

Qt programming for software development in various fields

Qt Programming

First Edition, Ver 1.4



Qt Framework enables software development using the same source code in various operating systems such as desktop, Mobile and Embedded.

Qt Korea Developer Community
www.qt-dev.com

Jesus loves you.

Qt Programming

Version First Edition, Version 1.4, 2020.02.27

Homepage URL www.qt-dev.com

Preface

Qt is a development framework that supports desktop-based operating system platforms such as MS Windows, Linux, and MacOS, operating system platforms such as Embedded Linux, QNX, etc., used in environments, and multi-platforms that can develop software that works on Android and iOS platforms used in mobile devices.

More than 2 million people around the world are using Qt to develop software in a variety of computing environments. We use Qt in a variety of areas ranging from embedded environments such as automobiles, defense, home appliances and mobile devices.

With today's diverse operating system platforms, developing software to support multiple platforms is a challenge. For example, developing the same software that works on Embedded Linux, such as Android, iOS, and so on, requires a lot of time and effort.

Source codes created using Qt can be compatible on a variety of operating system platforms, reducing development time. Therefore, we believe that Qt is the right software development framework to help companies gain competitive edge in rapidly changing markets.

To help those interested in Qt, we will distribute the written content to you as it is in the document file without publishing it as a paper book. I hope it helps those who are interested in Qt.

Lastly, I pray that the people of Israel be blessed and bring peace to Jerusalem. I pray that the people of Israel will realize that Jesus is a messiah. And I pray that God's gospel and love will be spread where the gospel has not yet been spread. I also pray that God's good grace will be with you. Amen

37. Jesus replied: " 'Love the Lord your God with all your heart and with all your soul and with all your mind.' 38. This is the first and greatest commandment. 39. And the second is like it: 'Love your neighbor as yourself.'

[NIV] Matthew 22:37~39

Jesus loves you.

Table of Contents

1. Introduction of Qt and Establishment of Development Environment.....	1
2. Start of Qt programming	15
3. Qt Basic	32
3.1. Qt GUI Widgets	33
3.2. Layout.....	75
3.3. Data type and container classes	81
3.4. Signal and Slot	98
3.5. Implementing the GUI using the Qt Designer tool.....	103
3.6. Dialog	112
3.7. MDI (Multi Document Interface)	125
3.8. File and Data stream	130
3.9. Qt Property	140
3.10. QTimer	146
3.11. QThread	149
3.12. Multilingual support using Qt Linguist Tool.....	157
3.13. Creating an external library.....	165
3.14. Model and View	176
4. GUI 2D Graphics Using QPainter Class.....	188
5. Qt Graphics View Framework	212
6. Animation Framework 와 State Machine	223
7. 3D Graphics Using Qt OpenGL Module.....	236
8. 3D Graphics Using Qt 3D Module	249

Jesus loves you.

9. XML	256
10. JSON.....	267
11. Qt Chart.....	272
12. Qt Data Visualization.....	281
13. Network programming	289
14. Qt WebSockets	327
15. Qt WebEngine	338
16. Inter Process Communication.....	345
16.1. Unix Domain Socket and Named pipe	346
16.2. Communication using QProcess class	355
16.3. Serial communication using the QSerialPort class.....	360
16.4. Share memory using QSharedMemory class.....	368
17. Database.....	375
18. Multimedia.....	395
18.1. Audio.....	397
18.2. Video	416
18.3. Camera	425
19. Qt Install Framework	432
20. Qt for Android	440
20.1. Deploy Android apps using Qt in MS Windows	441
20.2. Deploy Android apps using Qt on Linux	457

1. Introduction of Qt and Establishment of Development Environment

Qt provides the same development framework for developing applications on desktop-based operating systems such as MS Windows, Linux, and MacOS, saving time and money needed for development. And Qt framework has the advantage of being able to develop applications using the same Qt framework on Android (using the same kernel as Linux) and iOS mobile platforms.

Besides desktop and mobile-based platforms, Qt can also develop applications by using Qt on embedded platforms that are embedded in devices such as small devices. Qt framework allows application development using Qt development framework on Embedded Linux, QNX, and WinRT platforms.

Qt uses C++. In addition, the Qt framework provides Qt Quick (QML) in addition to C++. Qt Quick uses an interpreter language called QML. When developing applications with Qt, GUI or UX can be developed using QML. QML can be developed by dividing into business logic and design logic.

In this case, business logic is the process that works when you click a button on the GUI. Design logic refers to GUI. When developing applications with Qt, it can improve development maintainability by developing business logic with C++ and design logic with QML.

For example, by developing the GUI screen in QML and the user event and processing process in C++, two separate logic can be separated to enhance the re-use of the business logic.

GUI can also be developed using C++. There are pros and cons of developing GUI into QML. QML is appropriate if the GUI you use needs to use a touch screen and use a lot of graphic effects, such as Android or iOS.

However, if more information is provided to users, such as desktops, it is less efficient to develop a GUI.

In addition, if memory usage is limited embedded system or CPU (or GPU) performance

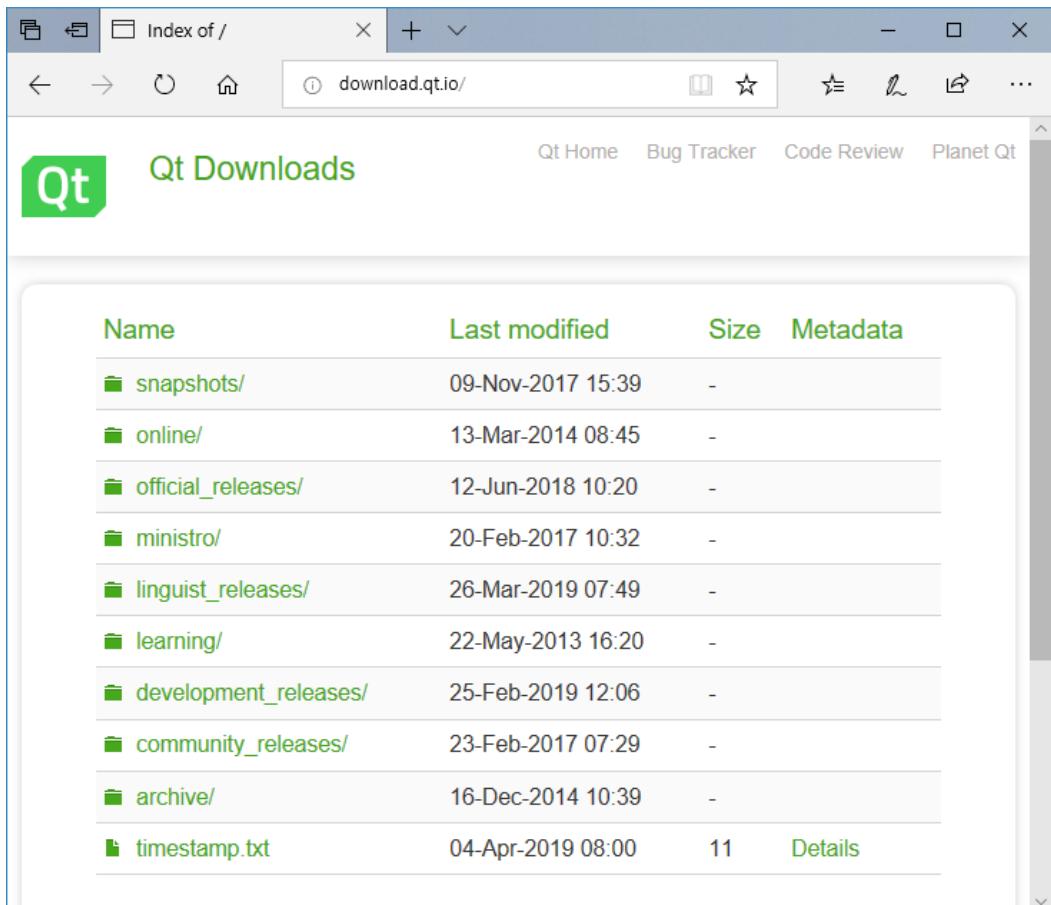
Jesus loves you.

is low, C++ has an advantage in performance over QML. Therefore, it is advisable to first look at the features of the software you are developing and decide whether to use QML or to develop the GUI (Design Logic).

- Establishing a development environment on a desktop-based platform

Qt is a desktop-based platform (or operating system) that supports MS Windows, Linux, and MacOS platforms. The Qt Framework SDK provides a website that can be downloaded by platform from Qt's official website.

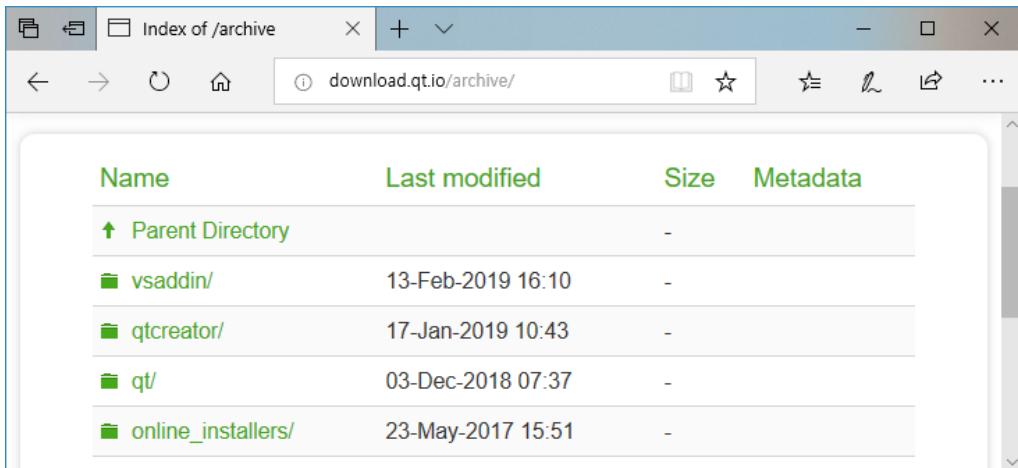
- ✓ Qt Download URL - <http://download.qt.io>



<FIGURE> Qt Download URL

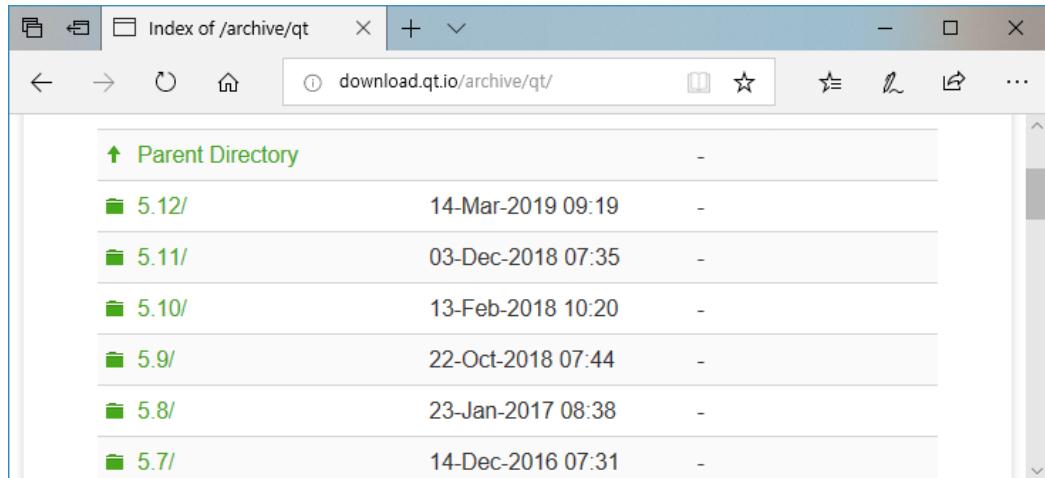
Click on the "archive" item of the item, as shown on the previous screen. Next click on the "qt" item of the item.

Jesus loves you.



<FIGURE> Qt Web Site

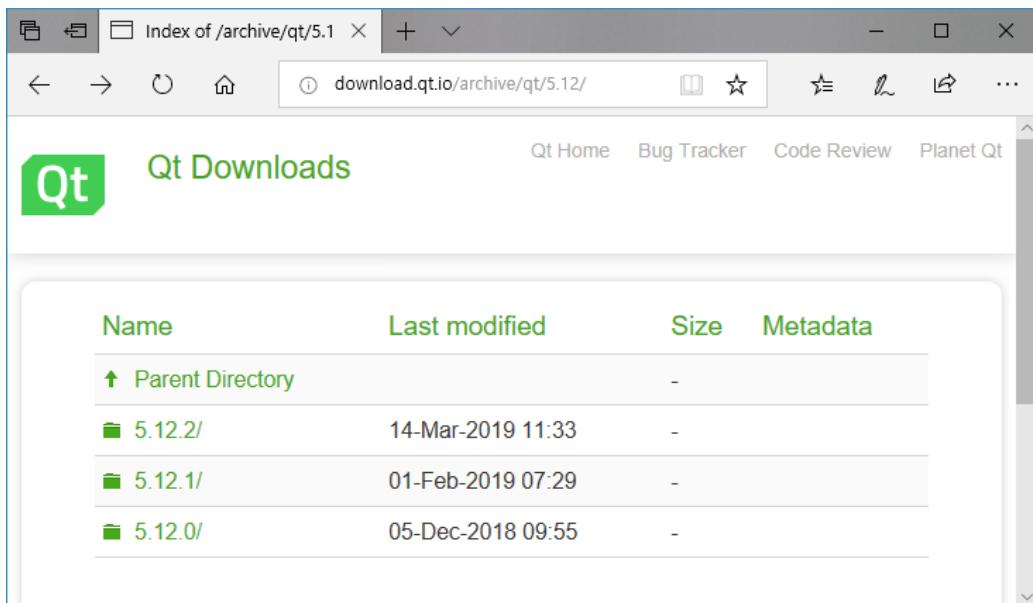
Click on the "qt" item, as shown in the figure above, to download the desired Qt development framework SDK by version, as shown on the next screen.



<FIGURE> Qt Web Site

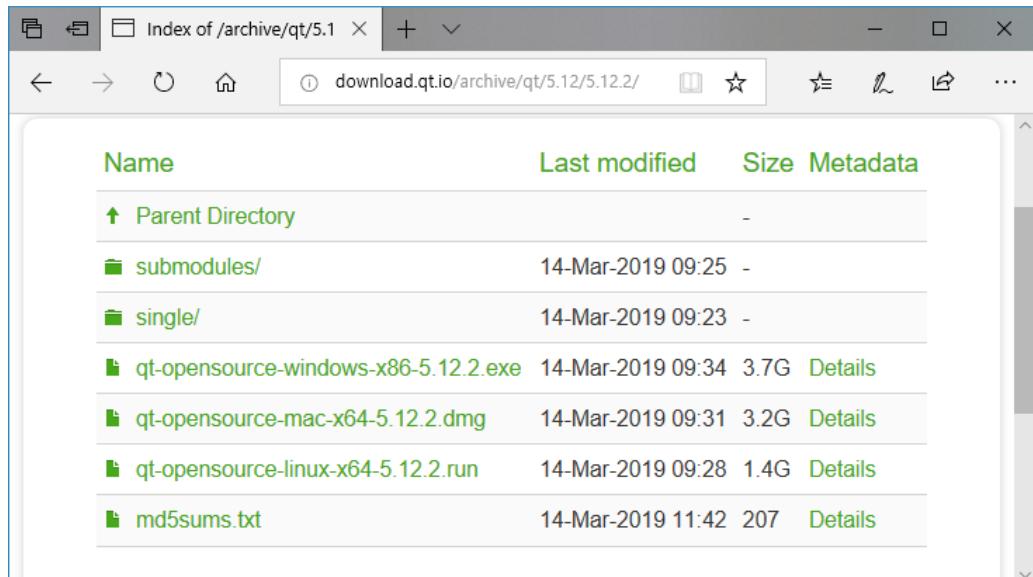
Select the 5.12 version item as shown in the previous figure. Selecting the 5.12 version item allows you to select a sub-version as shown in the following figure.

Jesus loves you.



<FIGURE> Qt Web Site

Click the 5.12.2 version item at the top of the sub-version as shown in the figure above. Then, you can download the Qt Development Framework SDK by platform (operating system) as shown below.

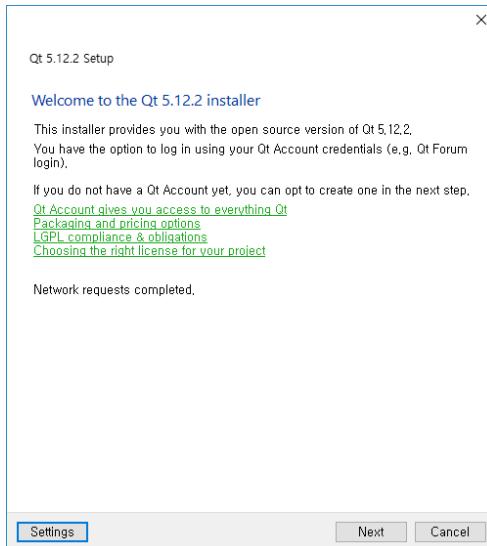


<FIGURE> Framework SDK for Qt Development by Platform

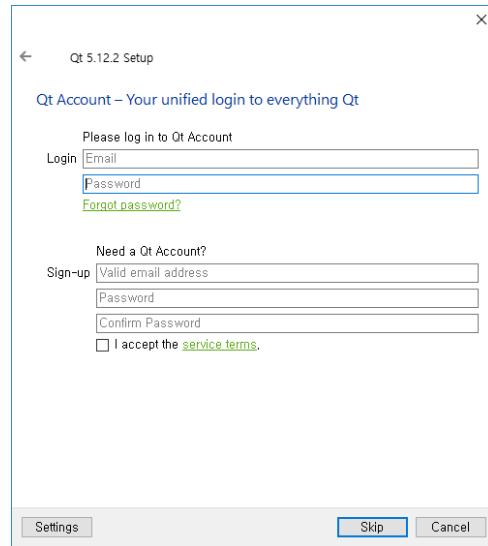
Jesus loves you.

- ✓ Building a Development Environment in MS Windows

If you downloaded the Qt Development Framework SDK from the MS Windows platform (operating system), start installing the downloaded SDK. Run the installation file on the MS Windows platform among the SDKs described in the previous section.

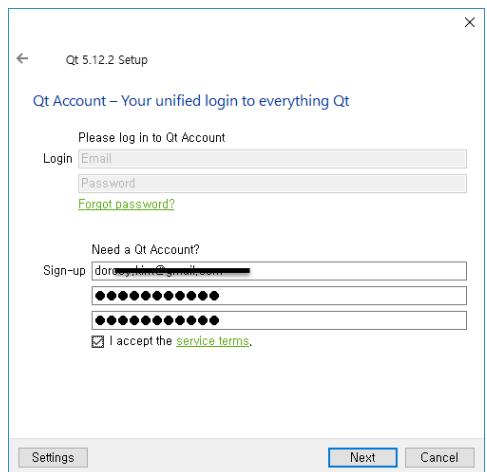


<FIGURE> First installation screen

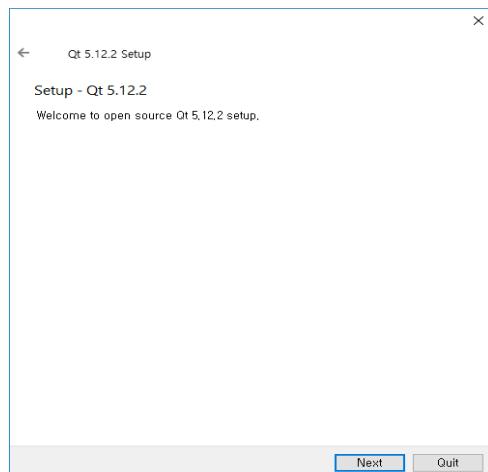


<FIGURE> Second installation screen

The second installation screen in the previous picture will enter the account ID and password. If you do not have an account, enter a new ID or password to use as shown at the bottom of the second installation screen, and the Skip button changes to the Next button. Click the [Next] button. The [Skip] button can be installed without an account.

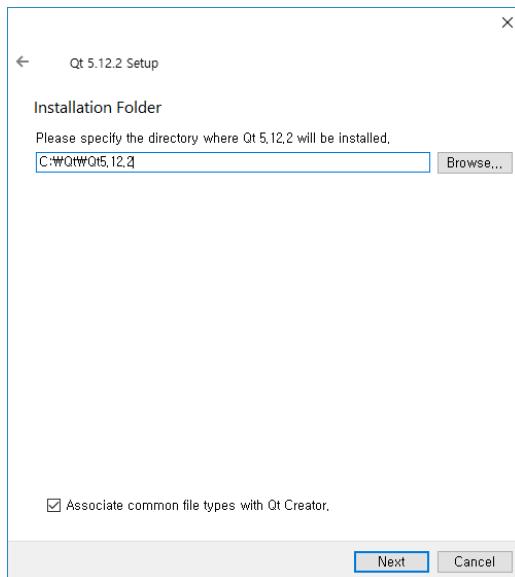


<FIGURE> Enter a new account



<FIGURE> Installation screen

Jesus loves you.



<FIGURE> Enter a new account



<FIGURE> Components 선택 화면

The left-hand picture allows you to enter the directory where you want to install the Qt development framework, or click the [Browse] button to select the directory to install in the directory dialogue window.

The left screen is the screen that selects the components to install.

Items in Qt > Qt 5.X.X. allow the user to choose which compiler to use. For example, if you select MSVC 2015 64-bit among Qt > Qt 5.X.X, you can build an application developed with Qt using Microsoft Visual C++ 2015 64-bit compiler.

Qt > Qt 5.X.X > MinGW 7.3.0 32-bit is a compiler of the 32Bit version of the Open GCC and can build applications developed with Qt.

Jesus loves you.

Items starting with Qt > Qt 5.12.2 > UWP xxx can build applications developed with the Universal Windows Platform compiler Qt. Each item can have multiple choices of compilers to build in the future development of applications. Select all the items here and click the [Next] button.

<TABLE> Components

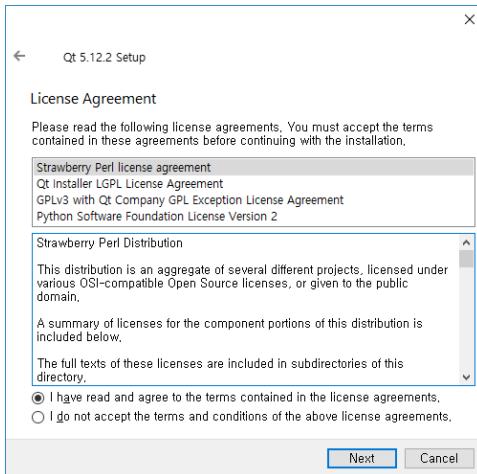
카테고리	Component	설명
Qt X.X.X	MSVC 2015 64-bit	Microsoft Visual C++ 2015 64 Bit
	MSVC 2017 32-bit	Microsoft Visual C++ 2017 32Bit
	MSVC 2017 64-bit	Microsoft Visual C++ 2017 64 Bit
	MinGW 7.3.0 32-bit	Open Source GCC/G++ 32 Bit (Prebuilt)
	MinGW 7.3.0 64-bit	Open Source GCC/G++ 64 Bit (Prebuilt)
	UWP ARMv7 (MSVC 2015)	Universal Windows Platform. (ARMv7 CPU Microsoft Visual C++ 2015)
	UWP x64 (MSVC 2015)	Universal Windows Platform. (ARMv7 CPU Microsoft Visual C++ 2015 64 bit).
	UWP ARMv7 (MSVC 2017)	Universal Windows Platform. (ARMv7 CPU Microsoft Visual C++ 2017)
	UWP x64 (MSVC 2017)	Universal Windows Platform (Intel x86 계열 CPU 용 Microsoft Visual C++ 2015 64 bit)
	UWP x86 (MSVC 2017)	Universal Windows Platform. (Intel x86 CPU Microsoft Visual C++ 2015)
	Android x86	Compiler for use with mobile Android platforms using Intel x86 series CPU.
	Android x64 ARM64-v8a	ARM64-v8a CPU Compiler
	Android ARM7	ARM7 CPU Compiler
	Source	Qt api to simplify debugging the necessity For example, a break for a specific point source code using qt api to simplify debugging the code

Jesus loves you.

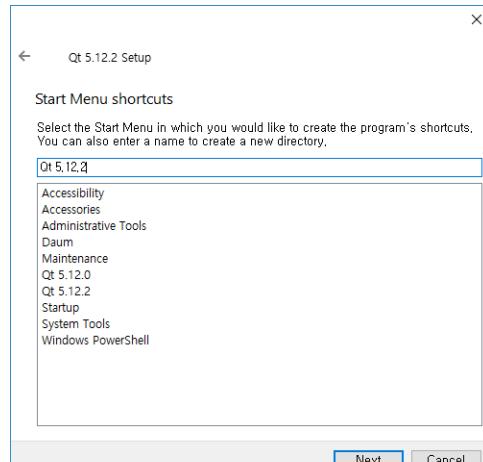
		required.
	Qt Charts	Graph Chart API Component.
	Qt Data Visualization	Chart API component in three-dimensional form.
	Qt Purchasing	APIs to support in-app purchases, such as Google Play and Apple Store.
	Qt Virtual Keyboard	Virtual Keyboard Component Support
	Qt WebEngine	Web browser-based APIs developed by Google
	Qt Network Authorization	Network APIs required to implement features requiring network authentication.
	Qt WebGL Streaming Plugin	Components such as Texture, Buffer, and Glyphs in Web browsers that support WebGL.
	Qt Script (Deprecated)	Items no longer in use
Developer and Designer Tools	Qt Creator 4.8.2	Qt IDE development tool. Users are not allowed to choose. This is a required installation item.
	Qt Creator 4.8.2 CDB Debugger Support	Component to support CDB for debugging purposes. MinGW uses GDB, but if you use MSVC compiler, you can debug only CDB.
	MinGW 7.3.0 32-bit	MinGW 7.3.0 32 bit ToolChains
	MinGW 7.3.0 64-bit	MinGW 를 빌드 된 7.3.0 64 bit ToolChains
	Strawberry Perl 5.22.1.3	Perl distribution for Microsoft Windows platform

The next installation screen selects the first item to accept the license and clicks the [Next] button.

Jesus loves you.

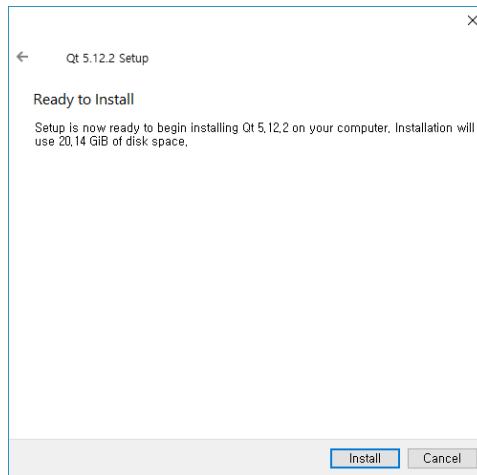


<FIGURE> License Acceptance Screen

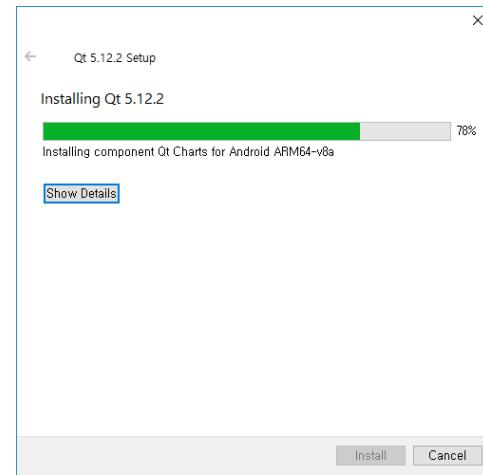


<FIGURE> Name to register with the Start menu

The Name screen to register with the Start menu can be viewed by clicking the [Next] button on the Select a name to register in the Windows Start menu. Click the [Next] button on this dialog screen to proceed with the installation.



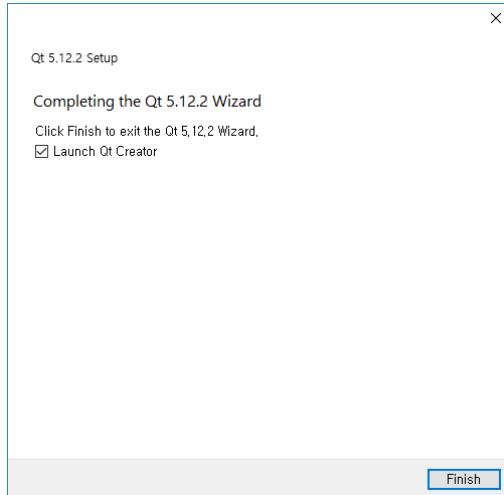
<FIGURE> Installation Preparation Screen



<FIGURE> Installation Process Screen

When the installation process is complete, there is a "Launch Qt Creator" checkbox entry. This item is a check box that asks if you want to start an IDE to be used when developing Qt. Click the [Finish] button below to complete the installation.

Jesus loves you.



Check the "Launch Qt Creator" item as shown in the figure and click the [Finish] button to run an IDE development tool called Qt Creator after the installation dialogue ends.

<FIGURE> 설치 완료 화면

✓ Building a Development Environment on Linux

The Linux platform (operating system) has multiple distribution editions, but uses the Ubuntu 18.04 64bit version as the platform for installing the Qt Framework SDK. The downloaded Qt framework SDK installation file for Linux should be added to the execution authority using the chmod command in the terminal, as shown in the following figure.

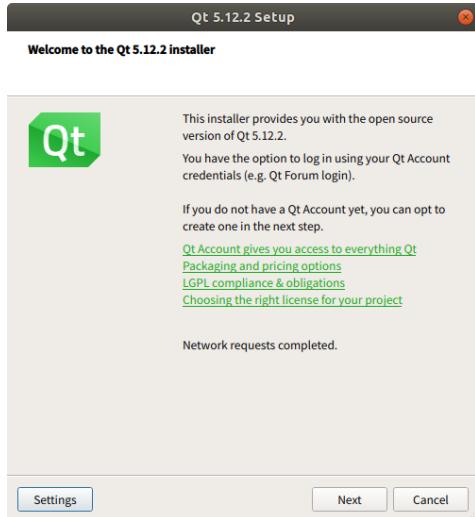
A screenshot of a terminal window titled 'incubic@linux1: ~/다운로드'. The terminal shows the following command sequence:

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

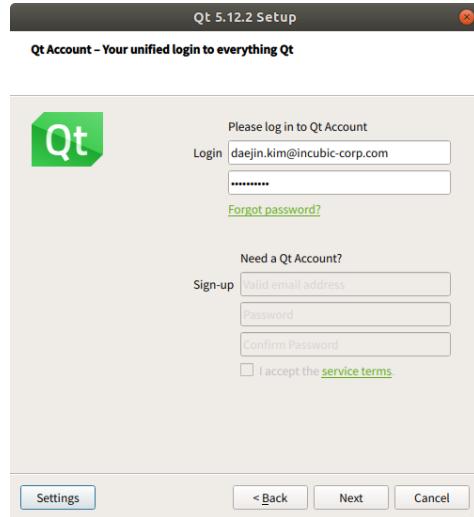
<FIGURE> Added execution authority using chmod

Once you have completed adding executable permissions, you can also use the file browser provided by Ubuntu Linux to follow the executable file. Alternatively, it may be run directly from the terminal as shown in the previous figure.

Jesus loves you.

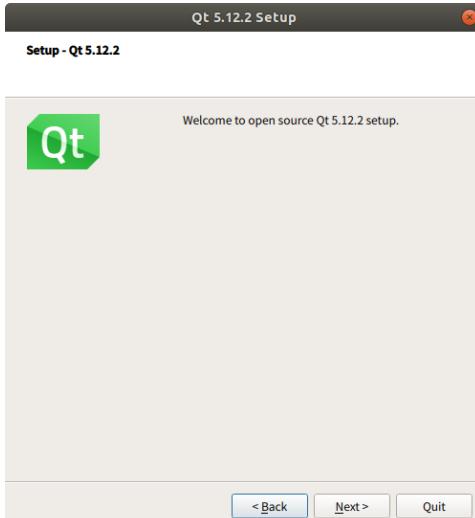


<FIGURE> Setup Welcome screen

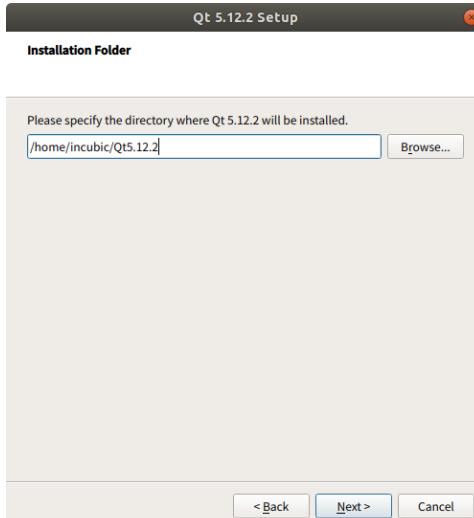


<FIGURE> Account Input Reception

In the previous picture, the account entry screen enters the account ID and password. If you do not have an account, enter a new ID and password to use as shown at the bottom of the account input dialogue screen and click the [Next] button.



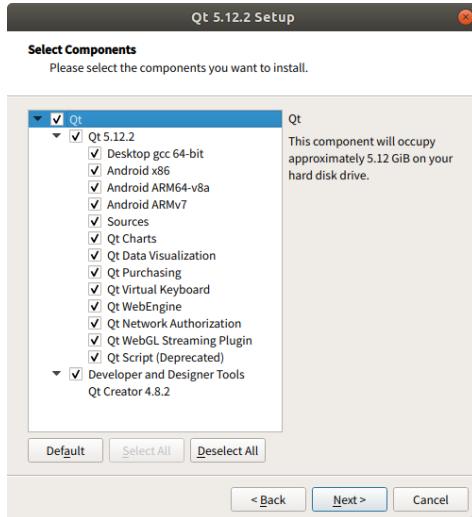
<FIGURE> Installation screen



<FIGURE> Specify the directory in which to install Qt

Select the directory where you want to install the Qt Framework SDK, then click the Next button.

Jesus loves you.



<FIGURE> Components Select screen

<TABLE> Linux Qt Framework SDK Installation Component description

카테고리	Component	설명
Qt X.X.X	Desktop gcc 64-bit	Compiler for building Qt applications using the Open GCC/G++ 64 bit compiler
	Android x86	Compiler for use with mobile Android platforms using Intel x86 CPUs.
	Android x64 ARM64-v8a	64 bit compiler for Android platforms with ARM64-v8a CPU.
	Android x64 ARM7	64 bit compiler for use with mobile Android platforms using ARM7 CPUs.
	Source	Required to use debugging within API provided by Qt in addition to developed source code.
	Qt Charts	Graph Chart API Component.
	Qt Data Visualization	It is a chart API Component in 3D
	Qt Purchasing	It is an API to support in-app purchases such as Google Play and Apple Store.
	Qt Virtual Keyboard	virtual keyboard

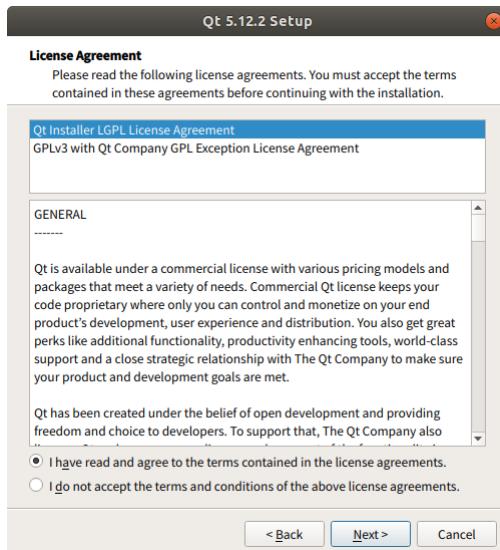
The picture on the left shows the component to be installed.

Refer to the following table for each component description. Select all components here, then click the [Next] button.

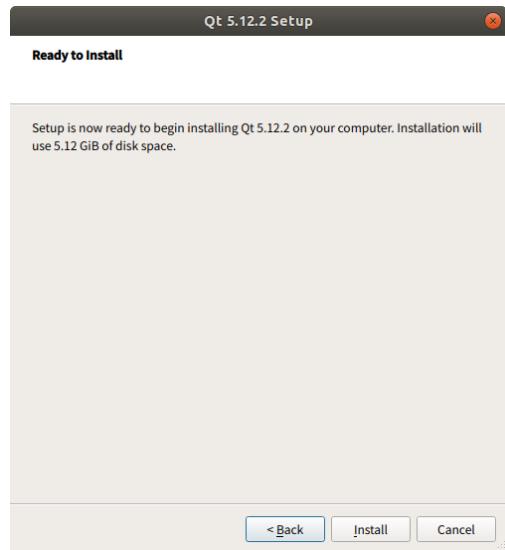
Jesus loves you.

	Qt WebEngine	Required when developing web browser-based applications on Qt.
	Qt Network Authorization	Network APIs required to implement features that require authentication in the network
	Qt WebGL Streaming Plugin	Components such as Texture, Buffer, and Glyphs on web browsers that support WebGL
	Qt Script (Deprecated)	Items no longer in use
Developer and Designer Tools	Qt Creator 4.8.2	Qt IDE development tool. Users are not allowed to choose. This is a required installation item.

Select all Components as shown in the previous illustration, then click the [Next] button. This is a license agreement dialogue. Select the first item, then click the [Next] button.



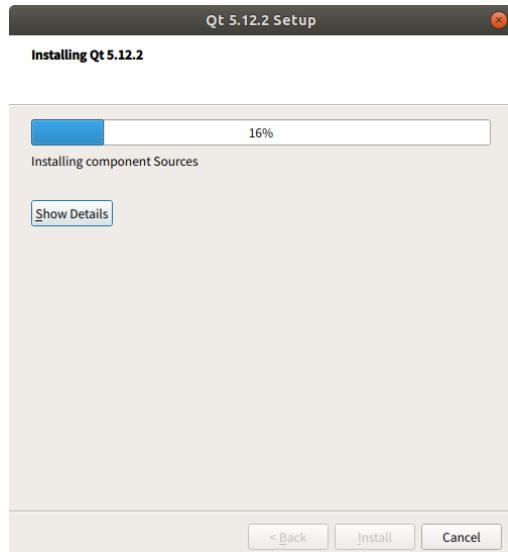
<FIGURE> License Acceptance Screen



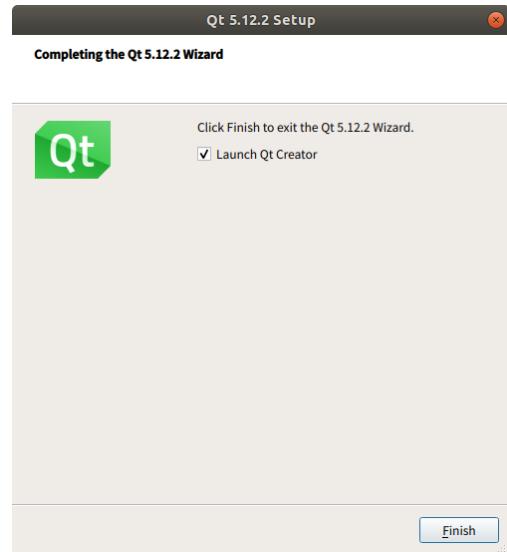
<FIGURE> Screens before starting the installation.

All setup is complete, as shown on the screen before installation. Click the [Install] button to start the installation.

Jesus loves you.



<FIGURE> Installation Progress Screen



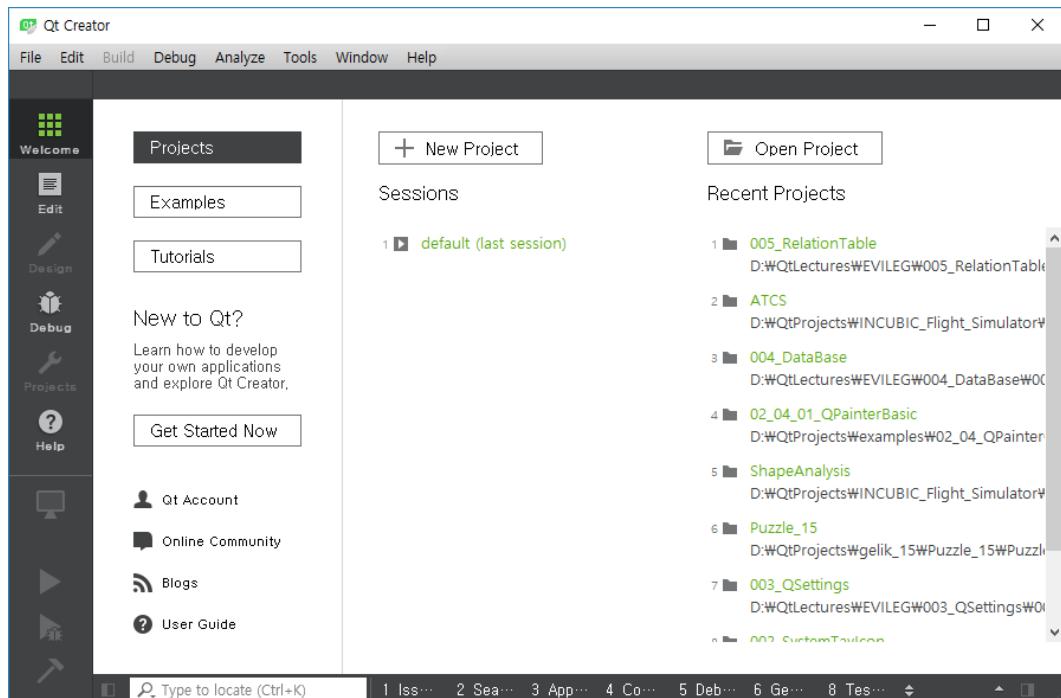
<FIGURE> Installation Completed Screen

2. Start of Qt programming

In this chapter, let's implement an application that outputs "Hello World" to a command window or terminal window without a GUI, and an application that outputs "Hello World" to a GUI window.

- Programming the "Hello World" output example to a command or terminal
- Qt Creator would have been installed if the Qt development framework had been installed. Qt Creator is an IDE tool. Provide developers with an application development environment. For example, development tools such as Visual Studio and Eclipse (IDE) provide Qt Creator development tools with IDE tools.

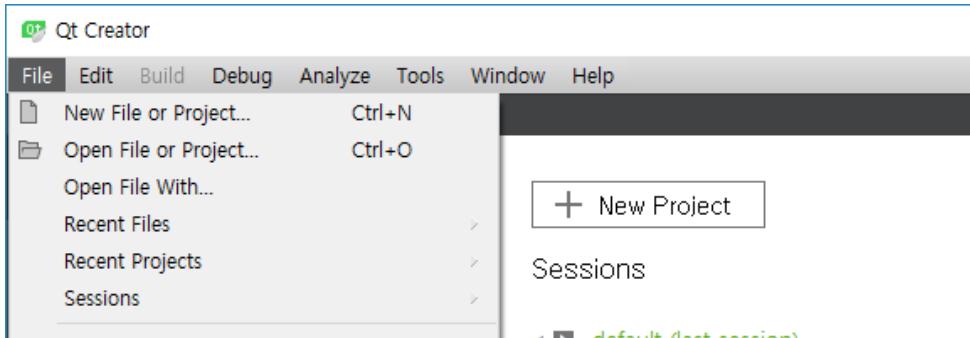
Running the installed Qt Creator will ensure that the Qt Creator is running as shown in the following figure.



<FIGURE> Qt Creator Run Screen

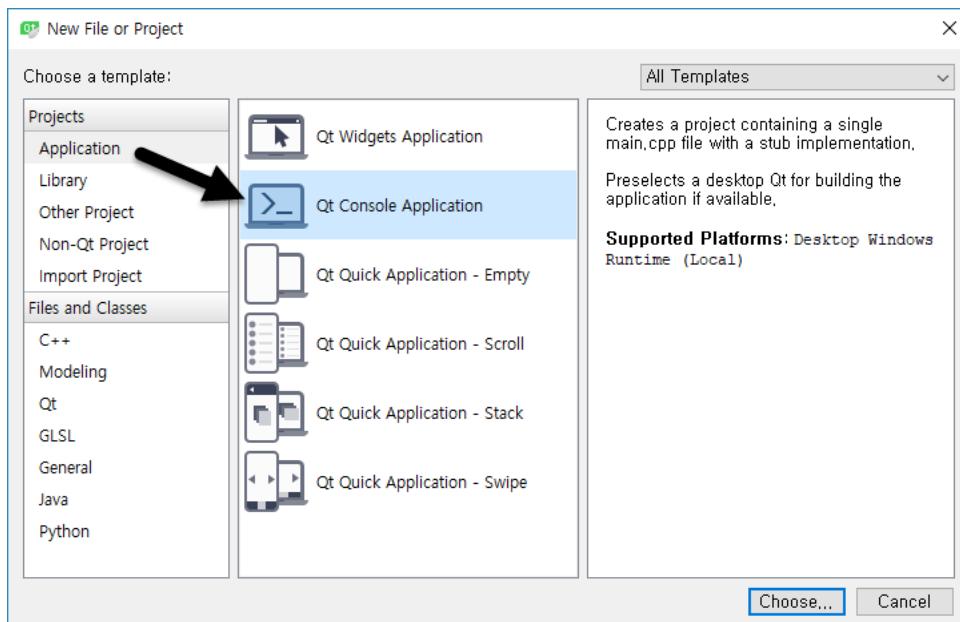
Click on the [File] -> [New File or Project] menu in the Qt Creator menu.

Jesus loves you.



<FIGURE> Qt Creator's [File] menu

The [New File or Project] menu automatically creates the project's outline (or framework) according to the characteristics of the application you want to develop. For example, if the characteristics I want to develop are libraries rather than applications, Qt Creator automatically creates the most basic architecture for library development.



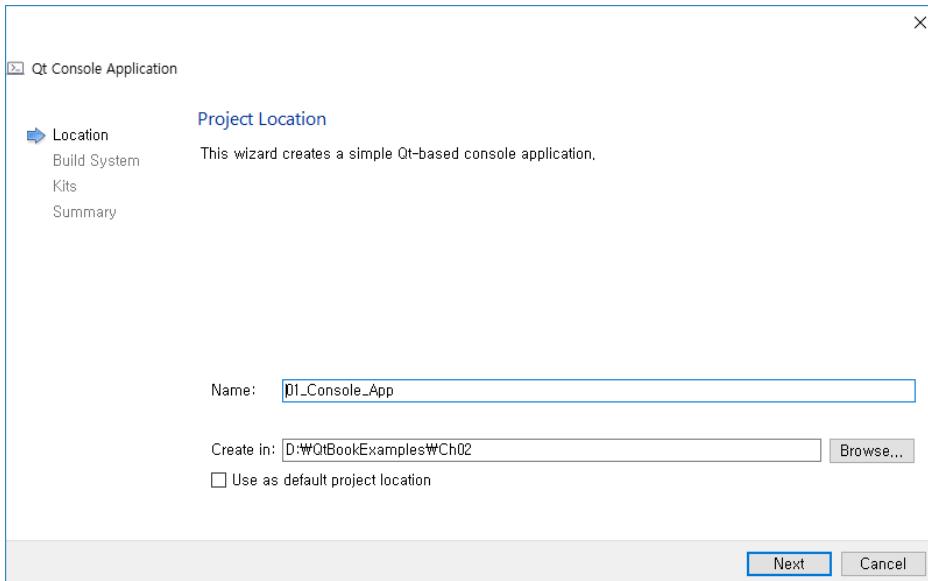
<FIGURE> [New File or Project] Dialog

Since the Command window will produce an example of printing "Hello World," select [Application] -> [Qt Console Application] as shown in the figure above and click the [Choose] button at the bottom.

Click the [Choose] button to specify where the project is created. Enter the project name and the parent directory where the project will be located, as shown in the following figure.

Jesus loves you.

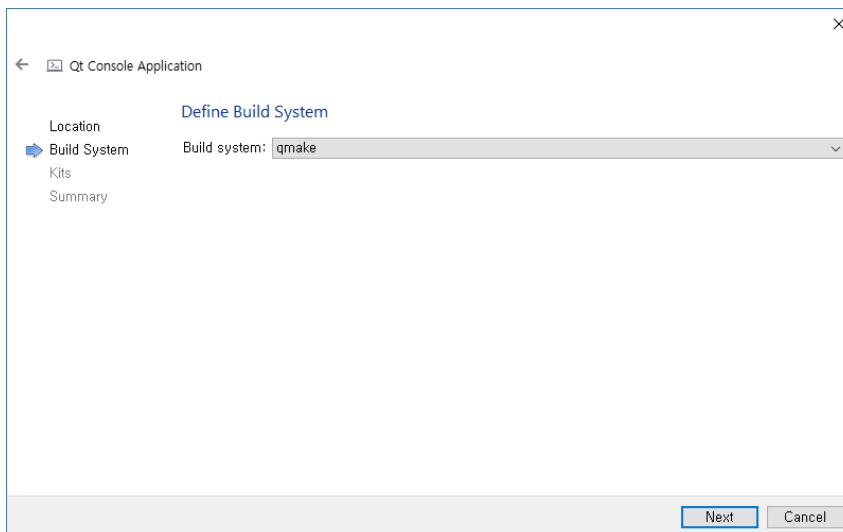
When selecting the name and location of the project, enter the English name because using Hangul as a point of caution may result in errors. And don't let Hangul enter the directory where the project is located, as shown in the following picture.



<FIGURE> Dialog that enter the name and location of the project

As shown in the figure, the [Name] entry enters the name of the project. For the [Create In] entry, enter the parent directory where the project will be located and click.

The following figure should select the build system to build the project. Select the first item [qmake] and click the [Next] button at the bottom, as shown in the following figure.



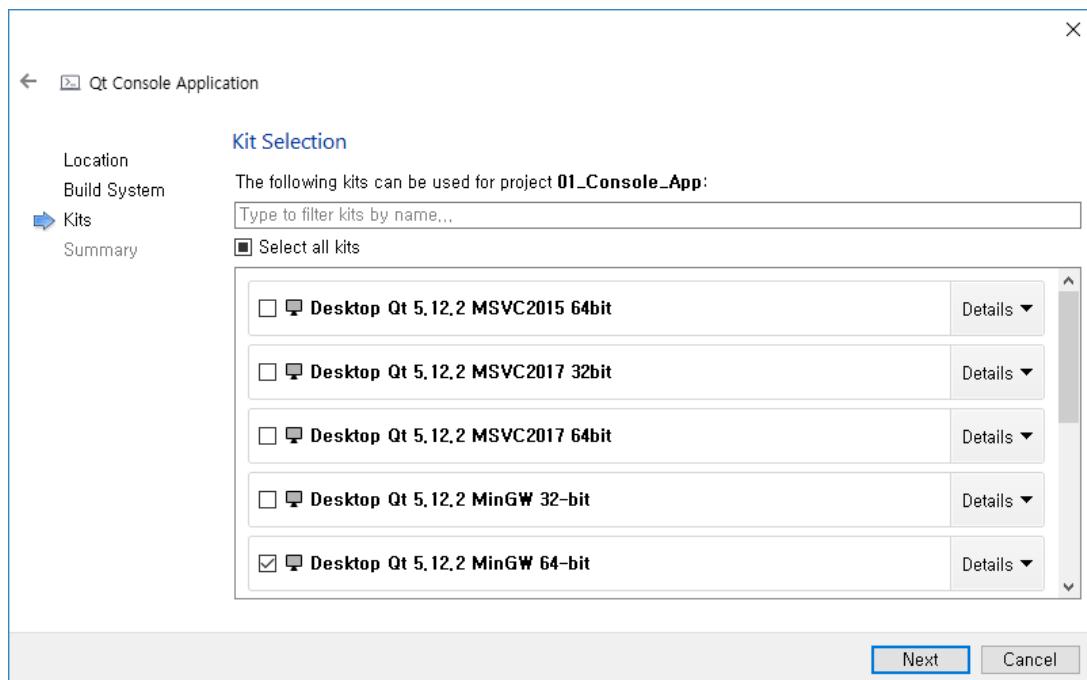
<FIGURE> Build System Selection Dialog

Jesus loves you.

The following screen is a selection of compilers to compile the application you want to create. Here, select the MinGW 64-bit item. MinGW is an open-source GCC compiler. You can also select MinGW 64-bit or MinGW 32-bit.

In addition to MinGW, the MSVC2017 compiler must be installed first to select the MSVC2017 64bit version. Therefore, if you select one of the MSVC items, as shown in the figure below, without MSVC 2017 installed on your PC, the compiler cannot be installed due to an error.

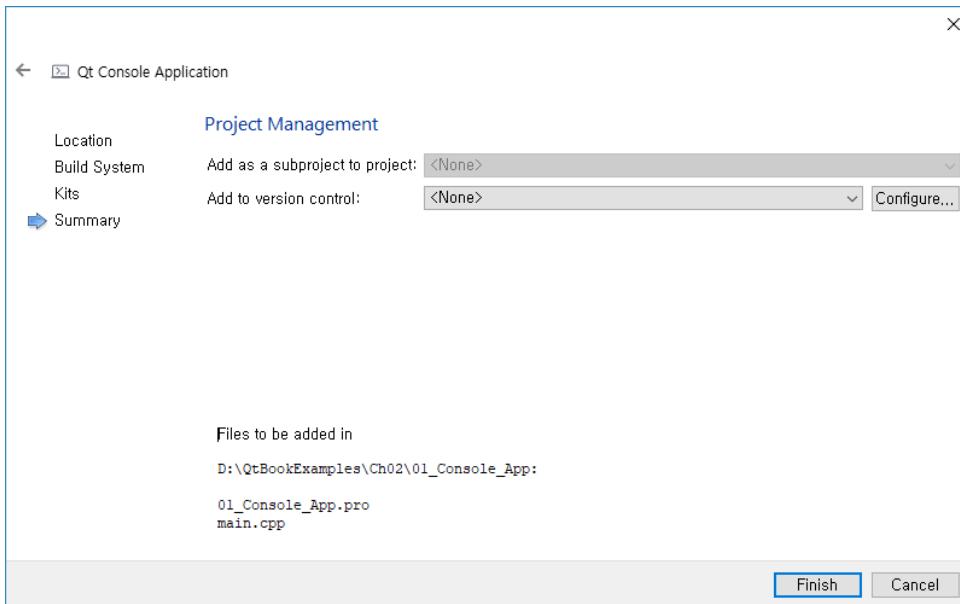
However, if you select MinGW, the MinGW compiler is automatically installed when installing the Qt development framework, so you can build applications developed with Qt without installing a separate compiler.



<FIGURE> Compiler selection dialogue to build

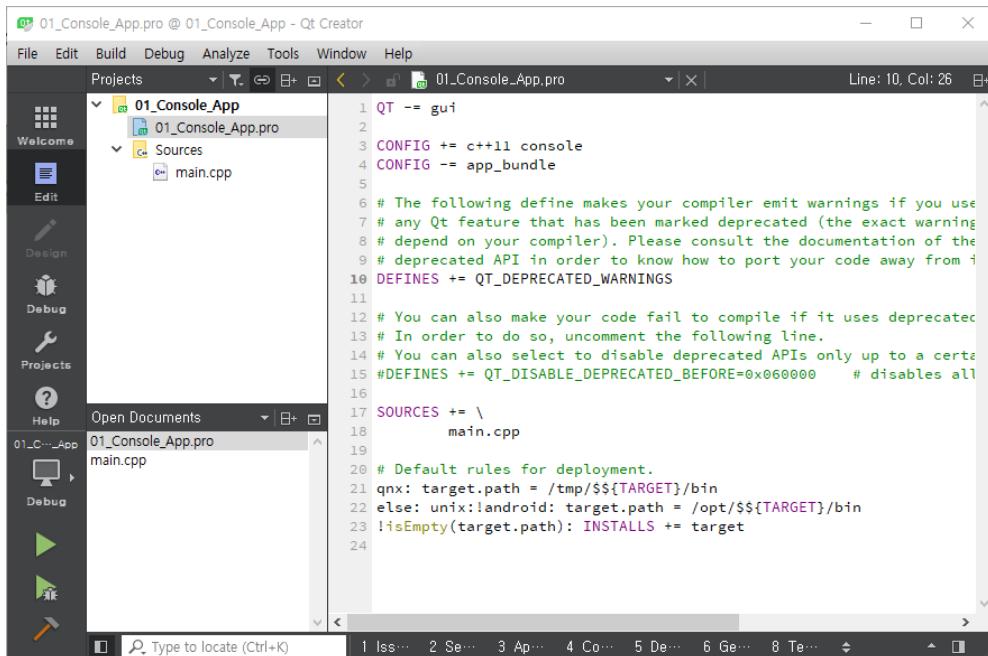
Select the [Desktop Qt 5.x.x MinGW 64Bit] item as shown in the figure above. Then click the [Next] button at the bottom.

Jesus loves you.



<FIGURE> Project Generation Last Dialog

As shown in the previous figure, clicking on the bottom [Finish] button completes all setup and automatically creates a basic source code for developing console-based applications, i.e. creating a "Hello World" program in a terminal or command window.



<FIGURE> The screen on which the project was created

As shown in the previous figure, it can be seen that all of the source codes for "Hello

Jesus loves you.

"World" output have been generated. You can verify that two files have been created. The 01_Console_App.pro file is a project file that specifies the properties of a project. The second generated main.cpp file is automatically created with the main function from which the program starts. Add the source code to the main.cpp source code file as follows:

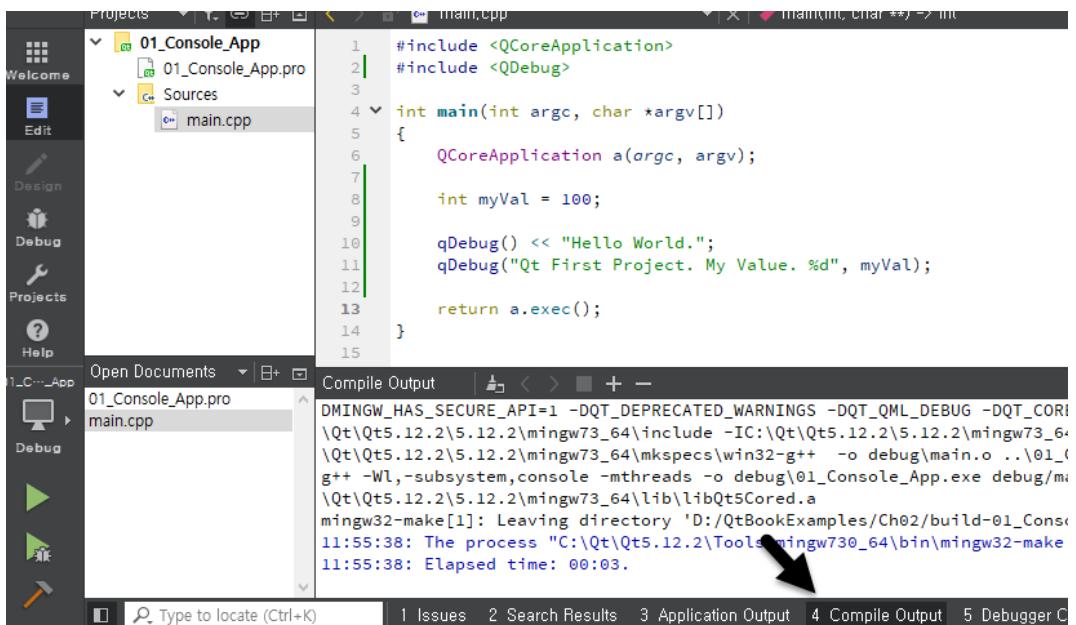
```
#include <QCoreApplication>
#include <QDebug>

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

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

    return a.exec();
}
```

The example source code above prints "Hello World" on the console. To build, the hammer shape at the bottom left of the Qt Creator tool window screen has a build icon button. Click this button to proceed with the build. The build shortcut is [Ctrl + B].

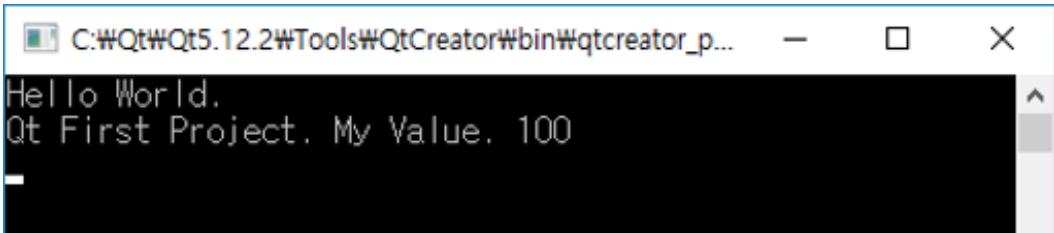


<FIGURE> Qt Creator Build Log Output

Jesus loves you.

You can view the build process through the log to see if an error occurred during the build process and if it was built normally. Click on [Compile Output] at the bottom of the Qt Creator tool to view the build log.

Click on the arrow shape at the bottom left of the Qt Creator to run the program you created after the build. The shortcut key is the [Ctrl + R] key.



<FIGURE> Program Execution Screen

- ✓ Project file (.pro)

The project file has a .pro extension. A project file is a file to define the properties of a project. For example, the purpose of this project is to specify attributes to distinguish between GUI-based applications, console applications, and libraries for other applications.

The following is the code for the project file.

```
QT -= gui

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

SOURCES += \
    main.cpp

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

The QT macro on the first line should specify which API items or groups will be used in the Qt framework. where "gui" means the GUI module provided by Qt. This means not using the GUI module.

For example, to use the network APIs and GUI provided by Qt, you must add the following as a QT macro:

Jesus loves you.

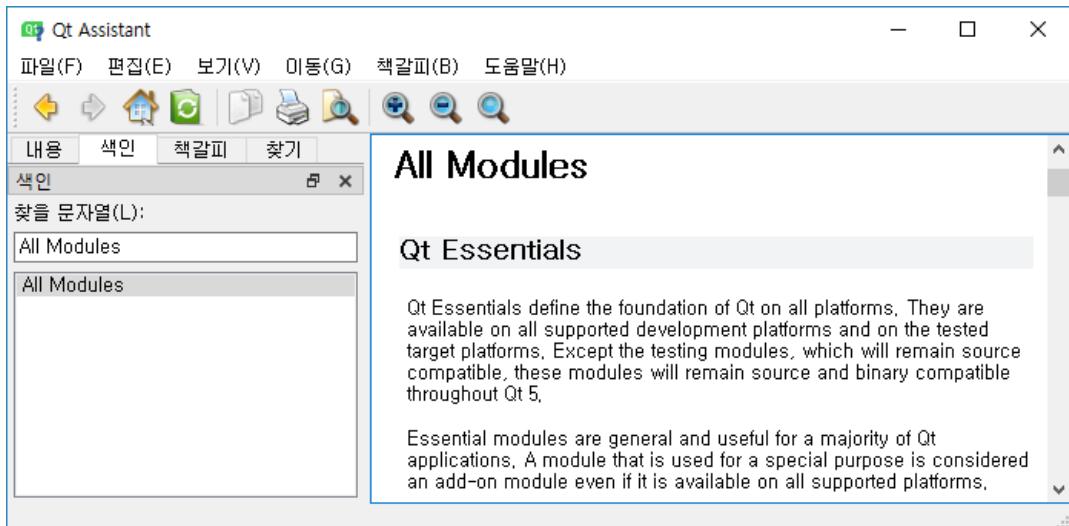
```
QT += network  
QT += gui
```

One may be used in the above-mentioned manner, but to specify several modules, the following may be used:

```
QT += network gui
```

As we just saw, not using unused modules or APIs in QT macros can reduce the capacity of modules (APIs) used by executables during post-application deployment. For example, if a network module is specified in a QT macro, it is recommended that you do not specify a module that is not used because the Qt Network Module Library file (such as dll or so) must also be distributed when distributing executable files.

Modules that can be used in Qt can be searched using the help of Qt Assistant, which is installed with Qt Select [Index] in the left-hand tab and enter "All Modules" in the string to be found, as shown in the figure below, to find out more about the modules available in Qt.



<FIGURE> Qt Assistant Screen

CONFIG on the next line is the build setup. C++11 means that you will use the C++11 version to build. console means that this project property is a console program.

The app_bundle in the following line shows the console program properties when built in MacOS. In Window, use console for CONFIG macro and app_bundle for MacOS.

In addition to CONFIG, use "-=" to add and remove items to the CONFIG option in the

Jesus loves you.

project file(.pro)

The DEFINE Macro allows you to add properties of a compilation by specifying options in the compilation. The use of "QT_DEPRECATED_WARNINGS" in the DEFINE macro means that an alert should be output when using APIs missing by Qt version.

The SOURCE macro specifies the source file. There is no header file here, but if there is a header file, specify the header file in the HAEDER macro. And the last three lines are examples of how a particular platform works.

qnx: target.path = /tmp/\${TARGET}/bin is the line performed on the platform QNX. If the following line is a Linux platform rather than a QNX, the last line means that both QNX and Linux.

Project files can use various options using conditional clauses such as different syntax and IF statements. For example, if the 32-bit version is a 64-bit version, you might use different compilation options. Next is the main.cpp source code.

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

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

    int myVal = 100;

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

    return a.exec();
}
```

The starting point of Qt in the main() function declares the object of the QCoreApplication class. This object processes requests from a user or event without ending the program through an event loop called Qt. Only use the return to a.exe() member function on the last line of the main() function to prevent the program from ending processing the main() function.

The argc and argv factors can be handed over from the main function. The qDebug() function is the same as the printf() function used in C language and provides the same

Jesus loves you.

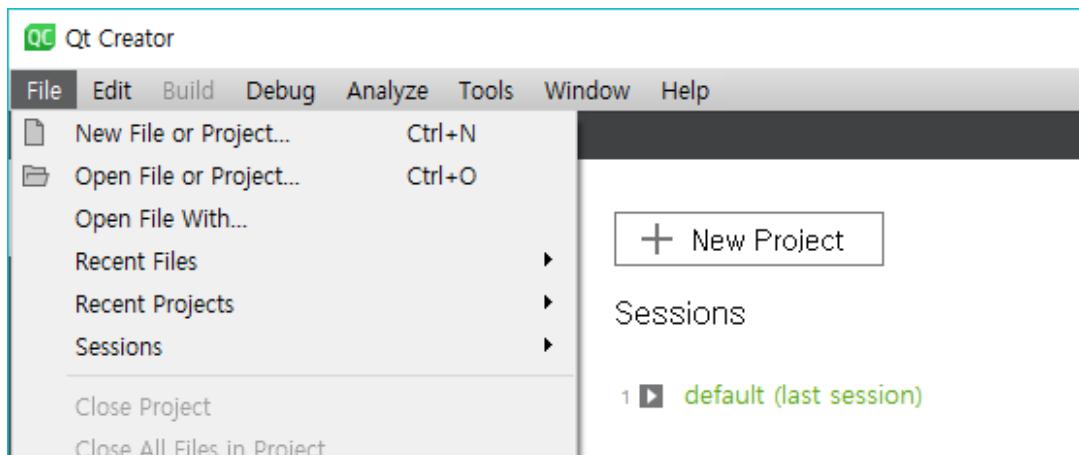
function as the System.out.println() function in Java.

- Programming a GUI-based "Hello World" output example.

In this example, let's implement a simple GUI-based application. This application will place a button widget on the GUI. Enter "Hello World" with the button's TEXT. Click this button to output the "Hello World" string to the debugging window using the qDebug() statement.

Qt can place widgets (e.g. buttons) by dragging them with the mouse using a designer tool built into the Qt Creator IDE tool as a way to develop GUI-based applications.

However, in this example, the purpose of how to develop the Qt development framework is to write the GUI code directly in C++ instead of using the designer tool. Run Qt Creator and create a new project.

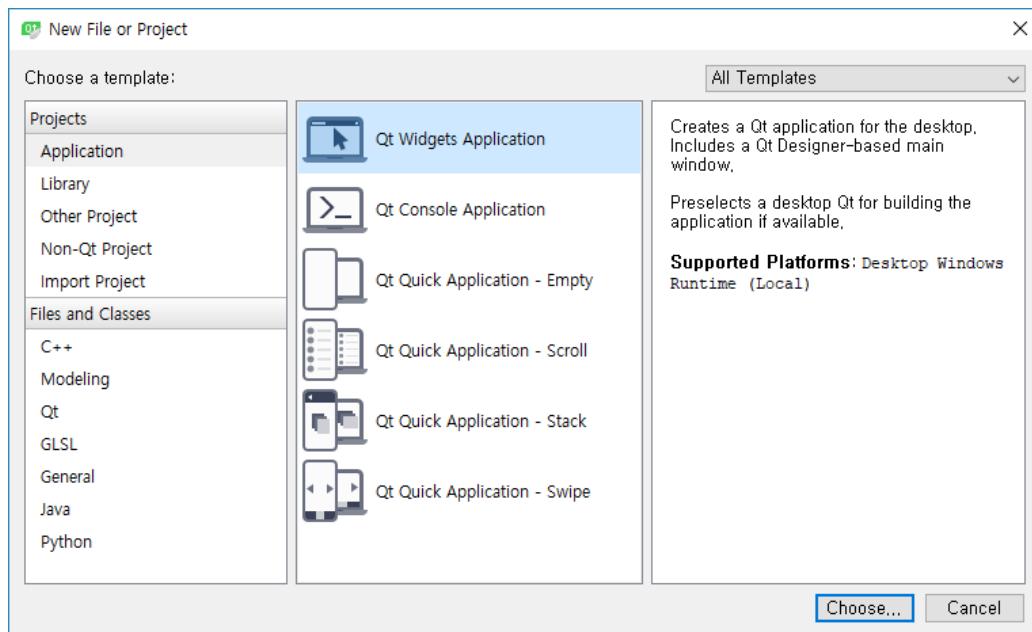


<FIGURE> Create a new project

[file], as you see the picture before - >, click the [new file or project]. Click the [new file or project] menu.

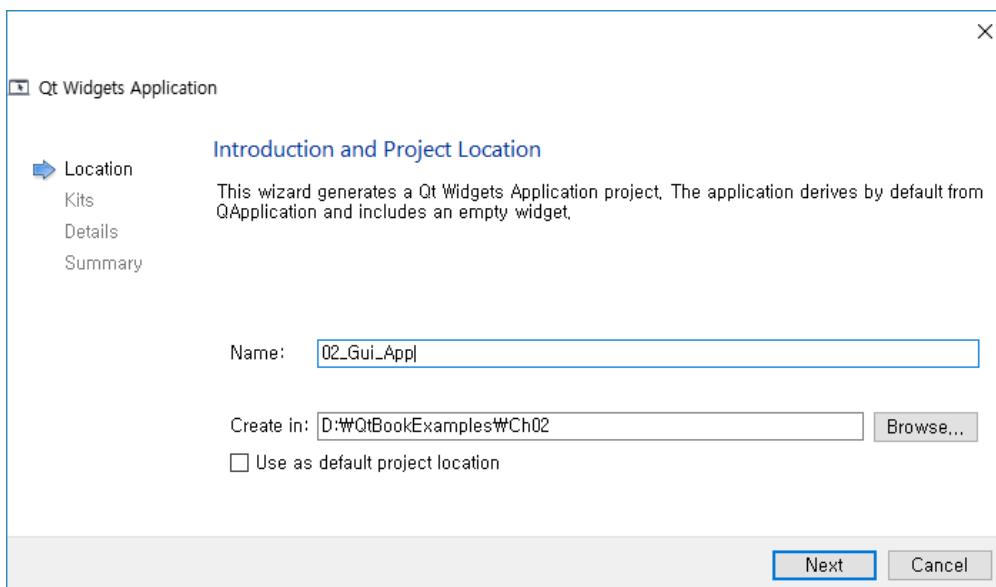
Then select [Application] from the left Project menu list in the dialog window, and select [Qt Widgets Application] from the right menu and click the [Choose] button at the bottom, as shown in the following figure.

Jesus loves you.



<FIGURE> Template Selection Dialog

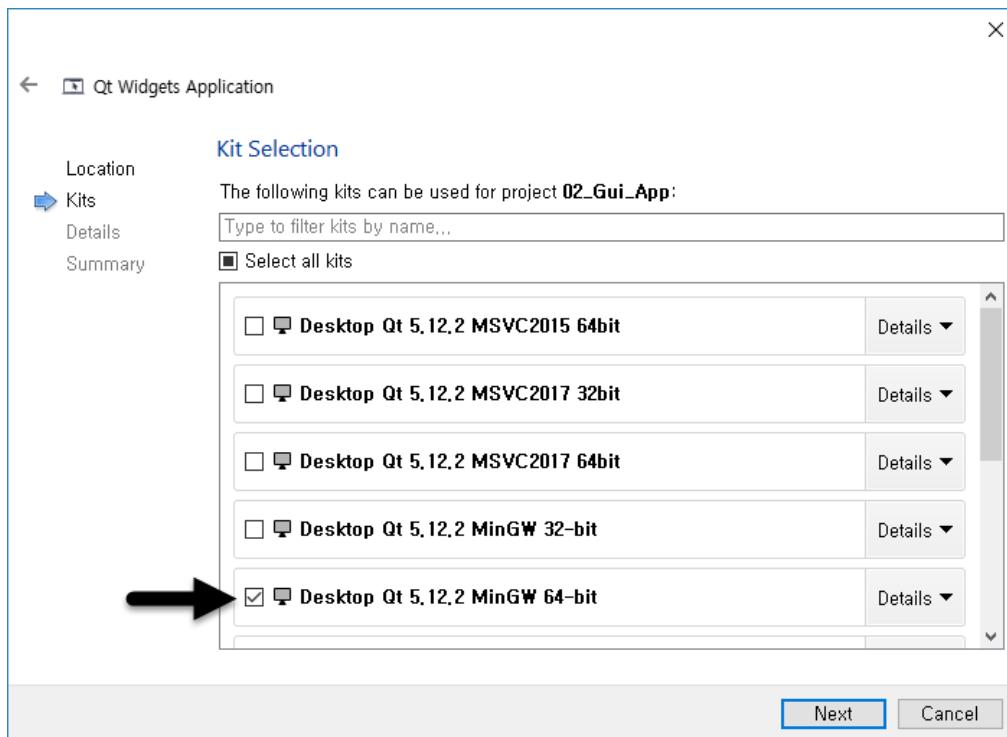
In the next dialogue window, enter the project name (Name) and the location of the directory where the project was created, then click the [Next] button at the bottom.



<FIGURE> Project Name and Location Dialog

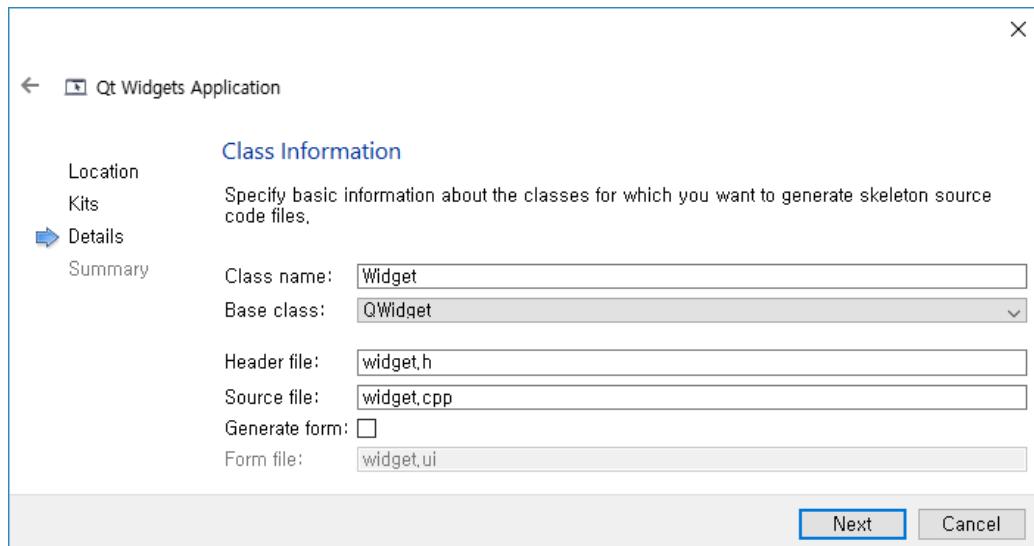
The next dialogue is a dialogue window that selects the compiler to build the application to be developed. Release all check boxes, select the MinGW 64-bit version and click the [Next] button at the bottom, as shown in the following figure.

Jesus loves you.



<FIGURE> Select Compiler to Build Dialog

The following specifies the name of the main class of the project (the first main window widget class to load), the header file name, the source file name, and the file name (.ui) of the design form, asking if you want to assign the design form to the main class.



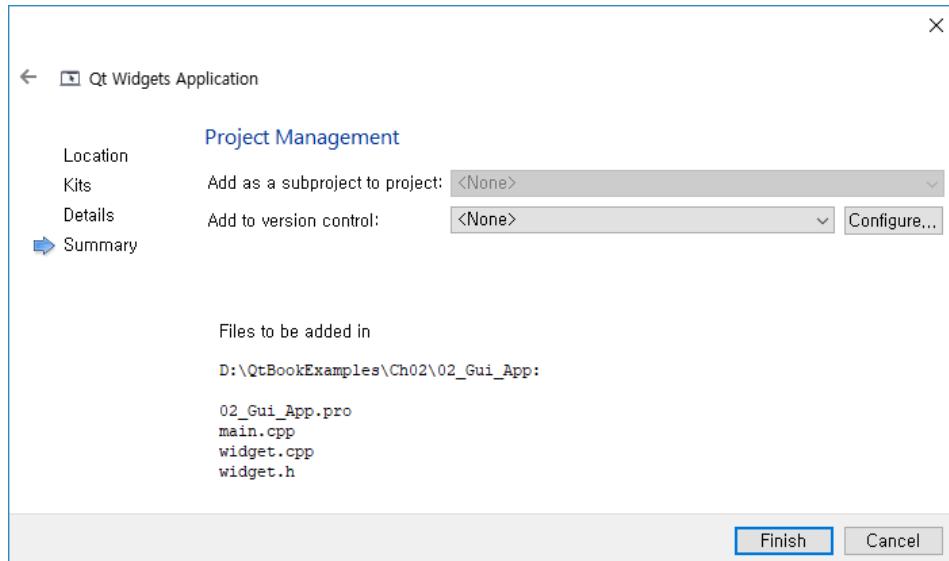
<FIGURE> Class Information Entry Dialog

In the dialogue window, the first [Class Name] specifies the name of the main class being

Jesus loves you.

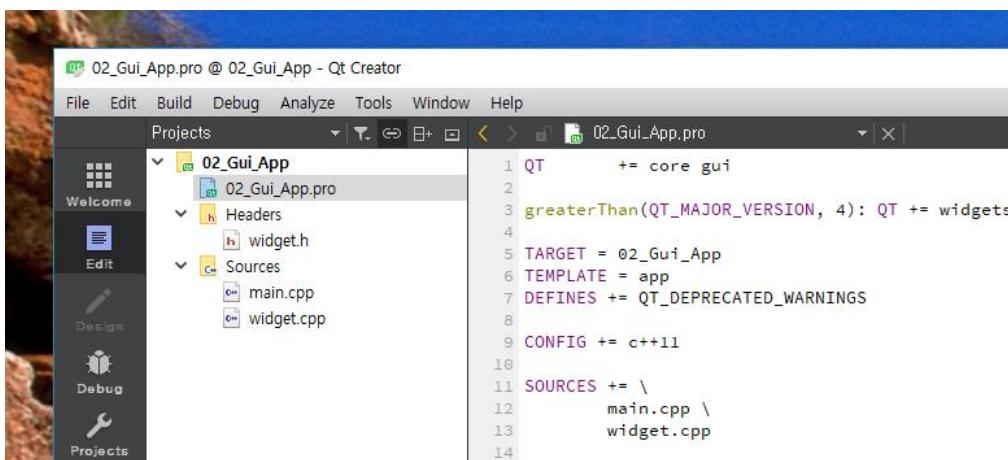
created. [Base class] allows you to select the class that the main class inherits. The type can be chosen from three, as shown in the following table.

Here, the second QWidget is selected as Base Class. The header file name and source file name are named Widget.h and Widget.cpp, and the [Generate form] item unlocks the check box as shown in the previous picture. Then click the [Next] button at the bottom.



<FIGURE> Project Management Dialog

The previous figure is the last generated dialogue of the last project. When the [Finish] button at the bottom is clicked, Qt Creator automatically generates the basic project files(.pro) file, main.cpp, widget.h, and widget.cpp files needed to create the widget.



<FIGURE> Qt Creator screen after completion of project creation

Jesus loves you.

The following is the project file(.pro) source code.

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

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

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

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

The QT macro (keyword) on the first line specifies the modules to be used in this project. The second line means that if the main parent version of Qt used by this project uses a version 4.x.x instead of the 5.x.x version, a module named widgets will be added.

TARGET Macro specifies the name of the project. TEMPLATE selects the same kind as whether this project is a library recognition application. This project is an application, so it specifies app. Using "QT_DEPRECATED_WARNINGS" in the DEFINE macro produces an alert when using APIs missing by Qt version.

The options available in CONFIG macros provide many options. For information on the options offered, you can use the Qt Assistant Help to examine in detail what options are available. The c++11 used here means the C++ version used by Qt. SOURCES and HEADERS Macro specify the source code file and header file for this project. And the last three lines are examples of how a particular platform works.

qnx: target.path = /tmp/\${TARGET}/bin is the line performed on the platform QNX. If the following line is a Linux platform rather than a QNX, the last line means that both QNX and Linux.

Project files can use a variety of options using conditional clauses such as various syntax and Syntax structures such as C For example, if the 32-bit version is a 64-bit version, the

Jesus loves you.

compilation options may vary. Here is the main.cpp example source code.

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

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

    Widget mywidget;
    mywidget.show();

    return a.exec();
}
```

Because this project is GUI-based, use QApplication instead of QCoreApplication. Mywidget object of the generated Widget class. This class is a Windows widget, so it declares the object as you see in the source code. The show() member function provides the ability to display the GUI widget on-screen. The following example is the widget.h source code.

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

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

private:
    QPushButton *btn;
    QString str;

public slots:
    void slot_btn();
};

#endif // WIDGET_H
```

Jesus loves you.

The QPushButton class widget specified in the private keyword is a button widget. As shown in the source code, btn object is declared and QString is a string processing class variable provided by Qt.

Public slots are event functions. In this event, it is called the SLOT function. This Slot function is used to specify which functions are executed when an event occurs. Therefore, in this project, click on the btn named "Hello World" to call the slot_btn() function. The following is an example source code for Widget.cpp.

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

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

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

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

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

Widget::~Widget()
```

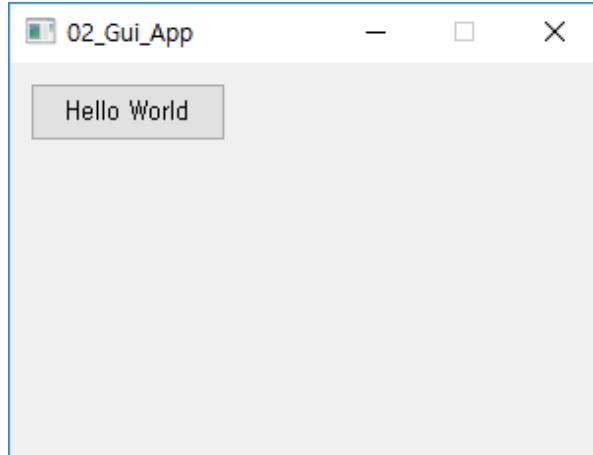
The setFixedSize() function is a member function for specifying the size of this widget. The first factor is the horizontal size. The second factor is vertical. The unit is Pixel.

The str variable of the following line is the QString class variable declared in the header. To save the string "hello world" in this class. Connect () event occurs, click the button function is btn slot btn () calls the function.

The first factor is an event occurring objects and the second is the type of event. The type of event, clicked mouse over, released, etc. The third factor specifies a definition

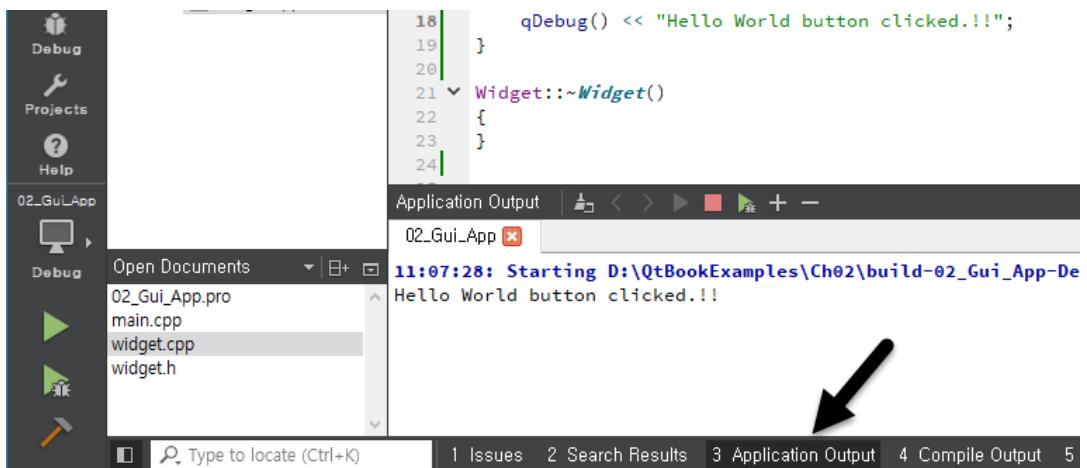
Jesus loves you.

function in the SLOT keyword in the header file of the class that receives the event, the class that receives the event in the fourth event. Events in Qt use Signal and Slot.



<FIGURE> Application Execution Screen

When you build and run an application, the function Slot_btn() is called when you click on the "Hello World" button in the window, as shown in the previous picture. The function uses the qDebug() statement to output the "Hello World button clicked.!!" statement to the debugging window. A string printed using the qDebug() function can be viewed by clicking on the [Application Output] tab at the bottom of the Qt Creator.



<FIGURE> qDebug() Output Screen After Application Execution

3. Qt Basic

In this chapter, let's look at the essential elements required to develop Qt applications. The contents of this chapter are as shown in the following table.

<TABLE> Summary of what will be discussed in this chapter.

Subject	Description
Qt GUI Widgets	Explain how Qt provides GUI widgets
Layout	How to Use Layout for Dynamic Size Transformation of GUI
Qt provides data types and classes	A method for using data types and classes.
Signal and Slot	Process events using Signal and Slot provided by Qt
Design and Fabrication of GUI Using Qt Designer	How to use the Qt Designer tool to place GUI widgets with your mouse using the GUI interface.
Dialog	Dialog message
Implementing Windows GUI based on MDI	Differences between Multi Document Interface and Single Document Interface and How to Implement it
File processing and data stream	How to efficiently handle files and large amounts of data using streams
Qt Property	Inter-Object Communication Based on Meta-Object System
QTimer	Process repetitive events using QTimer
QThread	Thread Implementation
Qt Linguist Tools	Multi-language processing using Qt Linguist tool
Creating libraries	Creating a Library with Qt
Model and View	How to display data in a widget in the form of a table

3.1. Qt GUI Widgets

In this section, let's look at the GUI widgets offered by Qt, as shown in the following figure.

- QCheckBox and QButtonGroup

The QCheckBox class widget provides a GUI interface for selecting multiple items. The first factor is the checkbox entry text. The second factor specifies the parent class.

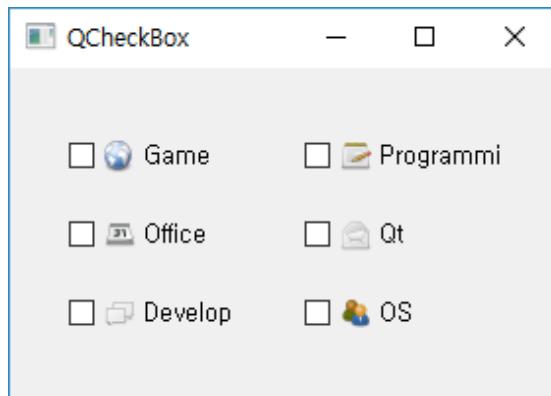
```
QCheckBox *chk = new QCheckBox("P&rogramming", this);
```

For the & characters used in the above "P&rogramming" string, a shortcut to the check box can be used, and the shortcut key can be enabled or disabled by pressing the [Alt + R] key. To separate the checkboxes into groups, use QButtonGroup as shown in the example source code below.

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

The first factor specifies the parent class. You can include check boxes in the group using the addButton() member function of the QButtonGroup class.

Although the check box features multiple selection features, the setExclusive() member function of the QButtonGroup class allows you to change the properties of the check box so that only a single item can be selected.



<FIGURE> QCheckBox Example Screen

Jesus loves you.

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

int xpos = 30;
int ypos = 30;

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

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

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

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

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

This example does not affect the first group because the check boxes registered in the second group and the boxes registered in the second group were separated into groups. Refer to Ch03 > 01_BasicWidget > 01_QCheckBox directory for complete examples.

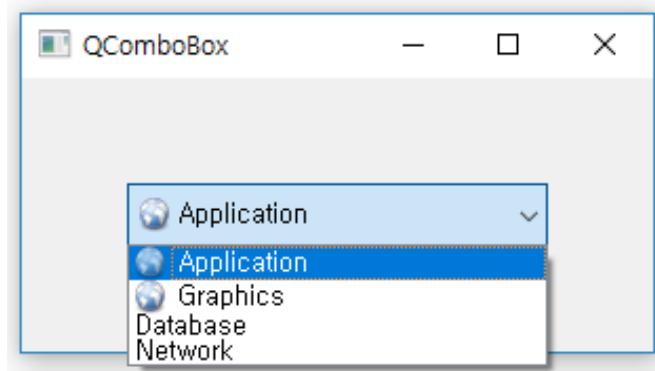
- QComboBox

When a user clicks on a widget, a pop-up menu appears and provides a GUI to select one of the registered items. To register an item on the QComboBox widget, you can use the text or the image in the item to mix the text.

```
combo = new QComboBox(this);
combo->setGeometry(50, 50, 200, 30);
...
combo->addItem(QIcon(":resources/browser.png"), "Application");
combo->addItem(QIcon(":resources/browser.png"), "Graphics");
```

Jesus loves you.

```
combo->addItem("Database");
combo->addItem("Network");
...
```



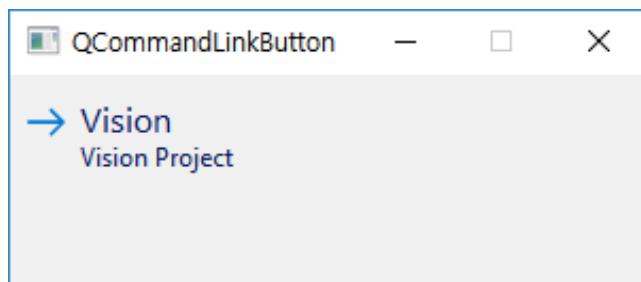
<FIGURE> Example Run Screen

For example, see Ch03 > 01_BasicWidget > 02_QComboBox directory for complete sources.

- **QCommandLinkButton**

This widget is a widget that provides the same functionality as the QPushButton widget. As a feature, the widget provides the same style as Link Button, which is available in MS Windows.

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



<FIGURE> QCommandLinkButton Execution screen

Jesus loves you.

- QDate and QDateEdit

QDateEdit provides a GUI that allows you to display or change dates. QDateEdit can display the date set in the QDate class. A QDate class can specify a year, month, or day, or obtain the current date from the system and display it by connecting it to the QDateEdit widget.

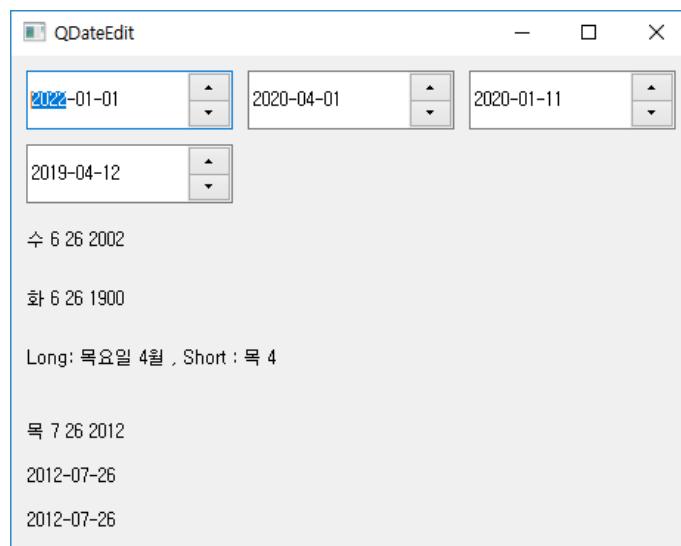
```
QDate dt1 = QDate(2020, 1, 1);
QDate dt2 = QDate::currentDate();

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);
...
```



<FIGURE> QDate and QDateEdit example screen

Jesus loves you.

The QDate class can use the membership function to return the QString data type as a way to display the date in the desired format.

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

<TABLE> Date Display Format

Expression character	Display form
d	1 to 31 mark (1 to 31)
dd	1-31 Marked by 2 digits (01-31)
ddd	Show days in 3-digit characters (Mon to Sun)
dddd	Display the day of the week in full text (Monday through Sunday)
M	1 to 12 numerals (1 to 12)
MM	01 to 12 numerals, 2 digits (01 to 12)
MMM	Mark the month in 3 digits (Jan to Dec)
MMMM	Mark the month as a complete letter (January through December)
yy	Mark the Year in 2 digits (00–99)
yyyy	Mark the Year in 4 digits (2002)

<TABLE> Change the QDate display style

12	Value	설명
Qt::TextDate	0	Show Default Type
Qt::ISODate	1	ISO8601 Displayed in Extended Format
Qt::SystemLocaleDate	2	Display in a national manner

The QDate class can change the style that displays the date to the following types:

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

Jesus loves you.

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

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

...
```

For example, see Ch03 > 01_BasicWidget > 04_QDateEdit directory.

- QTime class and QTimeEdit widget class

The QTime class can easily implement the various time-related functions required to develop an application, such as showing the current system setup time and elapsed time from the current time to a specific time, or comparing time. QTimeEdit is a widget that provides the ability to display the time taken from the QTime class on the GUI interface.

```
QTime ti = QTime(6, 24, 55, 432);

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

...
```

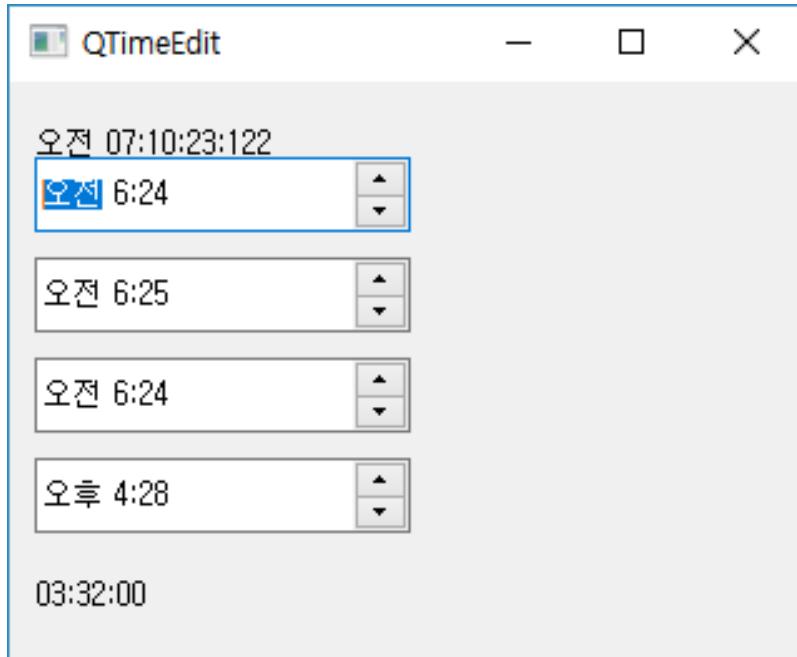
The QTime class provides various member functions to facilitate time-related operation. For example, to add seconds and milliseconds to the current time by the specified value, addSecs() and addMSsecs() member functions can add seconds and milliseconds() at that time to achieve the desired result.

```
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);

...
```

Jesus loves you.



