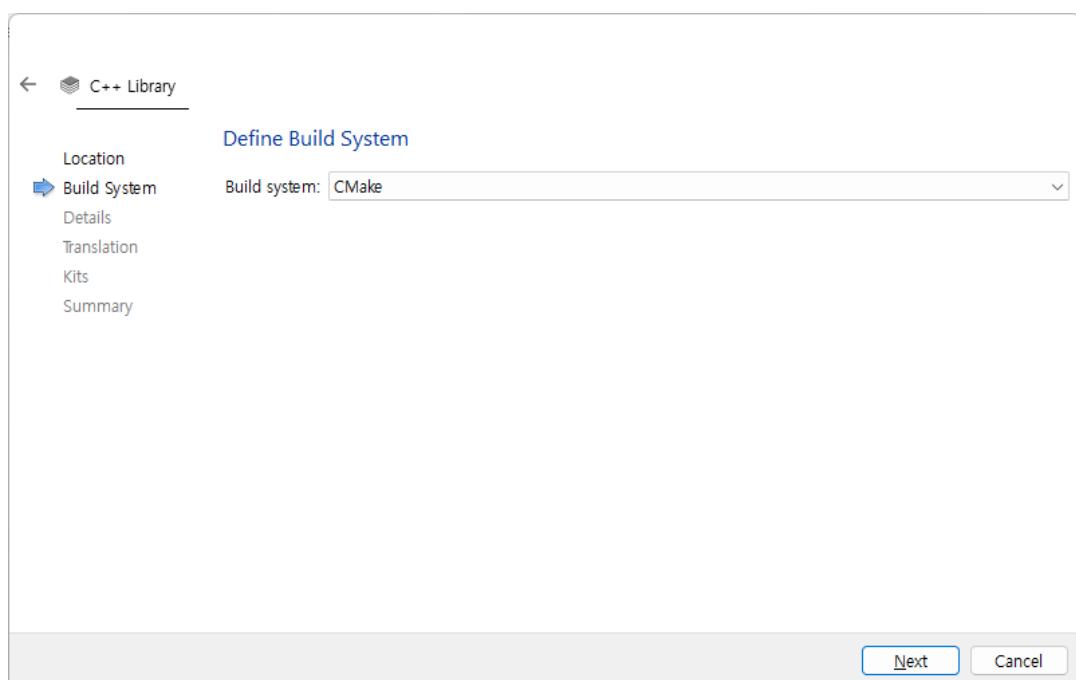


예수님은 당신을 사랑합니다.

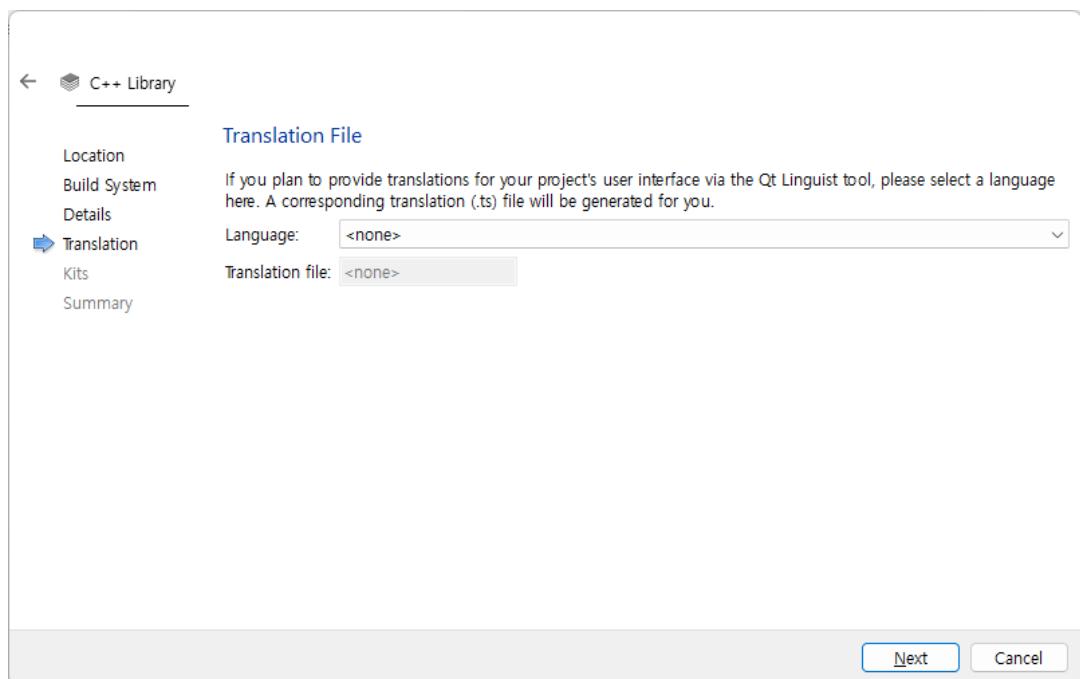
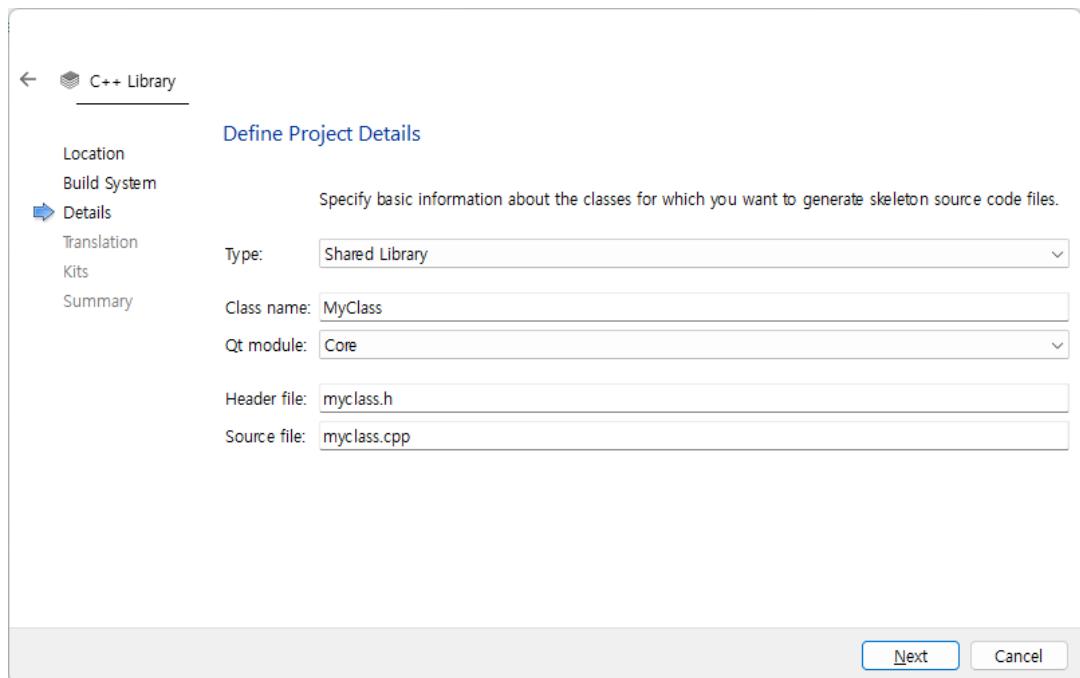
프로젝트의 이름은 "MyClass"를 입력해 프로젝트를 생성한다.

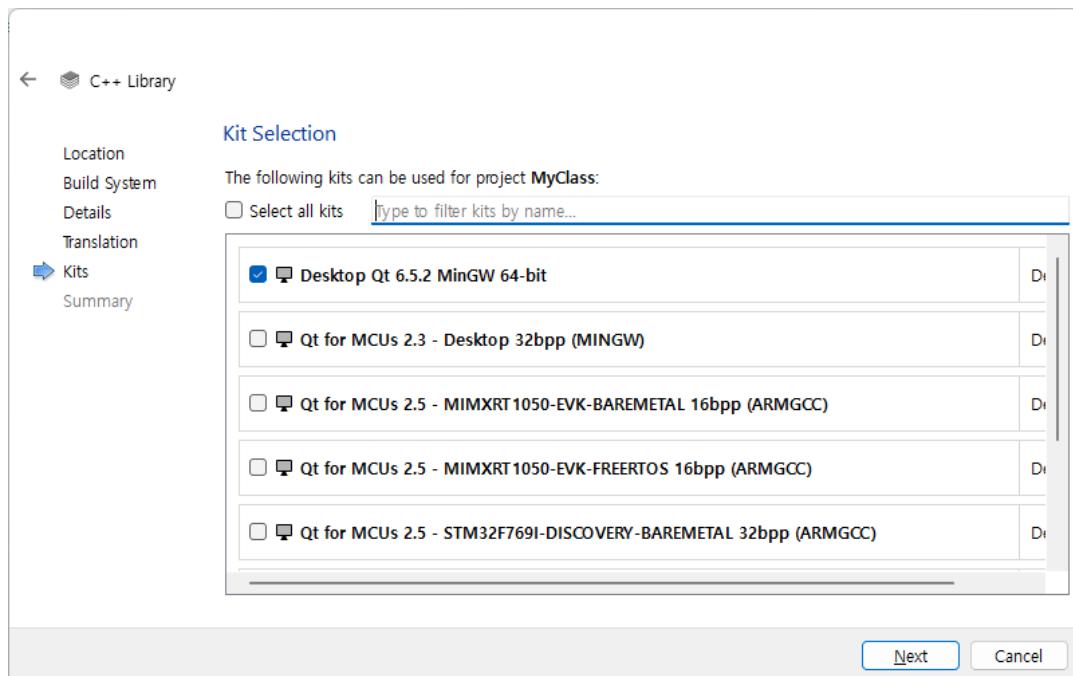


프로젝트 생성 시, 빌드 도구로 CMake를 선택한다.

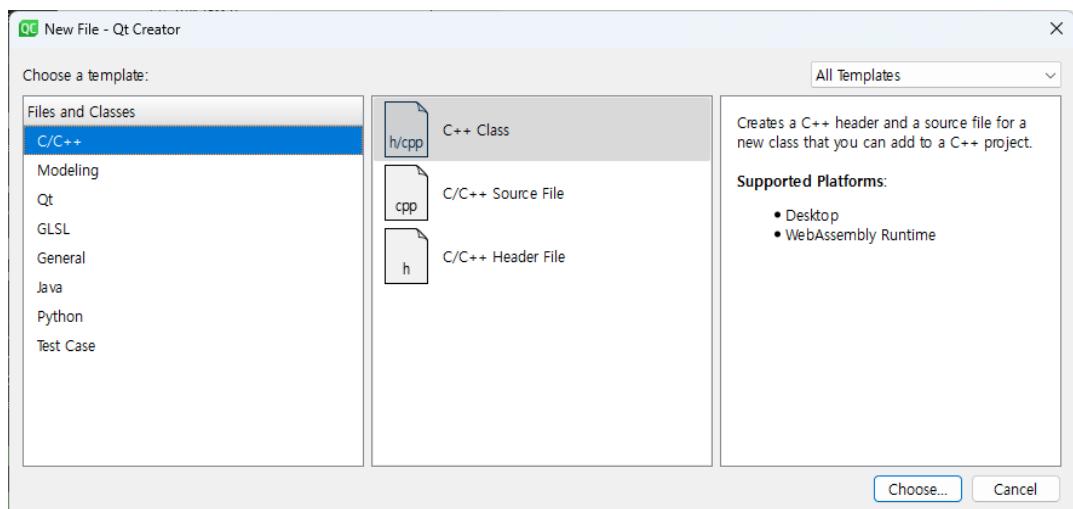


프로젝트의 타입으로 Shared Library를 선택하고 아래와 같이 입력한다.

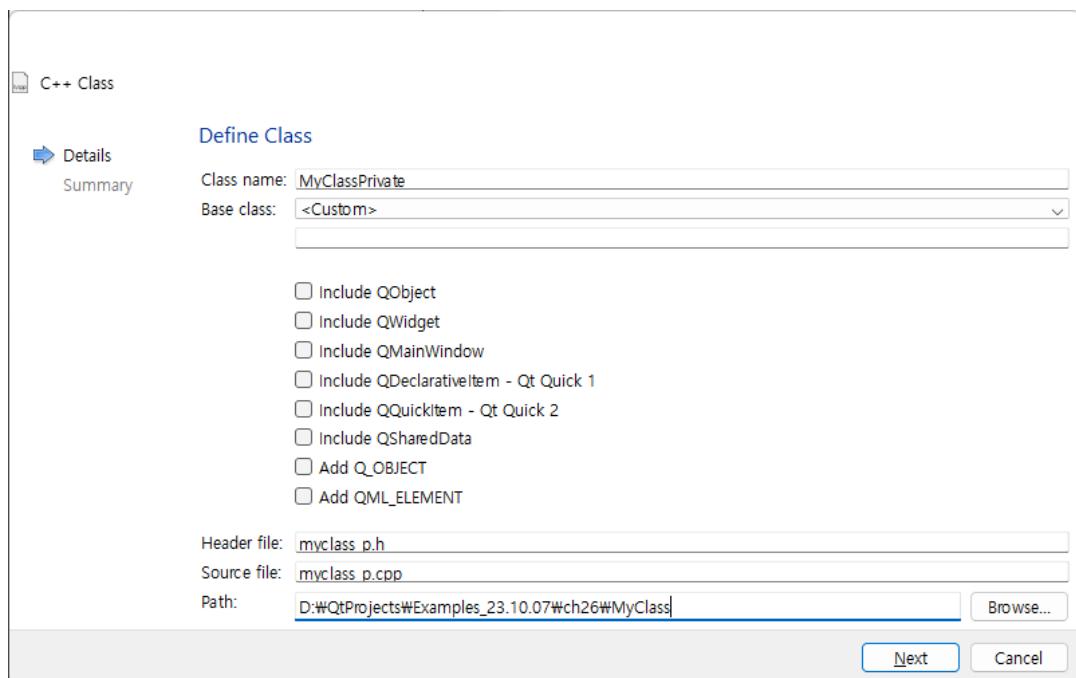




위와 같이 프로젝트를 생성하고 MyClassPrivate 클래스를 추가한다.



추가할 클래스의 클래스 명은 MyClassPrivate 을 입력한다. 헤더 파일명은 "myclass_p.h" 를 입력하고 소스코드 파일 명은 "myclass_p.cpp"를 입력한다.



위와 같이 입력하였다면 MyClass 와 MyClassPrivate 클래스를 작성해 보도록 하자. 첫 번째로 MyClassPrivate 를 먼저 작성해보도록 하자. Myclass_p.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef MYCLASSPRIVATE_H
#define MYCLASSPRIVATE_H

#include "myclass.h"
#include <QDebug>

class MyClassPrivate
{
    Q_DECLARE_PUBLIC(MyClass)

public:
    MyClassPrivate(MyClass *q);
    ~MyClassPrivate();

    void setValue(int);
    void printValue();

private:
    int      m_value;
    MyClass *q_ptr;
```

예수님은 당신을 사랑합니다.

```
};

#endif // MYCLASSPRIVATE_H
```

위에서 보는 것과 같이 setValue(int) 함수와 printValue() 함수를 MyClass 에서 호출 할 것이다. 그리고 printValue() 함수에서 다시 MyClass 의 getValue() 함수를 호출할 것이다. 아래는 MyClassPrivate.cpp 소스코드 파일을 아래와 같이 작성한다.

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

MyClassPrivate::MyClassPrivate(MyClass *q)
{
    q_ptr = q;
}

MyClassPrivate::~MyClassPrivate()
{
}

void MyClassPrivate::setValue(int val)
{
    m_value = val;
}

void MyClassPrivate::printValue()
{
    Q_Q(MyClass);

    qDebug() << " MyClass 의 m_value : " << q->getValue();
    qDebug() << " MyClassPrivate 의 m_value : " << m_value;
}
```

이 번에는 MyClass를 작성해 보도록 하자. MyClass.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef MYCLASS_H
#define MYCLASS_H

#include <QtGlobal>

#define MYCLASS_EXPORT Q_DECL_EXPORT
```

```
class MyClassPrivate;
class MYCLASS_EXPORT MyClass
{
    Q_DECLARE_PRIVATE(MyClass)

public:
    MyClass();
    ~MyClass();

    int getValue();
    void printValue();

private:
    int m_value;
    MyClassPrivate *d_ptr;
};

#endif // MYCLASS_H
```

다음으로 myclass.cpp 소스 코드 파일을 작성한다.

```
#include "myclass.h"
#include "myclass_p.h"

MyClass::MyClass()
{
    d_ptr = new MyClassPrivate(this);
    Q_D(MyClass);

    m_value = 100;
    d->setValue(200);
}

MyClass::~MyClass()
{
}

int MyClass::getValue()
{
    return m_value;
}

void MyClass::printValue()
```

```
{  
    Q_D(MyClass);  
    d->printValue();  
}
```

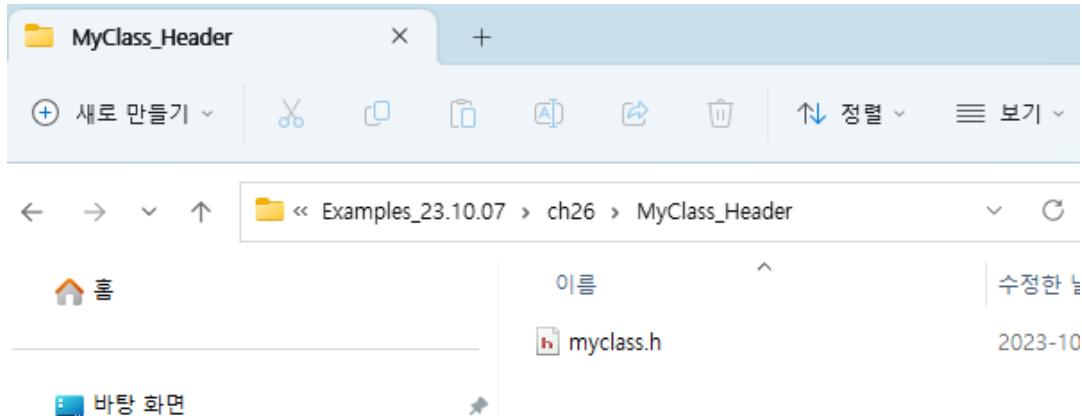
생성자 함수에서 MyClassPrivate 의 setValue() 멤버함수를 호출한다. 이 함수를 호출하기 위해서는 Q_D() 함수를 먼저 사용해야 한다. 즉 MyClassPrivate() 클래스의 멤버 함수를 호출하는 함수에서는 모두 Q_D() 함수를 호출해야 한다.

그리고 MyClassPrivate 에서 MyClass 의 멤버함수를 호출하기 위해서는 먼저 Q_Q() 함수를 호출해야 한다.

여기까지 작성했다면 이번에는 MyClass 프로젝트를 빌드해 보자. 에러가 없이 정상적으로 빌드가 되었다면 libMyClass.dll (리눅스인 경우 libMyClass.so가 생성됨) 가 생성되었을 것이다.

이번에는 MyClass를 사용하는 예제를 작성해 보도록 하자. 예제를 먼저 작성하기 전에 프로젝트가 위치한 동일한 디렉토리 위치에 MyClass_Header 디렉토리를 만들고 여기에 myclass.h 헤더 파일을 복사한다.

그 이유는 정말로 myclass_p.h를 배포하지 않아도 되는 것을 확인하기 위함이다.

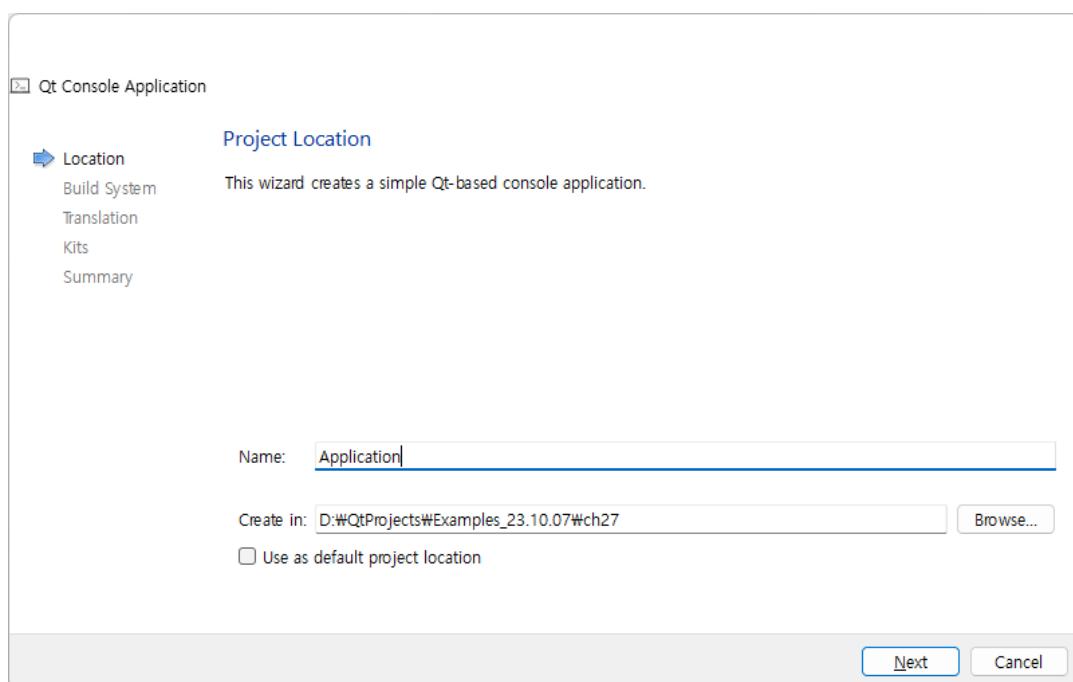
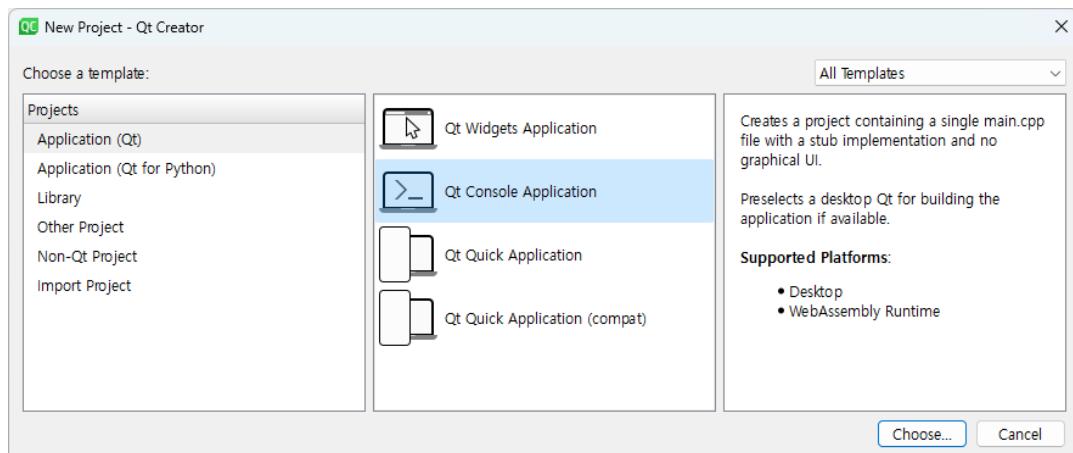


위의 그림에서 보는 것과 같이 myclass.h 헤더 파일을 MyClass_Header 디렉토리에 복사한다.

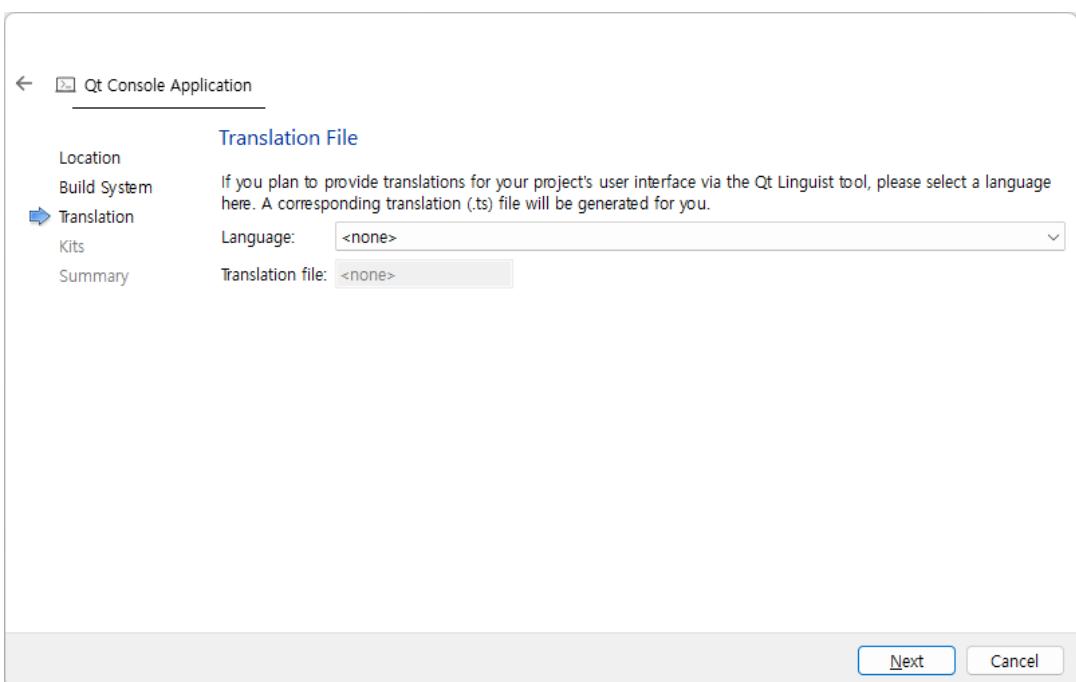
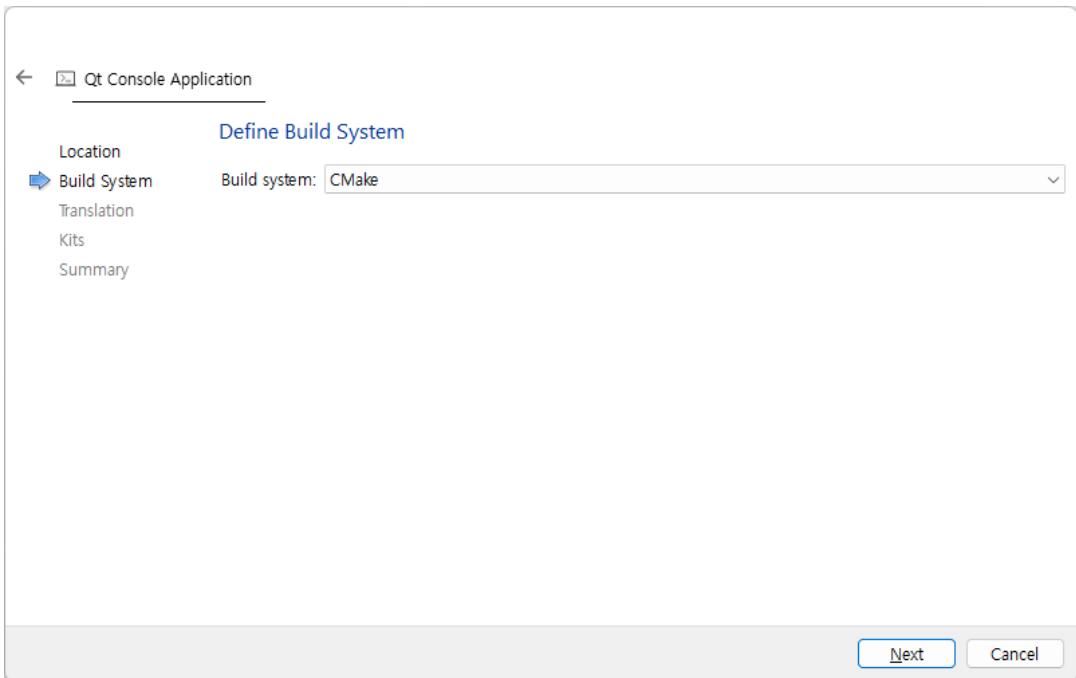
그리고 이번에는 프로젝트 이름을 Application 이라는 이름으로 프로젝트를 생성해 보도록 하자.

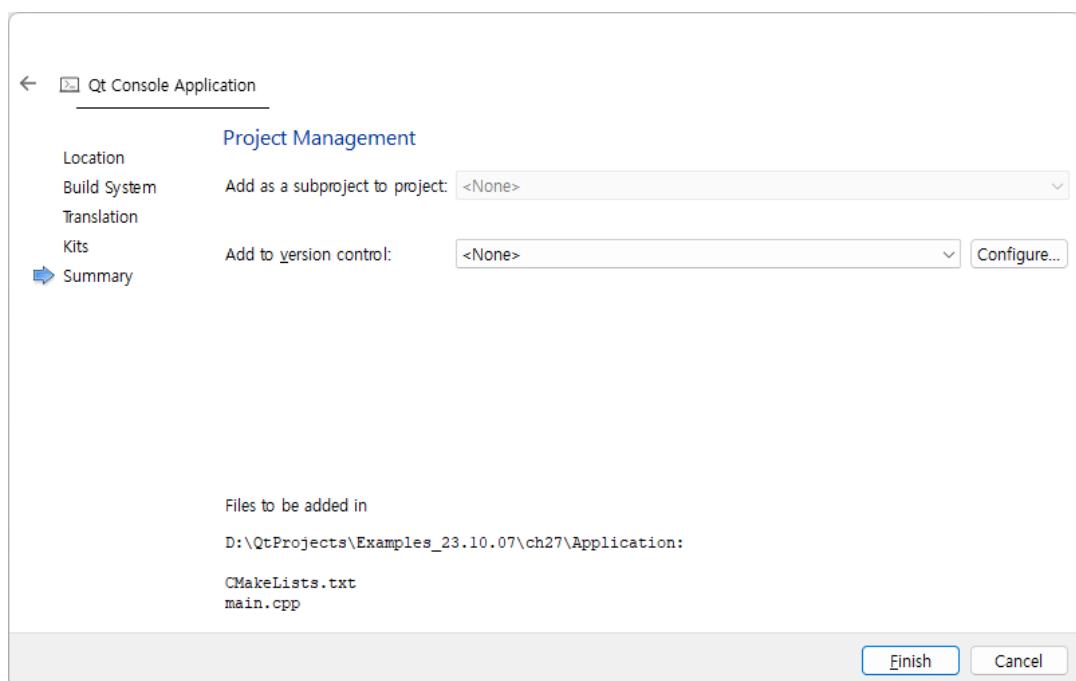
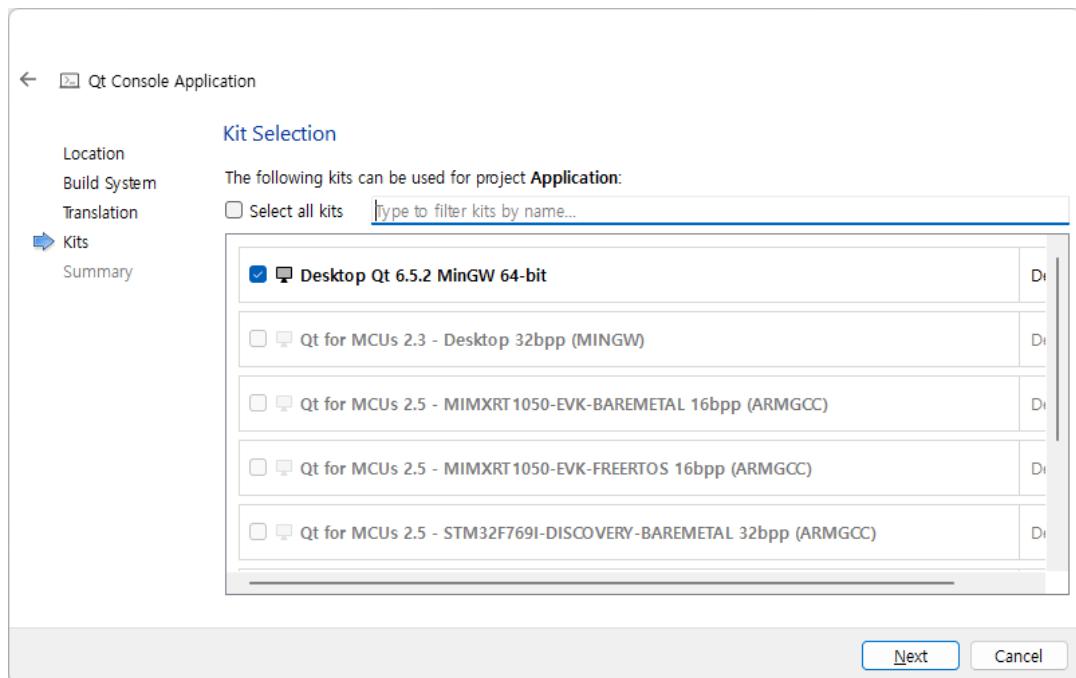
프로젝트 생성 시 아래 그림에서 보는 것과 같이 Qt Console Application을 선택한다.

예수님은 당신을 사랑합니다.



프로젝트 생성 시 빌드 도구로 CMake를 선택한다.





위와 같이 프로젝트를 생성하였다면 아래 그림에서 보는 것과 같이 CmakeList.txt 파일을 열어서 아래와 같이 MyClass 라이브러리를 사용할 수 있도록 라이브러리 파일의 위치와 헤더 파일의 위치를 명시해 준다. 그리고 target_link_libraries()에 사용할 라이브러리 명을 추가해 준다.

```
cmake_minimum_required(VERSION 3.14)

project(Application LANGUAGES CXX)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

include_directories(../MyClass_Header)
link_directories(../build-MyClass-Desktop_Qt_6_5_2_MinGW_64_bit-Debug)

add_executable(Application
    main.cpp
)

find_package(Qt NAMES Qt6 Qt5 REQUIRED COMPONENTS Core)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Core)

target_link_libraries(Application MyClass)
target_link_libraries(Application Qt${QT_VERSION_MAJOR}::Core)

include(GNUInstallDirs)
install(TARGETS Application
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

다음으로 main.cpp에서 MyClass 라이브러리를 사용하기 위해서 아래와 같이 소스코드를 작성해 보도록 하자.

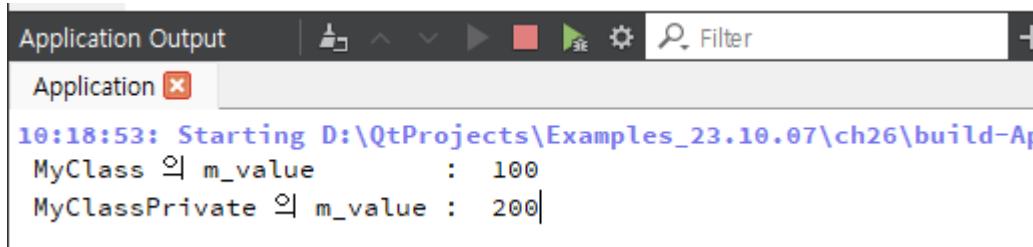
```
#include <QCoreApplication>
#include "myclass.h"

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

    MyClass myclass;
    myclass.printValue();
```

```
    return a.exec();
}
```

위의 소스코드는 MyClass 의 printValue() 함수를 호출한다. 그리고 이 함수는 다시 MyClassPrivate 의 printValue() 함수를 호출한다. 따라서 아래 그림에서 보는 것과 같이 출력한다.

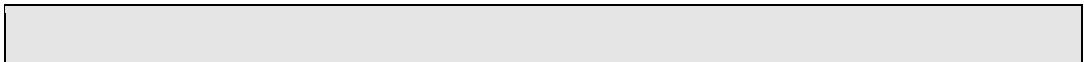


The screenshot shows the 'Application Output' window in Qt Creator. The title bar says 'Application'. The log output shows:

```
10:18:53: Starting D:\QtProjects\Examples_23.10.07\ch26\build-App
MyClass 의 m_value : 100
MyClassPrivate 의 m_value : 200
```

이번 장에서 다룬 라이브러리 프로젝트는 MyClass 와 MyClass_Header 디렉토리를 참조하면 된다. 그리고 라이브러리를 사용하는 예제는 Application 디렉토리의 프로젝트를 참조하면 된다.

예수님은 당신을 사랑합니다.



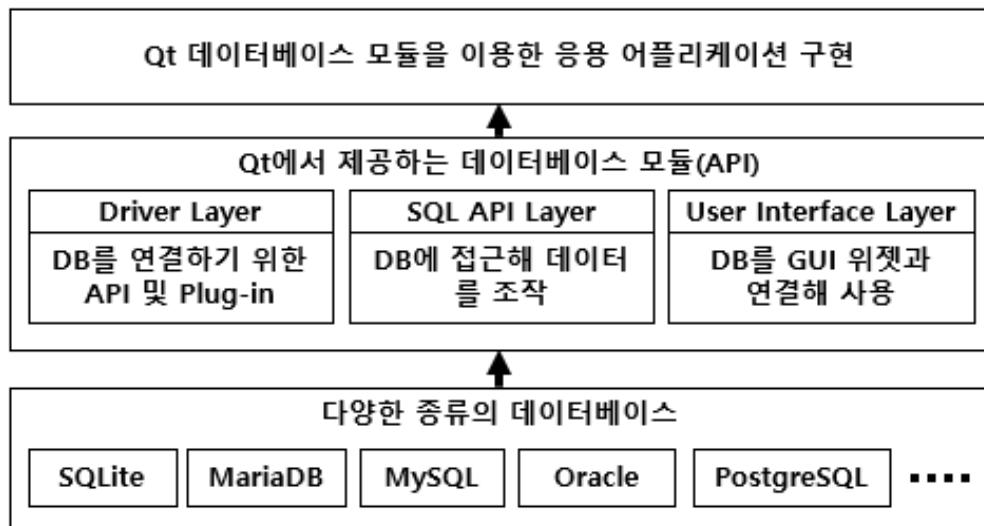
26. Database Programming

Qt에서 제공하는 데이터베이스 모듈(API)은 다른 개발 프레임워크에 비해 사용하기 쉽다. Qt에서 제공하는 데이터베이스 모듈을 이용하면 통일된 API를 이용해 다양한 데이터베이스에 접근할 수 있다.

예를 들어 MariaDB 데이터베이스에 저장된 데이터를 SQL문을 이용해 QUERY 하기 위해서 Qt에서 제공하는 QSqlQuery 클래스를 사용할 수 있다.

그리고 SQLite 데이터베이스에 저장된 데이터를 SQL문을 이용해 QUERY 하기 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.



위의 그림에서 보는 것과 같이 SQLite를 사용하든지 MariaDB를 사용하든지 Qt에서 제공하는 Qt SQL 모듈을 사용하면 각 데이터베이스를 핸들링 할 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

이 Layer는 특정 데이터베이스를 연결하기 위한 Low Level Bridge 역할을 담당한다. 즉 데이터베이스와 연결을 위한 드라이버 역할을 담당한다.

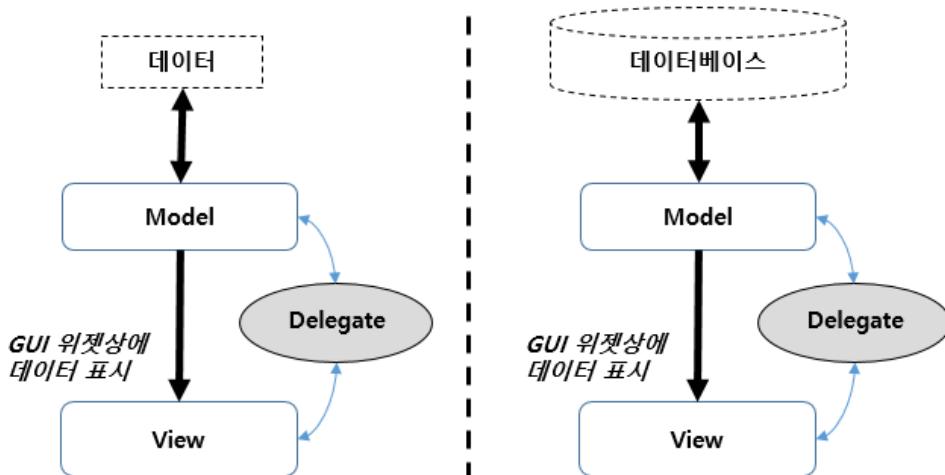
② SQL API Layer

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

예수님은 당신을 사랑합니다.

Model 클래스들을 이용해 View 위젯과 연결할 수 있다. Qt는 주로 사용하는 Model 클래스로써 QSqlQueryModel, QSqlTableModel, QSqlRelationalTableModel 클래스를 제공한다.

Qt에서 제공하는 데이터베이스 모듈을 사용하기 위해서는 프로젝트 파일에 다음과 같이 "sql" 키워드를 사용해야 한다.

```
Qt += sql
```

만약 CMake 를 사용한다면 아래와 같이 명시하면 된다.

```
find_package(Qt6 REQUIRED COMPONENTS Sql)
target_link_libraries(mytarget PRIVATE Qt6::Sql)
```

✓ 데이터베이스 연결

TCP에서 상대방과 데이터를 송/수신 하기 위해서, 패킷 송/수신 전에 Connection을 맺어야 하는 것과 같이 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.

| 드라이버 명 | 설명 |
|----------|------------------------------|
| QDB2 | IBM DB2 7.1 이상 버전 |
| QIBASE | Borland InterBase |
| QMYSQ | 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)");
```

예수님은 당신을 사랑합니다.

만약 많은 양의 데이터를 핸들링 하기 위해서 SQL 문을 사용 시 Placeholder 방식과 Positioning 방식을 사용할 수 있다. 성능 측면에서는 Placeholder 방식을 권장한다. 다음은 Placeholder 방식의 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 예제 이다.

```
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.

| 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 검색할 수 있는 Model 을 제공한다. 아래와 같이 2개의 테이블을 예로 들어 보자.

| MainTable | |
|-----------|----------|
| 필드 명 | 타입 |
| id | INTERGER |
| Date | DATE |
| Time | TIME |
| Hostname | INTERGER |
| IP | INTERGER |

| DeviceTable | |
|-------------|--------------|
| 필드 명 | 타입 |
| id | INTERGER |
| Hostname | VARCHAR(255) |
| IP | VARCHAR(16) |

위의 표에서 보는 것과 같이 MainTable 과 DeviceTable 이 있다. DeviceTable 에 레코드를 먼저 삽입한다. 그리고 그 후에 Main Table 의 데이터를 삽입 시 DeviceTable 에 있는 Hostname 과 IP 를 삽입하고자 한다면 다음과 같이 사용할 수 있다.

```
QSqlRelationalTableModel *modelMain;
QSqlRelationalTableModel *modelDevice;

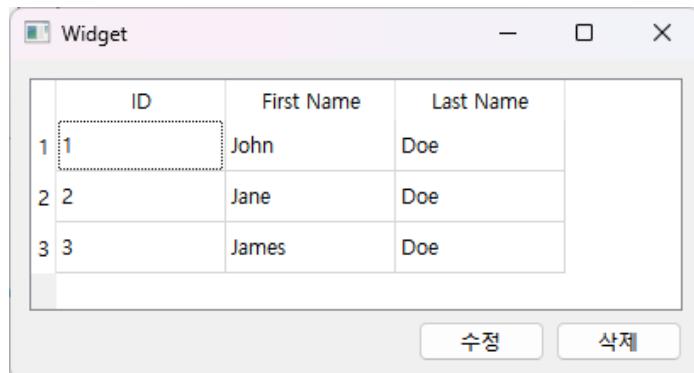
modelMain->setRelation(3, QSqlRelation("DeviceTable", "id", "Hostname"));
modelMain->setRelation(4, QSqlRelation("DeviceTable", "id", "IP"));
...
```

위의 예제와 같이 QSqlRelationalTableModel 클래스에서 제공하는 setRelation() 함수를 이용해 "DeviceTable" 테이블의 Hostname 과 IP 레코드 필드 데이터를 modelMain 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.



예수님은 당신을 사랑합니다.

위의 그림에서 보는 것과 같이 프로젝트를 빌드 후 실행하면 SQLite 데이터베이스를 생성한다. 데이터베이스가 생성되면 “names”라는 테이블이 생성된다. 그리고 이 테이블에 데이터를 삽입하고 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 00_SQLite_Example 디렉토리를 참조하면 된다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QSqlDatabase>
#include <QSqlTableModel>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

    QSqlDatabase m_db;
    QSqlTableModel *m_model;

    bool initializeDataBase();
    void creationTable();
    void insertDataToTable();
    void initializeModel();

private slots:
```

예수님은 당신을 사랑합니다.

```
void slot_pbtUpdate();
void slot_pbtDelete();
};

#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 코드이다.

```
#include "widget.h"
#include "./ui_widget.h"

#include <QSqlQuery>
#include <QSqlError>
#include <QFile>
#include <QDebug>

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

    if( initializeDataBase() )
    {
        creationTable();
        insertDataToTable();
        initializeModel();

        ui->tableView->setModel(m_model);

        connect(ui->pbtUpdate, SIGNAL(pressed()),
                this,           SLOT(slot_pbtUpdate()));
        connect(ui->pbtDelete, SIGNAL(pressed()),
                this,           SLOT(slot_pbtDelete()));

    }
}
```

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

bool Widget::initializeDataBase()
{
    QFile::remove("./my.db");

    m_db = QSqlDatabase::addDatabase("QSQLITE");
    m_db.setDatabaseName("./my.db");

    if( !m_db.open() ) {
        qDebug() << Q_FUNC_INFO << m_db.lastError().text();
        return false;
    }

    return true;
}

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().text();
}
}

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()
{
    m_model = new QSqlTableModel(this, m_db);

    m_model->setTable("names");
    m_model->setEditStrategy(QSqlTableModel::OnManualSubmit);
    m_model->select();

    m_model->setHeaderData(0, Qt::Horizontal, tr("ID"));
    m_model->setHeaderData(1, Qt::Horizontal, tr("First Name"));
    m_model->setHeaderData(2, Qt::Horizontal, tr("Last Name"));
}

void Widget::slot_pbtUpdate()
{
    QSqlQuery qry;
    qry.prepare( "UPDATE names "
                "SET firstname = 'Eddy', "
                "lastname = 'Kim' WHERE id = 1" );
    if( !qry.exec() )
        qDebug() << qry.lastError();
}
```

```
m_model->setTable("names");
m_model->select();
ui->tableView->setModel(m_model);
}

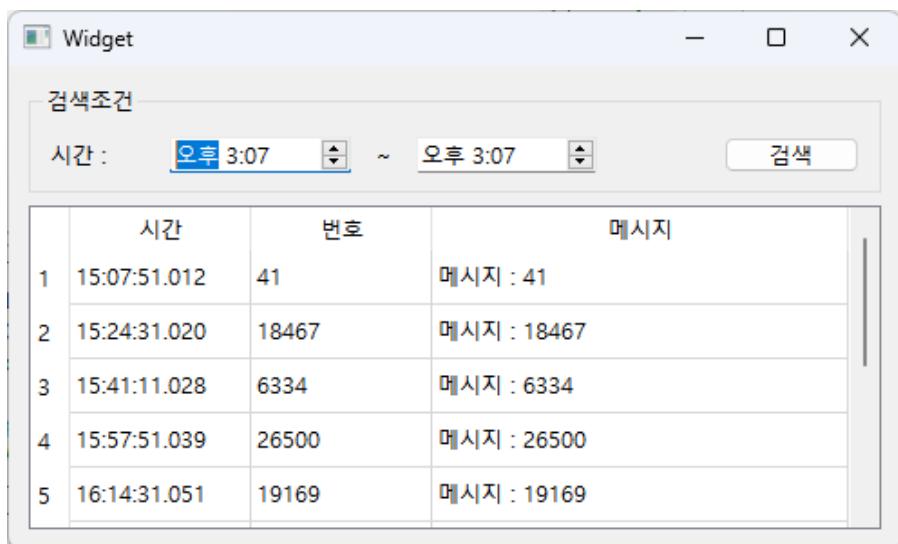
void Widget::slot_pbtDelete()
{
    QSqlQuery qry;

    qry.prepare( "DELETE FROM names WHERE id = 1" );
    if( !qry.exec() )
        qDebug() << qry.lastError();

    m_model->setTable("names");
    m_model->select();
    ui->tableView->setModel(m_model);
}
```

✓ 데이터베이스 테이블 검색 예제

이번 예제에서는 데이터베이스 테이블에 저장된 데이터를 QSqlTableModel로 가져와 QTableView 위젯에 출력한다. 그리고 시간을 기준으로 검색하는 기능을 구현해 보도록 하자.



예수님은 당신을 사랑합니다.

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 01_Search_Example 디렉토리를 참조하면 된다.

```
#ifndef DATABASEHANDLER_H
#define DATABASEHANDLER_H

#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();
}
```

예수님은 당신을 사랑합니다.

```
};

#endif // DATABASEHANDLER_H
```

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. 다음은 databasehandler.cpp 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;
```

예수님은 당신을 사랑합니다.

```
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.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QSqlTableModel>
#include "databasehandler.h"

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

예수님은 당신을 사랑합니다.

```
~Widget();

private:
    Ui::Widget *ui;

    DatabaseHandler *m_dbHandler;
    QSqlTableModel *m_model;

    void setupModel(const QString &tableName, const QStringList &headers);
    void createUserInterface();

private slots:
    void onPushButton();

};

#endif // WIDGET_H
```

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 QDateTimeEdit의 시간을 현재 시간으로 설정한다. 다음은 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_dbHandler = new DatabaseHandler();
    m_dbHandler->connectToDataBase();

    for(int i = 0; i < 10; i++)
    {
        QVariantList data;
```

```
QTime currTime = QTime::currentTime();
currTime = currTime.addSecs(i*1000);

int random = rand();

data.append(currTime);
data.append(random);
data.append("메시지 : " + QString::number(random));

m_dbHandler->insertIntoTable(data);
}

setupModel("RECEIVE_MAIL",
QStringList() << "ID"
             << "시간"
             << "번호"
             << "메시지"
);

createUserInterface();
}

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

void Widget::setupModel(const QString &tableName,
                      const QStringList &headers)
{
    m_model = new QSqlTableModel(this);
    m_model->setTable(tableName);

    for(int i = 0, j = 0; i < m_model->columnCount(); i++, j++) {
        m_model->setHeaderData(i, Qt::Horizontal, headers[j]);
    }

    m_model->setSort(0,Qt::AscendingOrder);
}

void Widget::createUserInterface()
{
```

```
ui->tableView->setModel(m_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);

m_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()
{
    QString str = QString("TIME between '%3' and '%4'")
                  .arg(ui->timeFROM->time().toString("hh:mm:ss"));

    m_model->setFilter(QString( "TIME between '%3' and '%4' ")
                        .arg(ui->timeFROM->time().toString("hh:mm:ss"),
                            ui->timeTO->time().toString("hh:mm:ss")));

    m_model->select(); // 테이블로부터 데이터를 Fetch
}
```

27. Qt for Android

이번 장에서는 Qt를 이용해 Android 응용 어플리케이션을 개발하는 방법에 대해서 살펴보도록 하자. 이번 장은 두 개의 단원으로 구성된다.

첫 번째 단원은 MS Windows 운영체제에서 Qt를 이용해 Android 어플리케이션을 다루는 방법을 다룬다. 이 단원에서는 MS Windows 운영체제에서 환경 구축을 구축하는 방법에 대해서 살펴본다. 그리고 Qt를 이용해 Android 응용 어플리케이션을 개발하고 안드로이드 스마트폰에 APK 설치하고 실행해 보자.

두 번째 단원은 Linux에서 Qt를 이용해 Android 어플리케이션을 다루는 방법에 대해서 다룬다. 이 단원에서는 Linux 운영체제에서 환경 구축을 구축하는 방법에 대해서 살펴본다. 그리고 Qt를 이용해 Android 응용 어플리케이션을 개발하고 안드로이드 스마트폰에 APK 설치하고 실행해 보도록 하자.

27.1. MS Windows 에서 Android 개발 환경 구축과 앱 배포

MS Windows에서 Android 개발 환경을 구축하기 위해서 Qt 설치 시 아래 그림에서 보는 것과 같이 설치 다이얼로그에서 "Android" 항목을 선택해야 한다.



위와 같이 선택 후 Qt를 설치해야만 Android 기반의 App 을 개발할 수 있다. 그리고 Qt 설치 후 몇가지 패키지를 설치해야 한다. 먼저 JDK를 설치해 보도록 하자.

- ✓ Open JDK 다운로드 및 설치

Open DJK 사이트 (jdk.java.net/java-se-ri/19) 에 방문하면 아래와 같이 Open JDK를 다운로드 받을 수 있다.

jdk.java.net

Java Platform, Standard Edition 19 Reference Implementations

The official Reference Implementation for Java SE 19 (JSR 394) is based solely upon open-source code available from the JDK 19 Project in the OpenJDK Community.

The binaries are available under the GNU General Public License version 2, with the Classpath Exception.

These binaries are for reference use only!

These binaries are provided for use by implementers of the Java SE 19 Platform Specification and are for reference purposes only. This Reference Implementation has been approved through the Java Community Process. Production-ready binaries under the GPL are available from Oracle; and will be in most popular Linux distributions.

RI Binary (build 19+36) under the GNU General Public License version 2

- Oracle Linux 8.5 x64 Java Development Kit (sha256) 187 MB
- Windows 11 x64 Java Development Kit (sha256) 186 MB

RI Source Code

The source code of the RI binaries is available under the GPLv2 in a single zip file (sha256) 170 MB.

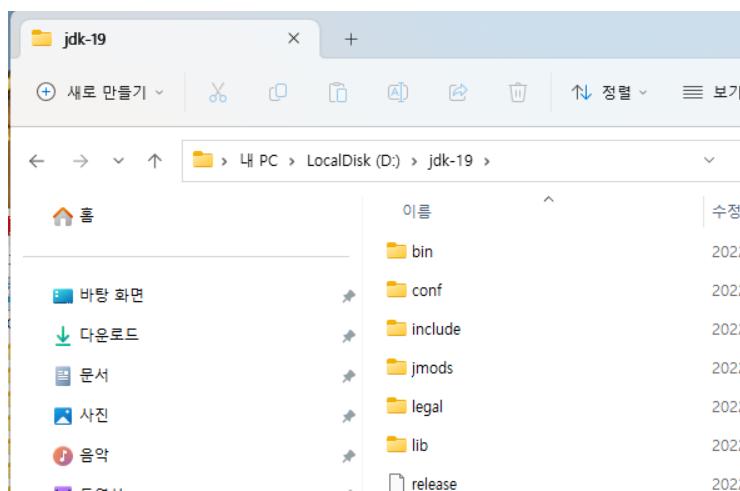
International use restrictions

Due to limited intellectual property protection and enforcement in certain countries, the JDK source code may only be distributed to an authorized list of

위에서 보는 것과 같이 Java SE 19 버전을 선택하고 Windows 11 버전을 선택한다. 만약 사용 하는 MS Windows 10 이거나 그 이하라면 다른 Java SE 18 또는 하위 버전을 선택하면 된다.

다운로드 받은 파일을 압축 해제 후 아래와 같이 디렉토리로 이동한다. (사용자가 원하는 위치로 이동하여도 무방하다.)

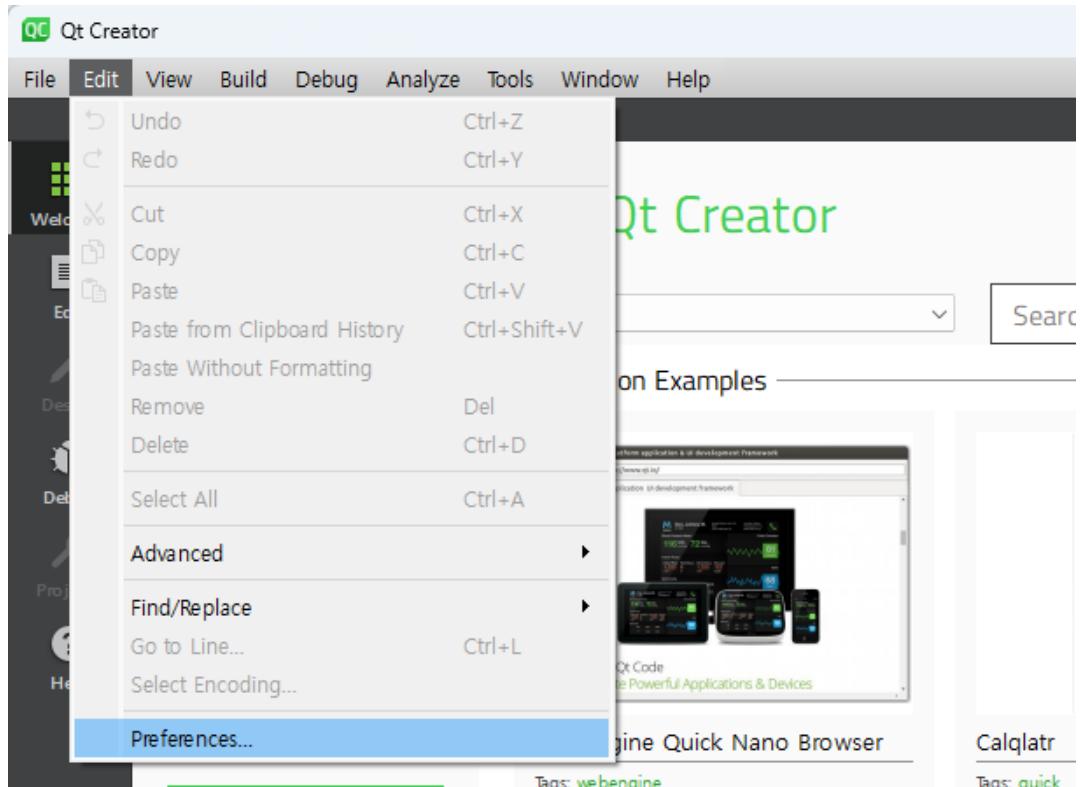
- ✓ 디렉토리 위치: D:\jdk-19



예수님은 당신을 사랑합니다.

Open JDK 사용시 주의 할 점으로 버전이 호환되지 않는 버전이 존재한다. 만약 버전이 호환되지 않는 경우 하위 버전을 사용하면 문제가 해결된다.

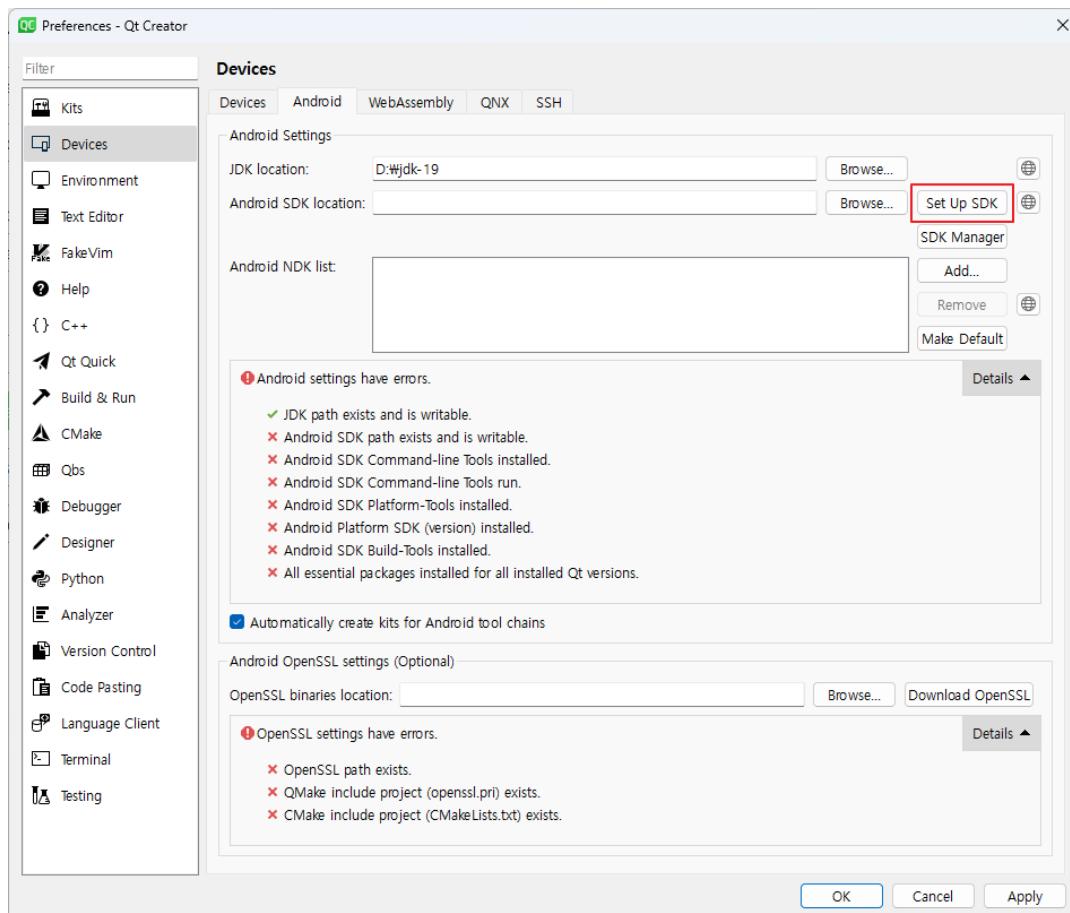
다음으로 Qt Creator 를 실행 후 [Edit] -> [Preferences]를 선택한다.



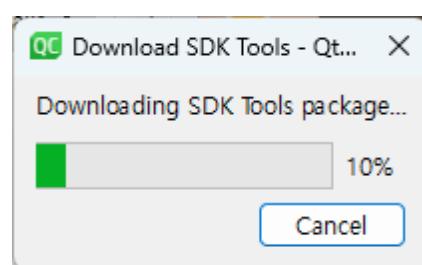
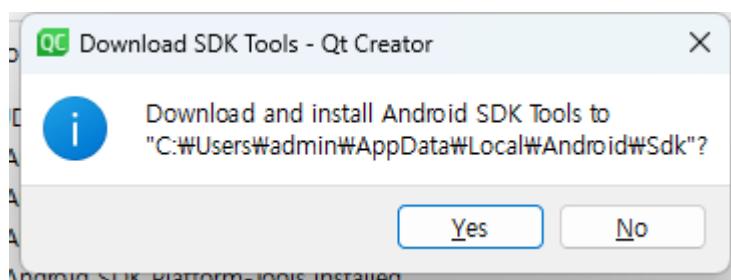
[Preferences] 를 실행하면 JDK location 이 설정된 것을 확인할 수 있을 것이다. 다음으로 Android SDK를 설치해야 한다.

아래 그림에서 보는 것과 같이 [Set Up SDK] 버튼을 클릭하면 다운로드와 설치를 쉽게 할 수 있는 기능을 제공해 준다. 따라서 [Set Up SDK] 버튼을 클릭해 SDK를 설치해 보도록 하자.

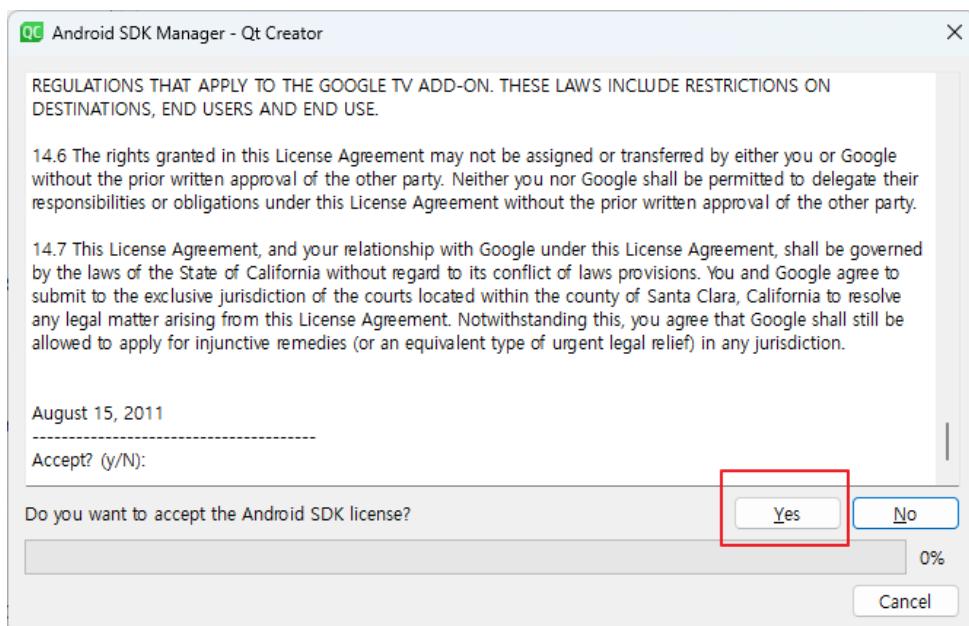
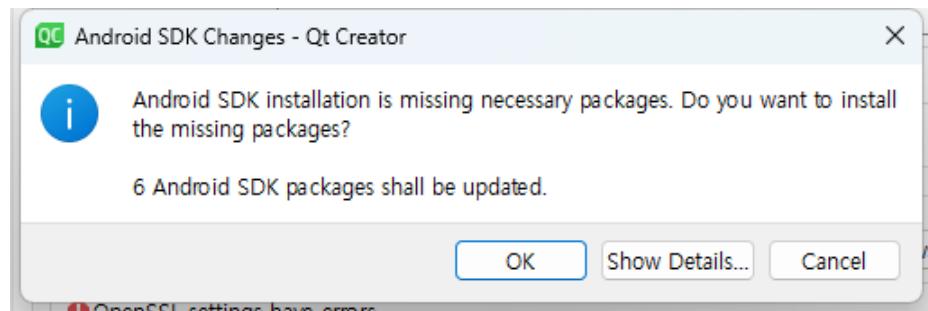
예수님은 당신을 사랑합니다.



아래 다이얼로그에서 [Yes]를 클릭한다.

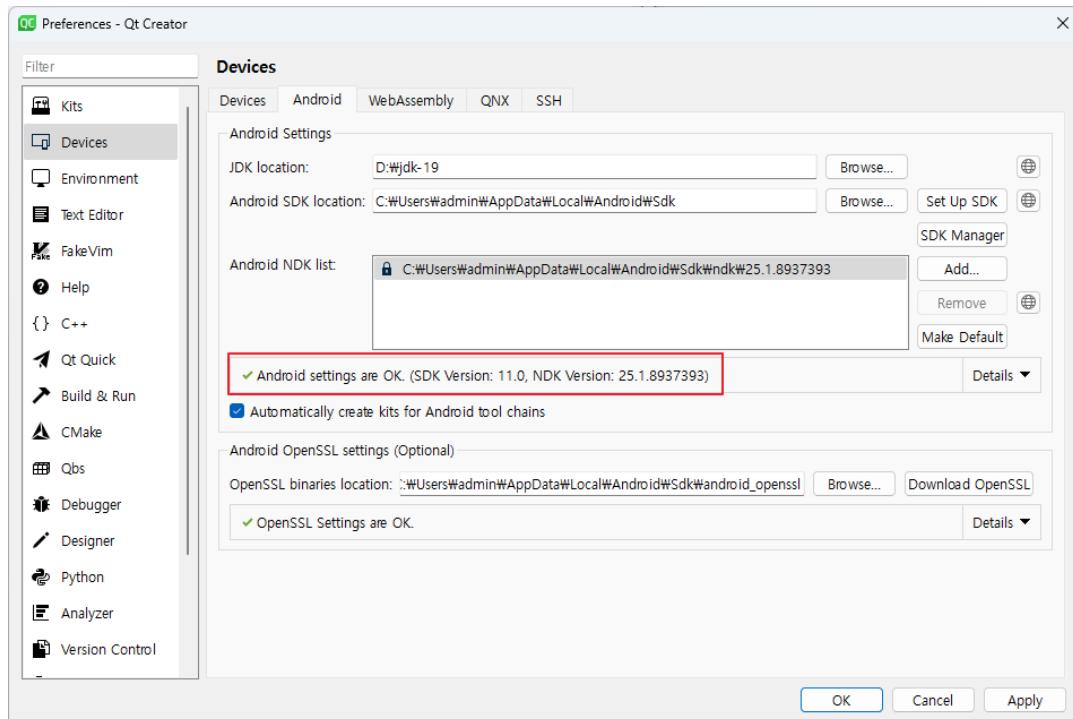


예수님은 당신을 사랑합니다.



위의 다이얼로그에서 여러 번 [Yes] 버튼을 클릭하는 것을 요구한다.

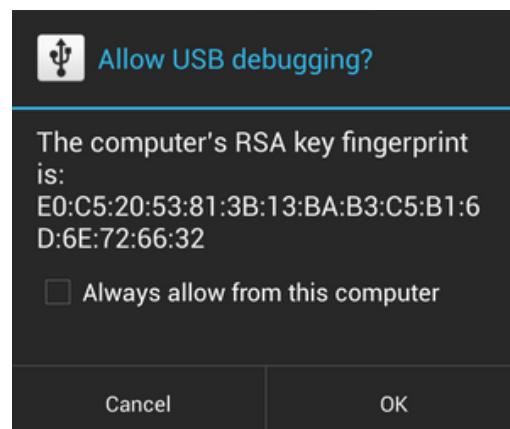
패키지 설치가 완료되면 아래 그림에서 보는 것과 같이 패키지 다운로드 및 설정이 완료된 것을 확인할 수 있다.



그리고 OpenSSL binaries location 이 자동으로 설치되지 않는 경우가 있다. 이런 경우 [Download OpenSSL] 버튼을 클릭하면 Open SSL 을 다운로드 받을 수 있는 사이트로 이동한다. 그 사이트에서 다운로드 받은 후 압축 해제한 디렉토리에 디렉토리를 명시해 주면 된다.

여기까지 Qt를 이용한 Android App 개발 환경 구성은 완료하였다. 이번에는 App 을 디버깅하고 배포할 실제 Android Device의 설정할 차례이다.

Android Device에서 Qt 로 작성한 App을 디버깅 하기 위해서 USB연결 시, 안드로이드의 USB 디버깅을 아래 그림에서 보는 것과 같이 허용해야 한다.



예수님은 당신을 사랑합니다.

위의 그림에서 보는 것과 같이 실제 Android Device의 USB 디버깅 모드를 허용한다. Android Device 의 USB 디버깅을 허용하는 방법은 인터넷상에서 검색을 해보면 자세한 설명이 나와있다.

- Android Device 개발자 옵션 활성화와 USB 디버깅 모드 허용하기

Android Device 상에서 Qt로 작성한 어플리케이션을 디버깅 및 배포하기 위해서는 개발자옵션을 활성화 해야 한다. 개발자 옵션을 활성화 하는 것도 제조사별로 조금씩 다를 수 있다. 아래 그림에서 보는 것과 같이 설정 메뉴로 들어간다.



설정 메뉴에서 휴대전화 정보를 선택한다.

예수님은 당신을 사랑합니다.

KT 1:47

72%

< 휴대전화 정보



RF9R4038K2W

IMEI

351142481232440

상태 정보

법률 정보

규제 정보

소프트웨어 정보

배터리 정보

삼성전자 AS 문의/예약

1588-3366 / www.3366.co.kr

다른 기능을 찾고 있나요?

[소프트웨어 업데이트](#)

[초기화](#)

[문의하기](#)



예수님은 당신을 사랑합니다.

KT 1:48 kt

5G 🔍 WiFi HD 72% 🔋

< 소프트웨어 정보

안드로이드 버전

13

Google Play 시스템 업데이트

2023년 6월 1일

기저 대역 버전

A325NKOU4DWH3

커널 버전

4.14.186-27270957

#1 Fri Aug 11 16:36:23 KST 2023

빌드번호

TP1A.220624.014.A325NKSU4DWH3

SE for Android 상태

Enforcing

SEPF_SM-A325N_12_0001

Fri Aug 11 16:48:38 2023

Knox 버전

Knox 3.9

Knox API level 36

HDM 2.0 - 1F



위의 그림에서 보는 것과 같이 [빌드번호]를 7번 클릭하면 아래 그림에서 보는 것과

27.1. MS Windows에서 Android 개발 환경 구축과 앱 배포 페이지 328

예수님은 당신을 사랑합니다.

같이 개발자 모드가 활성화 된다. 개발자 옵션을 선택한다.



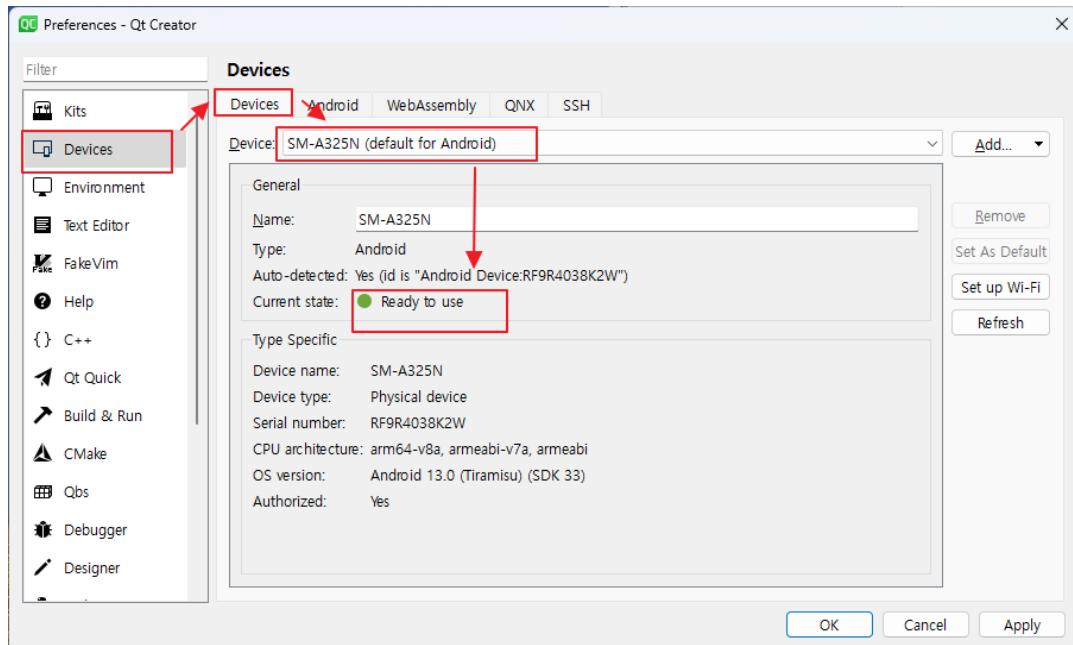
아래 그림에서 보는 것과 같이 USB 디버깅 모드를 활성화 한다.

예수님은 당신을 사랑합니다.



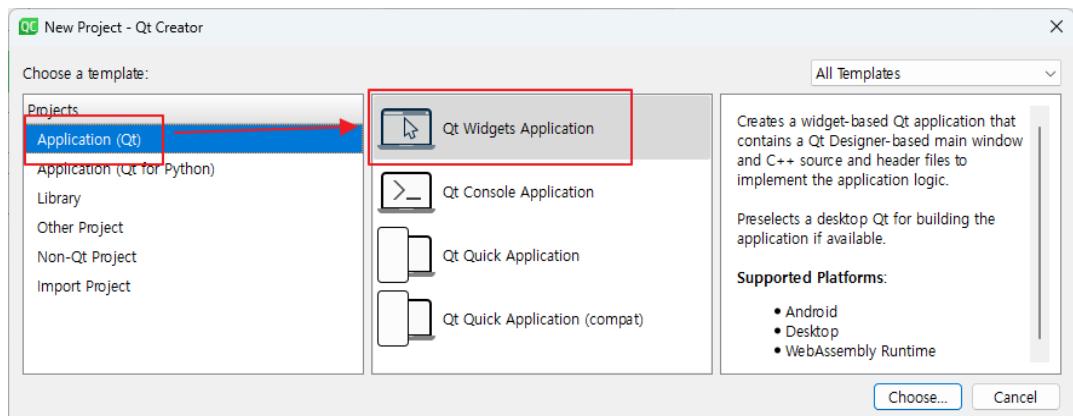
여기까지 완료하였다면 다음으로 Qt Creator에서 Android Device를 연결하는 작업을 진행해야 한다. [Edit] > [Preferences] 메뉴를 클릭한다. 그리고 오른쪽에서 [Device]를 선택한다. 그런 다음 Devices 항목에서 실제 Android Device를 선택한다.

그러면 아래 그림에서 보는 것과 같이 Current state 항목에 "Ready to use" 텍스트를 확인할 수 있다.

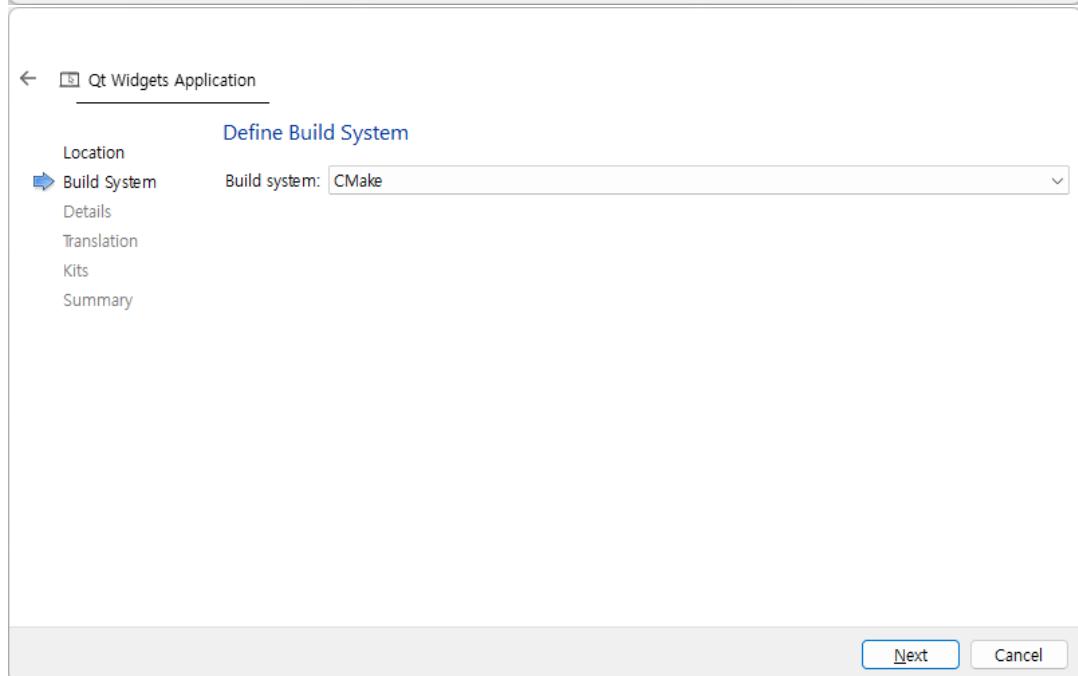
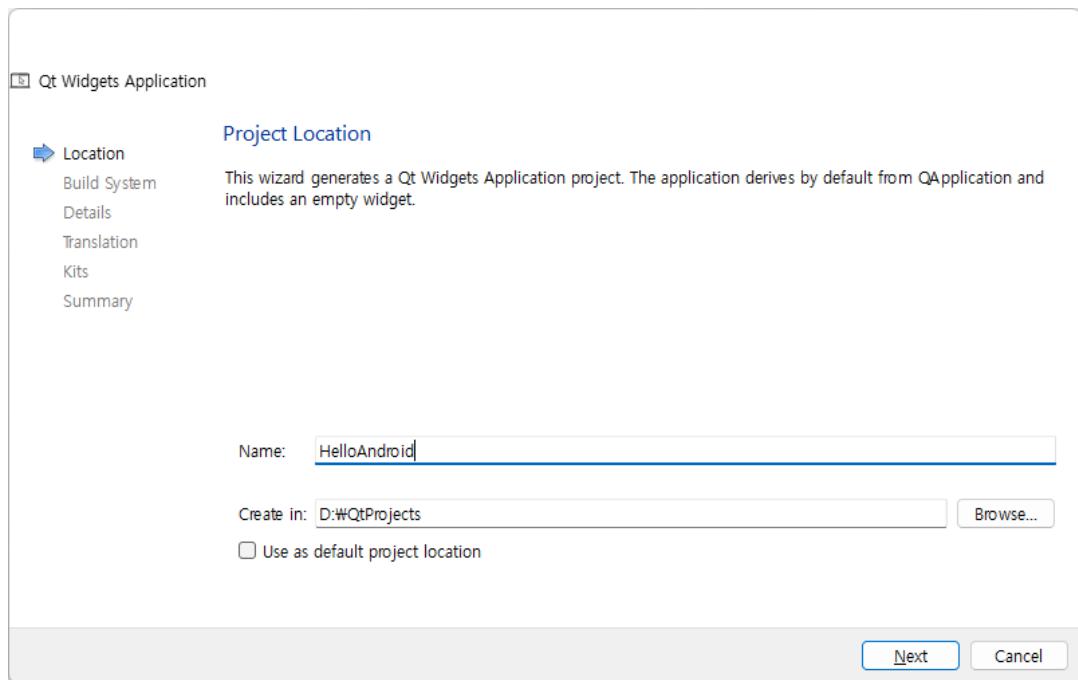


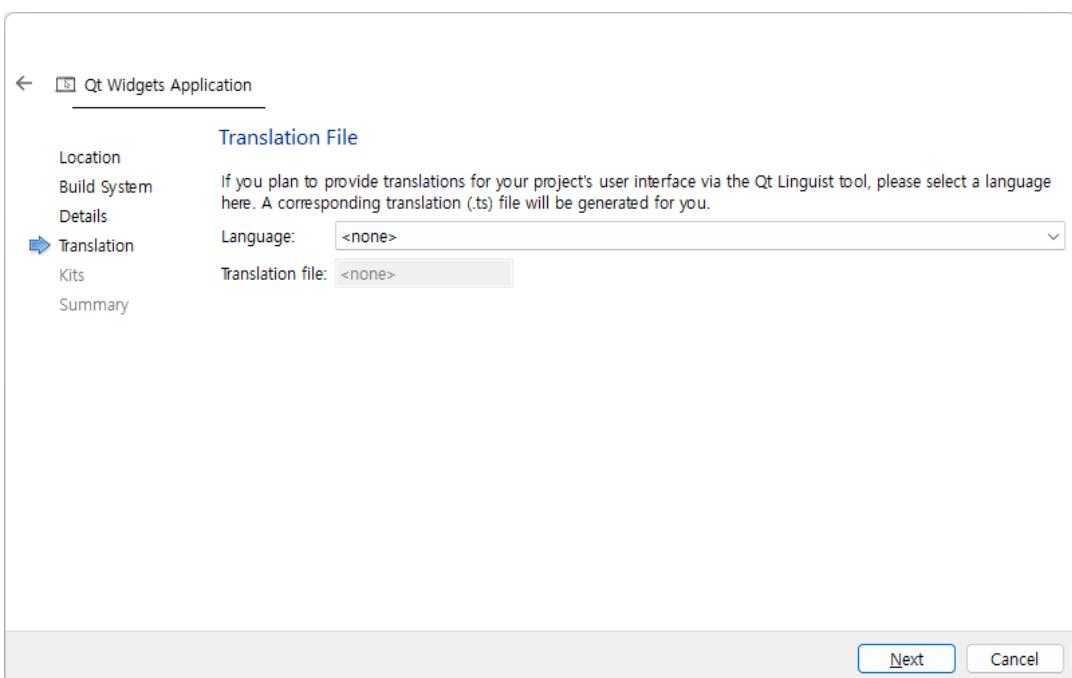
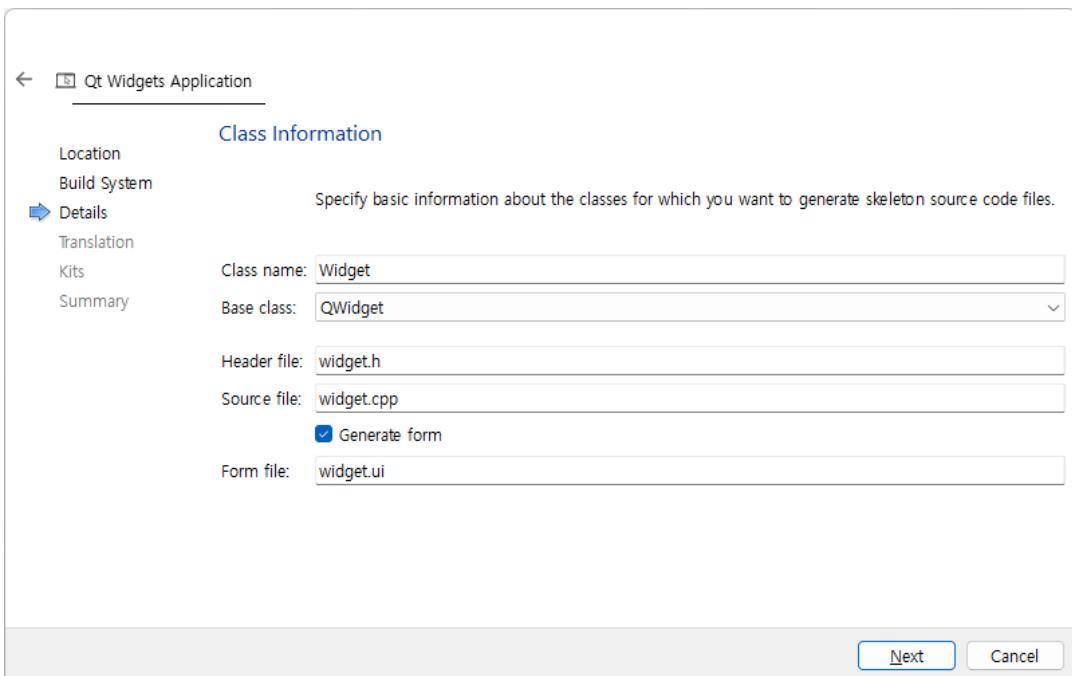
- Android Device 상에 Qt 로 작성한 어플리케이션 빌드 및 배포하기

Qt로 작성한 예제를 빌드 후 Android Phone 상에 배포하고 디버깅 해 보도록 하자.
Qt Creator를 실행하고 아래 그림에서 보는 것과 같이 프로젝트를 생성한다.

