Now, let's easily develop the web using Qt.

Qt for WebAssembly

Second Edition, Ver 2.0

Qt Korea Developer Community

www.qt-dev.com

Jesus loves you

Qt for WebAssembly

Release version 2.0 (July 21, 2024)

Homepage URL www.qt-dev.com

Preface

We distribute this book free of charge to anyone interested in Qt WebAssembly.

If there is one thing I pray for, it is that those who are still unbelievers will hear the gospel and turn to our Father.

In addition, Jesus, His only begotten Son, came to this earth and took our sins upon Himself. By the precious blood of Jesus, our sins have been forgiven, and God the Father is waiting for all unbelievers to return to Him out of love for His children.

If any of you who have read this book are still unbelievers, we pray and bless you with our earnest hope that you will accept Jesus as your Savior and be born again as a child of faith. We 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.'

Matthew 22:37~39

Table of Contents

1. What is Qt for WebAssembly

In order to build web-based applications, we need to use web scripting languages such as PHP, ASP, ASP.NET, and JavaScript.

However, WebAssembly technology makes it possible for applications written in C++ to run in a web browser.

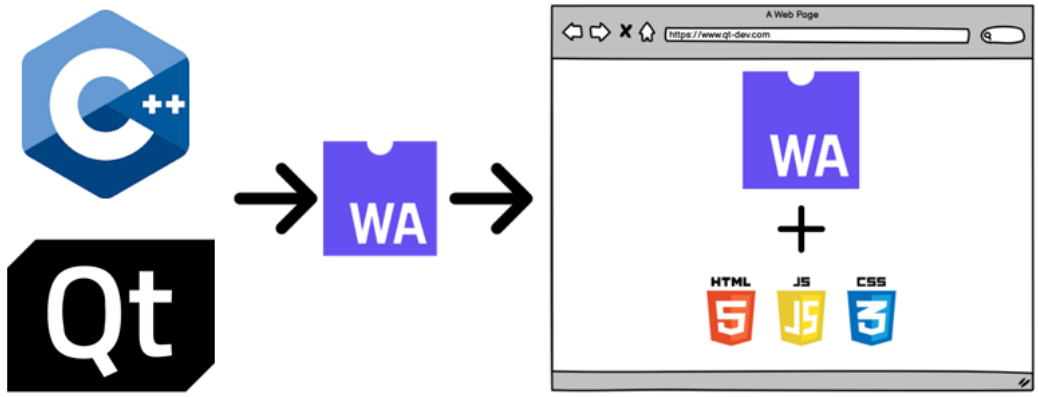
WebAssembly can be used to convert source code written in languages other than C++, such as C and RUST, into byte code that can then be run in a web browser.

The Qt Framework also provides an easy way to develop WebAssembly-based applications. This is a huge advantage for developers as it reduces development time.

For example, when developing a WebAssembly-based web application in C++, no developer would implement all the necessary libraries from scratch.

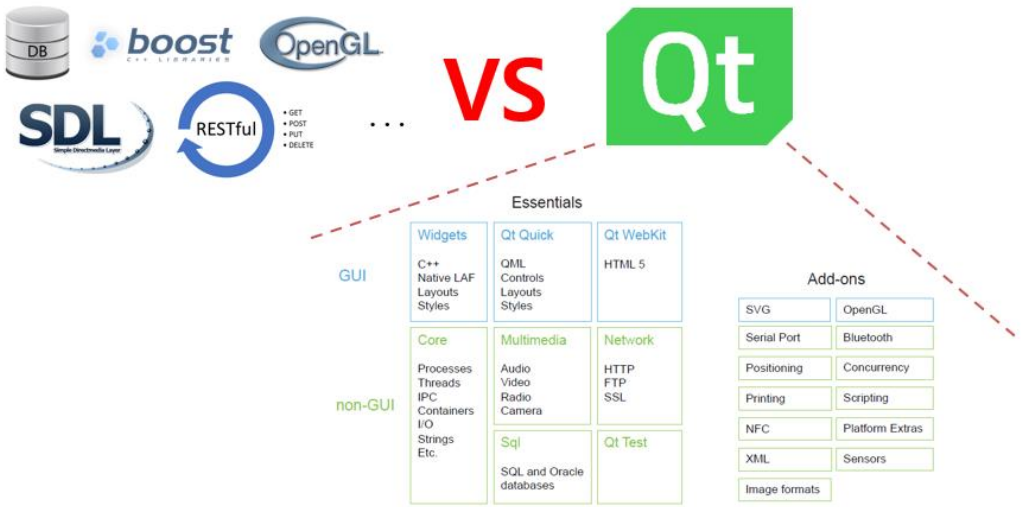
With the Qt Framework, the development time of WebAssembly-based web applications in C++ can be significantly reduced due to the vast range of libraries available. In addition,

debugging is possible with Qt Creator, which makes it very convenient to use debugging techniques.



Therefore, using the Qt Framework to develop WebAssembly-based applications is probably the best option for developers.

If you do not use the Qt Framework, you will have to port various necessary libraries yourself. For example, you will need to port various libraries such as DB, OpenGL, GUI related, etc. However, this is very difficult, because if you encounter any problems during the porting process, you have to solve them yourself. However, the advantage of using Qt for WebAssembly provided by Qt Framework is that you can use the extensive libraries provided by Qt. Therefore, using the Qt Framework can be the best option.

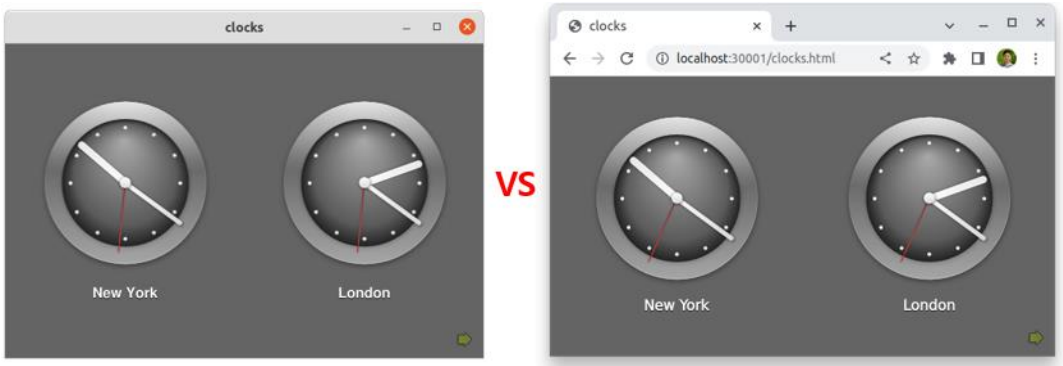


In addition, one of the biggest obstacles in WebAssembly is the availability of multi-threading.

When implementing applications using WebAssembly technology, it is very difficult to use multiple threads because the thread structure is single threaded. If you need to use multi-threaded applications, you have to solve this problem yourself. However, the Qt Framework has the advantage that it already supports multi-threading.

Therefore, when developing WebAssembly-based web applications, Qt for WebAssembly technology allows you to implement web applications that run in a web browser using C++ and Qt.

- Compared to an application built with Qt for WebAssembly



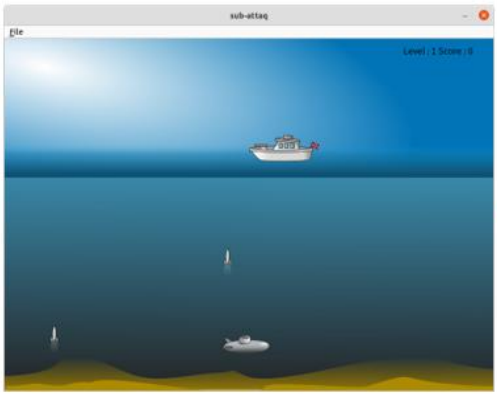
Build and run with a native compiler using Qt **Build and run with Qt for WebAssembly**

The run screen on the left is the one built and run in Desktop, and the one on the right is the one run in a web browser. Despite the different execution environments, both the left and right sides use the same

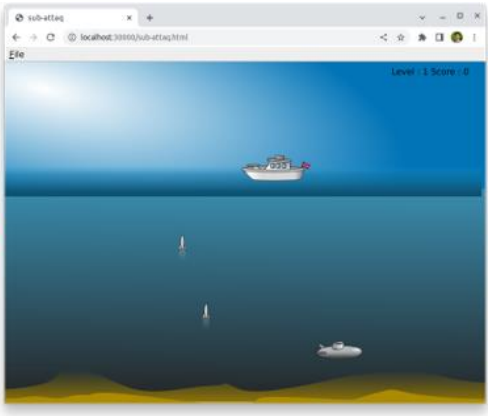
C++ source code and Qt libraries. If you need to create the same application that works in a web browser, you'll need to use a different web development language if you're not using C++/Qt.

However, with C++/Qt for WebAssembly, applications that work in a web browser can be implemented using C++ and Qt for WebAssembly.

The advantage of using C++ and Qt for WebAssembly is that you can use the same source code for applications that run in a web browser and applications that run on the desktop, without having to use different languages.



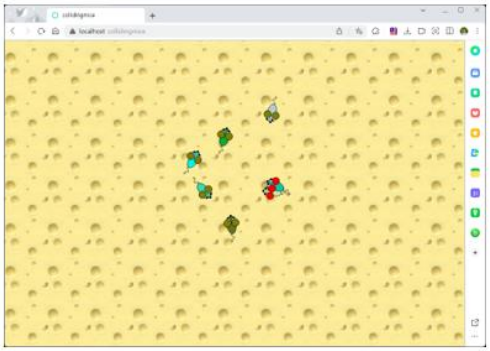
VS



Build and run with a native compiler using Qt in Ubuntu Linux

Build and run with Qt for WebAssembly in Ubuntu Linux

VS



Build and run with a native compiler using Qt in Chrome

Build and run with Qt for WebAssembly in Chrome

In conclusion, if you use C++ and Qt for WebAssembly to implement applications on the desktop without using PHP, ASP, JSP, Ruby, etc.

2. Building a development environment on MS Windows

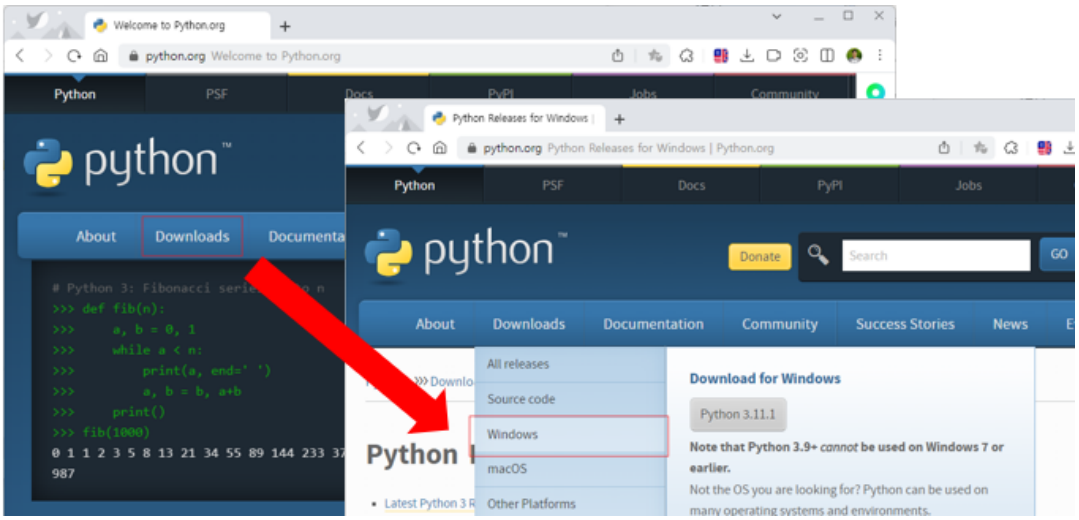
To build the development environment, you need to install the following items.

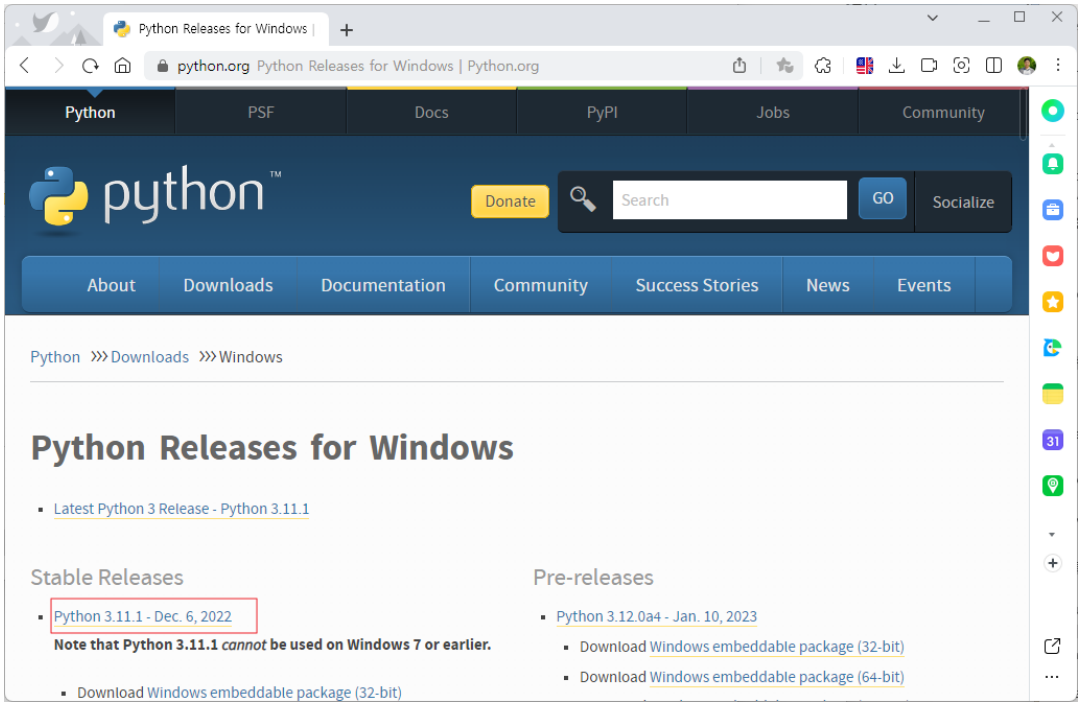
Item	Description
Python	Installing for use as a simple web server
Git	For downloading Emscripten to use as a standalone application
Emscripten	Install the SDK to build with WASM
Qt Framework	Qt version is Qt 6.5 LTS

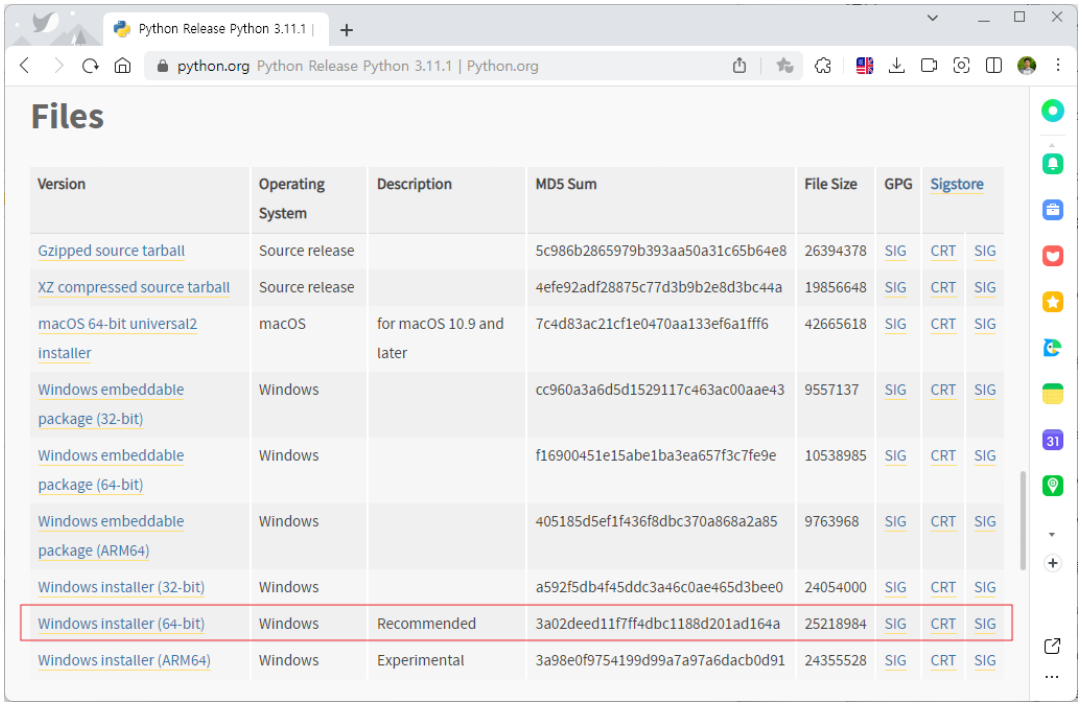
- Python Install

Python requires a web server to run with Web Bowser, which can be built directly with the Emscripten SDK before using Qt WebAssembly. Therefore, Python is required to use a simple web server. Qt does not use the web server provided by Python because Qt loads the web server automatically.

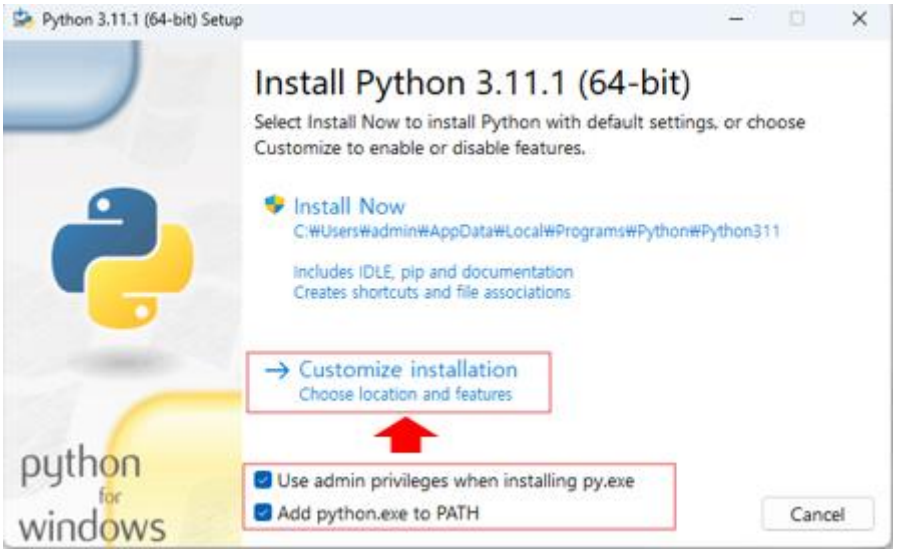
To download Python, visit python.org and download Python, as shown in the figure below.

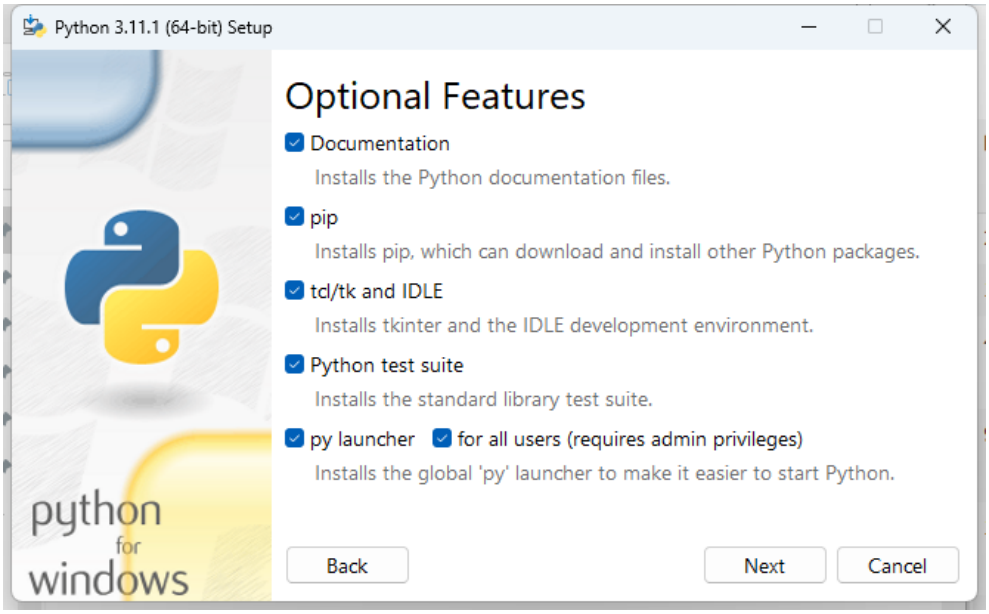


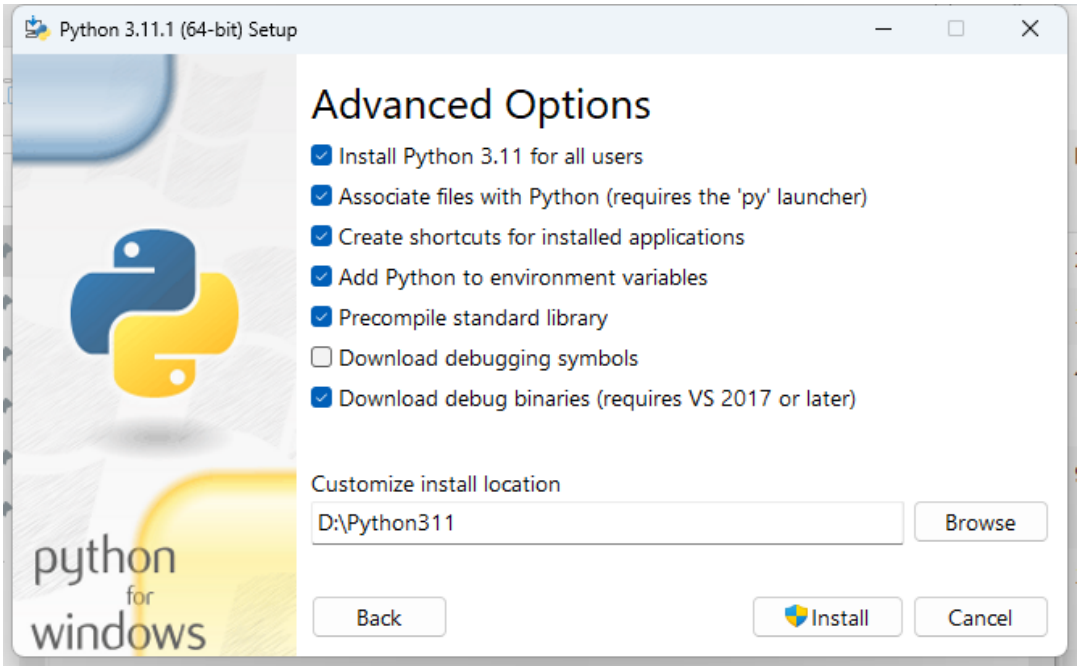


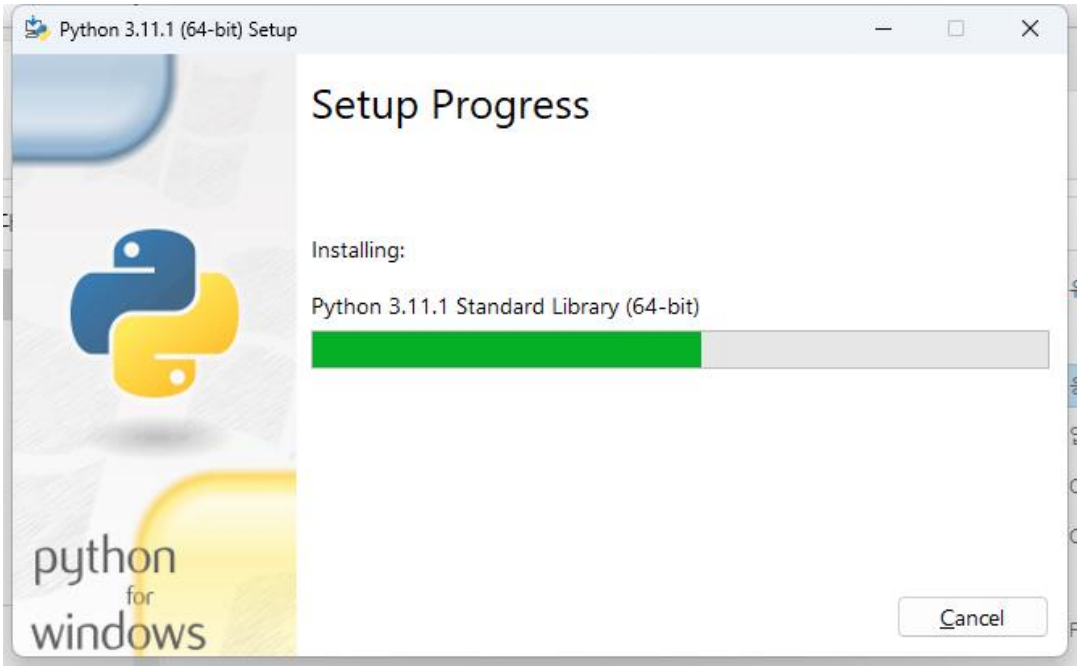


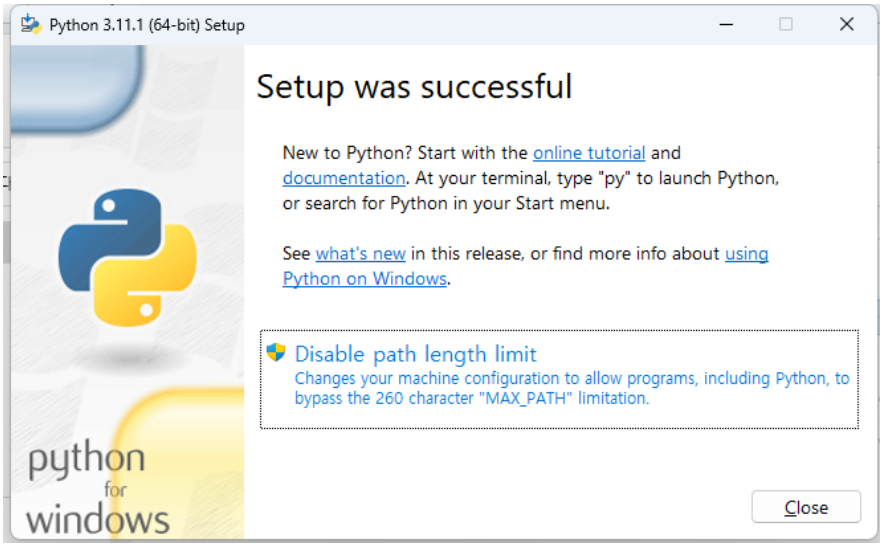
After downloading, run the installation file as shown below. Check all the checkboxes at the bottom of the dialog and click the [Customize installation] link button.


Python 3.11.1 (64-bit) Setup
Install Python 3.11.1 (64-bit)
Select Install Now to install Python with default settings, or choose
Customize to enable or disable features.
Install Now
C:\Users\admin\AppData\Local\Programs\Python\Python311
Includes IDLE, pip and documentation
Creates shortcuts and file associations
Customize installation
Choose location and features
python for windows
Use admin privileges when installing py.exe
Add python.exe to PATH
Cancel


Python 3.11.1 (64-bit) Setup
Optional Features
Documentation
Installs the Python documentation files.
pip
Installs pip, which can download and install other Python packages.
tcl/tk and IDLE
Installs tkinter and the IDLE development environment.
Python test suite
Installs the standard library test suite.
py launcher
for all users (requires admin privileges)
Installs the global 'py' launcher to make it easier to start Python.
python for windows
Back
Next
Cancel


Python 3.11.1 (64-bit) Setup
Advanced Options
Install Python 3.11 for all users
Associate files with Python (requires the 'py' launcher)
Create shortcuts for installed applications
Add Python to environment variables
Precompile standard library
Download debugging symbols
Download debug binaries (requires VS 2017 or later)
Customize install location
D:\Python311
Browse
Back
Install
Cancel
python
for
windows

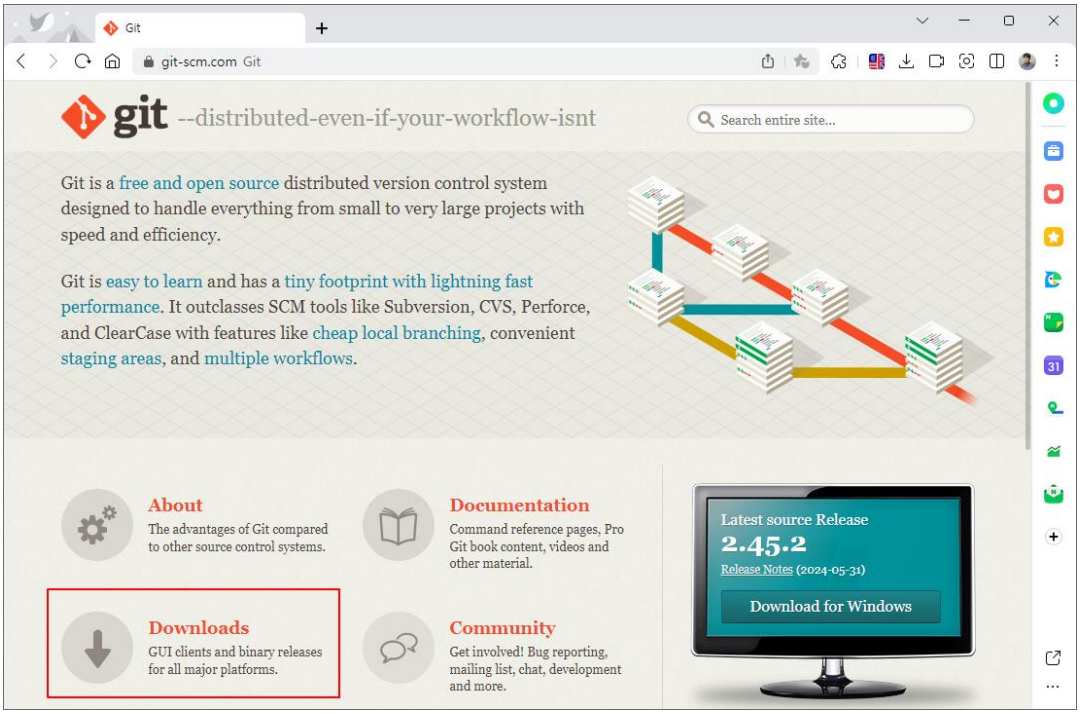

Python 3.11.1 (64-bit) Setup
Setup Progress
Installing:
Python 3.11.1 Standard Library (64-bit)
Cancel
python
for
windows



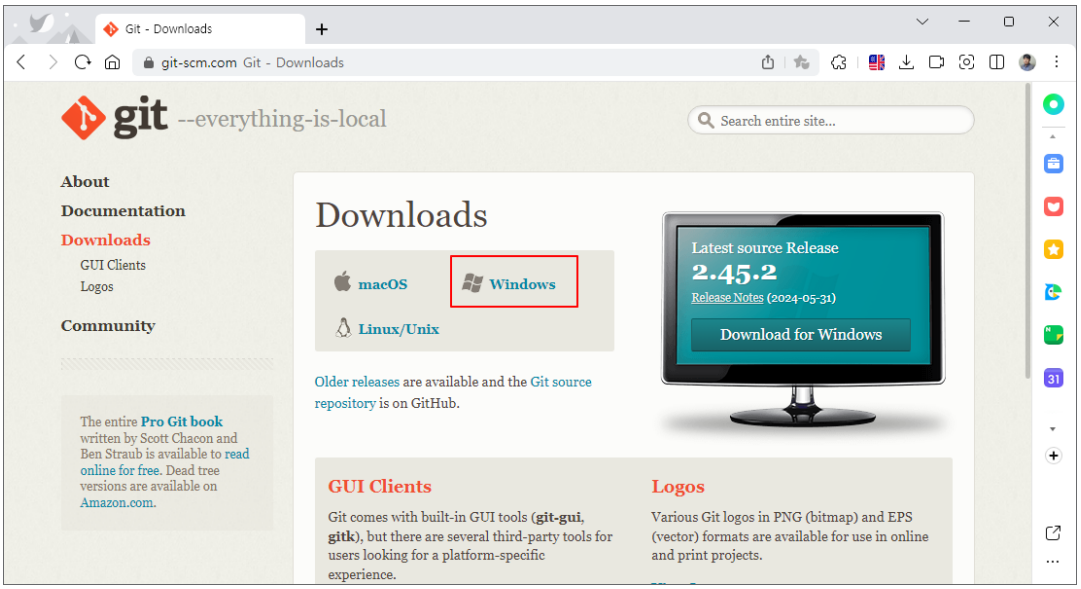
- Git install

Git is used to download Emscripten. However, if you don't use Git, you can still download Emscripten directly. Therefore, installing Git is not required.