<FIGURE> QTime and QTimeEdit example screen

One of the useful functions of the QTime class can be used as a function that obtains time elapsed from the time the start() member function is declared to the source code line from which it is called.

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

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

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

QTime class can display time in various formats using toString() member function.

```
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);
...
```

Jesus loves you.

<TABLE> Time Display Format

Expression	Time Display Format
h	Mark Hour as 0-23 Numeric
hh	Mark Hour as 00-23 Numeric (Always 2 digits)
m	Mark Minute 0-59
mm	Mark Minute as 00-59
s	Mark Second as 0-59
ss	Mark Second as 00-59
z	Mark Milli seconds as between 0 and 999
zzz	Milli seconds marked 000 to 999
AP	Capitalize AM/PM characters showing morning/afternoon
ap	AM/PM characters showing morning/afternoon in lowercase

For example, see the Ch03 > 01_BasicWidget > 05_QTimeEdit directory.

- QDateTime and QDateTimeEdit

The QDateTime class is a class that can handle both date and time, and the QDateTimeEdit class provides a GUI that can display date and time. The setDisplayFormat() member function of the QDateTimeEdit class can display the date and time depending on the format.

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

The QDateTimeEdit can change the date and time displayed in the widget to a spin box button and specify a range when changing the date and time.

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

Jesus loves you.

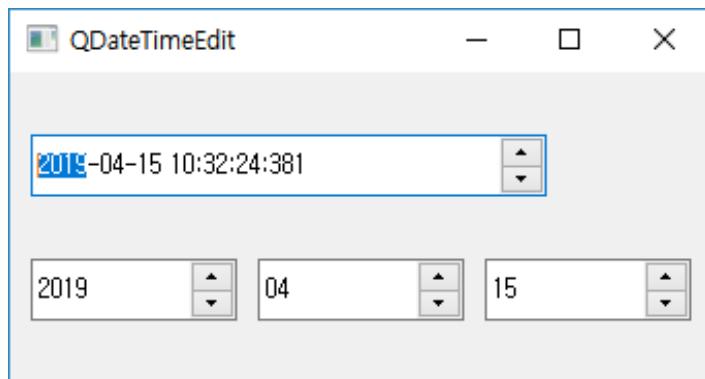
```
qde[0]->setDisplayFormat("yyyy");
qde[0]->setGeometry(10, 90, 100, 30);

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);

...
```

The `setMinimumDate()` and `setMaximumDate()` member functions can only change the date within a set range by specifying a range.



<FIGURE> QDateTime and QDateTimeEdit example screen

In addition to the date, the QDateTimeEdit class can specify the minimum and maximum values of time using the `setMinimumTime()` and `setMaximumTime()` member functions. For example, see Ch03 > 01_BasicWidget > 06_QDateTimeEdit directory.

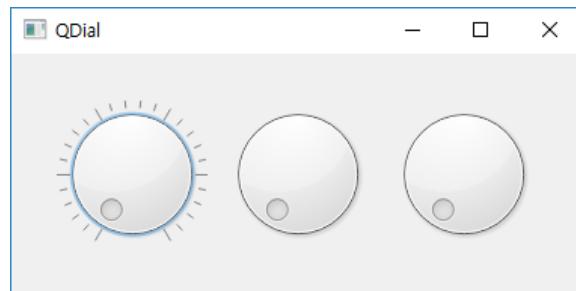
- QDial

The QDial widget class provides a dial-like GUI interface. For example, when adjusting a volume, it provides a GUI such as turning the dial to adjust it.

```
for(int i = 0 ; i < 3 ; i++, xpos += 110) {
    dial[i] = new QDial(this);
```

Jesus loves you.

```
dial[i]->setRange(0, 100);
dial[i]->setGeometry(xpos, 30, 100, 100);
}
dial[0]->setNotchesVisible(true);
connect(dial[0], &QDial::valueChanged, this, &Widget::changedData);
...
```



<FIGURE> QDial Example Run Screen

The `setRange()` member count can specify a range of QDial widgets. The `setNotchesVisible()` member count provides the ability to display scales in the QDial widget. Drag the QDial widget with your mouse to register a `valueChanged()` signal to obtain the current value of the changed QDial.

```
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());
}
```

When the value of the first dialogue in QDial changes, the `changeData()` Slot function is

Jesus loves you.

called. You have not yet learned how to handle events using Signal / Slot. So let's just understand that the function that handles events that occurred using the connect function is a Slot function. More details will be covered in the Signal and Slot sections. For example, see Ch03 > 01_BasicWidget > 07_QDail directory.

- QSpinBox and QDoubleSpinBox

The QSpinBox class provides a GUI that allows integer values of int-type data types to be changed using the up and down buttons. To use the double data type, you can use the QDoubleSpinBox widget.

The QSpinBox and QDoubleSpinBox widget classes can limit the range of values that users can change, and can use characters that point to specific characters or units in the Prefix and Suffix sections in which numbers are displayed. For example, currency symbols can be used within the widget.

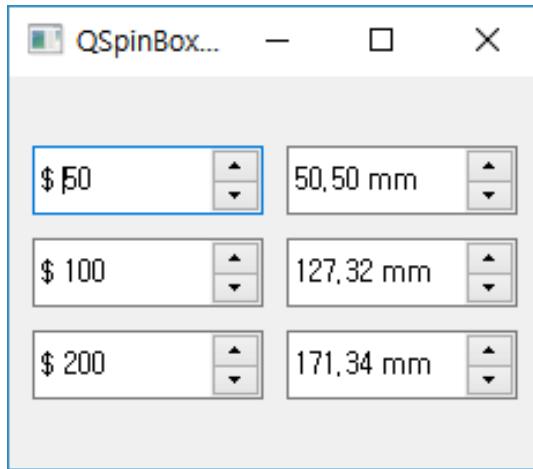
```
...
int ypos = 30;
int val[3] = {50, 100, 200};
double double_val[3] = {50.5, 127.32, 171.342};

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

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

    spin[i]->setPrefix("$ ");
    doublespin[i]->setSuffix(" mm");
    ypos += 40;
}
...
```

Jesus loves you.



<FIGURE> QSpinBox and QDoubleSpinBox example screen

For example, see Ch03 > 01_BasicWidget > 08_QSpinBox_QDoubleSpinBox directory for the complete source.

- QPushButton and QFocusFrame

The QPushButton widget provides button functionality. QFocusFrame is useful if you need to use Outer Line on the outside. In addition, QFocusFrame can use QStyle (tyle sheet used by HTML). Procedure for using QFocusFrame to draw Outer Line on the outside of the QPushButton widget is as follows.

```
QPushButton *btn[3];

int ypos = 30;
for(int i = 0 ; i < 3 ; i++) {
    QString str = QString("Frame's button %1").arg(i);
    btn[i] = new QPushButton(str, this);
    btn[i]->setGeometry(10, ypos, 300, 40);
    ypos += 50;
}

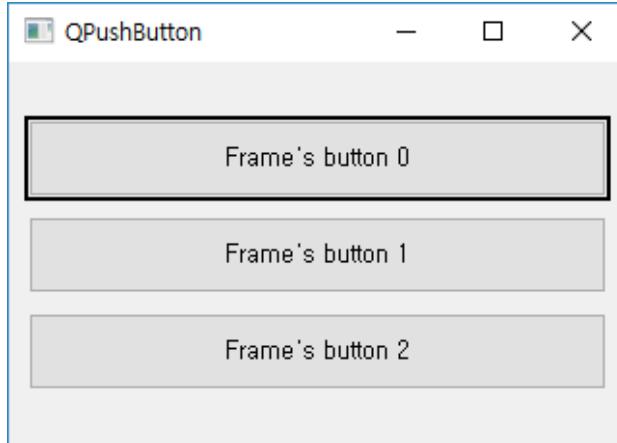
connect(btn[0], &QPushButton::clicked, this, &Widget::btn_click);
connect(btn[0], &QPushButton::pressed, this, &Widget::btn_pressed);
connect(btn[0], &QPushButton::released, this, &Widget::btn_released);

QFocusFrame *btn_frame = new QFocusFrame(this);
```

Jesus loves you.

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

The first factor in the QPushButton class constructor enters the text to display on the button. The second factor specifies the parent class of the QPushButton class.



<FIGURE> QPushButton and QFocusFrame example screen

For example, see Ch03 > 01_BasicWidget > 09_QPushButton_QFocusFrame directory.

- QFontComboBox

The QFontComboBox widget provides a GUI for selecting fonts on the GUI. This widget lists the fonts in alphabetical order and allows you to see how they look.

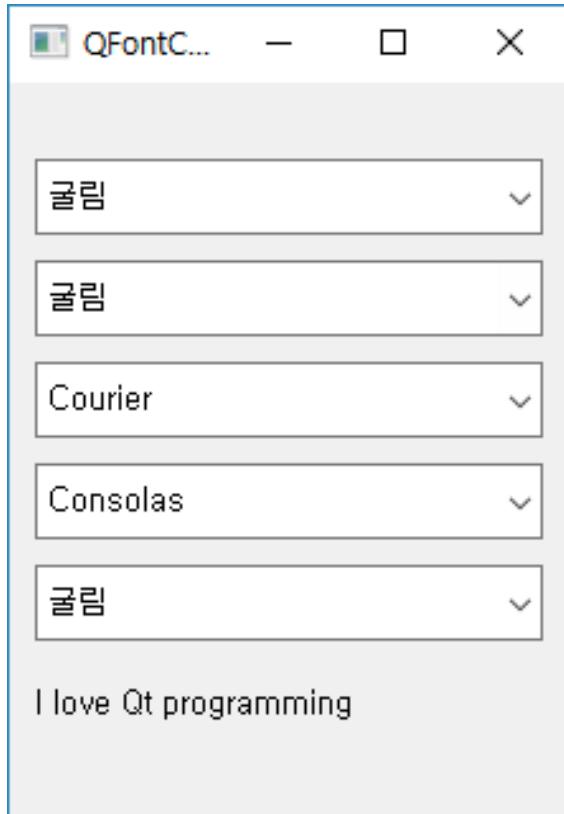
```
QFontComboBox *fontcb[5];
for(int i = 0 ; i < 5 ; i++)
    fontcb[i] = new QFontComboBox(this);

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

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

Jesus loves you.

```
ypos += 40;  
}  
...
```



setFontFilters() provides a constant display of a list of fonts to be listed on the QFontComboBox widget to display a specific font using the Filtering function.

See Ch03 > 01_BasicWidget > 10_QFontComboBox directory for sample sources.

<FIGURE> Example screen

<TABLE> constants available in setFontFilters() member functions

상수	값	설명
QFontComboBox::AllFonts	0	All fonts
QFontComboBox::ScalableFonts	0x1	Zoom-in/zoom dynamic convertable font
QFontComboBox::NonScalableFonts	0x2	Font without Dynamic Transformation
QFontComboBox::MonospacedFonts	0x3	Font that provides a constant character width form
QFontComboBox::ProportionalFonts	0x4	a font with a balance between width and height.

Jesus loves you.

- **QLabel** and **QLCDNumber**

The **QLabel** widget provides the ability to display text or images on applications. The **QLCDNumber** widget can only display numbers and display numbers in the same form as a digital clock. The **QLCDNumber** widget can be used together with the " :" characters used to display time.

```
...
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);

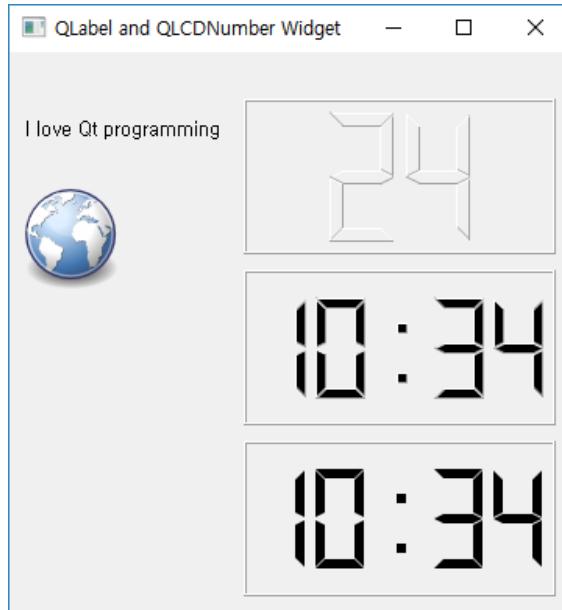
QLCDNumber *lcd[3];
lcd[0] = new QLCDNumber(2, this);
lcd[0]->display(24);
lcd[0]->setGeometry(150, 30, 200, 100);
lcd[0]->setSegmentStyle(QLCDNumber::Outline);

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);
...
```

The **QPixmap** class is an API provided for rendering images on the GUI. Using the **QPixmap** class, images can be displayed on the GUI by using the **setPixmap()** member function of the **QLabel** widget.

Jesus loves you.



<FIGURE> QLabel and QLCDNumber example screen

For example, see Ch03 > 01_BasicWidget > 11_QLabel_QLCDNumber directory for the complete source.

- QLineEdit

The QLineEdit widget provides a GUI for entering and modifying text. Provides functions such as copying, pasting, and cropping on the QLineEdit class widget.

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

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

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

    int ypos = 120;
    for(int i = 1 ; i < 5 ; i++) {
        edit[i] = new QLineEdit("I love qt.", this);
        edit[i]->setGeometry(10, ypos, 200, 40);
    }
}
```

Jesus loves you.

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



<FIGURE> QLineEdit example screen

QLineEdit allows text to be invisible or password processed. For this treatment, the

Jesus loves you.

following constants should be used as factors in the setEchoMode() member function.

<TABLE> constants available in setEchoMode () member functions

Contant	Value	Description
QLineEdit::Normal	0	Same style as default
QLineEdit::NoEcho	1	Style that does not see text and does not change the cursor's position
QLineEdit::Password	2	Text appears as "*"character
QLineEdit::PasswordEchoOnEdit	3	Same default style when text changes, but show "*" when focus is moved

For example, see the directory Ch03 > 01_BasicWidget > 12_QLineEdit.

● QMenu and QMenuBar

The QMenu and QMenuBar class widgets provide menu functions. The QMenu widget provides addAction() and addMenu() member functions as widgets that you provide to create menus. addAction() is used to connect functions to run immediately without submenus, and addMenu() is a member function used to connect submenus. You can connect a widget created with QMenu with QMenuBar

```
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]);
```

Jesus loves you.

```
menu[1]->addAction(act[1]);

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

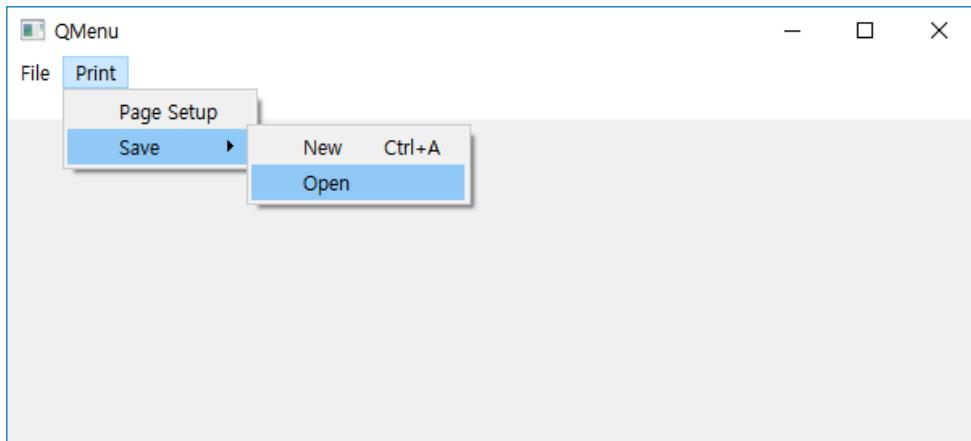
menuBar->addMenu(menu[0]);
menuBar->addMenu(menu[2]);

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

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

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

The QMenu class widget is the generated "File", "Save", and "Print" parent menus. And the menu created with QAction classes is a class widget for adding as a submenu.



<FIGURE> Menu Example Run Screen

For example, see the directory Ch03 > 01_BasicWidget > 13_QMenu_QMenuBar.

Jesus loves you.

- QProgressBar

The QProgressBar widget is available as a widget to display progress, and the widget can be oriented horizontally or vertically. The QProgressBar widget can change its progress from left to right and from right to left when placed horizontally. Conversely, if placed in a longitudinal direction, it may indicate the direction of progress from bottom to top to bottom.

```
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);
...
```

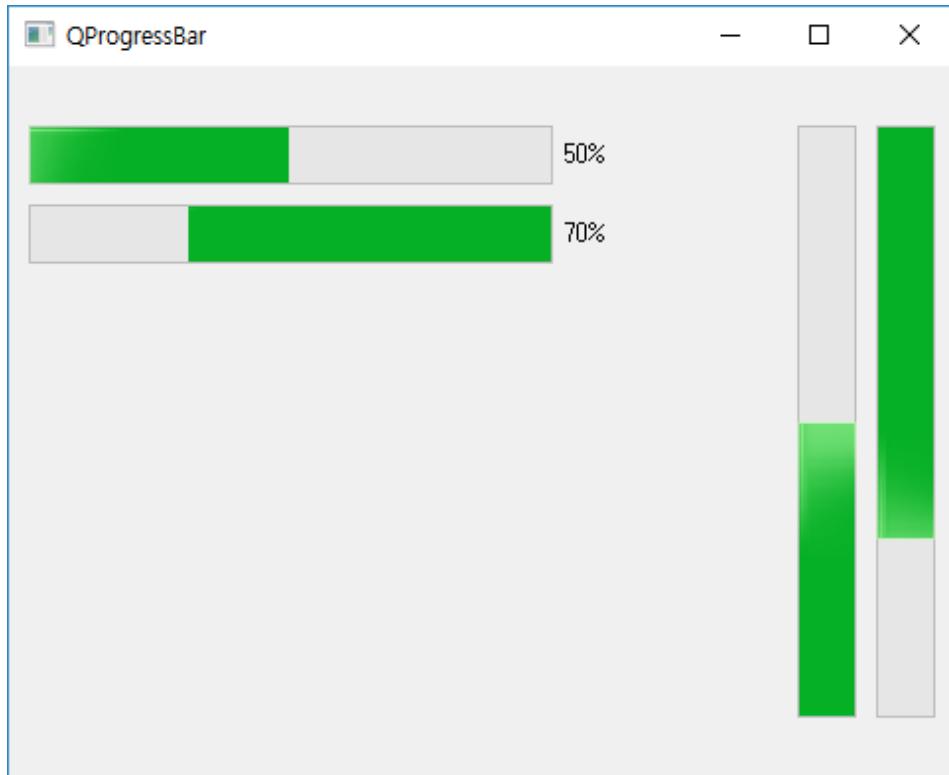
The setMinimum() and setMaximum() functions can be used to set the minimum and

Jesus loves you.

maximum values of QProgressBar and provide the setRange() member function with the same function.

setValue() is a member function used to set progress values between the minimum and maximum values of QProgressBar and the setOrientation() function can be displayed horizontally or vertically.

The setInvergedAppliance() member function is a member function for setting the direction of progress. Setting true as a factor can be reversed in the opposite direction of the default setting.



<FIGURE> QProgressBar example run screen

For example, see Ch03 > 01_BasicWidget > 14_QProgressBar directory.

- **QRadioButton**

The QRadioButton widget provides a GUI for users to select one of several items. For example, On or Off can be selected.

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

Jesus loves you.

```
QRadioButton *radio2[3];

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

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

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

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

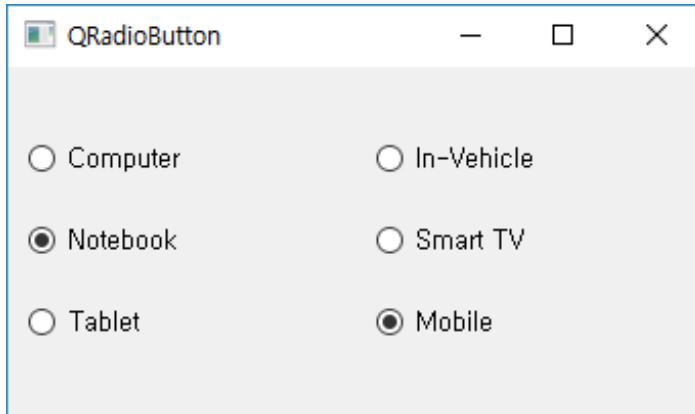
    radio2[i]->setGeometry(180, ypos, 150, 30);
    ypos += 40;
}

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

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

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

Jesus loves you.



<FIGURE> QRadioButton example screen

You can separate from the right QRadioButton by grouping the three left items in six categories, as shown in the example execution screen on the left. Refer to the Ch03 > 01_BasicWidget > 15_QRadioButton directory for the source of this project.

- QScrollArea

The QScrollArea widget provides a GUI function that moves to the hidden part using the scroll bar if it cannot be displayed on both the screen or on the GUI widget. For example, if you want to display an image on the screen without shrinking it, a lack of size of the GUI widget creates a scroll to the left or bottom, providing the same functionality that allows you to scroll through the image with the mouse.

```
QImage image;
QScrollArea *area;

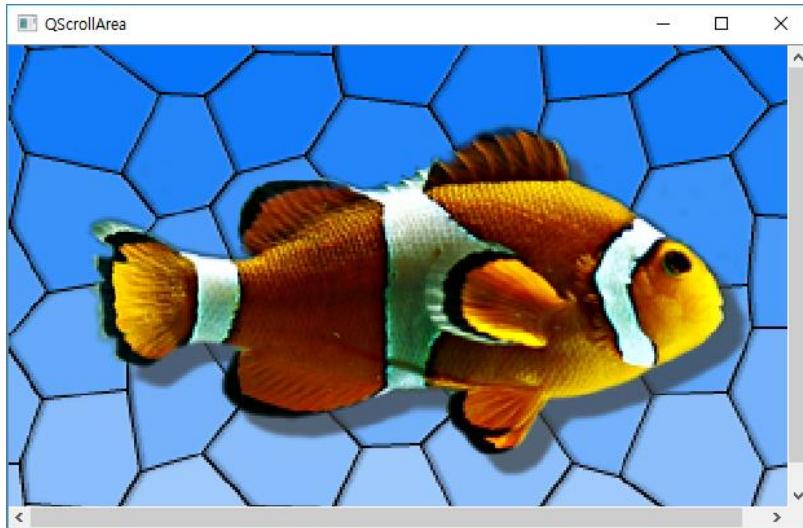
QLabel *lbl = new QLabel(this);
image = QImage(":resources/fish.png");
lbl->setPixmap(QPixmap::fromImage(image));

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

Use the QImage class to render images on the QLabel widget using the setPixmap() member function. The setWidget() member function of the QScrollArea widget allows

Jesus loves you.

you to include the QLabel widget in the QScrollArea widget area. If the image is larger than the QScrollArea widget, the scroll bar is automatically activated.



<FIGURE> QScrollArea Example Run Screen

For example, see Ch03 > 01_BasicWidget > 16_QScrollArea directory.

- **QScrollBar**

QScrollBar is similar to the shape of the slider widget. The QScrollBar widget provides the ability to position left, right, or right, or up and down, in a longitudinal or transversely.

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

Jesus loves you.

...

Declares objects from the QScrollBar and QLabel classes, as seen in the previous source code. And drag the scale from vscrollbar[0] to vscrollbar[2], which is an object in the longitudinal QScrollBar class, to display the changed value in the QLabel widget. Also, change the value of vscrollbar[0] to vscrollbar[2] and the value of each hscrollbar.

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

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

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

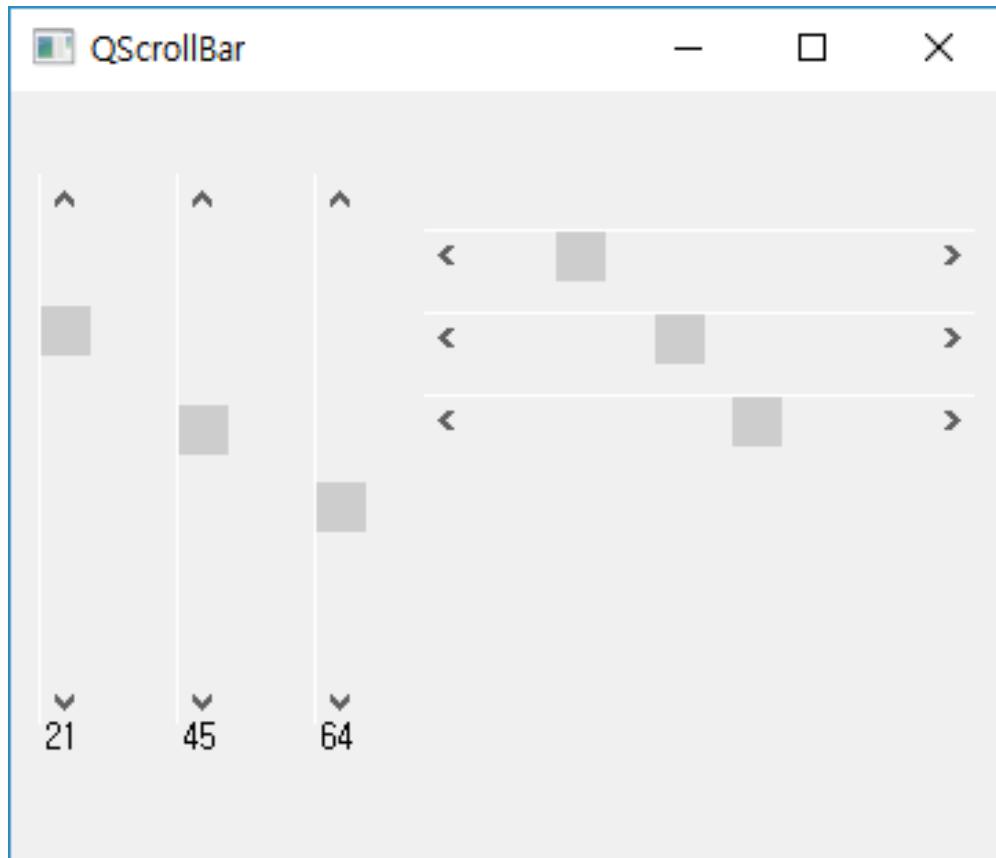
void Widget::valueChanged1(int value)
{
    lbl[0]->setText(QString("%1").arg(value));
    hscrollbar[0]->setValue(vscrollbar[0]->value());
}

void Widget::valueChanged2(int value)
```

Jesus loves you.

```
{  
    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());  
}  
...
```

The connect() function calls each Slot function when the value of vscrollbar[0] to vscrollbar[2] changes. For example, the valueChanged2() Slot function is called when the first division of vscrollbar[1] changes the value to the mouse drag. This Slot function changes the value of QLabel to the value received as a factor for the Slot function. Then change the value of hscrollbar[1].



<FIGURE> QScrollBar Example Run Screen

Jesus loves you.

For example, see Ch03 > 01_BasicWidget > 17_QScrollBar directory for the

- QSizeGrip

The QSizeGrip class widget can be resized within a window area of a limited size. For example, QSizeGrip can implement a GUI such as Splitter, just as you can drag the boundaries of a tree area with a mouse over a GUI showing file and directory properties on the left side, and then reduce or increase them. The next source code is the `widget.h` file source code.

```
...
class SubWindow : public QWidget
{
    Q_OBJECT

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

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

        layout()->addWidget(new QTextEdit);

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

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

class Widget : public QWidget
{
    Q_OBJECT

public:
```

Jesus loves you.

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

};

...
```

As shown in the example source code above, the Widget class is declared a parent class (window class) and the SubWindow class is a child window widget of the Widget window. On the source code, QVBoxLayout can place widgets using a vertical layout.

Layouts are covered in the following sections. Let's just know that we're declaring the Sub Window layout here.

The QTextEdit widget is a widget that is provided for editing text, such as Notepad. The QTextEdit widget can be set to the full area size of the SubWindow area and dynamically resized when changing the sub window size. The next source code is the source code of the main.cpp.

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

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

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

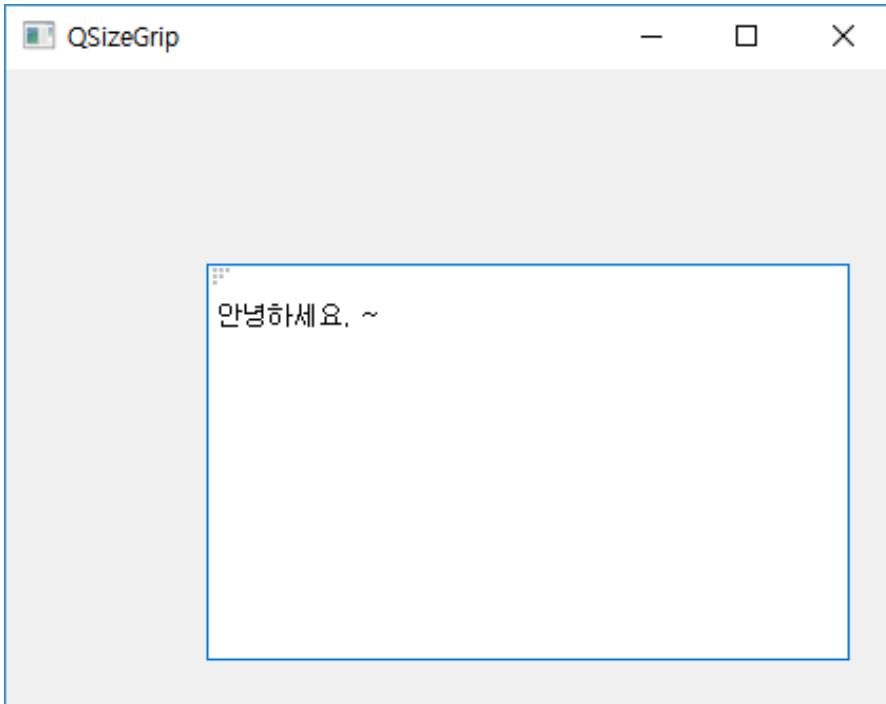
    SubWindow subWindow(&w);
    subWindow.move(200, 180);

    w.show();

    return a.exec();
}
```

In the example above a size total area of the area and Guy A widget qtextedit subwindow to. Dynamically change the size when to change the size of the window sub qtextedit.

Jesus loves you.



<FIGURE> QSizeGrip Example Run Screen

For example, see Ch03 > 01_BasicWidget > 18_QSizeGrip directory for complete sources.

- QSlider

The QSlider widget provides a GUI that allows you to change the minimum and maximum values within a specified range. Similar to QScrollBar. You can set the minimum and maximum values using the `setMinimum()` and `setMaximum()` member functions. You can set the Minimum and Maximum values for the QSlider widget by using the `setRange()` member function to set the minimum and maximum values.

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

Jesus loves you.

```
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());
}

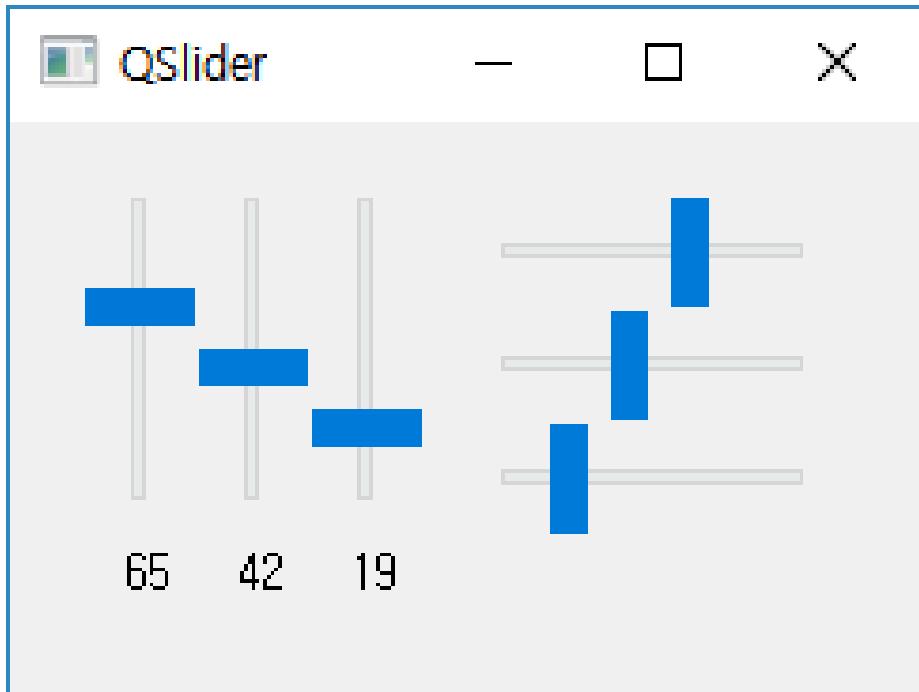
void Widget::valueChanged2(int value)
{
    lbl[1]->setText(QString("%1").arg(value));
    slider[4]->setValue(slider[1]->value());
}

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

Jesus loves you.

The QSlider widget provides a GUI in horizontal and vertical directions, and the first factor of the constructor function of the QSlider can be changed in the widget's orientation using the Qt::Vertical (vertical) or Qt::Horizontal (horizontal) constants.

The QLabel widget shows the value at which the scale of the QSlider is located in a vertical direction. In addition, when the tick position of the QSlider changes, a signal event occurs to change the text of the QLabel widget to the current value of the QSlider.



<FIGURE> QSlider example screen

Example : Ch03 > 01_BasicWidget > 19_QSlider

- QTabWidget

Tabs are useful for deploying many widgets, or when windows are limited in size. QTabWidget provides the ability to dynamically move pages if all tabs cannot be displayed in a limited size.

You can use the addTab() member function to place the widget declared QWidget on the tab instance widget. The way to place widgets on each tab is to specify the parent class as a tab widget, as you specify the parent class as a factor to place the widget, so that the widget can place within that tab.

Jesus loves you.

```
...
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);

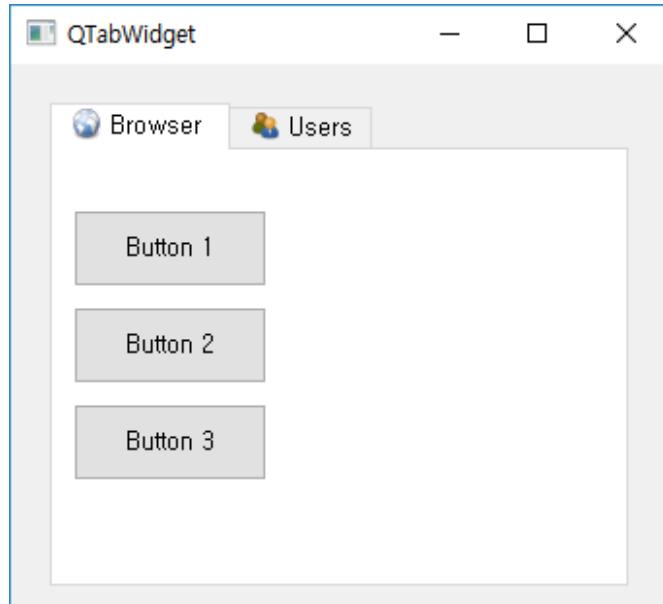
    QStringList btn_str[3] = {"Button 1", "Button 2", "Button 3"};
    QPushButton *btn[3];

    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);
}
...
```

Jesus loves you.



<FIGURE> QTabWidget Example Run Screen

Example - Ch03 > 01_BasicWidget > 20_QTabWidget

- QToolBar and QAction

Qtoolbar widget provides a window as in the tulmenu gui. A widget qtoolbar addaction the () toolbar using the member functions can be placed on the menu.

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

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

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

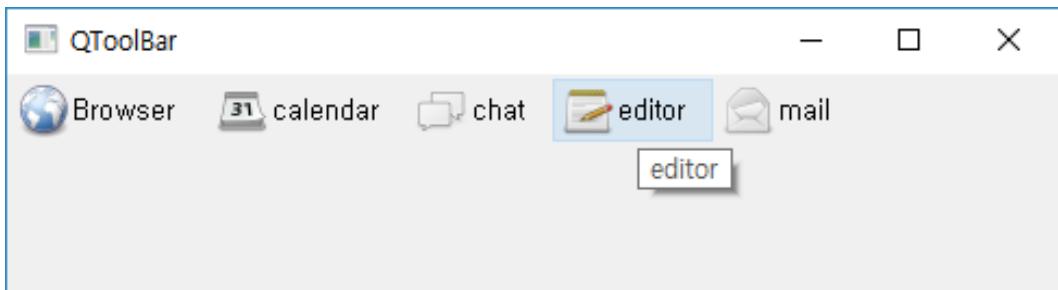
toolbar->setToolButtonStyle(Qt::ToolButtonTextBesideIcon);

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

Jesus loves you.

```
}
```

```
...
```



<FIGURE> QToolBar example screen

Provide the following constant values so that only the icons on the QAction menu buttons can be visible, as shown in the previous figure, or that the icons and button names can be displayed together.

<TABLE> Constant available in setToolButtonStyle() member function

Constant	value	Explain
Qt::ToolButtonIconOnly	0	Show Icons Only
Qt::ToolButtonTextOnly	1	Show Button Name Only
Qt::ToolButtonTextBesideIcon	2	Show icon inside text
Qt::ToolButtonTextUnderIcon	3	Show icon below text

Example - Ch03 > 01_BasicWidget > 21_QToolBar

- QWidget

The widgets we have looked at so far are those that have been inherited and implemented by QWidget. For example, events received from a mouse, keyboard, or window are available in widgets such as QPushButton because they are inherited from QWidget.

The QWidget class provides the ability to render text, shapes (lines, circles, rectangles, etc.) and images to widget areas using the paintEvent() virtual function. You can also

Jesus loves you.

invoke the paintEvent () function by calling the update() member function of the QWidget class. For example, if you click a button to use the update() member function within the called Slot function, the printEvent() virtual function is called.

The QWidget class provides resizeEvent() virtual function. This virtual function is invoked when the size of the QWidget changes. It provides various virtual functions, such as processing mouse events within the QWidget area, keyboard events, and whether activations are Focus in the area of the widget.

```
#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);
};
```

The following source code is the widget.cpp source code and the source code that implements each virtual function.

```
...
```

Jesus loves you.

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

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

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

void Widget::resizeEvent(QResizeEvent *event)
{
    Q_UNUSED(event);
    qDebug("[Resize Event call]");
    qDebug("width: %d, height: %d", this->width(), this->height());
}

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

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

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

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

Jesus loves you.

```
Q_UNUSED(event);
qDebug("[Mouse Double Click] x, y : %d , %d ", event->x(), event->y());
}

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

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

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

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

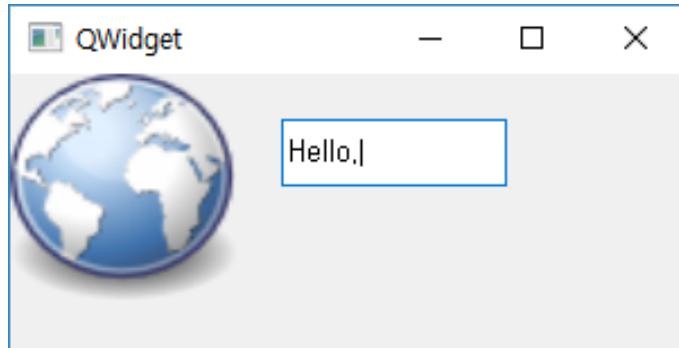
void Widget::keyReleaseEvent(QKeyEvent *event)
{
    Q_UNUSED(event);
    qDebug("Key Release Event.");
}

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

void Widget::focusOutEvent(QFocusEvent *event)
```

Jesus loves you.

```
{  
    Q_UNUSED(event);  
    qDebug("focusOutEvent Event.");  
}  
...
```



<FIGURE> QWidget example screen

Example - Ch03 > 01_BasicWidget > 22_QWidget

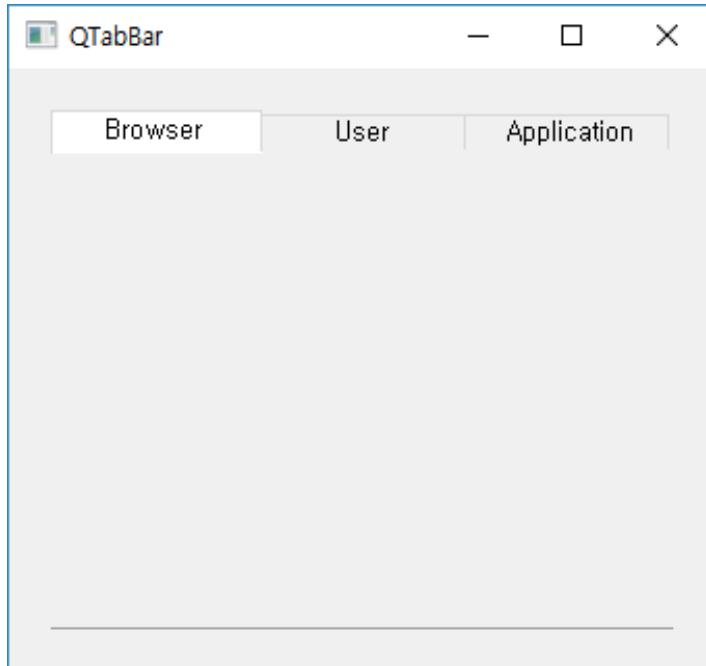
- QTabBar

QTabBar 클래스 위젯은 탭 GUI를 제공한다. 이 위젯은 QTabWidget 과 비슷한 기능을 제공한다. 탭에 아이콘을 사용 하기 위해서 setTabIcon() 함수를 사용하면 된다. 각 탭에 표시되는 텍스트가 구별되게 컬러 지정이 가능하다. 탭의 텍스트를 지정하기 위해 setTabTextColor() 함수를 사용하면 된다.

```
...  
Widget::Widget(QWidget *parent) : QWidget(parent)  
{  
    QTabBar *tab = new QTabBar(this);  
    tab->addTab("Browser");  
    tab->addTab("User");  
    tab->addTab("Application");  
    tab->setShape(QTabBar::RoundedNorth);  
    tab->setGeometry(20, 20, 300, 250);  
  
    connect(tab, SIGNAL(currentChanged(int)),  
            this, SLOT(currentTab(int)));  
}  
  
void Widget::currentTab(int index)
```

Jesus loves you.

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



<FIGURE> QTabBar example screen

QTabbar can change the appearance of tabs that have already been defined. For example, a round-shaped tab or a shape with rounded edges only the left corner of the tab, and can be specified using the setShape() function. The following table shows tabbed values defined by ENUM type.

<TABLE> constants available in setShape() member functions

Constants	value	Explain
QTabBar::RoundedNorth	0	default shape
QTabBar::RoundedSouth	1	Round at the bottom of the page
QTabBar::RoundedWest	2	The left side of the page is round.
QTabBar::RoundedEast	3	Round Right-hand side of page
QTabBar::TriangularNorth	4	triangular shape

Jesus loves you.

QTabBar::TriangularSouth	5	Forms similar to those used in spreadsheets
QTabBar::TriangularWest	6	The left side of the page is triangular
QTabBar::TriangularEast	7	The right side of the page is triangular

Example - Ch03 > 01_BasicWidget > 23_QTabBar

- QToolBox

The QToolbox Class widget provides a GUI for widget items in the form of a new Direction Tab column. Text and icons can be used as the names for each tab.

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

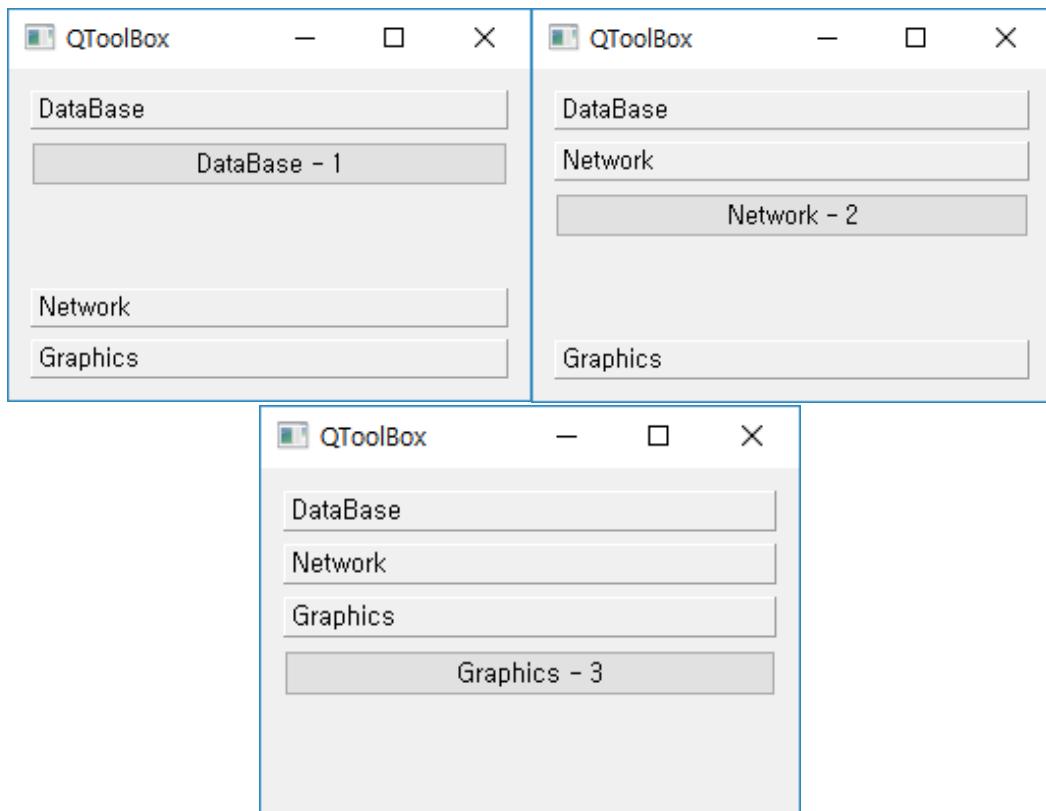
    but1 = new QPushButton("DataBase - 1",this);
    but2 = new QPushButton("Network - 2",this);
    but3 = new QPushButton("Graphics - 3",this);

    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);
}
...
```

Jesus loves you.



<FIGURE> QToolBox example screen

Example - Ch03 > 01_BasicWidget > 24_QToolBox

- **QToolButton**

Qtoolbutton widget that provides features like using text or icon button. The icon of the qtoolbutton QIcon, can be specified using the class. The icon can be displayed active or disabled depending on the status, and buttons are not available in the disabled state.

Settoolbarstyle, style can be changed using the member functions () seticonsize the size of an icon, using a () function can be specified.

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

    QToolButton *button = new QToolButton;
```

Jesus loves you.

```
button->setIcon(QIcon(":resources/browser.png"));

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

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

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

this->setLayout(layout);
}

...
```

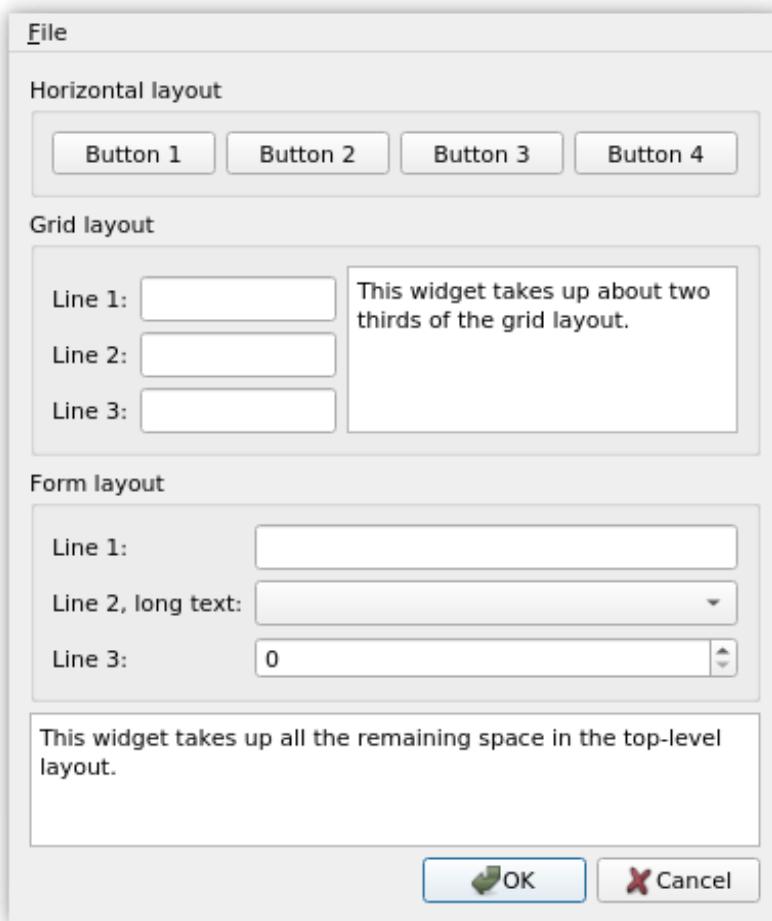


<FIGURE> QToolButton example screen

Example - Ch03 > 01_BasicWidget > 25_QToolBox

3.2. Layout

When you use the `setGeometry()` member function of the `QWidget` class to place the widget in a specific X and Y coordinate on the GUI, the widget's position does not change when the window size changes. However, if you place widgets using the layout as shown in the figure below, the location of the widgets changes dynamically on the GUI as the widgets change in size.



<FIGURE> Example screen for placing widgets using layout

As the size of the window changes, the layout allows the widgets to be arranged in the optimal position to maintain a consistent size appearance. The following table shows the

Jesus loves you.

layout classes used primarily in Qt.

<TABLE> Layouts

클래스	설명
QHBoxLayout	Place widgets horizontally
VBoxLayout	Place widgets in portrait orientation
GridLayout	Place widgets in grid or baduk style
FormLayout	The format in which you place widgets in two columns.

- **QHBoxLayout**

QHBoxLayout allows widgets to be placed horizontally. You can add the QPushButton widget using the addWidget() member function as shown in the following example.

```
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]);
}
```



<FIGURE> QHBoxLayout

- **VBoxLayout**

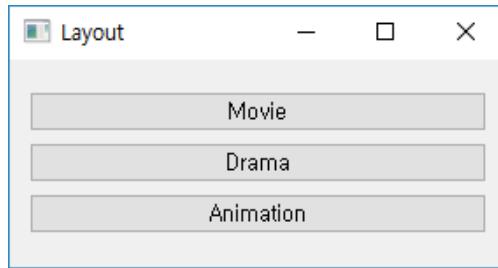
VBoxLayout can place widgets in a longitudinal direction.

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

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

Jesus loves you.

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

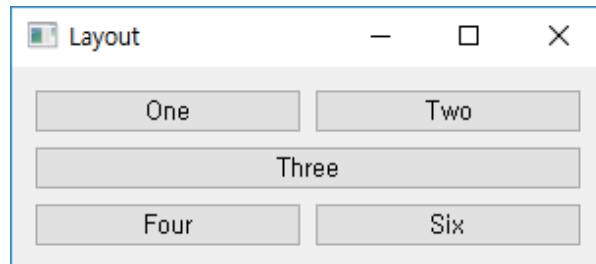


<FIGURE> QVBoxLayout

- QGridLayout

QGridLayout can place widgets in the same style as the grid. QGridLayout can merge a particular row so that only one widget can be placed, or it can be divided into several cells.

```
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);
```



<FIGURE> QGridLayout

Jesus loves you.

- QFormLayout

QFormLayout is a layout class that places widgets in two-column format. MS Windows, Linux GNOME, typically uses left-hand alignment, and MacOS uses right-hand alignment.

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

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

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

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

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

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



<FIGURE> MS Windows and GNOME Style



<FIGURE> Mac OS Style

- Use nested layout

Layouts can be placed within the layout. The following example is an example of an overlay structure.

```
...
```

Jesus loves you.

```
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);

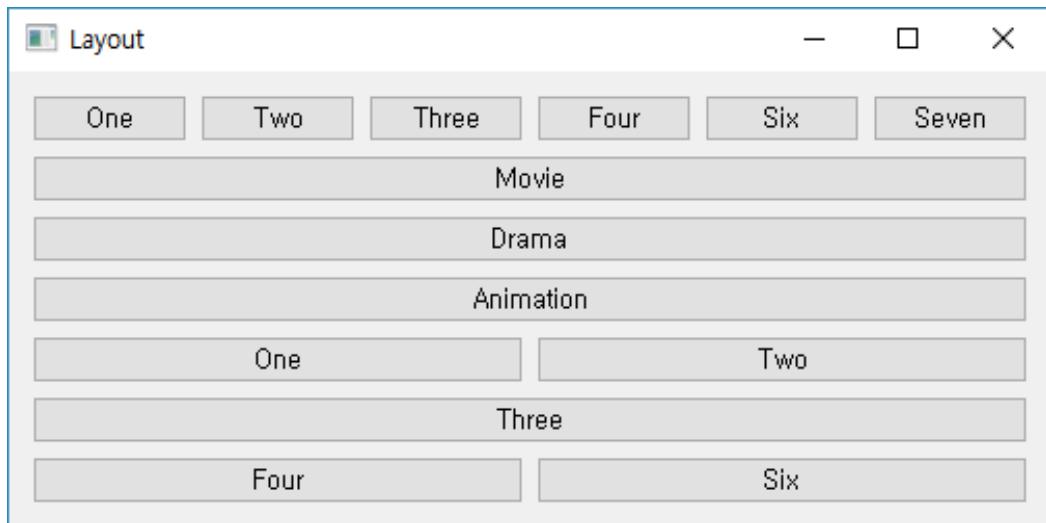
    QVBoxLayout *defaultLayout = new QVBoxLayout();
```

Jesus loves you.

```
defaultLayout->addLayout(hboxLayout);
defaultLayout->addLayout(vboxLayout);
defaultLayout->addLayout(gridLayout);

setLayout(defaultLayout);

}
```



<FIGURE> Overlay layout example screen

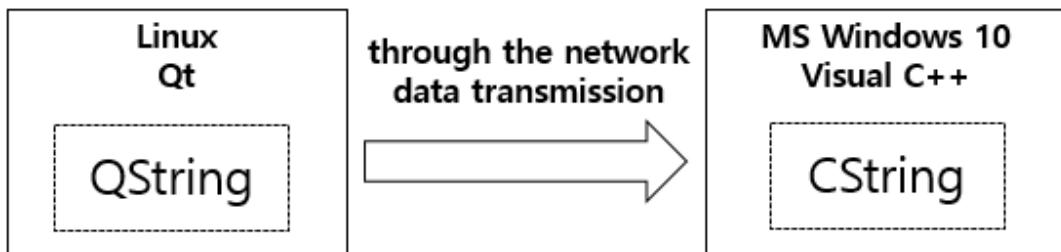
Example - Ch03 > 02_Layout > 01_Layout

3.3. Data type and container classes

Qt provides various data types for the convenience of developers. For example, to find a particular pattern within a string such as `QString`, regular expression expressions are supported, or specific characters are added to or deleted in the string.

Qt also supports data types to address the problem of data type changes when data is exchanged between these species. For example, within an application developed with Qt on the Ubuntu Linux operating system, a string called "Hello World" is sent by Qt, which is provided by Qt.

And if the receiving end of this string is developed with Visual C++ on the MS Windows operating system to store the "Hello World" string, two different data are stored.



<FIGURE> Problems with data exchange between these models

As shown in the figure above, converting and transmitting `QString` to `QByteArray` can be used by the receiving party as `Char` type to solve the problem between heterogeneous data. Qt also supports void-type data types, such as `QVariant` data types. The following table shows the data types used primarily in Qt.

<TABLE> Data types

Type	Size	Explain
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

Jesus loves you.

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	floating point number using IEEE 754 format
double	64 bit	floating point number using IEEE 754 format
const char*	32 bit	Pointer pointing to string constants, excluding zeros at the end

Qt provides a general function and a template function for judging/comparing values of data types. For example, qAbs(), qBound(), qMin(), and qMax() functions to obtain the maximum value, etc.

- ✓ Obtain absolute value

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

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

- ✓ To obtain a value between the minimum and maximum values

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

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

- ✓ Handle error messages

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

Jesus loves you.

```
}
```

- ✓ Function for comparing decimal points

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

- ✓ Obtain the maximum value

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

- ✓ int type rounding

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

- ✓ 64 bit int rounding

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

- String data type class

Besides dealing with simple strings, Qt offers a variety of classes that support Unicode 4.0 (Unicode Standard Version) characters in the form of data streams and multi-byte characters.

Jesus loves you.

<TABLE> Various string data types classes

Classes	Eplain
QByteArray	Classes for supporting array in bytes
QByteArrayMatcher	Class used to find out if there is a matching string using index in a byte array implemented with QByteArray
QChar	Classes to support 16-bit Unicode Character
QLatin1Char QLatin1String	Classes provided to support US-ASCII/Latin-1 encoding strings
QLocale	A class that converts numeric or character presentation to fit the way in which different languages are expressed.
QString	Class that supports Unicode string characters
QStringList	Aggregation class of string list
QStringMatcher	Classes provided to find text matching on a string
QStringRef	Sub-string wrapping classes such as size(), position(), and toString().
QTextStream	STREAM FUNCTIONS FOR TEXT-based WRITE/READ
QDataStream	STREAM FUNCTIONS FOR Binary-based WRITE/READ

✓ **QByteArray**

Qbytearray The class provides an array of byte (8 - bit) unit.

```

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**

A class provided to find a pattern of matching byte array in a byte array.

Jesus loves you.

```
QByteArray x(" hello Qt, nice to meet you.");
QByteArray y("Qt");

QByteArrayMatcher matcher(y);

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

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

✓ QChar

It is a character class to support 16 bit Unicode.

```
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

Classe provided to support US-ASCII/Latin-1 encoding strings.

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

if(str == latin)
    qDebug("Equal.");
else
    qDebug("Not equal.");

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

Jesus loves you.

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

✓ QLocale

Used to convert numbers and characters into various languages.

```
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 class supports Unicode strings and provides the ability to store 16-bit QChar.

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

QString provides a function that can replace string constants such as const char * with a function fromUtf8().

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

Provides the ability to store QChar in a specific location in a stored string.

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

To compare strings, QString can be compared using the if statement.

```
QString str;

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

Jesus loves you.

```
// ...  
}
```

If you want to know the location of a particular string you want to find, you can use the `indexOf()` function to find the location of the string you want to find.

```
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**

It provides functions for managing strings stored in `QString` in an array, such as `QList`, and provides member functions such as `append()`, `prepend()`, and `insert()`.

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

✓ **QByteArrayMatcher**

It provides a function for finding matching characters by comparing `QString` strings.

```
QString x("hello World, nice to meet you.");  
QString y("World");  
  
QStringMatcher matcher(y);  
int index = matcher.indexIn(x, 0);  
  
qDebug("index : %d", index); // index : 6
```

✓ **QTextStream**

The `QTextStream` class provides STREAM processing to handle large amounts of text data.

Jesus loves you.

STREAM allows users to quickly and efficiently access large amounts of data to READ or WRITE. The following example uses QTextStream with READ data from a file using the QFile class.

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

QTextStream in(&file);
while (!in.atEnd()) {
    QString line = in.readLine();
    ...
}
```

- Container classes

Container classes are used to store certain types of data items or sets. For example, if you have multiple items to store with QString, you can use a Container class such as QVector

Container classes are easier to use and safer than the Container provided by STL. It is also lightened. Container provided by Qt can be used in place of Container provided by STL.

<TABLE> Container Classes provided by Qt

Classes	Explain
QHash<Key, T>	Template classes that provide a Dictionary based on a hash table
QMap<Key, T>	Binary Search Tree Based Template Class
QPair<T1, T2>	Processing class of item data that exists in pairs.
QList<T>	Template classes provided to address values in the form of a list
QLinkedList<T>	Template classes to provide a link list
QVector<T>	Classes provided to address dynamic QVector arrays
QStack<T>	Provided for use with push(), pop(), etc. based on stack
QQueue<T>	Provided for FIFO structure data manipulation

Jesus loves you.

QSet<T>	Classes available for hash-based quick search
QMap<Key, T>	The data mapped by the key value is connected as a combination to provide the Dictionary.
QMultiMap<Key, T>	Class inherited from QMap and multiple mapping values are available.
QMultiHash<Key, T>	Class inherited from QHash and can use Hesh using multiple mapping values.

✓ QHash<Key, T>

The QHash class provides a hash table-based Dictionary. Key and Value are stored in pairs in the way that data is stored. It provides the ability to quickly retrieve the data that you want to find with the key value. QHash offers features that are very similar to QMap, but its internal algorithms are faster than QMap.

```
QHash<QString, int> hash;  
  
hash["one"] = 1;  
hash["three"] = 3;  
hash["seven"] = 7;
```

You can use the insert() function as a method for storing keys and values in pairs in QHash. You can also use a value() member function to know the value value value.

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

✓ QMap<Key, T>

The use method of QMap is similar to QHash. The following example is an example source code that uses QString as a key and int as a value.

```
QMap<QString, int> map;  
  
map["one"] = 1;  
map["three"] = 3;  
map["seven"] = 7;
```

Jesus loves you.

```
map.insert("twelve", 12);

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

The following example uses the contents() function as a way to obtain a value value using the key value.

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

✓ **QPair<T1, T2>**

The QPair class can store two items in a pair of one. As shown in the example below, the QPair class can be used by declaring.

```
QPair<QString, double> pair;

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

✓ **QList< T >**

QList<T> provides fast index-based access and very fast data erasure. QList is an index-based class that is easier to use than QLinkedList's Iterator base and faster than QVector at the rate at which memory is allocated when data is stored.

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

QList can use the return value through the comparison operator as shown in the example below.

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

The QList can easily search for a location stored on the list by using the at() function.

```
for (int i = 0; i < list.size(); ++i)
{
```

Jesus loves you.

```
if (list.at(i) == "Jane")
    cout << "Found Jane at position " << i << endl;
}
```

✓ **QLinkedList<T>**

The QLinkedList class is based on the indicator and provides the ability to save and delete items in the list. The declaration can be made as shown in the following example.

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

The operator can be used to add items to the list, as shown in the following example.

```
QLinkedList<QString> list;

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

✓ **QVector<T>**

QVector stores the item in a memory location and provides fast index-based access.

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

You can specify the size of the number of data to be stored in advance and assign the Vector size at declaration. For example, a Vector with 200 storage spaces could be declared as follows: And you can set the Default value on 200 pre-declared storage spaces.

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

✓ **QStack<T>**

QStack is a class for providing Stack algorithms. This is the structure (LIFO) in which data that is later inserted comes first.

```
QStack<int> stack;
```

Jesus loves you.

```
stack.push(1);
stack.push(2);
stack.push(3);

while (!stack.isEmpty())
    cout << stack.pop() << endl;
```

✓ **QQueue<T>**

QQueue is a class to provide a Queue algorithm, a structure in which first inserted data comes first (FIFO).

```
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(3);

while (!queue.isEmpty())
    cout << queue.dequeue() << endl;
```

✓ **QSet<T>**

QSet is very fast when searching. The internal structure of QSet is implemented as QHash. The following examples are declarations and how to use them.

```
QSet<QString> set;

set.insert("one");
set.insert("three");
set.insert("seven");

set << "twelve" << "fifteen" << "nineteen";
```

✓ **QMultiMap<Key, T>**

Class inherited from the QMap class, which provides the ability to use multiple mapping values and can expand and use QMap.

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

Jesus loves you.

```
map3 = map1 + map2; // map3.size() == 3
```

- ✓ `QMultiHash<Key, T>`

The `QMultiHash` class is suitable for storing multiple Hash values. `QHash` does not allow multiple identical values, but `QMultiHash` has features that allow the same multiple values.

```
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
```

- Data classes

In addition to the basic data types offered by Qt, various classes are provided for flexible data manipulation. For example, `QBitArray` to handle data in bits and `QByteArray` to handle data in an array. Here, let's look at some of the classes that are useful for dealing with data.

<TABLE> Data classes provided by Qt

Class	Explain
<code>QBitArray</code>	Bit Array to provide bit operations (AND, OR, XOR, NOT)
<code>QMargins</code>	Class for handling each Marine of left, top, right, and bottom rectangle
<code>QPoint</code>	Classes provided to handle X, Y, and Z values
<code>QQuaternion</code>	Classes for handling quaternions consisting of vectors and scalars
<code>QRect</code>	left (qint32), top (qint32), right (qint32), and bottom (qint32)
<code>QRegExp</code>	Classes for processing regularization expressions
<code>QRegion</code>	Used to define clipboard areas on the Pinterest
<code>QSize</code>	Class for storing width and height
<code>QVariant</code>	Class in union type that can be stored as void type
<code> QVector2D</code>	Classes for representing Vector or vertex in two-dimensional space

Jesus loves you.

QVector3D	x, y, and z Classes with 3 coordinates available
QVector4D	Used to represent Vector or vertex in a four-dimensional space.

✓ QBitArray

The QBitArray class provides the ability to manage Bit units as data. It provides bit operation function through AND, OR, XOR, and NOT computation.

```
QBitArray ba(200);

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

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

✓ QMargins

The QMargins class can store Left, Top, Right, and Bottom, and can store or modify values using the setLeft(), setRight(), setTop() and setBottom() member functions.

```
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)
```

Jesus loves you.

✓ QPoint

QPoint class provides the functions used to display coordinate points.

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

QPoint can be added as follows.

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

✓ QQuaternion

QQuaternion can use a quaternion consisting of vectors and scalars. Quaternion provides the ability to manage the data used in 3D at angles such as SCALAR, X, Y, and Z coordinates and rotations.

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

✓ QRect 와 QRectF

QRect class can be used to store coordinate values of rectangles using an integer and a QRectF with a Float You can save the X, Y, Width, and Height values of the rectangle.

```
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 class provides regular expression capabilities.

```
QRegExp rx("^\\d\\d?"); // 0~99 integer values
```

Jesus loves you.

```
rx.indexIn("123");           // return -1  
rx.indexIn("-6");           // return -1  
rx.indexIn("6");            // return 0
```

✓ QRegion

QRegion is used in conjunction with QPainter::setClipRegion() function. For use as a clipboard a specific area inside a rectangle on the Pinterest, the following may be used.

```
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 class is mainly used to store width and height using the int value.

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

✓ QVariant

QVariant class provides the ability to specify the type of data that has not been determined. For example, void *.

```
QDataStream out(...);  
QVariant v(123);  
int x = v.toInt();  
  
out << v;  
v = QVariant("hello");  
v = QVariant(tr("hello"));
```

Jesus loves you.

```
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 and QVector4D

QVector2D is used to handle Vector and Vertex in 2D coordinates. QVector3D is a class in which Z has been added to X and Y. And the QVector4D class offers the added functionality of W.

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

3.4. Signal and Slot

Qt uses Signal and Slot as a mechanism for handling events. For example, the act of a button being clicked is called Signal in Qt. And when a signal occurs, the function called is called a slot function. When an event called signal occurs, a slot function associated with the signal is called.

Signals are events that occur in a situation. Qt's handling of all events uses a mechanism called signal and slot.

For example, suppose in a chat program with Qt that a user named A sends a message to a user named B, the act of receiving a message from A from B could be defined as Signal

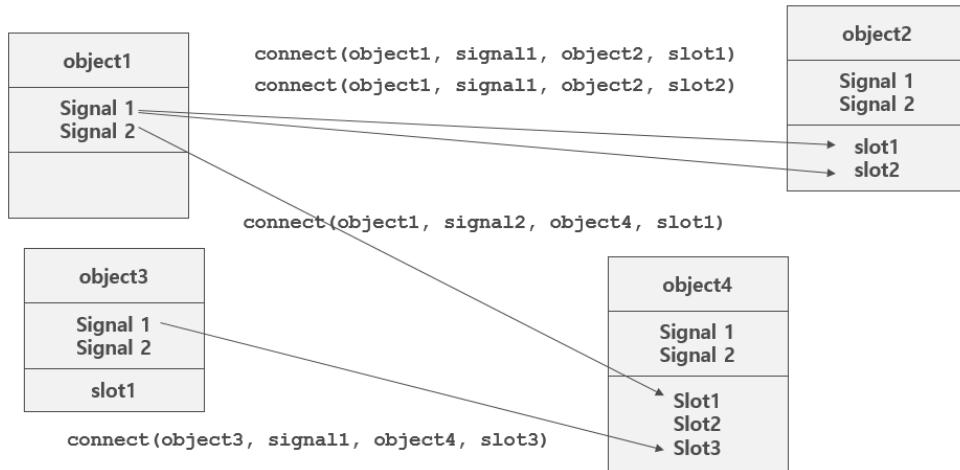
It then calls the slot function associated with the signal received the message. Qt uses signals and slots for all event processing. As an example just now, all APIs use event mechanisms called signal and slot, as well as GUI and network modules. Signals and slots simplify program source code, which can shorten development period and simplify complex program structure.

Suppose you develop a chat program without using Qt's signal slot. A program of several thread structures should be developed. For example, programs can become complex because they require the use of multiple threads, such as threads that handle messages sent by a particular user, threads that handle whispers, and threads that notify when a new user is registered. However, the use of signals and slots in Qt makes it very simple to implement because there is no need for threads.

All GUI widgets provided to Qt have a predetermined variety of signals. For example, various signals are defined, such as click, double click, mouse over, and so on in QPushButton.

Signals and slots can be thought of as one pipeline. A single signal can invoke multiple slot functions. In addition, multiple signals can call a single slot.

Jesus loves you.



<FIGURE> Signal and Slot Architecture

The function for connecting the signal and slot functions can connect the signal and slot using the `connect()` function of the `QObject` class. The first factor in the `connect()` member function is the object on which the event occurred and the second factor is the signal of the object.

For example, if you have a button named A, the object name of the button A is the first factor, and the second factor is the A button's click or double click. Therefore, the click event is specified as the second factor. The third factor specifies the signal and the name of the object with the slot function to be called. The fourth factor specifies the slot function to be called when the generated signal occurs.

So far we have looked at the concepts of Signal and Slot. Now let's try to create an example program using Signal and Slot.

- Signal and Slot example

This code example is a signal occurs, that outputs a The text of the window placed on the label.



<FIGURE> Signal and Slot example screen

Jesus loves you.

In this example, there are two classes, SignalSlot and Widget. The following example source code is a class called SignalSlot.

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

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

signals:
    void valueChanged(int newValue);

private:
    int m_value;
};
```

As shown in the example above, signs: at the bottom of the keyword there is a function called valueChanged(). This function is a signal function.

The Signal function has no Implementation Department and only needs to implement a Defined Part in the header, as shown by the source code. The valueChanged() Signal function specifies the int factor. This factor can pass the value as a factor to the Slot function when the signal is generated. One value is used here. If necessary, factors may not be used or several may be used as factors.

setValue() member function in the public keyword of the class SignalSlot of the example code above. When this member function is called, you can check the source code using the keyword emit in the function. The emit keyword generates a signal event. The following example source code is the header part of the Widget class.

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

Jesus loves you.

```
private:  
    QLabel *lbl;  
  
public slots:  
    void setValue(int val);  
};
```

If you look at the bottom of the Widget class, you will see that the public keyword has a combination of the slots keyword. The slots keyword can be used with a private or public keyword. A specified member function can define a Slot function using the keywords slots. The following example source code is the implementation part of the Widget class.

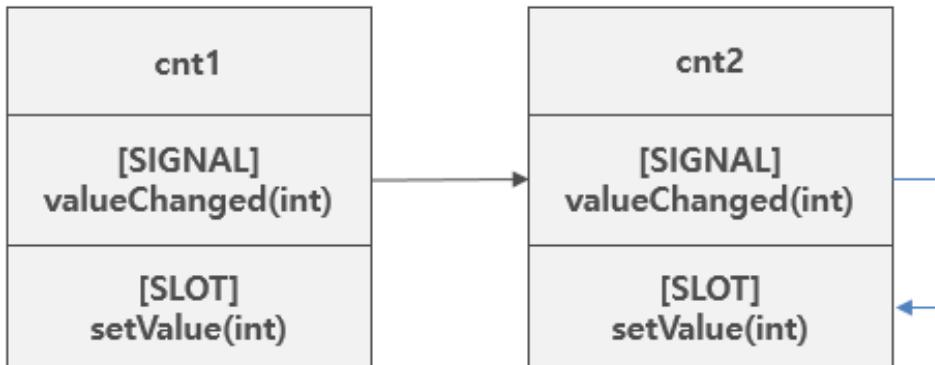
```
#include "widget.h"  
  
Widget::Widget(QWidget *parent) : QWidget(parent)  
{  
    lbl = new QLabel("", this);  
    lbl->setGeometry(10, 10, 250, 40);  
  
    SignalSlot myObject;  
  
    // New Style  
    connect(&myObject, &SignalSlot::valueChanged, this, &Widget::setValue);  
  
    /* Old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

You can use the connect() member function to connect Signal and Slot. There are two ways to connect Signal and Slot. The annotated New Style is a way to connect the signal and slot added from Qt 5.5 or later. Both New Syntax and Old Style are available. Only

Jesus loves you.

the Old Style method can be used in Qt 5.5 and earlier versions. New Style styles do not have multiple factors available. And the New Style can be used as the fourth factor in the connect() function, as well as the Slot function.

As shown in the example, the Signal and Slot functions should not be implemented only in other classes. Signals that exist in the same class can invoke the Slot function. And as you can see in the following figure, Signal can call up Signal in addition to the Slot function.



<FIGURE> Signal -> Signal

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

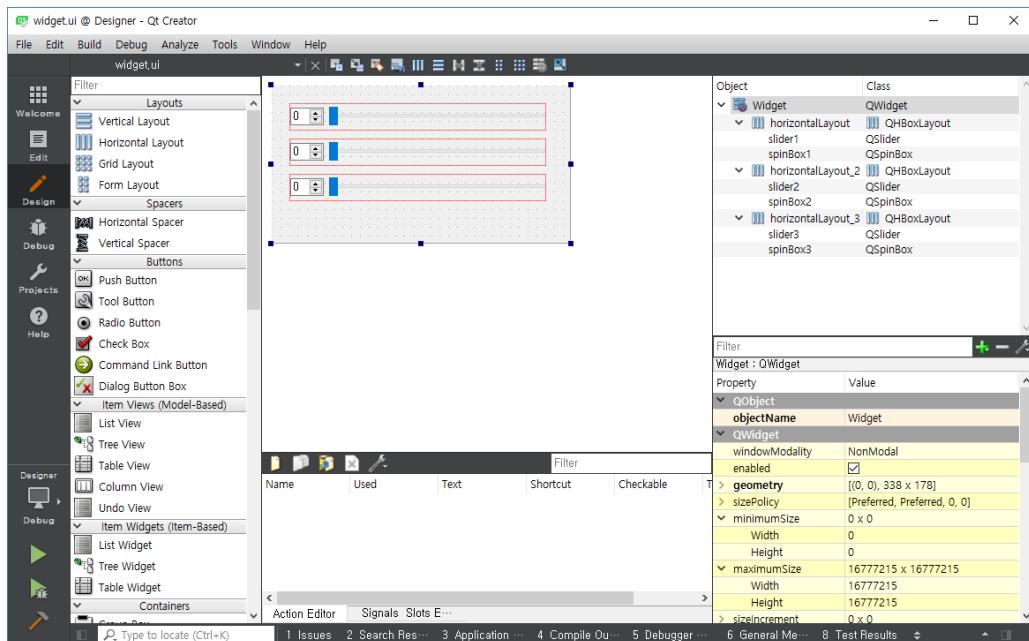
connect(cnt1, SIGNAL(valueChaged(int)), cnt2, SIGNAL(valueChaged(int)));
connect(cnt2, SIGNAL(valueChaged(int)), cnt2, SLOT(setValue(int)));
```

At the top of the class using Signal and Slot, you can see that you used the keyword Q_OBJECT. This keyword must be specified as viewing Q_OBJECT in the class header as required when using Signals and Slot in Qt. In some cases, Signals and Slot are used without declaring Q_OBJECT, which will result in an error, so care must be taken.

Exmple - ch03 > 04_SignalSlot > 01_CustomSignalSlot example.

3.5. Implementing the GUI using the Qt Designer tool

Qt provides Qt Designer to quickly implement the desired GUI. Qt Designer allows users to place widgets on the GUI while dragging them with the mouse. In the days when the Qt Creator IDE tool was not available, a tool called Qt Designer was provided independently. However, it is now integrated into the Qt Creator IDE tool. Qt Designer tools are still available as independent execution programs. The reason is still for developers using IDE tools such as Visual Studio.



<FIGURE> Designer included within Qt Creator tool

The file with the extension file, gui ui xml, such as source code format to be below. This allows the user to the mouse using the widget, qt creator designer of the tool automatically in xml. To build and files, the supposed xml gui c++ to source code and build.

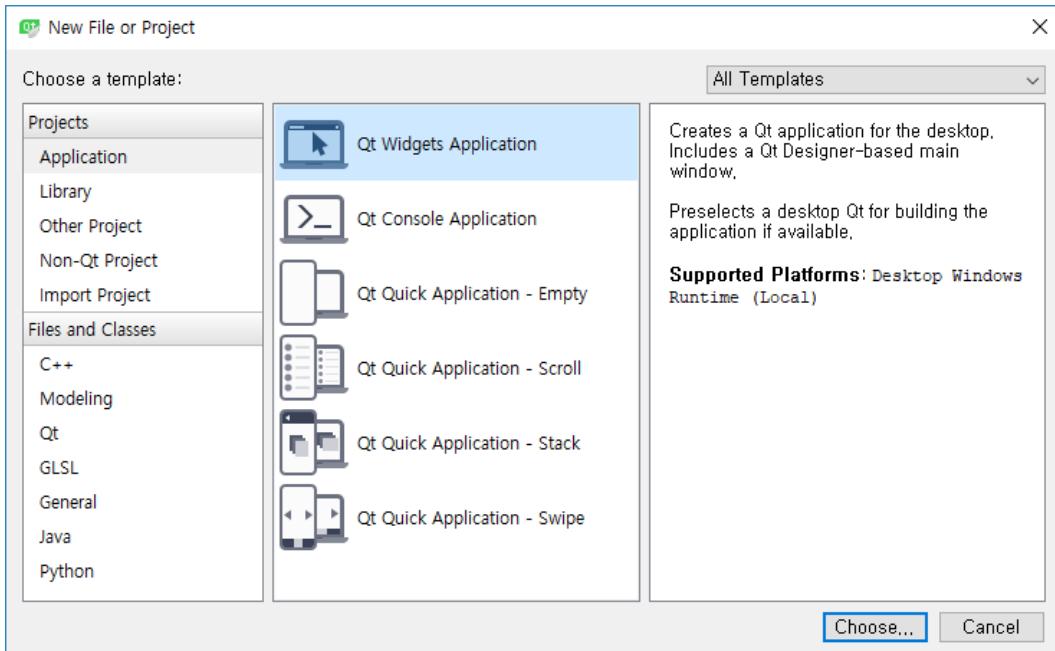
```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>Widget</class>
<widget class="QWidget" name="Widget">
<property name="geometry">
```

Jesus loves you.

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

- Example with Qt Designer

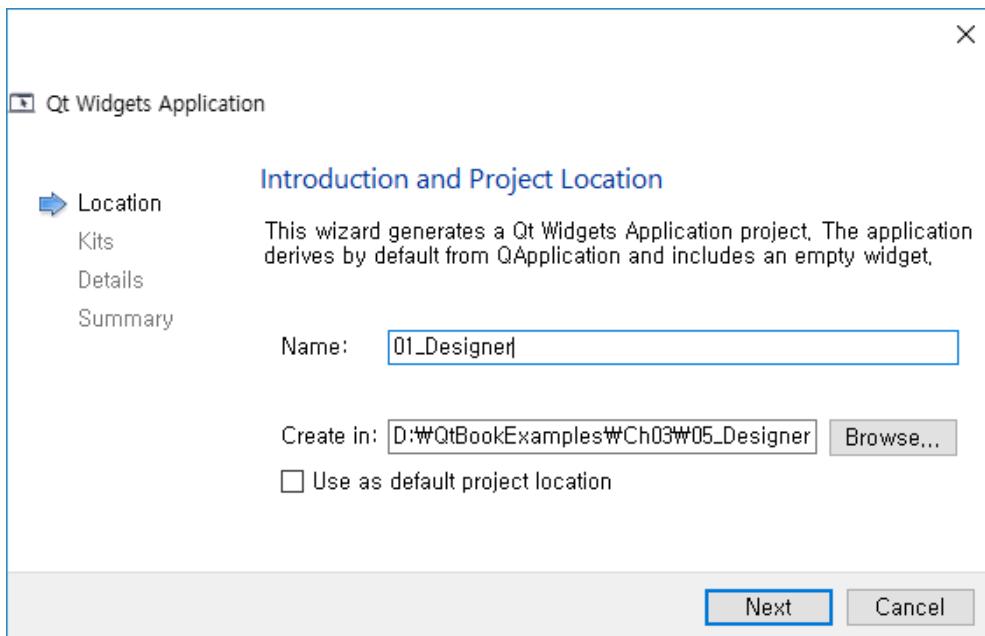
Run Qt Creator, as shown in the following figure, then run [File] > [New file or Project...]. Click on the menu. Once the dialog is loaded, select the [Application] item from the [Projects] item on the left side of the dialogue.



<FIGURE> [New File or Project] Dialog

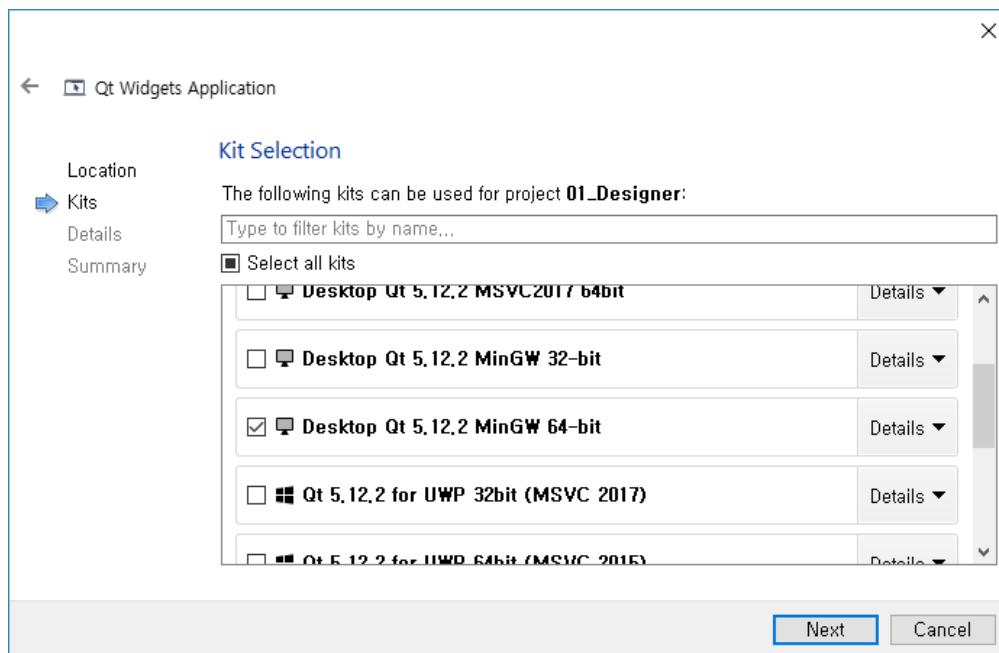
Then select Qt Widgets Application from the center and click the [Choose] button at the bottom. In the next screen, enter the project name and location to be created and click the [Next] button.

Jesus loves you.



<FIGURE> Name and Location Dialogs for Project

The following selects the compiler to build the application. Select either MinGW 32 bit or 64 bit version. When you are done making your selection, click the [Next] button at the bottom.

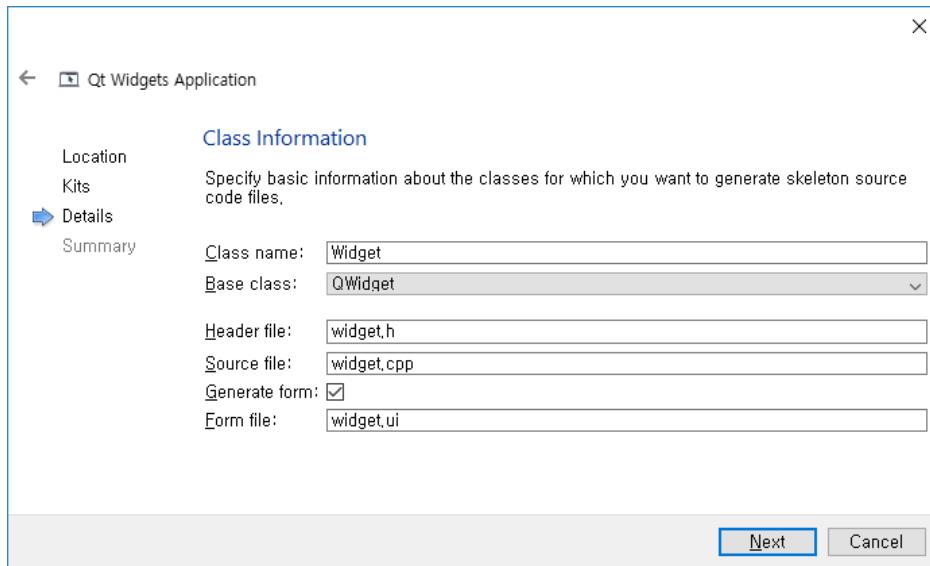


<FIGURE> Select compiler screen

Jesus loves you.

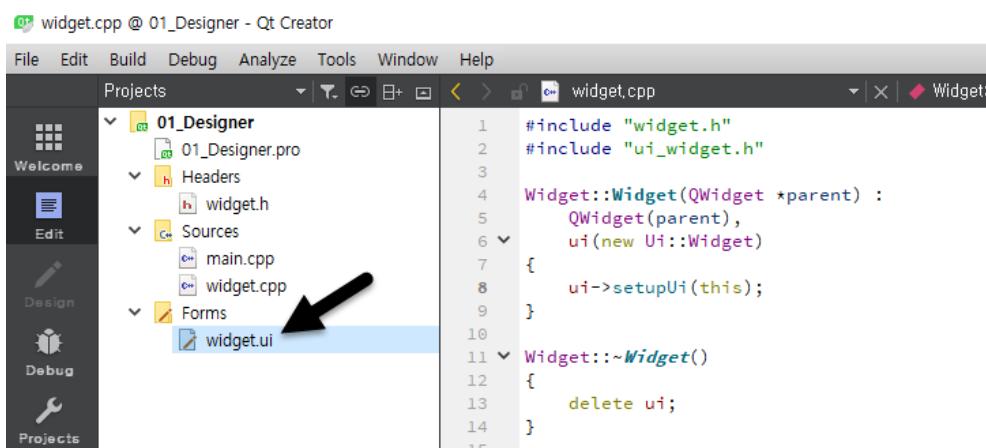
The next screen is a dialogue to define the class. In the Base class item, select QWidget. Then, enable (check) the Generate form check box.

This item is intended to use features for designing screens using the Designer described in this section. widget.ui automatically generates UI files for the widget class. Select and click the [Next] button at the bottom, as shown in the following illustration.



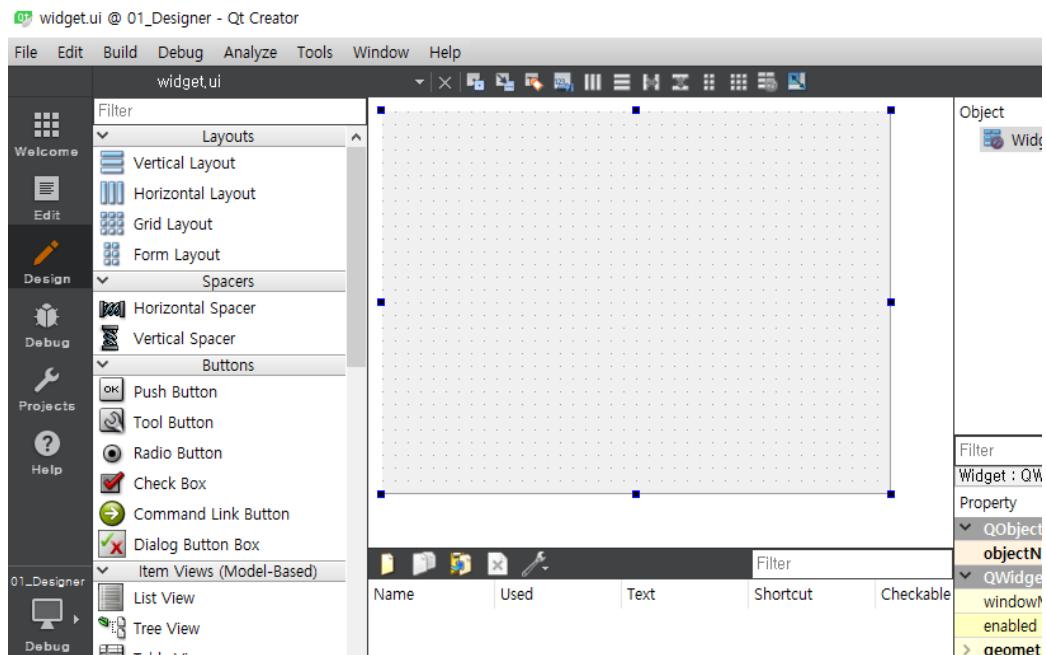
<FIGURE> Class Information Dialog

In the next dialogue window, click the [Finish] button at the bottom to complete the creation of the project. When the project creation is complete, double-click the widget.ui source code in the project window to the left of the Qt Creator, as shown in the following figure, to switch to the Designer screen.



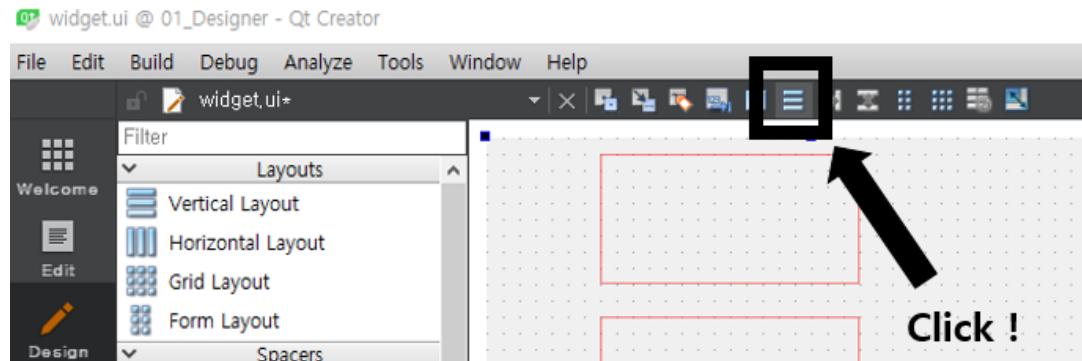
<FIGURE> Qt Creator screen

Jesus loves you.



<FIGURE> Designer screen

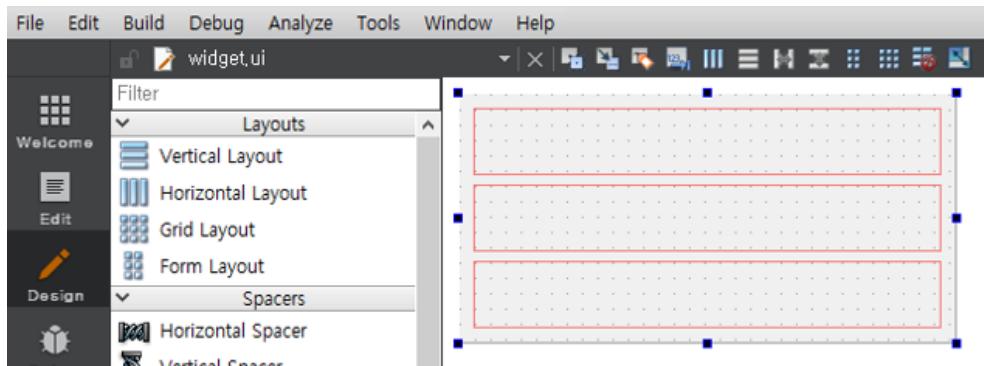
You see the picture before designer, such as tools to switch to Let's screen widgets placed in. First, left the [layout] tab, three horizontal layout in Use your mouse to drag a person out of gui widget. Then click the icon button with three rectangles stacked vertically from the toolbar at the top, as shown in the following figure.



<FIGURE> Selecting Layout

Old as you see the picture window by clicking the icon on horizontal layout the window the size of the changes, the horizontal layout this automatically adjust the size.

Jesus loves you.



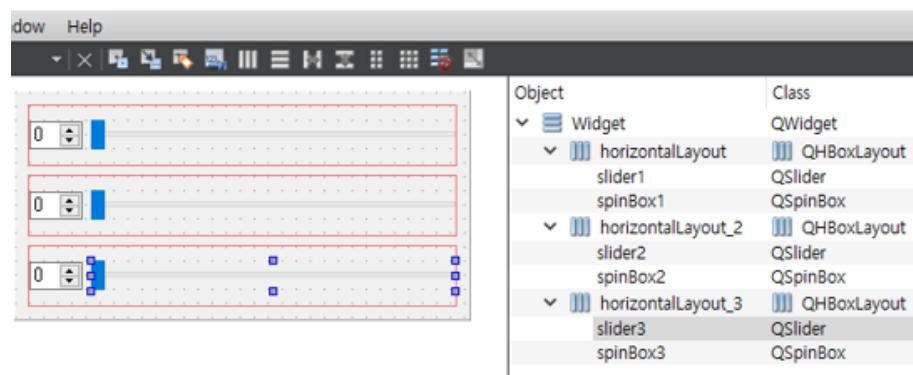
<FIGURE> Selecting Layout

When you have completed the screen layout as shown in the previous figure, place QSpinBox and QSlider on each Horizontal Layout.