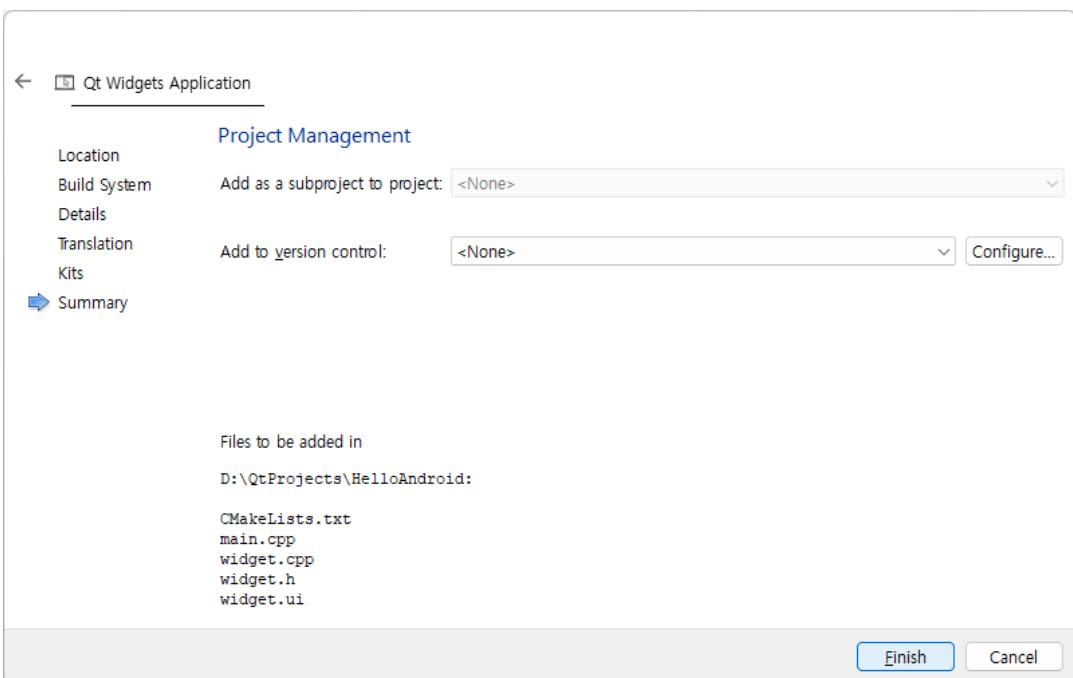
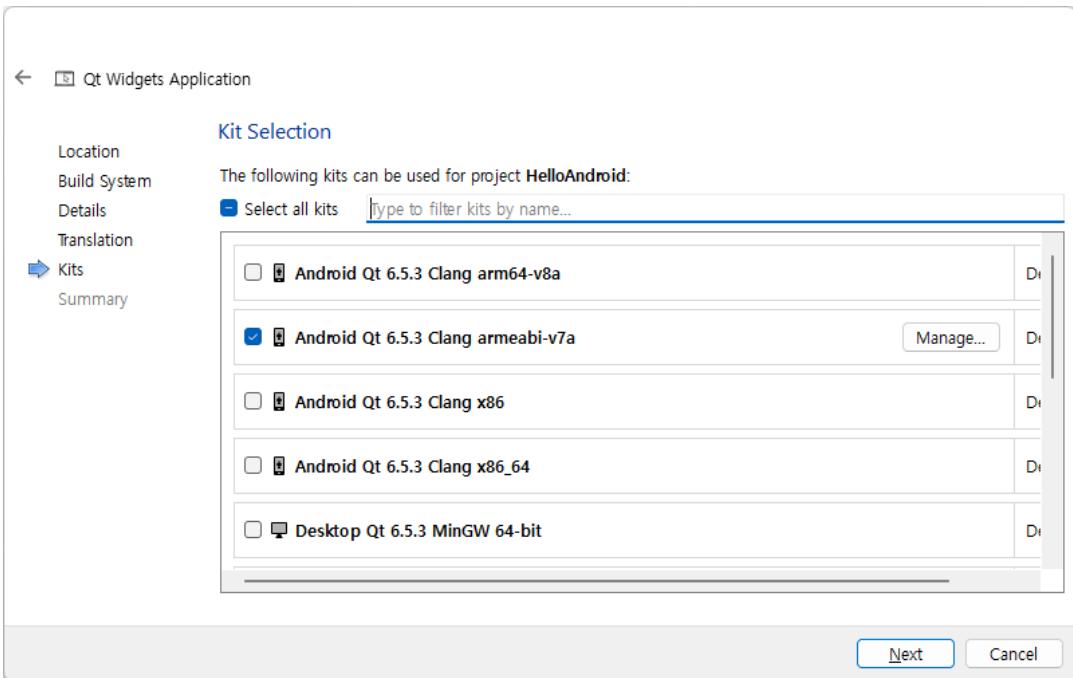


예수님은 당신을 사랑합니다.



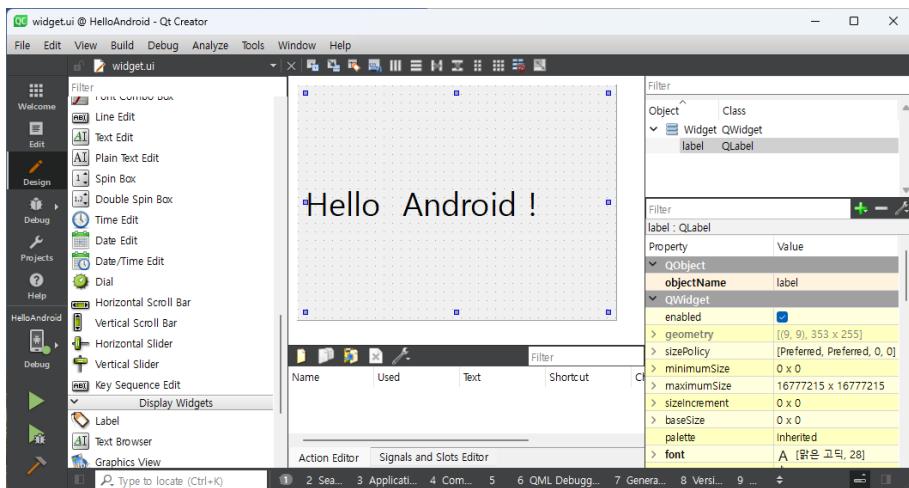


예수님은 당신을 사랑합니다.

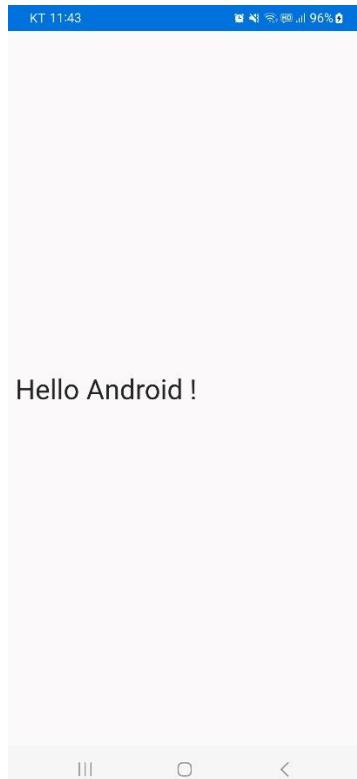


프로젝트 생성이 완료되면 widget.ui 파일을 더블 클릭한다. 그런 다음 GUI상에 QLabel을 배치한다. 배치한 QLabel에 표시되는 텍스트를 "Hello Android"로 변경하고 폰트 크기를 28로 변경한다.

예수님은 당신을 사랑합니다.



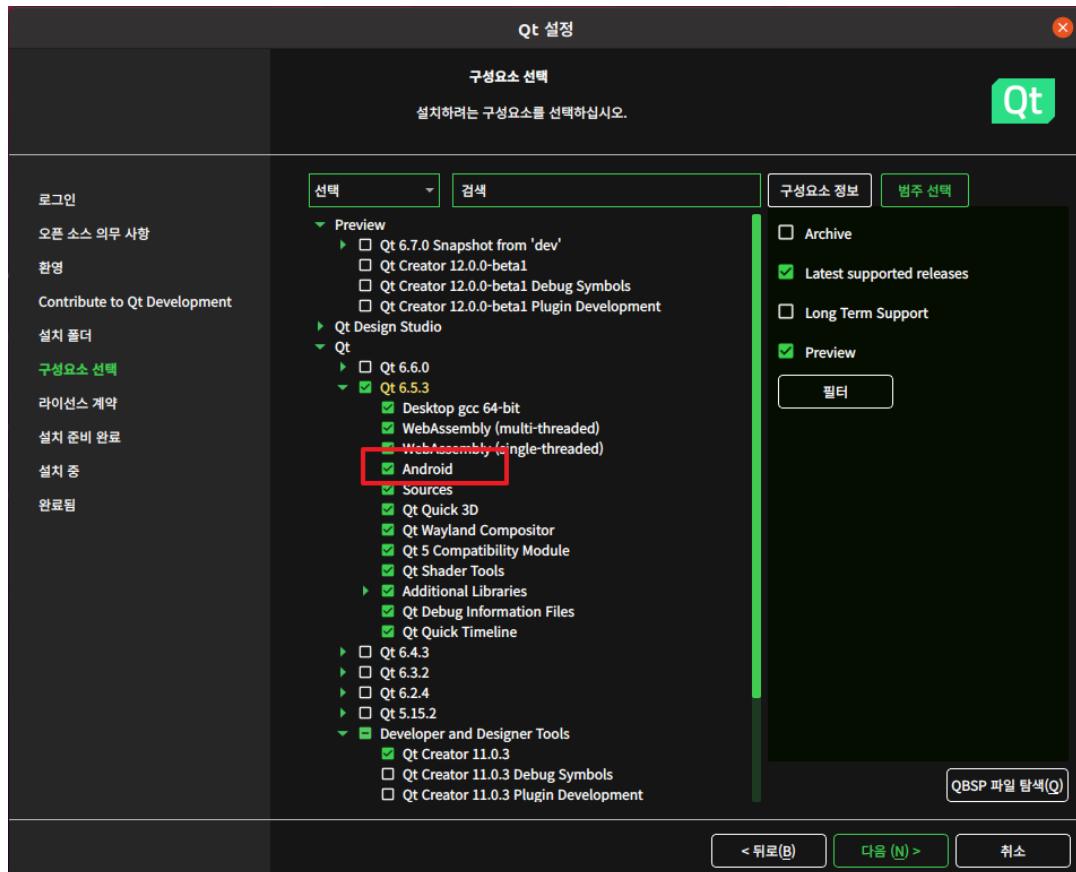
QLabel의 텍스트와 폰트 변경을 완료한 후 저장한 다음 소스코드로 돌아와 빌드 후 실행해 보도록 하자. 빌드 후 실행이 완료되면 실제 Android Device 사에 App 이 실행된 것을 확인할 수 있다.



27.2. Linux에서 Android 개발 환경 구축과 앱 배포

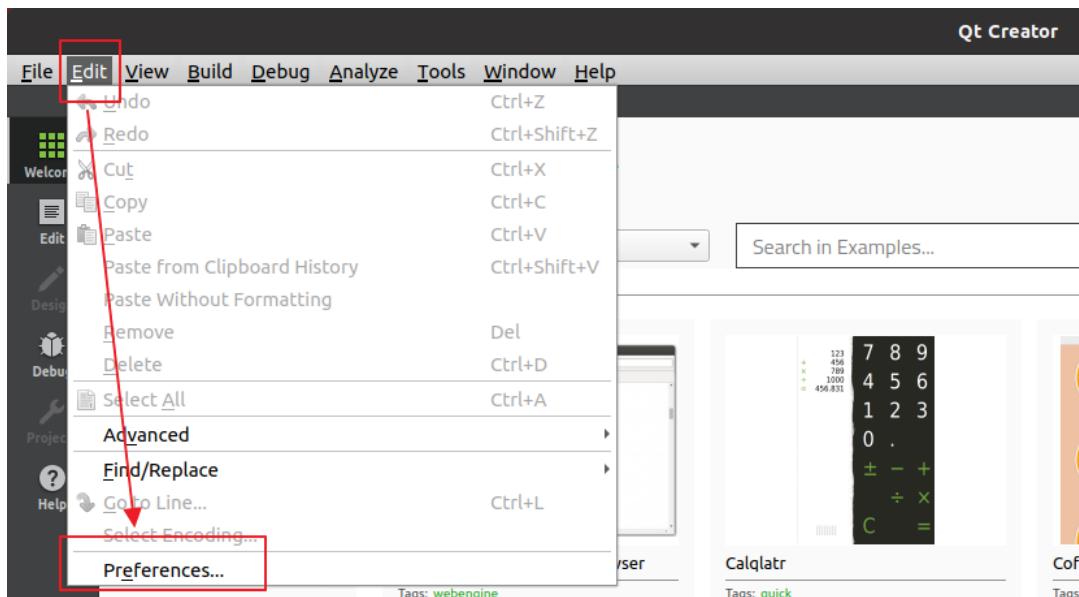
이번 장에서는 Ubuntu Linux 20.04에서 Android 개발 환경을 구축한 후 UI에 "Hello Android!"를 출력하는 간단한 App을 만들어보자.

Ubuntu Linux에서 Qt 설치 시, Android 항목을 추가해야 한다.



● Android 개발 환경 구축

위와 같이 선택 후 설치하면 된다. 그 다음으로 설치가 완료되면 Qt Creator를 실행한다. Qt Creator가 실행되면 메뉴에서 [Edit] 메뉴를 선택한다. 그리고 [Preferences...] 메뉴를 아래와 같이 선택한다.



아래 다이얼로그에서 가장 먼저 JDK location 항목에 JDK(Java Developer Kit)를 설치 후, 설치된 JDK의 위치를 입력한다.

여기서는 Open JDK를 사용하겠다. 그리고 버전으로 JDK 17 버전을 사용하겠다.

Java Platform, Standard Edition 17 Reference Implementations

The official Reference Implementation for Java SE 17 (JSR 392) is based solely upon open-source code available from the [JDK 17 Project](#) in the [OpenJDK Community](#).

The binaries are available under the [GNU General Public License version 2](#), with the [Classpath Exception](#).

These binaries are for reference use only!

These binaries are provided for use by implementers of the Java SE 17 Platform Specification and are for reference purposes only. This Reference Implementation has been approved through the Java Community Process. Production-ready binaries under the GPL are available from [Oracle](#); and will be in most popular Linux distributions.

RI Binary (build 17+35) under the GNU General Public License version 2

- Oracle Linux 8.3 x64 Java Development Kit (sha256) 179 MB
- Windows 10 x64 Java Development Kit (sha256) 178 MB

RI Source Code

The source code of the RI binaries is available under the [GPLv2](#) in a single zip file ([sha256](#)) 165 MB.

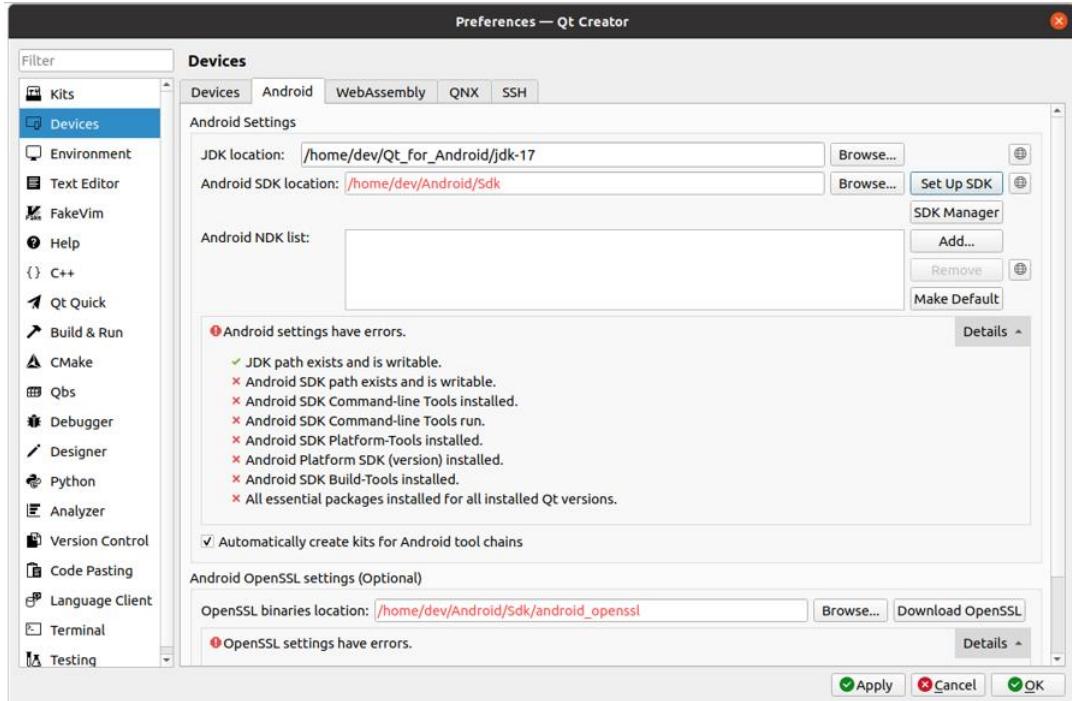
International use restrictions

Due to limited intellectual property protection and enforcement in certain countries, the JDK source code may only be distributed to an authorized list of countries. You will not be able to access the source code if you are downloading from a country that is not on this list. We are continuously reviewing this list for addition of other countries.

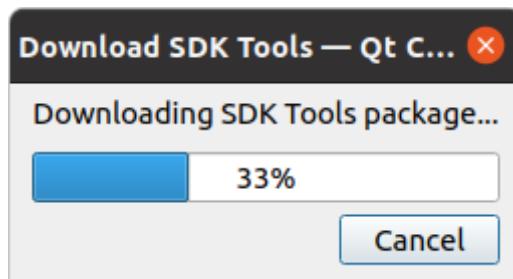
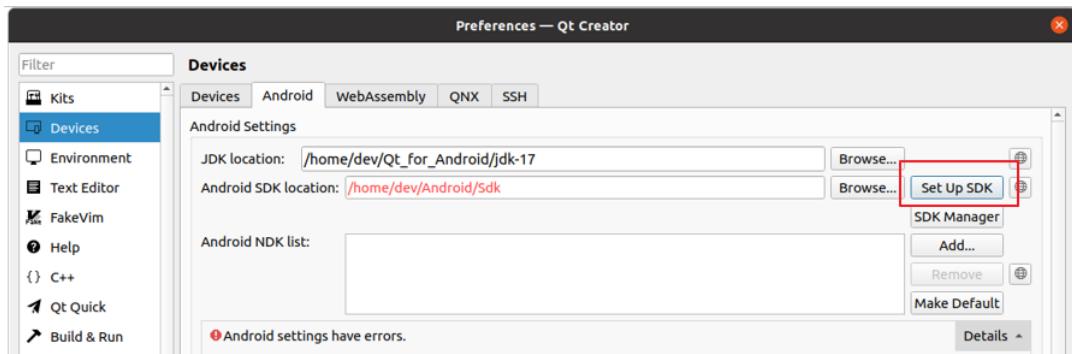
Open JDK를 다운로드 받은 후 압축을 해제한다. 그리고 해제한 디렉토리를 아래 그림

예수님은 당신을 사랑합니다.

에서 보는 것과 같이 Qt Creator 로 돌아와서 JDK Location 항목에 Open JDK 의 압축이 해제된 디렉토리 위치를 입력한다.

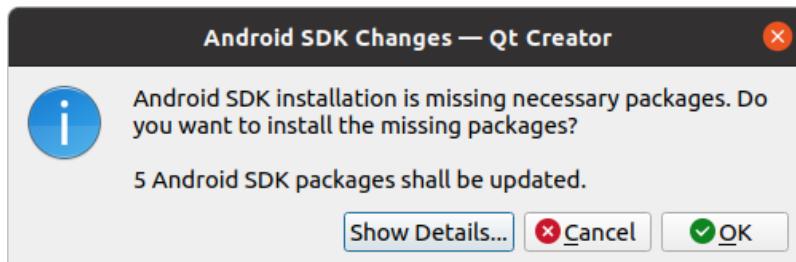


다음으로 [Set Up SDK] 버튼을 클릭한다.

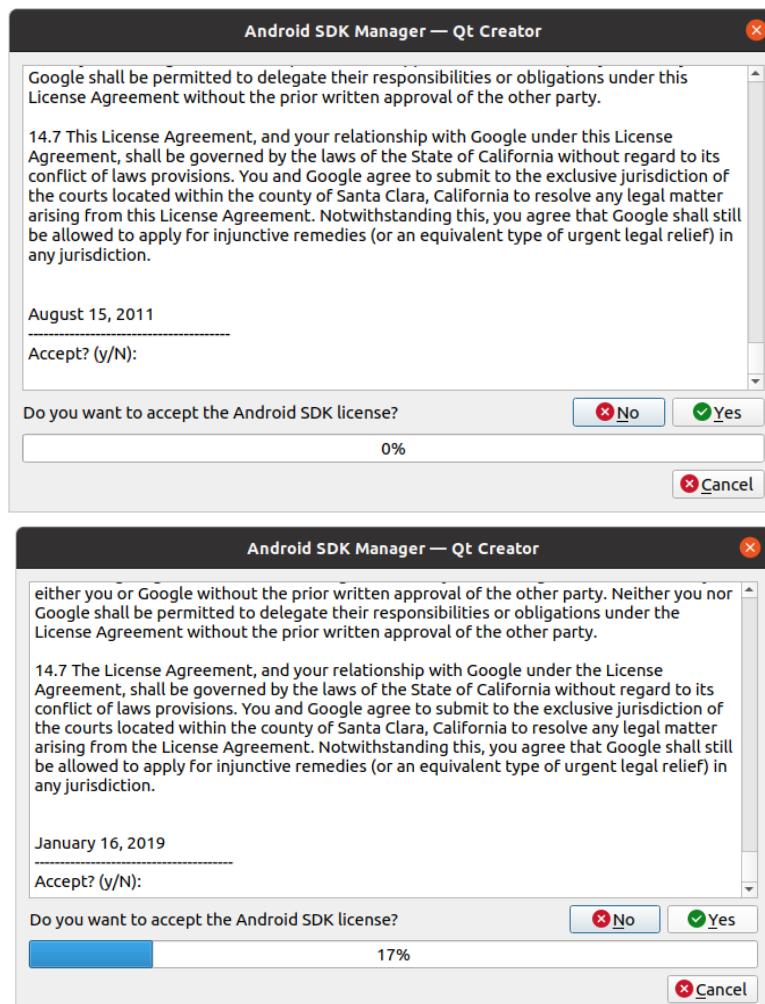


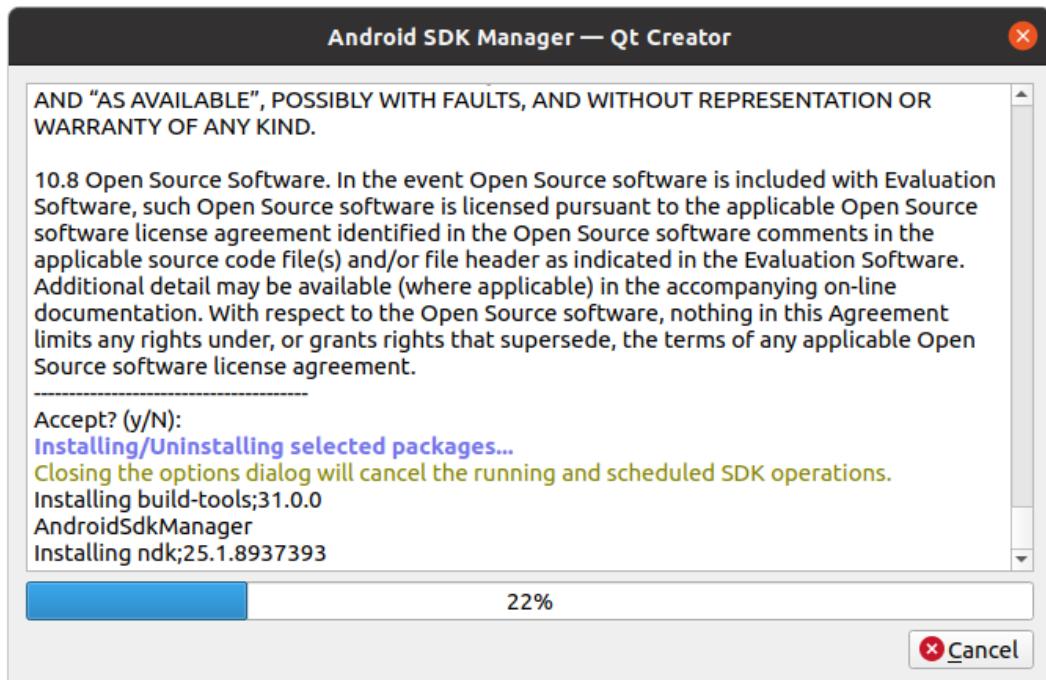
예수님은 당신을 사랑합니다.

다음은 Android SDK package를 설치하기 전에, 설치를 동의하는ダイ얼로그이다. [OK] 버튼을 클릭한다.

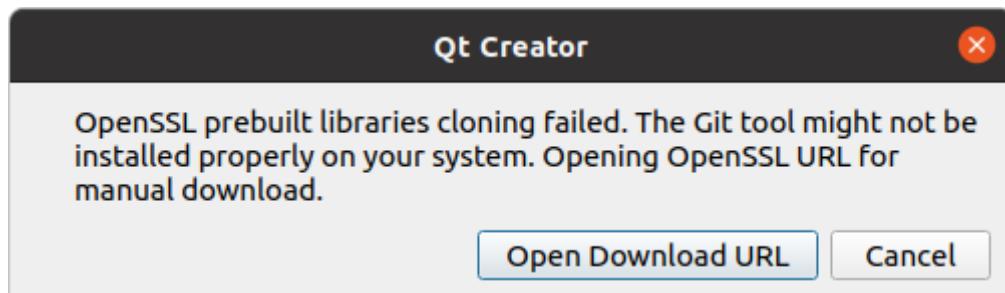
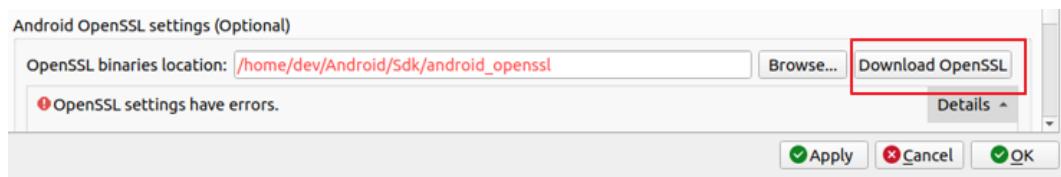


다음ダイ얼로그는 라이선스와 관련해 동의를 구하는ダイ얼로그이다. 여러 번 [Yes] 버튼을 누르기를 요구한다. 매번 [Yes]를 클릭한다.



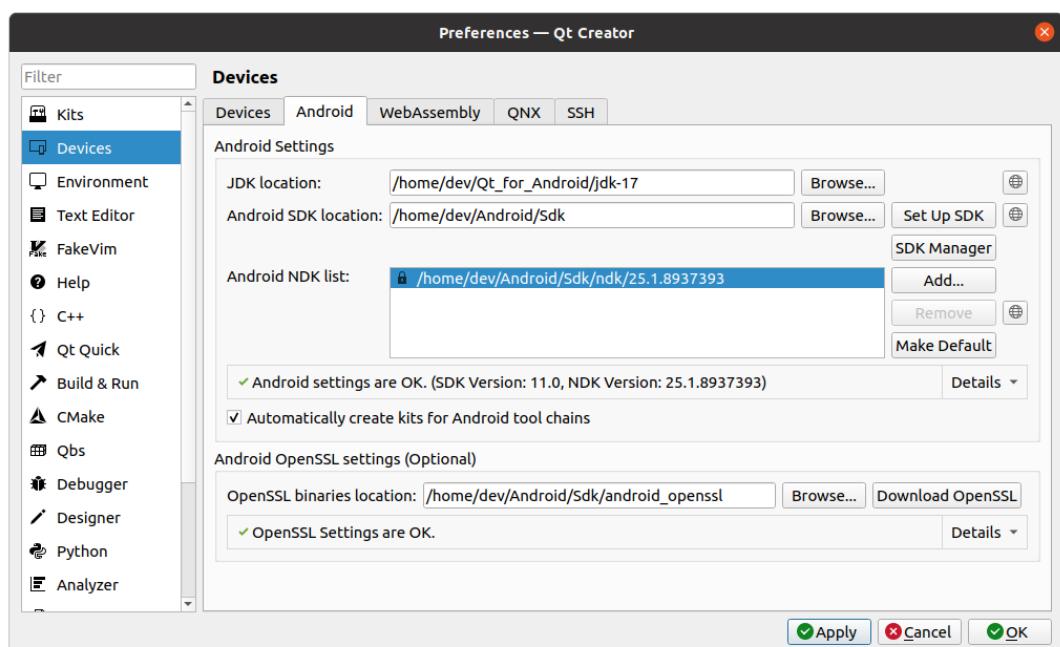


설치가 완료되면 다이얼로그 하단에 OpenSSL binaries location 항목에 OpenSSL을 다운로드 받은 후 압축해제한다. 그런 다음 압축 해제한 디렉토리 위치를 아래 그림에서 보는 것과 같이 명시해 준다.

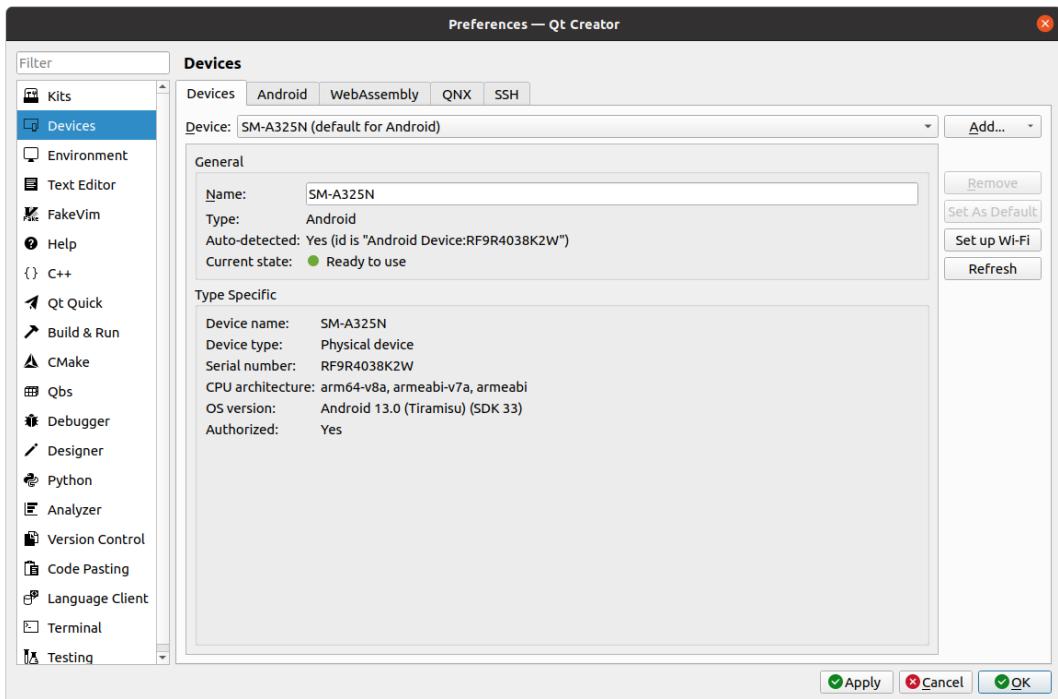


예수님은 당신을 사랑합니다.

The screenshot shows a GitHub repository page for 'KDAK/android_openssl'. The repository has 2 branches and 7 tags. The 'About' section includes links for Readme, Apache-2.0 license, Activity, 263 stars, 29 watching, 121 forks, and Report repository. The 'Releases' section shows 7 tags. There are no packages published.



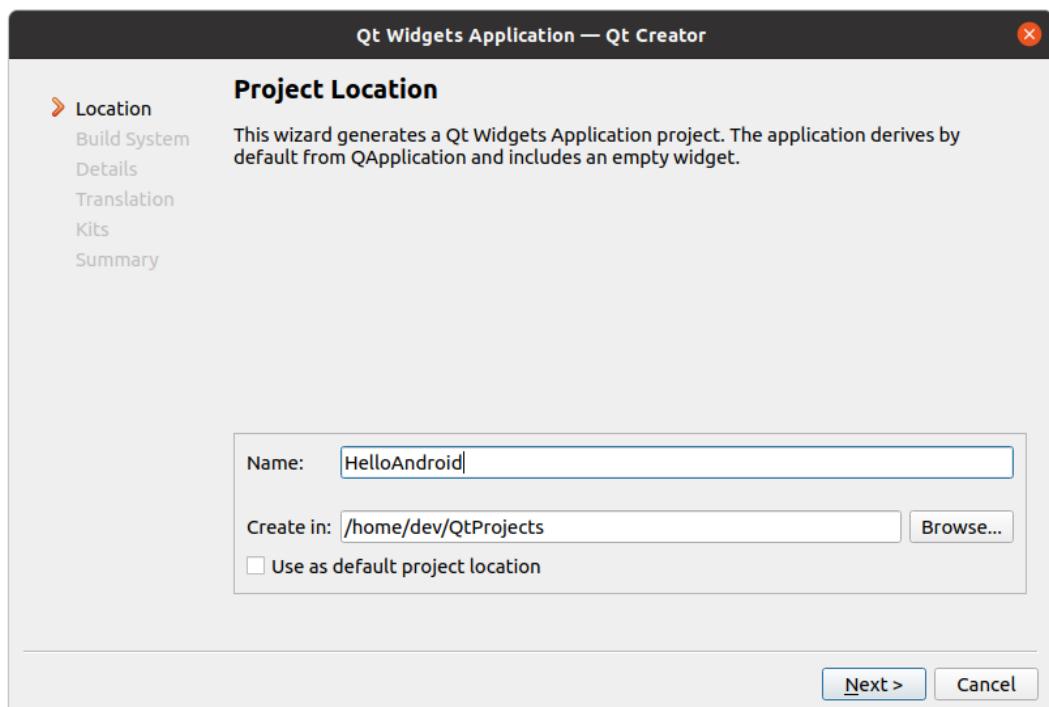
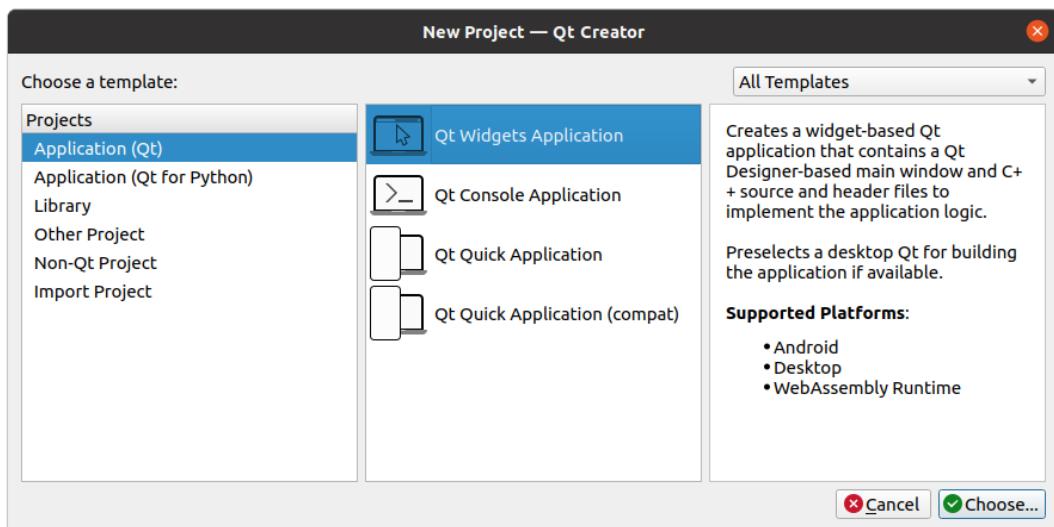
다음으로 다이얼로그 상단에 [Devices] 항목을 선택한다. 실제 연결한 Android Device 상에 빌드한 App을 실행하기 위해서는 아래 그림에서 보는 것과 같이 [Current state] 항목에 "Ready to use"라는 메시지가 보이면 연결한 Android device 가 정상적으로 연결된 것을 의미한다.



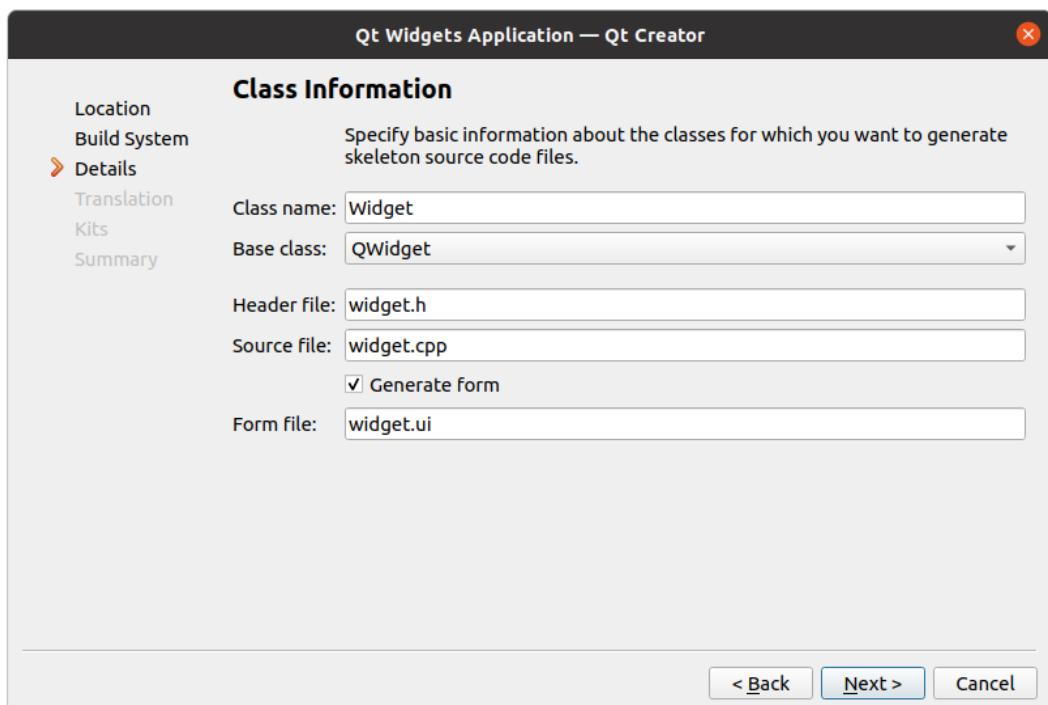
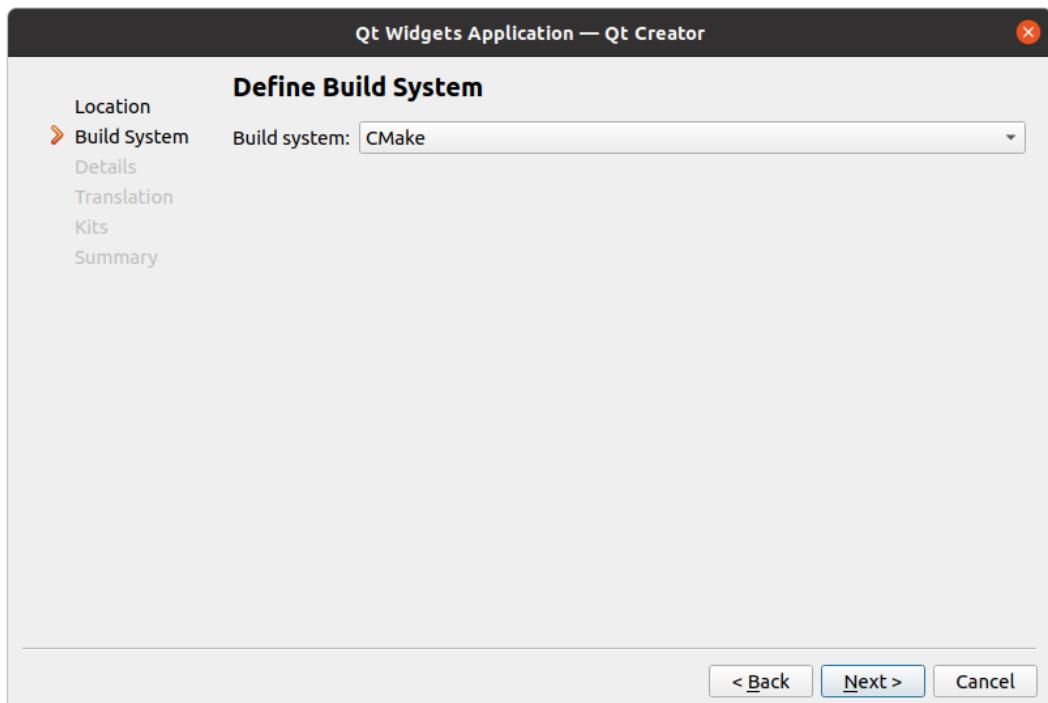
여기까지 되었으면 개발 환경 구축은 완료되었다. Android Device 의 필요한 설정은 이전 장에서 설명 하였으므로 이전 장에서 Android Device 설정 부분을 참조하기 바란다.

- Android Device 상에 Qt 로 작성한 어플리케이션 빌드 및 배포하기

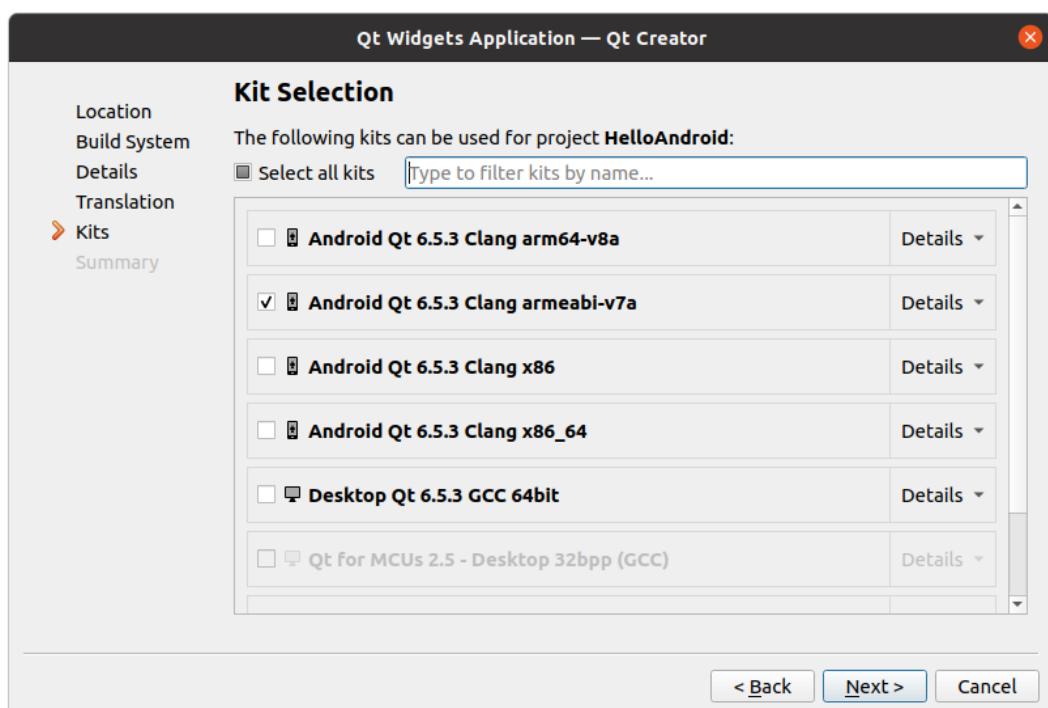
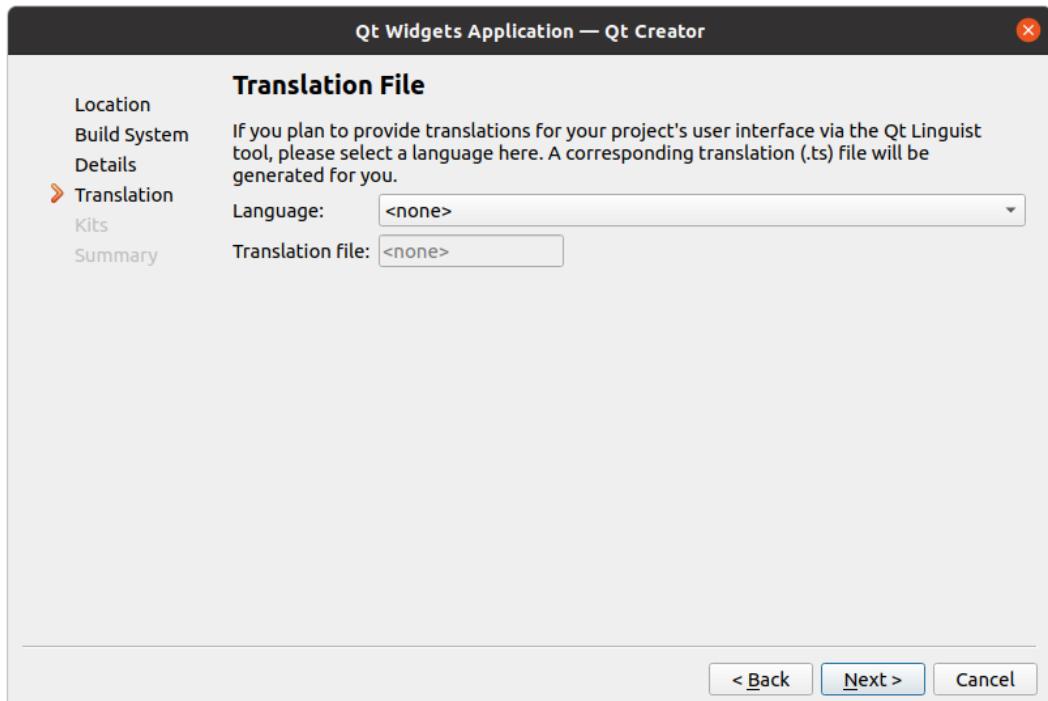
이번에는 간단한 App을 만들어보고 실제 Android Device 에서 실행해 보도록 하자. 프로젝트 생성 시 [Qt Widget Application]을 선택하고 [Chooses] 버튼을 클릭한다.

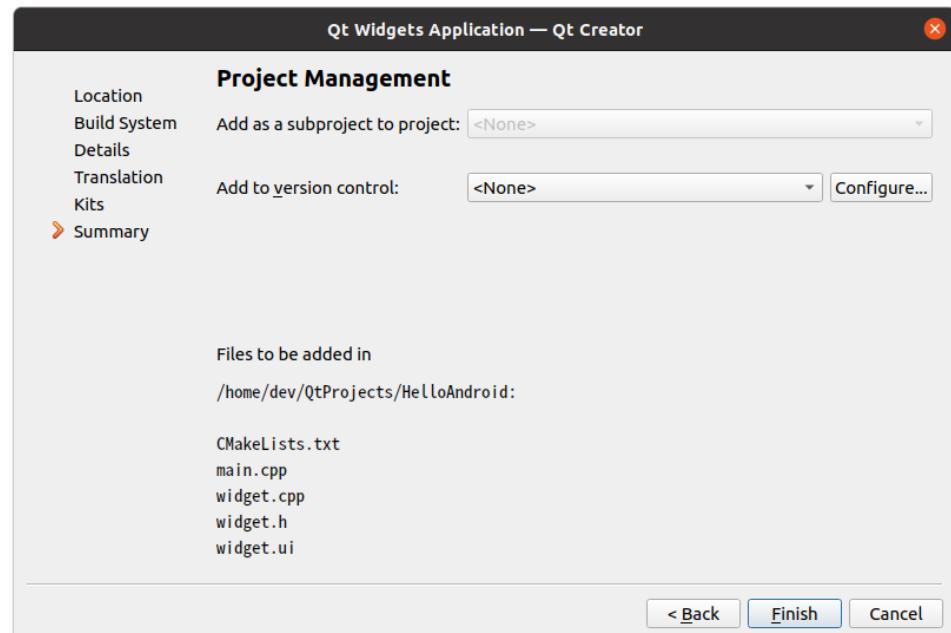


예수님은 당신을 사랑합니다.

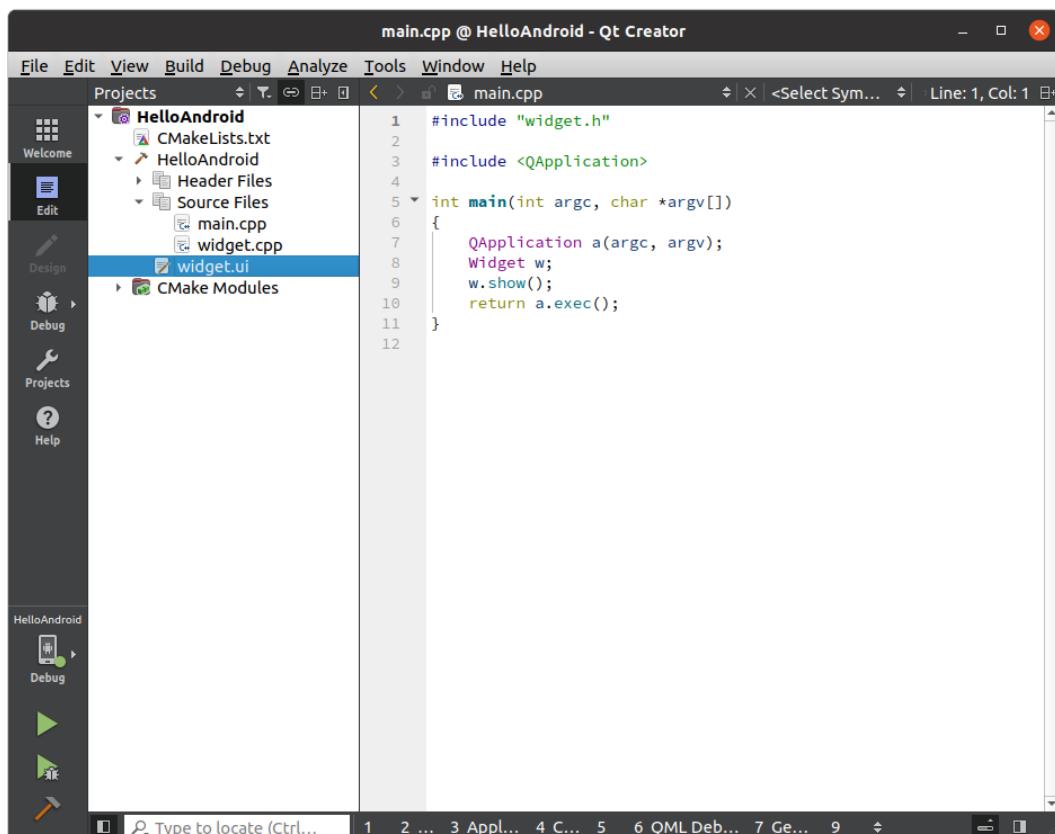


예수님은 당신을 사랑합니다.



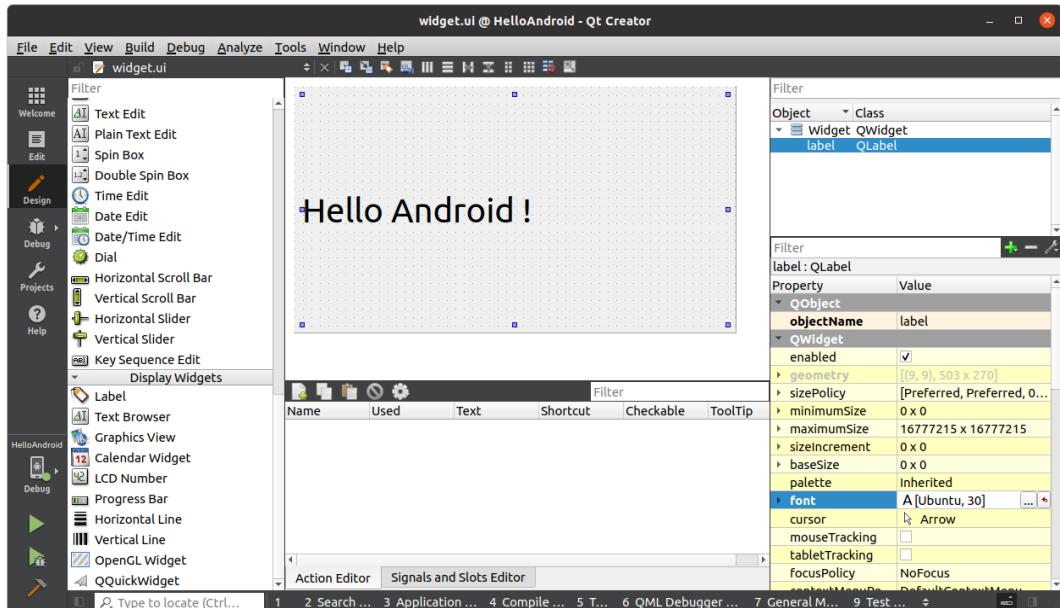


아래 그림에서 보는 것과 같이 프로젝트 생성이 완료되면 widget.ui 파일을 더블 클릭 한다.

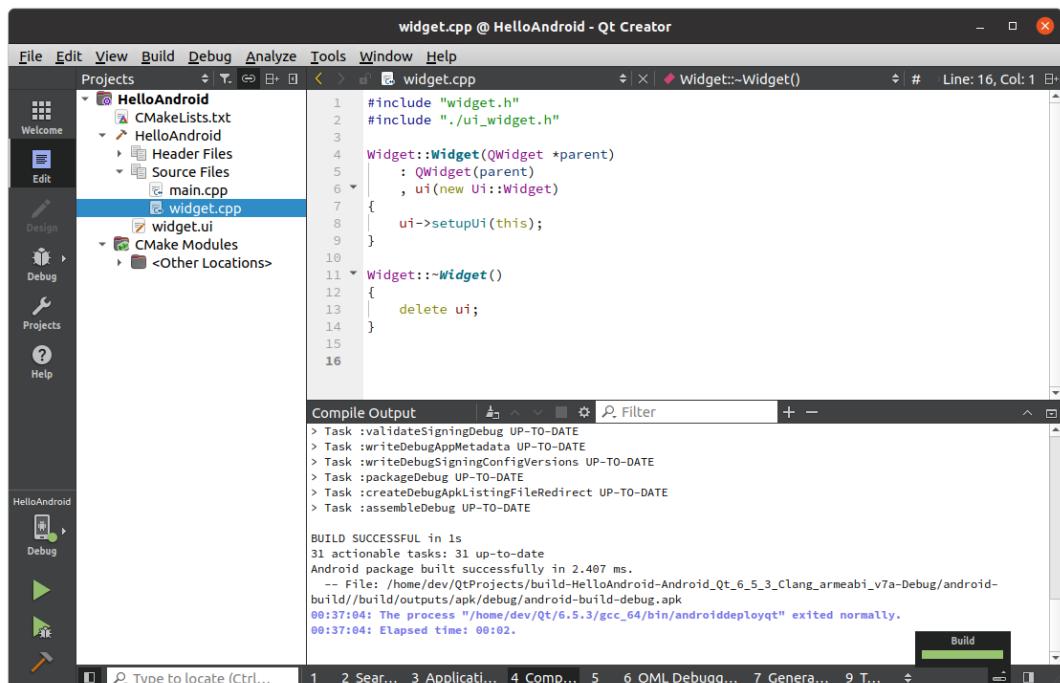


예수님은 당신을 사랑합니다.

GUI 상에 QLabel 을 배치하는 후 "Hello Android!"를 입력한다. 그리고 폰트 크기를 30 으로 변경한다.

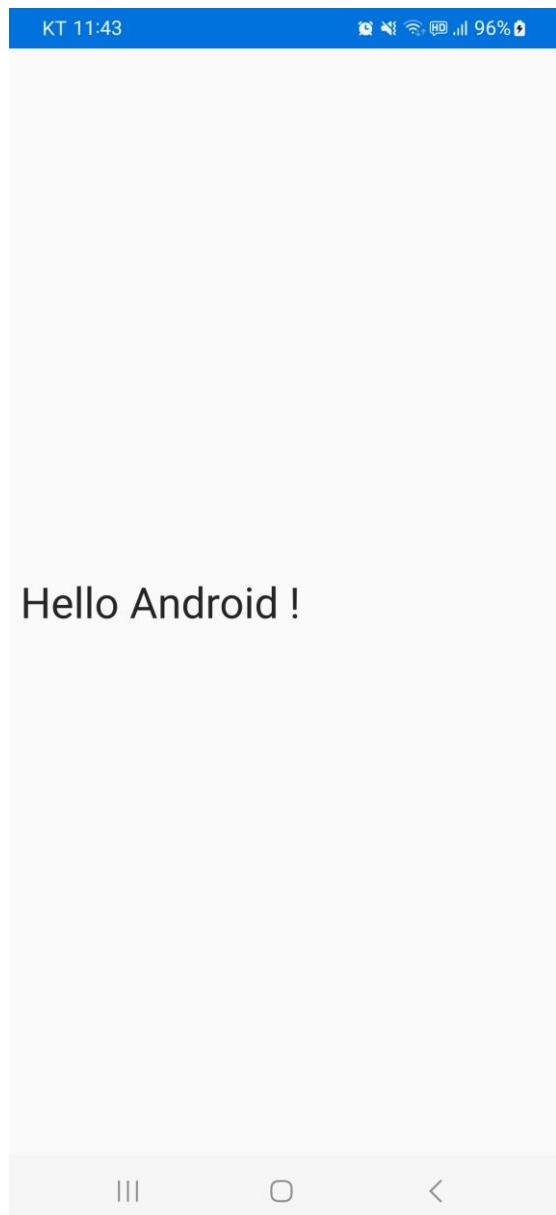


위의 그림에서 보는 것과 같이 수정 후 빌드한다.



빌드가 끝나면 실행해 보도록 하자. 실행 하면 아래 그림에서 보는 것과 같이 실제 Android Device 에서 실행되는 것을 확인할 수 있을 것이다.

예수님은 당신을 사랑합니다.



28. Essential Network Programming

Qt에 제공하는 네트워크 모듈을 이용해 응용 어플리케이션을 개발하는 방법은 C++에서 제공하는 네트워크 라이브러리를 사용하는 것보다 쉽고 빠르게 구현이 가능하다.

가장 큰 이유는 Qt에서는 네트워크 프로그래밍이 쉬운 이유는 Signal/Slot 기반으로 네트워크 기반 응용 어플리케이션을 구현할 수 있기 때문이다.

예를 들어 기존의 C++ 기반 채팅 서버를 구현한다고 가정해 보도록 하자. 구현해야 하는 기능 중 클라이언트의 메시지를 처리하는 부분을 한번 생각해 보자.

여러 명의 클라이언트들이 서버에 접속되어 있는 상황에서 특정 클라이언트가 메시지를 보내면, 그 메시지를 서버에 접속한 클라이언트들에게 모두 전송해야 한다. 이 기능을 C++에서 제공하는 표준 네트워크 라이브러리를 사용해 구현해보자.

먼저 클라이언트별로 Thread가 동작되고 있어야 한다. 10명의 클라이언트가 접속되어 있으면 10개의 Thread가 생성되고 동작되어야 한다.

하지만 Qt를 이용한다면 이 기능을 쉽게 구현할 수 있다. 예를 들어 특정 클라이언트가 메시지를 보냈다면 해당 클라이언트가 보내는 Signal을 특정 Slot 함수와 연결해주면 된다. 즉 Thread 대신, Signal과 Slot 만 연결해주면 된다.

아래 예제 소스코드는 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 소스코드는 QTcpSocket 클래스의 오브젝트를 선언한다. connect() 함수를 이

예수님은 당신을 사랑합니다.

용해 새로운 클라이언트가 서버에 접속하면 connected() Signal을 발생한다. 이 Signal은 connect() 함수를 이용해 slotConnected() Slot 함수가 호출된다.

즉 Qt에서는 새로운 접속자를 처리하기 위한 Thread 구현 없이 Slot 함수만 구현하면 되기 때문에 쉽게 구현할 수 있다.

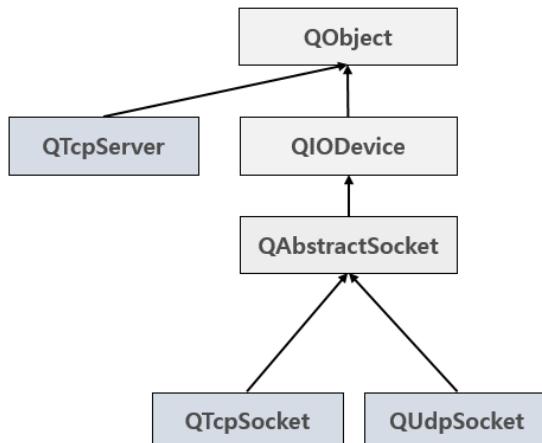
Qt 네트워크 모듈은 TCP/UDP 프로토콜 기반 서버 또는 클라이언트 구현을 위한 다양한 API를 제공한다.

TCP/UDP 프로토콜 기반 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 – UDP 프로토콜 기반의 네트워크 클래스

LOW 레벨 클래스들은 개발자가 세부적인 네트워크 기능을 구현에 적합하다.



<그림> LOW 레벨 주요 네트워크 클래스 다이어그램

QTcpSocket 클래스와 QUdpSocket 클래스는 QAbstractSocket 클래스로부터 상속받아 구현되었다. TCP/UDP 프로토콜의 기반이 아닌 새로운 네트워크 프로토콜을 구현하기 위해서 QAbstractSocket 클래스를 상속 받아 구현한다면 많은 시간을 절약할 수 있다.

QTcpServer는 QTcpSocket과 동일한 TCP 프로토콜 기반의 클래스이다. 차이점은 QTcpServer 클래스는 서버 기반의 응용 어플리케이션 구현에 필요한 기능을 제공한다.

예수님은 당신을 사랑합니다.

Qt에서 제공하는 네트워크 모듈을 사용하기 위해서는 프로젝트파일 상에 다음과 같이 네트워크 모듈을 사용하겠다고 명시해야 한다.

CMake를 사용하는 경우 다음과 같은 항목을 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS Network)
target_link_libraries(mytarget PRIVATE Qt6::Network)
```

qmake 를 사용한다면 다음과 같은 항목을 추가해야 한다.

```
QT += network
```

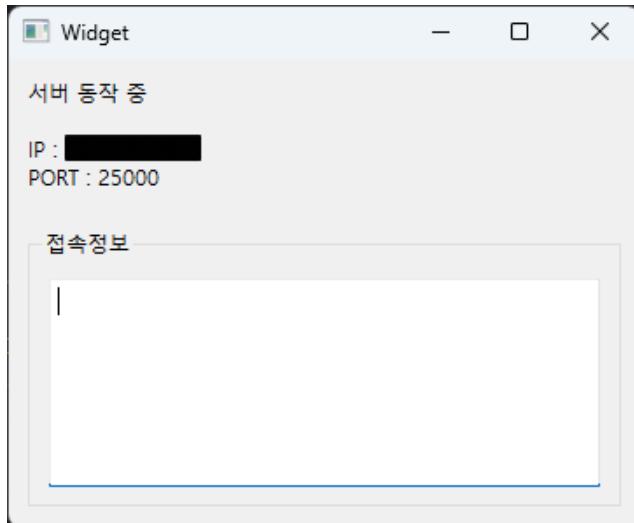
지금까지 Qt에서 제공하는 Network 에 대한 전반적이 내용을 알아보았다. 다음 소 단원에서부터는 구현에 필요한 내용을 자세히 알아보도록 하자.

28.1. TCP 프로토콜 기반 서버/클라이언트 구현

이번 장에서 QTcpServer 클래스와 QTcpSocket 클래스를 이용해 서버/클라이언트 예제를 구현해 보도록 하자.

- QTcpServer 클래스를 이용한 TCP서버 예제

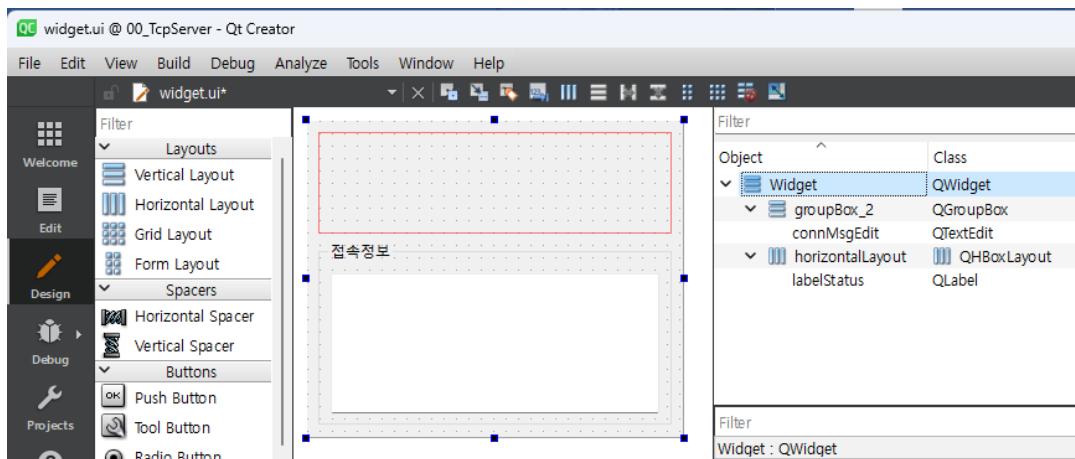
이번 예제는 QTcpServer 클래스를 이용해 서버와 클라이언트 어플리케이션을 구현해 보도록 하자.



위의 그림에서 보는 것과 같이 GUI상에 위젯을 배치해 보도록 하자. 이 예제 소스코드는 00_TcpServer 디렉토리를 참조하면 된다.

CMake 기반 프로젝트 생성 후, CMakeLists.txt 파일에 아래와 같이 항목을 수정 및 추가한다.

```
# 아래와 같이 Network 추가
find_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS Widgets Network)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Widgets Network)
...
# 아래 라인 추가
target_link_libraries(00_TcpServer PRIVATE Qt${QT_VERSION_MAJOR}::Network)
```



가장 상단에 QLabel 위젯을 배치하고 하단의 그룹 박스에는 클라이언트가 접속한 시간을 출력하는 텍스트로 출력할 수 있는 QTextEdit를 배치한다.

다음은 Widget 클래스의 헤더 파일에 다음과 같이 소스코드를 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QObject>
#include <QWidget>

#include <QtNetwork/QTcpServer>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
    QTcpServer *tcpServer;

    void initialize();
}
```

예수님은 당신을 사랑합니다.

```
private slots:  
    void newConnection();  
  
};  
#endif // WIDGET_H
```

위의 헤더 소스코드에서 initialize() 함수는 QTcpServer 클래스의 오브젝트를 초기화를 위한 함수이다. newConnection() Slot 함수는 클라이언트가 접속하면 Signal 이 발생하면 호출되는 함수이다. 다음은 widget.cpp 소스코드이다.

```
#include "widget.h"  
#include "ui_widget.h"  
#include "widget.h"  
#include "./ui_widget.h"  
  
#include <QtNetwork/QNetworkInterface>  
#include <QtNetwork/QTcpSocket>  
#include <QMessageBox>  
#include <QTime>  
  
Widget::Widget(QWidget *parent)  
    : QWidget(parent)  
    , ui(new Ui::Widget)  
{  
    ui->setupUi(this);  
    initialize();  
}  
  
void Widget::initialize()  
{  
    QHostAddress hostAddr;  
    QList<QHostAddress> ipList = QNetworkInterface::allAddresses();  
  
    // localhost(127.0.0.1) 가 아닌 것을 사용  
    for (int i = 0; i < ipList.size(); ++i) {  
        if (ipList.at(i) != QHostAddress::LocalHost &&  
            ipList.at(i).toIPv4Address()) {  
            hostAddr = ipList.at(i);  
            break;  
        }  
    }  
}
```

```
if (hostAddr.toString().isEmpty())
    hostAddr = QHostAddress(QHostAddress::LocalHost);

tcpServer = new QTcpServer(this);
if (!tcpServer->listen(hostAddr, 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(hostAddr.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;
}
```

예수님은 당신을 사랑합니다.

생성자 함수에서 호출되는 initialize() 함수는 클라이언트가 접속할 서버의 IP를 시스템으로부터 얻어온다. 그리고 QTcpServer 클래스의 tcpServer 오브젝트를 초기화 하고 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.

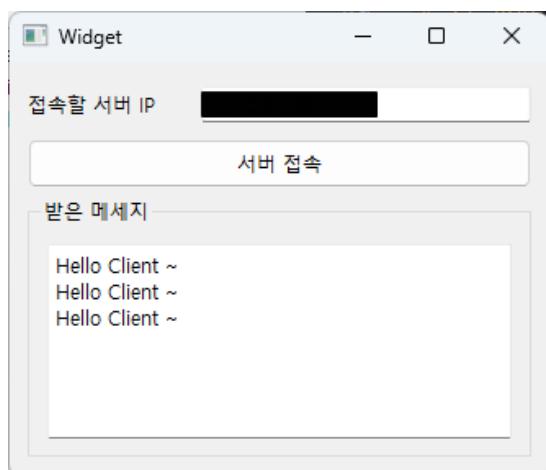
newConnection() 함수는 GUI 상에 QTextEdit 위젯에 클라이언트가 접속한 시간을 출력하고 접속한 클라이언트에게 “Hello Client ~” 메시지를 전송한다.

● QTcpSocket 클래스를 이용한 TCP 클라이언트 예제

CMake 기반 프로젝트 생성 후, CMakeLists.txt 파일에 아래와 같이 항목을 수정 및 추가한다.

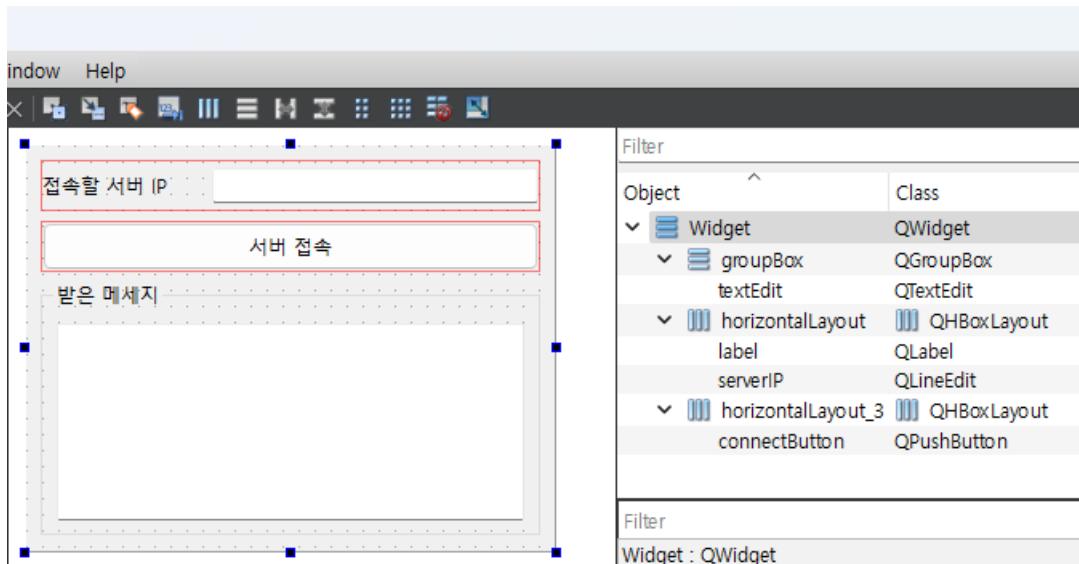
```
# 아래와 같이 Network 추가
find_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS Widgets Network)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Widgets Network)
...
# 아래 라인 추가
target_link_libraries(00_TcpServer PRIVATE Qt${QT_VERSION_MAJOR}::Network)
```

다음은 클라이언트 예제이다. QWidget 을 상속받는 프로젝트를 생성하고 다음 그림에서 보는 것과 같이 GUI상에 위젯을 배치한다. 클라이언트의 전체 예제 소스코드는 00_TcpClient 디렉토리를 참조하면 된다.



예수님은 당신을 사랑합니다.

위의 그림에서 보는 것과 같이 접속 할 서버의 IP와 PORT 를 입력할 수 있도록 QLineEdit 위젯을 배치한다. 그리고 [접속] 버튼과 하단의 QTextEdit 를 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 <QtNetwork/QTcpSocket>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
```

예수님은 당신을 사랑합니다.

```
QTcpSocket *tcpSocket;

void initialize();

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"

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()
```

예수님은 당신을 사랑합니다.

```
{  
    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.

28.2. 동기 방식 비 동기 방식 구현

네트워크에서 데이터 송/수신 처리 방식을 크게 두가지로 나누어 볼 수 있다. 첫 번째는 동기 방식이고 두 번째 방식은 비 동기 방식이다.

동기 방식은 서버가 클라이언트에게 Request 했을 때, 다른 처리는 하지 않고 응답을 기다리는 경우이다. 이러한 경우를 동기 방식이라 한다.

```
...
int writeBytes = socket->write("Hello server\r\n\r\n");
socket->waitForReadyRead(3000);
...
```

위의 소스코드에서 보는 것과 같이 write() 함수를 이용해 메시지를 전송하고 상대방으로부터 메시지를 받을 때까지 기다리기 위해서 waitForReadyRead() 함수를 사용할 수 있다.

비 동기 방식이란 서버가 클라이언트에게 Request를 하였지만 언제 응답을 받을지 모르는 상황을 비 동기 방식으로 처리할 수 있다.

```
QHttpSocket::QHttpSocket(QObject *parent) : QObject(parent)
{
    socket = new QTcpSocket(this);
    ...
    connect(socket, SIGNAL(readyRead()), this, SLOT(slotRead()));
}

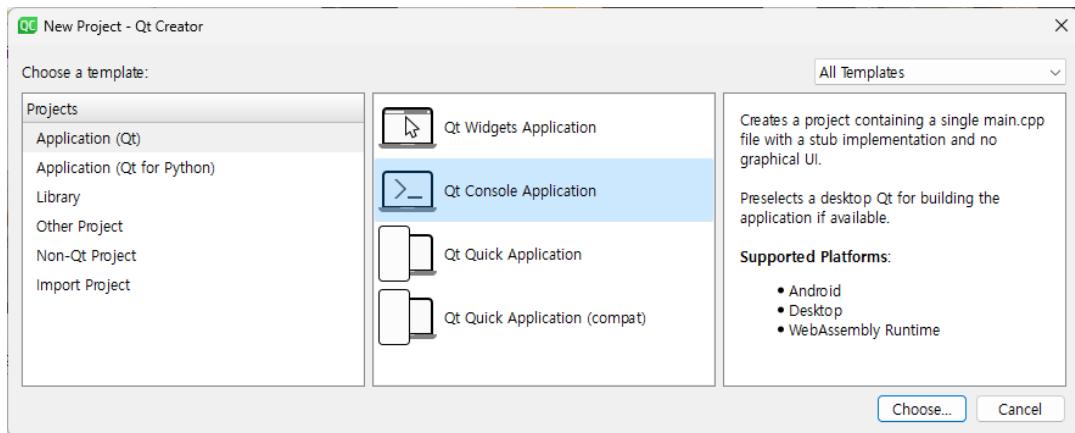
void QHttpSocket::slotRead ()
{
    qDebug() << socket->readAll();
}
```

위의 예에서 보는 것과 같이 상대방으로부터 메시지를 수신 받을 때 slotRead() Slot 함수가 호출 된다. 이런 방식을 비 동기 방식이라고 한다.

우리는 이번 장에서 우리는 qt-dev.com 웹 서버를 이용해 동기 방식과 비 동기 방식을 이용해 데이터를 송/수신 하는 방법에 대해서 알아보도록 하자.

● 동기 방식 예제

이번 예제는 GUI를 사용하지 않고 콘솔 기반의 프로젝트를 생성한다.



생성이 완료되면 CMakeList.txt 파일에 아래와 같이 Network 모듈을 추가한다.

```
cmake_minimum_required(VERSION 3.14)

project(00_Sync LANGUAGES CXX)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt NAMES Qt6 Qt5 REQUIRED COMPONENTS Core Network)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Core Network)

add_executable(00_Sync
    main.cpp
    qhttpsocket.h qhttpsocket.cpp
)

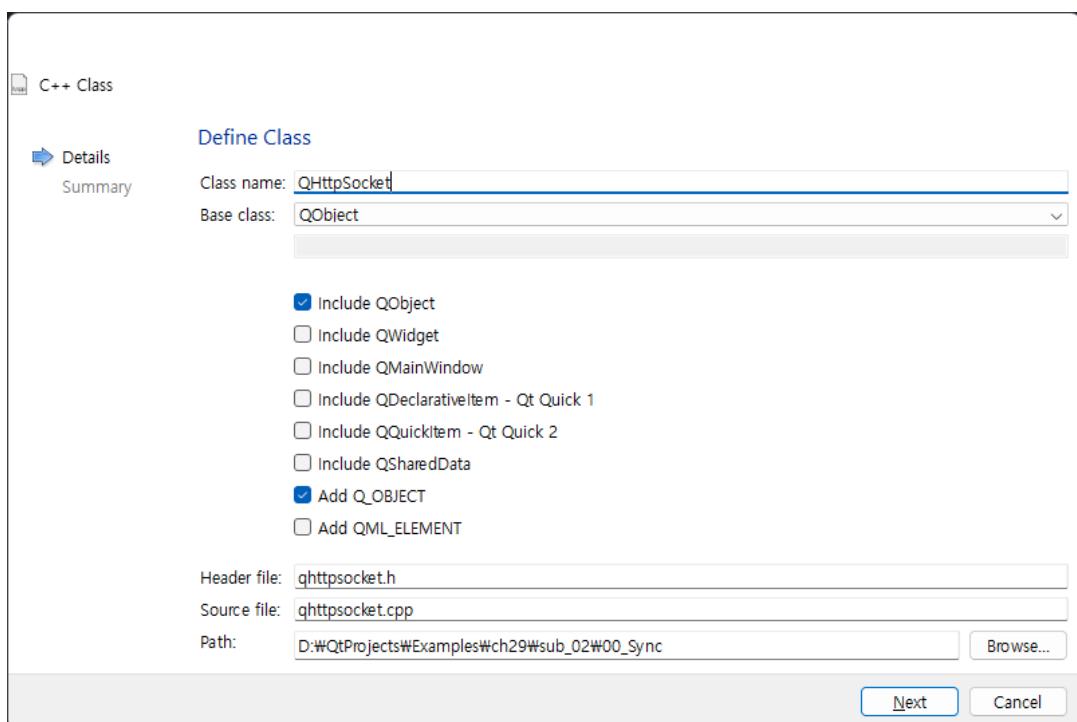
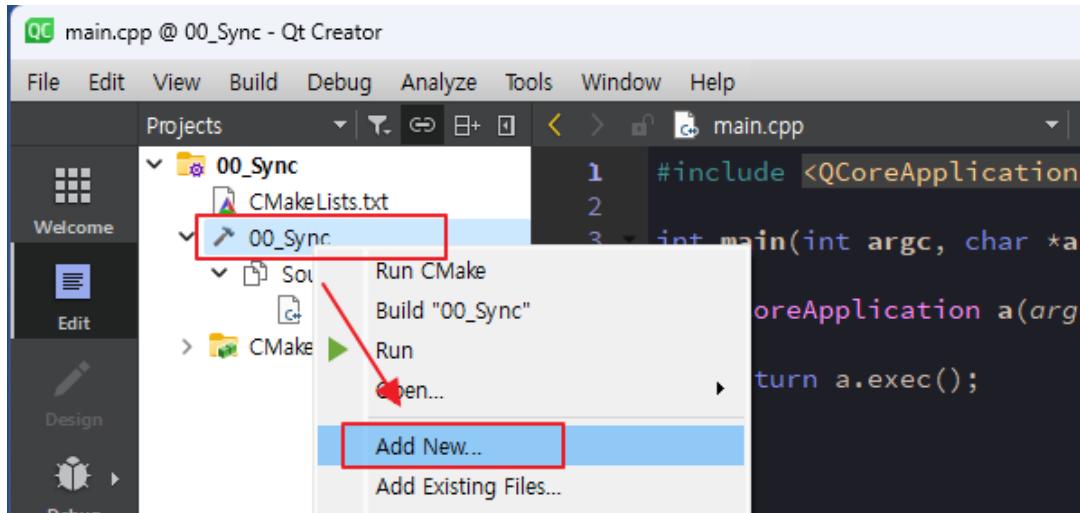
target_link_libraries(00_Sync Qt${QT_VERSION_MAJOR}::Core)
target_link_libraries(00_Sync Qt${QT_VERSION_MAJOR}::Network)

include(GNUInstallDirs)
install(TARGETS 00_Sync
```

예수님은 당신을 사랑합니다.

```
LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

다음으로 QObject 클래스를 상속받는 QHttpSocket 라는 이름의 클래스를 생성하자.



클래스의 헤더에는 아래와 같이 작성한다.

```
#ifndef QHTTPSOCKET_H
#define QHTTPSOCKET_H

#include <QObject>
#include <QTcpSocket>

class QHttpSocket : public QObject
{
    Q_OBJECT
public:
    explicit QHttpSocket(QObject *parent = nullptr);
    ~QHttpSocket();
    void httpConnect();

signals:

public slots:
    void httpDisconnected();

private:
    QTcpSocket *socket;

};

#endif // QHTTPSOCKET_H
```

다음은 qhttpsocket.cpp 소스코드를 아래와 같이 작성한다.

```
#include "qhttpsocket.h"

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))
    {
        qDebug() << "TCP Connected.";
        // send
        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 QTcpSocket 클래스에서 제공하는 connectToHost() 함수는 첫 번째 인자에 명시한 URL에 접속한다. 그리고 두 번째 인자는 접속할 Port 번호를 명시 한다.

connectToHost() 함수 다음 라인에서 사용한 waitForConnected() 함수는 웹 서버와 연결 될 때 까지 기다린다. 첫 번째 인자는 서버가 응답할 때까지 기다리는 시간이다.

다음으로 상대방으로부터 데이터 수신할 때 까지 기다리기 위해서 waitForReadyRead() 함수를 사용할 수 있다. 이러한 방식으로 동기 방식으로 소스코드를 구현할 수 있다.

다음은 main.cpp 에서 QHttpSocket 클래스의 오브젝트를 생성하고 httpConnect() 멤버 함수를 하도록 아래와 같이 작성해 보도록 하자.

```
#include <QCoreApplication>
#include "qhttpsocket.h"
```

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

    QHttpSocket *httpSocket = new QHttpSocket();
    httpSocket->httpConnect();

    return a.exec();
}
```

이 예제 소스코드는 00_Sync 디렉토리를 참조하면 된다.

● 비 동기 방식 예제

이번 예제는 Signal/Slot을 이용한 비 동기 방식 예제 소스코드이다. 이전 예제와 같이 Console 기반의 프로젝트를 생성하고 QHttpSocket 클래스를 추가한다. 그런 다음에 qhttpsocket.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef QHTTPSOCKET_H
#define QHTTPSOCKET_H

#include <QObject>
#include <QTcpSocket>

class QHttpSocket : public QObject
{
    Q_OBJECT
public:
    explicit QHttpSocket(QObject *parent = nullptr);
    ~QHttpSocket();

    void httpConnect();

signals:

public slots:
    void slotConnected();
    void slotDisconnected();
    void slotBytesWritten(qint64 bytes);
    void slotReadPendingDatagram();
```

예수님은 당신을 사랑합니다.

```
private:  
    QTcpSocket *socket;  
  
};  
  
#endif // QHTTPSOCKET_H
```

다음 예제 소스코드는 QHttpSocket 클래스의 구현 소스코드이다.

```
#include "qhttpsocket.h"  
  
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(slotReadPendingDatagram()));  
}  
  
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);  
}
```

예수님은 당신을 사랑합니다.

```
socket->write("HEAD / HTTP/1.0\r\n\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. 이전 동기 방식과 다르게 이번 예제에서는 Signal 과 Slot을 이용해 비 동기 방식을 사용할 수 있다. 이번 예제 소스코드는 01_Async 디렉토리를 참조하면 된다.

28.3. UDP 프로토콜 기반 네트워크 통신 구현

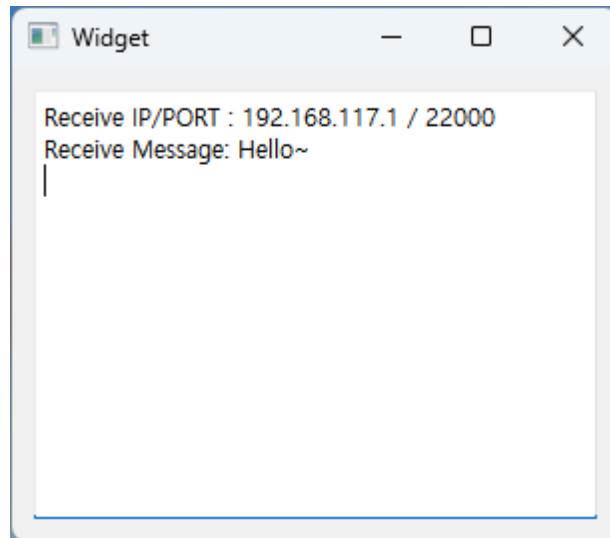
UDP 프로토콜은 비 신뢰성 이므로 TCP 와 같이 메시지를 송/수신 하기 전에 서버와 클라이언트가 연결을 맺어야 하지만 UDP 는 Connection을 맺지 않고 양자간 메시지를 송/수신 할 수 있다.

Qt에서는 UDP 프로토콜 기반 어플리케이션을 구현할 수 있도록 QUdpSocket 클래스를 제공한다.

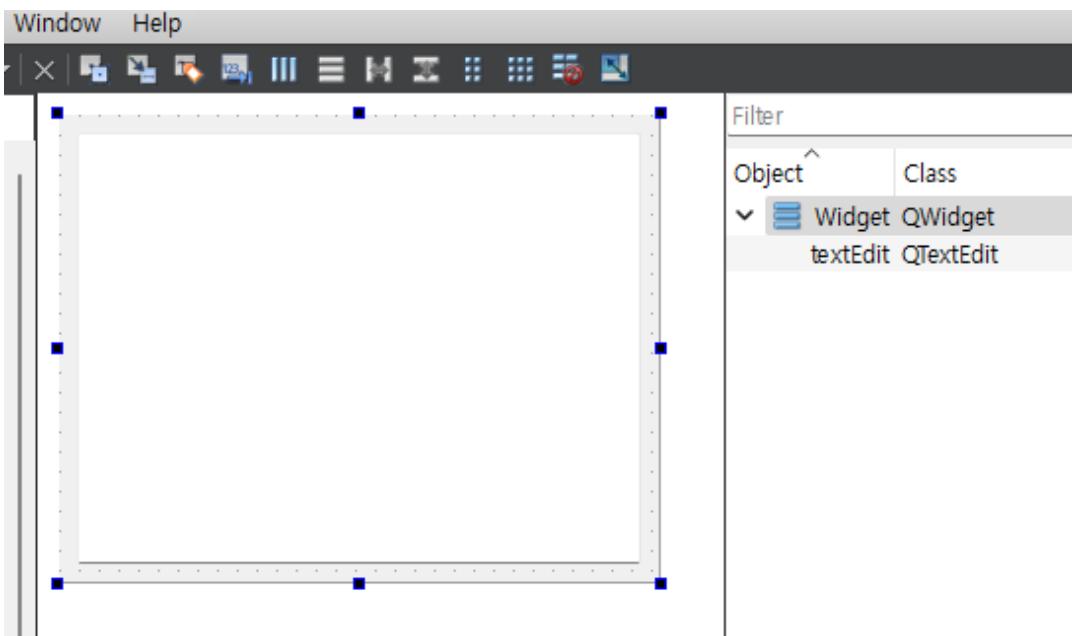
따라서 이번 장에서는 QUdpSocket 클래스를 이용해 서버/클라이언트 예제를 구현해 보도록 하자.

- QUdpSocket 클래스를 이용한 서버 구현

이번 예제는 아래 그림에서 보는 것과 같이 UDP 프로토콜 기반 메시지를 수신하면 QTextEdit 위젯 상에 상대방 Network 정보와 메시지를 출력한다. 그리고 상대방에게 “Hello UDP Client~” 라는 메시지를 전송한다.



프로젝트 생성 시 Qt Widget 클래스 기반으로 프로젝트를 생성한다. widget.ui 상에 아래와 같이 QTextEdit 위젯을 배치한다.



그리고 widget.h 헤더 파일에 아래와 같이 소스코드를 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QUdpSocket>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
    QUdpSocket *udpSocket;

private slots:
    void readPendingDatagram();
}
```

```
};

#endif // WIDGET_H
```

위와 같이 작성 하였다면 widget.cpp 소스 파일에 아래와 같이 작성한다.

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

#define SERVER_PORT 21000

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

    udpSocket = new QUdpSocket(this);
    udpSocket->bind(QHostAddress("192.168.117.1"), 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());
```

예수님은 당신을 사랑합니다.