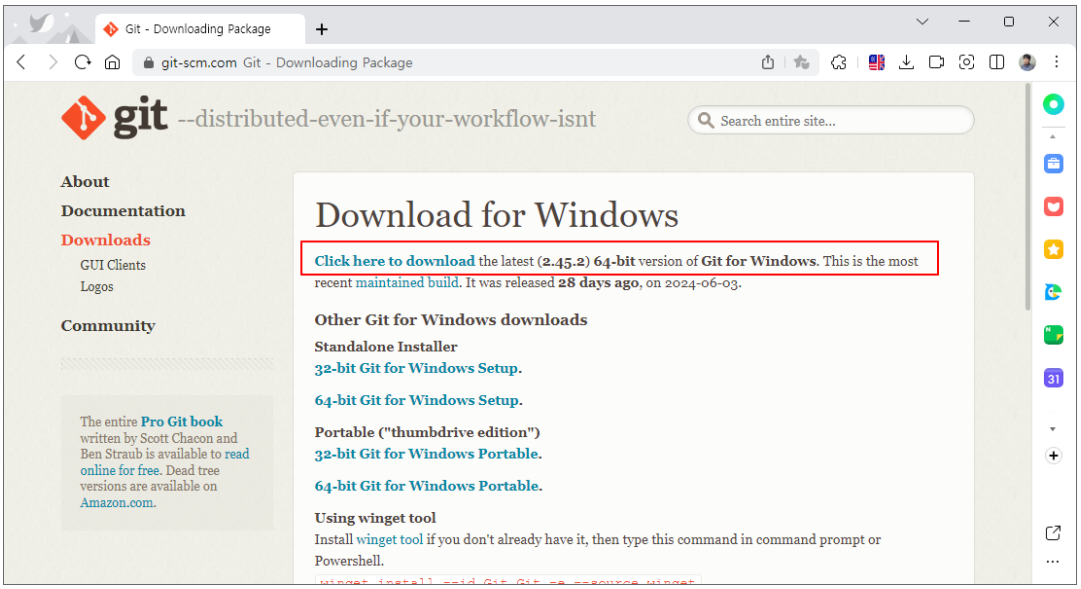
To install Git, go to git-scm.org and click the Downloads link as shown below.



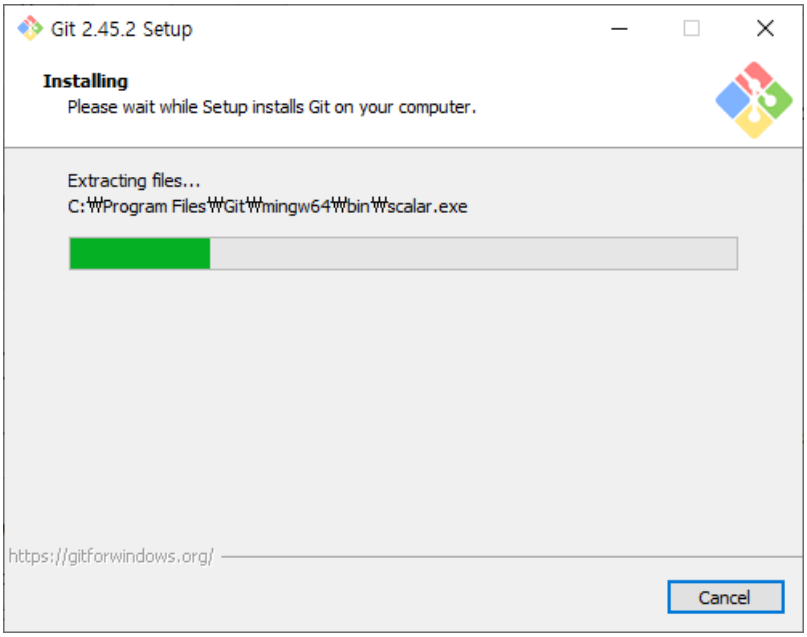
On the next page, click [Windows] link.



Click the [Click here to download] link as shown below.

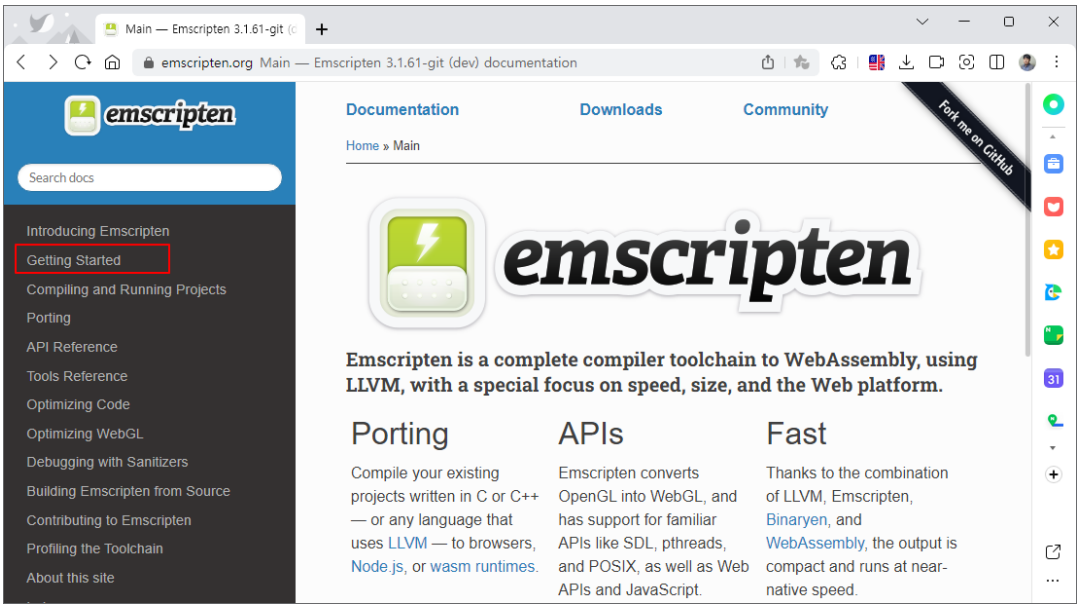


When the download is complete, run the installer as shown in the image below.

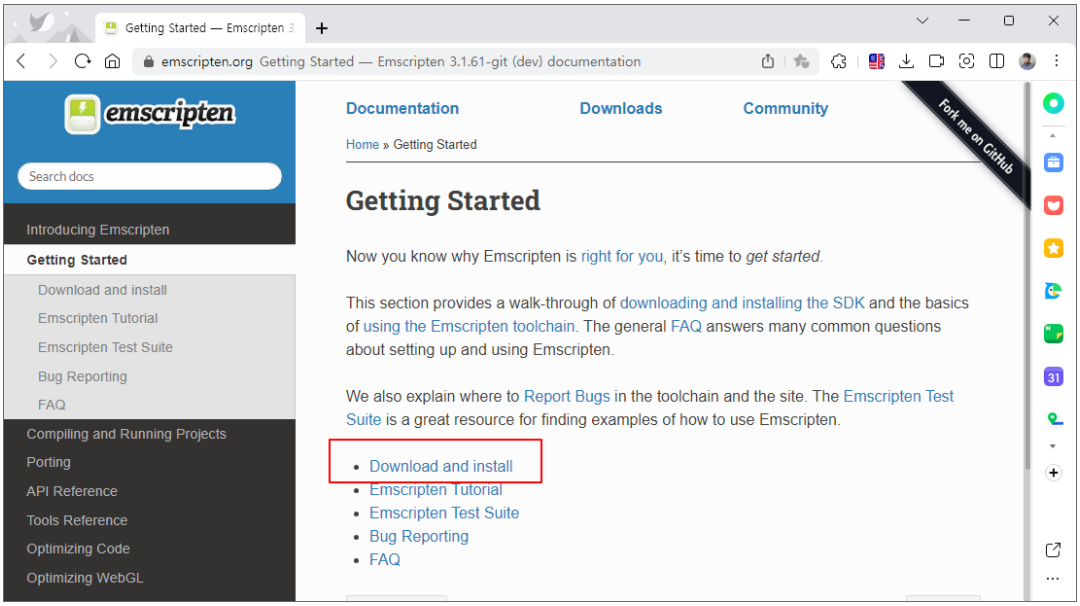


- Emscripten install

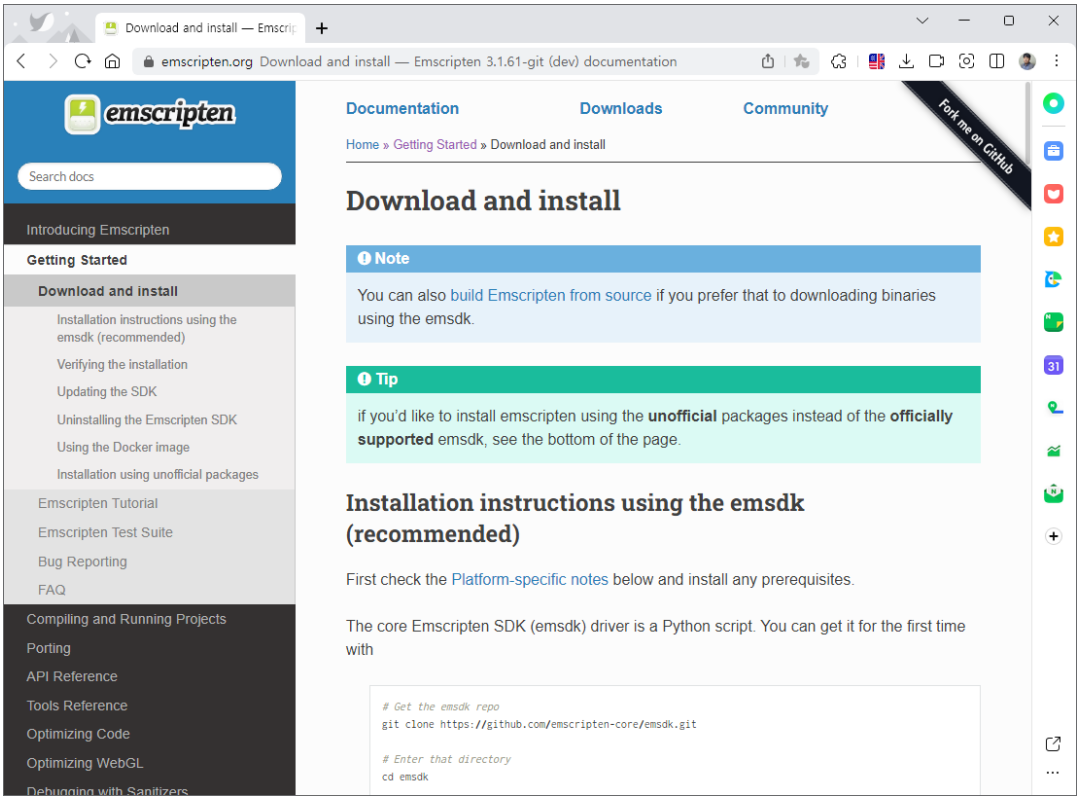
The Emscripten SDK is an SDK for generating WASM. It is located at emscripten.org.



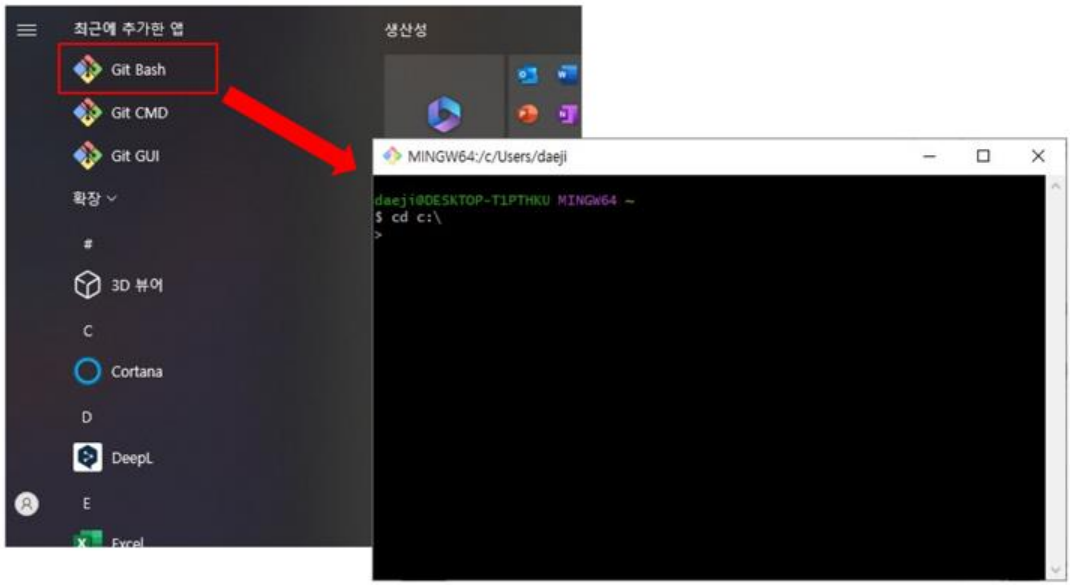
Click the Getting Started link, as shown on the page above.



Next, click the Download and install link.

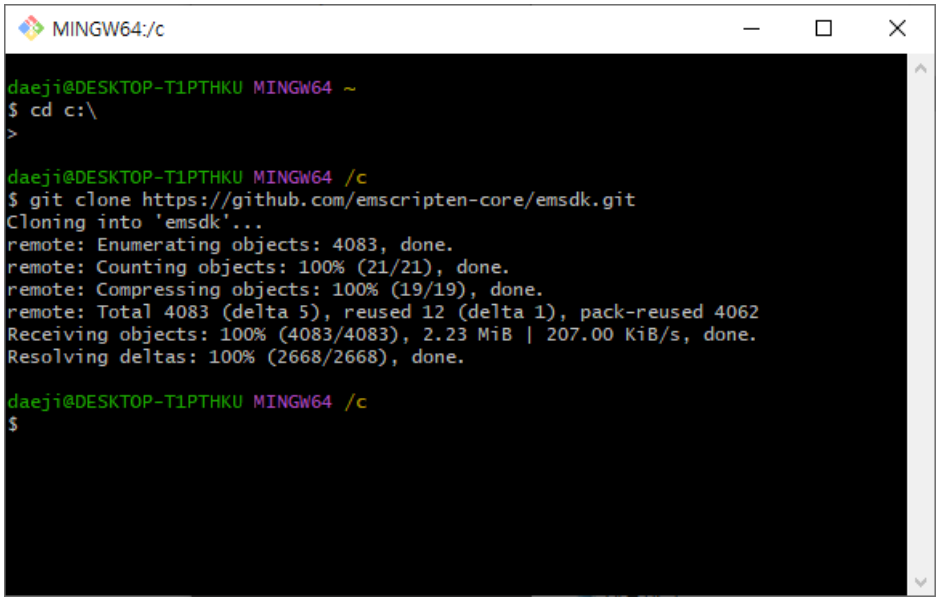


Launch the Git Bash window you installed, as shown on the page above.



Then enter the following

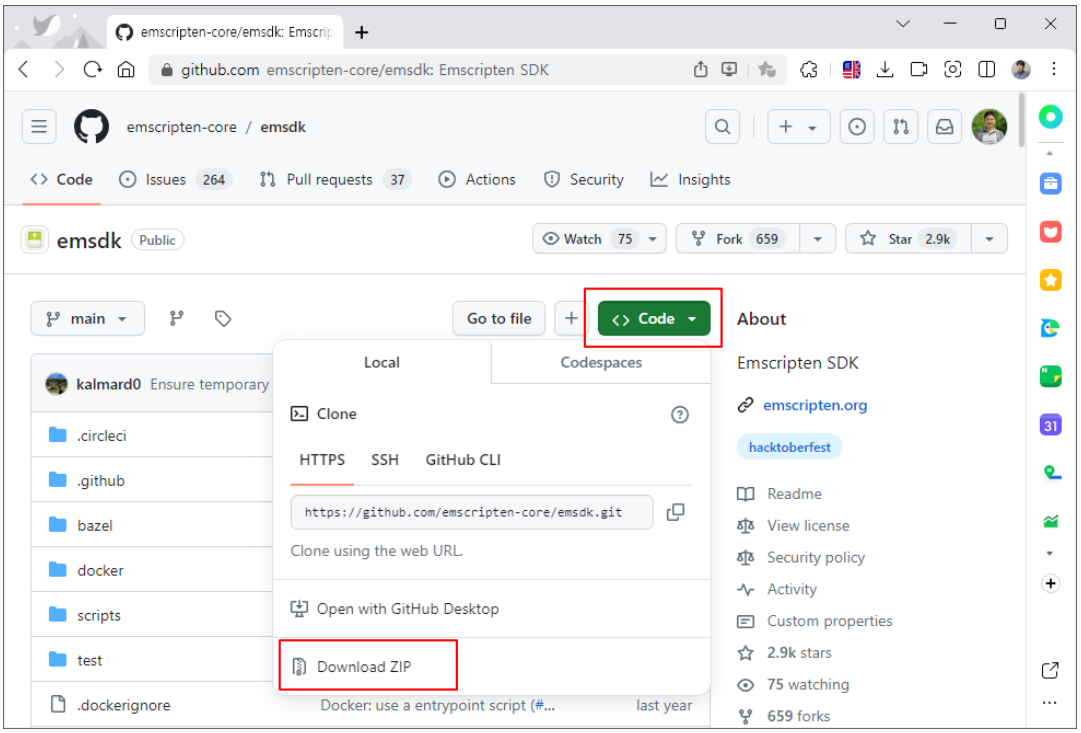
```
$ cd /d
$ git clone https://github.com/emscripten-core/emsdk.git
```



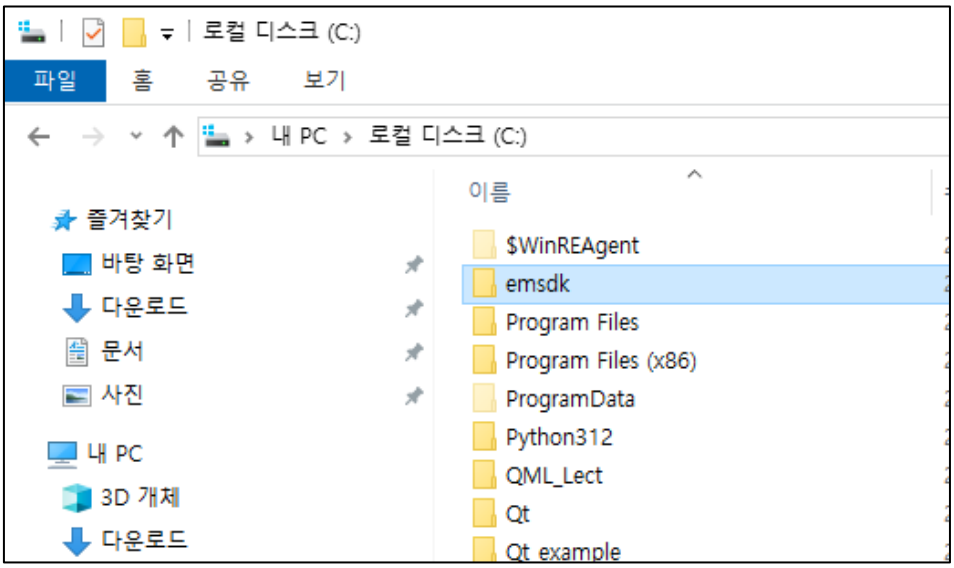
You can download it using Git, as shown in the screen shot above.

<if you're not using Git>

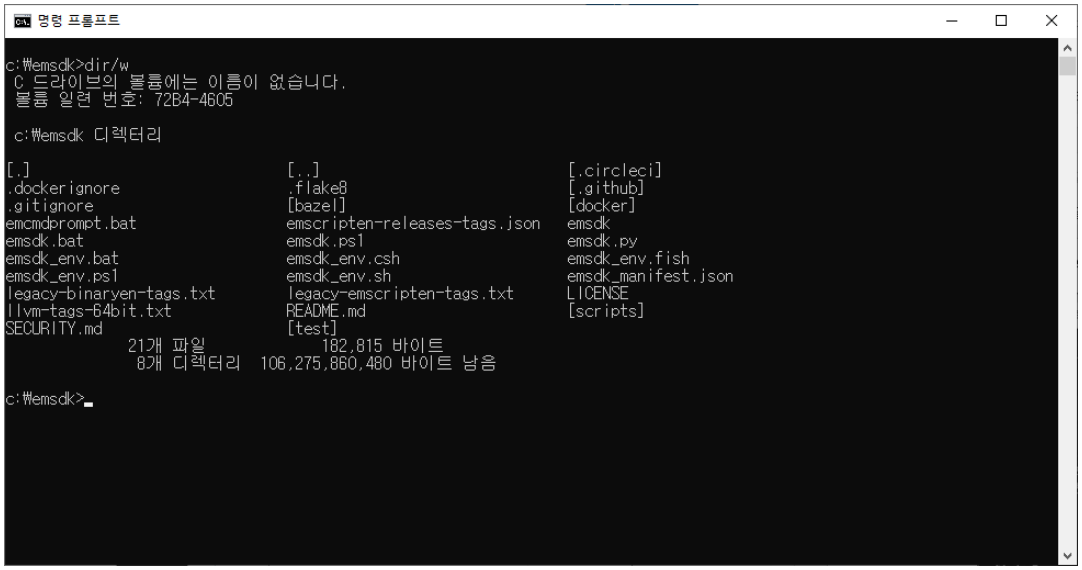
Visit https://github.com/emscripten-core/emsdk.git in your web browser. Then click the [Code] link, and then click the [Download ZIP] link, as shown below.



After downloading, rename the unzipped directory to emsdk.



Next, run a command prompt.



Then navigate to the emsdk directory as shown in the image above. Next, install the latest version from Emscripten as shown below.

```
D:₩emsdk> emsdk.bat install latest
...
Downloading: C:/emsdk/downloads/node-v18.20.3-win-x64.zip from https://storage.googleapis.com/webassembly/emscripten-releases-builds/deps/node-v18.20.3-win-x64.zip, 30476796 Bytes
Done installing tool 'node-18.20.3-64bit'.
Installing tool 'python-3.9.2-nuget-64bit'..
...
D:₩emsdk> emsdk.bat activate latest
Resolving SDK alias 'latest' to '3.1.61'
...
28e4a74b579b4157bda5fc34f23c7d3905a8bd6c-64bit'
Setting the following tools as active:
    node-18.20.3-64bit
    python-3.9.2-nuget-64bit
    java-8.152-64bit
    releases-28e4a74b579b4157bda5fc34f23c7d3905a8bd6c-64bit
...
```

Once you have installed the final version as shown above, make sure you have the latest version installed as shown below.

```
D:₩emsdk> emsdk_env.bat
Setting up EMSDK environment (suppress these messages with EMSDK_QUIET=1)
Setting environment variables:
...
D:₩emsdk> emcc --version
...
D:₩emsdk> em++ --version
emcc (Emscripten gcc/clang-like replacement + linker emulating GNU ld) 3.1.61 (67fa4c16496b157a7fc3377afd69ee0445e8a6e3)
Copyright (C) 2014 the Emscripten authors (see AUTHORS.txt)
This is free and open source software under the MIT license.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Next, before using Qt, let's write the source code to output a simple "Hello World" by hand, and then build and run it using the Emscripten SDK directly. Create a hello.c source file as shown below.

```
#include <stdio.h>
int main()
{
	printf("Hello World ~ ^^; ₩n");
		return 1;
}
```

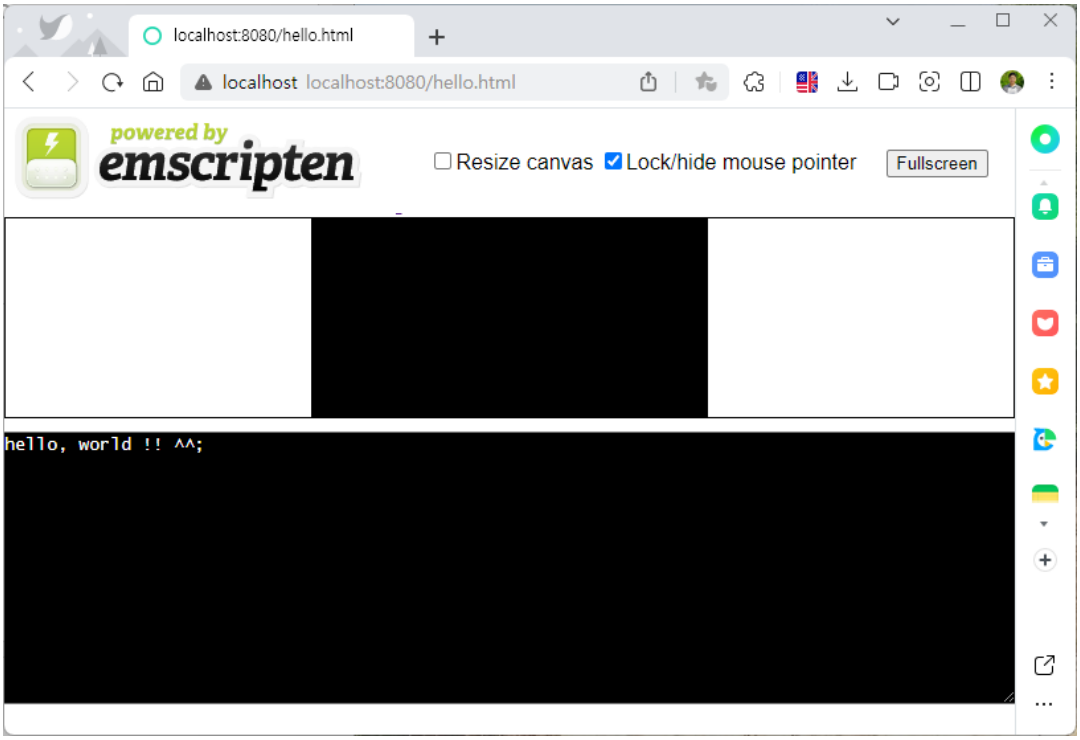
Write it like above and then build it like below.

```
D:₩Examples₩00_hello_world_c_lang> dir
```

```
2024-07-03 02:37    <DIR>          .
2024-07-03 02:37    <DIR>          ..
2024-07-01 11:11                93 hello.c

D:₩Examples₩00_hello_world_c_lang> emcc hello.c -s WASM=1 -o hello.html

D:₩Examples₩00_hello_world_c_lang> dir

2024-07-03 02:37    <DIR>          .
2024-07-03 02:37    <DIR>          ..
2024-07-01 11:11                93 hello.c
2024-07-03 02:37           103,800 hello.html
2024-07-03 02:37            68,982 hello.js
2024-07-03 02:37            12,091 hello.wasm
```

Next, load the webserver provided by Python in the same directory.

```
D:₩Examples₩00_hello_world_c_lang> python -m http.server 8080
Serving HTTP on :: port 8080 (http://[::]:8080/) ...
```



You can see the Hello World output in the web browser as shown above. Next, install the Qt Framework.

- Using the web server provided by the Emscripten SDK

In addition to the web server functionality provided by Python, the emrun command provided by the Emscripten SDK allows you to run HTML files through a web server and read them in a web browser. The following example shows how to run HTML files in a specific directory through a web server.