<TABLE> 배치할 위젯들

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

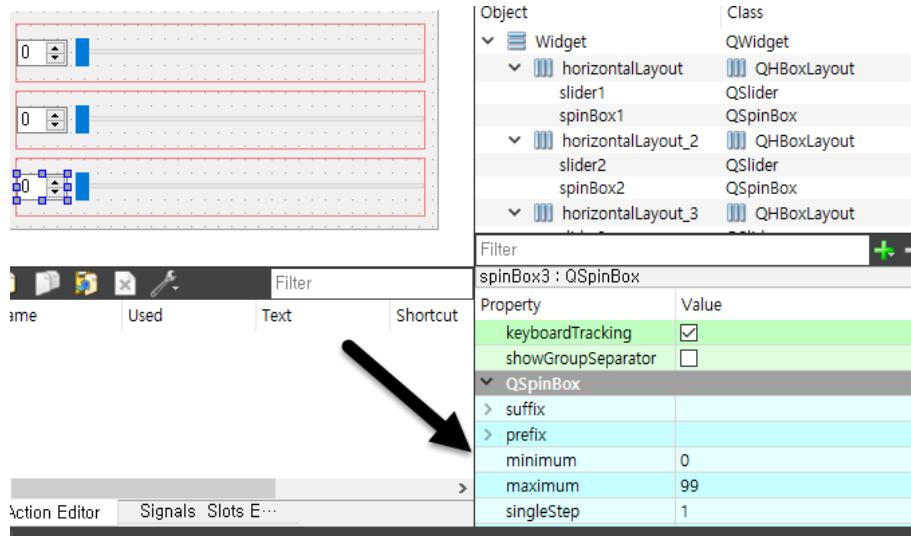
Drag each widget with the mouse to change the object name of the widgets, as shown in the table you have placed. You can change the Object Name change by double-clicking on the `objectName` item in the lower right corner.



<FIGURE> `objectName` change screen

Jesus loves you.

See figure below, such as object name and to zero the value after changing a person for each widget's minimum maximum value is changed to 99.



<FIGURE> Widget Detail Properties

If you have all the widgets in the GUI, try building the application and running it. When the application is running, try changing the size of the widget with your mouse and see if the widget automatically changes its size.



<FIGURE> Example screen

Changing the position of the scale of each QSlider widget that you place in the Designer tool changes the value of QSlider. Example of setting the changed value to the value of QSpinBox if this value is changed. The source code below is the widget.h header file.

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

namespace Ui { class Widget; }
```

Jesus loves you.

```
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
```

There is an Object who declared ui in the header file. This object name is the accessor to the widget that is placed in the Designer tool. For example, you can use a ui object in the source code to access an object called slider2 placed in the Designer tool.

Example below The Slot function at the bottom of the source code is to be doubled in Designer

```
#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() {
```

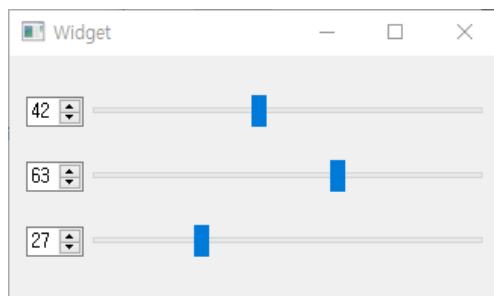
Jesus loves you.

```
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);  
}
```

One caveat in the above example is that the factor of the connect() function was used in the form of an Old Style. If you use New Style, an error called "no matching member function for call to connect" occurs. This error occurs because the valueChanged() member function provided by QSpinBox is an overloaded member function that provides two member functions: int and QString.

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

If there is an overloaded signal as shown in the source code above, use Old Style.



<FIGURE> Application examples screen

Example - Ch03 > 05_Designer > 01_Designer

3.6. Dialog

A dialogue is used to send a message to the user when an event occurs while the application is operating. It also provides a GUI to receive input values from users or to select one of several. The following table is a frequently used dialogue provided by Qt.

종류	설명
QInputDialog	Dialogue where you can enter a value from the user.
QColorDialog	Dialogue in which you can select a specific color.
QFileDialog	Provides a GUI interface for selecting files or directories.
QFontDialog	Dialogs for selecting fonts
QProgressDialog	Dialogue to show progress, such as percent.
QMessageBox	Modal dialogue

- QInputDialog

The QInputDialog class can receive values from users.

```
bool retValue;

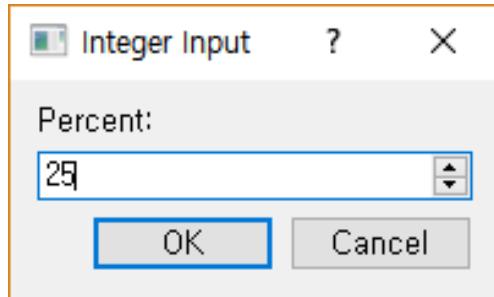
int i = QInputDialog::getInt(this, "Integer Input", "Percent:",
                             25, 0, 100, 1, &retValue);

if (retValue)
{
    qDebug("true %d", i);
}
```

getInt() member function of the QInputDialog class provides a dialogue that allows you to enter an integer value from the user.

getInt() member function of the QInputDialog class provides a dialogue that allows you to enter an integer value from the user.

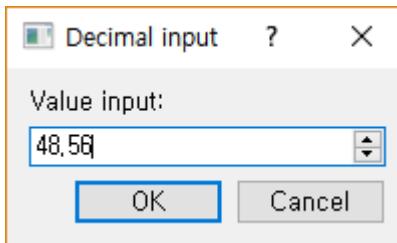
Jesus loves you.



<FIGURE> integer value input dialog

The first factor is the parent class, the second factor is the title to be displayed in the title bar of the window, and the third factor is the name of the item to be displayed on the left side of the input value widget item. The fourth factor is the default setting value, the fifth and sixth are the range of values that the user can enter, and the next factor is the increment in the spin box of the dialogue. The last factor in the dialogue Stores a value that can confirm that you click on [ok] or [cancel] button.

```
bool retValue;  
  
double dVal = QInputDialog::getDouble(this, "Decimal input", "Value input:",  
                                     48.56, -100, 100, 2, &retValue);
```

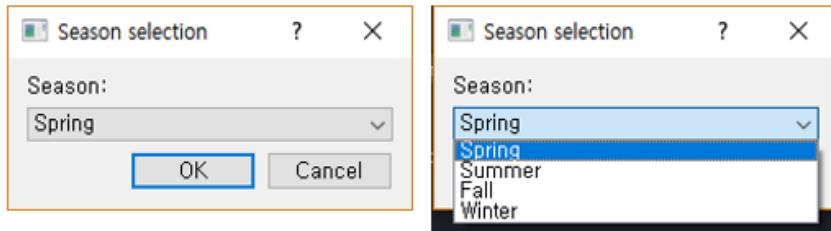


<FIGURE> Double value input dialog

getDouble() member function of the QInputDialog class allows you to enter a real value. getItem() member function provides the ability to select one of the strings (or words).

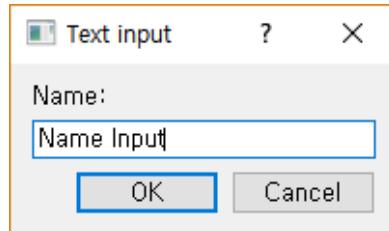
```
QStringList items;  
items << "Spring" << "Summer" << "Fall" << "Winter";  
  
bool ok;  
QString item = QInputDialog::getItem(this, "Season selection", "Season: ",  
                                      items, 0, false, &ok);
```

Jesus loves you.



<FIGURE> QInputDialog

getText() member function of the QInputDialog class provides a dialogue where you can enter and receive text from the user.

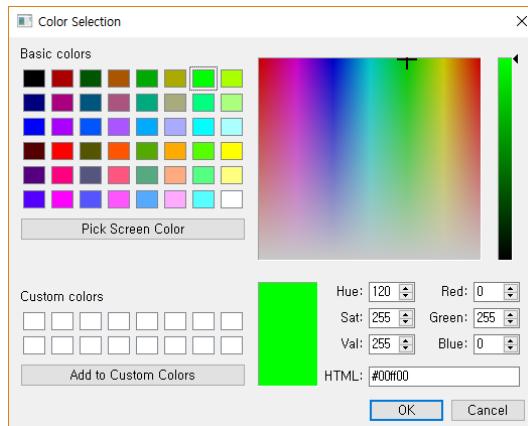


<FIGURE> TEXT input dialog

```
bool ok;  
QString text = QInputDialog::getText(this, "Text input", "Name: ",  
                                     QLineEdit::Normal, "Name Input", &ok);
```

- QColorDialog

QColorDialog class provides the ability for users to view color schemes and select the desired color.



<FIGURE> QColorDialog

Jesus loves you.

```
QColor color;
color = QColorDialog::getColor(Qt::green, this, "Color Selection",
                               QColorDialog::DontUseNativeDialog);

if (color.isValid())
{
    qDebug() << Q_FUNC_INFO << "Valid Color";
}
```

- **QFileDialog**

QFileDialog class provides a dialogue for selecting files from users. You can filter out specific extensions or specific files to show users.

getOpenFileNames() member function of the QFileDialog class provides the ability to multi-select files that exist within a directory. getExistingDirectory() member function allows the user to select a directory. In addition, getSaveFileName() member function can specify the files to be saved by the user.

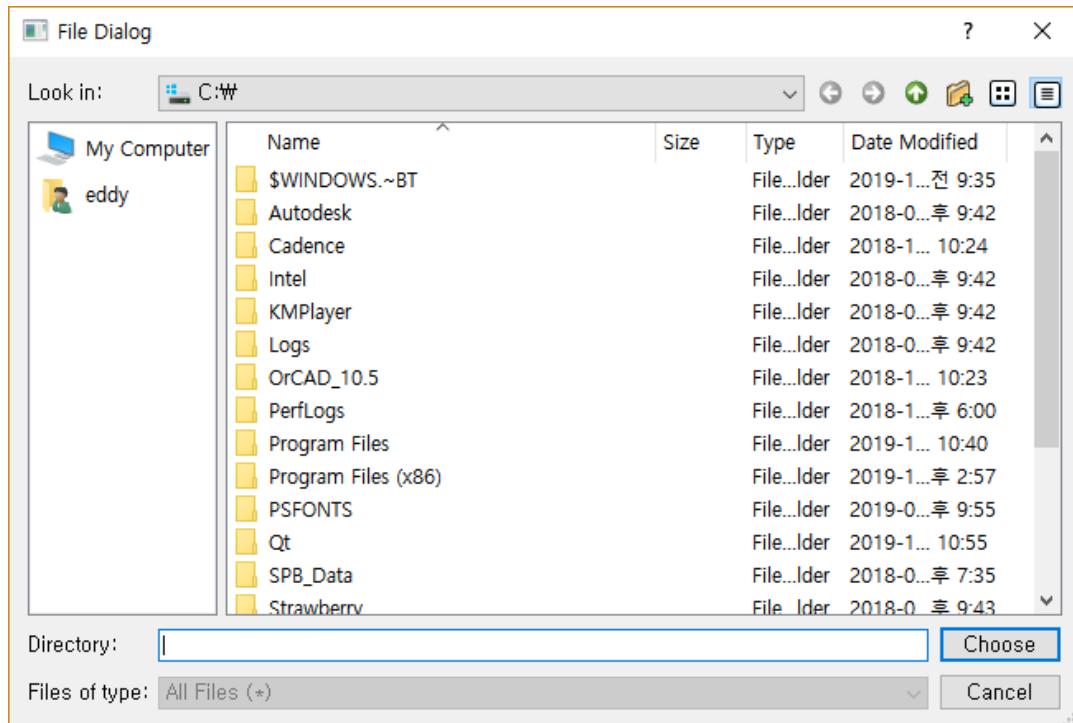
```
QFileDialog::Options options;

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

QString directory = QFileDialog::getExistingDirectory(this,
                                                       "File Dialog",
                                                       "C:",
                                                       options);
```

getExistingDirectory() member function provides the ability to select directories from users. The first factor is the parent class, the second factor is the title bar title of the dialogue, and the third factor is the factor that the specified directory uses to default. The last factor is an option for filtering using constant values in the file dialogue.

Jesus loves you.



<FIGURE> QFileDialog example screen

<TABLE> QFileDialog::Options

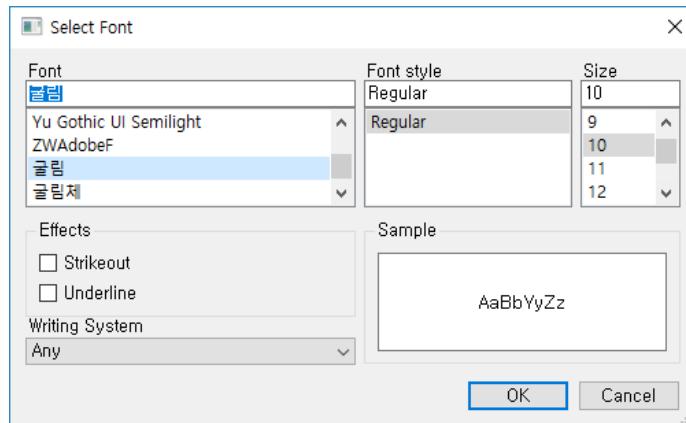
Constant	explanation
QFileDialog::ShowDirsOnly	Show Directory Only
QFileDialog::DontResolveSymlinks	Use to avoid displaying symbolic links
QFileDialog::DontConfirmOverwrite	Do not display warning messages when overwriting
QFileDialog::DontUseNativeDialog	Use to disable the system default file dialogue
QFileDialog::ReadOnly	Enable File Dialog in Read Mode
QFileDialog::HideNameFilterDetails	Use to hide files using filters

● QFontDialog

QFontDialog provides a dialogue from the user to select the font. The first factor specifies a variable to confirm whether the user clicks the [OK] button located at the bottom of

Jesus loves you.

the dialogue or the [CANCEL] button.



<FIGURE> QFontDialog

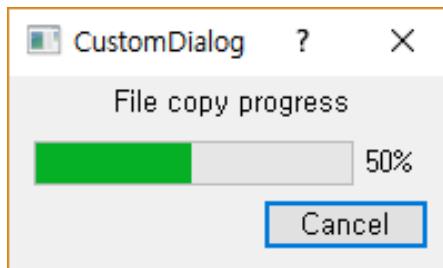
The second factor specifies the font that can be specified as the default selection on the font dialogue.

```
bool ok;
QFont font;

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

- QProgressDialog

QProgressDialog class is used to show users the current progress. For example, you can show the progress in the dialogue window on things that take some time, such as copying large files.



<FIGURE> QProgressDialog

The following is a dialogue example source code to show the user progress using the QProgressDialog class. Create a project from the QWidget class as Base class and create the widget.h header file as follows.

Jesus loves you.

```
#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
```

In the above example, QTimer called a timer event once a second to increase the progress value of the QProgressDialog by 1%. The next source code is the widget.cpp source code.

```
#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("File copy progress", "Cancel", 0, 100);

    connect(pd, SIGNAL(canceled()), this, SLOT(cancel()));
}
```

Jesus loves you.

```
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 class provides modal dialogue. The buttons that can be passed on to the user and that provide the QMessageBox class with access to the dialogue window provide a wide range of buttons, as shown in the following table.

<TABLE> Available buttons in the QMessageBox

Constant	Value	Explanation
QMessageBox::Ok	0x00000400	OK button
QMessageBox::Open	0x00002000	Open File Button
QMessageBox::Save	0x00000800	Save button
QMessageBox::Cancel	0x00400000	Cancel button
QMessageBox::Close	0x00200000	Close button

Jesus loves you.

QMessageBox::Discard	0x00800000	Unsaved and Discard button
QMessageBox::Apply	0x02000000	APPLY button
QMessageBox::Reset	0x04000000	RESET button
QMessageBox::RestoreDefaults	0x08000000	Save again button
QMessageBox::Help	0x01000000	Help button
QMessageBox::SaveAll	0x00001000	Save All button
QMessageBox::Yes	0x00004000	YES button
QMessageBox::YesToAll	0x00008000	Apply YES to All button
QMessageBox::No	0x00010000	NO BUTTON
QMessageBox::NoToAll	0x00020000	Apply all NO buttons
QMessageBox::Abort	0x00040000	Stop button
QMessageBox::Retry	0x00080000	Retry Button
QMessageBox::Ignore	0x00100000	Ignore button
QMessageBox::NoButton	0x00000000	An invalid button

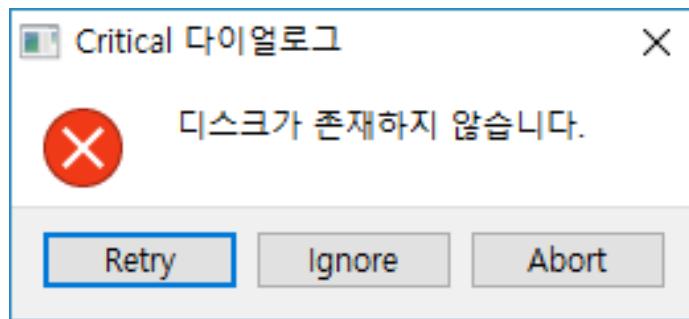
The following examples are for using the [Abort] button, [Retry] button, and [Ignore] button using the QMessageBox class.

```
QMessageBox::StandardButton reply;

reply = QMessageBox::critical(this,
    "Critical Dialog",
    "There is no diak.",
    QMessageBox::Abort |
    QMessageBox::Retry |
    QMessageBox::Ignore);

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

Jesus loves you.



<FIGURE> Critical Dialog

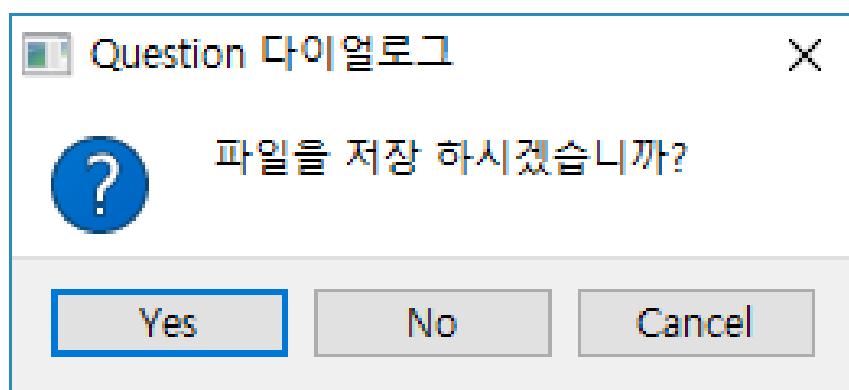
QMessageBox class provides information(), query(), and warning() member functions. The following is an example of using the query() member function.

```
QMessageBox::StandardButton reply;

reply = QMessageBox::question(this, "Question Dialog",
                             "Do you want to save the file?",
                             QMessageBox::Yes | QMessageBox::No | QMessageBox::Cancel);

// 다이얼로그 - Dialog
// 파일을 저장 하시겠습니까? - Do you want to save the file?

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



<FIGURE> Question Dialog

Jesus loves you.

- Custom dialog example

In this example, let's take over the QDialog class and try to implement the desired style of dialogue. The following example source code is the dialog.h header file.

```
#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
```

The dialog.h header file is a header file to implement the class inherited from the QDialog class. This dialogue allows you to enter names and has two buttons placed. The following example source code is the full source code of the dialog.cpp.

```
#include "dialog.h"

Dialog::Dialog(QWidget *parent) : QDialog(parent)
```

Jesus loves you.

```
{  
    setWindowTitle("Custom Dialog");  
  
    lbl = new QLabel("Name");  
    edit = new QLineEdit("");  
    okbtn = new QPushButton("OK");  
    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()  
{  
}
```

The slot_okbtn() Slot function is called when you click the OK button on the dialog. The slot_cancelbtn() Slot function is recalled by clicking the [Cancel] button. The following example is the source code main.cpp source code.

```
#include <QApplication>  
#include <QDebug>
```

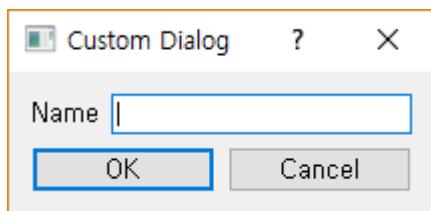
Jesus loves you.

```
#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();
}
```



<FIGURE> Custom Dialog example

As shown in the example above, the `dlg` object in the `Dialog` class returns the `int` value 1 when the user clicks the `OK` button. [Cancel] Click the button to return zero.

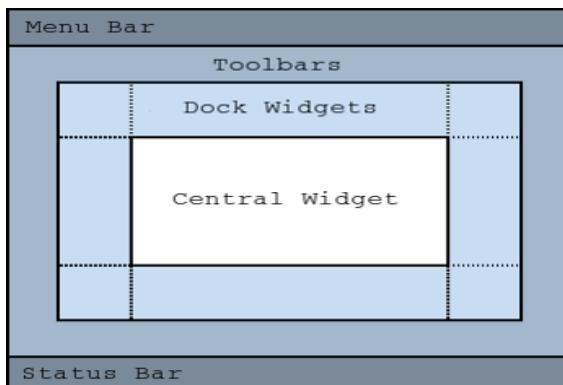
Example - Ch03 > 06_Dialog > 01_CustomDialog

3.7. MDI (Multi Document Interface)

So far, we have covered an example of placing a GUI-based widget on a window using Qt. This method has been dealt with through single document interface (SDI) method. The SDI method is suitable for simple GUI configuration in such a way that only one window screen exists.

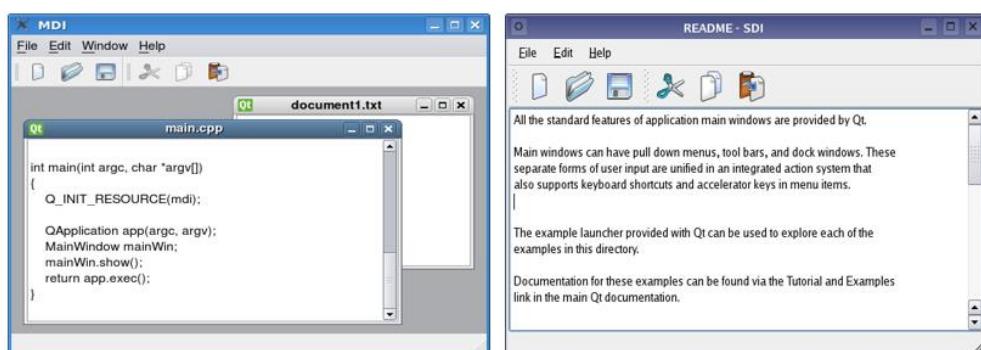
However, when functions are complex and many functions need to be provided to users, implementing a GUI using the Multi Document Interface (MDI) method rather than the SDI method would provide an intuitive GUI for users.

MDI method can be composed of different areas. For example, you can place widgets in specific areas, such as Menu Bar, Toolbars, Status Bar, Dock Widget, Central Widget, and so on.



<FIGURE> MDI Window

Unlike the SDI method, the SDI method can have one window area, but the MDI method can have multiple windows within the main window area using the QMdiArea class. Therefore, it is recommended to use the MDI method if it is necessary to implement a complex window GUI.



<FIGURE> MDI and SDI

Jesus loves you.

- QMdiArea example

In this example, let's use the QMdiArea class to implement the MDI-based GUI. This example is provided by installing Qt. Refer to the full source code in the example of MDI Examples. This example source code is implemented in two classes.

The MainWindow class is the implementation main window GUI, inheriting the QMainWindow class. This class implements Menu Bar, Tool Bar, Central Widget Area, etc.

The MDIMainWindow class is a widget class that inherits the QTextEdit widget class. This class is used as a multi-edit widget to center MainWindow. It is a class that is implemented to provide the ability to edit multiple text files within a single window area, such as an Ultra editor, a Notepad, and so on. The following example source code is the header source code of the MainWindow class.

```
#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 function is called when you click the [New] menu in the menu bar. open() function is called by clicking on the [Open] menu. The following example is the mainwindow.cpp source code.

```
#include "mainwindow.h"
#include <QMenu>
#include <QAction>
#include <QMenuBar>
#include <QToolBar>
```

Jesus loves you.

```
#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"));
}
```

Jesus loves you.

```
MainWindow::~MainWindow()
{
}

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

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

The creator of the MainWindow class defines the menu items to be placed in menus and toolbars on the MDI window GUI. Then, when you click on each menu, link it to the Slot function when a signal event occurs.

The QDockWidget is located on the left side of the MDI window and provides a GUI for the user to separate from the GUI into a new window. The MDIMinWindow class is used as a child window in the MDI window. It provides the same functionality as providing multiple Child windows within the GUI. The following is the header source code for the MDIMainWindow class.

```
#include <QMainWindow>
#include <QObject>

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

This class constructor uses the QMdiArea class and the QMdiSubWindow class to create a window to register as a sub-window under the MDIMainWindow. The following example source code is the MDIMainwindow.cpp source code.

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

Jesus loves you.

```
#include <QMdiSubWindow>
#include <QPushButton>

MDIMainWindow::MDIMainWindow(QWidget *parent) : QMainWindow(parent)
{
    setWindowTitle(QString::fromUtf8("My MDI"));

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

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

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



<FIGURE> MDI example

Example - Ch03 > 07_MDI > 01_MDIMainWindow

3.8. File and Data stream

Qt provides a QFile class to process files. In addition, QTextStream and QDataStream classes are provided for the purpose of efficiently re-READ/WRITE large For example, if a file with a smaller file size is READ / WRTIE and the member function provided by QFile is fine, but the QTextStream class and the QDataStream class are used to handle files with larger file capacities, then the file can be READ/WRITE efficiently to handle large files.

The following example uses only the QFile class instead of the data stream to READ the data from the file.

- ✓ Retrieve data from a file using the QFile class

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

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

open() member function of the QFile class provides the ability to open a file. When you read a file at open time, you can specify whether the file is OPEN in read mode or OPEN() in READ mode, WRITE mode, or READ/WRITE enabled mode.

QIODevice::ReadOnly allows only files to be READ. The QIODevice::Text option can be used to open files in TEXT mode.

<TABLE> File mode

Constant	value	explanation
QIODevice::NotOpen	0x0000	Use when not opening a file
QIODevice::ReadOnly	0x0001	Use in read-only mode
QIODevice::WriteOnly	0x0002	Use in write-only mode

Jesus loves you.

QIODevice::ReadWrite	Read Write	Use read / write mode together
QIODevice::Append	0x0004	Use to add to end of file
QIODevice::Truncate	0x0008	Delete previously used files for new connections if previously opened files exist.
QIODevice::Text	0x0010	Use 'Wn' at the end when reading in TEXT mode. Use 'WrWn' at the end in MS Windows.
QIODevice::Unbuffered	0x0020	Use the device immediately without using the buffer

In the example source code above, the `atEnd()` member function used as a condition in the `while` statement repeats until the last time of the file. The `readLine()` member function provides a read-in function until you meet the '`Wn`' character.

The `QFile` class can READ/WRITE files using `QTextStream` and `QDataStream` classes that process STREAM. The following example is a method for reading data from a file using the `QFile` and `QTextStream` classes.

```
#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();
}
```

Jesus loves you.

```
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[])
{
    QCoreApplication a(argc, argv);

    QString filename = "C:/Qt/MyFile.txt";

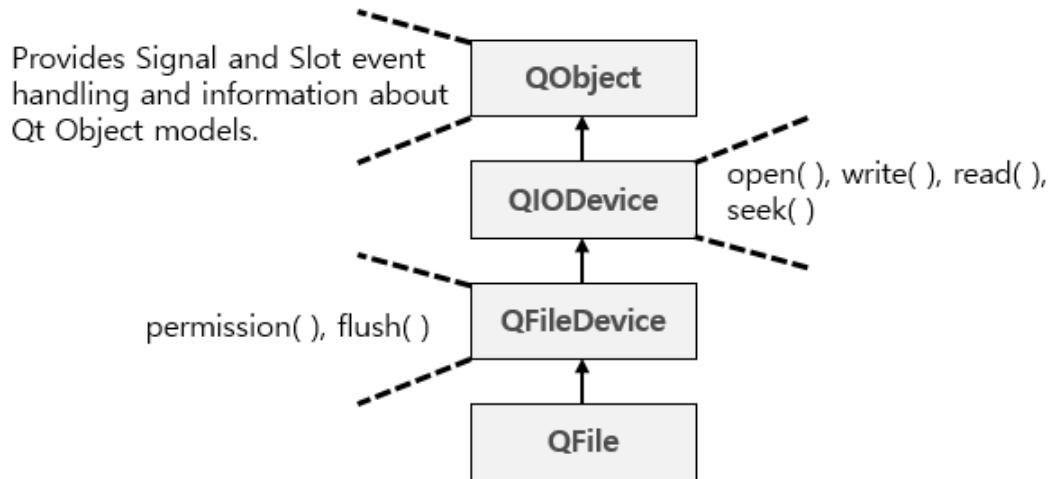
    write(filename);
    read(filename);

    return a.exec();
}
```

QFile class provides features such as Open, Exists, Link, Remove, Rename, and more for file processing. The QFile class was inherited from the QFileDevice class and implemented. The QFileDevice class implements exception handling functions such as Permission, Flush, and Error processing of files.

QFileDevice class inherited the QIODevice class. The QIODevice class implements such functions as Open, Write, Read, Seek, and so. In addition to QFile, QIODevice is inherited from several classes and used classes. For example, it may be used in multimedia to retrieve sound data, or media from MP3 files

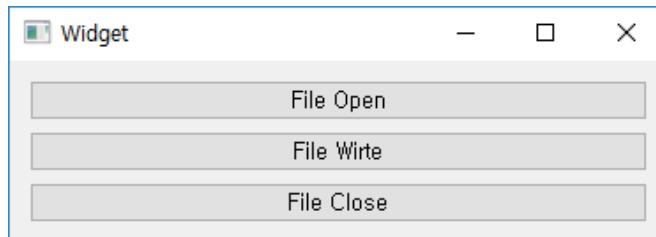
Jesus loves you.



<FIGURE> Inheritance Relationship of QFile Classes

The reason why QFile class inherited relationships are explained here is that if the application of an API that requires data to be READ/WRITE from a new device is implemented in the same inheritance relationship as QFile, features such as Open, Write, Read, Seek will be available.

- QFile example



<FIGURE> Example

Examples to be discussed this time are file open, write, and close functions. Click the [File Open] button, as shown in the figure above, to select one of the files from the file dialogue. The user then opens the file of their choice and reads the data from the file using the QTextStream class. Read the read data line by line and print it on the console using qDebug().

Click the [File Write] button to move the file to the last time using the view() member function of the QFile class, and then write the "Hello World" string to the file. Click the File Close button to close the file. The following example is the widget.h source

Jesus loves you.

code.

```
#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();
};

};
```

This class constructor initializes the mFile object. In addition, when a file has a close signal, it calls the slotAboutToClose() Slot function. The following example source code is the widget.cpp source code.

```
#include "widget.h"
#include "ui_widget.h"
#include <QDebug>
#include <QTStream>
#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()));
```

Jesus loves you.

```
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 << " Add file last message.\n";
}

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

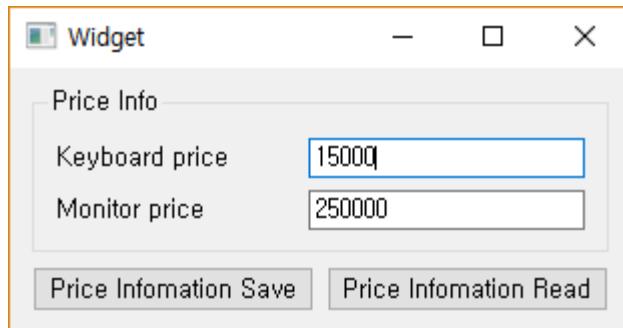
void Widget::slotAboutToClose()
{
```

Jesus loves you.

```
qDebug() << Q_FUNC_INFO;  
}
```

- Example using QFile class and QDataStream

This example is an example of using the QDataStream class. Click the [Price Information Save] button to save the keyboard and monitor prices to a file, as shown in the example execution screen in the left-hand picture.



<FIGURE> Example

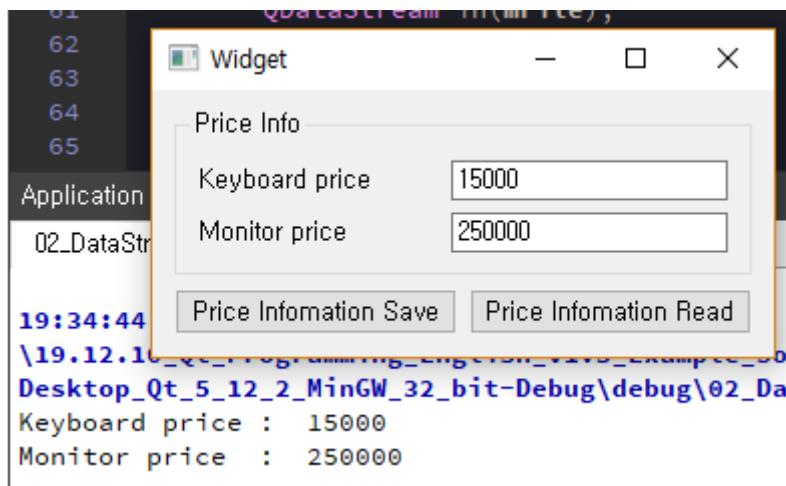
When saving a file on a TEXT basis, as shown in the previous example, "15000" is stored as a letter, and 6 bytes is required to store the monitor price "250000". Therefore, if saved as a string, it totals 11 bytes.

However, if you convert a string of keyboard and monitor prices into numbers, a total of 8 bytes is sufficient to store each value. This is because it is possible to save using two int-type 32bit (4Byte) variables

As with this method, data transmission and reception of defined protocols in binary data formats is often the case for heterogeneous data communication Advantages in this case are to use only as much storage space as is necessary, thereby minimizing unnecessary data waste.

For example, in embedded environments such as UART and I2S, it is often used for interprocess communication or for data communication between remote devices between these models. Then, click the [Price Information Read] button on the GUI to READ the data from the file using QDataStream. Then, the read data is printed to the console using the qDebug() function.

Jesus loves you.



<FIGURE> Example console message

The following example source code is the widget.h header file source code.

```
#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();
};
```

Click the [Price Information Save] button to call the slotPbtFileSave() Slot function. Click the [Price Information Read] button to call the slotPbtFileRead() Slot function. Next is the widget.cpp source code.

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

Jesus loves you.

```
#include <QDataStream>
#include <QMessageBox>

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

    connect(ui->pbtSave, SIGNAL(pressed()),
            this, SLOT(slotPbtFileSave()));
    connect(ui->pbtFileRead, SIGNAL(pressed()),
            this, SLOT(slotPbtFileRead()));

    mFile = new QFile();
}

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

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

    mFile->setFileName(fileName);
    if(!mFile->open(QIODevice::WriteOnly | QIODevice::Truncate))
    {
        qDebug() << "File open fail.";
        return;
    }
    else
    {
        qint32 keyboardPrice = ui->leKeyboard->text().toInt();
        qint32 monitorPrice = ui->leMonitor->text().toInt();

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

        mFile->close();
    }
}
```

Jesus loves you.

```
    }

}

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

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

        mFile->close();

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

Example - Ch03 > 08_File_Stream > 02_DataStream

3.9. Qt Property

Property system is provided by c++ property system and similar features. Sets the object and gets the value in the object is property if used. For example, some Gets or sets a value like The following example source code in a particular class, let's look at the case.

```
class Person : public Q0bject
{
    Q_OBJECT
public:
    explicit Person(Q0bject *parent = nullptr);

    QString getName() const {
        return m_name;
    }

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

private:
    QString m_name;
};
```

As shown in the example source code above, the class Person provides the functions of getName() and setName() members declared as public accesses. Returns the value of the getName() member variable m_name. The setName() member function is a function that sets the m_name value. Declare the object of the class Person and use it as follows.

```
Person goodman;

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

Person Class objects were declared and variable values were set using the setName() member function. It is an example source code that obtains a value declared as a setName() member function as a getName() member function and outputs it to the

Jesus loves you.

Console window. Let's take a look at the source code as a Qt example, the Property System. Let's add the Q_PROPERTY macro to the Person class, as shown in the following sources.

```
...
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;
};

...
```

As shown in the example source code above, the `getName()` and `setName()` member functions were registered in the `Q_PROPERTY` macro. In addition, the member functions registered in the `Q_PROPERTY` macro can be used as follows.

```
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"
```

Although you can obtain or set the value of the `name` variable by accessing the

Jesus loves you.

getName() and setName() member functions, you can use the Q_PROPERTY macro to obtain and set the value using setProperty() and property().

The Q_PROPERTY macro in the class does not seem very useful. However, it is very important when using QML. When the UI is developed with QML and functions are developed with C++, data between QML and C++ is often exchanged. The Q_PROPERTY macro must be used to obtain or change the value of the variable C++ from QML. In addition to the method covered by the example source code, the Q_PROPERTY macro offers a number of options.

```
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 keywords can specify values to read from READ keywords. For example, in the example source code above, you can specify the declared m_name variable in the private access limiter as follows.

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

Of the keywords provided by the Q_PROPERTY macro, NOTIFY keywords can be signal-specified. Signals can be used as shown in the following example.

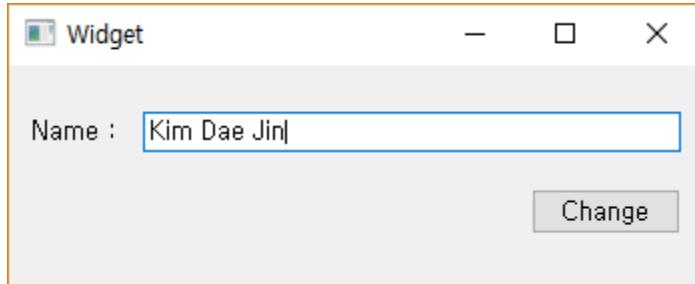
```
class Person : public QObject  
{  
    Q_OBJECT  
    Q_PROPERTY(QString name MEMBER m_name READ getName WRITE setName  
               NOTIFY nameChanged)
```

Jesus loves you.

```
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_PROPERTY example

The example to be discussed this time is to change the value by clicking the [Change] button, using the setProperty() member function provided by the QObject class.



<FIGURE> Example

In addition, when the setName() member function is called, a signal event specified by the NOTIFY keyword in the Q_PROPERTY macro will occur. The following example is the header source code for Widget.h.

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

Jesus loves you.

```
public slots:  
    void buttonPressed();  
    void nameChanged(const QString &n);  
  
private:  
    Ui::Widget *ui;  
    Person *goodman;  
};  
...
```

Clicking the [Change] button invokes the buttonPressed() Slot function. And the nameChanged() Slot function is called when a nameChanged() signal in the Person class occurs. The following example source code is the widget.cpp example source code.

```
...  
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;
```

Jesus loves you.

```
}
```

```
...
```

When the nameChanged() Slot function is called, the value is obtained using the property() function provided by the QObject class. The following example source code is the header file of the Person class.

```
...
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);
};

...
```

In the Person class, you can access getName() and setName() member functions using the Q_PROPERTY macro. The nameChanged() signal was then specified as the keyword NOTIFY in the Q_PROPERTY macro.

Therefore, when a signal event occurs using emit in the setName() member function, the associated Slot function of the Widget class is called.

Example - Ch03 > 09_Property > 01_Simple_Property.

3.10. QTimer

QTimer class can be invoked repeatedly based on the time you specify. The following example source code is an example using QTimer.

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

timer->start(1000);
```

The above example in source code, declaring QTimer, such as seeing an object in the class. And repeatedly calling the specified time to connect must be the () function shall be the function and slot.

Connect the first () function factor is QTimer specified object. The second is the signal. Timeout () specified by the time specified in the fourth factor has elapsed, silver signal update () The function is invoked slot.

The start() member function of the QTimer class of the last line is repeatedly invoked when the time specified in the connect() function has elapsed for the first factor. The first factor in the unit milliseconds (milliseconds).

Classes QTimer stop the member functions () to provide the ability to shut down the QTimer. Provides a singleShot() member function to ensure that the QTimer class is invoked only once, without repeating it. Singleshot members () function is available, such as watching in this case, a source code.

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

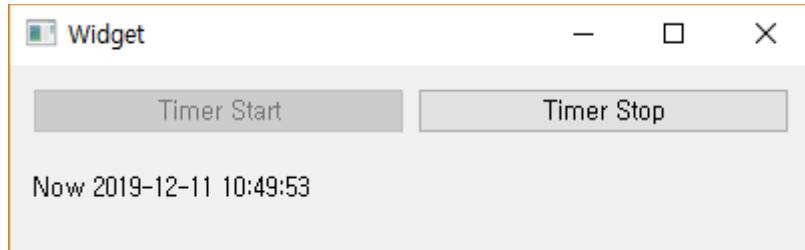
The first factor of the singleShot() member function is the elapsed time and the unit is Millie Second. The third factor can then specify the Slot function to be called after the time of the first factor.

- QTimer example

The following example calls the Slot function every one second. Click on the [Timer Start] button as shown in the following illustration to start the timer. Click the [Timer Stop]

Jesus loves you.

button to stop the timer.



<FIGURE> QTimer example screen

The current time at the bottom of the example run screen is an example of obtaining the current time from the system and outputting it to the QLabel widget in a cycle of 1000 milliseconds (one second) specified in QTimer. The following example source code is the `widget.h` source code.

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

private:
    Ui::Widget *ui;
    QTimer *m_timer;

public slots:
    void startPressed();
    void stopPressed();
    void elapsedTime();
};

...
```

`startPressed()` is a Slot function called by clicking the Start Timer button. `stopPressed()` is a Slot function called by clicking the [Timer Stop] button. `elapsedTime()` is called after the time specified in QTimer has elapsed. The following example is the `widget.cpp` source code.

```
...
```

Jesus loves you.

```
Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pbtStart, &QPushButton::pressed, this, &Widget::startPressed);
    connect(ui->pbtStop,  &QPushButton::pressed, this, &Widget::stopPressed);

    ui->pbtStart->setEnabled(true);
    ui->pbtStop->setEnabled(false);

    m_timer = new QTimer();
    connect(m_timer, &QTimer::timeout, this, &Widget::elapsedTime);
}

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

void Widget::startPressed()
{
    ui->pbtStart->setEnabled(false);
    ui->pbtStop->setEnabled(true);
    m_timer->start(1000);
}

void Widget::stopPressed()
{
    ui->pbtStart->setEnabled(true);
    ui->pbtStop->setEnabled(false);
    m_timer->stop();
}

void Widget::elapsedTime()
{
    QDateTime curr = QDateTime::currentDateTime();
    QString timeStr = curr.toString("Now yyyy-MM-dd hh:mm:ss");
    ui->leCurrentTime->setText(timeStr);
}
```

Example - Ch03 > 10_QTimer > 01_Timer

3.11. QThread

Qt offers QThread classes to support Thread. If you use Thread using the QThread class, synchronization may be required in certain situations. Synchronization can refer to variables referred to by multiple users in a thread environment, with one user changing the value of a variable, which can refer to the wrong value for another user.

Therefore, it provides a function to prevent other users from referring to or changing variables from the start to the end of the source code of a particular function operated by the thread, so that they do not refer to any false values before the variable changes.

These features are called synchronization features, and Qt provides a QMutex class to support synchronization. The following example source code is part of the class that inherited and implemented the QThread class.

```
#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);
        }
    }

    MyThread(int n);

private:
    bool m_threadStop;
```

Jesus loves you.

```
QMutex m_mutex;  
};  
...
```

The example shown in the example source code above is an example of implementing MyThread class using QThread. The run() function is a virtual member function inherited from QThread and can implement the function that you want to operate from Thread.

Since the run() function is an internally implemented function, invoking the start() function provided by QThread outside the class automatically invokes the run() function.

```
MyThread *thread = new MyThread(this);  
  
thread->start();
```

Sometimes the run() function is declared public and the run() function is called externally. In this case, the run() function does not operate as in the thread . That is, be careful not to call run() immediately, since it works like a normal function, not a thread. The QMutex class used in the run() function is a function for synchronization.

If the lock() member function provided by the QMutex class is declared to start synchronization and the last section of synchronization is declared through the unlock() member function, the variables declared within this interval cannot be accessed until the external class is called the unlock() member function, as shown in the example source code above.

For example, suppose you have 10 current connections in a network application. And if you have 10 current users, suppose you have saved the value to an int common variable called users. If the number of new accesses increases to 11, a process that checks the current accessor at the same time at which the user variable value is increased by one may give the wrong current accessor information to another accessor at the wrong.

Therefore, if synchronization is used to increase the value of the user variable by 1 for the source code interval, which increases the value of the user variable, the user variable is treated to wait in the queue so that the user variable is not changed or referenced.

If synchronization is required, the QMutex class is useful, but it is recommended to use it only when necessary because too often use may cause the program to block and cause a pause.

Jesus loves you.

- Thread Example satisfying Reentrancy and Thread-Safety

Reentrancy means that when two or more threads are in operation, one thread is performed and then the next thread is performed, regardless of the order in which the threads are performed.

Thread-Safety means using mechanisms such as Mutex to ensure stability when approaching shared memory areas, such as Static or Heap memory areas, in situations where two or more threads are operating. The following example creates two threads of the source code and implements threads that do not take into account Reentrancy and Thread-Safety.

```
static QMutex mutex;
static QWaitCondition incNumber;
static int numUsed;

class Producer : public QThread
{
    Q_OBJECT
public:
    void run() override {
        for(int i = 0 ; i < 10 ; i++) {
            sleep(1);
            ++numUsed;
            qDebug("[Producer] numUsed : %d", numUsed);
        }
    }
};

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

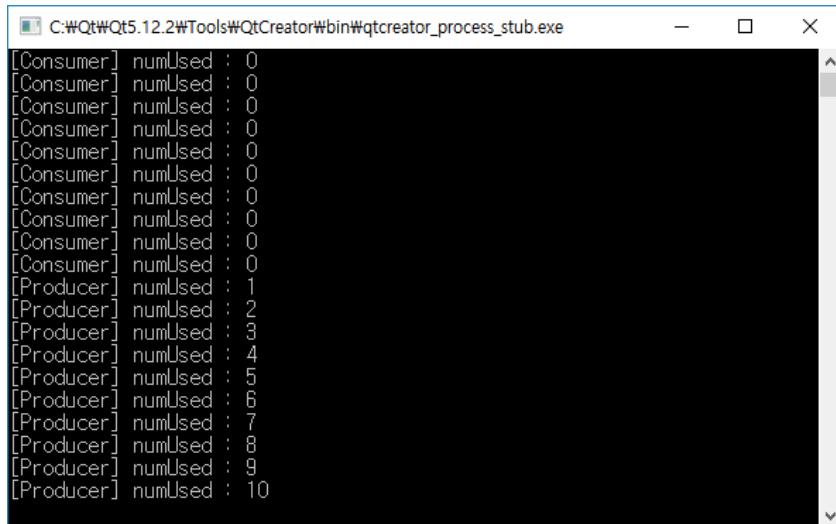
Jesus loves you.

```
Consumer() { }  
};
```

As shown in the example source code above, implement the Producer and Consumer classes and try to write them in main.cpp as follows:

```
#include <QCoreApplication>  
#include "mythread.h"  
  
int main(int argc, char *argv[])  
{  
    QCoreApplication a(argc, argv);  
  
    Producer producer;  
    Consumer consumer;  
    producer.start();  
    consumer.start();  
  
    return a.exec();  
}
```

Declaring two implemented Thread Class objects in the source code above and invoking the start() member function provided by QThread will initiate the Thread as shown in the following figure.



```
[Consumer] numUsed : 0  
[Producer] numUsed : 1  
[Producer] numUsed : 2  
[Producer] numUsed : 3  
[Producer] numUsed : 4  
[Producer] numUsed : 5  
[Producer] numUsed : 6  
[Producer] numUsed : 7  
[Producer] numUsed : 8  
[Producer] numUsed : 9  
[Producer] numUsed : 10
```

<FIGURE> 예제 실행 화면

As shown in the example run screen, the run() function of the Consumer class will be performed first and then the run() function of the Producer class.

Jesus loves you.

The following example is a Thread implementation that satisfies Reentrancy and Thread-Safety. Let's modify the Consumer and Producer classes implemented in the previous example as follows.

```
#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

    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
    void run() override {
        for(int i = 0 ; i < 10 ; i++) {
            mutex.lock();
            incNumber.wait(&mutex);
```

Jesus loves you.

```
    mutex.unlock();

    qDebug("[Consumer] numUsed : %d", numUsed);
}
}

public:
Consumer() { }

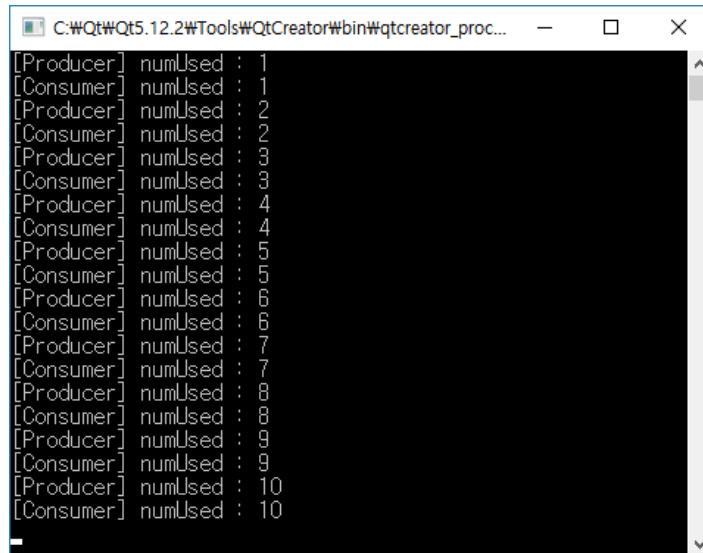
};

#endif // MYTHREAD_H
```

In the Producer and Consumer classes that were written in the previous example source code, the Thread operates regardless of the execution between the two threads. However, the Producer and Consumer classes modified are examples of Thread classes that satisfy Reentrancy and Thread-Safe.

Producer and Consumer Thread work when numUsed values change. That is, Consumer is Waited by the wait() member function of the QWaitCondition class. Increase the numUsed value by 1 in the Producer class and wakeAll() member function in the QWaitCondition class to wake up the Consumer class.

In this way, it is easy to implement a thread that satisfies two or more threads with Reentrancy and Thread-Safety.



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

<FIGURE> Exampe

Jesus loves you.

Example - Ch03 > 11_Thread > 02_WaitCondition

- Thread implementation using QtConcurrent class

Qt provides a QtConcurrent class that enables the implementation of Multi Threads rather than the implementation of Thread using QThread. The QtConcurrent class can behave like a thread without creating a thread class. The following example source code is an example source code using QtConcurrent.

```
#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);

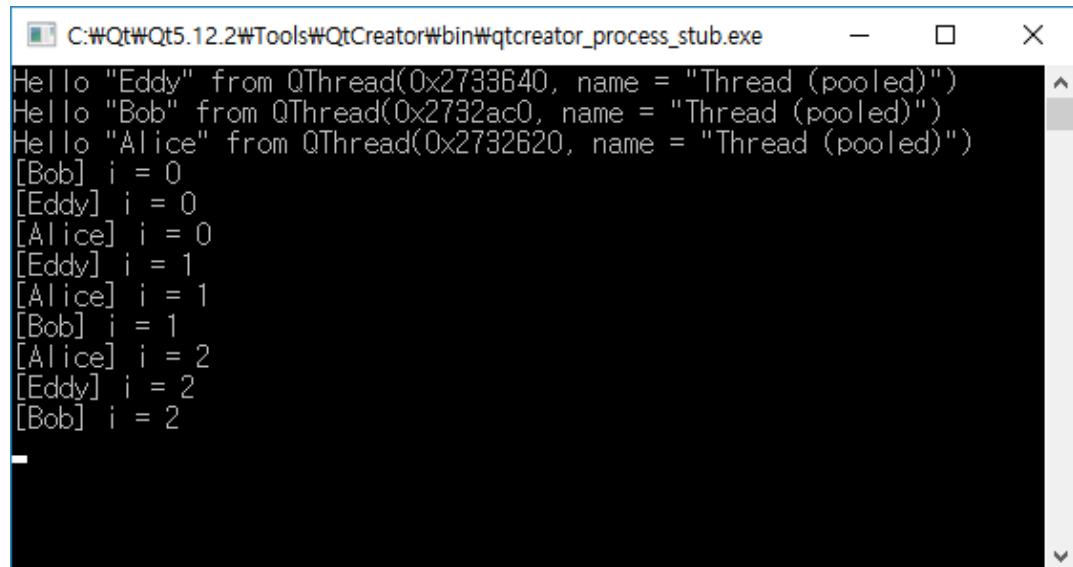
    QFuture<void> f1 = QtConcurrent::run(hello, QString("Alice"));
    QFuture<void> f2 = QtConcurrent::run(hello, QString("Bob"));
    QFuture<void> f3 = QtConcurrent::run(hello, QString("Eddy"));
}
```

Jesus loves you.

```
f1.waitForFinished();
f2.waitForFinished();
f3.waitForFinished();

    return a.exec();
}
```

Specifies the function name in the first factor of the run() member function of the QtConcurrent class, as shown in the example source code above. Then, you can operate the function like Thread.



The screenshot shows a terminal window titled 'C:\Qt\Qt5.12.2\Tools\QtCreator\bin\qtcreator_process_stub.exe'. The window displays the following text:

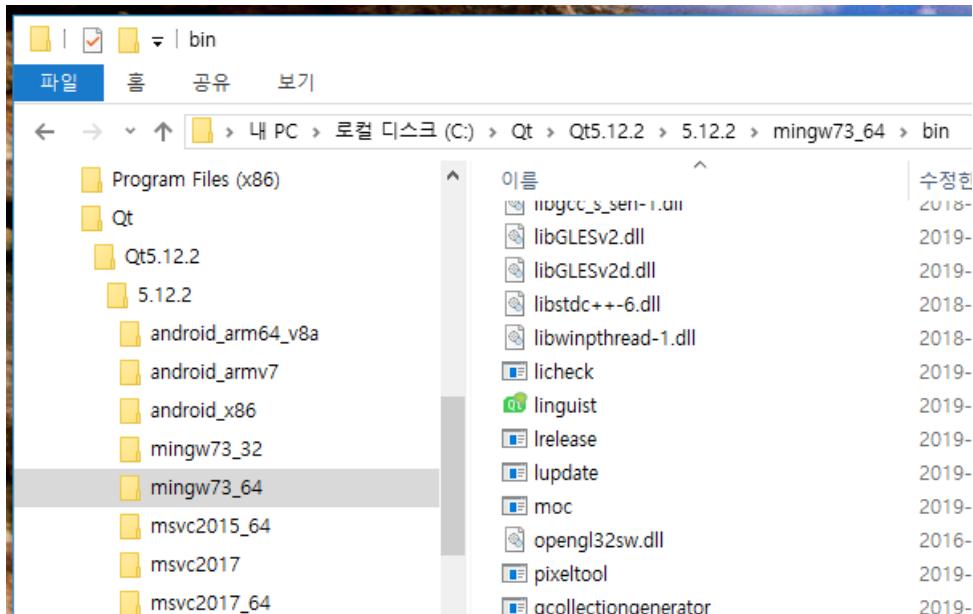
```
Hello "Eddy" from QThread(0x2733640, name = "Thread (pooled)")
Hello "Bob" from QThread(0x2732ac0, name = "Thread (pooled)")
Hello "Alice" from QThread(0x2732620, name = "Thread (pooled)")
[Bob] i = 0
[Eddy] i = 0
[Alice] i = 0
[Eddy] i = 1
[Alice] i = 1
[Bob] i = 1
[Alice] i = 2
[Eddy] i = 2
[Bob] i = 2
```

<FIGURE> QtConcurrent example screen

Example - Ch03 > 11_Thread > 03_WaitCondition

3.12. Multilingual support using Qt Linguist Tool

The Qt Linguist tool is provided to support multilingualism and will be installed together with Qt installation. Qt Linguist provides a Qt Linguist compiled with each compiler when installing Qt. For example, if implemented using the MinGW 64bit version, the Qt Linguist tool should also use the MinGW 64 Bit version.



<FIGURE> Qt Linguist Tool Locations

You can see examples of using the `tr()` function, as shown in the following example source code: The `tr()` member function used by the creator of the Widget class is a member function of the `QObject` class for multilingual processing. "Button" was entered in the first factor of the `tr()` function.

```
Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    ui->pushButton->setText(tr("Button"));
    ...
}
```

If a word registered in the `tr()` member function is found in a multilingual file registered with the Qt Linguist tool, the word is automatically replaced with a word.

Jesus loves you.

Therefore, if you want to change to a different language, you must use the tr() function. Use the tr() function to cover examples of multilingual support. Create a project with the Widget class as the Base class and place the buttons, as shown in the example source code above. And use the tr() function as shown in the setText() member function of the QPushButton class, as shown in the above source code.

If you have added a source code as shown in the example source code above, add a file to the project file for multilingual support as follows: To add multiple languages to the project file, use the keyword TRANSLATIONS.

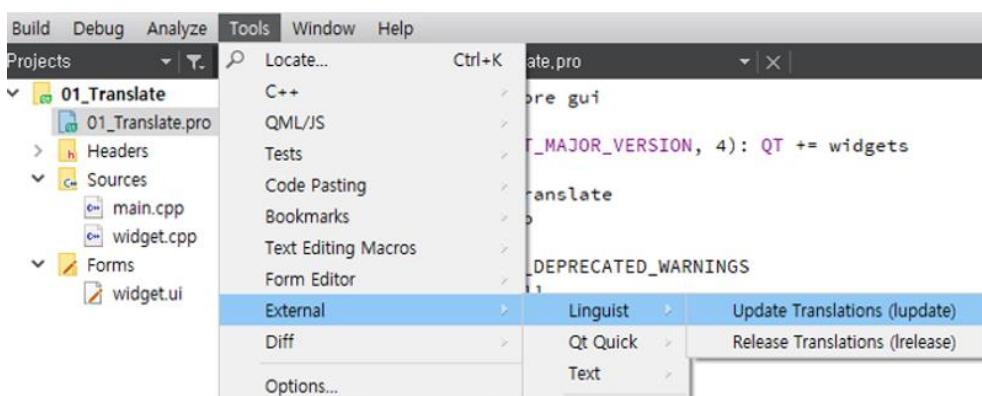
Add a multilingual file as shown in the following example source code: The file name of a multilingual file can be used with the desired file name. However, the file should have an extension of ".ts".

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

TARGET = 01_Translate
TEMPLATE = app
DEFINES += QT_DEPRECATED_WARNINGS
CONFIG += c++11

TRANSLATIONS = widget.ts
...
```

Enter the desired multi-lingual file name in the TRANSLATIONS keyword, as shown in the example project file above. In addition, the lupdate tool should be used to create the specified widget.ts file in the project file. Run the lupdate tool from the File menu of the Qt Creator tool as shown in the following figure. To run the lupdate tool, run the [Tools]>[External]> [Linguist] > [Update Transactions (lupdate)] menu.



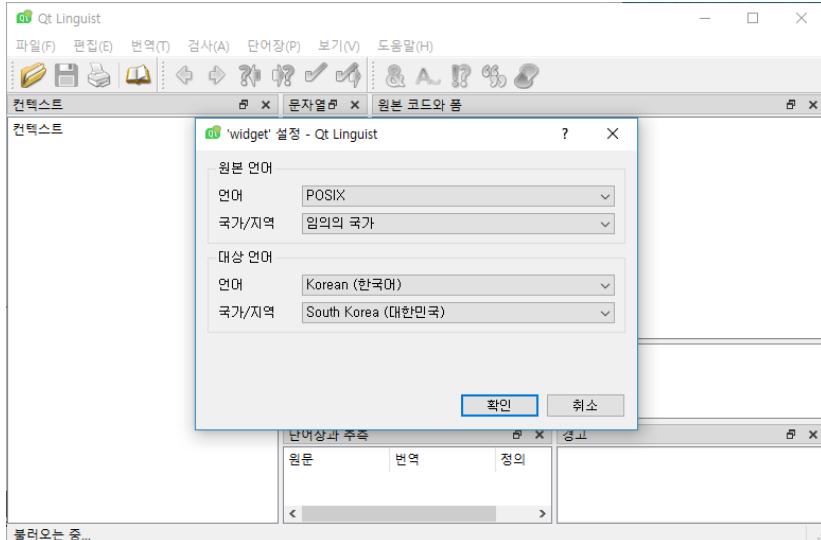
3.12. Multilingual support using Qt Linguist Tool

Jesus loves you.

<FIGURE> lupdate tool execution screen

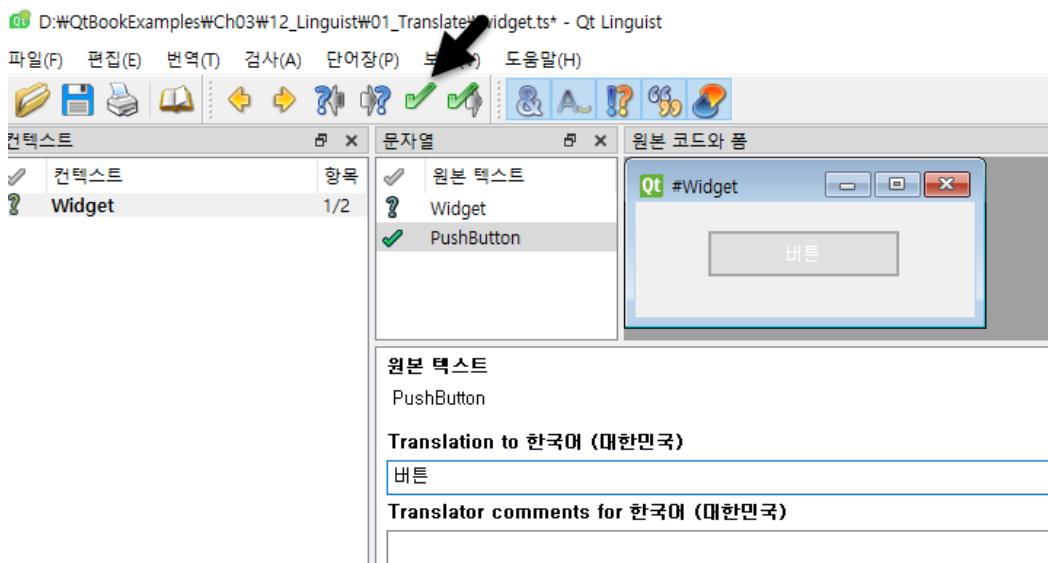
As shown in the figure above, the lupdate tool creates a widget.ts file in the directory where the project source code is located.

Use the Qt Linguist tool to open the generated widget.ts file. When you open the file, you can see the following dialogue window: Click the OK(확인) button at the bottom of the dialogue window, as shown in the following figure.



<FIGURE> Qt Linguist Tool screen

Click the [OK(확인)] button to see the Widget created in the project as shown in the following figure.

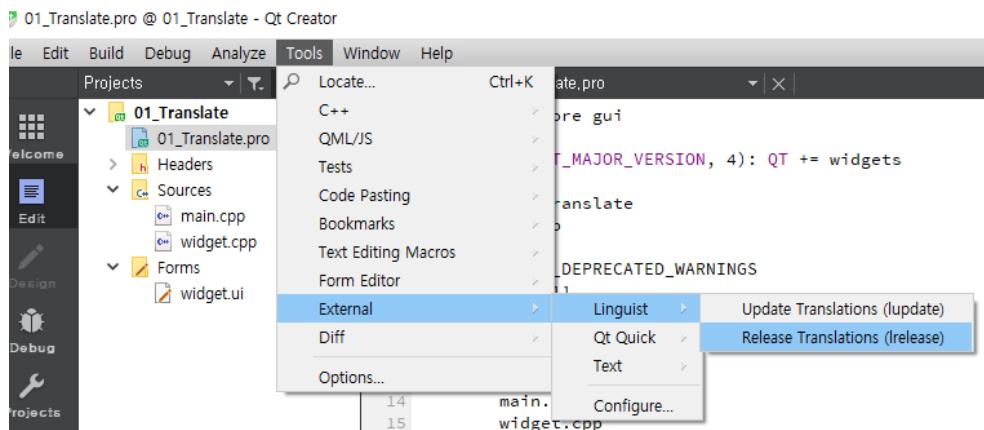


Jesus loves you.

<FIGURE> Qt Linguist Tool

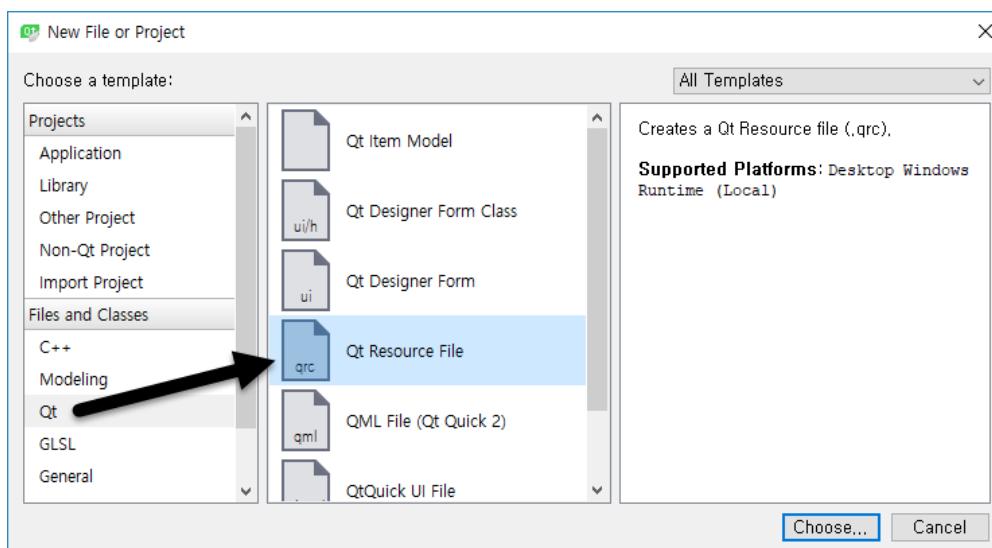
Select "PushButton" on the string screen and type "버튼(in English, Button)" in the lower window. Then click the check icon in the toolbar at the top.

Click on the check-shaped icon, as shown in the previous picture, and then save the edited file in the Qt Linguist tool. When the save is complete, return to the Qt Creator window and run the release tool. For release, perform the [Tools] [External] > [Linguist] > [Release Translations] menu of Qt Creator.



<FIGURE> Qt Linguist Tool

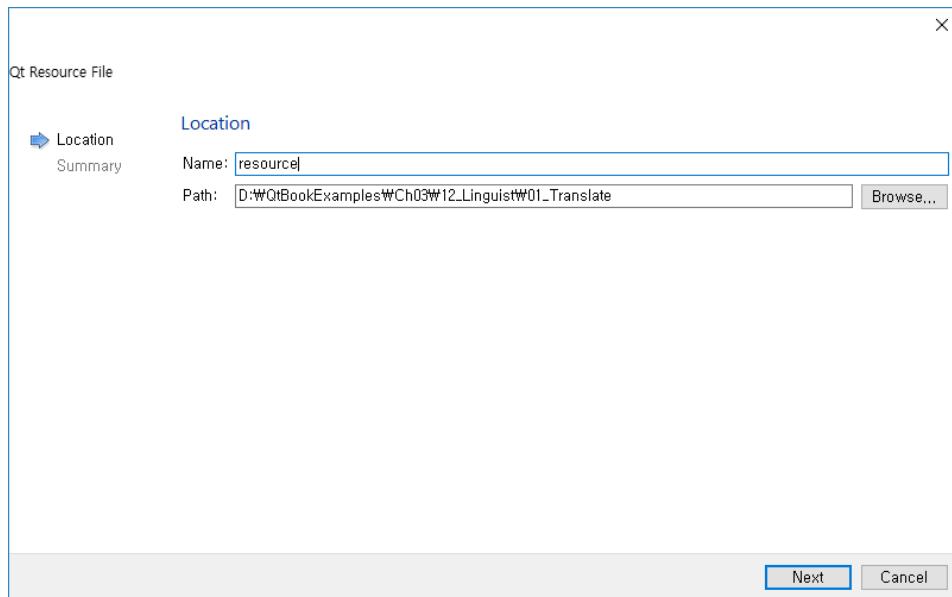
As shown in the figure above, the release tool can be run to see that the file was created in the directory where widget.ts is located. This file is a multi-lingual file created using the widget.ts file to actually use.



<FIGURE> Qt Resource File

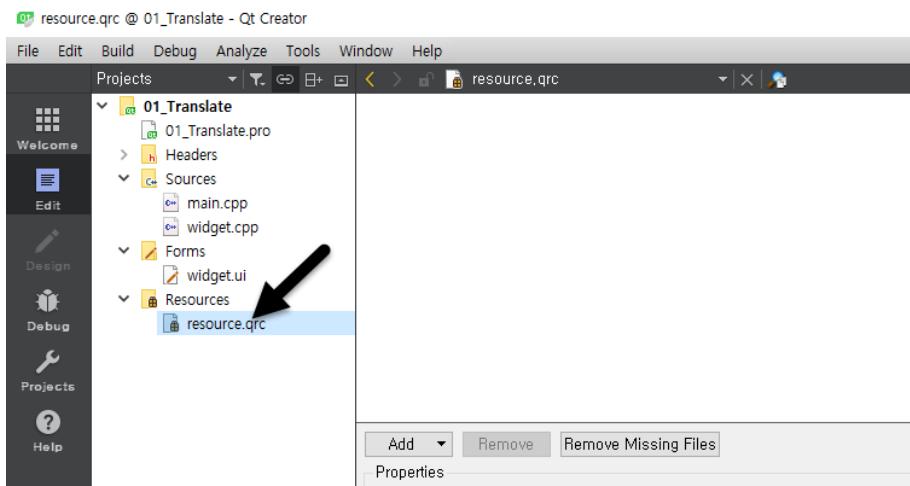
Jesus loves you.

Register the last generated file, Widget.qm, in the resource file to be distributed with the application build. To register in a resource file, click the [Ctrl + N] shortcut or [File] > [New File or Project] menu from the Qt Creator menu as shown in the following figure. Select Qt Resource File in the dialog window and click the [Choose] button at the bottom. On the next dialogue screen, type "resource" in the Name entry and click the [Next] button at the bottom as follows.



<FIGURE> Resource registration screen

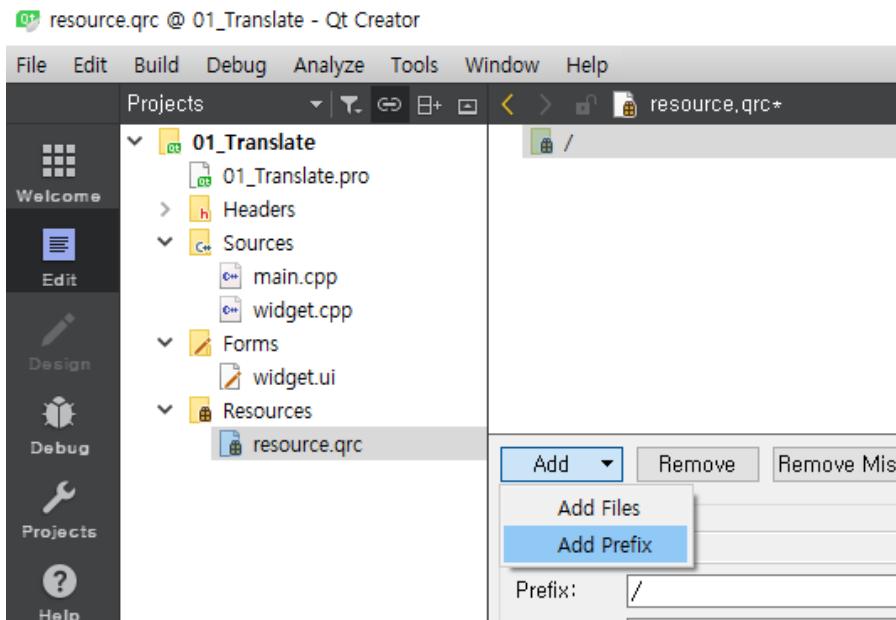
Double-click the resource.qrc file in the Qt Creator project navigation window, as shown in the following figure, to load the resource.qrc editor screen.



<FIGURE> Resource Edit Window

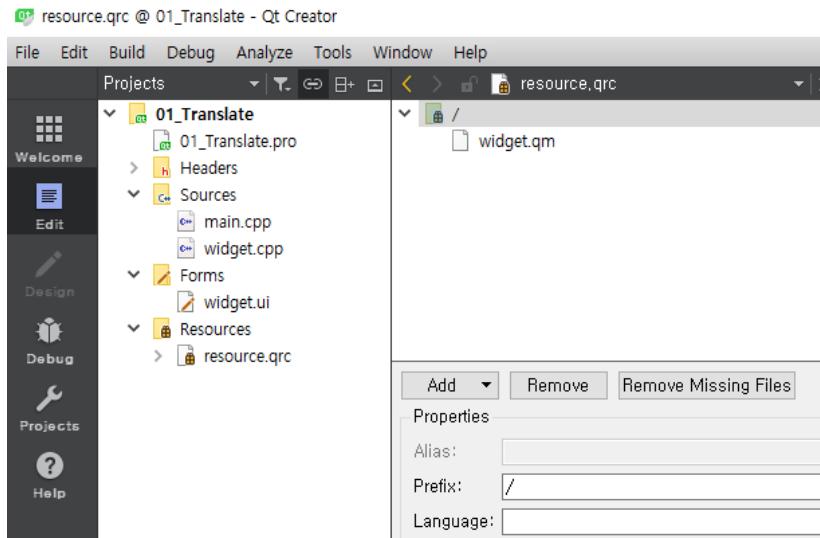
Jesus loves you.

Click the [Add] button in the Resource Edition window to register the generated widget.qm file. Before registering the widget.qm file, you must first register [Add Prefix] as shown in the following figure.



<FIGURE> Add Prefix registration screen

Click on [Add Prefix] as shown in the figure and type "/" into the Prefix entry at the bottom of the Resource Edition window. Then click the [Add Files] menu to register the widget.qm file.



<FIGURE> widget.qm registration screen

Jesus loves you.

Then add the source code to the main.cpp file as follows.

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

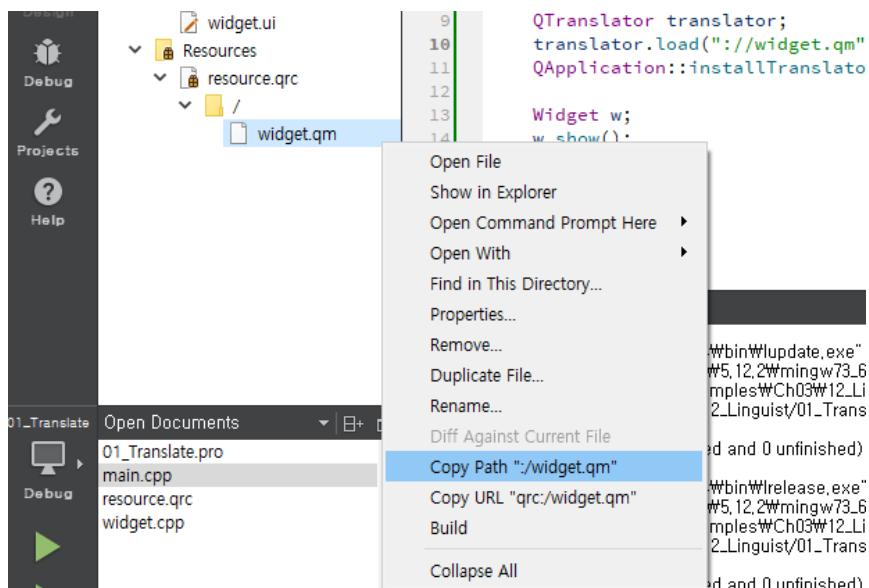
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    QTTranslator translator;
    translator.load(":/widget.qm");
    QApplication::installTranslator(&translator);

    Widget w;
    w.show();

    return a.exec();
}
```

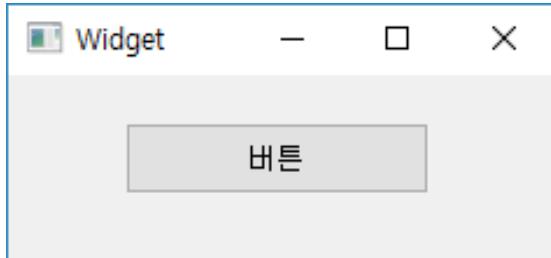
In the main.cpp source code file, the first factor of the transporter.load() function is copied by right-clicking on the registered resource file and clicking [Copy Path] or [Copy URL] in the context menu. Then paste into the first factor of the translator.load() function as shown in the source code.



<FIGURE> Copy Path selection screen

Jesus loves you.

And when you build and execute the changed source code, you can see that QPushButton's Text on the widget has been changed to "버튼" (e.g. "Button" -> "버튼")



<FIGURE> Example screen

The code added to the main.cpp source code must previously declare the widget to use multiple languages. If you create a Widget object and then add the multilingual source code that you added, it does not apply to Widget.

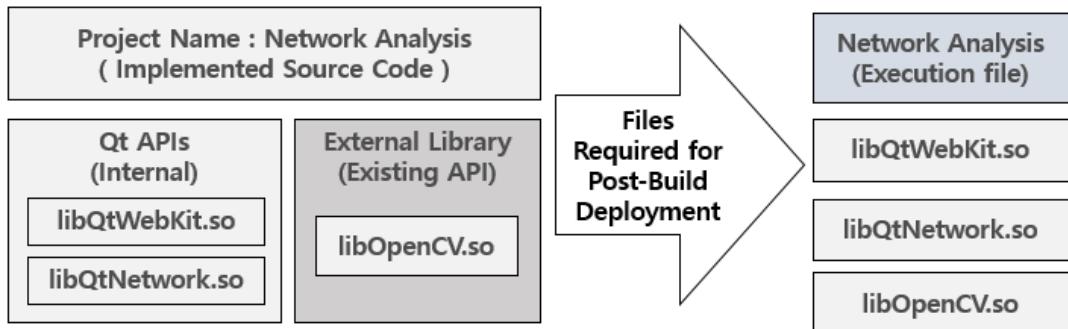
Therefore, be aware that you must declare the Widget object before it can be created.

Example - Ch03 > 12_Linguist > 01_Translate

3.13. Creating an external library

Qt can develop software by porting external libraries implemented with C/C++. For example, you can use a variety of external libraries, such as OpenCV, CUDA, OpenCL, and so on.

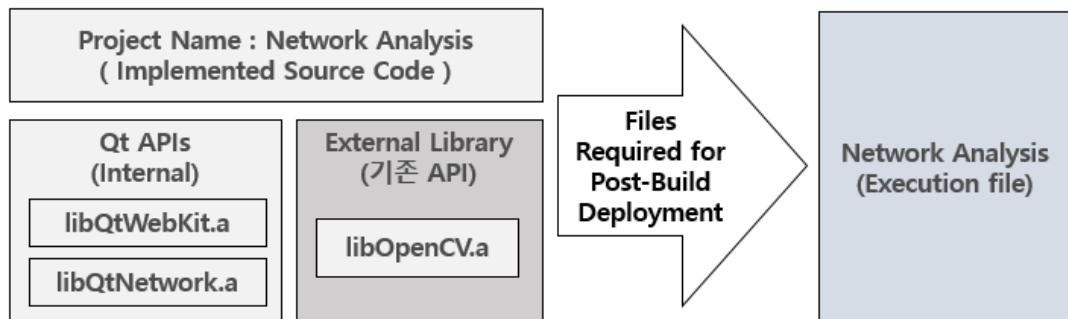
Qt is available in both shared and static libraries. If the software that you want to implement is implemented using external libraries, the software works only when you distribute the executable you have built and the shared library as well.



<FIGURE> Shared library

As shown in the figure above, a file with the so-called extension of the Linux operating system is a shared-style library file.

The Static library method combines libraries into executable files and a single file at the time of build. Therefore, if you have built it in a static fashion, you only need to distribute the executable at the time of deployment.



<FIGURE> Static library

As shown in the figure above, a library extension of a is used as an extension of the

Jesus loves you.

Static Library file in the Linux operating system. When using an external library in Qt, it should be built using the same compiler as the compiler that compiled.

For example, if you have built software developed with Qt into a MinGW-based compiler, you should also use a library built with a MinGW-based compiler.

There are also two MinGWs, 32Bit and 64Bit, and the software and libraries you want to implement must also use a library compounded in the same bit

For example, if the software compiler that you implement is built using the MinGW 32Bit compiler, you should also use a library that is compiled as a MinGW 32Bit compiler. To use an external library in Qt, you can specify the library in the project file as follows.

```
INCLUDEPATH += "C:/Intel/OpenCL/sdk/include"  
LIBS += -L"C:/Intel/OpenCL/sdk/lib/x64"  
LIBS += -lOpenCL
```

As shown in the example above, the INCLUDEPATH keyword can specify the location directory where the header file of the external library is located.

The LIBS keyword can specify the directory in which the library files are located, such as a file or a file. In front of the directory path, "-L" is the directory where the library is located.

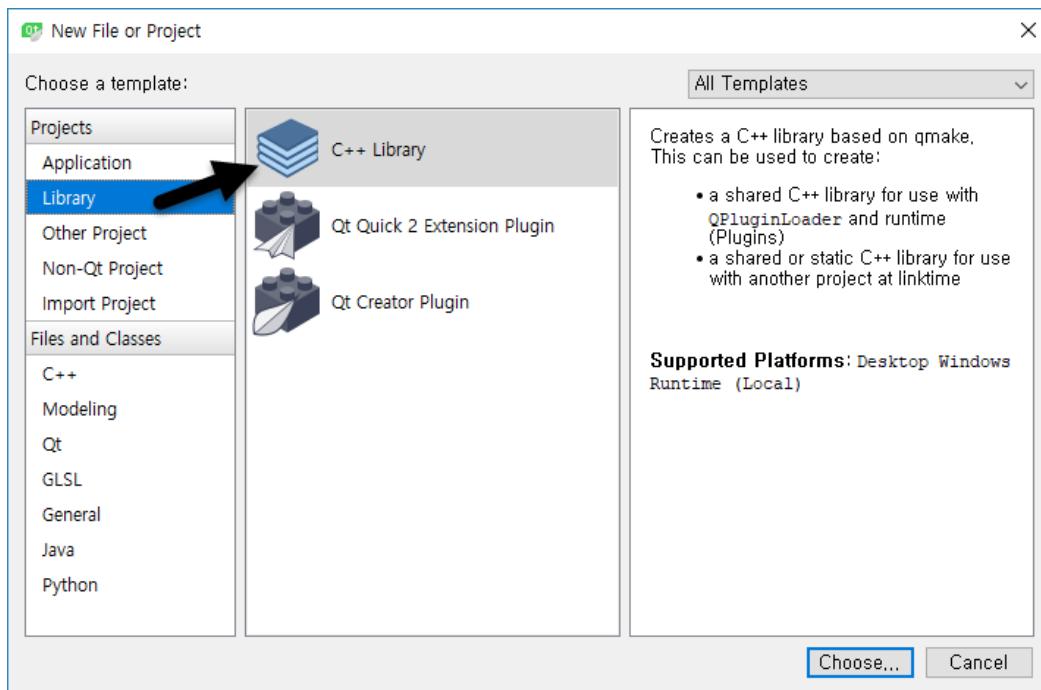
And the third line -lOpenCL can specify the shared library files to be used in practice. The "-l" option in front of the library name means that you want to specify the library file. In the directory located in the library file, the extension is so if Linux, or dll or lib for MS Windows.

- Implementing Qt libraries

In this example, we will create two Qt projects. The first project is a project for library implementation. The second project will use the libraries implemented in the first project.

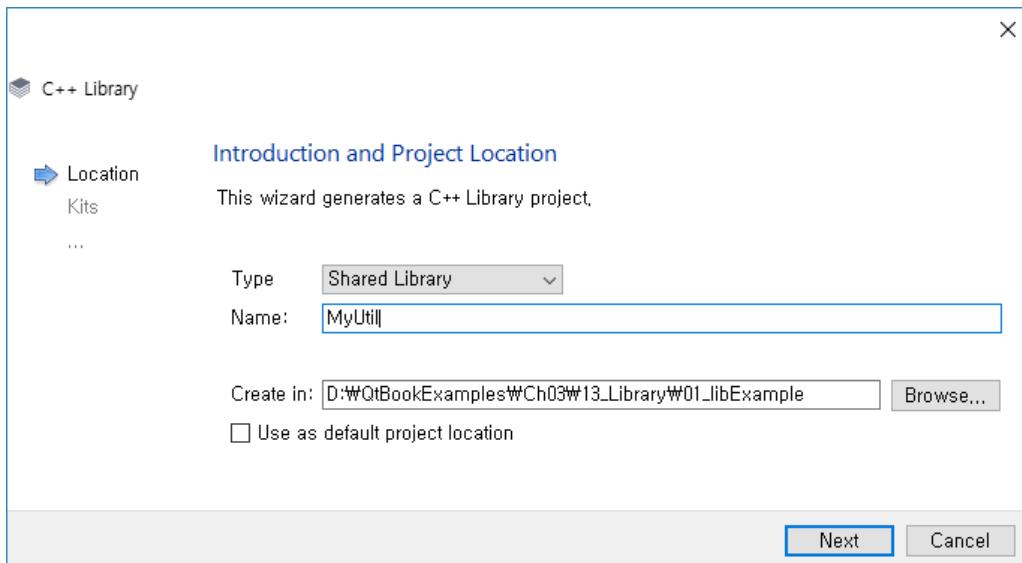
Create a new project in the Qt Creator Tool. In the Project Creation Dialog, select the Library item from the [Projects] tab in the upper left and select the C++ Library item from the list in the center as shown in the following figure below.

Jesus loves you.



<FIGURE> Create first library project screen

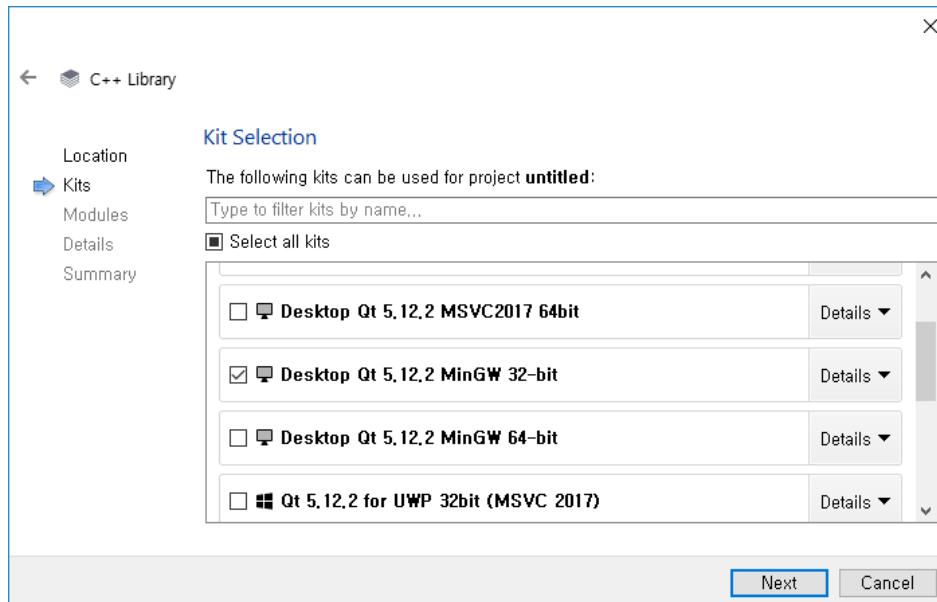
The next dialogue screen selects the library type, name, and location. The Type item selects the Shared Library item. In the Name entry, type MyUtil. The Create In entry specifies the library source code or the parent directory where the library project will be located.



<FIGURE> Create second project screen

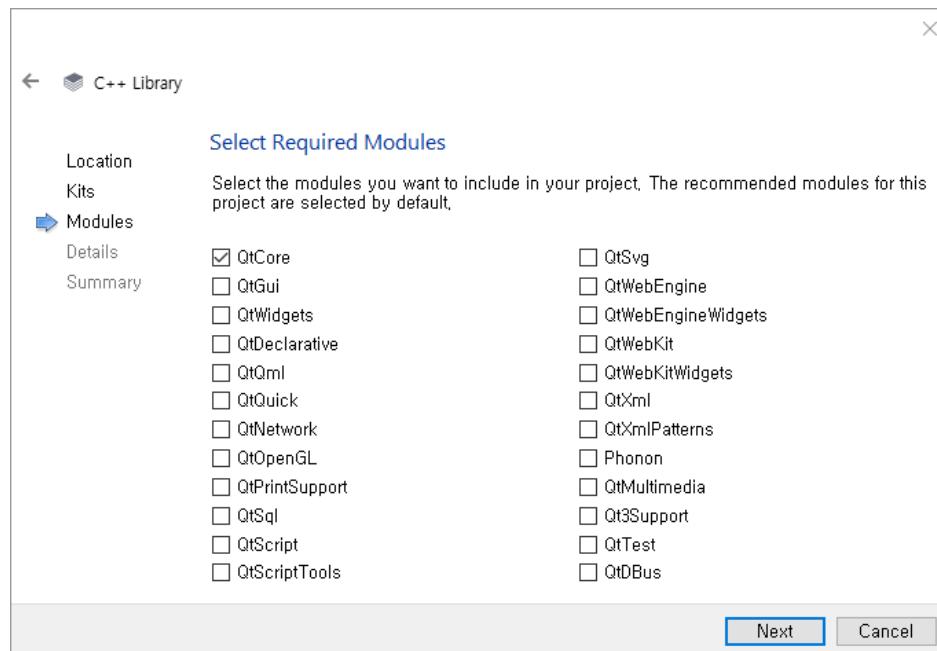
Jesus loves you.

Next, select the compiler to build. Select the MinGW 32-bit version as shown in the following figure.



<FIGURE> Compiler selection screen

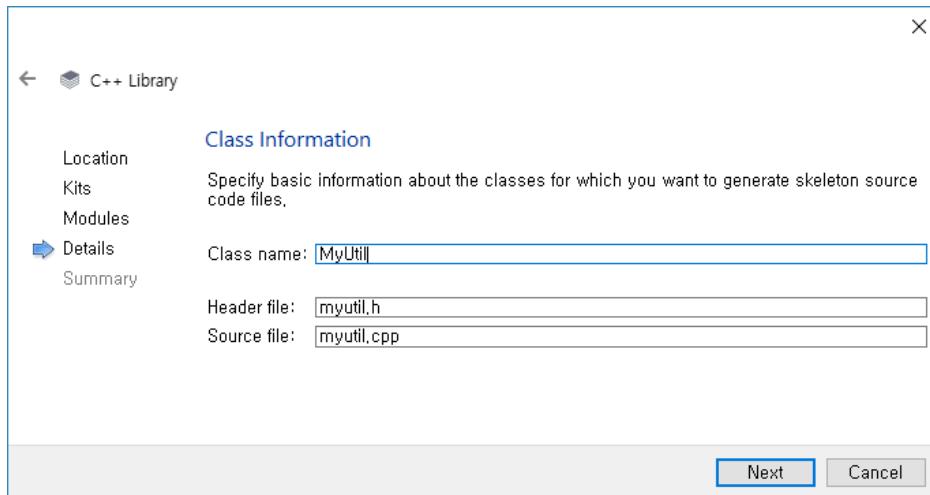
The next dialogue screen is the screen that selects the Qt module to use in the library. Only QtCore items will be used in this project.



<FIGURE> Select Qt library item to use screen

Jesus loves you.

Next, enter the information for the library class that you want to create. Class name is MyUtil, header file is myutil.h, and source code file is myutil.cpp.



<FIGURE> Class Information screen

The next dialogue will be completed by pressing the [Finish] button at the bottom. Let's take a look at the project as shown in the following example source code.

```
QT      -= gui

TARGET = MyUtil
TEMPLATE = lib

DEFINES += MYUTIL_LIBRARY
DEFINES += QT_DEPRECATED_WARNINGS
SOURCES += myutil.cpp

HEADERS += myutil.h myutil_global.h

unix {
    target.path = /usr/lib
    INSTALLS += target
}
```

General applications, projects and Library project, is the keyword template. General applications on keywords template app is specified, but libraries are set out by lib.

This factor will receive a value of the library has two. Two and a library that provides the ability to provide value plus the value of the implementation. Created a myutil h fill out

Jesus loves you.

the source code as follows : in the header file.

```
#ifndef MYUTIL_H
#define MYUTIL_H

#include "myutil_global.h"

class MYUTILSHARED_EXPORT MyUtil
{
public:
    MyUtil();
    qint32 getSumValue(qint32 a, qint32 b);
};

#endif // MYUTIL_H
```

The following adds the source code to the myutil.cpp source code file as follows.

```
#include "myutil.h"

MyUtil::MyUtil()
{
}

qint32 MyUtil::getSumValue(qint32 a, qint32 b)
{
    return a + b;
}
```

Add and build the source code as shown in the example source code above. A library would have been built if an error had not occurred and been built normally.

Ch03 > 13_Library > 01_libExample > build-MyUtil-Desktop_Qt_5_12_2_MinGW_32_bit-Debug > debug

이름	수정한 날짜	유형	크기
libMyUtil.a	2019-05-10 오후...	A 파일	8KB
moc_myutil.cpp	2019-05-10 오후...	C++ Source file	3KB
moc_myutil.o	2019-05-10 오후...	O 파일	321KB
moc_prelude.h	2019-05-10 오후...	C++ Header file	15KB
MyUtil.dll	2019-05-10 오후...	응용 프로그램 확장	620KB
myutil.o	2019-05-10 오후...	O 파일	313KB

<FIGURE> Build directory

As you can see in the previous picture, you will see that the libMyUtil.a and MyUtil.dll

Jesus loves you.

files were created under the built directory. This time, you create a project to use the built library. The project to be created creates a GUI-based widget.

When creating a project, the same compiler that built the library should be used as previously stated. Select the MinGW 32 Bit version of the project you want to build. It then adds the library to be used in the project file, as shown in the following example.

```
QT      += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = AppWithLibrary
TEMPLATE = app

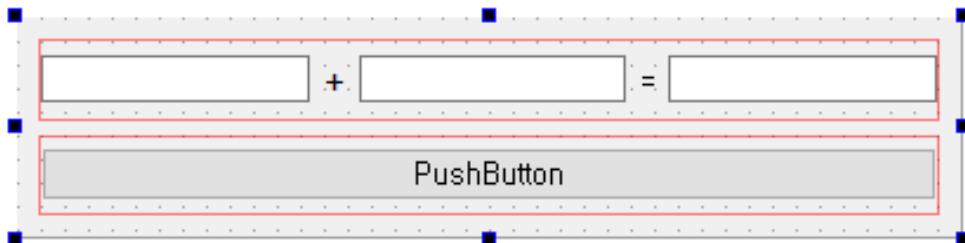
SOURCES += main.cpp\
           widget.cpp

HEADERS += widget.h

FORMS    += widget.ui

LIBS += -L$$PWD/../../[Directory name]/debug/ -lMyUtil
INCLUDEPATH += $$PWD/../../MyUtil
DEPENDPATH += $$PWD/../../MyUtil
```

Place the widget on the GUI as shown in the figure below.



<FIGURE> GUI Widget Layout Screen

As shown in the figure above, clicking the PushButton button will add two input values and output them to the resulting QLineEdit widget. Create the source code as shown in the header file below.

```
#ifndef WIDGET_H
#define WIDGET_H
```

Jesus loves you.