```
ui->textEdit->append(msg);

QByteArray writeData;
writeData.append("SERVER : Hello UDP Client.~ ");
udpSocket->writeDatagram(writeData, sender, senderPort);

}

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

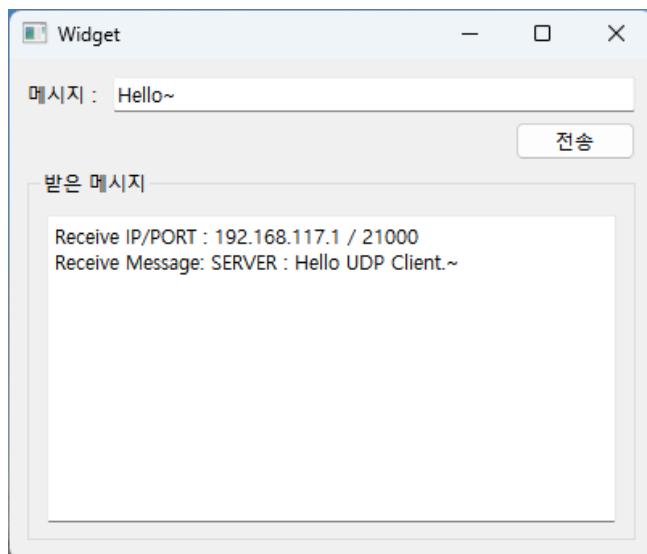
생성자 함수에서 QUdpSocket 클래스에서 제공하는 bind() 함수를 이용해 IP와 PORT를 설정한다. 그리고 connect() 함수에서 메시지를 수신 했을 때, readyRead() 시그널이 발생한다. 이 시그널은 readPendingDatagram() Slot 함수를 호출한다.

readPendingDatagram() Slot 함수에서 pendingDatagramSize() 멤버 함수는 수신 메시지의 크기를 리턴 한다. 그리고 수신 받은 메시지를 QTextEdit 위젯에 출력하고 수신 받은 클라이언트에게 다시 메시지를 전송한다.

이 예제의 소스코드는 00_UDP_Server 디렉토리를 참조하면 된다.

- QUdpSocket 클래스를 이용한 클라이언트 구현

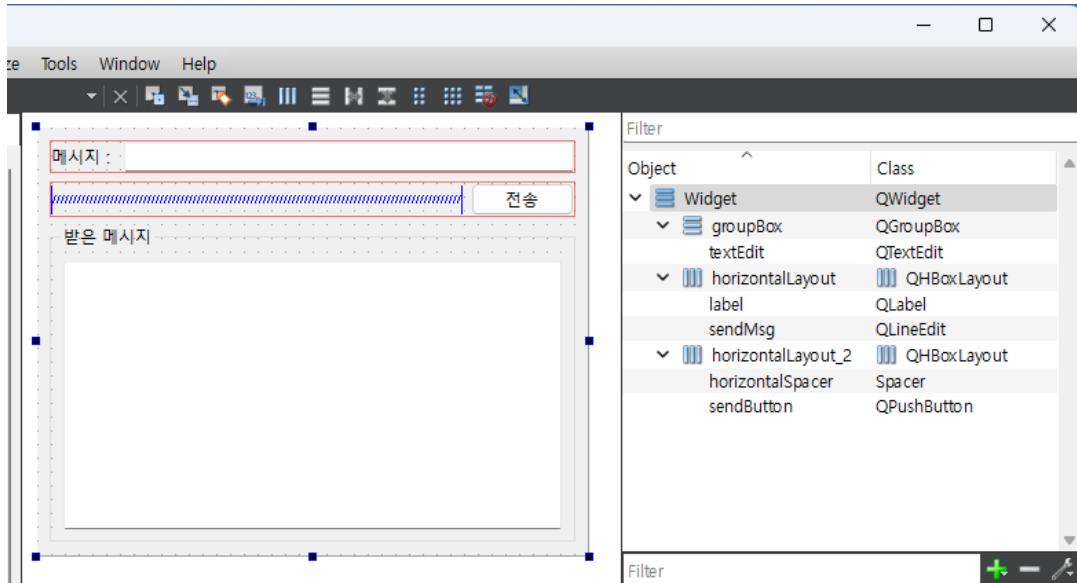
다음 예제는 클라이언트 구현 예제이다.



예수님은 당신을 사랑합니다.

위의 그림에서 보는 것과 같이 UDP 서버로 보낼 메시지를 입력하고 [전송] 버튼을 누르면 하단에 서버로부터 “Hello UDP Client~” 메시지를 수신하는 예제이다.

프로젝트 생성 시 Qt Widget 기반으로 프로젝트를 생성한다. 그리고 아래 그림에서 보는 것과 같이 GUI상에 위젯을 배치한다.



위와 같이 위젯을 배치하고 widget.h 헤더 파일에 아래와 같이 소스코드를 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QUdpSocket>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
```

예수님은 당신을 사랑합니다.

```
Ui::Widget *ui;
Q_udpSocket *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"

#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 Q_udpSocket(this);
    udpSocket->bind(QHostAddress("192.168.117.1"), CLIENT_PORT);

    connect(udpSocket, SIGNAL(readyRead()),
            this,           SLOT(readPendingDatagram()));
}

void Widget::sendButton()
{
    QByteArray msg;
    msg = ui->sendMsg->text().toLocal8Bit();

    QHostAddress sender("192.168.117.1");
```

예수님은 당신을 사랑합니다.

```
    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;
}
```

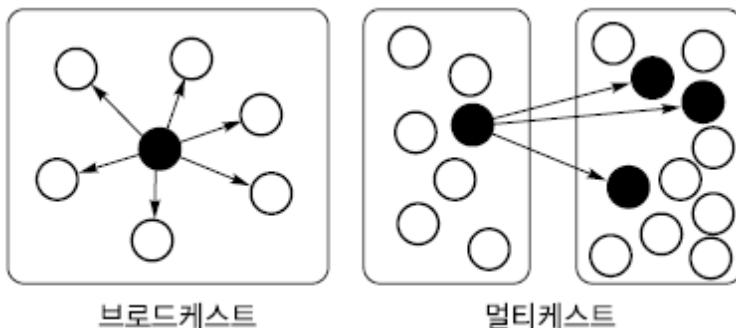
readPendingDatagram() Slot 함수는 서버로부터 메시지를 수신 받으면 QTextEdit 위젯에 수신 받은 메시지를 출력한다. 이 예제의 소스코드는 00_UDP_Client 디렉토리를 참조하면 된다.

28.4. Broadcast

컴퓨터 네트워크 상에서 UDP 기반 프로토콜을 이용해 송신자와 수신자 관점에서 나누어보면 브로드캐스트와 멀티캐스트로 방식으로 데이터 송/수신이 가능하다.

브로드캐스트 방식은 하나의 송신자가 모든 수신자에게 메시지를 전달할 수 있는 방식이다. 예를 들어 송신자가 특정 PORT로 데이터를 송신하고 있을 때 수신자는 송신자의 동일한 PORT로 수신하는 모든 수신자를 메시지를 수신할 수 있다.

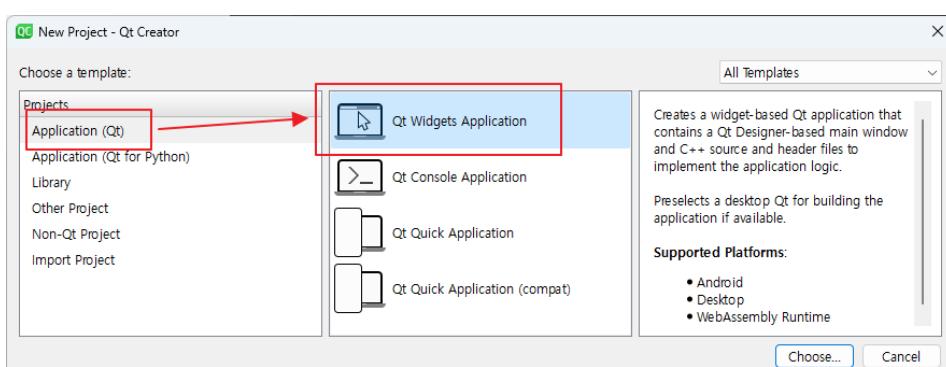
브로드캐스트 방식으로 송신자가 데이터를 송신하면 해당 네트워크 그룹에 포함된 수신자는 모두 수신할 수 있다.



멀티캐스트는 지정한 사용자만 메시지를 송/수신할 수 있는 방식이다. 따라서 이번 장에서는 브로드캐스트 방식과 멀티캐스트 방식에 대해서 예제를 통해 자세히 다루어 보도록 하겠다.

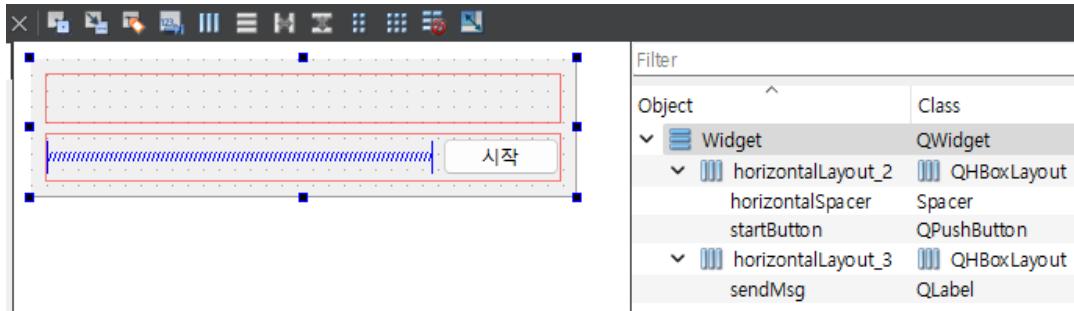
- 브로드캐스트 방식을 이용한 데이터 송신 예제

Qt Widget 기반 프로젝트를 생성한다.



예수님은 당신을 사랑합니다.

프로젝트 생성 후 widget.ui 파일을 더블 클릭해 아래와 같이 GUI상에 위젯을 배치한다.



시작 버튼을 클릭하면 네트워크 그룹 상에 모든 수신자들은 메시지를 수신할 수 있다.
다음으로 widget.h 헤더 파일에 아래와 같이 소스코드를 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QUdpSocket>
#include <QTimer>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

    QUdpSocket *udpSocket;
    QTimer     *timer;
    int         msgNumber;

private slots:
    void startButton();
}
```

예수님은 당신을 사랑합니다.

```
void broadcastSend();  
};  
#endif // WIDGET_H
```

QTimer 클래스의 timer 오브젝트는 지정한 시간을 반복해 브로드캐스트 메시지를 전송 한다. 다음은 widget.cpp 소스코드이다.

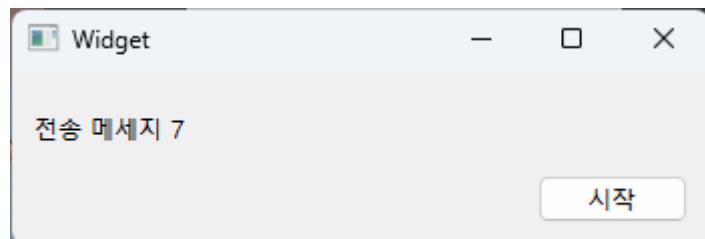
```
#include "widget.h"  
#include "./ui_widget.h"  
  
Widget::Widget(QWidget *parent)  
    : QWidget(parent)  
    , ui(new Ui::Widget)  
{  
    ui->setupUi(this);  
  
    timer = new QTimer(this);  
    udpSocket = new QD udpSocket(this);  
    msgNumber = 1;  
  
    connect(ui->startButton, SIGNAL(clicked()),  
            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.

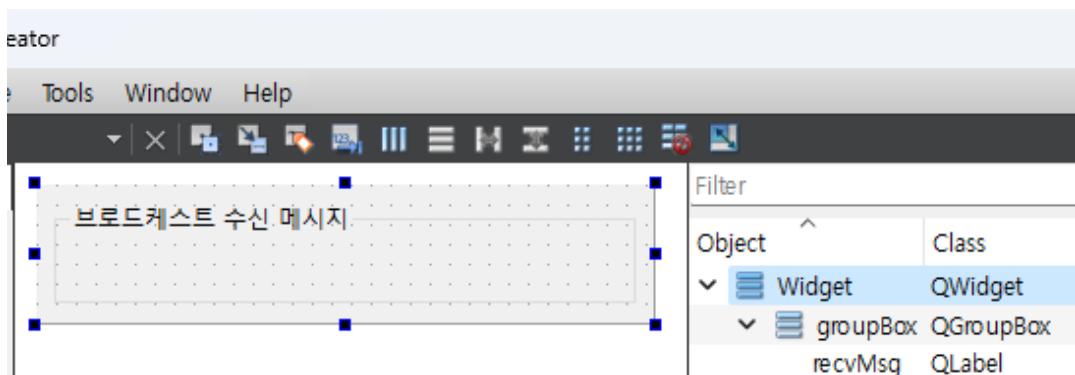
QHostAddress::Broadcast 값을 3번째 인자로 사용하면 네트워크 그룹 내에 모든 컴퓨터에게 메시지를 전송한다.



이 예제의 소스코드는 00_Broadcast_Sender 디렉토리를 참조한다.

- 브로드캐스트 수신 예제

다음 예제는 브로드캐스트 메시지를 수신 받는 예제이다. Qt Widget 기반 프로젝트 생성 후 아래와 같이 GUI상에 위젯을 배치한다.



그런 다음 widget.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef WIDGET_H
```

예수님은 당신을 사랑합니다.

```
#define WIDGET_H

#include <QWidget>
#include <QUdpSocket>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

private slots:
    void readDatagrams();

};

#endif // WIDGET_H
```

다음으로 widget.cpp 소스 파일을 아래와 같이 작성한다.

```
#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()));
}
```

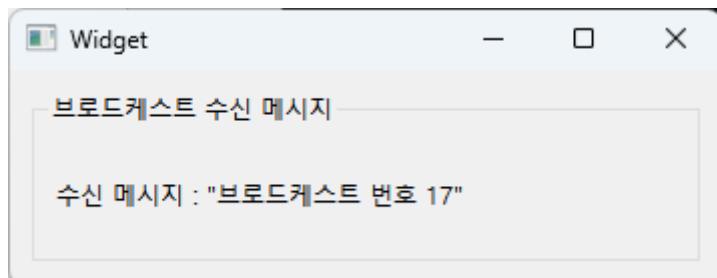
```
void Widget::readDatagrams()
{
    while (udpSocket->hasPendingDatagrams())
    {
        QByteArray datagram;
        datagram.resize(udpSocket->pendingDatagramSize());

        udpSocket->readDatagram(datagram.data(), datagram.size());

        ui->recvMsg->setText(tr("수신 메시지 : \"%1\"").arg(datagram.data()));
    }
}

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

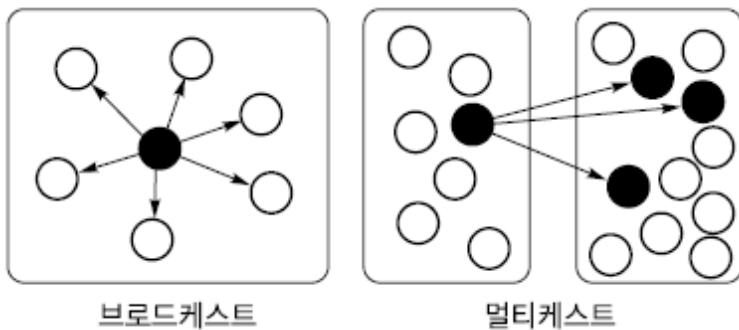
생성자 함수에서 bind() 함수를 사용할 때 송신 하는 예제와 동일한 PORT를 사용함으로써 송신 예제가 보내는 메시지를 수신 받을 수 있다.



이 예제 소스코드는 00_Broadcast_Receiver 디렉토리를 참조하면 된다.

28.5. Multicast

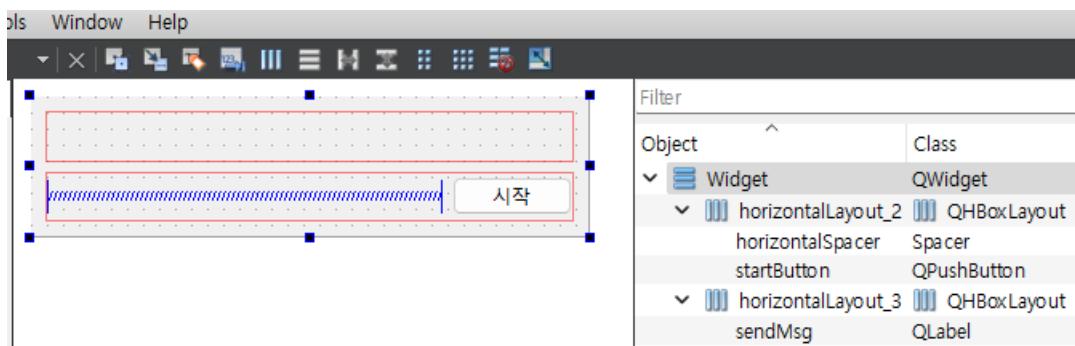
이번 장에서는 Multicast를 이용해 메시지를 송/수신 하는 방법을 예제를 통해 알아 다
루어 보도록 하겠다. Multicast 방식은 특정 컴퓨터(또는 사용자)에게 송/수신 할 수 있
다.



첫 번째 예제로써 Multicast 방식을 이용해 메시지를 전송하는 예제를 먼저 구현해보고
두 번째로 메시지를 수신하는 방법에 대해서 알아보도록 하자.

- Multicast를 이용한 Sender 예제 구현

QWidget 클래스를 상속받는 프로젝트를 생성하고 다음 그림에서 보는 것과 같이 GUI
상에 위젯을 배치한다.



위의 그림에서 보는 것과 같이 [시작] 버튼을 클릭하면 Multicast 방식을 이용해 메시지
를 전송한다. 다음으로 widget.h 헤더 파일을 열어서 아래와 같이 소스코드를 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H
```

```
#include <QWidget>
#include <QUdpSocket>
#include <QTimer>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

public:
    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 multicastSend();
};

#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)
```

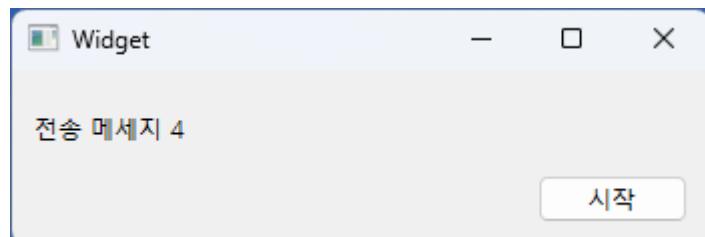
```
{  
    ui->setupUi(this);  
  
    groupAddress4 = QHostAddress(QStringLiteral("239.255.43.21"));  
    groupAddress6 = QHostAddress(QStringLiteral("ff12::2115"));  
  
    timer = new QTimer(this);  
    udpSocket4.bind(QHostAddress(QHostAddress::AnyIPv4), 0);  
    udpSocket6.bind(QHostAddress(QHostAddress::AnyIPv6),  
                    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 메시지 %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.

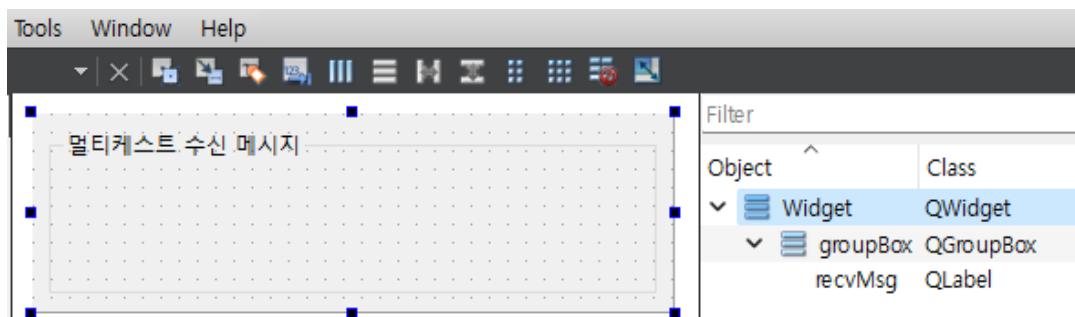
다음으로 빌드 후 실행 보도록 하자.



이 예제 소스코드는 00_Multicast_Sender 디렉토리를 참조하면 된다.

● Multicast를 이용한 Receiver 예제 구현

이번 예제는 이전 예제에서 전송한 메시지를 수신하는 예제를 작성해 보도록 하자. 프로젝트 생성 시, Qt Widget 기반으로 프로젝트를 생성한다. 그런 다음 widget.ui 를 열어서 아래와 같이 GUI상에 위젯들을 배치한다.



위와 같이 작성하고 widget.h 헤더 파일을 아래와 같이 작성한다.

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

예수님은 당신을 사랑합니다.

```
QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

private:
    QUdpSocket udpSocket4;
    QUdpSocket udpSocket6;
    QHostAddress groupAddress4;
    QHostAddress groupAddress6;

private slots:
    void readDatagrams();

};

#endif // WIDGET_H
```

다음은 widget.cpp 소스 파일을 열어서 아래와 같이 소스코드를 작성한다.

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

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,
```

예수님은 당신을 사랑합니다.

```
    QUpdSocket::ShareAddress);
udpSocket4.joinMulticastGroup(groupAddress4);

if (!udpSocket6.bind(QHostAddress::AnyIPv6, 45000,
                     QUpdSocket::ShareAddress) ||
    !udpSocket6.joinMulticastGroup(groupAddress6))
qDebug() << Q_FUNC_INFO << "IPv4 멀티캐스트만 사용 가능.";

connect(&udpSocket6, &QUpdSocket::readyRead,
        this,           &Widget::readDatagrams);
}

void Widget::readDatagrams()
{
    QByteArray datagram;

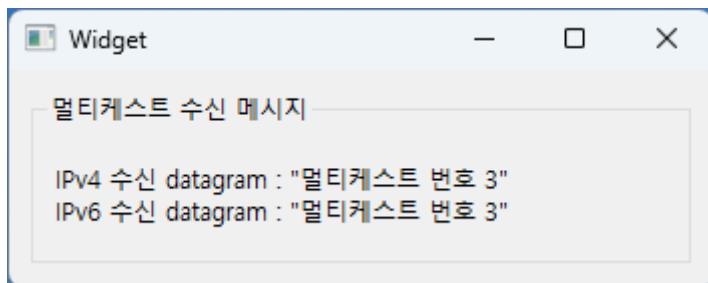
    while (udpSocket4.hasPendingDatagrams())
    {
        datagram.resize(int(udpSocket4.pendingDatagramSize()));
        udpSocket4.readDatagram(datagram.data(), datagram.size());
        ui->recvMsg->setText(tr("IPv4 수신 datagram : \"%1\"")
                               .arg(datagram.constData()));
    }

    while (udpSocket6.hasPendingDatograms())
    {
        QNetworkDatagram dgram = udpSocket6.receiveDatagram();
        ui->recvMsg->setText(ui->recvMsg->text() +
                               tr("\nIPv6 수신 datagram : \"%1\"")
                               .arg(dgram.data().constData()));
    }
}

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

송신 예제에서 보내는 메시지를 받기 위해서 동일한 Address를 사용해야 한다. 동일한 그룹 Address를 사용함으로써 Sender 예제에서 보내는 메시지를 수신할 수 있다. 소스 코드 작성 완료 후, 아래 그림에서 보는 것과 같이 빌드 후 실행 보도록 한다.

예수님은 당신을 사랑합니다.



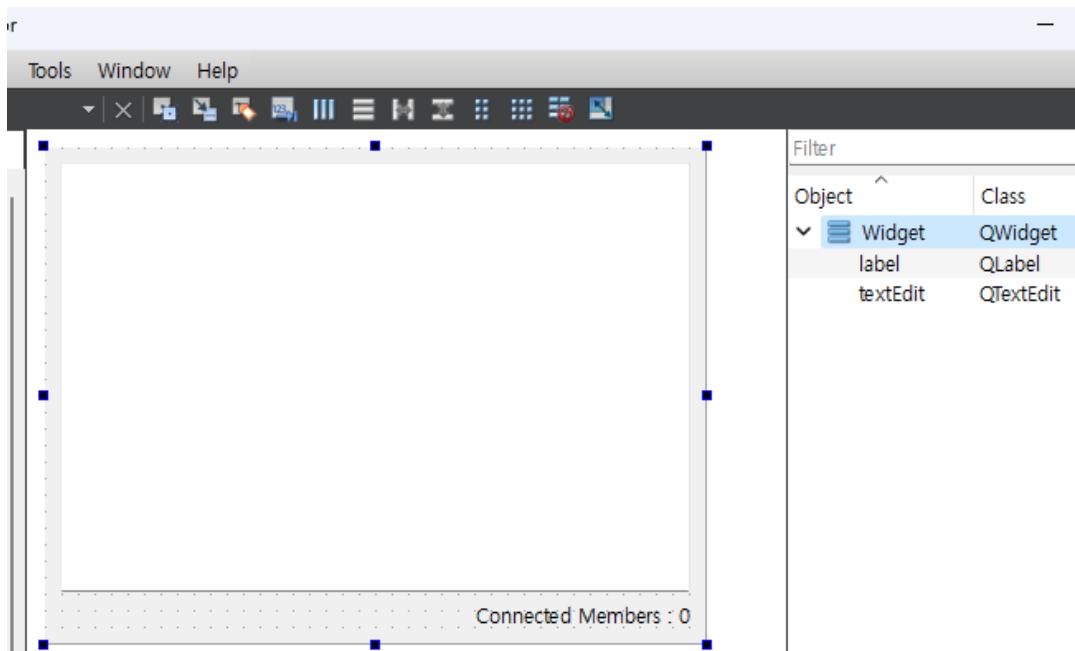
이번 예제의 전체 소스코드는 00_Multicast_Receiver 디렉토리를 참조하면 된다.

28.6. 채팅 서버/클라이언트 구현

이번 장에서는 QTcpServer 와 QTcpSocket 클래스를 이용해 간단한 채팅 서버 예제와 클라이언트 예제를 구현해 보도록 하자.

- 채팅 서버 예제 구현

위의 그림에서 보는 것과 같이 GUI 상에 위젯을 배치한다. GUI 상에서 QTextEdit 위젯은 새로운 사용자가 접속, 사용자들이 보내는 메시지 그리고 접속이 끊긴 사용자 정보를 표시한다.



위와 같이 배치 했으면, 다음으로 ChatServer 클래스를 프로젝트에 추가한다. 그리고 chatserver.h 헤더 파일에 아래와 같이 작성한다.

```
#ifndef CHATSERVER_H
#define CHATSERVER_H

#include <QObject>
#include <QTcpServer>
#include <QTcpSocket>
```

예수님은 당신을 사랑합니다.

```
class ChatServer : public QTcpServer
{
    Q_OBJECT
public:
    ChatServer(QObject *parent = nullptr);
    ~ChatServer();

private slots:
    void readyRead();
    void disconnected();
    void sendUserList();

signals:
    void clients_signal(int users);
    void message_signal(QString msg);

protected:
    void incomingConnection(qintptr 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() Slot 함수가 호출될 수 있도록 이 함수를 시그널과 연결한다. 따라서 클라이언트가 메시지를 보내면 readyRead() Slot 함수가 호출된다.

그리고 disconnected() 시그널은 클라이언트 접속을 종료하면 이 시그널이 발생하고 이 시그널과 연결된 disconnected() Slot 함수가 호출된다. 다음 예제 소스코드는 ChatServer.cpp 의 소스코드이다.

```
#include "chatserver.h"
#include <QRegularExpression>
#include <QRegularExpressionMatch>
```

```
ChatServer::ChatServer(QObject *parent)
    : QTcpServer(parent)
{
}

void ChatServer::incomingConnection(qintptr socketfd)
{
    QTcpSocket *client = new QTcpSocket(this);
    client->setSocketDescriptor(socketfd);
    clients.insert(client);

    emit clients_signal(clients.count());

    QString str;
    str = QString("New Member: %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;
        str = QString("Read line: %1").arg(line);

        emit message_signal(str);

        QRegularExpression meRegex("^/me:(.*)$");
        QRegularExpressionMatch match = meRegex.match(line);

        if(match.hasMatch())
```

```
{  
    QString user = match.captured(1);  
    users[client] = user;  
    foreach(QTcpSocket *client, clients)  
    {  
        client->write(QString("Server: %1 connected\n")  
                      .arg(user).toUtf8());  
    }  
  
    //sendUserList();  
}  
else if(users.contains(client))  
{  
    QString message = line;  
    QString user = users[client];  
  
    QString str;  
    str = QString("User name: %1, Message: %2")  
          .arg(user, 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;  
    str = QString("Disconnect: %1")  
          .arg(client->peerAddress().toString());  
  
    emit message_signal(str);  
  
    clients.remove(client);  
  
    emit clients_signal(clients.count());
```

예수님은 당신을 사랑합니다.

```
QString user = users[client];
users.remove(client);

sendUserList();
foreach(QTcpSocket *client, clients)
    client->write(QString("Server: %1 Disconnect").arg(user).toUtf8());
}

void ChatServer::sendUserList()
{
    QStringList userList;
    foreach(QString user, users.values())
        userList << user;

    foreach(QTcpSocket *client, clients)
        client->write(QString("User:" + userList.join(",") + "\n").toUtf8());
}

ChatServer::~ChatServer()
{
    deleteLater();
}
```

incomingConnection() 함수에서 QTcpSocket 을 새로 만드는 것은 새로운 클라이언트의 소켓 오브젝트를 생성하기 위함이다. 따라서 이 함수에서 생성된 QTcpSocket 클래스의 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 헤더 파일이다. 아래와 같이 소스코드를 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include "chatserver.h"
```

```
QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

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 "./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)));
```

예수님은 당신을 사랑합니다.

```
connect(server, SIGNAL(message_signal(QString)), this,
        SLOT(slot_message(QString)));

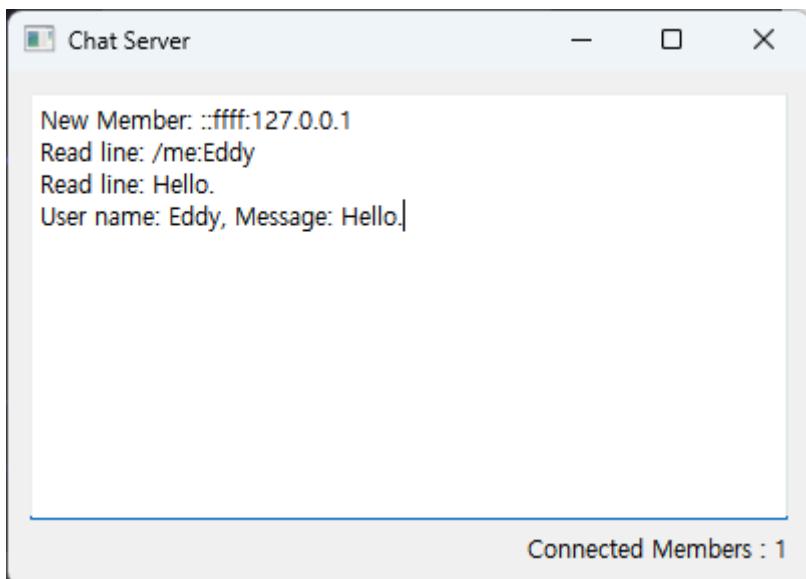
server->listen(QHostAddress::Any, 35000);
}

void Widget::slot_clients(int users)
{
    QString str = QString("Connected Members : %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. 그리고 slot_message() Slot 함수는 QTextEdit 창에 메시지를 출력한다. 위와 같이 작성이 완료 되었다면 빌드 후 실행해보자.

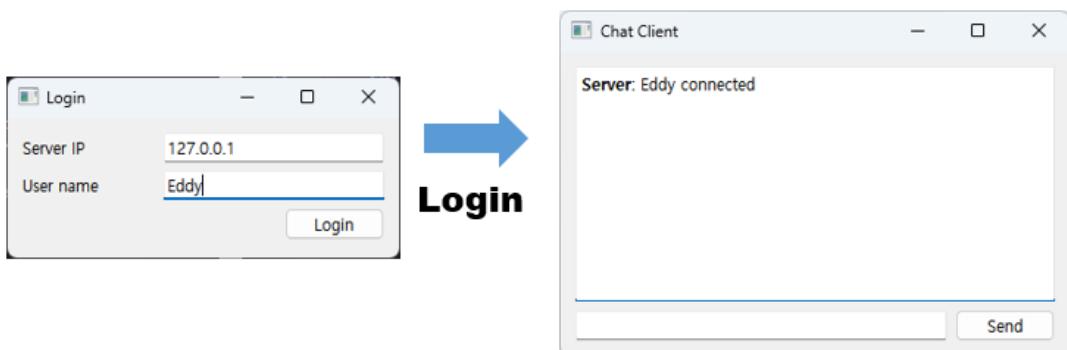


예수님은 당신을 사랑합니다.

위의 그림에서 보는 것과 같이 사용자가 접속한 정보, 보낸 메시지 정보가 GUI상에 표시된다. 이 예제의 소스코드는 00_ChatServer 디렉토리를 참조하면 된다.

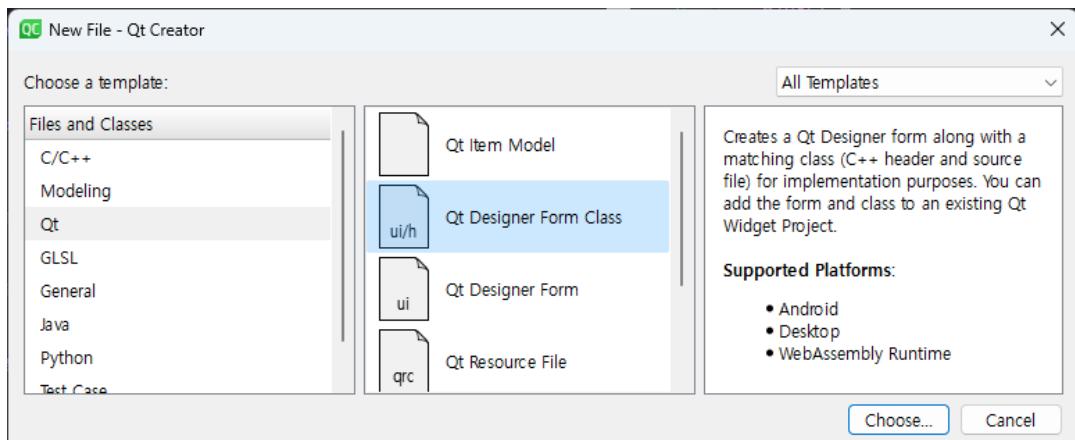
● 채팅 클라이언트 예제 구현

이번에는 채팅 서버와 메시지를 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 [login] 버튼을 클릭하면 서버 IP주소로 채팅 서버에 접속이 완료 된다. 그리고 로그인 위젯은 Hide 되고 채팅 클라이언트 위젯이 활성화(Show) 된다.



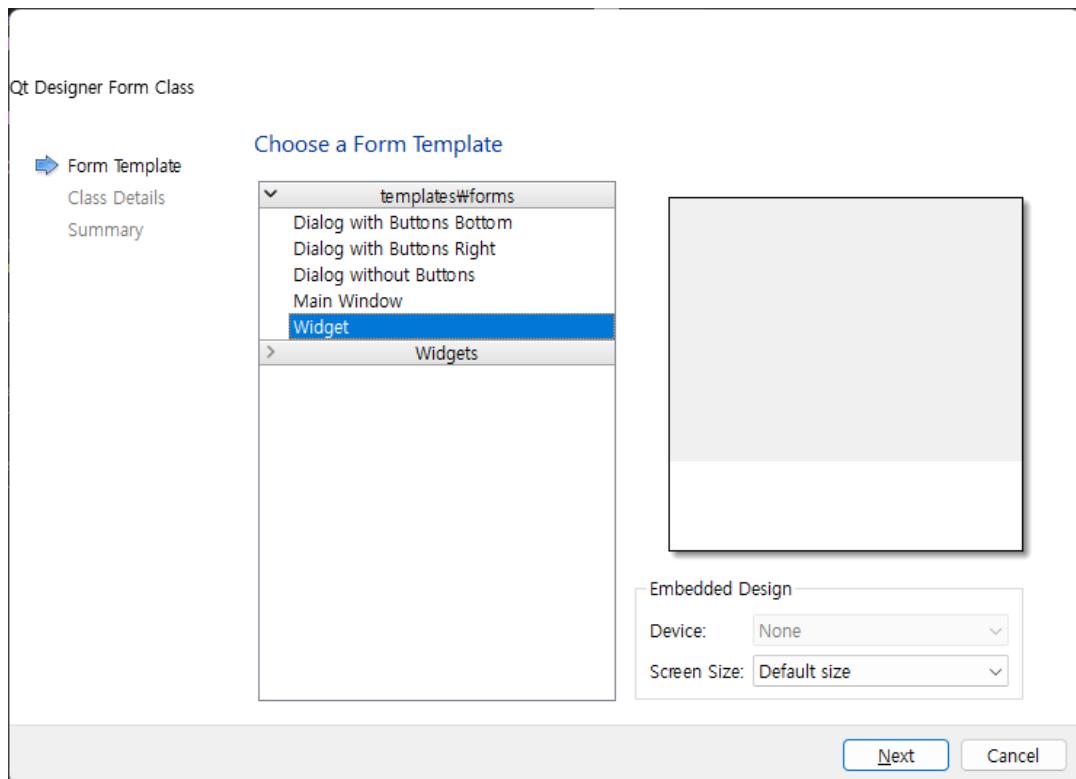
채팅 클라이언트 위젯에서 중간에 위치한 QTextEdit 위젯은 서버로부터 수신한 메시지를 출력한다. 하단의 QLineEdit 는 서버로 보낼 메시지를 입력한다. 그리고 [Send] 버튼을 클릭하면 메시지를 채팅 서버로 전송한다.

Qt Widget 기반 프로젝트를 생성한다. 그리고 아래 그림에서 보는 것과 같이 [Qt Designer Form Class] 를 선택하고 하단의 [Choose] 버튼을 클릭한다.

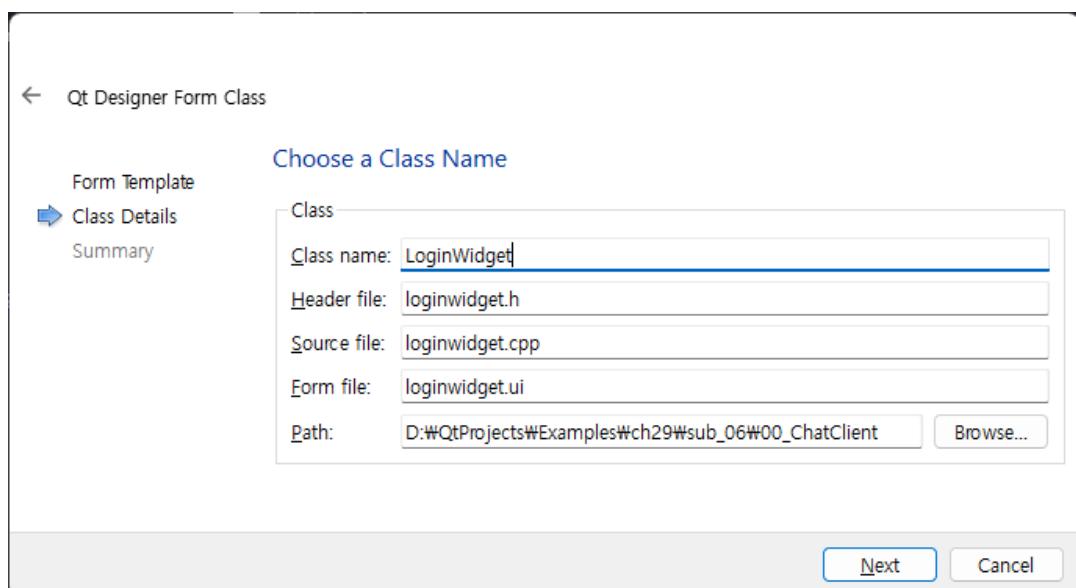


예수님은 당신을 사랑합니다.

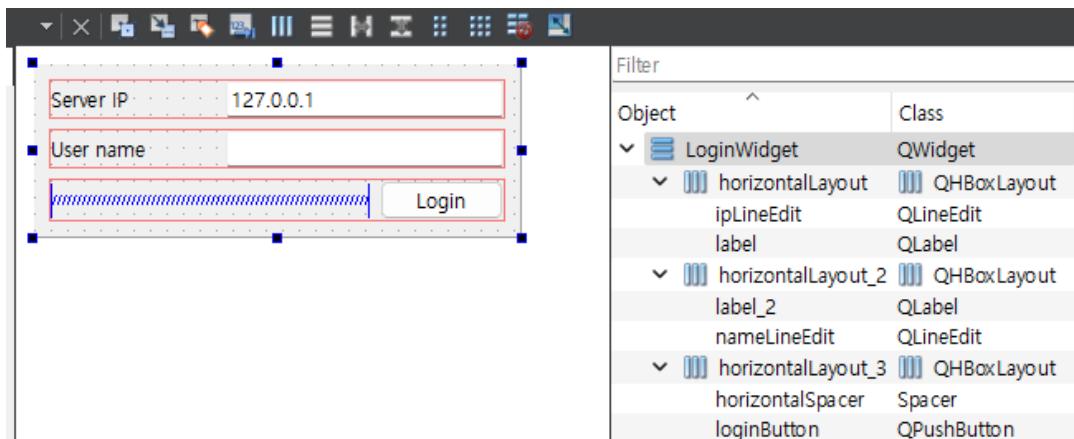
다음 다이얼로그 화면에서는 [Widget] 을 선택하고 [Next] 버튼을 클릭한다.



아래 다이얼로그에서 보는 것과 같이 Class 정보를 입력한다.



위와 같이 Form Class 를 추가 했다면 loginwidget.ui 파일을 열어서 아래 그림에서 보는 것과 같이 GUI상에 위젯들을 배치한다.



다음으로 loginwidget.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef LOGINWIDGET_H
#define LOGINWIDGET_H

#include <QWidget>

namespace Ui {
class LoginWidget;
}

class LoginWidget : public QWidget
{
    Q_OBJECT

public:
    explicit LoginWidget(QWidget *parent = nullptr);
    ~LoginWidget();

private:
    Ui::LoginWidget *ui;

private slots:
    void loginBtnClicked();

signals:
    void sig_loginInfo(QString addr, QString name);
};

#endif // LOGINWIDGET_H
```

예수님은 당신을 사랑합니다.

다음은 widget.cpp 소스 파일을 아래와 같이 작성한다.

```
#include "loginwidget.h"
#include "ui_loginwidget.h"

LoginWidget::LoginWidget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::LoginWidget)
{
    ui->setupUi(this);
    connect(ui->loginButton, &QPushButton::pressed,
            this, &LoginWidget::loginBtnClicked);
}

void LoginWidget::loginBtnClicked()
{
    QString serverIp = ui->ipLineEdit->text().trimmed();
    QString name = ui->nameLineEdit->text().trimmed();

    emit sig_loginInfo(serverIp, name);
}

LoginWidget::~LoginWidget()
{
    delete ui;
}
```

GUI 상에서 [Login] 버튼을 클릭하면 Server IP 와 User name 을 QString 에 저장한다. 그리고 QString 에 저장한 값을 sig_loginInfo() 시그널의 인자로 전달한다. 이 시그널은 Widget 클래스의 Slot 함수와 연결된다. 다음은 widget.h 헤더 파일의 소스코드이다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QTcpSocket>
#include "loginwidget.h"

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
```

```
{  
    Q_OBJECT  
  
public:  
    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();  
};  
#endif // WIDGET_H
```

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 <QRegularExpression>  
#include <QRegularExpressionMatch>  
  
Widget::Widget(QWidget *parent)  
    : QWidget(parent)  
    , ui(new Ui::Widget)  
{  
    ui->setupUi(this);
```

```
loginWidget = new LoginWidget();

connect(loginWidget, &LoginWidget::sig_loginInfo,
        this,           &Widget::loginInfo);

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();
```

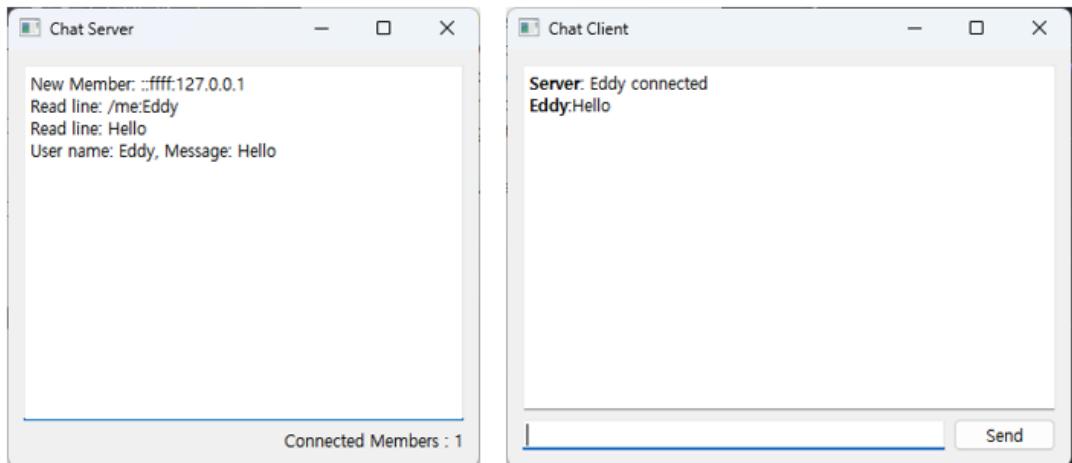
```
socket->write(QString("/me:" + userName + "\n").toUtf8());  
}  
  
void Widget::readyRead()  
{  
    while(socket->canReadLine())  
    {  
        QString line = QString::fromUtf8(socket->readLine()).trimmed();  
  
        QRegularExpression re("^(?:[^;]+);(.*)$");  
        QRegularExpressionMatch match = re.match(line);  
  
        if(match.hasMatch())  
        {  
            QString user = match.captured(1);  
            QString message = match.captured(2);  
  
            ui->roomTextEdit->append("<b>" + user + "</b>:" + message);  
        }  
    }  
}  
  
Widget::~Widget()  
{  
    delete ui;  
}
```

[Send] 버튼을 클릭하면 sayButton_clicked() Slot 함수가 호출된다. connected() Slot 함수는 채팅 서버와 연결이 완료되면 호출된다.

readyRead() 는 채팅 서버가 메시지를 전송하면 호출되는 Slot 함수이다. 이 Slot 함수에서 메시지를 QTextEdit 위젯에 출력한다.

이 예제를 빌드한 다음 실해 보도록 하자. 이 예제를 실행하기 전 서버 예제가 먼저 실행되어야 한다.

예수님은 당신을 사랑합니다.



이 예제의 소스코드는 00_ChatClient 디렉토리를 참조하면 된다.

29. Qt WebSocket

WebSocket 은 TCP 기반 프로토콜이며 웹에서 사용하는 HTTP의 단점을 보완하기 위해서 만들어진 기술이다.

HTTP 프로토콜은 웹에서 사용하는 프로토콜이다. 클라이언트와 서버가 데이터 통신을 할 때 HTTP 프로토콜을 사용하는 대표적인 예로 Web browser와 Web Server를 예로 들 수 있다.

Web browser 가 Web Server 에게 데이터를 요청하면 Web Server 는 Web browser 에게 요청한 데이터를 전송한다. 그런 후, 전송이 모두 완료하면 Web Browser와 Web Server 간의 연결은 종료된다.

항상 Web browser 가 요청이 있을 때마다 Web Server 는 응답을 종료하기 때문에 시간과 자원의 낭비가 심한 편이다.

이러한 단점을 극복하기 위해서 WebSocket 이 생겨났다. WebSocket 은 응답이 완료되더라도 연결이 종료되지 않는다. 따라서 Web browser 가 Web Server 간에 연결이 되어 있다면 연결된 Session을 재 활용한다. 따라서 재 연결하는데 시간과 자원 낭비를 줄일 수 있다.

WebSocket 은 Web browser, Web Server 뿐만 아니라 다양한 어플리케이션에서 사용할 수 있다.

즉 TCP Socket 을 이용해서 통신을 했던 것과 같이 서버/클라이언트 개념에서 TCP Socket을 사용했던 것과 같이 WebSocket을 사용할 수 있다.

Qt 에서는 WebSocket 을 제공하기 위해서 Qt WebSocket 모듈을 제공한다. Qt WebSockets 모듈을 이용하면 웹브라우저와 통신할 수 있는 어플리케이션을 쉽게 구현할 수 있다.

또한 Qt WebSockets 모듈을 이용하면 Node.js 를 이용해 서버 Side 어플리케이션을 쉽게 개발할 수 있는 것과 같이 Web browser와 통신할 수 있는 서버 어플리케이션을 Qt WebSockets 모듈을 이용해 쉽게 구현할 수 있다.

Qt 에서 WebSocket 모듈을 사용하기 위해서 프로젝트파일에 다음과 같이 추가해야 한다. CMake를 사용하는 경우 아래와 같이 추가하면 된다.

예수님은 당신을 사랑합니다.

```
find_package(Qt6 REQUIRED COMPONENTS WebSockets)
target_link_libraries(mytarget PRIVATE Qt6::WebSockets)
```

qmake를 사용한다면 아래와 같이 추가한다.

```
QT += websockets
```

Qt WebSockets 모듈은 쉽게 어플리케이션을 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 Signal과 Slot 을 사용할 수 있다.

예를 들어 QWebSocket 을 이용해 Connection이 완료되면 QWebSocket 에서 제공하는 connected() 시그널이 발생한다. 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 연결이 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.

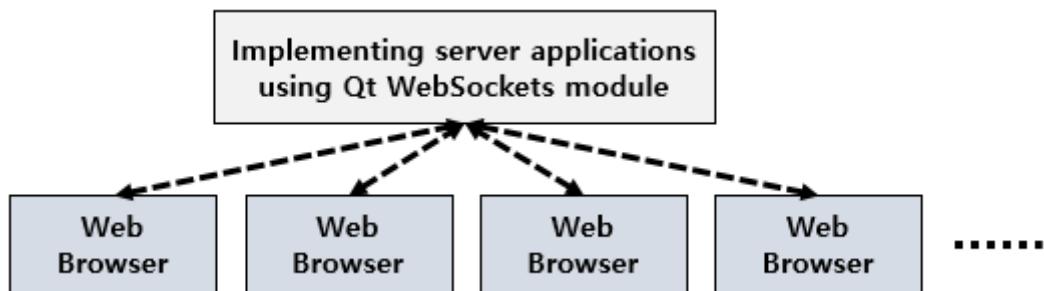
서버와 연결이 끊어지면(Close) disconnected() 시그널이 발생하고 closed()라는 Slot 함수가 호출된다.

위의 예제에서 보는 것과 같이 Qt WebSockets 모듈을 이용하면 웹브라우저와 통신할 수 있는 어플리케이션을 쉽게 구현할 수 있다.

다음은 웹브라우저와 Qt WebSockets 모듈을 이용해 구현한 예제 간 Web Socket 프로토콜을 이용해 예제를 구현해 보도록 하자.

✓ 웹브라우저와 통신을 위한 서버 어플리케이션 구현

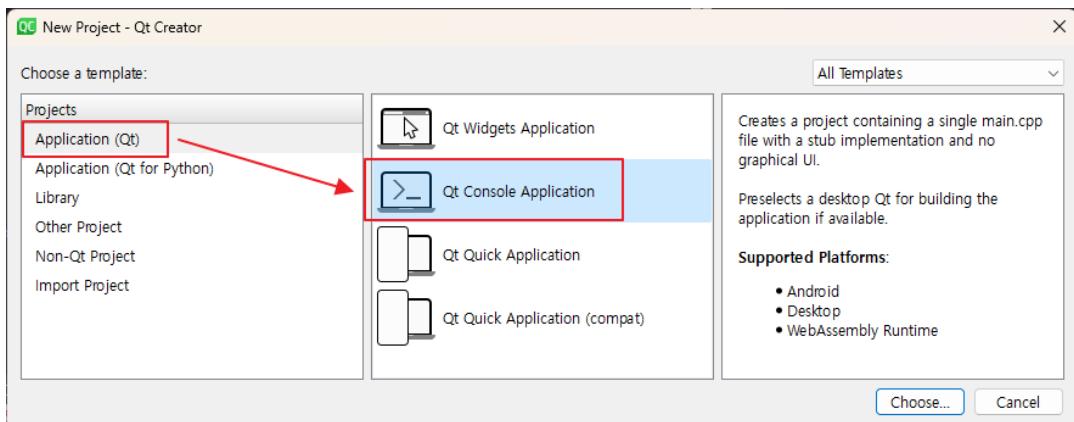
이번 예제에서는 Web Brower 에서 서버 어플리케이션과 통신하기 위한 소스코드를 먼저 작성해 본 후 Qt WebSockets 모듈을 이용해 서버 어플리케이션을 구현해 보도록 하자.



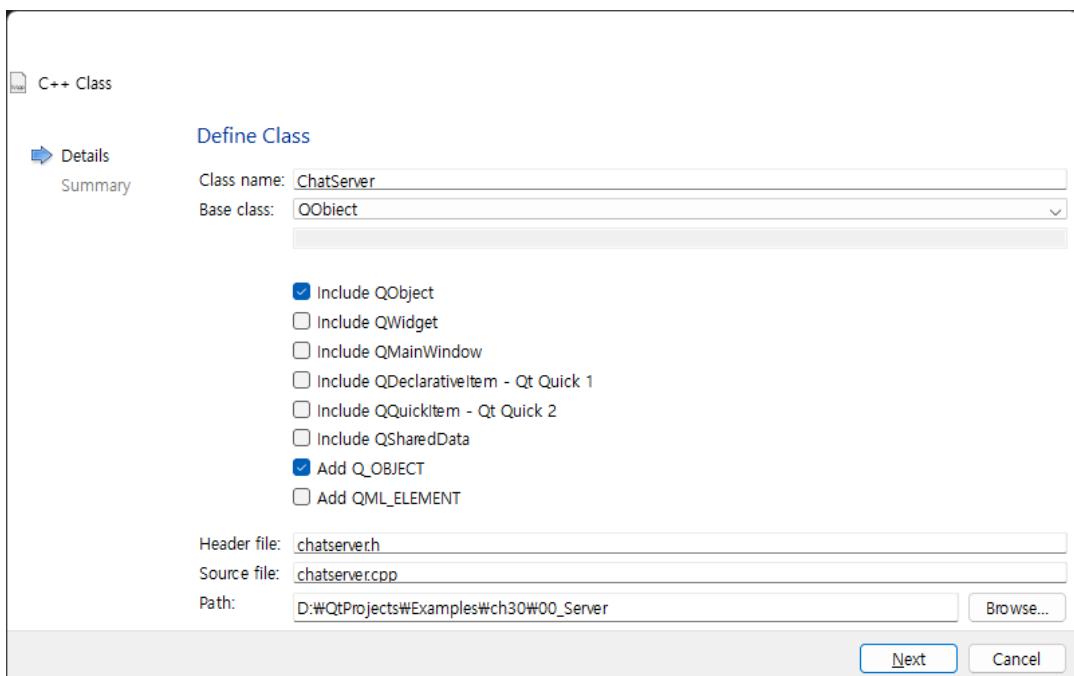
위의 그림에서 보는 것과 같이 웹 브라우저에서 동작하는 WebSocket을 HTML5를 이용해 구현할 것이다. 그리고 서버 어플리케이션을 Qt를 이용해 구현할 것이다.

먼저 Qt 기반의 서버 어플리케이션 예제를 구현해 보도록 하자.

아래 그림에서 보는 것과 같이 Console 기반의 프로젝트를 생성한다.



프로젝트 생성 후, 아래와 같이 ChatServer 클래스를 추가한다.



그리고 Qt WebSockets 모듈을 프로젝트에서 사용할 수 있도록 CMakeList.txt 파일에 아래와 WebSocket 모듈을 사용하도록 추가한다.

```
cmake_minimum_required(VERSION 3.14)

project(00_Server LANGUAGES CXX)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)
```

```
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt NAMES Qt6 Qt5 REQUIRED COMPONENTS Core WebSockets)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Core WebSockets)

add_executable(00_Server
    main.cpp
    chatserver.h chatserver.cpp
)
target_link_libraries(00_Server Qt${QT_VERSION_MAJOR}::Core)
target_link_libraries(00_Server Qt${QT_VERSION_MAJOR}::WebSockets)

include(GNUInstallDirs)
install(TARGETS 00_Server
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

다음으로 chatserver.h 헤더 파일을 열어서 아래와 같이 소스코드를 작성한다.

```
#ifndef CHATSERVER_H
#define CHATSERVER_H

#include <QObject>
#include <QWebSocketServer>
#include <QWebSocket>

class ChatServer : public QObject
{
    Q_OBJECT
public:
    explicit ChatServer(quint16 port, QObject *parent = 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
```

onNewConnection() Slot 함수는 새로운 접속자 Signal 이 발생하면 호출된다. processMessage() Slot 함수는 클라이언트가 보내온 메시지를 서버에 접속한 모든 클라이언트(Web browser)에게 메시지를 전달한다.

socketDisconnected() Slot 함수는 특정 클라이언트 접속을 종료하면 접속한 클라이언트의 QWebSocket 클래스 오브젝트를 제거한다. 다음 예제 소스코드는 chatserver.cpp 소스 파일이다. 아래와 같이 소스코드를 작성한다.

```
#include "chatserver.h"  
  
ChatServer::ChatServer(quint16 port, QObject *parent)  
    : QObject{parent}  
{  
    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()  
{  
    qDebug() << "New Connection ";  
  
    QWebSocket *pSocket = m_pWebSocketServer->nextPendingConnection();  
  
    connect(pSocket, &QWebSocket::textMessageReceived,
```

예수님은 당신을 사랑합니다.

```
        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()
{
    qDebug() << "Client disconnect";

    QWebSocket *pClient = qobject_cast<QWebSocket *>(sender());
    if (pClient)
    {
        m_clients.removeAll(pClient);
        pClient->deleteLater();
    }
}

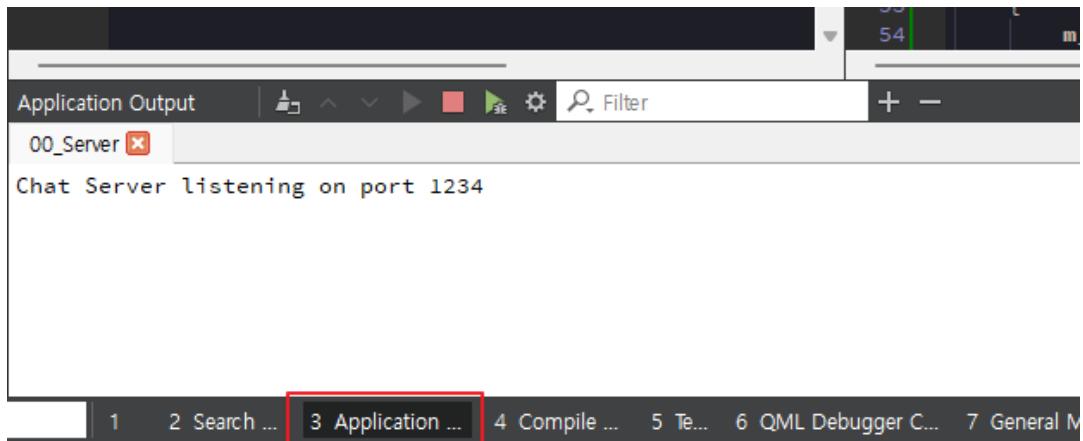
ChatServer::~ChatServer()
{
    m_pWebSocketServer->close();
    qDeleteAll(m_clients.begin(), m_clients.end());
}
```

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[])
{
    QApplication a(argc, argv);
    ChatServer server(1234);
    return a.exec();
}
```

위의 예제 소스코드에서 ChatServer 오브젝트의 첫 번째 인자는 포트번호이다. 위와 같이 작성하고 어플리케이션을 실행해보자. 실행하면 Qt Creator 의 Application Output 창에 아래와 같이 로딩된 것을 확인할 수 있다.



서버는 구현이 완료하였다. 이번에는 Web browser 에서 우리가 작성한 서버를 접속할 수 있도록 HTML을 작성해 보도록 하자. Web browser 에서 동작하는 클라이언트는 Qt Creator 가 아니더라도 소스코드 작성이 가능하다. 메모장과 같은 간단한 에디터 툴을 이용해 HTML 소스코드를 작성해도 무방하다.

아래와 같이 HTML 소스코드를 작성한 후 사용자가 원하는 이름으로 저장할 수 있다. 여기서는 chatclient.html 이름으로 저장한다.

```
<html>
  <head>
    <title>WebSocket Chatting Client</title>
  </head>
  <body>
    <h1>WebSocket Chatting Client</h1>
    <p>
      <button onClick="initWebSocket();">Server Connection</button>
    </p>
  </body>
</html>
```

예수님은 당신을 사랑합니다.

```
<button onClick="stopWebSocket();">Disconnect</button>
<button onClick="checkSocket();">Status</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);
}
}

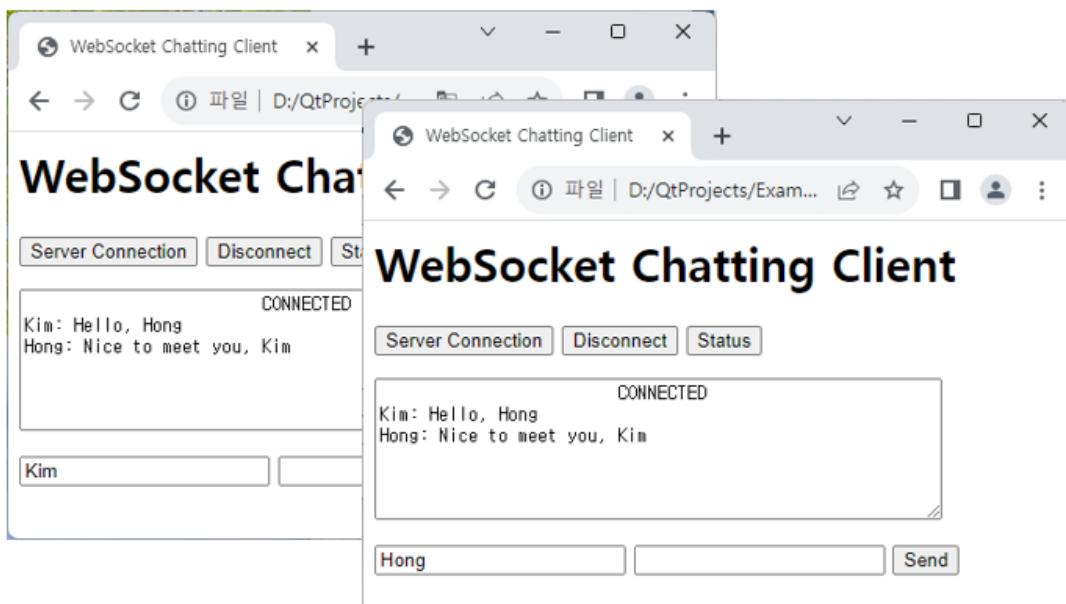
function stopWebSocket() {
    if (websocket)
        websocket.close();
}

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;
            }
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
        }
        case 3: {
            stateStr = "CLOSED";
            break;
        }
        default: {
            stateStr = "UNKNOW";
            break;
        }
    }
    debug("WebSocket state = " +
        websocket.readyState + " (" +
        stateStr + ")");
} else {
    debug("WebSocket is null");
}
}
</script>
</body>
</html>
```

위와 같이 작성은 완료하였다면 작성한 HTML 코드를 실행한 2개의 웹브라우저를 아래 그림에서 보는것과 같이 실행한다. 그리고 [서버연결] 버튼을 클릭하고 메시지를 입력 후 [Send] 버튼을 클릭해 보자.

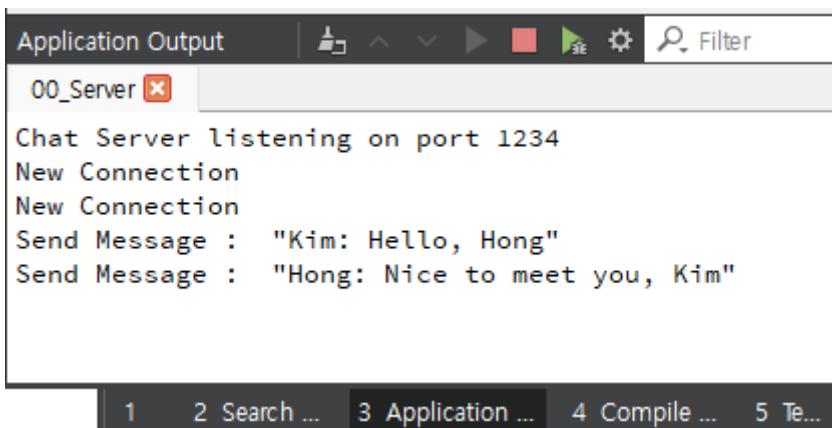


위의 그림에서 보는 것과 같이 [Server Connection] 버튼을 누르면 서버와 접속을 시도

예수님은 당신을 사랑합니다.

한다. 정상적으로 접속이 완료되면 메시지 창에 CONNECTED라는 메시지가 출력될 것이다. 그리고 아래 메시지를 입력하고 [Send] 버튼을 클릭하면 메시지가 서버로 전송된다. 그런 다음 서버는 연결된 클라이언트에게 받은 메시지를 다시 전송한다.

클라이언트가 보낸 메시지를 서버의 로그를 아래와 같이 Qt Creator의 [Application Output] 창에서 확인할 수 있다.



The screenshot shows the 'Application Output' window in Qt Creator. The title bar says 'Application Output'. Below it is a toolbar with icons for file operations, search, and filter. A tab labeled '00_Server' is selected. The main area displays the following log entries:

```
Chat Server listening on port 1234
New Connection
New Connection
Send Message : "Kim: Hello, Hong"
Send Message : "Hong: Nice to meet you, Kim"
```

At the bottom of the window, there are tabs labeled 1, 2 Search ..., 3 Application ..., 4 Compile ..., and 5 Te...

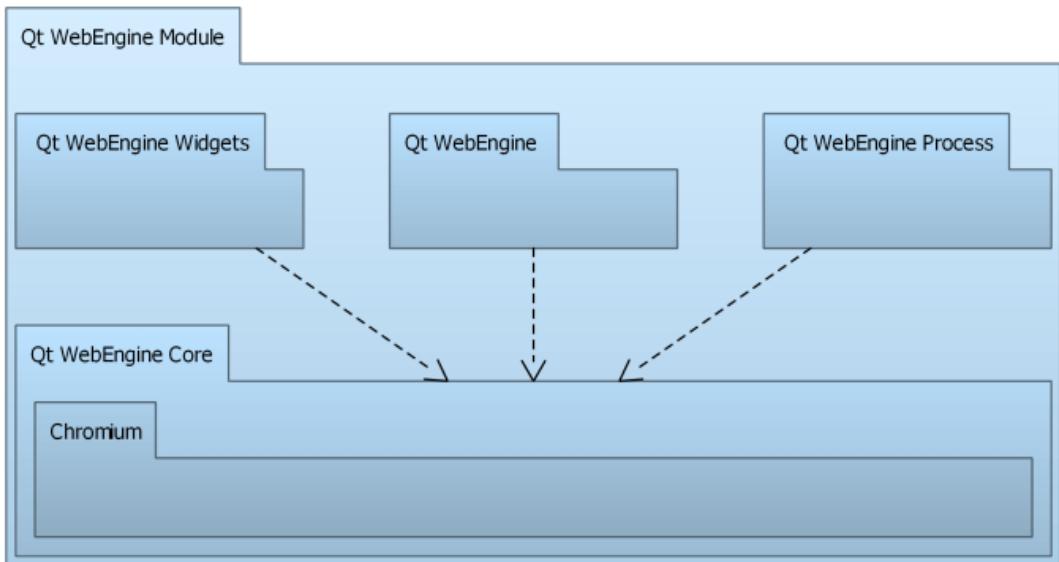
서버 예제 소스코드는 00_Server 디렉토리를 참조하면 된다. 그리고 HTML 소스코드는 00_HTML_Source를 참조하면 된다.

30. Qt WebEngine

Qt WebEngine 모듈은 Web browser 기반 응용 어플리케이션을 쉽게 만들 수 있는 기능을 제공한다.

Qt WebEngine 모듈은 C++ 과 QML 타입에서 사용할 수 있다. 주요 기능으로 HTML, XHTML 그리고 SVG Document를 랜더링 할 수 있는 기능을 제공한다. 그리고 CSS(Cascading Style Sheets)를 사용할 수 있고 JavaScript 와 HTML Documents 들도 사용할 수 있으면 제어할 수 있다.

Qt WebEngine 모듈의 Architecture 는 3가지 모듈로 분류된다.



✓ **Qt WebEngine Widgets** module

Qt WebEngine Widgets Module 은 Web 어플리케이션 기반의 위젯을 제공한다. 즉 QWidget 과 같이 Web 의 Contents 가 랜더링 되는 영역을 위젯 형태로 생성할 수 있는 위젯을 제공한다. 단 이 모듈은 C++ 에서 사용하는 위젯이다.

✓ **Qt WebEngine** module

이 모듈은 Qt WebEngine Widgets 모듈과 제공하는 기능은 동일하지만 QML에서 사용할 수 있는 모듈이다.

✓ **Qt WebEngine Process module**

Qt WebEngine 모듈은 Chromium Project를 기반으로 구현되었다. Chromium은 자체 네트워크 및 Painting Engine을 제공한다. 따라서 이 모듈은 Chromium에서 제공하는 Core와 통신을 하기 위한 기능을 제공한다.

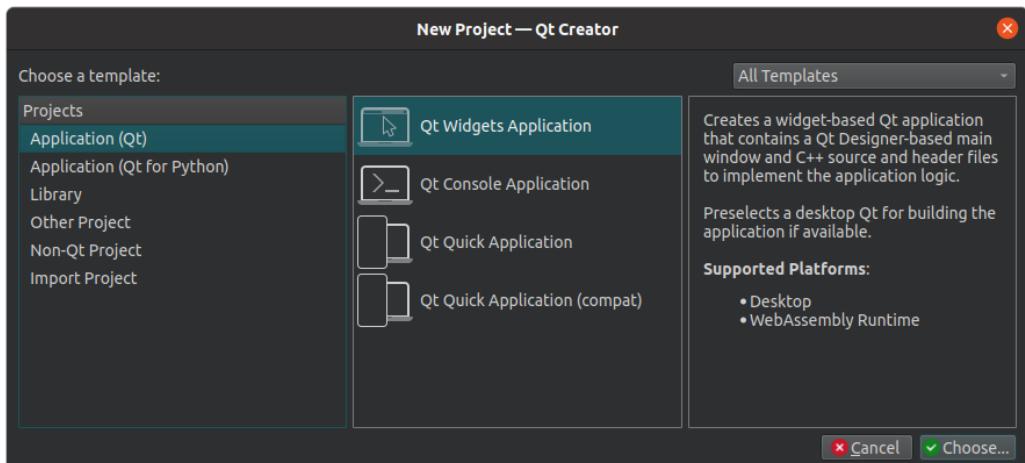
우리는 이번 장에서 Qt WebEngine 모듈을 이용해 간단한 웹 브라우저 예제를 다루어 보도록 하자.

주의 해야할 사항으로, Qt WebEngine 모듈을 사용할 경우 MS Windows에서 MinGW 컴파일러는 지원하지 않는다. MS Windows에서 Qt WebEngine 모듈을 사용하는 경우 MSVC 컴파일러를 사용해야 한다.

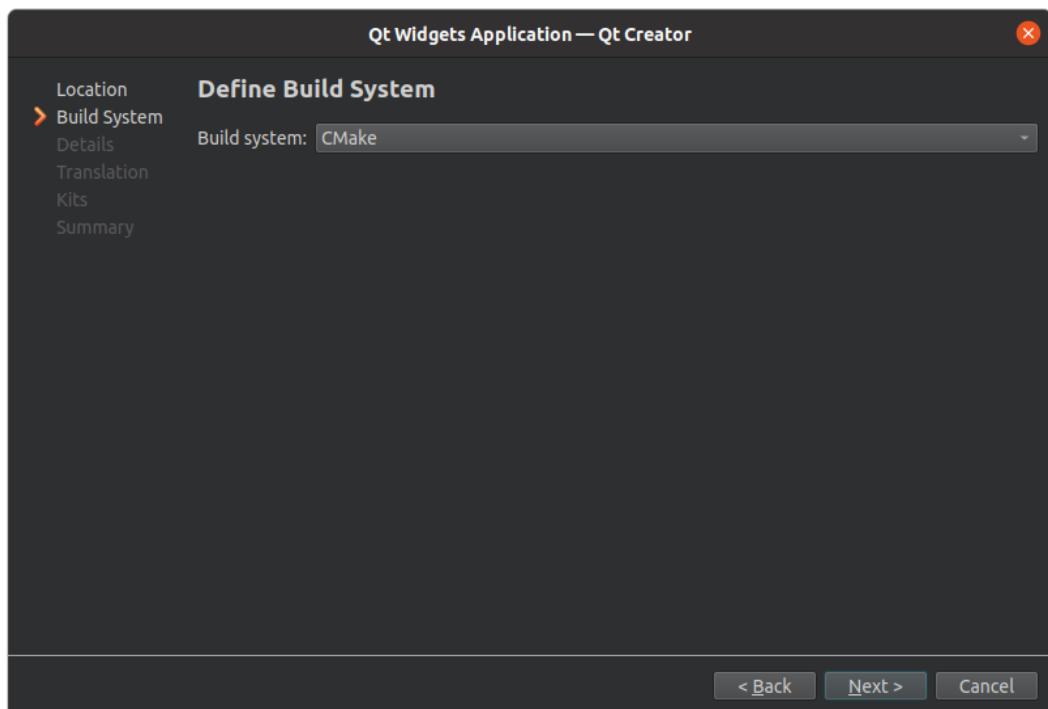
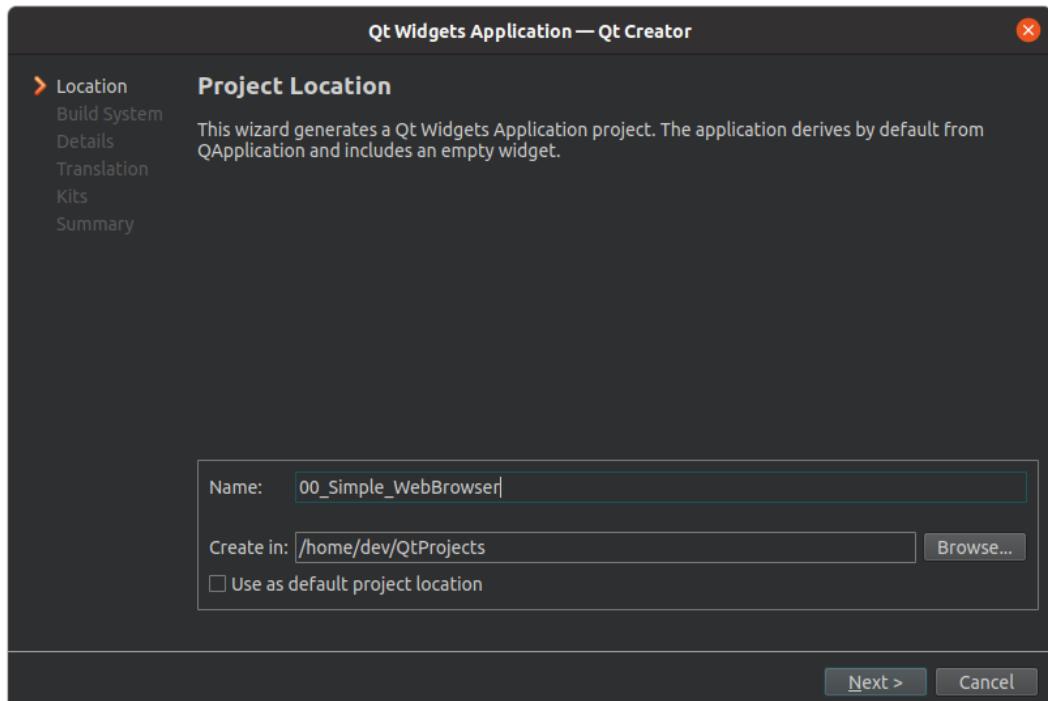
이번 장은 리눅스 플랫폼을 사용하고 Qt 6.5.3 버전을 사용할 것이다.

✓ **간단한 Web Browser 예제 구현**

프로젝트 생성 시 Qt Widget 기반의 프로젝트를 선택해 생성한다.

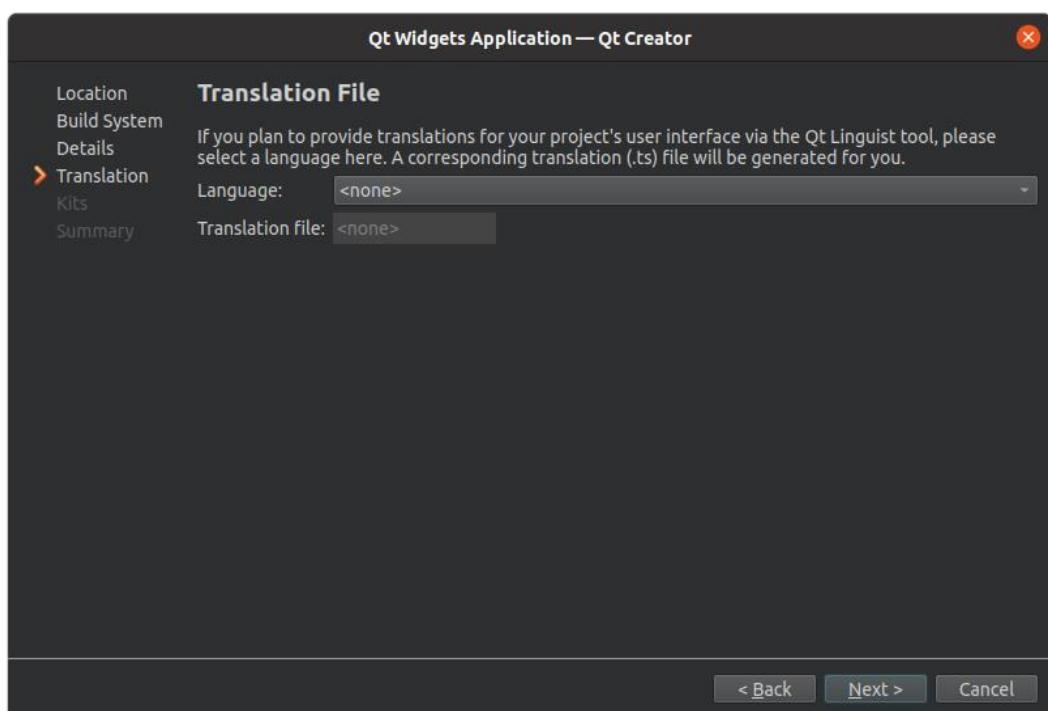
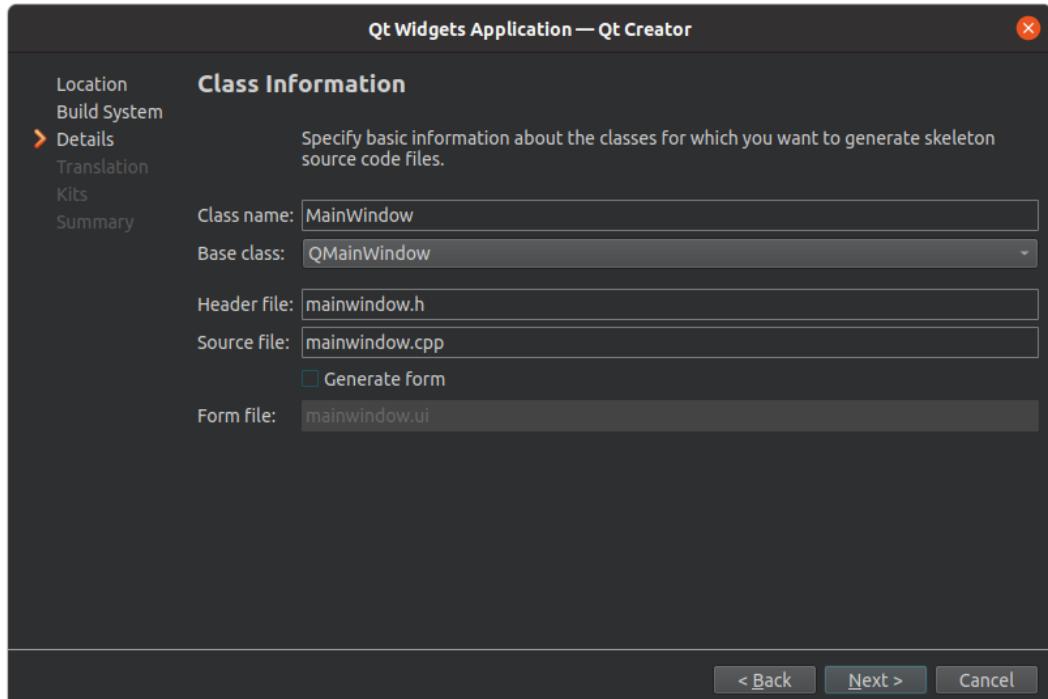


예수님은 당신을 사랑합니다.

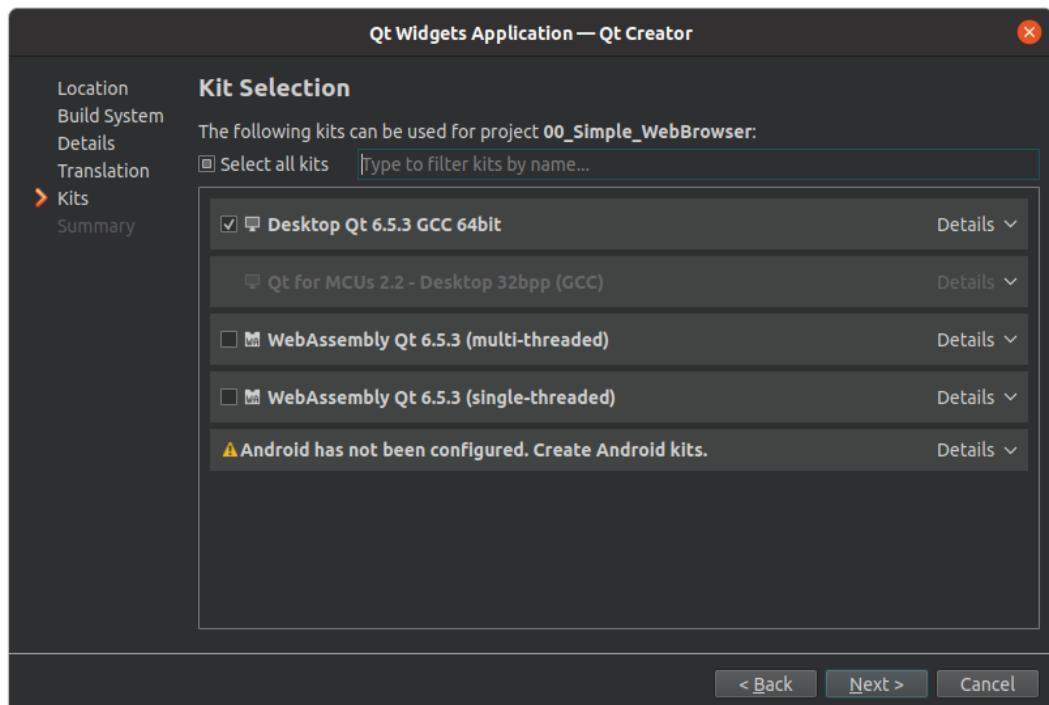


예수님은 당신을 사랑합니다.

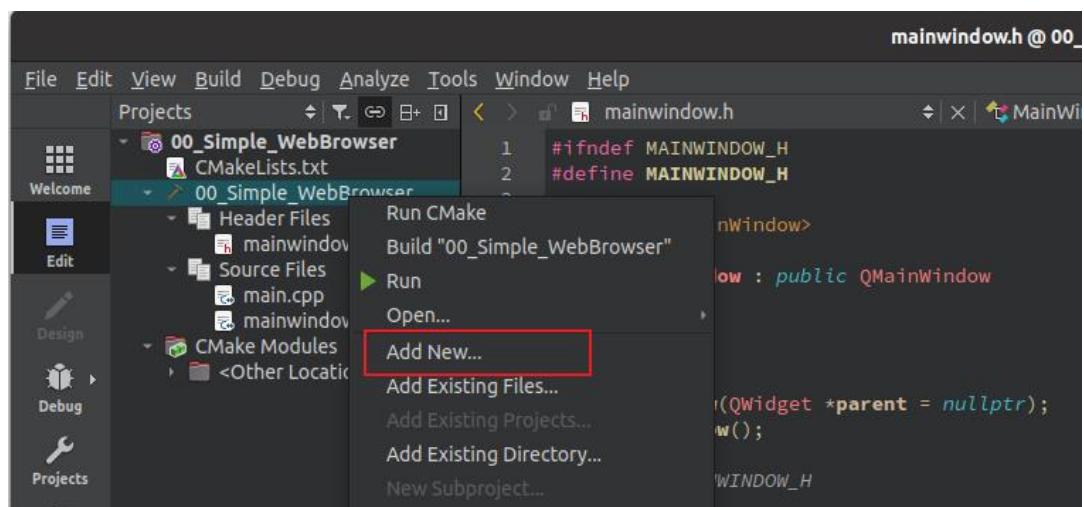
Class Informationダイアログ에서 아래와 같이 MainWindow 클래스를 생성한다. 그리고 [Generate form] 항목에서 CheckBox를 해제한다.



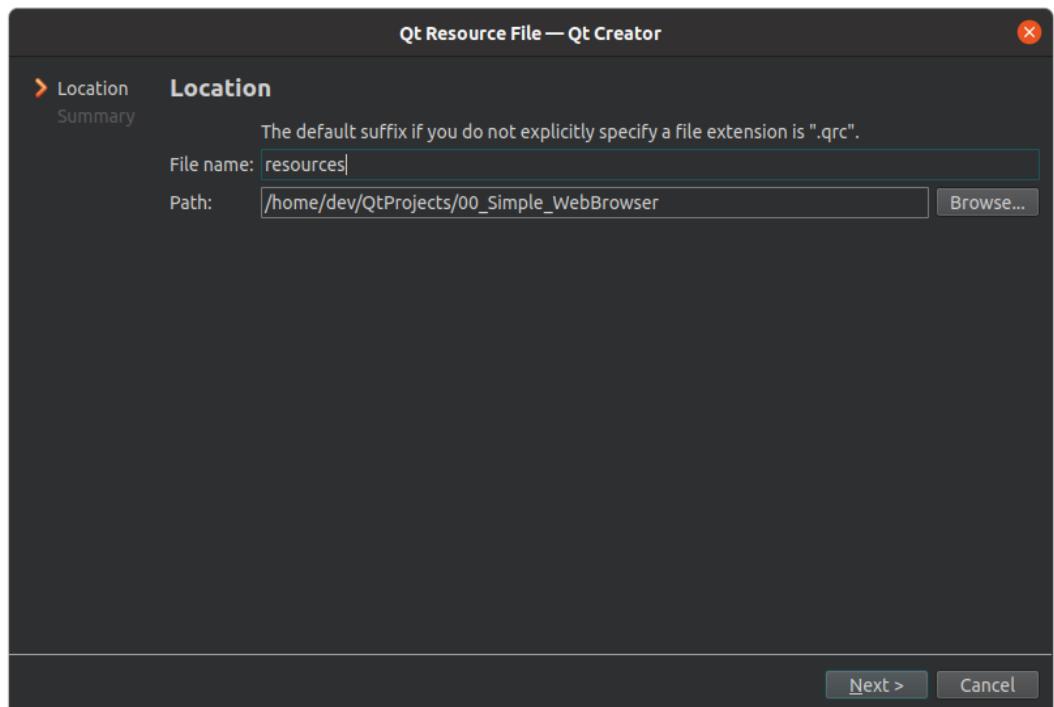
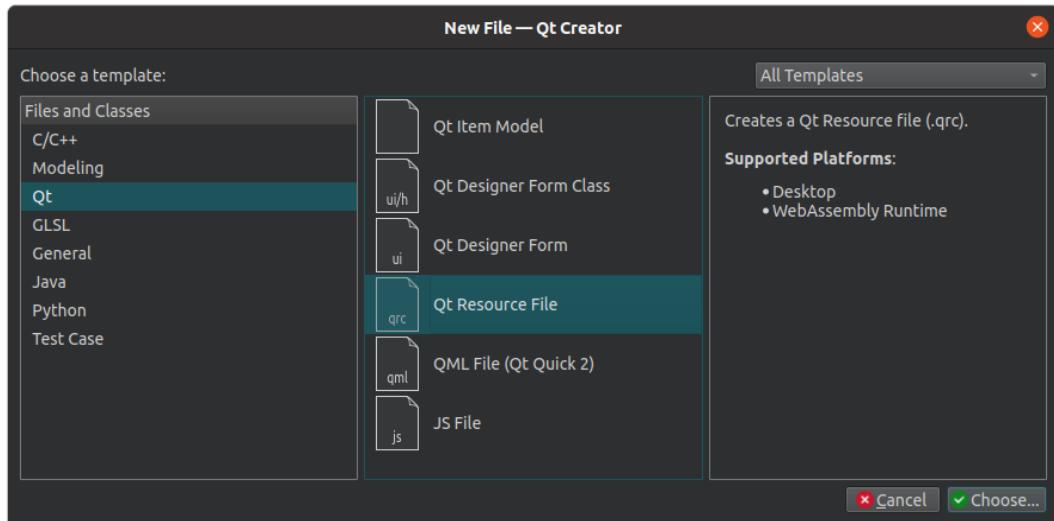
예수님은 당신을 사랑합니다.



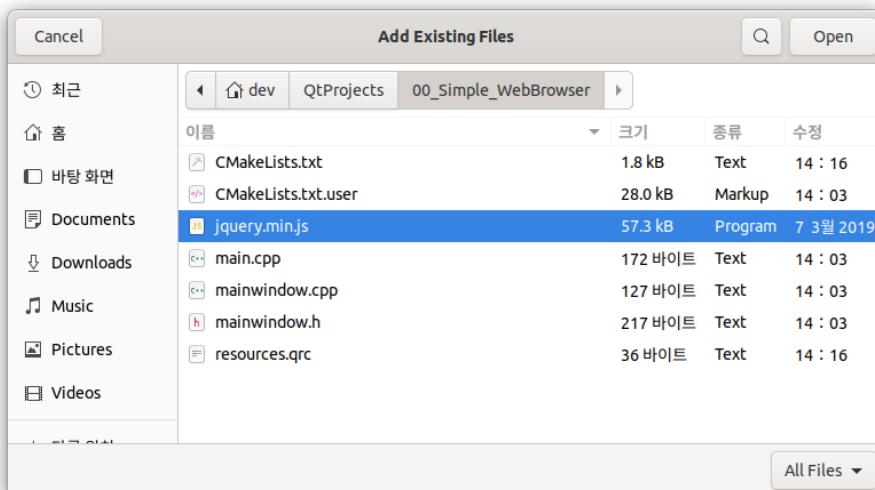
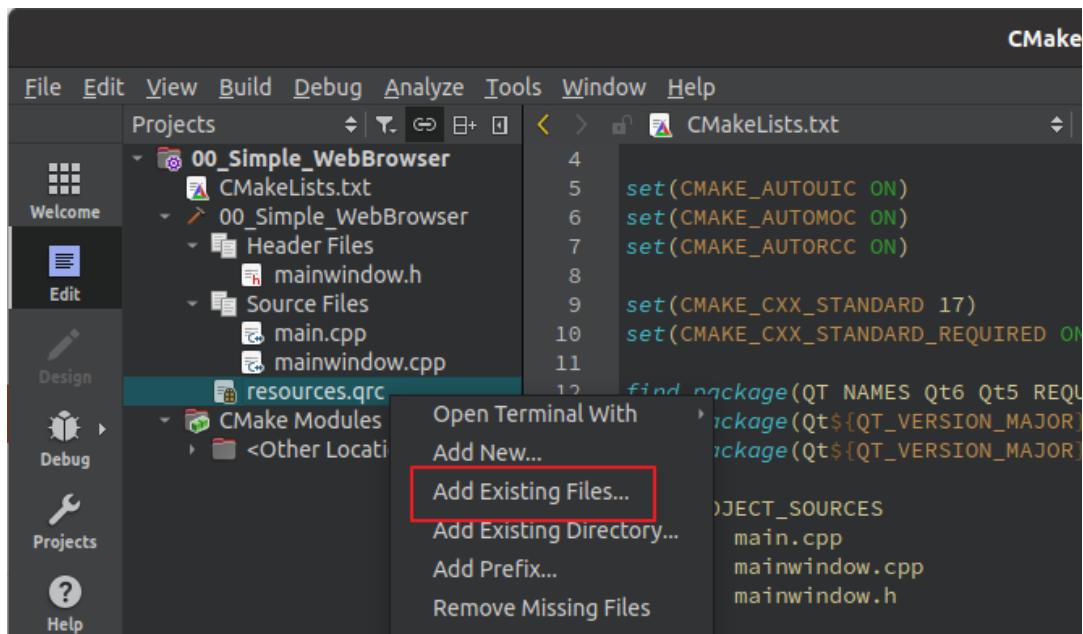
위와 같이 프로젝트를 생성했다면 프로젝트 상에서 Qt Resources를 추가한다.



예수님은 당신을 사랑합니다.



리소스 파일을 추가했다면 리소스에 jquery.min.js 파일을 추가 한다. 이 파일은 이 예제 소스코드 디렉토리에 보면 있다. 따라서 이 디렉토리에 있는 파일을 사용한다.



다음으로 프로젝트에서 Qt WebEngine 모듈을 사용하기 위해서는 CMakeList.txt 파일 상에 다음과 같이 추가해야 한다.