```
emrun --browser chrome d:₩root₩index.html
```

The name after --browser is the name of the web browser. You can retrieve the list of supported web browsers by running the following command

```
emrun –list_browsers
```

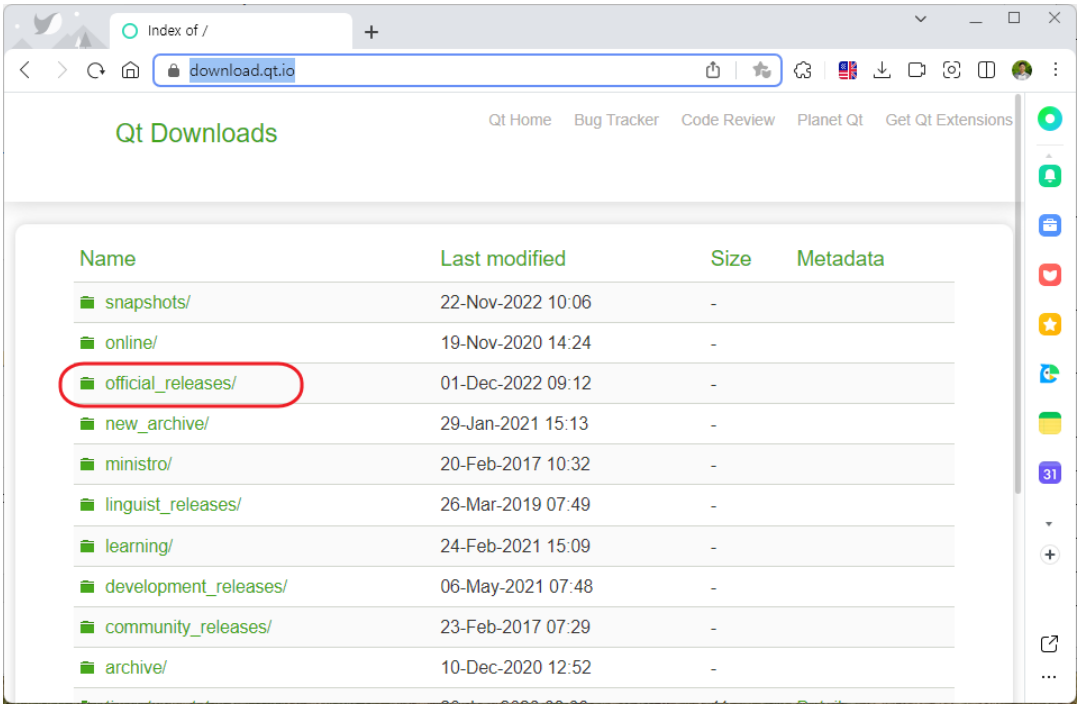
The last directory name and file name are the directory location and file name of the HTML file you want to run through the web server. When executed as above, the d:₩root₩index.html file will be executed in the Google Chrome Web Browser, just like

accessing a web server.

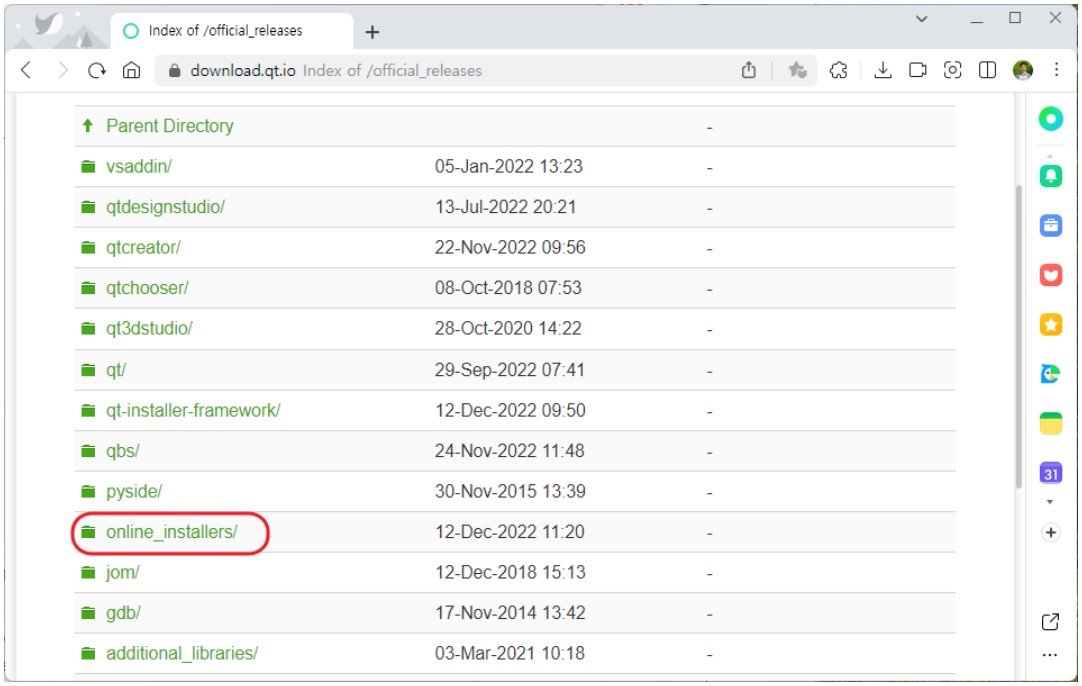
- Qt Framework install

For Qt version, install 6.5.x LTS. To install Qt version 6.5.x, visit the site as shown below.

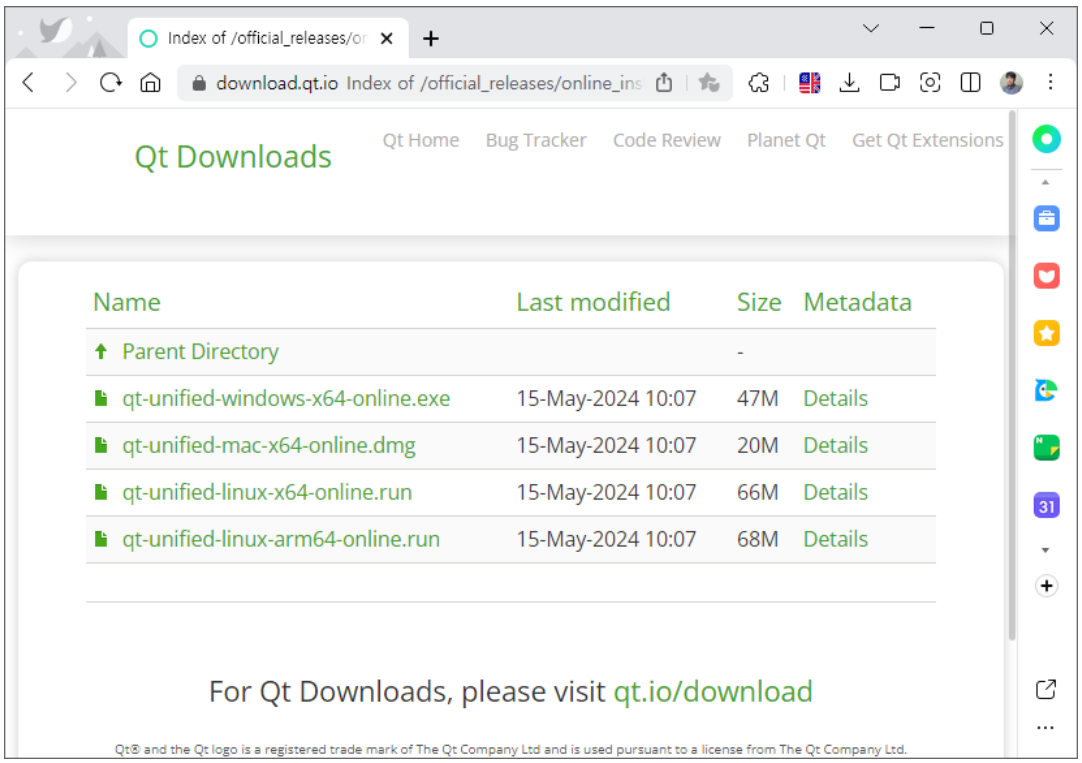
Site URL: https://download.qt.io



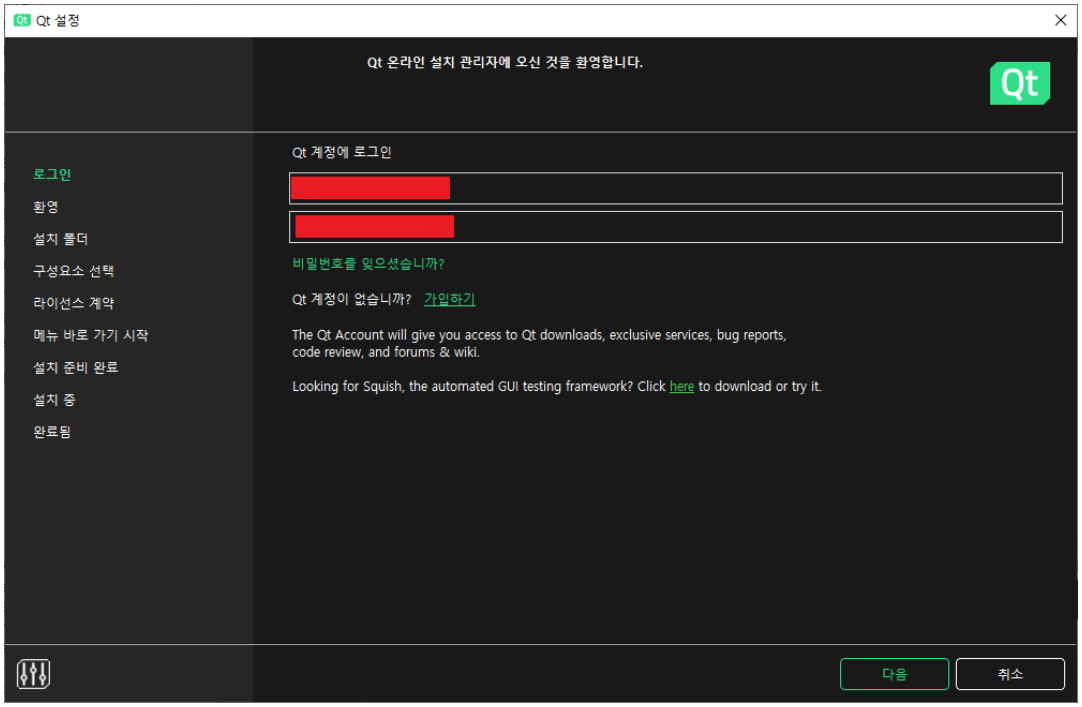
Next, click the [online_installers] link.



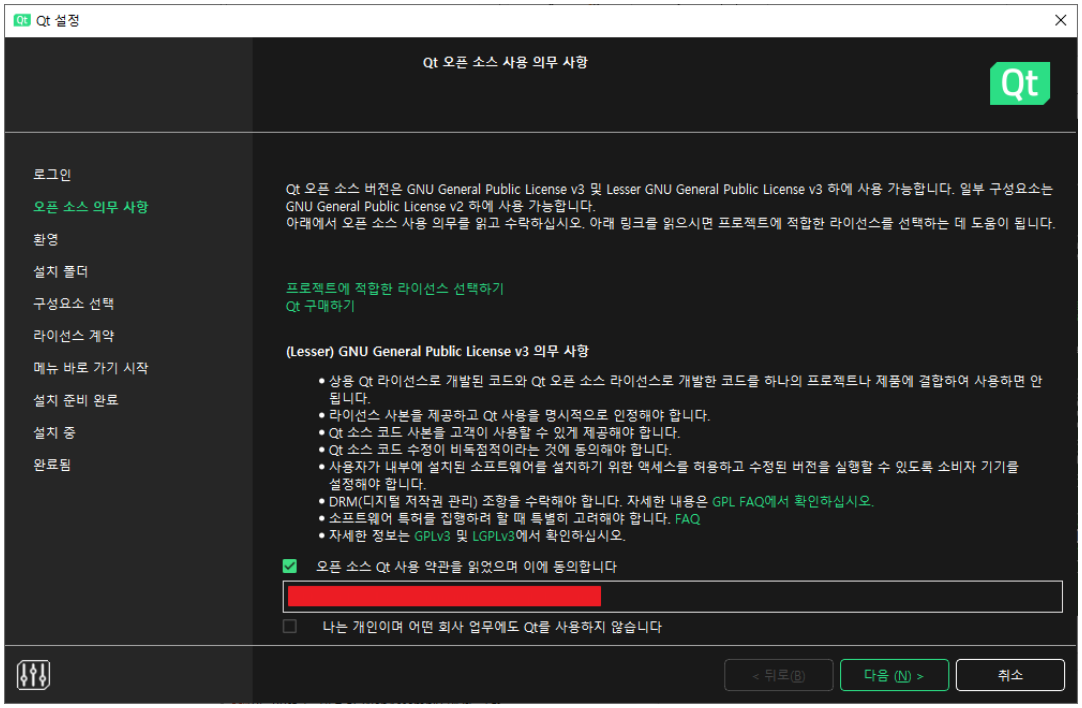
And you will download a file with the exe extension.



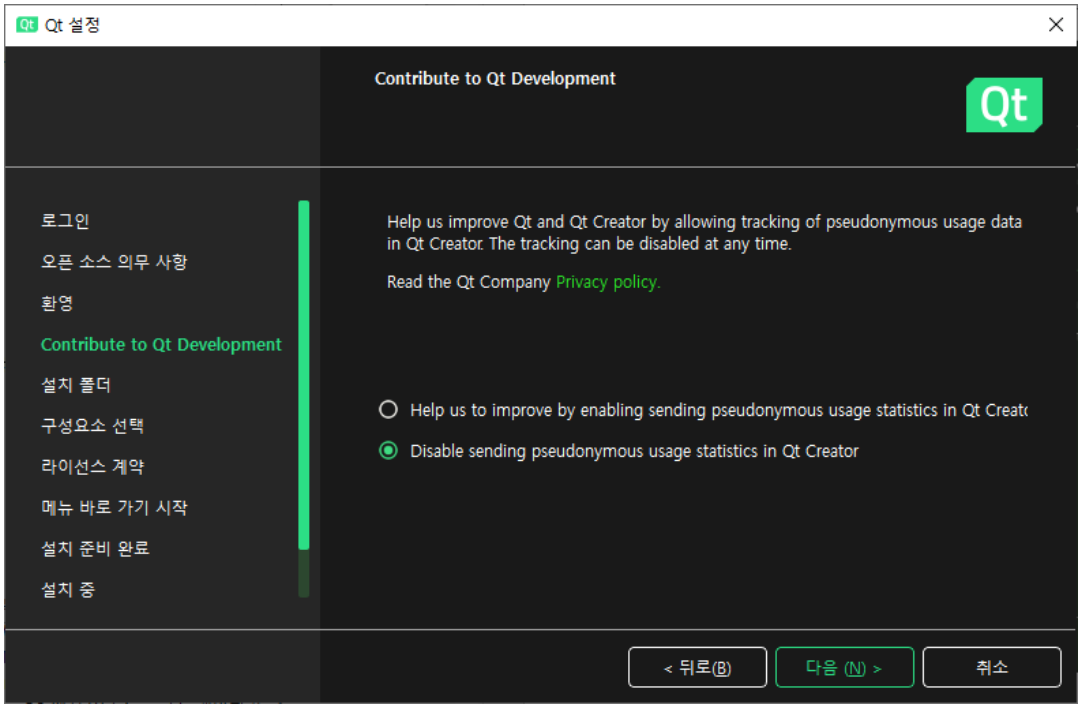
After the download is complete, run the downloaded file. Enter your account in the Qt account field and click the Next button at the bottom.

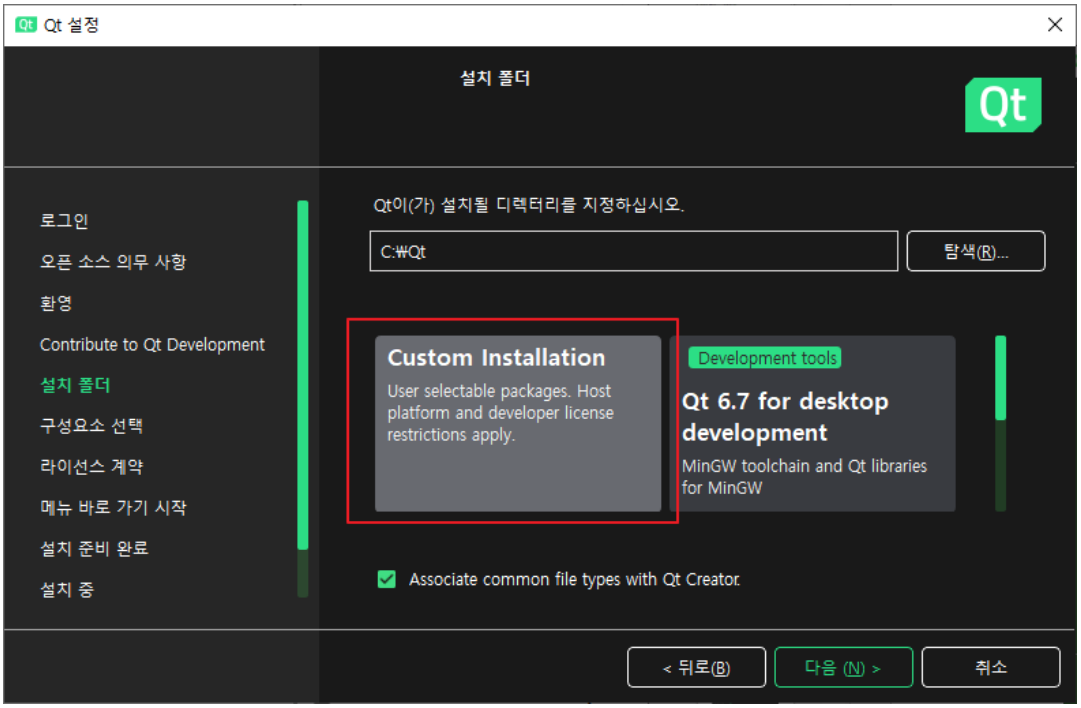


If you don't have an account, click the Sign Up link above.



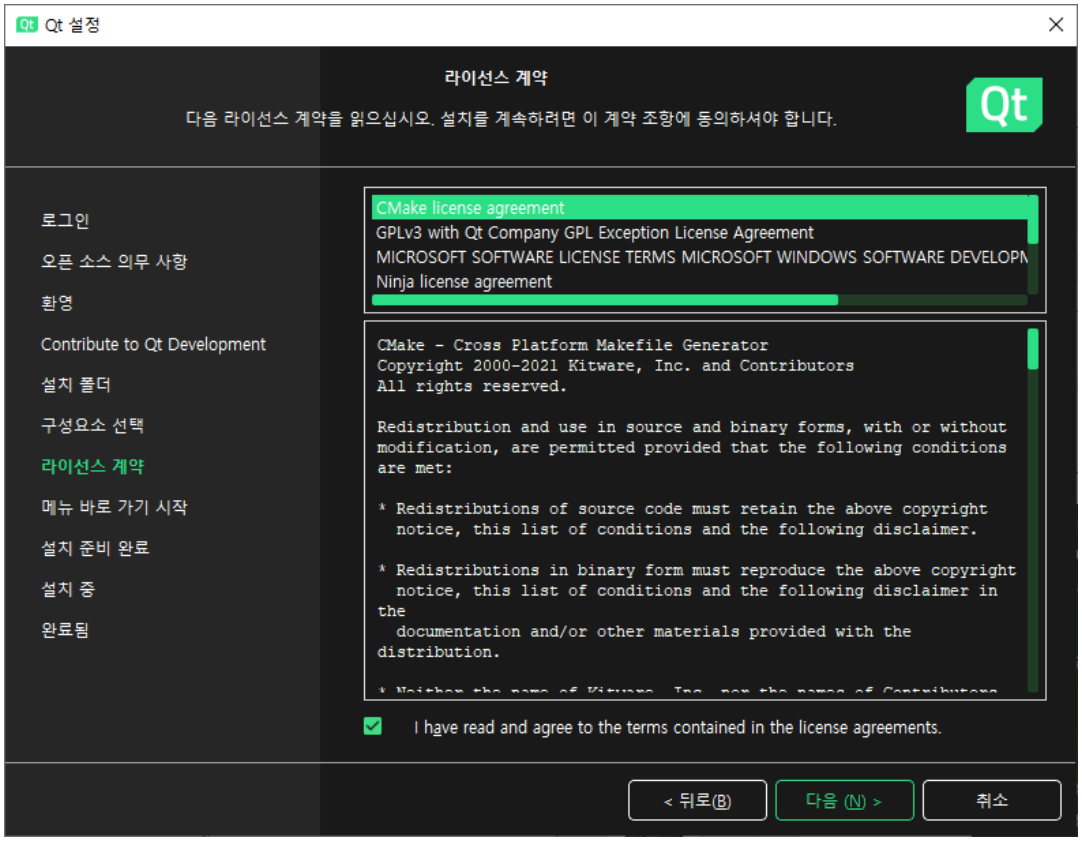

Qt 설정
환영
Qt
로그인
오픈 소스 의무 사항
환영
설치 폴더
구성요소 선택
라이선스 계약
메뉴 바로 가기 시작
설치 준비 완료
설치 중
완료됨
오픈 소스 Qt 설정에 오신 것을 환영합니다.
< 뒤로(B)
다음 (N) >
종료(Q)

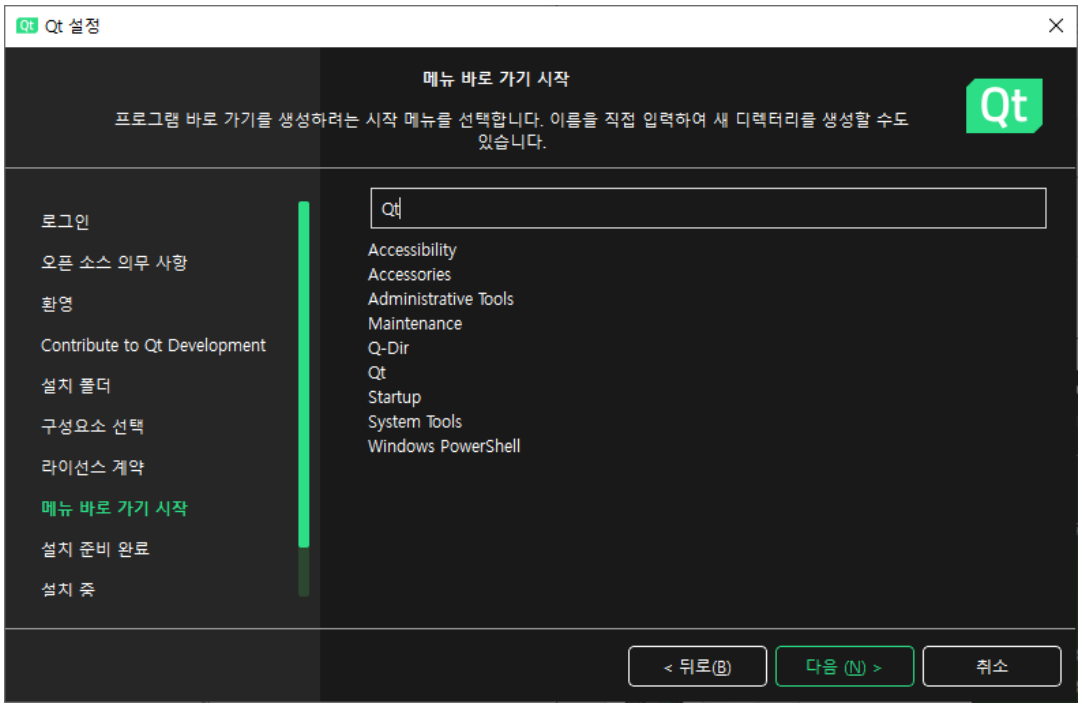

Qt 설정
Contribute to Qt Development
Qt
로그인
오픈 소스 의무 사항
환영
Contribute to Qt Development
설치 폴더
구성요소 선택
라이선스 계약
메뉴 바로 가기 시작
설치 준비 완료
설치 중
Help us improve Qt and Qt Creator by allowing tracking of pseudonymous usage data in Qt Creator. The tracking can be disabled at any time.
Read the Qt Company Privacy policy.
Help us to improve by enabling sending pseudonymous usage statistics in Qt Creat
Disable sending pseudonymous usage statistics in Qt Creator
< 뒤로(B)
다음 (N) >
취소

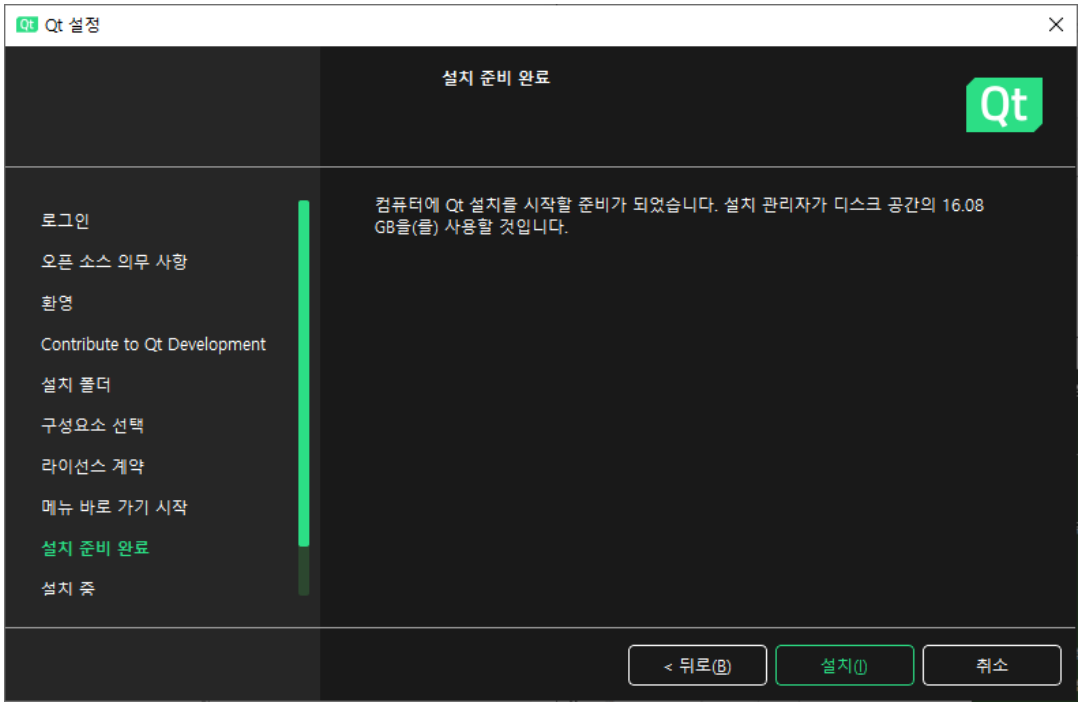


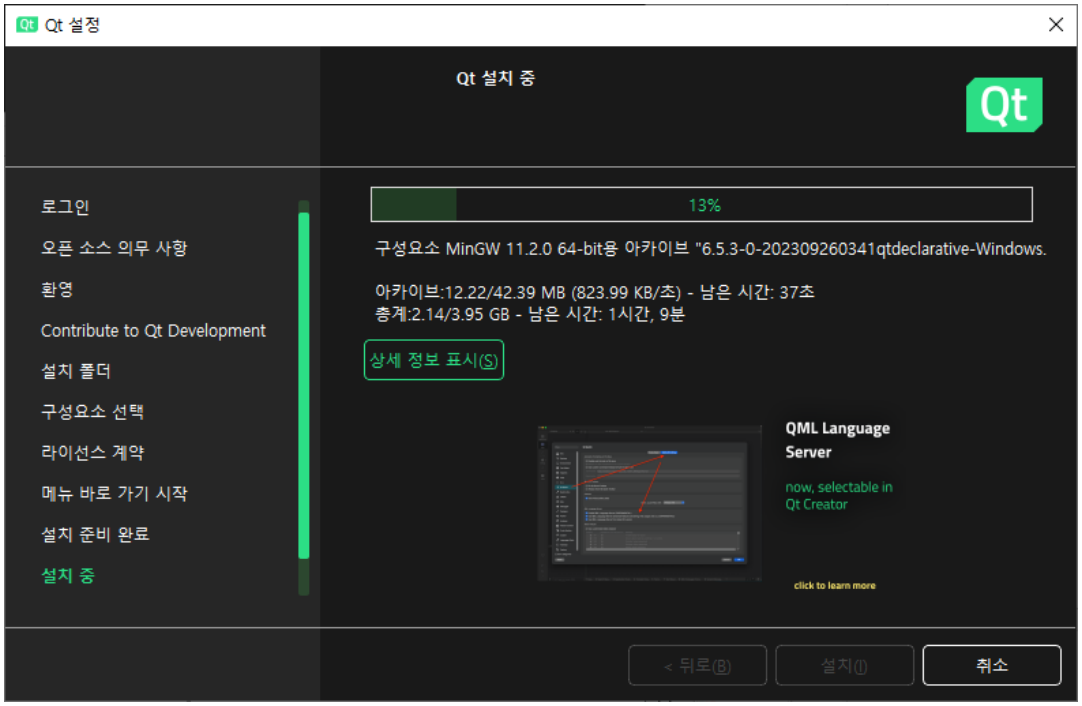
As shown in the screen above, select the [Custom Installation] item and click [Next]. On the next screen, select the following items.

Select the items as shown above, and then click the [Next] button.


Qt 설정
라이선스 계약
다음 라이선스 계약을 읽으십시오. 설치를 계속하려면 이 계약 조항에 동의하셔야 합니다.
로그인
오픈 소스 의무 사항
환영
Contribute to Qt Development
설치 폴더
구성요소 선택
라이선스 계약
메뉴 바로 가기 시작
설치 준비 완료
설치 중
완료됨
CMake license agreement
GPLv3 with Qt Company GPL Exception License Agreement
MICROSOFT SOFTWARE LICENSE TERMS MICROSOFT WINDOWS SOFTWARE DEVELOPN
Ninja license agreement
CMake - Cross Platform Makefile Generator
Copyright 2000-2021 Kitware, Inc. and Contributors
All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
I have read and agree to the terms contained in the license agreements.
< 뒤로(B)
다음 (N) >
취소


Qt 설정
메뉴 바로 가기 시작
프로그램 바로 가기를 생성하려는 시작 메뉴를 선택합니다. 이름을 직접 입력하여 새 디렉터리를 생성할 수도 있습니다.
로그인
오픈 소스 의무 사항
환영
Contribute to Qt Development
설치 폴더
구성요소 선택
라이선스 계약
메뉴 바로 가기 시작
설치 준비 완료
설치 중
Qt
Accessibility
Accessories
Administrative Tools
Maintenance
Q-Dir
Qt
Startup
System Tools
Windows PowerShell
< 뒤로(B)
다음 (N) >
취소

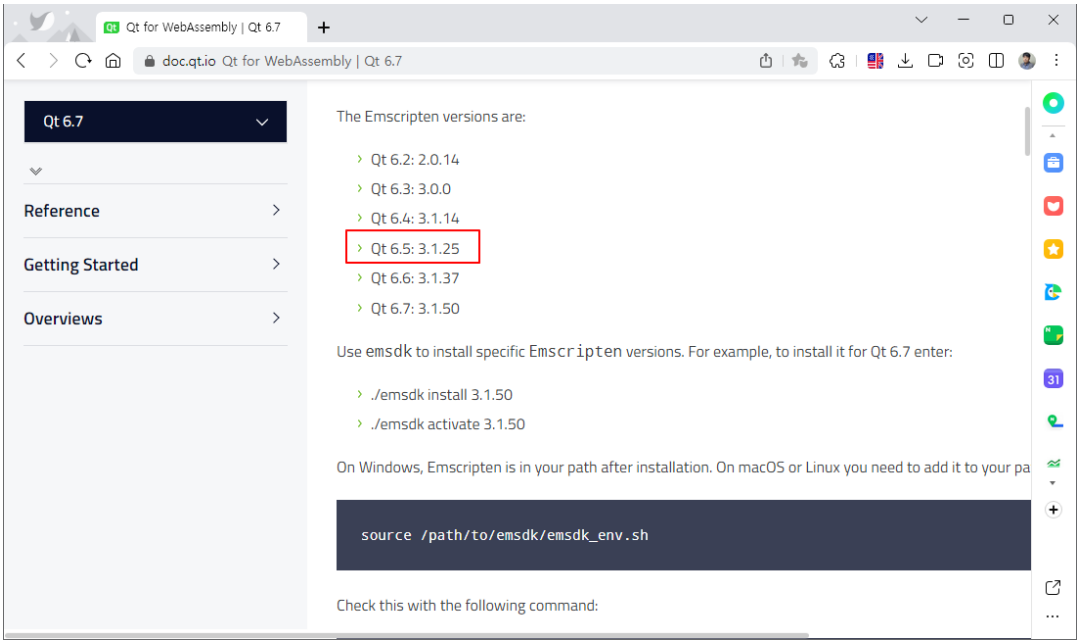




After the Qt installation is complete, you need to reinstall the Emscripten version for your Qt version. Previously, you installed the latest version, but you need to install a version of Emscripten that is compatible with Qt 6.5.x LTS. You can check the compatibility

information for each Qt version at the following sites.

- Site addresses to check by version: https://doc.qt.io/qt-6/wasm.html



As shown in the figure above, the version of Emscripten that is compatible with Qt 6.5 LTS is version 3.1.25.

D:₩emsdk> **emsdk.bat install 3.1.25**

...

D:₩emsdk> **emsdk.bat activate 3.1.25**

...

D:₩emsdk> **emsdk_env.bat**

...

D:₩emsdk> **em++ --version**

emcc (Emscripten gcc/clang-like replacement + linker emulating GNU ld) **3.1.25** (febd44b21ecaca86e2cb2a25ef3ed4a0a2076365)

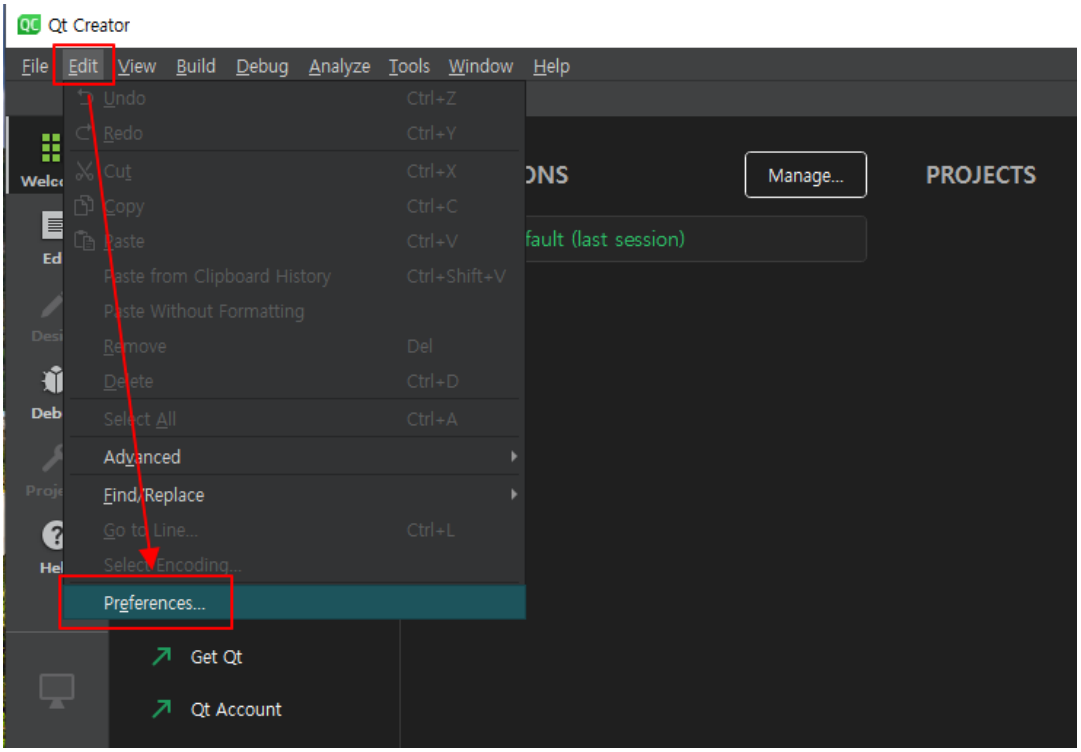
Copyright (C) 2014 the Emscripten authors (see AUTHORS.txt)

This is free and open source software under the MIT license.

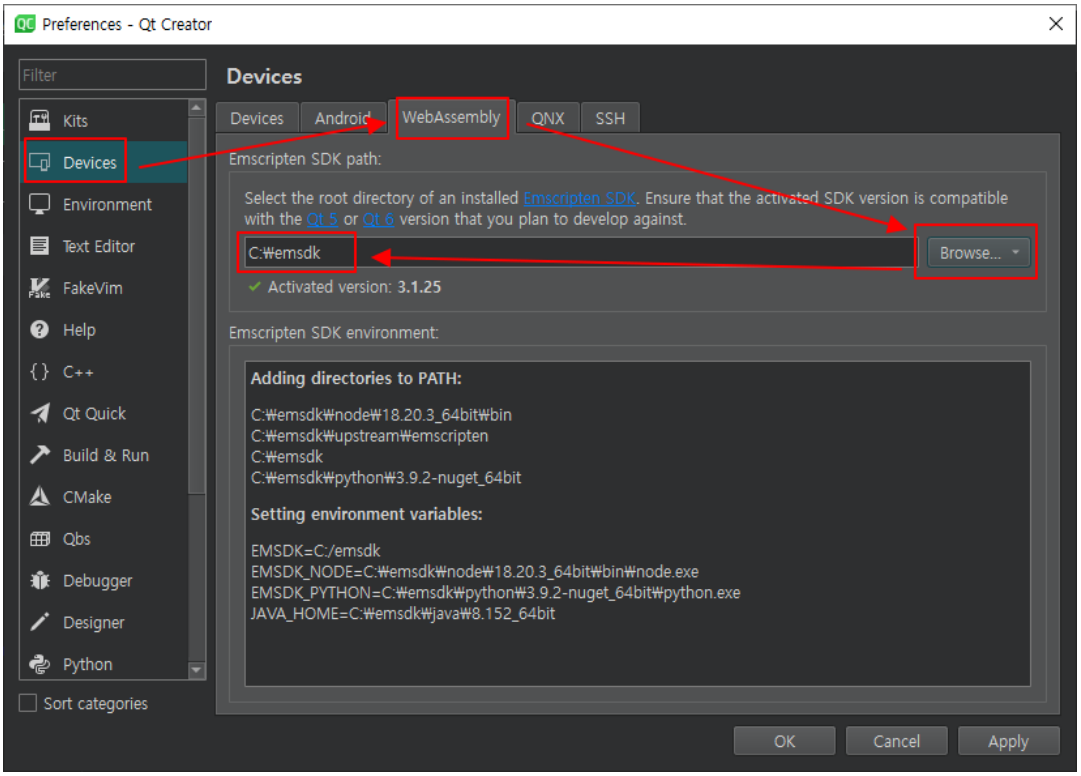
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

If you see version 3.1.25 in the versioning information as shown above, the installation

went well. Next, run Qt Creator.



[Edit] -> [Preferences...] Click



[Devices] -> [WebAssembly] -> [Browser] Click and specify the directory where Emscripten is installed.

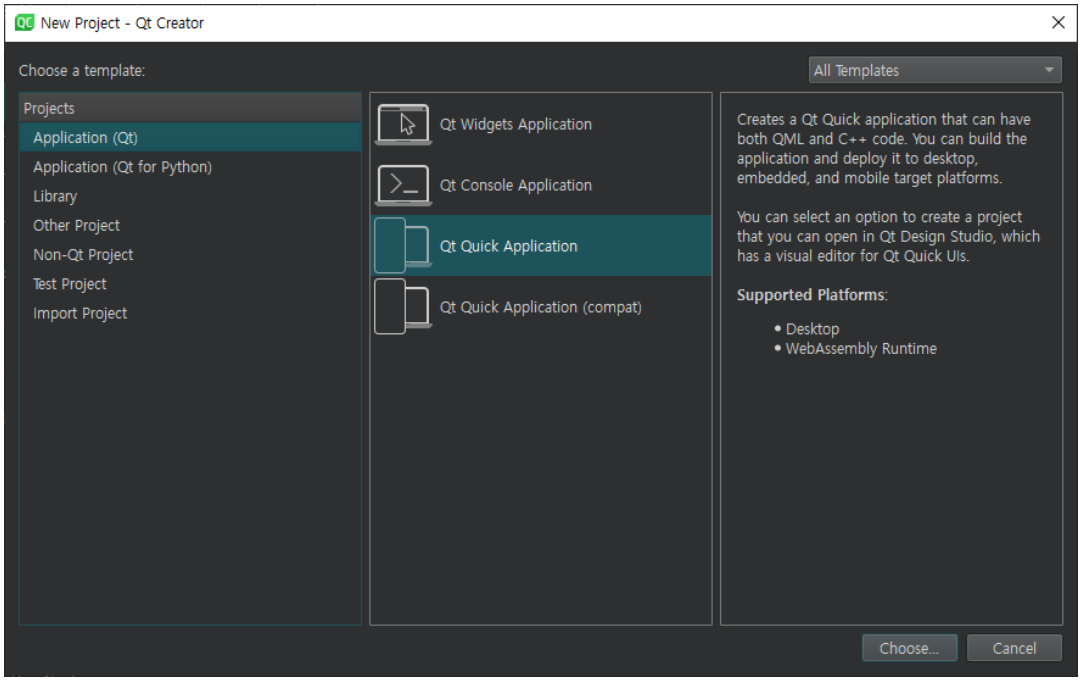
If you set the message [Activated version: 3.1.25] as shown above, you're all set.

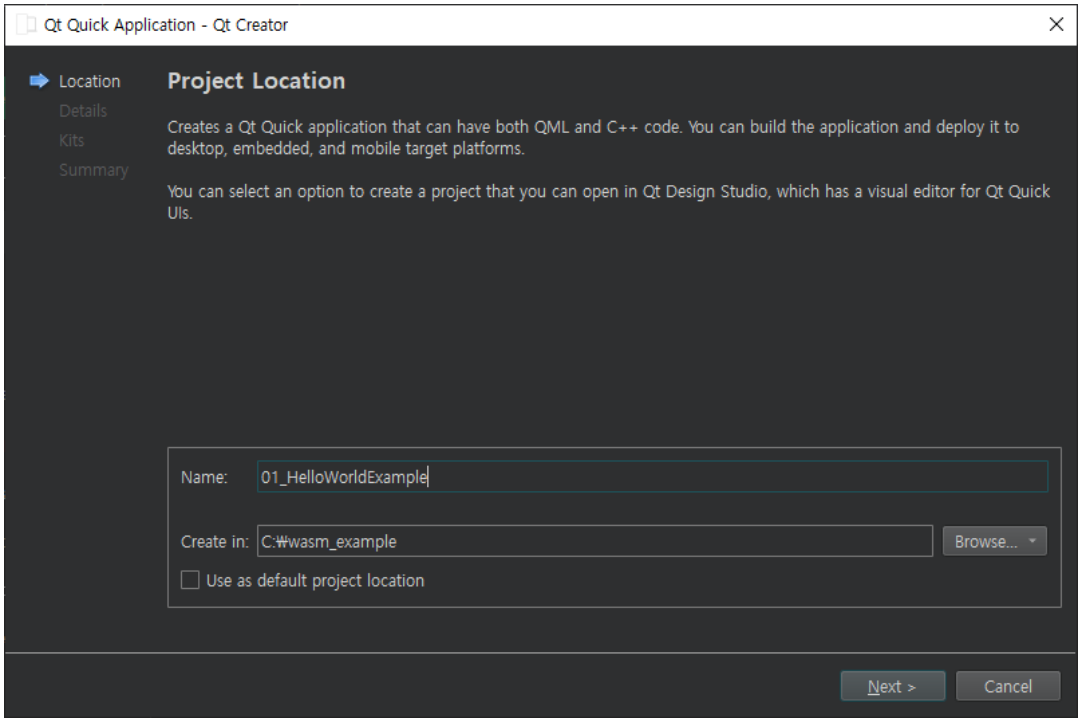
- Hello World output example

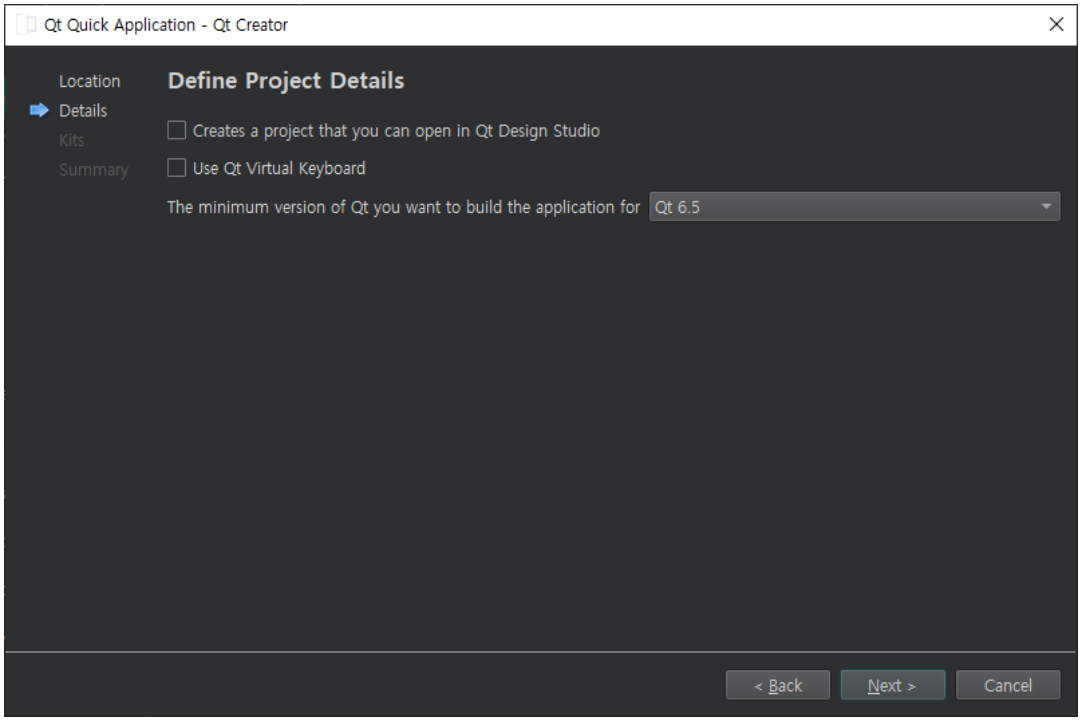
This time, we'll write an example that prints Hello World in a web browser to give you a secondary run-through.

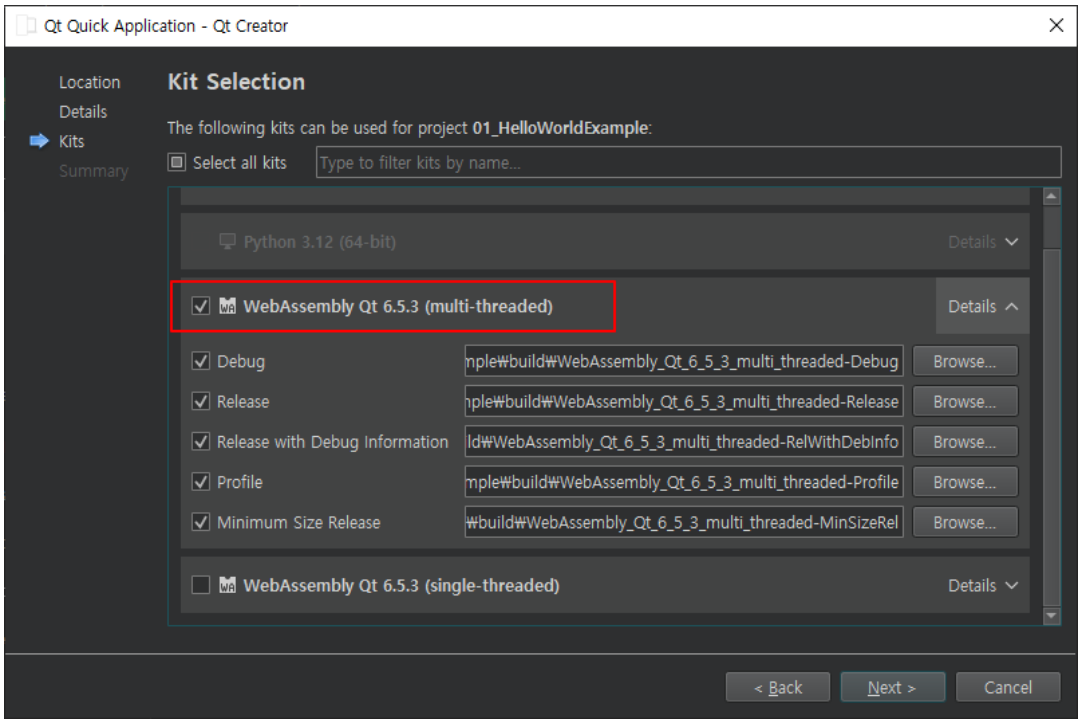
the example.

Create a project in the Qt Creator window. In the dialog shown below, click the [Application Qt] item and select [Qt Quick Application].

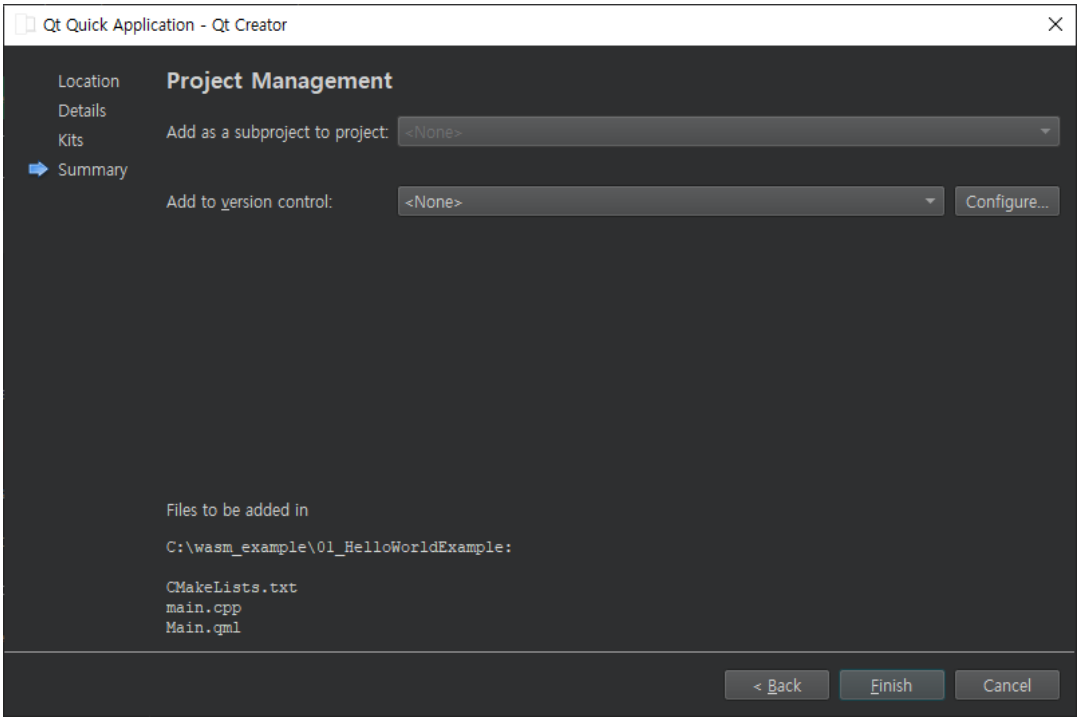

New Project - Qt Creator
Choose a template:
All Templates
Projects
Application (Qt)
Application (Qt for Python)
Library
Other Project
Non-Qt Project
Test Project
Import Project
Qt Widgets Application
Qt Console Application
Qt Quick Application
Qt Quick Application (compat)
Creates a Qt Quick application that can have both QML and C++ code. You can build the application and deploy it to desktop, embedded, and mobile target platforms.
You can select an option to create a project that you can open in Qt Design Studio, which has a visual editor for Qt Quick UIs.
Supported Platforms:
• Desktop
• WebAssembly Runtime
Choose...
Cancel


Qt Quick Application - Qt Creator
Location
Details
Kits
Summary
Project Location
Creates a Qt Quick application that can have both QML and C++ code. You can build the application and deploy it to desktop, embedded, and mobile target platforms.
You can select an option to create a project that you can open in Qt Design Studio, which has a visual editor for Qt Quick UIs.
Name:
01_HelloWorldExample
Create in:
C:₩wasm_example
Browse...
Use as default project location
Next >
Cancel





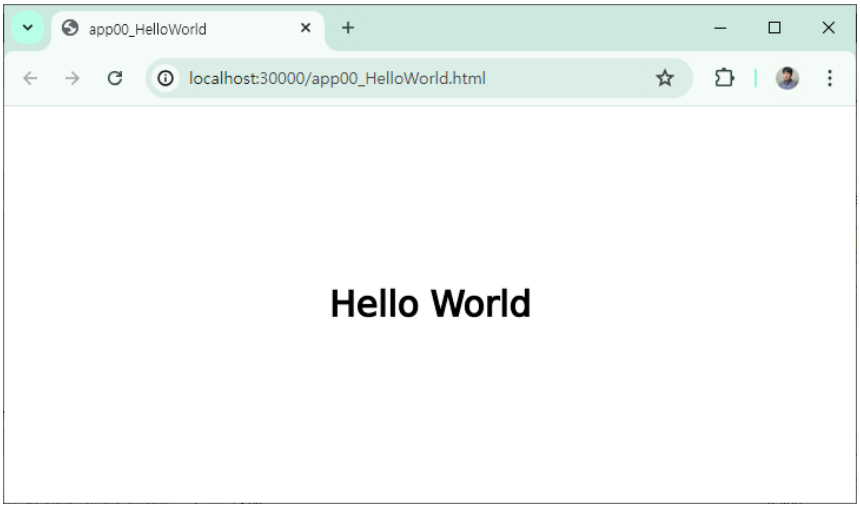
Select the [WebAssembly Qt 6.5.3 (multi-threaded)] item, as shown in the image above.



Once the project is created, open the Main.qml file and write the source code as shown below.