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

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    MyUtil *myUtil;

    Ui::Widget *ui;

private slots:
    void slotBtnClick();

};

#endif // WIDGET_H
```

Next, write the widget.cpp source code.

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

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

    myUtil = new MyUtil();
    connect(ui->pbtSum, SIGNAL(pressed()), this, SLOT(slotBtnClick()));
}

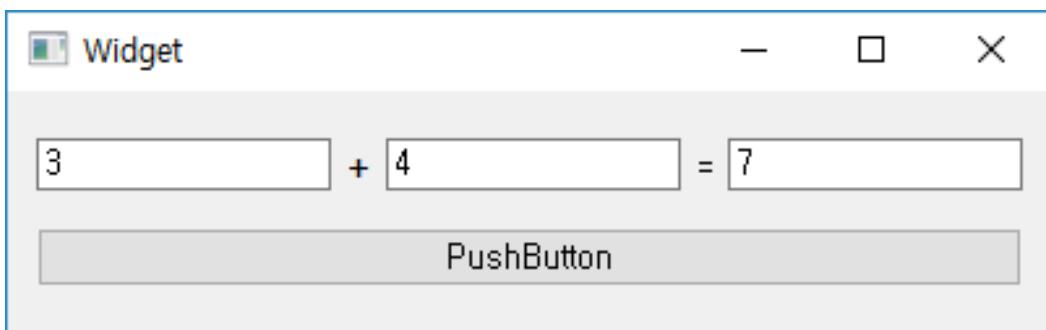
void Widget::slotBtnClick()
{
    qint32 arg1 = ui->leArg1->text().toInt();
```

Jesus loves you.

```
qint32 arg2 = ui->leArg2->text().toInt();

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

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



<FIGURE> Example screen

Enter the two integer values as shown in the figure above, then click the [PushButton] button to output the result values to the right.

Example - Ch03 > 13_Library > 01_libExample.

- Build with Library

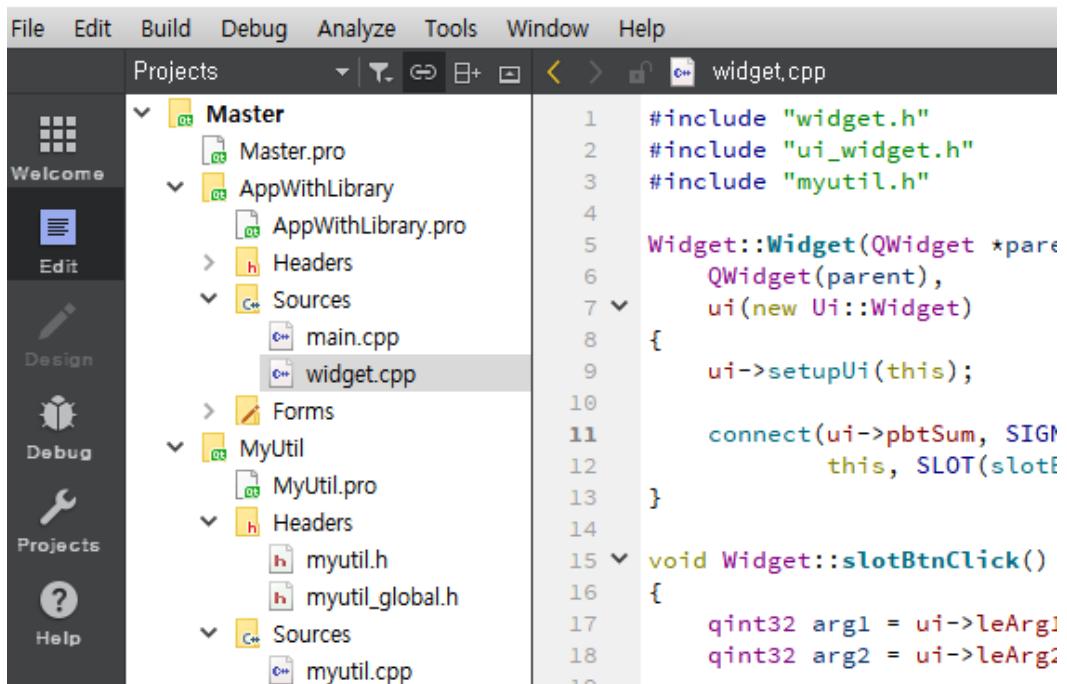
Library projects and application projects were built using separate Qt Creator tools. Assuming that there is a need to modify the library, there will be inconvenience in having to modify the library from the Qt Creator tool that opened the library project, then build it, then return to the Qt Creator window where the application project was opened, and apply the modified library.

However, Qt allows you to modify multiple projects in one Qt Creator window. This will allow the library and application source code to be simultaneously modified and applied in the same Qt Creator tool, provided that the library directory location on the project is correct as a benefit. The following figure shows two projects opening one Qt Creator

Jesus loves you.

tool.

widget.cpp (AppWithLibrary @ Master) - Qt Creator



The screenshot shows the Qt Creator interface with the following details:

- File Menu:** File, Edit, Build, Debug, Analyze, Tools, Window, Help.
- Projects View:** Shows three projects:
 - Master**: Contains `Master.pro`.
 - AppWithLibrary**: Contains `AppWithLibrary.pro`, `Headers`, `Sources` (containing `main.cpp` and `widget.cpp`), and `Forms`.
 - MyUtil**: Contains `MyUtil.pro`, `Headers` (containing `myutil.h` and `myutil_global.h`), and `Sources` (containing `myutil.cpp`).
- Code Editor:** The code editor displays the `widget.cpp` file with the following content:

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

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

    connect(ui->pbtSum, SIGNAL(clicked()), this, SLOT(slotBtnClick()));
}

void Widget::slotBtnClick()
{
    qint32 arg1 = ui->leArg1->text().toInt();
    qint32 arg2 = ui->leArg2->text().toInt();
    ui->labelResult->setText(QString::number(arg1 + arg2));
}
```

<FIGURE> The screen that opened multiple projects

To open several projects in Qt Creator, as shown in the figure above, create a project file as follows:

```
TEMPLATE      = subdirs

SUBDIRS      = MyUtil \
               AppWithLibray

MyUtil.file   = MyUtil/MyUtil.pro
AppWithLibray.file = AppWithLibrary/AppWithLibrary.pro
```

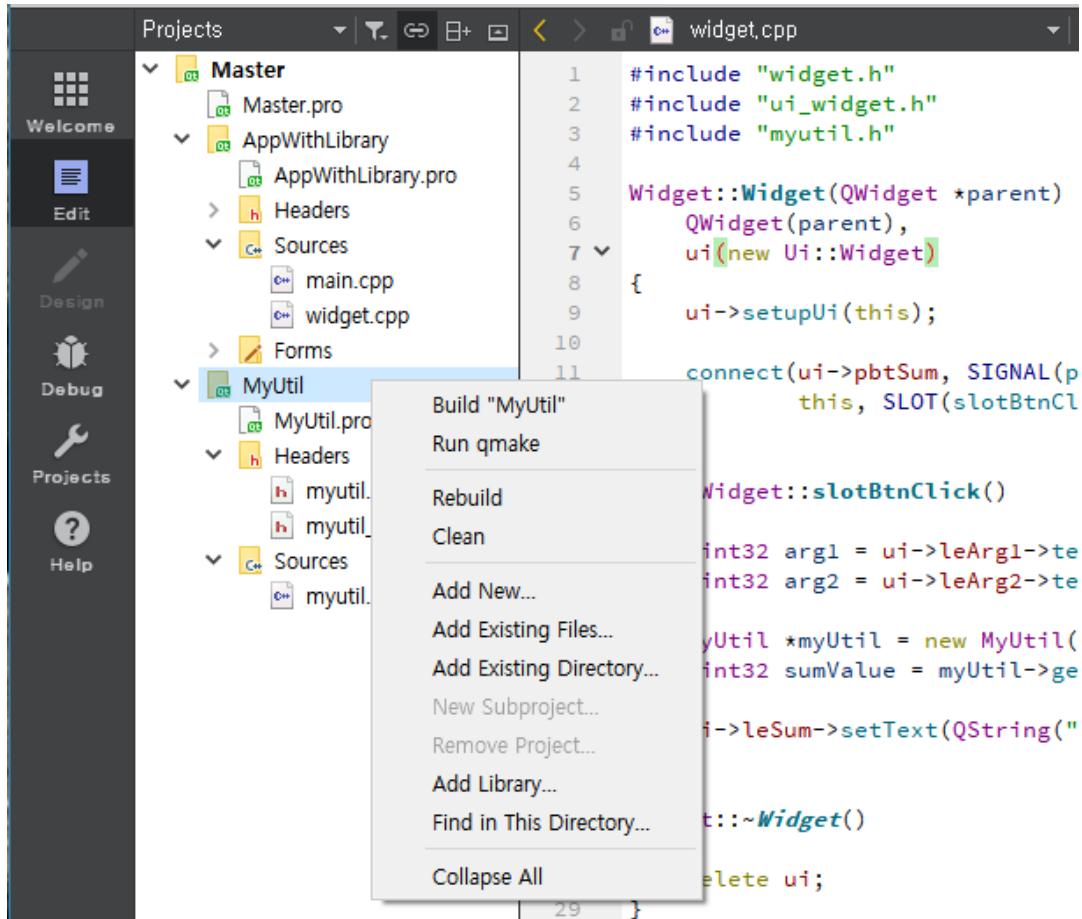
Create a project file(.pro) file as shown in the example above. By specifying two project files each, the project can be modified and built in one Qt Creator tool, which makes it very convenient to build.

For example, if you change the library source code, Qt Creator automatically builds the changed project, making it easier to build the library without having to build it separately.

However, you must first build the library separately because you must refer to the original

Jesus loves you.

library. To build a library separately, click on the library project and right-click again to create a menu as shown in the figure below.



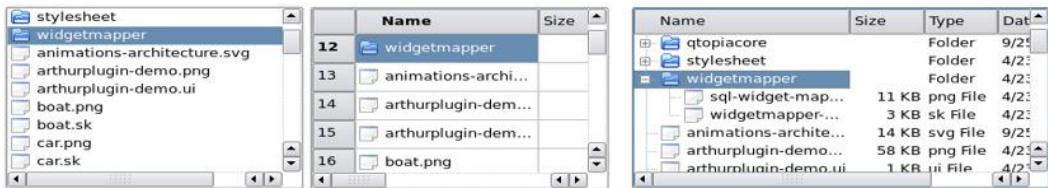
<FIGURE> Library Build Screen

As you see the picture on the Right Click the Library can build the project.

Example - Ch03 > 13_Library > 02_libExample

3.14. Model and View

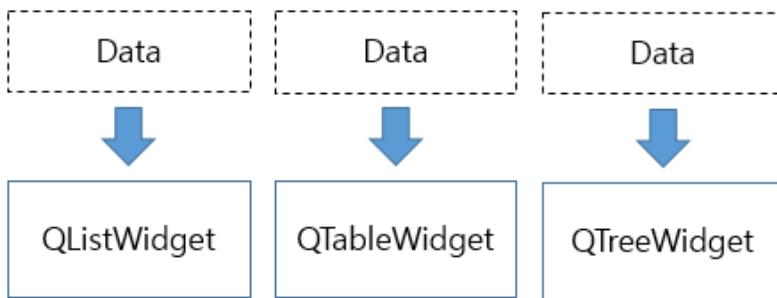
Qt provides various classes such as QListWidget, QTableWidget, QTreeWidget, QListView, QTableView, QTreeView, QTreeView, and QColumnView as widgets for displaying data in the same widget as shown in the following figure.



<FIGURE> Widgets for displaying widgets, such as tables.

QListWidget is the same form as QListView. A widget to display data from a column on multiple lines. However, QListWidget and QListView differ in inserting, modifying, and deleting data.

Classes using the word Widget at the end of the class provide a member function that allows data to be inserted/modified/deleted directly, as shown in the figure below.



<FIGURE> QListWidget, QTableWidget and QTreeWidget

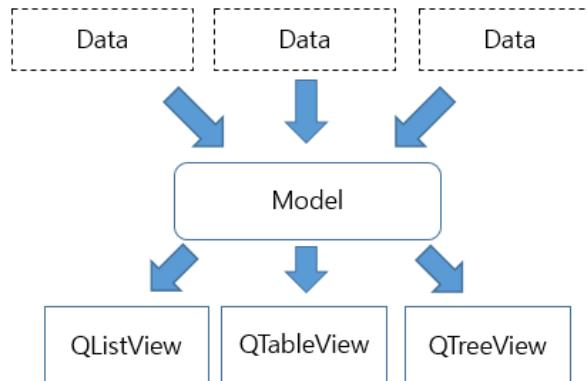
As shown in the figure above, widgets with the word Widget at the end of the class name all provide a membership function that enables direct data insertion/modification/deletion. For example, the QListWidget widget class provides insertItem().

The QTableWidget class provides a different member function for inserting data. In other words, because QListWidget has different ways of inserting data and QTableWidget has different ways of using it, it is important to learn how to insert the data provided by each

Jesus loves you.

class.

And widget classes that finally use the word View, such as QListView, QTableView, and QTreeView classes, can insert/modify/delete data using a medium called Model class without inserting/modifying/delete data using each member function.

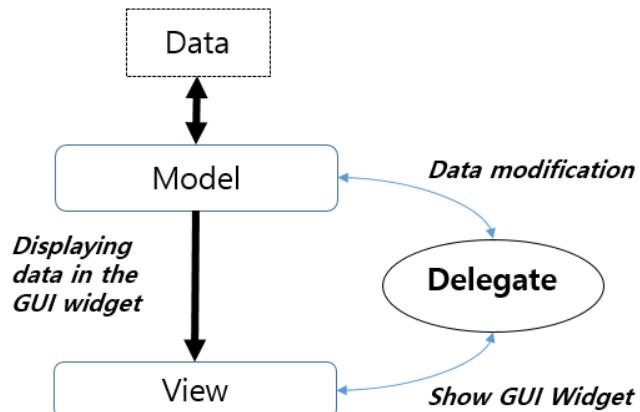


<FIGURE> Model / View architecture

As shown in the figure above, QListView, QTableView, and QTreeView classes use Model classes without the membership functions provided by each class to insert/modify/delete data into the widget.

This method has the advantage of using the same model, whether using QListView or using QTableView.

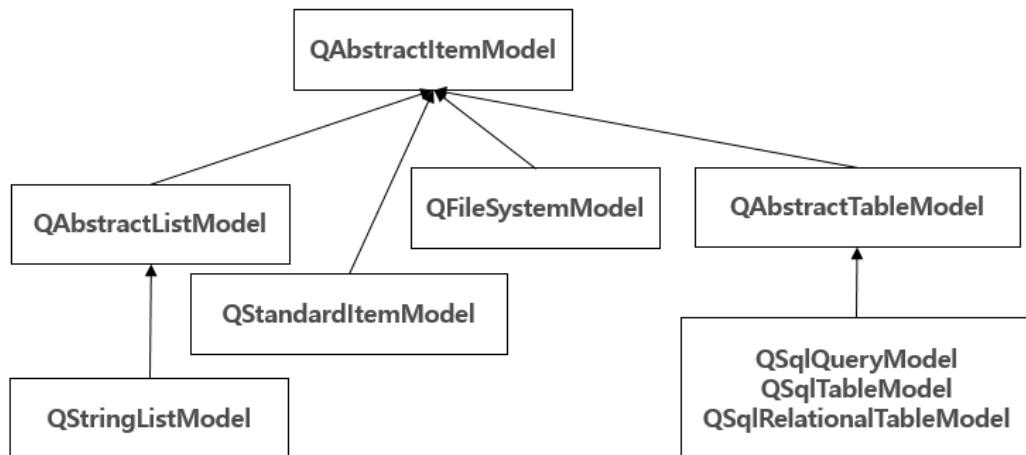
The Model/View provided by Qt can be used to control data, as shown in the following figure. For example, to modify data by double-clicking a specific item in the View widget, Qt might use Delegate to handle these events.



<FIGURE> Diagram Delegate

Jesus loves you.

Model classes provide the ability to manage (insert/modify/delete) data. For example, the QSqlQueryModel class provides a membership function that allows SQL statements to be queried directly (QUERY). Therefore, data imported into a separate database query can be edited using the Model, but the QSqlQueryModel class allows you to insert the data directly. In addition, various model classes are provided as shown in the following figure.



<FIGURE> Inheritance Relationship with Model Classes

Model classes offer a range of model classes, from model classes to complex model classes, to manage data lists that are simply composed of one column.

The **QStringListModel** class provides the ability to manage simple data lists of **QString** data types.

```
QStringListModel *model = new QStringListModel();

QStringList list;
list << "Hello World" << "Qt Programming" << "Model is Good";

model->setStringList(list);
...
```

The **QStandardItemModel** class provides the ability to manage data, such as in table form or tree form.

```
QStandardItemModel model(4, 4);

for (int row = 0; row < 4; ++row)
```

Jesus loves you.

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

The QFileSystemModel class provides the ability to manage files and directories that occur from the file system.

```
QFileSystemModel *model = new QFileSystemModel;  
model->setRootPath(QDir::currentPath());  
  
QTreeView *tree = new QTreeView(this);  
tree->setModel(model);  
...
```

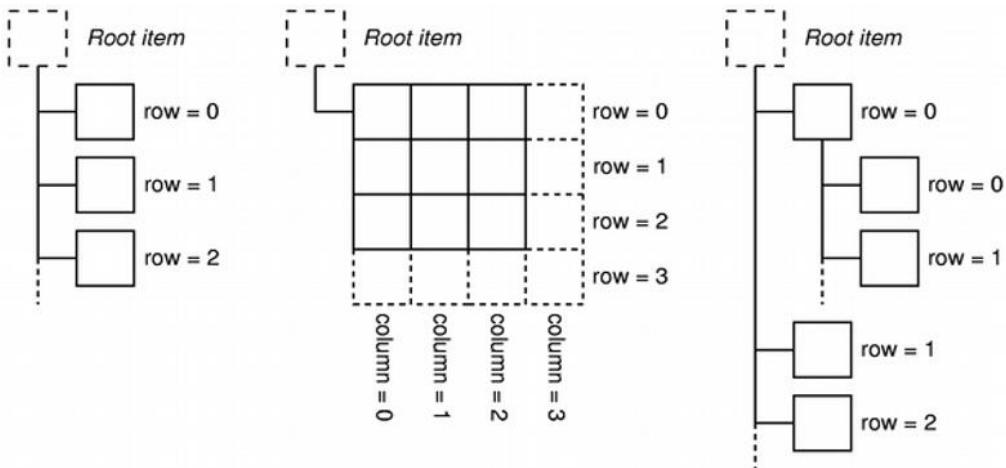
The QSqlQueryModel class provides access to data from database tables using SQL query (QUERY) statements, while the QSqlTableModel class provides the ability to import data into the model by passing a specific table name as a factor when importing data from database roll.

For example, setTable() Name of the table as the first factor in the member functions, database tables from data storage.

```
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();  
...
```

When displaying large amounts of data using Model / View, three types can be distinguished as shown in the following figure.

Jesus loves you.

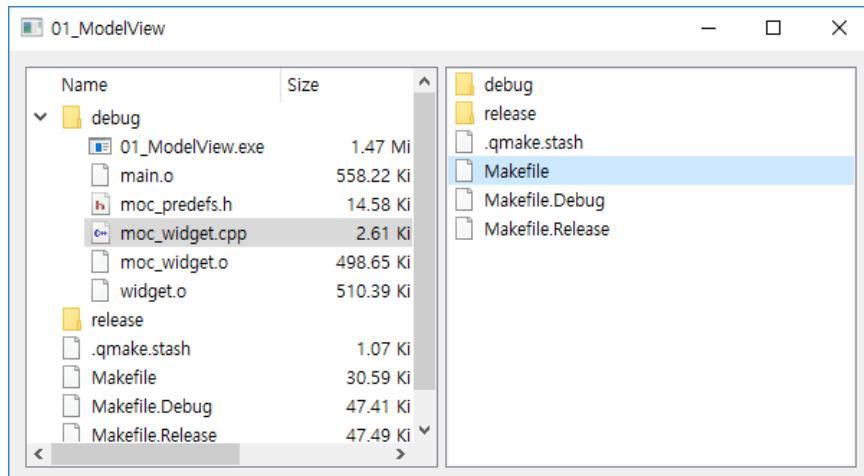


<FIGURE> Types of data display in View

For the most left-most data type, it is appropriate to use the `QListView` class. It is appropriate to use `QTableView` if it should be shown in the table. In addition, it is appropriate to use `QTreeView` if it should be displayed in tree form.

- `QTreeView` and `QListView` example

This example reads the data from the file, stores it in a model, and then links the model to View to display the file system.



<FIGURE> `QTreeView` and `QListView` example screen

Left was shown the data obtained from the file system using a class `qtreeview`. The right side of the `qlistview` A widget showing the current directory by and present in the files and directories.

Jesus loves you.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    resize(600, 300);
    QSplitter *splitter = new QSplitter(this);

    QFileSystemModel *model = new QFileSystemModel;
    model->setRootPath(QDir::currentPath());

    QTreeView *tree = new QTreeView(splitter);
    tree->setModel(model);
    tree->setRootIndex(model->index(QDir::currentPath()));

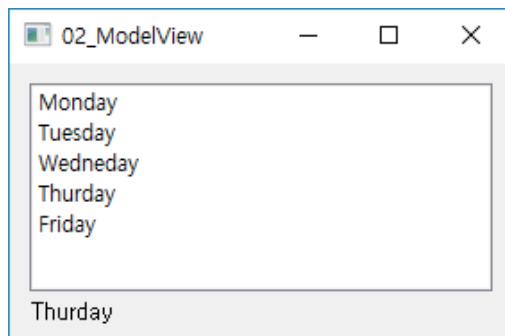
    QListView *list = new QListView(splitter);
    list->setModel(model);
    list->setRootIndex(model->index(QDir::currentPath()));

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

Example - Ch03 > 14_ModelView > 01_ModelView

- Example using QListView class

The figure below shows an example run using the QListView class and the QAbstractItemModel class. QListView is a widget that is suitable for single columns and multiple lines.



<FIGURE> QListView example screen

Jesus loves you.

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

strList << "Monday" << "Tuesday" << "Wednesday" << "Thursday" << "Friday";
QAbstractItemModel *model = new QStringListModel(strList);

QListView *view = new QListView();
view->setModel(model);

QModelIndex index = model->index(3, 0);
QString text = model->data(index, Qt::DisplayRole).toString();

QLabel *lbl = new QLabel("");
lbl->setText(text);

QVBoxLayout *lay = new QVBoxLayout();
lay->addWidget(view);
lay->addWidget(lbl);

setLayout(lay);
...
```

Example - Ch03 > 14_ModelView > 02_ModelView

- `QTableView` example

The `QTableView` class is a suitable widget for displaying data in tabular form, as shown in the figure below.

	Subject	Description	Date
Col 1	Monitor	LCD	2030-10-04 오전 12:00
Col 2	CPU	Samsung	2030-12-05 오전 12:00

<FIGURE> `QTableView` example

Jesus loves you.

```
...  
  
QStandardItemModel *model = new QStandardItemModel(0, 3);  
  
model->setHeaderData(0, Qt::Horizontal, QObject::tr("Subject"));  
model->setHeaderData(1, Qt::Horizontal, QObject::tr("Description"));  
model->setHeaderData(2, Qt::Horizontal, QObject::tr("Date"));  
  
model->setVerticalHeaderItem(0, new QStandardItem("Col 1"));  
model->setVerticalHeaderItem(1, new QStandardItem("Col 2"));  
  
model->setData(model->index(0, 0), "Monitor");  
model->setData(model->index(0, 1), "LCD");  
model->setData(model->index(0, 2),  
QDateTime(QDate(2030, 10, 4)));  
  
model->setData(model->index(1, 0), "CPU");  
model->setData(model->index(1, 1), "Samsung");  
model->setData(model->index(1, 2),  
QDateTime(QDate(2030, 12, 5)));  
  
QTableView *table = new QTableView();  
table->setModel(model);  
  
QVBoxLayout *lay = new QVBoxLayout();  
lay->addWidget(table);  
  
setLayout(lay);  
  
...
```

Example - Ch03 > 14_ModelView > 03_TableModel

- **QTableWidget example**

This example will cover the insertion of data using QTableWidget instead of using Model. This example will also cover how to insert the QCheckBox widget into the first column.

Jesus loves you.

1	2
1 <input checked="" type="checkbox"/> 2019.12.11	Notebook
2 <input type="checkbox"/> 2019.12.11	Mobile
3 <input checked="" type="checkbox"/> 2019.12.11	Desktop
4 <input type="checkbox"/> 2019.12.11	Keyboard
5 <input checked="" type="checkbox"/> 2019.12.11	Monitor

<FIGURE> QTableWidget example screen

위의 그림에서 보는 것과 같이 QTableWidget 헤더에 체크 박스가 있다. 이 체크박스를 변경하면 모든 라인의 컬럼에의 값이 헤더에 있는 체크 박스의 값과 동일하게 변경이 가능하다. 그리고 각각의 체크 박스의 값을 사용자가 변경할 수 있다.

Qt는 QTableWidget 의 헤더(Header)에 원하는 모양 또는 특정 위젯이 추가된 헤더를 커스터마이징 하기 위한 방법을 제공 한다.

QHeaderView 클래스를 상속받아 커스터마이징한 헤더(Header)를 QTableWidget 클래스의 setHorizontalHeader() 멤버 함수를 이용해 커스터마이징한 헤더를 추가 할 수 있다. 다음 예제 소스코드는 widget.cpp 소스코드이다.

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

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

    ui->tableWidget->setRowCount(5);
    ui->tableWidget->setColumnCount(2);

    CheckBoxHeader* header = new CheckBoxHeader(Qt::Horizontal,
                                                ui->tableWidget);
    ui->tableWidget->setHorizontalHeader(header);
    connect(header, &CheckBoxHeader::checkBoxClicked,
```

Jesus loves you.

```
this,    &Widget::checkBoxClicked);

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

for(int i = 0; i < 5 ; i++)
{
    ui->tableView->insertRow(i);
    QTableWidgetItem *dateItem = new QTableWidgetItem("2019.12.11");
    dateItem->setCheckState(Qt::Checked);

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

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

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

In the above example, the CheckBoxHeader class is an inherited and implemented class of QHeaderView. You can use the setHorizontalHeader() member function to use this class as a header.

And checkBoxClicked() is a Slot function associated with events that occurred when the header's check box was changed. In this Slot function, the value of the check box in the first column changes equally depending on the value of the check box in the header. The following example source code is the header file of the CheckBoxHeader class.

Jesus loves you.

```
...
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() is a Virtual function. When the mouse in the header area is clicked, this function is called to change the check box status according to the value of the check box. The following example is the checkboxheader.cpp source code.

```
#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();
```

Jesus loves you.

```
if (logicalIndex == 0) {
    QStyleOptionButton option;
    option.rect = QRect(1,3,20,20);
    option.state = QStyle::State_Enabled | QStyle::State_Active;

    if (isChecked_)
        option.state |= QStyle::State_On;
    else
        option.state |= QStyle::State_Off;

    option.state |= QStyle::State_Off;
    style()->drawPrimitive(QStyle::PE_IndicatorCheckBox, &option, painter);
}
}

void CheckBoxHeader::mousePressEvent(QMouseEvent* event)
{
    Q_UNUSED(event)
    setIsChecked(!isChecked());

    emit checkBoxClicked(isChecked());
}

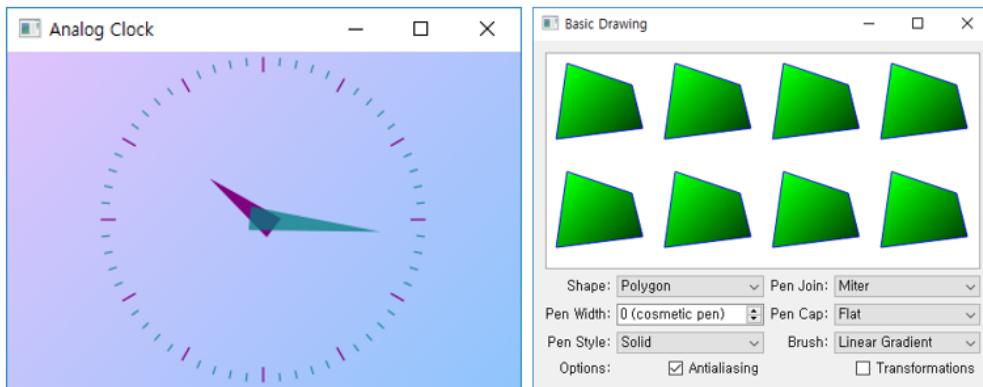
void CheckBoxHeader::redrawCheckBox()
{
    viewport()->update();
}

void CheckBoxHeader::setIsChecked(bool val)
{
    if (isChecked_ != val) {
        isChecked_ = val;
        redrawCheckBox();
    }
}
```

mousePressEvent() is a virtual function called when a mouse click event occurs in the header widget area. Example - ch03 > 14_ModelView > 04_TableWidget

4. GUI 2D Graphics Using QPainter Class

Qt can display text, lines, and shapes using QPainter classes in the GUI widget area. In addition to basic drawing functions, image files can be displayed in the widget area using QImage, QPixmap, and QPicture classes. It also provides functions that can be applied to QPainter areas using effects such as Gradients, Transformation, and Composition, etc.



<FIGURE> QPainter example

To use the QPainter class within the widget area, you can use the patternEvent() Virtual function of the QWidget class, as shown in the following example.

```
...
#include <QPainter>

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

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

...
```

As shown in the example above, the QWidget class can be inherited and the pintEvent() function can be used for implementation. The paintEvent () function is automatically

Jesus loves you.

invoked when a widget is hidden and it appears on the monitor, moves the widget, and zooms the widget.

In addition, if you want to generate an event that requires you to draw the widget area back, you can use the update() function to call the paintEvent() function. The following example is the example source code for drawing shapes from the paintEvent() function.

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

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

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

// Basic Drawing
void Widget::paintEvent(QPaintEvent *event)
{
    Q_UNUSED(event);

    QPainter painter;
    painter.begin(this);

    QPen pen(Qt::blue);

    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);
```

Jesus loves you.

```
painter.setPen(Qt::blue);
painter.drawLine(10, 10, 100, 40);
painter.drawRect(120, 10, 80, 80);

QRectF rect(230.0, 10.0, 80.0, 80.0);
painter.drawRoundedRect(rect, 20, 20);

QPointF p1[3] = {
    QPointF(10.0, 110.0),
    QPointF(110.0, 110.0),
    QPointF(110.0, 190.0)
};
painter.drawPolyline(p1, 3);

QPointF p2[3] = {
    QPointF(120.0, 110.0),
    QPointF(220.0, 110.0),
    QPointF(220.0, 190.0)
};
painter.drawPolygon(p2, 3);

painter.setFont(QFont("Arial", 16));
painter.setPen(Qt::black);
QRect font_rect(10, 150, 520, 180);
painter.drawText(font_rect, Qt::AlignCenter,
                 "Qt developer community (http://www.qt-dev.com)");

painter.end();
}
```

In the example source code above, the begin() member function and end() function of the QPainter class do not immediately draw actual drawing results. Start drawing in a virtual area. In other words, the begin() member function applies the drawing result between the begin() and end() functions to the actual drawing area when the end() member function is called after starting drawing in the virtual memory area.

For example, if you do not use the begin() and end() functions, they apply to the actual drawing area whenever you call up a function that draws each drawing element. If this is a simple drawing result, drawing is invisible.

However, if a computer system is performing complex operations, it can be seen drawing sequentially. Thus, to avoid this problem, begin() and end() draw the drawing into the

Jesus loves you.

virtual memory area, then when the end() function is called, copy the entire drawing to the actual drawn memory area, thereby avoiding the visible process of drawing.



<FIGURE> Example screen

QPainter offers a wide range of capabilities to express the colors, thickness and style of line and shape contours when drawing.

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

QPen pen(Qt::blue);
pen.setWidth(4);
painter.setPen(pen);
QRect rect1(10.0, 20.0, 80.0, 50);
painter.drawEllipse(rect1);

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



<FIGURE> Example screen

Jesus loves you.

The QPainter class can specify the edge style of a line when drawing a line using the setJoinStyle() member function of the QPen class.

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

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

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

QPointF p2[3] = {QPointF(130.0, 80.0),
QPointF(120.0, 40.0),
QPointF(180.0, 60.0) };

pen.setJoinStyle(Qt::MiterJoin);
painter.setPen(pen);
painter.drawPolyline(p2, 3);

QPointF p3[3] = {QPointF(230.0, 80.0),
QPointF(220.0, 40.0),
QPointF(280.0, 60.0) };

pen.setJoinStyle(Qt::RoundJoin);
painter.setPen(pen);
painter.drawPolyline(p3, 3);
...
```



<FIGURE> Example screen

Jesus loves you.

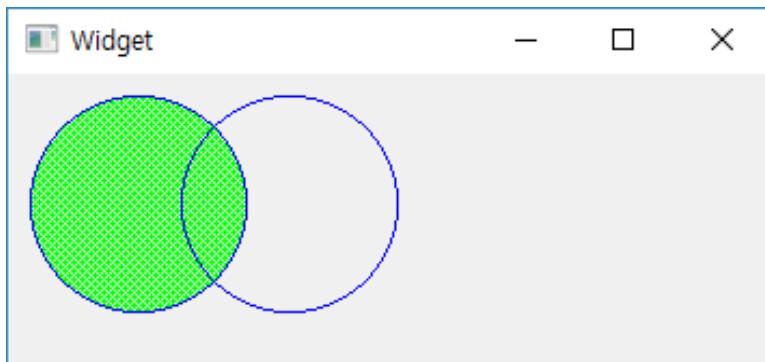
The setBrush() member function provided by the QPainter class allows you to fill the inside of the shape with a specific color.

```
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);

...
```



<FIGURE> Example screen

In addition to filling QBrush with a specific color, the image can be called to fill the interior.

```
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);

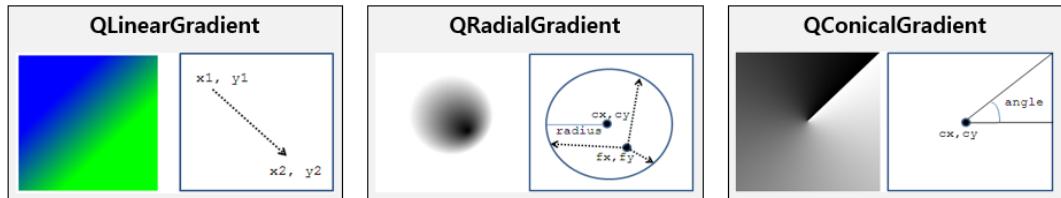
...
```

Jesus loves you.



<FIGURE> Example screen

To use the Gradients effect, you can use the Gradients effect as shown in the following figure.



<FIGURE> Kind of Gradient

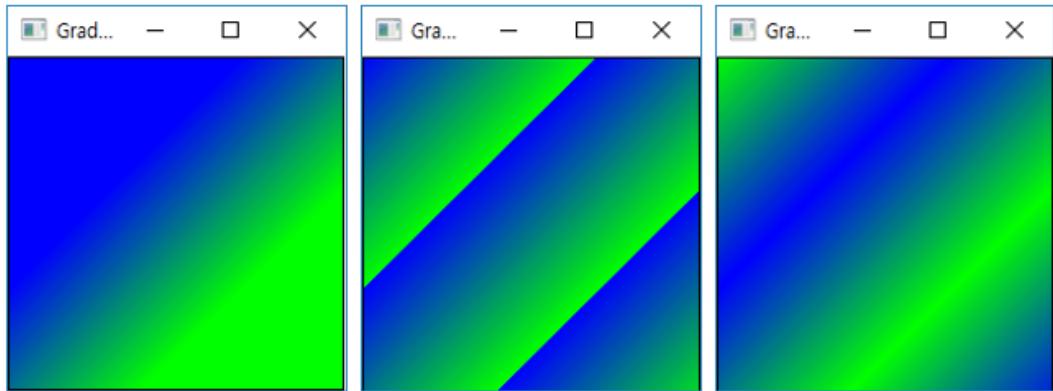
The following example source code is an example using the Gradients effect using the `QLinearGradient` class.

```
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);
...
```

Jesus loves you.



QGradient::PadSpread QGradient::RepeatSpread QGradient::ReflectSpread

<FIGURE> Example screen

Qt can use Scaling, Rotation, and Perspective techniques using the QTransform class.



<FIGURE> QTransform example

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

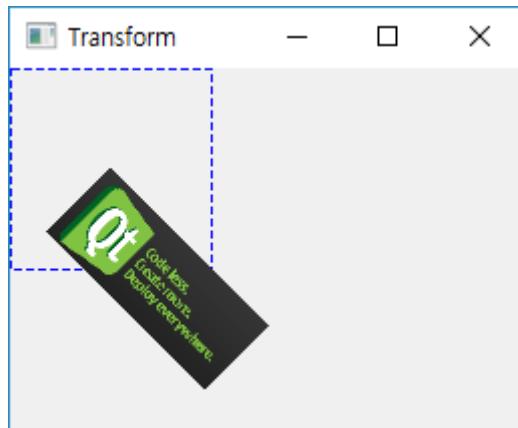
QPainter painter(this);
painter.setPen(QPen(Qt::blue, 1, Qt::DashLine));
painter.drawRect(0, 0, 100, 100);

QTransform transform;
transform.translate(50, 50);
transform.rotate(45);
transform.scale(0.5, 0.5);

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

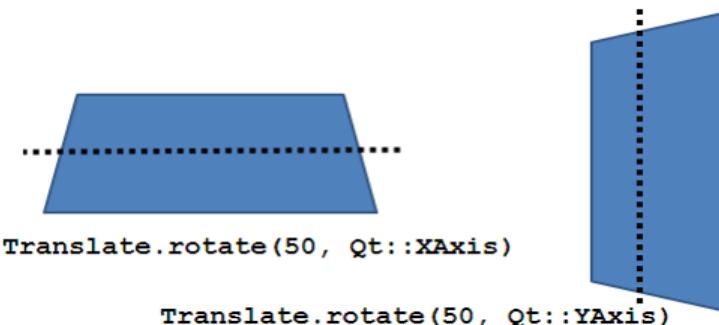
...
```

Jesus loves you.



<FIGURE> Example screen

The following example is the application of the Perspective technique. Provides the ability to move to the X-axis, Y-axis, or Z-axis as a means of applying the Perspective technique.



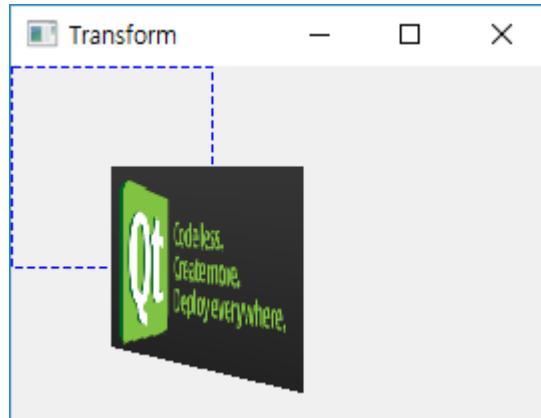
<FIGURE> Perspective Techniques

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

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

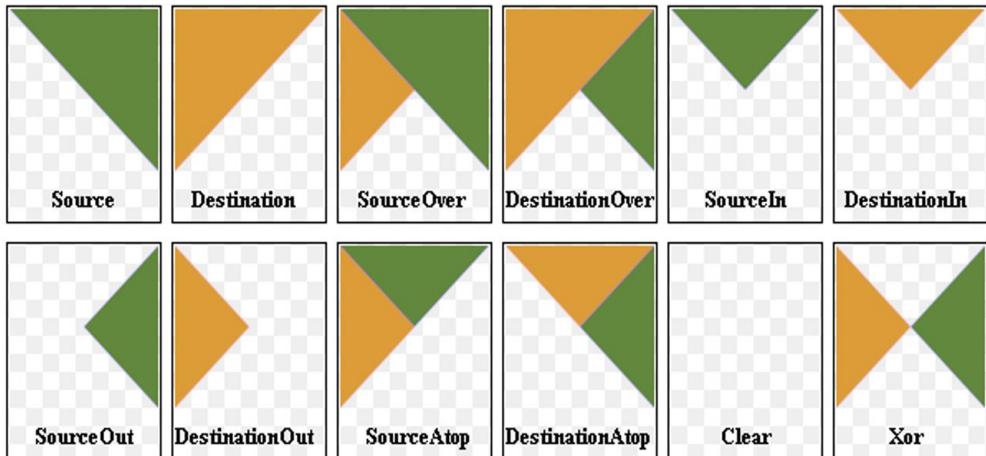
painter.setTransform(transform);
painter.drawImage(0, 0, image);
...
```

Jesus loves you.



<FIGURE> Example screen

QImage allows the overlaid areas to be applied with the Composition technique, as shown in the following figure.



<FIGURE> Kind of Composition

```
...
painter.drawImage(0, 0, destinationImage);
painter.setCompositionMode(QPainter::CompositionMode_DestinationOver);
painter.drawImage(0, 0, sourceImage);
...
```

- Implementing custom buttons

In this example, let's use the QPainter and QWidget classes to implement buttons such as the QPushButton widget. You can implement custom widgets primarily by using QPainter classes and classes for handling user-input events.

Jesus loves you.

So in this example, let's try implementing a button using the QPainter class and mouse events that handle user input. In order to implement a button, the following image is required. Attached to image example source code, please refer to it.



<FIGURE> Image resource

Create a header file with the ImageButton class name to implement the custom buttons as shown in the following example source code:

```
#ifndef IMAGEBUTTON_H
#define IMAGEBUTTON_H
#include <QWidget>
#include <QPainter>

class ImageButton : public QWidget
{
    Q_OBJECT
public:
    explicit ImageButton(QWidget *parent = 0);
    void setDisabled(bool val);

private:
    QString imgFileName;
    qint32 behaviour;
    bool disabled;

signals:
    void clicked();

protected:
    void paintEvent(QPaintEvent *event);
    virtual void enterEvent(QEvent* event);
    virtual void leaveEvent(QEvent* event);
    virtual void mousePressEvent(QMouseEvent* event);
    virtual void mouseReleaseEvent(QMouseEvent* event);
    virtual void mouseDoubleClickEvent(QMouseEvent *event);
};

#endif // IMAGEBUTTON_H
```

Jesus loves you.

In the example above, the setDisabled() member function provides a function for disabling buttons. The disable variable stores the value set in setDisabled. The paintEvent() function changes the image of the mouse according to the event on the mouse.

For example, if the mouse is located in the Button Widget area, change the image of the button to a second image, as shown in the picture above. Then click() signals occur when you click the mouse within the widget area with the . The following example is the implementation part source code for the ImageButton class.

```
#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);
```

Jesus loves you.

```
if(disabled == true) {
    imgFileName = QString(":/resources/disable.png");
}
else
{
    if(this->behaviour == BEHAVIOUR_NOMAL)
        imgFileName = QString(":/resources/normal.png");
    else if(this->behaviour == BEHAVIOUR_ENTER)
        imgFileName = QString(":/resources/enter.png");
    else if(this->behaviour == BEHAVIOUR_LEAVE)
        imgFileName = QString(":/resources/normal.png");
    else if(this->behaviour == BEHAVIOUR_PRESS)
        imgFileName = QString(":/resources/press.png");
}

QImage image(imgFileName);
painter.drawImage(0, 0, image);
painter.end();
}

void ImageButton::enterEvent(QEvent *event)
{
    Q_UNUSED(event);
    this->behaviour = BEHAVIOUR_ENTER;
    update();
}

void ImageButton::leaveEvent(QEvent *event)
{
    Q_UNUSED(event);
    this->behaviour = BEHAVIOUR_NOMAL;
    update();
}

void ImageButton::mousePressEvent(QMouseEvent *event)
{
    Q_UNUSED(event);
    this->behaviour = BEHAVIOUR_PRESS;
    update();

    emit clicked();
}
```

Jesus loves you.

```
void ImageButton::mouseReleaseEvent(QMouseEvent *event)
{
    Q_UNUSED(event);

    this->behaviour = BEHAVIOUR_ENTER;
    update();
}

void ImageButton::mouseDoubleClickEvent(QMouseEvent *event)
{
    Q_UNUSED(event)
}
```

enterEvent() is called when the mouse is located in the widget area. leaveEvent() occurs when the mouse moves out of the widget area. mousePressEvent() occurs when a mouse is clicked inside the widget area and mouseReleaseEvent() occurs when the mouse button is clicked and released. And mouseDoubleClick() happens when you double-click the mouse button.

In view of the mouse event Virtual function provided by Qt, the update() function was used. This function calls the paintEvent() function. If you have implemented the ImageButton class as the source file and header file above, let's declare and try out the objects in the ImageButton class that you have implemented. Let's try creating a header file for the Widget class, as shown in the following example source

```
#ifndef WIDGET_H
#define WIDGET_H

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

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
```

Jesus loves you.

```
private:  
    Ui::Widget *ui;  
  
public slots:  
    void clicked();  
  
};  
  
#endif // WIDGET_H
```

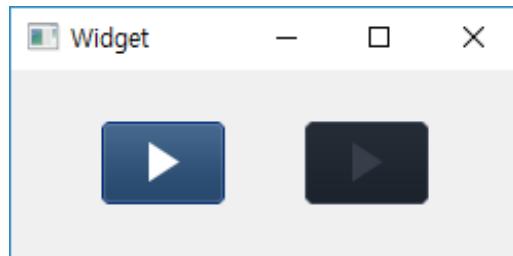
As shown in the example above, clicked() Slot function is a function that is called when a signal event of the ImageButton class occurs. Next, try filling out the source code file for the Widget class as follows:

```
#include "widget.h"  
#include "ui_widget.h"  
#include <QHBoxLayout>  
#include <QDebug>  
  
Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)  
{  
    ui->setupUi(this);  
  
    ImageButton *imgBtn1 = new ImageButton(this);  
    ImageButton *imgBtn2 = new ImageButton(this);  
  
    QHBoxLayout *hLay = new QHBoxLayout(this);  
    hLay->addWidget(imgBtn1);  
    hLay->addWidget(imgBtn2);  
  
    setLayout(hLay);  
    connect(imgBtn1, &ImageButton::clicked, this, &Widget::clicked);  
    imgBtn2->setDisabled(true);  
}  
  
Widget::~Widget()  
{  
    delete ui;  
}  
  
void Widget::clicked()
```

Jesus loves you.

```
{  
    qDebug() << Q_FUNC_INFO;  
}
```

Use the connect() function to connect to the Slot function when an event occurs for the imgBtn1 object. And a second imgbtn disable imagebutton class implements to make with setDisabled() function. See figure below, such as an example to build and run, Let's see.



<FIGURE> Example screen

- Implementing Example of Maintaining Image Scale Ratio

This example shows rendering (displaying) image picture files in PNG format in the widget area. In this example, the image is rendered and retains the image source as the size of the image changes as the widget changes.

For example, the ratio of the size of the image is enlarged or reduced in proportion to the size of the area in the widget. And depending on the ratio of image size, areas where no image is displayed are painted black and images are placed centrally, as shown in the figure below.



<FIGURE> Example screen

Jesus loves you.

As shown in the figure above, this is an example of images being calculated and rendered relative to the size of the widget as the widget grows in size. The following example source code is a paintEvent() function. As previously described, changing window size automatically calls without invoking the pintEvent() function because the pintEvent() function is called.

```
...
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;

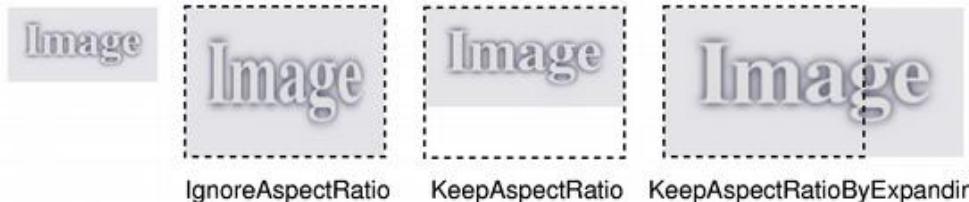
    if(w > imgPixmap.width())
        xPos = (w - imgWidth) / 2;
    else if( h > imgPixmap.height())
        yPos = (h - imgHeight) / 2;

    painter.drawPixmap(xPos, yPos, imgPixmap);
    painter.end();
}
```

QPixmap provides the ability to decode image files. The scaled() function of the QPixmap class specifies the first factor (arrow) and second factor (longitudinal) size. And the third

Jesus loves you.

factor is how to determine how to use images in rendering. The method available for the third factor can be selected as shown in the following table.



<FIGURE> Ratio image

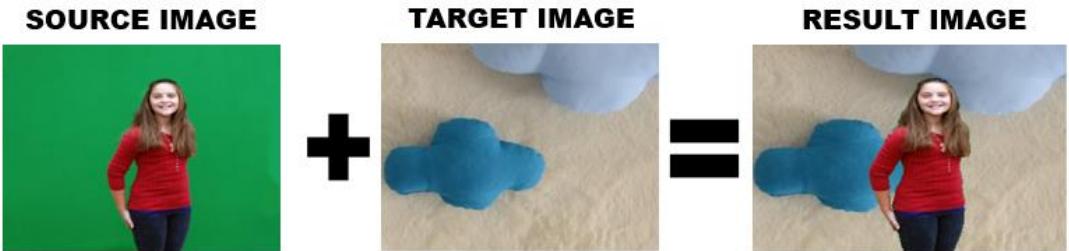
Constant	Value	Explanation
Qt::IgnoreAspectRatio	0	Full horizontal and vertical sizes, regardless of image size ratio
Qt::KeepAspectRatio	1	Maintain image rates regardless of aspect and vertical size
Qt::KeepAspectRatioByExpanding	2	Displayed according to image source size regardless of aspect size and image size ratio

The drawPixmap() member function of the QPainter class provides the ability to display rendered images as QPixmap classes in the QPainter area. The first and second factors are the X and Y starting coordinates. The last third factor can specify the objects in the QPixmap class. Example - Ch04 > 07_ScaledImageRender

- Image synthesis Using Chromakey techniques

In this example, let's implement a method for processing using Chromakey imaging techniques. Chromakey, the word here, refers to a method of filtering the color of a background to replace the background with another image. For example, you may have seen many videos that replace the background of weather news broadcasts with other images or images. In this example, let's change the background of the picture image format to a different image.

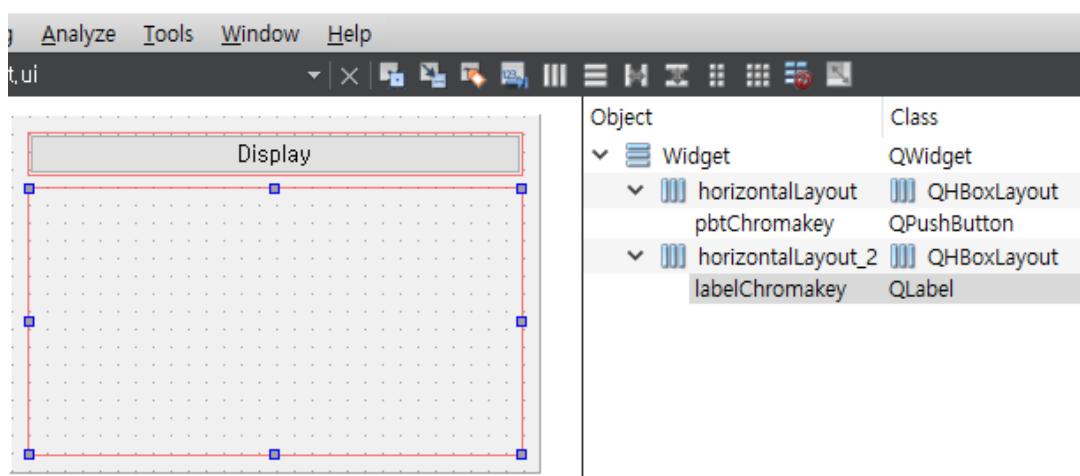
Jesus loves you.



<FIGURE> Example screen

Background image on the background is in the image above, green. Replaced by images to the background color in the picture image source target. To do so, soruce way to remove the man part of the image and background is needed. Sorce the background of images are green.

Therefore, in order to separate people from backgrounds, one can find a green Threshold value, the background color of the SOURCE image, to separate people from the green background. And if removed to change the image the target result be like the image.



<FIGURE> Widgets

Place QPushButton and QLabel as shown in the figure above. You then add the ImageProcessing class to the project, as shown in the following example, and then create the header file as follows:

```
#ifndef IMAGEPROCESSING_H
#define IMAGEPROCESSING_H

#include <QImage>
#include <QColor>
```

Jesus loves you.

```
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
```

The constructor first and second factors in the ImageProcessing class pass on the size as the factor to which you want to perform the kromaki processing. And the third factor hands over the image size. When handing over the image size size to the third factor, the horizontal size and vertical size should be multiplied by 4.

Because we use RGB32 here. In other words, RGBA uses each of the four values and one value is 1 byte size, so the overall size can be calculated as follows.

Width size X Height size X RGBA(4) = Image size

The unit of image size is Pixel. One of the things to be careful when using the kromaki technique should be the size of the SOURCE image, the size of the TARGET image, and the size of the RESULT image.

The chromakeyProcess() member function that will be implemented in the ImageProcessing class crisscrosses the SOURCE image and TARGET image to pass the result over to the third factor. The following example source code is the implementation source code of the ImageProcessing class header.

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

ImageProcessing::ImageProcessing(int width, int height, int dataSize)
{
    this->imageWidth = width;
    this->imageHeight = height;
```

Jesus loves you.

```
this->imageDataSize = dataSize;  
}  
  
void ImageProcessing::chromakeyProcess(QImage& sourceImage,  
                                      QImage& targetImage,  
                                      QImage& resultImage)  
{  
    uchar *pSourceData = sourceImage.bits();  
    uchar *pTargetData = targetImage.bits();  
    uchar *pResultData = resultImage.bits();  
  
    QColor maskColor = QColor::fromRgb(sourceImage.pixel(1,1));  
  
    int kred      = maskColor.red();  
    int kgreen    = maskColor.green();  
    int kblue     = maskColor.blue();  
  
    int sPixRed, sPixGreen, sPixBlue;  
  
    for (int inc = 0; inc < this->imageDataSize ; inc += 4)  
    {  
        sPixRed   = pSourceData[inc+2];  
        sPixGreen = pSourceData[inc+1];  
        sPixBlue  = pSourceData[inc];  
  
        if((abs(kred - sPixRed) + abs(kgreen - sPixGreen) +  
            abs(kblue - sPixBlue)) / 5 < 22 )  
        {  
            pResultData[inc+2] = pTargetData[inc+2];  
            pResultData[inc+1] = pTargetData[inc+1];  
            pResultData[inc]   = pTargetData[inc+0];  
        }  
        else  
        {  
            pResultData[inc+2] = pSourceData[inc+2];  
            pResultData[inc+1] = pSourceData[inc+1];  
            pResultData[inc]   = pSourceData[inc+0];  
        }  
    }  
}
```

QImage was used as a factor in the chromakeyProcess() member function. Despite having various classes, such as QPixmap, the reason for using the QImage class is that the RGBA

Jesus loves you.

values for each pixel in the image are accessible immediately.

In the RGBA of each pixel, the value of four values is Red, G is Green, B is Blue, and A is Alpha. In addition to RGBA, the QImage class offers a variety of formats. In addition, the order of RGBA according to each format is correct, since the format of the RGBA used in the QImage class can change in order of BGRA, RGBA, ABGR, etc.

In addition, QImage provides a format that does not use the Alpha value and can provide only one value of the RGB value. The following are the header file sources for the Widget class:

```
#ifndef WIDGET_H
#define WIDGET_H

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

namespace Ui {
class Widget;
}

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

private:
    ImageProcessing *imgProcess;

    QImage sourceQImage;
    QImage targetQImage;
    QImage resultQImage;

    int sourceQImageWidth;
    int sourceQImageHeight;
    int sourceQImageDataSize;

private slots:
    void slotChromakey();
}
```

Jesus loves you.

```
private:  
    Ui::Widget *ui;  
};  
  
#endif // WIDGET_H
```

SORUCE image for sourceQImage object in QImage class, targetQImage is a TARGET image, and resultQImage is an object for storing the results from the ImageProcessing class of the image to complete the Chromaki processing. The following example source code is the implementation source code of the Widget class.

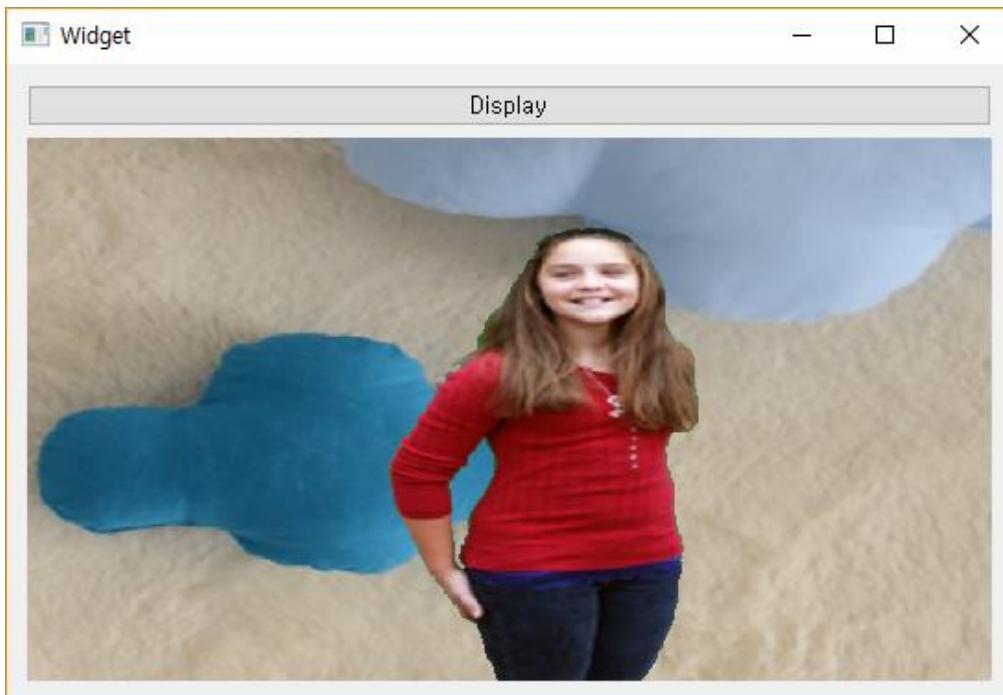
```
...  
#define IMGSOURCE(":/images/jana_480p.png"  
#define IMGTARGET(":/images/target_480p.png"  
  
Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)  
{  
    ui->setupUi(this);  
    connect(ui->pbtChromakey, SIGNAL(clicked()), this, SLOT(slotChromakey()));  
  
    sourceQImage = QImage(IMGSOURCE);  
    targetQImage = QImage(IMGTARGET);  
    resultQImage = QImage(targetQImage.width(),  
                          targetQImage.height(),  
                          QImage::Format_RGB32);  
  
    sourceQImageWidth = targetQImage.width();  
    sourceQImageHeight = targetQImage.height();  
    sourceQImageContentSize = targetQImage.width() * targetQImage.height() * 4;  
  
    imgProcess = new ImageProcessing(sourceQImageWidth,  
                                    sourceQImageHeight,  
                                    sourceQImageContentSize);  
}  
  
void Widget::slotChromakey()  
{  
    imgProcess->chromakeyProcess(sourceQImage, targetQImage, resultQImage);  
  
    int width = ui->labelChromakey->width();  
    int height = ui->labelChromakey->height();  
    QPixmap drawPixmap = QPixmap::fromImage(resultQImage).scaled(width, height);
```

Jesus loves you.

```
    ui->labelChromakey->setPixmap(drawPixmap);  
}  
  
Widget::~Widget()  
{  
    delete ui;  
}  
...
```

The third factor in the QImage class described earlier specifies the type of RGB format to be used by QImage. We used QImage::Format_RGB32. This format refers to a format that uses RGBA values for each pixel. For the types of formats provided by the QImage class, see Assistant Help for various formats available in QImage.

As shown in the example above, complete the creation of the source code and try to execute it after build. Run the example you created and click the [Display] button to see the results.



<FIGURE> Example screen

Example - Ch04 > 08_Chromakey

5. Qt Graphics View Framework

Suppose you use a 1080p resolution monitor that can display 1920 x 1080 pixels. But let's say that the application that we need to develop is a map with hundreds of objects, such as buildings, on a map like navigation, and the size of the map is 3840 x 2160 pixels, twice the resolution of the monitor.

It is not too difficult to display the size of a map with a size of 3840 x 2160 pixels on a 1080p resolution monitor. You can reduce the current image of the map to 1080p resolution and display it in the same area as QPainter.

But what would you do if the map was mapped to the standard GPS coordinate system (WGS84) used by GPS and each pixel coordinate of the map?

This will have to be very difficult to implement. Also, if each object shown on the map has the same coordinates as the GSP coordinate system, how would you implement zooming and zooming? It can be implemented well enough, but it is not something that can be easily implemented in a matter of days.

Take navigation maps for example, but let's take games as an example. On the map, you should mark the main character in the coordinates used in the game and other objects (e.g., monsters?) in addition to the main characters.

And how would you implement it if you had to be able to zoom in and out? There will be many ways, but it will be difficult to implement them easily. Qt provides Graphics View Framework for use in these situations. The Graphics View Framework is a component that provides three components:

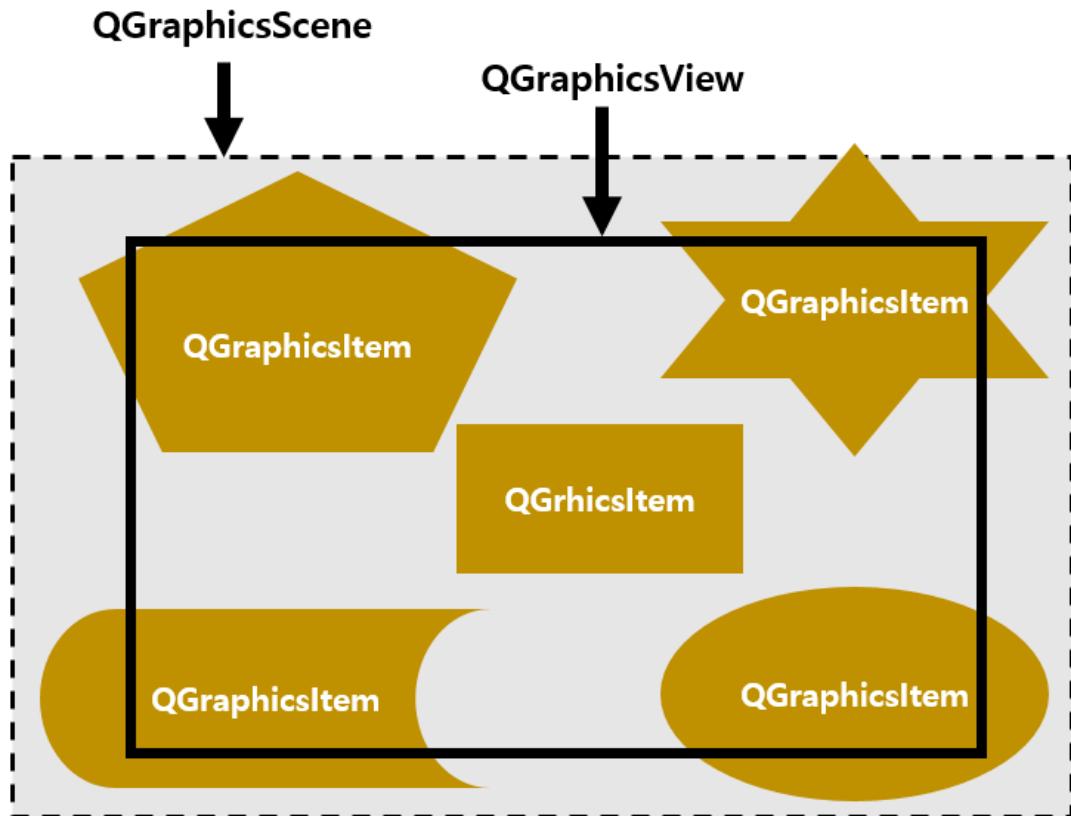
<TABLE> Qt Graphics View Framework 의 3가지 요소

Classes	Explanation
QGraphicsView	A visually visible area. For example, specific areas in the GUI. This class is a real-size area on the GUI.
QGraphicScene	The QGraphicScene class is a logical area. Appears in the QGraphicsView area.
QGraphicsItem	Appears on the QGraphicScene class object. For example, it is

Jesus loves you.

	used as a concept, such as an object displayed on a map.
--	--

The following figure illustrates three elements of the Qt Graphics View Framework.



<FIGURE> Structure of Graphics View Framework

As shown in the figure above, the Graphics View Framework uses QGraphicsView, QGraphicsScene, and QGraphicsItem. For example, the actual area shown in the application is the QGraphicsView area. The GUI widget, QGraphicsView, is also an area of the widget.

QGraphicsScene is a hypothetical area. As shown in the figure above, QGraphicsScene may be larger or smaller than the area of QGraphicsView.

The QGraphicsItem is located on the QGraphicsScene. For example, all QGraphicsItem created as if it were located on an object (such as a building) map on a map should be registered with QGraphicsScene. The QGraphicsItem class provides a paint() virtual function that can display text, image, diagram, etc. in the QGraphicsItem area, such as QPainter. The following is an example source code inherited and implemented by QGraphicsItem.

Jesus loves you.

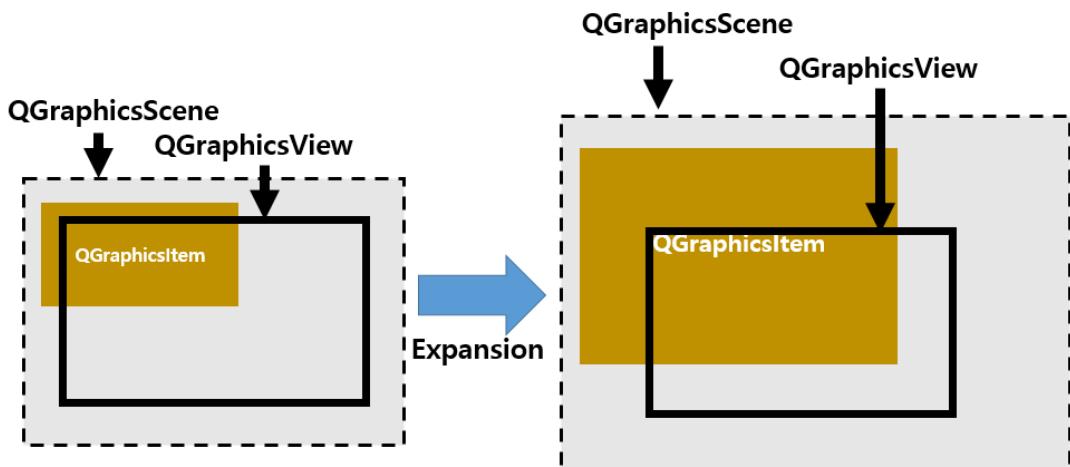
```
...
class SimpleItem : public QGraphicsItem
{
public:
    QRectF boundingRect() const override
    {
        qreal penWidth = 1;
        return QRectF(-10 - penWidth / 2, -10 - penWidth / 2,
                      20 + penWidth, 20 + penWidth);
    }

    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
               QWidget *widget) override
    {
        painter->drawRoundedRect(-10, -10, 20, 20, 5, 5);
    }
};

...
...
```

In the example code above, the `paint()` function provides the same function as the `paintEvent()` function in `QWidget`. In addition to the `QGraphicsItem` class, it also offers various types of item classes such as `QGraphicsRectItem`, `QGraphicsTextItem` and `QGraphicsPixmapItem`.

The Graphics View Framework provides a function to zoom in and out of `QGraphicsScene` to the `QGraphicsView` area.

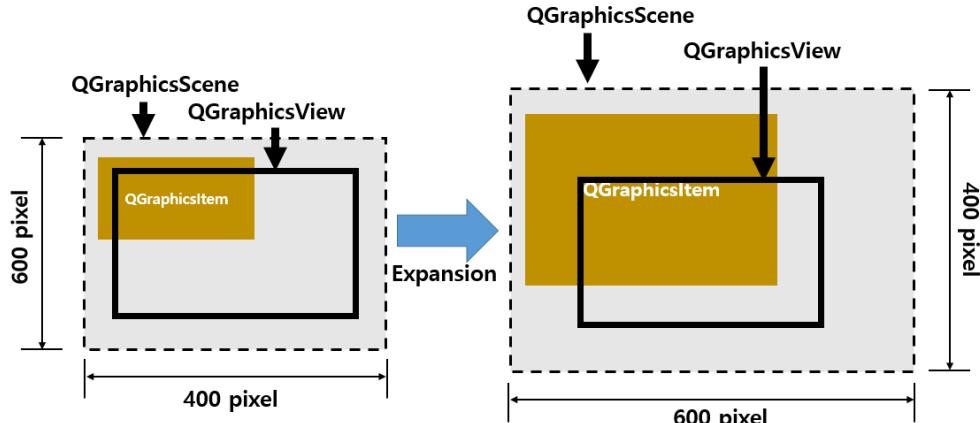


<FIGURE> a schematic picture with magnification

Jesus loves you.

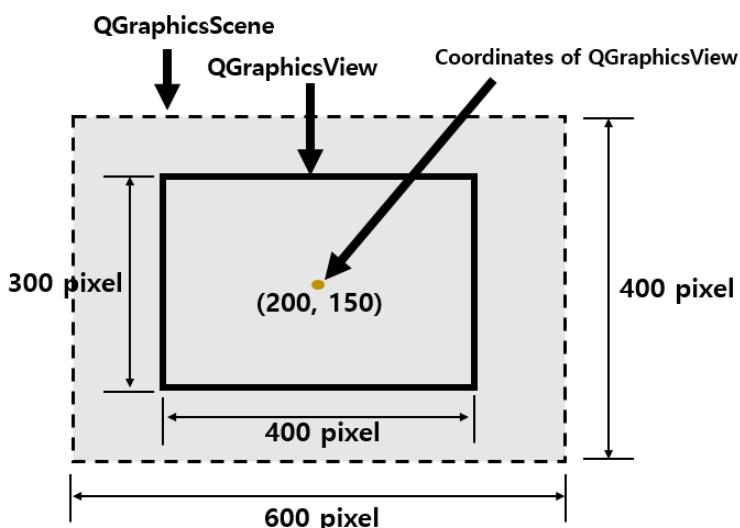
As shown above, the Qt Graphics View Framework logically increases the size of the QGraphicsScene when magnified without changing the size of the QGraphicsView.

However, this does not increase the horizontal and vertical size of the actual QGraphicsScene. The same is true of QGraphicsItem, which is registered with QGraphicsScene. As shown in the following figure, the original size of QGraphicsScene does not change when magnified, nor does it change when reduced.



<FIGURE> Expanding the size of the QGraphicsScene

As shown in the figure above, magnification does not alter the actual size of QGraphicsScene. This is the same for downsizing. And the coordinate systems of QGraphicsView and virtual QGraphicsScene can be different. For example, suppose the X and Y coordinates of QGraphicsView are 200, 150, as shown in the following figure.



<FIGURE> Coordinate System of QGraphicsView with QGraphicsScene

Jesus loves you.

Suppose the QGraphicsView is 400, 300, and the coordinates at the center coordinates of the QGraphicsView are 200, 150, as shown in the figure above. As shown in the figure above, 200 and 150 coordinates are the coordinates of QGraphicsView and not the coordinates of QGraphicsScene. In the above figure, for example, the coordinate points of 200, 150 for QGraphicsView will be approximately 300, 200 for QGraphicsScene.

As such, the Graphics View Framework provides a function to convert coordinates. It provides a function to change the coordinates of various QGraphicsViews to QGraphicsScene coordinates or vice versa, as shown in the figure above.

For example, mapFromScene() member functions and mapToScene() functions are provided. It also provides a function to change the coordinates of items registered in QGraphicsScene to those of QGraphicsView. To map QGraphicsView with QGraphicsScene as shown in the following example source code, you can be used as follows.

```
...
QGraphicsScene scene;
myPopulateScene(&scene);

QGraphicsItem *item;
scene.addItem(&item);

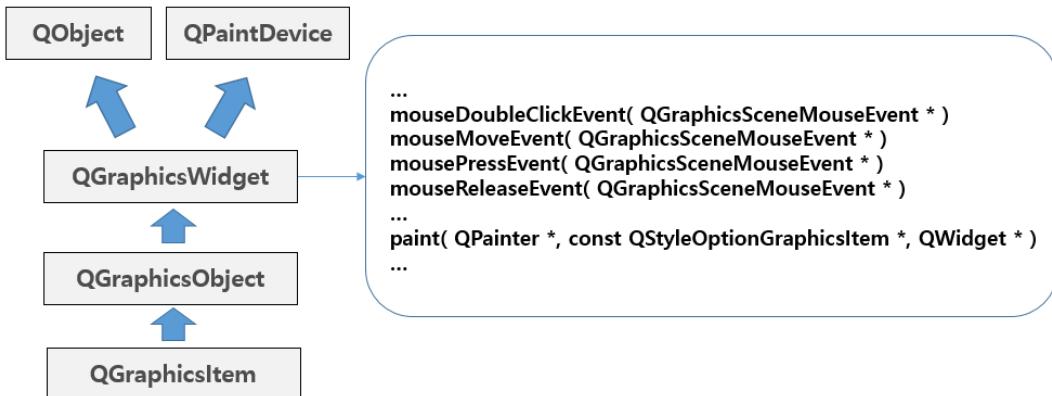
QGraphicsView view(&scene);
view.show();
...
```

You can use the addIItem() member function provided by the QGraphicsScene class to register QGraphicsItem with the QGraphicsScene award.

The QGraphicsItem class can draw shapes or display images using the 2D graphic elements in an area using the print() virtual function, as described earlier.

In addition, since QGraphicsItem was implemented in succession to QGraphicsWidget, it is possible to use a virtual function that can handle events such as user input in the QGraphicsItem area as shown in the following figure.

Jesus loves you.



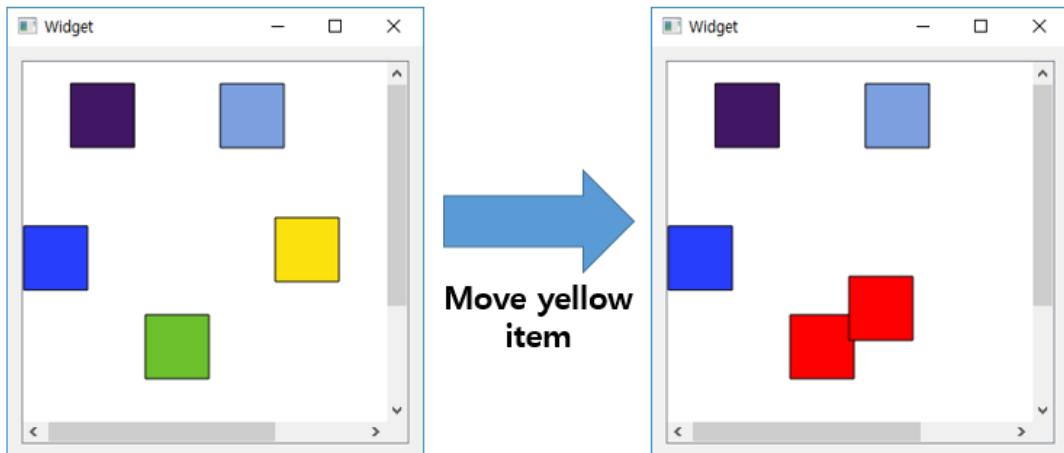
<FIGURE> The inheritance relationship of the `QGraphicsItem` class

Graphics View Framework uses BSP(Binary Space Partitioning) algorithms internally to enable rapid interaction of Zoom, Rotation, and numerous items. Therefore, more than hundreds of thousands of items are also available.

- Example of implementing crash detection in `QGraphicsItem`

The example to be dealt with this time is the implementation of a function that detects a collision of a Shape item created by the `QGraphicsItem` class. You can drag the Shape item registered in `QGraphicsScene` with the mouse.

If a drag one Shape item collides with another Shape item, change the color of the crashed item to red. The following picture shows an example.

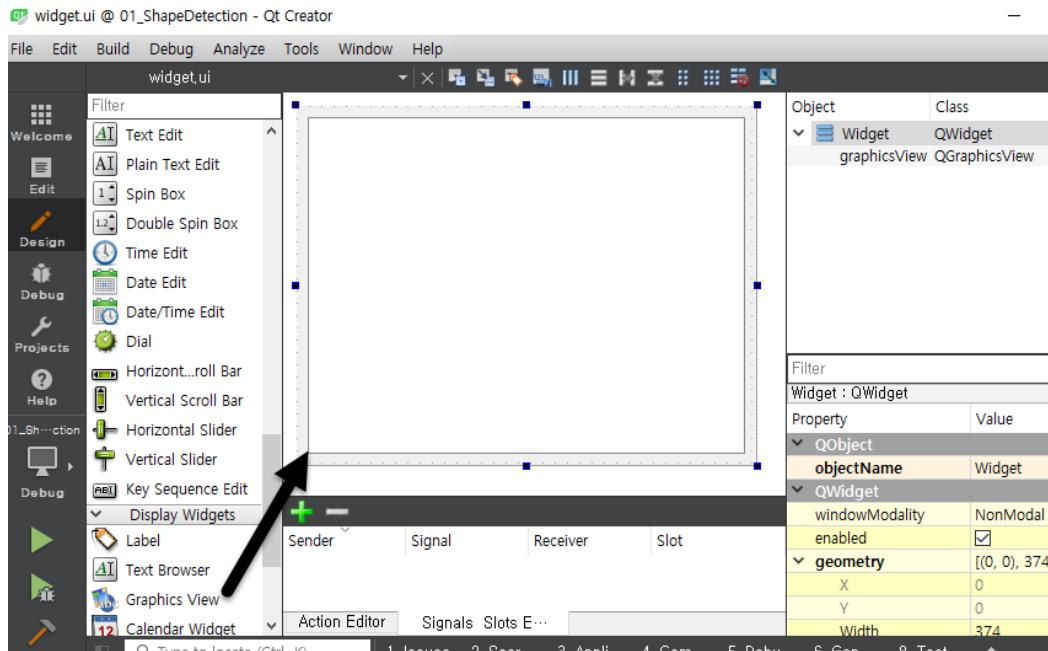


<FIGURE> Example screen

If one of the Shape items is dragged with the mouse and overlapped with another, as

Jesus loves you.

shown in the picture above, a collision will occur. Then, change the color of the two conflicting items to red. Create a new project based on QWidget. Then place the widget in the GUI form as follows.



<FIGURE> GUI From screen

Place the [Graphics View] item in the left widget list on the GUI as shown in the figure above. And create a new class called Shape in the project. Create a header file for the Shape class, as shown in the following example:

```
#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:
```

Jesus loves you.

```
void advance(int step) override;
void mouseMoveEvent(
    QGraphicsSceneMouseEvent *event) override;

private:
    QColor color;
};

#endif // SHAPE_H
```

The Shape class inherits and implements the QGraphicsItem class. The boundingRect() override function is a function used inside QGraphicsScene. In this function, you can pass the starting point x, y, width, and height values of this item.

The shape() redefined function is a function that detects a conflict. In this function, you can specify the area where the collision is detected. If a registered item occurs within QGraphicsScene, it detects the area registered by this function and detects the collision. The paint() function can draw shapes or desired elements in the QGraphicsItem area. Provides the same functionality as the paintEvent() function in the QWidget class.

The evaluation function used a timer in the Widget class. This function is automatically invoked when the timer is called. In addition, the mouseMoveEvent() function provides a function for dragging a specific QGraphicsItem with the mouse within QGraphicsScene. The following example source code is the functional implementation part of the Shape class.

```
#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));
```

Jesus loves you.

```
{}

QRectF Shape::boundingRect() const
{
    return QRectF(0, 0, 50, 50);
}

QPainterPath Shape::shape() const
{
    QPainterPath path;
    path.addRect(0, 0, 50, 50);
    return path;
}

void Shape::paint(QPainter *painter, const QStyleOptionGraphicsItem *,
                  QWidget *)
{
    if(scene()->collidingItems(this).isEmpty()) {
        painter->setBrush(color);
    } else {
        painter->setBrush(QColor(Qt::red));
    }

    painter->drawRect(0, 0, 50, 50);
}

void Shape::advance(int step)
{
    if (!step)
        return;
    update();
}

void Shape::mouseMoveEvent(QGraphicsSceneMouseEvent *event)
{
    QPointF eventpos = event->pos();
    QPointF shapePos = this->pos();

    QPointF wPos(eventpos.x() + shapePos.x() - 25,
                 eventpos.y() + shapePos.y() - 25);
    setPos(wPos);
    update();
}
```

Jesus loves you.

```
QGraphicsItem::mouseMoveEvent(event);  
}
```

In the constructor, the indicator of the setFlags() function can set a value for allowing the item to move. The color variable is used as the color of the Brush in the Shape.

Scene()->collidingItems(this) in the paint() function. The isEmpty() function can tell if an item is currently in a collision. Therefore, the constructor uses the value of the color variable if it is not in a collision state. Otherwise, if a collision is detected, use the red color specified by the Qt::red constant as Brush.

The following example source code is the Widget class for using the Shape class. Create a header file as shown in the following example:

```
#ifndef WIDGET_H  
#define WIDGET_H  
  
#include <QWidget>  
#include <QGraphicsScene>  
#include <QTimer>  
#include "shape.h"  
  
namespace Ui { class Widget; }  
  
class Widget : public QWidget  
{  
    Q_OBJECT  
public:  
  
    explicit Widget(QWidget *parent = nullptr);  
    ~Widget();  
    void timerStart();  
  
private:  
    Ui::Widget *ui;  
    QGraphicsScene *m_scene;  
    Shape        *m_shape[5];  
    QTimer       m_timer;  
};  
  
#endif // WIDGET_H
```

위의 클래스 생성자에서 QGraphicsScene 을 생성한다. 생성한 QGraphicsScene 상에서

Jesus loves you.

Register QGraphicsItem. In addition, QGraphicsScene is registered in QGraphicsView using the setScene () function. The following example source code is main.cpp. Here the objects in the QTimer class are called start() member functions.

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

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

    Widget w;
    w.show();
    w.timerStart();

    return a.exec();
}
```

Let's run the example you've created so far, and then drag the mouse over QGraphicsItem. If a collision occurs, the impacted Shape changes its interior color to red. When the collision is released, it changes to its original color.

Example - Ch05 > 01_ShapeDetection

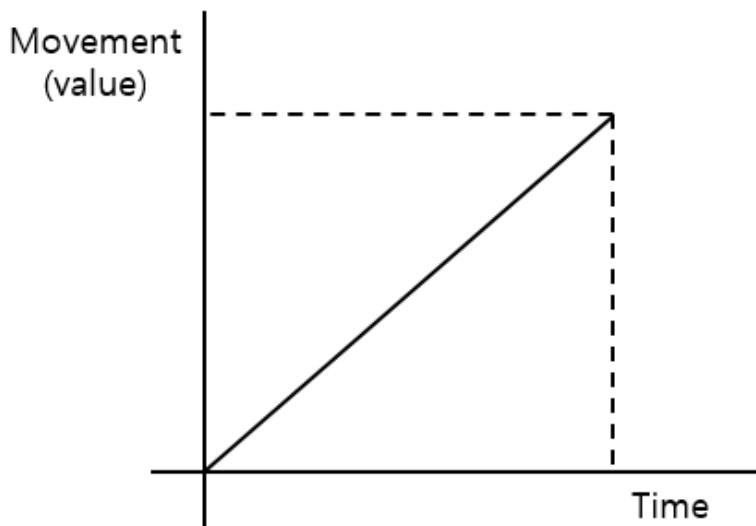
6. Animation Framework 와 State Machine

In order to use animation elements such as smooth screen switching when switching screens on the GUI, Qt provides Animation Framework. Animated elements are available on the window screen, and widgets placed on the GUI can also use each animation element.

Animated elements such as transparency, movement, and zoom can be used as animation elements. For example, if an application requires a window with a screen width and vertical size of 600 x 400 pixels to run, the larger window size from 100 x 100 to 600 x 400 over a specified time period can be used as an animation element.

In addition, animation effects can be used to vary the transparent from 0.0 to 1.0 over a specified time using Opacity at the same time. 0.0 is fully transparent and 1.0 is all transparent.

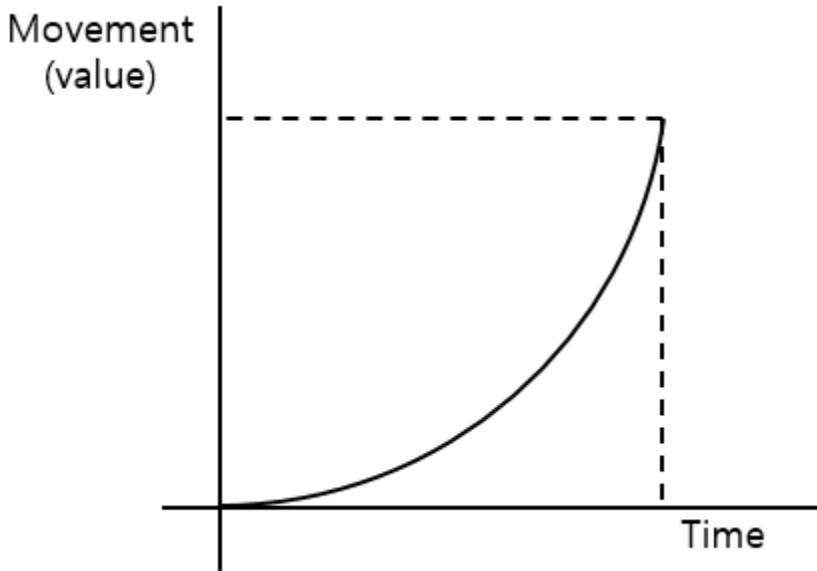
In addition, Easing Curves can be used for time values of animation. For example, suppose you have specified 1.0 seconds for the x, y positions of the GUI buttons to move and travel from positions of 100, 100 to coordinates of 200 and 200. This will travel at a constant speed from the x and y start coordinates to the destination coordinates.



<FIGURE> Movement and Time Graph

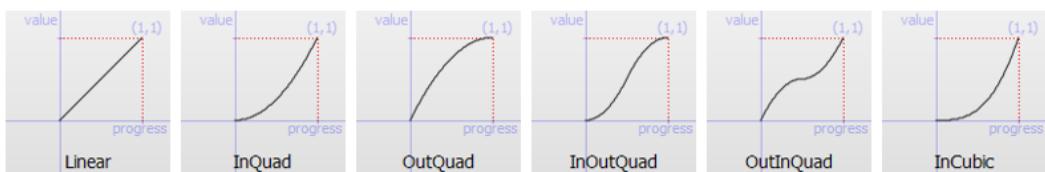
Jesus loves you.

As shown in the figure above, the y-axis of movement (value) is a graph that shows that the x, y-position coordinates of the GUI buttons take a constant 1.0 second to move from 100, 100 to 200 and 200 values. In other words, the time increases steadily (linear) depending on the movement value. However, using Easing Curves at the time allows you to change the value of time in the moving value as shown in the following figure.



<FIGURE> Graphs with Easing Curve

It is possible to apply values for the time taken to the final x, y coordinates as shown in the figure above. In addition, the Easing Curve can apply about 46 different kinds of Easing Curve.



<FIGURE> Various Using Curve

And Qt can use State Machine in Animation element. State Machine is an example of an on/off switch. Not only can you use ON/OFF to change one value, but you can also use a number of options. For example, a state machine can be used to turn on or off the fluorescent light in the living room with a fluorescent light in the front door of the house. In other words, under certain circumstances, you can use a technique called State Machine in which different values are applied at the same time depending on the

Jesus loves you.

situation.

Suppose there are A, B, and C buttons on the GUI. Press A button to move the position of the C button from 100 and 100 positions to 200 and 200, and to change the projection from 1.0 to 0.0 simultaneously, State Machine can be used. Of course, Animation offers parallel execution capabilities, but if you have to change dozens of states, it may be more effective to use State Machine technology. So far, we have briefly summarized what this chapter will cover. In this chapter, we will give examples of Animation, Easing Curve and State Machine.

- Implemented using Animation to move buttons on the GUI

In this example, QPushButton is placed on the GUI. Animation elements are used to move the deployed QPushButton to a specific location. Clicking the button moves the x and y coordinates of the button from the positions of 10, 10, 10, and to the positions of 200 and 150. It takes 3,000 milliseconds to move. The following is the source code for Widget.h.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QPushButton>
#include <QPropertyAnimation>

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

private:
    QPropertyAnimation *animation;

public slots:
    void btnClicked();
};

#endif // WIDGET_H
```

Jesus loves you.

Declares the objects of the QPropertyAnimation class as shown in the example source code above. The btnClicked() Slot function is invoked by clicking the button. Next is the widget.cpp source code.

```
#include "widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent)
{
    this->resize(350, 200);

    QPushButton *btn = new QPushButton("Button", this);
    connect(btn, &QPushButton::pressed, this, &Widget::btnClicked);
    btn->setGeometry(10, 10, 100, 30);

    animation = new QPropertyAnimation(btn, "geometry", this);

    animation->setDuration(3000);
    animation->setStartValue(QRect(10, 10, 100, 30));
    animation->setEndValue(QRect(200, 150, 100, 30));
}

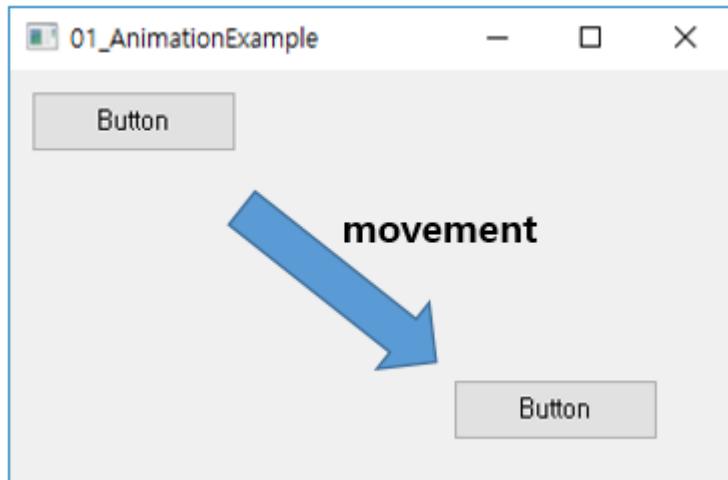
void Widget::btnClicked()
{
    animation->start();
}

Widget::~Widget()
{
```

The btn used for the first factor when executing the constructor function of the QPropertyAnimation class as the new operator specifies the object of QPushButton. And the second "geometry" is a moving factor.

setDuration() can specify the total time required to move. setStartValue() specifies the values of x, y, width, and height of the start coordinates. setEndValue() specifies the position, horizontal, and vertical size of the movement. Click the btnClicked() function to run QPropertyAnimation.

Jesus loves you.



<FIGURE> Example screen

Click the button, as shown in the figure above, to move the x- and y-coordinates to 200, 150 for three seconds. Here's an example of how to use the Easing Curve: Let's add the following source code at the end of the Widget generator, as shown in the following example.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    this->resize(350, 200);
    QPushButton *btn = new QPushButton("Button", this);
    ...
    animation->setEasingCurve(QEasingCurve::OutInQuart);
}
...
```

Add the `setEasingCurve()` function, as shown in the example source code above, and then try to build it and run it. Click the button to see that the Easing Curve is applied.

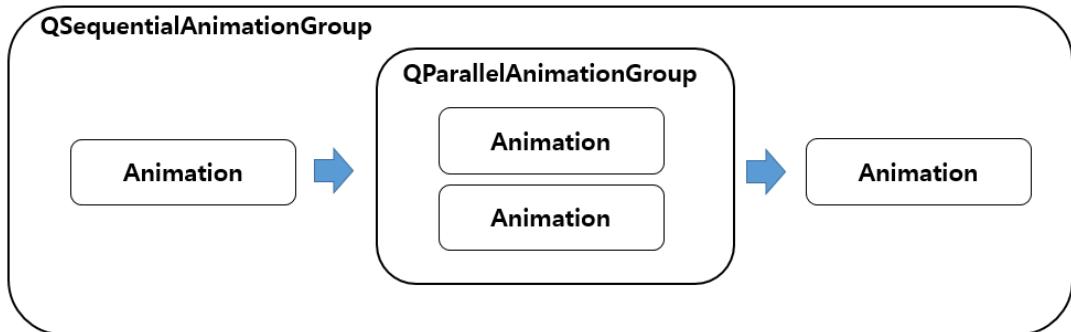
Easing Curve provides a function to change the pattern of motion by interpolating the progress of the path from the start coordinate to the last. In addition to moving coordinates, it can be used for transparency, zoom.

Example - Ch06 > 01_AnimationExample

Jesus loves you.

- Examples with Animation Groups

Let's take a look at the implementation of functions for simultaneous or sequential execution of the Animation elements using grouping. In the Animation Framework, if there are many animation elements that you want to use, you can use Animation in groups as shown in the following figure.



<FIGURE> Animation group

The QSequentialAnimationGroup class and QParallelAnimationGroup class can be used to group animation as shown above. QSequentialAnimationGroup can perform animation sequentially and the QParallelAnimationGroup class can perform animation simultaneously (parallel).

In addition, QParallelAnimationGroup can be used to internalize animation elements within QSequentialAnimationGroup as shown in the figure.

Conversely, QSequentialAnimationGroup can be used within QParallelAnimationGroup. The following example source code is the widget.h source code.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QPushButton>
#include <QPropertyAnimation>
#include <QSequentialAnimationGroup>
#include <QParallelAnimationGroup>

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

Jesus loves you.

```
~Widget();
private slots:
    void btn_clicked();
private:
    QPropertyAnimation *anim1;
    QPropertyAnimation *anim2;
    QSequentialAnimationGroup *sGroup;
};

#endif // WIDGET_H
```

As shown in the example code above, the QSequentialAnimationGroup class provides the ability to perform animation elements. The following example source code is Widget.cpp.

```
#include "widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent)
{
    resize(320, 270);
    QPushButton *btn1 = new QPushButton("First", this);
    btn1->setGeometry(10, 10, 100, 30);

    QPushButton *btn2 = new QPushButton("Second", this);
    btn2->setGeometry(10, 45, 100, 30);

    anim1 = new QPropertyAnimation(btn1, "geometry");
    anim1->setDuration(2000);
    anim1->setStartValue(QRect(10, 10, 100, 30));
    anim1->setEndValue(QRect(200, 150, 100, 30));

    anim2 = new QPropertyAnimation(btn2, "geometry");
    anim2->setDuration(2000);
    anim2->setStartValue(QRect(10, 45, 100, 30));
    anim2->setEndValue(QRect(200, 195, 100, 30));

    sGroup = new QSequentialAnimationGroup;
    sGroup->addAnimation(anim1);
    sGroup->addAnimation(anim2);

    connect(btn1, SIGNAL(clicked()), this, SLOT(btn_clicked()));
    connect(btn2, SIGNAL(clicked()), this, SLOT(btn_clicked()));
}
```

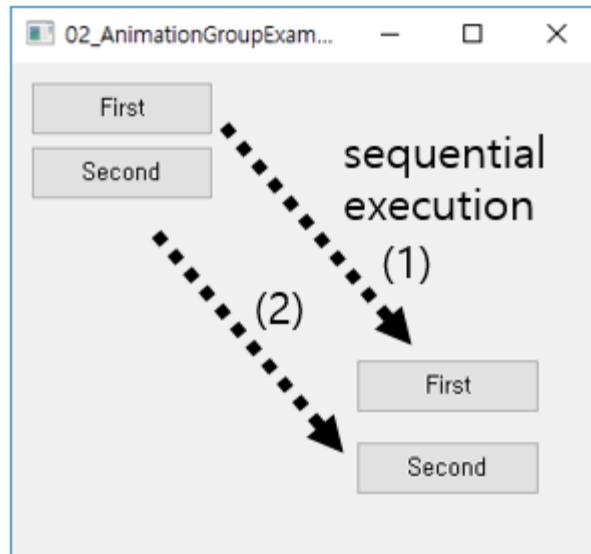
Jesus loves you.

```
void Widget::btn_clicked()
{
    sGroup->start(QPropertyAnimation::DeleteWhenStopped);
}

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

The example source code above places two buttons. When animation starts, move the [First] button first, and then move the [Second] button. The following figure shows an example run screen.

As shown in the figure below, two button animations are executed sequentially. For the following example, modify the source code so far and change it to parallel. On the widget.h source code, annotate it as follows and add the source code to the bottom.



<FIGURE> Example screen

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QPushButton>
#include <QPropertyAnimation>
#include <QSequentialAnimationGroup>
#include <QParallelAnimationGroup>
```

Jesus loves you.