```
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS WebEngineWidgets)
target_link_libraries(00_Simple_WebBrowser PRIVATE Qt6::WebEngineWidgets)
```

만약 qmake를 사용한다면 아래와 같이 추가해야 한다.

```
QT += webenginewidgets
```

예수님은 당신을 사랑합니다.

여기서는 CMake를 사용하므로 CMakeList.txt 파일에 아래와 같이 작성하면 된다.

소스코드를 작성하기 전에 이 예제의 실행 화면을 보면서 어떤 기능을 제공하는지 먼저 살펴보도록 하자.

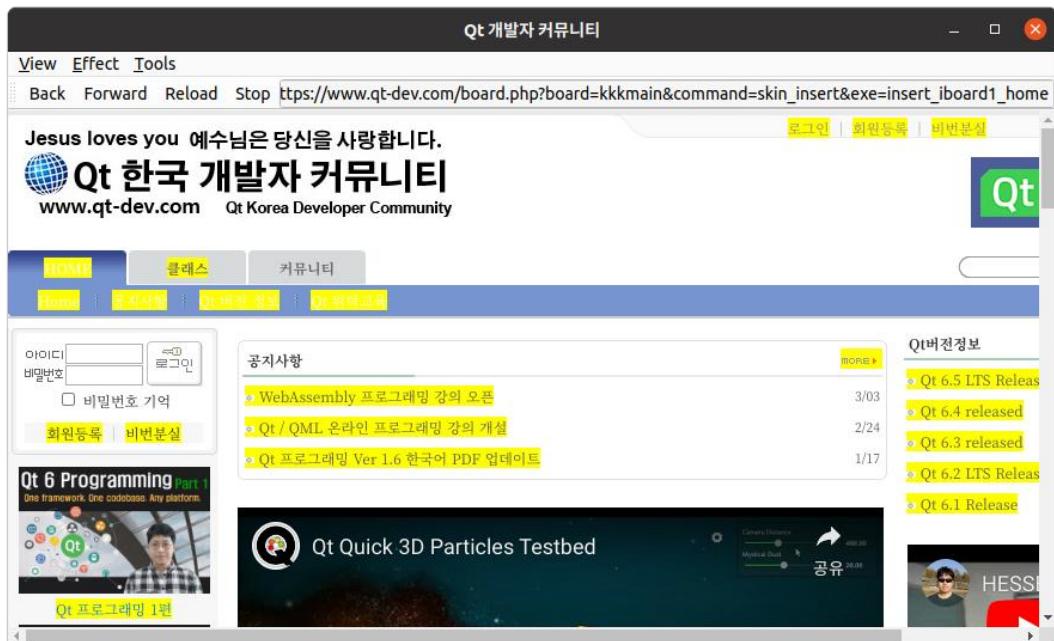


위의 그림에서 보는 것과 같이 예제를 실행하면 Qt 개발자 커뮤니티 사이트 처음 페이지를 로딩하는 예제이다. 메뉴에서 첫 번째 View 메뉴에는 [Page Source] 메뉴이다. 이 메뉴를 클릭하면 현재 웹 페이지의 소스코드를 볼 수 있는 창이 로딩된다.



다음은 메뉴에서 [Effect] -> [Highlight Links]를 선택하면 아래 그림에서 보는 것과 같이 링크가 모두 노란색 바탕으로 변경된다.

예수님은 당신을 사랑합니다.



Tools 메뉴에서 [Remove GIF images] 를 클릭하면 웹페이지에서 사용한 GIF 이미지를 모두 HTML 태그에서 제거한다.

메뉴 하단의 툴바 메뉴에서 첫 번째 [Back] 은 이전 페이지로 돌아가기 기능을 제공한다.

두 번째 [Forward]는 아이콘은 다음 페이지, 세 번째 [Reload]는 현재 페이지를 새로 고침 하는 기능을 제공한다. 네 번째 [Stop]은 현재 로딩 중일 때 중지 하는 기능을 제공한다. 그리고 우측의 주소 창은 현재 URL 주소 창이다.

지금까지 설명한 기능을 구현해 보도록 하자. 먼저 mainwindow.h 헤더 파일을 열어서 다음과 같이 소스코드를 작성한다.

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QWebEngineView>
#include <QLineEdit>

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
```

```
MainWindow(const QUrl& url,
           QWidget *parent = nullptr);
~MainWindow();

protected slots:

void adjustLocation();
void changeLocation();
void adjustTitle();
void setProgress(int p);
void finishLoading(bool);

void viewSource();

void highlightAllLinks();
void removeGifImages();

private:
QString jjQuery;
QWebEngineView *view;
QLineEdit *locationEdit;
int progress;

};

#endif // MAINWINDOW_H
```

adjustLocation() Slot 함수는 주소 창에 현재 웹 페이지 주소를 표시하는 기능을 제공한다. 웹 페이지 로딩이 완료 되면 QWebEngineView 클래스의 loadFinished() Signal 이 발생한다. 이 시그널 발생 시 adjusLocation() Slot 함수가 호출 된다.

changeLocation() Slot 함수는 주소 창에서 사용자가 Enter 키를 클릭하면 주소 창의 주소를 가져온다. 그리고 페이지를 랜더링 하기 위해서 주소 창에 입력한 웹 서버로 웹 페이지를 요청한다.

adjustTitle() Slot 함수는 웹 페이지의 <title> 태그의 문자열을 QMainWindow 위젯 타이틀 바에 표시한다. 그리고 finishLoading() 함수는 웹 페이지 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.

예수님은 당신을 사랑합니다.

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 JavaScript이다. 다음으로 mainwindow.cpp 소스코드를 아래와 같이 작성 한다.

```
#include "mainwindow.h"
#include <QFile>
#include <QToolBar>
#include <QMenu>
#include <QMenuBar>
#include <QAction>
#include <QTextEdit>

MainWindow::MainWindow(const QUrl& url, QWidget *parent)
    : QMainWindow(parent)
{
   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(tr("Page Source"), this);
 connect(viewSourceAction, &QAction::triggered,
         this, &MainWindow::viewSource);
 viewMenu->addAction(viewSourceAction);

QMenu *effectMenu = menuBar()->addMenu(tr("&Effect"));
 effectMenu->addAction(tr("Highlight all links"),
                       this, &MainWindow::highlightAllLinks);

QMenu *toolsMenu = menuBar()->addMenu(tr("&Tools"));
 toolsMenu->addAction(tr("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()
```

예수님은 당신을 사랑합니다.

```
{  
    QString code = QStringLiteral("qt.jQuery('[src*=gif]').remove()");  
    view->page()->runJavaScript(code);  
}  
  
MainWindow::~MainWindow()  
{  
}
```

지금까지 작성한 예제 소스코드는 00_Simple_WebBrowser 디렉토리를 참조한다.

31. Qt HTTP Server

Qt에서 제공하는 Qt HTTP Server 모듈은 Web Server를 쉽게 구현 할 수 있는 기능을 제공한다.

예를 들어 Web browser 또는 다른 응용 어플리케이션과 같은 클라이언트에게 HTTP 기반 REST API를 통해 서비스를 제공할 수 있다.

필자가 생각하기 이 모듈을 이용하는 가장 큰 이유로, 이 모듈을 이용하면 경량화된 Web server를 쉽게 구현할 수 있다. 또한 SSL 기반의 서비스도 쉽게 구현할 수 있다.

이 모듈을 사용하기 위해서는 프로젝트 파일에 아래와 같이 추가 해야 한다. 만약 CMake를 사용한다면 아래와 같이 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS HttpServer)
target_link_libraries(mytarget PRIVATE Qt6::HttpServer)
```

만약 qmake를 사용한다면 아래와 같이 추가 해야 한다.

```
QT += httpserver
```

Qt HTTP Server 모듈을 이용해 Web server를 구현하기 위해서 QHttpServer 클래스를 제공한다. 이 클래스를 이용하면 쉽게 Web server를 구현할 수 있다.

첫 번째로 Web browser (클라이언트)로부터 요청을 수신하기 위해서 listen() 멤버함수를 아래와 같이 사용해야 한다.

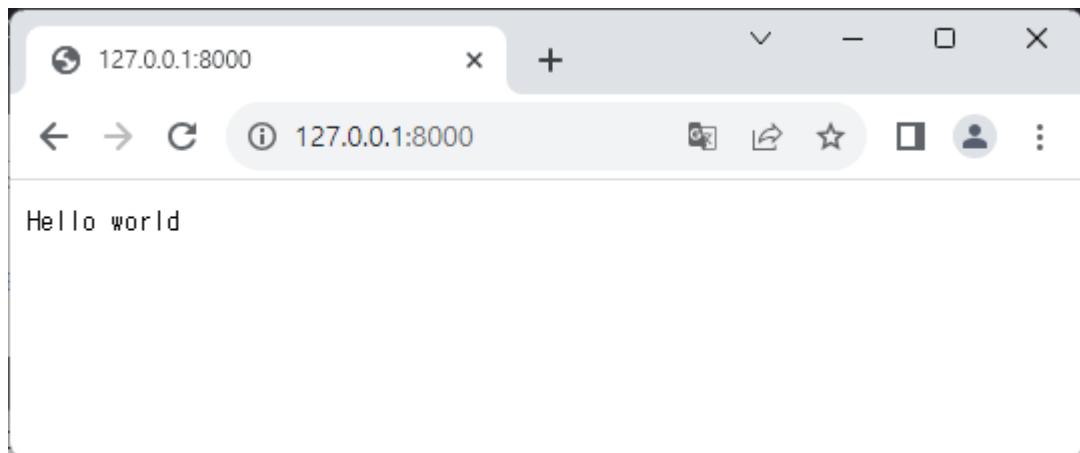
```
QHttpServer m_server;
m_server.listen(QHostAddress::Any, 8000);
```

첫 번째 인자는 IP 주소이다. 첫 번째 인자에서 직접 IP주소를 사용할 수 있다. 하지만 QHostAddress::Any 는 시스템상에 IP를 모두 사용할 수 있다. 두 번째 인자는 Port 번호이다.

따라서 Web browser 가 Web server에게 요청 시 [http://\[IP Address\]:\[Port\]](http://[IP Address]:[Port])를 사용해야 한다.

예를 들어 Web browser 가 Web server에게 요청할 때 <http://127.0.0.1:8000/> 주소로 요청을 했을 때 아래 그림에서 보는 것과 같이 "Hello world" 를 요청 할 경우

예수님은 당신을 사랑합니다.

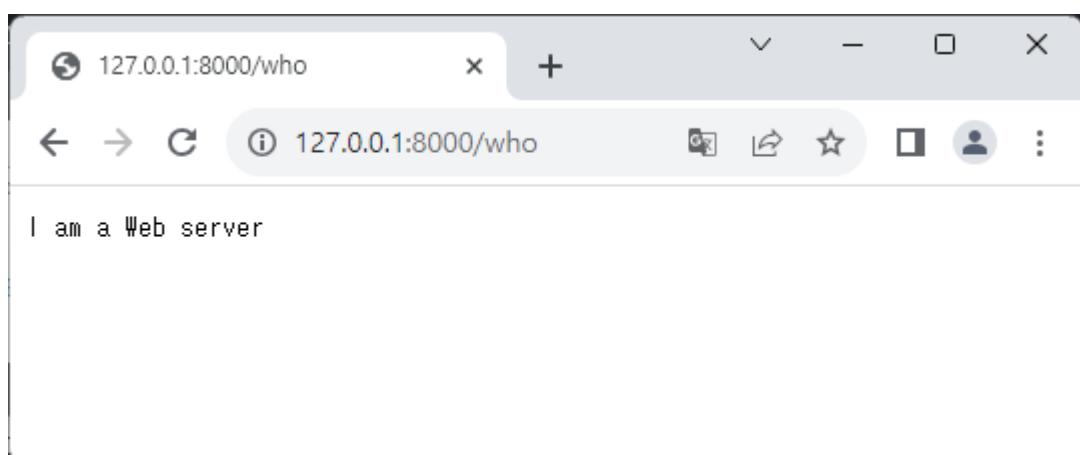


다음으로 위의 그림에서 보는 것과 같이 "Hello world"를 Web server로부터 응답 받기 위해서 route() 함수를 사용할 수 있다.

```
m_server.route("/", [] () {
    return "Hello world";
});
```

만약 특정 Filter 를 사용하고자 할 때 route() 함수의 첫 번째 인자에 Filter 옵션을 사용할 수 있다.

```
// [ URL ] http://127.0.0.1:8000/who
m_server.route("/who", [] () {
    return "I am a boy";
});
```



예수님은 당신을 사랑합니다.

JSON 타입으로 응답 하기 위해서 아래와 같은 방법을 사용할 수 있다.

```
// [ URL ] http://127.0.0.1:8000/hello_json
m_server.route("/hello_json", [] () {
    QJsonObject jsonObj;
    jsonObj[ "key1" ] = "value1";
    jsonObj[ "key2" ] = "value2";
    jsonObj[ "key3" ] = "value3";

    return jsonObj;
});
```



Web Browser 가 Web Server 에게 여러 개의 값을 전달하기 위해서 아래와 같은 방식으로 값을 전달 할 수 있다.

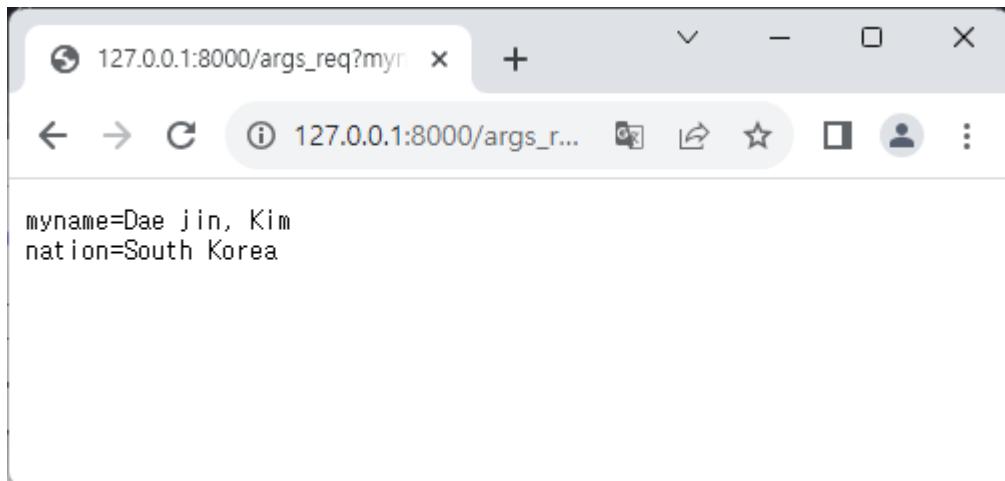
```
http://127.0.0.1:8000/args_req?myname=Dae jin, Kim&nation=South Korea
```

위의 예제에서 보는 것과 같이 myname 과 nation 이라는 값을 Web server에게 전달하기 위해서 아래와 같은 방법을 사용할 수 있다.

```
m_server.route("/args_req", [] (const QHttpServerRequest &req) {
    QString filter;
    QTextStream result(&filter);
    for (auto pair : req.query().queryItems()) {
        if (!filter.isEmpty())
            result << "\n";
```

예수님은 당신을 사랑합니다.

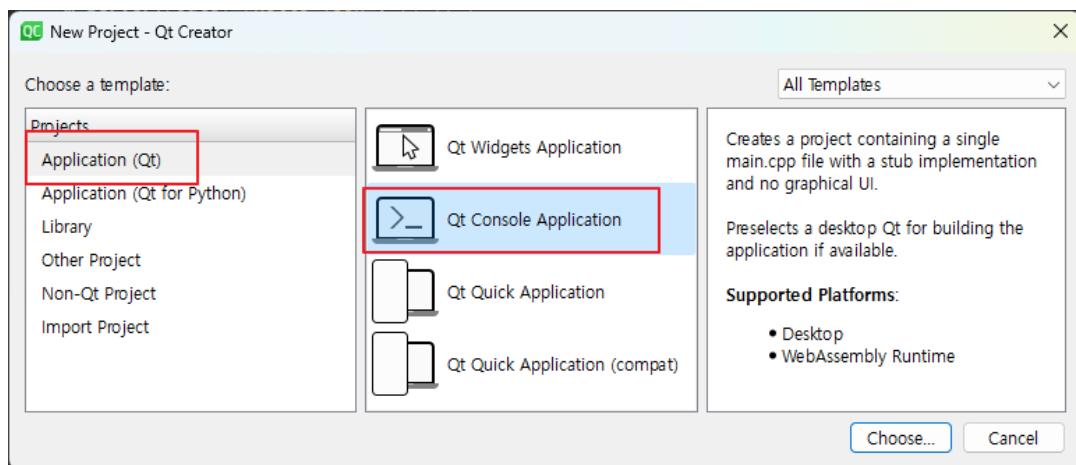
```
        result << pair.first << "=" << pair.second;
    }
    return filter;
});
```



지금까지 QHttpServer 클래스를 사용해 Web server를 구현하는 방법에 대해서 살펴보았다. 다음으로 예제를 통해 간단한 Web server 를 구현하는 방법에 대해서 살펴보도록 하자.

✓ 간단한 Web server 구현 예제

프로젝트 생성 시 Qt Console 기반의 어플리케이션을 선택한다.



프로젝트 생성 후 HttpServer 모듈을 사용할 수 있도록 프로젝트 파일에 아래와 같이

예수님은 당신을 사랑합니다.

추가 한다.

```
cmake_minimum_required(VERSION 3.14)

project(00_WebServer LANGUAGES CXX)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt NAMES Qt6 Qt5 REQUIRED COMPONENTS Core)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Core HttpServer)

add_executable(00_WebServer
    main.cpp
)

target_link_libraries(00_WebServer Qt${QT_VERSION_MAJOR}::Core)
target_link_libraries(00_WebServer Qt${QT_VERSION_MAJOR}::HttpServer)

include(GNUInstallDirs)
install(TARGETS 00_WebServer
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

다음으로 WebServer 클래스를 추가한다. 그리고 WebServer.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef WEBSERVER_H
#define WEBSERVER_H

#include <QObject>
#include <QtHttpServer/QHttpServer>
```

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

private:
    QHttpServer m_server;
    QString hostName(const QHttpRequest &request);

};

#endif // WEBSERVER_H
```

다음으로 WebServer.cpp 소스코드를 작성한다.

```
#include "WebServer.h"
#include <qloggingcategory.h>
#include <QJsonObject>
#include <QDebug>

using namespace Qt::StringLiterals;

WebServer::WebServer(QObject *parent)
    : QObject{parent}
{
    // [ URL ] http://127.0.0.1:8000
    m_server.route("/", [] () {
        return "Hello world";
    });

    // [ URL ] http://127.0.0.1:8000/who/
    m_server.route("/who", [] () {
        return "I am a Web server";
    });
}
```

```
// [ URL ] http://127.0.0.1:8000/hello/2/
m_server.route("/hello/<arg>/", [=] (qint32 id, const QHttpServerRequest &req) {
    QString hostStr = hostName(req) + u"/hello id ---> %1"_s.arg(id);
    return hostStr;
});

// [ URL ] http://127.0.0.1:8000/hello_json
m_server.route("/hello_json", [] () {
    QJsonObject jsonObj;
    jsonObj["key1"] = "value1";
    jsonObj["key2"] = "value2";
    jsonObj["key3"] = "value3";

    return jsonObj;
});

// [ URL ] http://127.0.0.1:8000/args_req?myname=Dae jin, Kim&nation=South Korea
m_server.route("/args_req", [] (const QHttpServerRequest &req) {
    QString filter;
    QTextStream result(&filter);
    for (auto pair : req.query().queryItems()) {
        if (!filter.isEmpty())
            result << "\n";
        result << pair.first << "=" << pair.second;
    }
    QLoggingCategory::setFilterRules(filter);

    return filter;
});

//m_server.listen(QHostAddress::LocalHost, 8000);
m_server.listen(QHostAddress::Any, 8000);
```

```
}

QString WebServer::hostName(const QHttpServerRequest &request)
{
    return QString::fromLatin1(request.value("Host"));
}

WebServer::~WebServer()
{
    deleteLater();
}
```

위의 예제 소스코드에서 보는 것과 같이 QHttpServer 클래스의 route() 함수는 Web browser (클라이언트) 의 요청을 처리한다. 그리고 listen() 함수는 Web browser 의 요청을 수신할 IP와 Port를 지정한다.

지금까지 간단한 Web server를 구현하는 방법에 대해서 알아보았다. 지금까지 전체 살펴본 전체 예제 소스코드는 00_WebServer 디렉토리를 참조하면 된다.

✓ SSL을 지원하는 Web server 예제

QHttpServer 클래스는 SSL(Secure Socket Layer) 을 지원한다. SSL을 지원하는 Web server를 구현하기 위해서는 Key 와 CSR 가 필요하다. 이 2개의 파일을 생성하기 위해서 여러가지 방법이 있으나 여기서는 OpenSSL을 이용하는 방법에 대해서 살펴보도록 하자.

```
# openssl genrsa -out test.key 2048
```

위와 같이 Key 생성 한다. 만약 Public Key를 사용한다면 아래와 같이 사용하면 된다.

```
# openssl rsa -in test.key -pubout -out test_pub.key
```

다음은 CSR 을 생성하는 방법이다.

```
# openssl req -new -key test.key -out test.csr
```

다음은 CRT를 생성하는 방법이다.

```
# openssl x509 -req -days 365 -in test.csr -signkey test.key -out test.crt
```

예수님은 당신을 사랑합니다.

위와 같이 Private Key 와 CRT를 생성하였으면 이 2개의 파일은 SSL을 지원하는 Web server 구현에 사용해야 한다.

다음으로 Qt Console 기반 프로젝트 생성 한다. 그리고 CMake상에 HttpServer 모듈을 추가한다.

그리고 생성한 SSL 파일 2개를 Resource 에 추가한다. 리소스 추가 후, 프로젝트상에 추가한다. SecureWebServer 클래스를 프로젝트에 추가한다. 아래는 SecureWebServer.h 헤더 파일이다.

```
#ifndef SECUREWEBSERVER_H
#define SECUREWEBSERVER_H

#include <QObject>
#include <QtHttpServer/QHttpServer>

class SecureWebServer : public QObject
{
    Q_OBJECT
public:
    explicit SecureWebServer(QObject *parent = nullptr);
    ~SecureWebServer();

private:
    QHttpServer m_server;

};

#endif // SECUREWEBSERVER_H
```

아래 예제 소스코드는 SecureWebServer.cpp 소스코드 이다.

```
#include "SecureWebServer.h"
#include <QCoreApplication>
#include <QFile>
#include <QDebug>

#if QT_CONFIG(ssl)
```

```
# include <QSslCertificate>
# include <QSslKey>
#endif

SecureWebServer::SecureWebServer(QObject *parent)
    : QObject{parent}
{
    m_server.route("/", [] () {
        return "hello world";
    });
}

#if QT_CONFIG(ssl)
const auto sslCertificateChain =
    QSslCertificate::fromPath(QStringLiteral(":/test.crt"));

QFile privateKeyFile(QStringLiteral(":/test.key"));
if (!privateKeyFile.open(QIODevice::ReadOnly)) {
    qDebug() << "Couldn't open file for reading: %1"
        << privateKeyFile.errorString();
    return;
}

m_server.sslSetup(sslCertificateChain.front(),
                  QSslKey(&privateKeyFile, QSsl::Rsa));
privateKeyFile.close();

// [ URL ] https://127.0.0.1:8001
const auto sslPort = m_server.listen(QHostAddress::Any, 8001);
if (!sslPort)
    qDebug() << "Server failed to listen on a port.";

#else
    qDebug() << "This Web Server does not support SSL.";
#endif
```

예수님은 당신을 사랑합니다.

```
}
```



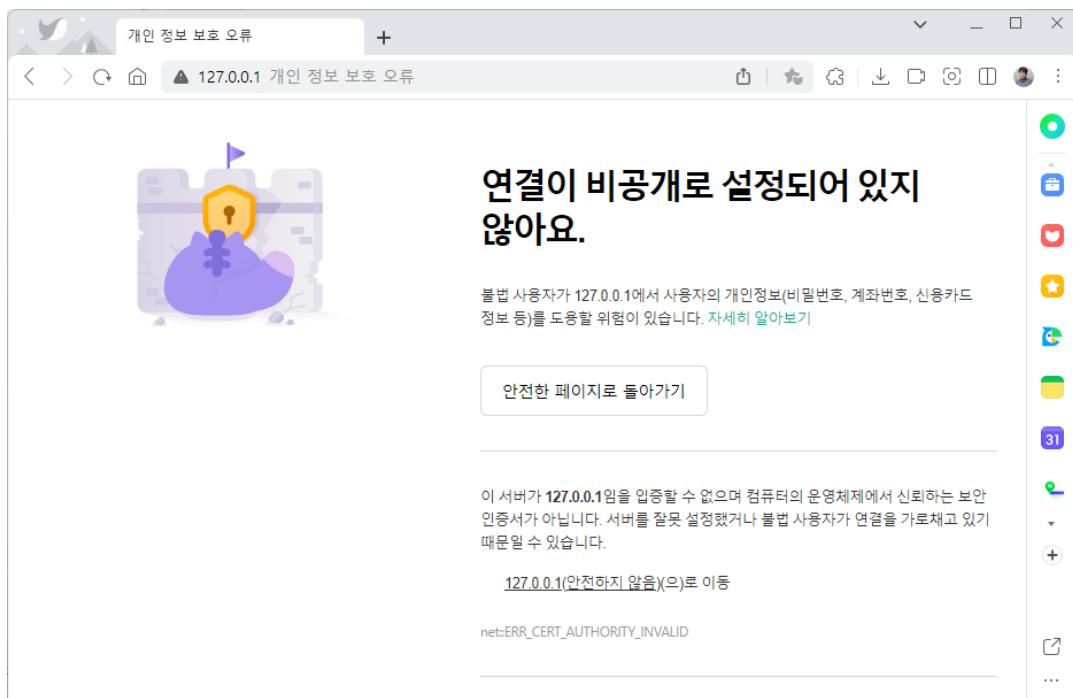
```
SecureWebServer::~SecureWebServer()
{
    deleteLater();
}
```

위의 예제에서 보는 것과 같이 QHttpServer 클래스에서 제공하는 sslSetup() 멤버 함수를 사용해 KEY 를 등록하면 된다.

위와 같이 소스코드 작성 후 빌드 후 실행해 보도록 하자. 그런 다음 Web browser 에서 아래 주소로 Web server에 접속해 보도록 하자.

```
https://127.0.0.1:8001
```

위의 주소로 접속하면 아래와 같은 화면이 로딩 될 것이다.

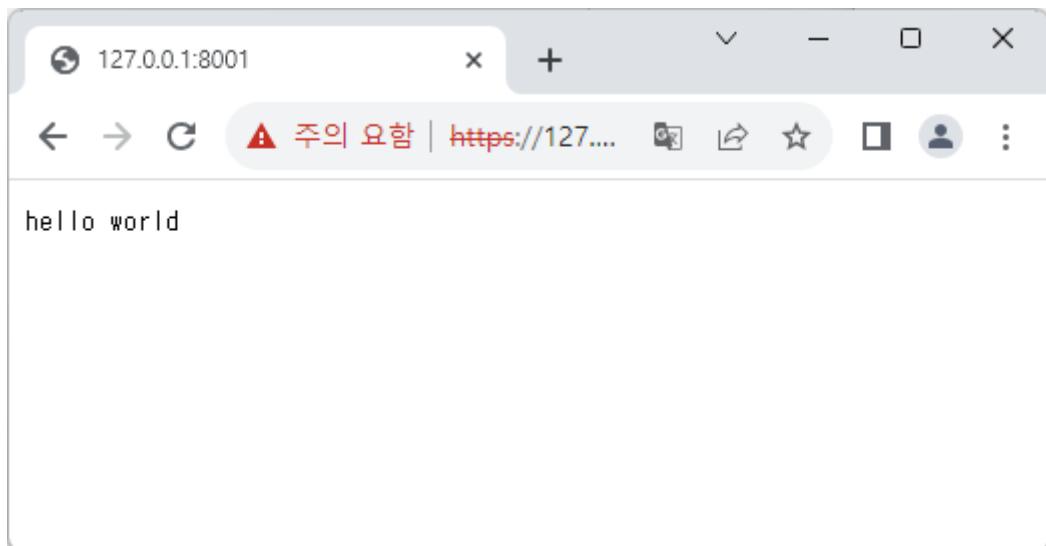


위의 화면에서 보는 것과 같이 안전하지 않다는 것을 확인할 수 있을 것이다. 이 화면이 먼저 로딩되는 이유는 Web browser가 URL이 도용할 수 있는 사이트로 판단하기 때문이다.

위의 문제를 해결하기 위해서 도메인 주소로 SSL 인증서를 발급받으면 된다. 여기서는 임시 URL을 사용하므로 화면 하단의 "127.0.0.1(안전하지 않음)로 이동"을 클릭한다. 그

예수님은 당신을 사랑합니다.

라면 아래와 같이 Web server 로부터 응답 받은 메시지를 확인할 수 있다.



이 예제 소스코드 01_Secure_WebServer 디렉토리를 참조하면 된다.

32. Deployment

이번장에서는 사용자에게 Qt를 이용해 구현한 어플리케이션을 배포하는 방법에 대해서 알아보도록 하자.

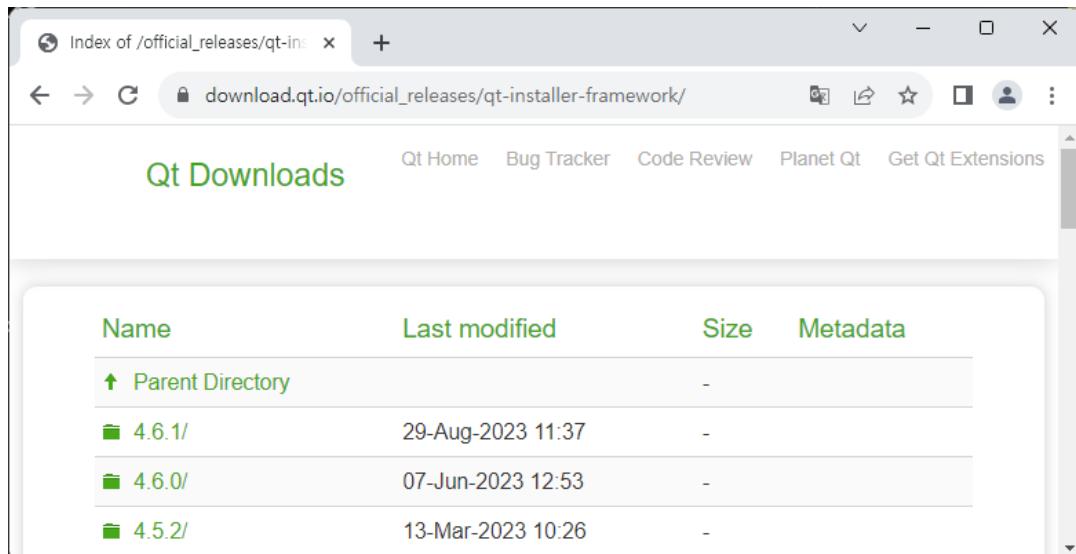
Qt는 사용자에게 배포하기 위해서 Qt Install Framework 툴을 제공한다. 이 툴은 실행 파일과 필요한 라이브러리를 사용자에게 배포 시 설치 파일을 배포하는 것과 같은 설치 패키지로 만들 수 있는 기능을 제공한다.

이 툴은 Qt 설치 시 옵션으로 설치할 수 있다. 그리고 이 툴을 별도로 설치하는 것도 가능하며 MS윈도, Linux 그리고 MacOS 플랫폼을 지원한다. 사용 방법은 동일하다.

- ✓ Qt Install Framework 다운로드 및 설치

https://download.qt.io/official_releases/qt-installer-framework/

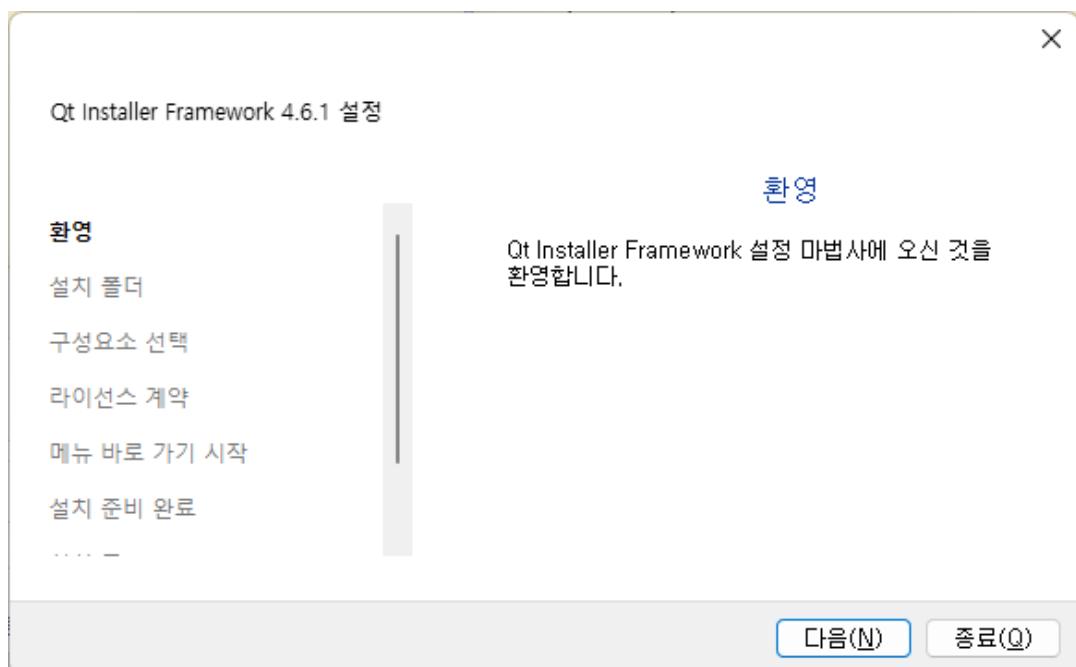
위의 URL에 접속하면 Qt Install Framework를 다운로드 받을 수 있다.



가장 최신을 버전을 클릭한다.

| Name | Last modified | Size | Metadata |
|---|-------------------|------|-------------------------|
| Parent Directory | - | - | - |
| md5sums.txt | 29-Aug-2023 11:37 | 388 | Details |
| installer-framework-everywhere-src-4.6.1.zip | 29-Aug-2023 11:35 | 5.9M | Details |
| installer-framework-everywhere-src-4.6.1.tar.xz | 29-Aug-2023 11:35 | 3.2M | Details |
| QtInstallerFramework-windows-x64-4.6.1.exe | 29-Aug-2023 11:34 | 66M | Details |
| QtInstallerFramework-macOS-x64-4.6.1.dmg | 29-Aug-2023 11:34 | 46M | Details |
| QtInstallerFramework-linux-x64-4.6.1.run | 29-Aug-2023 11:34 | 73M | Details |

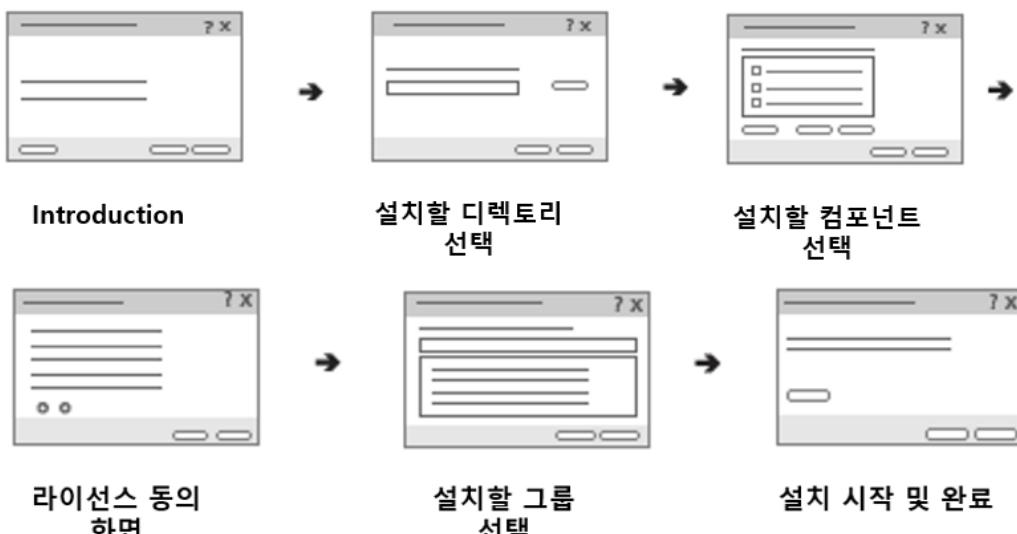
확장자가 "exe" 는 MS윈도우 플랫폼에서 사용할 수 있다. 그리고 확장자가 "run" 은 리눅스이며 "dmg" 는 MacOS 플랫폼에서 사용할 수 있다. 여기서는 "exe" 파일을 다운로드 받아 설치한다.





✓ 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 다운로드 받아 설치하는 방식이다. 그리고 어플리케이션을 배포하기 위한 방법으로 다이얼로그 형태의 Setup Wizard 형식으로 배포판을 제작할 수 있다.



위의 그림에서 보는 것과 같이 Workflow 형태의 템플릿 형태로 Qt Install Framework를

예수님은 당신을 사랑합니다.

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 00_Installer_Example라는 디렉토리를 생성한다. 그리고 생성한 디렉토리에 00_Installer_Example.pro 파일을 다음과 같이 작성한다.

```
TEMPLATE = aux

INSTALLER = 00_Installer_Example

INPUT = $$PWD/config/config.xml $$PWD/packages

myexam.input = INPUT
myexam.output = $$INSTALLER
myexam.commands = C:/Qt/QtIFW-4.6.1/bin/binarycreator \
                  -c $$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. 여기서는 aux를 입력한다.

INSTALLER 키워드는 설치 파일의 이름을 명시한다. 나중에 빌드하면 생성된 설치 파일 이름이 INSTALLER 키워드로 지정한 이름이 설치파일명으로 생성된다.

다음의 INPUT 키워드는 두 가지 항목을 명시해야 한다. 첫 번째는 config.xml 파일이 있는 위치와 파일명을 입력한다. 두 번째 항목은 packages 디렉토리를 명시한다.

config.xml 파일은 프로젝트의 이름, 버전, 공급자, MS원도우인 경우 시작 메뉴의 이름들을 명시한다.

packages 디렉토리에는 설치될 실행 파일과 라이브러리들을 여기에 복사하면 된다.

myexam.input 키워드는 INPUT 키워드를 명시한다. 두 번째 myexam.output 은 INSTALLER 키워드에서 명시한 설치 파일 명을 명시한다.

myexam.commands 는 설치 파일을 만들기 위해서 Qt Install Framework에서 제공하는 binarycreator가 있는 디렉토리 및 파일 명을 입력한다. "-c" 옵션에는 config.xml 파일의 위치와 이름을 지정한다.

예수님은 당신을 사랑합니다.

"-p" 는 패키지 파일의 디렉토리와 파일 명을 입력한다. 위와 같이 프로젝트 파일을 작성하고 config 디렉토리를 만들고 config.xml 을 아래와 같이 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>
```

이번에는 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, 실행 파일이 사용하는 라이브러리, 그리고 실행파일이 사용하는 리소스 파일 등을 이 디렉토리에 복사한다.

여기서는 Qt 예제 중 아날로그 예제 실행 파일을 MinGW 컴파일러의 Release 모드로 빌드한 후 data 디렉토리에 복사해 넣을 것이다. 이 때 실행 파일이 참조하는 라이브러리도 함께 복사해 넣어야 한다. 예제로 사용한 소스코드는 00_AnalogClock 을 참조하면 된다.

실행 파일이 어떤 라이브러리를 참조하는지 찾기 위해서 MS윈도우 플랫폼에서는 windeployqt.exe 또는 windeployqt6.exe 파일을 활용할 수 있다. 이 툴은 컴파일러마다 위치가 달라 질 수 있다. 예를 들어 MinGW 컴파일러를 사용한다면 아래와 같을 수 있다.

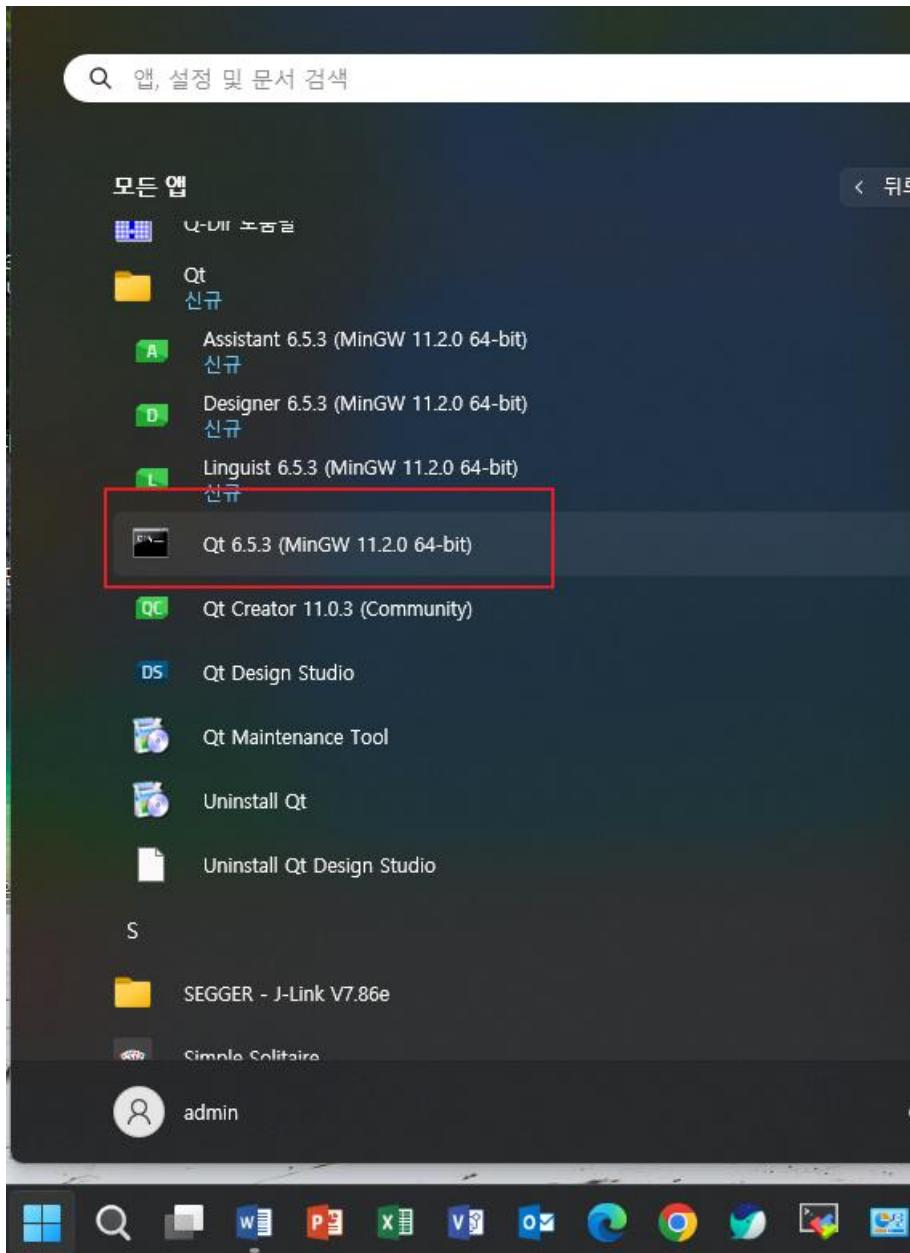
```
C:[ Qt Installed directory ]\mingw_64\bin
```

위의 디렉토리에 가면 windeployqt.exe 는 Qt 5 버전이며 windeployqt6.exe 는 Qt 6 버전에서 사용할 수 있다.

MS윈도우에서 제공하는 명령프롬프트를 사용해도 된다. 하지만 여기서는 Qt 설치 디렉토리가 PATH로 설정된 명령 프롬프트를 사용할 것이다.

예수님은 당신을 사랑합니다.

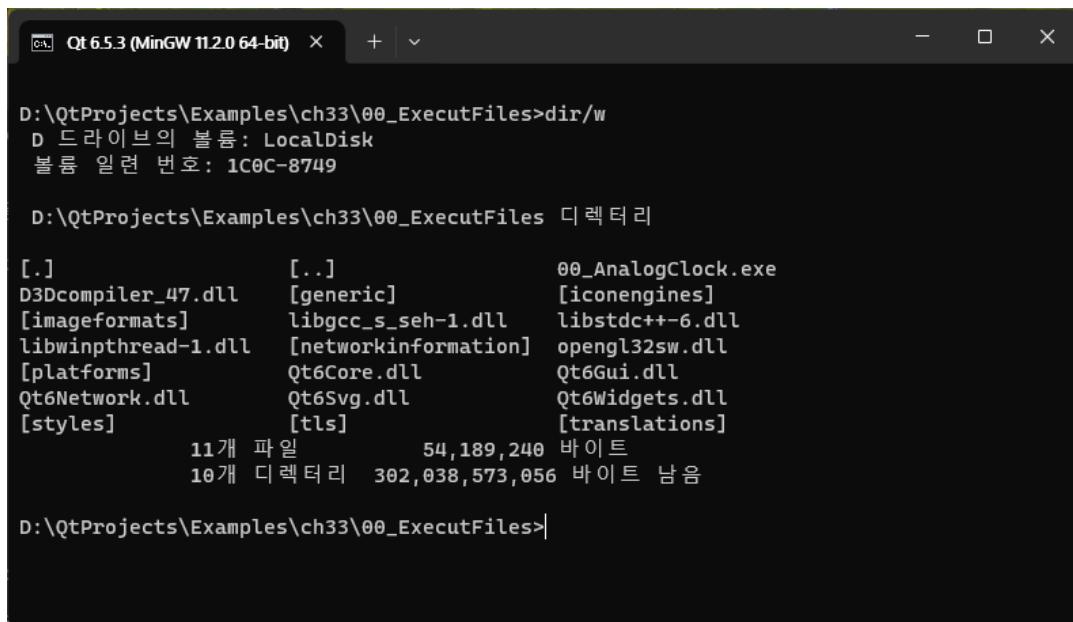
아래 그림에서 보는 것과 같이 Qt에서 제공하는 명령 프롬프트를 실행한다.



그리고 실행파일이 있는 위치로 이동한 다음 아래와 같이 실행한다.

```
> winedeployqt6.exe 00_AnalogClock.exe
```

위와 같이 입력하고 실행하면 00_AnalogClock.exe 이 필요한 파일을 복사까지 자동으로 한다.



```
D:\QtProjects\Examples\ch33\00_ExecutFiles>dir/w
D 드라이브의 볼륨: LocalDisk
볼륨 일련 번호: 1C0C-8749

D:\QtProjects\Examples\ch33\00_ExecutFiles 디렉터리

[.]          [...]          00_AnalogClock.exe
D3Dcompiler_47.dll [generic] [iconengines]
[imageformats] libgcc_s_seh-1.dll libstdc++-6.dll
libwinpthread-1.dll [networkinformation] opengl32sw.dll
[platforms] Qt6Core.dll Qt6Gui.dll
Qt6Network.dll Qt6Svg.dll Qt6Widgets.dll
[styles] [tls] [translations]

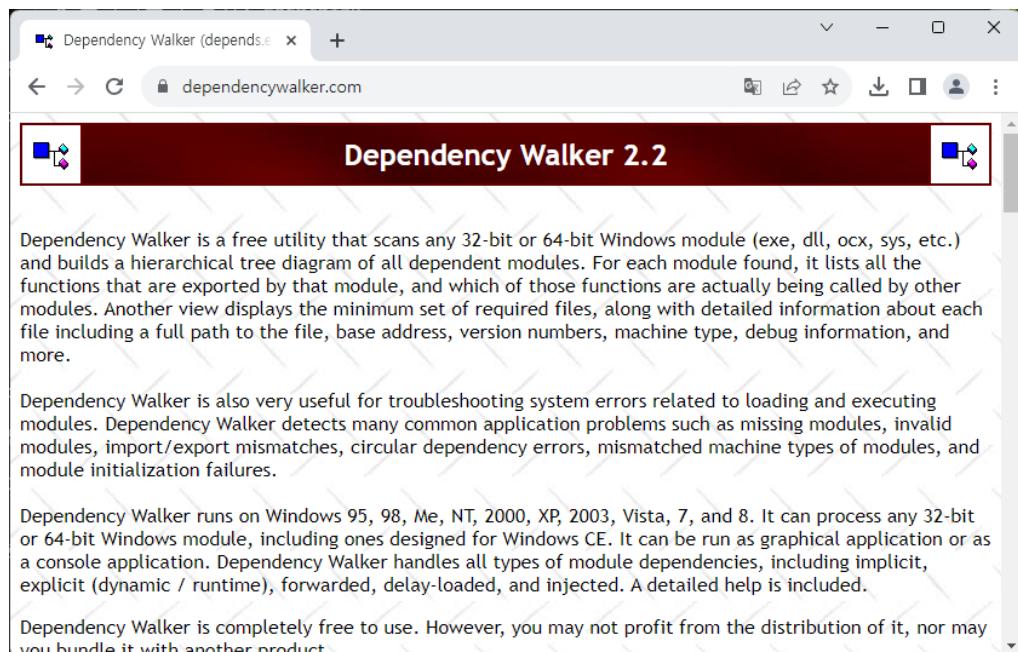
11개 파일      54,189,240 바이트
10개 디렉터리  302,038,573,056 바이트 남음

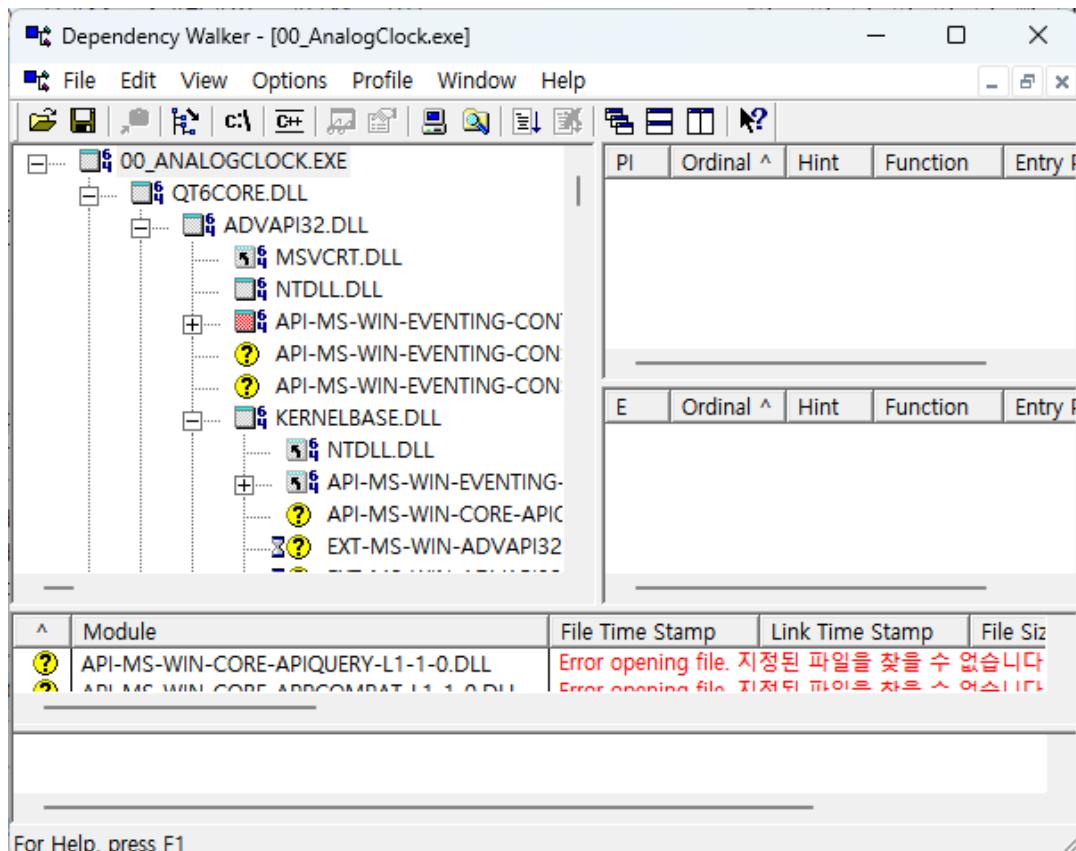
D:\QtProjects\Examples\ch33\00_ExecutFiles>
```

이 툴은 현재 MS 윈도우에서만 제공한다.

이 툴 이외에도 MS 윈도우에서는 참조하는 라이브러리를 확인하기 위해서 Dependency Walker라는 툴을 사용할 수 있다. (이 툴은 Qt에서 제공하는 툴은 아니다.)

이 툴은 참조하는 라이브러리들이 무엇인지 확인할 수 있다. 하지만 이 툴은 자동으로 라이브러리를 찾아서 복사해 주는 기능은 제공하지 않는다.





만약 리눅스인 경우 실행 파일이 참조하는 라이브러리가 무엇인지 확인하기 위해서 ldd 명령어를 사용할 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 플랫폼이 MacOS라면 otool 명령어를 이용해 실행파일이 참조하는 라이브러리를 확인할 수 있다.

```
otool -L MyApp.app/Contents/MacOS/MyApp
```

00_AnalogClock.exe 실행 파일과 필요한 라이브러리들을 com.vendor.product 디렉토리 아래에 meta 디렉토리를 만들 후 다음과 같이 파일을 작성한다.

이 디렉토리에는 main_icon.ico를 필자가 복사해 넣어 두었다. 이 파일은 메뉴에서 아

예수님은 당신을 사랑합니다.

이콘으로 사용할 것이다.

다음으로 com.vendor.product 디렉토리 아래에 meta 디렉토리를 만든 아래와 같이 파일들을 작성한다.

| 파일명 | 설명 |
|------------------|------------|
| package.xml | 패키지 설정 파일 |
| license.txt | 라이선스 정보 파일 |
| installscript.qs | 설치 스크립트 |

package.xml 파일은 표시할 이름, Description, 설치 디렉토리에서 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 예제 입니다.</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 Ink 파일, 실행 아이콘의 위치를 명시한다. Installscript.qs 는 아래와 같이 작성 한다.

```
function Component()
{
    // default constructor
}

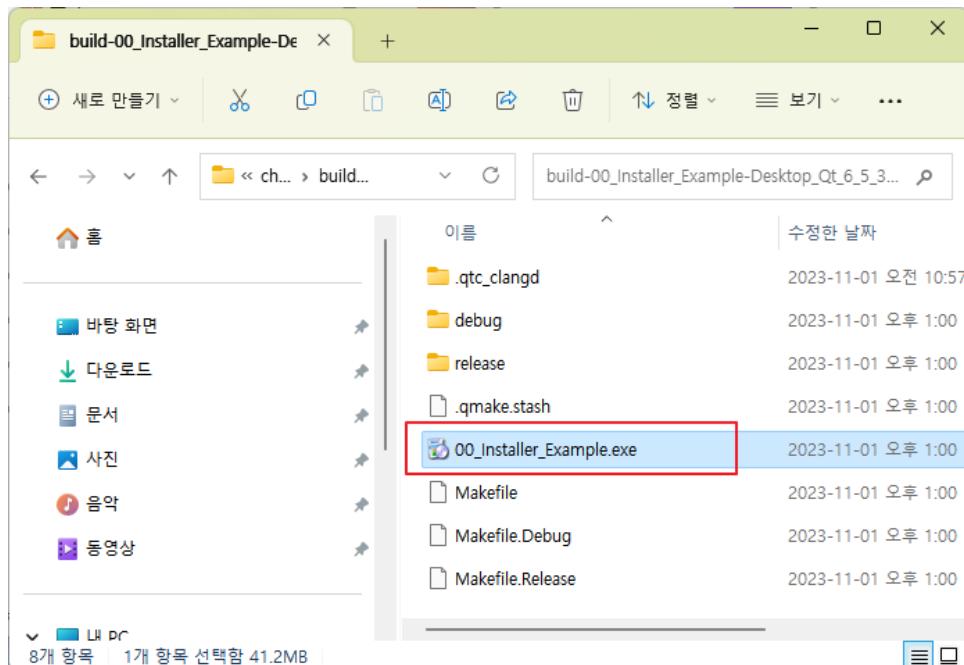
Component.prototype.createOperations = function()
{
    component.createOperations();

    if (systemInfo.productType === "windows")
    {
        component.addOperation(
            "CreateShortcut",
```

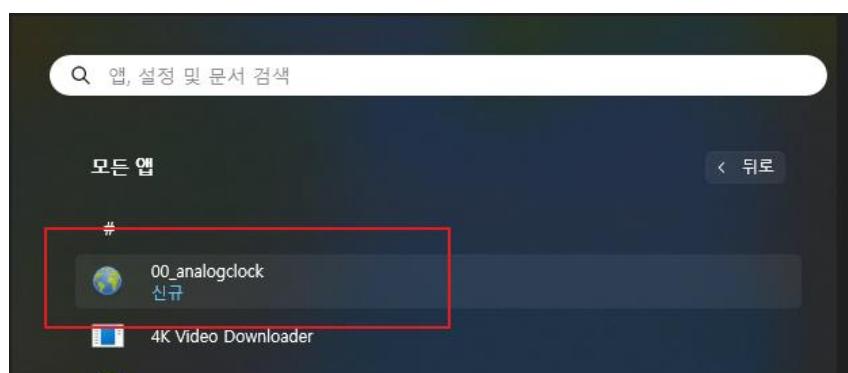
예수님은 당신을 사랑합니다.

```
    "@TargetDir@/00_analogclock",
    "@StartMenuDir@/00_analogclock.lnk",
    "workingDirectory=@TargetDir@",
    "iconPath=@TargetDir@/main_icon.ico");
}
```

위와 같이 작성하였으면 설치 배포판을 만들 모든 준비가 끝났다. Qt Creator 툴에서 00_Installer_Example.pro 파일을 Open 한 후 빌드를 한다. 빌드가 완료되면 빌드 디렉토리에 00_Installer_Example.exe 설치 배포판 파일이 생성된 것을 확인 할 수 있다.



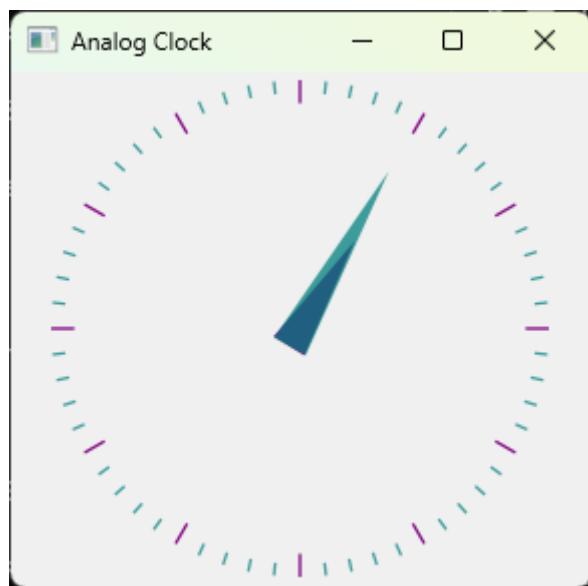
00_Installer_Example.exe 파일을 실행 해보자. 설치 완료 후 메뉴에 보면 메뉴가 등록된 것을 확인할 수 있다.



위의 그림에서 보는 것과 같이 메뉴에서 00_analogclock 을 실행하면 아래와 같이 예

예수님은 당신을 사랑합니다.

제 어플리케이션이 로딩되는 것을 확인할 수 있다.



지금 까지 다른 예제는 00_Installer_Example 디렉토리를 참조하면 된다.

33. Qt Graphics View Framework

어플리케이션을 개발하기 위해서 모니터 보다 큰 윈도우 영역을 사용해야 하는 경우가 있다. 예를 들어 Map 관련 어플리케이션을 구현해야 한다고 가정해 보자.

맵(Map) 상에 건물 등과 같은 수 만개 개 이상의 건물을 맵에 표시해야 한다면 어떻게 구현해야 할까?

아무리 큰 모니터를 가지고 있다고 해도 맵 전체를 모니터에 표시할 수 없을 것이다. 예를 들어 전 세계의 Map 을 모니터 상에 표시할 수 있겠는가? 모니터 크기에 맞게 축소하면 가능하지만 그렇지 않은 경우는 불가능 할 것이다.

그래서 Qt 에서는 맵, 건축 도면, 반도체 설계 도 등과 같은 CAD 등의 어플리케이션 구현 시 쉽게 구현할 수 있도록 Qt Graphics View Framework를 제공한다.

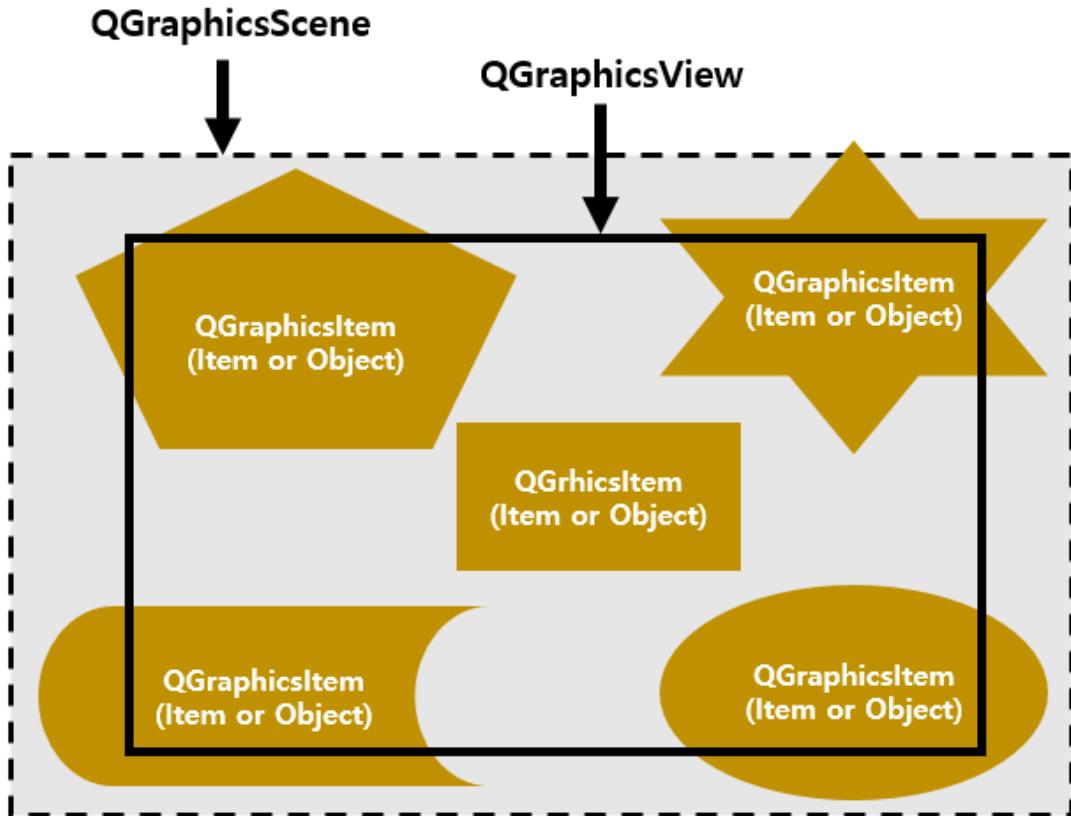
Qt Graphics View Framework 는 확대/축소 등 특정 오브젝트(예를 들어 지도상의 건물) 등을 표시하거나 축소 하는 경우 쉽게 구현이 가능하다.

그리고 Qt Graphics View Framework 는 내부 구현 알고리즘으로 BSP(Binary Space Partitioning) Tree 알고리즘을 사용한다. 따라서 수 많은 오브젝트를 빠르게 표시할 수 있다.

Graphics View Framework 는 요소로 다음과 같이 3가지를 제공한다.

| 요소 | 설명 |
|----------------|--|
| QGraphicsView | 시각적으로 표시가 가능한 영역. 예를 들어 GUI에서 특정 영역. 이 클래스는 GUI 상에서 실제 크기의 영역이다. |
| QGraphicsScene | QGraphicsScene 클래스는 논리 적인 영역 이다. QGraphicsView 영역 안에 표시된다. |
| QGraphicsItem | QGraphicsScene 클래스 오브젝트 상에 표시된다. 예를 들어 맵 상에 표시되는 오브젝트와 같은 개념으로 사용된다. |

다음 그림은 Qt Graphics View Framework 의 3가지 요소를 도식화한 구조이다.



모니터 하면에 GUI기반의 어플리케이션이 있다고 가정해 보자. GUI 기반의 어플리케이션의 GUI의 크기가 QGraphicsView 이다. 그리고 QGraphicsScene 은 가상의 영역이다. 그리고 QGraphicsItem 은 QGraphicsScene 에 배치된 아이템들이다.

예를 들어 QGraphicsItem 은 맵 상에서 건물(Item 또는 Object) 이다. 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);
    }
}
```

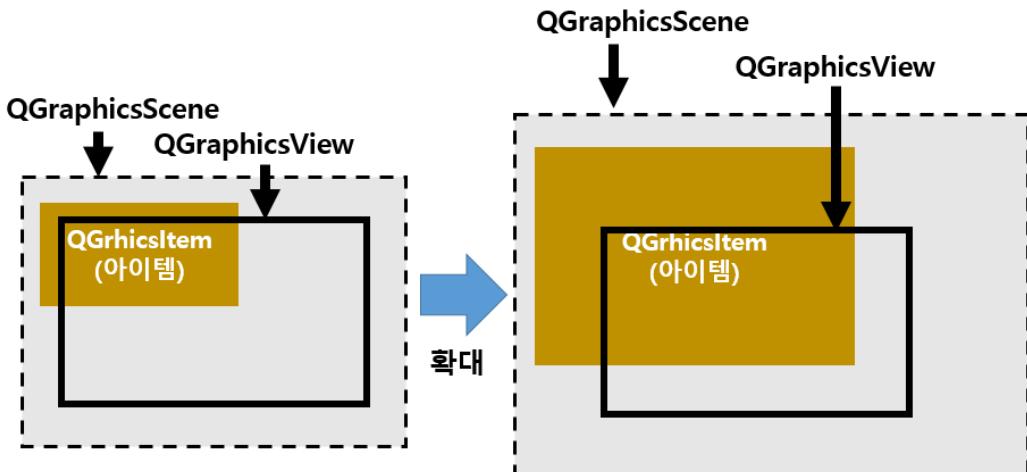
```
void paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
          QWidget *widget) override
{
    painter->drawRoundedRect(-10, -10, 20, 20, 5, 5);
}
};

...
```

위의 예제 소스코드에서 paint() 함수는 QWidget의 paintEvent() 함수와 동일한 기능을 제공한다.

QGraphicsItem 외에도 QGraphicsRectItem, QGraphicsTextItem, QGraphicsPixmapItem 등 다양한 형태의 아이템 클래스를 제공한다.

Qt는 Qt Graphics View Framework에서 제공하는 QGraphicsScene을 QGraphicsView 영역에 맞게 확대/축소할 수 있는 기능을 제공한다.

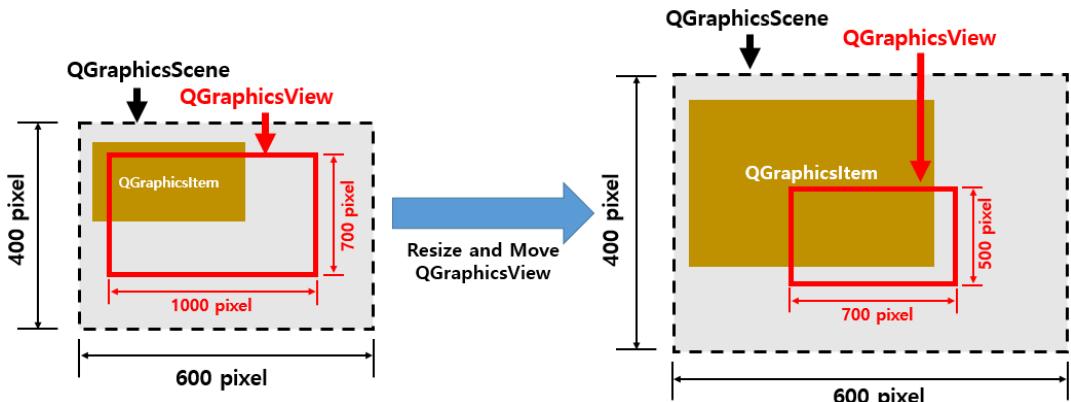


<그림> 확대 했을 때의 도식화한 그림

위의 그림과 같이 Qt Graphics View Framework는 확대 시 QGraphicsView의 크기는 변경되지 않는 상태에서 QGraphicsScene의 크기가 논리적으로 커진다.

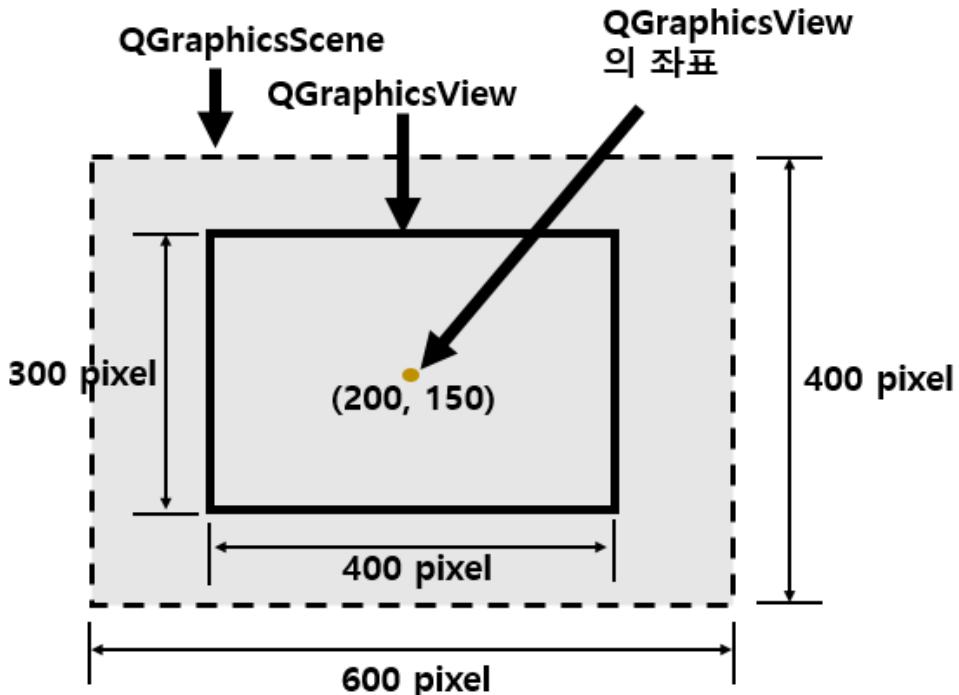
하지만 그렇다고 해서 실제 QGraphicsScene의 가로와 세로 크기가 커지지는 않는다. 또한 QGraphicsScene에 등록된 QGraphicsItem도 마찬가지이다. 다음 그림에서 보는 것과 같이 확대 시 QGraphicsScene의 원래 크기는 변경되지 않으며 축소 시에도 마찬가지다.

예수님은 당신을 사랑합니다.



위의 그림에서 보는 것과 같이 확대 또는 축소 해도 실제 `QGraphicsScene` 의 크기가 변경되지 않는다.

그리고 화면에서 보는 `QGraphicsView` 와 가상의 `QGraphicsScene` 의 좌표계가 서로 다를 수 있다. 예를 들어 다음 그림에서 보는 것과 같이 `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, 300이며 `QGraphicsView` 중심 좌표에 있는 좌표가 200, 150 이라고 가정해 보자.

위의 그림에서 보는 것과 같이 200, 150 좌표는 `QGraphicsView` 의 좌표일 뿐

예수님은 당신을 사랑합니다.

QGraphicsScene 의 좌표가 아니다. 위의 그림에서 예로 QGraphicsView 의 200, 150 의 좌표 점은 QGraphicsScene 에서 대략 300, 200 일 것이다.

Qt Graphics View Framework 는 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 아래와 같이 사용하면 된다.

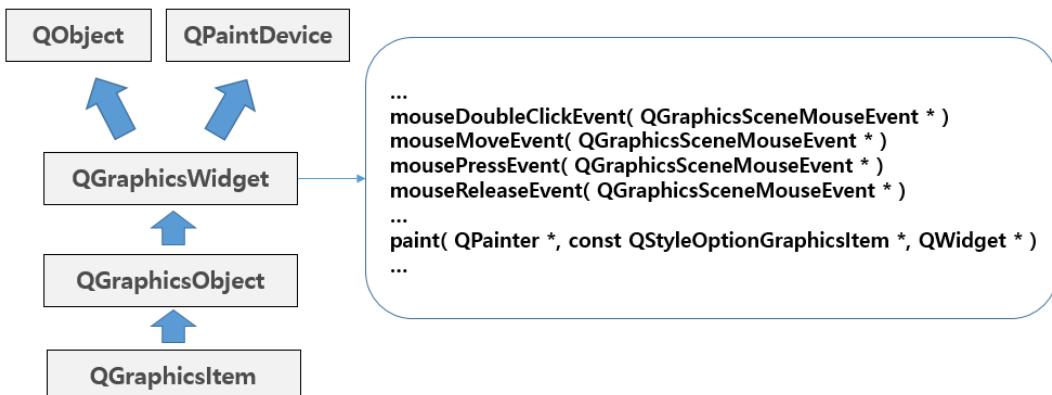
```
...
QGraphicsScene scene;
myPopulateScene(&scene);

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. 그리고 QGraphicsItem 영역에서 사용자 이벤트를 처리할 수 있다.

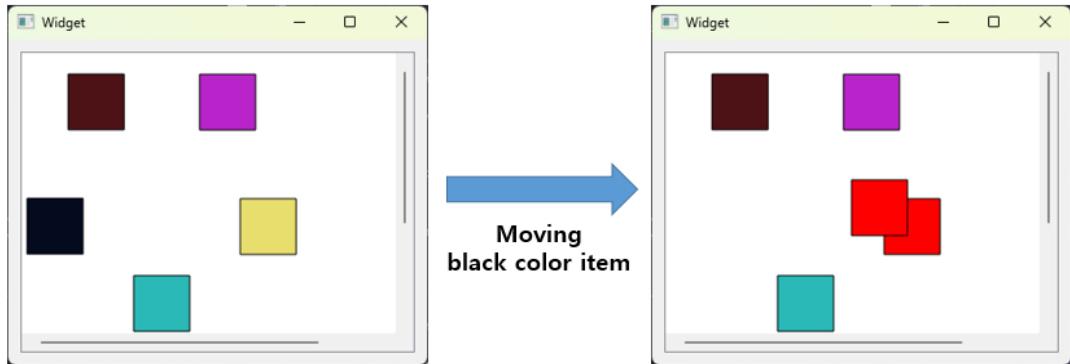


예수님은 당신을 사랑합니다.

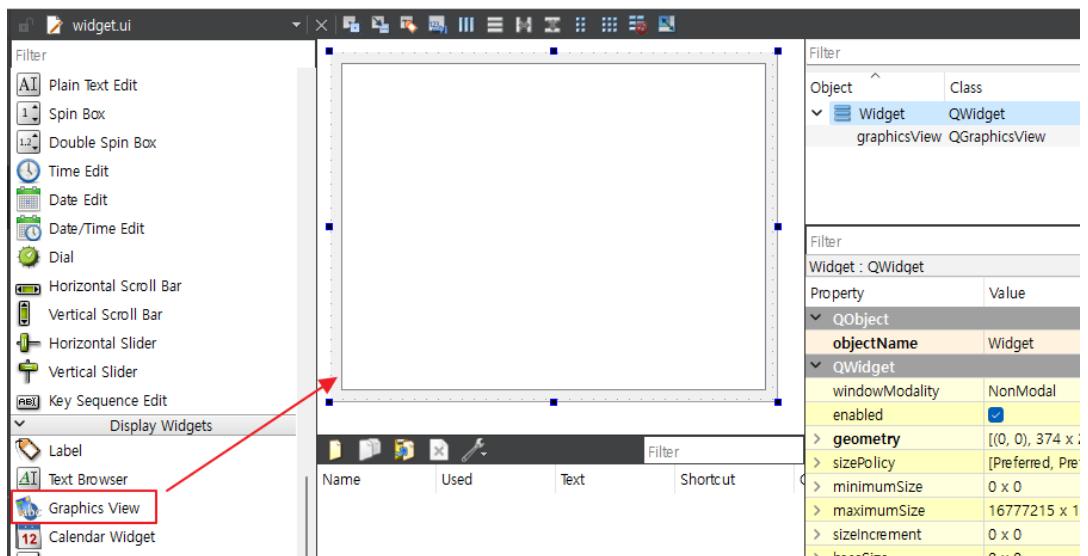
✓ `QGraphicsItem` 의 충돌 감지 구현 예제

이번에 다룰 예제는 `QGraphicsItem` 클래스로 생성한 Shape 아이템이 충돌하는 경우 감지하는 기능을 구현한 예제이다. `QGraphicsScene` 에 등록된 아이템을 마우스로 드래그 할 수 있다.

드래그 한 아이템이 다른 아이템과 충돌하면 충돌된 아이템의 컬러를 빨간색으로 변경 한다.



Qt Creator 상에서 프로젝트 생성 시 `QWidget` 기반의 새로운 프로젝트를 생성한다. 그리고 GUI 폼에 다음과 같이 위젯을 배치한다.



위의 그림에서 보는 것과 같이 좌측 위젯 리스트에서 [Graphics View] 아이템을 GUI 상에 배치한다. 그리고 Shape 라는 새로운 클래스를 프로젝트에 생성한 후 shape.h 헤더 파일에 아래와 같이 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.

Advance() 함수는 QGraphicsScene 업데이트되면 자동으로 이 함수를 호출한다.

Widget 클래스에서 타이머를 사용했다. 이 타이머는 30 Millisecond 마다 호출되어 QGraphicsScene 을 업데이트 한다.

QGraphicsScene 을 업데이트 하면, 이 QGraphicsScene 상에 등록된 Shape 클래스의 advanced() Virtual 함수를 호출 함으로써 Shape 클래스의 paint() Virtual 함수를 호출

예수님은 당신을 사랑합니다.

하게 된다.

그리고 mouseMoveEvent() 함수는 QGraphicsScene 내에서 특정 QGraphicsItem을 마우스로 드래그 하기 위한 기능을 제공한다. 다음 예제 소스코드는 Shape 클래스의 소스코드이다. 아래와 같이 작성한다.

```
#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()->bouned(256),
                  QRandomGenerator::global()->bouned(256),
                  QRandomGenerator::global()->bouned(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 *)
{
```

예수님은 당신을 사랑합니다.

```
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)
{
    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);
    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() 는 아이템이 현재 충돌했는지 알 수 있다.

클래스 생성자에서 설정한 color 변수 값을 사용한다. 그렇지 않고 충돌이 감지 되면 빨간색으로 아이템 Color를 변경한다.

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>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

    QGraphicsScene *m_scene;
    QTimer m_timer;
};

#endif // WIDGET_H
```

이 클래스의 생성자 함수에서 QGraphicsScene 을 생성하고 QGraphicsScene 상에서 QGraphicsItem 을 등록한다.

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.

34. Animation Framework and State Machine

GUI상에서 화면 전환 시 부드러운 화면 전환과 같은 애니메이션 요소를 사용하기 위해서 Qt 에서는 Animation Framework를 제공한다.

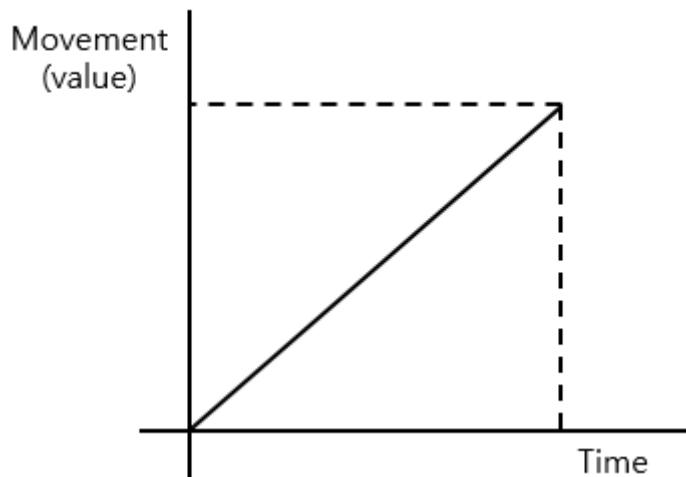
윈도우 화면에서 애니메이션 요소를 사용 가능하며 GUI 상에 배치된 위젯들도 각각의 애니메이션 요소를 사용할 수 있다.

Animation Framework은 QML에서 자주 사용한다. 하지만 C++에서도 Animation Framework를 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 애니메이션으로 처리 할 수 있다.

또한 동시에 Opacity(투명)를 사용해 설정 값을 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.