```
import QtQuick

Window {
    width: 640; height: 480; visible: true
    Text {
        anchors.centerIn: parent
        text: "Hello World"
        font.pixelSize: 30
        font.bold: true
    }
}
```

Let's write the source code as above and then build and run it.



If the build completes successfully, you should see your web browser running. If you go to the directory where the build was completed, you can see that the WASM file has been created. To run the WASM file, you also need a Web Server to create the HTML file and run the WASM.

Qt Creator runs a debugging Web server for you to debug, so Qt for WebAssembly conveniently provides everything you need to debug WebAssembly.

3. Building a development environment on Linux

To build the Qt for WebAssembly development environment on Linux, we will use Ubuntu Linux 20.04 LTS. First, we need to install the packages required to build the Qt for Web Assembly environment as shown below.

- Updating after installing Ubuntu Linux

```
$ sudo apt update
```

- g++ and make install

```
$ sudo apt install build-essential make
```

- OpenGL install

```
$ sudo apt install freeglut3-dev libglu1-mesa-dev mesa-common-dev
```

- libxcb install

```
$ sudo apt install libxcb*
```

- Python install

```
$ sudo apt install -y python3-pip
$ pip3 install package_name
$ sudo apt install -y libssl-dev libffi-dev python3-dev
$ sudo apt install -y python3-venv
$ pip install flask
$ pip install pymongo dnspython
```

- Git install

```
$ sudo apt install git
```

- clang install

```
$ sudo apt install clang
```

Next, we need to install the Emscripten SDK. Navigate to your home directory and download it using Git as shown below.

- Emsdk download and install

```
$ git clone https://github.com/emscripten-core/emsdk.git
```

Once the clone is complete as shown above, install the Emscripten SDK as follows.

```
$ cd emsdk
$ git pull
$ ./emsdk install 3.1.25
$ ./emsdk activate 3.1.25
$ source ./emsdk_env.sh
```

Next, let's use emcc to print out the version information to make sure the installation went well.

```
$ emcc –v
emcc (Emscripten gcc/clang-like replacement + linker emulating GNU ld) 3.1.25
(febd44b21ecaca86e2cb2a25ef3ed4a0a2076365)
clang version 16.0.0 (https://github.com/llvm/llvm-project)
Target: wasm32-unknown-emscripten
Thread model: posix
InstalledDir: /home/dev/emsdk/upstream/bin
```

- Script to run automatically at boot time

To make the emsdk script run automatically when booting on Linux, make the following modifications.

```
$ sudo vi /etc/profile
```

Next, add the following lines to your HOME/.profile file

```
source /home/dev/emsdk/emsdk_env.sh
```

Next, let's write an example that prints Hello world, build it using the Job Emscripten SDK, and run it in a web browser.

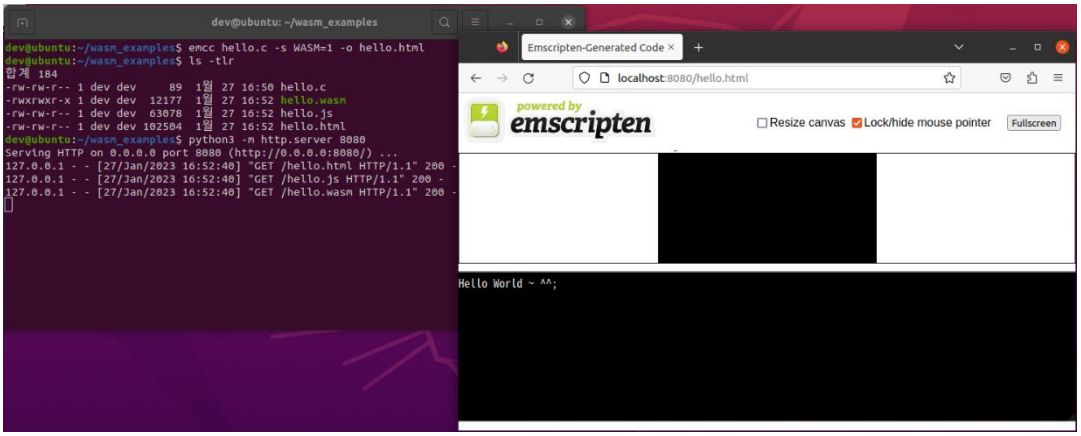
Create a hello.c source file and add the following code to it.

```
#include <stdio.h>
int main()
{
        printf("Hello World ~ ^^; ₩n");
        return 1;
}
```

Next, after building, load the web server provided by Python as shown below.

```
$ emcc hello.c -s WASM=1 -o hello.html
shared:INFO: (Emscripten: Running sanity checks)
cache:INFO: generating system asset: ...
cache:INFO:  - ok
$ ls -tlr
합계 448
-rw-rw-r-- 1 dev dev      477  00:17 hello.cpp
-rwxrwxr-x 1 dev dev 158679  00:19 hello.wasm
-rw-rw-r-- 1 dev dev 184873  00:19 hello.js
-rw-rw-r-- 1 dev dev 102507  00:19 hello.html

$ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

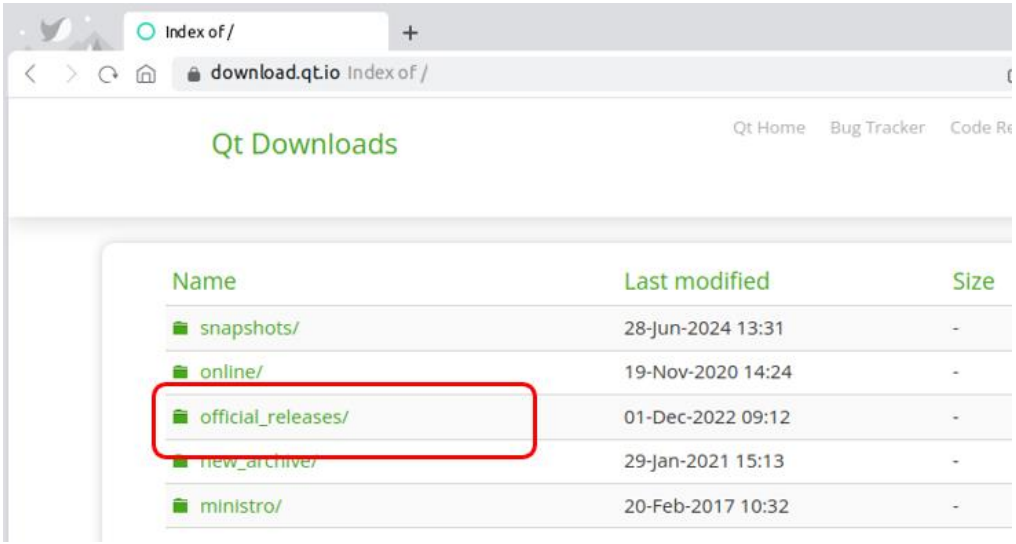
Next, let's try running the URL http://localhost:8080/hello.html using a web browser.



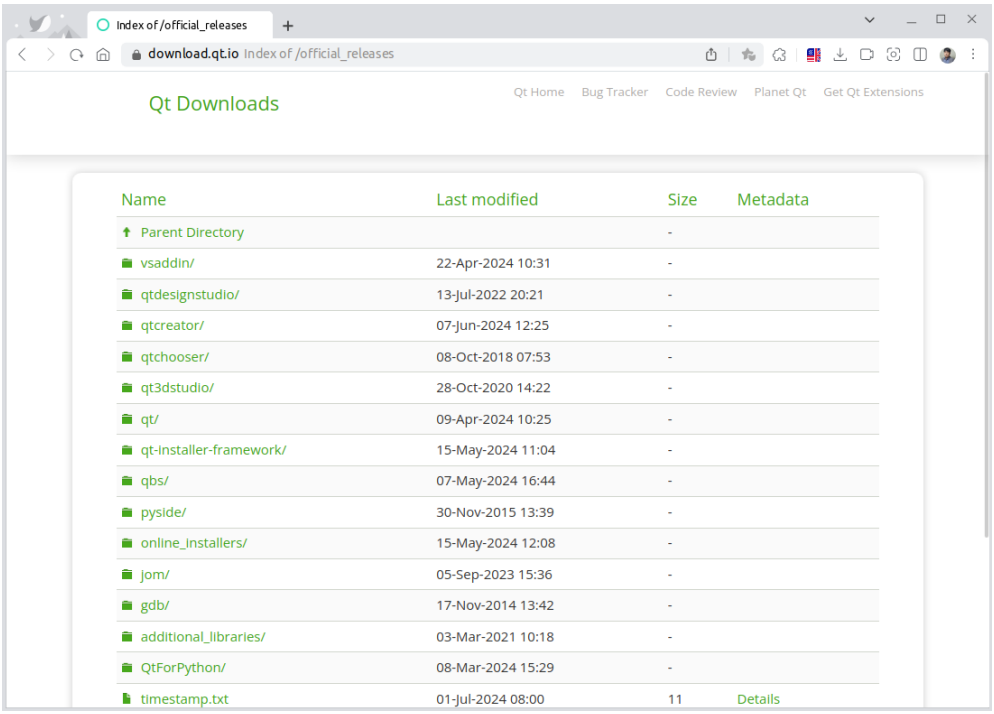
Next, let's install the Qt Framework.