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

private slots:
    void btn_clicked();

private:
    QPropertyAnimation *anim1;
    QPropertyAnimation *anim2;
    //QSequentialAnimationGroup *sGroup;
    QParallelAnimationGroup *pGroup;
};

#endif // WIDGET_H
```

Next, let's modify and add the QSequentialAnimationGroup class in Widget.cpp and the Slot function to allow animation to work in parallel as shown in the example source code below.

```
#include "widget.h"

Widget::Widget(QWidget *parent) : QWidget(parent)
{
    resize(320, 270);
    QPushButton *btn1 = new QPushButton("First", this);
    btn1->setGeometry(10, 10, 100, 30);

    QPushButton *btn2 = new QPushButton("Second", this);
    btn2->setGeometry(10, 45, 100, 30);

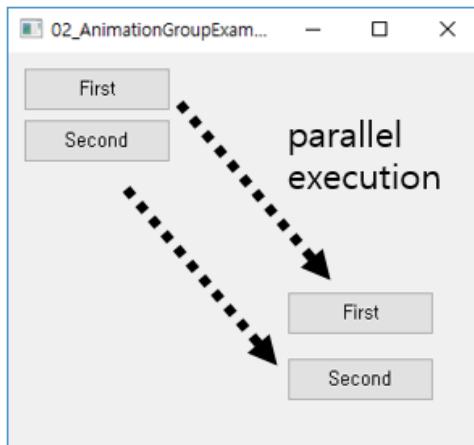
    anim1 = new QPropertyAnimation(btn1, "geometry");
    anim1->setDuration(2000);
    anim1->setStartValue(QRect(10, 10, 100, 30));
    anim1->setEndValue(QRect(200, 150, 100, 30));

    anim2 = new QPropertyAnimation(btn2, "geometry");
    anim2->setDuration(2000);
```

Jesus loves you.

```
anim2->setStartValue(QRect(10, 45, 100, 30));  
anim2->setEndValue(QRect(200, 195, 100, 30));  
  
pGroup = new QParallelAnimationGroup;  
pGroup->addAnimation(anim1);  
pGroup->addAnimation(anim2);  
  
connect(btn1, SIGNAL(clicked()), this, SLOT(btn_clicked()));  
connect(btn2, SIGNAL(clicked()), this, SLOT(btn_clicked()));  
}  
  
void Widget::btn_clicked()  
{  
//    sGroup->start(QPropertyAnimation::DeleteWhenStopped);  
    pGroup->start(QPropertyAnimation::DeleteWhenStopped);  
}  
  
Widget::~Widget()  
{  
}
```

예제를 실행하면 [First] 버튼과 [Second] 버튼의 애니메이션이 동시에 실행되는 것을 확인할 수 있을 것이다.



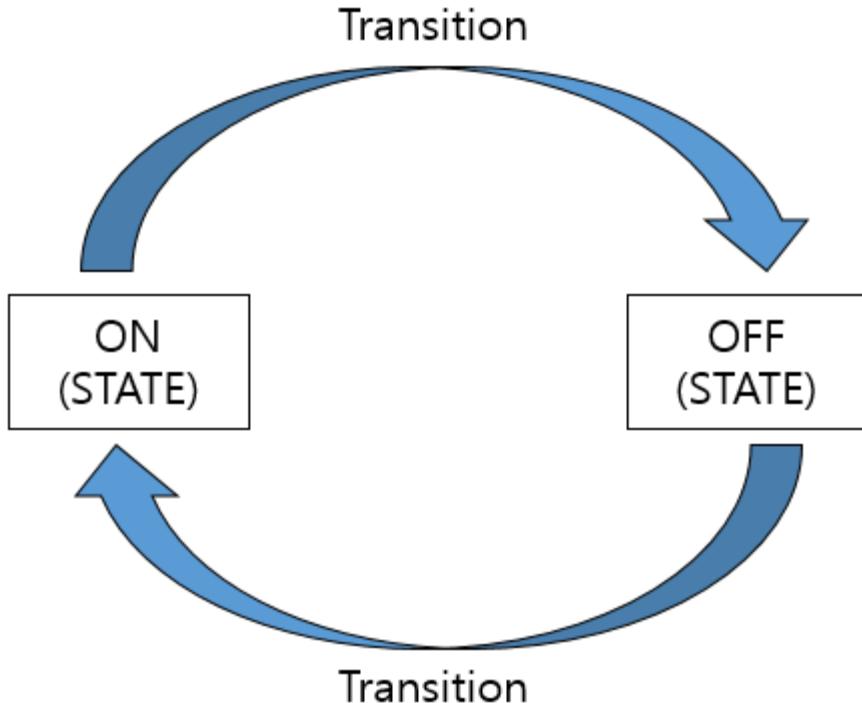
<FIGURE> Example screen

- Animation and State Machine example

State Machine is compared to ON/OFF. It is possible to distinguish between ON and OFF actions or actions that change state from ON to OFF as transition. In other words, State

Jesus loves you.

indicates the state of recognition on or off. Transition can be seen as an action that changes from ON to OFF.



<FIGURE> State 외 Transition

State Machine techniques can be used in Animation for the purpose of performing animation elements at the same time if many Animation elements are used. For example, suppose you perform dozens of animation elements, you can group them, but the source code becomes more complex. However, it is easy to implement State Machine by applying it.

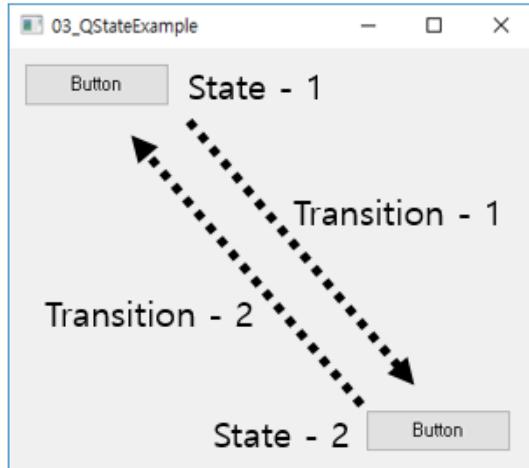
In Qt, the State Machine technique is divided into State and Transition, as shown in the figure above. Provide QState classes to use State. It also provides a QTransition class to apply Transition.

The picture below shows the running screen of the State Machine example. State – 1 is the state – 1 with the x, y coordinates of the buttons 10, 10.

And the state with x and y coordinates of 250 and 250 is State – 2.

Moving from State – 1 to State – 2 is designated Transition – 1 and moving from State – 2 to State – 1 is designated Transition – 2.

Jesus loves you.



<FIGURE> Example screen

Therefore, when you click the button in State – 1, Transition – 2 is performed and State – 2 is changed. Conversely, when you click the button in State – 2, Transition – 2 is performed and the status is State – 1. The following is part of the widget.cpp source code.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    resize(360, 290);
    QPushButton *button = new QPushButton("Button", this);
    button->setGeometry(10, 10, 100, 30);

    QStateMachine *machine = new QStateMachine;
    QState *state1 = new QState(machine); // state-1
    state1->assignProperty(button, "geometry", QRect(10, 10, 100, 30));
    machine->setInitialState(state1);

    QState *state2 = new QState(machine); // state-2
    state2->assignProperty(button, "geometry", QRect(250, 250, 100, 30));

    // transition-1
    QSignalTransition *transition1;
    Transition1 = state1->addTransition(button, SIGNAL(clicked()), state2);
    transition1->addAnimation(new QPropertyAnimation(button, "geometry"));

    // transition-2
    QSignalTransition *transition2;
```

Jesus loves you.

```
transition2 = state2->addTransition(button, SIGNAL(clicked()), state1);
transition2->addAnimation(new QPropertyAnimation(button, "geometry"));
machine->start();
}
...
}
```

Create state1 and state2 objects with QState classes. The state of the object that you created was declared as location coordinates. In addition, QSignalTransition was used to change state1 and state2 status from one At the end, the start() function was performed.

The start() function is not executed immediately, but the transition is performed according to the State when the button is clicked.

Example - Ch06 > 03_QStateExample

7. 3D Graphics Using Qt OpenGL Module

Qt offers a Qt OpenGL module with OpenGL wrapped to make 3D graphics programming more user-friendly using OpenGL. Qt OpenGL module is identical to OpenGL. To render 3D using OpenGL as we use QWidget for GUI programming, the Qt OpenGL module provides QOpenGLWidget and QGLWidget classes.

OpenGL is a 3D graphics standard API developed by Silicon Graphics and provides approximately 250 functions. It provides the ability to express complex 3D graphics in geometry and is classified as OpenGL and OpenGL ES. OpenGL is an API used in environments such as desktop PCs.

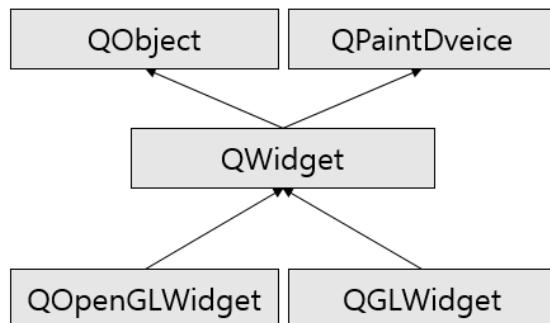
OpenGL ES is used for 3D graphics in resource-constrained environments, such as embedded systems, and Qt also supports both OpenGL and OpenGL ES. In order to use OpenGL in Qt, the project file shall contain the following statements

```
QT + opengl
```

In addition, the following exclude shall be used in the header.

```
#include <QtOpenGL>
```

QOpenGLWidget and QGLWidget provide the same functions for rendering 3D graphic elements. QOpenGLWidget is recommended for Qt 5.4 versions, although either class may be used. QGLWidget can be used to render OpenGL like QWidget.



<FIGURE> The inheritance relationship between QOpenGLWidget and QGLWidget

QOpenGLWidget provides three virtual functions that are easy to implement, such as those implemented using OpenGL.

Jesus loves you.

- ✓ `paintGL()` – A virtual function for rendering (display) OpenGL. This function is invoked when the `QGLWidget` widget is updated.
- ✓ `resizeGL()` – Set the viewport, project, etc. of OpenGL. Called automatically when the size of the widget changes. A resize event is automatically invoked when `QGLWidget` is first run.
- ✓ `initializeGL()` – Set the OpenGL rendering context. `paintGL()`, `resizeGL()` is called for initialization before the function is called.

The following is an example source code using `QOpenGLWidget`.

```
...
class MyGLDrawer : public QOpenGLWidget
{
    Q_OBJECT
public:
    MyGLDrawer(QWidget *parent)
        : QGLWidget(parent) {}

protected:
    void initializeGL() override {
        ...
        glClearColor(0.0, 0.0, 0.0, 0.0);
        glEnable(GL_DEPTH_TEST);
        ...
    }

    void resizeGL(int w, int h) override {
        glViewport(0, 0, (GLint)w, (GLint)h);
        ...
        glFrustum(...);
        ...
    }

    void paintGL() override {
```

Jesus loves you.

```
// draw the scene:  
glRotatef(...);  
glMaterialfv(...);  
glBegin(GL_QUADS);  
glVertex3f(...);  
glVertex3f(...);  
...  
glEnd();  
...  
}  
};  
...
```

You can use updateGL() to update QGLWidget, such as using update() to update the QWidget widget area. When this function is called, paintGL() is called.

And for OpenGL-enabled systems, QGLWidget supports three overlay functions to use overlays on the system.

- ✓ paintOverlayGL() – Same as paintGL(). An override function that is used to calculate the overlay of the widget instead of the main context of the widget.
- ✓ resizeOverlayGL() – resizegl() and is similar in function. Of the widget main context of the overlay function of the fiscal for context widget.
- ✓ initializeOverlayGL() – equal to initializeGL() An override function that uses the widget's overlay operation instead of the widget's main context. This function is called only once before calling paintOverlayGL() or resizeOverlayGL(). Use this function to set the context rendering flag for OpenGL

Because the QOpenGLWidget and QGLWidget are inherited from the QWidget, they can use pintEvent() for 2D graphics rendering used in the QWidget class.

```
...  
class GLWidget : public QOpenGLWidget  
{
```

Jesus loves you.

```
Q_OBJECT

public:
    GLWidget(Helper *helper, QWidget *parent);

public slots:
    void animate();

protected:
    void paintEvent(QPaintEvent *event) override;

private:
    Helper *helper;
    int elapsed;
};

...
```

In addition, the `paintGL()` used by `QOpenGLWidget` and `QGLWidget` can use the `paintEvent()` function to render 2D graphic elements, such as rendering 3D graphic elements using OpenGL.

- Triangle 3D Rendering example

The purpose of this example is a simple example of how 3D graphics are rendered.

This is an example of how to use `initializeGL()`, `resizeGL()`, and `paintGL()` in some way using the `QOpenGLWidget` class. In this example, let's also take a look at how to use the `QGLShader` class to use GLSL(GL Shading Language)

The following example source code is a source code written in GLSL, the file name is `verexShader.vsh`, and you create the source code as follows:

```
uniform mat4 mvpMatrix;
in vec4 vertex;

void main(void)
{
    gl_Position = mvpMatrix * vertex;
}
```

The following example source code is `FragmentShader.fsh` source code.

Jesus loves you.

```
#version 130

uniform vec4 color;
out vec4 fragColor;

void main(void)
{
    fragColor = color;
}
```

Create two GLSL files as shown in the example above. Then, write the source code of glwidget.h and glwidget.cpp by inheriting the QOpenGLWidget class as follows: The following example source code is the glwidget.h header file source code.

```
#ifndef GLWIDGET_H
#define GLWIDGET_H

#include <QtOpenGL>
#include <QOpenGLWidget>
#include <QGLShaderProgram>

class GlWidget : public QOpenGLWidget
{
    Q_OBJECT

public:
    GlWidget(QWidget *parent = 0);
    ~GlWidget();
    QSize sizeHint() const;

protected:
    void initializeGL();
    void resizeGL(int width, int height);
    void paintGL();

private:
    QMatrix4x4 pMatrix;
    QGLShaderProgram shaderProgram;
    QVector<QVector3D> vertices;
};

#endif // GLWIDGET_H
```

Jesus loves you.

As shown in the example above, the GlWidget class is inherited and implemented from the QOpenGLWidget class. The following example source code is the glwidget.cpp source code.

```
#include "glwidget.h"

GlWidget::GlWidget(QWidget *parent) : QOpenGLWidget(parent)
{
}

GlWidget::~GlWidget()
{
}

QSize GlWidget::sizeHint() const
{
    return QSize(640, 480);
}

void GlWidget::initializeGL()
{
    QOpenGLFunctions *f = QOpenGLContext::currentContext()->functions();

    f->glEnable(GL_DEPTH_TEST);
    f->glEnable(GL_CULL_FACE);
    f->glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

    shaderProgram.addShaderFromSourceFile(QGLShader::Vertex,
                                        (":/vertexShader.vsh"));

    shaderProgram.addShaderFromSourceFile(QGLShader::Fragment,
                                        (":/fragmentShader.fsh"));
    shaderProgram.link();

    vertices << QVector3D(1, 0, -2)
        << QVector3D(0, 1, -2)
        << QVector3D(-1, 0, -2);
}

void GlWidget::resizeGL(int width, int height)
{
    if (height == 0)
```

Jesus loves you.

```
height = 1;

pMatrix.setToIdentity();
pMatrix.perspective(60.0, (float)width / (float)height, 0.001, 1000);

QOpenGLFunctions *f = QOpenGLContext::currentContext()->functions();
f->glViewport(0, 0, width, height);
}

void GlWidget::paintGL()
{
    QOpenGLFunctions *f =
    QOpenGLContext::currentContext()->functions();

    f->glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    QMatrix4x4 mMatrix;
    QMatrix4x4 vMatrix;

    shaderProgram.bind();

    shaderProgram.setUniformValue("mvpMatrix", pMatrix * vMatrix * mMatrix);
    shaderProgram.setUniformValue("color", Color(Qt::white));

    shaderProgram.setAttributeArray("vertex", vertices.constData());
    shaderProgram.enableAttributeArray("vertex");

    f->glDrawArrays(GL_TRIANGLES, 0, vertices.size());
    shaderProgram.disableAttributeArray("vertex");

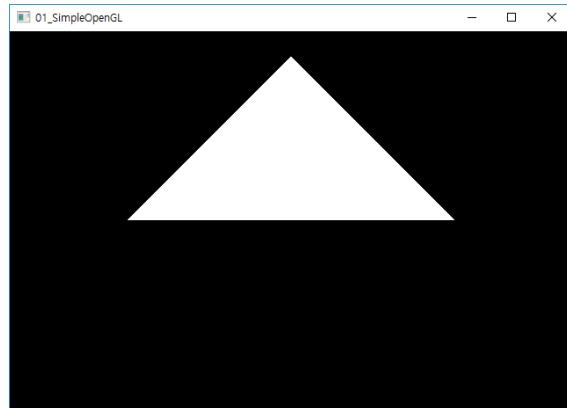
    shaderProgram.release();
}
```

The shaderProgram object of the QGLShaderProgram class declared in the header file reads the source code written in GLSL and provides functions that can be used in Qt.

GLSL can be used using functions such as setUniformValue() of QGLShaderProgram class, setAttributeArray() and enableAttributeArray() within the paintGL() Virtual function.

Example - Ch07 > 01_SimpleOpenGL

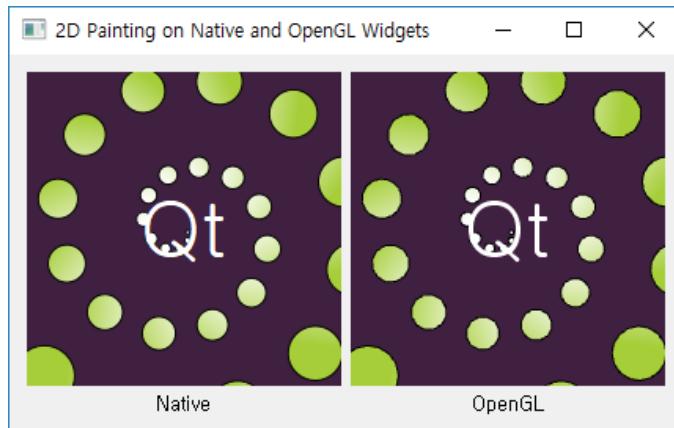
Jesus loves you.



<FIGURE> Example screen

- Example for using the paintEvent() function in QOpenGLWidget

The `paintEvent()` function can also be used in `QOpenGLWidget`, such as using `paintEvent()` as a method for rendering 2D graphic elements used in `QWidget`. In this example, let's learn how to use the `paintEvent()` function in `QOpenGLWidget`. This example source code is an example provided by the accompanying Exam when you install Qt.



<FIGURE> Example screen

As shown above, the widget on the left inherited the `QWidget` class and used `paintEvent()`. The right-hand widget is an example of inheriting the `QOpenGLWidget` class and using `paintEvent()`. This example source code consists of four classes.

Window classes are classes for placing Widget Class widgets and GLWidget Class widgets on the GUI in Windows as shown in the figure above. The Helper class also implements

Jesus loves you.

the ability to render 2D graphics elements as shown in the above figure in the paintEvent() function of the Widget class and GLWidget class. The following example source code is the header source code of the Helper class.

```
...
class Helper
{
public:
    Helper();
public:
    void paint(QPainter *painter, QPaintEvent *event, int elapsed);

private:
    QBrush background;
    QBrush circleBrush;
    QFont textFont;
    QPen circlePen;
    QPen textPen;

};
```

The first factor in the paint() function of this class passes over the pointer on the QPainter used by the Widget class and GLWidget class. Therefore, it is a class that provides the ability to draw 2D graphic elements as shown in the example execution screen above the QPainter in this function. The following example source code is the function implementation sub-source code of the Helper class.

```
...
Helper::Helper()
{
    QLinearGradient gradient(QPointF(50, -20), QPointF(80, 20));
    gradient.setColorAt(0.0, Qt::white);
    gradient.setColorAt(1.0, QColor(0xa6, 0xce, 0x39));

    background = QBrush(QColor(64, 32, 64));
    circleBrush = QBrush(gradient);
    circlePen = QPen(Qt::black);
    circlePen.setWidth(1);
    textPen = QPen(Qt::white);
    textFont.setPixelSize(50);
```

Jesus loves you.

```
{}

void Helper::paint( QPainter *painter, QPaintEvent *event, int elapsed )
{
    painter->fillRect(event->rect(), background);
    painter->translate(100, 100);

    painter->save();
    painter->setBrush(circleBrush);
    painter->setPen(circlePen);
    painter->rotate(elapsed * 0.030);

    qreal r = elapsed / 1000.0;
    int n = 30;

    for (int i = 0; i < n; ++i)
    {
        painter->rotate(30);
        qreal factor = (i + r) / n;
        qreal radius = 0 + 120.0 * factor;
        qreal circleRadius = 1 + factor * 20;
        painter->drawEllipse(QRectF(radius, -circleRadius,
                                     circleRadius * 2, circleRadius * 2));
    }

    painter->restore();
    painter->setPen(textPen);
    painter->setFont(textFont);
    painter->drawText(QRect(-50, -50, 100, 100),
                      Qt::AlignCenter, QStringLiteral("Qt"));
}

...
```

As shown in the example above, the paint() member function is used in the Widget class and GLWidget class. The following example source code is the header source code of the GLWidget class.

```
#include <QOpenGLWidget>

class Helper;
class GLWidget : public QOpenGLWidget
{
```

Jesus loves you.

```
Q_OBJECT
public:
    GLWidget(Helper *helper, QWidget *parent);

public slots:
    void animate();

protected:
    void paintEvent(QPaintEvent *event) override;

private:
    Helper *helper;
    int elapsed;
};
```

The helper object is used in the paintEvent() function. The following example is the function implementation source code for this class.

```
GLWidget::GLWidget(Helper *helper, QWidget *parent)
    : QOpenGLWidget(parent), helper(helper)
{
    elapsed = 0;
    setFixedSize(200, 200);
    setAutoFillBackground(false);
}

void GLWidget::animate()
{
    elapsed = (elapsed + qobject_cast<QTimer*>(sender())->interval()) % 1000;
    update();
}

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

    painter.begin(this);
    painter.setRenderHint(QPainter::Antialiasing);
    helper->paint(&painter, event, elapsed);
    painter.end();
}
```

The Helper class provides the ability to render actual 2D graphic elements as shown in

Jesus loves you.

the example source code above. The patternEvent() function of the Widget class inherited and implemented by the QWidget class uses the Helper class in the same way as above.

In the example source code above, the animate() function is called in 50 ms(Millisecond) units with the QTimer used in the Window class, the parent window in which QGLWidget is placed. Therefore, the paintEvent() function is called every 50 ms period. The following example source code is part of the function implementation part of the window class.

```
...
Window::Window()
{
    setWindowTitle(tr("2D Painting on Native and OpenGL Widgets"));
    Widget *native = new Widget(&helper, this);
    GLWidget *openGL = new GLWidget(&helper, this);

    QLabel *nativeLabel = new QLabel(tr("Native"));
    nativeLabel->setAlignment(Qt::AlignHCenter);

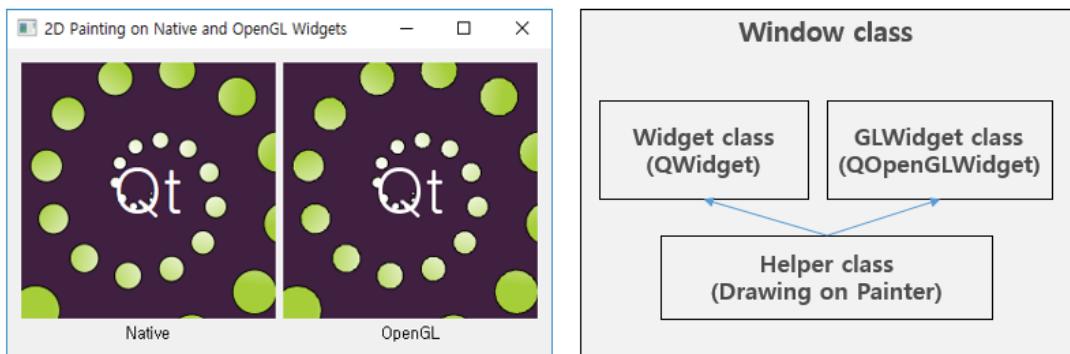
    QLabel *openGLLabel = new QLabel(tr("OpenGL"));
    openGLLabel->setAlignment(Qt::AlignHCenter);

    QGridLayout *layout = new QGridLayout;
    layout->addWidget(native, 0, 0);
    layout->addWidget(openGL, 0, 1);
    layout->addWidget(nativeLabel, 1, 0);
    layout->addWidget(openGLLabel, 1, 1);
    setLayout(layout);

    QTimer *timer = new QTimer(this);
    connect(timer, &QTimer::timeout, native, &Widget::animate);
    connect(timer, &QTimer::timeout, openGL, &GLWidget::animate);
    timer->start(50);
}
...
```

As shown in the example above, the Widget class and the paintEvent() of the GLWidget class continue to be called at a 50 ms interval with QTimer.

Jesus loves you.



<FIGURE> The relationship between the example run screen and the class

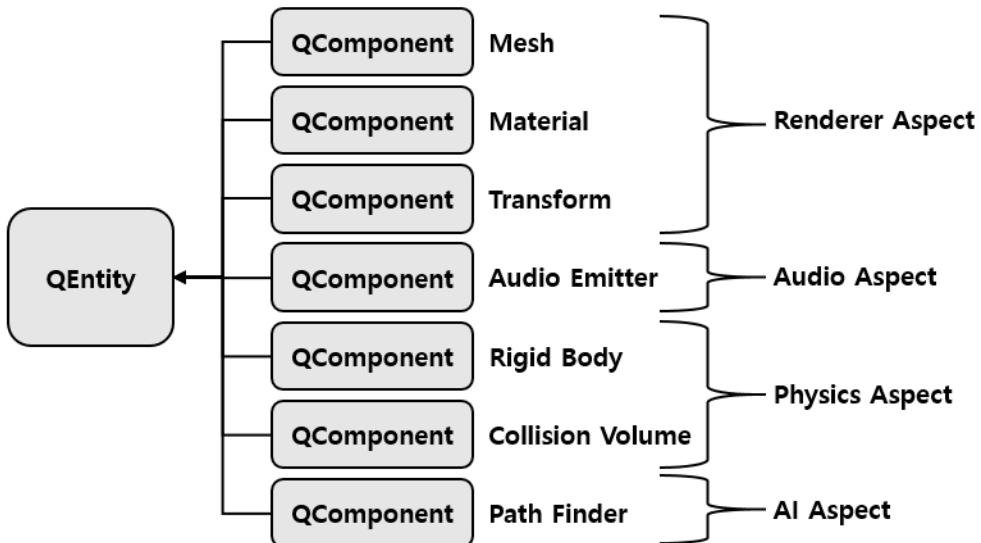
Example - Ch07 > 2dpainting

8. 3D Graphics Using Qt 3D Module

Qt 3D module is a module that supports 3D graphics rendering. It provides an API that is easier and more intuitive than OpenGL. In addition to 3D graphics rendering, the Qt 3D module also includes physics, audio, collision detection and artificial intelligence.

The Qt 3D module was first introduced in Qt 5.5 version and was released as a full version from Qt 5.7. GPU is used to ensure graphics performance on various platforms (operating systems), such as OpenGL. The Qt 3D module supports 3D graphics rendering as well as 2D graphics rendering. Qt 3D meets different rendering approaches in the OpenGL version and enables multiple rendering passes. It is also available in C++ and QML. Qt 3D modules can easily map GLSL Shader.

Qt 3D supports Vertex, Tessellation control, Tessellation evaluation, geometry, and Fragment shaders like OpenGL. Qt 3D modules use ECS (Entity Component System).



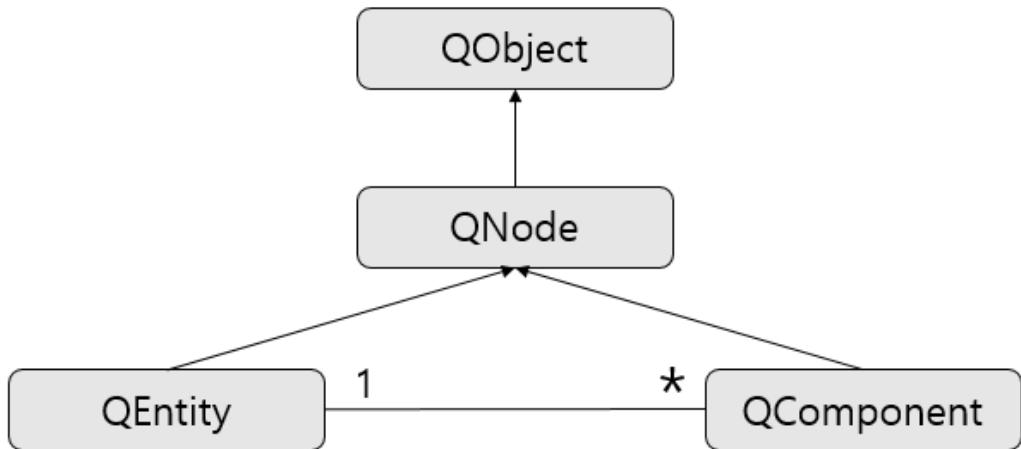
<FIGURE> ECS Structure

The ECS itself does not represent any 3D itself, but the Entity includes one or more components. In other words, ECS represents a simulated object, but does not have any specific behavior or characteristics in itself. As shown in the following figure, a single entity can construct multiple components by adding one or more components.

Jesus loves you.

As shown in the figure above, the Qt 3D module implements the ECS as a simple class layer. QEntity and QComponent classes were implemented in succession to the QNode class and QNode was implemented in succession to QObject.

Qt3DCore::QEntity can contain multiple Qt3DCore::QComponent as objects.



<FIGURE> class hierarchy

In addition to 3D graphics, Qt 3D also supports 2D graphics rendering and various classes supported by Qt 3D. It supports a considerable number of classes and can be classified into seven classes as follows

<TABLE> Qt 3D modules

Classification	Class / Module	Explanation
C++	Qt3DCore	Includes the ability to support real-time simulation systems.
	Qt3DInput	Class for processing user input
	Qt3DLogic	Class to support Qt3D Backend and frame synchronization.
	Qt3DRender	Classes to support 3D and 2D rendering.
	Qt3DAnimation	Class for providing animation.
	Qt3DExtras	Provides pre-built 3D elements.
	Qt3DScene2D	Rendering QML content with Qt 3D texture

Jesus loves you.

QML	Qt3D.Core	Includes the ability to support real-time simulation systems.
	Qt3D.Input	QML module for processing user inputs
	Qt3D.Logic	Module to support frame synchronization with Qt3D Backend
	Qt3D.Render	Classes to support 3D and 2D rendering.
	Qt3D.Animation	QML module to provide animation
	Qt3D.Extras	Provides pre-built 3D elements.
	QtQuick.Scene2D	Scened2d Module as Qt 3D QML Type
	QtQuick.Scene3D	Scened3d Module as Qt 3D QML Type

As shown in the table above, the Qt3DAnimation, Qt3DExtras and Qt3DRender class items are available as Preview but are still under development.

Qt3D in QML.Animation, Qt3D.Extras, QtQuick.Scene2D, qtQuick.Scene 3D module is an item that is still being developed. To use the Qt 3D module, add the following to the project file(.pro)

```
QT += 3dcore 3drender 3dinput 3dlogic 3dextras 3danimation
```

In addition, the Qt 3D module can be used only with the following exclude:

```
#include <Qt3DCore>
#include <Qt3DRender>
#include <Qt3DInput>
#include <Qt3DLogic>
#include <Qt3DExtras>
#include <Qt3DAnimation>
```

In order to use the Qt3D module in QML, the following shall be added to the project file:

```
QT += 3dcore 3drender 3dinput 3dlogic 3dextras
QT += qml quick 3dquick 3danimation
```

The Qt 3D module supports MS Windows, Linux (X11-based), macOS, Android, and Embedded Linux, and WinRT is not yet supported.

Jesus loves you.

- 3D example



<FIGURE> Example screen

This example is a 3D rendering using TEXT rendering API. As shown in the figure above, the QSlider widgets on the right are widgets that allow real-time camera positioning of the 3D area on the left. Each QSlider widget can change the x, y, and z locations of the camera.

If you change the X, Y, and Z values of the QSlider, you can change the x, y, and z values using the `setPosition()` function provided by the `Qt3DRender::QCamera` class. The following is the header file source code.

```
#ifndef WIDGET_H
#define WIDGET_H

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

private:
    Ui::Widget *ui;
    Qt3DEExtras::Qt3DWindow *m_view;
    Qt3DCore::QEntity      *m_rootEntity;
    Qt3DRender::QCamera     *m_camera;
    Qt3DEExtras::QOrbitCameraController *camController;
    Qt3DCore::QEntity *m_textEntity;
```

Jesus loves you.

```
void createRootEntry();

QVector3D m_camPos3D;
QVector3D m_camUpVector3D;
QVector3D m_camViewCenter3D;

public slots:
    void setCamPosX_ValueChanged();
    void setCamPosY_ValueChanged();
    void setCamPosZ_ValueChanged();
};

#endif // WIDGET_H
```

Qt3DExtras::Qt3DWindow class is a widget of Qt 3D. A VIEW area widget to render 3D graphics elements. In the header file source code above, the m_view object is a real area widget. m_rootEntry is the highest QEntity. The next is the widget.cpp source code.

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

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

    m_camPos3D      = QVector3D(0, 0, 40.0);
    m_camUpVector3D = QVector3D(0, 1, 0);
    m_camViewCenter3D = QVector3D(0, 0, 0);

    m_view = new Qt3DExtras::Qt3DWindow();
    m_view->defaultFrameGraph()->setClearColor(QColor(77, 77, 77));

    m_rootEntity = new Qt3DCore::QEntity;
    m_view->setRootEntity(m_rootEntity);

    m_camera = m_view->camera();
    m_camera->lens()->setPerspectiveProjection(45.0f, 2.0f, 0.1f, 2000.0f);

    m_camera->setPosition(m_camPos3D);
    m_camera->setUpVector(m_camUpVector3D);
    m_camera->setViewCenter(m_camViewCenter3D);
```

Jesus loves you.

```
Qt3DEExtras::QOrbitCameraController *camController;
camController = new Qt3DEExtras::QOrbitCameraController(m_rootEntity);

camController->setLinearSpeed( 50.0f );
camController->setLookSpeed( 180.0f );
camController->setCamera(m_camera);

createRootEntry();
QWidget *container = QWidget::createWindowContainer(m_view);
ui->horizontalLayout->addWidget(container);

connect(ui->xSlider, &QSlider::valueChanged,
        this,           &Widget::setCamPosX_ValueChanged);
connect(ui->ySlider, &QSlider::valueChanged,
        this,           &Widget::setCamPosY_ValueChanged);
connect(ui->zSlider, &QSlider::valueChanged,
        this,           &Widget::setCamPosZ_ValueChanged);
}

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

void Widget::setCamPosX_ValueChanged()
{
    m_camPos3D.setX(ui->xSlider->value());
    m_camera->setPosition(m_camPos3D);
}

void Widget::setCamPosY_ValueChanged()
{
    m_camPos3D.setY(ui->ySlider->value());
    m_camera->setPosition(m_camPos3D);
}

void Widget::setCamPosZ_ValueChanged()
{
    m_camPos3D.setZ(ui->zSlider->value());
    m_camera->setPosition(m_camPos3D);
}
```

Jesus loves you.

```
void Widget::createRootEntry()
{
    // Root entity
    Qt3DExtras::QExtrudedTextMesh *urlTextMesh;
    urlTextMesh = new Qt3DExtras::QExtrudedTextMesh();

    urlTextMesh->setText("www.qt-dev.com");

    Qt3DExtras::QPhongMaterial *urlTextMaterial;
    urlTextMaterial = new Qt3DExtras::QPhongMaterial();
    urlTextMaterial->setDiffuse(QColor(Qt::green));

    Qt3DCore::QTransform *urlTextTransform = new Qt3DCore::QTransform();
    urlTextTransform->setScale(2.0f);
    urlTextTransform->setTranslation(QVector3D(-10.0f, -4.0f, 15.0f));

    m_textEntity = new Qt3DCore::QEntity(m_rootEntity);
    m_textEntity->addComponent(urlTextMesh);
    m_textEntity->addComponent(urlTextMaterial);
    m_textEntity->addComponent(urlTextTransform);
}

...
```

m_camera is an object in the Qt3DRender::QCamera class. As shown in the above source code, the setPerspectiveProjection() function provided by this class provides a function for displaying the camera Perspective in the 3D area.

The first factor is the vertical viewing angle (Y axis), the second factor is the horizontal viewing angle (X axis), the third factor is the front cutting plane, and the fourth factor is the rear cutting plane.

Qt3DRender::The setPosition() member function of the QCamera class can set the camera position in 3D space. The setUpVector() is the Up Vector of the camera, and the setViewCenter() can set the Center View position of the camera.

createRootEntry() added a text object to the highest QEntity in m_rootEntity. And the Slot function is a function called when the value of QSlider changes on the GUI.

Example - Ch08 > 01_Simple3D

9. XML

Qt provides an API that can address XML (extensible markup language). To use the XML module provided by Qt, add the following to the project file:

```
QT += xml
```

Qt provides a Document object model (DOM) method and a Simple API for XML (SAX) method as a way to use XML.

DOM stores all XML content in memory in a tree and then reads it. DOM method has in memory, so it is easy to edit such as modification and deletion.

SAX method is easier to implement than DOM method. In addition, it is difficult to modify or delete the read data or the analyzed data because they do not store it in memory. Both DOM and SAX methods provide a variety of APIs that are available on Qt

The QXmlStreamReader class provided by Qt provides the ability to READ the data in XML format, while the QXmlStreamWriter class provides the ability to WRITE in XML format.

The QXmlStreamReader and QXmlStreamWriter classes can be used by connecting the QFile objects provided by Qt.

As shown in the following example, you can assign a QFile object as the first factor to the setDevice() member function provided by the QXmlStreamReader class. The following is an example of an XML format data file.

```
<?xml version="1.0" encoding="utf-8"?>
<students>
    <student>
        <firstName>Dae Jin</firstName>
        <lastName>Kim</lastName>
        <grade>3</grade>
    </student>
    <student>
        <firstName>Kim</firstName>
        <lastName>Min Soo</lastName>
        <grade>4</grade>
```

Jesus loves you.

```
</student>  
...  
</students>
```

As shown in the example above, the QXmlStreamReader class can be used to READ files stored in XML format. The following is part of an example source code that uses the QXmlStreamReader class to retrieve XML from files stored in the above XML format.

```
...  
  
QFile file(QFileDialog::getOpenFileName(this, "Open"));  
  
QXmlStreamReader xmlReader;  
xmlReader.setDevice(&file);  
  
QList<Student> students;  
xmlReader.readNext();  
  
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();  
    }  
}  
...  
...
```

The first factor in the setDevice() member function of the QXmlStreamReader class reads the selected file as a file dialogue. And an example is to save the read data to a QList called studies.

As shown in the example above, Qt can read XML format data stored in files. In addition, the QXmlStreamWriter class allows you to WRITE XML format data to files.

Jesus loves you.

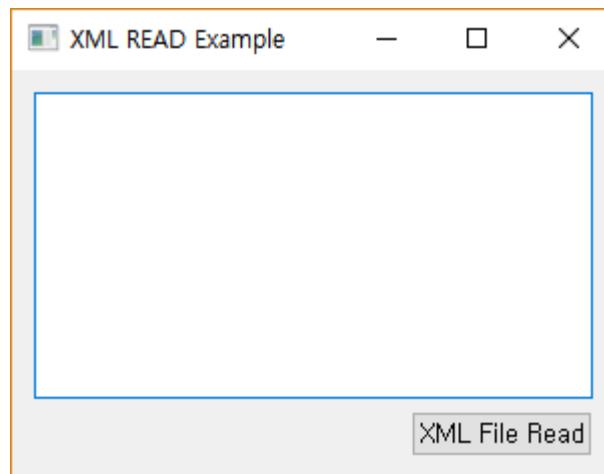
```
QXmlStreamWriter stream(&output);

stream.setAutoFormatting(true);
stream.writeStartDocument();
...
stream.writeStartElement("bookmark");
stream.writeAttribute("href", "http://www.qt-dev.com");
stream.writeTextElement("title", "Qt developer community");
stream.writeEndElement(); // bookmark
...
```

In the example above, such as watching easy to use the class `qxmlstreamwriter`. The following is a Let's take a look at how to handle the actual example allows XML

- Example of an XML file READ using `QXmlStreamReader`

This example reads the XML format data stored in the file from the file and outputs it on `QTextEdit`, and the example execution screen is as follows.



<FIGURE> Example screen

Click the [XML file Read] button as shown in the figure above to select the file from the file dialogue. The selected file is an XML file, and the `sample.xml` file is provided in this example directory with a `sample.xml` package and the contents of the XML file are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xbel version="1.0">
    <folder folded="yes">
```

Jesus loves you.

```
<title>Programming</title>
<bookmark href="http://qt-dev.com">
<title>Qt Developer Community</title>
</bookmark>

<bookmark href="http://www.google.com">
<title>Google</title>
</bookmark>
</folder>
</xbel>
```

An example of reading and outputting XML format data stored in the sample.xml file to the QTextEdit widget, as shown above, is the example header file.

```
#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();

};
```

The mReadFile object is an object in the QFile class that will read the XML file. And the readButtonClicked() function is a Slot function that is invoked when you click the [Read from XML file] button, as shown on the example execution screen. The following example source code is the widget.cpp source code.

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

Jesus loves you.

```
#include "ui_widget.h"
#include <QFileDialog>
#include <QXmlStreamReader>
#include <QDebug>

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

    mReadFile = new QFile();
}

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

void Widget::readButtonClicked()
{
    QString fName = QFileDialog::getOpenFileName(this, "Open XML File",
                                                QDir::currentPath(),
                                                "XML Files (*.xml)");
    mReadFile->setFileName(fName);

    if(! QFile::exists(fName)) {
        ui->textEdit->setText("There is no XML file. ");
        return;
    }

    if(!mReadFile->open(QIODevice::ReadOnly)) {
        ui->textEdit->setText("File Open Fail");
        return;
    }

    QXmlStreamReader reader(mReadFile);
    QString inputData;
    while(!reader.atEnd())
    {
        reader.readNext();
        if(!reader.text().isEmpty()) {
```

Jesus loves you.

```
QString data = reader.text().toString();
data.replace('\n', "");
data.replace('\t', "");

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

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

When the ReadButtonClicked() Slot function is called, as shown in the example source code above, the file dialogue is loaded.

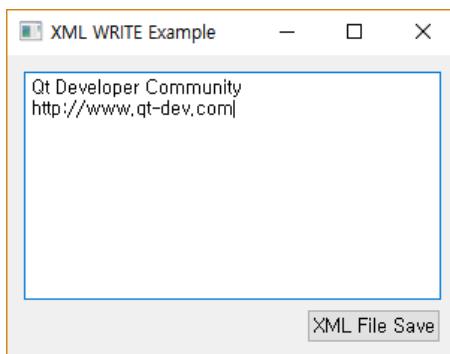
If you select an XML file in the file dialogue, the XML data is read using the QXmlStreamReader class. The readNext() member function in the QXmlStreamReader class provides the ability to read XML tags sequentially, and the text() member function reads the text() text of the actual tag.

Then, save the read text in QString to output it to QTextEdit. After saving the XML file READ, print the text stored in the saved QString to QTextEdit .

Example - Ch09 > 01_ReaderExample

- Example implementation of saving to an XML file using QXmlStreamWriter

This example is an example of saving XML format data to a file. An example is to read the data printed on the QTextEdit, change it to XML format data, and save it to a file, as shown in the figure below.



<FIGURE> Example screen

Jesus loves you.

Click on the [XML File Save] button as shown in the example execution screen above to save the XML in the "C:/output.xml" file. The following is the source code file, 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           *mWriteFile;
    QList<QString> mOriData;

public slots:
    void writeButtonClicked();
};


```

The mWriteFile object is a QFile object. mOriData is a variable that stores the data that is output to QTextEdit. The following example source code is the widget.cpp source code.

```
#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::writeButtonClicked);

    mWriteFile = new QFile();
}


```

Jesus loves you.

```
mOriData.append("Qt Developer Community");
mOriData.append("http://www.qt-dev.com");

for(int i = 0 ; i < mOriData.count() ; i++)
    ui->textEdit->append(mOriData.at(i));
}

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

void Widget::writeButtonClicked()
{
    QFile file("C:/output.xml");
    file.open(QIODevice::WriteOnly);

    QXmlStreamWriter xmlWriter(&file);
    xmlWriter.setAutoFormatting(true);
    xmlWriter.writeStartDocument();

    xmlWriter.writeStartElement("Qt");

    xmlWriter.writeStartElement("Info");

    xmlWriter.writeTextElement("Name", mOriData.at(0));
    xmlWriter.writeTextElement("URL", mOriData.at(1));

    xmlWriter.writeEndElement(); // Info End tag
    xmlWriter.writeEndElement(); // Qt End tag

    file.close();
}
```

In the writeButtonClicked() function, the writeStartElement() function of the QXmlStreamWriter class is an XML starting tag. In addition, the writeTextElement() function allows you to enter the text that you want to save to the tag. Finally, the writeEndElement() function stores the last tag in the actual file. If the file is closed after the growth as shown above, the XML data is stored in the "C:/output.xml" file and the result is as follows.

Jesus loves you.

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

Example - Ch09 > 02_WriteExample

- DOM Example

This example uses the DOM (Document object model) to retrieve XML format data from a file. Save the dom.xml file name as shown in the picture below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Building>
    <Korea ID="0" Name="Korea Building 0">
        <Korea ID="0" Name="Room 0"/>
        <Korea ID="0" Name="Room 1"/>
        <Korea ID="0" Name="Room 2"/>
    </Korea>
    <Korea ID="1" Name="Korea Building 1">
        <Korea ID="1" Name="Sub Room 0"/>
        <Korea ID="1" Name="Sub Room 1"/>
        <Korea ID="1" Name="Sub Room 2"/>
    </Korea>
</Building>
```

The QDomDocument class can be used to retrieve the above XML data from the file and the source code is written as follows.

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

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

    for(int i = 0; i < nodes.count(); i++)
    {
```

Jesus loves you.

```
QDomNode elm = nodes.at(i);
if(elm.isElement()){
    QDomElement e = elm.toElement();
    qDebug() << "Attribute : " << e.attribute(att);
}
}

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

    QDomDocument document;
    QFile file("c:/dom.xml");
    if(!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        qDebug() << "Failed to open file.";
        return -1;
    } else {
        if(!document.setContent(&file)) {
            qDebug() << "Failed to reading.";
            return -1;
        }
        file.close();
    }

    QDomElement root = document.firstChildElement();
    retrievElements(root, "Korea", "Name");

    return a.exec();
}
```

The first factor in the retrievElements() function is the object of the QDomElement class. The second factor specifies "Korea" tags in XML data and the third factor specifies "Attr" attributes. Outputs the specified tags and attributes using qDebug().

```
Node Counts = 8
Attribute : "Korea Building 0"
Attribute : "Room 0"
Attribute : "Room 1"
Attribute : "Room 2"
Attribute : "Korea Building 1"
Attribute : "Sub Room 0"
```

Jesus loves you.

Attribute : "Sub Room 1"

Attribute : "Sub Room 2"

Example - Ch09 > 03_DomExample

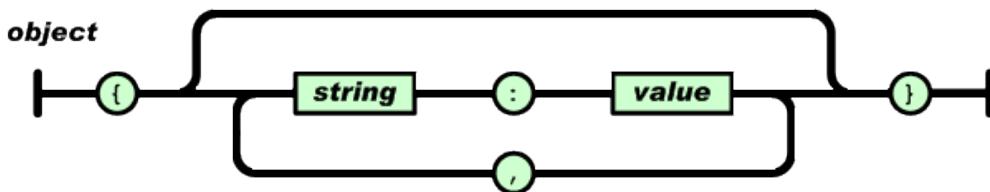
10. JSON

JSON (JavaScript Object Notation) is the same as XML. The difference is the lightweight DATA exchange method, which has the advantage of having a smaller data usage capacity than XML. For example, XML wastes a lot of data to store the words needed for the tag. However, JSON uses the symbols " and [] instead of tags, which saves space in storing data compared to XML. The following is a comparison of XML and JSON data formats.

XML	JSON
<pre><?xml version="1.0" encoding="UTF-8"?> <note> <to>Tove</to> <from>Jani</from> <heading>Reminder</heading> </note> <note> <to>Tove</to> <from>Jani</from> <heading>Reminder</heading> </note></pre>	<pre>{ "FirstName": "John", "Age": 43, "Address": { "Street": "Downing Street 10", "Country": "Great Britain" }, "Phone numbers": ["+44 1234567", "+44 2345678"] }</pre>

<FIGURE> XML and JSON

As shown in the example above, JSON has the potential to use less data than XML when communicating data between these models. JSON is stored as a pair in string/value form.



<FIGURE> JSON Structure

Use ":" to distinguish between string and value and one pair is divided by "," The following example source code is JSON format data.

```
{ "FirstName": "John", "LastName": "Doe", "Age": 43,
  "Address": {
    "Street": "Downing Street 10", "City": "Seoul", "Country": "Korea"
  },
```

Jesus loves you.

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

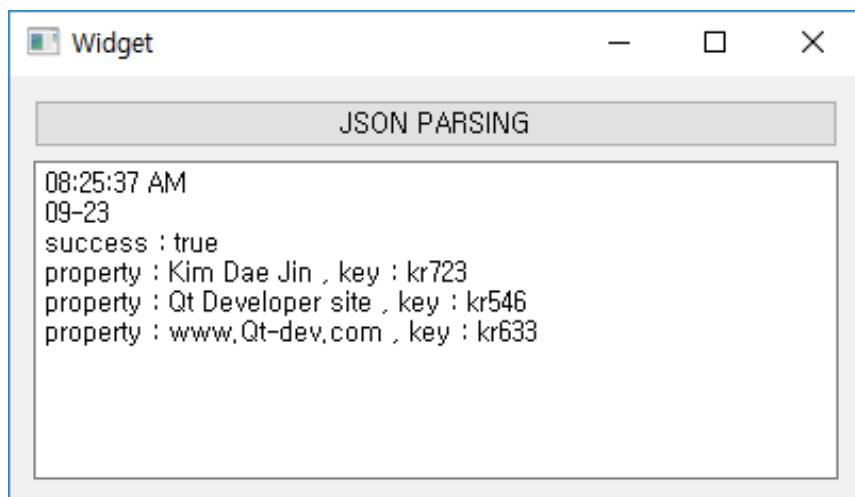
As shown in the example above, the JSON module provided by Qt can use six data types: Bool, Double, String, Array, Object, and Null. Start with { (left square) and end with } (right square bracket).

And [(column) and] (column) can be used as the beginning and end. In addition, brackets and brackets can be used in overlapping structures. Let's take a look at how to use JSON through a real-world example.

- JSON Parsing Example

This example takes JSON data from the Sample.json file and outputs it from the GUI to the QPlainTextEdit widget. The sample Sample.json file contains the following contents.

```
{  
    "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" }  
    ]  
}
```



<FIGURE> Example screen

Jesus loves you.

Click the [JSON PARSING] button as shown in the figure above to output the results to the QPlainTextEdit widget on the GUI. The following is the source code for this example.

```
#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
```

In the example above, the slotPbtJSONParser() Slot function is a Slot function invoked when the [JSON PARSING] button is clicked. The parsJSON() function is a function that parses data that is read from a JSON file. The addText() function is a function for outputting data extracted by the parsJSON() function to the widget. Next is the widget.cpp source code.

```
...
Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pbtJSONParser, SIGNAL(pressed()),
            this,           SLOT(slotPbtJSONParser()));
}
```

Jesus loves you.

```
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();
    }
}
```

Jesus loves you.

```
QString property = obj["PropertyName"].toString();
QString key      = obj["key"].toString();

QString arrayData; = QString("property : %1 , key : %2 \n")
                     .arg(property).arg(key);
addText(arrayData);
}

}

void Widget::addText(const QString &addLine)
{
    ui->textEdit->insertPlainText(addLine);
}
```

The first factor in the parseJSON() function is the original JSON data read from the JSON file. The original data you have read is used by the jsonResponse object in QJsonDocument.

QJsonDocument::fromJson() The first factor in the function is QByteArray. Therefore, to replace the QString original data with QByteArray, the toLocal8Bit() member function of the QByteArray class was used.

The jsonResponse object in the QJsonDocument class stores information about the JSON source data processed fromJson() function. Pass over the QJsonObject type using the object() member function of the QJsonDocument class so that the stored data can be extracted using the QJsonObject class.

And to access the actual data, pass the QJsonObject class type back to QJsonArray. You can then use the QJsonValue class, as used in the foreach statement, to access each data.

Example - Ch09 > 01_JSON_Parser

11. Qt Chart

Qt provides a graphical API that allows data to be represented in graph charts. It offers various kinds of charts such as Area Chart, Pie Chart, Line Chart, Bar Chart, Spline Chart, and Scatter Chart.



<FIGURE> Chart example

To use the Chart API provided by Qt, the Qt Chart Module must be added to the project file(.pro) as follows:

```
QT += charts
```

The Qt Chart module consists of an API for use and is available in C++ and QML. For example, you can use the `QLineSeries` class to create a source code as follows.

```
QLineSeries* series = new QLineSeries();
series->add(0, 6);
series->add(2, 4);
...
chartView->chart()->addSeries(series);
chartView->chart()->createDefaultAxes();
```

To add X and Y data, you can add data using the `add()` member function of the `QLineSeries` class. The first factor of the `add()` member function is the X value and the second factor is the value of the Y axis. The following is an example of using a pie chart with QML.

Jesus loves you.

```
import QtQuick 2.0
import QtCharts 2.0

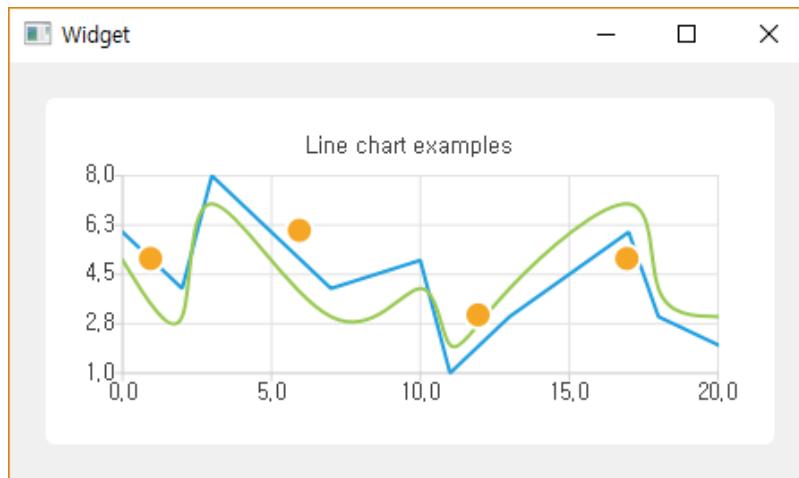
ChartView {
    width: 400
    height: 300
    theme: ChartView.ChartThemeBrownSand
    antialiasing: true

    PieSeries {
        id: pieSeries
        PieSlice { label: "eaten"; value: 94.9 }
        PieSlice { label: "not yet eaten"; value: 5.1 }
    }
}
```

The example source code above is the example source code that created the pie chart in QML. Both C++ and QML are available. The Qt Chart module can use animation effects when displaying charts. The Qt Chart module supports different kinds of graph charts. So let's look at the example.

- Line chart

This example shows a chart on the X and Y axes. Three types of data were displayed. The first was a straight graph, the second was a curved graph, and the third was a graph of the Scatter method.



<FIGURE> Example screen

Jesus loves you.

Straight as you see the picture above is qlineseries can be displayed using the class. The second curve type is qsplineseries, you can use the class. Lastly, the way scatter qscatterseries, you can use the class.

Example - Ch11 > 01_LineChart

```
...
#include <QHBoxLayout>
#include <QtCharts>
#include <QGraphicsView>

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

    QLineSeries* series = new QLineSeries();
    series->append(0, 6);
    series->append(2, 4);
    series->append(3, 8);
    series->append(7, 4);
    series->append(10, 5);
    *series << QPointF(11, 1) << QPointF(13, 3) << QPointF(17, 6)
        << QPointF(18, 3) << QPointF(20, 2);

    QSplineSeries* series1 = new QSplineSeries();
    series1->append(0, 5);
    series1->append(2, 3);
    series1->append(3, 7);
    series1->append(7, 3);
    series1->append(10, 4);
    *series1 << QPointF(11, 2) << QPointF(13, 4) << QPointF(17, 7)
        << QPointF(18, 4) << QPointF(20, 3);

    QScatterSeries* series2 = new QScatterSeries();
    *series2 << QPointF(1,5) << QPointF(6,6) << QPointF(12,3)
        << QPointF(17,5);

    QChart *chart = new QChart();
    chart->legend()->hide();
    chart->addSeries(series);
    chart->addSeries(series1);
    chart->addSeries(series2);
```

Jesus loves you.

```
chart->createDefaultAxes();
chart->setTitle("Line chart example");

QChartView *chartView = new QChartView(chart);
chartView->setRenderHint(QPainter::Antialiasing);

QHBoxLayout *hLay = new QHBoxLayout();
hLay->addWidget(chartView);

setLayout(hLay);
}

...
```

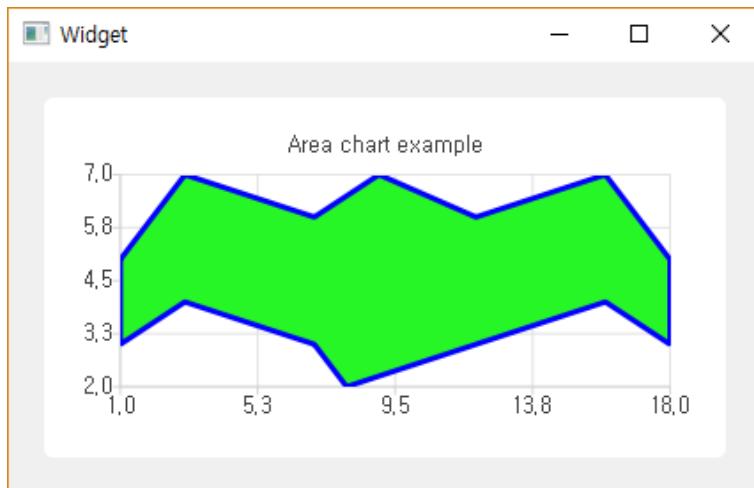
Insert data into objects in the QLineSeries, QSplineSeries, and QScatterSeries classes and add each graph using the addSeries() member function in the QChart class.

In addition, when declaring a QChartView class, you can declare the objects of the QChart class as the first factor of the constructor.

The QPainter::Antialiasing option used in the setRenderHint() function of the QChartView class is an option for using anti-aliasing.

- Area chart

Area charts can be used to display certain ranges in the graph ranges on the X and Y axes and can use the QAreaSeries classes provided by Qt.



<FIGURE> Example screen

Jesus loves you.

QAreaSeries classes can display the area on X and Y charts. In addition to setting up the Color inside the graph, Gradient can be used. The following is an example source code for the Area Chart. Example - Ch11 > 02_AreaChart

```
...
#include <QHBoxLayout>
#include <QtCharts>
#include <QGraphicsView>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    QLineSeries *series1 = new QLineSeries();
    QLineSeries *series2 = new QLineSeries();

    *series1 << QPointF(1, 5) << QPointF(3, 7) << QPointF(7, 6)
        << QPointF(9, 7) << QPointF(12, 6) << QPointF(16, 7)
        << QPointF(18, 5);

    *series2 << QPointF(1, 3) << QPointF(3, 4) << QPointF(7, 3)
        << QPointF(8, 2) << QPointF(12, 3) << QPointF(16, 4)
        << QPointF(18, 3);

    QAreaSeries *series = new QAreaSeries(series1, series2);
    series->setName("Area Data");
    QPen pen(Qt::blue);
    pen.setWidth(3);
    series->setPen(pen);

    QLinearGradient gradient(QPointF(0, 0), QPointF(0, 1));
    gradient.setColorAt(0.0, 0x3cc63c);
    gradient.setColorAt(1.0, 0x26f626);
    series->setBrush(gradient);

    QChart *chart = new QChart();
    chart->legend()->hide();
    chart->addSeries(series);

    chart->createDefaultAxes();
    chart->setTitle("Area chart example");

    QChartView *chartView = new QChartView(chart);
```

Jesus loves you.

```
chartView->setRenderHint(QPainter::Antialiasing);

QHBoxLayout *hLay = new QHBoxLayout();
hLay->addWidget(chartView);

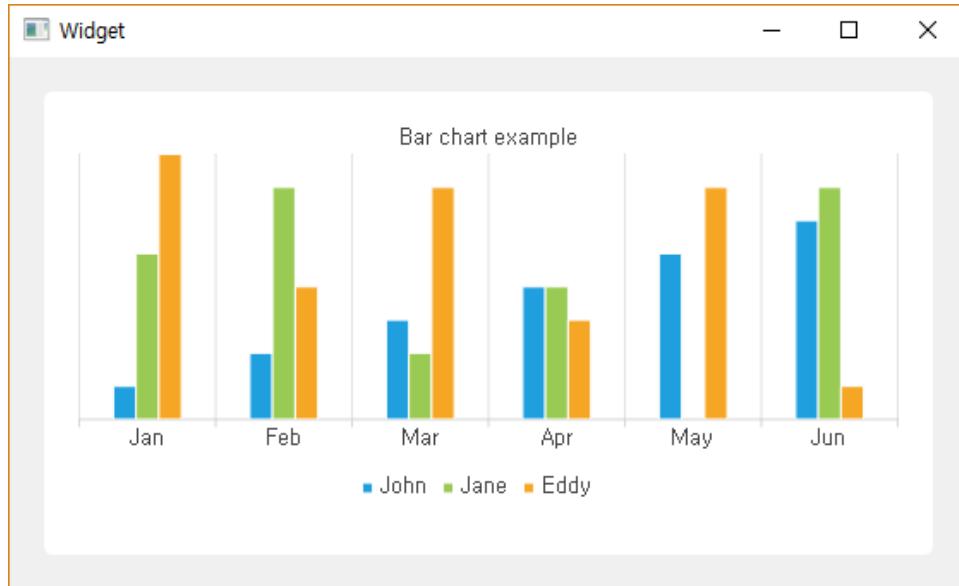
setLayout(hLay);
}

...
```

You can use the Gradient by using the QLinearGradient class object as the first factor in the setBrush() member function of the QAreaSeries class.

- Bar chart

Bar chart provides the ability to display bar graphs on the X and Y axes using the QBarSeries class.



<FIGURE> Example screen

You can use the QBarSeries class to use the bar graph as shown in the figure above. You can also use the QBarSet class to add data to the QBarSeries class. The following is part of the example source code for widget.cpp. Example - Ch11 > 03_BarChart

```
...
#include <QHBoxLayout>
#include <QtCharts>
```

Jesus loves you.

```
#include <QGraphicsView>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    QBarSet *set1 = new QBarSet("John");
    QBarSet *set2 = new QBarSet("Jane");
    QBarSet *set3 = new QBarSet("Eddy");

    *set1 << 1 << 2 << 3 << 4 << 5 << 6;
    *set2 << 5 << 7 << 2 << 4 << 0 << 7;
    *set3 << 8 << 4 << 7 << 3 << 7 << 1;

    QBarSeries *series = new QBarSeries();
    series->append(set1);
    series->append(set2);
    series->append(set3);

    QStringList categories;
    categories << "Jan" << "Feb" << "Mar" << "Apr" << "May" << "Jun";

    QBarCategoryAxis *axis = new QBarCategoryAxis;
    axis->append(categories);

    QChart *chart = new QChart();
    chart->addSeries(series);
    chart->setTitle("Bar chart example");
    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);
}

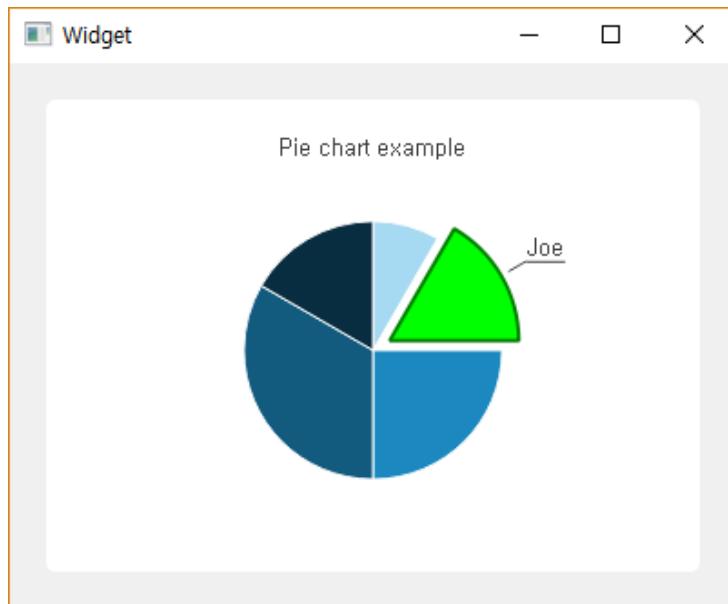
...
```

Jesus loves you.

QBarCategoryAxis can be used to use categories such as Jan, Feb, and Mar on the X and Y axes. By passing data as the first factor in the append() member function of the QBarCategoryAxis class, you can display the data in categories as shown in the figure above.

- Pie chart

The pie chart can graph the pie chart using the QPieSeries class. You can use the QPieSeries class to display data in a pie format, as shown in the following figure. You can use the append() member function to insert data. The first factor in the append() function can display the name of each data.



<FIGURE> Example screen

And the second factor is the size of the data. The following is an example source code.

Example - Ch11 > 04_PieChart

```
...
#include <QHBoxLayout>
#include <QtCharts>
#include <QGraphicsView>

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

Jesus loves you.

```
ui->setupUi(this);

QPieSeries *series = new QPieSeries();
series->append("Jane", 1);
series->append("Joe", 2);
series->append("Andy", 3);
series->append("Barbara", 4);
series->append("Axel", 2);

QPieSlice *slice = series->slices().at(1);

slice->setExploded();
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 example");

QChartView *chartView = new QChartView(chart);
chartView->setRenderHint(QPainter::Antialiasing);

QHBoxLayout *hLay = new QHBoxLayout();
hLay->addWidget(chartView);
setLayout(hLay);
}

...
```

12. Qt Data Visualization

Qt Data Visualization module provides visualization of data based on 3D. Data can be visualized in 3D, such as bar graphs, variance graphs, etc. Because Qt Data Visualization module uses hardware acceleration based on OpenGL, rendering speed is fast.



<FIGURE> Qt Data Visualization example screen

The Qt Data Visualization module can visualize 3D data with bar graphs, meter graphs, and surface graphs (e.g., sea level display on land and sea level) in 3D format.

The Qt Data Visualization module uses the concept of Camera to provide a function for users to move up/down/left/right the direction of the graph, zoom in and visualize specific data by clicking the mouse.

For example, it provides functions such as changing the color of a particular legend's data. To use the Qt Data Visualization module, you must add the following to the project file.

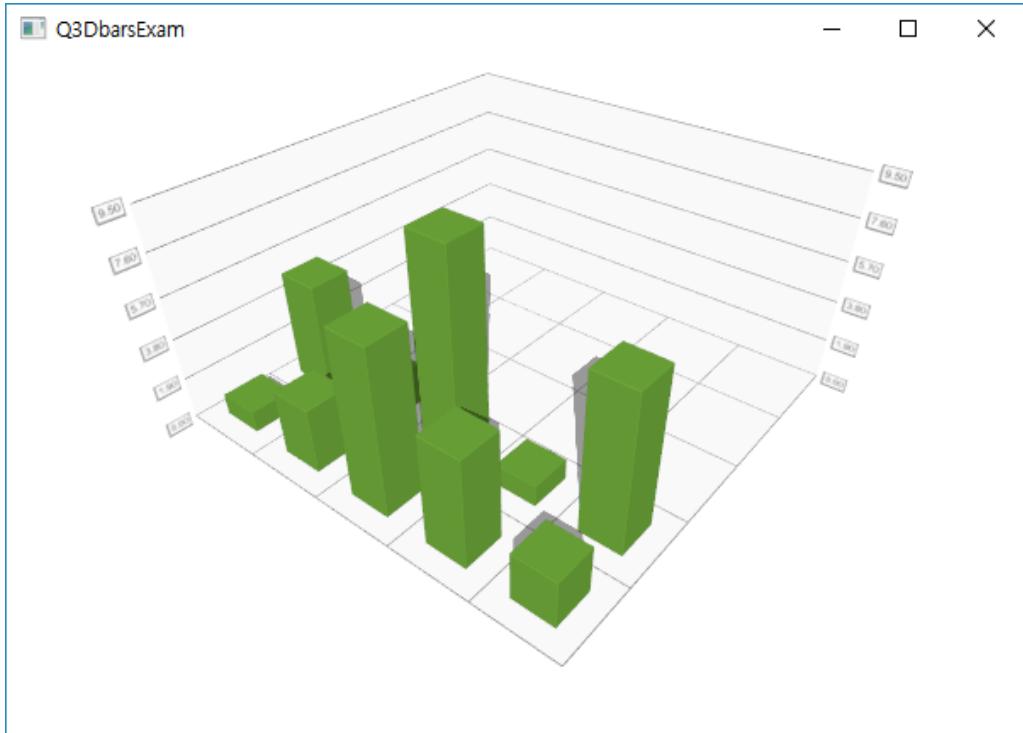
```
QT += datavisualization
```

Let's take a look at the bar graph, the Scatter graph, and the Surface graph provided by the Qt Data Visualization module in real-world examples.

Jesus loves you.

- Bar 3D example

The Q3Dbar class allows you to display a 3D bar graph. The Q3Dbar class is free to zoom in and out of the screen. The following is an example run screen.



<FIGURE> Example screen

The Q3DBars class can use the `rowAxis()->setRange()` member function to set the range of rows (Row). The `columnAxis()->setRange()` member function can be used to set the range of the column. The following example is Bar 3D Implementation Source Code.

Example - Ch12 > 01_BarsExample

```
#include <QApplication>
#include <QtDataVisualization>

using namespace QtDataVisualization;

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

    QBarDataRow *data1 = new QBarDataRow;
    *data1 << 1.0f << 3.0f << 7.5f << 5.0f << 2.2f;
```

Jesus loves you.

```
QBarDataRow *data2 = new QBarDataRow;
*data2 << 5.0f << 2.0f << 9.5f << 1.0f << 7.2f;

QBar3DSeries *series = new QBar3DSeries;
series->dataProxy()->addRow(data1);
series->dataProxy()->addRow(data2);

Q3DBars *bars = new Q3DBars;
bars->setFlags(bars->flags() ^ Qt::FramelessWindowHint);

bars->scene()->activeCamera()->setCameraPreset(
    Q3DCamera::CameraPresetFront);

bars->rowAxis()->setRange(0,4);
bars->columnAxis()->setRange(0,4);

bars->addSeries(series);
bars->setFloorLevel(0);

bars->setGeometry(50,50,600,400);
bars->show();

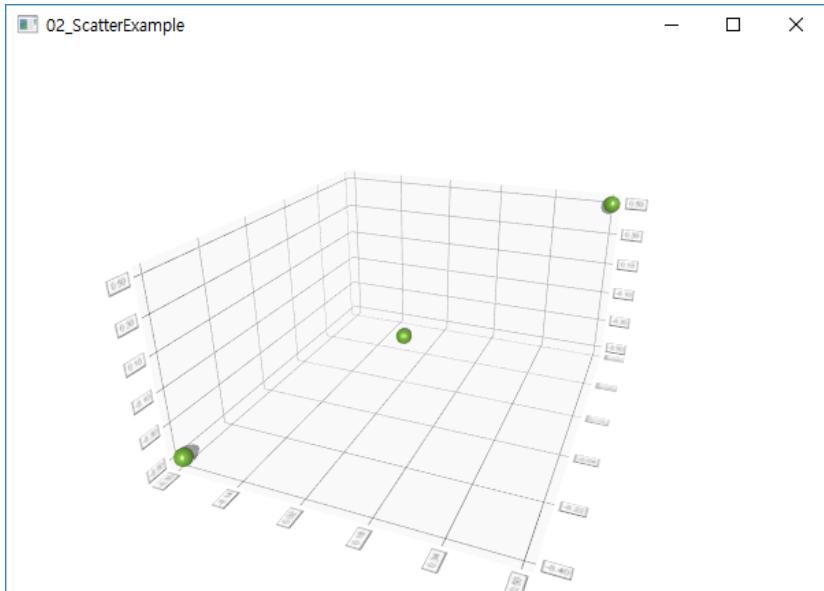
return a.exec();
}
```

You can use the QBarDataRow class and QBar3DSeries to add data to the 3D Bar. You must add an object from the QBarDataRow class to which you have added data. You can then pass the QBar3DSeries class object over to the addSeries() member function of the Q3DBars class.

- Scatter 3D example

Graph displays the specific point on the coordinates, x y scatter graph kind of form, such as distribution. Qt provides a Q3DScatter class to represent the Scatter graph in 3D.

Jesus loves you.



<FIGURE> 예제 실행 화면

You can add data to the Q3DScatter class using the QSCatter3DSeries and the QScatterdataArray class.

Example - Ch12 > 02_ScatterExample

```
#include <QApplication>
#include <QtDataVisualization>
using namespace QtDataVisualization;

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

    QScatter3DSeries *series = new QScatter3DSeries;
    QScatterdataArray data;
    data << QVector3D(0.5f, 0.5f, 0.5f) << QVector3D(-0.3f, -0.5f, -0.4f)
        << QVector3D(0.0f, -0.3f, 0.2f);

    Q3DScatter *scatter = new Q3DScatter;
    scatter->setFlags(scatter->flags() ^ Qt::FramelessWindowHint);
    scatter->scene()->activeCamera()->setCameraPreset(
        Q3DCamera::CameraPresetFront);

    series->dataProxy()->addItems(data);
    scatter->addSeries(series);
```

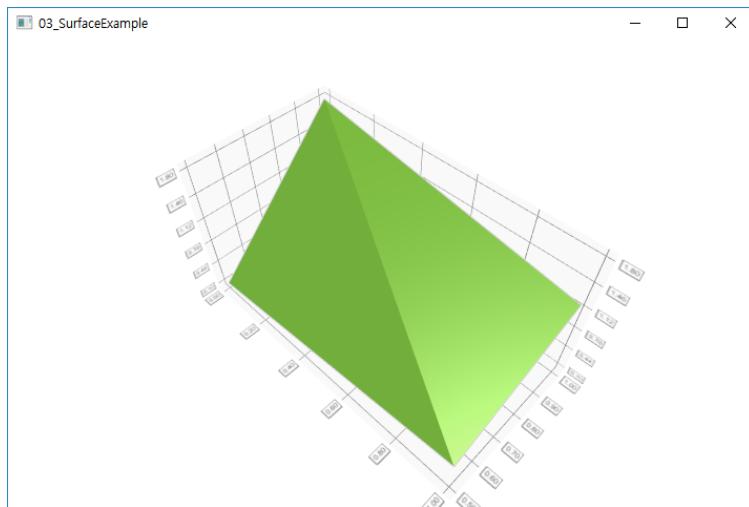
Jesus loves you.

```
scatter->setGeometry(50,50,600,400);
scatter->show();

return a.exec();
}
```

- Surface 3D example

The Surface graph can display sections on the X and Y axes. For example, it can be used to indicate the surface of land.



<FIGURE> Example screen

A Q3DSurface graph can be used to display a graph in the form of Surface. In addition, QSurfacedataArray and QSurfaceDataRow classes are provided to add data.

Example - Ch12 > 03_SurfaceExample

```
#include <QApplication>
#include <QtDataVisualization>

using namespace QtDataVisualization;

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

    Q3DSurface surface;
    surface.setFlags(surface.flags() ^ Qt::FramelessWindowHint);
```

Jesus loves you.

```
QSurfacedataArray *data = new QSurfacedataArray;
QSurfaceDataRow *dataRow1 = new QSurfaceDataRow;
QSurfaceDataRow *dataRow2 = new QSurfaceDataRow;

*dataRow1 << QVector3D(0.0f, 0.1f, 0.5f) << QVector3D(1.0f, 0.5f, 0.5f);
*dataRow2 << QVector3D(0.0f, 1.8f, 1.0f) << QVector3D(1.0f, 1.2f, 1.0f);
*data << dataRow1 << dataRow2;

QSurface3DSeries *series = new QSurface3DSeries;
series->dataProxy()->resetArray(data);
surface.addSeries(series);
surface.show();
return a.exec();
}
```

- Example of Displaying Contour 3D Surface with Contour Extraction Image

This example is an example to display 3D Surface using images taken for contour extraction as shown in the figure below.

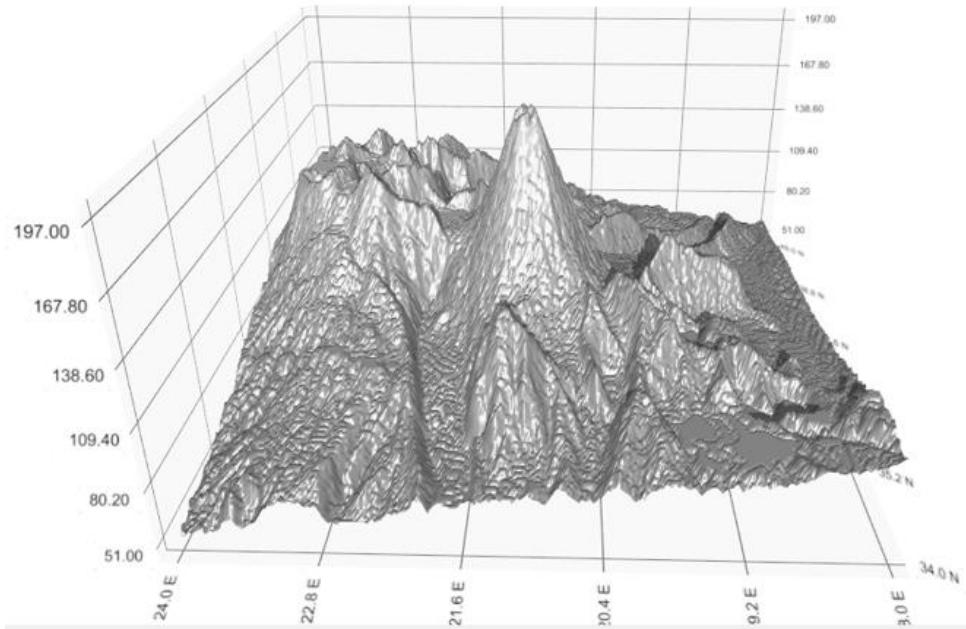


<FIGURE> Image for contour extraction

The above figure is an image taken for contour extraction. The higher the white, the higher the height. In other words, high height means higher altitude. Let's use Q3DSurface to visualize these photos on a 3D graph. The following figure shows an

Jesus loves you.

example run screen.



<FIGURE> Example screen

Load the original image into the QImage Class object as shown in the figure above. Then hand over the QImage class object to the QHeightMapSurfaceDataProxy class object.

This class provides the ability to convert the colors of an image into X, Y, and Z axes in order to visualize the image as Surface. This class provides the ability to extract RGB data to convert QImage class data into visualization data and convert it to values of X, Y, and Z. And passing the data stored in the QHeightMapSurfaceDataProxy class object as a factor to QSurface3DSeries enables Q3DSurface to convert it into available data.

Passing an object from the QSurface3DSeries class as the first factor in the Q3DSurface class' addSeries() member function allows you to display 3D as shown in the figure above. The following is an example source code.

Example - Ch12 > 04_ImageHeightMap

```
#include <QApplication>
#include <QtDataVisualization>
using namespace QtDataVisualization;

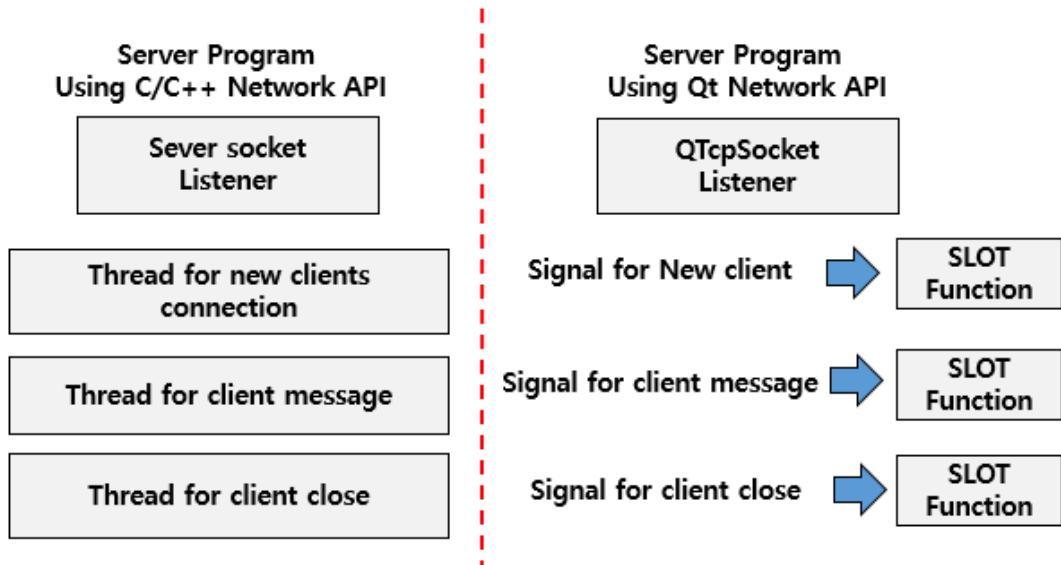
int main(int argc, char *argv[])
```

Jesus loves you.

```
{  
    QApplication a(argc, argv);  
  
    Q3DSurface *surface = new Q3DSurface;  
    surface->setFlags(surface->flags() ^ Qt::FramelessWindowHint);  
  
    surface->scene()->activeCamera()->setCameraPreset(  
        Q3DCamera::CameraPresetFront);  
  
    QImage heightMapImage(":/mountain.png");  
    QHeightMapSurfaceDataProxy *heightMapProxy;  
    heightMapProxy = new QHeightMapSurfaceDataProxy(heightMapImage);  
  
    QSurface3DSeries *series = new QSurface3DSeries(heightMapProxy);  
    series->setItemLabelFormat(QStringLiteral("@xLabel, @zLabel"): @yLabel));  
  
    heightMapProxy->setValueRanges(34.0f, 40.0f, 18.0f, 24.0f);  
  
    surface->axisX()->setLabelFormat("0.1f N");  
    surface->axisZ()->setLabelFormat("0.1f E");  
    surface->axisX()->setRange(34.0f, 40.0f);  
    surface->axisY()->setAutoAdjustRange(true);  
    surface->axisZ()->setRange(18.0f, 24.0f);  
  
    surface->axisX()->setTitle(QStringLiteral("Latitude"));  
    surface->axisY()->setTitle(QStringLiteral("Height"));  
    surface->axisZ()->setTitle(QStringLiteral("Longitude"));  
  
    surface->addSeries(series);  
    surface->setGeometry(50,50,600,400);  
    surface->show();  
  
    return a.exec();  
}
```

13. Network programming

Qt can develop network-based client or server software faster than using standard network libraries. Because Qt uses the concepts of Signal and Slot in implementing network programming, it is easy to deploy network applications. The following figure compares the method of development using the standard network API and the method of development using the network module provided by Qt.



<FIGURE> Comparison of two methods

From the above figure, outer is implemented using standard network API. This approach requires a thread that creates a socket riser for the server program and continues to check if new clients are accessing it. And you need a thread that checks for messages sent by a particular client and passes them on to the client you are logged on to. And you need a thread to check if the client you are connected to has shut down.

For example, three threads are cited as examples, but actual server programs require more Thread implementations than this. The more threads are used, the more complex the program source code is, the more difficult it is to maintain.

However, suppose you use a network module provided by Qt to develop a server program. While using standard network APIs to handle new connections requires a thread

Jesus loves you.

to handle new clients, Qt can connect Signal and Slot as connect() function to handle new clients. For example, when a new client connection signal occurs, you can connect as a Slot function. The following example is an example of a network module provided by the source code Qt implemented with Signal and Slot.

```
...
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\r\n");
}
...
```

Declares the objects of the QTcpSocket class and uses the connect() function to provide a new accessor event connected() signal. Associate this Signal with the slotConnected() function.

Therefore, when you connect Signal and Slot, the slotConnected() function is called when a new contact signal occurs. In other words, Qt can easily implement network programs because it requires only the implementation of a Slot function without the need for a thread implementation to handle new

The Qt network module provides various APIs for TCP/UDP protocol-based server or client implementation, APIs for network status, and APIs that support SSL(Secure Sockets Layer).

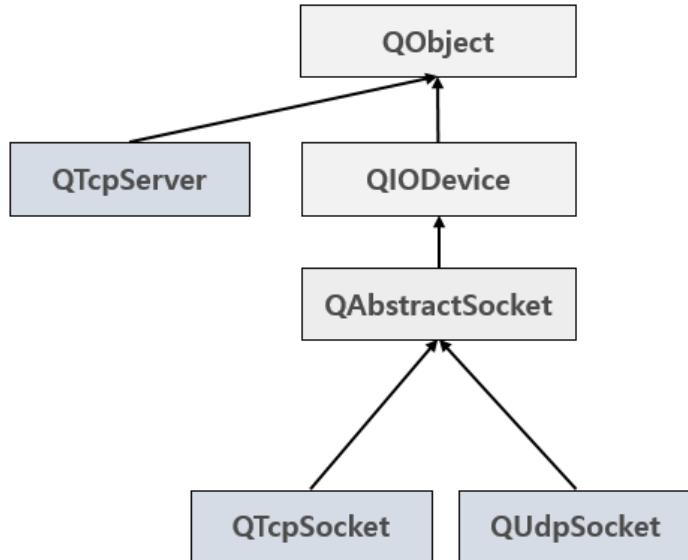
Provides QTcpSocket, QTcpServer, and QUdpSocket classes as the main classes for developing TCP/UDP protocol-based application servers and client applications.

- ✓ LOW Level network classes

Jesus loves you.

- QTcpSocket – Network class based on TCP protocol
- QTcpServer – Classes suitable for deploying servers based on TCP protocol
- QUdpSocket – Network class based on UDP protocol

LOW-level classes are suitable for developers to implement detailed network functions.



<FIGURE> LOW Lecel classes diagram

The **QTcpSocket** class and **QUdpSocket** class were inherited and implemented from the **QAbstracktSocket** class. It can save a lot of time if the **QAbstracktSocket** class is inherited and implemented in order to implement new network protocols that are not based on the TCP/UDP

QTcpServer is the same TCP protocol-based class as **QTcpSocket**. The difference is that the **QTcpServer** class provides convenient functionality for implementing server-based applications. To use the network modules provided by Qt, you must add the following to the project file(.pro)

```
QT += network
```

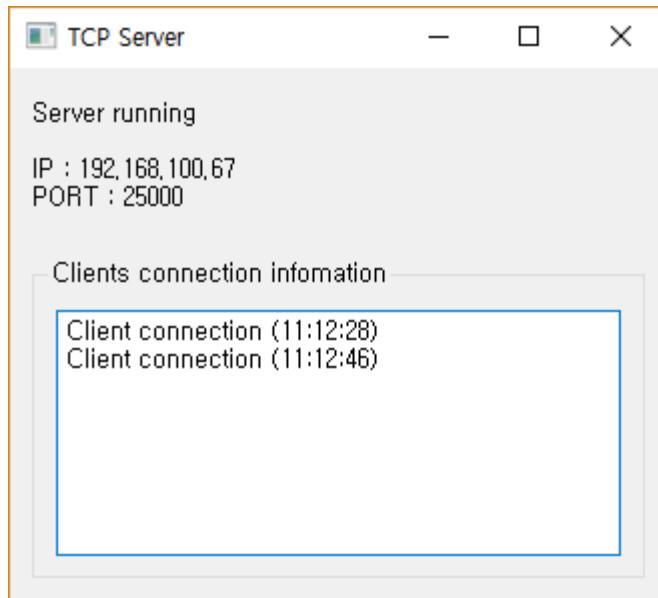
Jesus loves you.

- Implementing servers/clients using QTcpServer and QTcpSocket classes

In this example, let's use the QTcpServer class to implement server and client applications.

Create a project that inherits QWidget as shown in the figure below.

Server Example - Ch13 > 01_TcpServer



<FIGURE> Server example screen

Place the QLabel widget at the top of the GUI as shown in the figure above, and in the group box at the bottom place a QTextEdit that can output time accessed by the client as text output, and write the header source code for the Widget class as follows.

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

class QTcpServer;
class QNetworkSession;

namespace Ui { class Widget; }

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

Jesus loves you.

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

private:
    Ui::Widget *ui;
    void initialize();
    QTcpServer *tcpServer;

private slots:
    void newConnection();
};

#endif // WIDGET_H
```

위의 소스코드에서 initialize() 함수는 QTcpServer 클래스의 오브젝트를 초기화를 위한 함수이며 newConnection() Slot 함수는 클라이언트가 접속 Signal 이 발생하면 호출되는 함수이다. 다음은 widget.cpp 소스코드이다.

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

#include <QtWidgets>
#include <QDebug>

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

    initialize();
}

void Widget::initialize()
{
    QHostAddress hostAddress;
    QList<QHostAddress> ipAddressesList = QNetworkInterface::allAddresses();

    // Use non localhost(127.0.0.1)
    for (int i = 0; i < ipAddressesList.size(); ++i) {
        if (ipAddressesList.at(i) != QHostAddress::LocalHost &&
            ipAddressesList.at(i).toIPv4Address()) {
            hostAddress = ipAddressesList.at(i);
            break;
    }
}
```

Jesus loves you.

```
    }

}

if (hostAddress.toString().isEmpty())
    hostAddress = QHostAddress(QHostAddress::LocalHost);

tcpServer = new QTcpServer(this);
if (!tcpServer->listen(hostAddress, 25000)) {
    QMessageBox::critical(this, tr("TCP Server"),
        tr("It can not server, error message : %1.")
        .arg(tcpServer->errorString()));

    close();
    return;
}

ui->labelStatus->setText(tr("Server running \n\n"
    "IP : %1\n"
    "PORT : %2\n")
    .arg(hostAddress.toString())
    .arg(tcpServer->serverPort()));

connect(tcpServer, SIGNAL(newConnection()), this, SLOT(newConnection()));

ui->connMsgEdit->clear();
}

void Widget::newConnection()
{
    QTcpSocket *clientConnection = tcpServer->nextPendingConnection();
    connect(clientConnection, SIGNAL(disconnected()),
            clientConnection, SLOT(deleteLater()));

    QString currTime = QTime::currentTime().toString("hh:mm:ss");
    QString text = QString("Client connection (%1)").arg(currTime);

    ui->connMsgEdit->append(text);
    QByteArray message = QByteArray("Hello Client ~ ");
    clientConnection->write(message);
    clientConnection->disconnectFromHost();
}

Widget::~Widget()
```

Jesus loves you.

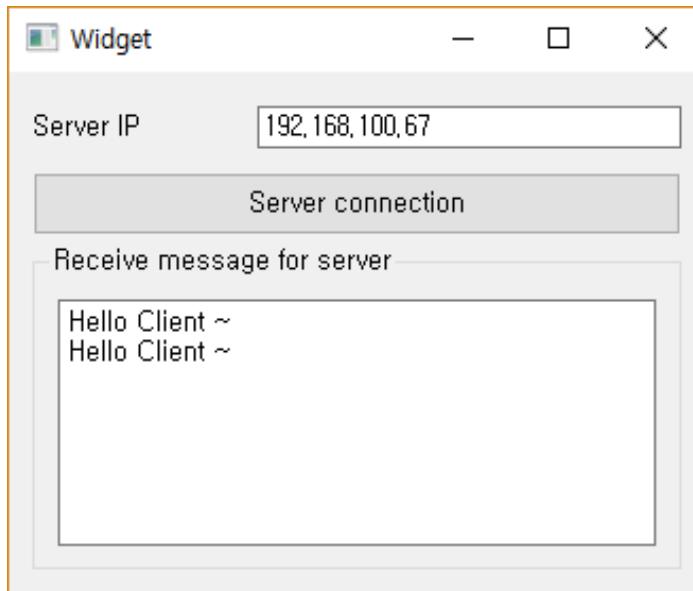
```
{  
    delete ui;  
}
```

In the initialize() function called from the constructor, the system obtains the IP of the server to which the client will connect. It also initializes the tcpServer object in the QTcpServer class and enables it to be queued for client access requests using the list() member function. The first factor in the list() function is the IP address and the second factor is the server port number.

The connect() function at the bottom of the initialize() function connected the signal and slot functions that were called when the client connected. Therefore, the newConnection() function is called when a new client is connected.

The newConnection() function outputs the time the client connects to the QTextEdit widget on the GUI and sends the message "Hello Client through " to the client that connects to it. The following is an example of a client: Create a project that inherits QWidget and place the widget on the GUI as shown in the following figure.

Client Example - Ch13 > 01_TcpClient



<FIGURE> Client Example screen

Place the QLineEdit widget so that you can enter the IP and PORT of the server you want to connect to, as shown in the figure above. Then place the [Server connection] button and the QTextEdit at the bottom.

Jesus loves you.

Click the [Server connection] button to try connecting with the server. When the connection with the server is complete, the QTextEdit widget outputs messages from the server. Create the widget.h header source code as shown in the example source code below.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QTcpSocket>

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
    void initialize();
    QTcpSocket *tcpSocket;

private slots:
    void connectButton();
    void readMessage();
    void disconnected();
};

#endif // WIDGET_H
```

The connectButton() Slot function is called when the Connect button is clicked. The readMessage() Slot function is a function that is invoked when a message is received from the server.

And the disconnected() Slot function is a function called when a signal terminates connection from the server occurs. The following is an example source code for

Jesus loves you.

widget.cpp.