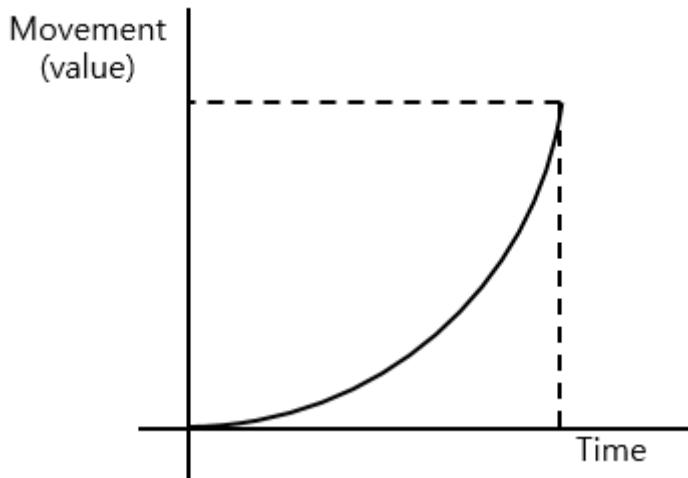
추가로 애니메이션에 Easing Curves를 추가해 사용할 수 있다. 예를 들어 1.0 초 동안 GUI 버튼의 위치(X, Y)가 (100, 100)에서 (200, 200) 좌표로 이동하고, 이동하는 시간을 1.0 초로 지정했다고 가정해보자. 이때 시작 좌표부터 목적지 좌표까지 이동하는데 일정한 속도로 이동할 것이다.



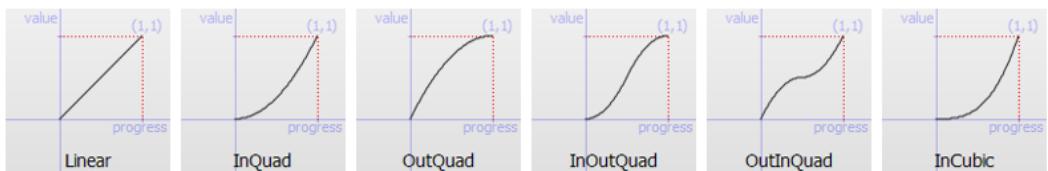
예수님은 당신을 사랑합니다.

위의 그림에서 보는 것과 같이 이동(값)이라는 Y축은 GUI 버튼의 x, y 위치 좌표가 100, 100에서 200, 200 값으로 이동하는 데 1.0 초 시간이 일정하게 소요되는 것을 나타내는 그래프이다. 즉, 이동 값에 따라 시간이 일정하게(Linear) 증가한다.

하지만 시간에 Easing Curves를 사용하면 이동 값에 시간의 값을 다음 그림에서 보는 것과 같이 변경할 수 있다.



최종 목적지의 X, Y 좌표까지 이동하는 시간의 값을 위의 그림에서 보는 것과 같이 적용이 가능하다.



위의 그림에서 보는 것 이외에도 40가지가 넘는 Easing Curve 타입을 사용할 수 있다. 그리고 Qt는 Animation 요소에 State Machine이라는 기법을 사용할 수 있다. 예를 들어 ON/OFF 스위치를 예를 들 수 있다.

한가지 값을 변경하기 위해 ON/OFF를 사용할 수 있을 뿐만 아니라 여러가지 옵션을 함께 사용할 수 있다. 예를 들어 집 현관의 형광등과 거실의 형광등을 함께 켜지게 하고자 할 때 State Machine이라는 기법을 사용해 함께 켜지거나 꺼지게 할 수 있다. 즉 어떤 상황이 되면 그 상황에 따라 여러가지 값이 동시에 적용되도록 사용할 수 있다.

GUI 상에 A버튼, B버튼 그리고 C버튼이 있다고 가정해 보자. A버튼을 누르면 GUI 상에 C버튼의 위치를 100, 100의 위치에서 200, 200으로 이동하고 투명을 1.0에서 0.0으로

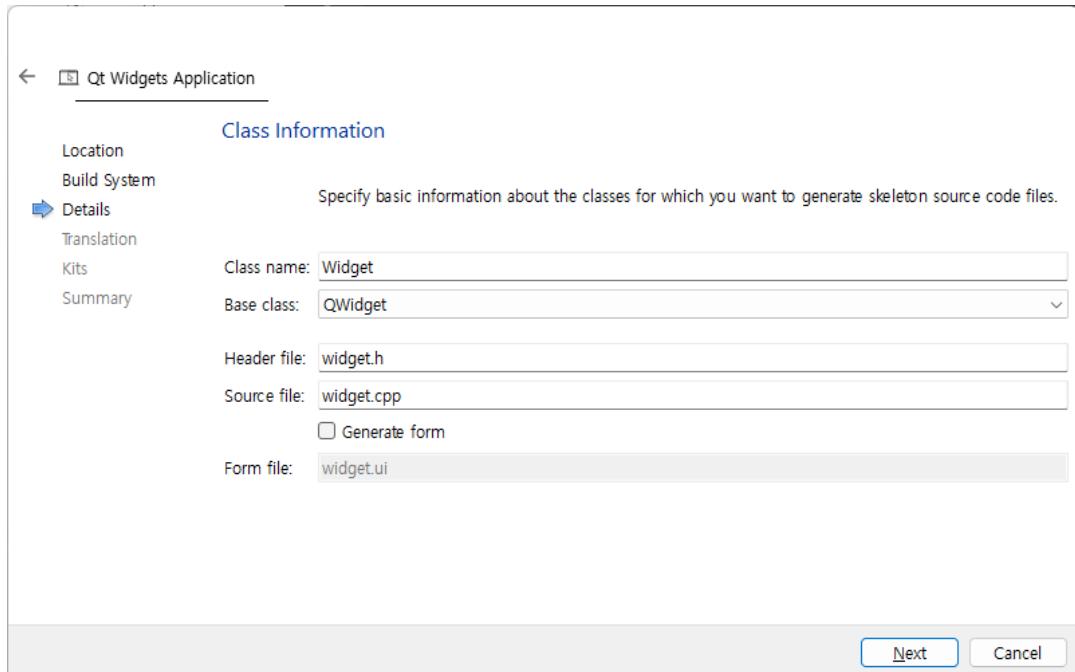
예수님은 당신을 사랑합니다.

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.

- ✓ GUI 상에서 Animation 을 사용한 예제

프로젝트 생성 시 Qt Widget 기반으로 생성한다. 그리고 프로젝트 생성 시 widget.ui 는 사용하지 않는다.



이번 예제에서는 GUI 상에 QPushButton 을 배치한다. 배치한 QPushButton을 특정 위치로 이동하기 위해서 Animation 요소를 사용한다.

버튼을 클릭하면 x, y 위치가 10, 10 의 위치에서 200, 150 의 위치로 이동한다. 이동하는데 시간은 3000 밀리세컨드이다. 다음은 widget.h 헤더파일 이다. 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QPropertyAnimation>

class Widget : public QWidget
```

예수님은 당신을 사랑합니다.

```
{  
    Q_OBJECT  
  
public:  
    Widget(QWidget *parent = nullptr);  
    ~Widget();  
  
private:  
    QPropertyAnimation *animation;  
  
public slots:  
    void btnClicked();  
  
};  
#endif // WIDGET_H
```

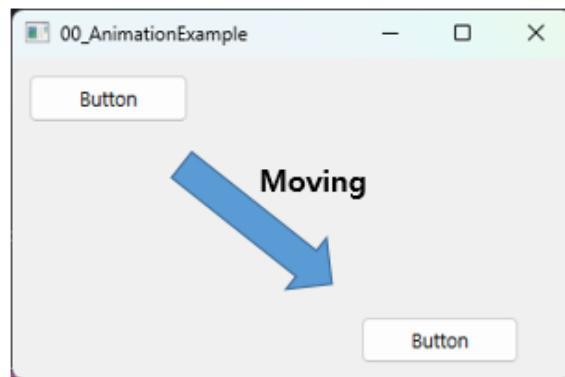
위의 예제 소스코드에서 보는 것과 같이 QPropertyAnimation 클래스의 오브젝트를 선언한다. btnClicked() Slot 함수는 버튼을 클릭하면 호출된다. 다음은 widget.cpp 소스코드이다.

```
#include "widget.h"  
#include <QPushButton>  
  
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); // 3 seconds  
    animation->setStartValue(QRect(10, 10, 100, 30)); // Start position  
    animation->setEndValue(QRect(200, 150, 100, 30)); // End position  
}  
  
void Widget::btnClicked()  
{  
    animation->start();  
}
```

```
Widget::~Widget()
{
}
```

Animation 은 QPropertyAnimation 클래스의 Object이다.

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. btnClicked() Slot 함수는 버튼 클릭 시 호출 된다.



위의 그림에서 보는 것과 같이 버튼을 클릭하면 x, y 좌표가 200, 150 로 3초간 이동한다. 다음은 이 예제에 다음과 같이 Easing Curve를 사용해 보도록 하자.

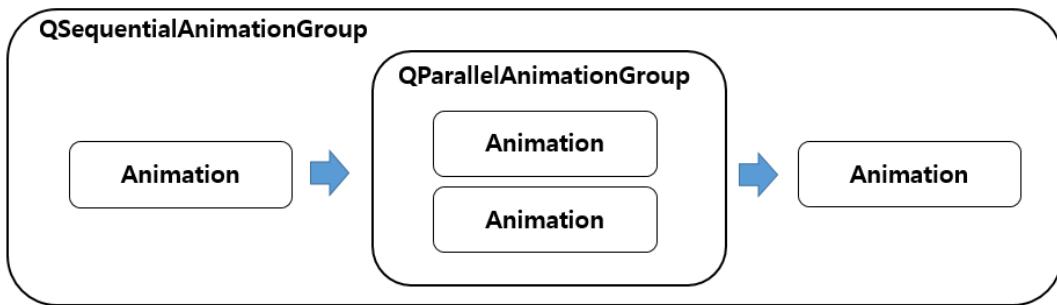
아래에서 보는 것과 같이 Widget 클래스 생성자 함수 마지막에 소스코드를 추가 하자.

```
...
// 아래 소스코드 추가
animation->setEasingCurve(QEasingCurve::OutInQuart);
...
```

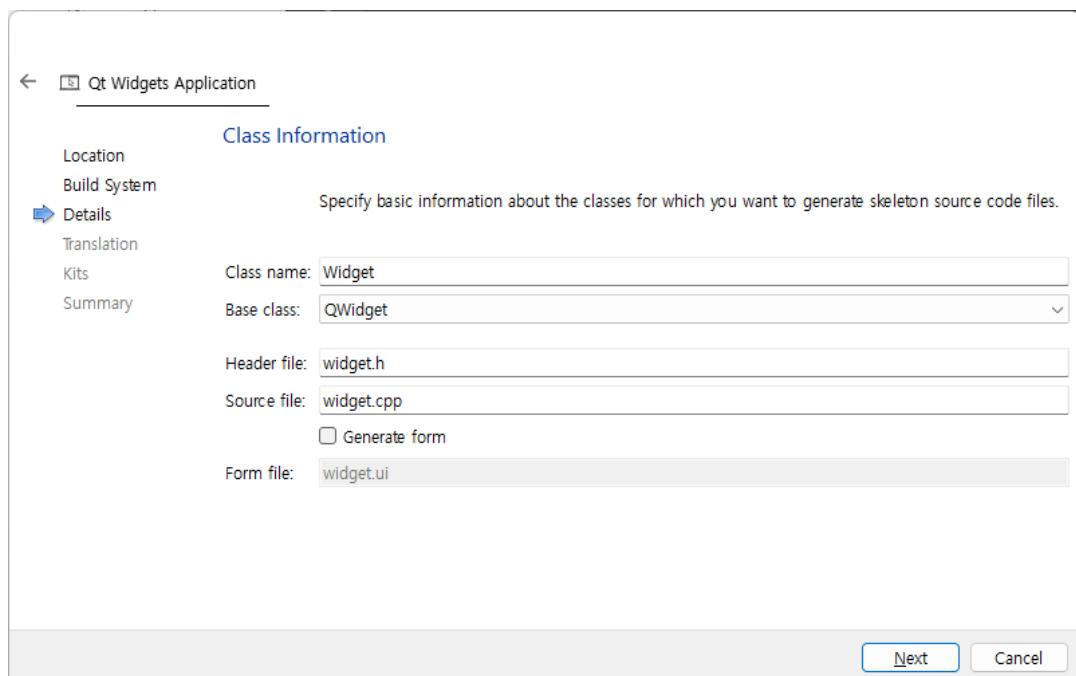
지금 다룬 이 예제 소스코드는 00_AnimationExample 디렉토리를 참조하면 된다.

✓ Animation 그룹을 사용한 예제

이번에 다룰 예제는 그룹화를 사용해 Animation 요소를 동시 또는 순차적 실행하기 위한 방법에 대해서 알아보도록 하자. Animation Framework 에서 사용하고자 하는 애니메이션 요소가 많은 경우 다음 그림에서 보는 것과 같이 그룹화 기능을 이용해 Animation 들을 그룹화 할 수 있다.



QSequentialAnimationGroup 클래스는 Animation을 순차적으로 실행할 수 있다. 반대로 QParallelAnimationGroup 클래스는 애니메이션을 동시(병렬)에 수행 시킬 수 있다. 위의 그림에서 보는 것과 같이 Animation들을 그룹화해 사용하는 예제를 작성해 보도록 하자. 프로젝트 생성 시 Qt Widget 기반으로 프로젝트를 생성한다. 그리고 이 프로젝트에서는 이전 예제와 마찬가지로 widget.ui를 사용하지 않는다.



아래는 widget.h 헤더파일이다. 아래와 같이 작성해 보도록 하자.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QPropertyAnimation>
#include <QSequentialAnimationGroup>
```

예수님은 당신을 사랑합니다.

```
class Widget : public QWidget
{
    Q_OBJECT

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

private slots:
    void slot_sequential();

private:
    QPropertyAnimation *anim1;
    QPropertyAnimation *anim2;
    QSequentialAnimationGroup *sGroup;

};

#endif // WIDGET_H
```

다음 예제 소스코드는 widget.cpp 이다.

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

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); // 2 Seconds (Unit: Miliseconds)
    anim1->setStartValue(QRect(10, 10, 100, 30)); // Start Position
    anim1->setEndValue(QRect(200, 150, 100, 30)); // End Position

    anim2 = new QPropertyAnimation(btn2, "geometry");
    anim2->setDuration(2000); // 2 Seconds (Unit: Miliseconds)
    anim2->setStartValue(QRect(10, 45, 100, 30)); // Start Position
    anim2->setEndValue(QRect(200, 195, 100, 30)); // End Position
```

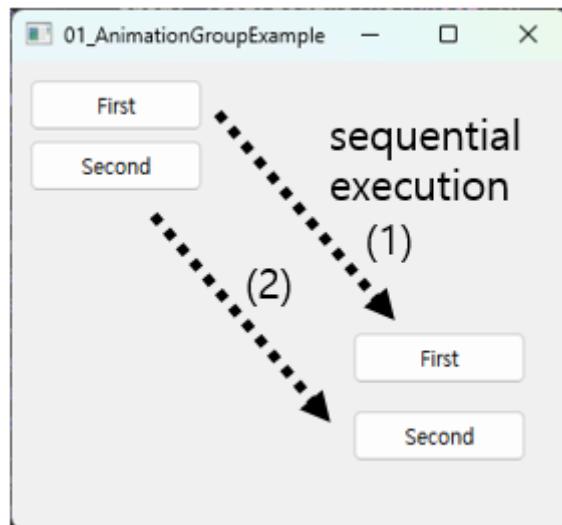
```
sGroup = new QSequentialAnimationGroup; // Sequential Group
sGroup->addAnimation(anim1);
sGroup->addAnimation(anim2);

connect(btn1, SIGNAL(clicked()), this, SLOT(slot_sequential()));

void Widget::slot_sequential()
{
    sGroup->start(QPropertyAnimation::DeleteWhenStopped);
}

Widget::~Widget()
```

위의 예제 소스코드는 두개의 버튼을 배치한다. [First] 버튼을 클릭하면 [First] 버튼을 먼저 이동한 후 [Second] 버튼을 이동한다. 다음 그림은 예제 실행 화면이다.



이번에는 동시에 버튼이 이동되도록 수정해 보도록 하자. 위에서 작성한 Widget 클래스의 소스코드를 아래와 같이 수정해 보도록 하자. 먼저 widget.h 헤더 파일을 먼저 아래와 같이 변경 하자.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
```

예수님은 당신을 사랑합니다.

```
#include <QPropertyAnimation>
#include <QParallelAnimationGroup>

class Widget : public QWidget
{
    Q_OBJECT

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

private slots:
    void slot_parallel();

private:
    QPropertyAnimation *anim1;
    QPropertyAnimation *anim2;
    QParallelAnimationGroup *pGroup;

};

#endif // WIDGET_H
```

다음은 widget.cpp 소스코드를 아래와 같이 수정한다.

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

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); // 2 Seconds (Unit: Miliseconds)
    anim1->setStartValue(QRect(10, 10, 100, 30)); // Start Position
    anim1->setEndValue(QRect(200, 150, 100, 30)); // End Position

    anim2 = new QPropertyAnimation(btn2, "geometry");
```

예수님은 당신을 사랑합니다.

```
anim2->setDuration(2000); // 2 Seconds (Unit: Miliseconds)
anim2->setStartValue(QRect(10, 45, 100, 30)); // Start Position
anim2->setEndValue(QRect(200, 195, 100, 30)); // End Position

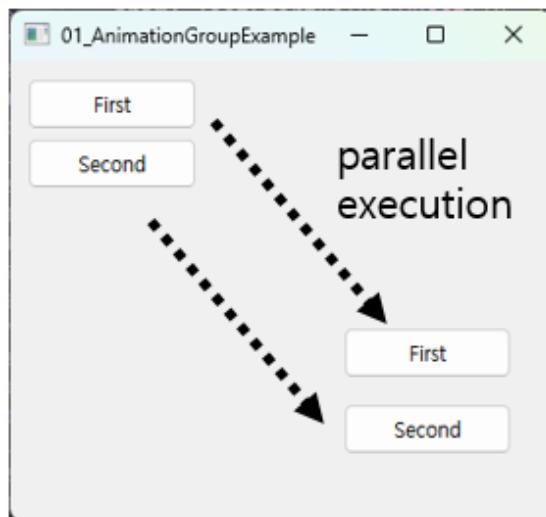
pGroup = new QParallelAnimationGroup; // Parallel Group
pGroup->addAnimation(anim1);
pGroup->addAnimation(anim2);

connect(btn2, SIGNAL(clicked()), this, SLOT(slot_parallel()));
}

void Widget::slot_parallel()
{
    pGroup->start(QPropertyAnimation::DeleteWhenStopped);
}

Widget::~Widget()
{
```

이 예제 소스코드를 빌드한 다음 실행 한다. 그리고 [First] 버튼을 클릭한다. 그러면 [First] 버튼과 [Second] 버튼의 애니메이션이 동시에 실행되는 것을 확인할 수 있을 것이다.

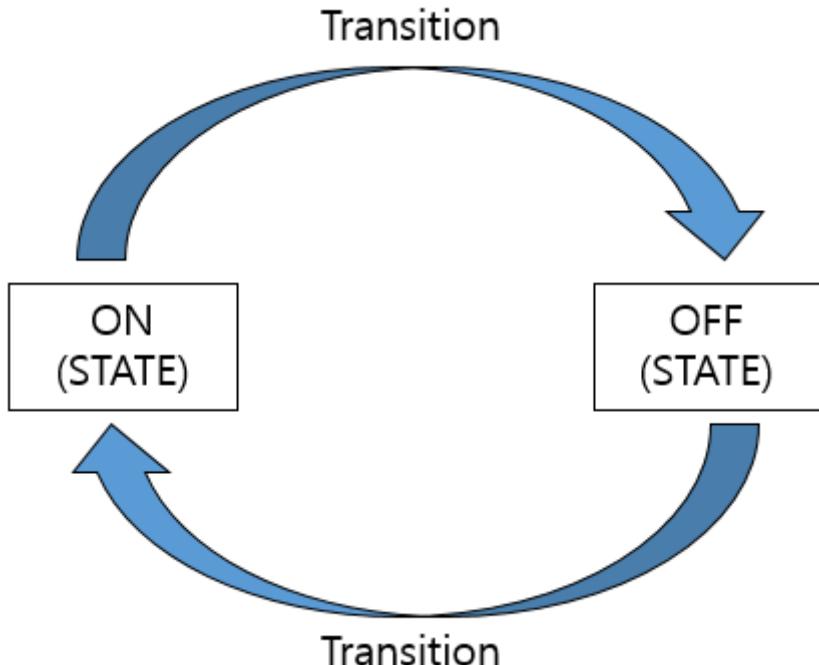


이 예제의 소스코드는 01_AnimationGroupExample 디렉토리를 참조하면 된다.

예수님은 당신을 사랑합니다.

- ✓ Animation 과 State Machine 을 사용한 예

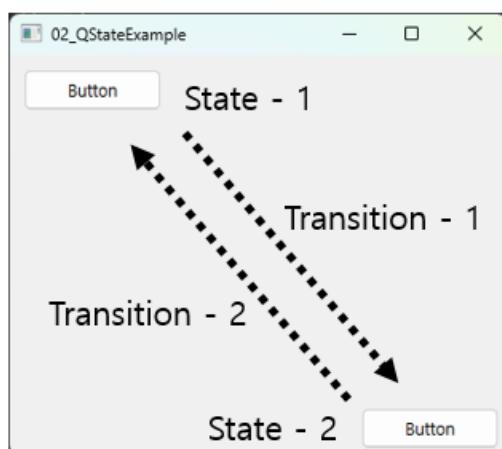
ON State 에서 OFF State로 바뀌는 행동 또는 행위를 Transition 이라고 한다.



State Machine 기법을 Animation 에서 사용하는 이유는 많은 Animation 들을 한꺼번에 처리 할 수 있다.

예를 들어 수십 개의 애니메이션들을 수행한다고 가정해보자 그룹화를 시킬 수 있으나 소스코드가 복잡해 진다. 하지만 State Machine 을 적용하면 쉽게 구현이 가능하다.

State Machine 기법은 State와 Transition 으로 구분된다.



위의 그림은 State Machine 예제를 실행한 화면이다. State - 1 은 버튼의 x, y 좌표가 10,

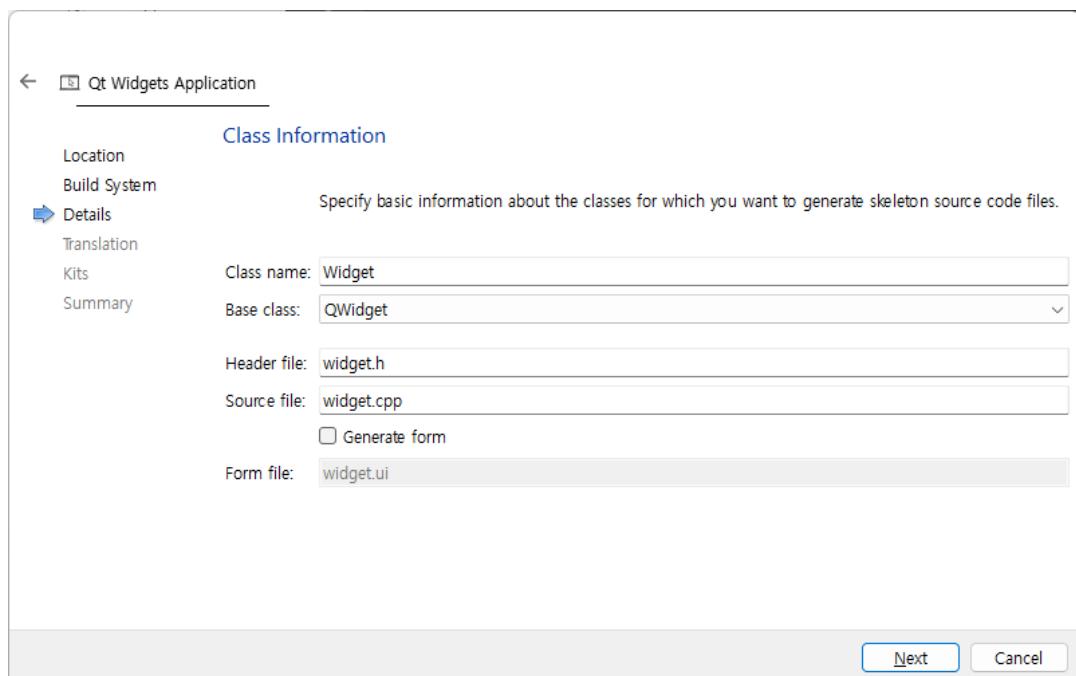
예수님은 당신을 사랑합니다.

10 좌표이다. 그리고 x, y 좌표가 250, 250 인 상태가 State - 2 이다.

State - 1 에서 State - 2 로 이동하는 것을 Transition - 1 이라고 한다. State - 2 에서 State - 1 상태로 이동하는 것을 Transition - 2라고 한다.

따라서 State - 1 상태에서 버튼을 클릭하면 Transition - 2 이 수행 되어 State - 2 상태가 된다.

반대로 State - 2에서 버튼을 클릭하면 Transition - 2 가 수행 되어 State - 1 상태가 된다. 프로젝트 생성 시 Qt Widget 기반 프로젝트를 생성한다. 그리고 이전 예제와 같이 widget.ui를 사용하지 않는다.



프로젝트 생성 후 widget.cpp 소스코드를 아래와 같이 작성한다.

```
#include "widget.h"
#include <QPushButton>
#include <QtStateMachine/QStateMachine>
#include <QSignalTransition>
#include <QStateMachine>
#include <QPropertyAnimation>
#include <QSignalTransition>

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);
state1->assignProperty(button, "geometry", QRect(10, 10, 100, 30));
machine->setInitialState(state1);

QState *state2 = new QState(machine);
state2->assignProperty(button, "geometry", QRect(250, 250, 100, 30));

QSignalTransition *t1 =
    state1->addTransition(button, SIGNAL(clicked()), state2);
t1->addAnimation(new QPropertyAnimation(button, "geometry"));

QSignalTransition *t2 =
    state2->addTransition(button, SIGNAL(clicked()), state1);
t2->addAnimation(new QPropertyAnimation(button, "geometry"));

machine->start();
}

Widget::~Widget()
{
```

이 예제는 02_QStateExample 디렉토리를 참조하면 된다.

35. Qt Chart

Qt 는 쉽게 Chart Component 를 사용할 수 있도록 Qt Chart 모듈을 제공한다. Qt Chart 모듈의 내부는 QWidget 과 QGraphicsWidget 에서 사용할 수 있다. 또한 QML에서도 사용할 수 있다.

Qt Chart 모듈은 CMake 와 qmake 에서 사용할 있다. CMake를 사용하는 경우 프로젝트파일에 아래와 같이 추가하면 된다.

```
find_package(Qt6 REQUIRED COMPONENTS Charts)
target_link_libraries(mytarget PRIVATE Qt6::Charts)
```

만약 qmake를 사용한다면 아래와 같이 프로젝트파일에 아래와 같이 추가해 주면 된다.

```
QT += charts
```

Qt Chart 모듈은 Qt Commercial Licenses 가 있어야 사용할 수 있다. 그렇지 않은 경우 GPLv3 License 하에서 사용해야 한다.

Qt Chart 모듈은 Line, Pie, Bar 등 다양한 그래프로 표현할 수 있는 기능을 제공한다. 예를 들어 Line 차트를 제공하기 위해서 QLineSeries 클래스를 사용하면 된다.

```
QLineSeries* series = new QLineSeries();
series->setName("Line");
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);
```

setName() 멤버 함수는 Legend(범례)의 제목을 설정할 수 있다. Append() 멤버 함수의 첫 번째 인자는 X축의 값이고 두 번째 인자는 Y축의 값이다. Append() 외에도 Operator 를 사용해 데이터를 삽입 할 수 있다.

다음으로 QLineSeries 클래스의 오브젝트를 표시할 QChart 클래스 오브젝트를 아래와 같이 사용하면 된다.

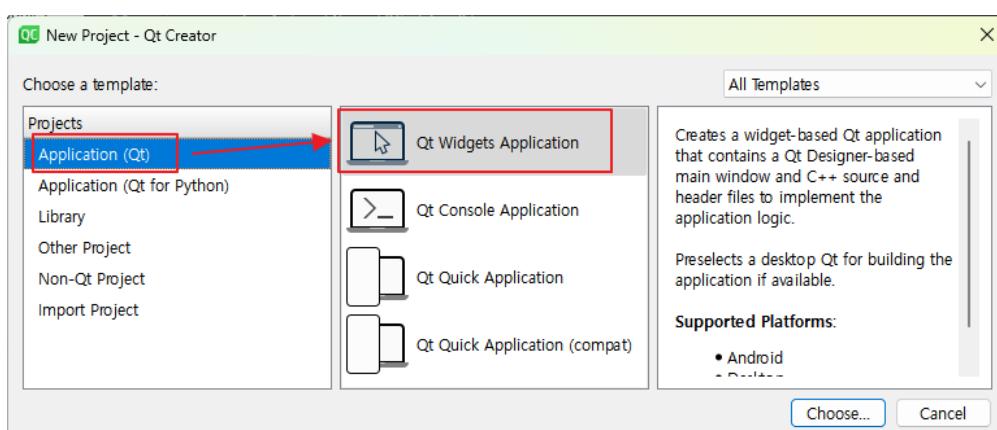
```
QChart *chart = new QChart();
chart->addSeries(series);
chart->createDefaultAxes();
chart->setTitle("Line");
```



QChart 클래스는 여러 개의 QLineSeries 클래스의 오브젝트를 추가할 수 있다. 다음은 예제를 통해서 Qt Chart 모듈을 활용하는 방법에 대해서 좀더 자세히 알아보도록 하자.

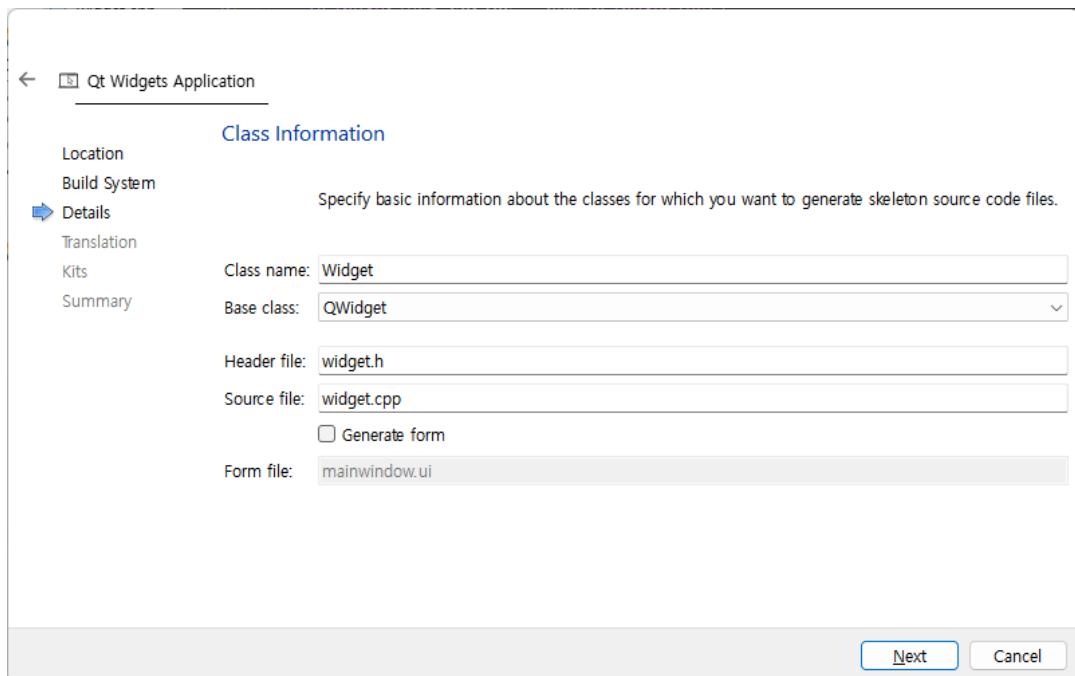
✓ Line and Scatter Chart 예제 구현

프로젝트 생성 시 Qt Widget 기반으로 프로젝트를 생성한다.



예수님은 당신을 사랑합니다.

이 프로젝트에서는 UI form 을 사용하지 않는다. 따라서 프로젝트 생성ダイ얼로그 중 Details 디아일로그에서 [Generate form] 항목의 체크박스를 선택하지 않는다.



프로젝트 생성 후, CMakeList.txt 파일을 열어서 Charts 모듈을 아래와 같이 추가해준다.

```
cmake_minimum_required(VERSION 3.5)
...
find_package(Qt NAMES Qt6 Qt5 REQUIRED COMPONENTS Widgets)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Widgets Charts)
...
target_link_libraries(00_LineAndScatter PRIVATE Qt${QT_VERSION_MAJOR}::Widgets)
target_link_libraries(00_LineAndScatter PRIVATE Qt${QT_VERSION_MAJOR}::Charts)
...
if(QT_VERSION_MAJOR EQUAL 6)
    qt_finalize_executable(00_LineAndScatter)
endif()
```

다음으로 widget.cpp 소스코드 파일을 아래와 같이 작성한다.

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

```
Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    window()->setMinimumSize(600, 400);

    QLineSeries* series = new QLineSeries();
    series->setName("Line");
    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->setName("Spline");
    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->setName("Scatter");
    *series2 << QPointF(1,5) << QPointF(6,6)
        << QPointF(12,3) << QPointF(17,5);

    QChart *chart = new QChart();
    chart->addSeries(series);
    chart->addSeries(series1);
    chart->addSeries(series2);
    chart->createDefaultAxes();
    chart->setTitle("Line and scatter chart");
```

```
QChartView *chartView = new QChartView(chart);
chartView->setRenderHint(QPainter::Antialiasing);

QHBoxLayout *hLay = new QHBoxLayout();
hLay->addWidget(chartView);
setLayout(hLay);

}

Widget::~Widget()
{
}
```

QLineSeries 는 Line Chart를 표시할 수 있다. QSplineSeries 는 곡선 형태의 그래프 차트를 표시할 수 있다. QScatterSeries 는 Scatter 형태로 데이터를 Chart에 표시할 수 있다.

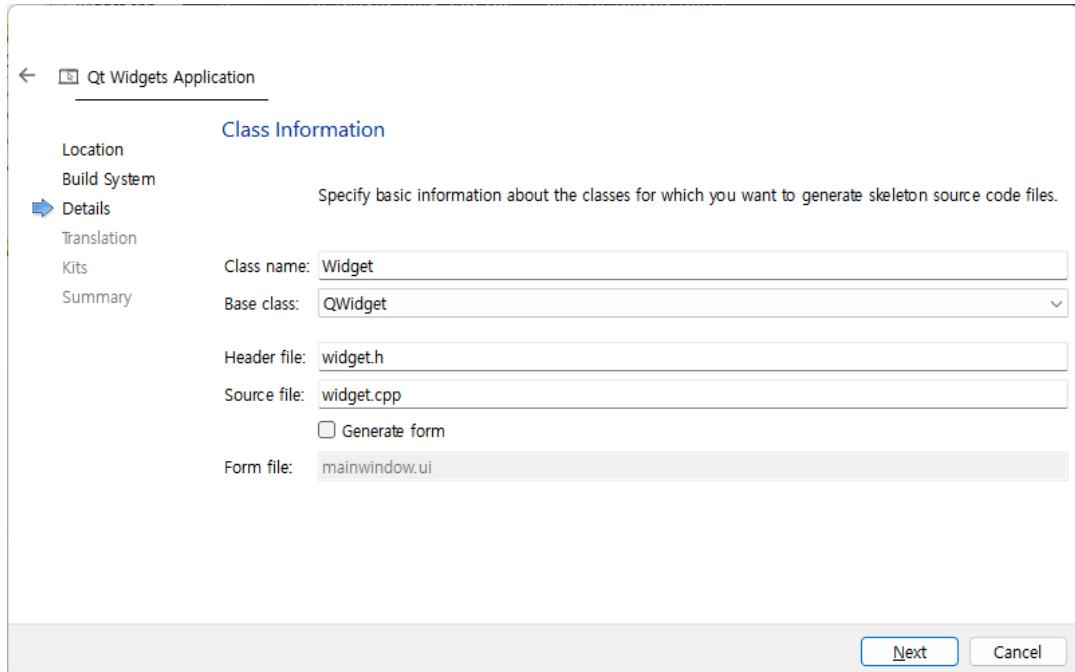
마지막으로 위의 그래프들을 Chart 에 표시하기 위해서 QChart 클래스의 addSeries() 멤버 함수를 사용하면 된다. addSeries() 멤버 함수의 첫 번째 인자에 QLineSeries, QSplineSeries 그리고 QScatterSeries 의 오브젝트를 명시하면 된다.



이 예제는 00_LineAndScatter 디렉토리를 참조하면 된다.

✓ Area Chart 구현

이번 예제에서는 Area Chart를 구현해 보도록 하자. Qt Widget 기반의 프로젝트를 생성한다. 이번 예제에서도 이전 예제에서 마찬가지로 Form UI를 사용하지 않는다. 따라서 프로젝트 생성 시 [Generate form] 항목에 체크박스를 해제한다.



다음으로 widget.cpp 소스코드 파일을 열어서 아래와 같이 작성한다.

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

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    window()->setMinimumSize(600, 400);

    QLineSeries *series1 = new QLineSeries();

    *series1 << QPointF(1, 5) << QPointF(3, 7) << QPointF(7, 6)
        << QPointF(9, 7) << QPointF(12, 6) << QPointF(16, 7)
        << QPointF(18, 5);
```

예수님은 당신을 사랑합니다.

```
QLineSeries *series2 = new QLineSeries();

*series2 << QPointF(1, 3) << QPointF(3, 4) << QPointF(7, 3)
    << QPointF(12, 3) << QPointF(16, 4);

QAreaSeries *series = new QAreaSeries(series1, series2);
series->setName("Area Data");

QPen pen(Qt::blue); // Applying a outline
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); // Applying a gradient

QChart *chart = new QChart();
chart->legend()->hide(); // Hiding legend
chart->addSeries(series);
chart->createDefaultAxes();
chart->setTitle("Area chart");

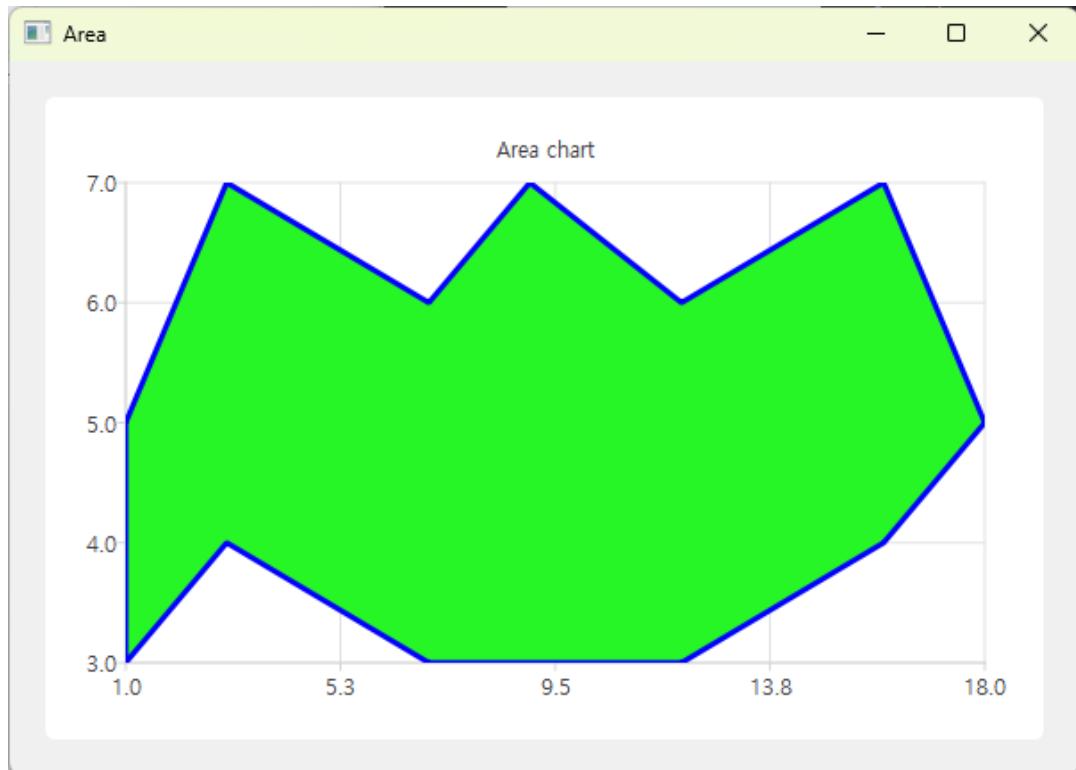
QChartView *chartView = new QChartView(chart);
chartView->setRenderHint(QPainter::Antialiasing);

QHBoxLayout *hLay = new QHBoxLayout();
hLay->addWidget(chartView);
setLayout(hLay);

}

Widget::~Widget()
{
```

Area Chart 를 표시하기 위해서 QAreaSeries 클래스 사용한다. 그리고 데이터를 표시하기 위해서 QLineSeries 를 QAreaSeries 클래스를 사용하면 된다. QLineSeries 클래스 오브젝트를 QAreaSeries 클래스 오브젝트 선언 시, 인자로 넘겨주면 된다.



이 예제의 소스코드는 01_Area 디렉토리를 참조하면 된다.

✓ Bar Chart 예제 구현

이번에는 Bar Chart를 구현해 보도록 하자. Qt 기반의 프로젝트를 생성한다. 그리고 이전 예제와 마찬가지로 UI Form 을 사용하지 않는다. 따라서 프로젝트 생성 시 Detailダイ얼로그에서 [Generate form] 체크박스를 해제한다.

프로젝트 생성이 완료되면 CMakeList.txt 에 Chart 모듈을 추가해 준다. 그런 다음 widget.cpp 소스코드를 연 다음 아래와 같이 작성한다.

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

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    window()->setMinimumSize(600, 300);
```

```
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");
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);

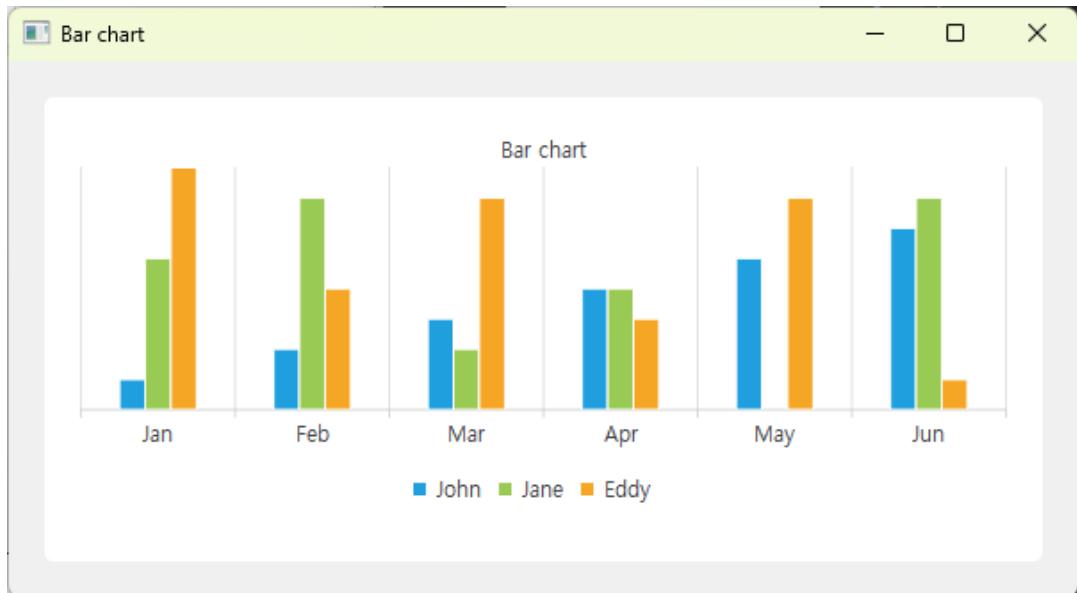
QHBoxLayout *hLay = new QHBoxLayout();
hLay->addWidget(chartView);
setLayout(hLay);

}

Widget::~Widget()
```

```
{  
}
```

Bar 형태의 Chart를 표시하기 위해서 QBarSeries 클래스를 사용한다.



이 예제 소스코드는 02_Bars 디렉토리를 참조하면 된다.

✓ Pie Chart 예제 구현

프로젝트 생성 시 이전 예제 방식과 동일하게 Qt Widget 기반 프로젝트를 생성한다. 또한 UI Form을 사용하지 않는다. 프로젝트 생성이 완료되면 CMakeList.txt 상에 Chart 모듈을 추가해준다. 그런 다음 widget.cpp 소스코드 파일을 아래와 같이 작성한다.

```
#include "widget.h"  
#include <QtCharts>  
  
Widget::Widget(QWidget *parent)  
    : QWidget(parent)  
{  
    window()->setMinimumSize(600, 400);  
  
    QPieSeries *series = new QPieSeries();  
    series->append("Jane", 1); // Inserting name and ratio  
    series->append("Joe", 2);
```

```
series->append("Andy", 3);
series->append("Barbara", 4);
series->append("Axel", 2);

QPieSlice *slice = series->slices().at(1); // Selecting second item
slice->setExploded(); // Separating item
slice->setLabelVisible();
slice->setPen(QPen(Qt::darkGreen, 2));
slice->setBrush(Qt::green);

QChart *chart = new QChart();
chart->legend()->hide();
chart->addSeries(series);
chart->createDefaultAxes();
chart->setTitle("Pie chart");

QChartView *chartView = new QChartView(chart);
chartView->setRenderHint(QPainter::Antialiasing);

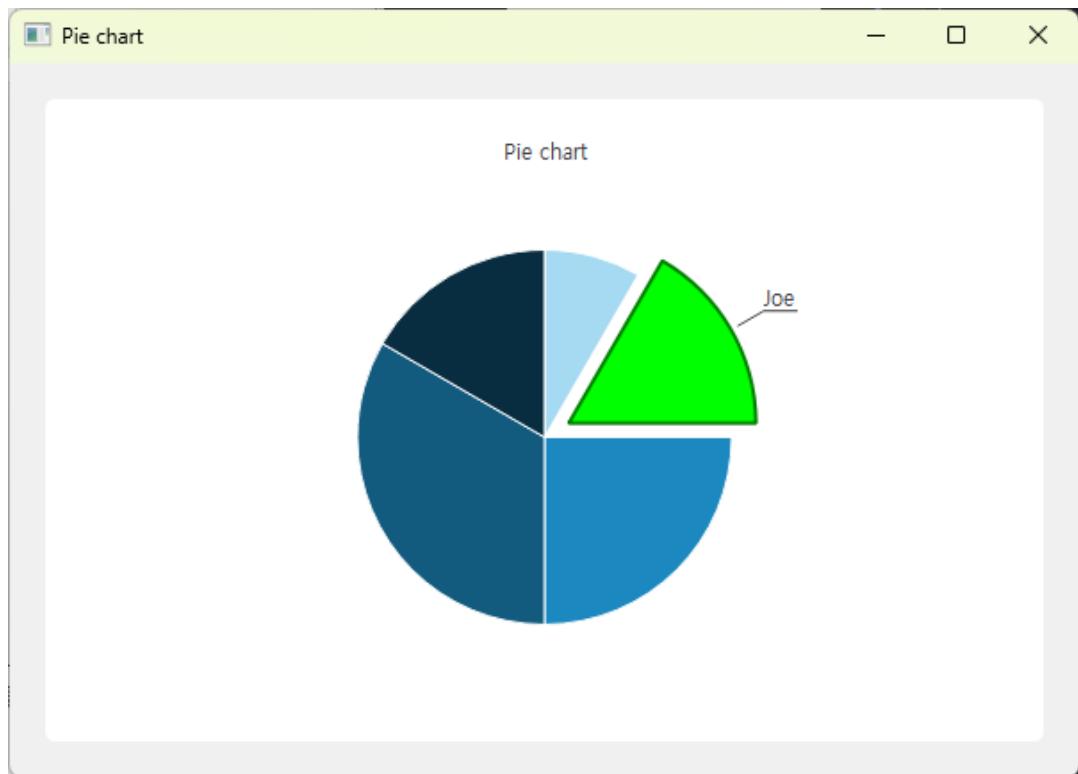
QHBoxLayout *hLay = new QHBoxLayout();
hLay->addWidget(chartView);
setLayout(hLay);

}

Widget::~Widget()
{
}
```

Pie 그래프를 표시하기 위해서 QPieSeries 클래스를 사용한다. 그리고 Pie 중에서 특정 영역을 강조하기 위해서 QPieSlice 클래스를 사용할 수 있다.

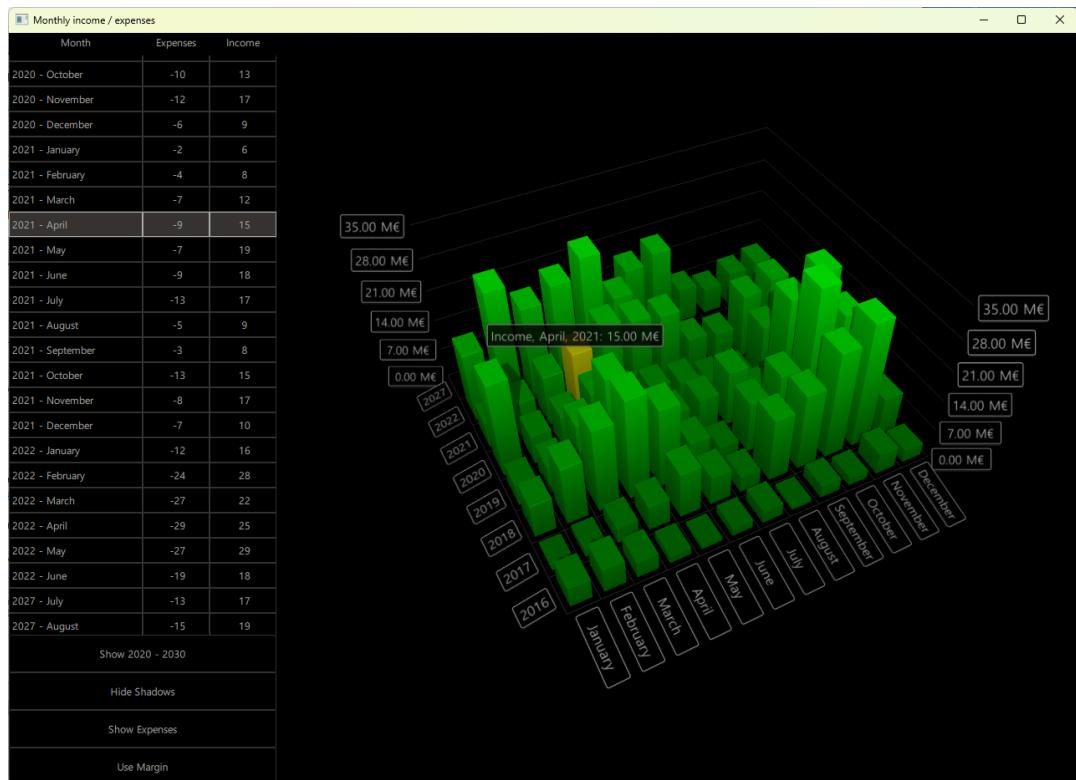
예수님은 당신을 사랑합니다.



이 예제의 소스코드는 03_Pie 디렉토리를 참조하면 된다.

36. Qt Data Visualization

Qt Data Visualization 모듈은 3D를 이용해 데이터를 시각화하는 기능을 제공한다. Bar, Scatter, Surface (예를 들어 육지와 바다의 해수면 표시) 등과 같이 3D로 데이터를 시각화 할 수 있다. Qt Data Visualization 모듈은 OpenGL을 기반으로 하드웨어 가속을 사용하기 때문에 랜더링 속도가 빠르다.



Qt Data Visualization 모듈은 Camera의 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.

프로젝트상에서 CMake를 사용하는 경우 Qt Data Visualization 모듈을 사용하기 위해서는 프로젝트 파일에 다음과 같이 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS DataVisualization)
target_link_libraries(mytarget PRIVATE Qt6::DataVisualization)
```

예수님은 당신을 사랑합니다.

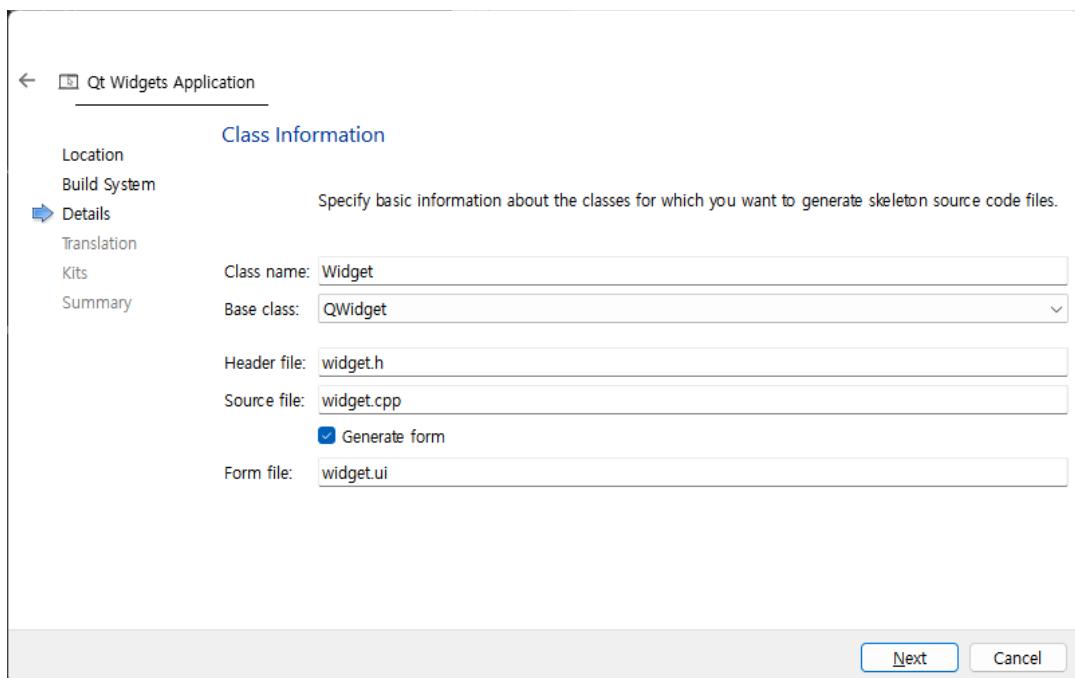
만약 qmake를 사용한다면 프로젝트파일에 아래와 같이 추가해야 한다.

```
QT += dataVisualization
```

✓ Bar 3D 예제 구현

Q3DBars 클래스를 이용하면 3D Bar 3D 그래프를 표시할 수 있다. Q3DBars 클래스는 화면을 상/하/좌/우 이동과 확대/축소를 자유롭게 할 수 있다.

프로젝트 생성 시 Qt Widget 기반의 프로젝트를 생성한다. 그리고 이 프로젝트에서는 UI Form 을 사용하지 않는다. 따라서 프로젝트 생성 다이얼로그 중 Class Information 다이얼로그에서 [Generate form] 체크 항목을 해제한다.



프로젝트 생성이 완료되면 CMakeList.txt 파일에서 DataVisualization를 아래와 같이 추가한다.

```
cmake_minimum_required(VERSION 3.5)

project(00_BarExample VERSION 0.1 LANGUAGES CXX)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
```

```
set(CMAKE_AUTORCC ON)
...
find_package(Qt NAMES Qt6 Qt5 REQUIRED COMPONENTS Widgets)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Widgets)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS DataVisualization)
...
target_link_libraries(00_BaExample PRIVATE Qt6::Widgets)
target_link_libraries(00_BaExample PRIVATE Qt6::DataVisualization)
...
if(QT_VERSION_MAJOR EQUAL 6)
    qt_finalize_executable(00_BaExample)
endif()
```

다음으로 widget.cpp 소스코드 파일을 열어서 아래와 같이 추가한다.

```
#include "widget.h"
#include <QtDataVisualization>
#include <QHBoxLayout>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    window()->setMinimumSize(600, 400);

    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->scene()->activeCamera()->setCameraPreset(
        Q3DCamera::CameraPresetFront);
    bars->rowAxis()->setRange(0,4);
```

예수님은 당신을 사랑합니다.

```
bars->columnAxis()->setRange(0,4);
bars->addSeries(series);

QHBoxLayout *hLay = new QHBoxLayout();
hLay->addWidget(QWidget::createWindowContainer(bars));
setLayout(hLay);

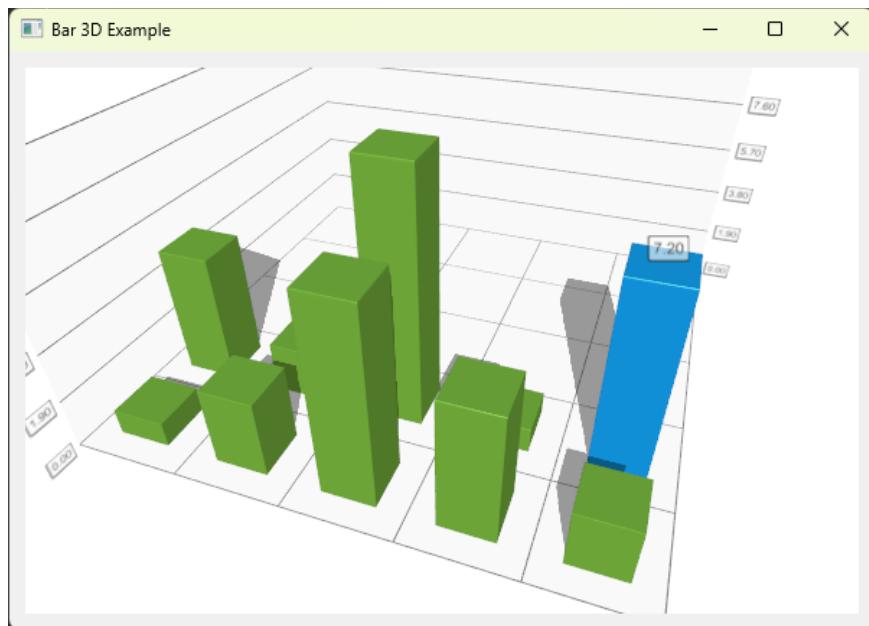
}

Widget::~Widget()
{
}
```

3D Bar 에 데이터를 추가 하기 위해서 QBarDataRow 클래스와 QBar3DSeries를 사용하면 된다.

데이터가 삽입된 QBarDataRow 클래스의 오브젝트를 QBar3DSeries 클래스에서 제공하는 addRow() 멤버 함수를 이용해 오브젝트를 추가하면 된다.

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.



이 예제의 소스코드는 00_BarLayout 디렉토리를 참조하면 된다.

✓ Scatter 3D 예제 구현

Scatter 그래프는 특정 X, Y 좌표에 점을 표시하는 것과 같이 분산 형태의 그래프 종류이다. Qt에서는 Scatter 그래프를 3D로 표현하기 위해서 Q3DScatter 클래스를 제공한다.

프로젝트 생성 시 Qt Widget 기반으로 프로젝트를 생성한다. 이전 예제와 마찬가지로 UI Form을 사용하지 않는다. 그리고 프로젝트 생성이 완료되면 CMakeList.txt 파일에서 DataVisualization을 추가해야 한다.

그런 다음 widget.cpp 소스코드 파일을 열어서 아래와 같이 추가한다.

```
#include "widget.h"
#include <QtDataVisualization>
#include <QHBoxLayout>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    window()->setMinimumSize(600, 400);

    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;

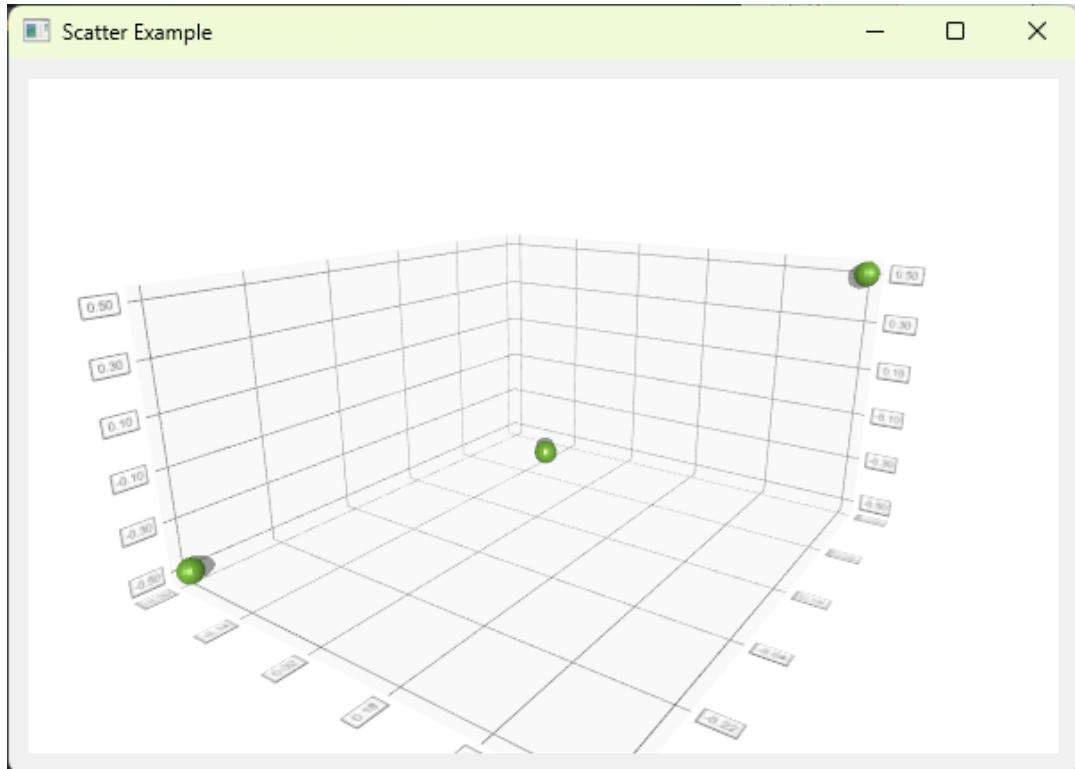
    scatter->scene()->activeCamera()->setCameraPreset(
        Q3DCamera::CameraPresetFront);
    series->dataProxy()->addItems(data);
    scatter->addSeries(series);

    QHBoxLayout *hLay = new QHBoxLayout();
    hLay->addWidget(QWidget::createWindowContainer(scatter));
    setLayout(hLay);
}

Widget::~Widget()
{}
```

예수님은 당신을 사랑합니다.

Q3DScatter 클래스에서 QSCatter3DSeries 와 QScatterdataArray 클래스를 이용해 데이터를 추가할 수 있다.



이 예제의 소스코드는 01_ScatterExample 디렉토리를 참조하면 된다.

✓ Surface 3D 예제

Q3DSurface 클래스를 이용해 Surface(단면) 표시할 수 있다. 예를 들어 지형을 표시할 때 육지의 표면을 시각화 할 수 있다. 이번 예제에서는 간단하게 Q3DSurface 클래스를 이용하는 방법에 대해서 다루어 보도록 하자.

이번 예제도 이전 예제와 같이 UI Form 을 사용하지 않는 프로젝트를 생성한다. 프로젝트 생성이 완료되면 CMakeList.txt 파일에 DataVisualization 모듈을 추가한다.

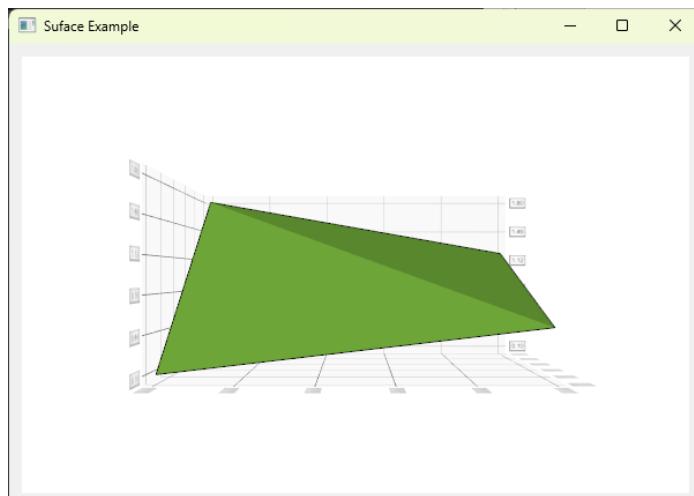
그런 다음 widget.cpp 소스코드 파일을 열어 아래와 같이 작성한다.

```
#include "widget.h"
#include <QtDataVisualization>
#include <QHBoxLayout>

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

```
{  
    window()->setMinimumSize(600, 400);  
  
    Q3DSurface *surface = new Q3DSurface();  
    QSurfacedataArray *data = new QSurfacedataArray;  
  
    QSurfaceDataRow *dataRow1 = new QSurfaceDataRow;  
    *dataRow1 << QVector3D(0.0f, 0.1f, 0.5f) << QVector3D(1.0f, 0.5f, 0.5f);  
    QSurfaceDataRow *dataRow2 = new QSurfaceDataRow;  
    *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);  
  
    QBoxLayout *hLay = new QBoxLayout();  
    hLay->addWidget(QWidget::createWindowContainer(surface));  
    setLayout(hLay);  
}  
  
Widget::~Widget()  
{  
}
```

Surface 형태의 그래프를 표시하기 위해서 Q3DSurface 클래스를 사용하면 된다. 그리고 데이터를 추가하기 위해 QSurfacedataArray 와 QSurfaceDataRow 클래스를 사용하면 된다.



예수님은 당신을 사랑합니다.

이 예제의 소스코드는 02_SurfaceExample 디렉토리를 참조하면 된다.

- ✓ 이미지를 이용한 등고선 3D Surface 표시 예제 구현

이번 예제에서는 아래 그림에서 보는 것과 같이 Height Map 을 이용해 등고선을 표시하는 예제를 구현해 보도록 하자.

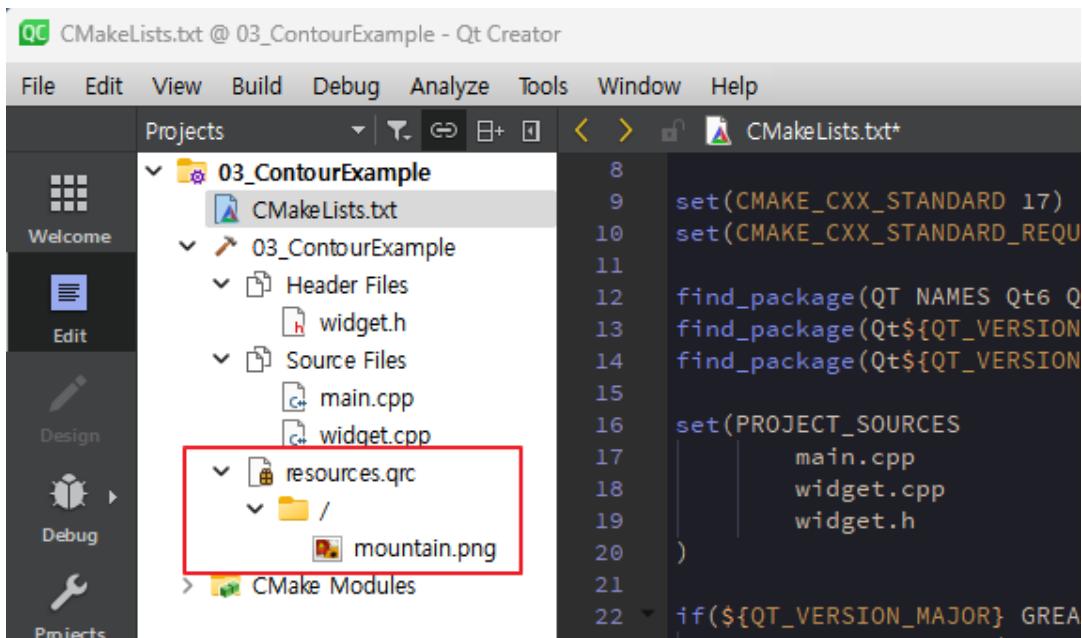


Height Map 은 지형을 위에서 바라본 상태에서 지형의 높이 값을 저장한 Texture 이다. 지리 분야에서는 위성에서 찍은 사진을 Height Map 이미지로 변환하여 일반인 들에게 제공한다.

Height Map 은 가장 낮은 높이의 값은 0이 되고 가장 높은 값은 255 로 표시한 한다.

프로젝트 생성 시, 이전 예제와 동일하게 Qt Widget 기반으로 프로젝트를 생성한다. 프로젝트 생성이 완료되면 CMakeList.txt 에 DataVisualization 모듈을 추가한다.

그런 다음 예제에서 사용할 Height Map 이미지를 리소스에 등록한다. Height Map 이미지는 03_ContourExample 디렉토리에 보면 mountain.png 파일이 있다. 프로젝트 상에 리소스를 추가 후, mountain.png 파일을 리소스에 등록한다.



다음으로 widget.cpp 소스코드 파일에 아래와 같이 작성한다.

```
#include "widget.h"
#include <QtDataVisualization>
#include <QHBoxLayout>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    window()->setMinimumSize(800, 700);

    Q3DSurface *surface = new Q3DSurface;
    surface->scene()->activeCamera()->setCameraPreset(
        Q3DCamera::CameraPresetFront);

    QImage heightMapImage(":/mountain.png");
    QHeightMapSurfaceDataProxy *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->addSeries(series);
surface->setGeometry(50,50,600,400);

QHBoxLayout *hLay = new QHBoxLayout();
hLay->addWidget(QWidget::createWindowContainer(surface));
setLayout(hLay);
}

Widget::~Widget()
{
}
```

이번 예제는 아래 그림에서 보는 것과 같이 등고선 추출을 위해 촬영한 이미지를 이용해 3D Surface를 표시하기 위한 예제이다.

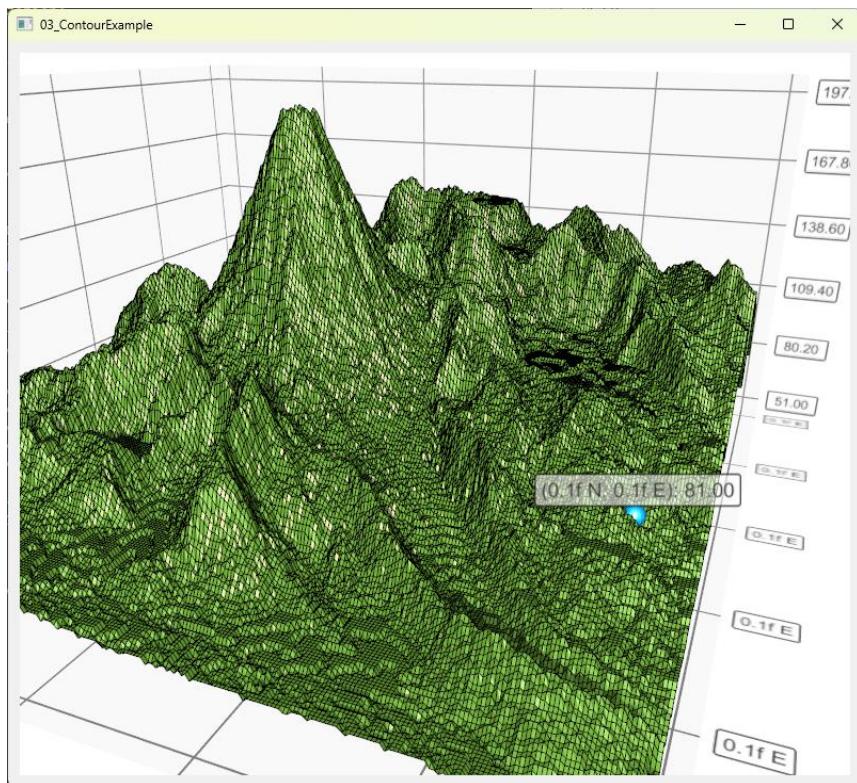
좌측의 그림은 등고선 추출을 위해 촬영된 이미지이다. 흰색이 높을수록 높이가 높다. 즉 높이가 높다는 것은 고도가 높아짐을 의미한다. 이와 같은 사진을 Q3DSurface를 이용해 3D 그래프로 시각화 해보도록 하자. 다음 그림은 예제 실행 화면이다.

QImage 클래스 오브젝트에 리소스에 등록한 Height map 이미지를 로딩 한다. 그리고 QImage 오브젝트를 QHeightMapSurfaceDataProxy 클래스 오브젝트상에 넘겨준다.

이 클래스는 이미지를 Surface로 시각화 하기 위해서, Surface의 높이 데이터를 처리한다.

QSurface3DSeries 클래스는 사용자가 그래프상에서 특정 데이터를 클릭하면 해당 데이터를 표시하는 방법을 지정할 수 있다. 그리고 setValueRanges() 함수를 통해 범위를 지정할 수 있다.

예수님은 당신을 사랑합니다.



위의 예제의 소스코드는 03_ContourExample 디렉토리를 참조하면 된다.

37. Inter Process Communication

IPC (Inter Process Communication)는 같은 시스템 내에 존재하는 프로세스간의 통신을 하기 위한 방법이다. 우리는 이번 장에서 아래와 같은 IPC 방법에 대해서 살펴볼 것이다.

① Unix Domain Socket and Named pipe

리눅스에서는 UDS(Unix Domain Socket), MS Windows 에서는 Named pipe 이름을 사용하는 프로세스간 통신.

② QProcess 클래스를 이용한 외부 프로세스와의 통신.

외부 프로세스를 실행 또는 결과를 얻어 오기 위해 사용

③ Shared Memory를 이용해 외부 프로그램에서 메모리를 공유.

QSharedMemory 클래스를 이용해 외부 프로그램 간 공유 메모리를 사용.

37.1. Unix Domain Socket and Named pipe

UDS (Unix Domain Socket) 란 리눅스 플랫폼 사용하는 프로세스간 통신 방법 중 하나이다. Name pipe 는 UDS를 좀더 확장 시켜 MS Windows 플랫폼에서 사용하는 프로세스간 통신 방법이다. 사용 목적은 동일하나 명칭이 다르다.

하지만 Qt 에서는 리눅스를 사용하든지 MS Windows에서 사용하든지 QLocalSocket 과 QLocalServer 클래스를 플랫폼 상관없이 모두 사용할 수 있다. 또한 MacOS에서도 사용할 수 있다.

UDS 또는 Name pipe를 사용하기 위해서는 프로젝트 파일에 Network 모듈을 사용해야 한다. 만약 CMake 프로젝트를 사용한다면 아래와 같이 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS Network)
target_link_libraries(mytarget PRIVATE Qt6::Network)
```

만약 qmake를 사용한다면 프로젝트 파일에 아래와 같이 추가해야 한다.

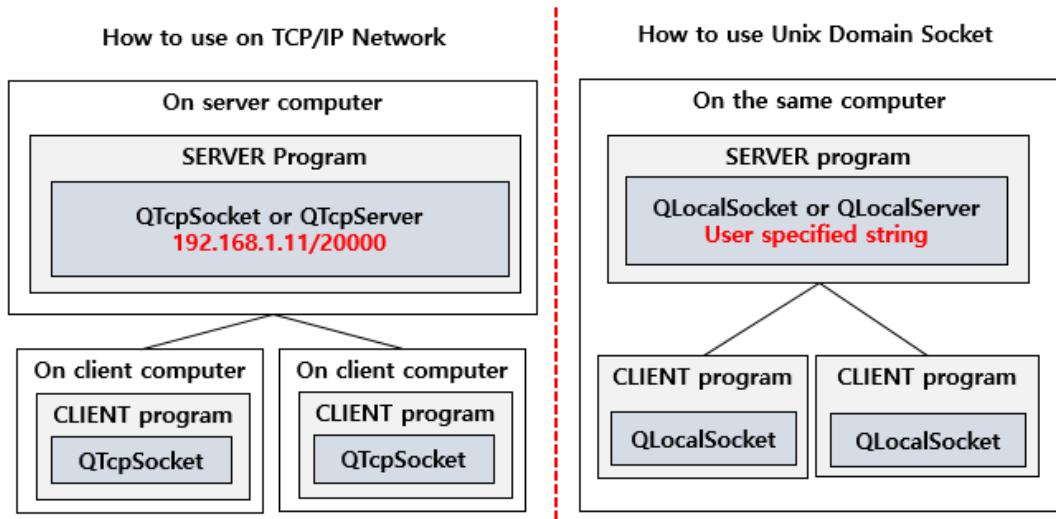
```
QT += network
```

QLocalSocket 과 QLocalServer 클래스는 상대방을 식별 할 수 있는 방법으로 사용자가 지정한 특정 문자열을 사용한다.

QLocalSocket 은 프로세스간 통신을 하기 위한 기능이 구현되었다. UDS에서는 Server / Client 개념으로 송/수신이 가능하다.

이는 Server 와 Client 간 1:N 통신이 가능하다. QLocalServer 는 Server 구현에 필요한 기능을 쉽게 구현할 수 있는 기능을 제공한다.

네트워크에서 특정 서버와 통신을 위해서는 식별자로 IP와 PORT 를 사용한다. 하지만 사용자가 정의한 문자열(단어)을 식별자로 사용할 수 있다.



QLocalSocket 과 QLocalServer 클래스는 Signal 과 Slot 을 이용해 비 동기 방식의 데이터 수신 모듈을 쉽게 구현할 수 있다. 따라서 Thread를 사용하지 않고 수신 모듈을 쉽게 구현할 수 있다.

TCP/IP 네트워크에서 서버가 클라이언트 접속을 대기하기 위해 QTcpServer 클래스가 제공하는 listen() 멤버 함수를 사용한 것과 같이 QLocalServer 클래스도 클라이언트 접속을 기다리기 위해 listen() 멤버 함수를 사용할 수 있다.

```
...
server = new QLocalServer(this);

if (!server->listen("MyServer")) {
    qDebug() << "Server error : " << server->errorString();
    ...
}

connect(server, SIGNAL(newConnection()), this, SLOT(clientConnection()));
...
```

위의 예제에서 connect() 멤버 함수는 새로운 클라이언트가 접속하면 newConnection() 시그널이 발생한다. 이 시그널이 발생했을 때 clientConnection() Slot 함수와 호출 될 수 있도록 연결 하였다.

아래 예제 소스코드는 QLocalSocket 클래스를 이용해 클라이언트가 서버에 접속하고 서버로부터 메시지를 받거나 예기 않은 에러가 발생하면 처리하기 위한 시그널이다.

그리고 아래 소스코드의 마지막 부분의 connectToServer() 함수는 Server에 연결하는 기능을 제공한다. 이 함수의 첫 번째 인자는 서버의 고유한 이름이다.

예수님은 당신을 사랑합니다.

여기서는 "MyServer" 라고 이름을 명시하였지만 사용자가 원하는 문자열로 변경할 수 있다.

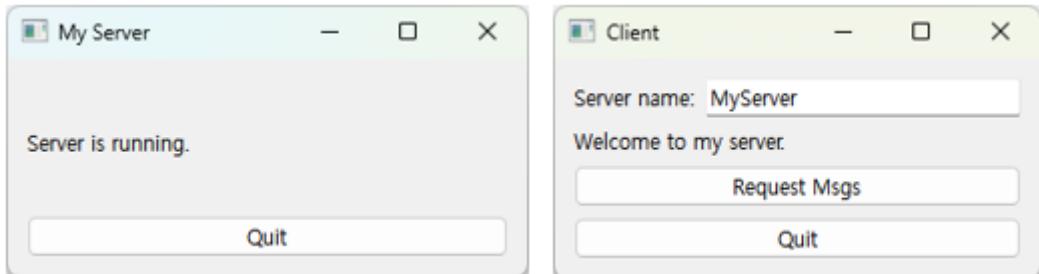
```
...
socket = new QLocalSocket(this);

connect(socket, SIGNAL(readyRead()), this, SLOT(readData()));

connect(socket, SIGNAL(error(QLocalSocket::LocalSocketError)),
        this, SLOT(sockError(QLocalSocket::LocalSocketError)));

socket->connectToServer("MyServer");
...
```

다음은 실제 UDS 상에서 실제 Server 와 Client를 통해 좀더 자세히 알아보도록 하자.



좌측의 예제는 Server이고 우측은 Client 예제이다.

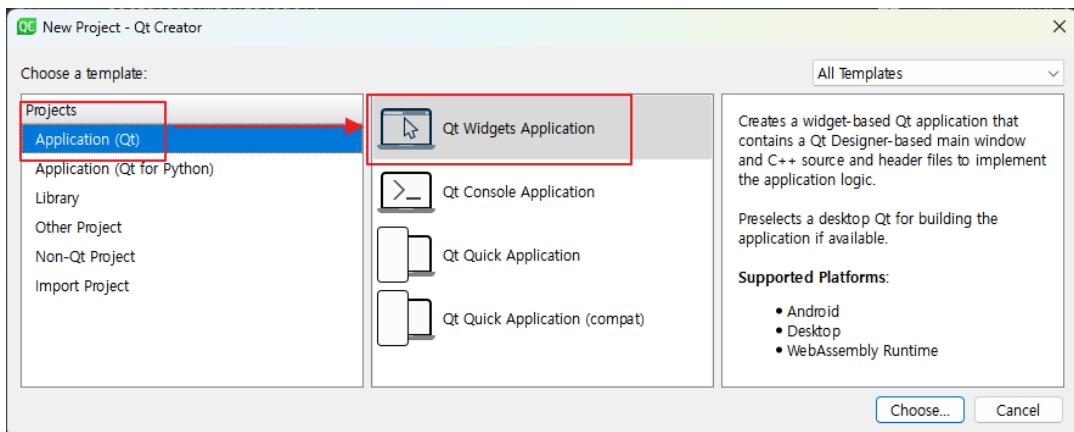
Server부터 실행하고 Client를 실행 한다. Client 가 실행 된 후, [Request Msgs] 버튼을 클릭하면 Server 와 연결을 요청한다.

그러면 Server 는 Client 와 연결하고 Client 에게 메시지를 송신한다. 그리고 바로 Client와 연결을 종료한다.

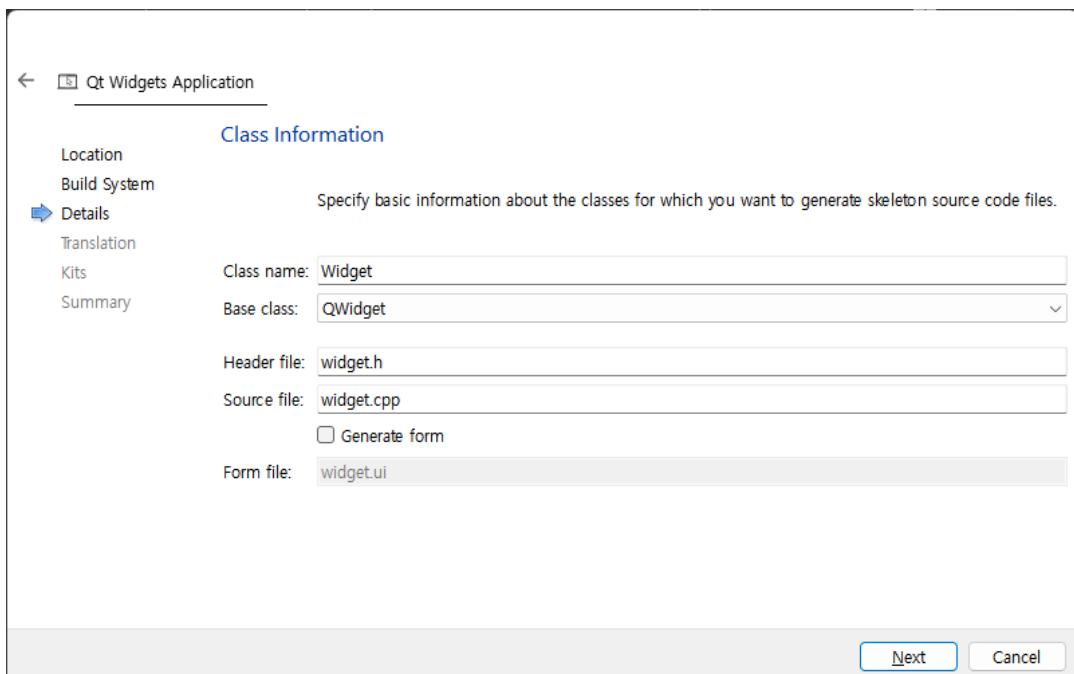
이번 장에서는 위의 그림에서 보는 것과 같이 2개의 예제를 구현해보도록 하자. 2개의 예제 중 Server 예제를 먼저 구현해본 다음 Client 예제를 구현해 보도록 하자.

✓ Server 예제 구현

프로젝트 생성 시 Qt Widget 기반 어플리케이션을 선택한다.



이번 프로젝트에서는 UI Form 을 사용하지 않는다. 따라서 프로젝트 생성 중 Class Information 다이얼로그에서 [Generate form] 항목이 체크되어 있다면 체크를 해제해야 한다.



프로젝트 생성이 완료 되면 CmakeLists.txt 파일을 열어서 아래와 같이 Network 모듈을 추가한다.

```
cmake_minimum_required(VERSION 3.5)
...
find_package(Qt NAMES Qt6 Qt5 REQUIRED COMPONENTS Widgets)
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Widgets)
```

예수님은 당신을 사랑합니다.

```
find_package(Qt${QT_VERSION_MAJOR} REQUIRED COMPONENTS Network)

set(PROJECT_SOURCES
    main.cpp
    widget.cpp
    widget.h
)
...
target_link_libraries(@@_LocalServer PRIVATE Qt${QT_VERSION_MAJOR}::Widgets)
target_link_libraries(@@_LocalServer PRIVATE Qt${QT_VERSION_MAJOR}::Network)
...
if(QT_VERSION_MAJOR EQUAL 6)
    qt_finalize_executable(@@_LocalServer)
endif()
```

다음은 widget.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QLabel>
#include <QPushButton>
#include <QLocalServer>

class Widget : public QWidget
{
    Q_OBJECT

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

private slots:
    void clientConnection();

private:
```

```
QLabel *statusLabel;
QPushButton *quitButton;
QLocalServer *server;
QStringList sendMsgs;
bool msgKind;

};

#endif // WIDGET_H
```

clientConnection() Slot 함수는 newConnection() Signal 이 발생되면 호출 된다. 따라서 새로운 Client 접속하면 clientConnection() Slot 함수가 호출 된다.

다음으로 widget.cpp 소스코드 파일을 열어서 아래와 같이 작성한다.