- Qt Framework install

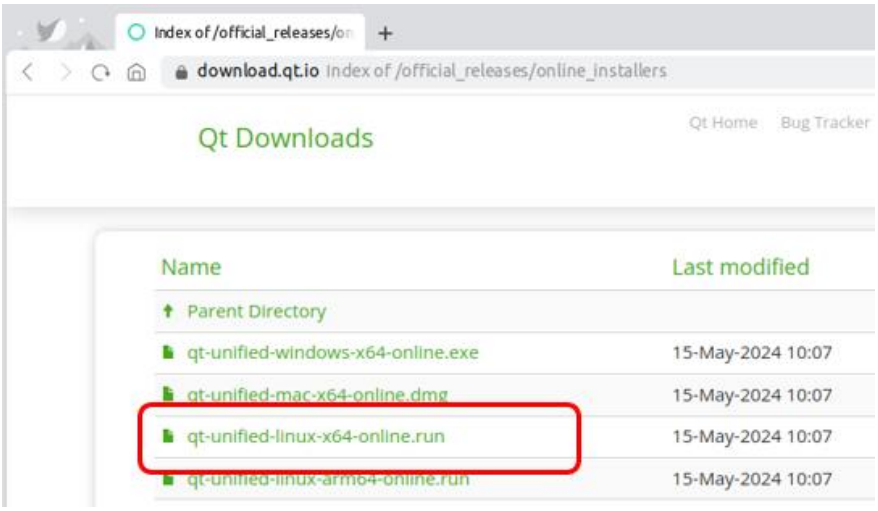
To install the Qt Framework, download the online installation file. The online installation file can be downloaded from the URL https://download.qt.io.



Click the [official_releases] link, as shown in the image above. Next, click the [online_installers] link, as shown in the image below.

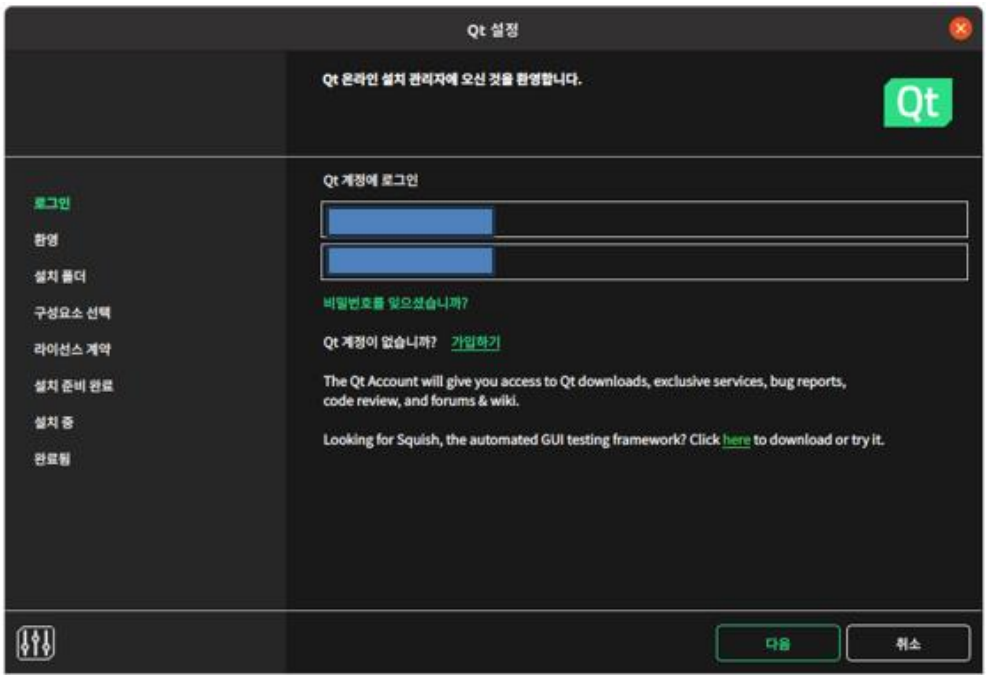


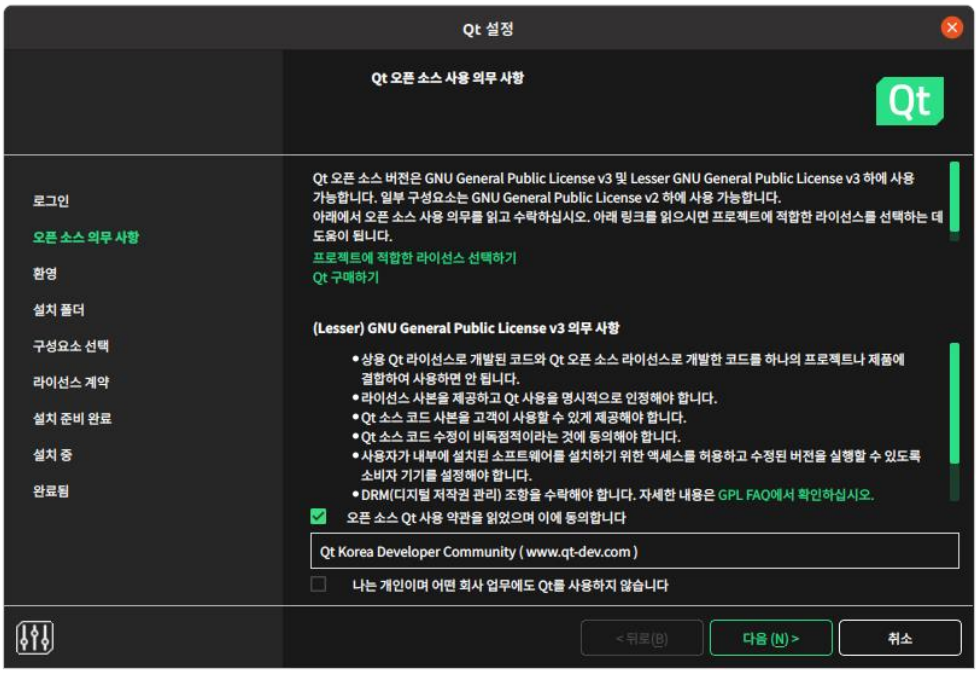
Next, download the file qt-online-installer-linux-x64-4.8.0.run with the extension as shown in the figure below.

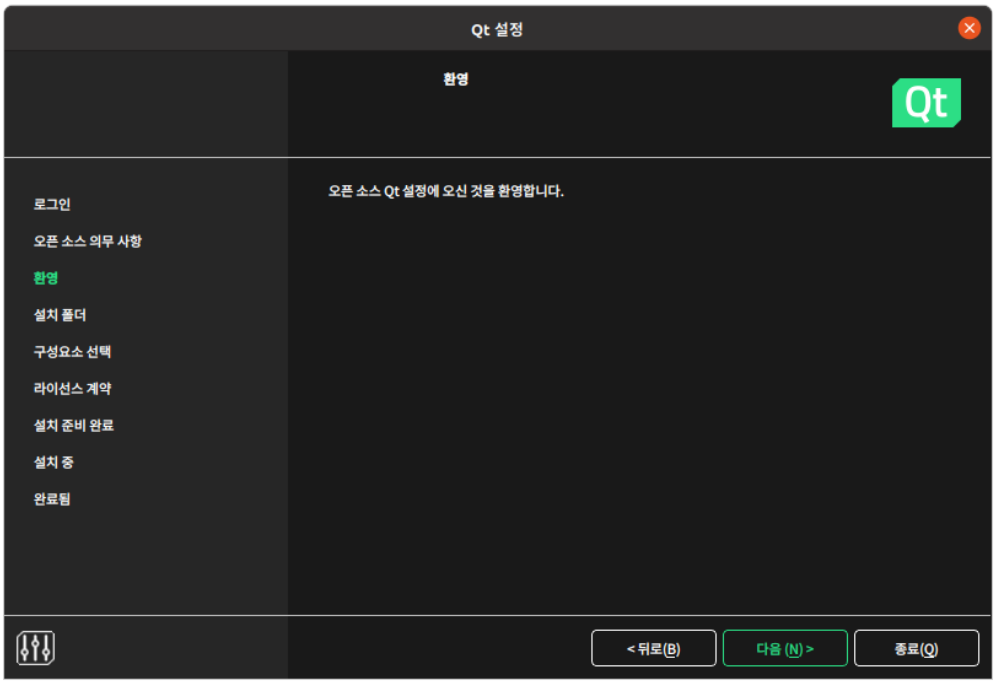


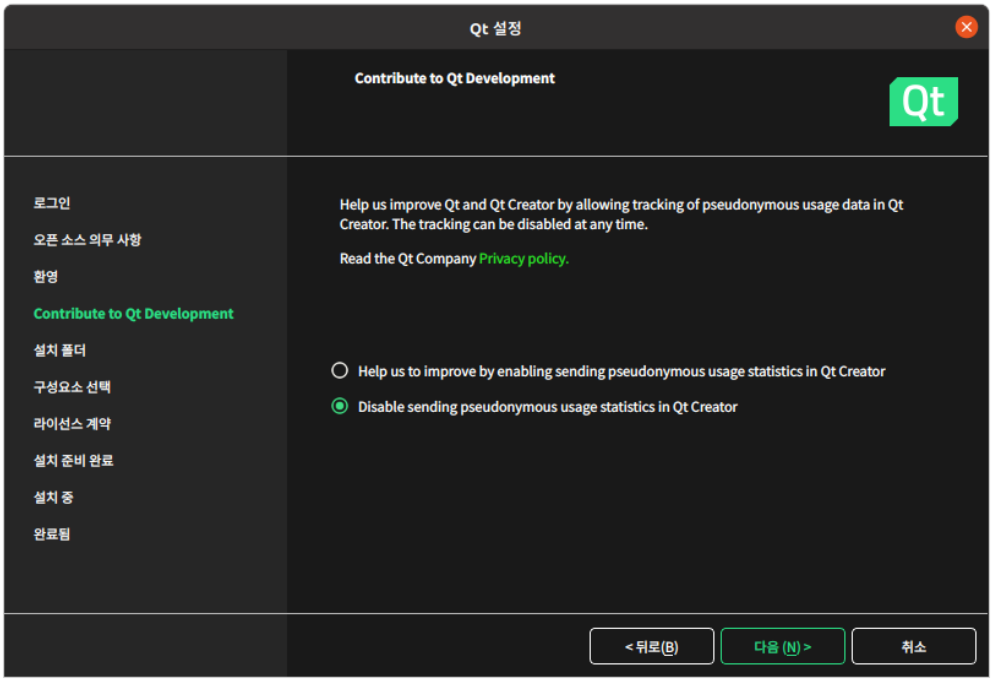
Once the qt-online-installer-linux-x64-4.8.0.run file has finished downloading, grant it execute permissions. Then run it.

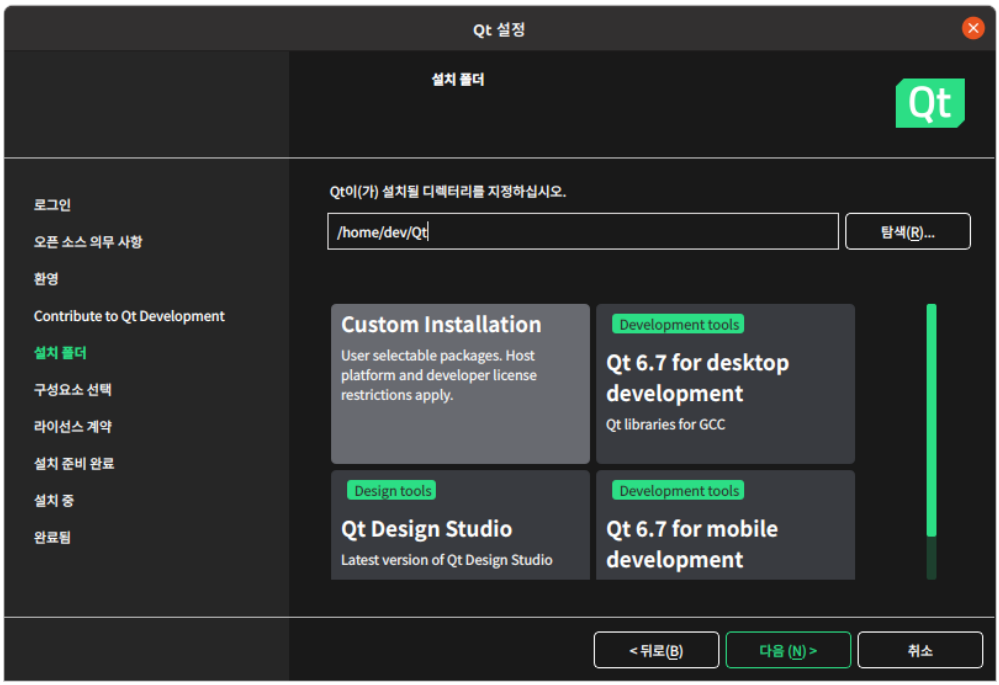
```
$ chmod 755 qt-online-installer-linux-x64-4.8.0.run
$ ./qt-online-installer-linux-x64-4.8.0.run
```


Qt 설정
Qt 온라인 설치 관리자에 오신 것을 환영합니다.
Qt
로그인
환영
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
Qt 계정에 로그인
비밀번호를 잊으셨습니까?
Qt 계정이 없습니까? 가입하기
The Qt Account will give you access to Qt downloads, exclusive services, bug reports, code review, and forums & wiki.
Looking for Squish, the automated GUI testing framework? Click here to download or try it.
다음
취소

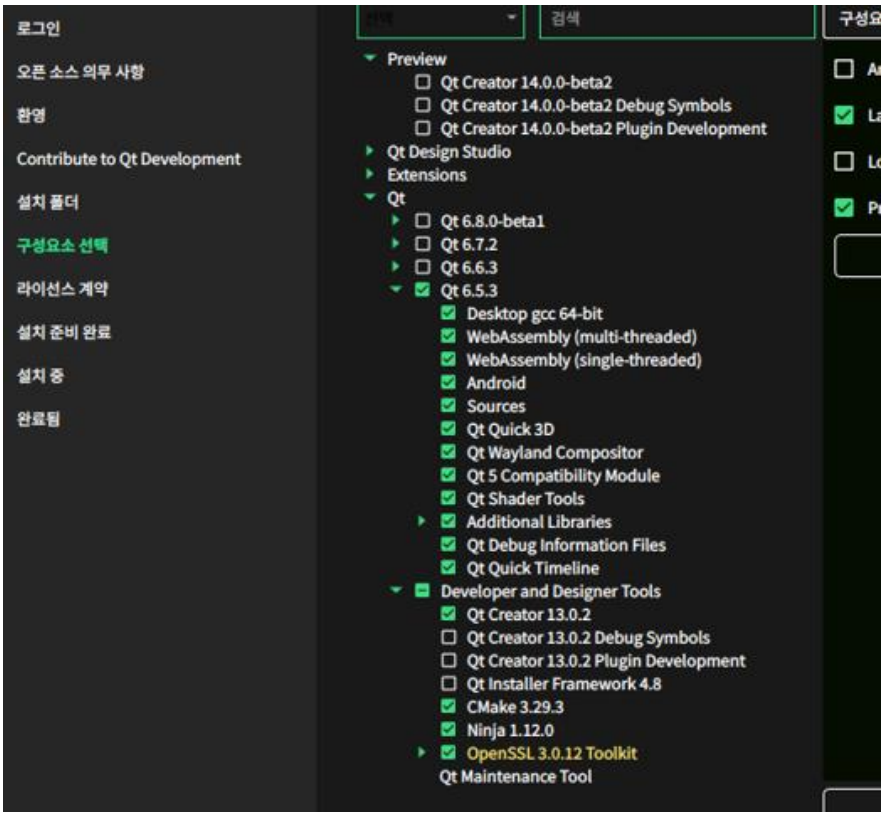

Qt 설정
Qt 오픈 소스 사용 의무 사항
Qt
로그인
오픈 소스 의무 사항
환영
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
Qt 오픈 소스 버전은 GNU General Public License v3 및 Lesser GNU General Public License v3 하에 사용 가능합니다. 일부 구성요소는 GNU General Public License v2 하에 사용 가능합니다.
아래에서 오픈 소스 사용 의무를 읽고 수락하십시오. 아래 링크를 읽으시면 프로젝트에 적합한 라이선스를 선택하는 데 도움이 됩니다.
프로젝트에 적합한 라이선스 선택하기
Qt 구매하기
(Lesser) GNU General Public License v3 의무 사항
• 상용 Qt 라이선스로 개발된 코드와 Qt 오픈 소스 라이선스로 개발한 코드를 하나의 프로젝트나 제품에 결합하여 사용하면 안 됩니다.
• 라이선스 사본을 제공하고 Qt 사용을 명시적으로 인정해야 합니다.
• Qt 소스 코드 사본을 고객이 사용할 수 있게 제공해야 합니다.
• Qt 소스 코드 수정이 비독점적이라는 것에 동의해야 합니다.
• 사용자가 내부에 설치된 소프트웨어를 설치하기 위한 액세스를 허용하고 수정된 버전을 실행할 수 있도록 소비자 기기를 설정해야 합니다.
• DRM(디지털 저작권 관리) 조항을 수락해야 합니다. 자세한 내용은 GPL FAQ에서 확인하십시오.
오픈 소스 Qt 사용 약관을 읽었으며 이에 동의합니다
Qt Korea Developer Community (www.qt-dev.com)
나는 개인이며 어떤 회사 업무에도 Qt를 사용하지 않습니다
< 뒤로(B)
다음 (N) >
취소

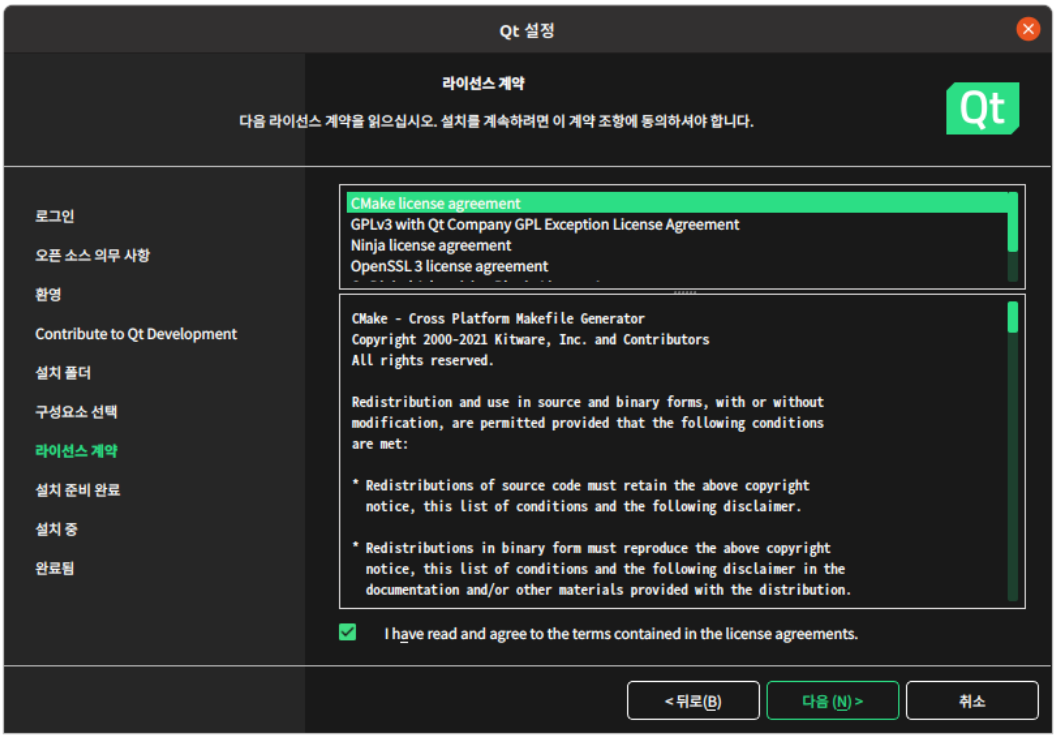

Qt 설정
환영
Qt
로그인
오픈 소스 의무 사항
환영
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
오픈 소스 Qt 설정에 오신 것을 환영합니다.
< 뒤로(B)
다음 (N) >
종료(Q)

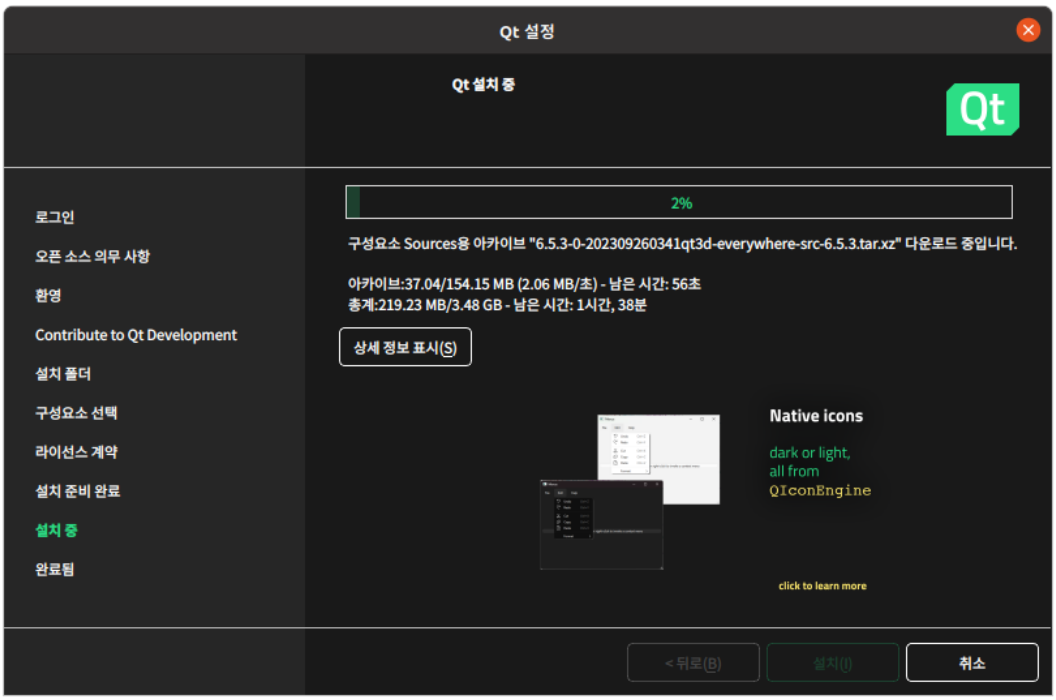

Qt 설정
Contribute to Qt Development
Qt
로그인
오픈 소스 의무 사항
환영
Contribute to Qt Development
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
Help us improve Qt and Qt Creator by allowing tracking of pseudonymous usage data in Qt Creator. The tracking can be disabled at any time.
Read the Qt Company Privacy policy.
Help us to improve by enabling sending pseudonymous usage statistics in Qt Creator
Disable sending pseudonymous usage statistics in Qt Creator
< 뒤로(B)
다음 (N) >
취소



Select the [Custom Installation] item, as shown in the figure above. In the next dialog, select the Qt 6.5 version and select the items as shown below.




Qt 설정
라이선스 계약
다음 라이선스 계약을 읽으십시오. 설치를 계속하려면 이 계약 조항에 동의하셔야 합니다.
로그인
오픈 소스 의무 사항
환영
Contribute to Qt Development
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
CMake license agreement
GPLv3 with Qt Company GPL Exception License Agreement
Ninja license agreement
OpenSSL 3 license agreement
CMake - Cross Platform Makefile Generator
Copyright 2000-2021 Kitware, Inc. and Contributors
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
* Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
I have read and agree to the terms contained in the license agreements.
< 뒤로(B)
다음 (N) >
취소


Qt 설정
Qt 설치 중
로그인
오픈 소스 의무 사항
환영
Contribute to Qt Development
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
2%
구성요소 Sources용 아카이브 "6.5.3-0-202309260341qt3d-everywhere-src-6.5.3.tar.xz" 다운로드 중입니다.
아카이브:37.04/154.15 MB (2.06 MB/초) - 남은 시간: 56초
총계:219.23 MB/3.48 GB - 남은 시간: 1시간, 38분
상세 정보 표시(S)
Native icons
dark or light,
all from
QIconEngine
click to learn more
< 뒤로(B)
설치(I)
취소

After installing Qt, check if the version of Emscripten is compatible with Qt 6.5 like below.