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

#include <QHostAddress>
#include <QDebug>

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

    initialize();
}

void Widget::initialize()
{
    tcpSocket = new QTcpSocket(this);
    connect(tcpSocket, SIGNAL(readyRead()), this, SLOT(readMessage()));
    connect(tcpSocket, SIGNAL(disconnected()), this, SLOT(disconnected()));
}

void Widget::connectButton()
{
    QString serverip = ui->serverIP->text().trimmed();
    QHostAddress serverAddress(serverip);
    tcpSocket->connectToHost(serverAddress, 25000);
}

void Widget::readMessage()
{
    if(tcpSocket->bytesAvailable() >= 0) {
        QByteArray readData = tcpSocket->readAll();
        ui->textEdit->append(readData);
    }
}

void Widget::disconnected()
{
    qDebug() << Q_FUNC_INFO << "Server disconnect. ";
}
```

Jesus loves you.

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

The initialize() function declares the tcpSocket object of the QTcpSocket class and associates the signal() that receives a message from the server with the readMessage() Slot function. Then, when you receive a connection termination signal with the server, you connect to the disconnected() Slot function. Therefore, when a message is received from the server, the readMessage() function is called and the disconnection() Slot function is called when the connection with the server is terminated.

In the readMessage() Slot function, the tcpSocket->bytesAvailable() function can get the number of bytes of messages sent by the server. And the readAll() member function provides the ability to read messages sent by the server.

- Comparison between synchronous and asynchronous implementation using QTcpSocket class

In the previous example, we implemented servers and clients. In the previous example, Signal / Slot was used to handle clients when they accessed the server because they were not accessible at any time. However, it may be necessary to implement network applications in a synchronous manner.

The motivational method described here refers to cases in which a particular order must be carried out. For example, if a client sends a message to a server, it might have to wait for a message to be received without any other processing. In other words, if a processing process has to wait for a response to be received upon request from the server, it should be implemented in a synchronous manner rather than in the asynchronous mode previously dealt with.

In this example, when connecting to theqt-dev.com web server, let's try to implement sync and non-sync approaches and compare the differences.

The following example does not include the GUI. Therefore, create a console-based project and create a class named QHttpSocket that inherits the QObject class as follows: The following example source code is the header file of QHttpSocket.

Jesus loves you.

Example - Ch13 > 02_QTcSocket_Sync

```
#ifndef QHTTPSOCKET_H
#define QHTTPSOCKET_H
#include <QObject>
#include <QTcpSocket>

class QHttpSocket : public QObject
{
    Q_OBJECT
public:
    explicit QHttpSocket(QObject *parent = 0);
    ~QHttpSocket();

    void httpConnect();

public slots:
    void httpDisconnected();

private:
    QTcpSocket *socket;

};

#endif // QHTTPSOCKET_H
```

The `httpConnect()` function provides the ability to connect to and send/receive messages to the `qt-dev.com` web server. The following example is the source code for the `QHttpSocket` class.

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

QHttpSocket::QHttpSocket(QObject *parent) : QObject(parent)
{
    socket = new QTcpSocket(this);
    connect(socket, SIGNAL(disconnected()), this, SLOT(httpDisconnected()));
}

QHttpSocket::~QHttpSocket()
{}
```

Jesus loves you.

```
void QHttpSocket::httpConnect()
{
    socket->connectToHost("qt-dev.com", 80);
    if(socket->waitForConnected(5000))
    {
        int writeBytes = socket->write("Hello server\r\n\r\n");
        socket->waitForBytesWritten(1000);
        qDebug() << "write bytes : " << writeBytes;

        socket->waitForReadyRead(3000);
        qDebug() << "Reading: " << socket->bytesAvailable();
        qDebug() << socket->readAll();
    }
    else
    {
        qDebug() << "Not connected!";
    }
}

void QHttpSocket::httpDisconnected()
{
    qDebug() << Q_FUNC_INFO;
}
```

Wait 5 seconds for the httpConnect() function to complete the connection to theqt-dev.com web server. Use the waitForConnected() function to wait for a 5 second connection. The unit of the waitForConnected() function factor is Milliscond.

When the connection is complete, connect to theqt-dev.com web server and use the write() function to send the message to theqt-dev.com web server. Then wait until the transmission is complete. You can use the waitForBytesWritten() function to wait for messages sent to the web server as a write() function.

In addition, the return value of the write() function returns the number of bytes sent. If an error occurs, return zero.

The waitForReadyRead() function waits until you receive a message from the server, and the factor of this function can specify the amount of time you wait. Therefore, wait for 3 seconds for messages received from the server. The following example source code is an example of a main.cpp source code that creates and performs a QHttpSocket class object.

Jesus loves you.

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

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

    QHttpSocket *httpSocket = new QHttpSocket();
    httpSocket->httpConnect();

    return a.exec();
}
```

So far we have covered examples of motivation. Let's try to implement it in the same way as this example. Try implementing the QHttpSocket class as shown below in sync. The following example source code is the header file of the QHttpSocket class.

Example - Ch13 > 02_QTcpSocket_Async

```
#ifndef QHTTPSOCKET_H
#define QHTTPSOCKET_H

#include <QObject>
#include <QTcpSocket>

class QHttpSocket : public QObject
{
    Q_OBJECT

public:
    explicit QHttpSocket(QObject *parent = 0);
    ~QHttpSocket();
    void httpConnect();

public slots:
    void slotConnected();
    void slotDisconnected();
    void slotBytesWritten(qint64 bytes);
    void slotReadPendingDatagram();

private:
    QTcpSocket *socket;
};
```

Jesus loves you.

```
#endif // QHTTPSOCKET_H
```

The example above is the header source for the non-synchronous QHttpSocket class. The slotConnected() function is a Slot function that is called when a connection with a web server is complete. The slotDisconnected() function is a Slot function that terminates when a connection with a web server is terminated.

The slotBytesWritten() function is a Slot function that is called upon completion of the transfer to the server, and the slotReadPandingDatagram() function is a Slot function that is called when a message is received from the server. The following example source code is the implementation source code for the QHttpSocket class.

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

QHttpSocket::QHttpSocket(QObject *parent) : QObject(parent)
{
    socket = new QTcpSocket(this);

    connect(socket, SIGNAL(connected()),
            this, SLOT(slotConnected()));
    connect(socket, SIGNAL(disconnected()),
            this, SLOT(slotDisconnected()));
    connect(socket, SIGNAL(bytesWritten(qint64)),
            this, SLOT(slotBytesWritten(qint64)));
    connect(socket, SIGNAL(readyRead()),
            this, SLOT(slotReadPandingDatagram()));
}

QHttpSocket::~QHttpSocket()
{
}

void QHttpSocket::httpConnect()
{
    socket->connectToHost("qt-dev.com", 80);
    if(!socket->waitForConnected(3000))
    {
        qDebug() << "Socket Error : " << socket->errorString();
    }
}
```

Jesus loves you.

```
void QHttpSocket::slotConnected()
{
    qDebug("\n [%s] CONNECTED", Q_FUNC_INFO);
    socket->write("HEAD / HTTP/1.0\r\n\r\n\r\n\r\n\r\n");
}

void QHttpSocket::slotDisconnected()
{
    qDebug("\n [%s] DISCONNECTED", Q_FUNC_INFO);
}

void QHttpSocket::slotBytesWritten(qint64 bytes)
{
    qDebug("\n [%s] Bytes Written [size :%d]", Q_FUNC_INFO, bytes);
}

void QHttpSocket::slotReadPendingDatagram()
{
    qDebug() << socket->readAll();
}
```

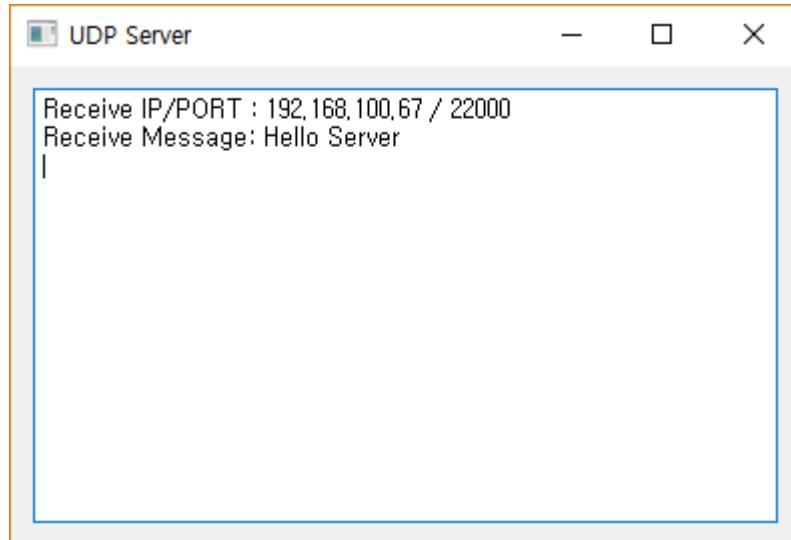
The creator declares the QTcpSocket class object and associates Signal with the Slot function when connecting to the server, terminating, sending messages, and receiving messages from the Web server. Running the httpConnect() function will attempt to connect to the Web server. As such, in this example, a non-sync method can be used using Signal and Slot.

- Implement servers and clients using QUdpSocket class

Because UDP protocol is non-reliable, servers and clients must connect before sending/receiving messages, such as TCP, but UDP can send/receive bilateral messages without having a connection. In other words, UDP can send or receive messages directly to the IP and PORT addresses it wants to send.

This example is protocol, Let's see the example to incoming and outgoing messages based on ratio plausibility udp. In the graphic below, the qtextedit, such as widget to see.

Jesus loves you.



<FIGURE> UDP Server example screen

In this example, receives the message from the client sends a message client's ip, port and example that prints a message. The following example is the widget.h header source code. Server Example – Ch13 > 03_UDP_Server

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

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
    QUdpSocket *udpSocket;

private slots:
    void readPendingDatagram();
};


```

Jesus loves you.

```
#endif // WIDGET_H
```

The readPendingDatagram() Slot function is a Slot function that is called when a message is received from the client. Next is the widget.cpp source code.

```
#include "widget.h"
#include "ui_widget.h"
#include <QDebug>
#define SERVER_PORT 21000

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

    udpSocket = new QUdpSocket(this);
    udpSocket->bind(QHostAddress("192.168.100.67"), SERVER_PORT);
    connect(udpSocket, SIGNAL(readyRead()), this, SLOT(readPendingDatagram()));
}

void Widget::readPendingDatagram()
{
    QByteArray buffer;
    buffer.resize(udpSocket->pendingDatagramSize());

    QHostAddress sender;
    quint16 senderPort;
    udpSocket->readDatagram(buffer.data(), buffer.size(),
                           &sender, &senderPort);

    QString msg = QString("Receive IP/PORT : %1 / %2 <br>"
                          "Receive Message: %3 <br>")
                  .arg(sender.toString())
                  .arg(senderPort)
                  .arg(buffer.data());
    ui->textEdit->append(msg);

    QByteArray writeData;
    writeData.append("SERVER : Hello UDP Client.~ ");
    udpSocket->writeDatagram(writeData, sender, senderPort);
}

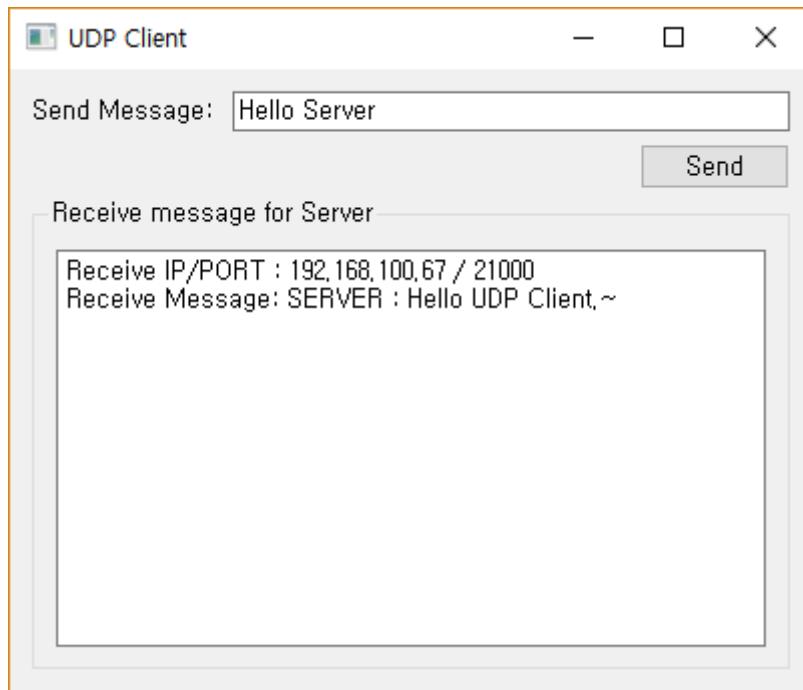
Widget::~Widget()
```

Jesus loves you.

```
{  
    delete ui;  
}
```

In the `readPendingDatagram()` Slot function, the `sendingDatagramSize()` member function returns the size of the incoming message. It then outputs the received messages to the `QTextEdit` widget and sends the messages back to the received. The following example is an example of a client implementation.

Client Example - Ch13 > 03_UDP_Client



<FIGURE> UDP Client example screen

Place the widget as shown in the figure above. Then write the source code of the `widget.h` header file as shown below.

```
#ifndef WIDGET_H  
#define WIDGET_H  
  
#include <QWidget>  
#include <QUdpSocket>  
  
namespace Ui {  
class Widget;
```

Jesus loves you.

```
{}

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
    QUdpSocket *udpSocket;

public slots:
    void sendButton();
    void readPendingDatagram();
};

#endif // WIDGET_H
```

As shown in the example above, the sendButton() Slot function is called by clicking the [Send] button. And the readPendingDatagram() Slot function is called upon receipt of the message. The following example is the widget.cpp source code.

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

#define SERVER_PORT 21000
#define CLIENT_PORT 22000

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

    udpSocket = new QUdpSocket(this);
    udpSocket->bind(QHostAddress("192.168.100.110"), CLIENT_PORT);
    connect(udpSocket, SIGNAL(readyRead()), this, SLOT(readPendingDatagram()));
}
```

Jesus loves you.

```
void Widget::sendButton()
{
    QByteArray msg;
    msg = ui->sendMsg->text().toLocal8Bit();

    QHostAddress sender("192.168.100.110");
    udpSocket->writeDatagram(msg, sender, SERVER_PORT);
}

void Widget::readPendingDatagram()
{
    QByteArray buffer;
    buffer.resize(udpSocket->pendingDatagramSize());

    QHostAddress sender;
    quint16 senderPort;
    udpSocket->readDatagram(buffer.data(), buffer.size(), &sender, &senderPort);

    QString msg = QString("Receive IP/PORT : %1 / %2 <br>"
                           "Receive Message: %3 <br>")
                           .arg(sender.toString())
                           .arg(senderPort)
                           .arg(buffer.data());
    ui->textEdit->append(msg);
}

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

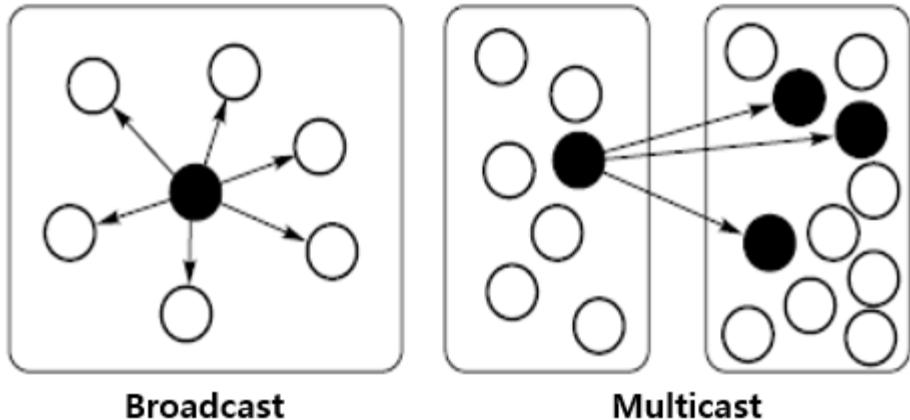
The `readPendingDatagram()` Slot function outputs messages received in the `QTextEdit` widget when they are received from the server.

- Implementing broadcasts using `QUdpSocket` class

Broadcasts and multicasts can be distinguished from transmitters and receivers using UDP-based protocols over computer networks.

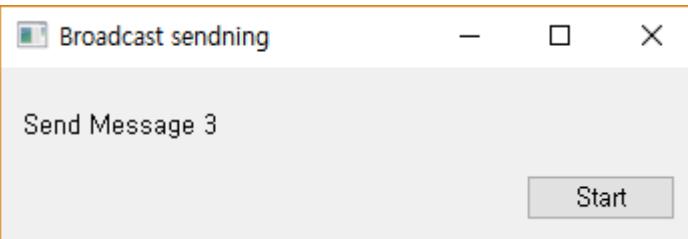
The Broadcast method allows one sender to send messages to all receivers, and the Multicast method allows one or more users to send data to more than one sender.

Jesus loves you.



<FIGURE> Broadcast and Multicast

Create a project that inherits the QWidget class and place the GUI as shown in the following figure. Example - Ch13 > 04_Broadcast_Sender



<FIGURE> Broadcast Sender example screen

The following example source code is widget.h.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QUdpSocket>
#include <QTimer>

namespace Ui { class Widget; }

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

Jesus loves you.

```
private:  
    Ui::Widget *ui;  
    QUDpSocket *udpSocket;  
    QTimer *timer;  
    int msgNumber;  
  
private slots:  
    void startButton();  
    void broadcastSend();  
};  
  
#endif // WIDGET_H
```

Timer objects in the QTimer class repeat the specified time to send the broadcast message. Next is the widget.cpp source code.

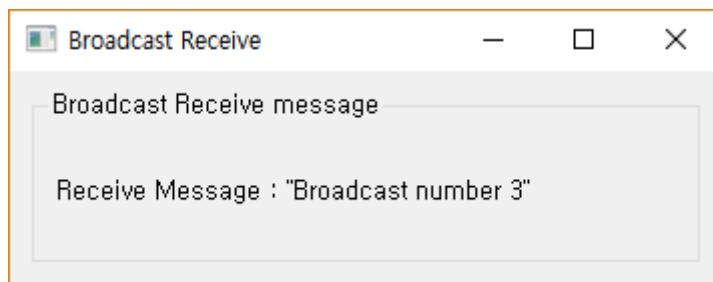
```
#include "widget.h"  
#include "ui_widget.h"  
  
Widget::Widget(QWidget *parent)  
    : QWidget(parent), ui(new Ui::Widget)  
{  
    ui->setupUi(this);  
  
    timer = new QTimer(this);  
    udpSocket = new QUdpSocket(this);  
    msgNumber = 1;  
  
    connect(ui->startButton, SIGNAL(clicked()),  
            this, SLOT(startButton()));  
  
    connect(timer, SIGNAL(timeout()),  
            this, SLOT(broadcastSend()));  
}  
  
void Widget::startButton()  
{  
    if(!timer->isActive())  
        timer->start(1000);  
}  
  
void Widget::broadcastSend()  
{
```

Jesus loves you.

```
ui->sendMsg->setText(QString("Send Message %1").arg(msgNumber));  
  
QByteArray datagram = "Broadcast number "  
                     + QByteArray::number(msgNumber);  
udpSocket->writeDatagram(datagram.data(), datagram.size(),  
                           QHostAddress::Broadcast, 35000);  
msgNumber++;  
}  
  
Widget::~Widget()  
{  
    delete ui;  
}
```

Click the [Start] button to periodically execute timer objects in QTimer class once a second. Then execute the broadcastSend() function every second. The third factor of the writeDatagram() function in the broadcastSend() function can be used when transferring data in a broadcast manner.

QHostAddress::Using the Broadcast value as the third factor sends a message to all computers within the network group (255.255.255.255). The following example is receiving a broadcast message.



<FIGURE> Broadcast Receiver example screen

When a broadcast message is received, as shown in the figure above, it outputs the message on the widget. Next is widget.cpp.

Example - Ch13 > 04_Broadcast_Receiver

```
#include "widget.h"  
#include "ui_widget.h"  
  
Widget::Widget(QWidget *parent) :  
    QWidget(parent), ui(new Ui::Widget)
```

Jesus loves you.

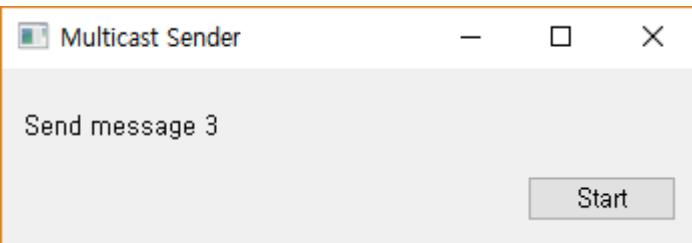
```
{  
    ui->setupUi(this);  
  
    udpSocket = new QUdpSocket(this);  
    udpSocket->bind(35000, QUdpSocket::ShareAddress);  
  
    connect(udpSocket, SIGNAL(readyRead()),  
            this, SLOT(readDatagrams()));  
}  
  
void Widget::readDatagrams()  
{  
    while (udpSocket->hasPendingDatagrams())  
    {  
        QByteArray datagram;  
        datagram.resize(udpSocket->pendingDatagramSize());  
  
        udpSocket->readDatagram(datagram.data(), datagram.size());  
  
        ui->recvMsg->setText(tr("Receive Message : \"%1\"")  
                             .arg(datagram.data()));  
    }  
}  
  
Widget::~Widget()  
{  
    delete ui;  
}
```

- Multi-cast implementation using QUdpSocket class

In this example, let's learn how to implement an application using a multi-cast method. The multi-cast method is where more than one user sends data to more than one sender. Create a project that inherits the QWidget class and place the widget in the GUI as shown in the following figure.

Example - Ch13 > 05_Multicast_Sender

Jesus loves you.



<FIGURE> Multicast sender example screen

The following example source code is the widget.h source code.

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

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();
private:
    Ui::Widget *ui;
    QUdpSocket udpSocket4;
    QUdpSocket udpSocket6;
    QHostAddress groupAddress4;
    QHostAddress groupAddress6;

    QTimer      *timer;
    int         msgNumber;

private slots:
    void startButton();
    void broadcastSend();
};

#endif // WIDGET_H
```

The timer object in the QTimer class repeats the specified time to send a broadcast message. The following is the widget.cpp source code.

Jesus loves you.

```
#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("Send message %1").arg(msgNumber));

    QByteArray datagram = "Multicast number "
                          + QByteArray::number(msgNumber);

    udpSocket4.writeDatagram(datagram, groupAddress4, 45000);
    if (udpSocket6.state() == QAbstractSocket::BoundState)
        udpSocket6.writeDatagram(datagram, groupAddress6, 45000);
}
```

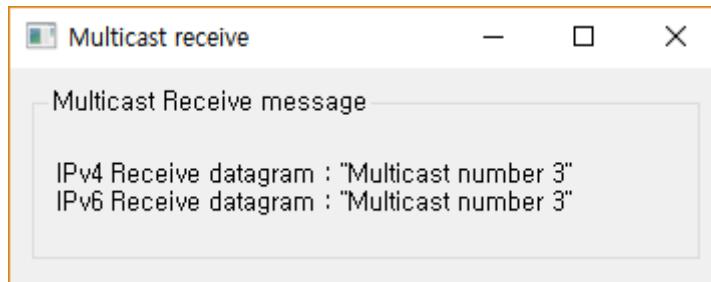
Jesus loves you.

```
    msgNumber++;
}

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

When you click the [Start] button, the timer object in the QTimer class is executed periodically once a second. Then, execute the multicastSend () function every second.

In the multicastSend () function, the first factor of the writeDatagram() function is the message to be sent, the second message is the specific group on the network, and the third factor is the PORT number to be sent. The following example is the receipt of a multicast message.



<FIGURE> Multicast receive example screen

When you receive a multicasting message, as shown in the figure above, you output the message on the widget. Next is the widget.cpp source code.

Example - Ch13 > 05_Multicast_Receiver

```
#include "widget.h"
#include "ui_widget.h"
#include <QtWidgets>
#include <QtNetwork>

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

    groupAddress4 = QHostAddress(QStringLiteral("239.255.43.21"));
    groupAddress6 = QHostAddress(QStringLiteral("ff12::2115"));
```

Jesus loves you.

```
udpSocket4.bind(QHostAddress::AnyIPv4,
                 45000,
                 QUpdSocket::ShareAddress);
udpSocket4.joinMulticastGroup(groupAddress4);

if (!udpSocket6.bind(QHostAddress::AnyIPv6, 45000,
                     QUpdSocket::ShareAddress) ||
    !udpSocket6.joinMulticastGroup(groupAddress6))
    qDebug() << Q_FUNC_INFO << "Only IPv4 Multicast";

// Old Signal Slot style
connect(&udpSocket4, SIGNAL(readyRead()),
        this,           SLOT(readDatagrams()));

// New Signal Slot style
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 Receive datagram : \"%1\"")
                               .arg(datagram.constData()));
    }

    // using QUpdSocket::receiveDatagram (API since Qt 5.8)
    while (udpSocket6.hasPendingDatograms()) {
        QNetworkDatagram dgram = udpSocket6.receiveDatagram();
        ui->recvMsg->setText(ui->recvMsg->text() +
                               tr("\nIPv6 Receive datagram : \"%1\"")
                               .arg(dgram.data().constData()));
    }
}

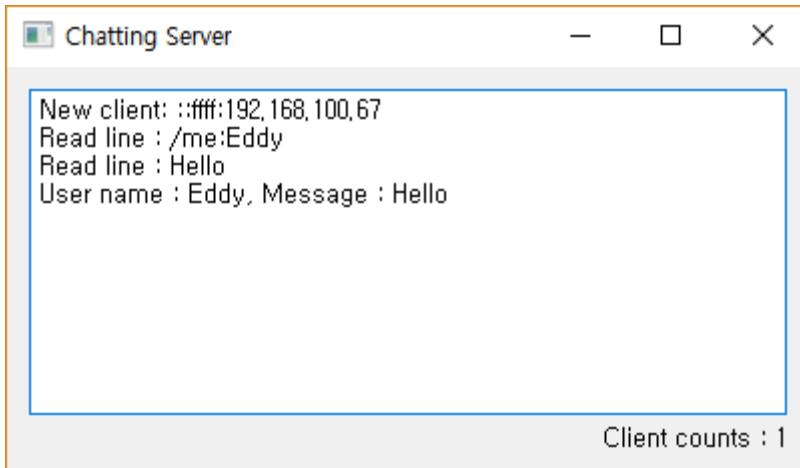
Widget::~Widget()
```

Jesus loves you.

```
{  
    delete ui;  
}
```

- Implement chat servers and clients

In this example, let's try implementing a chat server and a chat client. First, let's look at the chat server implementation. The GUI displays the user information accessed and the accesses connected to the server, as shown below. Example - Ch13 > 06_ChatServer



<FIGURE> Chatting Server example screen

Place the widget on the GUI as shown in the figure above. The centrally placed widget is the QTextEdit widget, and the widget that displays the number of contacts at the bottom is the QLabel widget. When you create a project, you create a Widget class with the QWidget class as the Base. It then creates the ChatServer class that inherits the QTcpServer class. The following is the header file of the ChatServer class.

```
#ifndef CHATSERVER_H  
#define CHATSERVER_H  
  
#include <QtNetwork/QTcpServer>  
#include <QtNetwork/QTcpSocket>  
  
class ChatServer : public QTcpServer  
{  
    Q_OBJECT
```

Jesus loves you.

```
public:  
    ChatServer(QObject *parent=0);  
  
private slots:  
    void readyRead();  
    void disconnected();  
    void sendUserList();  
  
signals:  
    void clients_signal(int users);  
    void message_signal(QString msg);  
  
protected:  
    void incomingConnection(int socketfd);  
  
private:  
    QSet<QTcpSocket*> clients;  
    QMap<QTcpSocket*,QString> users;  
};  
  
#endif // CHATSERVER_H
```

The incomingConnection() function, declared by the protected access restrictor in the header above, is a function called when a new client is connected. clients_signals() occur in this function. The factors in this signal hand over the number of users currently accessing the server as a factor. When you receive a message for the client, you connect it to the ReadyRead() function.

Therefore, the ReadyRead() function is called when the client sends the message. In addition, the disconnected() signal generates a signal when the client connection is terminated and calls the disconnected() function associated with this signal. The following example source code is the implementation source code of the ChatServer class.

```
#include <QtWidgets>  
#include <QRegExp>  
#include "chatserver.h"  
#include <QDebug>  
  
ChatServer::ChatServer(QObject *parent)
```

Jesus loves you.

```
: QTcpServer(parent)
{
}

void ChatServer::incomingConnection(int socketfd)
{
    qDebug() << Q_FUNC_INFO;

    QTcpSocket *client = new QTcpSocket(this);
    client->setSocketDescriptor(socketfd);
    clients.insert(client);

    emit clients_signal(clients.count());

    QString str;
    str = QString("New client: %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);

        QRegExp meRegex("^/me:(.*)$");

        if(meRegex.indexIn(line) != -1)
    {
```

Jesus loves you.

```
QString user = meRegex.cap(1);
users[client] = user;
foreach(QTcpSocket *client, clients)
{
    client->write(QString("Server : %1 connection")
                  .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).arg(message);
    emit message_signal(str);

    foreach(QTcpSocket *otherClient, clients)
        otherClient->write(QString(user+": "+message+"\n")
                           .toUtf8());
}
}
}

void ChatServer::disconnected()
{
    QTcpSocket *client = (QTcpSocket*)sender();

    QString str;
    str = QString("Connection close :: %1")
        .arg(client->peerAddress().toString());

    emit message_signal(str);

    clients.remove(client);

    emit clients_signal(clients.count());

    QString user = users[client];
```

Jesus loves you.

```
users.remove(client);

sendUserList();
foreach(QTcpSocket *client, clients)
    client->write(QString("Server: %1 close").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());
}
```

Creating a new QTcpSocket in the incomingConnection() function is to create a socket object for a new client. Thus, objects from the QTcpSocket class generated by this function are stored in a client QMap container named users.

The readyRead() function is a Slot function that is called when a client that connects to the server sends a message. This function sends a message from the client to all clients that have access to the server.

The disconnected() function is a Slot function that is called when a client terminates a connection. This function removes objects from the QTcpSocket class of clients that have terminated access from a QMap container named users. The following is the source code of widget.h.

```
#ifndef WIDGET_H
#define WIDGET_H

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

namespace Ui { class Widget; }

class Widget : public QWidget
{
```

Jesus loves you.

```
Q_OBJECT
public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();

private:
    Ui::Widget *ui;
    ChatServer *server;

private slots:
    void slot_clients(int users);
    void slot_message(QString msg);
};

#endif // WIDGET_H
```

In the Widget class, the slot_clients() Slot function is a signal that occurs in the ChatServer class when a new client is connected or shut down, and sends the end client's access. And the slot_message() function is a Slot function called when a client sends a message or terminates a connection in the ChatServer class. The following example source is the widget.cpp source code.

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

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

    server = new ChatServer();
    connect(server, SIGNAL(clients_signal(int)),
            this, SLOT(slot_clients(int)));
    connect(server, SIGNAL(message_signal(QString)),
            this, SLOT(slot_message(QString)));

    server->listen(QHostAddress::Any, 35000);
}

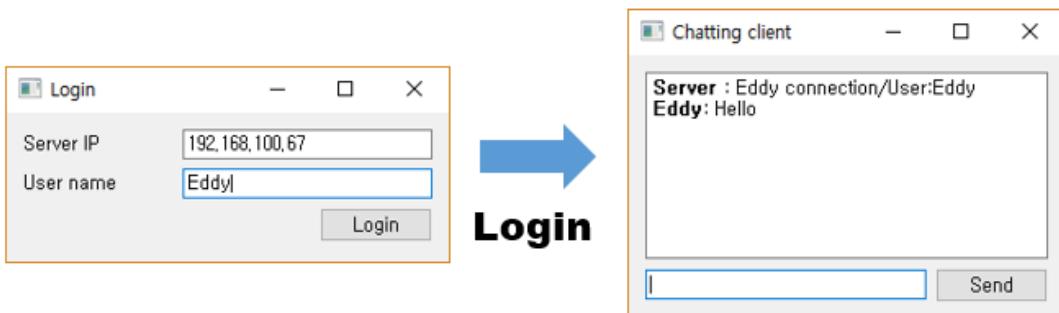
void Widget::slot_clients(int users)
{
    QString str = QString("전속자수 : %1").arg(users);
```

Jesus loves you.

```
ui->label->setText(str);  
}  
  
void Widget::slot_message(QString msg)  
{  
    ui->textEdit->append(msg);  
}  
  
Widget::~Widget()  
{  
    delete ui;  
}
```

Update the number of contacts on the GUI when the slot_clients() Slot function is called. The slot_message() Slot function outputs a mesh in the QTextEdit window. So far, I have found out about the chat server. Next, let's implement a chat server and a client that sends and receives messages.

The chat client consists of two screens. The first is the login screen as shown in the figure below. As you can see in the Login widget, clicking the Login button completes access to the chat server with the server IP address, the login widget is hidden, and the chat client widget is activated (Show).



<FIGURE> Chatting client example screen

The QTextEdit widget, located in the middle of the chat client widget, outputs messages from the server. QLineEdit at the bottom enters the message to be sent to the server. Then click the Send button to send the message to the chat server. The following example is the source code widget.h. Example - Ch13 > 06_ChatClient

```
#ifndef WIDGET_H  
#define WIDGET_H
```

Jesus loves you.

```
#include <QWidget>
#include <QtNetwork/QTcpSocket>
#include "loginwidget.h"

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
    LoginWidget *loginWidget;

    QTcpSocket *socket;
    QString     ipAddr;
    QString     userName;

private slots:
    void loginInfo(QString addr, QString name);
    void sayButton_clicked();
    void connected();
    void readyRead();
};

#endif // WIDGET_H
```

The loginInfo() Slot function is a Slot function called when you click the Login button in the LoginWidget class to generate a signal to communicate the server IP address and user name entered in the login window. The first factor is the server IP address and the second factor is the user name. The following example is the widget.cpp source code.

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

Jesus loves you.

```
Widget::Widget(QWidget *parent) :  
    QWidget(parent),  
    ui(new Ui::Widget)  
{  
    ui->setupUi(this);  
    loginWidget = new LoginWidget();  
    connect(loginWidget, SIGNAL(loginInfo(QString, QString)),  
            this,           SLOT(loginInfo(QString, QString)));  
    connect(ui->sayButton, &QPushButton::pressed,  
            this,           &Widget::sayButton_clicked);  
  
    loginWidget->show();  
  
    socket = new QTcpSocket(this);  
    connect(socket, SIGNAL(readyRead()),  
            this,           SLOT(readyRead()));  
    connect(socket, SIGNAL(connected()),  
            this,           SLOT(connected()));  
}  
  
void Widget::loginInfo(QString addr, QString name)  
{  
    qDebug() << Q_FUNC_INFO << addr << name;  
    ipAddr = addr;  
    userName = name;  
    socket->connectToHost(ipAddr, 35000);  
}  
  
void Widget::sayButton_clicked()  
{  
    QString message = ui->sayLineEdit->text().trimmed();  
  
    if(!message.isEmpty())  
    {  
        socket->write(message + "\n").toUtf8());  
    }  
  
    ui->sayLineEdit->clear();  
    ui->sayLineEdit->setFocus();  
}
```

Jesus loves you.

```
void Widget::connected()
{
    loginWidget->hide();
    this->window()->show();

    socket->write(QString("/me:" + userName + "\n").toUtf8());
}

void Widget::readyRead()
{
    while(socket->canReadLine())
    {
        QString line = QString::fromUtf8(socket->readLine()).trimmed();

        QRegExp messageRegex("^([:]+):(.*$)");

        if(messageRegex.indexIn(line) != -1)
        {
            QString user = messageRegex.cap(1);
            QString message = messageRegex.cap(2);

            ui->roomTextEdit->append("<b>" + user + "</b>: " + message);
        }
    }
}

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

The SayButton_clicked() Slot function is a function called by clicking the Send button. The Connected() Slot function is called when the connection with the chat server is complete. readyRead() is a Slot function that is called when a chat server sends a message. This Slot function outputs messages to the QTextEdit widget.

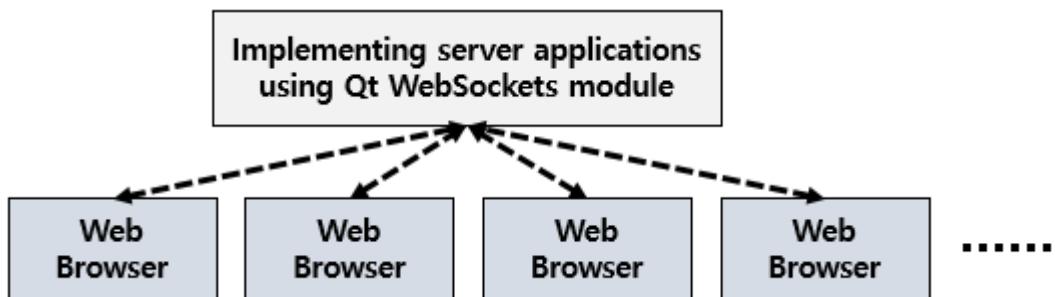
14. Qt WebSockets

Using a web browser, users can access portal sites (web servers) to get the information they want. When a user requests information from a Web browser to a Web server, TCP Connection is formed based on the HTTP protocol. And when the connection is complete, the web browser requests the desired information from the web server. When a Web server delivers the information that the user wants, it terminates the TCP Connection associated with the Web browser.

Because a Web server terminates TCP Connection when it delivers the requested information to the web browser, it must re-engage TCP Connection to obtain the information again.

To eliminate these shortcomings, webSockets can be used by web servers to update information in real time when certain portions of the page information currently being viewed in the web browser are updated without any refresh, such as when using QTcpSocket learned in the previous chapter.

The existing HTTP protocol terminates TCP Connection when the web server passes the desired information to the Web browser, but the WebSocket continues to connect without terminating TCP Connection. WebSockets are the same as the Asynchronous JavaScript and XML (AJAX) usage objectives.



<FIGURE> Web browser and server using webSockets

As shown in the figure above, the Qt WebSockets module makes it easy to implement applications that can communicate with the web browser.

Qt WebSockets module makes it easy to implement server side applications using Qt

Jesus loves you.

WebSockets module, just as it is easy to develop server side applications using Node.js.

Node.js also has the disadvantage of not being able to use the multi-thread required by the However, Qt WebSockets has the advantage of being able to use multi-thread. To use the Qt WebSockets module, you must add the following to the project file.

```
QT += websockets
```

Qt WebSockets provides the QWebSocket class for easy development. In addition, QWebSocket can be used for server programming, but the QWebSocketServer class is provided to facilitate server application development.

The QWebSocket class can be used in much the same way as the QTcpSocket covered in the network programming chapter and can handle events using Signal and Slot to facilitate implementation without the need for complex thread implementations.

For example, when you connect to a server application implemented with QWebSocket and the TCP connection is complete, the connected() signal provided by QWebSocket will occur. When this signal occurs, it can easily be implemented without the thread because it calls the associated Slot function. The following are some of the example source codes that connect Signal and Slot.

```
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!"));
}
```

Jesus loves you.

```
void EchoClient::onTextMessageReceived(QString message)
{
    qDebug() << "Message received:" << message;
}
```

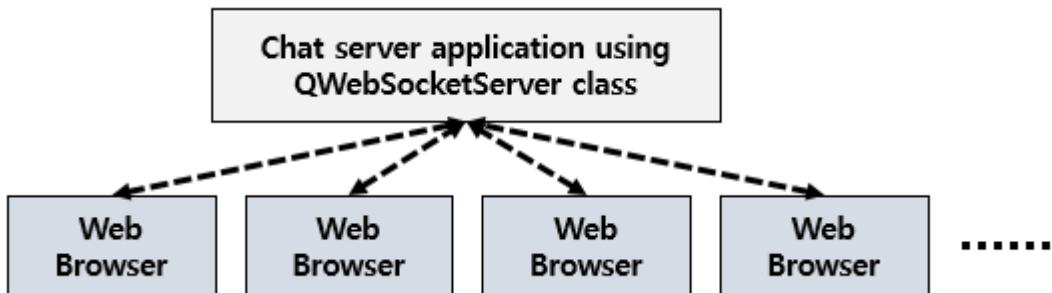
The example above is part of the example source code implemented using the QWebSocket class. When the connection with the server is complete, a connected() signal occurs and the onConnected() Slot function associated with this signal is called. And when the server receives a message sent, as seen in the onTextMessageReceived() function, a signal occurs and the onTextMessageReceived() Slot function associated with this signal is executed.

When the server is disconnected, a signal occurs and a Slot function named closed() is called. As shown in the example above, the Qt WebSockets module makes it easy to implement applications that can communicate with web browsers. The following will implement examples of communication using web browsers and Qt WebSockets modules.

- Implementing server applications for communication with web browsers

In this example, let's try implementing a server application using the Qt WebSockets module to communicate with the web browser. To implement server applications, try implementing them using the QWebSocketServer class.

In this example, let's use Google Chrome, Firefox, Safari, etc. for the web browser to communicate with the server that we're going to implement. You can use a different web browser. However, you need to make sure that the web browser you are using supports HTML5.



<FIGURE> Structure between Web browsers to communicate with chat servers

In order to implement it as shown in the figure above, let's first try creating a web

Jesus loves you.

browser HTML source code. The following example source code is HTML source code.

Example - Ch14 > 01_Chat_Example

```
<html>
  <head>
    <title>Websocket Chatting Client</title>
  </head>
  <body>
    <h1>Web Socket Chatting Client</h1>
    <p>
      <button onClick="initWebSocket()">Sever connection</button>
      <button onClick="stopWebSocket()">Connection close</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+"' " );
        }
      }
    </script>
  </body>
</html>
```

Jesus loves you.

```
        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: {
```

Jesus loves you.

```
stateStr = "CONNECTING";
break;
}

case 1: {
    stateStr = "OPEN";
    break;
}

case 2: {
    stateStr = "CLOSING";
    break;
}

case 3: {
    stateStr = "CLOSED";
    break;
}

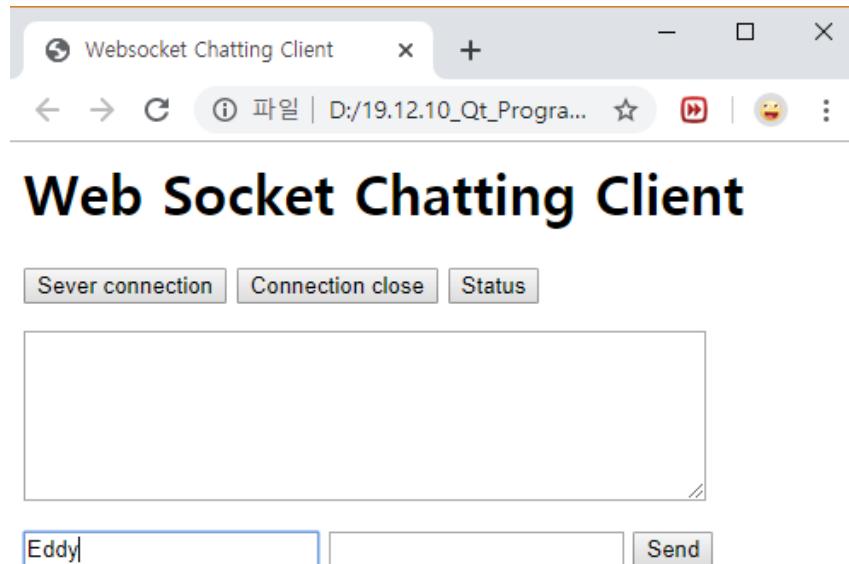
default: {
    stateStr = "UNKNOW";
    break;
}
}

debug("WebSocket state = " +
      websocket.readyState + " (" +
      stateStr + ")");

} else {
    debug("WebSocket is null");
}
}
</script>
</body>
</html>
```

Try running a WebSocket-based chat client with the HTML source code above in the web browser as shown in the picture below.

Jesus loves you.



<FIGURE> The web browser screen that ran the HTML file you created.

Run the HTML source code you created in the web browser as shown in the picture above. This time, let's create a server application to communicate with the web browser. Qt Creator creates a console-based project without a GUI Create a ChatServer class and create a source code for the header file source code as follows:

```
#ifndef CHATSERVER_H
#define CHATSERVER_H

#include <QtCore/QObject>
#include <QtCore/QStringList>
#include <QtCore/QByteArray>

QT_FORWARD_DECLARE_CLASS(QWebSocketServer)
QT_FORWARD_DECLARE_CLASS(QWebSocket)

class ChatServer : public QObject
{
    Q_OBJECT
public:
    explicit ChatServer(quint16 port,
QObject *parent = Q_NULLPTR);

    virtual ~ChatServer();
```

Jesus loves you.

```
private Q_SLOTS:  
    void onNewConnection();  
    void processMessage(QString message);  
    void socketDisconnected();  
  
private:  
    QWebSocketServer *m_pWebSocketServer;  
    QList<QWebSocket *> m_clients;  
};  
  
#endif //CHATSERVER_H
```

In this example source code, the `QWebSocketServer` class was used to implement a web server. The `onNewConnection()` Slot function is invoked when a new access signal occurs. The `processMessage()` Slot function passes messages sent by clients to all clients (web browsers) who have accessed the server.

The `socketDisconnected()` Slot function implements the ability to remove `QWebSocket` class objects to communicate with clients that are connected when a particular client connection is terminated. The following example source code is the implementation part of the `ChatServer` class.

```
#include "chatserver.h"  
#include "QtWebSockets/QWebSocketServer"  
#include "QtWebSockets/QWebSocket"  
#include <QtCore/QDebug>  
  
QT_USE_NAMESPACE  
  
ChatServer::ChatServer(quint16 port, QObject *parent) :  
    QObject(parent), m_pWebSocketServer(Q_NULLPTR), m_clients()  
{  
    m_pWebSocketServer = new QWebSocketServer(QStringLiteral("Chat Server"),  
                                              QWebSocketServer::NonSecureMode,  
                                              this);  
  
    if (m_pWebSocketServer->listen(QHostAddress::Any, port))  
    {  
        qDebug() << "Chat Server listening on port" << port;  
        connect(m_pWebSocketServer, &QWebSocketServer::newConnection,  
                this, &ChatServer::onNewConnection);  
    }  
}
```

Jesus loves you.

```
{}

void ChatServer::onNewConnection()
{
    QWebSocket *pSocket = m_pWebSocketServer->nextPendingConnection();
    connect(pSocket, &QWebSocket::textMessageReceived,
            this, &ChatServer::processMessage);
    connect(pSocket, &QWebSocket::disconnected,
            this, &ChatServer::socketDisconnected);

    m_clients << pSocket;
}

void ChatServer::processMessage(QString message)
{
    QWebSocket *pSender = qobject_cast<QWebSocket *>(sender());
    Q_FOREACH (QWebSocket *pClient, m_clients) {
        if (pClient != pSender)
            pClient->sendTextMessage(message);
    }
    qDebug() << "Send Message : " << message;
}

void ChatServer::socketDisconnected()
{
    QWebSocket *pClient = qobject_cast<QWebSocket *>(sender());
    if (pClient) {
        m_clients.removeAll(pClient);
        pClient->deleteLater();
    }
}

ChatServer::~ChatServer()
{
    m_pWebSocketServer->close();
    qDeleteAll(m_clients.begin(), m_clients.end());
}
```

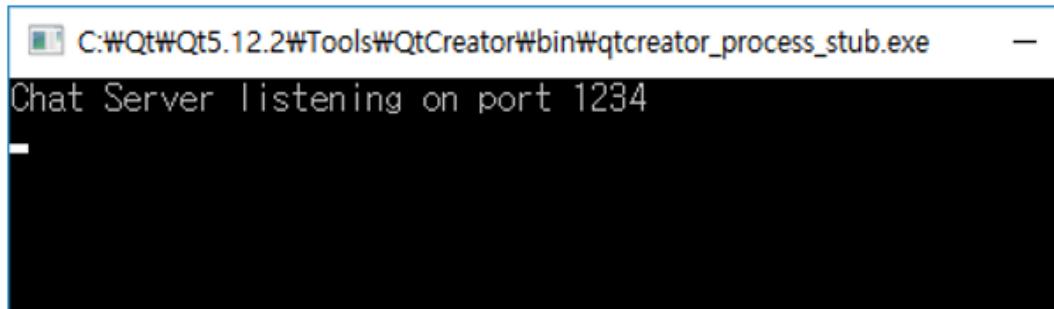
In Main.cpp, perform the following ChatServer classes: The following example source code is the main.cpp source code.

```
#include <QtCore/QCoreApplication>
#include "chatserver.h"
```

Jesus loves you.

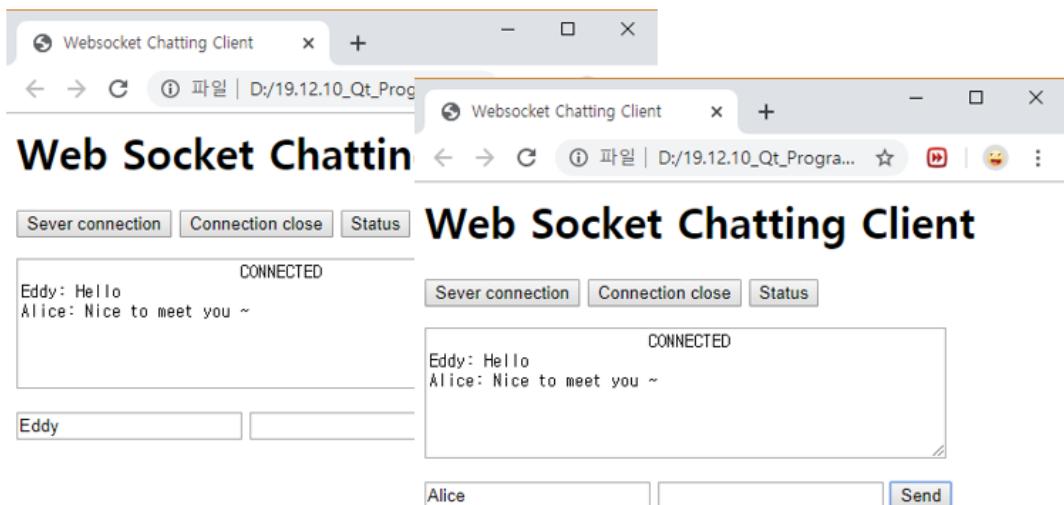
```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    ChatServer server(1234);
    return a.exec();
}
```

In the example source code above, the first factor for the ChatServer object is the port number. Let's fill out the above and run the application. When you run it, the following console windows will be launched.



<FIGURE> Sever example screen

The following runs the two web browsers that executed the HTML code you created as shown in the picture below. Then click the [Server connection button], type a message, and click the [Send] button.



<FIGURE> Web browser example screen

Jesus loves you.

In addition, the following logs can be found in the console window that ran the server.



A screenshot of a Windows command-line window titled "C:\Qt\Qt5.12.2\Tools\QtCreator\bin\qtcreator_process_stu...". The window contains the following text:

```
Chat Server Listening on port 1234
New Connection
New Connection
Send Message : "Eddy: Hello"
Send Message : "Alice: Nice to meet you ~"
```

<FIGURE> Server example screen

15. Qt WebEngine

The Qt WebEngine module provides a function to develop application applications using web browser engines. WebEngine modules use an open source web browser engine based on Google Chrome.

In Qt 5.5 and earlier, the Qt WebKit module was used instead of the Qt WebEngine module. The Qt WebKit module is based on WebKit open source and the Qt WebEngine module is based on the Google Chrome Web engine.

On the MS Windows operating system, the MinGW compiler does not support Qt WebEngine modules.

Therefore, it is necessary to use an MSVC compiler to implement applications using Qt WebEngine in MS Windows. To use the Qt WebEngine module, you must use the project file as follows:

```
QT += webenginewidgets
```

- Simple web browser implementation using QWebEngineView class

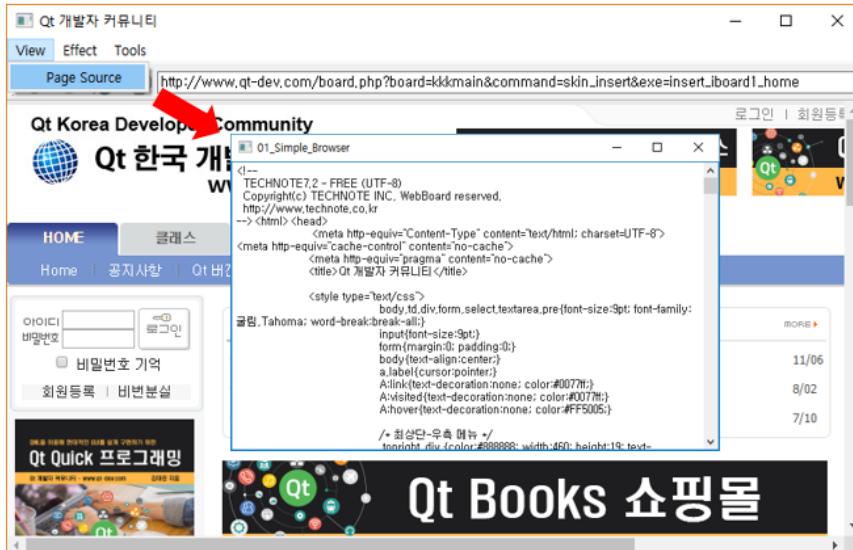
In this example, let's try to implement a simple web browser feature using the QWebEngineView class. The following picture shows an example.



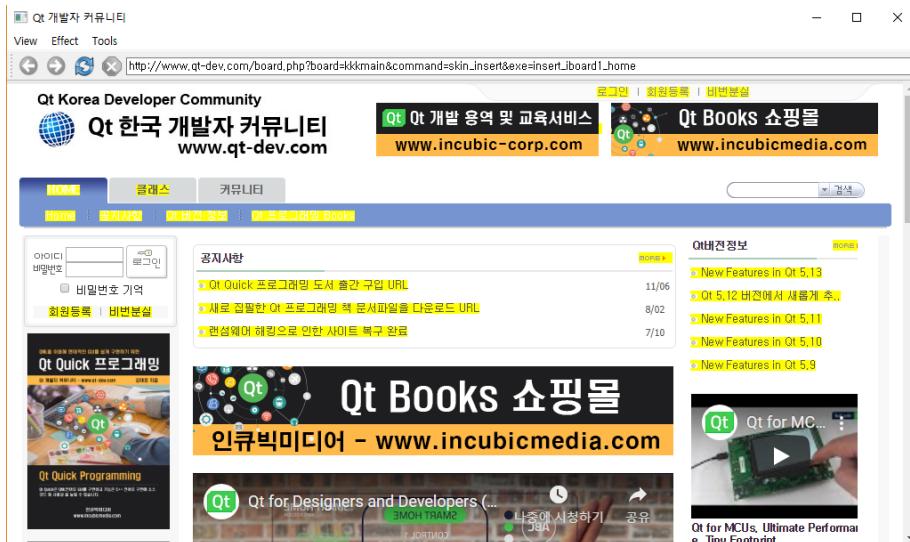
<FIGURE> Example screen

Jesus loves you.

As shown in the figure above, the example is to load the first page of the Qt developer community site. The first View menu in the menu is the [Page Source] menu. Clicking on this menu loads a window to view the source code of the current web page.



<FIGURE> Load Source Code View Window When Running [Page Source] Menu



<FIGURE> Click on the [Highlight all links] menu from the Effect menu

In the menu above, change the color of the hyperlink HTML tag to yellow on the webpage, as shown in the picture above the [Effect] menu > [Highlight all links] menu.

Click Remove GIF images on the Tools menu to remove all GIF images used in webpages from HTML tags. In the toolbar menu at the bottom of the menu, the first icon provides

Jesus loves you.

the ability to return to the previous page from the current webpage. The second icon is the next page and the third icon provides the ability to refresh the current page. The fourth icon provides the ability to stop when it is currently loading. And the address window on the right is the current webpage URL address window.

When you create a project, create a project with the QMainWindow class as the Base class and create the headers of the MainWindow class as follows.

Example - Ch15 > 01_Simgle_Browser

```
#include <QtWidgets>

QT_BEGIN_NAMESPACE
class QWebEngineView;
class QLineEdit;
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(const QUrl& url);

protected slots:
    void adjustLocation();
    void changeLocation();
    void adjustTitle();
    void setProgress(int p);
    void finishLoading(bool);
    void viewSource();
    void highlightAllLinks();
    void removeGifImages();

private:
    QString jjQuery;
    QWebEngineView *view;
    QLineEdit *locationEdit;
    int progress;
};
```

The `adjustLocation()` Slot function provides the ability to display the current web page address in the address window. When the web page is finished loading, `loadFinished()`

Jesus loves you.

Signals in QWebEngineView class occur and the adjuLocation() Slot function is called upon this signal.

The changeLocation() Slot function requests a web page from the web server you entered in the address bar to load the page by importing the address from the address bar when the user clicks the entry in the address window.

The adjustTitle() Slot function displays the string of the <title> tag on a webpage in the QMainWindow widget title bar. And the finishLoading() function is called when the web page import is complete.

The viewSource() Slot function is a function invoked by clicking [Page Source] in the View menu. The highlightAllLinks() Slot function is invoked by clicking the [Highlight all links] button in the Effect menu. Then click [Remove GIF images] in the Tools menu to call the removeGifImages() function.

jQuery was used in this project. jQuery is an open source JavaScript for easier use of JavaScript. The following example is source code MainWindow function implementation source code.

```
#include <QtWidgets>
#include <QtWebEngineWidgets>
#include "mainwindow.h"

MainWindow::MainWindow(const QUrl& url)
{
    setAttribute(Qt::WA_DeleteOnClose, true);
    progress = 0;

    QFile file;
    file.setFileName(":/jquery.min.js");
    file.open(QIODevice::ReadOnly);
    jjQuery = file.readAll();
    jjQuery.append("\nvar qt={'jQuery': jQuery.noConflict(true)};");

    file.close();

    view = new QWebEngineView(this);
    view->load(url);
    connect(view, &QWebEngineView::loadFinished,
            this, &MainWindow::adjustLocation);
```

Jesus loves you.

```
connect(view, &QWebEngineView::titleChanged,
        this, &MainWindow::adjustTitle);
connect(view, &QWebEngineView::loadProgress,
        this, &MainWindow::setProgress);
connect(view, &QWebEngineView::loadFinished,
        this, &MainWindow::finishLoading);

locationEdit = new QLineEdit(this);
locationEdit->setSizePolicy(QSizePolicy::Expanding,
                             locationEdit->sizePolicy().verticalPolicy());

connect(locationEdit, &QLineEdit::returnPressed,
        this, &MainWindow::changeLocation);

QToolBar *toolBar = addToolBar(tr("Navigation"));

toolBar->addAction(view->pageAction(QWebEnginePage::Back));
toolBar->addAction(view->pageAction(QWebEnginePage::Forward));
toolBar->addAction(view->pageAction(QWebEnginePage::Reload));
toolBar->addAction(view->pageAction(QWebEnginePage::Stop));
toolBar->addWidget(locationEdit);

QMenu *viewMenu = menuBar()->addMenu(tr("&View"));
 QAction *viewSourceAction = new QAction(Page Source, this);

connect(viewSourceAction, &QAction::triggered,
        this, &MainWindow::viewSource);

viewMenu->addAction(viewSourceAction);
QMenu *effectMenu = menuBar()->addMenu(tr("&Effect"));

effectMenu->addAction("Highlight all links",
                      this,
                      &MainWindow::highlightAllLinks);

QMenu *toolsMenu = menuBar()->addMenu(tr("&Tools"));
toolsMenu->addAction("Remove GIF images",
                      this,
                      &MainWindow::removeGifImages);
setCentralWidget(view);
}
```

Jesus loves you.

```
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;
```

Jesus loves you.

```
adjustTitle();
view->page()->runJavaScript(jQuery);
}

void MainWindow::highlightAllLinks()
{
    QString code = QStringLiteral(
        "qt.jQuery('a').each( function () "
        "{ qt.jQuery(this).css('background-color',"
        " 'yellow') } )"
    );

    view->page()->runJavaScript(code);
}

void MainWindow::removeGifImages()
{
    QString code;
    code = QStringLiteral("qt.jQuery('[src*=gif]').remove()");
    view->page()->runJavaScript(code);
}
```

16. Inter Process Communication

IPC(Inter Process Communication) refers to communication between processes that exist within the same system. Qt provides the following methods to support IPC.

- ① Unix Domain Socket and Named pipe

Linux uses the name Unix Domain Socket. MS Windows uses the name Named Pipe.

- ② Communication using QProcess class.

used to execute or obtain results from external processes.

- ③ Serial communication using the QSerialPort class.

Serial communication using UART(Universal Asynchronous Receiver/Transmitter).

- ④ Share memory using QSharedMemory class

Use the QSharedMemory class to use shared memory between external programs.

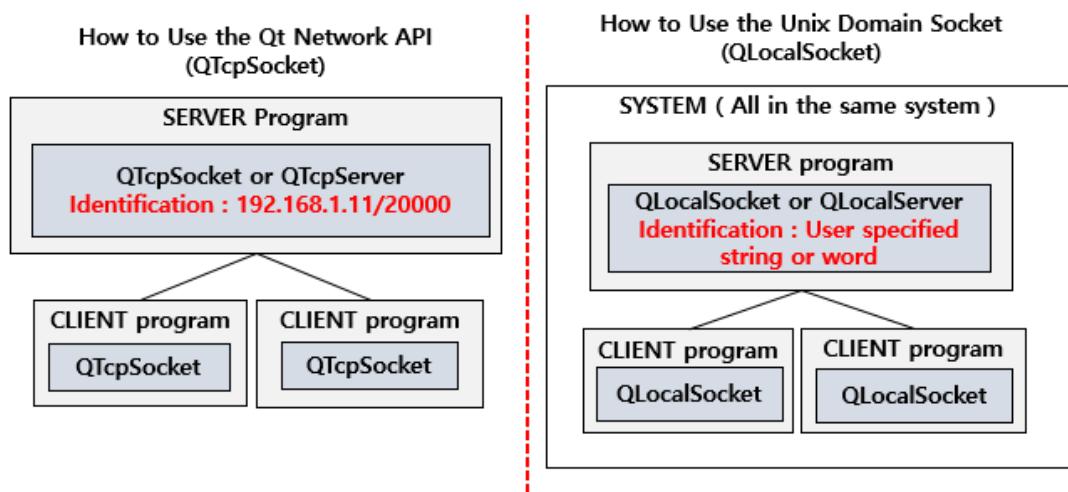
In this chapter, let's explore how to communicate between processes within the same system as mentioned above.

16.1. Unix Domain Socket and Named pipe

The name UDS (Unix Domain Socket) is the name of the interprocess communication method using the Linux platform. Both use the name "Named Pipe" on MS Windows platforms is the same for the same purpose, but have different names.

Qt allows you to use QLocalSocket and QLocalServer classes, regardless of whether you use Linux or MS Windows. The Unix Domain Socket method uses the name of the process in a way that identifies the opponent.

For example, Unix Domain Socket can use a user-defined string (word) of a process name (or name of an application) as an identifier, such as using IP and PORT as identifiers to communicate with a specific server on the network.

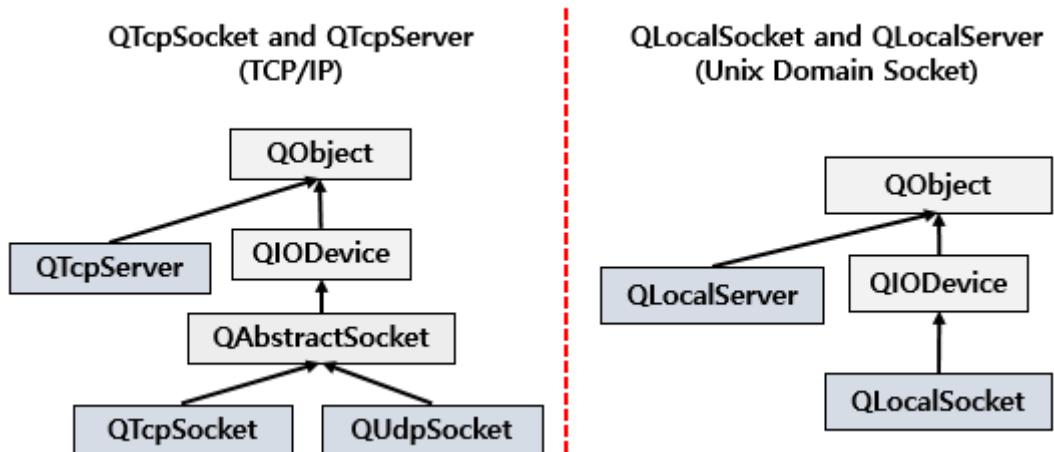


<FIGURE> TCP communication vs Unix Domain Socket

The QLocalSocket and QLocalServer classes are very similar in their usage to the QTcpSocket and QTcpServer classes used in the network and can be easily implemented using Signal and Slot.

The QLocalServer class is suitable for server side implementations and the QLocalSocket class is suitable for client implementations. For example, the QLocalServer class provides functions for listing clients to connect to the server side.

Jesus loves you.



<FIGURE> class inheritance relationship

It is recommended to use the **QLocalServer** class, although it can be implemented to list clients waiting for access using the **QLocalSocket** class.

In TCP/IP network programming, the **QLocalServer** class also provides a **list()** member function to wait for the client to connect, such as using the **list()** member function provided by the **QTcpServer** class to wait for the client to connect and can.

```
server = new QLocalServer(this);

if (!server->listen("MyServer")) {
    qDebug() << "Server error : " << server->errorString();
    ...
}

connect(server, SIGNAL(newConnection()), this, SLOT(clientConnection()));
...
```

In TCP/IP protocol networks, clients use IP and PORT to access the server, but Unix Domain Socket allows users to use the name they want.

The **connect()** member function connected the **newConnection()** signal to the **clientConnection()** Slot function when a new client connected. The following example source code is part of the example source code for clients to connect to the server using the **QLocalSocket** class.

```
socket = new QLocalSocket(this);
```

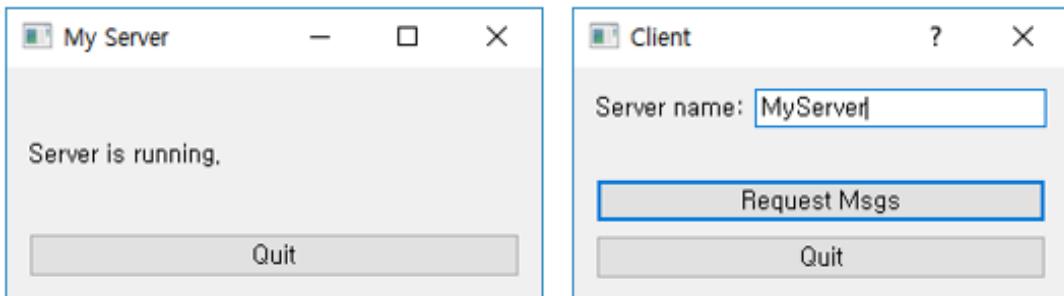
Jesus loves you.

```
connect(socket, SIGNAL(readyRead()), this, SLOT(readData()));  
connect(socket, SIGNAL(error(QLocalSocket::LocalSocketError)),  
       this, SLOT(sockError(QLocalSocket::LocalSocketError)));  
  
socket->connectToServer("MyServer");  
...
```

In the connect() function, readyRead() signal is the signal that occurs when a client receives a message. Here, we linked the readData() Slot function to process the message signal.

And the connectionToServer() member function of the QLocalSocket class at the bottom provides the ability to connect to the server. The first factor in the connectToServer() member function is the name of the server's identifier.

- Implement communication between processes using QLocalServer and QLocalSocket classes



<FIGURE> Server and Client example screen.

The example execution screen on the left is a server program and the right is a client program. When the client program runs, enter the server name on the GUI and click the [Request msgs] button to access the server program. Let's look at the server program first, the server or the client. The following example source code is the header source of the server class deployed on the server.

```
#ifndef SERVER_H  
#define SERVER_H  
  
#include <QWidget>  
  
QT_BEGIN_NAMESPACE
```

Jesus loves you.

```
class QLabel;
class QPushButton;
class QLocalServer;
QT_END_NAMESPACE

class Server : public QWidget
{
    Q_OBJECT

public:
    Server(QWidget *parent = 0);

private slots:
    void clientConnection();

private:
    QLabel *statusLabel;
    QPushButton *quitButton;
    QLocalServer *server;
    QStringList sendMsgs;
    bool msgKind;
};

#endif
```

The clientConnect() function is a Slot function that is called when a new client requests a connection. The following example is the implementation sub-source code for the Server class.

```
#include <QtWidgets>
#include <QtNetwork>
#include "server.h"
#include <qlocalserver.h>
#include <qlocalsocket.h>

Server::Server(QWidget *parent) : QWidget(parent)
{
    statusLabel = new QLabel;
    statusLabel->setWordWrap(true);
    quitButton = new QPushButton(tr("Quit"));
    quitButton->setAutoDefault(false);
```

Jesus loves you.

```
server = new QLocalServer(this);

if (!server->listen("MyServer"))
{
    qDebug() << "Server error : " << server->errorString();
    close();
    return;
}

connect(quitButton, SIGNAL(clicked()), this, SLOT(close()));
connect(server, SIGNAL(newConnection()), this, SLOT(clientConnection()));

statusLabel->setText(tr("Server is running."));

QVBoxLayout *mainLayout = new QVBoxLayout;
mainLayout->addWidget(statusLabel);
mainLayout->addWidget(quitButton);

setLayout(mainLayout);

setWindowTitle(tr("My Server"));
}

void Server::clientConnection()
{
    QByteArray writeData;

    if(msgKind)
    {
        writeData.append("Welcome to my server.");
        msgKind = false;
    }
    else
    {
        writeData.append("Who are you");
        msgKind = true;
    }

    QLocalSocket *clientConnection;
    clientConnection = server->nextPendingConnection();

    connect(clientConnection, SIGNAL(disconnected()),
```

Jesus loves you.

```
    clientConnection, SLOT(deleteLater()));

    clientConnection->write(writeData, writeData.size());
    clientConnection->flush();
    clientConnection->disconnectFromServer();
}
```

In the above example, the constructor function used the list() function to declare objects in the QLocalServer class and wait for client access. The factor in the list() function is the unique identifier that the client will connect to the server.

In the connect() function, when a new client connects, a newConnection() signal event occurs and the clientConnection() Slot function is called. This Slot function sends a message to the connected client and then terminates the connection. The following is the header source code for the Client class implemented in the client program.

```
#ifndef CLIENT_H
#define CLIENT_H

#include <QDialog>
#include <QLocalSocket>
#include <qlocalsocket.h>

QT_BEGIN_NAMESPACE
class QDialogButtonBox;
class QLabel;
class QLineEdit;
class QPushButton;
class QLocalSocket;
QT_END_NAMESPACE

class Client : public QDialog
{
    Q_OBJECT

public:
    Client(QWidget *parent = 0);

private slots:
    void requestNewMsg();
    void readData();
    void sockError(QLocalSocket::LocalSocketError socketError);
```

Jesus loves you.

```
private:  
    QLabel *hostLabel;  
    QLineEdit *hostLineEdit;  
    QLabel *statusLabel;  
    QPushButton *reqButton;  
    QPushButton *quitButton;  
    QLocalSocket *socket;  
    quint16 blockSize;  
};  
  
#endif
```

As shown in the example above, the `requestNewMsg()` Slot function is a Slot function called when the [Request msg] button is clicked. The `readData()` Slot function is called when a server receives a message sent, and the `sockError()` Slot function is a Slot function that is invoked when an error occurs. The following is the implementation part source code for the Client class.

```
#include <QtWidgets>  
#include <QtNetwork>  
#include "client.h"  
  
Client::Client(QWidget *parent) : QDialog(parent)  
{  
    hostLabel      = new QLabel(tr("Server name:"));  
    hostLineEdit  = new QLineEdit("MyServer");  
    statusLabel    = new QLabel(tr(""));  
    reqButton     = new QPushButton(tr("Request Msgs"));  
    quitButton    = new QPushButton(tr("Quit"));  
  
    socket = new QLocalSocket(this);  
    connect(reqButton, SIGNAL(clicked()), this, SLOT(requestNewMsg()));  
    connect(quitButton, SIGNAL(clicked()), this, SLOT(close()));  
    connect(socket, SIGNAL(readyRead()), this, SLOT(readData()));  
    connect(socket, SIGNAL(error(QLocalSocket::LocalSocketError)),  
            this, SLOT(sockError(QLocalSocket::LocalSocketError)));  
  
    QGridLayout *mainLayout = new QGridLayout;  
  
    mainLayout->addWidget(hostLabel, 0, 0);  
    mainLayout->addWidget(hostLineEdit, 0, 1);
```

Jesus loves you.

```
mainLayout->addWidget(statusLabel, 2, 0, 1, 2);
mainLayout->addWidget(reqButton, 3, 0, 1, 2);
mainLayout->addWidget(quitButton, 4, 0, 1, 2);
setLayout(mainLayout);

setWindowTitle(tr("Client"));
hostLineEdit->setFocus();
}

void Client::requestNewMsg()
{
    reqButton->setEnabled(false);
    socket->connectToServer(hostLineEdit->text());
}

void Client::readData()
{
    statusLabel->setText(socket->readAll());
    reqButton->setEnabled(true);
}

void Client::sockError(QLocalSocket::LocalSocketError ocketError)
{
    switch (socketError)
    {
        case QLocalSocket::ServerNotFoundError:
            qDebug() << "Server Not Found Error";
            break;
        case QLocalSocket::ConnectionRefusedError:
            qDebug() << "Connection Refused Error";
            break;
        case QLocalSocket::PeerClosedError:
            qDebug() << "Peer Closed Error";
            break;
        default:
            qDebug() << "Error : " << socket->errorString();
    }

    reqButton->setEnabled(true);
}
```

The Client Class Creator function places the GUI widget and declares the object for the QLocalSocket class. In the connect() function, the reqButton object is called the

Jesus loves you.

requestNewMsg() Slot function when the [Request msg] button is clicked.

This Slot function uses the server identifier entered in QLineEdit to connect to the server program. In addition, the sockError() Slot function is called when an error occurs in the client's QLocalSocket. This function provides a function to check which error has occurred.

Sever Example - Ch16 > 01_LocalServer, Client Example - Ch16 > 01_LocalClient

16.2. Communication using QProcess class

The QProcess class provides the ability to run external programs or to bring about the results of running them within the application you have implemented. For example, on Linux, "/bin/ls" prints directory and file information. For detailed output of directory and file information in the "/usr" directory rather than in the current directory using "/bin/ls", you can use the following.

```
/bin/ls -ltr /usr
```

As shown in the example above, QProcess can be used to output directory and file information from the Linux terminal within the /usr directory.

```
QString program = "/bin/ls";
QStringList arguments;
arguments << "-ltr" << "/usr";

QProcess *myProcess = new QProcess(parent);
myProcess->start(program, arguments);
```

As shown in the example above, the start() member function of the QProcess class executes an external program and the first factor enters the name (or file name) of the program to be executed. The second aggregate is available after the program, and if there is no option, only the first factor is required. Results executed by QProcess can be obtained by Signal and Slot, and let's take a closer look at the following examples.

- Get results from external programs running using QProcess class

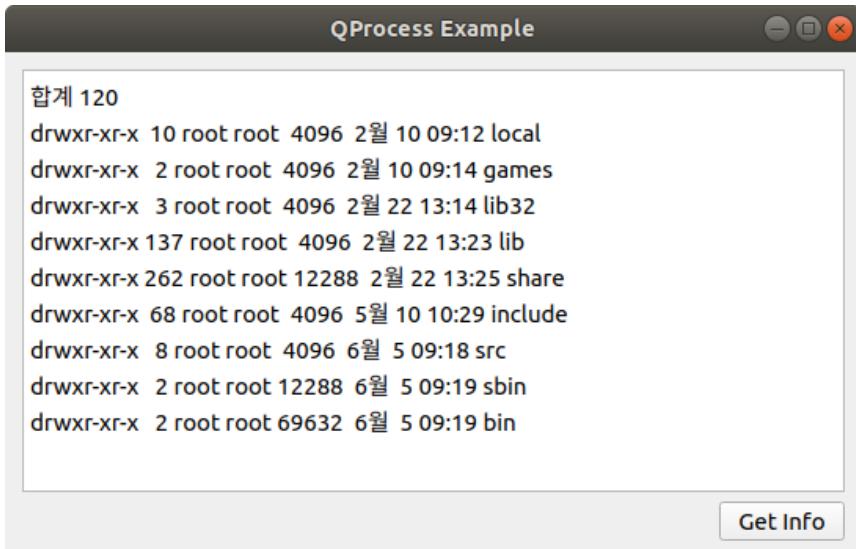
For this example, let's use QProcess class to run an external program and get the results and print the results in the GUI widget. When you run the command "/bin/ls -ltr /usr" on Linux, you output the following results:

```
qtdev@linux1:~$ ls -ltr /usr
Total 120
drwxr-xr-x 10 root root 4096 2월 10 09:12 local
drwxr-xr-x  2 root root 4096 2월 10 09:14 games
drwxr-xr-x  3 root root 4096 2월 22 13:14 lib32
```

Jesus loves you.

```
drwxr-xr-x 137 root root 4096 2월 22 13:23 lib  
...
```

As you can see above, the "-r" option used with the "/bin/ls" command is an option to output file and directory information in detail, and the /usr option is the directory name to output. The following is a screen with an example.



<FIGURE> Example screen

As shown in the figure above, clicking the [Get Info] button at the bottom executes external commands, obtains results, and outputs the results in the QTextEdit window. When creating a project, create a class that inherits QWidget and create the widget.h file as follows. Example - Ch16 > 02_QProcessExample

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

namespace Ui { class Widget; }

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

Jesus loves you.

```
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
```

The finished() Slot function is invoked when external program seal is completed using QProcess class. The ReadyReadStandardError() Slot function is invoked when an error occurs after running an external program using the QProcess class.

The readyReadStandardOutput() Slot function obtains the execution results and the started() Slot function is called when an external program is run by QProcess . The following is the source code of widget.cpp.

```
#include "widget.h"  
#include "ui_widget.h"  
#include <QDebug>  
  
Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)  
{  
    ui->setupUi(this);  
    connect(ui->getInfoButton, &QPushButton::pressed,  
            this,           &Widget::getInfoButton);  
  
    m_process = new QProcess();  
    connect(m_process, SIGNAL(finished(int, QProcess::ExitStatus)),  
            this,           SLOT(finished(int, QProcess::ExitStatus)));  
  
    connect(m_process, SIGNAL(readyReadStandardError()),  
            this,           SLOT(readyReadStandardError()));  
  
    connect(m_process, SIGNAL(readyReadStandardOutput()),
```

Jesus loves you.

```
        this,      SLOT(readyReadStandardOutput()));

    connect(m_process, SIGNAL(started()), this, SLOT(started()));
}

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

void Widget::getInfoButton()
{
    QString program = "/bin/ls";
    QStringList arguments;
    arguments << "-ltr" << "/usr";
    m_process->start(program, arguments);
}

void Widget::finished(int exitCode, QProcess::ExitStatus exitStatus)
{
    qDebug() << "Exit Code :" << exitCode;
    qDebug() << "Exit Status :" << exitStatus;
}

void Widget::readyReadStandardError()
{
    qDebug() << Q_FUNC_INFO << "ReadyError";
}

void Widget::readyReadStandardOutput()
{
    QByteArray buf = m_process->readAllStandardOutput();
    ui->textEdit->setText(buf);
}

void Widget::started()
{
    qDebug() << "Proc Started";
}
```

Declares the objects of the QProcess class in the class constructor as shown above and associates the Signals and Slot provided by QProcess. Clicking the [Get Info] button on

Jesus loves you.

the GUI calls the getInfoButton() Slot function and executes the external program using the start() member function of the QProcess class. In the ReadyReadStandardOutput() function, the readAllStandardOutput() member function of the QProcess class brings results, saves the imported results to the buffer, and outputs them to the QTextEdit widget.

16.3. Serial communication using the QSerialPort class

Qt's QSerialPort class offers features for 1:1 data transmission/receiving using Serial. It is slow compared to the current network speed. Serial, however, is often used as communication between heterogeneous or physically separated embedded device chips.

The biggest advantage of the QSerialPort class, whether it is serial communication on the Linux platform or serial communication on MS Windows, is that the QSerialPort class can use the features provided by the QSerialPort class regardless of operating system platform.

For example, the source code written using the QSerialPort class on the Linux platform can be used in MS Windows as well. Linux uses a name such as /dev/ttyUSB0 to recognize the serial port, and MS Windows uses a device name such as COM1 through COM4, so all functions for sending and receiving data follow the standard, so QSerialPort can be used regardless of the operating system platform.

In order to use cereals such as Qt provided by Qt, the following shall be added to the project file.

```
QT += serialport
```

Signal and Slot are also used in QSerialPort. Therefore, it has a signal and slot structure similar to QTcpSocket or QUdpSocket. The following is an example of connecting a device using the QSerialPort class.

```
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;
```

Jesus loves you.

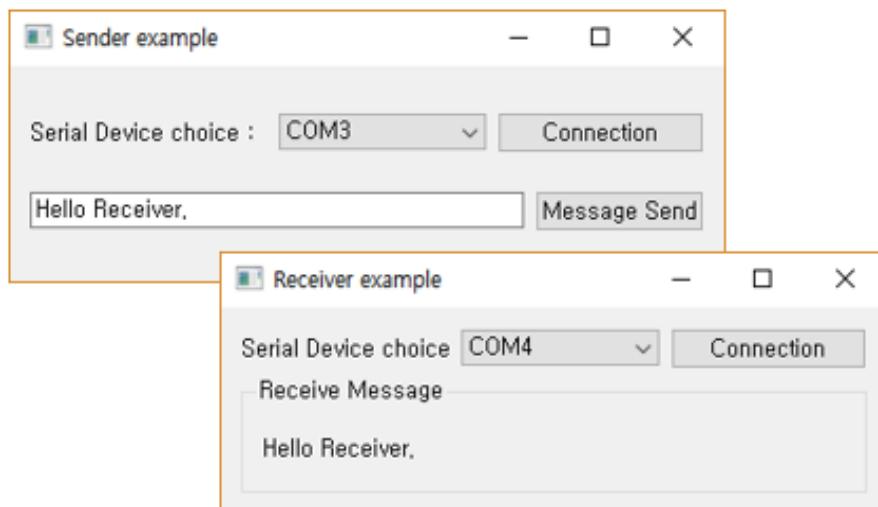
```
m_serial->setPortName(name);  
m_serial->setBaudRate(baudRate);  
m_serial->setDataBits(dataBits);  
m_serial->setParity(parity);  
m_serial->setStopBits(stopBits);  
m_serial->setFlowControl(flowControl);  
  
m_serial->open(QIODevice::ReadWrite);
```

The QSerialPort class generates an errorOccurred() signal when an error occurs, as shown in the example source code above. In the event of an error, the Slot function can be called. You can also receive data from the other party using the ReadyRead() signal.

Serials should be set to the same standard in order to connect them to the other side. The setPortName() member function enters the name of the serial device to use. setBaudRate can specify the speed. The setDataBits() function sets the number of data bits per character, and setParity() sets the parameters to be used.

To complete these settings and use serial communications, OPEN devices for sending/receiving data using the open() function that provides the QSerialPort class. Let's try to implement serial communication through an example.

- Implementing Data Transmit and Receive Examples with QSerialPort Class



<FIGURE> Sender and Receiver example screen

Jesus loves you.

An example to be discussed this time is sending/receiving data using two applications. The first example application uses the serial to transmit data. The second application is an example of outputting data received as serial ports onto the GUI.

In the figure above, the left side is an example of a sender. Select the device to be used in the GUI and click the [Connection] button. And just like the Receiver example on the right, if you select the [Connection] button and select it for use in the GUI, you are ready to send/receive data between the two applications.

Enter the message to be sent to the QLineEdit widget at the bottom of the first sender application, then click the [Message Send] button to send the message to the Receiver as shown in the figure above.

Receiver outputs the mesh sent by the sender on the GUI. Let's take a look at the source code of the sender example among the two applications and then look at the Receiver example.

Sender Example - Ch16 > 03_Sender, Receiver Example – Ch16 > 03_Receiver

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QSerialPort>
#include <QSerialPortInfo>

namespace Ui { class Widget; }

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

    struct SerialSettings {
        QString portName;
        qint32 baudRate;
        QSerialPort::DataBits dataBits;
        QSerialPort::Parity parity;
    };
}
```

Jesus loves you.

```
QSerialPort::StopBits stopBits;
QSerialPort::FlowControl flowControl;
};

SerialSettings m_serialSettings;
QSerialPort *m_serial = nullptr;

private slots:
    void connectButton();
    void sendButton();
};

#endif // WIDGET_H
```

The SerialSettings structure is a structure for storing the values that will set up the objects of the QSerialPort class. connectButton() is a Slot function called by clicking the Connect button on the GUI of the sender application. sendButton() is a Slot function called when the Send Message button is clicked. Next is the widget.cpp source code.

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

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->connectionButton, SIGNAL(pressed()),
            this,           SLOT(connectButton()));

    connect(ui->sendButton, SIGNAL(pressed()), this, SLOT(sendButton()));

    m_serialSettings.baudRate      = 115200;
    m_serialSettings.dataBits     = QSerialPort::Data8;
    m_serialSettings.parity       = QSerialPort::NoParity;
    m_serialSettings.stopBits     = QSerialPort::OneStop;
    m_serialSettings.flowControl  = QSerialPort::NoFlowControl;

    const auto infos = QSerialPortInfo::availablePorts();

    for (const QSerialPortInfo &info : infos)
        ui->comboBox->addItem(info.portName());
```

Jesus loves you.

```
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 message :" << m_serial->errorString();
    }
}

void Widget::sendButton()
{
    if(!m_serial->isOpen()) return;

    QString sendMsg = ui->sendLineEdit->text().trimmed();
    QByteArray msg = sendMsg.toLocal8Bit();

    m_serial->write(msg);
}

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

In the class creator, the `QSerialPortInfo::availablePorts()` member function provides the ability to obtain information from all available serial devices on the currently functioning platform. Obtain information on available serial devices and register them in combobox on GUI.

Jesus loves you.

When the connectButton() Slot function is called, set an option value for the serial connection saved in the class generator and connect the serial port device. In the sendButton() Slot function, the value of the character variable entered in QLineEdit is converted to QByteArray and the message is sent to the Receiver using the write() member function of the QSerialPort class. The following example source code is the widget.h source code in the Receiver example.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QSerialPort>
#include <QSerialPortInfo>

namespace Ui { class Widget; }

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

private:
    Ui::Widget *ui;

    struct SerialSettings {
        QString portName;
        qint32 baudRate;
        QSerialPort::DataBits dataBits;
        QSerialPort::Parity parity;
        QSerialPort::StopBits stopBits;
        QSerialPort::FlowControl flowControl;
    };
    SerialSettings m_serialSettings;
    QSerialPort *m_serial;

private slots:
    void connectButton();
    void readData();
    void error(QSerialPort::SerialPortError err);
```

Jesus loves you.

```
};

#endif // WIDGET_H
```

The example source code above is the header file source code of the Widget class in the Receiver example. The readData() and error() Slot function were added as a difference from the sender. The readData() function is invoked when it is received by the serial port.

The error() Slot function is then invoked when an error occurs in the serial port. The following example source code is widget.cpp.

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

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->connectionButton, SIGNAL(pressed()),
            this,           SLOT(connectButton()));

    m_serialSettings.baudRate     = 115200;
    m_serialSettings.dataBits     = QSerialPort::Data8;
    m_serialSettings.parity       = QSerialPort::NoParity;
    m_serialSettings.stopBits     = QSerialPort::OneStop;
    m_serialSettings.flowControl = QSerialPort::NoFlowControl;

    const auto infos = QSerialPortInfo::availablePorts();

    for (const QSerialPortInfo &info : infos)
        ui->comboBox->addItem(info.portName());

    m_serial = new QSerialPort(this);
    connect(m_serial, SIGNAL(error(QSerialPort::SerialPortError)),
            this,      SLOT(error(QSerialPort::SerialPortError)));
    connect(m_serial, SIGNAL(readyRead()),
            this,      SLOT(readData()));
}

void Widget::connectButton()
{
    if(m_serial->isOpen())
```

Jesus loves you.

```
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 Message : " << m_serial->errorString();
}
}

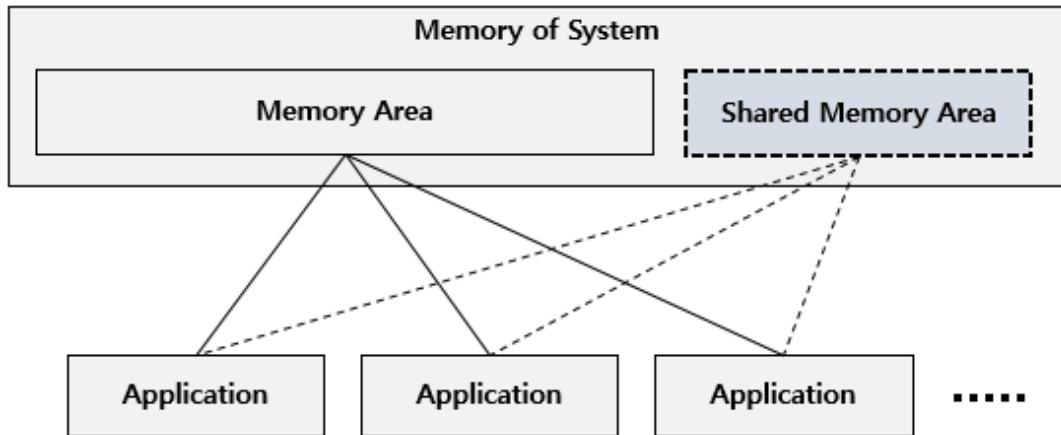
void Widget::error(QSerialPort::SerialPortError err)
{
    if (err == QSerialPort::ResourceError) {
        qDebug()<<"Critical Error: " << m_serial->errorString();
        m_serial->close();
    }
}

void Widget::readData()
{
    const QByteArray data = m_serial->readAll();
    ui->receiveMsgLabel->setText(data);
}

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

16.4. Share memory using QSharedMemory class

The use of shared memory using shared memory refers to the sharing of memory areas for the purpose of exchanging data between programs within the same system. Qt offers QSharedMemory classes for data sharing between independent applications.



<FIGURE> Independent applications share shared memory zones

To READ/WRITE a shared memory area in an independent application as shown above, use a medium that can access a specific memory region.

At Qt, data can be READed or WRITE can be used in shared memory areas by using unique KEY values. The following example source code is a method for setting the KEY value using the QSharedMemory class.

```
QString key = QString("qt-dev.com");
QSharedMemory *m_sharedMemory = new QSharedMemory(key);
```

As shown in the example above, when declaring a QSharedMemory class object, the KEY value shared as a factor can be discussed.

You can also set a unique KEY value by passing the KEY value as a factor into the setKey() member function provided by the QSharedMemory class. You can use the key() member function to refer to the set KEY value.

Always set the KEY value first, as shown in the example above, before WRITE data to a shared memory area using the QSharedMemory class. And to write the data, it can be

Jesus loves you.

used as shown in the following example.

```
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();
```

As shown in the example above, you can use lock() to prevent externally referenced data from being written to a shared memory area. In addition, the data() function can be used to obtain the address number of the shared memory area and the data can be WRITE using memcpy().

To retrieve data from a shared memory area, as shown in the example source code above, you can use the constData() member function to retrieve data from the shared memory area. The following example is for READ one WRITE data from the shared memory area.

```
QBuffer buffer;
...
m_sharedMemory->lock();
buffer.setData((char*)m_sharedMemory->constData(),
m_sharedMemory->size());
...
m_sharedMemory->unlock();
m_sharedMemory->detach();
```

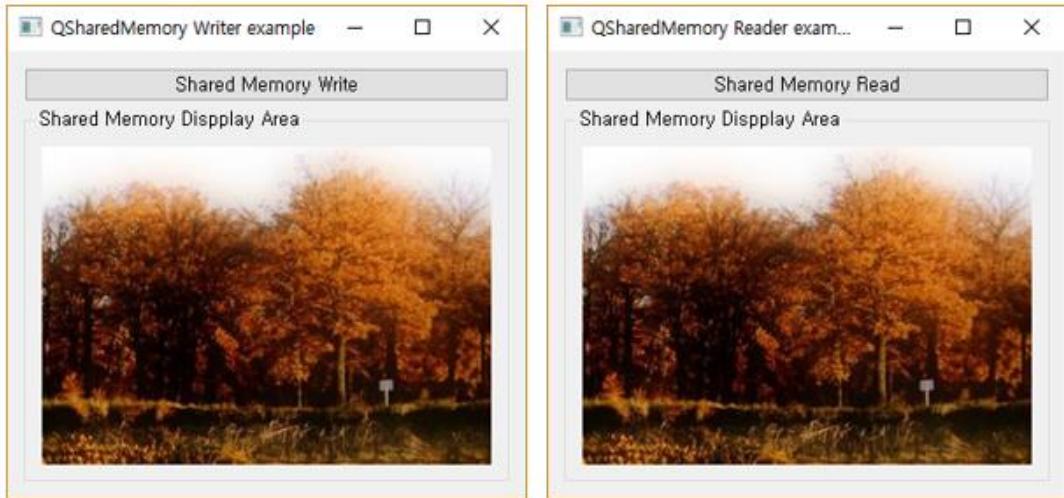
The QSharedMemory class is very easy to use, as shown above. Next, let's implement a real-world example application for data WRITE/READ in the shared memory area.

- Implement shared memory WRITE/READ using QSharedMemory class

It will implement two independent applications. The first application sets the KEY value to "qt-dev.com" and reads the image file and displays it on the GUI. Then, WRITE the binary data of the image file in the shared memory area.

Jesus loves you.

The second application will READ a WRTIE data from the first application. Example of saving READ one image binary data as QImage and then outputting it as an image to the GUI.



<FIGURE> Example screen

As shown in the figure above, the application on the left will WRITE the data from the image in the shared memory area to the shared memory area. Image files are read for WRITE in the shared data area. In other words, clicking the [Shared Memory Write] button on the GUI on the left creates a file dialogue and selects an image file.

It also WRITE the binary data of the already selected file in the shared memory area and outputs the image to the QLabel widget in the GUI. The example on the right reads the data from the shared memory area, converts it to QImage, and then outputs it to the QLabel widget as shown in the GUI.

The full source code in the above two examples is Ch16 > 04_QSharedMemory_Write directory, the application on the left is the full source code for WRITE in the shared memory area, and the 04_QSharedMemory_Reader directory is the full source code for the example application on the right.

Let's first look at the WRITE example on the left of the two example applications. The following example is the widget.h source code.

```
#include <QWidget>
#include <QSharedMemory>
```

Jesus loves you.

```
namespace Ui { class Widget; }

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

private:
    Ui::Widget *ui;
    QSharedMemory *m_sharedMemory;

private slots:
    void writeButton();
};
```

The constructor of this class declares the QSharedMemory class object and defines the key value to be used in shared memory. Key values use unique key values and QString for the type. The following example is the widget.cpp source code.

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

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

    QString key = QString("qt-dev.com");
    m_sharedMemory = new QSharedMemory(key);
}

void Widget::writeButton()
{
    if (m_sharedMemory->isAttached()) {
```

Jesus loves you.

```
if (!m_sharedMemory->detach())
    ui->label->setText(tr("Shared memory detach fail."));
}

QString fileName;
fileName = QFileDialog::getOpenFileName(
            0, QString(), QString(),
            tr("Images (*.png *.xpm *.jpg)"));

QImage image;
if (!image.load(fileName)) {
    ui->label->setText(tr("Select 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 fail."));
    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;
}
```

The writeButton() Slot function is invoked when the [Shared Memory Write] button is clicked. In this Slot function, a dialogue that allows you to select one of the files is selected. Select an image from the file to output the selected image to the GUI. Then, read binary data of the image file and WRITE in the shared memory area.