```
#include "widget.h"
#include <QLocalSocket>
#include <QVBoxLayout>

Widget::Widget(QWidget *parent)
: QWidget(parent)
{
    window()->setMinimumSize(300, 100);

    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 Widget::clientConnection()  
{  
    QByteArray writeData;  
    if(msgKind) {  
        writeData.append("Welcome to my server.");  
        msgKind = false;  
    } else {  
        writeData.append("Who are you");  
        msgKind = true;  
    }  
  
    QLocalSocket *clientConnection =  
        server->nextPendingConnection();  
  
    connect(clientConnection, SIGNAL(disconnected()),  
            clientConnection, SLOT(deleteLater()));  
  
    clientConnection->write(writeData, writeData.size());  
    clientConnection->flush();  
    clientConnection->disconnectFromServer();  
}  
  
Widget::~Widget()  
{  
}
```

이 클래스 생성자에서 QLocalServer 클래스 오브젝트를 선언한다. 그리고 Server 의 고

예수님은 당신을 사랑합니다.

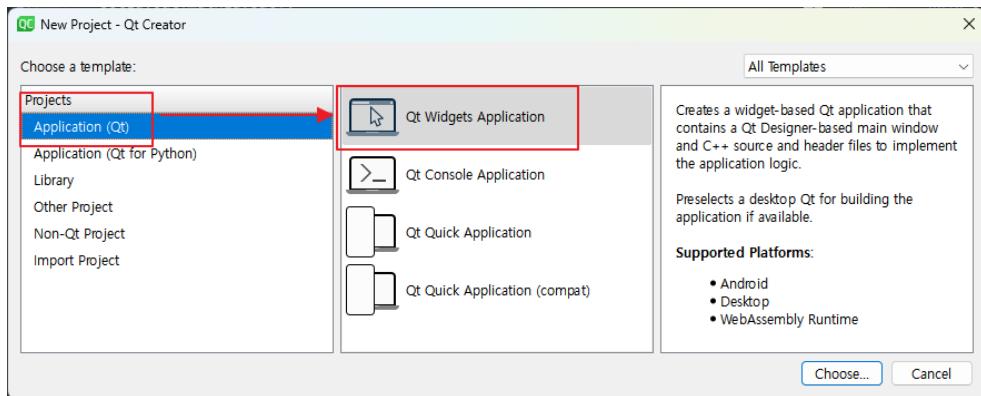
유한 이름을 "MyServer"로 지정하였다.

위의 clientConnection() Slot 함수에서는 클라이언트가 접속하면 클라이언트에게 메시지를 전송한 후 연결을 Disconnect 한다.

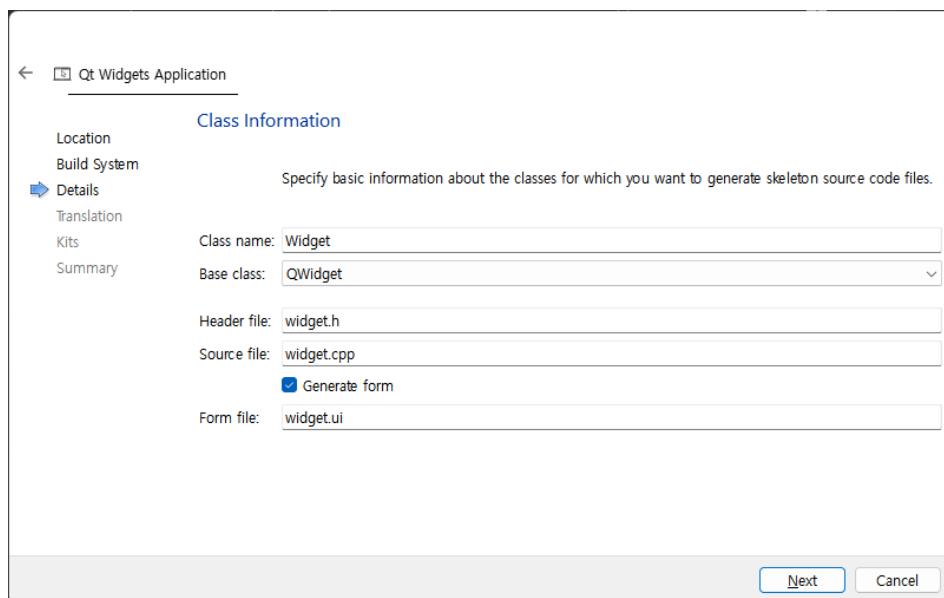
여기까지 Server 예제에 대해서 알아보았다. 이 예제의 전체 소스코드 00_LocalServer 디렉토리를 참조하면 된다. 다음으로 Client 예제에 대해서 구현해 보도록 하자.

✓ Client 예제 구현

프로젝트 생성 시 Qt Widget 기반 어플리케이션을 선택한다.



Client 프로젝트에서는 UI Form 을 사용하지 않는다. 따라서 프로젝트 생성 중 Class Information diálog에서 [Generate form] 항목이 체크되어 있다면 체크를 해제한다.



프로젝트 생성이 완료 되면 CmakeLists.txt 파일을 열어서 아래와 같이 Network 모듈을

예수님은 당신을 사랑합니다.

추가한다.

다음은 widget.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QLocalSocket>

class Widget : public QWidget
{
    Q_OBJECT

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

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 // WIDGET_H
```

다음으로 widget.cpp 파일을 열어서 아래와 같이 작성한다.

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

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    hostLabel = new QLabel(tr("Server name:"));
    hostLineEdit = new QLineEdit("MyServer");
    statusLabel = new QLabel("");
    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 Widget::requestNewMsg()
{
    reqButton->setEnabled(false);
    socket->connectToServer(hostLineEdit->text());
}

void Widget::readData()
```

```
{  
    statusLabel->setText(socket->readAll());  
    reqButton->setEnabled(true);  
}  
  
void Widget::sockError(QLocalSocket::LocalSocketError socketError)  
{  
    switch (socketError) {  
        case QLocalSocket::ServerNotFoundError:  
            qDebug() << "ServerNotFoundError";  
            break;  
        case QLocalSocket::ConnectionRefusedError:  
            qDebug() << "ConnectionRefusedError";  
            break;  
        case QLocalSocket::PeerClosedError:  
            qDebug() << "PeerClosedError";  
            break;  
        default:  
            qDebug() << "Error : " << socket->errorString();  
    }  
  
    reqButton->setEnabled(true);  
}  
  
Widget::~Widget()  
{  
}
```

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. 이 예제 소스코드는 01_LocalSocket 디렉토리를 참조하면 된다.

37.2. QProcess

QProcess 클래스는 구현한 응용 프로그램 내에서 외부 프로그램을 실행하고 결과를 가져올 수 있는 기능을 제공한다.

예를 들어 리눅스에서 "/bin/ls" 명령어는 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 아래와 같이 QProcess 클래스를 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 할 수 있다.

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 있다. 자세한 사용 방법은 다음 예제를 통해서 알아보도록 하자.

✓ QProcess 클래스를 이용한 예제

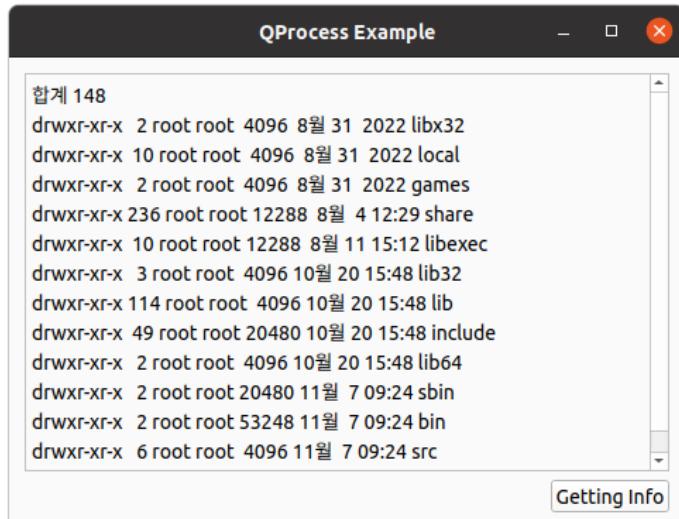
이번 예제는 QProcess 클래스를 이용해 외부 프로그램을 실행 후 결과를 얻어와 GUI 위젯에 결과를 출력해 보도록 하자. 리눅스에서 "/bin/ls -ltr /usr" 명령을 실행하면 다음과 같은 결과를 출력한다.

```
qtdev@linux1:~$ ls -ltr /usr
합계 120
```

예수님은 당신을 사랑합니다.

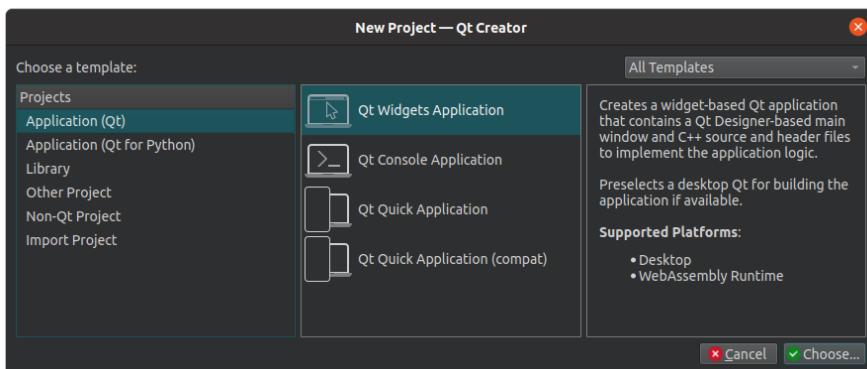
```
drwxr-xr-x 10 root root 4096 2월 10 09:12 local
drwxr-xr-x  2 root root 4096 2월 10 09:14 games
drwxr-xr-x  3 root root 4096 2월 22 13:14 lib32
drwxr-xr-x 137 root root 4096 2월 22 13:23 lib
...
```

위에서 보는 것과 같이 “/bin/ls” 명령어와 함께 사용한 “-ltr” 옵션은 파일 및 디렉토리 정보를 자세히 출력하는 옵션이다. /usr 옵션은 출력할 디렉토리 명이다. 다음은 예제를 실행한 화면이다.



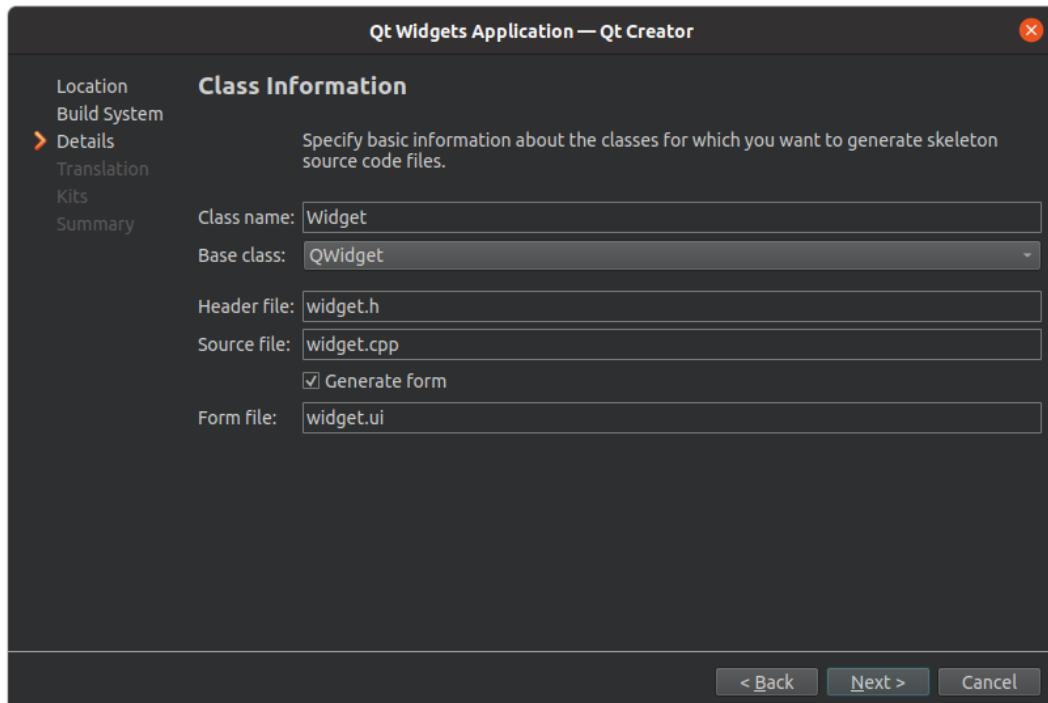
위의 그림에 보는 것과 같이 하단의 [Getting Info] 버튼을 클릭하면 외부 명령을 실행하고 결과를 얻어온다. 그리고 QTextEdit 창에 결과를 출력한다.

프로젝트 생성 시 QWidget 을 상속받는 클래스를 생성한다.

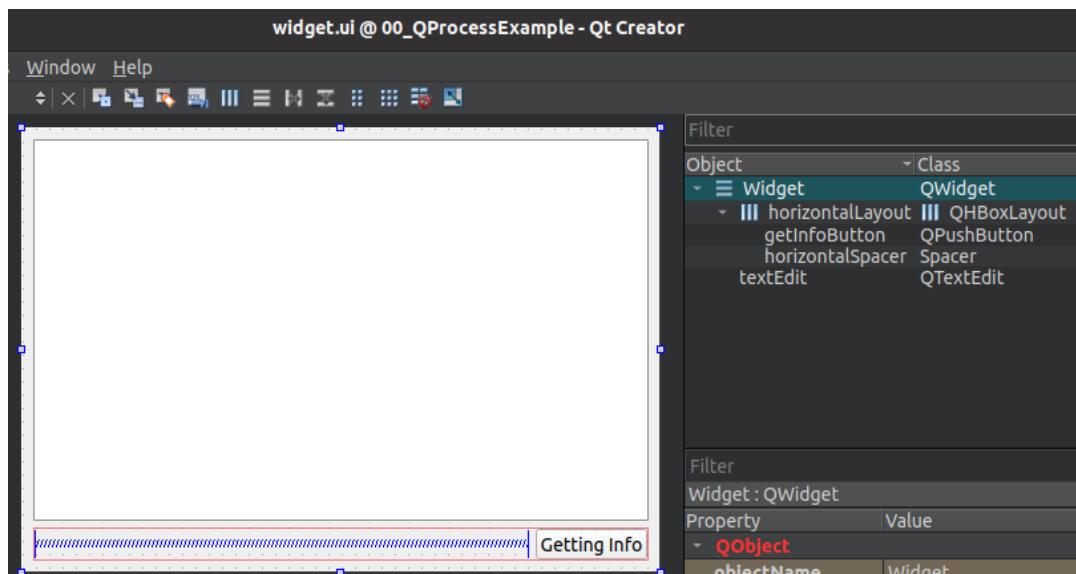


이번 프로젝트에서는 UI Form을 사용할 것이다. 따라서 [Generate form] 항목을 체크한다.

예수님은 당신을 사랑합니다.



프로젝트 생성이 을 완료 하였으면 아래 그림에서 보는 것과 같이 widget.ui 파일을 열어서 위젯들을 배치한다.



다음으로 widget.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef WIDGET_H  
#define WIDGET_H
```

```
#include <QWidget>
#include <QProcess>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

public:
    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"

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()));  
}  
  
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";
}

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

위의 예제에서 보는 것과 같이 클래스 생성자에서 QProcess 클래스의 오브젝트를 선언하고 QProcess 에서 제공하는 Signal 과 Slot 을 연결한다.

GUI 상에서 [Getting Info] 버튼을 클릭하면 getInfoButton() Slot 함수가 호출되고 QProcess 클래스의 start() 멤버 함수를 이용해 외부 프로그램을 실행한다.

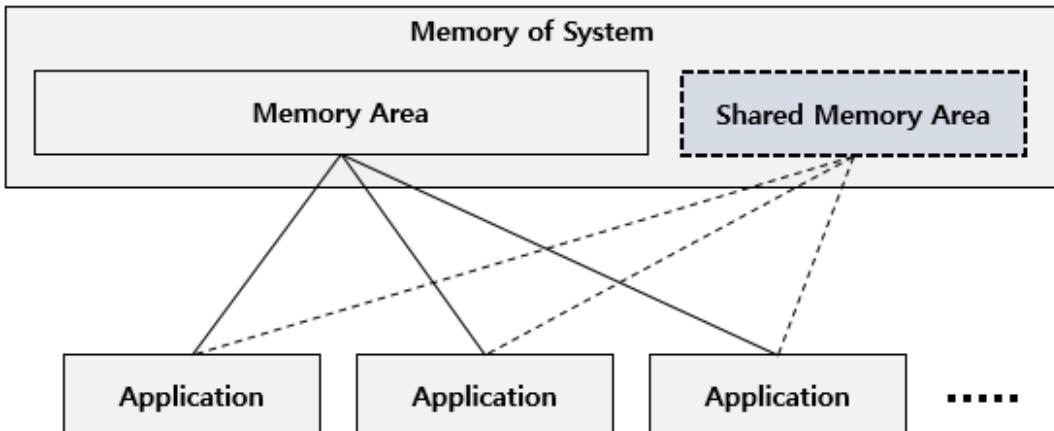
readyReadStandardOutput() 함수에서 QProcess 클래스의 readAllStandardOutput() 멤버 함수는 결과를 가져오고 가져온 결과를 buf 에 저장한다. 그리고 buf의 내용을 QTextEdit 위젯에 출력한다.

이 예제의 소스코드는 00_QProcessExample 디렉토리를 참조하면 된다.

37.3. Shared Memory

Shared Memory 를 이용한 공유 메모리를 사용하는 방식은 같은 시스템 내에서 프로그램간의 데이터를 교환 하기 위한 목적으로 공유 메모리 영역을 사용할 수 있다.

Qt는 어플리케이션간 Shared Memory 영역을 사용하기 위해서 QSharedMemory 클래스를 제공한다.



위의 그림과 같이 각각의 어플리케이션에서 공유 메모리 영역을 READ/WRITE 하기 위해서는 특정 메모리 영역을 접근할 수 있는 매개체를 이용한다.

Qt에서는 고유한 KEY값을 이용해 공유 메모리 영역에 데이터를 READ 하거나 WRITE 할 수 있다. 다음 예제 소스코드는 KEY값을 설정하는 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.

설정된 KEY 값을 참조하기 위해서는 QSharedMemory 클래스의 key() 멤버 함수를 사용하면 된다. 다음은 공유 메모리 영역에 데이터를 Writing 하는 방법이다.

```
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 공유 메모리 영역 주소 번지를 얻어와 memcpy()를 이용해 데이터를 WRITE 할 수 있다.

다음으로 공유 메모리 영역에 데이터를 읽어 오기 위해서 아래와 같이 사용할 수 있다.

```
...
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));
...
```

QSharedMemory 클래스는 위에서 보는 것과 같이 매우 사용하기 쉽다.

다음은 공유 메모리 영역에 데이터 WRITE/READ 하는 실제 예제를 통해서 구현해 보도록 하자.

✓ 공유 메모리에 Writing 하는 예제 구현

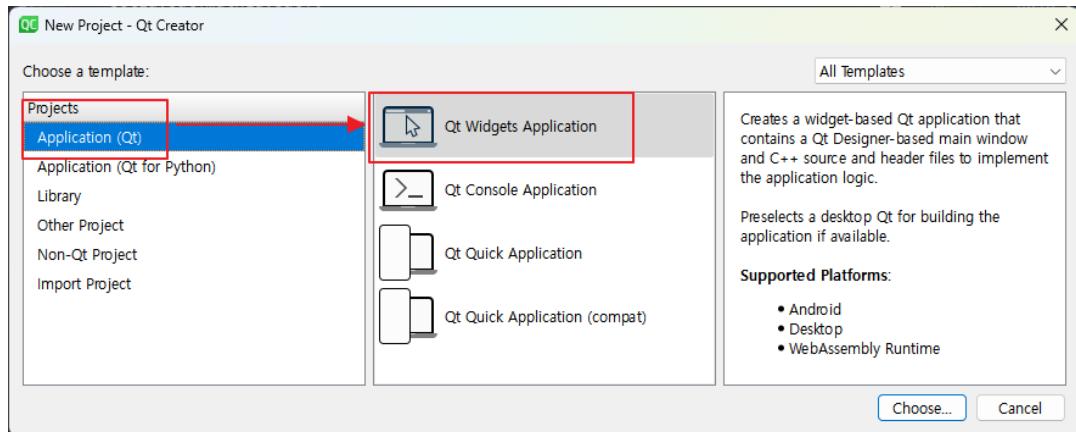
두 개의 독립된 어플리케이션을 구현할 것이다. 첫 번째로 어플리케이션은 KEY 값을 "qt-dev.com"으로 설정하고 이미지파일을 읽어와 GUI상에 표시한다. 그리고 이 이미지의 바이너리 데이터를 공유 메모리 영역에 Writing 할 것이다.

예수님은 당신을 사랑합니다.

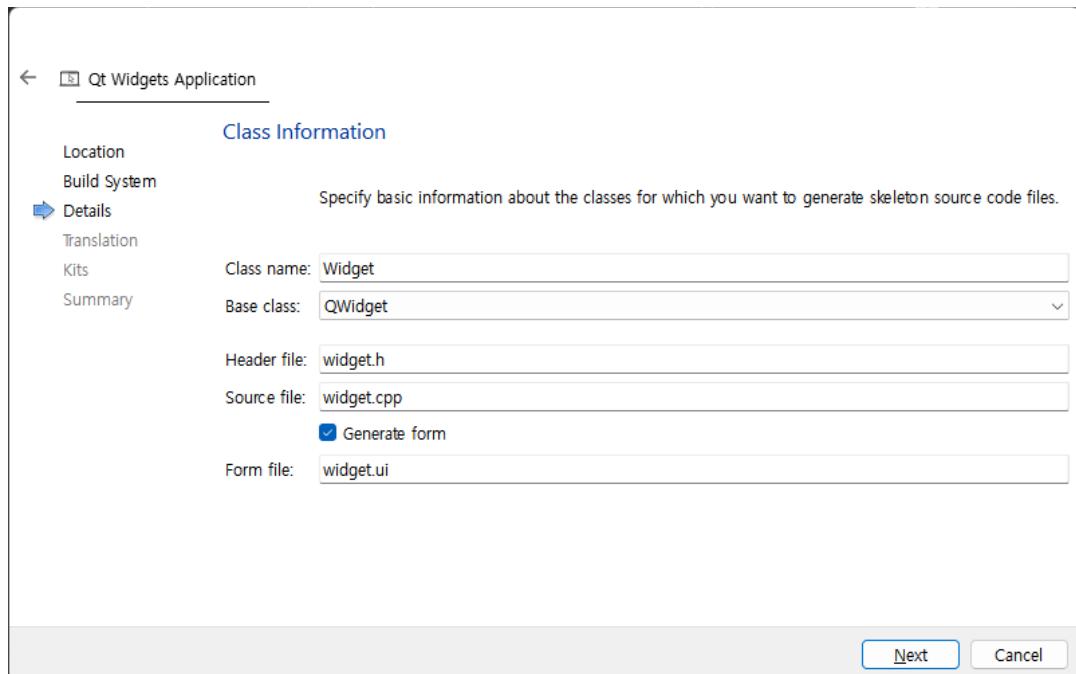
두 번째 예제에서는 공유 메모리 영역에서 데이터를 읽어와 QPixmap 이미지로 변환 후 GUI상에 표시할 것이다.

아래 그림에서 보는 것과 같이 첫 번째 예제에서 [Writing to the shared memory] 버튼을 클릭한다. 그러면 GUI상에 이미지를 표시하고 이미지 파일의 Binary 데이터를 읽어와 Shared Memory 영역에서 Writing 한다.

첫 번째 예제를 작성하기 위해서 프로젝트 생성 시 Qt Widget 기반으로 프로젝트를 생성한다.

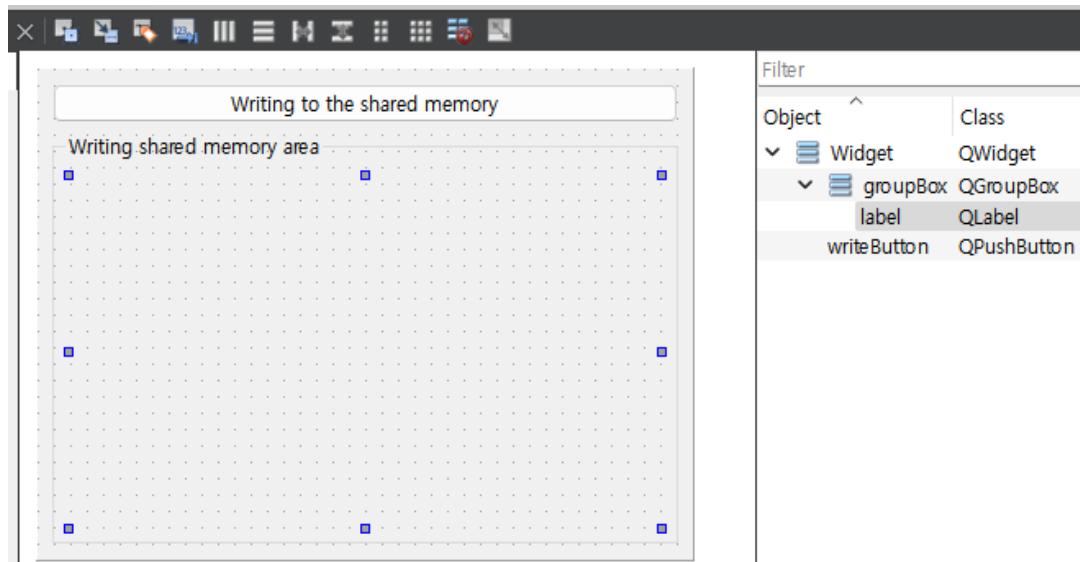


그리고 첫 번째 예제에서는 UI Form 을 사용할 것이다. 따라서 [Generate form] 항목을 체크한다.



예수님은 당신을 사랑합니다.

프로젝트 생성이 완료되면 widget.ui를 열어서 아래와 같이 위젯들을 배치한다.



다음으로 widget.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QSharedMemory>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
```

```
QSharedMemory *m_sharedMemory;

private slots:
    void writeButton();
};

#endif // WIDGET_H
```

다음으로 widget.cpp 소스코드 파일을 열어서 아래와 같이 작성한다.

```
#include "widget.h"
#include "./ui_widget.h"
#include <QFileDialog>
#include <QBuffer>

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(tr("Shared memory detach failed."));
    }

    QString fileName;
    fileName = QFileDialog::getOpenFileName(
        0, QString(), QString(),
```

```
tr("Images (*.png *.xpm *.jpg)");  
QImage image;  
if (!image.load(fileName)) {  
    ui->label->setText(tr("Could you select a image file?"));  
    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(tr("Shared memory Segment create failed."));  
    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;  
}
```

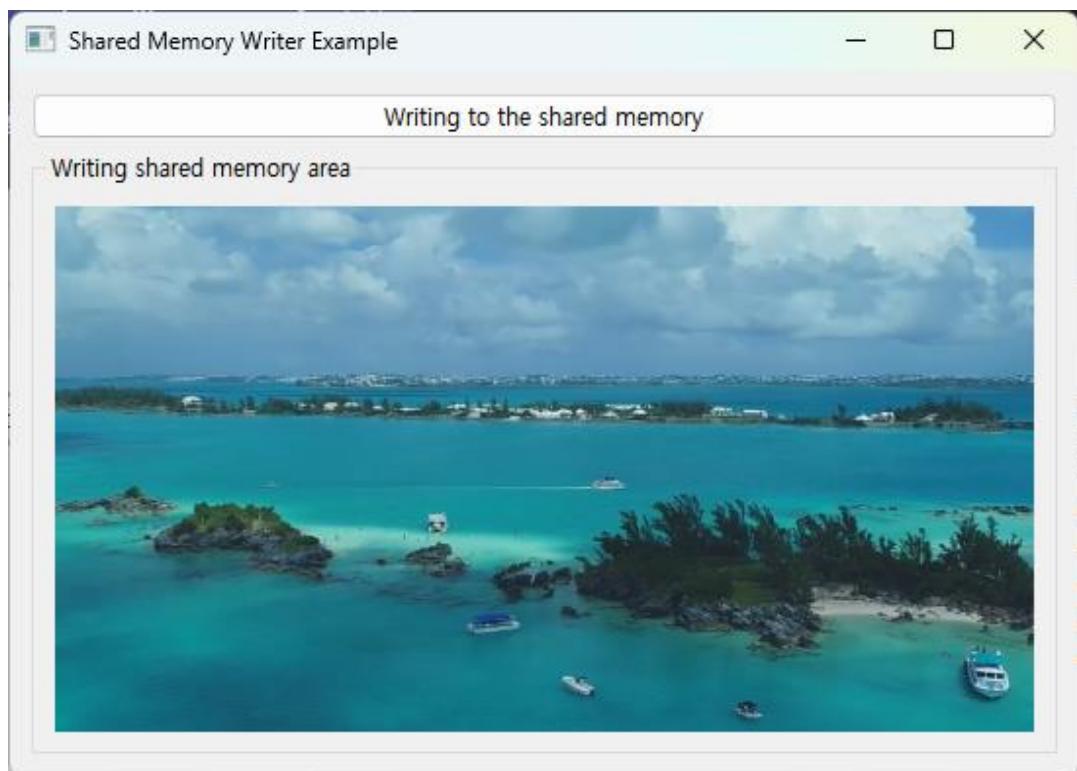
이 클래스의 생성자 함수에서는 QSharedMemory 클래스 오브젝트를 선언하고 공유메모리에서 사용할 키 값을 명시하였다. 키는 고유한 값을 사용하며 타입은 QString 을 사용한다.

writeButton() Slot 함수는 [Shared Memory 쓰기] 버튼 클릭 시 호출된다. 이 Slot 함수에서는 파일을 선택할 수 있는 다이얼로그가 로딩된다.

파일 중 이미지 파일을 선택하면 선택한 이미지를 GUI에 출력하고 이미지파일의

예수님은 당신을 사랑합니다.

Binary 데이터를 읽어와 공유 메모리 영역에 Writing 한다.



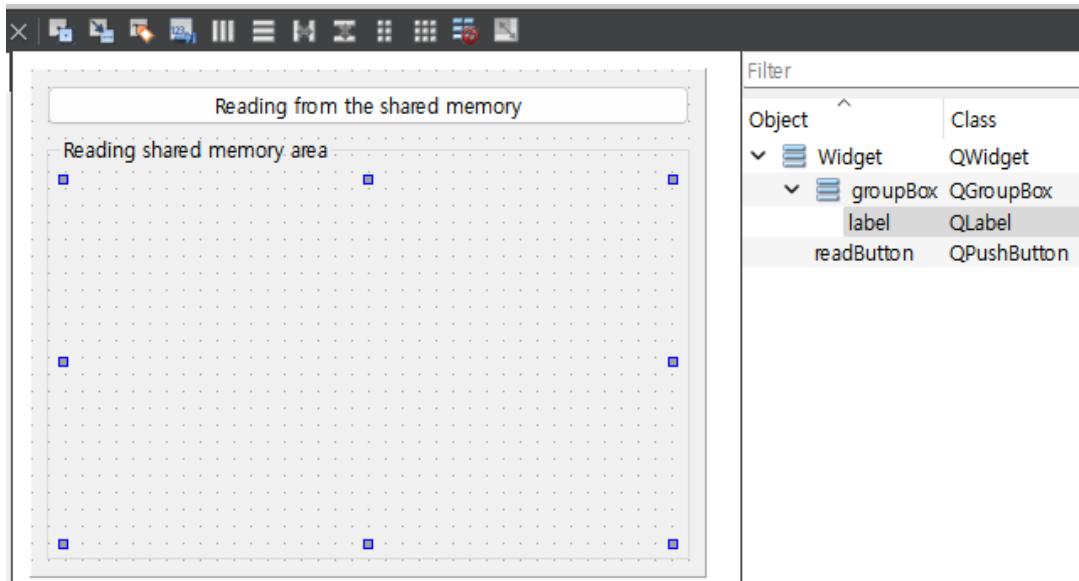
위의 사용한 이미지는 아무 이미지나 사용해도 된다.

이 예제의 소스코드는 00_SharedMemory_Writer 디렉토리를 참조하면 된다.

- ✓ 공유 메모리로부터 데이터를 Read 하는 예제 구현

두 번째 예제에서는 첫 번째 예제에서 Writing 한 공유 메모리 영역에서 데이터를 읽어와 QPixmap 이미지로 변환 후 GUI상에 이미지를 출력하는 예제이다. 프로젝트 생성 시 Qt Widget 기반으로 프로젝트를 생성한다.

그리고 이번 예제에서는 UI Form 을 사용할 것이다. 따라서 [Generate form] 항목을 체크한다. 프로젝트 생성이 완료되면 widget.ui를 열어서 아래와 같이 위젯들을 배치한다.



다음으로 widget.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QSharedMemory>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
    QSharedMemory *m_sharedMemory;

private slots:
    void readButton();
};

#endif // WIDGET_H
```

```
#endif // WIDGET_H
```

다음으로 widget.cpp 소스코드 파일을 열어서 아래와 같이 작성한다.

```
#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("Reading Failed from shared memory");
        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;
```

예수님은 당신을 사랑합니다.

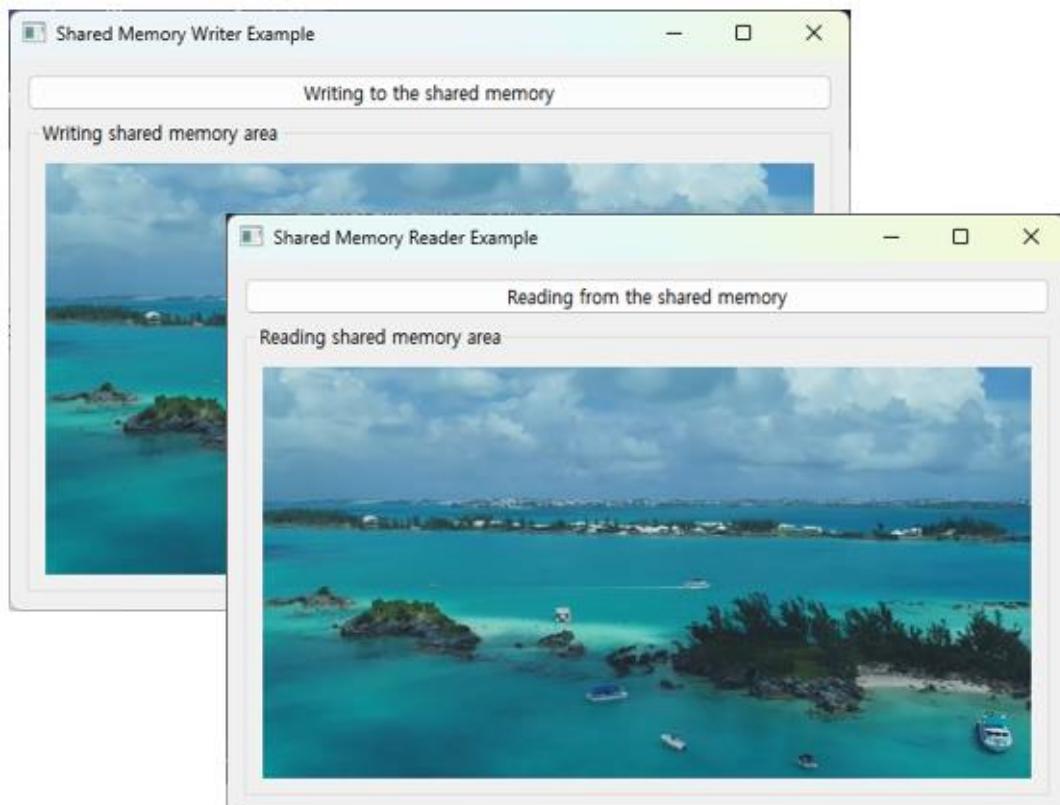
```
m_sharedMemory->unlock();

m_sharedMemory->detach();
ui->label->setPixmap(QPixmap::fromImage(image));
}

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

readButton() Slot 함수는 GUI 상에서 [Reading from the shared memory] 버튼을 클릭하면 호출된다.

이 Slot 함수에서는 공유 메모리 영역에서 데이터를 읽어와 QImage에 저장한다. 그리고 GUI 상에서 QLabel 위젯에 이미지를 표시하기 위해서는 QImage을 다시 QPixmap 이미지로 변환한다.



이 예제 소스코드는 01_SharedMemory_Reader 디렉토리를 참조하면 된다. 주의 사항은
페이지 525

37.3. Shared Memory

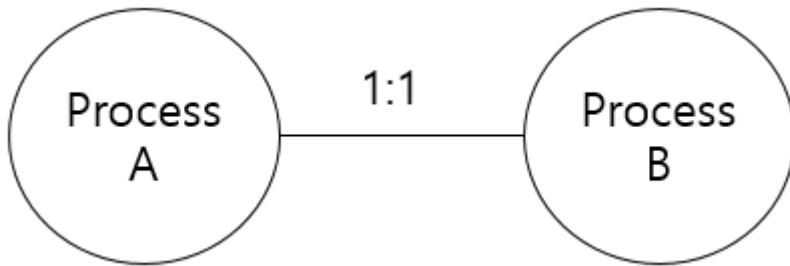
예수님은 당신을 사랑합니다.

로 Writing 예제를 먼저 수행한다. 그리고 Writing 예제로 로딩된 상태에서 Reading 하는 예제를 수행하면 된다.

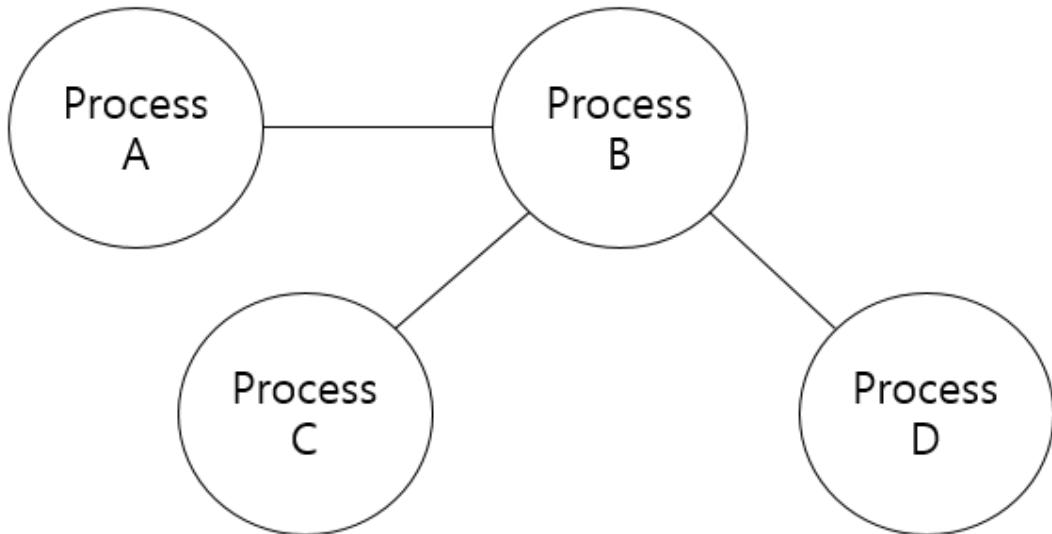
37.4. Qt D-Bus

D-Bus는 원래 Linux용으로 개발된 프로세스 간 통신(IPC) 및 원격 프로시저 호출(RPC) 메커니즘으로, 리눅스의 기존의 IPC를 D-Bus로 통합된 프로토콜로 대체하기 위해 개발되었다.

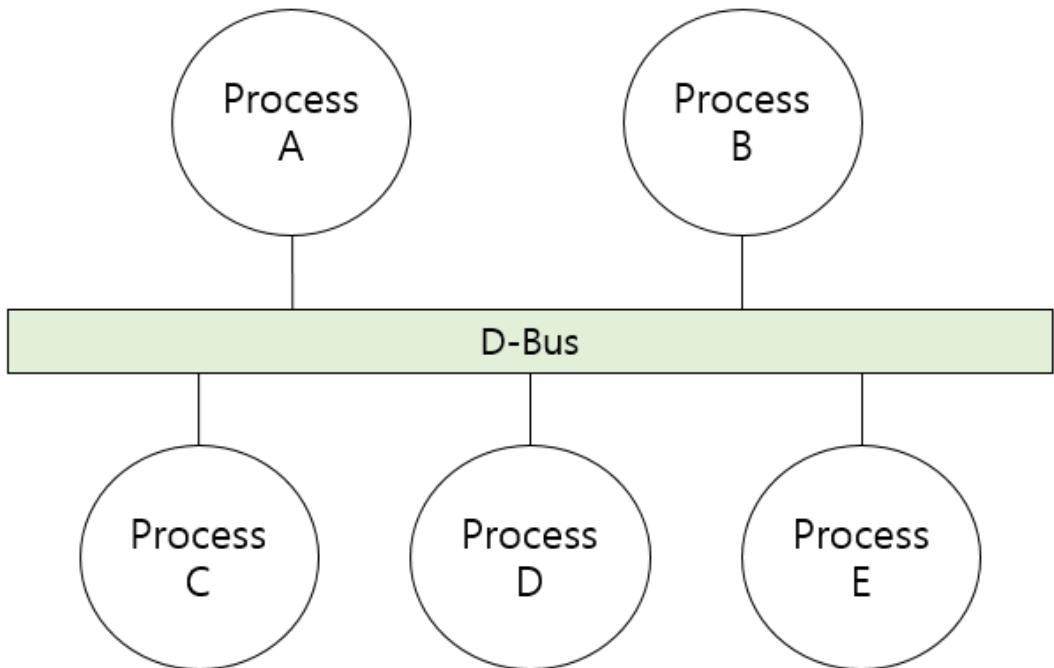
기존의 IPC 들은 1:1 또는 1:N 방식으로 통신을 하게 된다.



위의 그림에서 보는 것과 같이 IPC 는 1:1 방식으로 연결된다. 또한 아래 그림에서 보는 것과 같이 IPC를 이용한 Process 간 통신은 1:N 방식으로 이루어질 수 있다.



따라서 기존의 IPC 방식은 항상 통신 하는 상대방이 존재한다. 하지만 D-Bus 는 IPC는 다른 구조를 가지고 있다. D-Bus 에서는 Process 간 통신이 이루어 지지 않고 아래 그림에서 보는 것과 같이 D-Bus 서비스를 통해서 통신이 이루어 진다.



D-Bus 서비스는 Process 간 통신하기 위해서 D-Bus 가 통신을 관리한다. 예를 들어 Process A 가 D-Bus 에게 메시지를 전달하면 해당 D-Bus 와 연결된 모든 Process 에게 메시지를 전달할 수 있다.

이 그림에서는 Process가 5개가 있지만 1개만 있을 수 있고 그렇지 않을 수 도 있다. 또한 Process 가 6개 또는 그 이상이 될 수 있다.

즉 Process A 가 송/수신 하기 위해서 특정 Process 와 연결된 것이 아니라 D-Bus 서비스와 연결되어 있기 때문에 D-Bus 서비스에 연결된 모든 Process 로부터 메시지를 송/수신 할 수 있다.

여기서 D-Bus 1개의 서비스만 존재하지 않는다. D-Bus 의 각 고유한 이름은 D-Bus 의 Service Name 과 Object Path 로 구분 된다.

Service Name 은 예를 들어 “org.freedesktop.DBus” 과 같은 형태로 사용자가 지정할 수 있다. 그리고 Object Path 는 “pub/something/”으로 구분 된다.

✓ D-Bus 의 종류

D-Bus 는 종류로 System Bus 와 Session Bus 로 구분된다. System Bus 는 리눅스 커널과 같은 Process 상에서 통신을 위해서 사용하고 Session Bus 는 주로 어플리케이션과 통신하기 위한 목적으로 제공된다.

예수님은 당신을 사랑합니다.

✓ Method 와 Signal

D-Bus 는 Method 와 Signal 로 구분된다. Method 는 Qt에서 Slot으로 이해하면 된다.
그리고 Signal 은 Qt에서 Signal 과 동일한 것으로 이해하면 된다.

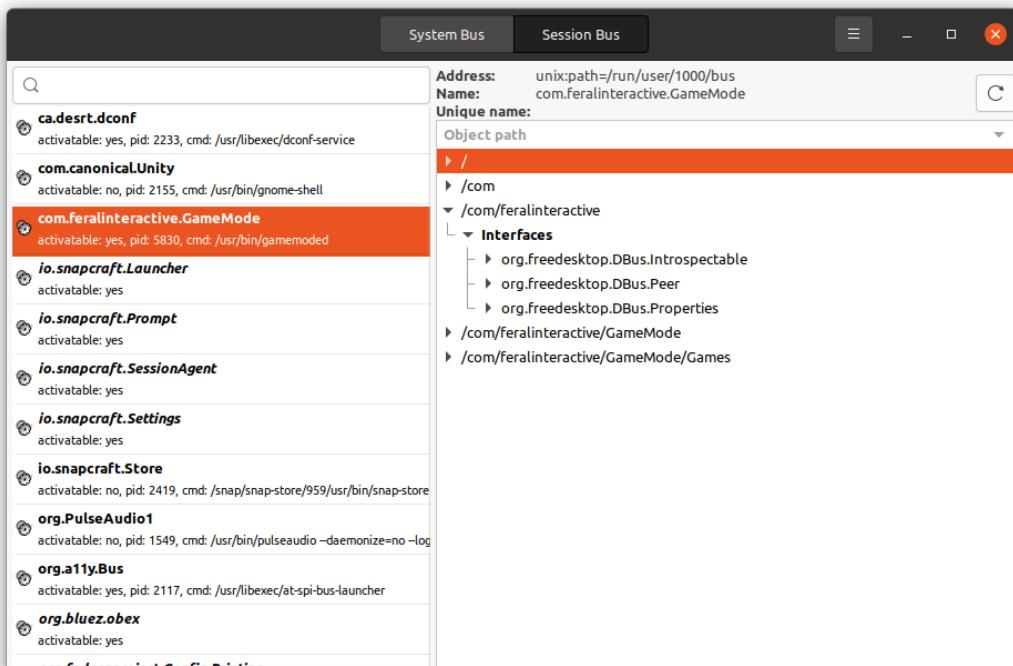
Qt 에서 D-Bus를 사용하기 위해서 CMake의 프로젝트 파일인 CMakeList.txt 파일에 아래와 같이 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS DBus)
target_link_libraries(mytarget PRIVATE Qt6::DBus)
```

✓ 유용한 D-Bus 디버깅 툴

리눅스에선 D-Bus를 디버깅하기 위한 툴로 d-feet를 제공한다. d-feet 를 설치하기 위해서는 아래와 같이 apt를 사용해 설치하면 된다.

```
$ sudo apt install d-feet
```



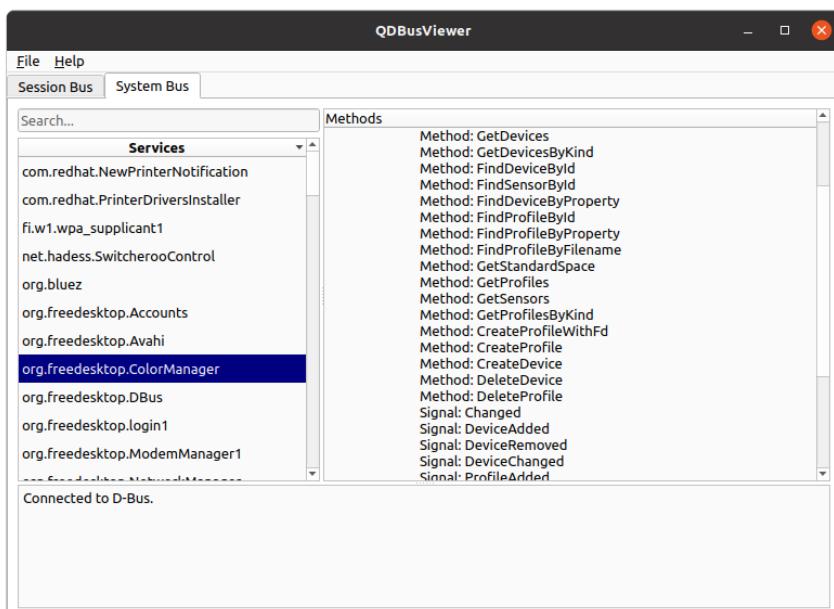
그리고 D-Bus 상에서 송/수신 되는 메시지를 실시간 디버깅하는 방법으로 "dbus-monitor" 라는 command 툴을 제공한다.

```
dev@ubuntu:~/QtProjects/examples$ dbus-monitor
signal time=1699861043.056620 sender=org.freedesktop.DBus -> destination=:1.101 serial=2 path=/org/freedesktop/DBus; interface=org.freedesktop.DBus; member=NameAcquired
    string ":1.101"
signal time=1699861043.056645 sender=org.freedesktop.DBus -> destination=:1.101 serial=4 path=/org/freedesktop/DBus; interface=org.freedesktop.DBus; member=NameLost
    string ":1.101"
```

Qt에서는 D-Bus를 디버깅하기 위한 툴로 “qdbusviewer”툴을 제공한다.

```
dev@ubuntu:~/Qt/6.5.3/gcc_64/bin$ pwd
/home/dev/Qt/6.5.3/gcc_64/bin
dev@ubuntu:~/Qt/6.5.3/gcc_64/bin$ ls
androiddeployqt           lrelease                               qmake6
androiddeployqt.debug      lrelease.debug                         qml
androiddeployqt6            lupdate                                qml.debug
androidtestrunner          lupdate.debug                         qmldom
androidtestrunner.debug    materialeditor                      qmldom.debug
assistant                  materialeditor.debug                   qmleasing
assistant.debug             meshdebug                            qmleasing.debug
balsam                     meshdebug.debug                     qmlformat
balsam.debug               pixeltool                           qmlformat.debug
balsamui                   pixeltool.debug                     qmllint
balsamui.debug             qdbus                                 qmllint.debug
canbusutil                 qdbus.debug                          qmls
canbusutil.debug           qdbuscpp2xml                      qmls.debug
cooker                     qdbuscpp2xml.debug                   qmlplugindump
cooker.debug               qdbusviewer                         qmlplugindump.debug
designer                   qdbusviewer.debug                   qmlpreview
designer.debug             qdbusviewer.debug
```

QDBusViewer 툴을 이용해 D-Bus를 디버깅할 수 있다.



✓ D-Bus 의 Interface

고유한 D-Bus Service를 만들기 위해서 XML 형식의 Interface 파일을 먼저 만들어야 한다.

```
<node>
  <interface name="local.DBusChat">
    <method name="accelerate"/>
    <signal name="message">
      <arg name="nickname" type="s" direction="out"/>
      <arg name="text" type="s" direction="out"/>
    </signal>
    <signal name="action">
      <arg name="nickname" type="s" direction="out"/>
      <arg name="text" type="s" direction="out"/>
    </signal>
  </interface>
</node>
```

위의 예제는 D-Bus 서비스의 Interface 이다. 여기서는 method 는 Qt에서 Slot 함수와 동일하다. 예를 들어 위의 method 는 Qt 에서 accelerate() Slot 함수이고 이 method 가 D-Bus 에서 호출되면 Qt에서 accelerate() Slot 함수가 호출된다.

```
public slots:
    void accelerate();
```

그리고 signal 은 Qt 에서 Signal 과 동일하다. 예를 들어 위의 XML에서 message 시그널이 발생하면 Qt에서 Signal 이 발생한다. 이때 Signal 의 인자는 nickname 과 text 이다. 따라서 아래와 같은 시그널이 호출된다.

```
signals:
    void message(const QString &nickname, const QString &text);
```

Qt에서는 XML 파일을 자동으로 만들어주는 툴을 제공한다. "dbuscpp2xml" 명령어를 이용하면 자동으로 XML 파일을 만들어 준다. 이와 관련해서는 예제에서 자세히 알아보도록 하자.

✓ Interface Classes 와 Adaptor Classes

Qt에서는 Interface XML 파일 이외에도 Interface class 와 Adaptor class를 만들어야 한다. Qt는 개발자의 편의를 위해서 자동으로 만들어준다. Interface class 와 Adaptor class 를 만들기 위해서 CMakeList.txt 파일에 아래와 같이 추가해 주면 필요한 Class 들을 자동으로 생성해 준다.

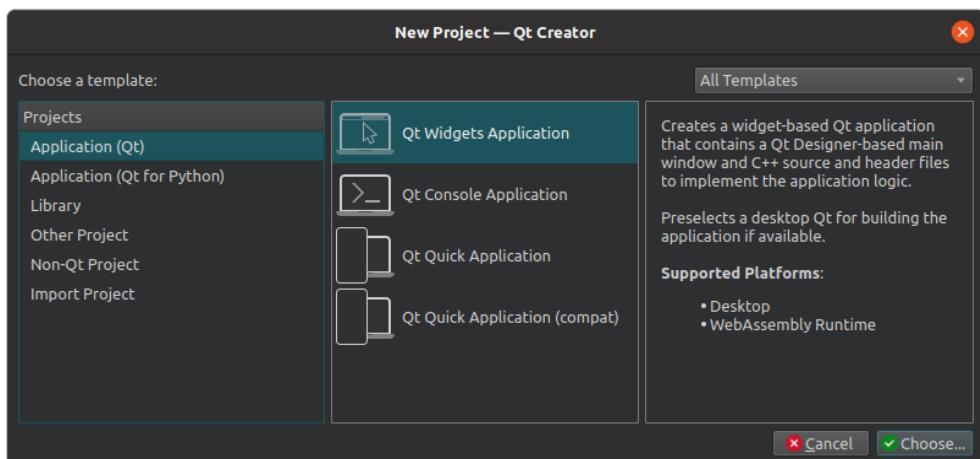
```
set(dbuschat_SRCS)
qt_add_dbus_interface(dbuschat_SRCS
    org.example dbuschat.xml
    dbuschat_interface
)

qt_add_dbus_adaptor(dbuschat_SRCS
    org.example dbuschat.xml
    qobject.h
    QObject
    dbuschat_adaptor
)
```

Interface Class 와 Adaptor Class를 자동으로 만드는 방법에 대해서는 예제에서 사용하는 방법을 자세히 다루어 보도록 하겠다.

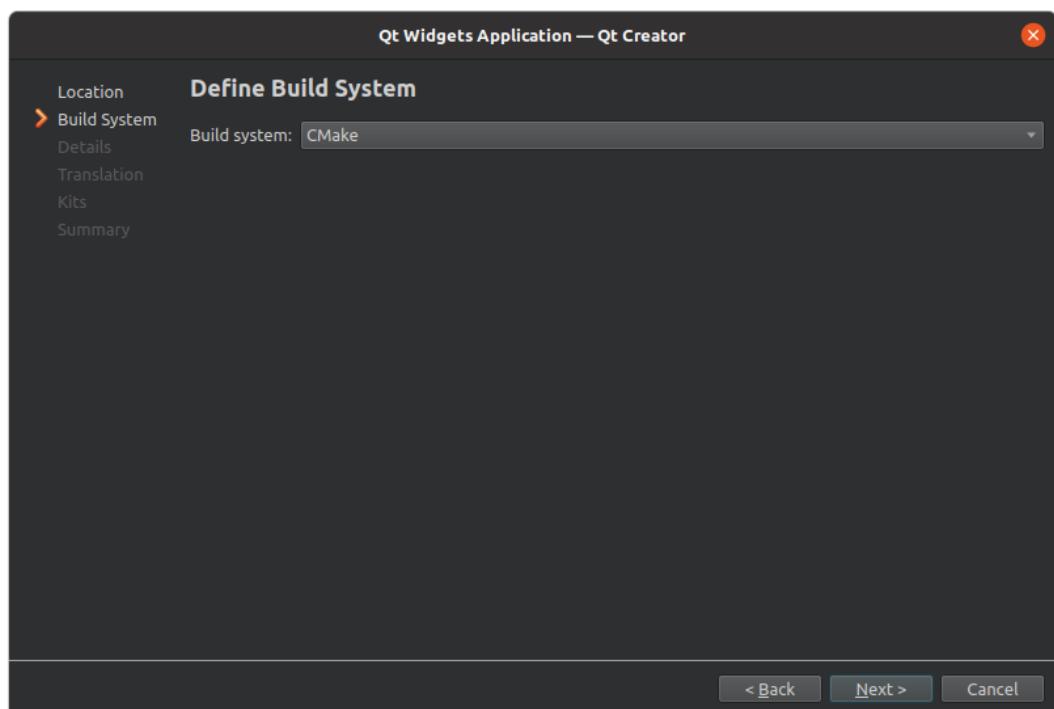
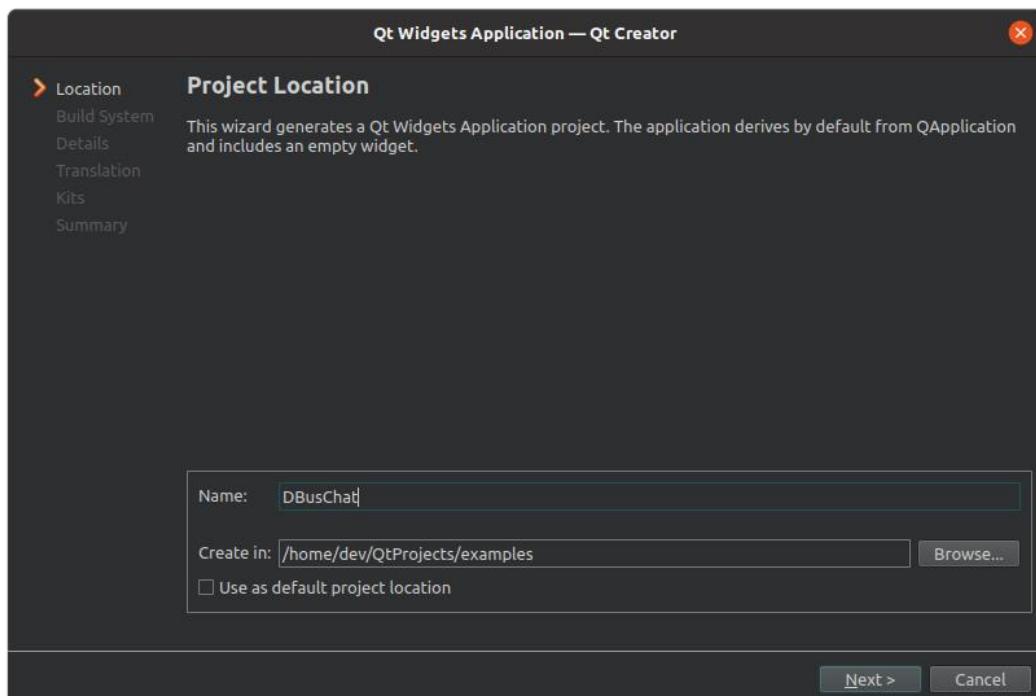
✓ D-Bus를 이용한 간단한 Chatting 예제

프로젝트 생성 시 Qt Widget 기반 프로젝트를 생성한다.

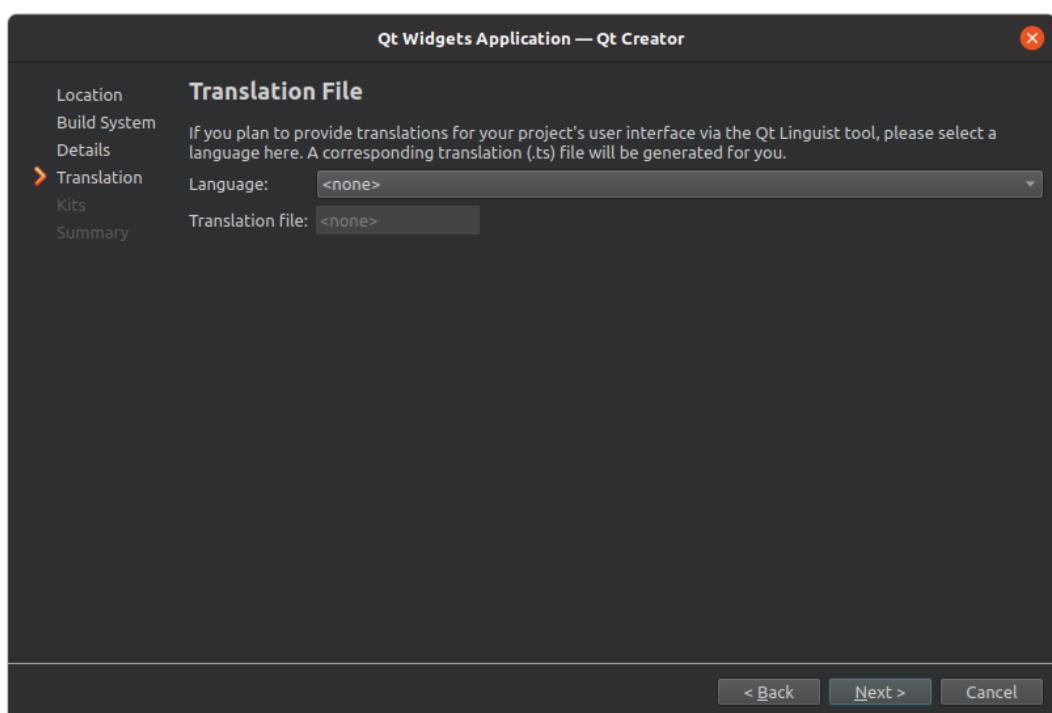
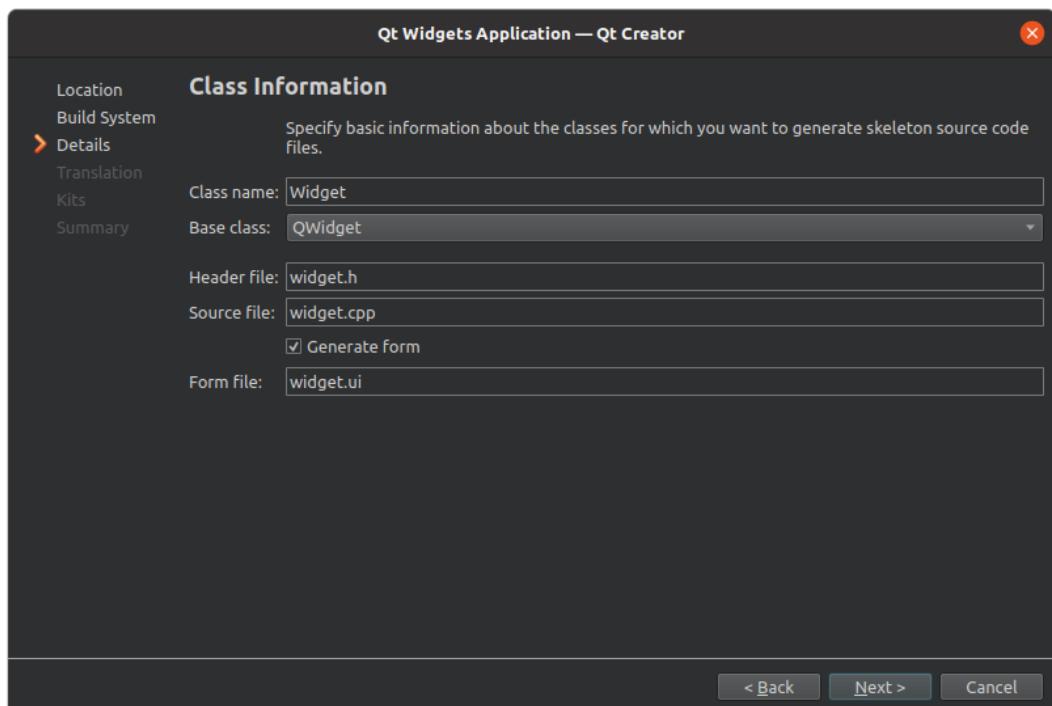


예수님은 당신을 사랑합니다.

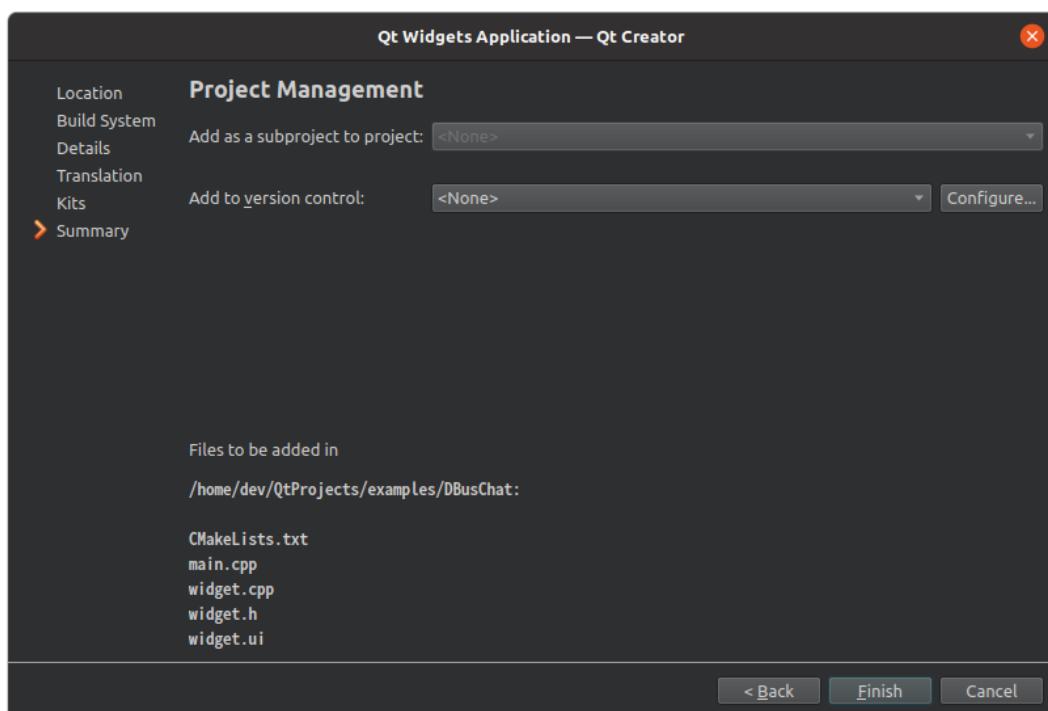
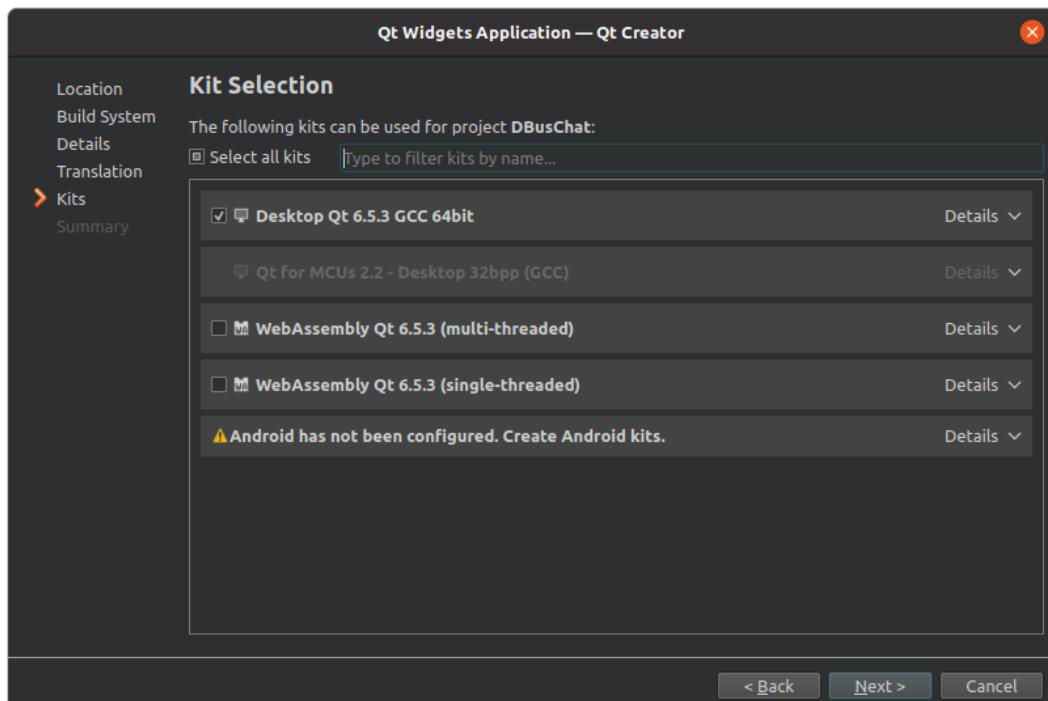
프로젝트 이름은 “DBusChat” 이라고 입력한다.



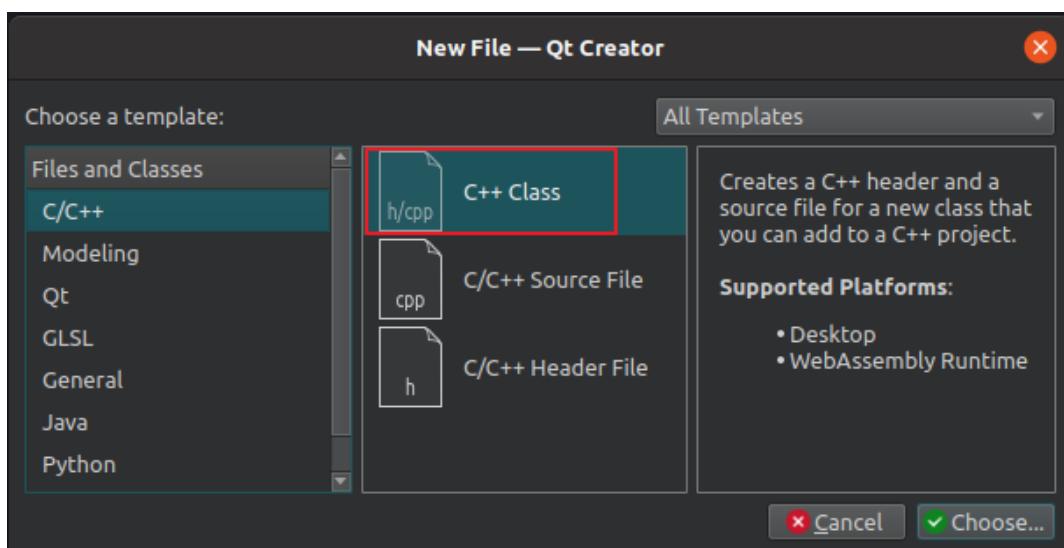
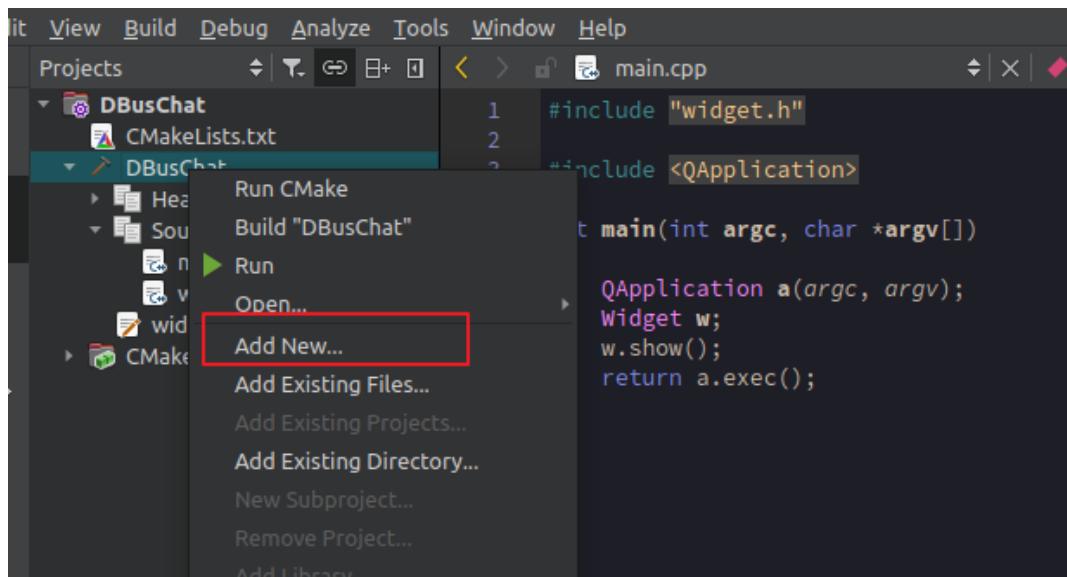
예수님은 당신을 사랑합니다.



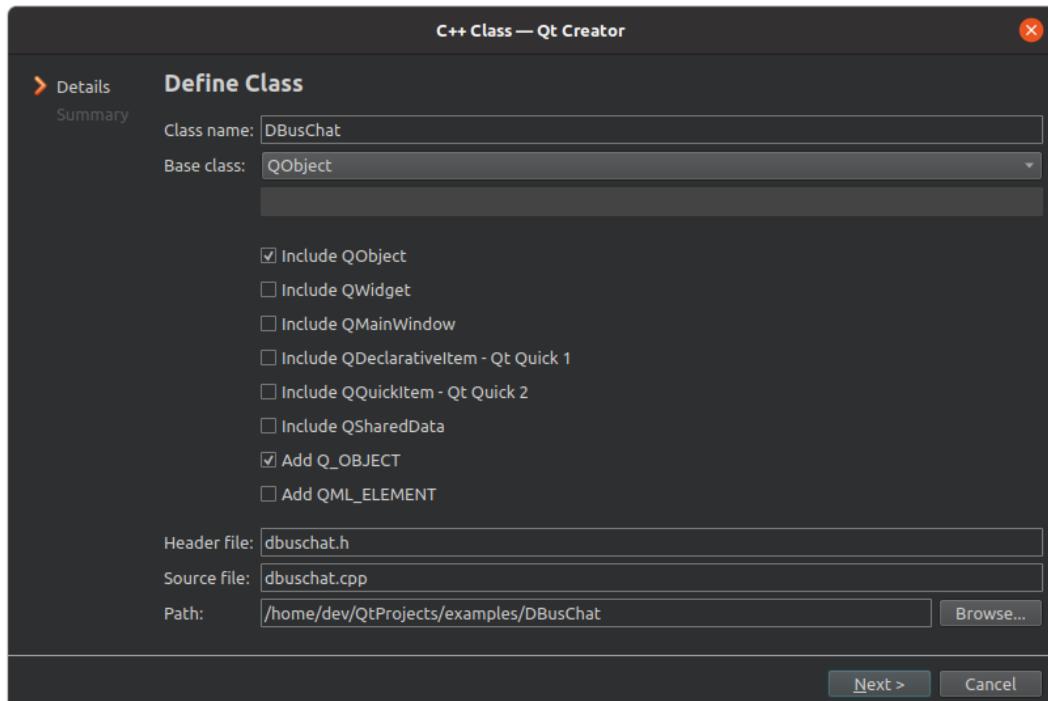
예수님은 당신을 사랑합니다.



프로젝트 생성이 완료된 후, 프로젝트 상에 DBusChat Class를 추가한다.



예수님은 당신을 사랑합니다.



프로젝트 생성이 완료되면 CMakeLists.txt 파일을 열어서 DBus 모듈을 아래와 같이 추가한다.

```
cmake_minimum_required(VERSION 3.5)
...
find_package(Qt6 REQUIRED COMPONENTS Widgets DBus)
...
target_link_libraries(DBusChat PRIVATE
    Qt6::DBus
    Qt6::Widgets
)
...
```

다음으로 dbuschat.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef DBUSCHAT_H
#define DBUSCHAT_H

#include <QObject>

class DBusChat : public QObject
```

예수님은 당신을 사랑합니다.

```
{  
    Q_OBJECT  
public:  
    explicit DBusChat(QObject *parent = nullptr);  
    void newConnection(const QString &nickname, const QString &text);  
    void newMessage(const QString &text);  
  
signals:  
    void message(const QString &nickname, const QString &text);  
    void action(const QString &nickname, const QString &text);  
  
    void uiDisplayMessage(const QString &text);  
  
private:  
    QString m_nickname;  
    void displayMessage(const QString &message);  
};  
  
#endif // DBUSCHAT_H
```

다음으로 dbuschat.cpp 소스코드 파일을 열어서 아래와 같이 작성한다.

```
#include <QApplication>  
#include <QDebug>  
  
#include "dbuschat.h"  
  
DBusChat::DBusChat(QObject *parent)  
    : QObject{parent}  
{  
}  
  
void DBusChat::displayMessage(const QString &message)  
{  
}  
  
void DBusChat::newConnection(const QString &nickname, const QString &text)
```

예수님은 당신을 사랑합니다.

```
{  
}  
  
void DBusChat::newMessage(const QString &text)  
{  
}
```

위와 같이 작성했다면 DBus 서비스의 인터페이스 파일(XML 형식의 파일)을 만들어야 한다. 일전에 설명 했던 것과 같이 직접 만들 수 있다. 하지만 Qt 자동으로 생성할 수 있는 기능을 제공한다.

DBusChat Class 에 등록된 Signal 과 Slot 함수를 추출해 자동으로 인터페이스 파일을 만들 수 있다.

터미널을 실행 후 소스코드 있는 디렉토리로 이동한다. 그리고 아래와 같이 터미널 창에 명령을 입력한다.

```
dev@ubuntu:~/examples/DBusChat$ ~/Qt/6.5.3/gcc_64/bin/qdbuscpp2xml ./dbuschat.h  
<!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS Object Introspection 1.0//EN"  
"http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">  
<node>  
  <interface name="local.DBusChat">  
    <signal name="message">  
      <arg name="nickname" type="s" direction="out"/>  
      <arg name="text" type="s" direction="out"/>  
    </signal>  
    <signal name="action">  
      <arg name="nickname" type="s" direction="out"/>  
      <arg name="text" type="s" direction="out"/>  
    </signal>  
    <signal name="uiDisplayMessage">  
      <arg name="text" type="s" direction="out"/>  
    </signal>  
  </interface>  
</node>  
dev@ubuntu:~/QtProjects/examples/DBusChat$
```

위와 같이 출력되면 <node> Tag 의 내용을 복사한 다음 “local.DbusChat.xml” 이라는 파일 이름으로 아래와 같이 저장한다.

예수님은 당신을 사랑합니다.

그리고 위의 XML 상에서 uiDisplayMessage Tag 는 DBus 에서 사용하지 않고 GUI 사용할 것이므로 이 Tag 는 삭제하고 아래와 같이 저장한다.

```
<node>
  <interface name="local.DBusChat">
    <signal name="message">
      <arg name="nickname" type="s" direction="out"/>
      <arg name="text" type="s" direction="out"/>
    </signal>
    <signal name="action">
      <arg name="nickname" type="s" direction="out"/>
      <arg name="text" type="s" direction="out"/>
    </signal>
  </node>
```

참고로 Qt는 “qdbusxml2cpp” 명령어를 제공한다. 이 명령어는 인터페이스 파일에서 Class를 추출하는 방법도 제공한다.

다시 내용으로 돌아와서, 위의 XML 형식의 내용을 local.DBusChat.xml 파일로 저장한다. 일전에 Qt에서 Adaptor class 와 Interface class 에 대해서 설명하였다.

여기서는 Adaptor class 와 Interface class를 자동으로 생성해 보겠다. 프로젝트 파일을 열어서 아래와 같은 라인을 추가한다.

```
cmake_minimum_required(VERSION 3.5)
project(DBusChat VERSION 0.1 LANGUAGES CXX)
...
find_package(Qt6 REQUIRED COMPONENTS WidgetsDBus)

set(dbuschat_SRCS)
qt_add_dbus_interface(dbuschat_SRCS
  local dbuschat.xml
  dbuschat_interface
)
qt_add_dbus_adaptor(dbuschat_SRCS
  local dbuschat.xml
  qobject.h
```

```
QObject
dbuschat_adaptor
)

qt_add_executable(DBusChat
    main.cpp
    widget.cpp
    widget.h
    widget.ui
    dbuschat.h dbuschat.cpp
    ${dbuschat_SRCS}
)

target_link_libraries(DBusChat PRIVATE
    Qt6::DBus
    Qt6::Widgets
)

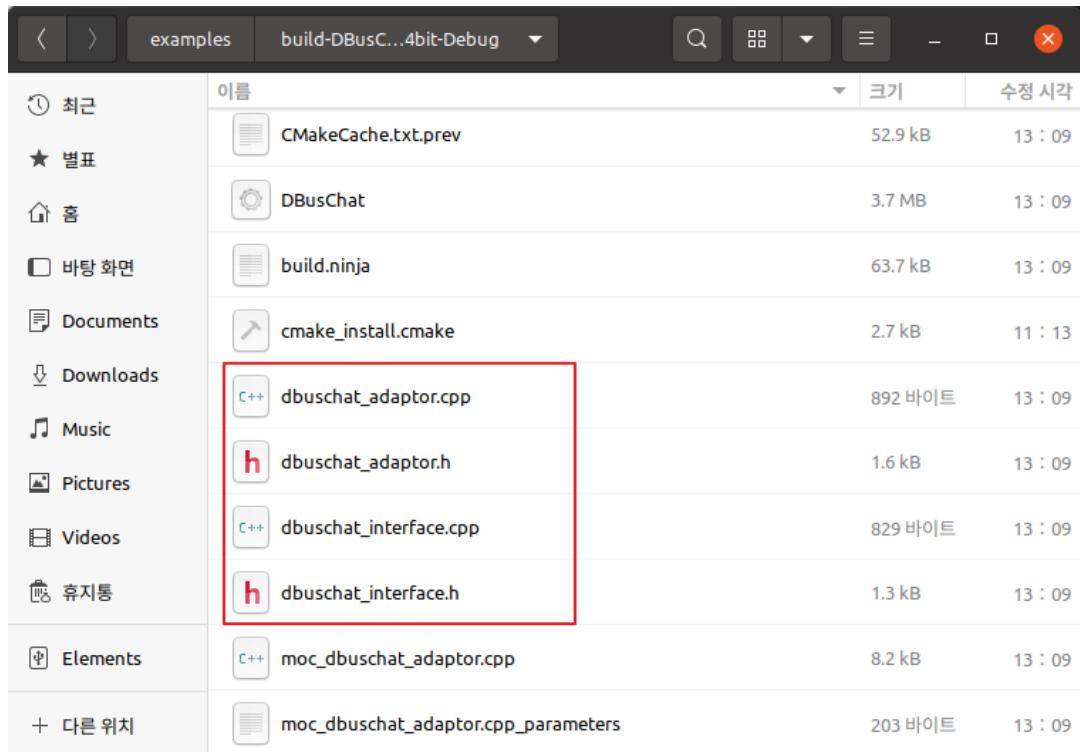
INCLUDE_DIRECTORIES(
    "../build-DBusChat-Desktop_Qt_6_5_3_GCC_64bit-Debug"
)

install(TARGETS DBusChat
    BUNDLE DESTINATION .
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

qt_add_dbus_interface() 는 Interface 클래스를 XML 파일 기반으로 자동으로 생성해준다. 그리고 qt_add_dbus_adaptor() 는 Adaptor 클래스를 XML파일 기반으로 자동으로 생성해준다.

그리고 하단의 INCLUDE_DIRECTORIES() 는 이 프로젝트의 빌드 디렉토리를 명시해 준다. 그 이유는 빌드 된 디렉토리에 Interface class 와 Adaptor class 가 생성되기 때문이다. 따라서 DBusChat 클래스에서 참조할 수 있기 때문이다.

위와 같이 작성하고 빌드한다. 빌드가 완료되면, 빌드 된 디렉토리로 이동하고 실제 이 디렉토리에 Interface class 와 Adapter class 가 생성되었는지 확인해보도록 하자.



위의 그림에서 보는 것과 같이 Interface class 와 Adaptor class 가 생성된 것을 확인 할 수 있을 것이다.

다시 dbuschat.cpp 소스코드 파일을 열어서 아래와 같이 소스코드를 추가한다.

```
#include <QApplication>
#include <QDebug>

#include "dbuschat.h"
#include "dbuschat_adaptor.h"
#include "dbuschat_interface.h"

DBusChat::DBusChat(QObject *parent)
    : QObject{parent}
{
    qDebug() << Q_FUNC_INFO;

    new DBusChatAdaptor(this);

    auto connection = QDBusConnection::sessionBus();
```

예수님은 당신을 사랑합니다.

```
connection.registerObject("/", this);

using local::DBusChat;

auto *iface = new DBusChat({}, {}, connection, this);

connect(iface, &DBusChat::message, this, [this](const QString &nickname,
const QString &text) {
    displayMessage(tr("<%1> %2").arg(nickname, text));
});

connect(iface, &DBusChat::action, this, [this](const QString &nickname, const
QString &text) {
    displayMessage(tr("* %1 %2").arg(nickname, text));
});
}

void DBusChat::displayMessage(const QString &message)
{
    emit uiDisplayMessage(message);
}

void DBusChat::newConnection(const QString &nickname, const QString &text)
{
    m_nickname = nickname;

    emit action(nickname, text);
}

void DBusChat::newMessage(const QString &text)
{
    emit message(m_nickname, text);
}
```

이 클래스 생성자 함수에서는 D-Bus를 초기화한다. 그리고 D-Bus 서비스에서 Signal이 발생하면 displayMessage() 함수를 실행하였다. Connect() 함수에서는 C++ lambda를 사용하였다.

예수님은 당신을 사랑합니다.

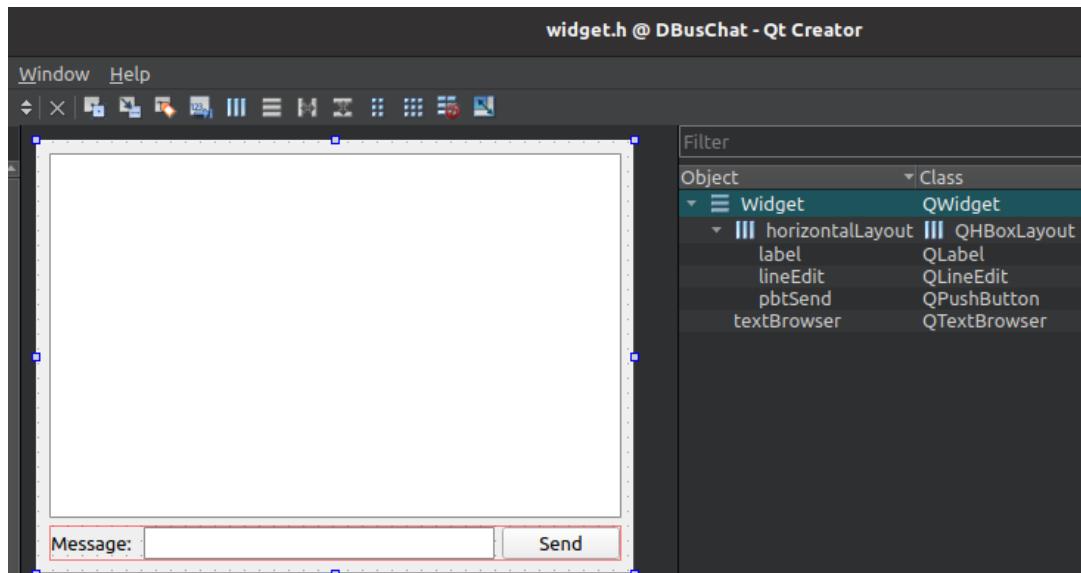
newConnection() 멤버 함수가 호출 되면, D-Bus 서비스에 action() Signal을 호출한다.
따라서 D-Bus 서비스에 연결된 Process 들은 action() Signal 이 발생한다.

그리고 newMessage() 멤버 함수가 호출 되면 D-Bus 서비스의 message() Signal 이 호출된다. 그러므로 D-Bus 서비스에 연결된 Process 들은 message() Signal 이 발생한다.

다음으로 main.cpp 에서 w.show() 함수를 제거하고 아래와 같이 작성한다.

```
#include "widget.h"  
#include <QApplication>  
  
int main(int argc, char *argv[])  
{  
    QApplication a(argc, argv);  
    Widget w;  
  
    return a.exec();  
}
```

다음으로 widget.ui 파일을 열어서 아래와 같이 GUI상에 위젯들을 배치한다.



다음으로 widget.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef WIDGET_H  
#define WIDGET_H
```

```
#include <QWidget>
#include "dbuschat.h"

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

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

signals:
    void message(const QString &nickname, const QString &text);
    void action(const QString &nickname, const QString &text);

private:
    Ui::Widget *ui;
    DBusChat *m_dbusChat;

    QStringList m_msgList;
};

#endif // WIDGET_H
```

다음은 widget.cpp 소스코드 파일을 열어서 아래와 같이 작성한다.

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

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

예수님은 당신을 사랑합니다.