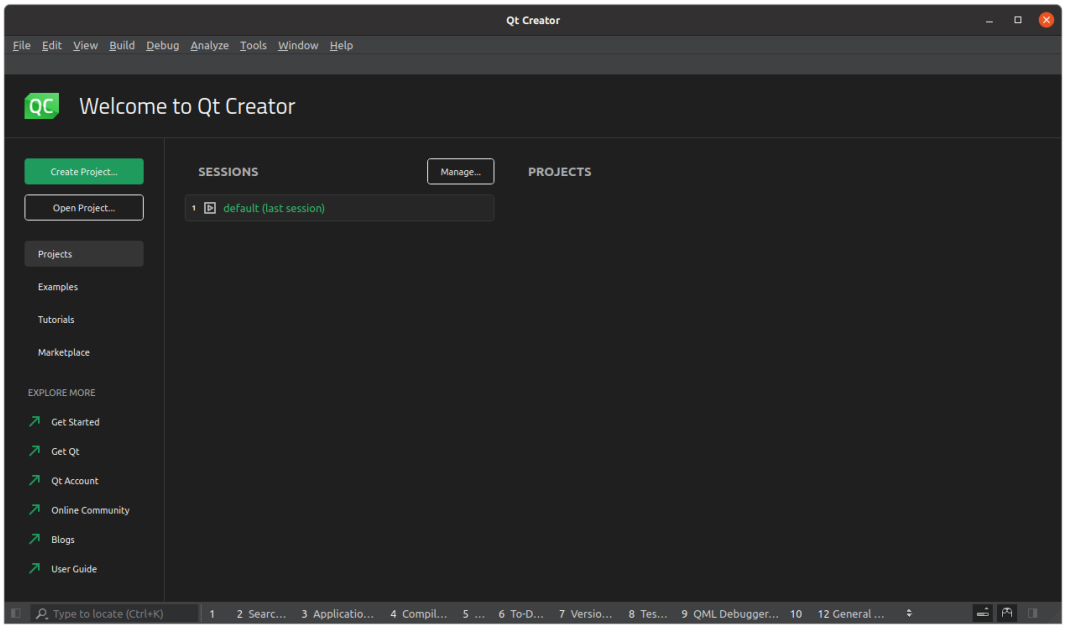
```
$ source /home/dev/emsdk/emsdk_env.sh
$ em++ --version
emcc (Emscripten gcc/clang-like replacement + linker emulating GNU ld) 3.1.25 (4343cbec72b7db283ea3bda1adc6cb1811ae9a73)
Copyright (C) 2014 the Emscripten authors (see AUTHORS.txt)
This is free and open source software under the MIT license.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

It is recommended that you use the LTS version of Qt whenever possible. If you are using a different version of Qt, you can refer to the URL below.

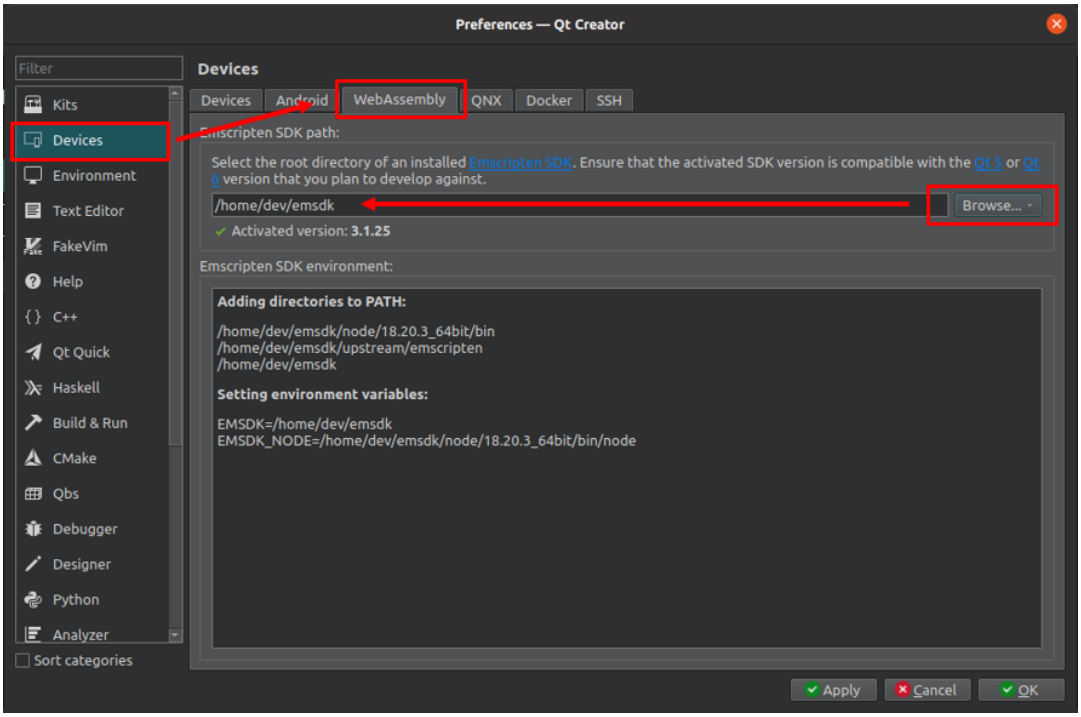
- https://doc.qt.io/qt-6/wasm.html

- Qt WebAssembly Configuration

Run Qt Creator.

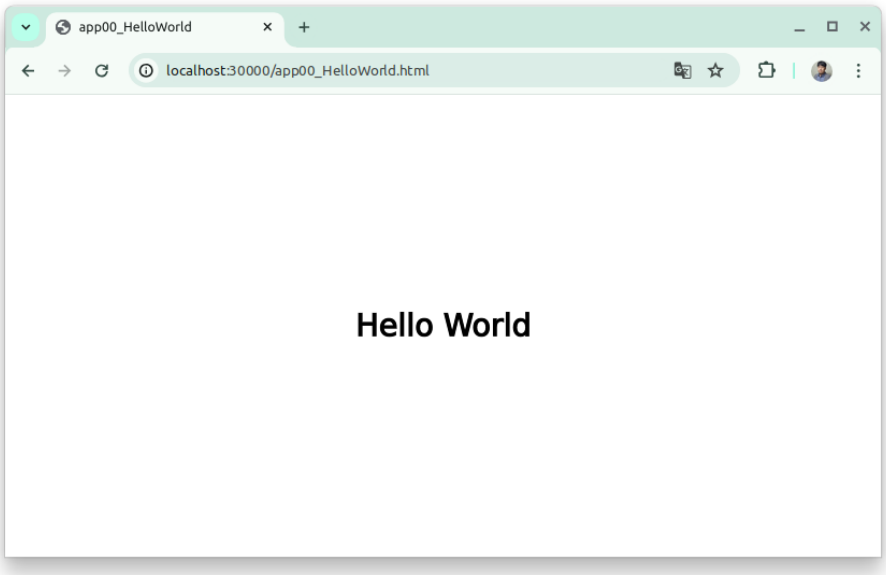


After running Qt Creator, click the [Edit] menu and then click the [Preferences...] menu.

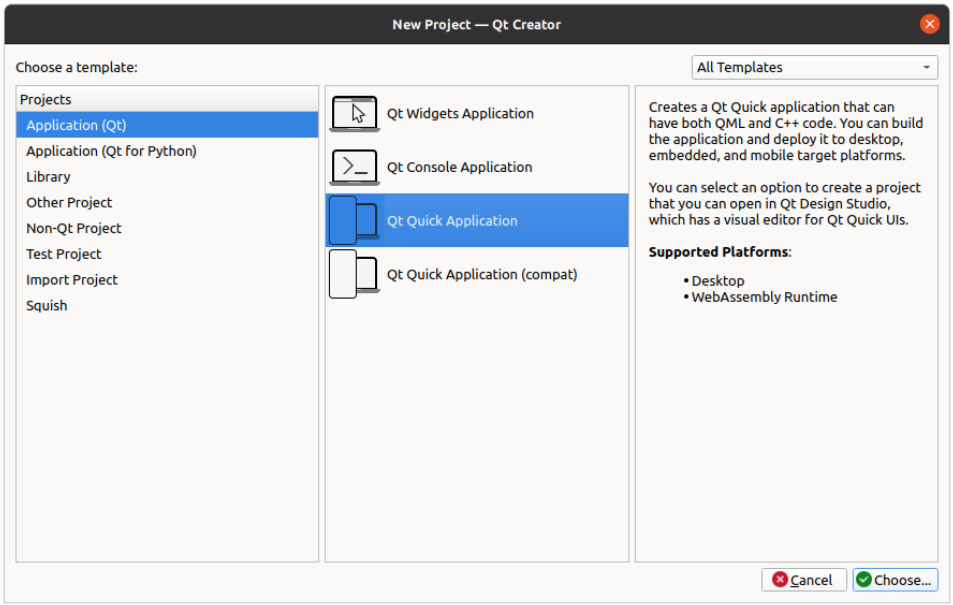


In the [Preferences] dialog, select [Devices] from the left tab and [WebAssembly] from the right tab. Next, click the [Browse] button and select the directory where the Emscripten SDK is installed.

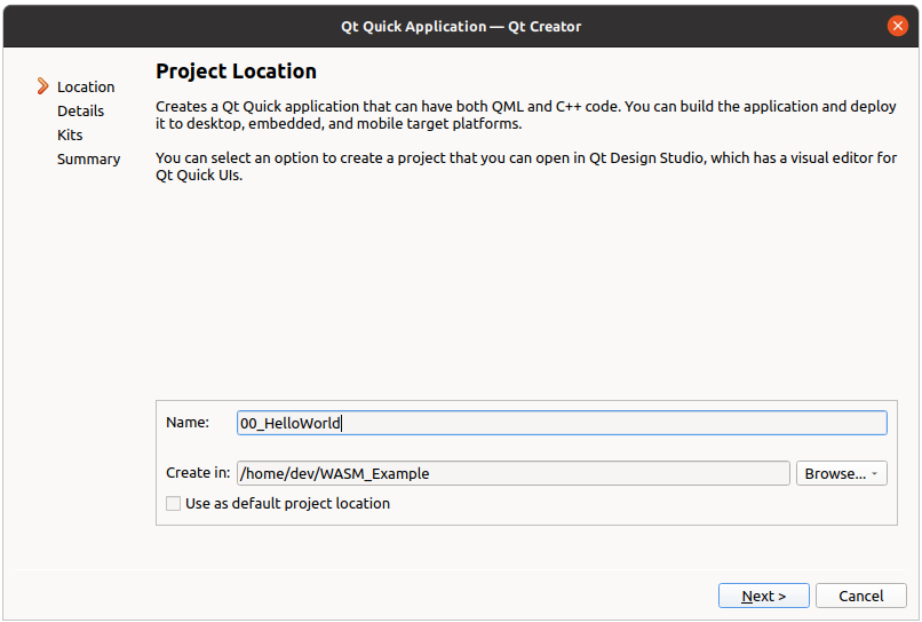
- Example of outputting Hello World to a web browser



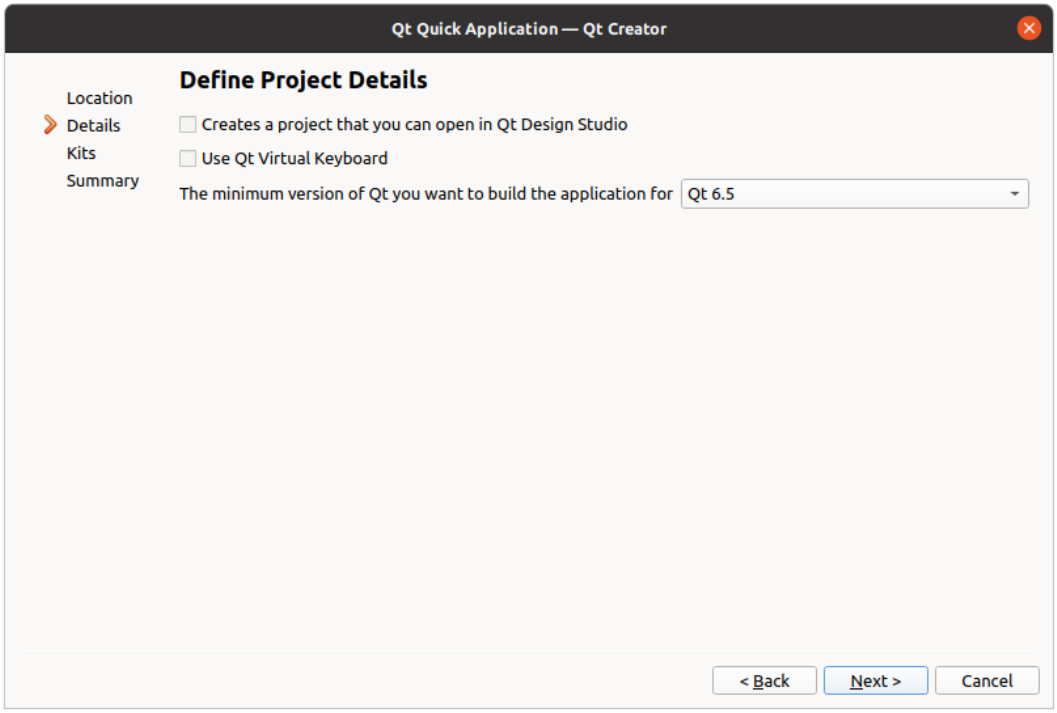
Let's write an example that prints "Hello World" to the web browser as shown in the image above. Create a project like the one below.



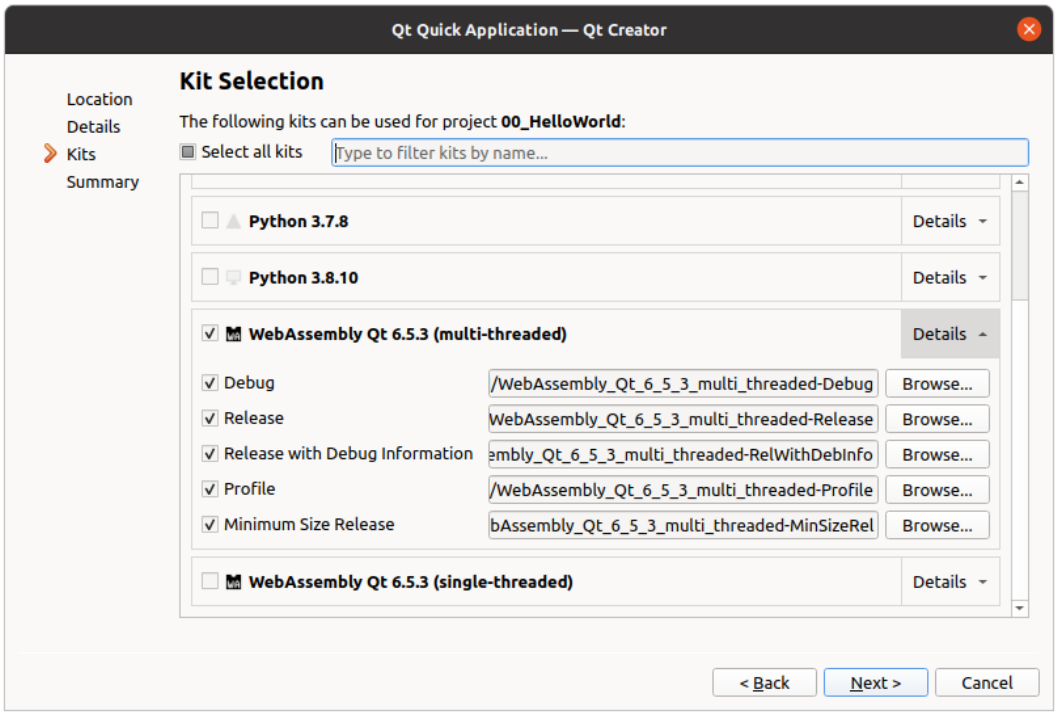
Select the [Application (Qt)] item, as shown in the image above, and then select the [Qt Quick Application] item from the middle of the list. Then click the [Choose] button.

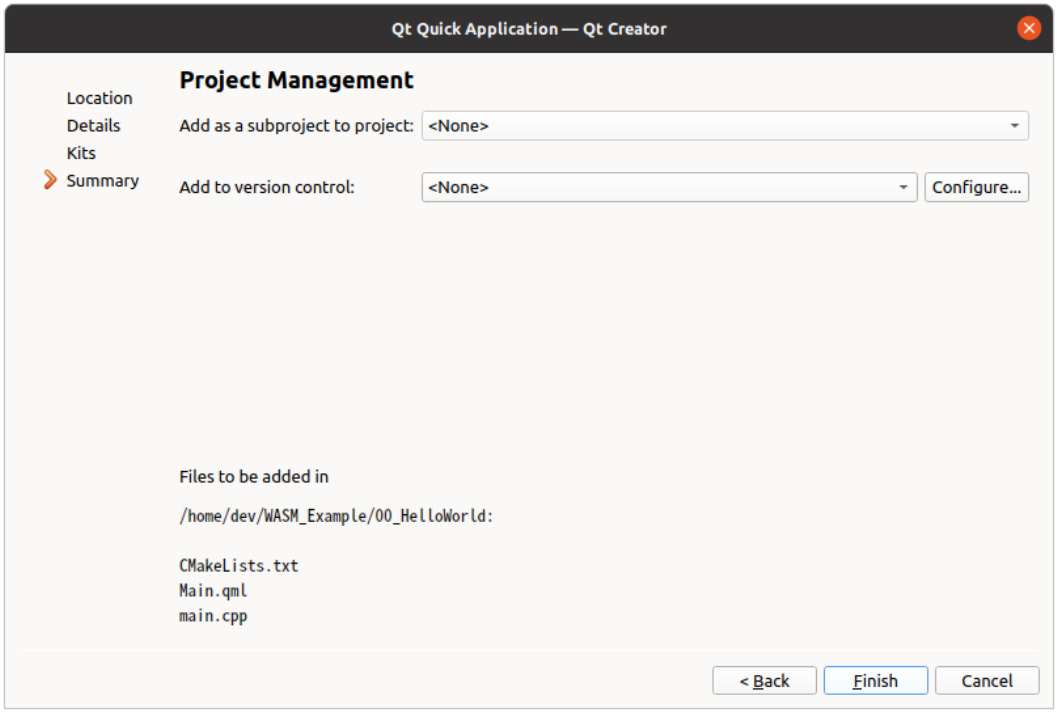


Enter a name for your project, as shown in the image above, and click the [Next] button.

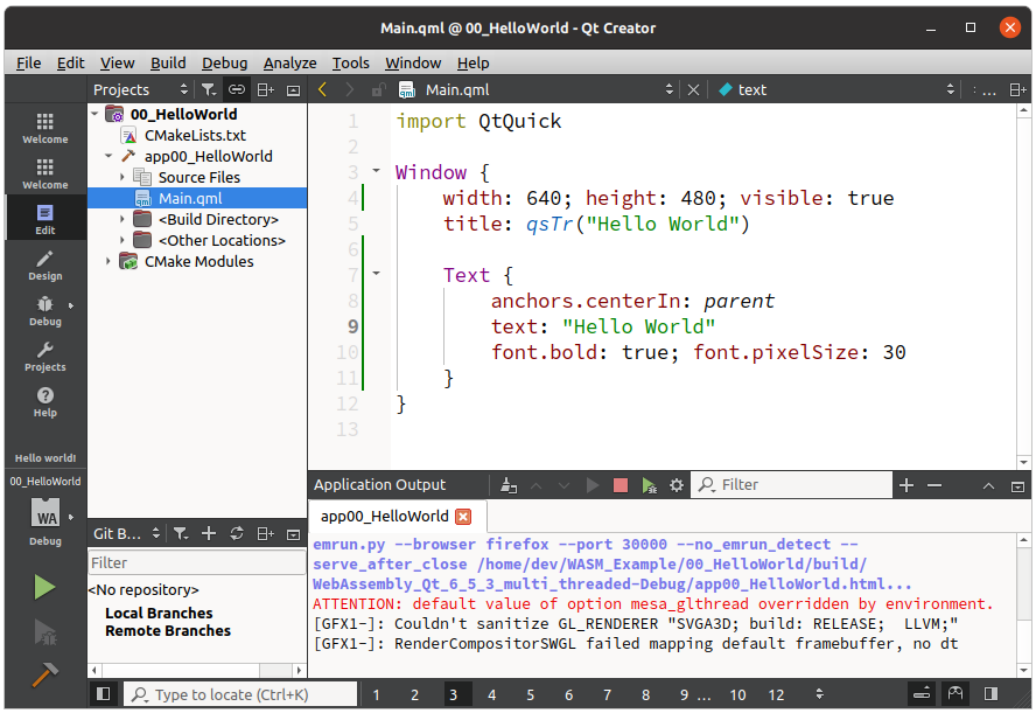


Select Qt 6.5 as the Minimum Version of Qt, and then select the WebAssembly Qt 6.x.x (multi-threaded) item in the Kits section, as shown in the image below.



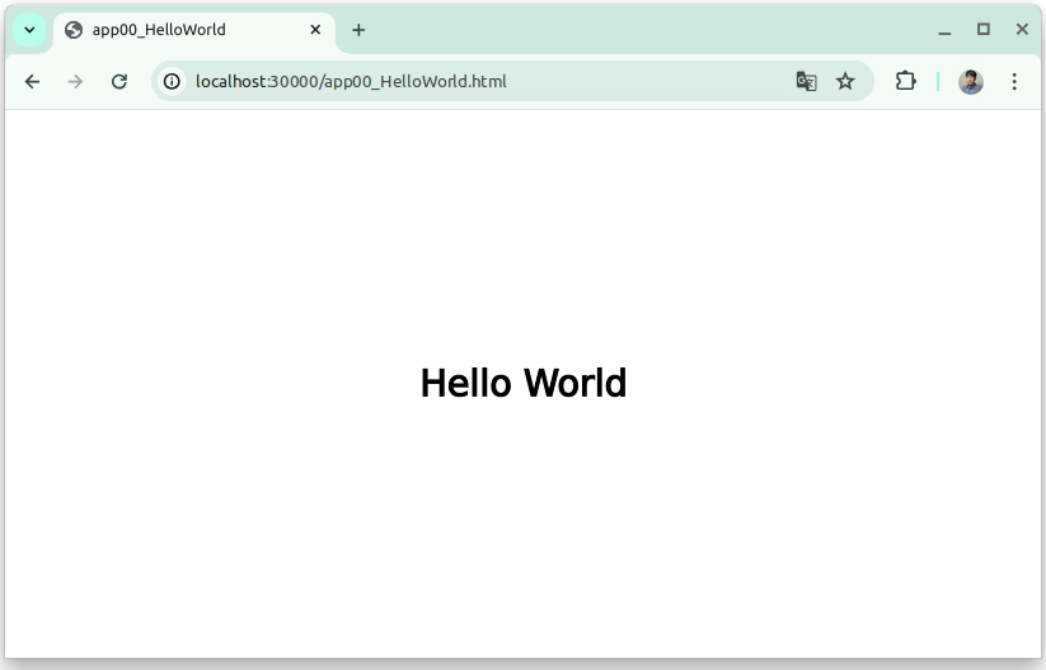


Click the [Finish] button to complete the project creation. Next, open the Main.qml file and write the source code like below.



```
import QtQuick

Window {
    width: 640; height: 480; visible: true
    title: qsTr("Hello World")

    Text {
        anchors.centerIn: parent
        text: "Hello World"
        font.bold: true; font.pixelSize: 30
    }
}
```

If you write the source code as above, build it and run it, you should see "Hello World" displayed on the web browser like below.



4. Building Dev Environments on macOS

In order to build a Qt WebAssembly development environment on macOS, you must first install the following packages

- Homebrew install

```
% /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

- Check the version after installing Homebrew

```
% brew --version
```

- Install Python3 with Homebrew

```
% brew install python3
```

- Check the version after installing Python3

```
% python3 --version
Python 3.x.x
```

- cmake install

```
% brew install cmake
```

- Emscripten download and install

```
% git clone https://github.com/juj/emsdk.git
% cd emsdk
emsdk % ./emsdk update
```

```
emsdk % git pull
emsdk % ./emsdk install 3.1.14
emsdk % ./emsdk activate 3.1.14
emsdk % source ./emsdk_env.sh
```

- Emscripten version check

```
emsdk % emcc --version
emcc (Emscripten gcc/clang-like replacement + linker emulating GNU ld) 3.1.14 (cfe2bdfe2692457cb5f5770672f6e5ccb3ffc2f2)
Copyright (C) 2014 the Emscripten authors (see AUTHORS.txt)
This is free and open source software under the MIT license.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

- Hello world example

Let's write a simple example using the Emscripten SDK to print Hello World. Create a directory like below and navigate to the directory you created.

```
% mkdir wasm_examples
% cd wasm_examples
wasm_examples % mkdir hello
wasm_examples % cd hello
dev@devui-Mac hello % pwd
/Users/dev/wasm_examples/hello
```

Next, create a hello.c file.

```
hello % touch hello.c
hello % ls -tlr
total 0
-rw-r--r--  1 dev  staff  0  1 22 19:42 hello.c
dev@devui-Mac hello %
```

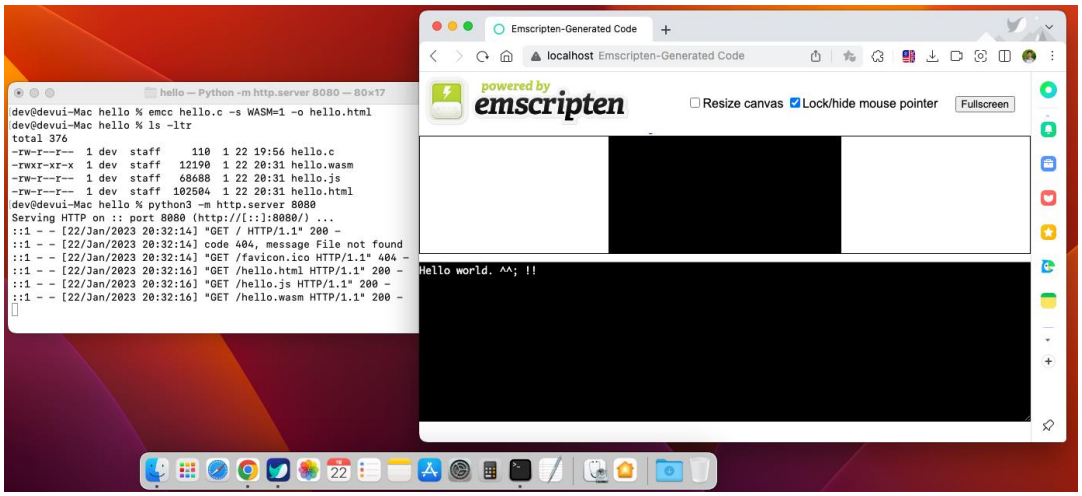
In the hello.c file you created, write the source code like below.

```
#include <stdio.h>
int main(int argc, char **argv)
{
	printf("Hello world. ^^; !!₩n");
	return 1;
}
```

Let's build the hello.c source file you created using the Emscripten SDK as shown below.

```
hello % emcc hello.c -s WASM=1 -o hello.html
hello % ls -tlr
total 376
-rw-r--r--   1 dev   staff       110   1 22 19:56 hello.c
-rwxr-xr-x   1 dev   staff     12190   1 22 20:00 hello.wasm
-rw-r--r--   1 dev   staff     68688   1 22 20:00 hello.js
-rw-r--r--   1 dev   staff    102504   1 22 20:00 hello.html