Jesus loves you.

The following example application source code on the right is an example of READING a data WRITE in the shared memory area. The following is the wishget.h source code.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QSharedMemory>

namespace Ui { class Widget; }

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

private:
    Ui::Widget *ui;
    QSharedMemory *m_sharedMemory;

private slots:
    void readButton();
};

#endif // WIDGET_H
```

The readButton() Slot function is invoked by clicking the [Read Memory] button. In this Slot function, data is read from the shared memory area, stored in QImage, and images are output to QLabel on the GUI. The following is the widget.cpp source code.

```
#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");
```

Jesus loves you.

```
m_sharedMemory = new QSharedMemory(key);
}

void Widget::readButton()
{
    if (!m_sharedMemory->attach()) {
        ui->label->setText("Read fail.");
        return;
    }

    QBuffer buffer;
    QDataStream in(&buffer);
    QImage image;

    m_sharedMemory->lock();
    buffer.setData((char*)m_sharedMemory->constData(), m_sharedMemory->size());
    buffer.open(QBuffer::ReadOnly);
    in >> image;
    m_sharedMemory->unlock();

    m_sharedMemory->detach();
    ui->label->setPixmap(QPixmap::fromImage(image));
}

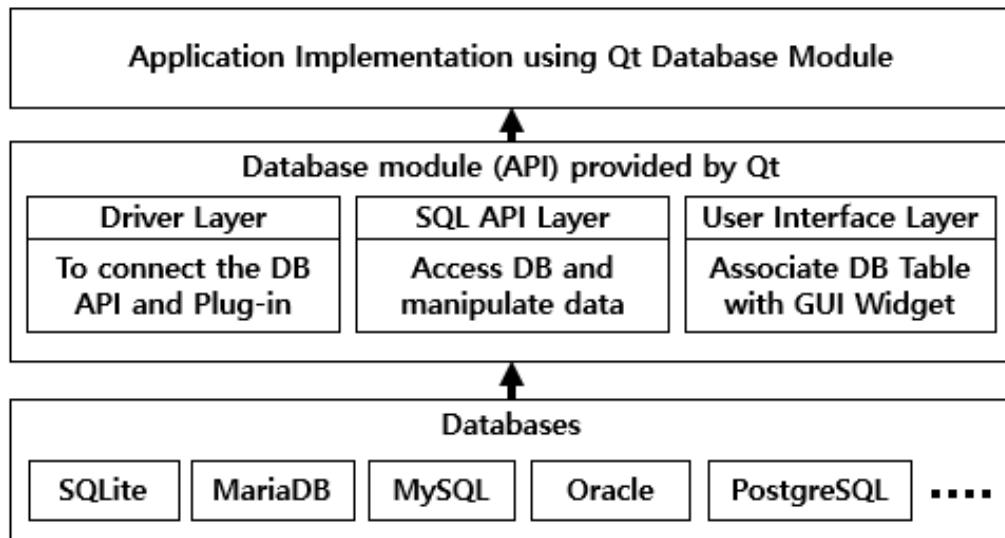
Widget::~Widget()
{
    delete ui;
}
```

17. Database

The database module (API) provided by Qt is easier to use than other development frameworks. The database module provided by Qt allows access to various databases using a unified API. For example, you can use the QSqlQuery class provided by Qt to QUERY the data stored in the MariaDB database using SQL statements.

In addition, the same QSqlQuery class that accessed the MariaDB database can be used to QUERY the data stored in the SQLite database using SQL statements.

In other words, Qt applications can be developed using common database modules provided by Qt rather than using APIs provided to access each database and use data.



<FIGURE> Qt Database Module Architecture and Layer

You can use SQLite or MariaDB as shown in the figure above, or you can use the database module provided by Qt.

That is, Qt can use a common database module (API) to access various databases. There are three categories of database modules provided by Qt:

- ① Driver Layer

The first is Driver Layer. This layer acts as a low level bridge for connecting specific

Jesus loves you.

databases. It acts as a driver for connecting to the database.

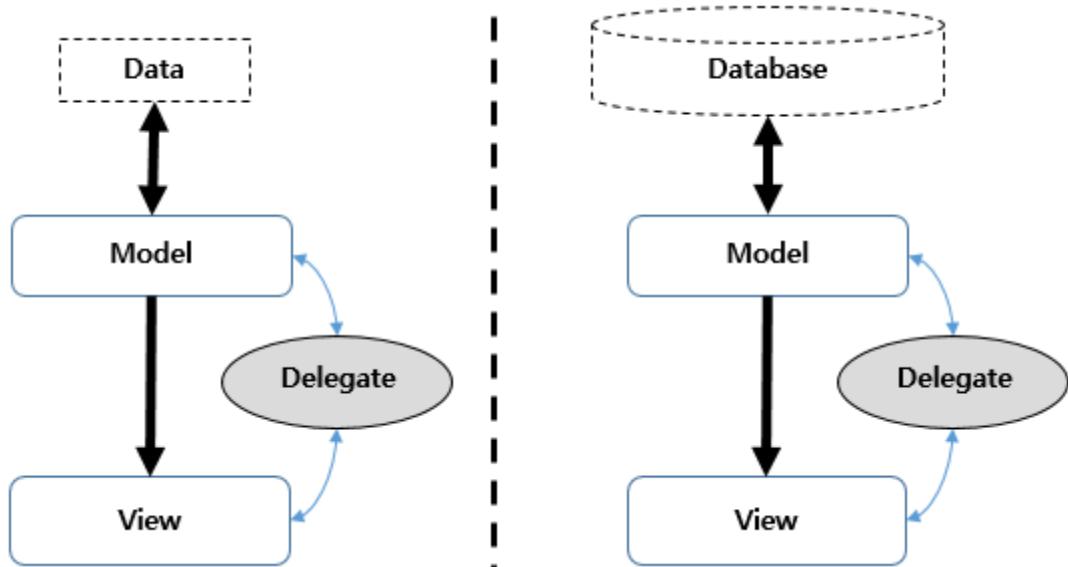
② SQL API Layer

Provides a class for connecting to the database and provides a database for utilizing data in the database table. For example, you can use the QSqlQuery class to connect to the database using the QSqlDatabase class and to QUERY.

③ User Interface Layer

This layer dealt with Model/View the other day when we were dealing with data. A model acts like a Container where data is stored. View is called View or View widgets for GUI, such as QTableView. Qt provides a database model, as you have linked a model to the View widget.

For example, a QSqlQueryModel stores data from tables stored in a database in a QUERY year. Therefore, you can associate the QSqlQueryModel with the View widget.



<FIGURE> Generic Model/View and Database Model/View

As shown in the figure above, the left side is the general model/view concept and the right side is the database model class and View. As such, you can use the database model classes provided by Qt to connect to the View widget. Qt is a mainly used model class

Jesus loves you.

that offers QSqlQueryModel, QSqlTableModel, and QSqlRelationalTableModel classes. To use the database module provided by Qt, the following keywords must be used in the project file:

```
Qt += sql
```

- ✓ Database connection

The database must first be connected to the database in order to use the database in Qt, such as the connection must be made to send/receive data from the counterpart in TCP.

```
QSqlDatabase db1 = QSqlDatabase::addDatabase("QMYSQL");
QSqlDatabase db2 = QSqlDatabase::addDatabase("QSQLITE");
QSqlDatabase db3 = QSqlDatabase::addDatabase("QPSQL");
```

The QSqlDatabase class provides the ability to connect to a database. It also supports connectivity with its own user account. The following is an example source code for accessing the database.

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");

db.setHostName("bigblue");           // IP or DNS Host name
db.setDatabaseName("flightdb");     // DB name
db.setUserName("acarlson");        // Id
db.setPassword("1uTbSbAs");        // Password

bool ok = db.open();
```

The database driver name you want to use as the first factor of the addDatabase() function should be specified, as shown in the example source code above. The following table is the name of the database driver provided by Qt.

<TABLE> 드라이버 명

DB driver name	Explanation
QDB2	IBM DB2 7.1 or later
QIBASE	Borland InterBase
QMYSQ	MariaDB or MySQL

Jesus loves you.

QOCI	Oracle Call Interface Driver
QODBC	Open Database Connectivity
QPSQL	PostgreSQL 7.3 or later
QSQLITE2	SQLite Version 2
QSQLITE	SQLite Version 3
QTDS	Sybase Adaptive Server

- ✓ Database QUERY using SQL statement

To QUERY the data in the database table, you can use the QSqlQuery class as shown in the following example.

```
QSqlQuery query;

QString qry;
qry = "SELECT name, salary FROM employee WHERE salary > 500";
query.exec(qry);
```

In order to retrieve the following data from the table in turn and obtain the data from the desired field, you can use the following.

```
while (query.next())
{
    QString name = query.value(0).toString();
    int salary = query.value(1).toInt();
}
```

In order to insert data into the table, it can be used as follows above.

```
QSqlQuery query;
query.exec("INSERT INTO employee (id, name, salary) "
          "VALUES (1001, 'Thad Beaumont', 65000)");
```

If you use SQL statements for large amounts of data, you can use Placeholder and Positioning. In terms of performance, the Placeholder approach is recommended. The following is an example source code of the Placeholder approach.

```
QSqlQuery query;
```

Jesus loves you.

```
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();
}
```

The following is an example of positioning.

```
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

Qt can handle the transaction using the member function provided by QSqlDatabase class. The following example uses the transaction() and comm() member functions.

```
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();
```

Jesus loves you.

- ✓ Database Model classes

Among the database model classes provided by Qt, the most active class provides the following classes:

<TABLE> Model classes

Name	Explanation
QSqlQueryModel	How to READ Data with SQL
QSqlTableModel	Suitable for READ/WRITE data on the table.
QSqlRelationalTableModel	How to Use the Foreign Key

The following example source code is an example source code that uses the QSqlQueryModel class to store a QUERY data in the Model.

```
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;
}
```

You can use the setQuery() member function of the QSqlQueryModel class to set up SQL queries and invoke the record(int) member function to retrieve records from tables stored in the database. The following example is an example source code using the QSqlTableModel class.

```
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);
}
```

Jesus loves you.

```
model.submitAll();
```

Records stored in the table can be retrieved through the record() member function and records in each row can be modified using the setRecord() member function. Using the setData() member function, the specific row (row) and column (column) can be modified as follows.

```
QSqlTableModel model;
model.setTable("employee");

model.setData(model.index(row, column), 75000);
model.submitAll();
```

In addition, data can be deleted in batches using the removeRows() member function of QSqlTableModel class.

```
model.removeRows(row, 5);
model.submitAll();
```

The first factor in the removeRows() member function is the number of the row to delete. The second factor is the number of records to delete. Next, let's learn how to use the QSqlRelationalTableModel class. This class offers a model for searching tables using the Foreign Key. Take two tables for example, as shown below.

<TABLE> MainTable Table

Field name	Type
id	INTERGER
Date	DATE
Time	TIME
Hostname	INTERGER
IP	INTERGER

<TABLE> DeviceTable Table

Field name	Type
id	INTERGER
Hostname	VARCHAR(255)

Jesus loves you.

IP	VARCHAR(16)
----	-------------

There are Main Table and Device Table as shown in the table above. Insert the record into the Device Table first. Then, if you want to insert Hostname and IP in the Device Table when you insert data from the main table, you can use the following.

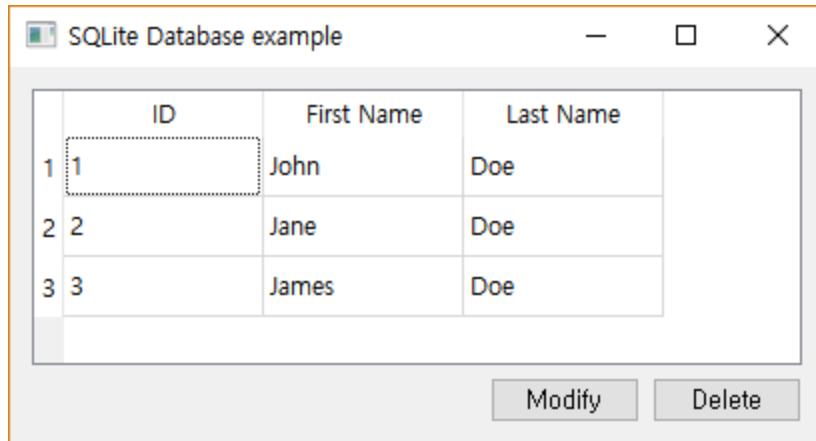
```
QSqlRelationalTableModel *modelMain;
QSqlRelationalTableModel *modelDevice;

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

Using the setRelation() member function provided by the QSqlRelationalTableModel class, as shown above, the Hostname of the "DeviceTable" table and the IP record field data are inserted in the maModelMain object equally So far we have looked at the database modules offered by Qt. Let's try to write an example for ourselves.

- SQLite Database example

This example is designed to fill out an example to use the database sqlite Let's see. The following example to run the screen.



<FIGURE> Example screen

When you run the program, as shown in the figure above, you create a SQLite database. When a database is created, a table named name creates . This is an example of inserting data into the name table and outputting data to the QTableView widget as shown in the figure above.

Jesus loves you.

Click the [Modify] button to change the first name of the record with id 1 to 'Eddy'. Then, change the Last Name to 'Kim'. Click the [Delete] button to delete data with id number 1. Create a project based on QWidget and create a header source code as follows:

Example - Ch17 > 01_SQLite_Example

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QDebug>
#include <QSqlDatabase>
#include <QSqlError>
#include <QSqlRecord>
#include <QSqlQuery>
#include <QSqlTableModel>
#include <QTableView>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();

private:
    Ui::Widget *ui;

    QSqlDatabase db;
    QSqlTableModel *model;

    void initializeDataBase();
    void creationTable();

    void insertDataToTable();
    void initializeModel();

private slots:
    void slotPbtUpdate();
```

Jesus loves you.

```
void slotPbtDelete();
};

#endif // WIDGET_H
```

The initializeDataBase() function creates SQLite database files. The creationTable() function creates a table within the database that you create. InsertDataToTable() inserts data into the table that you created.

The initializeModel() function creates a QSqlTableModel class object and sets the names table to model data. Then, set the header. The following example is the widget.cpp source code.

```
#include "widget.h"
#include "ui_widget.h"
#include <QTableView>
#include <QFile>

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

    initializeDataBase();
    creationTable();
    insertDataToTable();
    initializeModel();

    ui->tableView->setModel(model);
    connect(ui->pbtUpdate, SIGNAL(pressed()), this, SLOT(slotPbtUpdate()));
    connect(ui->pbtDelete, SIGNAL(pressed()), this, SLOT(slotPbtDelete()));
}

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

void Widget::initializeDataBase()
{
    QFile::remove("./testdatabase.db");
    db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("./testdatabase.db");
```

Jesus loves you.

```
if( !db.open() )
    qDebug() << db.lastError();
}

void Widget::creationTable()
{
    QSqlQuery qry;
    qry.prepare("CREATE TABLE IF NOT EXISTS names "
               "("
               "    id INTEGER UNIQUE PRIMARY KEY, "
               "    firstname VARCHAR(30), "
               "    lastname  VARCHAR(30)"
               ")");
}

if( !qry.exec() )
    qDebug() << qry.lastError();
}

void Widget::insertDataToTable()
{
    QSqlQuery qry;
    qry.prepare("INSERT INTO names (id, firstname, lastname) "
               "VALUES (1, 'John', 'Doe')");

    if( !qry.exec() )
        qDebug() << qry.lastError();

    qry.prepare("INSERT INTO names (id, firstname, lastname) "
               "VALUES (2, 'Jane', 'Doe')");
    if( !qry.exec() )
        qDebug() << qry.lastError();

    qry.prepare("INSERT INTO names (id, firstname, lastname) "
               "VALUES (3, 'James', 'Doe')");
    if( !qry.exec() )
        qDebug() << qry.lastError();
}

void Widget::initializeModel()
{
    model = new QSqlTableModel(this, db);
```

Jesus loves you.

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

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

void Widget::slotPbtUpdate()
{
    QSqlQuery qry;

    qry.prepare("UPDATE names SET firstname = 'Eddy', "
               "lastname = 'Kim' WHERE id = 1" );

    if( !qry.exec() )
        qDebug() << qry.lastError();

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

void Widget::slotPbtDelete()
{
    QSqlQuery qry;
    qry.prepare( "DELETE FROM names WHERE id = 1" );

    if( !qry.exec() )
        qDebug() << qry.lastError();

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

Jesus loves you.

- Example of searching database tables

In this example, the data stored in the database table is imported into the QSqlTableModel and output to the QTableView widget. And let's try to implement a time-based search.

	Time	Number	Message
1	16:53:02.428	41	Message : 41
2	17:09:42.441	18467	Message : 18467
3	17:26:22.453	6334	Message : 6334
4	17:43:02.466	26500	Message : 26500
5	17:59:42.479	19169	Message : 19169

<FIGURE> Example screen

Click the [Search] button based on time, as shown in the figure above, to display only the data that matches the time condition of the data below. You can use the setFilter() member function provided by the QSqlTableModel class to retrieve data.

When you create a project, you inherit a Widget class that inherits the QWidget class. Create additional DatabaseHandler classes. The following example source code is the header source code of the DatabaseHandler class.

Example - Ch17 > 02_Search_Example

```
#ifndef DATABASEHANDLER_H
#define DATABASEHANDLER_H

#include <QObject>
#include <QSql>
#include <QSqlQuery>
#include <QSqlError>
#include <QSqlDatabase>
#include <QFile>
```

Jesus loves you.

```
#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();
};

#endif // DATABASEHANDLER_H
```

The openDataBase() function creates a database file. And declare the objects of the QSqlDatabase class. The restoreDataBase() function calls the openDataBase() function.

Then, when you return true from this function, call the createTable() function. This function creates a table defined by the name RECEIVE_MAIL. The following example is the function implementation sub-source code of the DatabaseHandler class.

```
#include "databasehandler.h"
#include <QDir>
#include <QDebug>

DatabaseHandler::DatabaseHandler(QObject *parent)
    : QObject(parent)
{
```

Jesus loves you.

```
{}

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() << "Database connection fail.";
        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()
{
```

Jesus loves you.

```
    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() << "Table create fail error : "
                  << query.lastError().text();
        return false;
    } else {
        return true;
    }
}

bool DatabaseHandler::insertIntoTable(const QVariantList &data)
{
    QSqlQuery query;
    query.prepare("INSERT INTO "
                  "RECEIVE_MAIL (TIME, RANDOM, MESSAGE) "
                  "VALUES (:TIME, :RANDOM, :MESSAGE )");

    query.bindValue(":TIME",           data[0].toTime());
    query.bindValue(":MESSAGE",       data[1].toInt());
    query.bindValue(":RANDOM",        data[2].toString());

    if(!query.exec()){
        qDebug() << "Insert error - "
                  << query.lastError().text();
        return false;
    } else {
        return true;
    }
}

DatabaseHandler::~DatabaseHandler()
```

Jesus loves you.

```
{  
}
```

Of the above functions, insertIntoTable() inserts a record into the table you created. When inserting records into a table, insert records using Placeholder. The following example source code is the widget.h header source code.

```
#ifndef WIDGET_H  
#define WIDGET_H  
  
#include <QWidget>  
#include <QSqlTableModel>  
#include "databasehandler.h"  
  
namespace Ui {  
class Widget;  
}  
  
class Widget : public QWidget  
{  
    Q_OBJECT  
  
public:  
    explicit Widget(QWidget *parent = nullptr);  
    ~Widget();  
  
private:  
    Ui::Widget *ui;  
  
    DatabaseHandler *dbHandler;  
    QSqlTableModel *model;  
  
    void setupModel(const QString &tableName,  
                   const QStringList &headers);  
    void createUserInterface();  
  
private slots:  
    void onPushButton();  
};
```

Jesus loves you.

```
#endif // WIDGET_H
```

Of the above functions, opPushButton() is a Slot function called when the [Search] button is clicked. The setupModel() function declares the QSqlTableModel class object and declares the header. And arrange them in ascending order.

The createUserInterface() function sets the QTableView widget that is placed on the GUI. And set the time in QTimeEdit to the current time to search for time, as seen in the GUI. The following example is the source code for Widget.cpp.

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

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);
    ui->timeFROM->setDisplayFormat("hh:mm:ss");
    ui->timeTO->setDisplayFormat("hh:mm:ss");

    dbHandler = new DatabaseHandler();
    dbHandler->connectToDataBase();

    for(int i = 0; i < 10; i++)
    {
        QVariantList data;

        QTime currTime = QTime::currentTime();
        currTime = currTime.addSecs(i*1000);

        int random = qrand();
        data.append(currTime);
        data.append(random);
        data.append("Message : "
                   + QString::number(random));

        dbHandler->insertIntoTable(data);
    }

    this->setupModel("RECEIVE_MAIL",
```

Jesus loves you.

```
 QStringList() << "ID"
             << "Time"
             << "Number"
             << "Message"
         );

    this->createUserInterface();
}

void Widget::setupModel(const QString &tableName,
                       const QStringList &headers)
{
    model = new QSqlTableModel(this);
    model->setTable(tableName);

    for(int i = 0, j = 0; i < model->columnCount(); i++, j++) {
        model->setHeaderData(i, Qt::Horizontal, headers[j]);
    }

    model->setSort(0,Qt::AscendingOrder);
}

void Widget::createUserInterface()
{
    ui->tableView->setModel(model);
    ui->tableView->setColumnHidden(0, true);
    ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
    ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection);
    ui->tableView->resizeColumnsToContents();
    ui->tableView->horizontalHeader()->setStretchLastSection(true);

    for(int i = 0 ; i < 4 ; i++)
        ui->tableView->setColumnWidth(i, 100);

    model->select();
    ui->timeFROM-> setTime(QTime::currentTime());
    ui->timeTO-> setTime(QTime::currentTime());

    connect(ui->pbtSearch, SIGNAL(pressed()),
            this,           SLOT(onPushButton())));
}
```

Jesus loves you.

```
void Widget::onPushButton()
{
    QString str = QString("TIME between '%3' and '%4'")
                  .arg(ui->timeFROM->time().toString("hh:mm:ss"));

    qDebug() << Q_FUNC_INFO << str;

    model->setFilter(QString( "TIME between '%3' and '%4'" )
                      .arg(ui->timeFROM->time().toString("hh:mm:ss"),
                            ui->timeTO->time().toString("hh:mm:ss")));
    model->select();
}

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

18. Multimedia

Qt offers multimedia modules for developing applications using audio, video and camera. In addition to simple audio playback functions, it also provides various APIs necessary for implementing multimedia applications, such as the ability to extract, analyze, or transfer sound sources from a certain time segment to a network by extracting Bit rate and Sample rate. In order to use multimedia modules in Qt, the following must be added to the project file.

```
QT += multimedia multimediawidgets
```

The multimedia modules provided by Qt use Signal and Slot to process events. This allows developers to easily implement very complex functions in dealing with multimedia. For example, if you have a 60 second sound source, you can divide 60 seconds of music data into 100 milliseconds to play it back. If you're using an API that internally splits and plays data without replaying, you don't have to worry about the implementation of splitting and replaying, but if the media sound doesn't play.

Suppose you do not break down into 100 millisconds, but you realize that you can divide into 100 millisconds and play back the following sound data at the end of each 100 milliscond. The ability to easily import and play data from the next sound source as a Slot function would make it easy to implement a very complex implementation, provided that 100 milliseconds are at the end of the signal.

Another example is Internet radio stations. If a sound input from a microphone is to be transmitted over the network and played on the receiving side application, such as an Internet radio station that must play the music at every moment, it cannot be realized by simply playing the MP3. As described just now, real-time music that has been split and transmitted over the network should be played.

This requires the control of the sound source using a lower level audio API rather than simply using a simple API to play MP3, which has the advantage of easily implementing complex implementations using Signal and Slot.

Qt also supports multi-platforms. As a result, multimedia applications implemented on

Jesus loves you.

Linux can be used in MS Windows. So far we have briefly explored the features and benefits of the Qt multimedia module and the main points to be covered in this chapter are as follows.

① Audio

Consider implementing a function that decodes (repliesced or decompressed) encoded music (such as MP3s) after restoration or decompression). We will also look at how to handle data received from the microphone, as well as how to handle low level music.

② Video

Let's take a look at how to play a video file. In addition to the video-related APIs provided by Qt, let's look at the open source APIs that support video decoding.

③ Camera

Let's look at how the system can detect available camera devices and output images from the camera to the GUI.

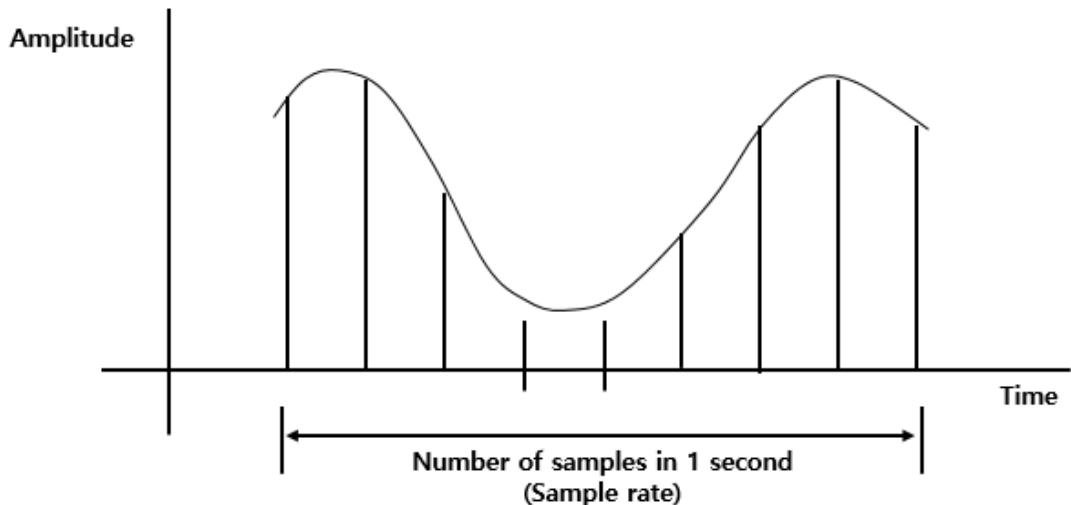
18.1. Audio

Before we look at the implementation of the audio functions provided by Qt, let's first look at the essential basic concepts required when handling audio.

Although sound waves are transmitted to other people's ears in the form of analog waveforms when people speak, analog waveform signals such as sound sources must be converted to binary forms in order to play the sound source on a digital device on the computer. In order to control a binary-converged sound source, the concepts of sample rate and bit rate must be understood first.

- ✓ Sample rate

Sample rate means a specific value extracted by converting a continuous waveform into binary digital, such as an analog signal. That is, sample rate means the number of samples taken per second or the number of samples taken. The terms used in audio are expressed in 52.3 KHz (52,300 Hz) or 22.4 KHz (22,400 Hz). This means the number of samples repeated per second.



<FIGURE> Sample rate

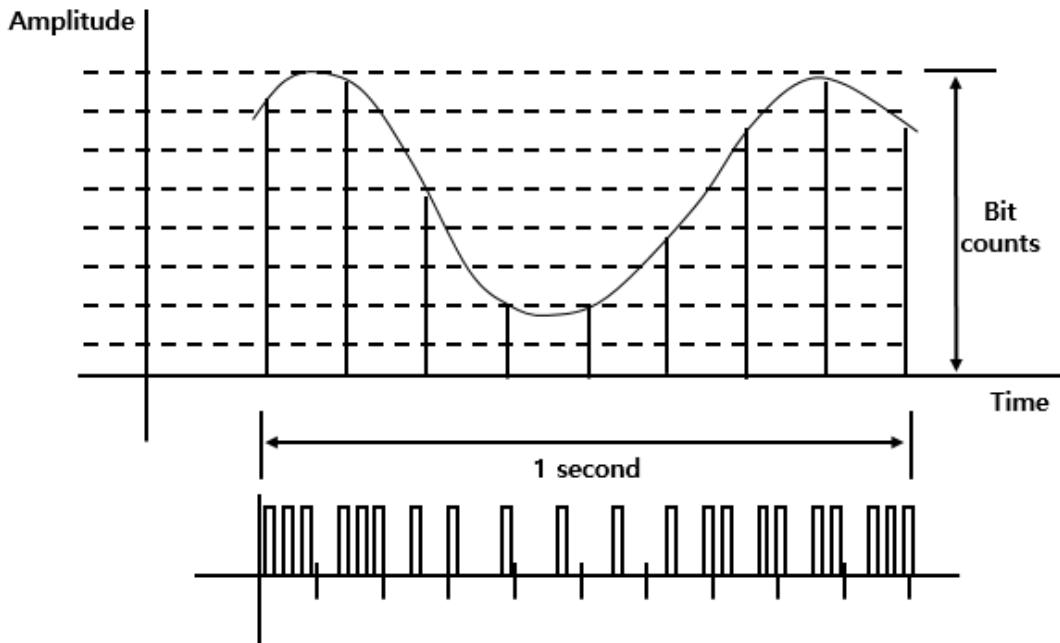
For example, 52.1 KHz means that 52,100 samples are taken in one second. You can see that the number of samples is getting bigger, and the number is getting more detailed.

Jesus loves you.

The sound quality we use as CD sound uses 44.1 Khz.

✓ Bit rate

Bit rate means the number of bits per second. i.e. the number of bits used in 1 second, i.e. use the bps unit. When we talk about the quality of music, we refer to bps like 96 Kbps, 128 Kbps, 192 Kbps, etc.



<FIGURE> Bit rate

The sound quality on the phone like the cell phone we use is 8 Kbps, AM radio 32 Kbps, MP3 56 Kbps, FM radio 96 Kbps, CD 192 Kbps, FLAC uses 500 Kbps to 1 Mbps. In other words, the higher the bps, the better the sound quality.

As shown in the figure above, analog conversion signals are expressed as 1 and 0 in digital, but only these two states are possible. And digitizing analog to 1 and 0 is called Pulse Code Modulation (PCM).

If you obtain the sample rate and bit rate you have looked at so far, you can calculate the bytes of the entire sound after decoding the MP3 file. For example, the sample.mp3 file has a replay time of 299 seconds (4:59) and Bit Rate is 56 Kbps, and Sample rate is 22,050 Hz, instant noodle play time of * Bit rate / 8 is the number of bytes in the sample.mp3 file.

Jesus loves you.

And PCM can be obtained as follows. PCM file size can be calculated by doing PCM processing per second * playback time / 8.

File : sample.mp3

Play time : 299.277
secs
Bit rate: 56Kbps (CBR)
Channel: 2 ch
Bit : 16 bit
Sample rate: 22050 Hz

Size: 2,094,939 byte

File name : stop.pcm
Size : 26,394,624 byte

Calculate the size of MP3 files

$$\text{File size} = \text{Play time} * \text{Bit rate} / 8$$

$$299.277 * 56,000(\text{Hz}) / 8 = [2,093,000 \text{ Bytes}]$$

Differences in actual file size and calculated results are different because no time of less than 1 second was taken into account.

$$\text{PCM (Amount of data per second)} = \text{Sample rate} * \text{Channel} * \text{Bit} \\ 22,050(\text{Hz}) * 2 * 16 (\text{bit}) = 705,600 \text{ Bit (88,200 Kbytes)}$$

Amount of data per second

The amount of data per second is 705 Kbps. For example, the audio quality of the MP3 is 56 Kbps, so it is about 12 times more different than that of Kbps

$$\text{PCM File size} = \text{PCM (Amount of data per second)} * \text{Play time} / 8 \\ 705,600 * 299.277 / 8 = 26,394,624 \text{ Bytes}$$

The difference between actual file size and calculated results is slightly different because no time of less than 1 second was taken into account.

<FIGURE> Calculate MP3 file size and PCM file size

In audio, a channel is called a mono for transmitting sound in one direction, and two channels for transmitting sound in two directions is called a stereo channel. Just the same sound from two speakers is not called stereo channels.

This means, for example, how many directions are divided into sounds in the recording process? The channel referred to here in calculating MP3 files means the channel just mentioned. And where bit means the number of bits per sample. The larger the number of bits, the smaller the analog waveform can be.

So far we have identified the basic concepts that we need before implementing audio. Sample rate, bit rate, number of bits, and channel reviewed so far are used to set the actual audio format.

Even if the concept is not known exactly, it is not a problem to implement the function of playing MP3 files. In order to play MP3, the function can be implemented easily using

Jesus loves you.

the QMediaPlayer class.

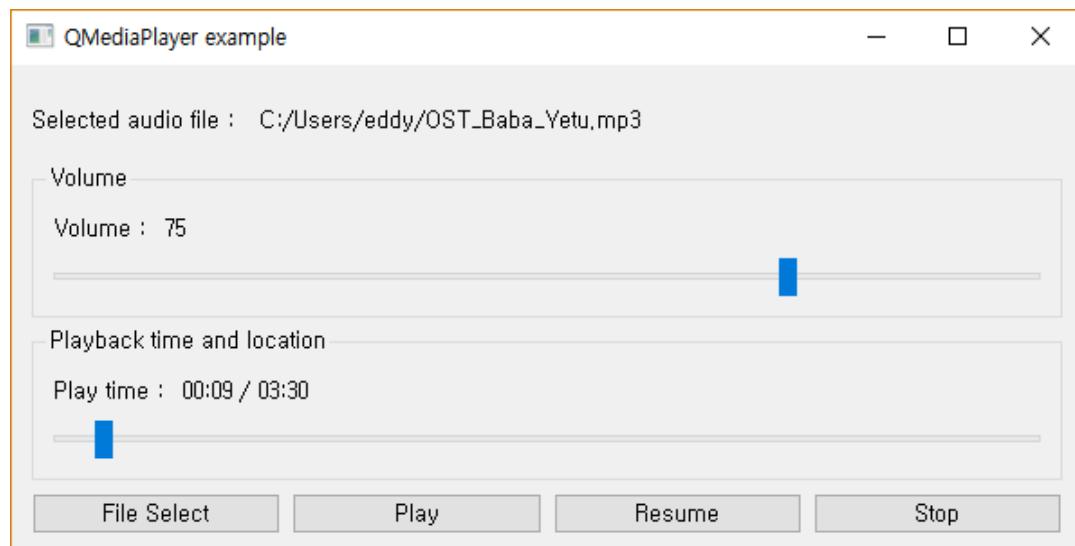
```
player = new QMediaPlayer;  
...  
player->setMedia(QUrl::fromLocalFile("/My/Music/song.mp3"));  
player->setVolume(50);  
player->play();
```

In addition to the ability to play local files stored in the system's file system, the QMediaPlayer class also provides the ability to play files located externally through the URL using the HTTP protocol.

```
player = new QMediaPlayer;  
playlist = new QMediaPlaylist(player);  
  
playlist->addMedia(QUrl("http://example.com/myfile1.mp3"));  
playlist->addMedia(QUrl("http://example.com/myfile2.mp3"));  
...  
playlist->setcurrentIndex(1);  
player->play();
```

We briefly looked at how to play MP3 files. Next, let's learn more about the audio features offered by Qt through a real-world example.

- Example of playing audio using the QMediaPlayer class



<FIGURE> Example screen

Jesus loves you.

The picture above shows the running screen of this example. In this example, let's implement a feature that plays audio MP3 files using the QMediaPlayer class. Click the File Select button, as shown in the example execution screen above, to select an MP3 file from the file dialogue. The [Play] button plays the selected music file.

The first QSlider on the GUI controls the volume. The second QSlider shows where it is currently playing and allows the user to change the playback position using the QSlider. The following example source code is the widget.h header source file.

Example - Ch18 > 01_QMediaPlayer_Example

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QMediaPlayer>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;
    QString        m_fName;
    QMediaPlayer   *m_player;
    qint64         m_duration;

private slots:
    void onOpenBtn();
    void onPlayBtn();
    void onPauseBtn();
    void onStopBtn();

    void sliderValueChange(int val);
    void durationChanged(qint64 duration);
    void positionChanged(qint64 progress);
```

Jesus loves you.

```
void seek(int seconds);
};

#endif // WIDGET_H
```

The `m_fName` variable stores the path and filename of the selected file in the file dialogue. `m_duration` saves the total playback time of the music file in seconds. The `sliderValueChange()` Slot function provides the ability to control the volume of the `QMiaPlayer` currently playing.

The `durationChanged()` function is a Slot function that is called when you click the Select File button and the total playback time of the music file that you want to play changes. The `positionChanged()` function is a Slot function that is called when the current replay position changes during playback.

The `view()` Slot function also changes and plays the replay position again when the user drags the QSlider widget pointing to the current position in the GUI. The following example is the source code for `Widget.cpp`.

```
#include "widget.h"
#include "ui_widget.h"
#include <QFileDialog>
#include <QTime>

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    m_duration = 0;
    ui->setupUi(this);
    connect(ui->volSlider, SIGNAL(valueChanged(int)),
            this,           SLOT(sliderValueChange(int)));

    ui->sliderPosition->setEnabled(false);
    connect(ui->pbtOpen,   SIGNAL(clicked()), this, SLOT(onOpenBtn()));
    connect(ui->pbtPlay,   SIGNAL(clicked()), this, SLOT(onPlayBtn()));
    connect(ui->pbtPause,  SIGNAL(clicked()), this, SLOT(onPauseBtn()));
    connect(ui->pbtStop,   SIGNAL(clicked()), this, SLOT(onStopBtn()));

    m_player = new QMediaPlayer();
    connect(m_player, &QMediaPlayer::durationChanged,
            this,     &Widget::durationChanged);
    connect(m_player, &QMediaPlayer::positionChanged,
            this,     &Widget::positionChanged);
```

Jesus loves you.

```
connect(ui->sliderPosition, &QSlider::sliderMoved,
        this,                      &Widget::seek);
}

void Widget::sliderValueChange(int val)
{
    ui->labelVolume->setText(QString("%1").arg(val));
    m_player->setVolume(val);
}

void Widget::onOpenBtn()
{
    m_fName = QFileDialog::getOpenFileName(this,
                                            "Open File",
                                            QDir::homePath(),
                                            "MP3 files (*.mp3);");

    if(!m_fName.isNull())
        ui->lblFileName->setText(m_fName);
}

void Widget::onPlayBtn()
{
    if(!m_fName.isNull())
    {
        m_player->setMedia(QUrl::fromLocalFile(m_fName));
        ui->sliderPosition->setEnabled(true);
        m_player->play();
    }
}

void Widget::onPauseBtn()
{
    int state = m_player->state();
    if(state == QMediaPlayer::PausedState)
    {
        m_player->play();
    }
    else if(state == QMediaPlayer::PlayingState)
    {
        m_player->pause();
    }
}
```

Jesus loves you.

```
void Widget::onStopBtn()
{
    int state = m_player->state();
    if(state == QMediaPlayer::PlayingState)
    {
        m_player->stop();
    }
}

void Widget::durationChanged(qint64 duration)
{
    m_duration = duration / 1000;
    ui->sliderPosition->setMaximum(m_duration);
}

void Widget::positionChanged(qint64 progress)
{
    if (!ui->sliderPosition->isSliderDown())
        ui->sliderPosition->setValue(progress / 1000);

    qint64 currentInfo = progress / 1000;

    QString playTimeStr;
    if (currentInfo || m_duration) {
        QTime currentTime((currentInfo / 3600) % 60, (currentInfo / 60) % 60,
                           currentInfo % 60, (currentInfo * 1000) % 1000);

        QTime totalTime((m_duration / 3600) % 60, (m_duration / 60) % 60,
                        m_duration % 60, (m_duration * 1000) % 1000);

        QString format = "mm:ss";

        if (m_duration > 3600)
            format = "hh:mm:ss";

        playTimeStr = currentTime.toString(format)
                     + " / " + totalTime.toString(format);
    }

    ui->labelPlayTime->setText(playTimeStr);
}
```

Jesus loves you.

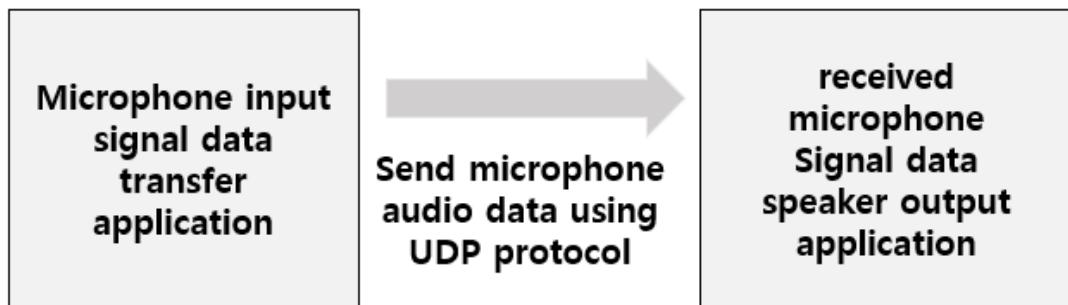
```
void Widget::seek(int seconds)
{
    m_player->setPosition(seconds * 1000);
}

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

The onPauseBtn() Slot function is a Slot function called by clicking the [PauseBtn] button. In this Slot function, pause if play is currently on. Conversely, if it is paused, it is replayed. The durationChanged() Slot function is called when the MP3 file is selected. When MP3 files are called here, the unit is Milliscond. Therefore, it was divided by 1000 to store it in seconds. The positionChanged() Slot function is constantly invoked with a period of time, as the position played during playback continues to change. This Slot function shows the time currently being played and the total replay time on the GUI.

- Implementing microphone input signal network send/receive examples

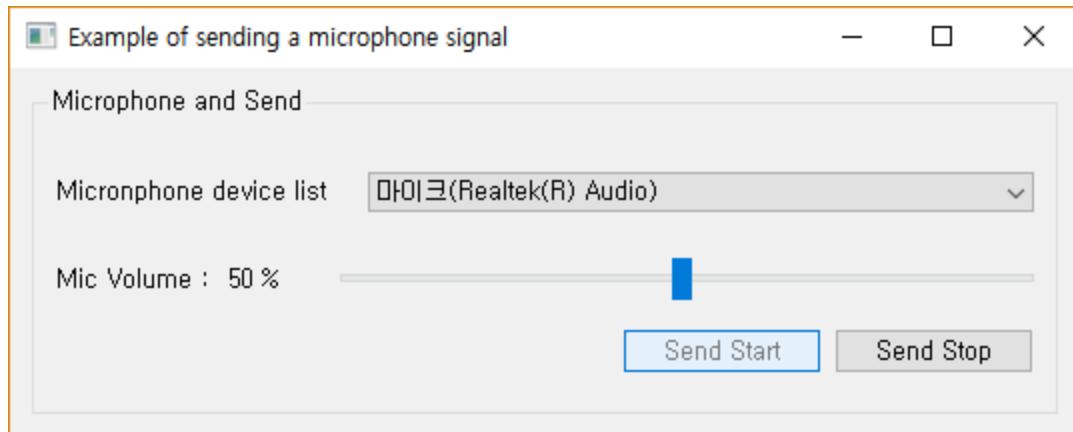
An example to be dealt with this time is an application that uses QAudioInput class to transmit micro-input signals to network UDP. The output is then exported from the received application to the speaker, which is the data sent to the network UDP protocol.



<FIGURE> Role of Example Applications

The transmission application uses the QAudioInput class to obtain signal data from the microphone and transmits the data using the network UDP protocol. Applications that receive microphone signal data over UDP will output to speakers using QAudioOutput class.

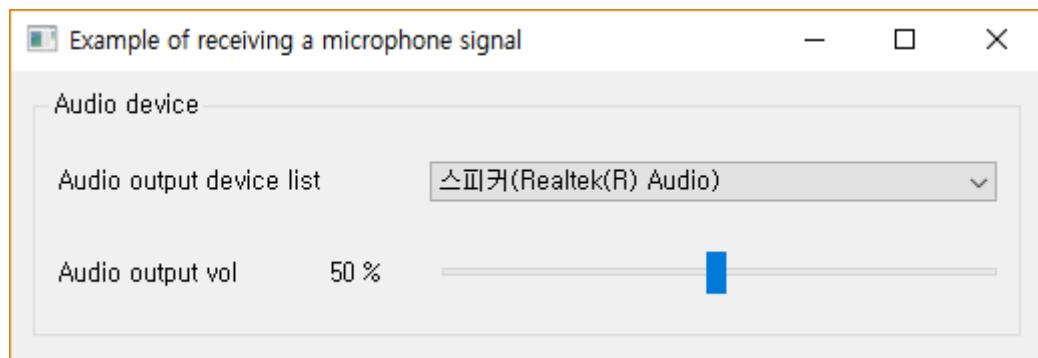
Jesus loves you.



<FIGURE> Example of sending a microphone signal

The above figure shows an example of a microphone signal transmission. The Microphone Devices list outputs a microphone device that is recognized by the current system.

Then click the [Send Start] button below to extract data from the microphone signal input from the user-selected microphone device and send it to the receiving application via UDP protocol.



<FIGURE> Example of receiving a microphone signal

The picture above shows the running screen of the microphone signal reception application. When the application runs, it outputs the microphone signal data received from the current combobox to the selected audio output device. If you select a different device from the output device list combobox in the GUI above, change the output to the selected device.

Let's take a look at some of the examples that we've seen so far, the ones that we've sent. The full example is below the source code Ch18 directory and the entire source

Jesus loves you.

code for the outgoing application can be referenced to the 01_AudioSender directory.
The following example source is the widget.h source code for the sending application.

Sender Exam – Ch18 > 01_AudioSender, Receiver Exam – Ch18 > 01_AudioReceiver

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QAudioInput>
#include <QUdpSocket>
#include <QHostAddress>
#include <QEapsedTimer>

namespace Ui { class Widget; }

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

    QList<QAudioDeviceInfo> devList;

    QAudioFormat      mFormat;
    QAudioDeviceInfo  mDevInfo;
    QAudioInput       *mAudioInput;
    QIODevice        *mInput;

    QByteArray        mBuffer;
    QEapsedTimer      mElapsedTimer;
    int               mInputVolume;

    void audioInitialize();
    QUdpSocket        mUdpSocket;

private slots:
    void volSliderChanged(int val);
```

Jesus loves you.

```
void sendStartBtn();
void sendStopBtn();

void notified();
void stateChanged(QAudio::State state);
void readMore();
};

#endif // WIDGET_H
```

In the example source above, QAudioFormat provides functions for setting the audio format, such as sample rate, channel, sample size, etc. The QAudioInput class provides the ability to extract signal data from microphone devices.

QIODevice is used to implement the function of converting microphone signals from the QAudioInput class into QByteArray. Therefore, the mBuffer variable stores data from the microphone device (QIODevice) signal. It then transmits this data to the network. The following is the widget.cpp source code for the transmission example.

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

Widget::Widget(QWidget *parent) : QWidget(parent), ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->volSlider,      &QSlider::valueChanged,
            this,                  &Widget::volSliderChanged);
    connect(ui->pbtSendStart,   &QPushButton::clicked,
            this,                  &Widget::sendStartBtn);
    connect(ui->pbtSendStop,    &QPushButton::clicked,
            this,                  &Widget::sendStopBtn);

    ui->volSlider->setEnabled(false);
    ui->pbtSendStop->setEnabled(false);
    audioInitialize();
}

void Widget::audioInitialize()
{
    devList = QAudioDeviceInfo::availableDevices(QAudio::AudioInput);
```

Jesus loves you.

```
for(int i = 0; i < devList.size(); ++i)
    ui->comboDevList->addItem(devList.at(i).deviceName());

mDeviceInfo = devList.at(ui->comboDevList->currentIndex());

mFormat.setSampleRate(8000);
mFormat.setChannelCount(1);
mFormat.setSampleSize(16);
mFormat.setSampleType(QAudioFormat::SignedInt);
mFormat.setByteOrder(QAudioFormat::LittleEndian);
mFormat.setCodec("audio/pcm");

mAudioInput = new QAudioInput(mDeviceInfo, mFormat, this);
mInputVolume = ui->volSlider->value();

ui->volLabel->setText(QString("%1 %").arg(mInputVolume));

mBuffer = QByteArray(14096, 0);
mElapsedTimer.start();
}

void Widget::volSliderChanged(int val)
{
    mInputVolume = val;
    ui->volLabel->setText(QString("%1 %").arg(mInputVolume));
    mAudioInput->setVolume(mInputVolume);
}

void Widget::sendStartBtn()
{
    mDeviceInfo = devList.at(ui->comboDevList->currentIndex());
    mAudioInput = new QAudioInput(mDeviceInfo, mFormat, this);
    connect(mAudioInput, SIGNAL(notify()),
            this,           SLOT(notified()));
    connect(mAudioInput, SIGNAL(stateChanged(QAudio::State)),
            this,           SLOT(stateChanged(QAudio::State)));

    mAudioInput->setVolume(qreal(mInputVolume) / 100);

    mInput = mAudioInput->start();
    connect(mInput, SIGNAL(readyRead()), this, SLOT(readMore()));
}
```

Jesus loves you.

```
{}

void Widget::sendStopBtn()
{
    if(mInput != nullptr) {
        disconnect(mInput, nullptr, this, nullptr);
        mInput = nullptr;
    }
    mAudioInput->stop();
    mAudioInput->disconnect(this);
    delete mAudioInput;
}

void Widget::notified()
{
    qDebug() << "elapsedUSecs=" << mAudioInput->elapsedUSecs()
           << "processUSecs=" << mAudioInput->processedUSecs();
}

void Widget::stateChanged(QAudio::State state)
{
    qDebug() << "state = " << state;
    if(state == QAudio::IdleState || state == QAudio::ActiveState) {
        ui->volSlider->setEnabled(true);
        ui->pbtSendStart->setEnabled(false);
        ui->pbtSendStop->setEnabled(true);
    }
    else if(state == QAudio::StoppedState)
    {
        ui->volSlider->setEnabled(false);
        ui->pbtSendStart->setEnabled(true);
        ui->pbtSendStop->setEnabled(false);
    }
}

void Widget::readMore()
{
    if(!mAudioInput) return;

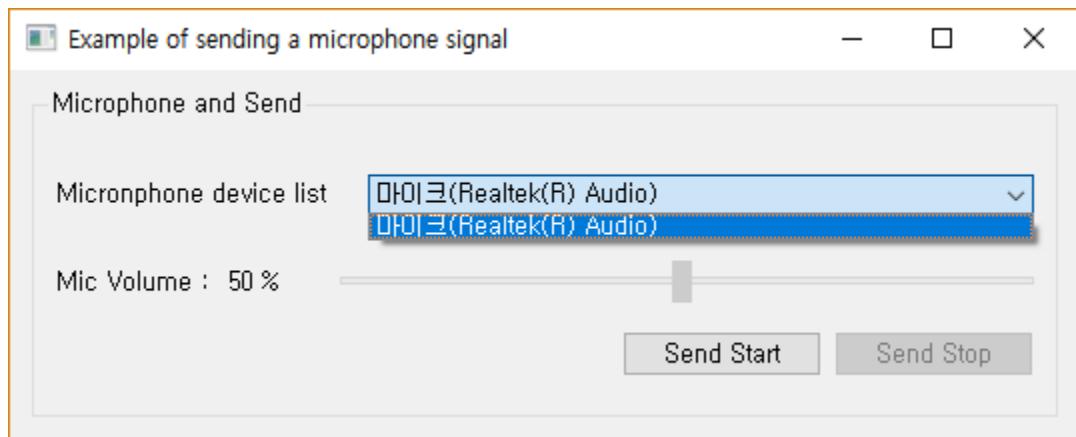
    qint64 len = mAudioInput->bytesReady();
    if(len > 4096)
        len = 4096;
```

Jesus loves you.

```
qint64 readLen = mInput->read(mBuffer.data(), len);
if(readLen > 0) {
    mUdpSocket.writeDatagram(mBuffer.data(), len, QHostAddress::LocalHost,
                             15000);
}
}

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

Link the GUI widget's Signal and Slot functions in the class constructor function as shown in the example source above. Then call the audioInitialize() function. In the audioInitialize() function, an input device such as a microphone is obtained from the system using the QAudioDeviceInfo class and registered in the combobox.



<FIGURE> Microphone device list

Then set the audio format and declare QAudioInput class. The first factor of the constructor during the QAudioInput class object declaration is the device of the selected item in the microphone device list combobox item in the GUI. The second factor passes over the QAudioFormat class object.

Click the Start Transfer button to start extracting the microphone signal using the start() function of the QAudioOutput class. The ReadMore() Slot function is called when the microphone signal is extracted. In this Slot function, the extracted data is stored as QByteArray and then transferred to UDP. The following is the source code for the receive

Jesus loves you.

application: widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QtMultimedia>
#include <QtNetwork>

namespace Ui { class Widget; }

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

private:
    Ui::Widget *ui;

    QList<QAudioDeviceInfo> devList;
    QAudioDeviceInfo mDeviceInfo;
    int mOutputVolume;
    QAudioFormat mFormat;
    QAudioOutput *mAudioOutput;
    QIODevice *mOutput;

    void audioInitialize();
    void audioOutputProcess(const QByteArray &ba);

    QUdpSocket *mUdpSocket;
    void networkInitialize();

private slots:
    void devListIndexChanged(int index);
    void volSliderChanged(int val);
    void readUdpData();

};

#endif // WIDGET_H
```

Jesus loves you.

The QAudioDeviceInfo class declares an object for an audio device that is outputable by the system.

It provides a function to export data received through UDP protocol to an output device that is selected by using QAudioOutput. Next is the widget.cpp source code.

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

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

    connect(ui->comboDevList, SIGNAL(currentIndexChanged(int)),
            this,                      SLOT(devListIndexChanged(int)));
    connect(ui->volSlider,      &QSlider::valueChanged,
            this,                      &Widget::volSliderChanged);
}

void Widget::audioInitialize()
{
    devList = QAudioDeviceInfo::availableDevices(QAudio::AudioOutput);
    for(int i = 0; i < devList.size(); ++i) {
        ui->comboDevList->addItem(devList.at(i).deviceName());

    mDevInfo = devList.at(ui->comboDevList->currentIndex());

    mFormat.setSampleRate(8000);
    mFormat.setChannelCount(1);
    mFormat.setSampleSize(16);
    mFormat.setSampleType(QAudioFormat::SignedInt);
    mFormat.setByteOrder(QAudioFormat::LittleEndian);
    mFormat.setCodec("audio/pcm");

    mAudioOutput = new QAudioOutput(mDevInfo, mFormat, this);
    mOutput = mAudioOutput->start();

    mOutputVolume = ui->volSlider->value();
    ui->volLabel->setText(QString("%1 %").arg(mOutputVolume));
}
```

Jesus loves you.

```
mAudioOutput->setVolume(mOutputVolume);
}

void Widget::networkInitialize()
{
    mUdpSocket = new QUdpSocket(this);
    mUdpSocket->bind(QHostAddress::LocalHost, 15000);
    connect(mUdpSocket, SIGNAL(readyRead()), this, SLOT(readUpdData()));
}

void Widget::devListIndexChanged(int index)
{
    mDevInfo = devList.at(index);
    if(mOutput != nullptr) {
        disconnect(mOutput, nullptr, this, nullptr);
        mOutput = nullptr;
    }

    mAudioOutput->stop();
    mAudioOutput->disconnect(this);
    delete mAudioOutput;

    mAudioOutput = new QAudioOutput(mDevInfo, mFormat, this);
    mOutput = mAudioOutput->start();
}

void Widget::volSliderChanged(int val)
{
    mOutputVolume = val;
    ui->volLabel->setText(QString("%1 %").arg(mOutputVolume));
    mAudioOutput->setVolume(mOutputVolume);
}

void Widget::readUpdData()
{
    while (mUdpSocket->hasPendingDatagrams())
    {
        QNetworkDatagram datagram = mUdpSocket->receiveDatagram();

        QByteArray audioData = datagram.data();
        audioOutputProcess(audioData);
    }
}
```

Jesus loves you.

```
}

void Widget::audioOutputProcess(const QByteArray &ba)
{
    qint64 len = ba.size();
    if(len > 0)
        mOutput->write(ba.data(), ba.size());
}

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

The start() member function of the QAudioOutput class starts the output. This function passes the pointer address of QIODevice on the output device.

Therefore, the mOutput function of QIODevice class stores the pointer passed over by the start() function and writes the data received from the actual microphone to the mOutput object and outputs it to the speaker device of the output audio device of the user's choice.

In addition, the networkInitialize() function called by the creator declares and initializes the objects of the QUdpSocket class to be received through the network UDP protocol.

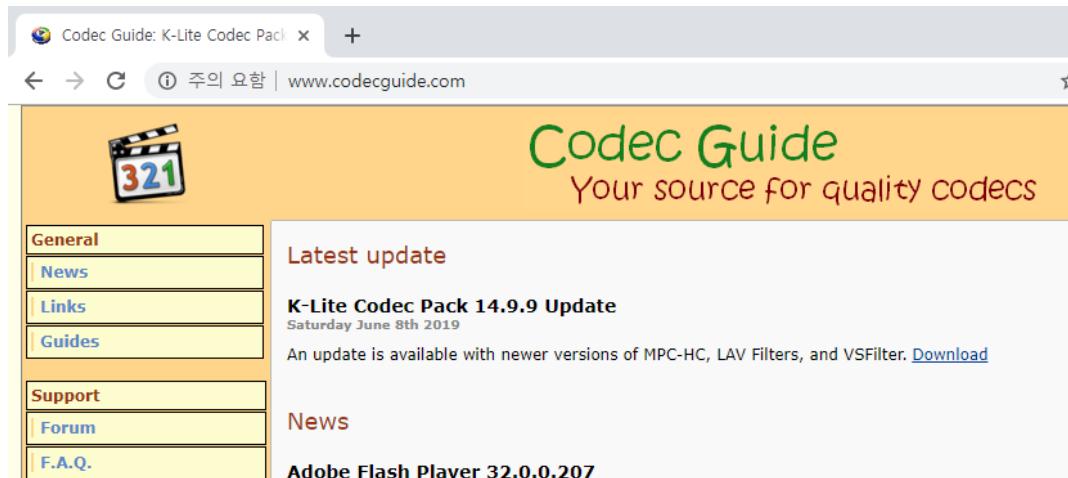
In this case, when you receive the received data signal and the settings for receiving the received data signal with Localhost and 15000 ports, you use the Connect function to connect the signal and slot so that the readUpdpData() Slot function is called.

When this Slot function is called, the received data is converted to QByteArrary and then write to the object in the QIODevice class. The output device currently selected in the GUI will then output the microphone signal data to the speakers.

18.2. Video

In order to play a video, compression codec is provided to decode the video encoded image file. Compression Codec has various compression Codecs such as H.264 and MOV. For example, in order to decode (play) video encoded with H.264 Codec, Codec that supports H.264 must be installed on the system. Qt does not provide various video codecs with Qt due to licensing issues.

The reason for this is that each Codec is associated with a development licence and cannot be provided together by Qt. Therefore, you must download and install the required Codec. Try the K-Lite Codec Pack as an integrated, easily accessible Codec. There are many kinds of K-Lite Codec Packets when you log on to the www.codecguide.com site. Among these, Basic, Standard, Full, etc. are provided, where the Full version is downloaded and installed.



<FIGURE> K-Lite Codec Pack Homepage

Although it is not a problem to implement video playback application using Qt multimedia module, it will not be played if Codec used by the video you want to play is not installed. Therefore, make sure to install Codec as shown in the picture above.

Suppose you use a multimedia module provided by Qt to implement a video player. In order to implement the video player provided by Qt, two classes must be used: The first is QMiaPlayer. MP3 files, such as music files, do not need to print out the video footage.

Jesus loves you.

So it can be played using the QMediaPlayer class. But the video includes the video along with the song. For this reason, video should be displayed and Qt multimedia should use the QVideoWidget class.

That is, you can decode the music and images of a movie with the QMediaPlayer class. If you need a GUI widget to print on the screen of a video, you can implement a video player by using QVideoWidget, which is provided by the QtMultimedia module. The following example sources are examples of using the QMediaPlayer and QVideoWidget classes.

```
player = new QMediaPlayer;
videoWidget = new QVideoWidget;
player->setVideoOutput(videoWidget);

player->setMedia(QUrl("http://example.com/movie1.mp4"));
//player->setMedia(QUrl::fromLocalFile("/My/ movie2.mp4"));

videoWidget->show();
player->play();
```

To set up widgets for video output, as shown in the example source code above, you can set up the video display widget using the setVideoOutput() member function provided by the QMediaPlayer class.

In addition to the QVideoWidget class, the QGraphicsVideoItem class can be used as a widget to output videos. As one of the items in the Graphics View Framework previously covered, this class can use items that can play videos in the QGraphicsView area.

```
player = new QMediaPlayer(this);

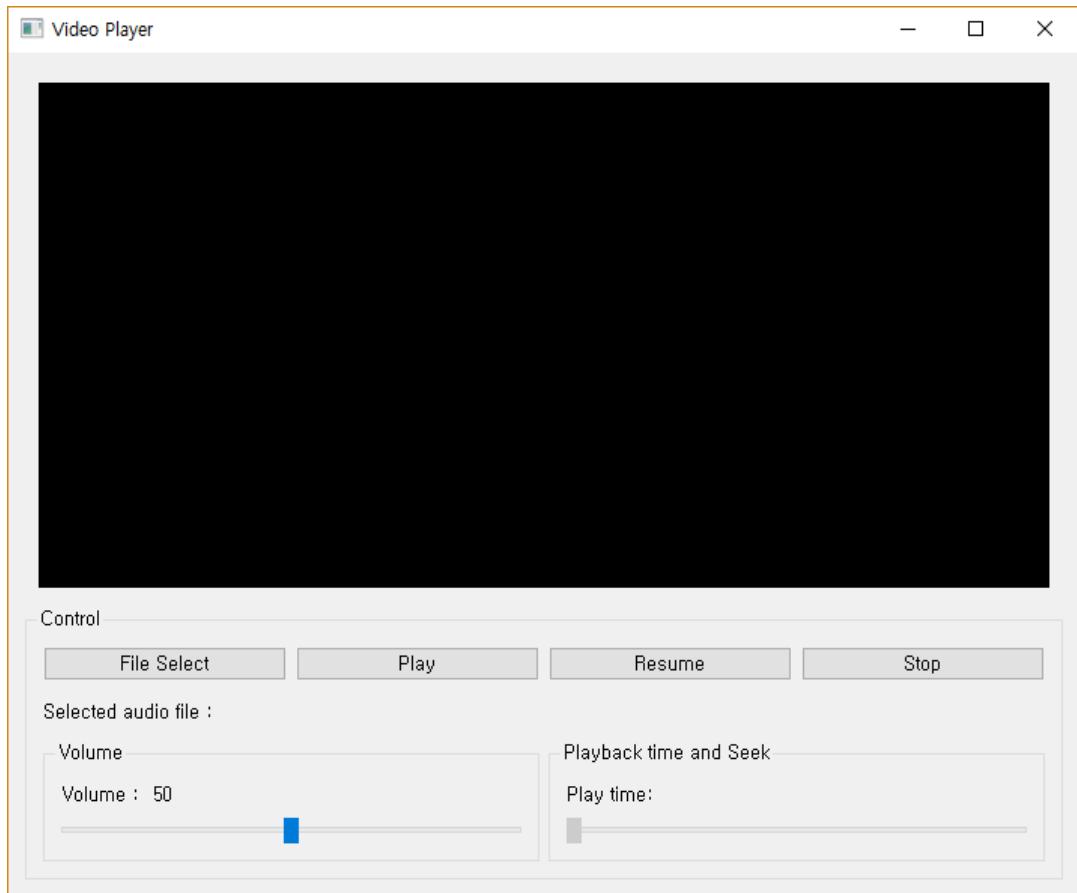
QGraphicsVideoItem *item = new QGraphicsVideoItem;
player->setVideoOutput(item);
graphicsView->scene()->addItem(item);
graphicsView->show();

player->setMedia(QUrl("http://example.com/my.mp4"));
player->play();
```

Next, let's try implementing a video player using the QMediaPlayer class and QVideoWidget class.

Jesus loves you.

- Implementing a video player



<FIGURE> 예제 실행 화면

QVideoWidget was used to print videos on GUI and see above. In addition, double-clicking on the part where the video is printed changes the area where the video is printed to the full screen.

In the full screen state, click ESC to return to the original size screen. To implement these functions, the DisplayWidget class that inherits the QVideoWidget class was implemented. The following example source code is the header source code for the DisplayWidget class. Example - Ch18 > 02_VideoPlayer

```
#ifndef DISPLAYWIDGET_H
#define DISPLAYWIDGET_H

#include <QVideoWidget>

class DisplayWidget : public QVideoWidget
```

Jesus loves you.

```
{  
    Q_OBJECT  
public:  
    explicit DisplayWidget(QWidget *parent = nullptr);  
protected:  
    void keyPressEvent(QKeyEvent *event) override;  
    void mouseDoubleClickEvent(QMouseEvent *event) override;  
    void mousePressEvent(QMouseEvent *event) override;  
};  
#endif // DISPLAYWIDGET_H
```

The function used in the protected keyword is a Virtual function. The mouseDoubleClickEvent() Virtual function is invoked when you double-click it. In this function, when you double-click the mouse, the movie rendering widget switches to the full screen. The following example source code is the DisplayWidget class source code.

```
#include "displaywidget.h"  
#include <QKeyEvent>  
#include <QMouseEvent>  
  
DisplayWidget::DisplayWidget(QWidget *parent) : QVideoWidget(parent)  
{  
    setSizePolicy(QSizePolicy::Ignored, QSizePolicy::Ignored);  
  
    QPalette p = palette();  
    p.setColor(QPalette::Window, Qt::black);  
    setPalette(p);  
  
    setAttribute(Qt::WA_OpaquePaintEvent);  
}  
  
void DisplayWidget::keyPressEvent(QKeyEvent *event)  
{  
    if (event->key() == Qt::Key_Escape && isFullScreen())  
    {  
        setFullScreen(false);  
        event->accept();  
    }  
    else if (event->key() == Qt::Key_Enter && event->modifiers() & Qt::Key_Alt)  
    {  
        setFullScreen(!isFullScreen());  
        event->accept();  
    }  
}
```

Jesus loves you.

```
    }
    else {
        QVideoWidget::keyPressEvent(event);
    }
}

void DisplayWidget::mouseDoubleClickEvent(QMouseEvent *event)
{
    setFullScreen(!isFullScreen());
    event->accept();
}

void DisplayWidget::mousePressEvent(QMouseEvent *event)
{
    QVideoWidget::mousePressEvent(event);
}
```

keyPressEvent() is called when a keyboard event occurs and switches to the original screen size when the ESC (Escape) key or ALT + Enter key is clicked.

For the following example source code, consider using the DisplayWidget class to explore the Widget class using. The following example is the widget.h source code.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QMediaPlayer>
#include "displaywidget.h"

namespace Ui { class Widget; }

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

private:
    Ui::Widget *ui;
    QString      m_fName;
    QMediaPlayer *m_player;
```

Jesus loves you.

```
DisplayWidget *m_displayWidget;
qint64 m_duration;

private slots:
    void onOpenBtn();
    void onPlayBtn();
    void onPauseBtn();
    void onStopBtn();

    void sliderValueChange(int val);
    void durationChanged(qint64 duration);
    void positionChanged(qint64 progress);
    void seek(int seconds);
};

#endif // WIDGET_H
```

onOpenBtn() is a Slot function called by clicking the Select File button. In this function, you can use the QFileDialog class to select a video file during a file selection dialogue.

The onPlayBtn() function plays video files. Use the setMedia() function of the QMediaPlayer class to set up the files to play.

It also plays video files using the play() function. onPauseBtn() provides the ability to pause a video playback day, while onStopBtn() provides the ability to stop a video. The following example source code is the widget.cpp source code.

```
#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()));
```

Jesus loves you.

```
connect(ui->pbtStop, SIGNAL(clicked()), this, SLOT(onStopBtn()));

m_displayWidget = new DisplayWidget();

QVBoxLayout *vLay = new QVBoxLayout();
vLay->addWidget(m_displayWidget);
ui->containerWidget->setLayout(vLay);

m_player = new QMediaPlayer();
m_player->setVideoOutput(m_displayWidget);
connect(m_player,           &QMediaPlayer::durationChanged,
        this,                 &Widget::durationChanged);
connect(m_player,           &QMediaPlayer::positionChanged,
        this,                 &Widget::positionChanged);
connect(ui->sliderPosition, &QSlider::sliderMoved,
        this,                 &Widget::seek);
}

void Widget::sliderValueChange(int val)
{
    ui->labelVolume->setText(QString("%1").arg(val));
    m_player->setVolume(val);
}

void Widget::onOpenBtn()
{
    m_fName = QFileDialog::getOpenFileName(this,
                                           tr("Open File"), QDir::homePath());
    if(!m_fName.isNull())
        ui->lblFileName->setText(m_fName);
}

void Widget::onPlayBtn()
{
    if(!m_fName.isNull())
    {
        m_player->setMedia(QUrl::fromLocalFile(m_fName));
        ui->sliderPosition->setEnabled(true);
        m_player->play();
    }
}
```

Jesus loves you.

```
void Widget::onPauseBtn()
{
    int state = m_player->state();
    if(state == QMediaPlayer::PausedState) {
        m_player->play();
    }
    else if(state == QMediaPlayer::PlayingState) {
        m_player->pause();
    }
}

void Widget::onStopBtn()
{
    int state = m_player->state();
    if(state == QMediaPlayer::PlayingState) {
        m_player->stop();
    }
}

void Widget::durationChanged(qint64 duration)
{
    m_duration = duration / 1000;
    ui->sliderPosition->setMaximum(m_duration);
}

void Widget::positionChanged(qint64 progress)
{
    if (!ui->sliderPosition->isSliderDown())
        ui->sliderPosition->setValue(progress / 1000);

    qint64 currentInfo = progress / 1000;
    QString playTimeStr;
    if (currentInfo || m_duration)
    {
        QTime currentTime((currentInfo / 3600) % 60, (currentInfo / 60) % 60,
                          currentInfo % 60, (currentInfo * 1000) % 1000);

        QTime totalTime((m_duration / 3600) % 60, (m_duration / 60) % 60,
                        m_duration % 60, (m_duration * 1000) % 1000);

        QString format = "mm:ss";
        if (m_duration > 3600) format = "hh:mm:ss";
    }
}
```

Jesus loves you.

```
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;
}
```

In the class creator, place DisplayWidget on the GUI and hand over the object in the DisplayWidget class as the first factor in the setVideoOutput() function of the QMediaPlayer class to play the movie. When QMediaPlayer plays a video, it outputs a video screen to the DisplayWidget class widget.

sliderValueChange() is invoked when the value of the volume QSlider widget changes over the GUI. This Slot function allows you to adjust the volume when playing a movie using the setVolume() function in the QMediaPlayer class.

The durationChanged() function is invoked once the video file is selected and played normally by the QMediaPlayer . This function tells the replay time in Millisecond units. positionChanged() provides a function that tells you where you are currently playing. This Slot function changes the overall time to "hour:minutes:seconds" and outputs it to the GUI widget.

18.3. Camera

It provides various camera APIs that can be developed using cameras in Qt multimedia module. The QCamera class, most commonly used among the various classes, can be developed as an application using physical camera hardware devices.

QCamera class obtains image images from camera devices. Then, for output to the GUI, the QCameraViewfinder class can be used to output to the GUI. The following example source code is a method for using camera devices using QCamera class and QCameraViewfinder.

```
QMcamea *camera = new QCamera;  
viewfinder = new QCameraViewfinder;  
  
camera->setViewfinder(viewfinder);  
viewfinder->show();  
  
camera->start();
```

Outputs images obtained by QCamera from the camera device in the QCameraViewfinder class widget area. Specify the QCameraViewfinder class object as the first factor in the setViewfinder() member function provided by the QCamera class, as shown in the example source code above.

It also uses the start() member function of QCamera class to obtain images from actual camera devices. In addition, the QCamera class and QCameralImageCapture classes allow users to capture the desired camera images with images. The following example is an example for implementing a capture function.

```
imageCapture = new QCameraImageCapture(camera);  
camera->setCaptureMode(QCamera::CaptureStillImage);  
camera->start();  
...  
camera->searchAndLock();  
imageCapture->capture();  
camera->unlock();  
...
```

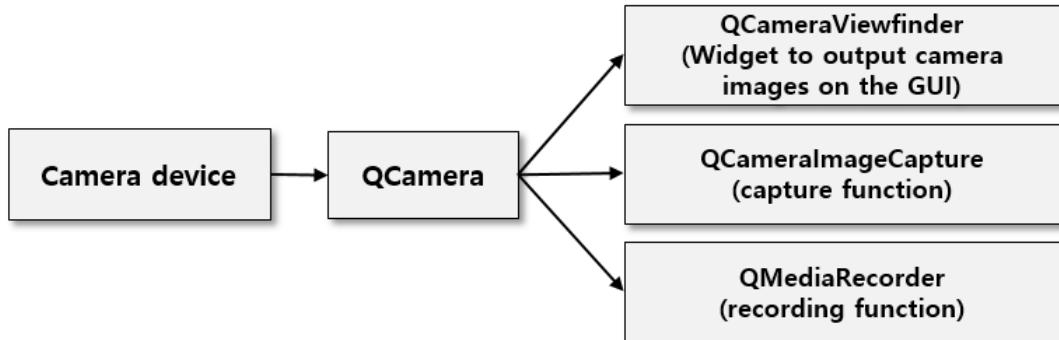
Jesus loves you.

The QCamera class also provides a QMediaRecorder class to record images from the camera.

Like the QCameralmageCapture class, the QMiaRecorder class does not import capture images directly from the camera device, and the QMediaRecorder class can use the QCamera class as an interface to record images obtained from the camera.

```
camera = new QCamera;  
recorder = new QMediaRecorder(camera);  
  
camera->setCaptureMode(QCamera::CaptureVideo);  
camera->start();  
  
recorder->record();  
...  
recorder->stop();
```

We have obtained the camera images we have seen so far and looked at how to display them on the GUI, how to capture camera images, and how to record them. Therefore, the QCamera class provides an interface for handling images from physical camera hardware devices.



<FIGURE> QCamera Class Relationship

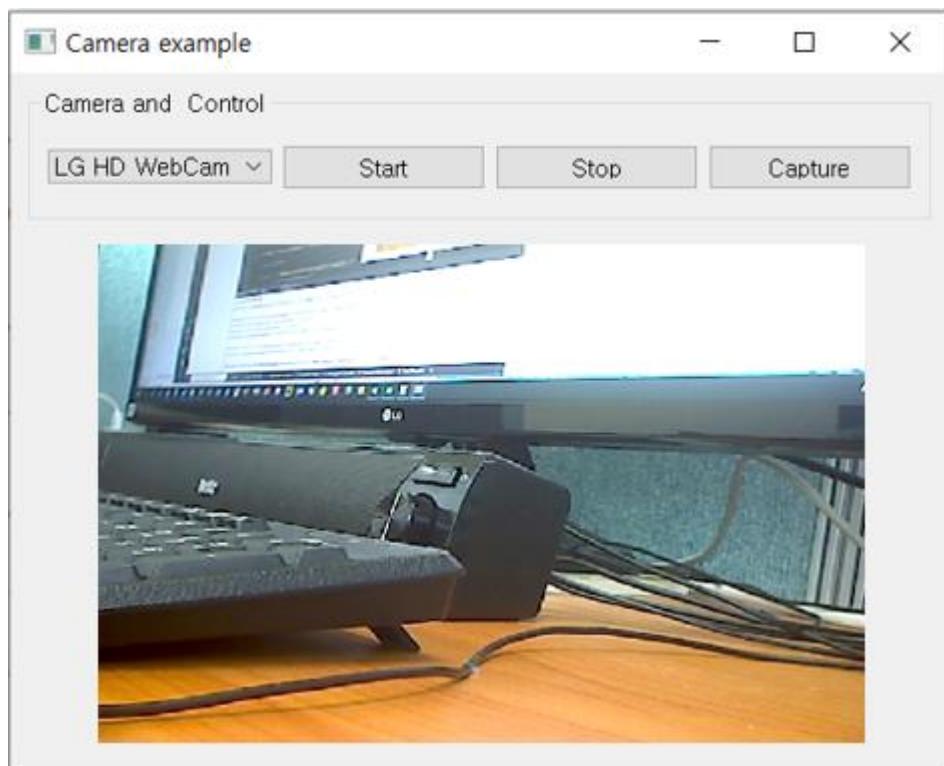
QCameraFocus to handle camera focus, QCameraExposure class to set up the camera's position, etc. So far, we have looked at ways to handle cameras. Here's an example of how to handle images from a real camera device.

- Implementing an example using a camera device

This example outputs camera images from a USB camera device to the GUI. Let's

Jesus loves you.

implement additional functions that capture camera images with additional functions. The following figure shows an example run screen.



<FIGURE> Example screen

As shown in the figure above, the QComboBox widget registers all camera device devices retrieved from the system. Click the [Start] button to output the image from the camera onto the GUI. The [Stop] button stops importing camera images.

The [Capture] button then captures the currently output image as an image. The following is the header of the example, widget.h.

Example - Ch18 > 03_CameraCapture

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QCamera>
#include <QCameraViewfinder>
#include <QCameraImageCapture>

namespace Ui { class Widget; }
```

Jesus loves you.

```
class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
    void initCamera();
    void cameraDevicesSearch();

private:
    Ui::Widget *ui;
    QCamera           *mCamera;
    QCameraViewfinder *mViewfinder;
    QCameraImageCapture *mCapture;
    QList<QByteArray>   camDevNameLists;

private slots:
    void onStartBtn();
    void onStopBtn();
    void onCaptureBtn();
    void camError(QCamera::Error error);
    void imageCaptured(int pId, QImage pPreview);
};

#endif // WIDGET_H
```

In the example above, the `initCamera()` function initializes the `QCamera` and `QCameraImageCapture` classes. In addition, place the `QCameraImageCapture` class widget on the GUI.

The `cameraDevicesSearch()` function registers all camera devices found in the system with the widget in the `QComboBox` class.

The `onStartBtn()` function is invoked when the [Start] button is clicked. The `onStopBtn()` function is called by clicking the [Stop] button and the `onCaptureBtn()` function is called by clicking the [Capture] button. The `camError()` function is invoked when an error signal occurs from the camera.

The last `imageCaptured()` function is then invoked when the camera capture function is performed. The following example is the `widget.cpp` source code.

Jesus loves you.

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

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

    connect(ui->pbtStart,    SIGNAL(clicked()), this, SLOT(onStartBtn()));
    connect(ui->pbtStop,     SIGNAL(clicked()), this, SLOT(onStopBtn()));
    connect(ui->pbtCapture,  SIGNAL(clicked()), this, SLOT(onCaptureBtn()));
}

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

void Widget::initCamera()
{
    mCamera      = new QCamera;
    mCapture     = new QCameraImageCapture(mCamera);
    mViewfinder  = new QCameraViewfinder();

    QVBoxLayout *vLay = new QVBoxLayout();
    vLay->addWidget(mViewfinder);
    ui->camWidget->setLayout(vLay);
}

void Widget::cameraDevicesSearch()
{
    camDevNameLists.clear();
    ui->comboBox->clear();

    camDevNameLists.clear();
    ui->comboBox->clear();

    foreach (const QByteArray &deviceName, QCamera::availableDevices())
    {
```

Jesus loves you.

```
QString description;
description = mCamera->deviceDescription(deviceName);
camDevNameLists.append(deviceName);
ui->comboBox->addItem(description);
}

}

void Widget::onStartBtn()
{
    delete mCapture;
    delete mCamera;

    if(camDevNameLists.count() < 1) {
        mCamera = new QCamera;
    }else{
        int curIndex = ui->comboBox->currentIndex();
        mCamera = new QCamera(camDevNameLists.at(curIndex));
    }

    connect(mCamera, SIGNAL(error(QCamera::Error)),
            this, SLOT(camError(QCamera::Error)));

    mCamera->setViewfinder(mViewfinder);
    mCamera->setCaptureMode(QCamera::CaptureVideo);
    mCapture = new QCameraImageCapture(mCamera);

    //imageCapture->setCaptureDestination(
    //    QCameraImageCapture::CaptureToBuffer);

    mCapture->setCaptureDestination(
        QCameraImageCapture::CaptureToBuffer |
        QCameraImageCapture::CaptureToFile);

    mCapture->setBufferFormat(QVideoFrame::Format_RGB32);
    connect(mCapture, SIGNAL(imageCaptured(int,QImage)),
            this, SLOT(imageCaptured(int,QImage)));

    mCamera->start();
}

void Widget::onStopBtn()
{
```

Jesus loves you.

```
mCamera->stop();
}

void Widget::onCaptureBtn()
{
    mCapture->capture("c:/CaptuerImage.jpg");
}

void Widget::camError(QCamera::Error error)
{
    qDebug() << Q_FUNC_INFO << "Error : " << error;
}

void Widget::imageCaptured(int pId, QImage pPreview)
{
    Q_UNUSED(pId);

    qDebug() << "IMAGE CAPTUE SIZE (WIDTH X HEIGHT) : "
           << pPreview.byteCount();
}
```

Suppose there are several camera devices in the system. If there are multiple cameras, several camera devices will be registered on the combo box on the GUI. Select the camera device to be used in the combobox and press the [Start] button to recall the onStartBtn() Slot function.

The QCamera class and QCameredImageCapture class were initialized and declared in the onStartBtn() Slot function.

19. Qt Install Framework

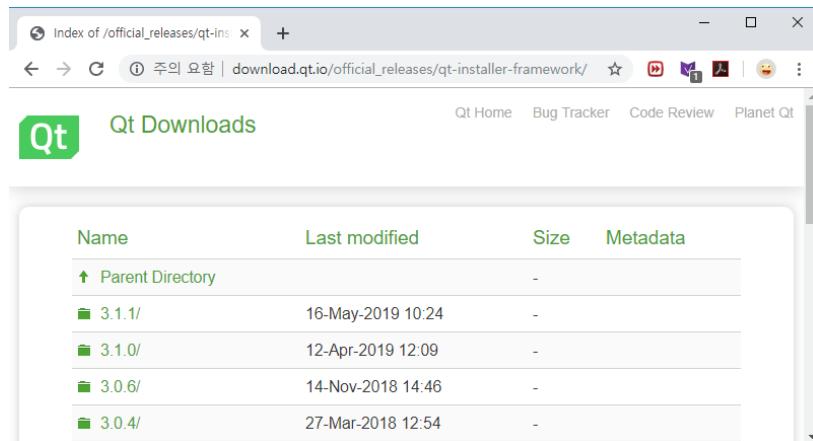
The Qt Install Framework provides the ability to create an installation SDK or an installation distribution plate so that users can install user-implemented applications. In addition to the installation files, libraries and various resource files used by the installation files can be produced and provided to users with a single installation distribution version.

For example, you might use the Qt Install Framework, such as providing an installation distribution version of an application implemented using the Install Shield in MS Windows. Qt Install Framework supports multiple platforms with Qt and Marge. The Qt Install Framework can create installation files using the same Qt Install Framework method, whether you are using Linux, MS, or MacOS.

- ✓ Qt Install framework download and install

You can download the Qt Install Framework from the following site using the web browser:

- http://download.qt.io/official_releases/qt-installer-framework

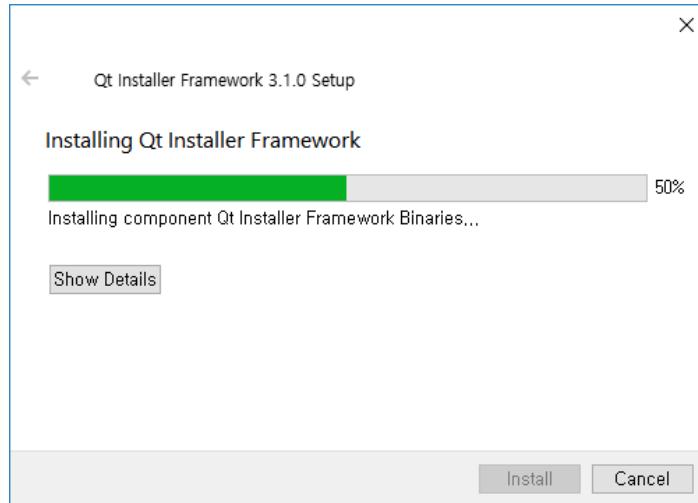


<FIGURE> Qt Install Framework download URL

Each version directory is provided as shown in the figure above. Click on the desired version to provide the Qt Install Framework for each operating system platform. Let's try the MS Window version here.

Jesus loves you.

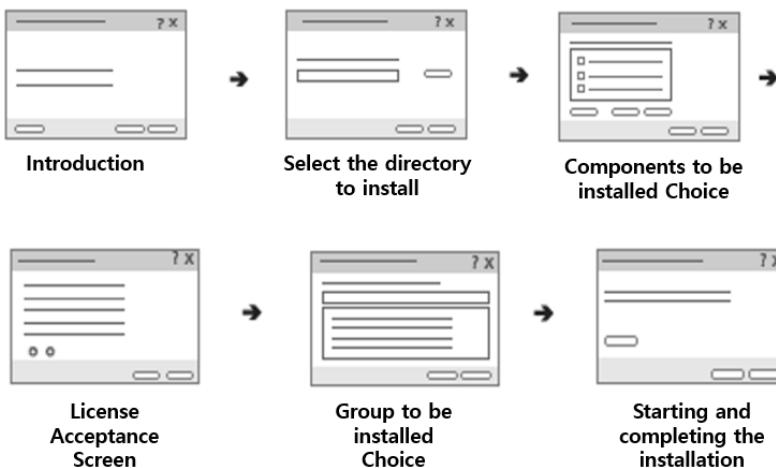
Download and install the QtInstallerFramework-win-x86.exe version, as shown in the following figure.



<FIGURE> Qt Install Framework install screen

- ✓ How to deploy using Qt Install framework

The Qt Install Framework provides an online and offline approach. The online method automatically downloads and installs by connecting to a specific Web server repository. The Offline method downloads and installs the installation files directly. In addition, distribution boards can be created in the form of a dialogue-type Setup Wizard as a way to distribute applications.



<FIGURE> Wizard Setup Dialog Example

As shown in the figure above, the installation file can be created using the Qt Install

Jesus loves you.

Framework in the form of a workflow-type template. Let's use the following example to create an installation distribution version.

- Create an Offline Installation Distribution with Qt Install Framework

Create a directory called 01_Installer_Example as shown in the example source code below. Then write the 01_Installer_Example.pro file in the directory you created as follows:

```
TEMPLATE = aux

INSTALLER = 01_Installer_Example
INPUT = $$PWD/config/config.xml $$PWD/packages

myexam.input = INPUT
myexam.output = $$INSTALLER

myexam.commands = C:/Qt/QtIFW-3.1.0/bin/binarycreator \
                  -c $$PWD/config/config.xml \
                  -p $$PWD/packages ${QMAKE_FILE_OUT}

myexam.CONFIG += target_predeps no_link combine

QMAKE_EXTRA_COMPILERS += myexam
```

The TEMPLATE keyword is specified to distinguish if the installation file is a library-recognized application. If you type library lib. If it is an application, type aux. This example is to deploy an application, so type aux as shown in the example above.

The INSTALLER keyword specifies the name of the installation file. When you build later, the name of the installation file that you create is created with the keyword INSTALLER.

The following INPUT keywords should specify two items: First, enter the config.xml file. The second entry specifies the packages directory.

The config.xml file specifies the project's name, version, provider, or, if MS Windows, the names of the Start menu.

The packages directory is the directory where the installed executable files, the developed application executable files and libraries, and license information, file information, and installation scripts are located during installation. Myexam.input keyword specifies the INPUT keyword. The second myexam.output specifies the installation file name specified

Jesus loves you.

in the INSTALLER keyword. To create an installation file, myexam.commands enters the binarycreator provided by the Qt Install Framework and specifies the location and name of the config.xml file for the –c option. And –p enters the package.

If you have completed creating the project file as shown above, create the config directory. Then, create the config.xml in the config directory as shown in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Installer>
    <Name>Example Software</Name>
    <Version>1.0.0</Version>
    <Title>Example Software</Title>
    <Publisher>Qt Developer</Publisher>
    <StartMenuDir>Qt Developer Menu</StartMenuDir>
    <TargetDir>C:/Example_Software</TargetDir>
</Installer>
```

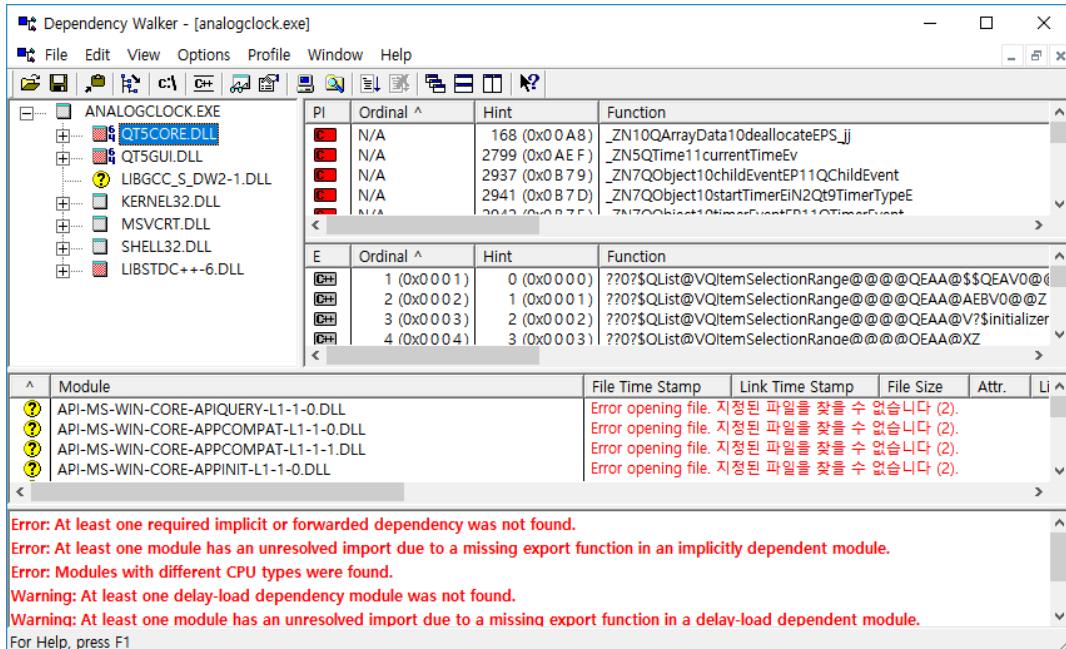
Create the packages directory in the project's top directory and create the com.vendor.product directory inside the packages directory. Then, create the data and meta directory in the com.vendor.product directory. The data directory copies the executable file of the application to be installed, the library that the executable file uses, and the resource file that the executable file uses.

Here we will build the analog example executable file of the Qt example in MinGW 32 Bit Release mode and put it in the data directory. In addition to this example, you can copy other executables and libraries.

So far, we've built and run the Qt Creator IDE tool, but this time let's run the executable that we've built. This will result in an error that says there is no library used by the executable file. The Qt Creator IDE tool automatically locates the library location, but when you run it yourself, the library must also exist in the location where the executable file is located.

If you are wondering which libraries you need, you can use the Dependency Walker to see a list of libraries that the executable file references, as shown in the figure below. Dependency Walker is free to download from the Internet.

Jesus loves you.



<FIGURE> Dependency Walker

If it is Linux, you can check the library that the executable file references with the ldd.

```
# ldd /usr/sbin/qtApplication
    linux-vdso.so.1 => (0x00007ffff85bff000)
    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)
    ...
```

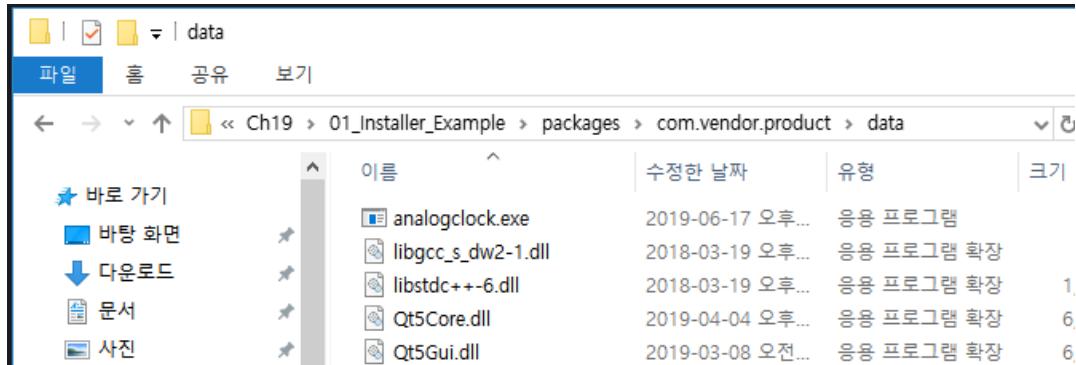
If your operating system uses the MacOS Ramen otool command, you can check the reference library for your deployment application as follows.