```
auto newNickname = QInputDialog::getText(this,
                                         tr("Set nickname"),
                                         tr("New nickname:"));

if(newNickname.isEmpty())
{
    deleteLater();
}
else
{
    this->show();
    m_dbusChat = new DBusChat(this);
    m_dbusChat->newConnection(newNickname, "joins the chat.");

    connect(m_dbusChat, &DBusChat::uiDisplayMessage,
            this, [this](const QString &text) {

        m_msgList.append(text);
        auto history = m_msgList.join(QLatin1String("\n"));
        ui->textBrowser->setPlainText(history);
    });

    connect(ui->pbtSend, &QPushButton::clicked, this, [this]() {
        m_dbusChat->newMessage(ui->lineEdit->text().trimmed());
    });
}

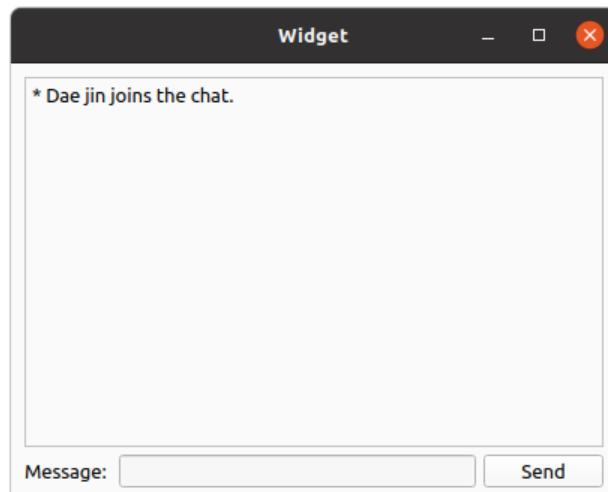
Widget::~Widget()
{
    delete ui;
}
```

Widget 클래스 생성자 함수에서 QInputDialog 클래스를 이용해 사용자로부터 입력을 받을 수 있는ダイ얼로그가 먼저 로딩된다. 여기서 Nick Name 을 입력한다.

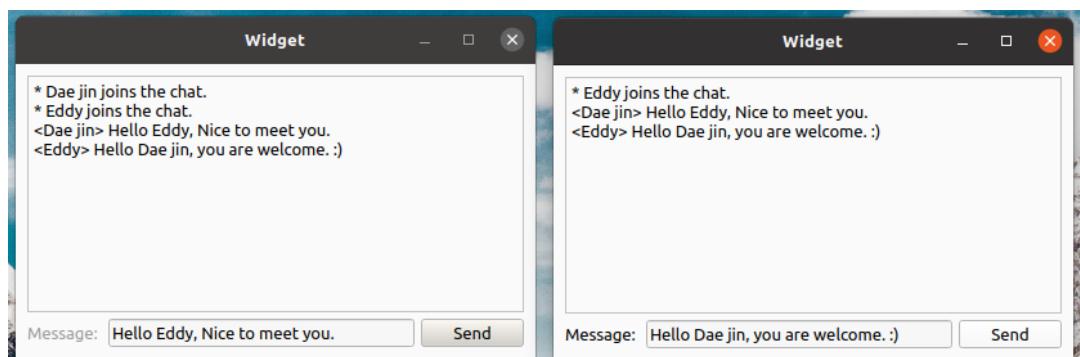
예수님은 당신을 사랑합니다.



그러면 아래와 Widget 이 로딩된다.



위와 같이 실행되었다면 정상적으로 메시지가 D-Bus 를 통해서 다른 Process 에게 전달되는지 확인해보자. 이 예제를 추가로 아래와 같이 실행해 보도록 하자. 그런 다음 메시지를 전달해 보도록 하자.



이 예제의 전체 소스코드는 00_DBusChat 디렉토리를 참조하면 된다.

38. Multimedia

Qt에서 제공하는 Multimedia 모듈은 오디오, 비디오, 카메라를 이용해 응용 어플리케이션을 개발할 수 있다.

주의 사항으로 Qt 5.x.x 버전과 Qt 6.5.x 이후 버전에서 많은 변화가 있었다. 따라서 Qt5에서 제공하는 Multimedia 모듈을 이용해 구현한 소스코드가 Qt 6에서는 호환 되지 않는 부분이 많이 있다.

따라서 Qt 5.x.x 버전에서 구현한 소스코드를 Qt 6 이후 버전에서 그대로 사용하는 것은 불가능 하다. 여기서는 Qt 6.5.x 이후 버전을 사용할 것이다. 만약 Qt 5.x 버전을 사용하고 있다면 Qt 6.5.x 이상 버전으로 재 설치해야 한다.

그럼 다시 본론으로 돌아와, 계속해 Qt Multimedia에 대해서 알아보도록 하겠다.

Qt에서 Multimedia 모듈을 사용하기 위해서는 프로젝트 파일에 아래와 같이 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS Multimedia)
target_link_libraries(my_project PRIVATE Qt6::Multimedia)
```

이외에도 만약 여러분이 QVideoWidget 등과 같은 멀티미디어 GUI 위젯을 사용해야 하는 경우에 아래와 같이 MultimediaWidgets를 아래와 같이 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS Multimedia MultimediaWidgets)
target_link_libraries(my_project PRIVATE
    Qt6::Multimedia
    Qt6::MultimediaWidgets
)
```

만약 qmake를 사용한다면 프로젝트 파일에 아래와 같이 추가하면 된다.

```
QT += multimedia multimediawidgets
```

Multimedia 모듈 이벤트가 Signal과 Slot을 사용하므로 구현하기가 매우 쉬운 편이다.

예수님은 당신을 사랑합니다.

물론 단순히 음악 파일 또는 동영상 파일을 재생하는 기능을 구현하는 것은 간단하다. 하지만 좀더 세부적인 기능을 구현해야 할 때, Signal 과 Slot을 사용하면 매우 쉽게 구현할 수 있다.

예를 들어 인터넷 라디오 방송국을 구현하는 예를 들어보자. 마이크로 입력된 음원을 100 Millisecond Seconds 단위로 컴퓨터 네트워크 망을 경유해 송신하고, 수신 받은 컴퓨터에서 이 음원을 재생하는 응용 어플리케이션을 구현해야 한다고 한다면

단순히 음악 파일을 재생하는 API로는 구현하기 어렵다. 따라서 Qt에서 제공하는 Multimedia 모듈은 단순한 기능부터 좀더 복잡한 멀티미디어 응용 어플리케이션을 보다 쉽게 구현할 수 있는 장점이 있다.

또한 플랫폼에 구애 받지 않고 Qt에서 제공하는 Multimedia 모듈을 동일하게 사용할 수 있다. 예를 들어 MS Windows 플랫폼에서 Qt Multimedia 모듈을 이용해 구현한 응용 어플리케이션을 Linux 또는 MacOS에서도 동일하게 사용할 수 있을 뿐만 아니라 Mobile 플랫폼에서도 동일하게 사용할 수 있다.

이번 장에서는 Qt에서 제공하는 Multimedia 모듈을 아래와 같이 3개의 Chapter로 나누어 알아보도록 하자.

① Audio

음악 파일을 단순히 재생하는 기능부터 Low Level에서 음원을 다루는 방법에 대해서 살펴볼 것이다.

② Video

동영상 파일을 재생하는 방법에 대해서 살펴보도록 하자.

③ Camera

시스템에서 사용 가능한 카메라 디바이스를 검색하고 카메라로부터 받은 영상을 GUI에 출력하는 방법에 대해서 살펴보도록 하자.

38.1. Audio

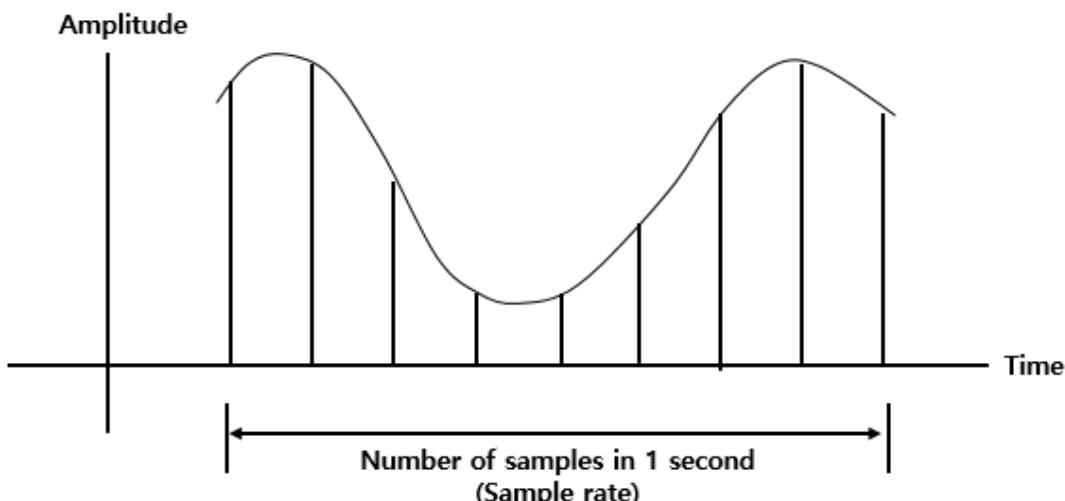
Qt Multimedia 모듈을 이용해 어플리케이션을 구현하는 방법을 알아보기 전에 필수적인 Audio 기초 지식을 살펴 보도록 하자. Audio 대한 기초 지식이 필요 없다면 Audio basics 부분을 건너뛰어도 무방하다.

- ✓ **Audio basics**

사람이 말할 때 음파가 아날로그 파형 형태로 다른 사람의 귀에 전달되지만 컴퓨터에서 음원을 디지털 기기에서 재생하기 위해서 아날로그 파형을 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.

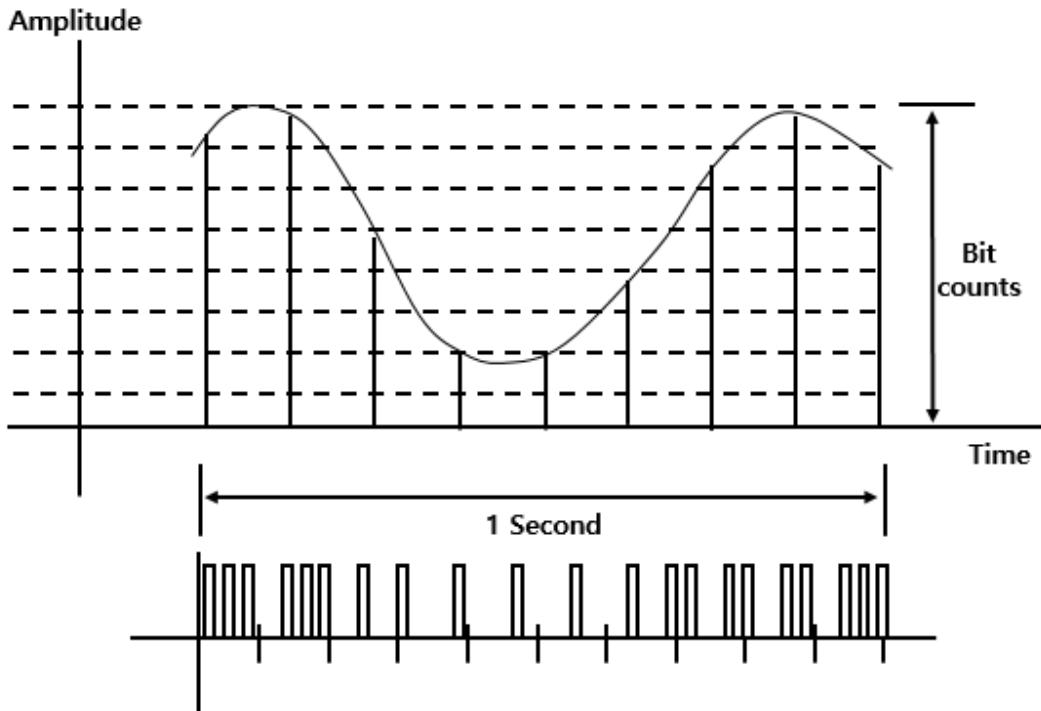


예를 들어 52.1 KHz는 1초 동안에 52,100개의 샘플을 추출한다는 뜻이다. 샘플 개수가 더 커질 수록 더욱 세밀해진다는 것을 알 수 있다. 우리가 CD음질로 사용하는 음질은

44.1Khz를 사용한다.

✓ Bit rate

1초동안 사용한 Bit 수를 의미하며 bps 단위를 사용한다. 우리가 음원의 품질을 말할 때 96 Kbps, 128 Kbps, 192 Kbps 등과 같이 bps를 말한다.



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 59초)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 수가 된다.

그리고 PCM 을 다음과 같이 구할 수 있다. PCM 파일 크기는 “PCM 초당 데이터 처리량 * 재생 시간 / 8” 을 하면 PCM 파일 크기를 계산할 수 있다.

Example calculation using duration, sample rate and bit rate

About playback

Duration: 299.277 seconds (4 minutes, 59 seconds)
Bitrate: 56 Kbps (CBR)
Channels: 2
Bit: 16-bit
Sample Rate: 22050 Hz

Sample.mp3 – Calculate file size for MP3 format

File size = Duration * Bit rate / 8
 $299.277 * 56,000(\text{Hz}) / 8 = \textcolor{blue}{2,094,939} \text{ Bytes}$

Sample.pcm – Calculate file size for PCM format

PCM File size = PCM data throughput per second * Duration / 8
 $705,600 * 299.277 / 8 = \textcolor{blue}{26,394,624} \text{ Bytes}$

PCM data throughput per second = Sample rate * Channel * Bit
 $22,050(\text{Hz}) * 2 * 16 \text{ (bit)} = 705,600 \text{ Bit (88,200 Kbytes)}$

오디오에서 Channel 이란 한 방향으로 소리를 전달하는 것을 Mono, 두 개의 채널, 즉 2두 개의 방향으로 소리를 전달하는 것을 스테레오 채널이라고 한다.

예를 들어 녹음 과정에서 몇 개의 방향으로 소리를 나누어 구분하는 것인지를 의미한다. 여기에서 MP3 파일 계산 시 참조한 Channel은 방금 말한 채널을 의미한다. 그리고 여기에서 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.

다음으로 Qt Multimedia 모듈을 이용해 간단한 MP3 파일을 재생하는 방법을 알아보도록 하다.

단순히 MP3를 재생하기 위해서 QMediaPlayer 클래스를 이용해 쉽게 재생하는 기능을 구현할 수 있다.

```
m_player = new QMediaPlayer();
m_audioOutput = new QAudioOutput;
m_player->setAudioOutput(m_audioOutput);
...
float volValue = ui->volSlider->value() / 100.0;
m_audioOutput->setVolume(volValue);
```

QMediaPlayer 클래스는 Container 라는 개념으로 생각하면 된다. QMediaPlayer 클래스는 Audio 이외에도 Video, Camera의 Container 기능을 제공한다.

음악파일을 재생하기 위해서는 QMediaPlayer 상에 QAudioOutput 클래스의 오브젝트를 setAudioOutput() 멤버 함수를 이용해 연결해야 한다.

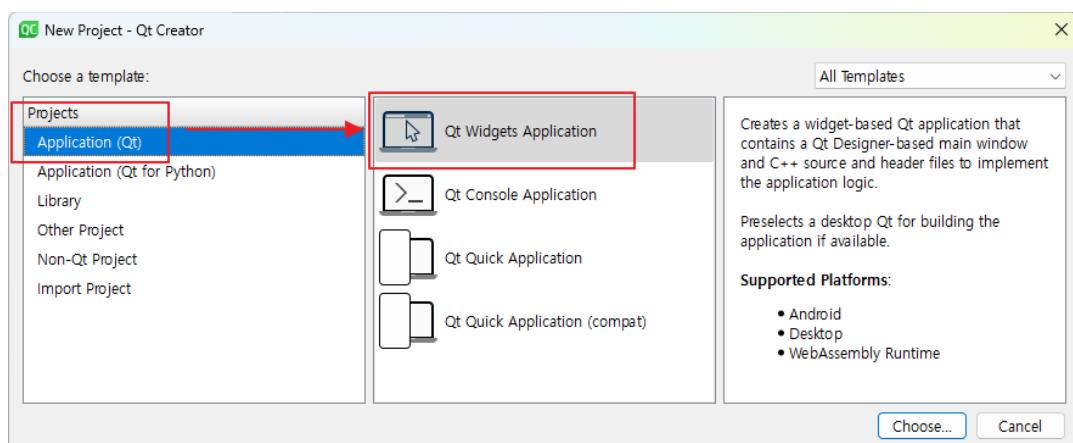
다음으로 음악파일을 지정하기 위해서 QMediaPlayer 클래스의 setSource() 멤버 함수를 사용하면 된다. 그런 다음 음악파일을 재생하기 위해서 play() 멤버 함수를 사용하면 된다.

```
m_player->setSource(QUrl::fromLocalFile(m_fName));
ui->sliderPosition->setEnabled(true);
m_player->play();
```

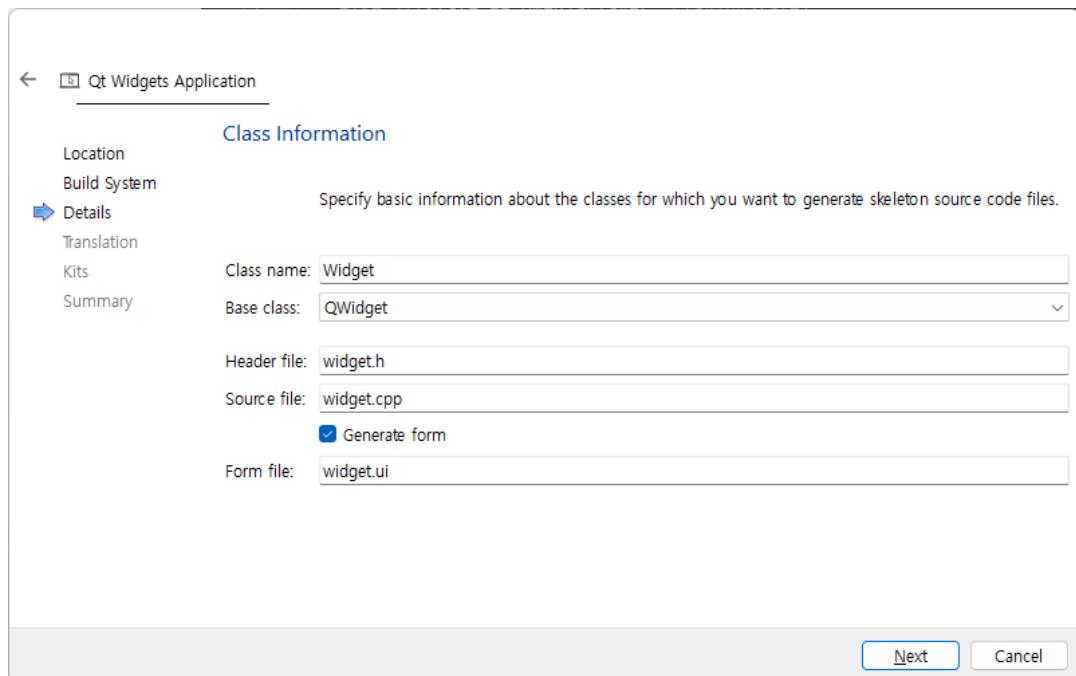
음악 파일을 재생하는 방법에 대해 간단히 살펴 보았다. 다음으로 실제 예제를 통해 간단한 음악파일 재생 예제를 구현해 보도록 하자.

✓ 간단한 Audio 재생 예제 구현

프로젝트 생성 시, Qt Widget 기반의 Application 을 선택한다.



UI Form을 사용할 것이므로 아래와 같이 [Generation form]에 체크한다.



프로젝트 생성이 완료되면 CMakeLists.txt 파일을 열어서 Multimedia 모듈을 아래와 같이 추가한다.

```
cmake_minimum_required(VERSION 3.5)

project(00_SimpleAudioPlayer VERSION 0.1 LANGUAGES CXX)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

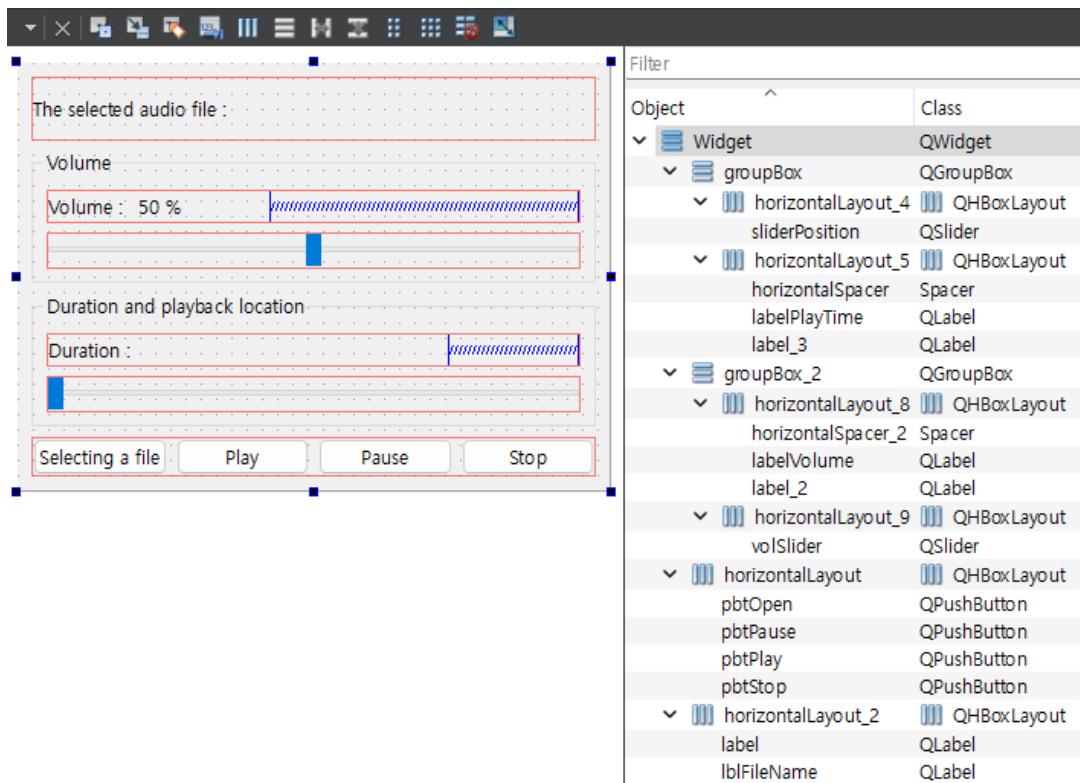
find_package(Qt6 REQUIRED COMPONENTS Widgets Multimedia)

set(PROJECT_SOURCES
    main.cpp
    widget.cpp
    widget.h
    widget.ui)
```

예수님은 당신을 사랑합니다.

```
)  
  
qt_add_executable(00_SimpleAudioPlayer  
    ${PROJECT_SOURCES}  
)  
  
target_link_libraries(00_SimpleAudioPlayer PRIVATE  
    Qt6::Widgets  
    Qt6::Multimedia  
)  
  
install(TARGETS 00_SimpleAudioPlayer  
    BUNDLE DESTINATION .  
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}  
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}  
)
```

다음으로 widget.ui 파일을 열어 아래와 같이 GUI 위젯들을 배치한다.



다음으로 widget.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QMediaPlayer>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

    QString        m_fName;
    QMediaPlayer   *m_player;
    QAudioOutput   *m_audioOutput;
    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
```

다음으로 widget.cpp 소스코드 파일을 아래와 같이 작성한다.

```
#include "widget.h"
#include "./ui_widget.h"
#include <QFileDialog>
#include <QTime>
#include <QAudioOutput>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    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();
    m_audioOutput = new QAudioOutput;
    m_player->setAudioOutput(m_audioOutput);

    float volValue = ui->volSlider->value() / 100.0;
    m_audioOutput->setVolume(volValue);

    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));
    float volValue = val / 100.0;
    m_audioOutput->setVolume(volValue);
}

void Widget::onOpenBtn()
{
    m_fName = QFileDialog::getOpenFileName(this,
                                           tr("Open File"),
                                           QDir::homePath(),
                                           tr("MP3 files (*.mp3);"));

    if(!m_fName.isNull())
        ui->lblFileName->setText(m_fName);
}

void Widget::onPlayBtn()
{
    if(!m_fName.isNull())
    {
        m_player->setSource(QUrl::fromLocalFile(m_fName));
        ui->sliderPosition->setEnabled(true);
        m_player->play();
    }
}
```

```
void Widget::onPauseBtn()
{
    int state = m_player->playbackState();
    if(state == QMediaPlayer::PausedState)
    {
        m_player->play();
    }
    else if(state == QMediaPlayer::PlayingState)
    {
        m_player->pause();
    }
}

void Widget::onStopBtn()
{
    int state = m_player->playbackState();

    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;
}
```

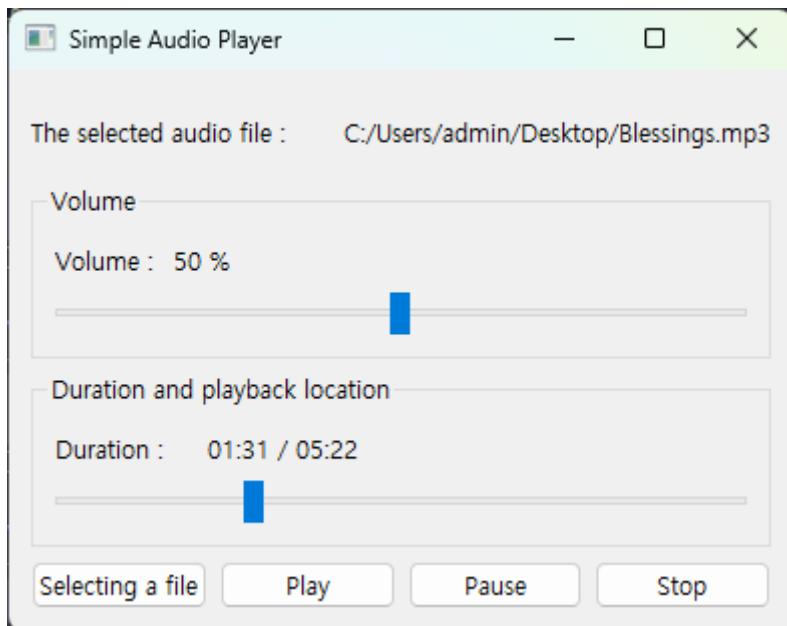
생성자 함수에서는 GUI 상에의 각 버튼의 Signal 과 Slot을 연결하고 QMediaPlayer 클래스와 QAudioOutput 클래스 오브젝트를 생성한다. 그리고 QMediaPlayer 클래스의 setAudioOutput() 멤버 함수에 QAudioOutput 클래스의 오브젝트를 인자로 넘겨준다.

그리고 QMediaPlayer 클래스에서 재생 시간이 변경되었을 때 durationChanged() 시그

예수님은 당신을 사랑합니다.

널이 발생한다. 따라서 이 Signal 을 durationChanged() Slot 함수와 연결한다.

QMediaPlayer 클래스의 positionChanged 는 재생 되고 있는 위치가 변경 되었을 때 알려주는 Signal 이다. 이 Signal 은 positionChanged() Slot 함수와 연결되어 있다. 주의 사항으로 Volume 을 조절할 때 값의 최소값은 0.0 ~ 1.0 이다. 여기까지 소스코드 작성을 완료 하였다면 빌드 후 실행해 보도록 하자.

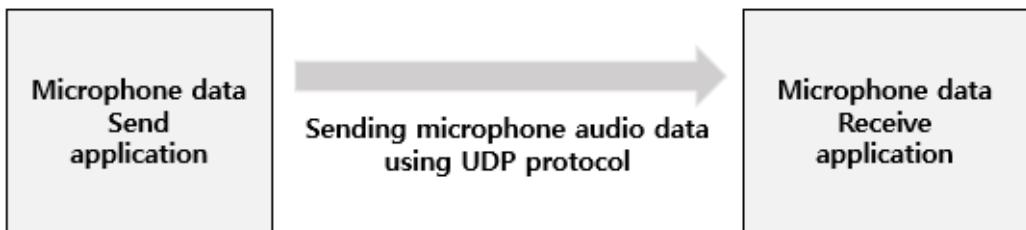


이 예제의 전체 소스코드는 00_SimpleAudioPlayer 디렉토리를 참조하면 된다.

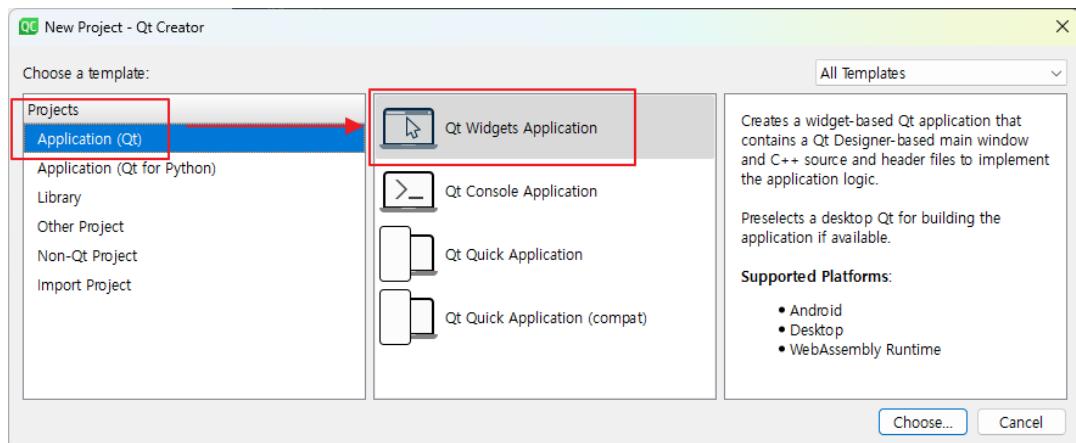
- ✓ 마이크 입력 신호를 네트워크(UDP)로 전송하는 예제

이번에는 2개의 예제를 구현할 것이다. 첫 번째는 첫 번째 예제는 마이크로부터 Audio 데이터를 추출해 네트워크로 전송하는 예제이다.

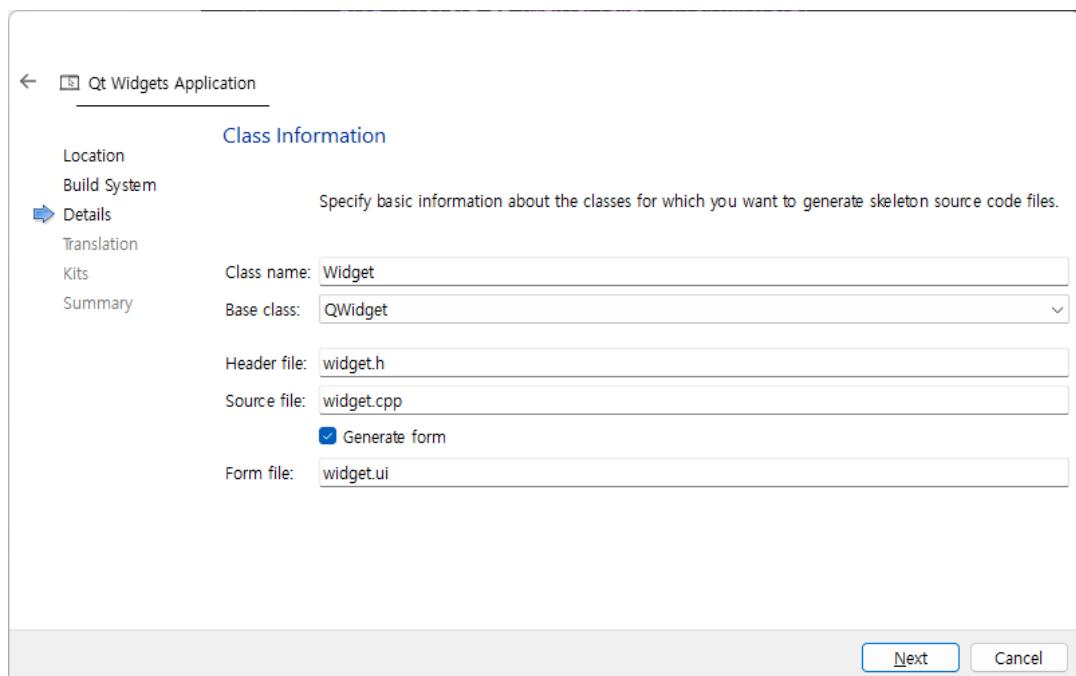
두 번째 예제는 네트워크를 경유해 수신한 데이터를 스피커로 출력하는 예제이다.



첫 번째 어플리케이션을 먼저 구현해 보도록 하자. 프로젝트 생성 시, Qt Widget 기반의 Application 을 생성한다.



이 프로젝트에서는 UI Form을 사용할 것이므로 아래와 같이 [Generation form]에 체크 한다.



프로젝트 생성이 완료되면 CMakeLists.txt 파일에 Multimedia 모듈과 Network 모듈을 아래와 같이 추가한다.

```
cmake_minimum_required(VERSION 3.5)

project(01_AudioSender VERSION 0.1 LANGUAGES CXX)

set(CMAKE_AUTOUIC ON)
```

```
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 REQUIRED COMPONENTS
    Widgets
    Multimedia
    Network
)

set(PROJECT_SOURCES
    main.cpp
    widget.cpp
    widget.h
    widget.ui
)

qt_add_executable(01_AudioSender
    ${PROJECT_SOURCES}
)

target_link_libraries(01_AudioSender PRIVATE
    Qt6::Widgets
    Qt6::Multimedia
    Qt6::Network
)

install(TARGETS 01_AudioSender
    BUNDLE DESTINATION .
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

다음으로 widget.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QtMultimedia>
#include <QUdpSocket>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

    QMediaDevices      *m_devices = nullptr;
    QList<QAudioDevice> devList;
    QAudioFormat       mFormat;
    QAudioDevice       mDeviceInfo;

    QAudioInput        *mAudioInput = nullptr;
    Q AudioSource      *m_audioSource = nullptr;

    QIODevice          *mInput;
    QByteArray         mBuffer;
    int                mInputVolume;

    QUdpSocket         mUdpSocket;
    void audioInitialize();
```

```
private slots:  
    void volSliderChanged(int val);  
    void sendStartBtn();  
    void sendStopBtn();  
  
    void stateChanged(QAudio::State state);  
    void readMore();  
};  
  
#endif // WIDGET_H
```

다음은 widget.cpp 소스코드 파일을 열어서 아래와 같이 작성한다.

```
#include "widget.h"  
#include "./ui_widget.h"  
  
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()  
{
```

```
m_devices = new QMediaDevices(this);

devList = m_devices->audioInputs();
for(int i = 0; i < devList.size(); ++i)
    ui->comboDevList->addItem(devList.at(i).description());

mDevInfo = devList.at(ui->comboDevList->currentIndex());

mFormat.setSampleRate(8000);
mFormat.setChannelCount(1);

mFormat.setSampleFormat(QAudioFormat::Int16);

mAudioInput = new QAudioInput(mDevInfo, this);
mInputVolume = ui->volSlider->value();
ui->volLabel->setText(QString("%1 %").arg(mInputVolume));

mBuffer = QByteArray(14096, 0);
}

void Widget::volSliderChanged(int val)
{
    mInputVolume = val;
    ui->volLabel->setText(QString("%1 %").arg(mInputVolume));
    mAudioInput->setVolume(qreal(mInputVolume) / 100);
}

void Widget::sendStartBtn()
{
    mDevInfo = devList.at(ui->comboDevList->currentIndex());
    mAudioInput = new QAudioInput(mDevInfo, this);
    mAudioInput->setVolume(qreal(mInputVolume) / 100);
    m_audioSource = new Q AudioSource(mDevInfo, mFormat);
    m_audioSource->setBufferSize(200);

    connect(m_audioSource, SIGNAL(stateChanged(QAudio::State)),
```

예수님은 당신을 사랑합니다.

```
this,           SLOT(stateChanged(QAudio::State));  
  
mInput = m_audioSource->start();  
connect(mInput, SIGNAL(readyRead()), this, SLOT(readMore()));  
}  
  
void Widget::sendStopBtn()  
{  
    if(mInput != nullptr) {  
        disconnect(mInput, nullptr, this, nullptr);  
        mInput = nullptr;  
    }  
  
    m_audioSource->stop();  
}  
  
void Widget::stateChanged(QAudio::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 = m_audioSource->bytesAvailable();  
    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;
}
```

audioInitialize() 멤버 함수는 생성자 함수에서 호출된다. 이 함수에서는 Audio Input Device의 목록을 얻어와 Combo Box 에 등록한다. Audio Input Device 목록을 얻어오기 위해서 QMediaDevices 클래스를 이용한다.

QAudioInput 클래스는 Microphone 과 같은 입력 디바이스로부터 입력되는 음성 데이터를 추출하기 위한 목적으로 제공된다.

그리고 실제 음성 데이터를 읽어오는 클래스로 QIODevice 로부터 데이터를 읽어온다. 따라서 QIODevice 클래스는 마이크 신호를 QByteArray 저장한다.

그런 다음 QByteArray 에 저장된 데이터를 네트워크(UDP)를 이용해 송신한다.

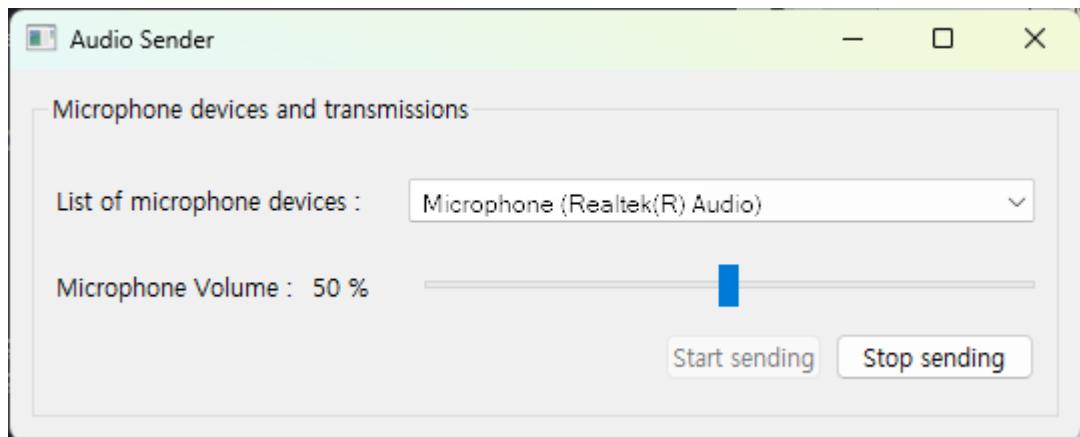
volSliderChanged() Slot 함수는 GUI 상에서 Volume 을 조절할 수 있는 기능을 제공한다. sendStartBtn() Slot 함수는 GUI 상에서 [Start sending] 버튼을 클릭하면 호출된다. 이 함수에서는 Combo box 상에서 선택되어 있는 Microphone device를 QAudioInput 클래스의 오브젝트 선언 시, 첫 번째 인자로 넘겨준다.

그리고 QAudioInput 클래스 오브젝트가 Microphone device 로부터 데이터를 추출한다. 데이터 추출은 Signal 과 Slot 에 의해서 추출된다.

예를 들어서 Microphone device 로부터 추출되면 readyRead() Signal 이 호출된다. 그러면 이 Signal 과 연결된 readMore() Slot 함수가 호출된다.

readMore() Slot 함수에서는 Microphone device 로부터 추출된 데이터를 네트워크를 경유해 전송한다.

예수님은 당신을 사랑합니다.

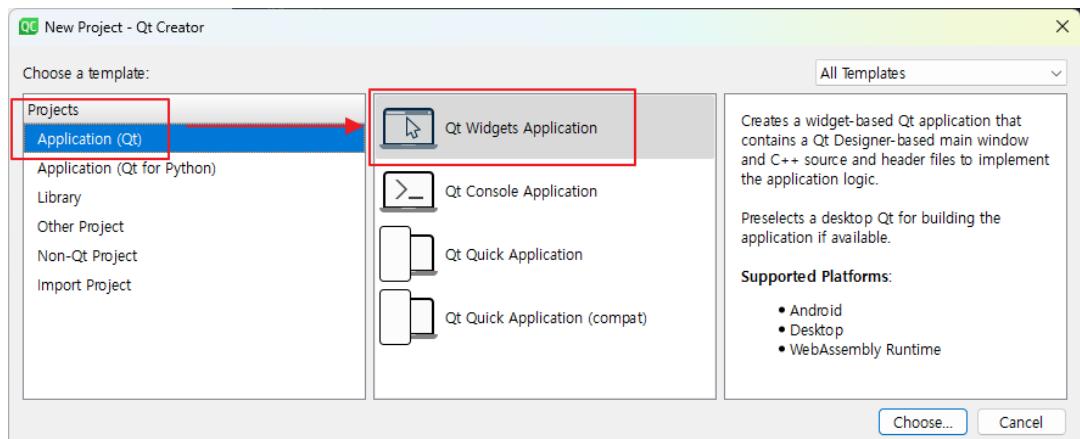


이 예제의 소스코드는 01_AudioSender 디렉토리를 참조하면 된다.

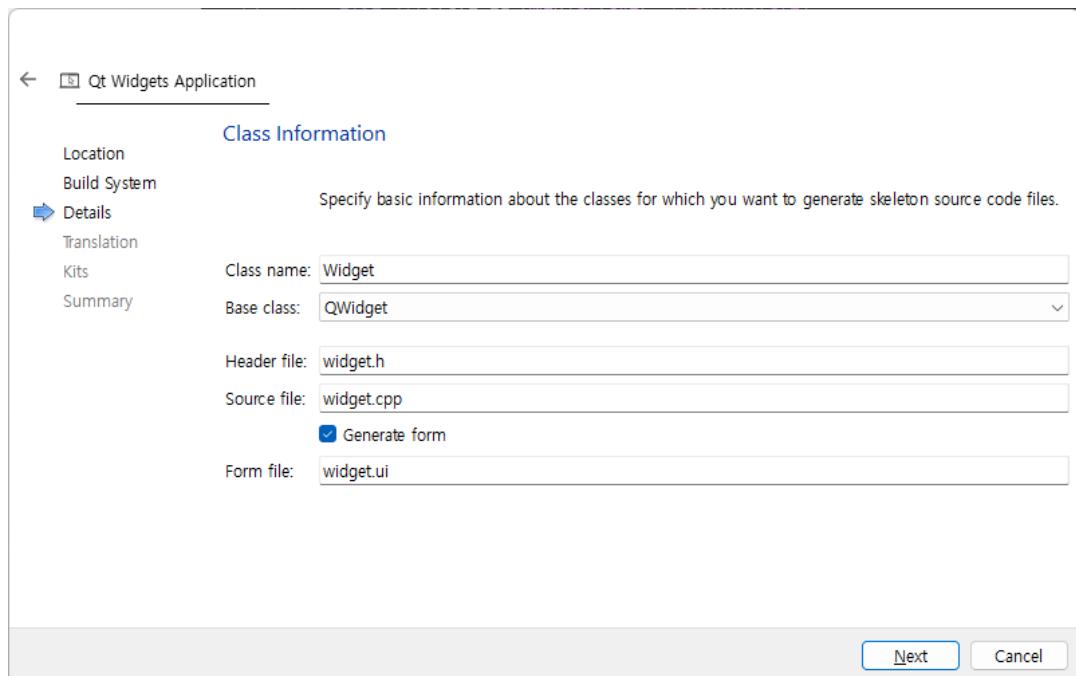
- ✓ 네트워크로 전달 받은 데이터를 스피커로 출력하는 예제

이번에는 송신 예제에서 수신 받은 데이터를 스피커로 출력하는 예제를 작성해 보도록 하자.

프로젝트 생성 시 Qt Widget 기반으로 프로젝트를 생성한다.



이 프로젝트에서는 UI Form을 사용할 것이므로 아래와 같이 [Generation form]에 체크 한다.



프로젝트 생성이 완료되면 CMakeLists.txt 파일에 Multimedia 모듈과 Network 모듈을 아래와 같이 추가한다.

```
cmake_minimum_required(VERSION 3.5)

project(02_AudioReceiver VERSION 0.1 LANGUAGES CXX)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 REQUIRED COMPONENTS
    Widgets
    Multimedia
    Network
)

set(PROJECT_SOURCES
```

```

main.cpp
widget.cpp
widget.h
widget.ui
)

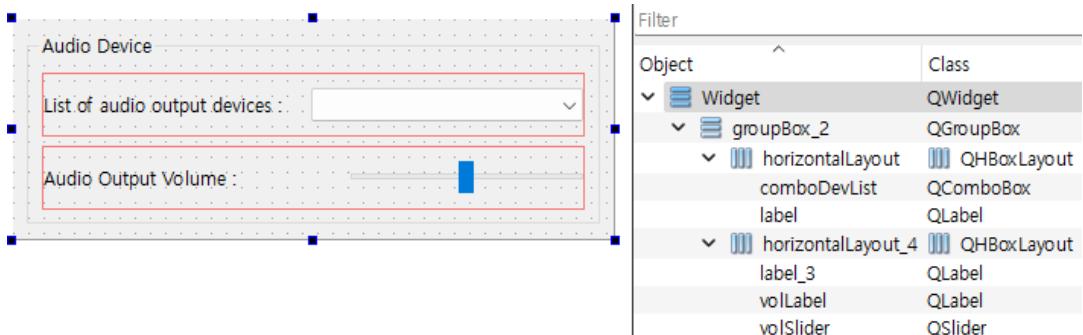
qt_add_executable(02_AudioReceiver
    ${PROJECT_SOURCES}
)

target_link_libraries(02_AudioReceiver PRIVATE
    Qt6::Widgets
    Qt6::Multimedia
    Qt6::Network
)

install(TARGETS 02_AudioReceiver
    BUNDLE DESTINATION .
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)

```

다음으로 widget.ui 파일을 열어서 아래와 같이 GUI 위젯을 배치한다.



다음으로 widget.h 헤더 파일을 아래와 같이 작성한다.

```

#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

```

```
#include <QtMultimedia>
#include <QtNetwork>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

    QMediaDevices      *m_devices = nullptr;
    QList<QAudioDevice> mDevList;
    QAudioDevice       mDeviceInfo;
    float              mOutputVolume;
    QAudioFormat        mFormat;
    QAudioSink          *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();
```

예수님은 당신을 사랑합니다.

```
};

#endif // WIDGET_H
```

다음으로 widget.cpp 소스코드 파일을 아래와 같이 작성한다.

```
#include "widget.h"
#include "./ui_widget.h"

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()
{
    m_devices = new QMediaDevices(this);
    mDevList = m_devices->audioOutputs();

    for(int i = 0; i < mDevList.size(); ++i)
        ui->comboDevList->addItem(mDevList.at(i).description());

    mDevInfo = mDevList.at(ui->comboDevList->currentIndex());

    mFormat.setSampleRate(8000);
    mFormat.setChannelCount(1);
    mFormat.setSampleFormat(QAudioFormat::Int16);
```

예수님은 당신을 사랑합니다.

```
mAudioOutput = new QAudioSink(mDeviceInfo, mFormat, this);

mOutput = mAudioOutput->start();

mOutputVolume = ui->volSlider->value() / 100.0;
ui->volLabel->setText(QString("%1 %")
                        .arg(ui->volSlider->value()));

mAudioOutput->setVolume(mOutputVolume);
}

void Widget::networkInitialize()
{
    mUdpSocket = new QUdpSocket(this);
    mUdpSocket->bind(QHostAddress::LocalHost, 15000);

    connect(mUdpSocket, SIGNAL(readyRead()),
            this,           SLOT(readUdpData()));
}

void Widget::devListIndexChanged(int index)
{
    mDeviceInfo = mDevList.at(index);

    if(mOutput != nullptr) {
        disconnect(mOutput, nullptr, this, nullptr);
        mOutput = nullptr;
    }

    mAudioOutput->stop();
    mAudioOutput->disconnect(this);
    delete mAudioOutput;

    mAudioOutput = new QAudioSink(mDeviceInfo, mFormat, this);
    mAudioOutput->setVolume(mOutputVolume);
```

```
mOutput = mAudioOutput->start();  
  
}  
  
void Widget::volSliderChanged(int val)  
{  
    mOutputVolume = val / 100.0;  
    mAudioOutput->setVolume(mOutputVolume);  
  
    ui->volLabel->setText(QString("%1 %").arg(val));  
}  
  
void Widget::readUdpData()  
{  
    while (mUdpSocket->hasPendingDatagrams())  
    {  
        QNetworkDatagram datagram;  
        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()  
{  
}
```

예수님은 당신을 사랑합니다.

audioInitialize() 멤버함수에서는 출력할 디바이스 목록을 얻어온다. 그리고 Combo box에 등록한다.

Combo box 상에 등록된 항목 중에서 Combo box에 현재 선택되어진 Device를 출력할 디바이스로 설정한다.

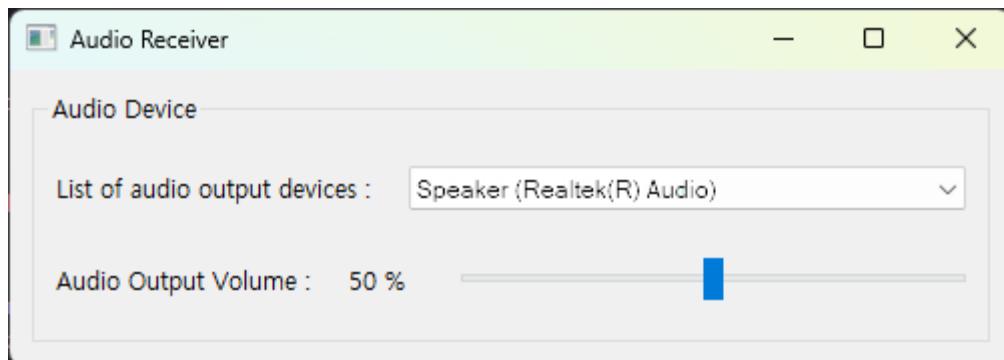
그리고 QAudioFormat 클래스는 출력할 디바이스의 Sample rate, Channel counts, Sample 당 Bytes 수를 설정한다.

다음으로 QAudioSink 클래스는 출력할 디바이스를 설정한다.

첫 번째 인자로 QAudioOutput 클래스의 오브젝트를 넘겨준다. 두 번째 인자로 QAudioFormat 클래스 오브젝트를 넘겨주면 된다.

마지막으로 네트워크로부터 읽어온 QByteArray 데이터를 Speaker로 출력하기 위해서는 QIODevice 클래스를 사용한다. 따라서 audioInitialize() 멤버 함수 마지막 부분에서 QIODevice 클래스의 오브젝트 Pointer를 QAudioSink 클래스로부터 얻어와 저장한다.

readUdpData() Slot 함수는 UDP 프로토콜에서 데이터를 수신 받으면 호출된다. 이 함수에서는 수신 받은 데이터를 QIODevice 클래스에서 제공하는 write() 멤버 함수에 첫 번째 인자로 넘겨준다. 그러면 Speaker를 통해 출력된다.



아래 그림에서 보는 것과 같이 어플리케이션을 실행 후 Microphone device로부터 수신 받은 데이터가 Speaker 잘 출력되는지 확인해 보도록 하자. 이 예제의 소스코드는 02_AudioReceiver 디렉토리를 참조하면 된다.

38.2. Video

Qt에서 제공하는 Video 관련 멀티미디어 기능을 살펴보기 전에, 동영상을 재생하기 위해서 필요한 Codec에 대해서 살펴보자.

동영상 파일은 압축된 상태이다. 동영상 파일을 재생하면 동영상을 재생하는 어플리케이션은 내부적으로 실시간 Decoding을 해 화면상에 Rendering하게 된다. 반대로 녹화 시 동영상을 Encoding(예를 들어 압축)하게 된다.

따라서 동영상을 Decoding하거나 Encoding하기 위해서는 Codec이 필요하다. Codec은 다양한 종류가 있고 이 시간에도 파일 압축률 더 높이기 위해서 개선된 Codec이 개발되고 있다.

Qt는 다양한 동영상 Codec을 라이선스 문제로 인해 Qt와 함께 제공하지 않는다. 이것은 다른 개발 Framework도 마찬가지이다.

만약 Qt로 구현한 동영상 재생 어플리케이션이 동영상 파일을 Decoding하지 못한다면 내 플랫폼(운영체제)에 동영상을 재생하기 위해 필요한 Codec 없기 때문이다.

동영상 Codec으로 무료로 제공해주는 여러 Codec이 있다. 여러가지 통합 Codec으로 K-Like Codec Pack (www.codecguide.com)을 무료로 사용할 수 있다.

이 중에서 Basic, Standard, Full 등을 제공하는데 여기서는 Full 버전을 다운로드 받아 설치한다.

다시 Qt로 돌아와서, 동영상을 재생한 화면을 GUI상에 표시하기 위해서 Multimedia GUI 위젯을 제공한다. 따라서 Multimedia GUI를 사용하기 위해서 프로젝트 파일 상에 아래와 같이 Multimedia 모듈과 MultimediaWidgets 모듈을 함께 사용해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS
    Widgets
    Multimedia
    MultimediaWidgets
)

target_link_libraries(00_VideoPlayer PRIVATE
    Qt6::Widgets
    Qt6::Multimedia
```

```
Qt6::MultimediaWidgets  
})
```

Qt에서 동영상을 재생하는 어플리케이션을 구현하기 위해서 QMediaPlayer 클래스를 사용해야 한다.

그리고 동영상을 재생하면 동영상 화면과 Sound 가 필요하다. 따라서 동영상 화면을 Rendering 하기 위해서 QMediaPlayer 클래스에 QVideoWidget 클래스 오브젝트를 설정해야 한다.

QMediaPlayer 클래스의 setVideoOutput() 멤버 함수를 이용해 QVideoWidget 클래스의 오브젝트를 연결할 수 있다.

그리고 동영상 재생 시 Sound를 재생하기 위해서 QAudioOutput 클래스를 사용하면 된다. Sound를 재생하기 위해서 QMediaPlayer 클래스에서 제공하는 setAudioOutput() 멤버 함수에 첫 번째 인자로 QAudioOutput 클래스 오브젝트를 넘겨주면 된다.

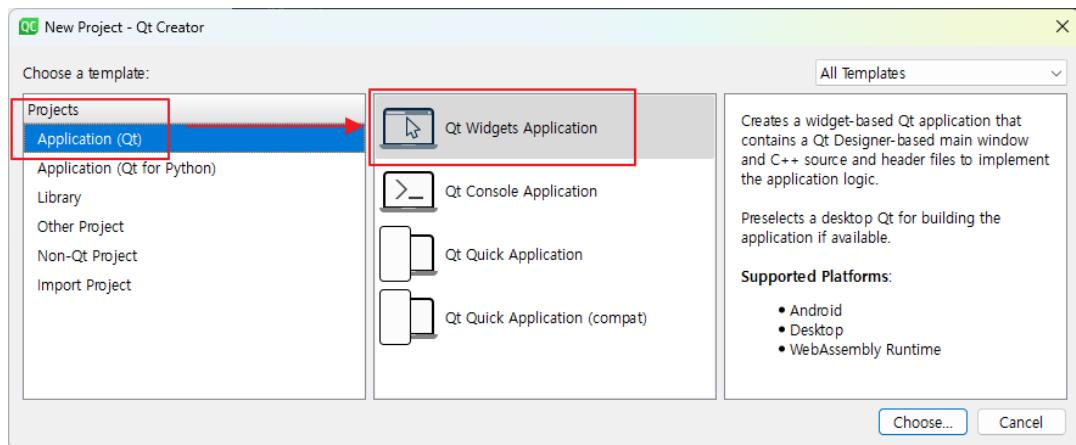
```
player = new QMediaPlayer;  
  
vWidget = new QVideoWidget();  
player->setVideoOutput(vWidget);  
  
audioOutput = new QAudioOutput;  
player->setAudioOutput(audioOutput);  
...  
audioOutput->setVolume(50);  
...  
player->setSource(QUrl::fromLocalFile("d:/sample.mp4"));  
player->play();
```

✓ Video Player 예제 구현

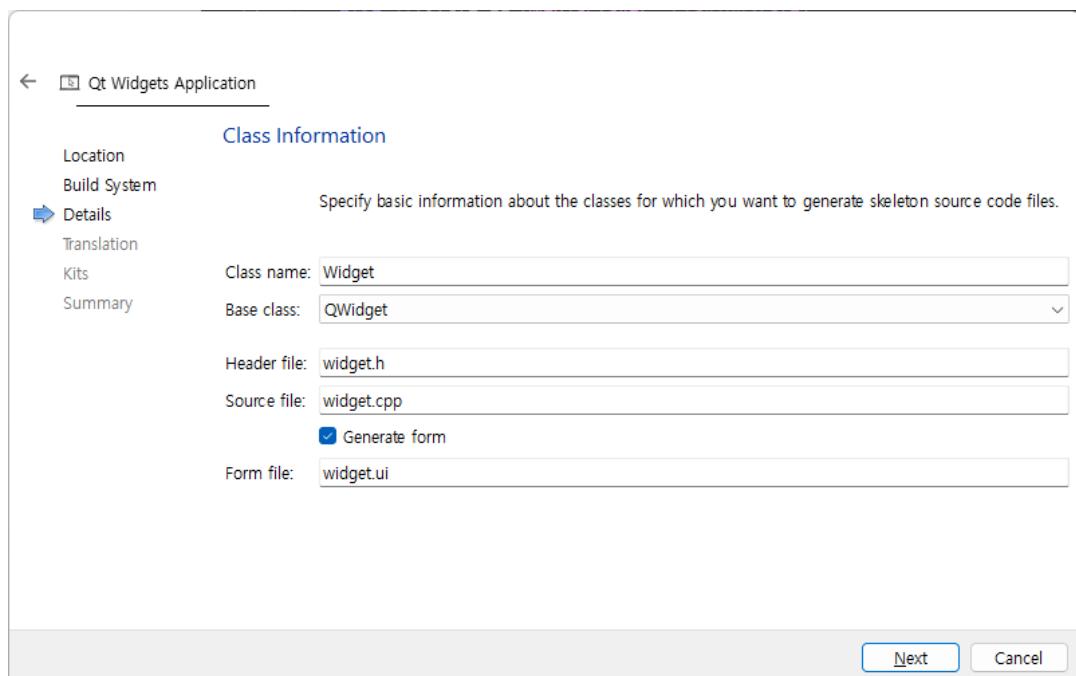
이번 예제는 QMediaPlayer 클래스를 이용해 동영상 파일을 재생하는 예제를 작성해 보도록 하자.

예수님은 당신을 사랑합니다.

프로젝트 생성 시 Qt Widget 기반으로 프로젝트를 생성한다.



이 프로젝트에서는 UI Form을 사용할 것이므로 아래와 같이 [Generation form]에 체크 한다.



프로젝트 생성이 완료되면 프로젝트 파일에 Multimedia 모듈과 MultimediaWidgets 모듈을 아래와 같이 추가한다.

```
cmake_minimum_required(VERSION 3.5)  
  
project(00_VideoPlayer VERSION 0.1 LANGUAGES CXX)
```

```
set(CMAKE_AUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 REQUIRED COMPONENTS
    Widgets
    Multimedia
    MultimediaWidgets
)

set(PROJECT_SOURCES
    main.cpp
    widget.cpp
    widget.h
    widget.ui
)

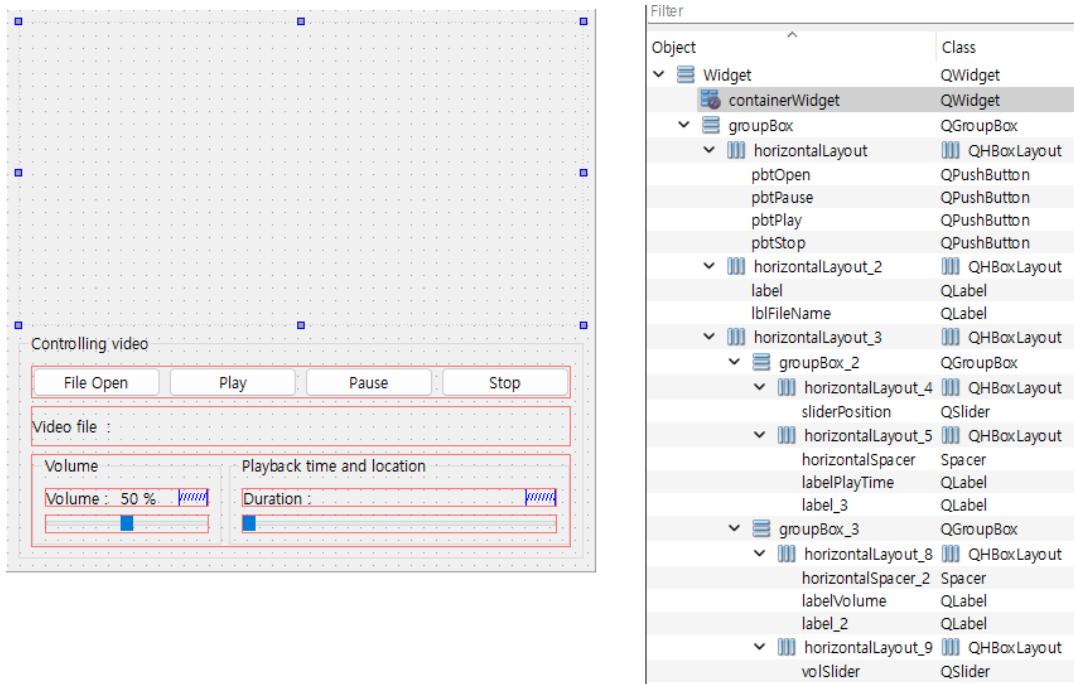
qt_add_executable(00_VideoPlayer
    ${PROJECT_SOURCES}
)

target_link_libraries(00_VideoPlayer PRIVATE
    Qt6::Widgets
    Qt6::Multimedia
    Qt6::MultimediaWidgets
)

install(TARGETS 00_VideoPlayer
    BUNDLE DESTINATION .
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

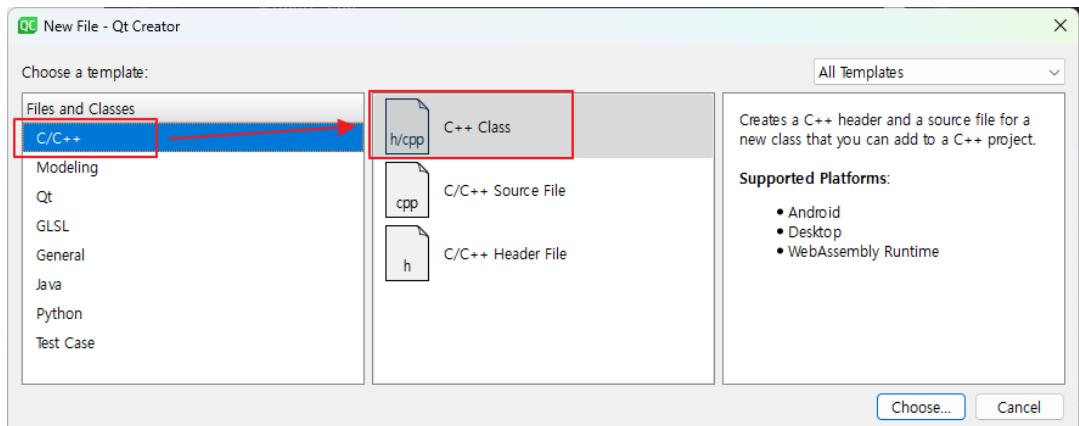
예수님은 당신을 사랑합니다.

다음으로 widget.ui 파일을 열어서 아래와 같이 GUI 위젯들을 배치한다.



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. 프로젝트상에 DisplayWidget 클래스를 아래와 같이 추가한다.



DisplayWidget 클래스를 추가하고 displaywidget.h 헤더 파일에 아래와 같이 작성한다.

```
#ifndef DISPLAYWIDGET_H  
#define DISPLAYWIDGET_H
```

```
#include <QVideoWidget>

class DisplayWidget : public QVideoWidget
{
    Q_OBJECT
public:
    DisplayWidget();

protected:
    void keyPressEvent(QKeyEvent *event) override;
    void mouseDoubleClickEvent(QMouseEvent *event) override;
    void mousePressEvent(QMouseEvent *event) override;
};

#endif // DISPLAYWIDGET_H
```

mouseDoubleClickEvent() Virtual 함수는 마우스 더블 클릭 시 호출된다. 마우스 더블 클릭 시 동영상 랜더링 위젯이 전체 화면으로 전환된다.

다음은 displaywidget.cpp 소스코드 파일을 아래와 같이 작성한다.

```
#include "displaywidget.h"
#include <QMouseEvent>

DisplayWidget::DisplayWidget()
{
    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.

다음으로 widget.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef WIDGET_H  
#define WIDGET_H  
  
#include <QWidget>  
#include <QMediaPlayer>  
#include <QAudioOutput>  
#include "displaywidget.h"  
  
QT_BEGIN_NAMESPACE  
namespace Ui { class Widget; }  
QT_END_NAMESPACE  
  
class Widget : public QWidget  
{  
    Q_OBJECT  
  
public:  
    Widget(QWidget *parent = nullptr);  
    ~Widget();
```

```

private:
    Ui::Widget *ui;

    QString      m_fName;
    QMediaPlayer *m_player;
    DisplayWidget *m_displayWidget;
    QAudioOutput *m_audioOutput;
    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() Slot 함수는 [File Open] 버튼을 클릭하면 호출되는 Slot 함수이다. 이 함수에서는 QFileDialog 클래스를 이용해 파일 선택 디아ログ 중 동영상 파일을 선택할 수 있다.

onPlayBtn() 함수는 동영상 파일을 재생한다. onPauseBtn() 은 동영상 재생일 일시정지 기능을 제공하고 onStopBtn() 은 동영상을 중지 하는 기능을 제공한다. 다음으로 widget.cpp 소스 코드를 아래와 같이 작성한다.

```

#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_audioOutput = new QAudioOutput;
m_player->setAudioOutput(m_audioOutput);
m_audioOutput->setVolume(0.5);

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_audioOutput->setVolume(val / 100.0);
}
```

```
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->setSource(QUrl::fromLocalFile(m_fName));
        ui->sliderPosition->setEnabled(true);
        m_player->play();
    }
}

void Widget::onPauseBtn()
{
    int state = m_player->playbackState();
    if(state == QMediaPlayer::PausedState)
        m_player->play();
    else if(state == QMediaPlayer::PlayingState)
        m_player->pause();
}

void Widget::onStopBtn()
{
    int state = m_player->playbackState();
    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;
}
```

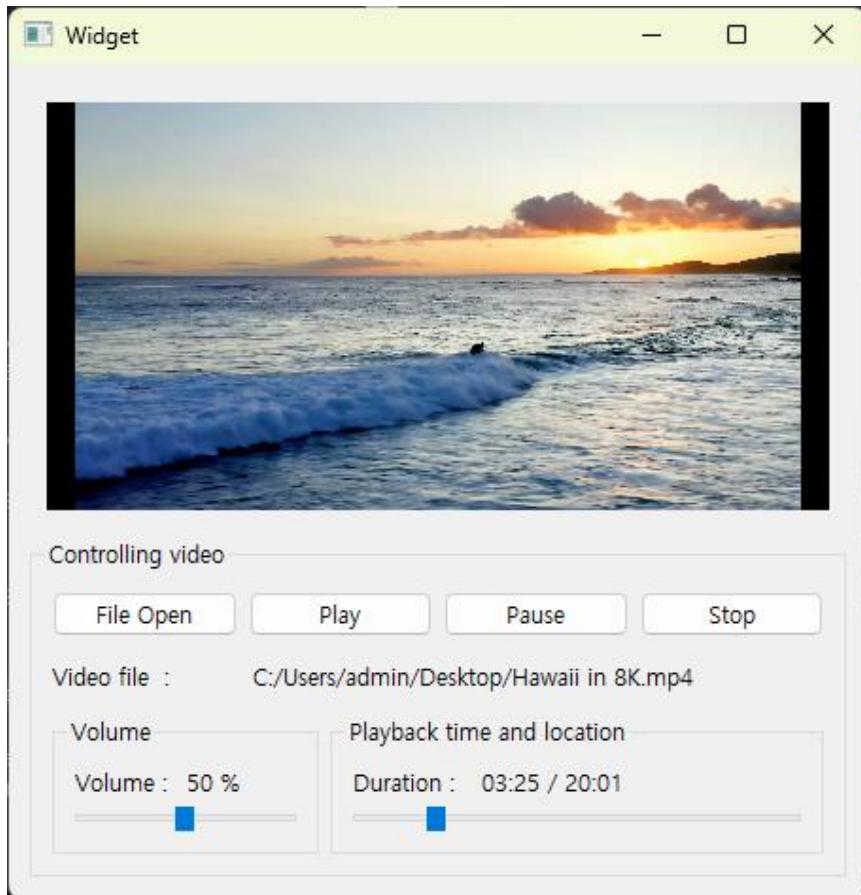
클래스 생성자에서 DisplayWidget 을 GUI 상에 배치하고 QMediaPlayer 클래스의 setVideoOutput() 함수의 첫 번째 인자로 DisplayWidget 클래스의 오브젝트를 넘겨주면 QMediaPlayer 클래스가 동영상을 재생한다.

예수님은 당신을 사랑합니다.

QMediaPlayer가 동영상을 재생하면 Decoding 된 결과를 DisplayWidget 클래스 위젯에 출력한다.

sliderValueChange() 는 GUI 상에서 볼륨을 조절할 수 있다. durationChanged() 함수는 동영상 파일의 재생 위치가 임의로 변경되면 호출된다.

positionChanged() 는 현재 재생 되고 있는 위치를 알려주는 기능을 제공한다. 이 Slot 함수에서는 전체 시간을 “시:분:초”로 변경한 후 GUI 위젯에 출력한다.



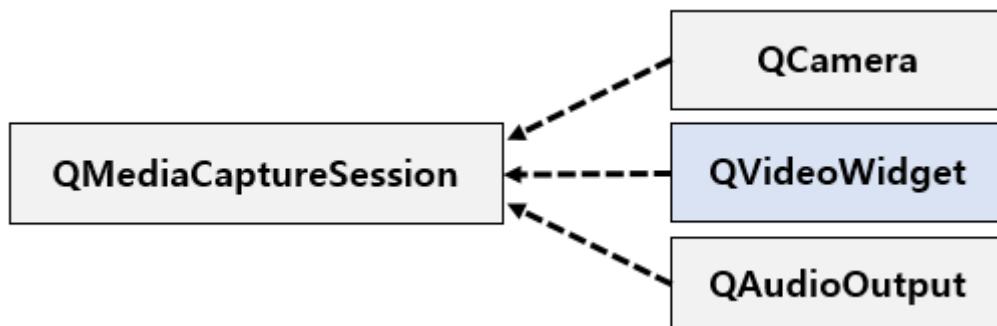
이 예제의 소스코드는 00_VideoPlayer 디렉토리를 참조하면 된다.

38.3. Camera

Qt Multimedia 모듈에서 Camera device를 이용해 응용 어플리케이션 개발할 수 있는 API를 제공한다.

Camera device를 제어하기 위한 클래스로 QMediaCaptureSession 클래스를 제공한다. 이 클래스는 카메라 영상을 추출 하기 위해서 QCamera 클래스를 이용한다. 그리고 추출한 영상을 GUI 상에 표시하기 위해서 QVideoWidget 클래스를 이용한다.

그리고 Camera device 는 Microphone device 가 함께 제공된다. Camera device 에 포함된 Microphone device 로부터 입력되는 데이터를 처리하기 위해서 QAudioInput 클래스 오브젝트를 QMediaCaptureSession 에 연결해야 한다.



마지막으로 Camera device를 시작하기 위해서 QCamera 클래스에서 제공하는 start() 멤버 함수를 사용하면 된다.

```
QMediaCaptureSession          m_captureSession;
QScopedPointer<QAudioInput>   m_audioInput;
QScopedPointer<QCamera>        mCamera;
QVideoWidget                  *mVideoWidget;

...
m_audioInput.reset(new QAudioInput);
mCamera.reset(new QCamera(0));
mVideoWidget = new QVideoWidget();

m_captureSession.setAudioInput(m_audioInput.get());
...
m_captureSession.setCamera(mCamera.data());
m_captureSession.setVideoOutput(mVideoWidget);
...
```

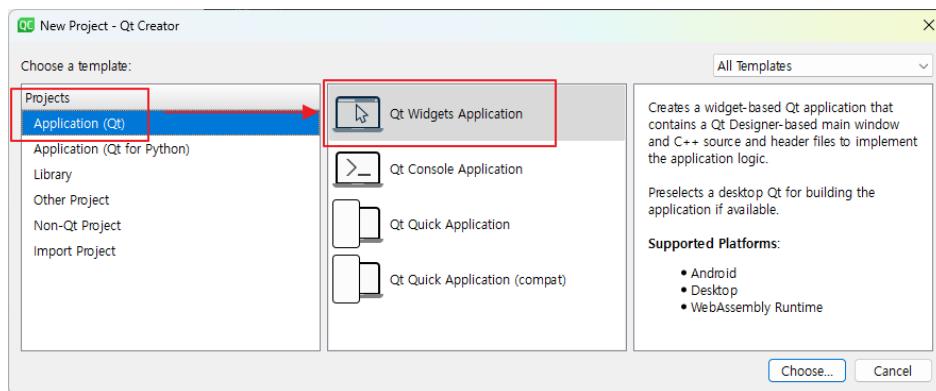
```
mCamera->start();  
...
```

다음은 실제 Camera Device로부터 영상을 다루는 예제를 구현해 보도록 하자.

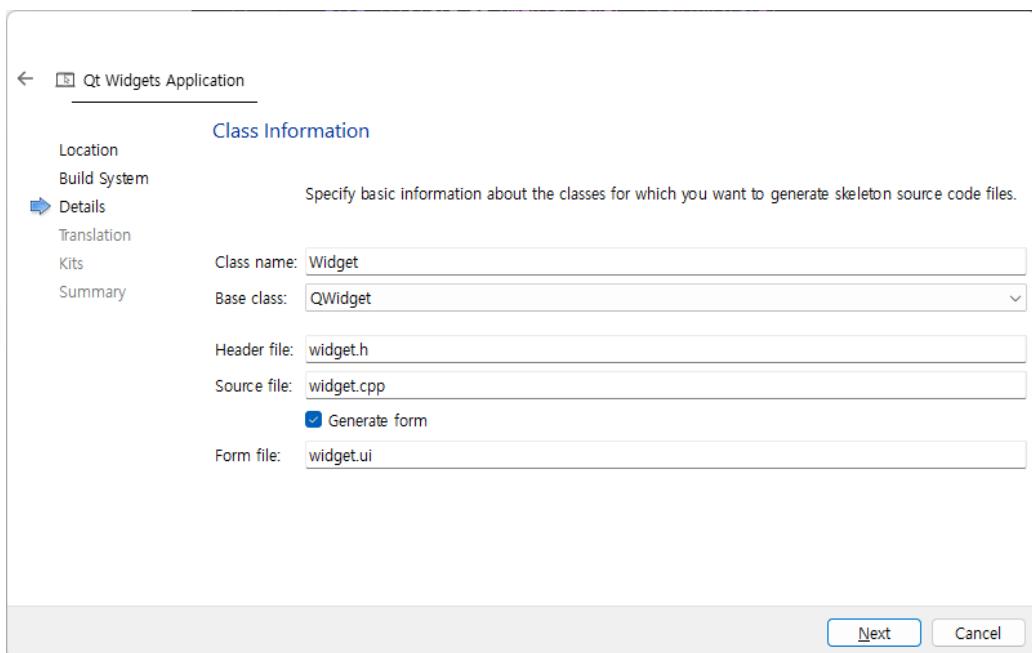
- ✓ Camera device를 이용한 예제 구현

이번 예제는 USB Camera device로부터 영상을 GUI상에 표시하는 예제를 구현해 보도록 하자.

프로젝트 생성 시 Qt Widget 기반으로 프로젝트를 생성한다.



이 프로젝트에서는 UI Form을 사용할 것이므로 아래와 같이 [Generation form]에 체크 한다.



예수님은 당신을 사랑합니다.

프로젝트 생성이 완료되면 프로젝트 파일에 Multimedia 모듈과 MultimediaWidgets 모듈을 아래와 같이 추가한다.

```
cmake_minimum_required(VERSION 3.5)

project(00_CameraCapture VERSION 0.1 LANGUAGES CXX)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 REQUIRED COMPONENTS
    Widgets
    Multimedia
    MultimediaWidgets
)

set(PROJECT_SOURCES
    main.cpp
    widget.cpp
    widget.h
    widget.ui
)

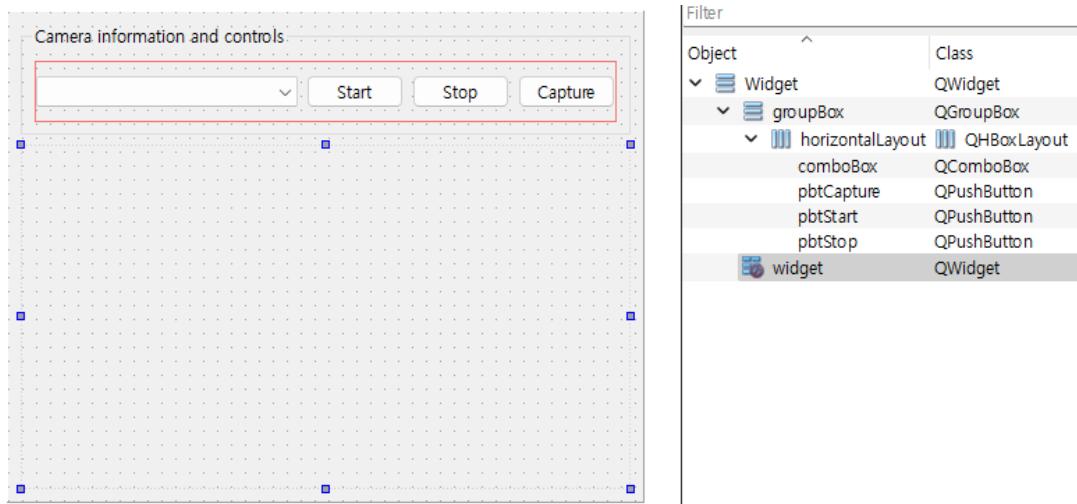
qt_add_executable(00_CameraCapture
    ${PROJECT_SOURCES}
)

target_link_libraries(00_CameraCapture PRIVATE
    Qt6::Widgets
    Qt6::Multimedia
    Qt6::MultimediaWidgets
)

install(TARGETS 00_CameraCapture
    BUNDLE DESTINATION .
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

예수님은 당신을 사랑합니다.

다음으로 widget.ui 파일을 열어서 아래와 같이 GUI 위젯들을 배치한다.



다음으로 widget.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QVideoWidget>
#include <QCamera>
#include <QImageCapture>
#include <QMediaCaptureSession>
#include <QAudioInput>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
```

예수님은 당신을 사랑합니다.

```
QList<QCameraDevice>      mCamList;

QVideoWidget                 *m_videoWidget;

QScopedPointer<QImageCapture>   m_imageCapture;
QMediaCaptureSession           m_captureSession;
QScopedPointer<QCamera>        mCamera;
QScopedPointer<QAudioInput>    m_audioInput;

void cameraDevicesSearch();
void viewCaptureImgDialog(const QImage &img);

private slots:
void onStartBtn();
void onStopBtn();
void onCaptureBtn();
void camError();

void imageCaptured(int requestId, const QImage &img);
};

#endif // WIDGET_H
```

다음으로 widget.cpp 소스코드 파일을 열어서 아래와 같이 작성한다.

```
#include "widget.h"
#include "./ui_widget.h"
#include <QMediaDevices>
#include <QMMessageBox>
#include <QLabel>
#include <QHBoxLayout>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pbtStart, SIGNAL(clicked()), this, SLOT(onStartBtn()));
    connect(ui->pbtStop,  SIGNAL(clicked()), this, SLOT(onStopBtn()));
    connect(ui->pbtCapture, SIGNAL(clicked()), this, SLOT(onCaptureBtn()));

    cameraDevicesSearch();

    ui->pbtStart->setEnabled(true);
    ui->pbtStop->setEnabled(false);
    ui->pbtCapture->setEnabled(false);
```

```
m_videoWidget = new QVideoWidget();
QHBoxLayout *hLay = new QHBoxLayout;
hLay->addWidget(m_videoWidget);
ui->widget->setLayout(hLay);
}

void Widget::cameraDevicesSearch()
{
    ui->comboBox->clear();
    mCamList = QMediaDevices::videoInputs();
    foreach(const QCameraDevice &camDev, mCamList)
        ui->comboBox->addItem(camDev.description());
}

void Widget::viewCaptureImgDialog(const QImage &img)
{
    QDialog viewDialog(this, Qt::Dialog);
    viewDialog.setWindowTitle("Capture Image");

    QLabel doneLabel("");
    doneLabel.setPixmap(QPixmap::fromImage(img));

    QHBoxLayout lay;
    lay.addWidget(&doneLabel);

    viewDialog.setLayout(&lay);
    viewDialog.resize(img.size());
    viewDialog.exec();
}

void Widget::onStartBtn()
{
    m_audioInput.reset(new QAudioInput);
    m_captureSession.setAudioInput(m_audioInput.get());

    int currCamIndex = ui->comboBox->currentIndex();

    mCamera.reset(new QCamera(mCamList.at(currCamIndex)));
    connect(mCamera.data(), &QCamera::errorOccurred,
            this,             &Widget::camError);
}
```

```
m_captureSession.setCamera(mCamera.data());
m_captureSession.setVideoOutput(m_videoWidget);

if (!m_imageCapture) {
    m_imageCapture.reset(new QImageCapture);
    m_captureSession.setImageCapture(m_imageCapture.get());

    connect(m_imageCapture.get(), &QImageCapture::imageCaptured,
            this,                      &Widget::imageCaptured);
}

mCamera->start();

ui->pbtStart->setEnabled(false);
ui->pbtStop->setEnabled(true);
ui->pbtCapture->setEnabled(true);
}

void Widget::onStopBtn()
{
    mCamera->stop();
    m_videoWidget->clearMask();

    ui->pbtStart->setEnabled(true);
    ui->pbtStop->setEnabled(false);
    ui->pbtCapture->setEnabled(false);
}

void Widget::onCaptureBtn()
{
    m_imageCapture->capture();
}

void Widget::camError()
{
    if ( mCamera->error() != QCamera::.NoError ) {
        QMessageBox::warning(this,
                            tr("Camera Error"),
                            mCamera->errorString());
    }
}

void Widget::imageCaptured(int requestId, const QImage &img)
```

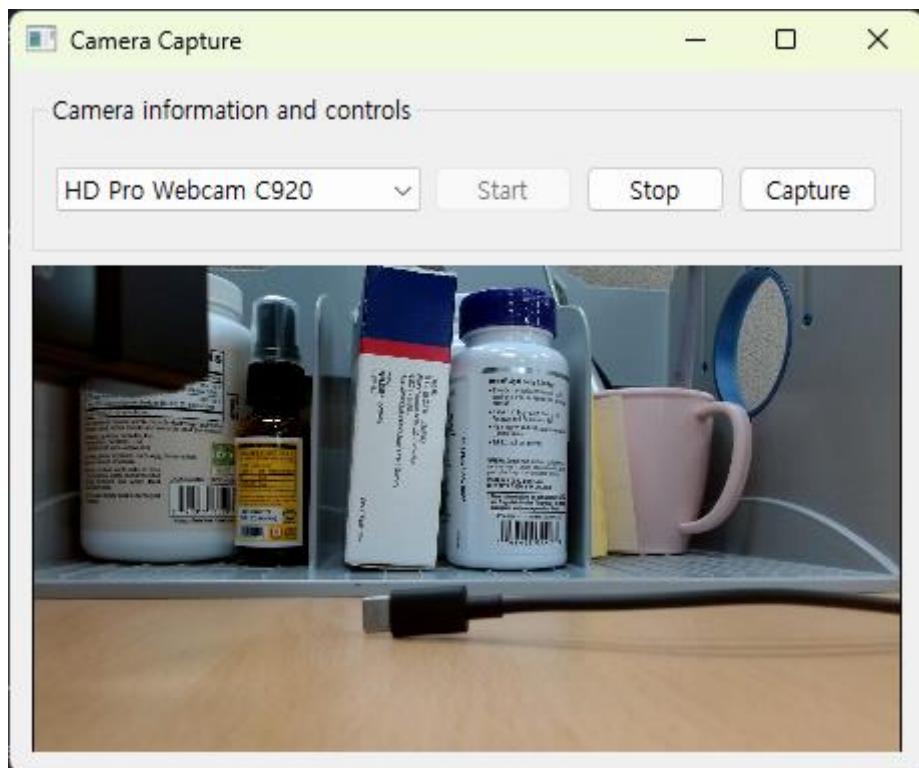
예수님은 당신을 사랑합니다.

```
{  
    QImage scaledImgImage = img.scaled(m_videoWidget->size(),  
                                         Qt::KeepAspectRatio,  
                                         Qt::SmoothTransformation);  
    viewCaptureImgDialog(scaledImgImage);  
}  
  
Widget::~Widget()  
{  
    delete ui;  
}
```

cameraDevicesSearch() 함수는 시스템에서 검색된 모든 카메라 장치를 QComboBox 클래스의 위젯에 등록한다.

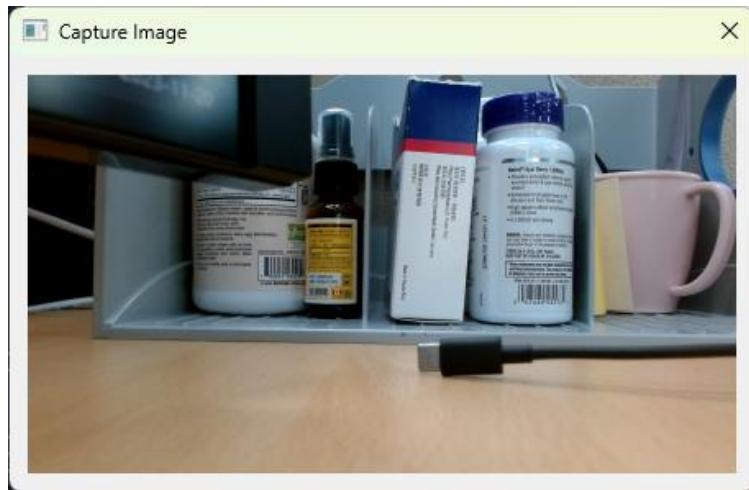
onStartBtn() 함수는 [Start] 버튼 클릭 시 호출된다. onStopBtn() 함수는 [Stop] 버튼 클릭 히 호출된다.

viewCaptureImgDialog() 함수는 [Capture] 버튼을 클릭하면 호출된다. 이 함수는 Camera device 가 활성화된 상태에서 이 함수가 호출되면 Dialog 로 Capture 한 이미지를 표시한다.



예수님은 당신을 사랑합니다.

[Capture 버튼을 클릭하면 아래와 같이 Dialog 상에 Capture 한 이미지가 표시된다. 이 예제의 소스코드는 00_CameraCapture 디렉토리를 참조하면 된다.



39. Serial Communication

Qt에서 제공하는 SerialPort 모듈은 RS-232 pinout 의 제어 시그널을 통해 데이터를 송/수신할 수 있다. Serial 통신은 물리적으로 떨어져 있는 컴퓨터 또는 Embedded device 간의 통신으로 많이 사용된다.

Qt SerialPort 모듈의 가장 큰 장점은 MS Windows, Linux 그리고 MacOS 플랫폼에 모두 호환된다. 예를 들어 MS Windows에서 Qt SerialPort 모듈에서 제공하는 클래스들을 이용해 구현한 소스코드가 Linux와 MacOS에서도 호환 된다는 것이다.

Qt에서 제공하는 QSerialPort 클래스와 같은 시리얼을 이용하기 위해서 프로젝트파일에 다음과 같이 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS SerialPort)
target_link_libraries(mytarget PRIVATE Qt6::SerialPort)
```

만약 qmake를 사용한다면 아래와 같이 추가해주면 된다.

```
QT += serialport
```

QSerialPort에서도 Signal과 Slot을 사용한다. 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);
```

예수님은 당신을 사랑합니다.