dev@devui-Mac hello % python3 -m http.server 8080
Serving HTTP on :: port 8080 (http://[::]:8080/) ...
```

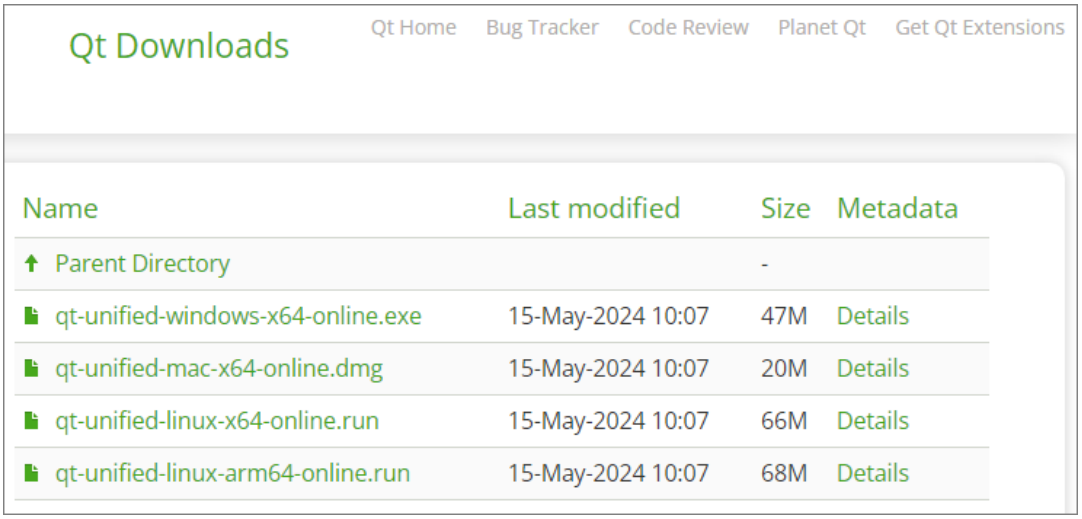
Python에는 WASM을 실행할 수 있는 웹서버를 제공한다. 위와 같이 8080 포트 번호로 웹서버를 로딩 하면 Web Browser 로 WebAssembly 파일을 실행할 수 있다.



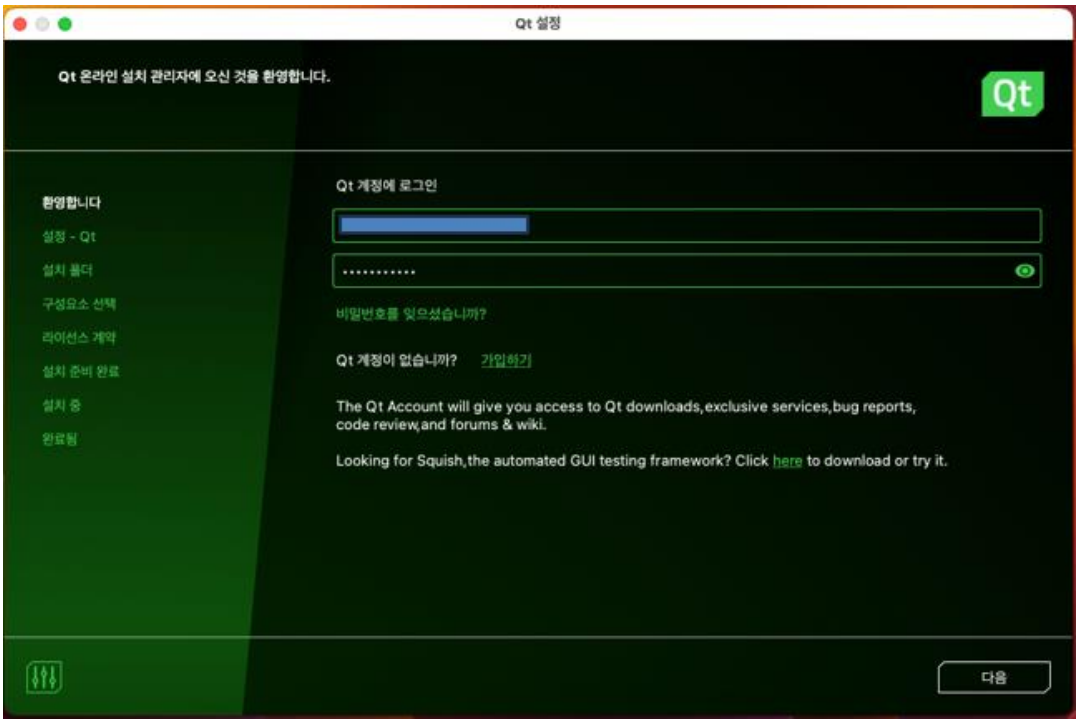
- Qt install

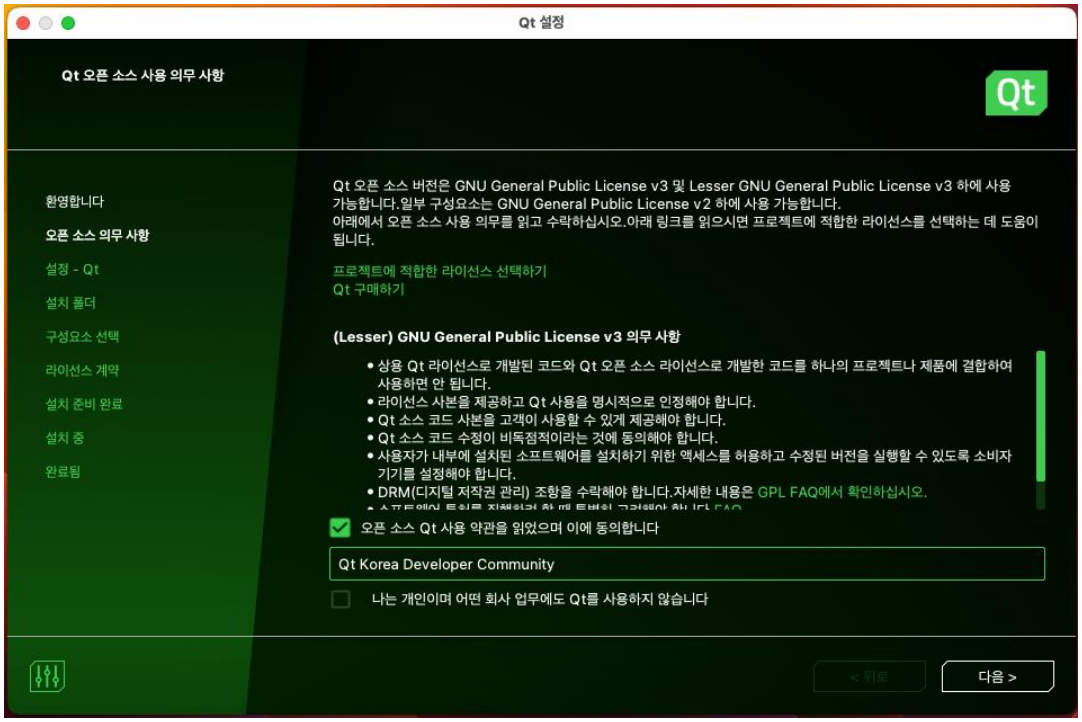
Download the Qt online install file from the download.qt.io site. The file for MacOS has a dmg extension.

- Download URL: https://download.qt.io/official_releases/online_installers/
- Download file: qt-unified-mac-x64-online.dmg

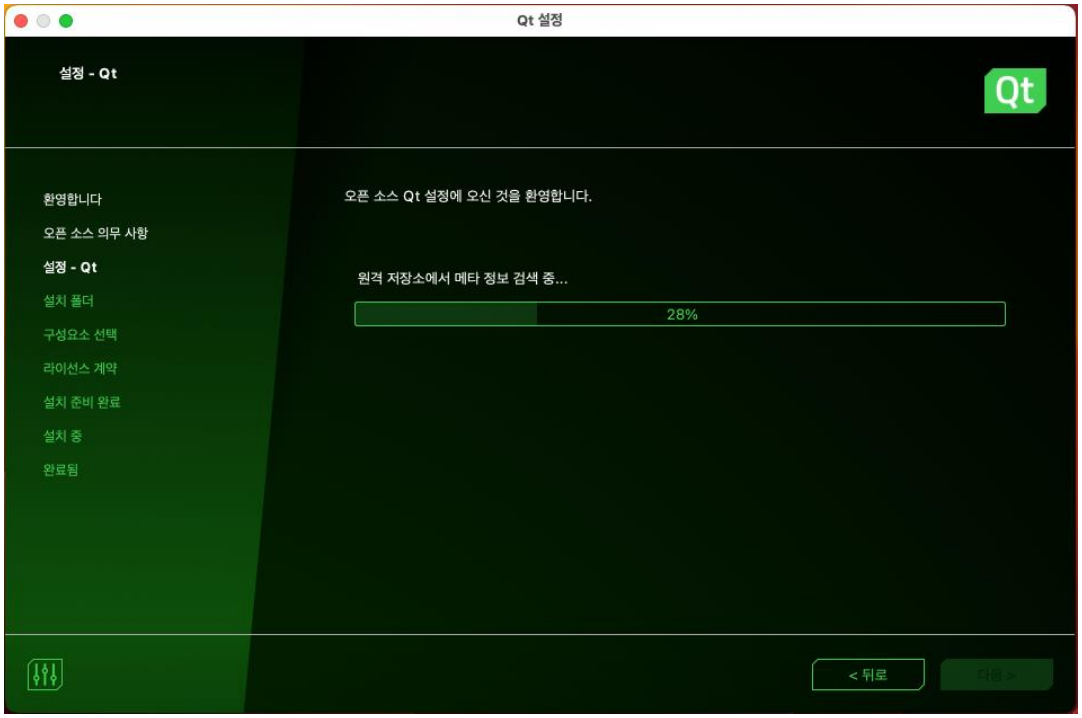


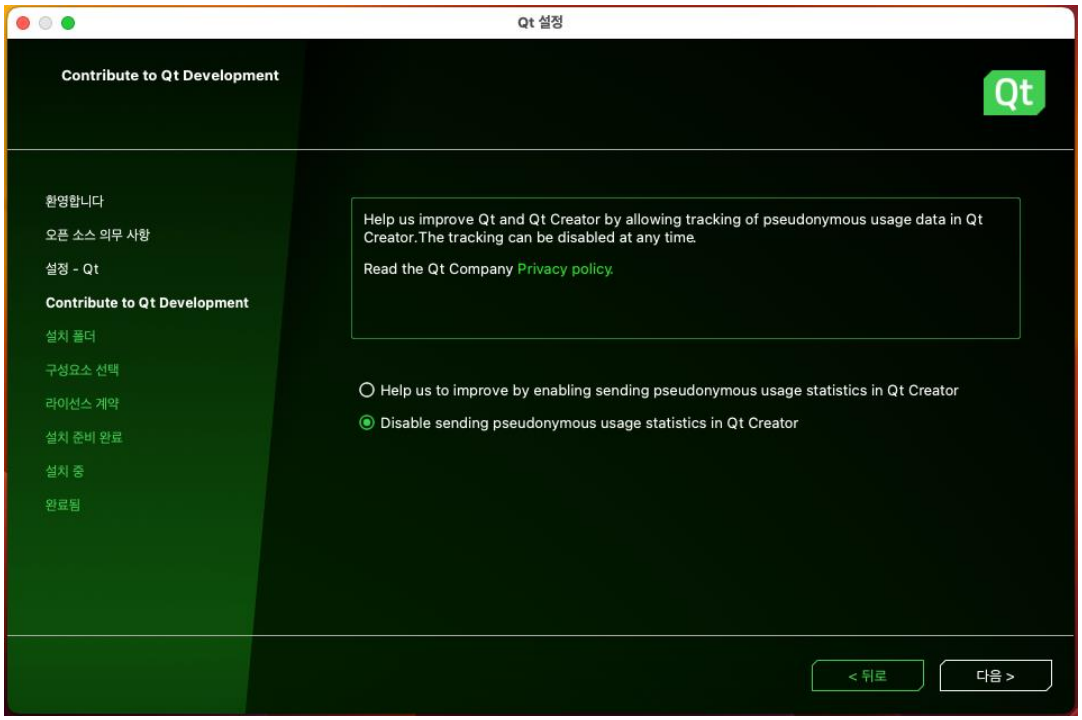
Download and run the Qt online install file as shown above.

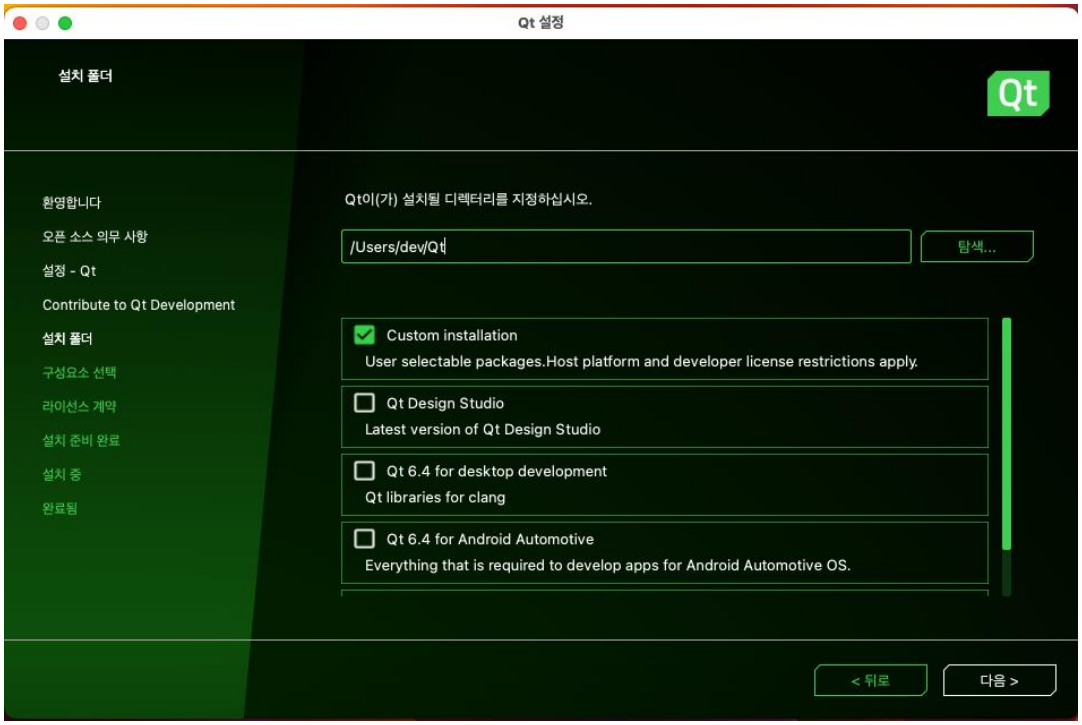


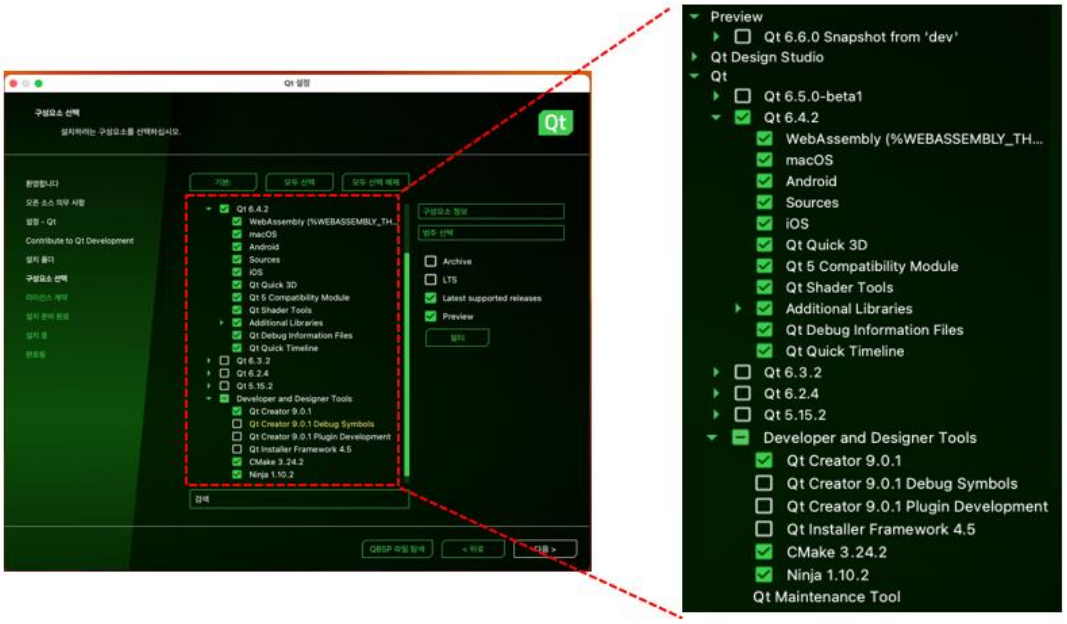

Qt 설정
Qt 오픈 소스 사용 의무 사항
환영합니다
오픈 소스 의무 사항
설정 - Qt
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
Qt 오픈 소스 버전은 GNU General Public License v3 및 Lesser GNU General Public License v3 하에 사용 가능합니다.일부 구성요소는 GNU General Public License v2 하에 사용 가능합니다.
아래에서 오픈 소스 사용 의무를 읽고 수락하십시오.아래 링크를 읽으시면 프로젝트에 적합한 라이선스를 선택하는 데 도움이 됩니다.
프로젝트에 적합한 라이선스 선택하기
Qt 구매하기
(Lesser) GNU General Public License v3 의무 사항
• 상용 Qt 라이선스로 개발된 코드와 Qt 오픈 소스 라이선스로 개발한 코드를 하나의 프로젝트나 제품에 결합하여 사용하면 안 됩니다.
• 라이선스 사본을 제공하고 Qt 사용을 명시적으로 인정해야 합니다.
• Qt 소스 코드 사본을 고객이 사용할 수 있게 제공해야 합니다.
• Qt 소스 코드 수정이 비독점적이라는 것에 동의해야 합니다.
• 사용자가 내부에 설치된 소프트웨어를 설치하기 위한 액세스를 허용하고 수정된 버전을 실행할 수 있도록 소비자 기기를 설정해야 합니다.
• DRM(디지털 저작권 관리) 조항을 수락해야 합니다.자세한 내용은 GPL FAQ에서 확인하십시오.
오픈 소스 Qt 사용 약관을 읽었으며 이에 동의합니다
Qt Korea Developer Community
나는 개인이며 어떤 회사 업무에도 Qt를 사용하지 않습니다
< 뒤로
다음 >

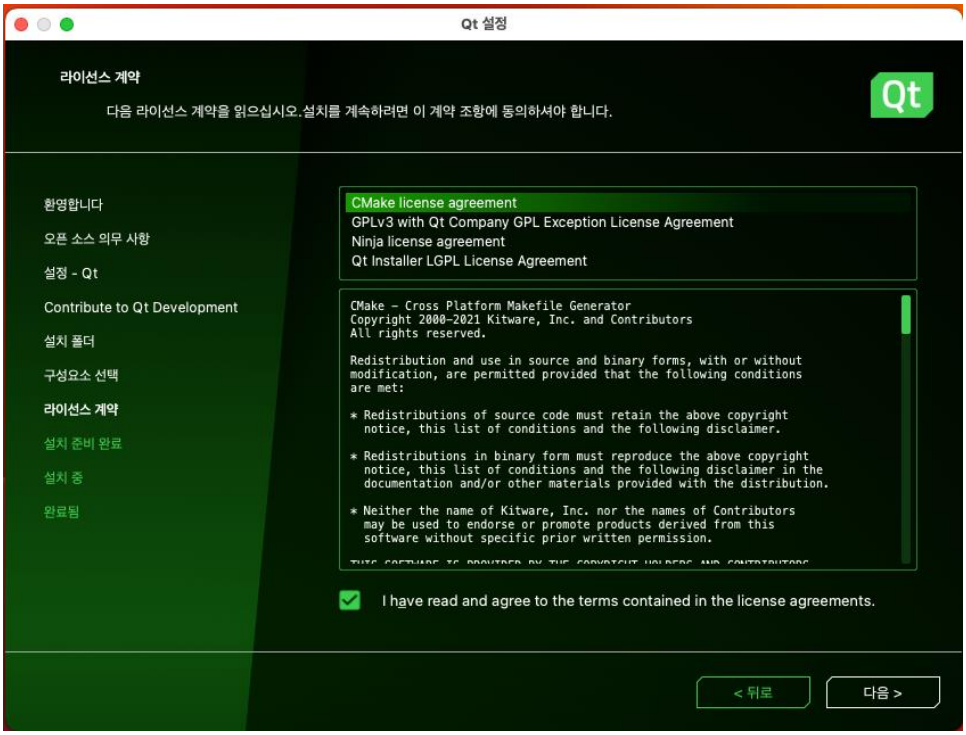

Qt 설정
설정 - Qt
환영합니다
오픈 소스 의무 사항
설정 - Qt
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
오픈 소스 Qt 설정에 오신 것을 환영합니다.
< 뒤로
다음 >

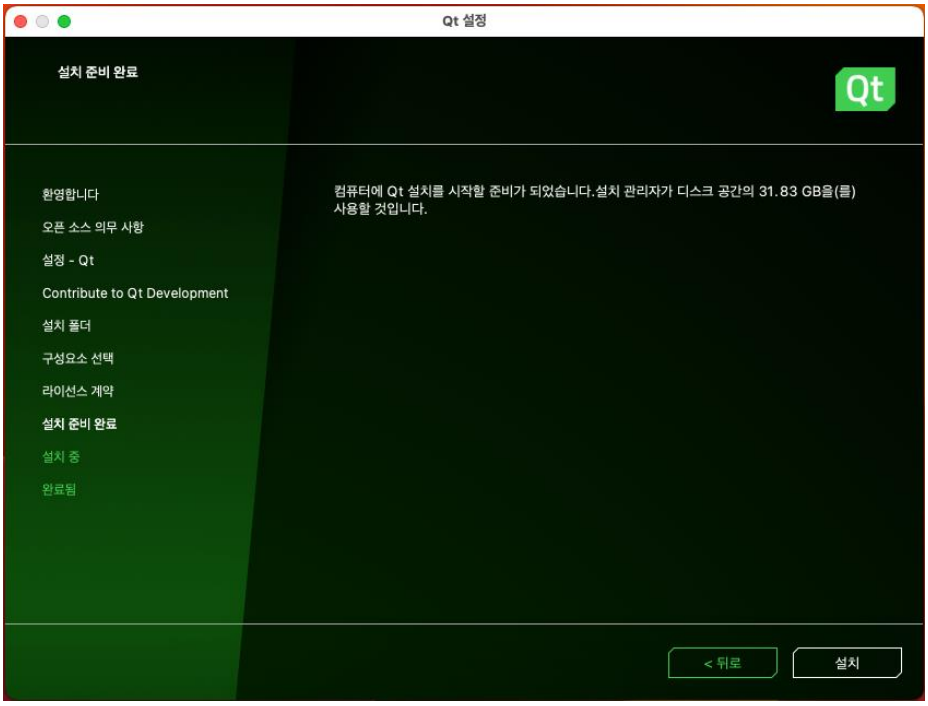

Qt 설정
설정 - Qt
환영합니다
오픈 소스 의무 사항
설정 - Qt
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
오픈 소스 Qt 설정에 오신 것을 환영합니다.
원격 저장소에서 메타 정보 검색 중...
28%
< 뒤로
다음 >

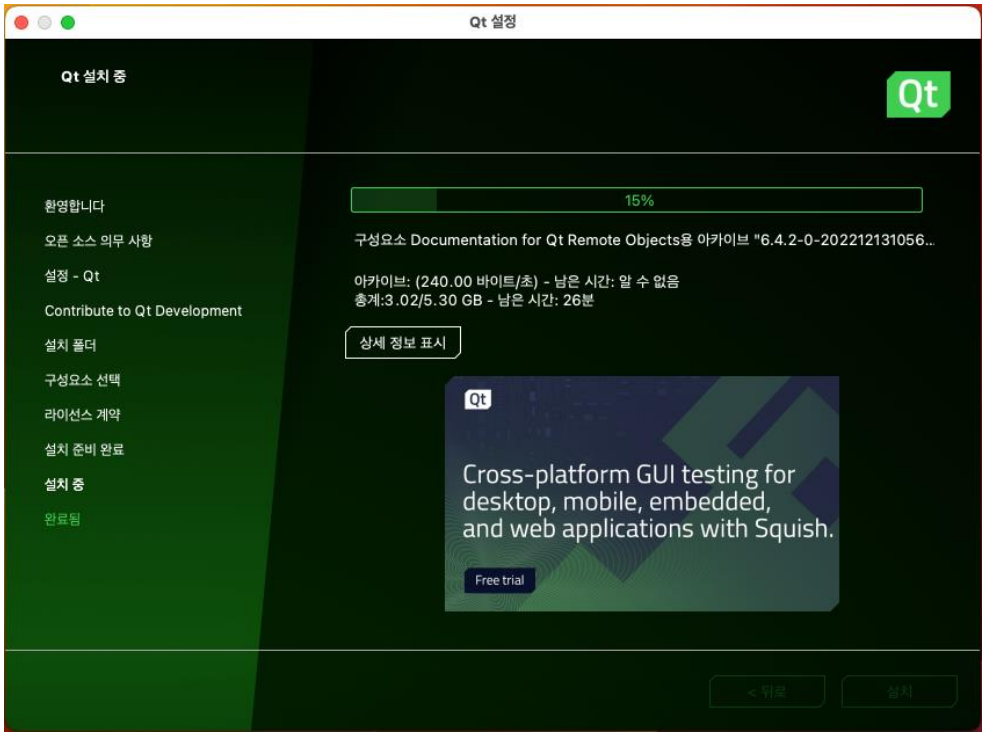

Qt 설정
Contribute to Qt Development
환영합니다
오픈 소스 의무 사항
설정 - Qt
Contribute to Qt Development
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
Help us improve Qt and Qt Creator by allowing tracking of pseudonymous usage data in Qt Creator.The tracking can be disabled at any time.
Read the Qt Company Privacy policy.
Help us to improve by enabling sending pseudonymous usage statistics in Qt Creator
Disable sending pseudonymous usage statistics in Qt Creator
< 뒤로
다음 >

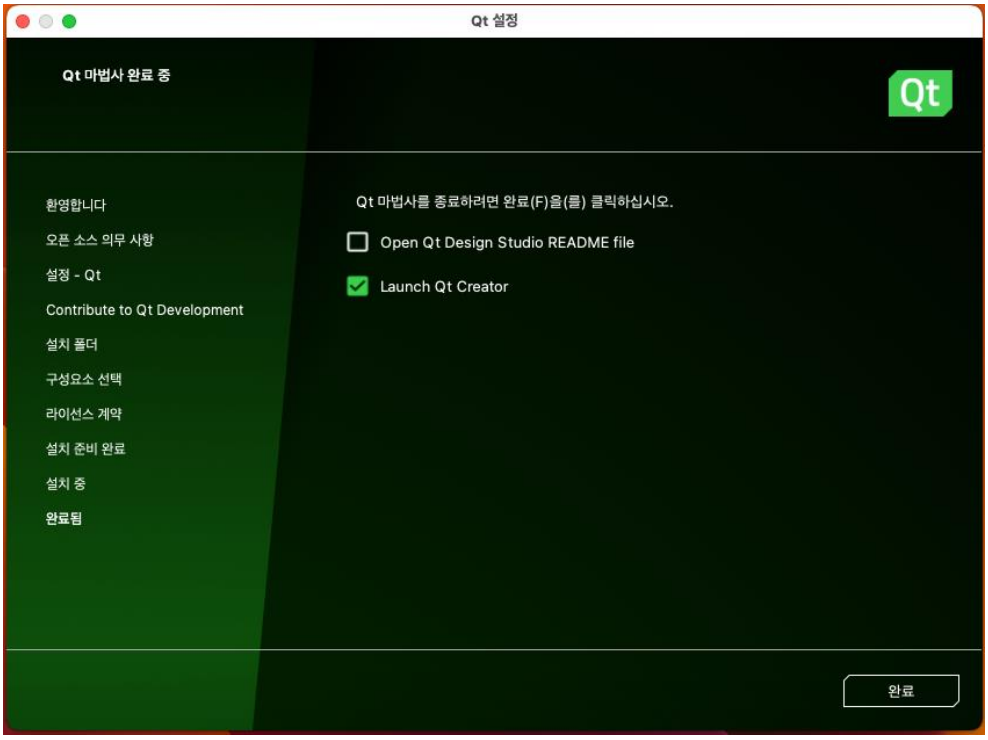

Qt 설정
설치 폴더
환영합니다
오픈 소스 의무 사항
설정 - Qt
Contribute to Qt Development
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
Qt이(가) 설치될 디렉터리를 지정하십시오.
/Users/dev/Qt
탐색...
Custom installation
User selectable packages.Host platform and developer license restrictions apply.
Qt Design Studio
Latest version of Qt Design Studio
Qt 6.4 for desktop development
Qt libraries for clang
Qt 6.4 for Android Automotive
Everything that is required to develop apps for Android Automotive OS.
< 뒤로
다음 >

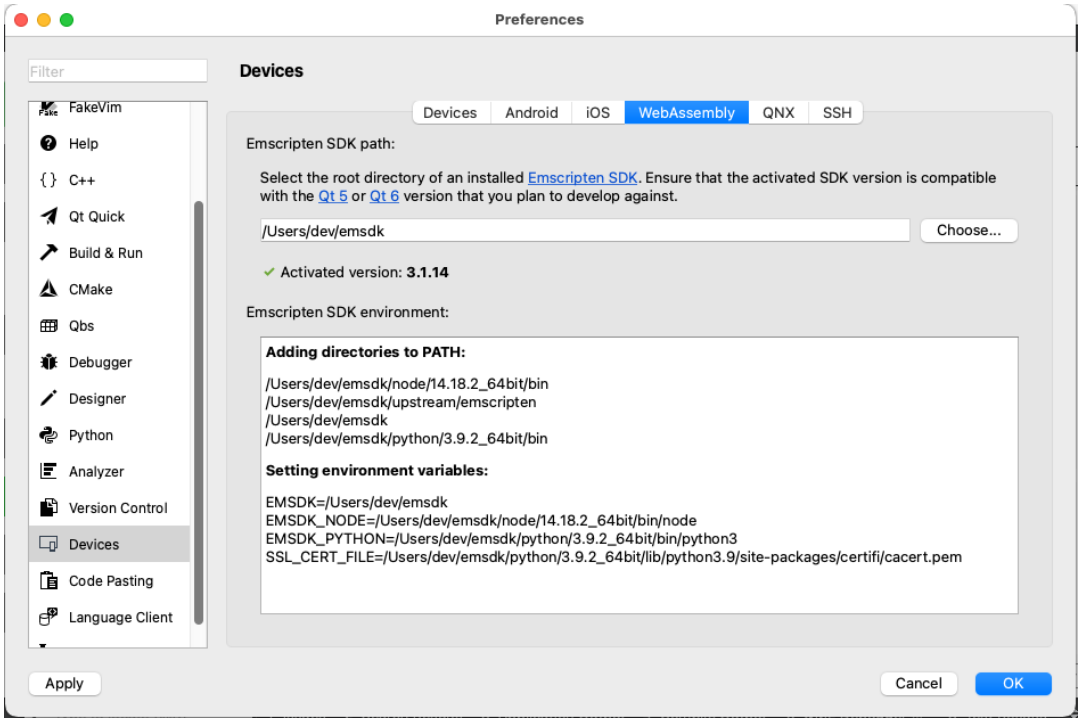

Preview
Qt 6.6.0 Snapshot from 'dev'
Qt Design Studio
Qt
Qt 6.5.0-beta1
Qt 6.4.2
WebAssembly (%WEBASSEMBLY_TH...
macOS
Android
Sources
iOS
Qt Quick 3D
Qt 5 Compatibility Module
Qt Shader Tools
Additional Libraries
Qt Debug Information Files
Qt Quick Timeline
Qt 6.3.2
Qt 6.2.4
Qt 5.15.2
Developer and Designer Tools
Qt Creator 9.0.1
Qt Creator 9.0.1 Debug Symbols
Qt Creator 9.0.1 Plugin Development
Qt Installer Framework 4.5
CMake 3.24.2
Ninja 1.10.2
Qt Maintenance Tool


Qt 설정
라이선스 계약
다음 라이선스 계약을 읽으십시오.설치를 계속하려면 이 계약 조항에 동의하셔야 합니다.
환영합니다
오픈 소스 의무 사항
설정 - Qt
Contribute to Qt Development
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
CMake license agreement
GPLv3 with Qt Company GPL Exception License Agreement
Ninja license agreement
Qt Installer LGPL License Agreement
CMake – Cross Platform Makefile Generator
Copyright 2000-2021 Kitware, Inc. and Contributors
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
* Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
* Neither the name of Kitware, Inc. nor the names of Contributors
may be used to endorse or promote products derived from this
software without specific prior written permission.
I have read and agree to the terms contained in the license agreements.
< 뒤로
다음 >


Qt 설정
설치 준비 완료
환영합니다
오픈 소스 의무 사항
설정 - Qt
Contribute to Qt Development
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
컴퓨터에 Qt 설치를 시작할 준비가 되었습니다.설치 관리자가 디스크 공간의 31.83 GB을(를) 사용할 것입니다.
< 뒤로
설치


Qt 설정
Qt 설치 중
환영합니다
오픈 소스 의무 사항
설정 - Qt
Contribute to Qt Development
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
15%
구성요소 Documentation for Qt Remote Objects용 아카이브 "6.4.2-0-202212131056...
아카이브: (240.00 바이트/초) - 남은 시간: 알 수 없음
총계:3.02/5.30 GB - 남은 시간: 26분
상세 정보 표시
Cross-platform GUI testing for desktop, mobile, embedded, and web applications with Squish.
Free trial
< 뒤로
설치


Qt 설정
Qt 마법사 완료 중
환영합니다
오픈 소스 의무 사항
설정 - Qt
Contribute to Qt Development
설치 폴더
구성요소 선택
라이선스 계약
설치 준비 완료
설치 중
완료됨
Qt 마법사를 종료하려면 완료(F)을(를) 클릭하십시오.
Open Qt Design Studio README file
Launch Qt Creator
완료



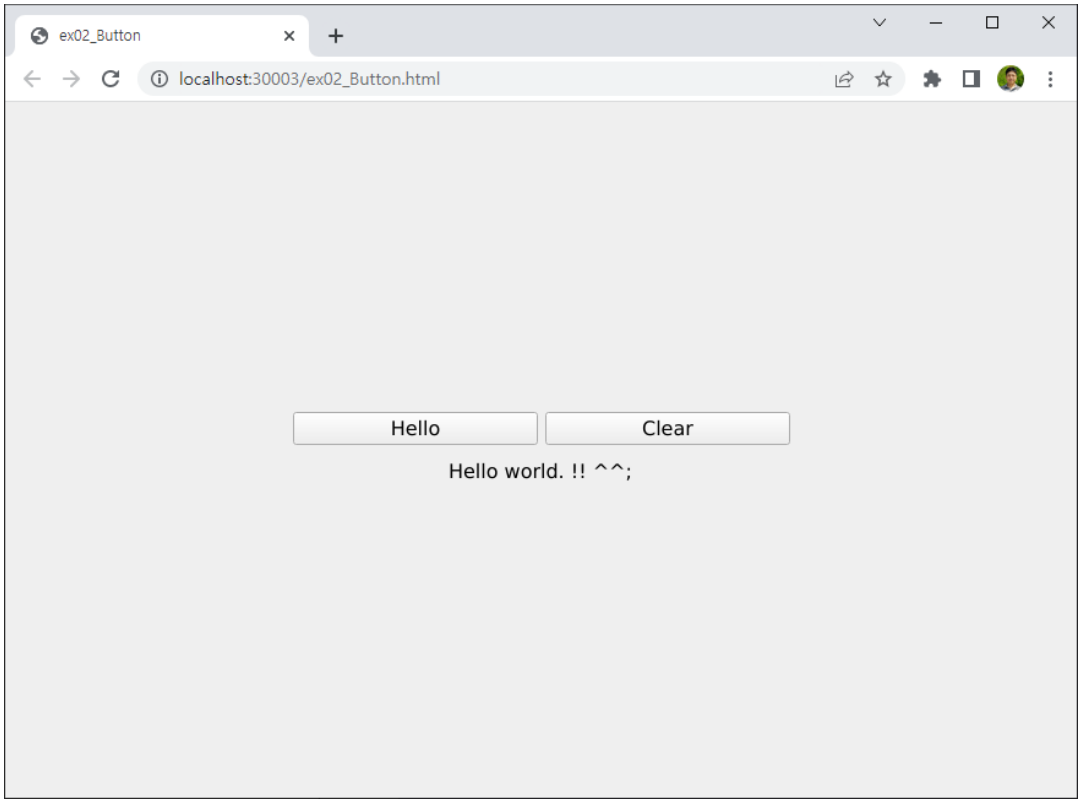
Click [Devices] on the left tab, as shown in the image above. Then, on the right tab, select the WebAssembly tab and select the version as shown above. In this example, we used Qt version 6.4. Therefore, the Emscripten version is 3.1.14.

If you want to use Qt 6.5 on macOS, your Emscripten version should be 3.1.25.

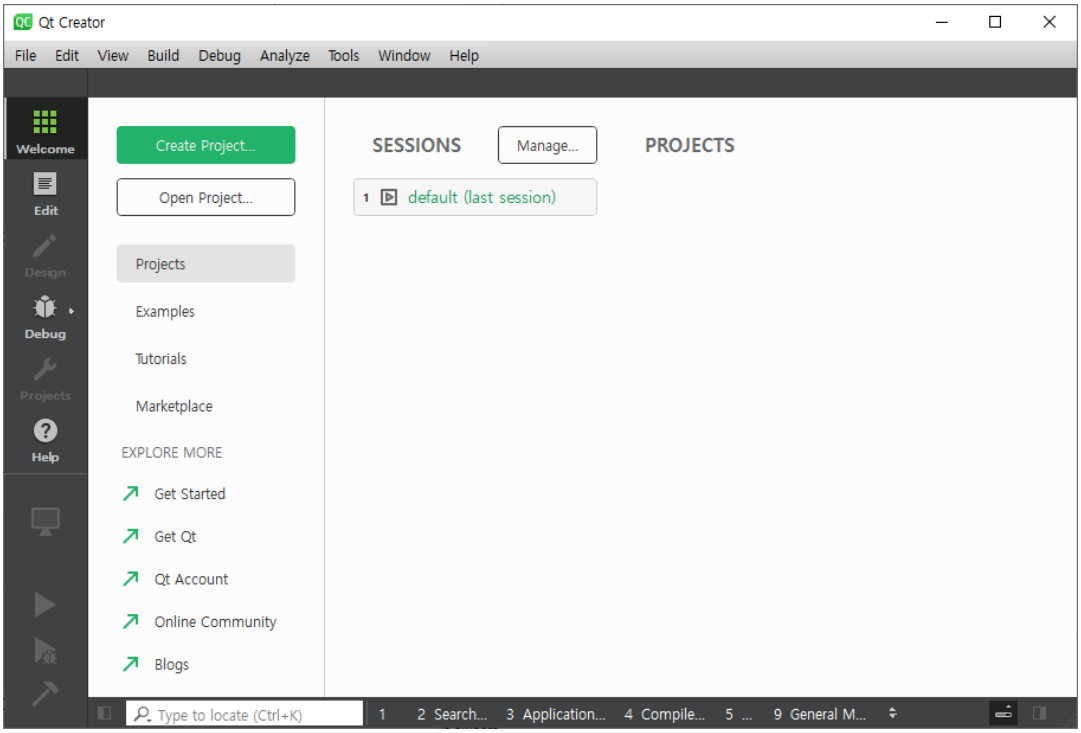
5. Getting Started with Qt for WebAssembly Programming

In this article, we will implement a simple Qt WebAssembly application using Qt for WebAssembly.

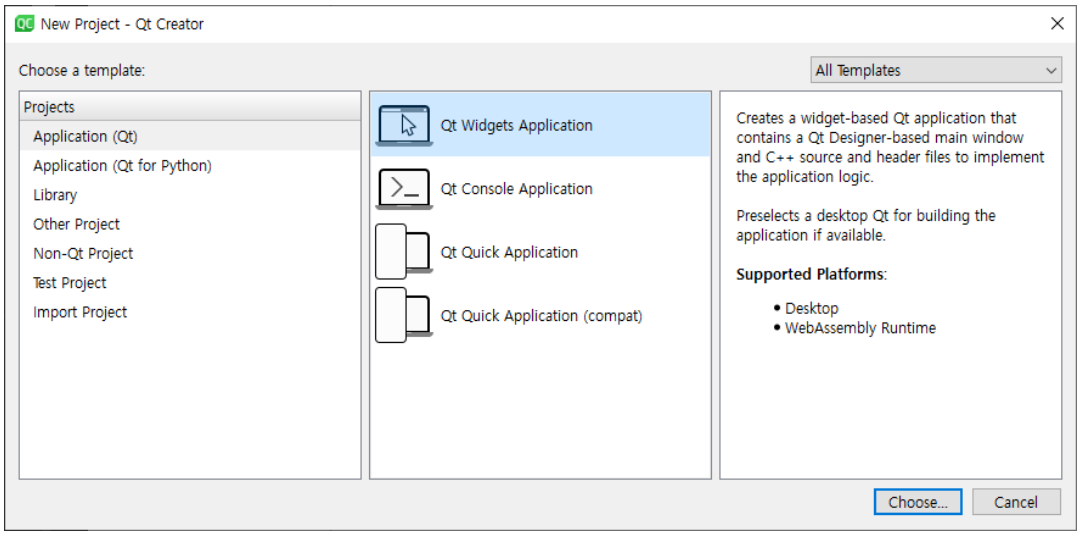
The application we will implement will be as shown in the figure below, where when you click on the [Hello] button, it will display the message "Hello world. !! ^^;" message is printed below the button. We will also implement a simple function to delete the message below the button when the [Clear] button is clicked.



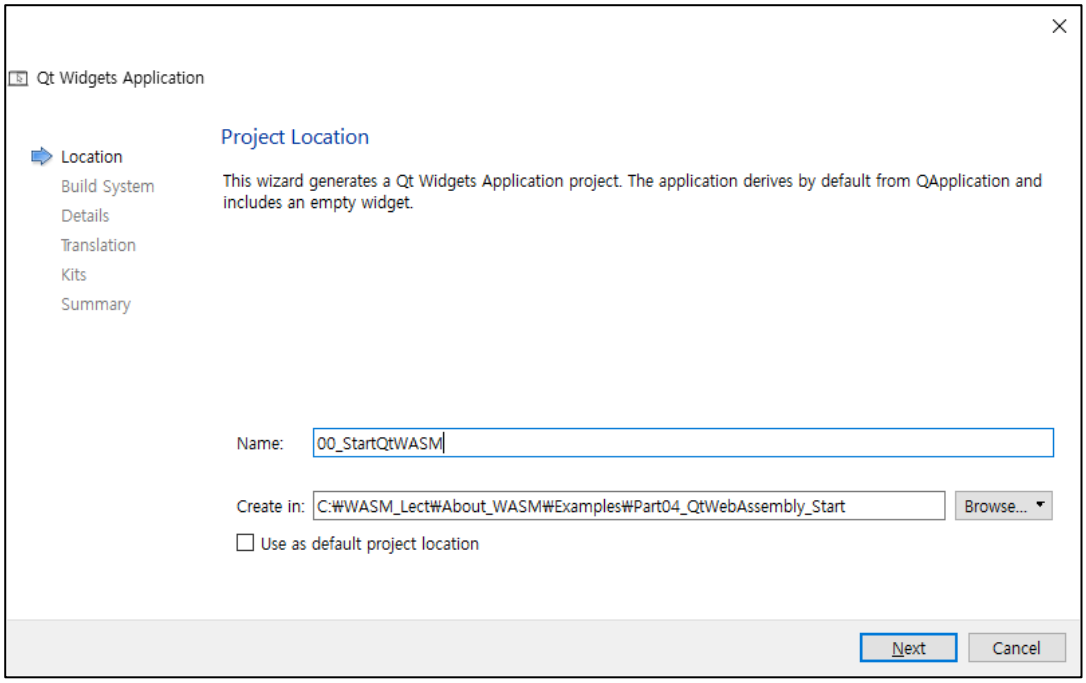
Run Qt Creator as shown in the image below.



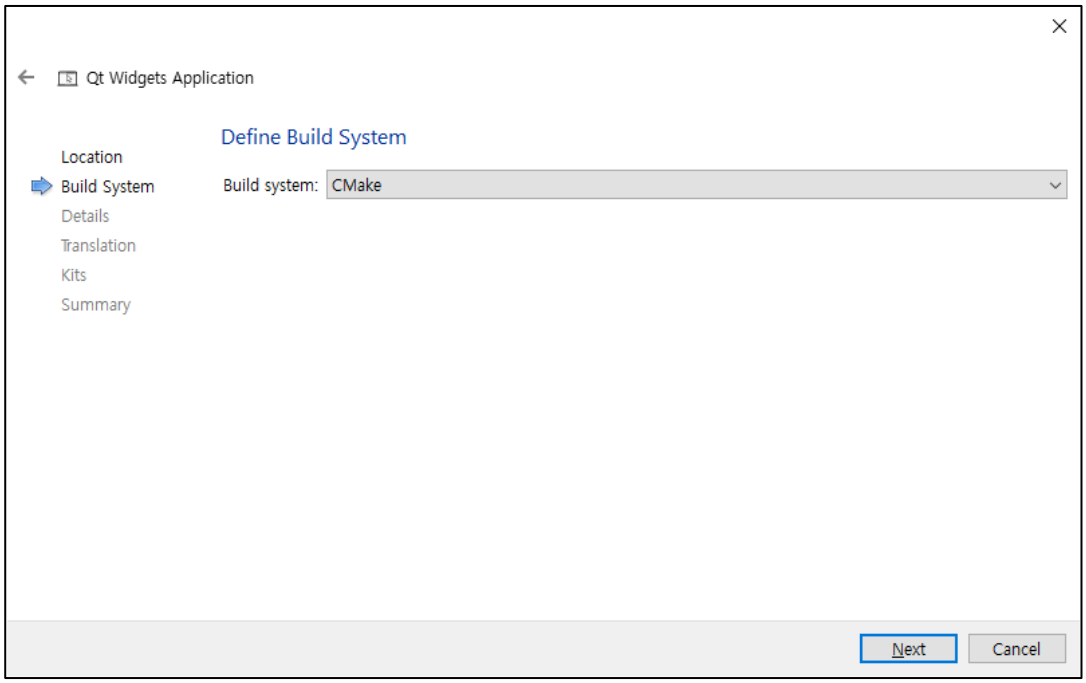
Click the Create project menu.



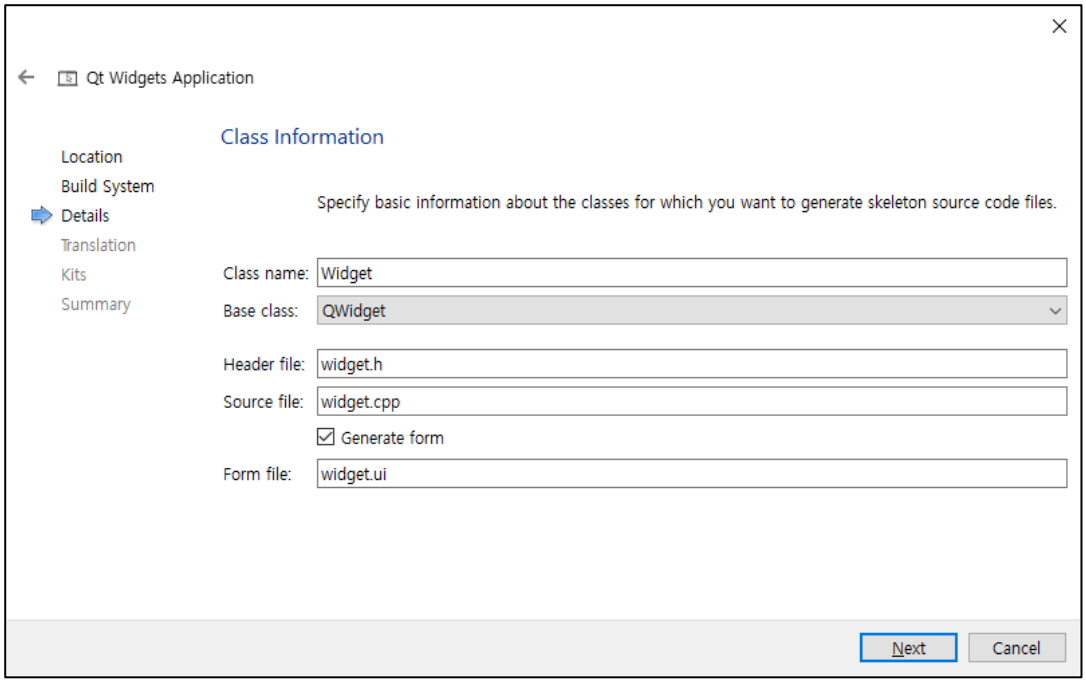
Select the [Application (Qt)] item, as shown in the image above, and then select the [Qt Widgets Application] from the middle tab.

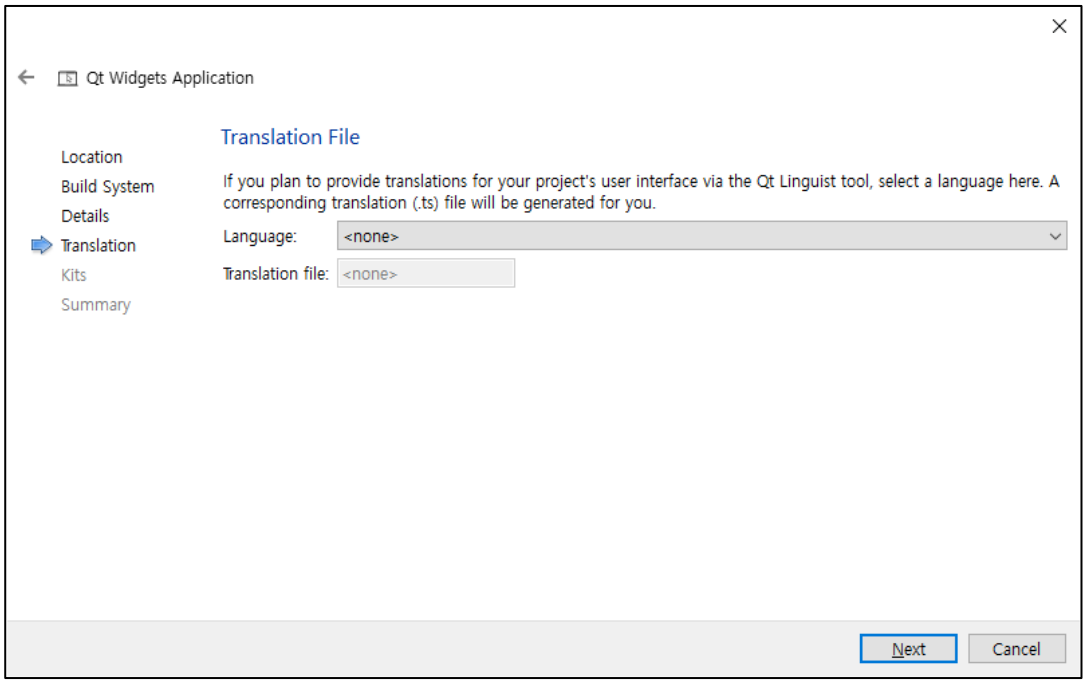


Under Build System, select CMake.

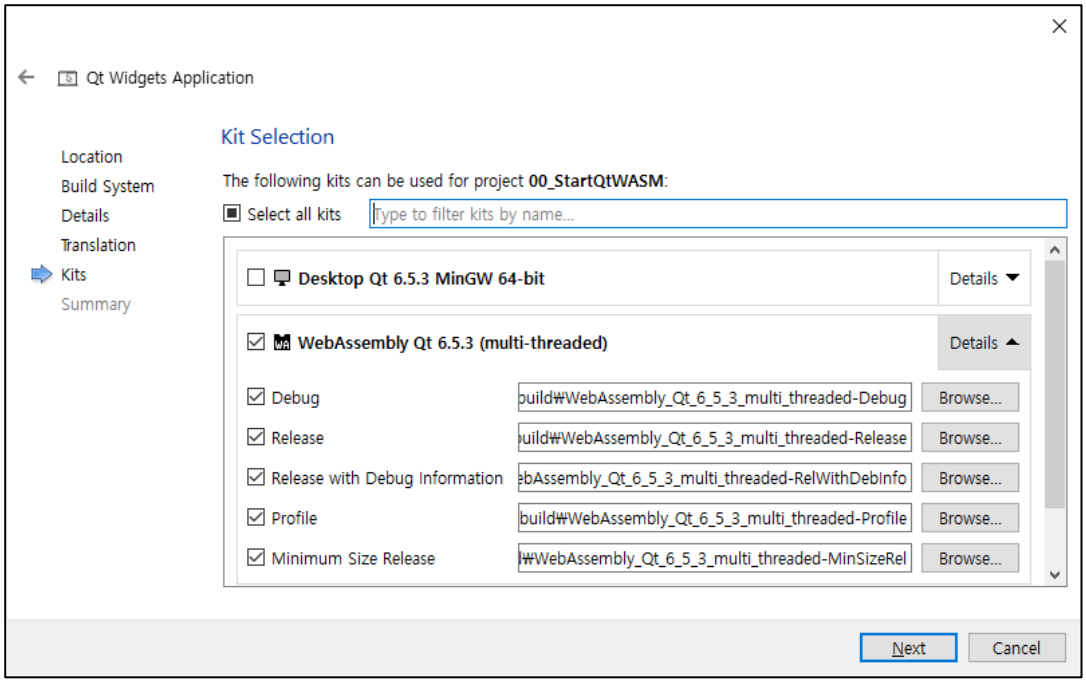


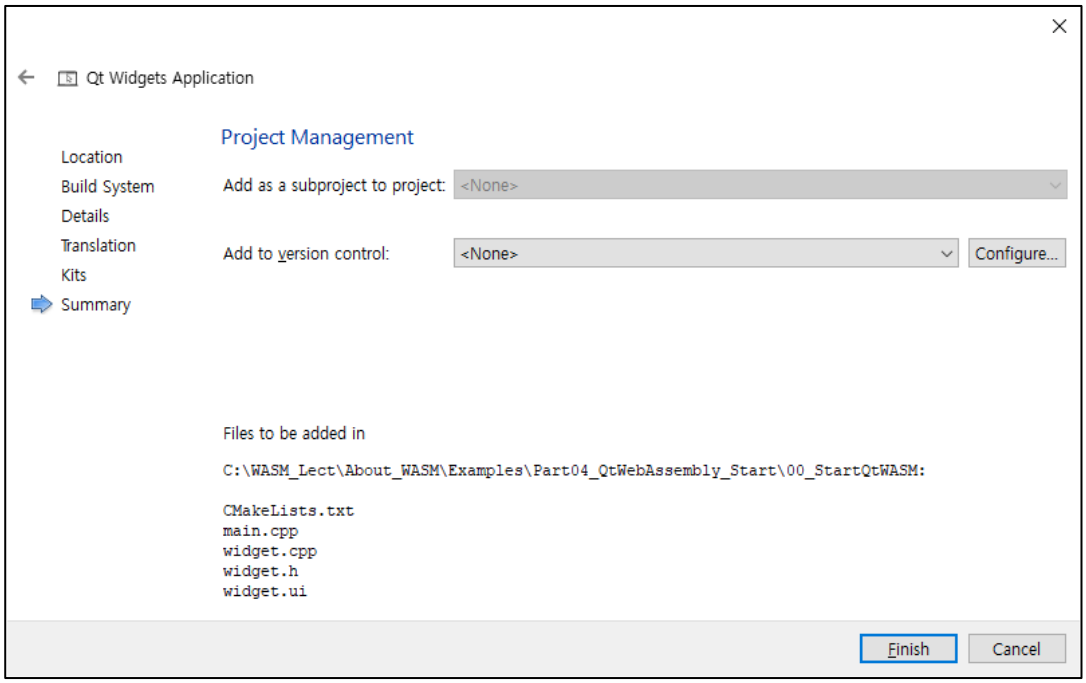
In the [Class Information] dialog, check the [Generate form] checkbox.



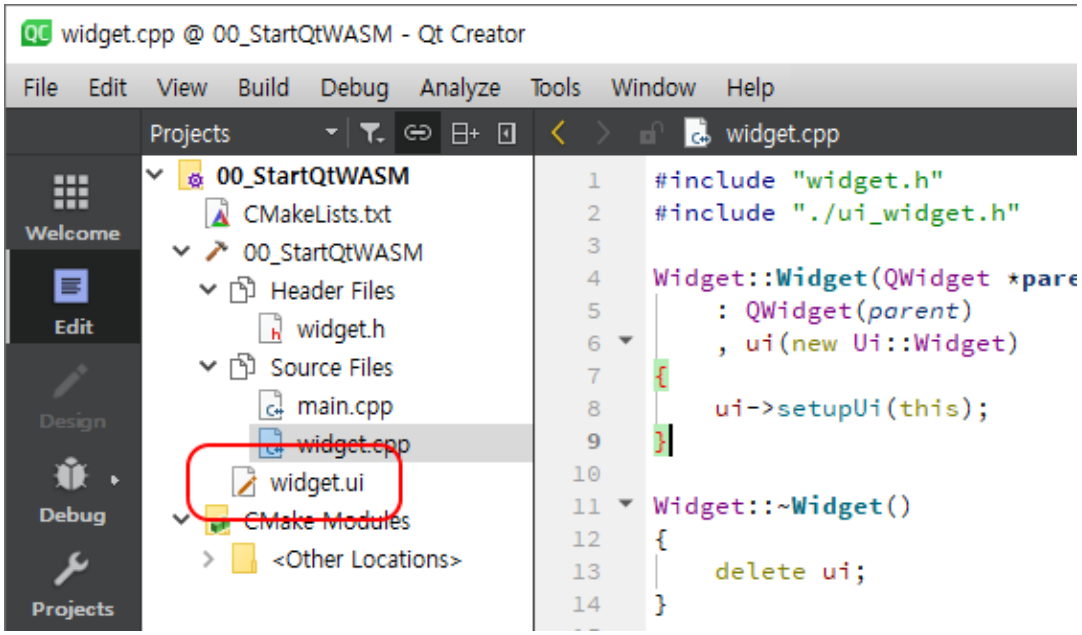


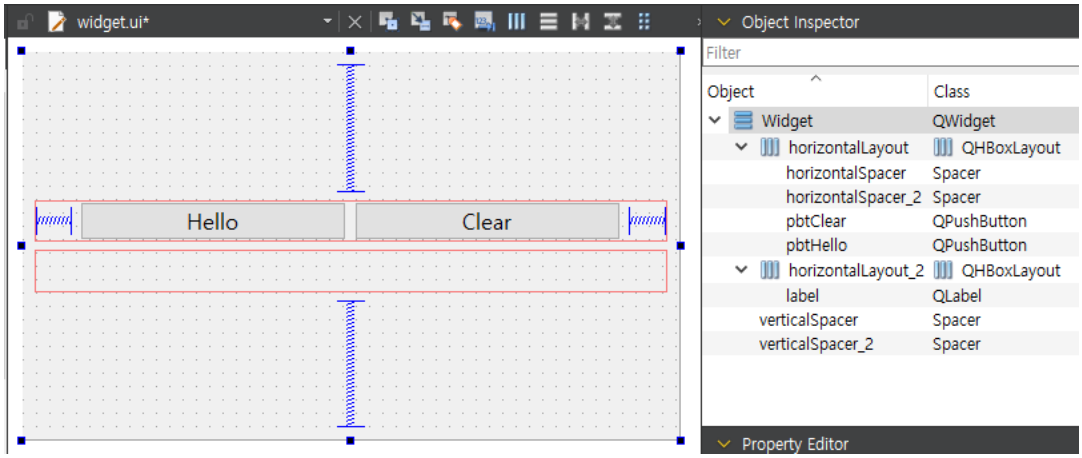
In the [Kit Selection] dialog, select [WebAssembly Qt 6.5.3 MinGW 64-bit].





Once the project is created, double-click the [widget.ui] file. This will load the Qt Designer screen.





Place and save the GUI Widgets as shown above. Then, create a widget.h header file like below.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

QT_BEGIN_NAMESPACE
namespace Ui {
```