```
otool -L MyApp.app/Contents/MacOS/MyApp
```

Since we have built the example of analogclock provided by Qt into a MinGW 32 Bit version compiler, if you look at the bin directory by each compiler at the bottom of the Qt installation directory, there will be the necessary files.

Copy to the corresponding compiler subbin directory the library that requires the analogclock example executable file to the same location.

Jesus loves you.



<FIGURE> Required libraries and executables

Then, create the Meta directory under the com.vendor.product directory and create the file as follows.

<TABLE> Files Required for Creating an Installation Distribution

File name	Explanation
package.xml	package settings file
license.txt	License Information File
installscript.qs	installation script

The package.xml file specifies the name to display, the description, the name of the file with the license information to be displayed in the installation dialogue, and the script file required for the installation.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package>
    <DisplayName>Example Software</DisplayName>
    <Description>Example Software - Analog clock</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 specifies the location of the Short Cut menu for the package to be installed, the Ink file on the Start menu, and the Run icon. Write Installscript.qs as follows.

Jesus loves you.

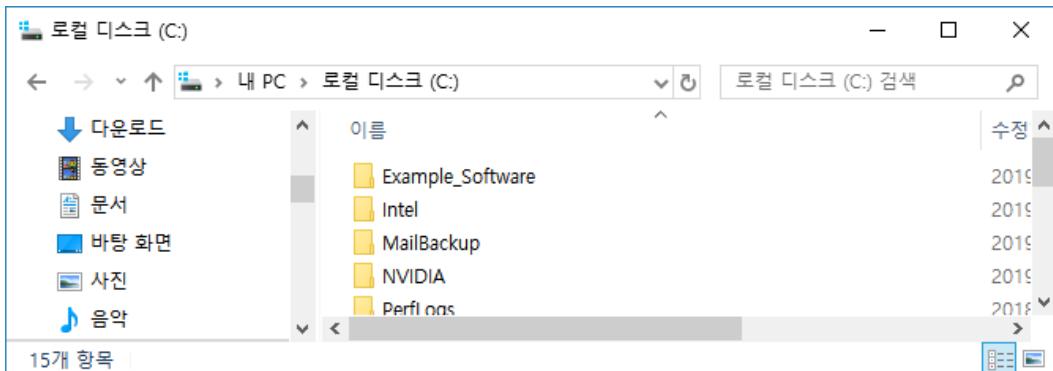
```
function Component()
{
    // default constructor
}

Component.prototype.createOperations = function()
{
    component.createOperations();

    if (systemInfo.productType === "windows")
    {
        component.addOperation(
            "CreateShortcut",
            "@TargetDir@/analogclock",
            "@StartMenuDir@/analogclock.lnk",
            "workingDirectory=@TargetDir@",
            "iconPath=@TargetDir@/main_icon.ico");
    }
}
```

If you have completed the above, you are ready to create the installation distribution. When you build from the Qt Creator tool, you can verify that the 01_Installer_Example.exe installation distribution file has been created in the build directory.

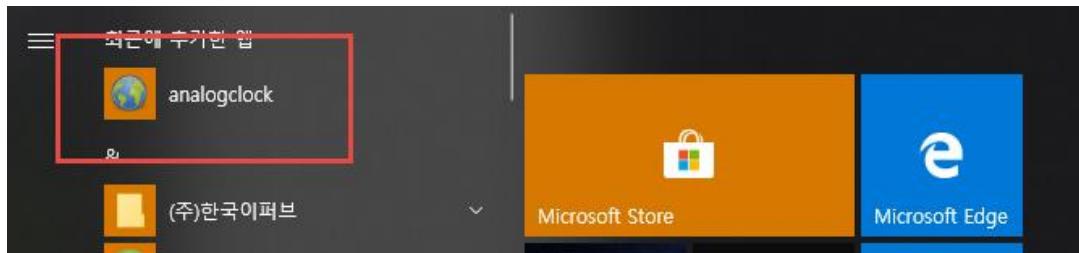
After installing the 01_Installer_Example.exe installation, check the config.xml file to see if the directory specified by the tag TargetDir is created and required.



<FIGURE> Example_Software directory

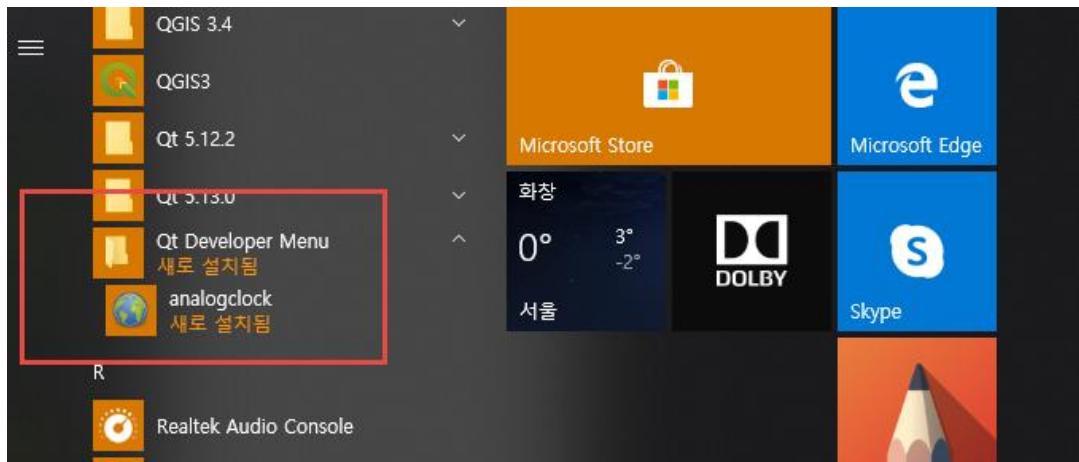
Then check to see if the program you installed on the Start menu is registered as shown in the picture below.

Jesus loves you.



<FIGURE> Analogclock Menu on the Start Menu

You can also see that the Start menu group is registered as specified in the config.xml file, as shown in the figure below.



<FIGURE> Start Menu Group

Example - Ch19 > 01_Installer_Example

20. Qt for Android

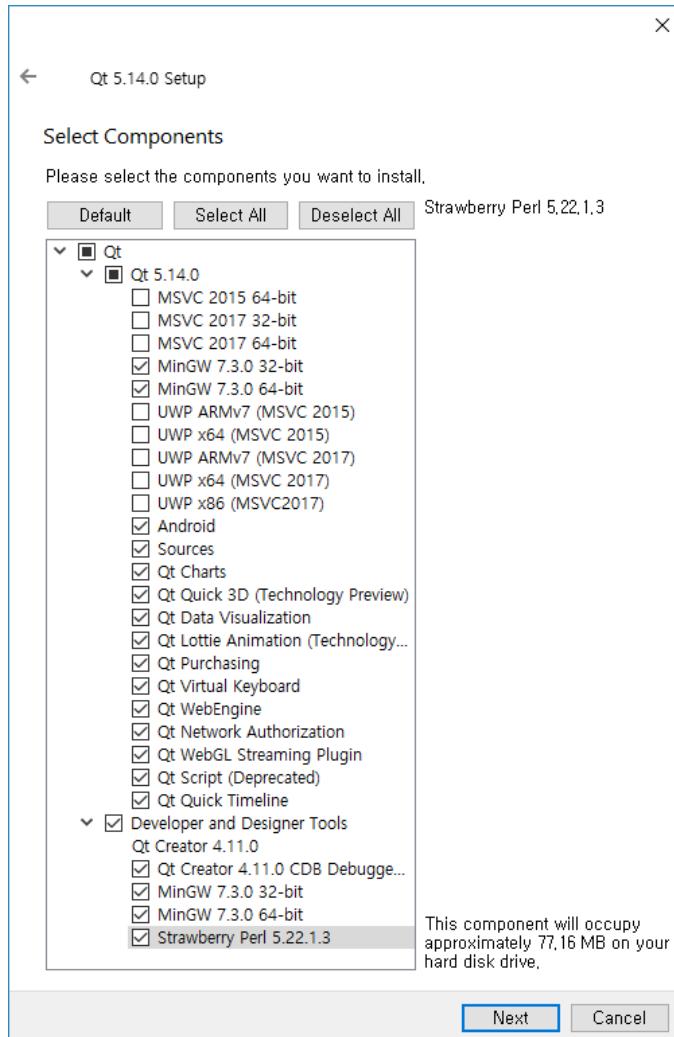
In this chapter, we will explore how to develop Android applications using Qt. This chapter consists of two units.

The first section deals with how to handle Android applications using Qt in the MS Windows operating system. In this section, you will learn how to build an environment on the MS Windows operating system. Then use Qt to develop Android application applications and install and run APKs on Android smartphones.

The second section covers how to handle Android applications using Qt on Linux. This section looks at how to build an environment on a Linux operating system. And let's use Qt to develop Android applications and install and run APKs on Android smartphones.

20.1. Deploy Android apps using Qt in MS Windows

We will use the 64-bit version of the MS Windows 10 operating system to develop the Android App. And Qt will use 5.14.0 version. Select Components as shown in the figure below.



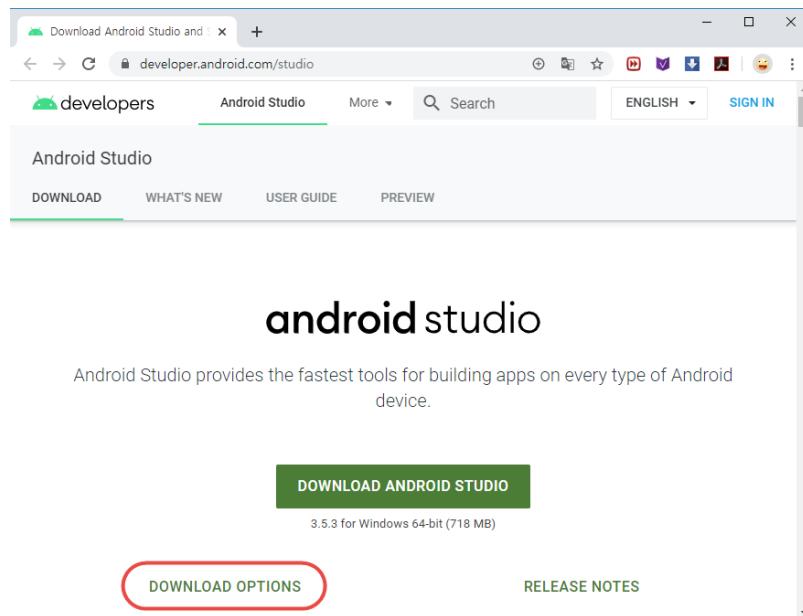
<FIGURE> Components select screen

Select Components as shown above and click the [Next] button at the bottom. Next, when the Qt installation is complete, the required software must be downloaded as shown in the table below.

Jesus loves you.

- Android SDK Tools download and install, URL - <https://developer.android.com/studio>

If the Qt version uses a version of 5.9 or lower, the SDK Tools package v25.2.5 or lower should be used. You can download the SDK by accessing the URL address below using a web browser. When you access the URL address above, click [DOWNLOAD OPTIONS] as shown in the figure below.



<FIGURE> SDK Tools download screen

Download the [Windows] entry from the [Command line tools only] item as shown in the figure below. Download and create a directory called "Qt_For_Android" as shown in the following figure. And unzip this directory.

When decompression is released, [tools] > [bin] has a file named sdkmanager.bat under the decompressed directory as shown in the figure below. Update the SDK tools package with the Command window as follows.

```
 sdkmanager.bat --update
```

The default USB driver for Windows does not allow debugging using the Android Debug Bridge (ADB) tool. Therefore, additional USB drivers provided by Android SDK packages must be installed. The following commands should be used to install additional packages,

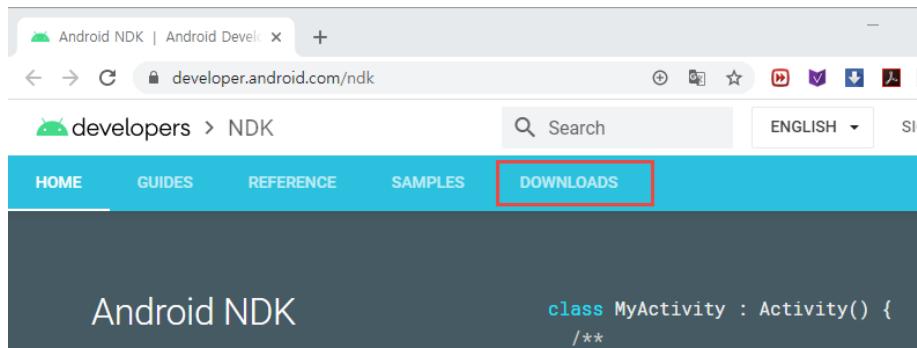
Jesus loves you.

as shown in the example below.

```
 sdkmanager.bat "extras;google;usb_driver"
```

- Android NDK download and install (URL - <https://developer.android.com/ndk>)

Access the above URL address and download NDK. Click the [DOWNLOADS] link in the menu when loading the webpage as shown in the figure below.



<FIGURE> NDK download screen

Next, download the latest version of NDK.

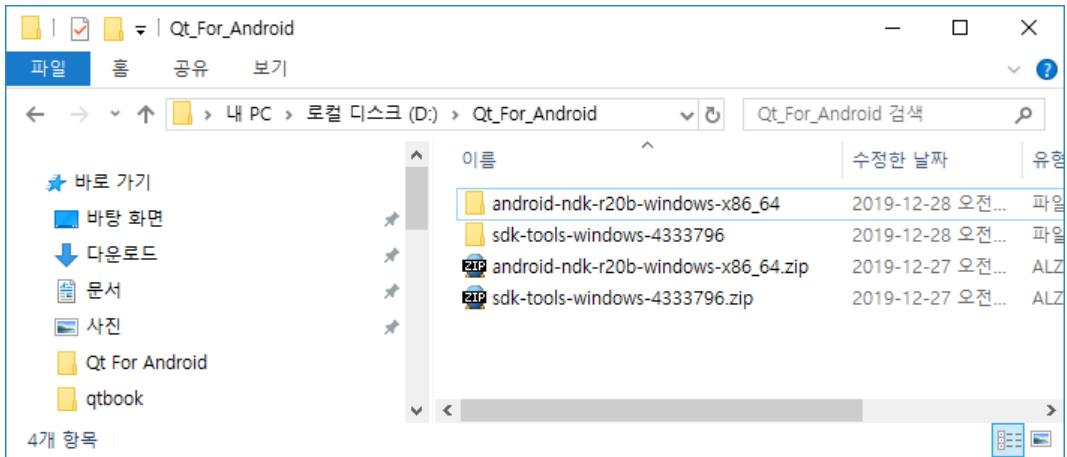
A screenshot of a web browser displaying the 'Latest Stable Version (r20b)' page for the NDK. The URL in the address bar is /ndk/downloads. The page has a header with 'y/ndk/downloads' and a search bar. Below the header, there's a 'SIGN IN' button. The main content area shows a table of download links for different platforms. The table has columns for Platform, Package, Size (Bytes), and SHA1 Checksum. A red box highlights the 'Windows 64-bit' row, which contains the link 'android-ndk-r20b-windows-x86_64.zip'. Other rows include 'Windows 32-bit', 'Mac', and 'Linux 64-bit (x86)'.

<FIGURE> NDK download

- Note: The Android NDK version r10e is required to use Qt for Android with the GCC toolchain. For Qt 5.12 and later, use the latest version of the NDK with an Android-clang tool chain.

Jesus loves you.

When you download the NDK, unzip it as shown in the figure below.



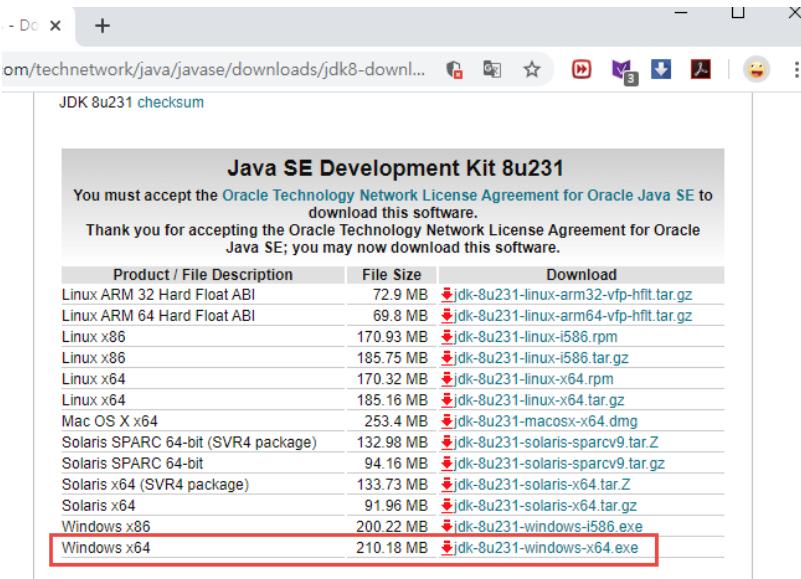
<FIGURE> uncompress screen

- Java SE Development Kit(JDK) download and install

Java SE Development Kit versions must be installed with a minimum of v6 versions.

- URL - <https://www.oracle.com/technetwork/java/javase/downloads/index.html>

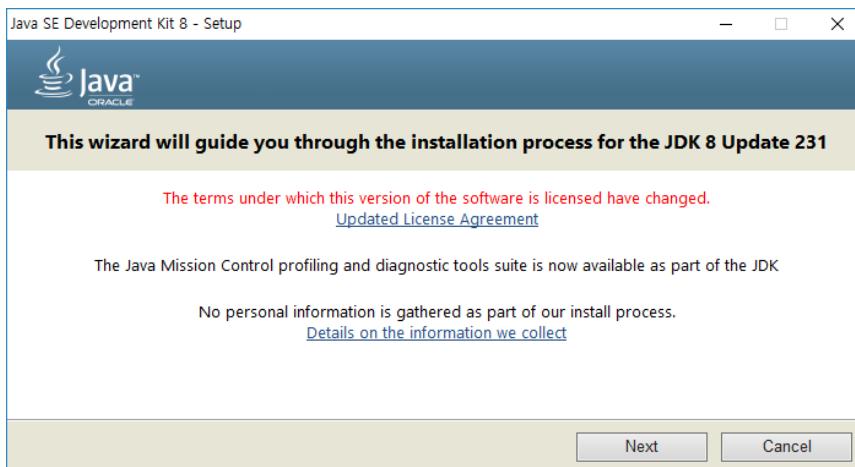
Download the Java SE Development Kit as shown in the figure below.



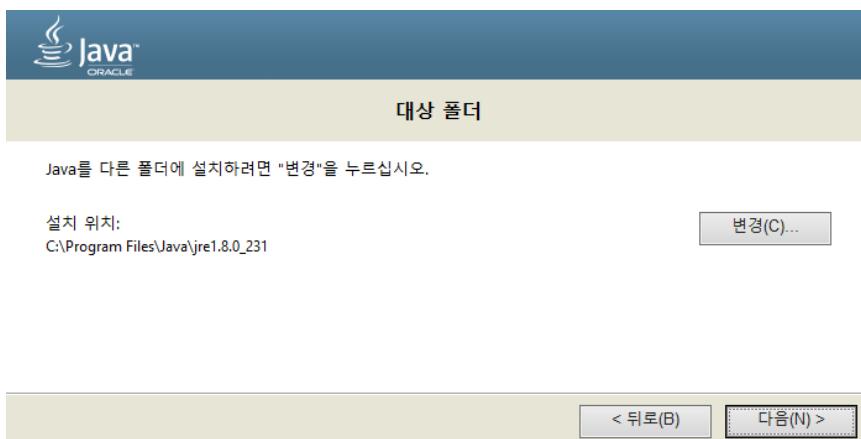
<FIGURE> Java SE Development Kit download

Download the Java SE Development Kit and install it as shown in the figure below.

Jesus loves you.



<FIGURE> Java SE Development Kit install screen

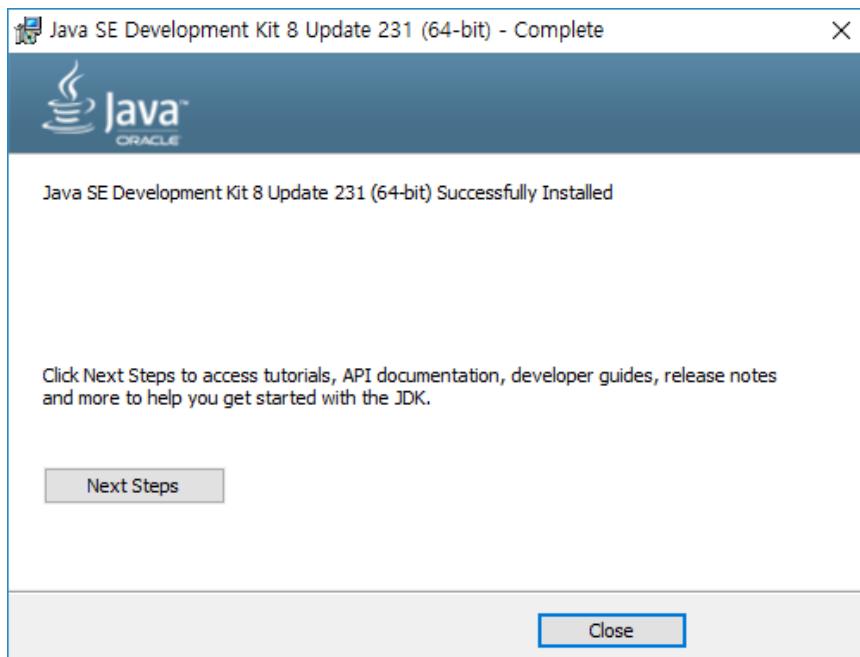


<FIGURE> Specify the Java SE Development Kit installation directory screen



<FIGURE> Java SE Development Kit Installation Progress Screen

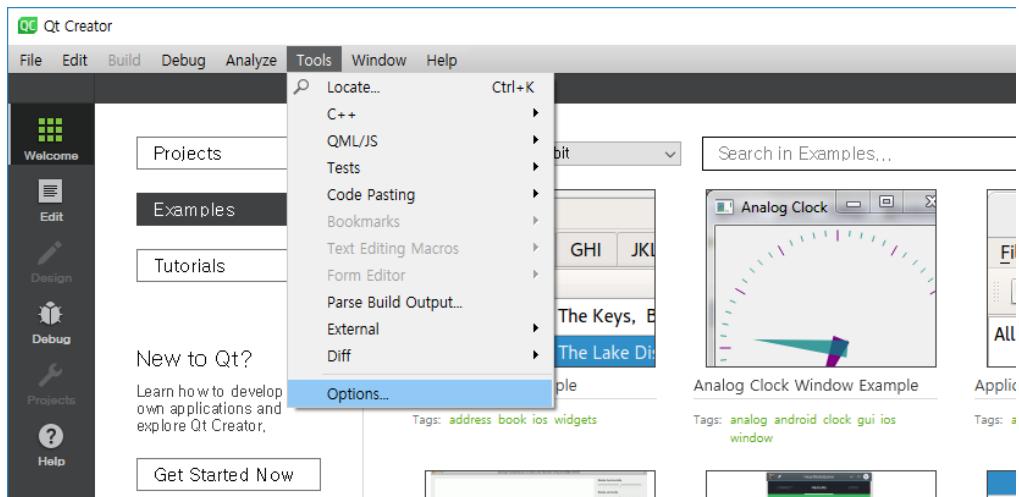
Jesus loves you.



<FIGURE> Java SE Development Kit Installation Complete Screen

- Qt Creator 설정

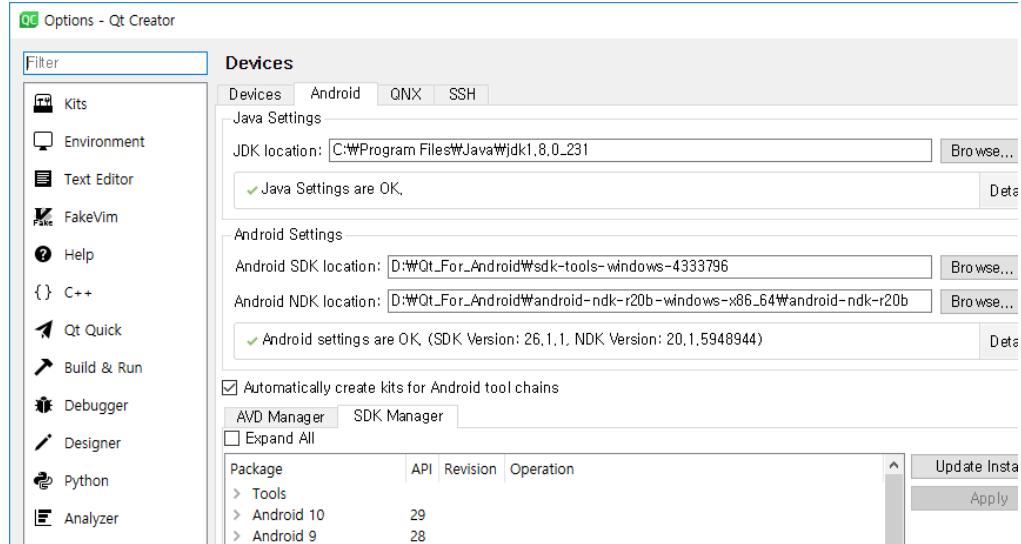
Run Qt Creator: And as you can see in the picture below, [Tools] -pt [Options...] Click on the menu.



<FIGURE> [Options...] Qt Creator screen

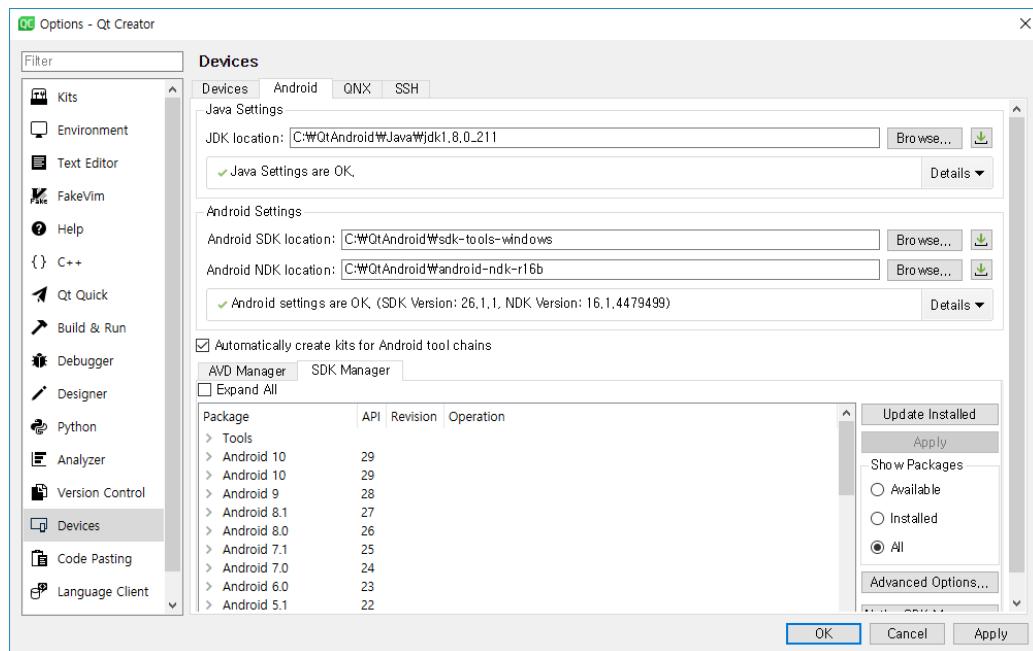
Then select the Devices menu on the left side of the Options dialog and click the Android menu on the right, as shown in the figure below.

Jesus loves you.



<FIGURE> Options dialog

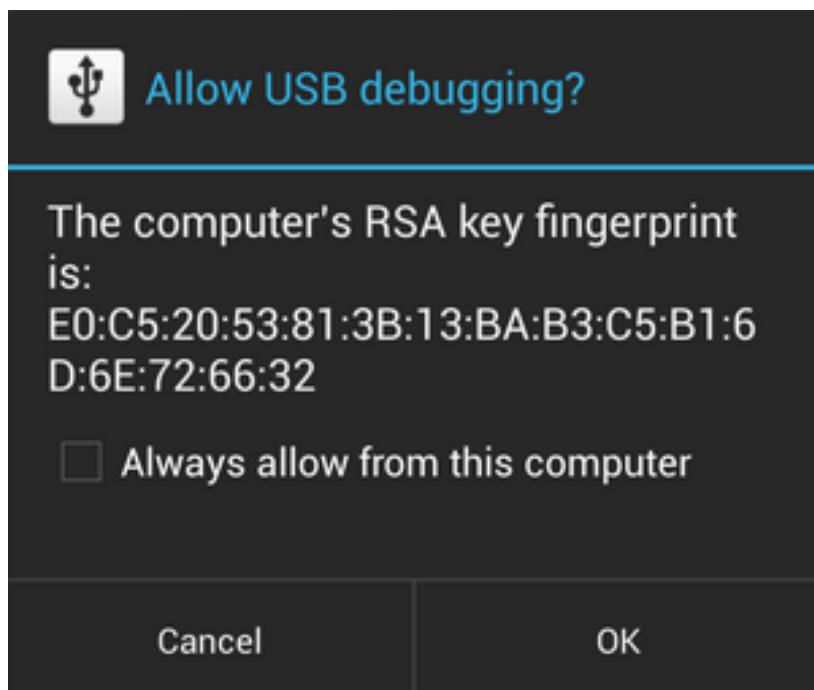
Specify the JDK, SDK, and NDK directories as shown in the figure above.



<FIGURE> Options Dialog

As shown in the picture above, the preparation for Android development is complete.

In order to debug apps written in Qt on Android Phone, you must allow USB debugging on Android as shown below.

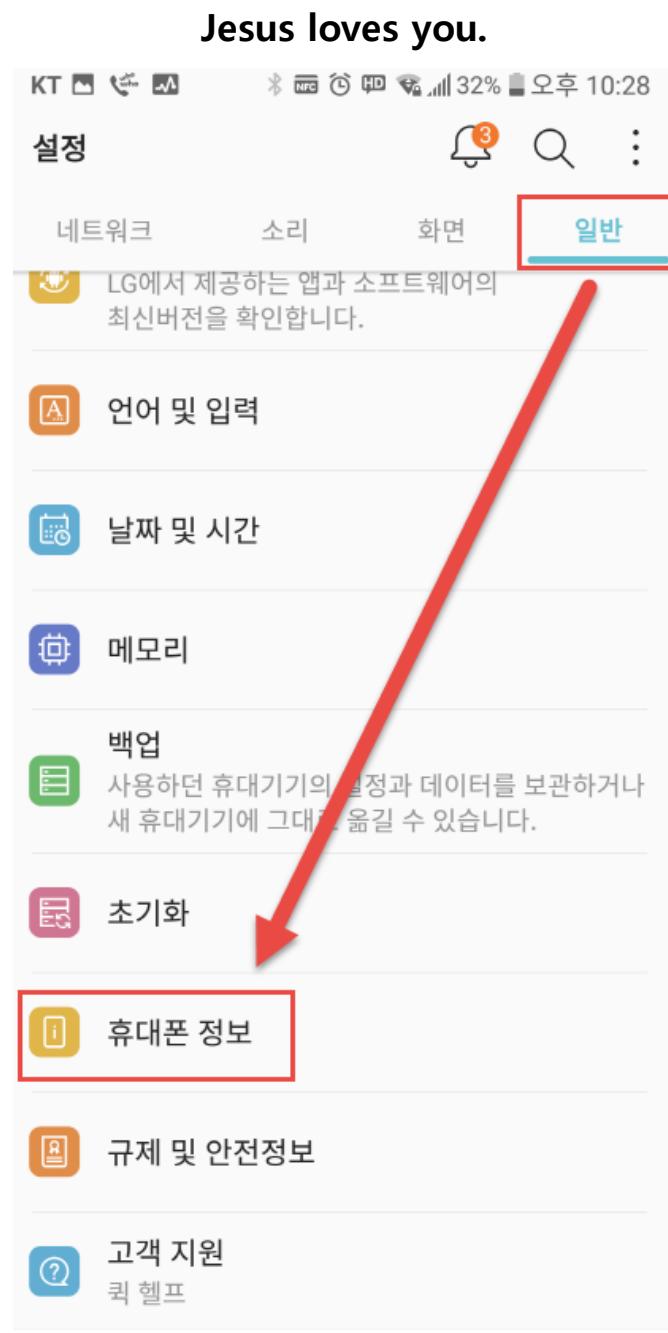


<FIGURE> USB Debugging Allowed for Android Smartphones screen

This allows USB debugging mode of actual Android smartphones, as shown in the picture above. How to allow USB debugging of Android smartphones can be described in more detail by searching on the Internet.

- Enabling Android Phone Developer Options and Allow USB Debugging Mode

Enabling Android Phone Developer Options and Allow USB Debugging Mode To debug and distribute applications written in Qt on Android phones, developer options must be enabled. Enter the setup menu as shown in the picture below.



<FIGURE> Android Phone Settings Menu Screen

Click on the [Settings] > [Mobile Information] menu of the Android phone as shown in the picture above. Then click the Software Information button as shown in the figure below.

Jesus loves you.

KTLTE 3G 4G 5G 33% 오후 10:34

← 휴대폰 정보

휴대폰 이름

V30

네트워크

네트워크, 서비스 상태, 모바일 네트워크 상태 등

상태

내 휴대폰 번호, IMEI 등

배터리

배터리 상태, 수준 등

하드웨어 정보

모델번호, Wi-Fi MAC 주소, Bluetooth 주소 등

소프트웨어 정보

Android 버전, Baseband 버전, 소프트웨어 버전 등

법률 정보

LG 소프트웨어 사용 동의, 오픈소스 라이선스, Google
법적 고지

사용 기록

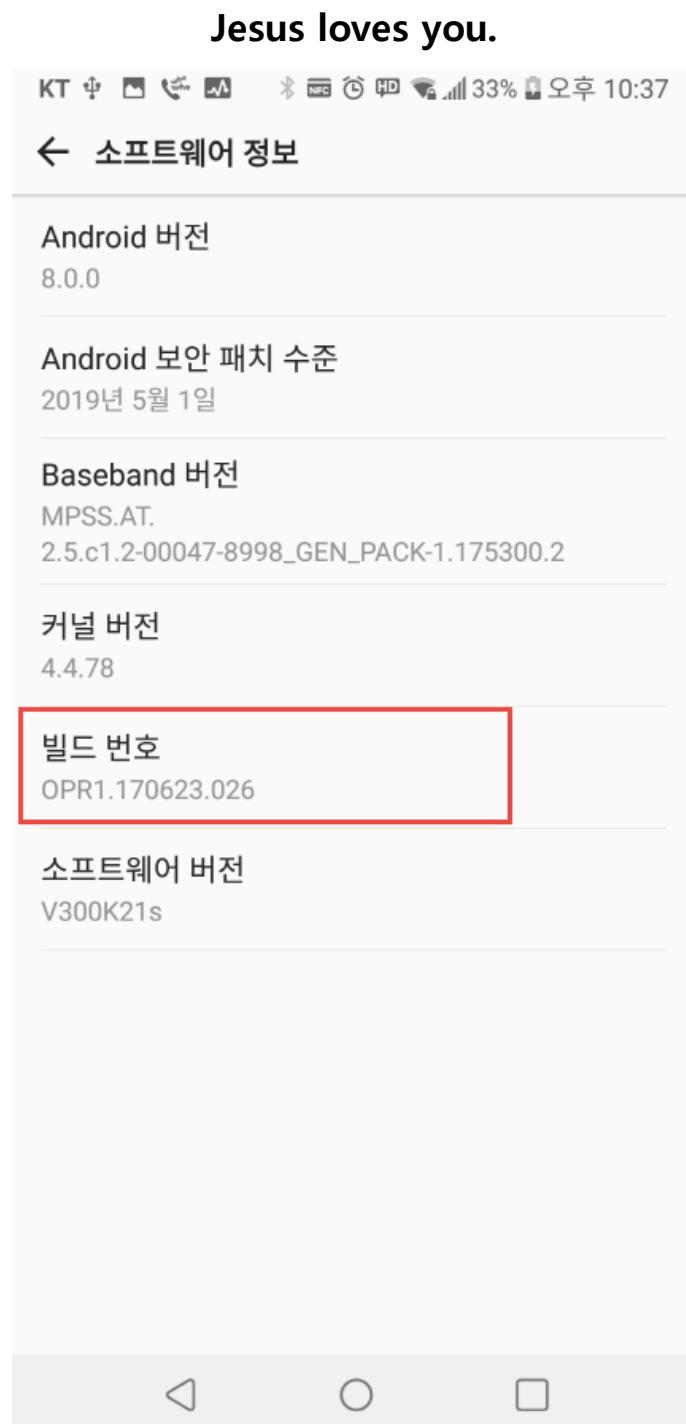
휴대기기 고장 발생 시 진단 서비스의 품질 및 성능
개선을 위하여 사용 기록을 저장합니다.

전자파흡수율 등급



<FIGURE> Android Phone's Mobile Information Menu Screen

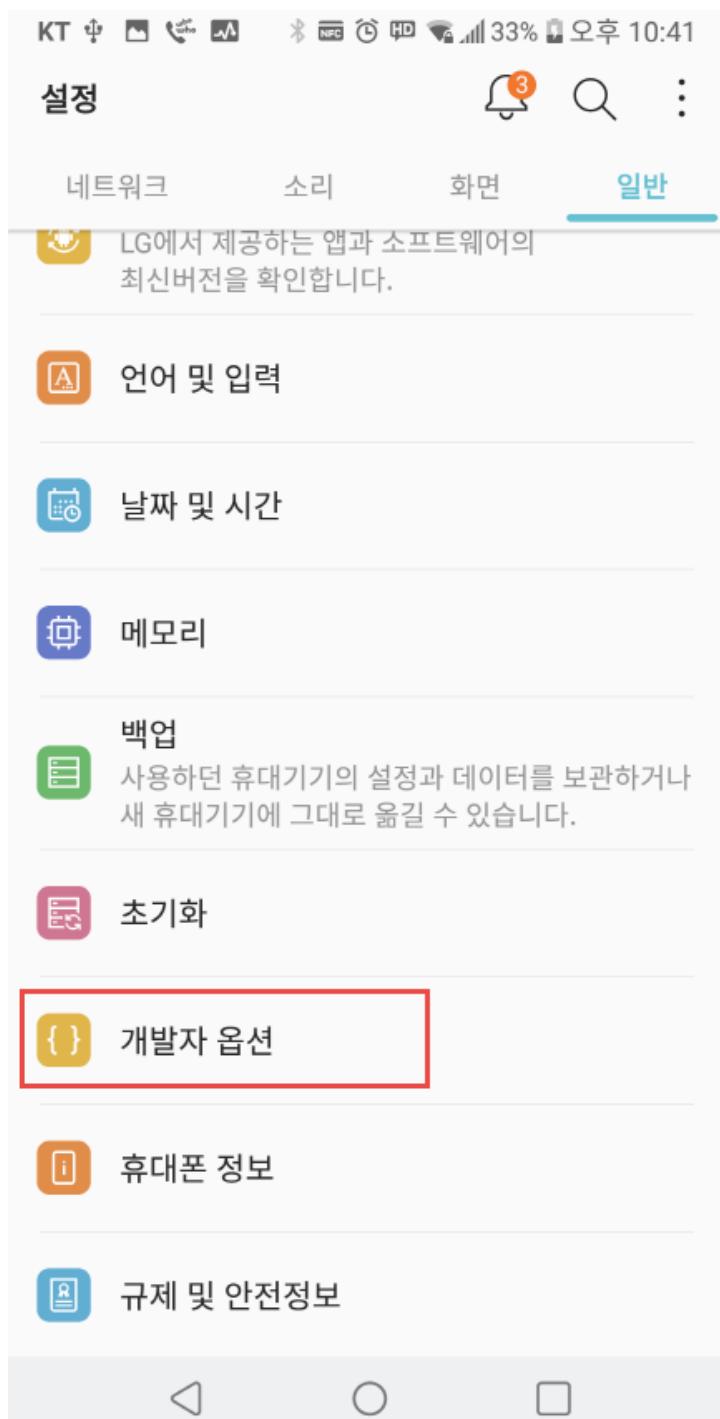
Click Software Information as shown in the picture above, as shown in the mobile phone information menu screen.



<FIGURE> Android Phone's Software Information Screen

Click [Build Number] 7 times as shown in the figure above to activate the developer mode as shown in the figure below.

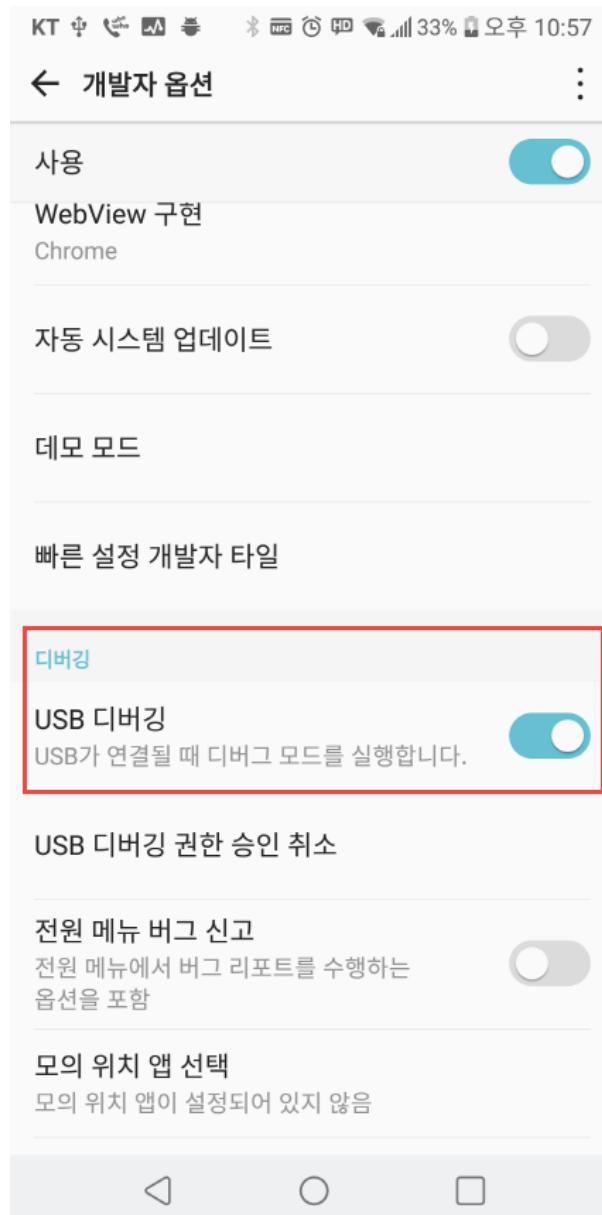
Jesus loves you.



<FIGURE> Activating Android Phone's Developer Options

The following enables the USB debugging mode as shown in the figure below.

Jesus loves you.

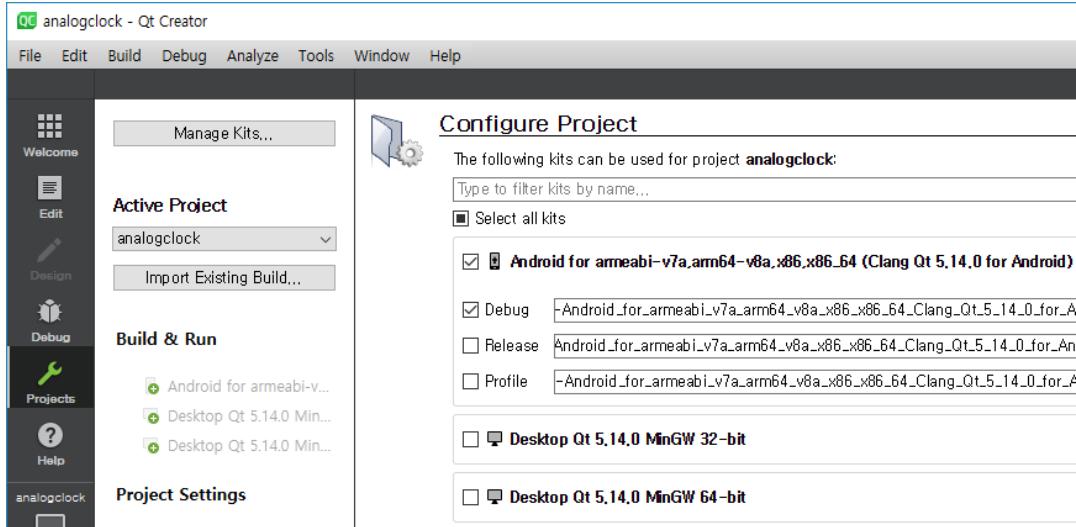


<FIGURE> USB Debugging Mode Activation Screen

- Build and distribute applications written in Qt on Android phones

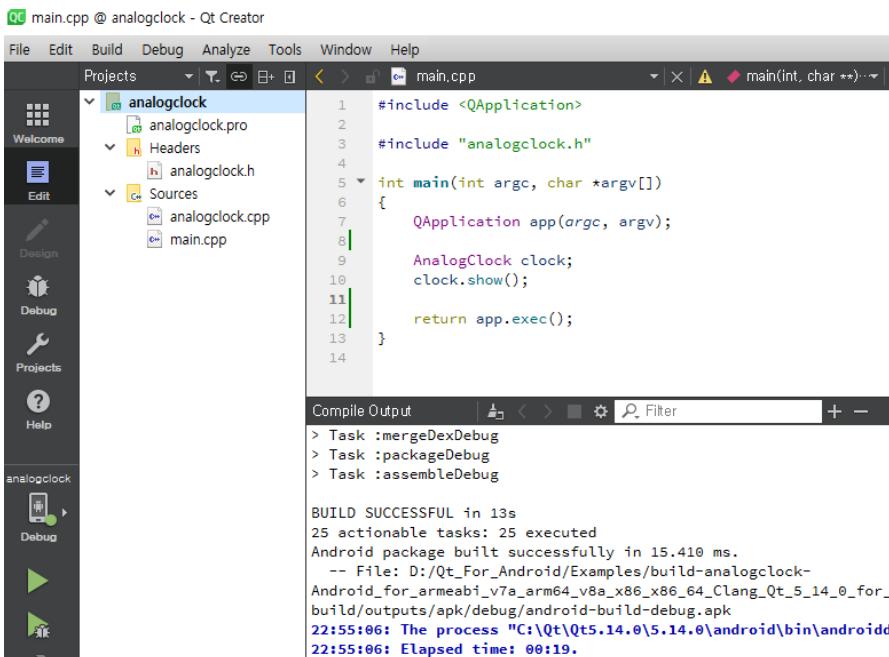
Build an example in Qt and distribute it on Android phones and try debugging it. An example to use this time is the Analog Clock example. This example is a native example of installing Qt.

Jesus loves you.



<FIGURE> Select Kit to Build screen

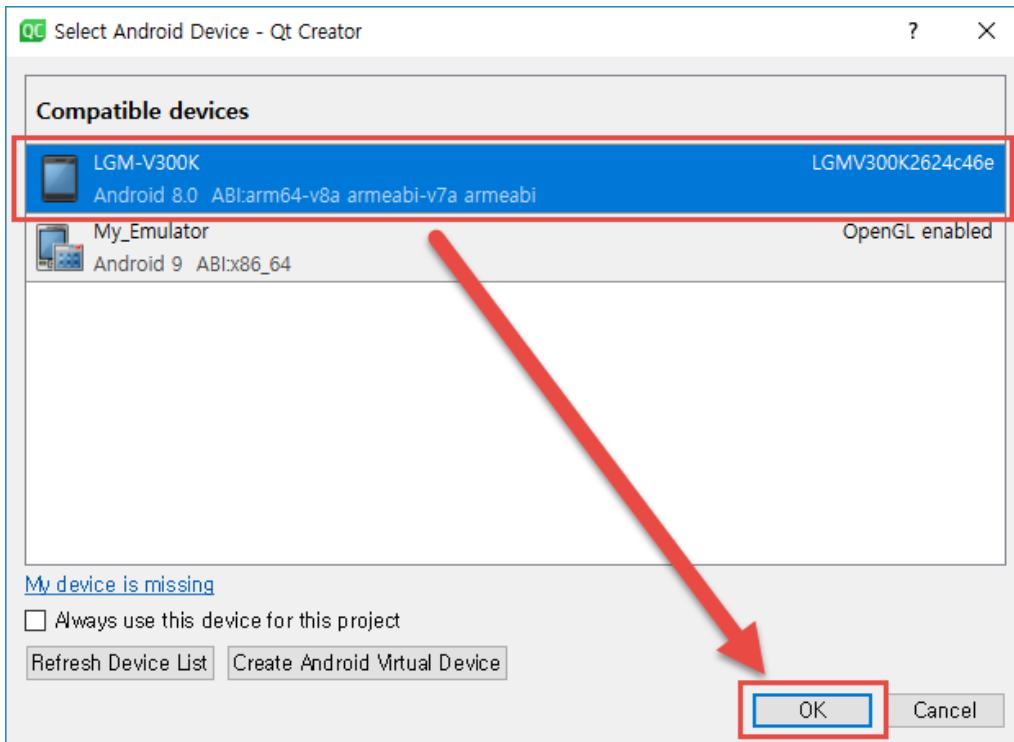
Select Android Kit as the kit to build as shown above. When the build is complete as shown in the figure below, click on the [Run] menu to run it.



<FIGURE> Build completed in Qt Creator

The following figure runs [Run] on the Qt Creator and loads a dialog window that selects the Android phone to be deployed and executed. In the dialog window, select the Android phone to load the application created with Qt and click the [OK] button below.

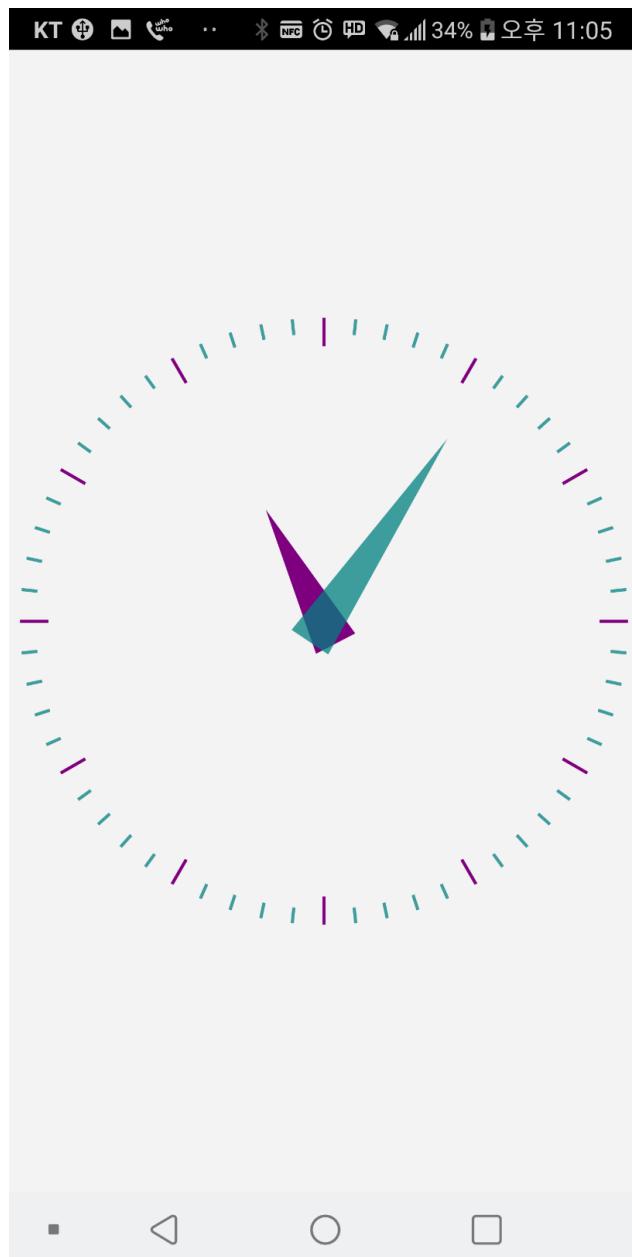
Jesus loves you.



<FIGURE> The dialog screen that selects the Android phone to load the application.

The following picture shows an app built on Qt over Android Phone loaded on Android Phone.

Jesus loves you.

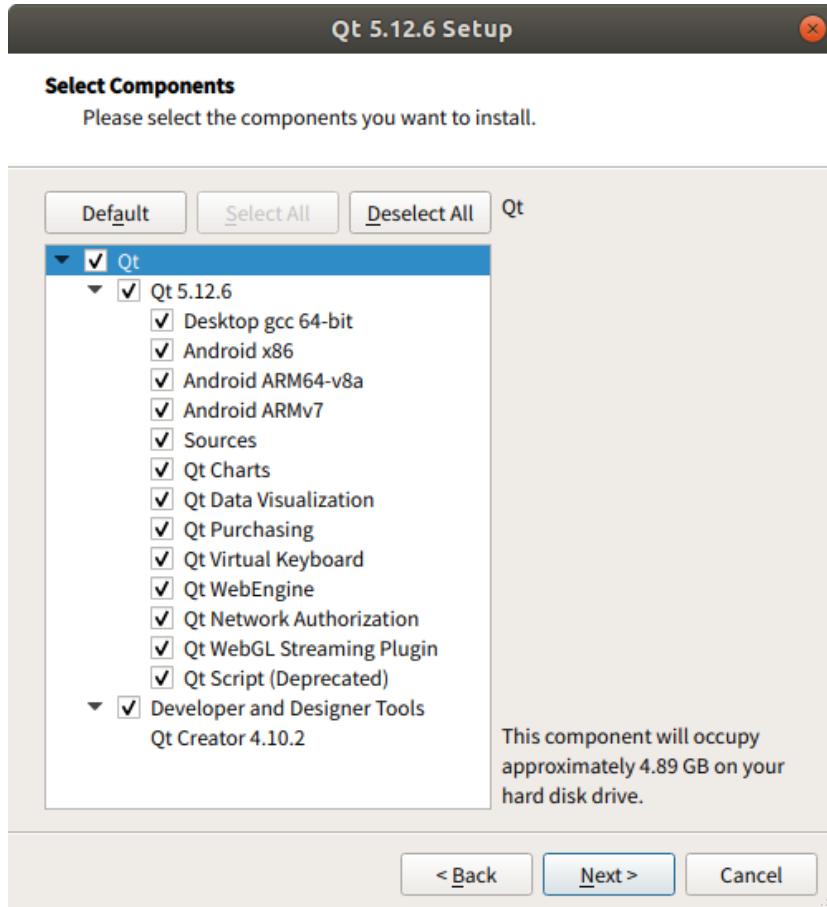


<FIGURE> App screen running on Android Phone

Example directory – Ch20 > 01_analogclock

20.2. Deploy Android apps using Qt on Linux

We will use Ubuntu Linux to develop apps that work on Android phones. The Ubuntu Linux version used the 18.04 64bit version. And Qt used version 5.12.6. Select all components as shown in the figure below.



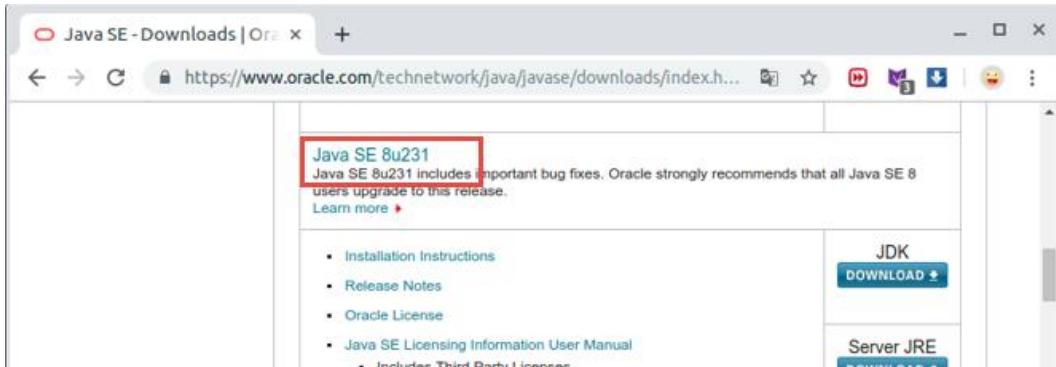
<FIGURE> Qt Select Components at Installation Screen

- Java SE Development Kit(JDK) download and install

The JDK version shall use the 1.6 version. You can download it from the URL below.

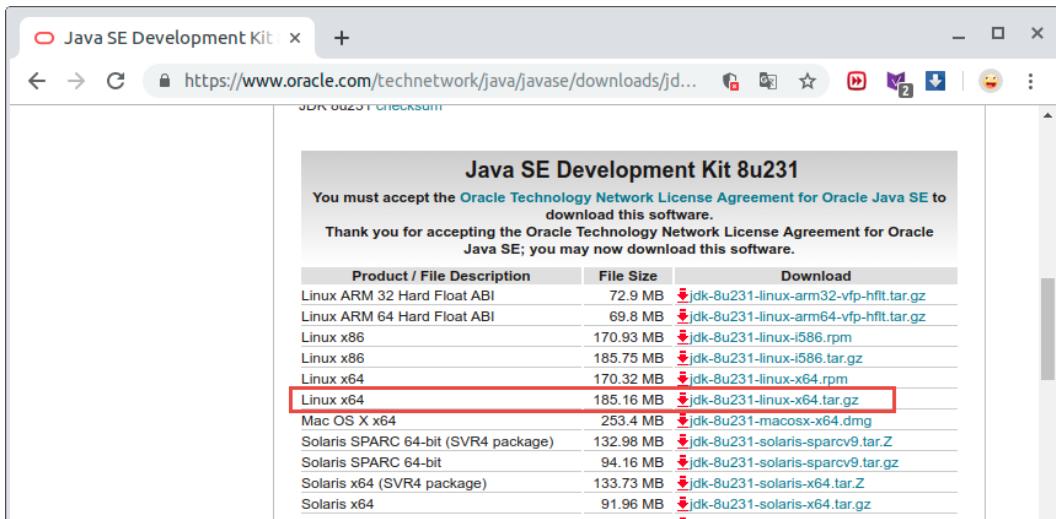
- <https://www.oracle.com/technetwork/java/javase/downloads/index.html>

Jesus loves you.



<FIGURE> Java SE download

Click on the link [Java SE 8u231] as shown in the figure above. And download the Linux x64 version as shown in the figure below.



<FIGURE> Java SE download

Unzip the download as shown in the following example.

```
tar xfz jdk-8u231-linux-x64.tar.gz
```

After decompression, the JAVA_HOME environment variable should be registered on the .pro file as shown in the example below.

```
export JAVA_HOME=/home/eddykim/Qt/Qt_For_Android/jdk1.8.0_231  
export PATH=$PATH:$JAVA_HOME/bin
```

Jesus loves you.

As shown in the above example, you can verify that the JAVA_HOME environment variables are normally applied.

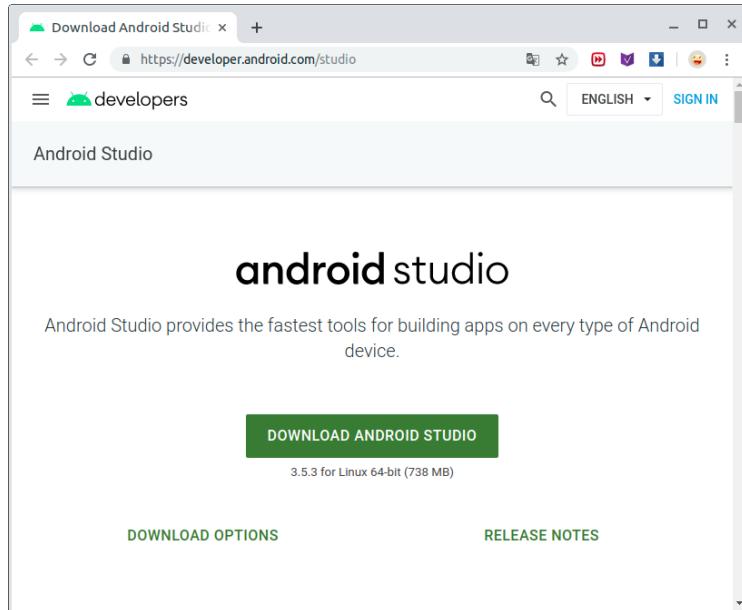
```
eddykim@eddykim-note:~$ java -version
java version "1.8.0_231"
Java(TM) SE Runtime Environment (build 1.8.0_231-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.231-b11, mixed mode)
```

In addition to the JDK version provided by Oracle, OpenJDK can be used as an alternative. To install the OpenJDK, you can install it using apt-get as shown in the example.

```
apt-get install openjdk-8-jdk
```

- Android SDK Tools download and instal, URL - <https://developer.android.com/studio>

If you use versions of Qt 5.9 or earlier, you should use SDK Tools package v25.5 or lower. You can download the SDK by accessing the above URL address using a web browser. Click the [DOWNLOAD OPTIONS] link as shown in the figure below.

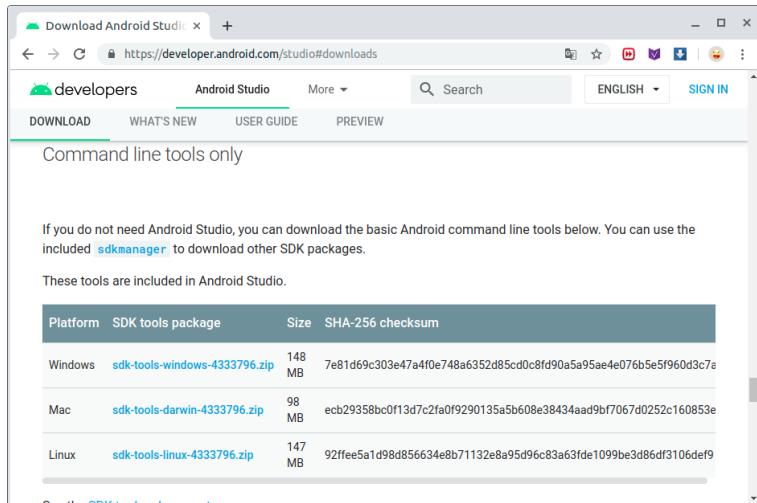


<FIGURE> SDK Tools download screen

Download the [Linux] entry from the [Command line tools only] item as shown in the

Jesus loves you.

figure below.



<FIGURE> SDK Tools download screen

Unzip the download after completion. After decompression, update the SDK using the sdkmanager executable file in the [tools] > [bin] directory.

```
./sdkmanager --update
```

A screenshot of a terminal window on a Linux system. The command ./sdkmanager --update is entered, and the output shows the progress of computing updates. The terminal window has a dark background with white text.

<FIGURE> sdkmanager screen

You can use the sdkmanager of <ANDROID_SDK_ROOT>/tools/bin to install a specific package of SDK on the Ubuntu Linux operating system (platform). For example, to install an Android-10 platform package, you can use the following example.

```
./sdkmanager "platforms;android-10"
```

In addition to this method, it can be installed later in the Options dialog window provided by the Qt Creator IDE tool. If you are using the version of Ubuntu Linux that you use, you should install the package using app-get as follows:

Jesus loves you.

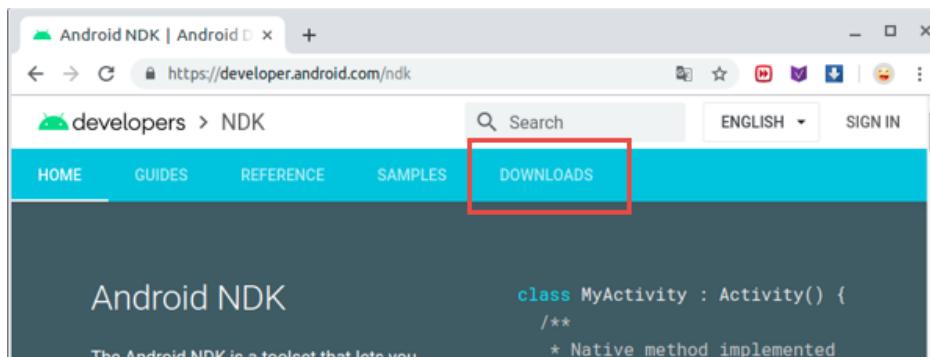
```
sudo apt-get install libstdc++6:i386 libgcc1:i386 zlib1g:i386 libncurses5:i386
```

In addition, the following packages must be installed to use emulator.

```
sudo apt-get install libsdl1.2debian:i386
```

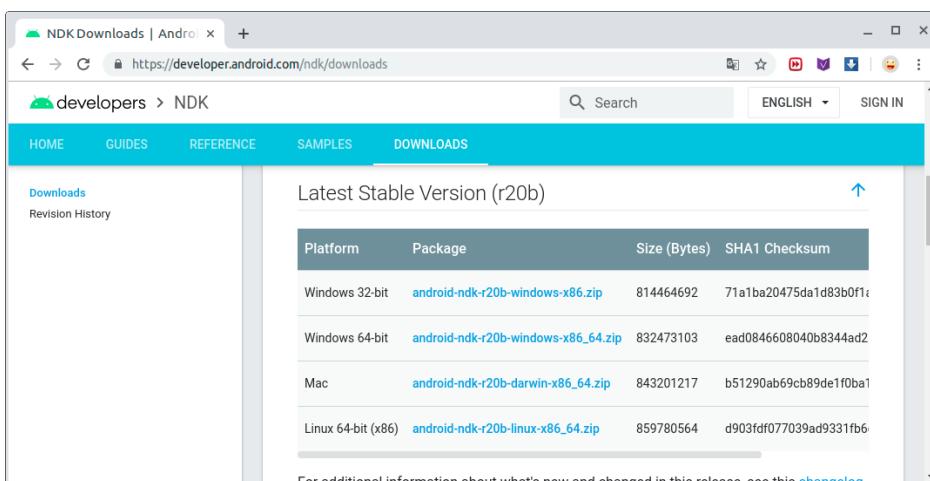
- Android NDK download and install (URL : <https://developer.android.com/ndk>)

Access the above URL and download the NDK. Click on the [DOWNLOAD] link as shown in the figure below.



<FIGURE> NDK download

Next, download the latest version of NDK.



Platform	Package	Size (Bytes)	SHA1 Checksum
Windows 32-bit	android-ndk-r20b-windows-x86.zip	814464692	71a1ba20475da1d83b0f1c
Windows 64-bit	android-ndk-r20b-windows-x86_64.zip	832473103	ead0846608040b8344ad2
Mac	android-ndk-r20b-darwin-x86_64.zip	843201217	b51290ab69cb89de1f0ba1
Linux 64-bit (x86)	android-ndk-r20b-linux-x86_64.zip	859780564	d903fdf077039ad9331fb6

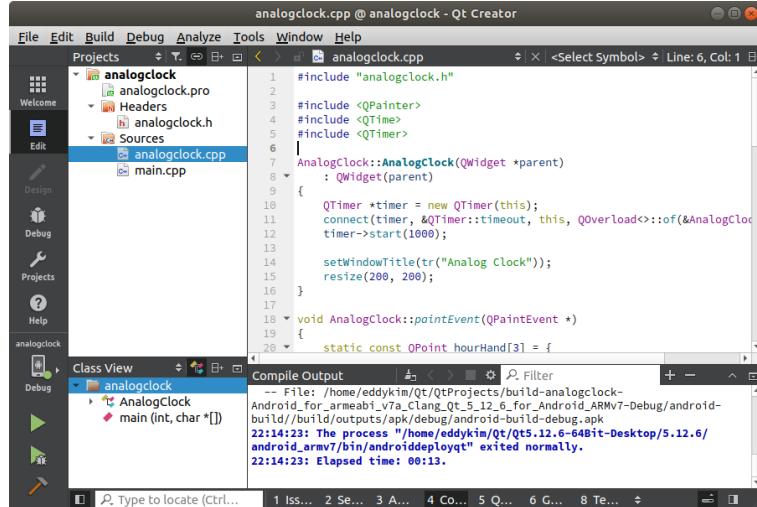
<FIGURE> NDK download

Unzip the NDK after the download completes.

Jesus loves you.

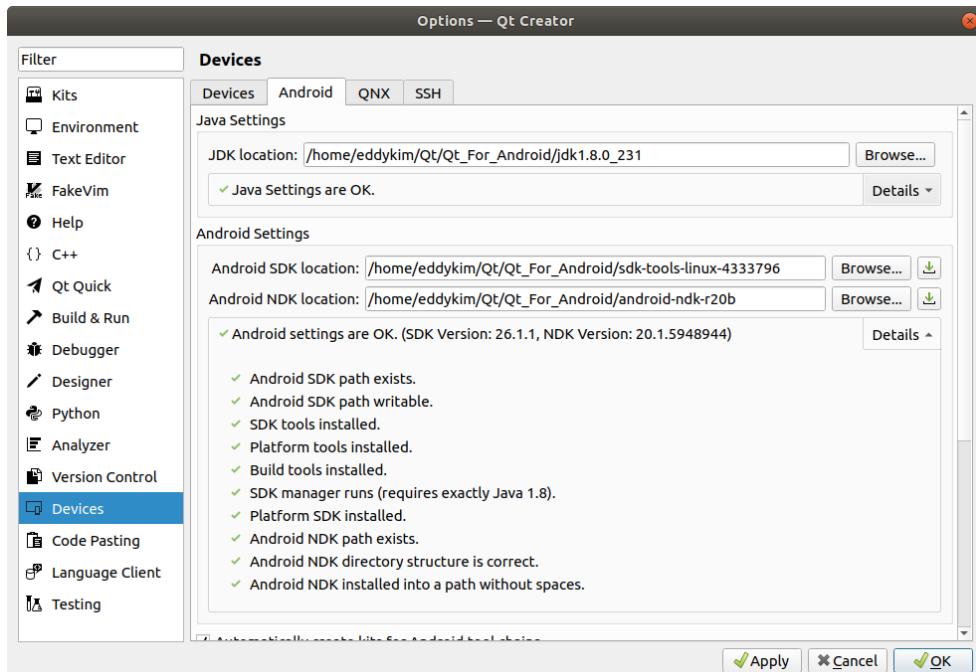
- Qt Creator Settings

Run the Qt Creator. Click the [Tools] > [Options] menu in the Qt Creator IDE menu.



<FIGURE> Qt Creator screen

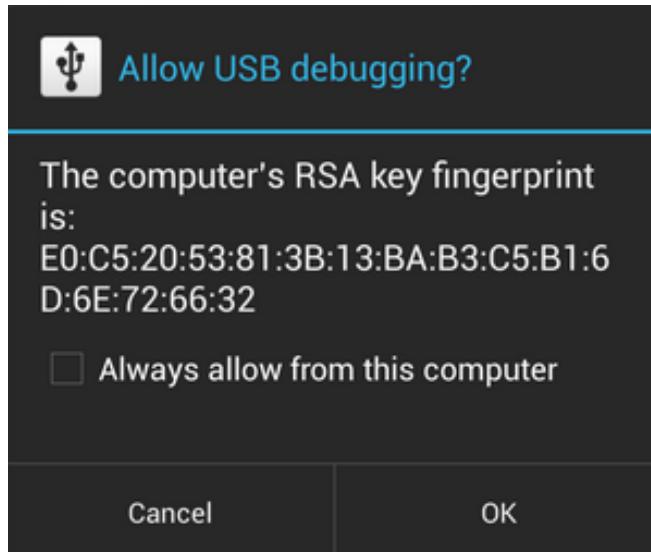
Specify the JDK, SDK, and NDK directories as shown in the figure above.



<FIGURE> [Options] dialog screen

Jesus loves you.

When JDK, SDK and NDK directories are specified without error messages as shown in the figure above, USB debugging should be permitted as shown below to debug and distribute Qt-written apps on Android phones.



<FIGURE> USB Debugging Allowed Screen for Android Phone

Refer to the previous section for how to activate Android Phone's developer options and USB debugging mode.

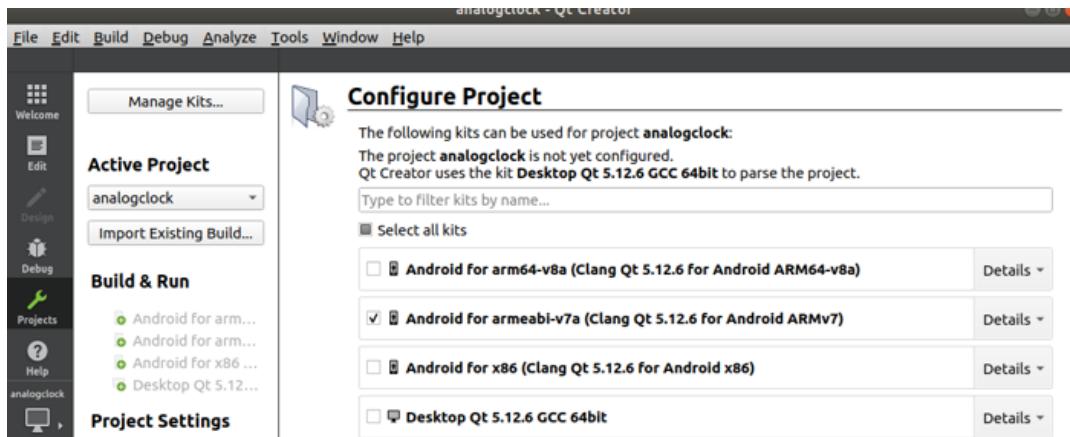
To build and distribute an app created with Qt on Android Phone, connect Android Phone as shown in the example below and verify that the device is detected normally using the adb command. The adb tool exists in the [ANDROID_SDK_ROOT]/platform-tools directory.

```
$ ./adb devices
List of devices attached
LGMV300K2624c46e        device
```

- Build and distribute app sources created with Qt on Android phones

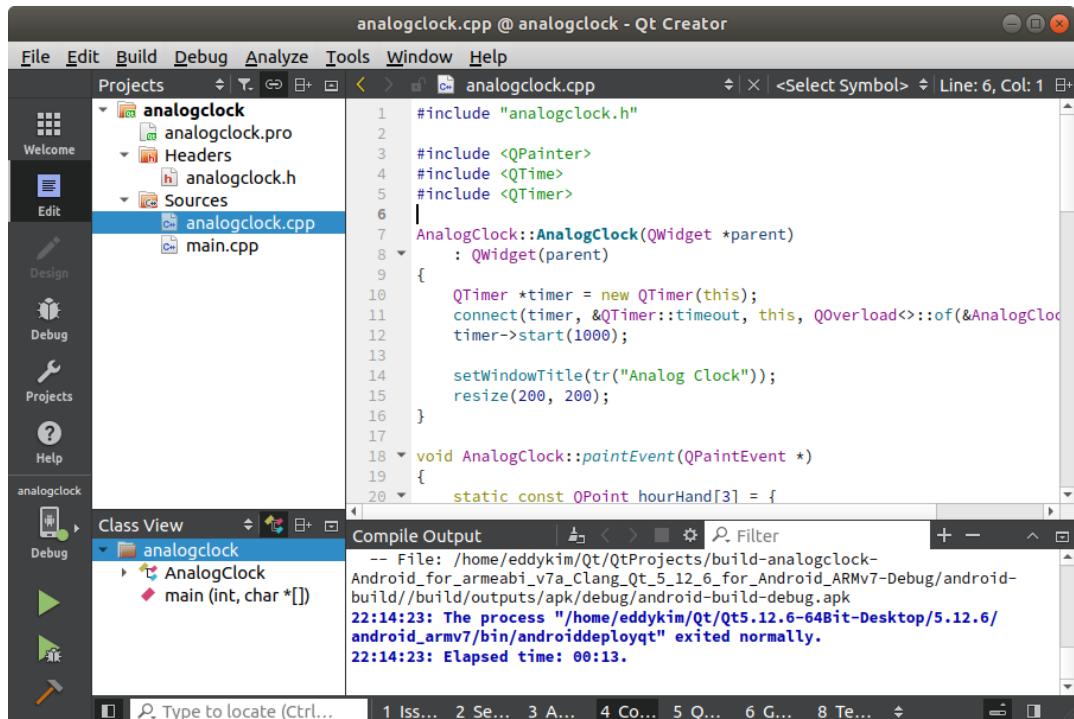
Let's distribute the app source code written with Qt on the Android phone and try debugging it. An example to use this time is the Analog Clock example. This example is provided by installing Qt.

Jesus loves you.



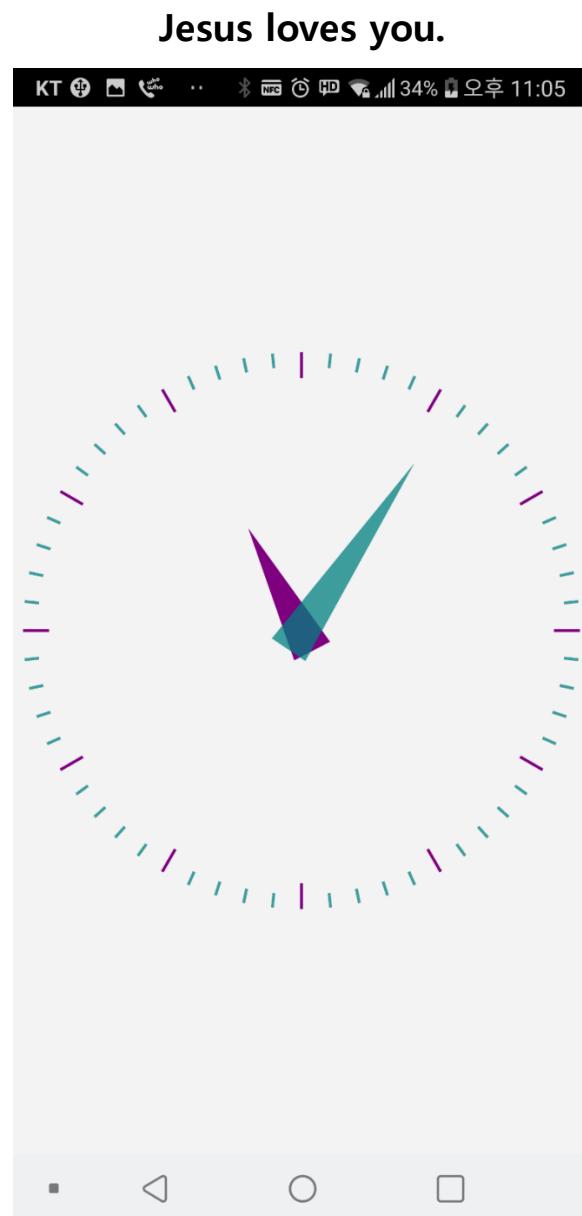
<FIGURE> Select Kit to Build

Select [Android for ameabi-v7a] as the kit to build as shown in the picture above. Then build and run the App [Run] menu as shown in the picture below.



<FIGURE> Qt Creator build screen

As you can see in the picture above, when the build is complete, let's run the App.

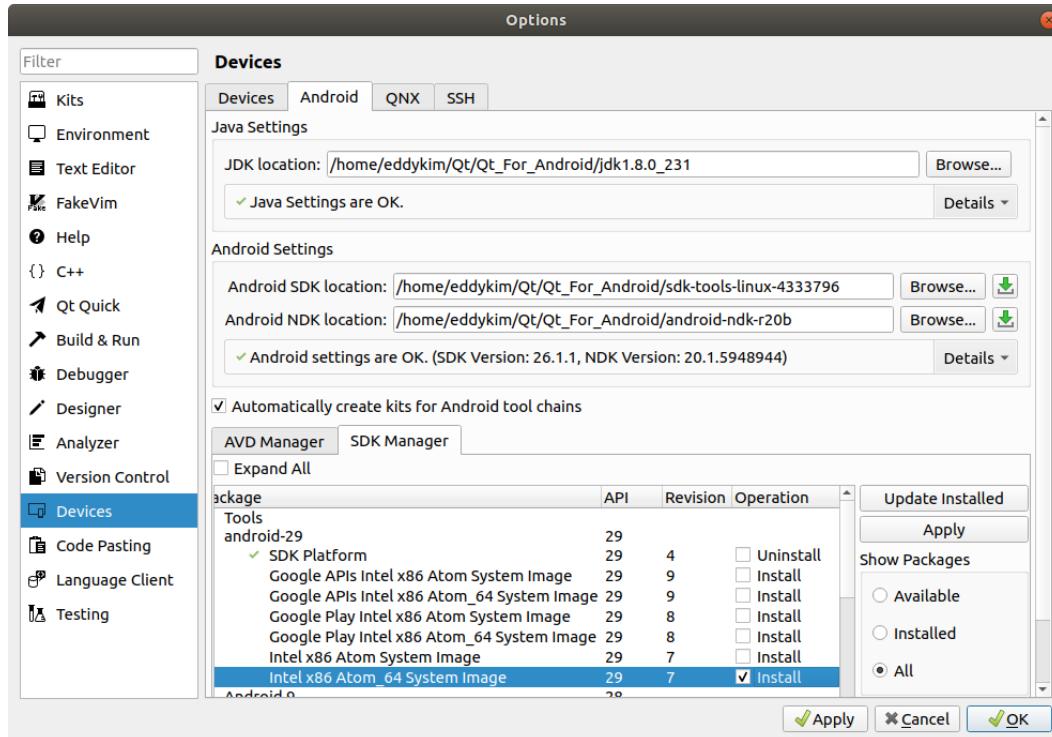


<FIGURE> App screen

- Deploy Qt-built apps on Android Emulator

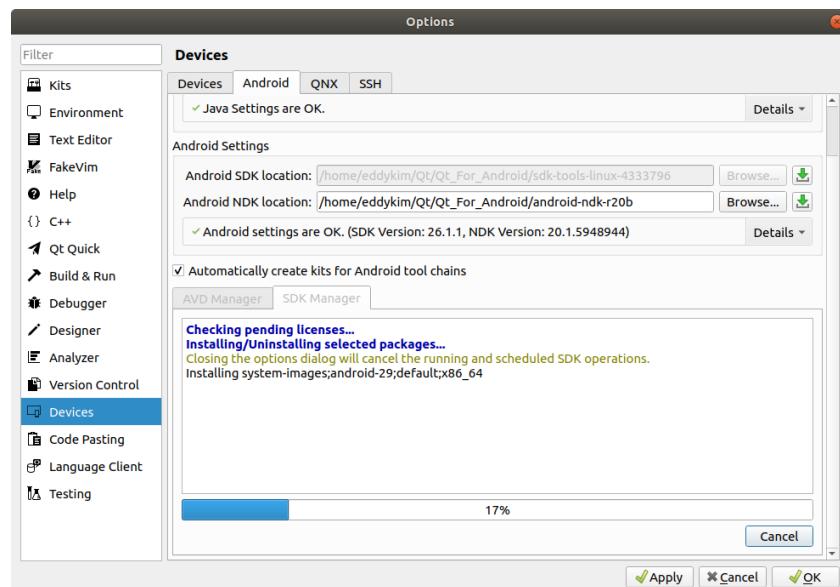
This time, let's build and distribute the Qt App built on Android Emulator. To use the Android Emulator, run [Tools] > [Options] from the Qt Creator menu.

Jesus loves you.



<FIGURE> Options dialog

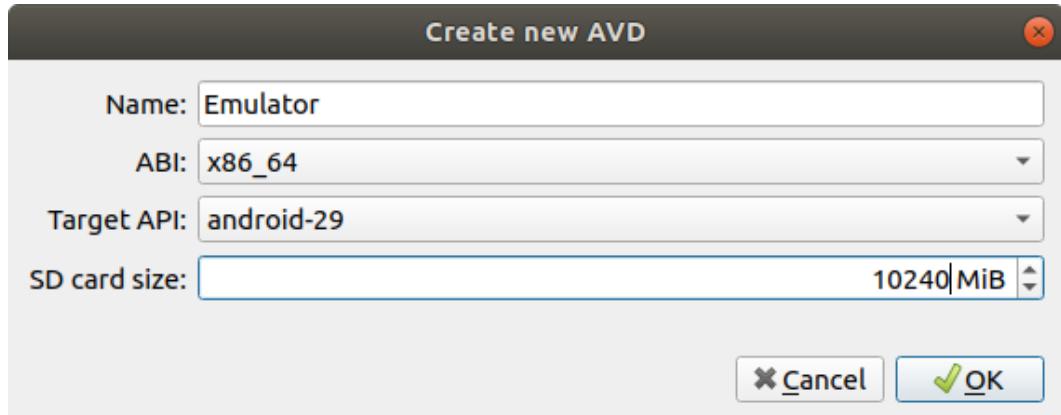
Install the [Intel x86 Atom_64 System Image] item and click the [Apply] button on the right to proceed with the installation as shown in the figure above.



<FIGURE> Android SDK packages install screen

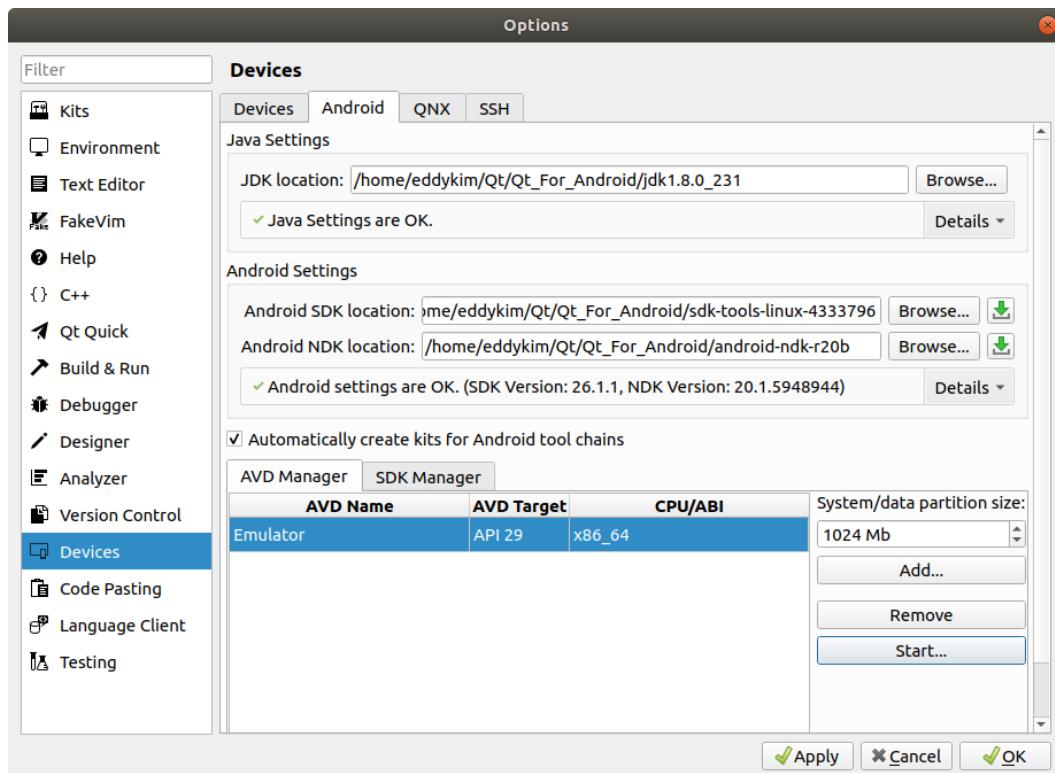
Jesus loves you.

When the package installation is complete, click the [AVD Manager] tab. Then click the [Add] button on the right to create an emulator.



<FIGURE> Android Emulator creation screen dialog

Click the [OK] button below after setting up as shown in the figure above. Click the [Start] button on the right when the emulator is completed as shown in the figure below.



<FIGURE> Emulator creation complete screen

Jesus loves you.

Click the [Start] button. If the [Start] button is clicked, the following error may occur:

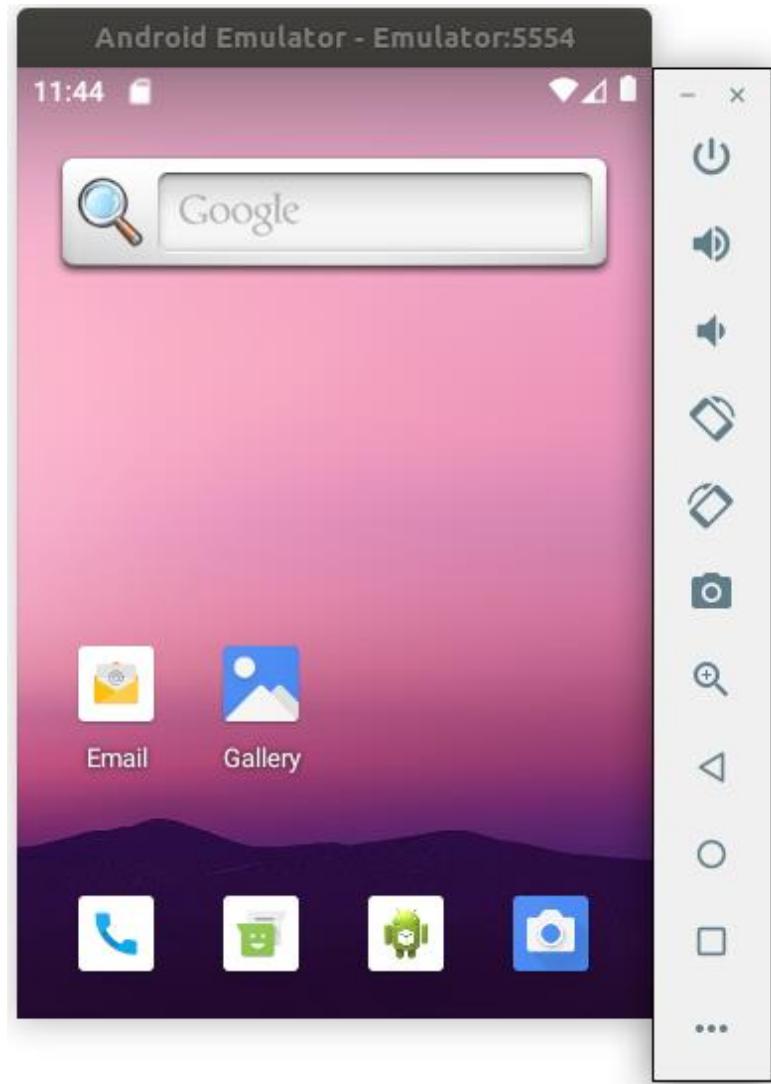
```
emulator: ERROR: x86_64 emulation currently requires hardware acceleration!
Please ensure KVM is properly installed and usable.
CPU acceleration status: This user doesn't have permissions to use KVM
(/dev/kvm)
More info on configuring VM acceleration on Linux:
https://developer.android.com/studio/run/emulator-acceleration#vm-linux
General information on acceleration:
https://developer.android.com/studio/run/emulator-acceleration
```

If an error occurs as shown above, run the following command in the example below at the terminal:

```
$ sudo apt-get install qemu-kvm libvirt-bin ubuntu-vm-builder bridge-utils
$ sudo modprobe kvm_intel
$ sudo chmod 777 -R /dev/kvm
```

If you click the [Start] button and run without error, the Android Emulator will run as shown in the figure below.

Jesus loves you.

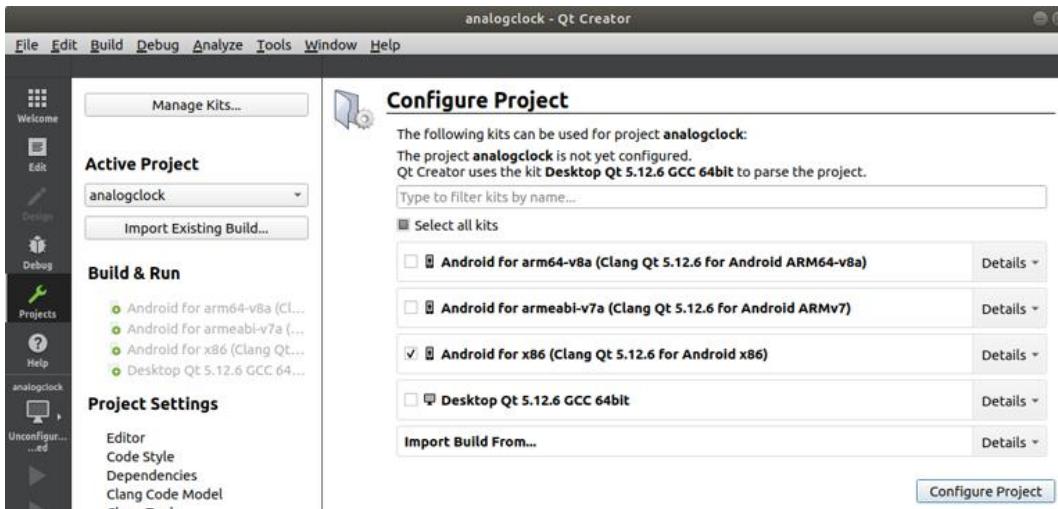


<FIGURE> Android Emulator screen

If the Android Emulator is running normally as shown in the figure above, let's try to distribute and execute the Analog Clock example created with Qt on the Emulator.

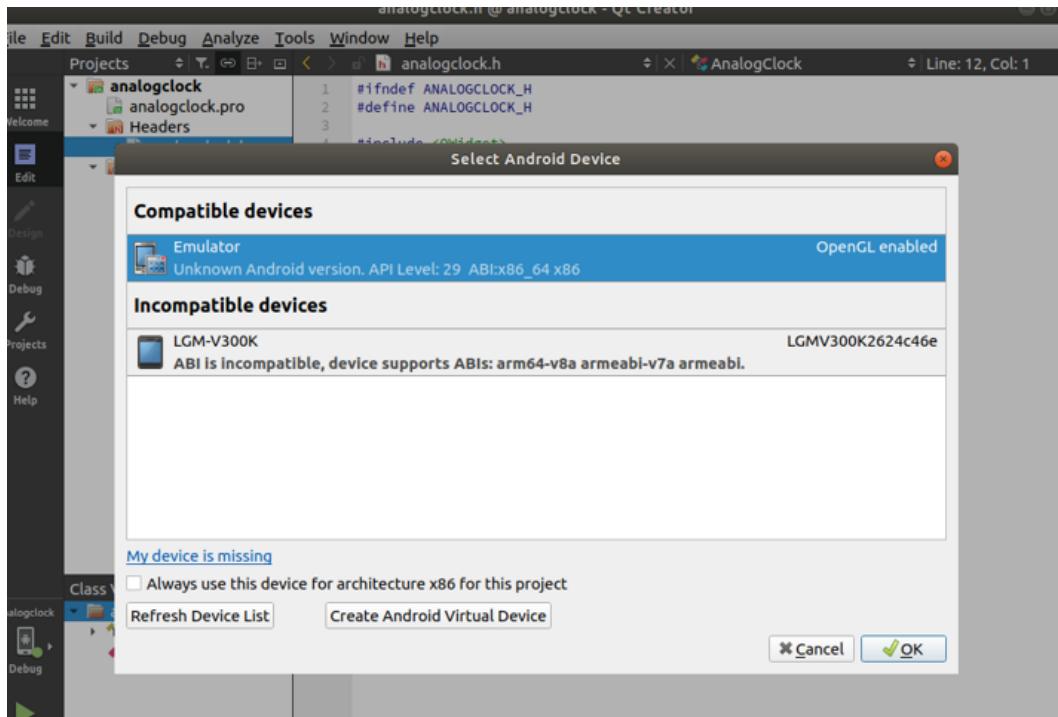
[Open] the Analog Clock example project after running Qt Creator. Then select the [Android for x86] item as shown in the figure below.

Jesus loves you.



<FIGURE> Select Kit to Build

When you build and run on the Qt Creator, a dialogue asking if you want to run on the Emulator or on the actual Android phone is executed as shown in the figure below. Select the emulator as shown in the figure below.

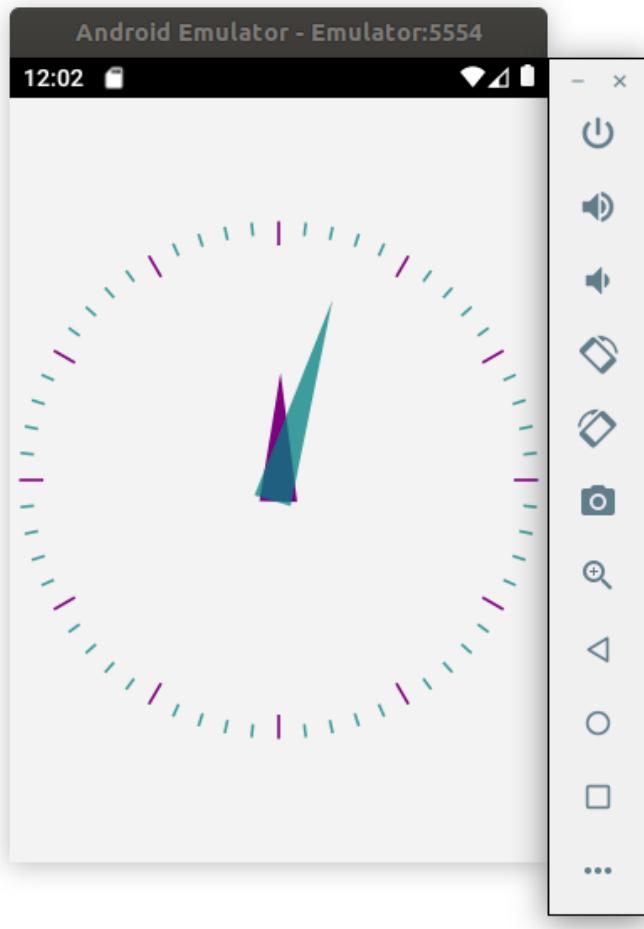


<FIGURE> Android Device selection dialog

Select the Emulator in the Android Device selection dialog as shown in the figure above.

Jesus loves you.

You can then see that the Analog Clock example has been performed on the Emulator as shown in the figure below.



<FIGURE> The screen that ran the Analog Clock example on the Android Emulator.

Example source code: Ch20 > 01_analogclock