```
m_serial->setStopBits(stopBits);  
m_serial->setFlowControl(flowControl);  
  
m_serial->open(QIODevice::ReadWrite);
```

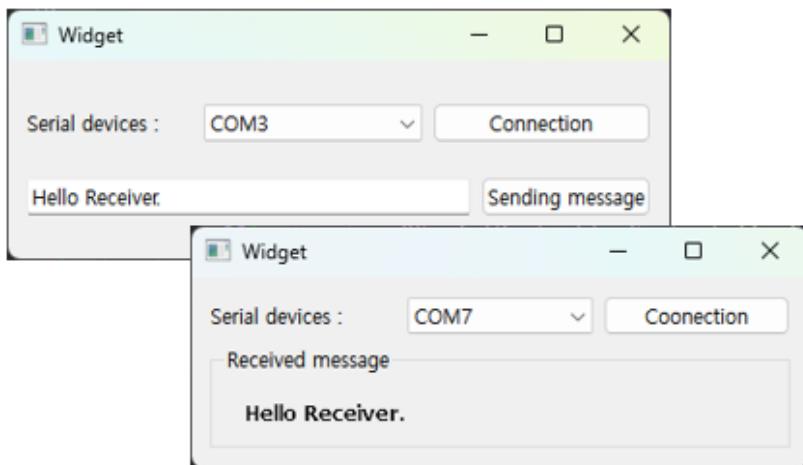
QSerialPort 클래스는 위의 예제 소스코드에서 보는 것과 같이 에러가 발생하면 errorOccurred() Signal 이 발생한다.

그리고 readyRead() 시그널을 이용해 상대방이 전송하는 데이터를 수신 받을 수 있다. 시리얼은 상대방 측과 시리얼로 연결하기 위해서는 동일하게 설정해야 한다.

setPortName() 멤버 함수는 사용할 시리얼 디바이스 이름(예를 들어 Linux 인 경우 /dev/USB0와 같은 이름을 사용)을 입력한다. setBaudRate() 멤버 함수는 속도를 지정할 수 있다. setDataBits() 멤버 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



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 device 를 선택하고 [Connection] 버튼을 클릭한다. 그리고 우측의 Receiver 예제도 마찬가지로 GUI상에서

예수님은 당신을 사랑합니다.

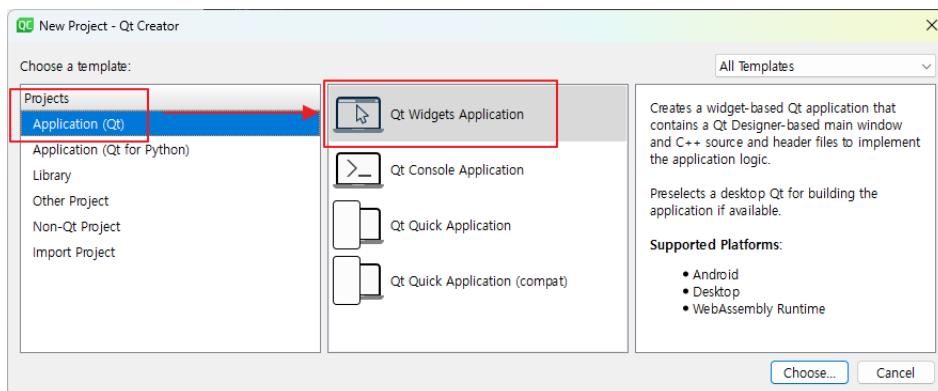
사용할 device 를 선택하고 [Connection] 버튼을 클릭하면 두 어플리케이션 간 데이터를 송/수신 할 준비가 모두 완료된다.

첫 번째 Sender 어플리케이션의 하단의 QLineEdit 위젯에 전송할 메시지를 입력 후 [Sending message] 버튼을 클릭하면 위의 그림에서 보는 것과 같이 Receiver 에게 메시지를 송신한다.

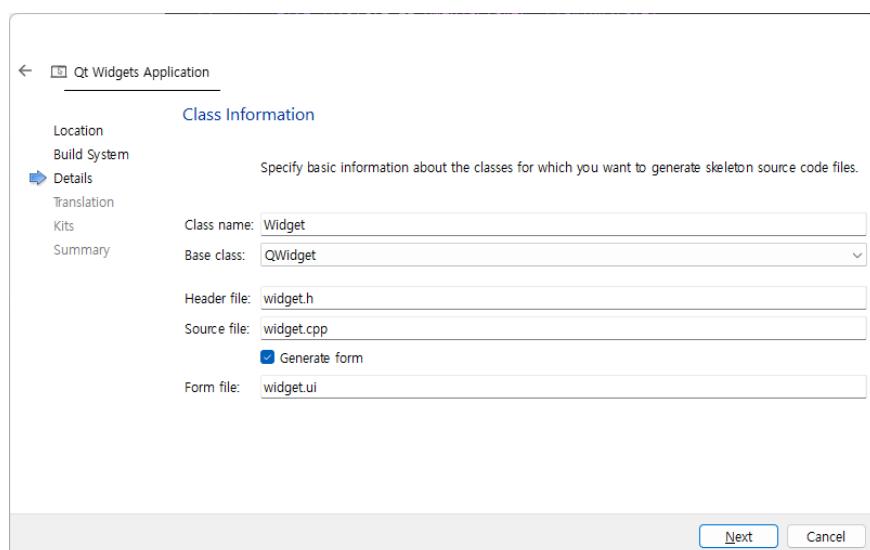
Receiver 는 Sender 가 보낸 메시지를 GUI 상에 출력한다. 두 개의 어플리케이션 중 Sender 예제의 소스코드를 먼저 살펴본 후 Receiver 예제를 살펴보도록 하자.

✓ Sender 예제

프로젝트 생성 시 Qt Widget 기반으로 프로젝트를 생성한다.



이 프로젝트에서는 UI Form을 사용할 것이므로 아래와 같이 [Generation form]에 체크 한다.

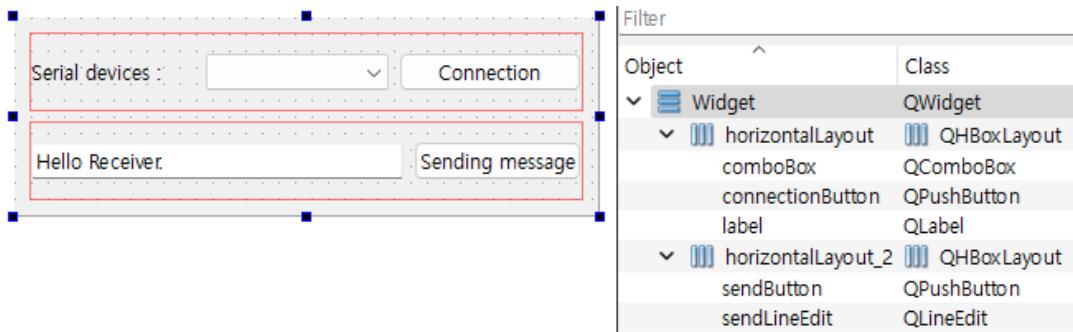


예수님은 당신을 사랑합니다.

프로젝트 생성이 완료되면 프로젝트 파일에 SerialPort 모듈을 아래와 같이 추가한다.

```
cmake_minimum_required(VERSION 3.5)
...
find_package(Qt6 REQUIRED COMPONENTS Widgets SerialPort)
...
target_link_libraries(00_Sender PRIVATE
    Qt6::Widgets
    Qt6::SerialPort
)
...
```

다음으로 widget.ui 파일을 열어서 아래와 같이 GUI 위젯을 배치한다.



다음으로 widget.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QSerialPort>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT
```

```
public:  
    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
```

다음으로 widget.cpp 소스코드 파일을 열어서 아래와 같이 작성한다.

```
#include "widget.h"  
#include "./ui_widget.h"  
#include <QSerialPortInfo>  
  
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() << "Error : " << m_serial->errorString();
    }
}
```

예수님은 당신을 사랑합니다.

```
}

void Widget::sendButton()
{
    if(!m_serial->isOpen())
        return;

    QString sendMsg = ui->sendLineEdit->text().trimmed();
    QByteArray msg = sendMsg.toLocal8Bit();

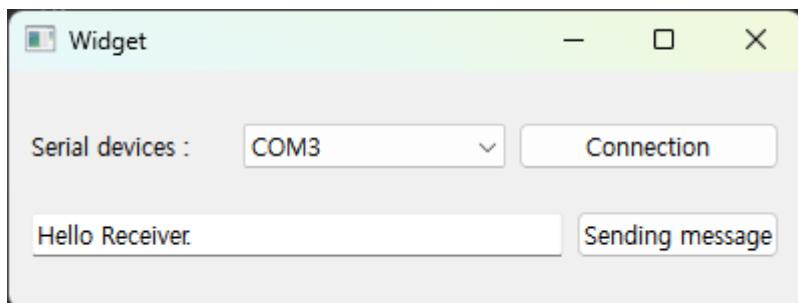
    m_serial->write(msg);
}

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

클래스 생성자에서 QSerialPortInfo::availablePorts() 멤버 함수는 활용 가능한 모든 시리얼 디바이스의 정보를 얻어오는 기능을 제공한다. 활용 가능한 시리얼 디바이스 정보를 얻어와 GUI 상에서 Combo box에 등록한다.

connectButton() Slot 함수가 호출되면 클래스 생성자에서 저장한 시리얼 연결을 위한 옵션 값을 설정하고 Serial device를 연결한다.

sendButton() Slot 함수에서는 QLineEdit에서 입력한 문자 변수 값을 QByteArray로 변환 후 QSerialPort 클래스의 write() 멤버 함수를 이용해 메시지를 전송한다.

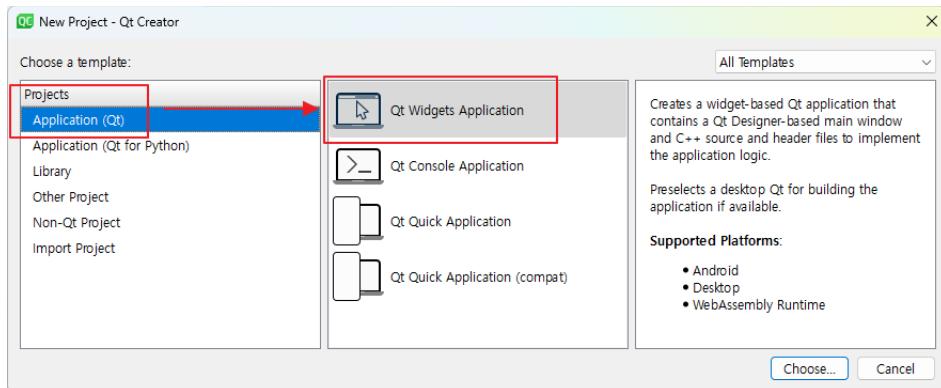


이 예제의 소스코드는 00_Sender 예제를 참조하면 된다.

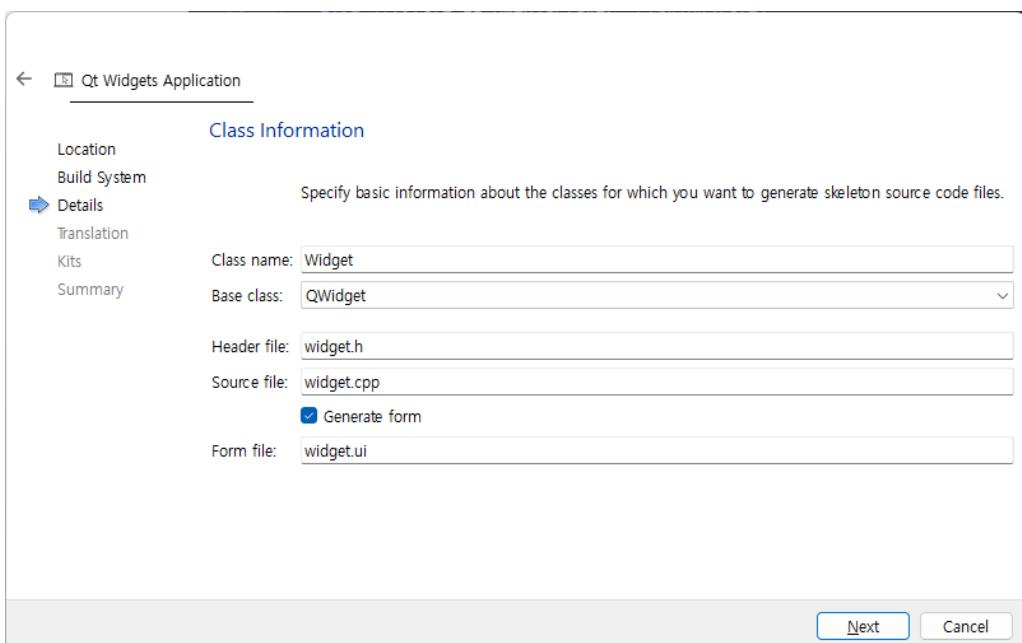
예수님은 당신을 사랑합니다.

✓ Receiver 예제

프로젝트 생성 시 Qt Widget 기반으로 프로젝트를 생성한다.



이 프로젝트에서는 UI Form을 사용할 것이므로 아래와 같이 [Generation form]에 체크 한다.

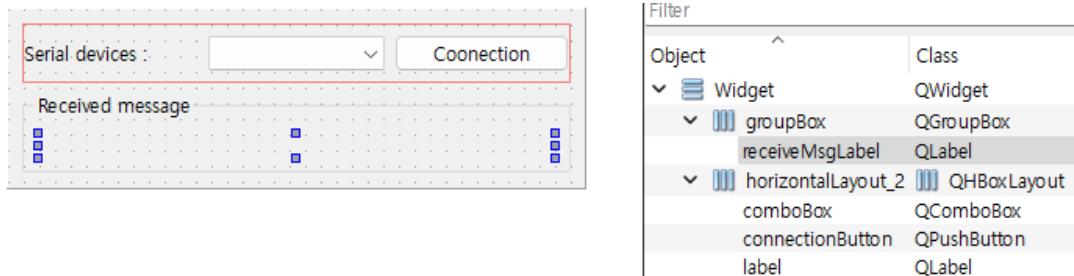


프로젝트 생성이 완료되면 프로젝트 파일에 SerialPort 모듈을 아래와 같이 추가한다.

```
cmake_minimum_required(VERSION 3.5)
...
find_package(Qt6 REQUIRED COMPONENTS Widgets SerialPort)
...
target_link_libraries(01_Receiver PRIVATE
    Qt6::Widgets
```

```
Qt6::SerialPort  
)  
...
```

다음으로 widget.ui 파일을 열어서 아래와 같이 GUI 위젯들을 배치한다.



다음으로 widget.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QSerialPort>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

    struct SerialSettings {
        QString portName;
```

예수님은 당신을 사랑합니다.

```
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 errorOccurred(QSerialPort::SerialPortError err);
};

#endif // WIDGET_H
```

다음으로 widget.cpp 소스코드 파일을 아래와 같이 작성한다.

```
#include "widget.h"
#include "./ui_widget.h"
#include <QSerialPortInfo>

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)
```

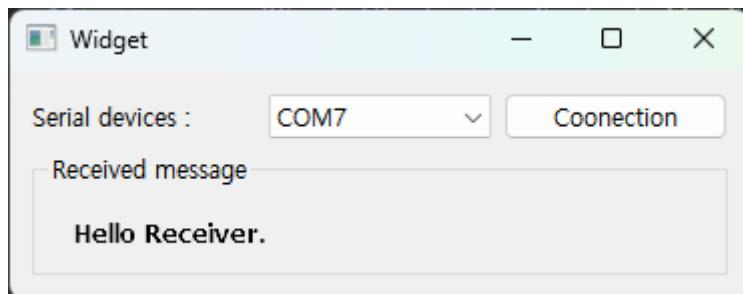
```
ui->comboBox->addItem(info.portName());  
  
m_serial = new QSerialPort(this);  
connect(m_serial, SIGNAL(errorOccurred(QSerialPort::SerialPortError)),  
        this, SLOT(errorOccurred(QSerialPort::SerialPortError)));  
connect(m_serial, SIGNAL(readyRead()),  
        this, SLOT(readData()));  
}  
  
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() << "Error :" << m_serial->errorString();  
    }  
}  
  
void Widget::errorOccurred(QSerialPort::SerialPortError err)  
{  
    if (err == QSerialPort::ResourceError) {  
        qDebug() << Q_FUNC_INFO  
              << "Critical Error : "  
              << m_serial->errorString();  
    }  
}
```

예수님은 당신을 사랑합니다.

```
m_serial->close();  
}  
}  
  
void Widget::readData()  
{  
    const QByteArray data = m_serial->readAll();  
    ui->receiveMsgLabel->setText(data);  
}  
  
Widget::~Widget()  
{  
    delete ui;  
}
```

[Connection] 버튼을 클릭하면 connectButton() Slot 함수가 호출된다. 이 함수에서는 Serial device를 Open 하고 Sender 예제에서 송신한 데이터를 수신한다.

Sender로부터 메시지를 수신하면 readyRead() Signal 이 호출된다. 이 시그널이 호출되면 readData() Slot 함수가 호출된다. 이 Slot 함수는 수신한 데이터를 QByteArray에 저장한다. 그리고 GUI 상에 표시한다.



이 예제의 소스코드는 01_Receiver 디렉토리를 참조하면 된다.

40. Qt Positioning

Qt Position 모듈은 Latitude(위도)와 Longitude(경도)를 이용해 거리, 방위각, 속도 등을 쉽게 계산하기 위한 기능을 제공한다. 또한 Latitude 와 Longitude 를 표기하는 방법으로 DD, DMS 등의 방식이 있다. 이러한 표기식 변환도 쉽게 할 수 있다.

예를 들어 Map과 연동해 QGeoCoordinate 클래스에서 제공하는 distanceTo() 멤버 함수를 이용해 두 지점 간의 거리를 계산할 수 있다. 또한 azimuthTo() 멤버 함수를 이용해 방위각을 계산할 수 있다.

따라서 Qt Positioning 모듈은 Map 과 관련한 응용 어플리케이션을 구현할 때 유용하게 사용할 수 있다.

프로젝트 파일로 CMake를 사용하는 경우, 아래와 같이 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS Positioning)
target_link_libraries(mytarget PRIVATE Qt6::Positioning)
```

QGeoCoordinate 클래스와 QGeoPositionInfo 클래스를 이용해 다음과 같이 Date/Time, Latitude 그리고 Longitude를 표시할 수 있다.

```
double latitude = -27.572321;
double longitude = 153.090718;

QString timeStr = QString("2009-08-24T22:24:37");
QDateTime timestamp = QDateTime::fromString(timeStr, Qt::ISODate);

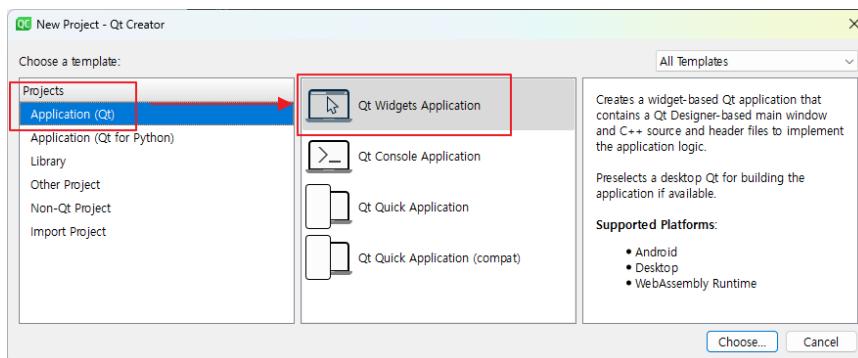
QGeoCoordinate coordinate(latitude, longitude);
QGeoPositionInfo info(coordinate, timestamp);

QString posDateTime;
posDateTime = QString("Position Date/time = %1")
               .arg(info.timestamp().toString());
QString posGeo;
posGeo = QString("Coordinate = %1")
           .arg(info.coordinate().toString());
```

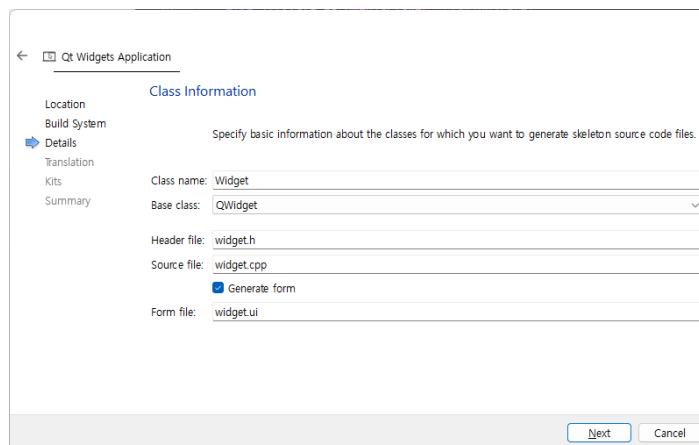
```
qDebug() << posDateTime;  
qDebug() << posGeo;  
  
/*  
[ Result ]  
Position Date/time = Mon Aug 24 22:24:37 2009  
Coordinate = 27 ° 34' 20.4" S, 153 ° 5' 26.6" E  
*/
```

● Distance 예제 구현

이번 예제는 두 도시간의 직선거리를 구하는 예제를 구현해 보도록 하자. 프로젝트 생성 시 Qt Widget 기반으로 프로젝트를 생성한다.



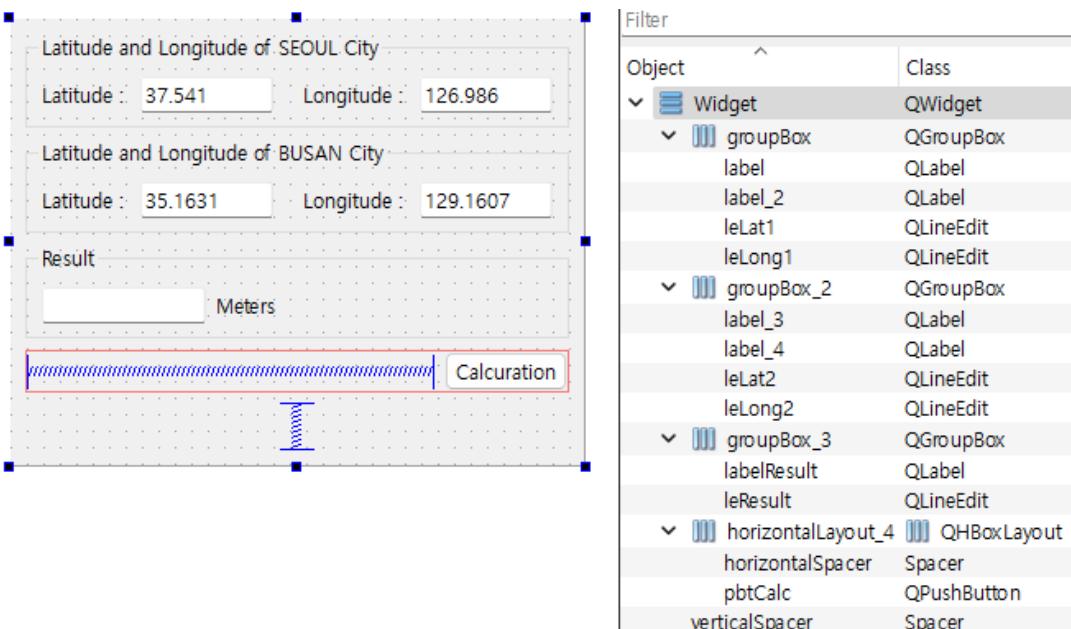
이 프로젝트에서는 UI Form을 사용할 것이므로 아래와 같이 [Generation form]에 체크 한다.



프로젝트 생성이 완료되면 프로젝트 파일에 Positioning 모듈을 아래와 같이 추가한다.

```
cmake_minimum_required(VERSION 3.5)
...
find_package(Qt6 REQUIRED COMPONENTS Widgets Positioning)
...
target_link_libraries(00_DistanceExample PRIVATE
    Qt6::Widgets
    Qt6::Positioning
)
...
...
```

다음으로 widget.ui 파일을 열어서 아래와 같이 GUI 위젯들을 배치한다.



다음으로 widget.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE
```

예수님은 당신을 사랑합니다.

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

private:
    Ui::Widget *ui;

private slots:
    void slotCalculation();
};

#endif // WIDGET_H
```

다음으로 widget.cpp 소스코드 파일을 열어 아래와 같이 작성한다.

```
#include "widget.h"
#include "./ui_widget.h"
#include <QDateTime>
#include <QGeoCoordinate>
#include <QGeoPositionInfo>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pbtCalc, &QPushButton::clicked,
            this,           &Widget::slotCalculation);
}

void Widget::slotCalculation()
{
    double lat1      = ui->leLat1->text().toDouble();
    double long1     = ui->leLong1->text().toDouble();
    double lat2      = ui->leLat2->text().toDouble();
    double long2     = ui->leLong2->text().toDouble();
```

```
QGeoCoordinate coord1(lat1, long1);
QGeoCoordinate coord2(lat2, long2);

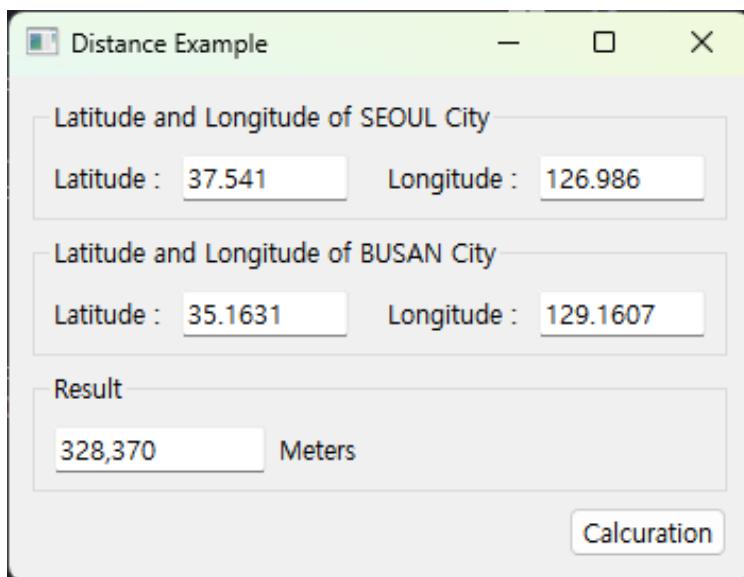
int distance_meter = coord1.distanceTo(coord2);
QString distanceStr = QString("%L1").arg(distance_meter);

ui->leResult->setText(distanceStr);
}

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

GUI 상에서 [Calculation] 버튼을 클릭하면 slotCalculation() Slot 함수가 호출된다. 이 함수에서 SEOUL City 와 BUSAN City 간의 거리를 Meter 로 계산한다.

두 도시간의 거리는 QGeoCoordinate 클래스에서 제공하는 distanceTo() 멤버 함수를 사용하면 두 도시간의 거리를 계산할 수 있다. 여기서 두 도시간의 거리는 직선 거리이다.



이 예제의 소스코드는 00_DistanceExample 디렉토리를 참조하면 된다.

41. Qt PDF

Qt에서 제공하는 PDF 모듈은 GUI 상에 PDF를 Rendering 하기 위한 기능을 제공한다. 단순히 PDF 문서를 표시하는 것 이외에도 ZOOM In/Out, Page back, Page Forward, Page Preview 등 PDF Rendering 에 필요한 다양한 기능을 제공한다.

Qt PDF 모듈을 사용하기 위해서는 CMake 의 프로젝트 파일에 아래와 같이 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS Pdf)
target_link_libraries(mytarget Qt6::Pdf)
```

GUI 상에 PDF 문서를 표시할 수 있는 위젯 클래스를 사용하기 위해서는 아래와 같이 CMake 프로젝트 파일에 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS PdfWidgets)
target_link_libraries(mytarget Qt6::PdfWidgets)
```

만약 qmake를 사용한다면 프로젝트 파일에 아래와 같이 추가해야 한다.

```
QT += pdf
```

주의해야 할 사항으로 현재 Qt PDF 모듈은 MS Windows 플랫폼에서 지원하지 않는다. 현재를 기준으로 Qt 6.5.x 버전에서는 지원하지 않지만 향후 상위 버전에서는 지원할 수 있다.

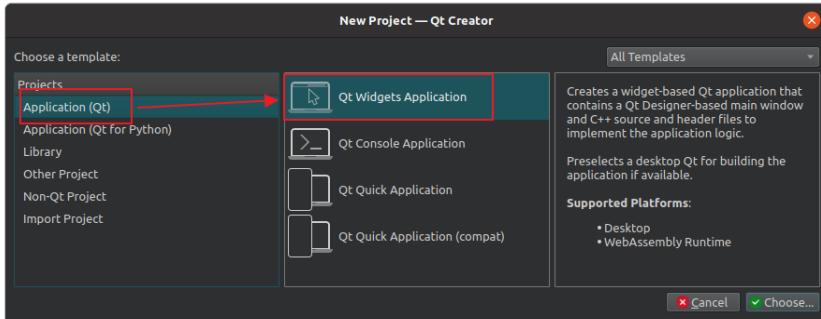
Qt PDF를 사용하는 방법은 매우 쉽게 사용할 수 있는 API를 제공한다. 따라서 간단하게 PDF를 GUI상에 Rendering 하는 예제를 통해서 사용하는 방법에 대해서 다루어 보도록 하자.

- PDF Viewer 예제 구현

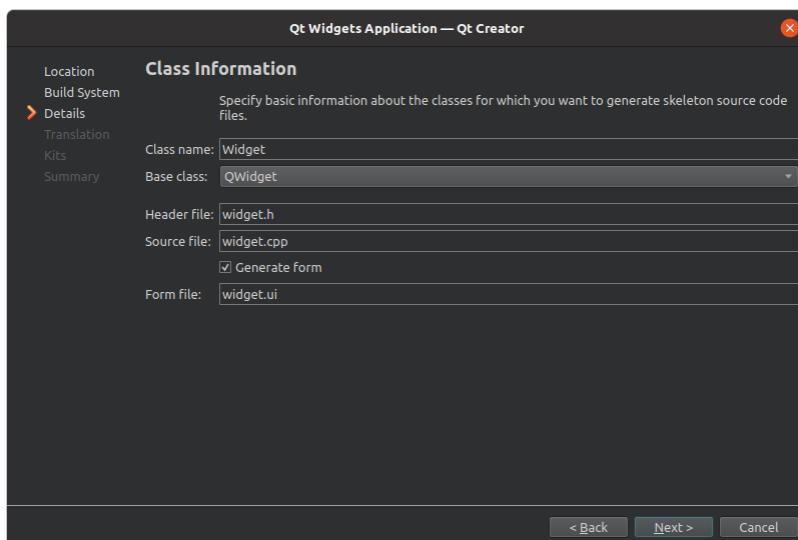
MS Windows 플랫폼(Qt 6.5.x)에서는 지원하지 않으므로 Linux에서 예제를 다루어 보도록 하겠다.

프로젝트 생성 시, Qt Widget 기반의 어플리케이션을 작성한다.

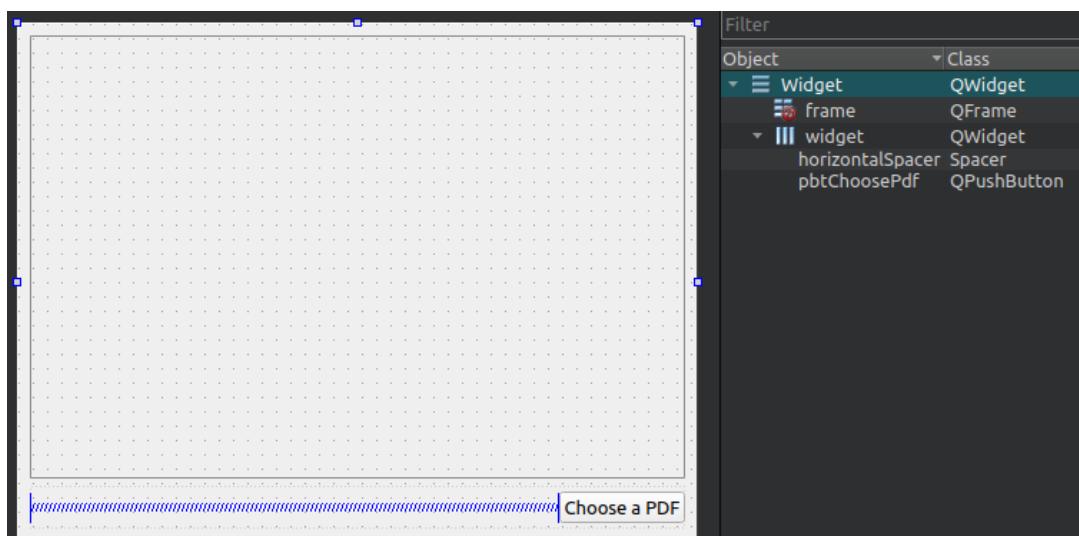
예수님은 당신을 사랑합니다.



이 프로젝트에서는 UI Form을 사용할 것이므로 아래와 같이 [Generation form]에 체크 한다.



다음으로 widget.ui 파일을 열어서 아래와 같이 GUI 위젯들을 배치한다.



예수님은 당신을 사랑합니다.

다음으로 widget.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QPdfView>
#include <QPdfDocument>
#include <QFileDialog>
#include <QPdfPageNavigator>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
    QFileDialog *m_fileDialog = nullptr;

    QPdfView     *m_pdfView;
    QPdfDocument *m_document;

    void slot_pbtChoosePDF();
};

#endif // WIDGET_H
```

다음으로 widget.cpp 소스코드 파일을 아래와 같이 작성한다.

```
#include "widget.h"
```

```
#include "./ui_widget.h"
#include <QHBoxLayout>
#include <QStandardPaths>
#include <QDebug>

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

    m_pdfView = new QPdfView;
    m_document = new QPdfDocument(this);

    m_pdfView->setDocument(m_document);

    QHBoxLayout *hLay = new QHBoxLayout();
    hLay->addWidget(m_pdfView);

    ui->frame->setLayout(hLay);

    connect(ui->pbtChoosePdf, &QPushButton::clicked,
            this,           &Widget::slot_pbtChoosePDF);
}

void Widget::slot_pbtChoosePDF()
{
    qDebug() << Q_FUNC_INFO;

    if (m_fileDialog == nullptr) {
        m_fileDialog = new QFileDialog(this, tr("Choose a PDF"),
                                      QStandardPaths::writableLocation(
                                          QStandardPaths::DocumentsLocation));

        m_fileDialog->setAcceptMode(QFileDialog::AcceptOpen);
        m_fileDialog->setMimeTypeFilters({"application/pdf"});
    }
}
```

```
}

if (_m_fileDialog->exec() == QDialog::Accepted)
{
    const QUrl toOpen = _m_fileDialog->selectedUrls().constFirst();
    _m_document->load(toOpen.toLocalFile());

    const auto docTitle =
        _m_document->metaData(QPdfDocument::MetaDataField::Title)
            .toString();

    setWindowTitle(!docTitle.isEmpty() ? docTitle : "PDF Viewer");

    _m_pdfView->setPageMode(QPdfView::PageMode::MultiPage);
}

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

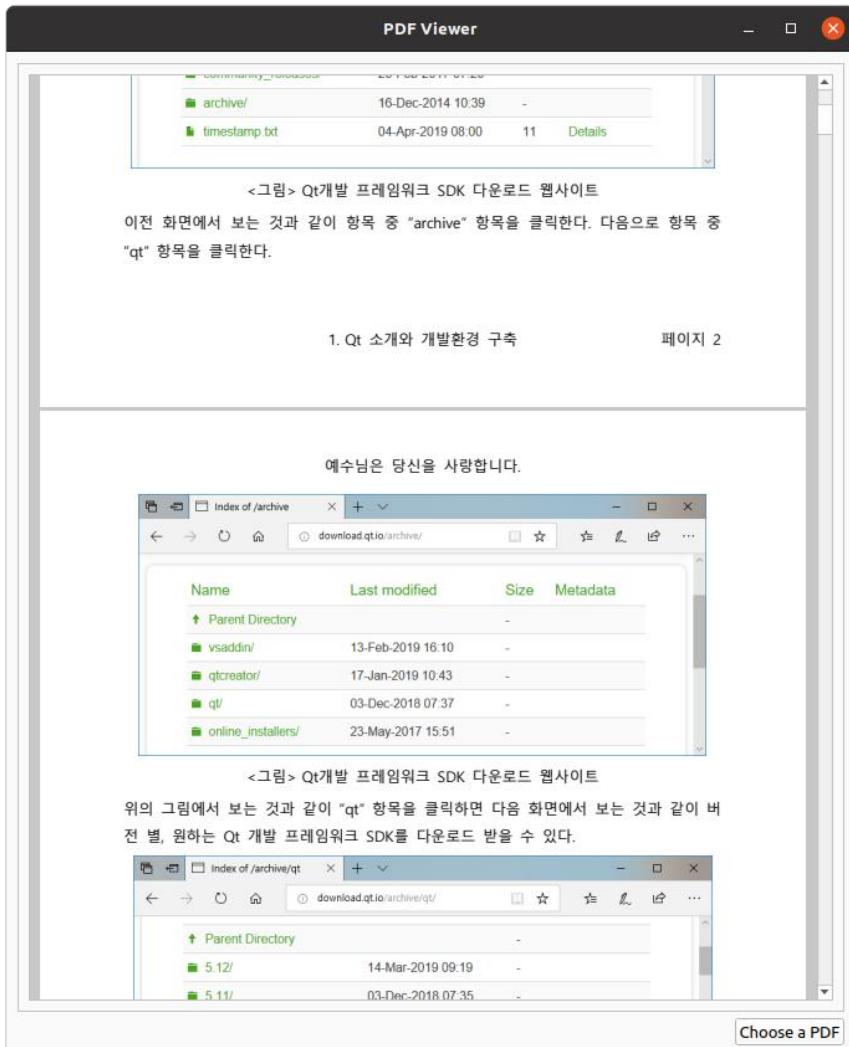
생성자 함수에서는 QPdfView 클래스 오브젝트를 생성한다. 이 클래스는 GUI상에 PDF를 Rendering 하는 기능을 제공한다.

그리고 QPdfDocument 클래스는 PDF 문서의 각 페이지를 로딩하는 기능을 제공한다. 따라서 QPdfDocument 클래스의 오브젝트를 QPdfView 와 연결해야 한다.

QPdfView 클래스와 연결하기 위해서 QPdfView 클래스에서 제공하는 setDocument() 멤버 함수를 사용하면 된다.

GUI 상에서 [Choose a PDF] 버튼을 클릭하면 slot_pbtChoosePDF() Slot 함수가 호출된다. 이 함수가 호출되면 File Dialog 가 로딩된다. 이 Dialog 에서 PDF 파일을 선택한다. 그러면 QPdfView 위젯 클래스에 PDF 문서가 로딩된다.

예수님은 당신을 사랑합니다.



이 예제 소스코드는 00_PDFViewer 디렉토리를 참조하면 된다.

42. Qt Printer Support

Qt에서 제공하는 Print Support 모듈은 연결된 프린터로 인쇄하거나 네트워크를 통해 원격 프린터로 인쇄할 수 있다.

또한 Qt Print Support 모듈은 PDF 파일로 생성할 수 있다. (참고: Qt Print Support 모듈은 현재 iOS에서는 지원하지 않는다.)

Qt Printer Support 모듈을 사용하기 위해서는 CMake 를 사용한다면 프로젝트 파일에 아래와 같이 모듈을 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS PrintSupport)
target_link_libraries(mytarget PRIVATE Qt6::PrintSupport)
```

만약 qmake를 사용한다면 프로젝트 파일에 아래와 같이 추가하면 된다.

```
QT += printsupport
```

Qt Print Support 모듈에서 제공되는 클래스 중 QPainter 클래스는 Qt GUI 위젯 또는 QPainter 영역을 프린트 할 수 있다.

```
QPrinter printer;
...
QPainter painter;
painter.begin(&printer);

...
myWidget->render(&painter);
...
```

QTextEdit 하는 GUI Widget 상에 표시된 텍스트를 프린트 하기 위해서 아래와 같이 사용할 수 있다.

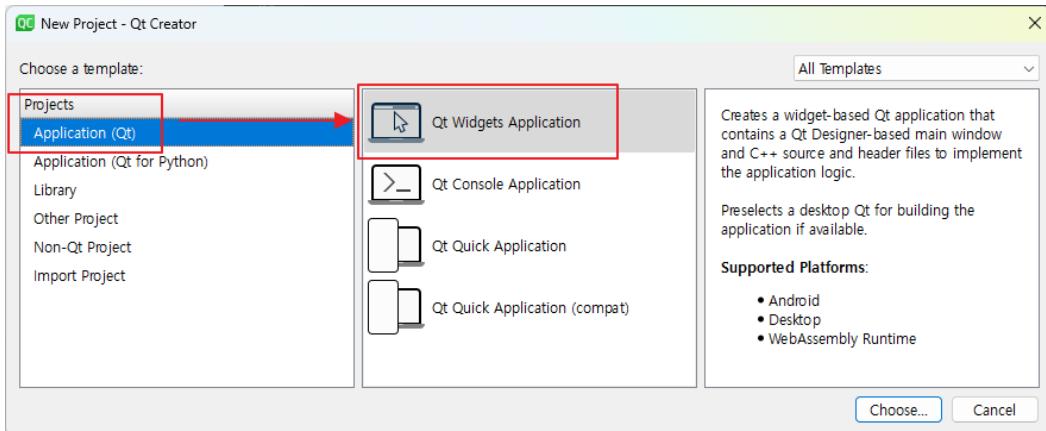
```
QTextEdit *textEdit = new QTextEdit();
...
textEdit.setText("Hello world.");
QPrinter printer(QPrinter::HighResolution);
```

```
TextEdit->print(&printer);
```

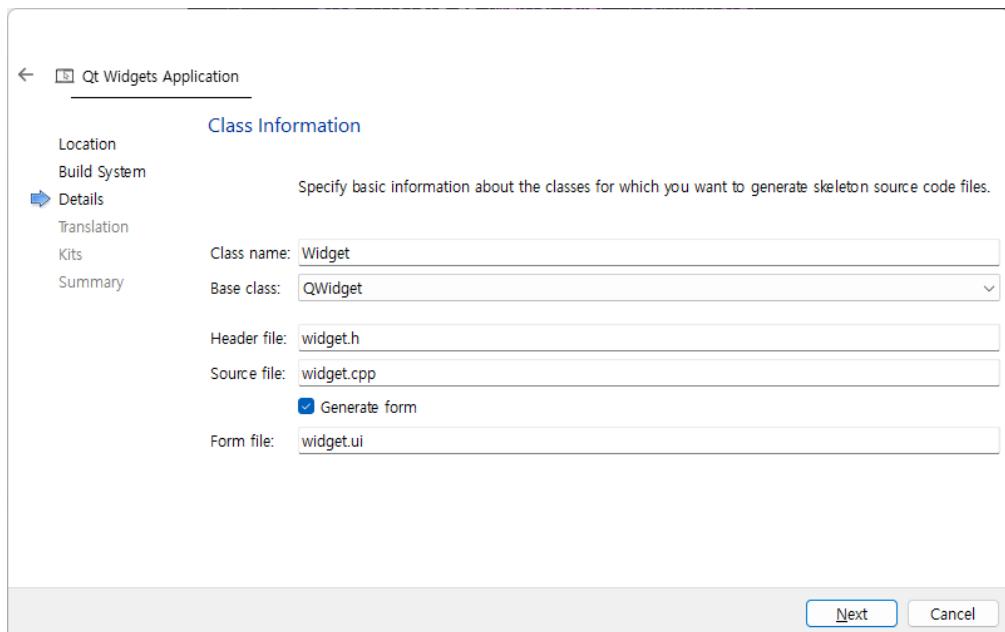
● 프린트 예제 구현

이번에는 간단한 예제를 통해서 GUI 위젯인 QTextEdit 상의 Text를 컴퓨터와 연결된 프린터로 프린트하는 기능을 구현해 보도록 하자.

프로젝트 생성 시 Qt Widget 기반의 프로젝트를 생성한다.



이 프로젝트에서는 UI Form을 사용할 것이므로 아래와 같이 [Generation form]에 체크 한다.



프로젝트 생성이 완료되면 CMakeLists.txt 파일에 아래와 같이 Printer Support 모듈을

예수님은 당신을 사랑합니다.

추가한다.

```
cmake_minimum_required(VERSION 3.5)

project(00_Printer_Example VERSION 0.1 LANGUAGES CXX)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 REQUIRED COMPONENTS Widgets PrintSupport)

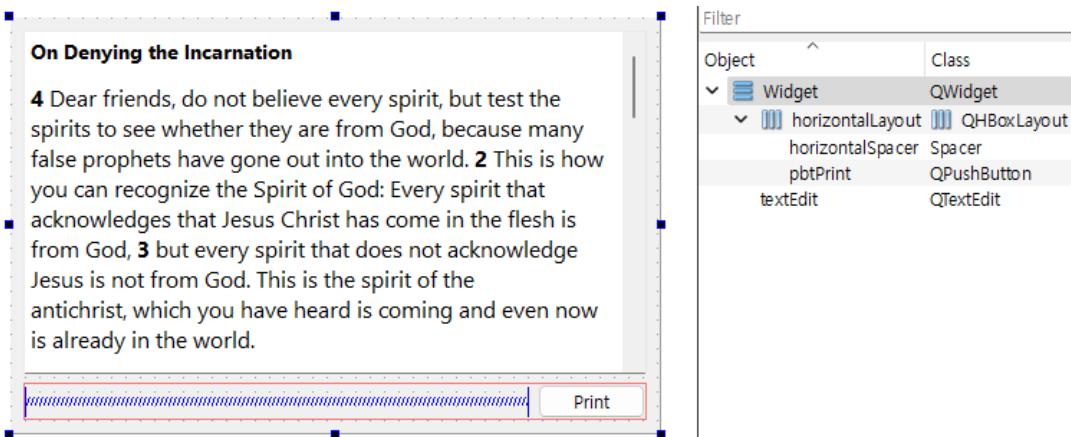
set(PROJECT_SOURCES
    main.cpp
    widget.cpp
    widget.h
    widget.ui
)

qt_add_executable(00_Printer_Example
    ${PROJECT_SOURCES}
)

target_link_libraries(00_Printer_Example PRIVATE
    Qt6::Widgets
    Qt6::PrintSupport
)

install(TARGETS 00_Printer_Example
    BUNDLE DESTINATION .
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

다음으로 widget.ui 파일을 열어서 아래와 같이 GUI 위젯들을 배치한다.



QTextEdit 위젯에는 어떤 내용을 넣어도 상관없다. 여기서는 프린트를 할 것으로 위와 같이 입력하였다. 따라서 원하는 텍스트를 입력해도 무방하다.

위와 같이 GUI 위젯들을 배치하고 widget.h 헤더 파일을 아래와 같이 작성한다.

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

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT
public:
    Widget(QWidget *parent = nullptr);
    ~Widget();
private:
    Ui::Widget *ui;
    void slot_pbtPrint();
};

#endif // WIDGET_H
```

예수님은 당신을 사랑합니다.

다음으로 widget.cpp 소스코드 파일을 열어서 아래와 같이 작성한다.

```
#include "widget.h"
#include "./ui_widget.h"
#include <QDebug>
#include <QFileDialog>
#include <QPrintDialog>

Widget::Widget(QWidget *parent)
    : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pbtPrint, &QPushButton::clicked,
            this,           &Widget::slot_pbtPrint);
}

void Widget::slot_pbtPrint()
{
    QPrinter printer(QPrinter::HighResolution);

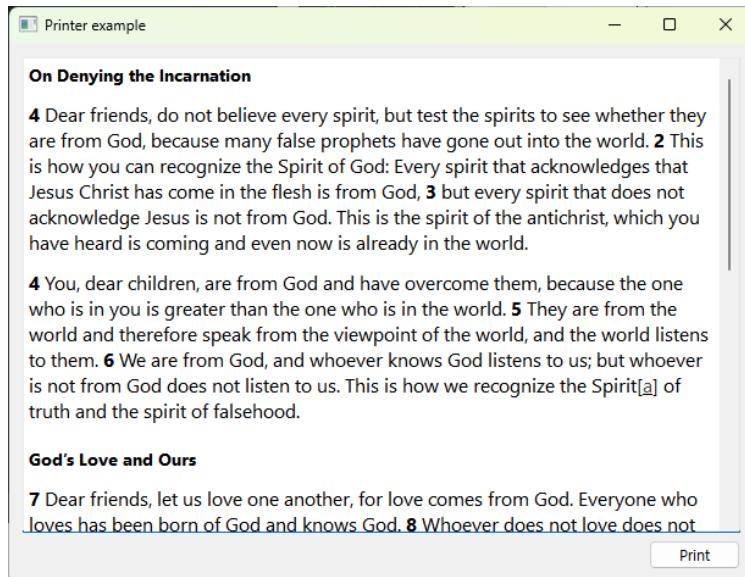
    QPrintDialog dialog(&printer, this);
    dialog.setWindowTitle(tr("Print Document"));

    if (dialog.exec() == QDialog::Accepted) {
        ui->textEdit->print(&printer);
    }
}

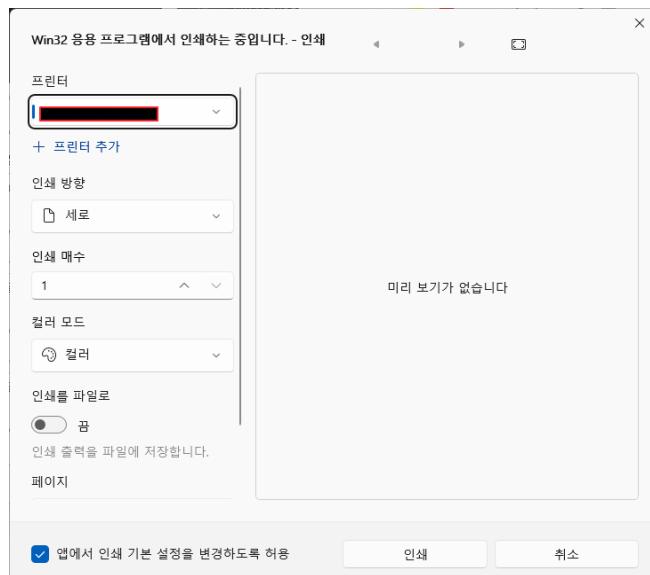
Widget::~Widget()
{
    delete ui;
}
```

위와 같이 작성하였다면 빌드 후 실행한 다음 QTextEdit 의 내용을 프린트로 출력해 보도록 하자.

예수님은 당신을 사랑합니다.



출력은 GUI 상에서 [Print] 버튼을 누르면 된다. [Print] 버튼을 클릭하면 아래 같이 Printダイ얼로그가 로딩된다.



위의 예제 소스코드는 00_Printer_Example 디렉토리를 참조하면 된다.

43. Qt Pprotobuf

Qt에서 제공하는 Protobuf는 Google에서 제공하는 Protocol Buffers(또는 Protobuf)를 쉽게 사용할 수 있도록 제공하는 모듈이다.

Protocol Buffers는 구조화된 데이터를 Serialized를 쉽게 할 수 있다. 예를 들어 두개의 프로그램 상에서 데이터를 송수신 할 때 아래와 같은 Structure가 있다고 가정해 보자.

```
struct EmployeeInfo {  
    int32 number;  
    string name;  
    string department;  
}
```

송신하는 프로그램에서 위의 structure 데이터를 QByteArray로 변환(또는 serialized)한 다음 네트워크를 통해 전송한다.

그러면 수신 프로그램에서 QByteArray 데이터를 수신한 다음, 위의 Structure에 데이터를 삽입한다고 해보자.

이럴 때 Protocol Buffer는 Serialized와 Deserialized하는 과정을 쉽게 할 수 있도록 클래스를 자동으로 만들어준다. 따라서 QByteArray에 저장된 데이터를 쉽게 핸들링할 수 있는 기능을 제공한다.

Protocol Buffers는 C++ 이외에도 다양한 프로그래밍 언어에서 지원한다. 따라서 다양한 프로그래밍 언어에서 proto 파일만 동일하게 만들어 놓으면 언어가 달라지더라도 쉽게 활용할 수 있다는 장점이 있다.

이와 관련해 자세한 내용은 Protocol Buffers 공식 홈페이지를 참조하면 더 많은 정보를 참조할 수 있다. (<https://protobuf.dev>)

- Protocol Buffers 설치

Qt에서 제공하는 Protocol Buffers는 MS Windows, Linux 그리고 MacOS에서 설치할 수 있다. MS Windows에서 Protocol Buffers를 설치하기 위해서 아래 GitHub에서 다운로드 받을 수 있다.

예수님은 당신을 사랑합니다.

<https://github.com/protocolbuffers/protobuf/releases>

The screenshot shows the GitHub releases page for the protocolbuffers/protobuf repository. The main heading is "Protocol Buffers v25.1" (Latest). Below it, there's a section titled "Announcements" with a bullet point: "• [Protobuf News](#) may include additional announcements or pre-announcements for upcoming changes." At the bottom of this section, there's a "Python" link. To the left, there's a sidebar for the release "v25.1" by user "anandolee", posted 2 weeks ago. The sidebar includes links for "Compare" and "Tags".

MS Windows에서 Protocol Buffers를 다운로드 후 설치해 보도록 하자. 위의 그림에서 보는 것과 같이 하단에 보면 MS Windows 64 bit 버전을 다운로드 받는다.

The screenshot shows the GitHub releases page for the protocolbuffers/protobuf repository. The main heading is "Protocol Buffers v25.1" (Latest). Below it, there's a table of download links:

| Link | Size | Posted |
|--|---------|-------------|
| protoc-25.1-linux-x86_64.zip | 5.22 MB | 2 weeks ago |
| protoc-25.1-linux-x86_64.zip | 2.96 MB | 2 weeks ago |
| protoc-25.1-osx-aarch_64.zip | 2.11 MB | 2 weeks ago |
| protoc-25.1-osx-universal_binary.zip | 4.23 MB | 2 weeks ago |
| protoc-25.1-osx-x86_64.zip | 2.13 MB | 2 weeks ago |
| protoc-25.1-win32.zip | 3.03 MB | 2 weeks ago |
| protoc-25.1-win64.zip | 3 MB | 2 weeks ago |
| Source code (zip) | | 2 weeks ago |
| Source code (tar.gz) | | 2 weeks ago |

The "protoc-25.1-win64.zip" link is highlighted with a red box. At the bottom of the page, there are reaction counts: 13 thumbs up, 1 smiley face, 3 hearts, 4 stars, and 4 rocket icons. The URL at the bottom of the page is https://github.com/protocolbuffers/protobuf/releases/download/v25.1/protoc-25.1-osx-x86_64.zip.

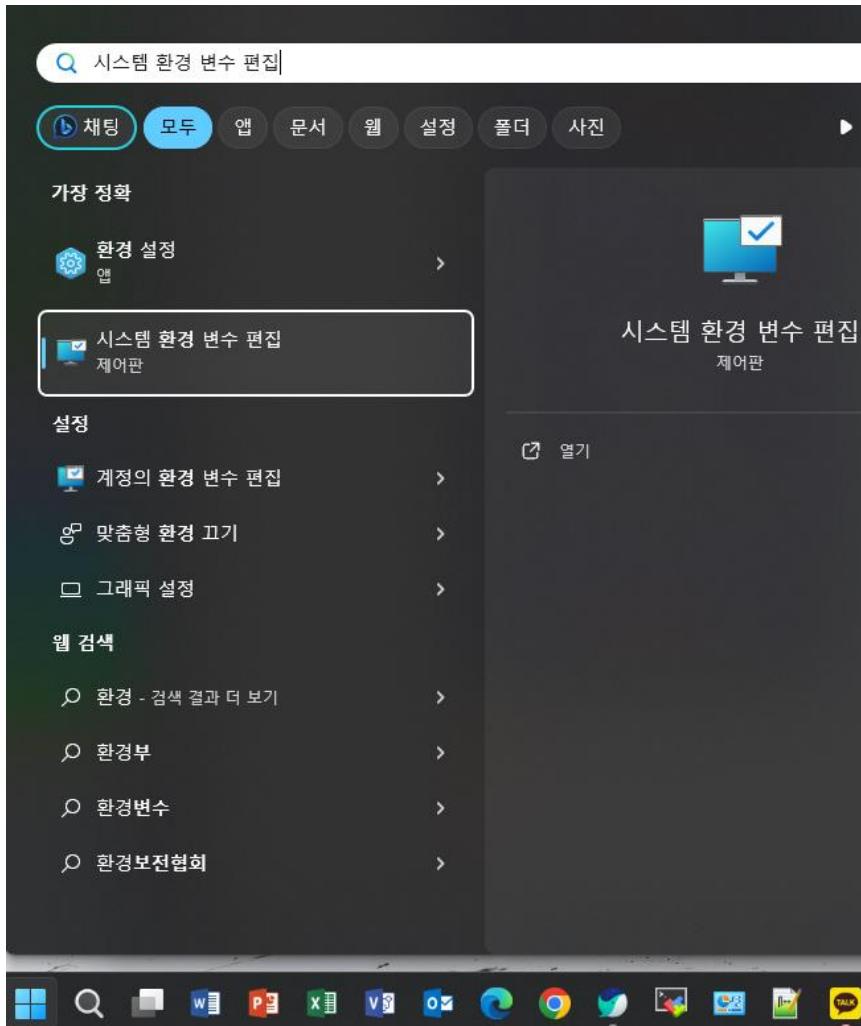
예수님은 당신을 사랑합니다.

위의 파일을 다운로드 받는다. 그리고 이 파일은 실행 바이너리가 아니므로 설치가 필요하지 않는다.

다운로드 받은 파일의 압축을 푼 다음, 압축 해제한 디렉토리는 사용자가 원하는 위치로 이동하면 된다. 필자는 D:\ 로 이동하였다.

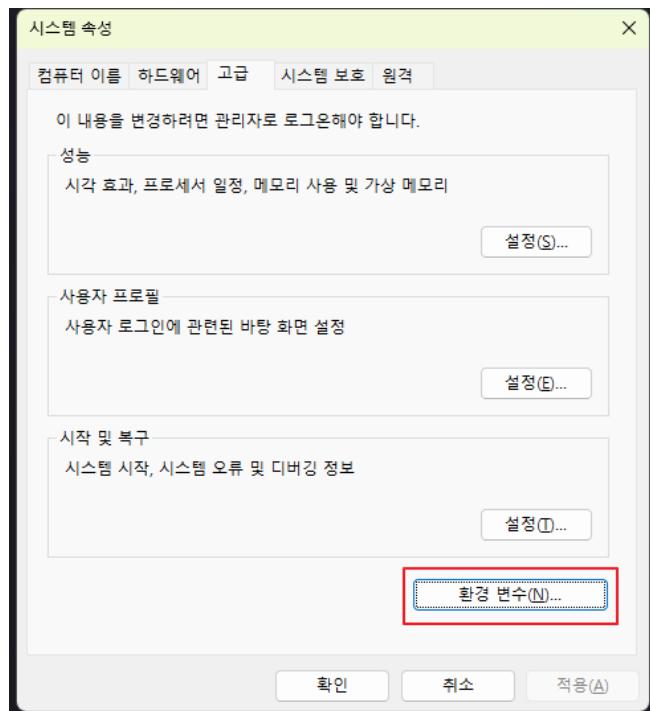
```
D:\protoc-25.1-win64
```

다음으로 위의 Protocol Buffers 디렉토리를 Qt Creator 가 찾을 수 있도록 시스템 환경 변수에 등록한다.

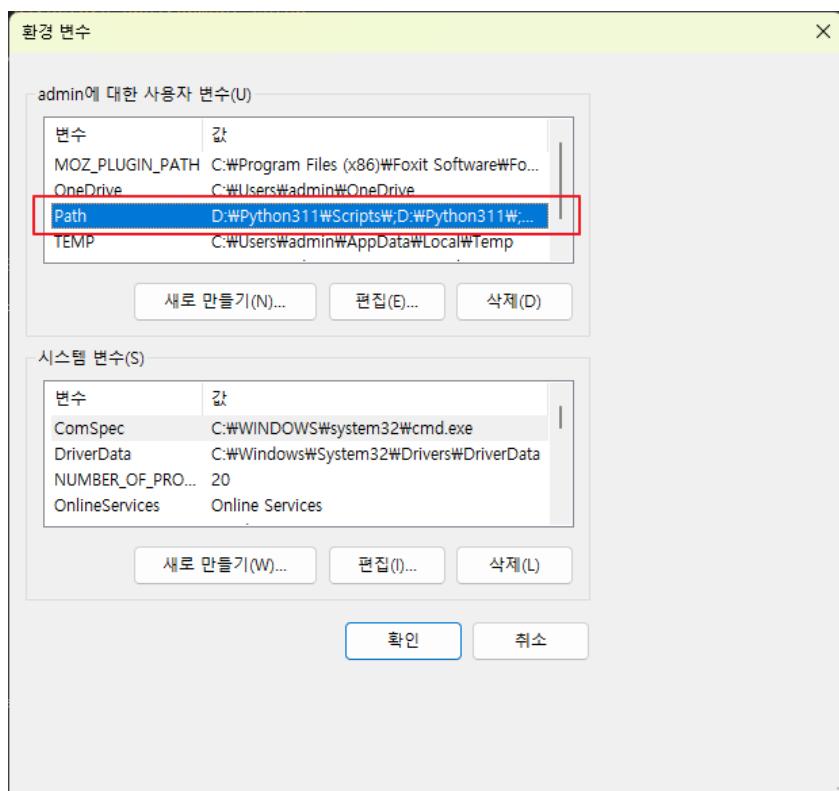


위의 그림에서 보는 것과 같이 [시스템 환경 변수 편집] 을 클릭한다.

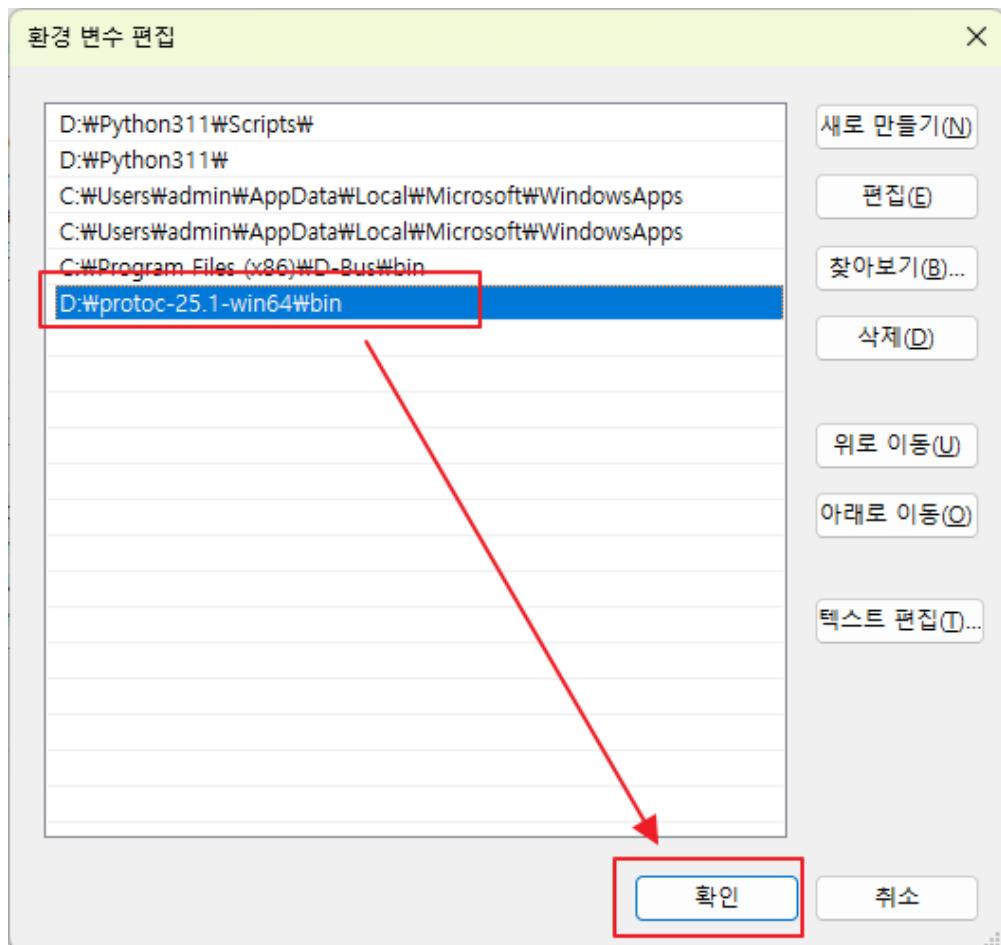
예수님은 당신을 사랑합니다.



위의 그림에서 보는 것과 같이 [환경 변수]를 클릭한다.



예수님은 당신을 사랑합니다.



위의 그림에서 보는 것과 같이 환경 변수에 추가 한 다음 [확인] 버튼을 클릭한다. 다음으로 환경 변수가 정상적으로 추가가 잘 되었는지 아래와 같이 확인 해 보도록 하자.

```
명령 프롬프트
Microsoft Windows [Version 10.0.22621.2715]
(c) Microsoft Corporation. All rights reserved.

C:\Users\admin>protoc --version
libprotoc 25.1

C:\Users\admin>
```

Linux와 MacOS 에서도 위와 같은 방법으로 등록하면 된다. 여기까지 완료하였다면 Qt

예수님은 당신을 사랑합니다.

에서 제공하는 Protobuf 모듈을 사용할 준비가 끝났다.

Qt Protobuf 모듈을 사용하기 위해서는 프로젝트 파일에 아래와 같이 추가해야 한다.

```
find_package(Qt6 REQUIRED COMPONENTS Protobuf)
target_link_libraries(mytarget PRIVATE Qt6::Protobuf)
```

그리고 Qt Protobuf 모듈에서 사용하는 .proto 파일은 아래와 같은 형태로 만들면 된다.

```
syntax = "proto3";

package my.employee;

message EmployeeInfo {
    int32 num = 1;
    string name = 2;
    string department = 3;
}
```

첫 번째 라인은 버전 정보이다. 두 번째 “my.employee” 는 C++에서 namespace가 된다. 예를 들어 C++에서 아래와 같은 namespace로 사용된다.

```
namespace my::employee {
class EmployeeInfo;
}
```

그리고 message 는 C++에서 structure 와 동일한 개념으로 이해하면 된다. 예를 들어 EmployeeInfo 라는 Structure 안에 3개의 변수가 정의되어 있다고 생각하면 된다.

여기까지 작성하였다면 .proto 파일을 참조해 자동으로 EmployeeInfo 클래스를 만들어 준다. 이 클래스를 자동으로 만들기 위해서 프로젝트 파일에 아래와 같이 추가해 주면 된다.

```
qt_add_protobuf(protobuf_employee
    PROTO_FILES
        ../protobuf_files/employee.proto
)
```

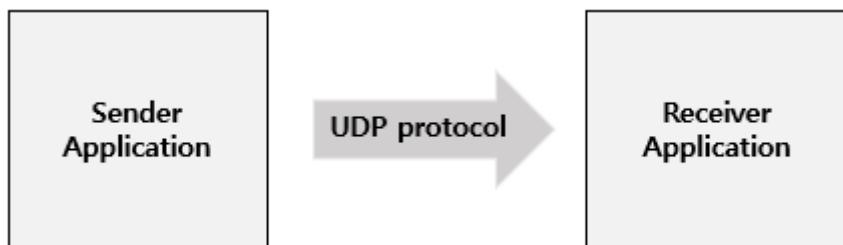
위의 protobuf_employee를 아래에서 보는 것과 같이 target_link_libraries() 항목에 등록하면 된다.

```
target_link_libraries(00_Sender PRIVATE
    Qt6::Core
    Qt6::Widgets
    Qt6::Protobuf
    Qt6::Network
    protobuf_employee
)
```

여기까지 Qt Protobuf 모듈을 사용하는 방법에 대해서 알아보았다. 이번에는 예제를 통해서 사용하는 방법을 알아보도록 하자.

- Qt Protobuf를 이용한 데이터 송/수신 예제

이번 예제는 2개의 예제를 구현해 볼 것 이다. Sender 예제는 Qt Protobuf 모듈을 이용해 Serialized 한 QByteArray 를 UDP Protocol 로 전송한다. 그리고 Receiver 예제는 UDP Protocol 을 이용해 수신한 데이터를 Qt Protobuf 모듈을 이용해 Deserialized 하는 예제이다.



먼저 2개의 예제에서 동일한 Structure를 사용해야 한다. 따라서 아래와 같이 "protobuf_files" 라는 디렉토리를 만든다. 그리고 이 디렉토리 안에 employee.proto 라는 파일을 생성한다.

다음으로 employee.proto 파일을 아래와 같이 작성한다.

```
syntax = "proto3";

package my.employee;

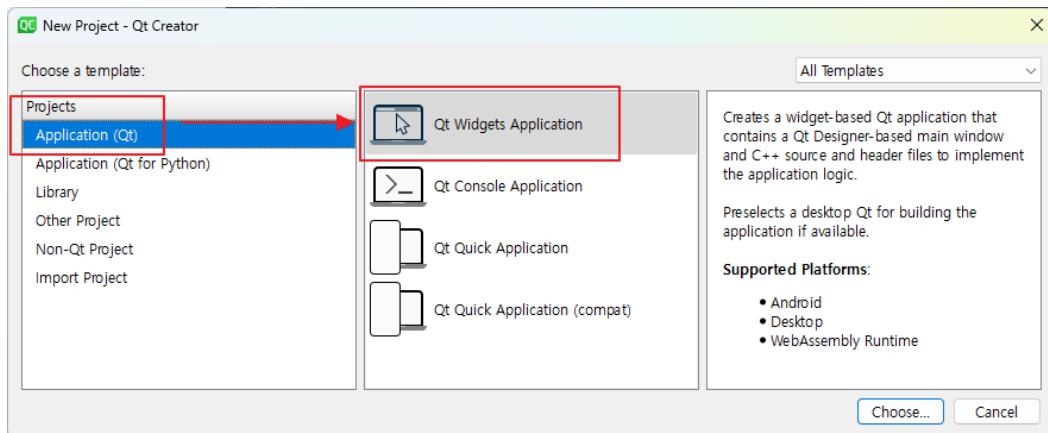
message EmployeeInfo {
    int32 num = 1;
    string name = 2;
```

```
string department = 3;  
}
```

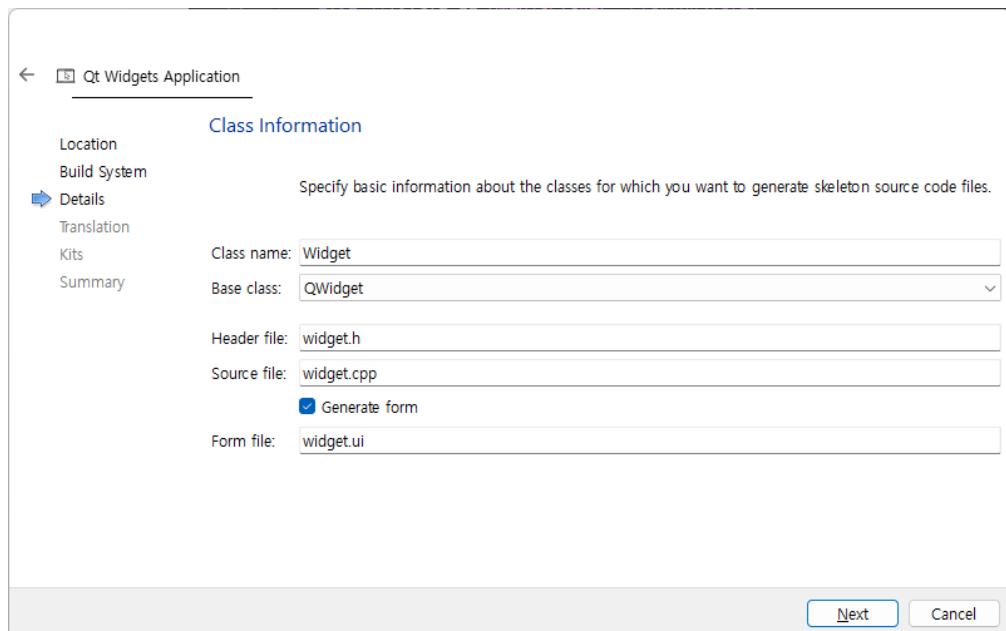
위와 같이 작성하였다면 이번에는 Sender 예제를 만들어 보도록 하자.

✓ Sender 예제 구현

프로젝트 생성 시 Qt Widget 기반의 프로젝트를 생성한다.



이 프로젝트에서는 UI Form을 사용할 것이므로 아래와 같이 [Generation form]에 체크 한다.



프로젝트 생성이 완료되면 CMakeLists.txt 파일에 아래와 같이 작성한다.

```
cmake_minimum_required(VERSION 3.16)

project(00_Sender VERSION 0.1 LANGUAGES CXX)

set(CMAKE_AUIC ON)
set(CMAKE_AUTORCC ON)
set(CMAKE_AUTOMOC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 REQUIRED COMPONENTS
    Core
    Widgets
    Protobuf
    Network
)

qt_standard_project_setup()

set(PROJECT_SOURCES
    main.cpp
    widget.cpp
    widget.h
    widget.ui
)

qt_add_executable(00_Sender
    ${PROJECT_SOURCES}
)

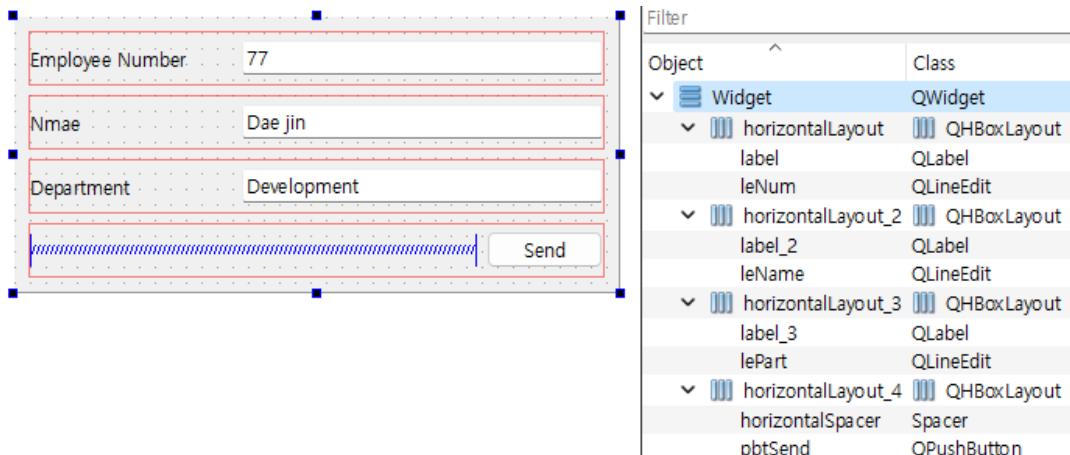
qt_add_protobuf(protobuf_employee
    PROTO_FILES
        ./protobuf_files/employee.proto
)
```

```
target_link_libraries(00_Sender PRIVATE
    Qt6::Core
    Qt6::Widgets
    Qt6::Protobuf
    Qt6::Network
    protobuf_employee
)

install(TARGETS 00_Sender
    BUNDLE DESTINATION .
    LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
    RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
)
```

위와 같이 작성한다. 만약 MOC 관련해 파일을 찾지 못한다는 에러가 발생한다면 위에서 보는 것과 같이 `cmake_minimum_required()` 함수에 CMake 버전 때문에 발생할 수 있다. 따라서 위에서 보는 것과 같이 버전을 3.16으로 수정해 준다.

다음으로 `widget.ui` 파일을 열어서 아래와 같이 GUI 위젯을 배치한다.



다음으로 `widget.h` 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
```

예수님은 당신을 사랑합니다.

```
#include <QProtobufSerializer>
#include <QUdpSocket>

namespace my::employee {
class EmployeeInfo;
}

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

    QUdpSocket m_socket;
    QProtobufSerializer m_serializer;

    void send(const QByteArray &data);

private slots:
    void slot_pbtSend();
};

#endif // WIDGET_H
```

다음으로 widget.cpp 소스코드 파일을 아래와 같이 작성한다.

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

```
#include "employee.qpb.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pbtSend, &QPushButton::clicked, this, &Widget::slot_pbtSend);
}

void Widget::send(const QByteArray &data)
{
    if (m_socket.writeDatagram(data, QHostAddress::LocalHost, 61122) == -1) {
        qWarning() << "Unable to send data of size: "
                    << data.size()
                    << "to UDP port 61122";
    }
}

void Widget::slot_pbtSend()
{
    my::employee::EmployeeInfo eInfo;

    QtProtobuf::int32 tNum      = ui->leNum->text().toInt();
    QString          tName     = ui->leName->text();
    QString          tDepart   = ui->lePart->text();

    eInfo.setNum(tNum);
    eInfo.setName(tName);
    eInfo.setDepartment(tDepart);

    QByteArray sendData = eInfo.serialize(&m_serializer);
    send(sendData);
}
```

예수님은 당신을 사랑합니다.

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

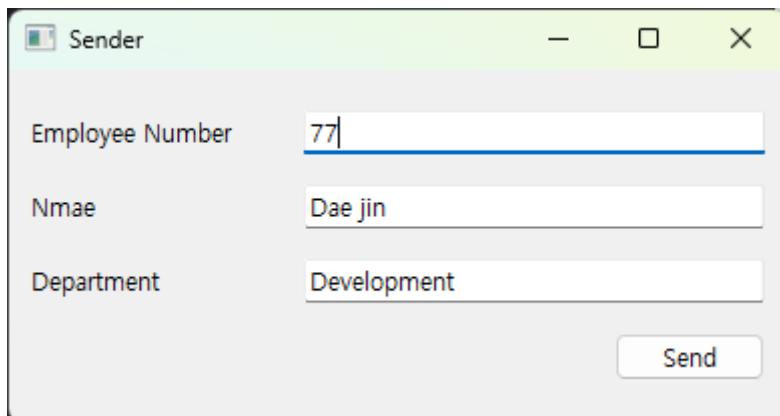
GUI 상에서 [Send] 버튼을 클릭하면 slot_pbtSend() Slot 함수가 호출된다. GUI 상에서 값을 읽어와 int32 와 QString 에 값을 저장한다.

저장한 값을 Qt Protobuf 모듈에서 자동으로 생성해 준 EmployeeInfo 클래스에서 제공하는 멤버 함수를 이용해 값을 저장한다.

그리고 EmployeeInfo 클래스 오브젝트에 저장된 데이터를 QByteArray 에 저장하기 위해서 이 클래스에서 제공하는 serialize() 멤버 함수를 이용하였다.

이 멤버 함수는 EmployeeInfo 클래스 오브젝트에 저장된 데이터를 QByteArray 로 저장해주는 기능을 제공한다.

그런 다음 QByteArray 데이터를 UDP Protocol 을 이용해 전송해 준다.

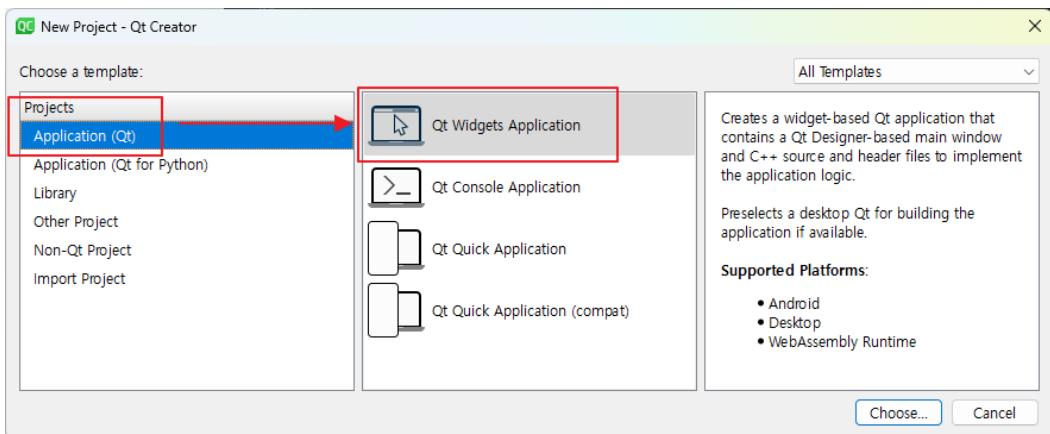


이 예제의 소스코드는 00_Sender 디렉토리를 참조하면 된다.

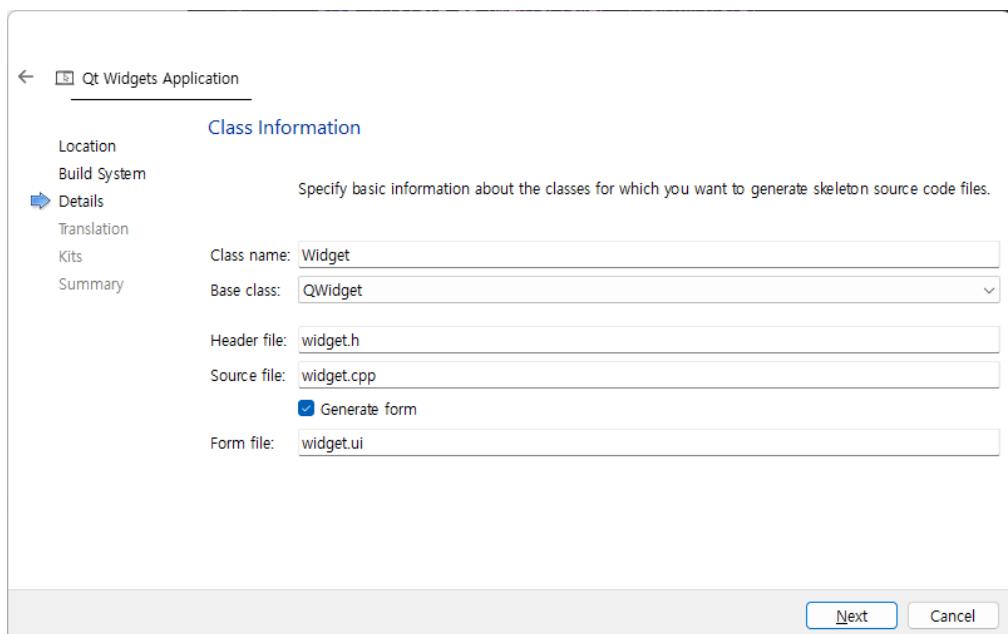
✓ Receiver 예제 구현

Receiver 예제는 UDP Protocol 을 통해 수신 받은 데이터를 GUI에 표시하는 예제이다.

프로젝트 생성 시 Qt Widget 기반의 프로젝트를 생성한다.



이 프로젝트에서는 UI Form을 사용할 것이므로 아래와 같이 [Generation form]에 체크 한다.



프로젝트 생성이 완료되면 CMakeLists.txt 파일에 아래와 같이 작성한다.

```
cmake_minimum_required(VERSION 3.16)
project(01_Receiver VERSION 0.1 LANGUAGES CXX)

set(CMAKE_AUIC ON)
set(CMAKE_AUTORCC ON)
set(CMAKE_AUTOMOC ON)
```

```
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 REQUIRED COMPONENTS
    Core
    Widgets
    Protobuf
    Network
)

qt_standard_project_setup()

set(PROJECT_SOURCES
    main.cpp
    widget.cpp
    widget.h
    widget.ui
)

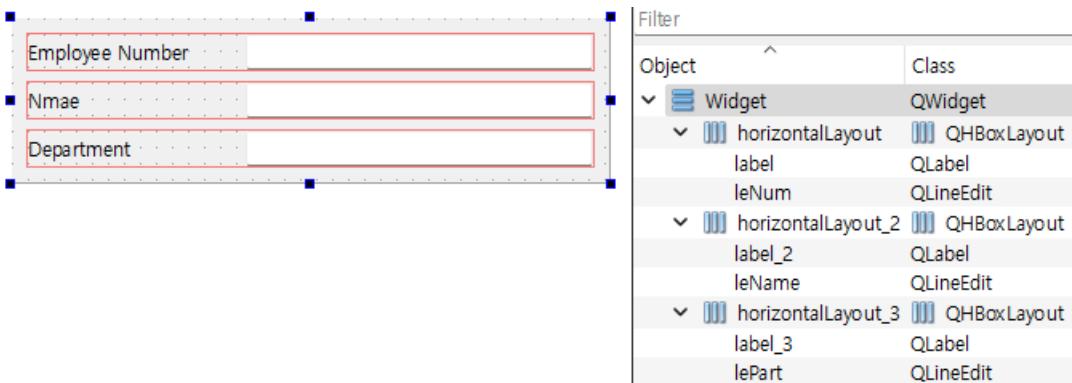
qt_add_executable(01_Receiver
    ${PROJECT_SOURCES}
)

qt_add_protobuf(protobuf_employee
    PROTO_FILES
        ./protobuf_files/employee.proto
)

target_link_libraries(01_Receiver PRIVATE
    Qt6::Core
    Qt6::Widgets
    Qt6::Protobuf
    Qt6::Network
    protobuf_employee
)
```

```
install(TARGETS 01_Receiver
        BUNDLE DESTINATION .
        LIBRARY DESTINATION ${CMAKE_INSTALL_LIBDIR}
        RUNTIME DESTINATION ${CMAKE_INSTALL_BINDIR}
    )
```

다음으로 widget.ui 파일을 열어서 아래와 같이 GUI 위젯들을 배치한다.



다음으로 widget.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QProtobufSerializer>
#include <QUdpSocket>

namespace my::employee {
class EmployeeInfo;
}

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT
```

```
public:  
    Widget(QWidget *parent = nullptr);  
    ~Widget();  
  
    void receive();  
  
private:  
    Ui::Widget *ui;  
  
    QUdpSocket m_client;  
    QProtobufSerializer m_serializer;  
  
    void display(const QtProtobuf::int32,  
                 const QString &,  
                 const QString &);  
  
};  
#endif // WIDGET_H
```

다음으로 widget.cpp 소스코드 파일을 열어서 아래와 같이 작성한다.

```
#include "widget.h"  
#include "ui_widget.h"  
  
#include <QNetworkDatagram>  
#include <QDebug>  
  
#include "employee.qpb.h"  
  
Widget::Widget(QWidget *parent)  
    : QWidget(parent)  
    , ui(new Ui::Widget)  
{  
    ui->setupUi(this);  
  
    Q_ASSERT_X(m_client.bind(QHostAddress::LocalHost, 61122),  
               "Receiver",
```

예수님은 당신을 사랑합니다.

```
"Unable to bind to port 61122");

QObject::connect(&m_client, &QUdpSocket::readyRead,
                 this,      &Widget::receive);
}

void Widget::receive()
{
    while (m_client.hasPendingDatagrams())
    {
        const auto datagram = m_client.receiveDatagram();

        my::employee::EmployeeInfo eInfo;
        eInfo.deserialize(&m_serializer, datagram.data());

        if(m_serializer.serializationError()
           == QAbstractProtobufSerializer::NoError)
        {
            QtProtobuf::int32 tNum      = eInfo.num();
            QString          tName     = eInfo.name();
            QString          tDepart   = eInfo.department();

            display(tNum, tName, tDepart);
        }
    }
}

void Widget::display(const QtProtobuf::int32 vNum,
                     const QString &vName,
                     const QString &vDepartment)
{
    QString strNum      = QString("%1").arg(vNum);
    QString strName     = QString("%1").arg(vName);
    QString strDepartment = QString("%1").arg(vDepartment);

    ui->leNum->setText(strNum);
```

```
ui->leName->setText(strName);
ui->lePart->setText(strDepartment);
}

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

UDP Protocol 을 이용해 데이터를 수신하면 receive() Slot 함수가 호출된다. 이 함수에서 수신 받은 QByteArray 데이터를 Qt Protobuf 모듈을 이용해 자동으로 생성된 클래스의 deserialize() 멤버 함수를 이용해 EmployeeInfo 클래스 오브젝트에 저장한다. 그리고 EmployeeInfo 클래스에서 제공하는 멤버함수를 이용해 값을 읽어 온 다음 GUI 상에 값을 표시한다.



이 예제의 소스코드는 01_Receiver 디렉토리를 참조하면 된다.