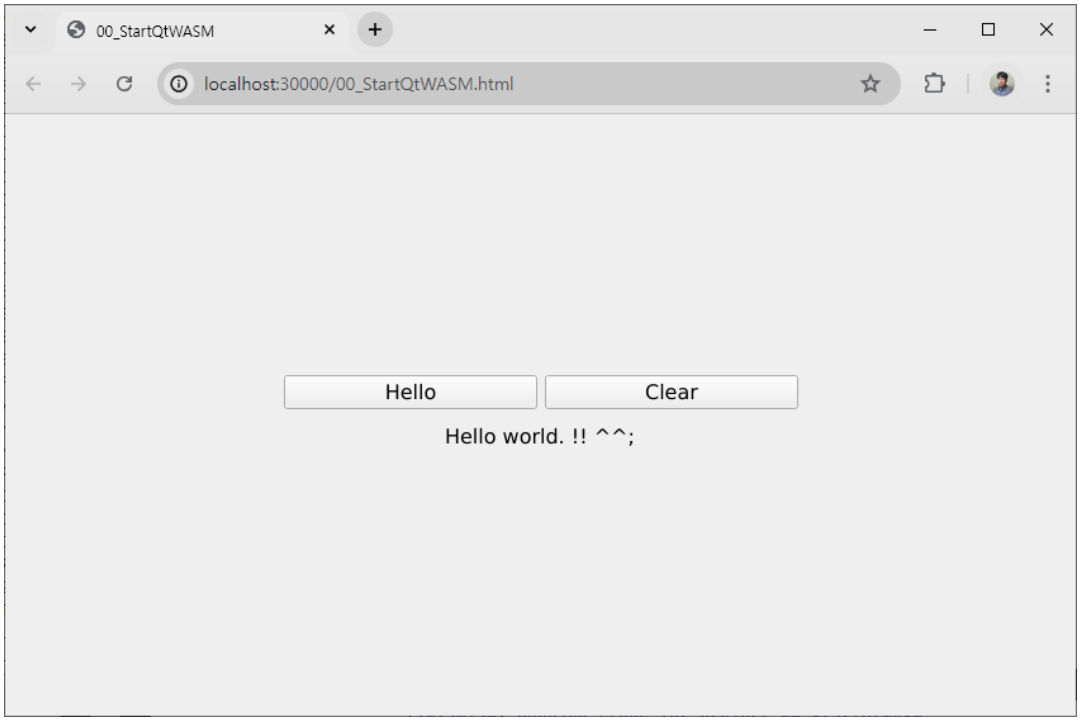
```
class Widget;
}
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

private slots:
    void slot_helloBtn();
    void slot_clearBtn();

};
#endif // WIDGET_H
```

Next, write the widget.cpp source code as shown below.

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

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pbtHello, SIGNAL(pressed()), this, SLOT(slot_helloBtn()));
    connect(ui->pbtClear, SIGNAL(pressed()), this, SLOT(slot_clearBtn()));
```

```
}

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

void Widget::slot_helloBtn()
{
    ui->label->setText("Hello world. !! ^^;");
}

void Widget::slot_clearBtn()
{
    ui->label->setText("");
}
```

Next, build the project. Then, let's run it.



Jesus loves you

6. Signal and Slot

Qt uses Signals and Slots as mechanisms for handling events. For example

that a button is clicked is called a Signal in Qt. And when a signal

is raised, the function that is called is called a Slot function. When an event called a signal occurs, the slot function associated with the signal is called.

A signal is an event that occurs when something happens. All events in Qt use the mechanism of signals and slots.

For example, in a chat program using Qt, user A sends a message to user B. From B's perspective, receiving the message from A is a signal.

It then calls the Slot function associated with the signal that received the message. Qt uses signals and slots for all event handling. In addition to the GUI and network, all modules (APIs) provided by Qt use the event mechanisms of signals and slots. Signals and slots simplify program source code, which can speed up development and simplify complex program structures.

Suppose you want to develop a network chat program without using Qt's signal slots. You need to develop a program with multiple threads. This can make your program complex because you need to use multiple threads, for example, a thread to handle messages from a particular user, a thread to handle whispers, and a thread to notify you when a new user is registered.

However, Qt's use of signals and slots makes it simple to implement a chat program without using threads.

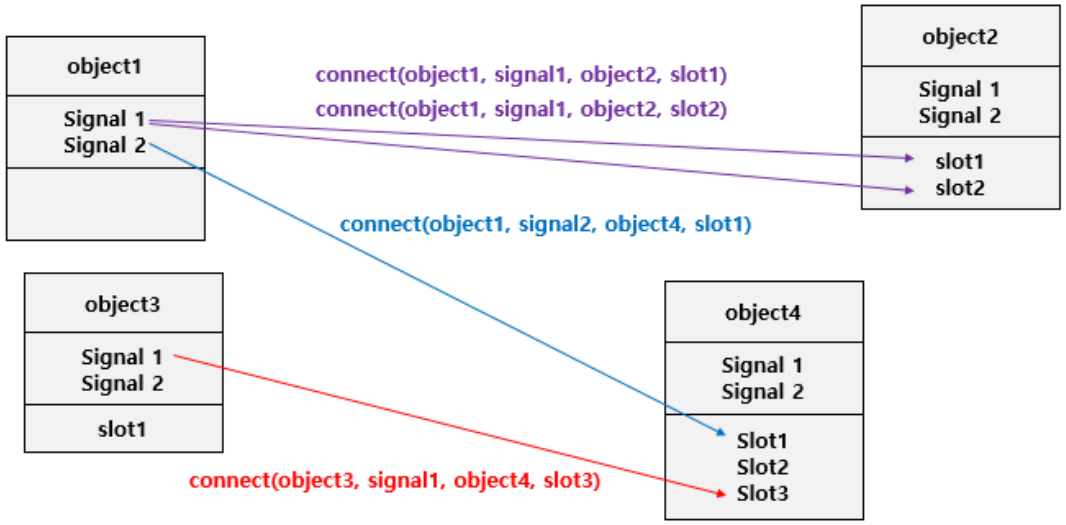
All GUI widgets provided by Qt have a variety of predefined signals. For example

QPushButton has various signals defined, such as click, double click, mouse over, etc.

for QPushButton.

You can think of signals and slots as a pipeline. A single signal can call multiple

slot functions. And multiple signals can call a single slot.



The function for connecting Signal and Slot functions is the connect( ) function of the QObject class.

to connect a Signal and a Slot. The first argument of the connect( ) member function is the object (instance of the class) on which the event occurred.

object (instance of the class) on which the event occurred, and the second argument is the object's signal (event).

For example, if you have a button named A, the object name of the button named A is the first argument, and the second

The second argument could be the signal of a click or a double-click on button A.

So we specify the click event as the second argument. The third argument specifies the name of the object that contains the signal and the slot function to call. The fourth argument specifies the slot function to call when the signal is fired.

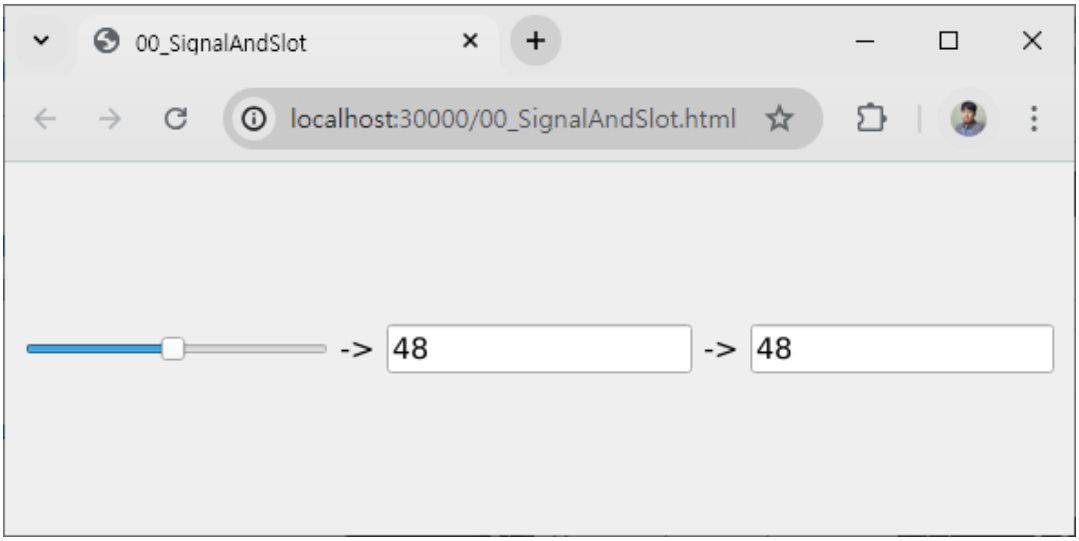
So far, we've covered the concepts of Signal and Slot. Now let's write an example program using them.

- Signal and Slot example

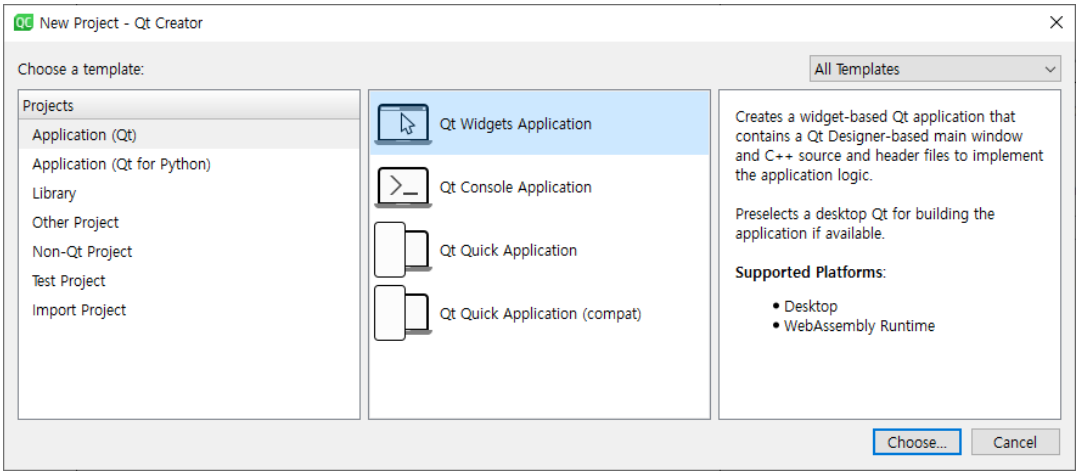
As shown in the figure below, when the first QSlider widget fires a Signal, it calls the Slot function associated with the Signal.

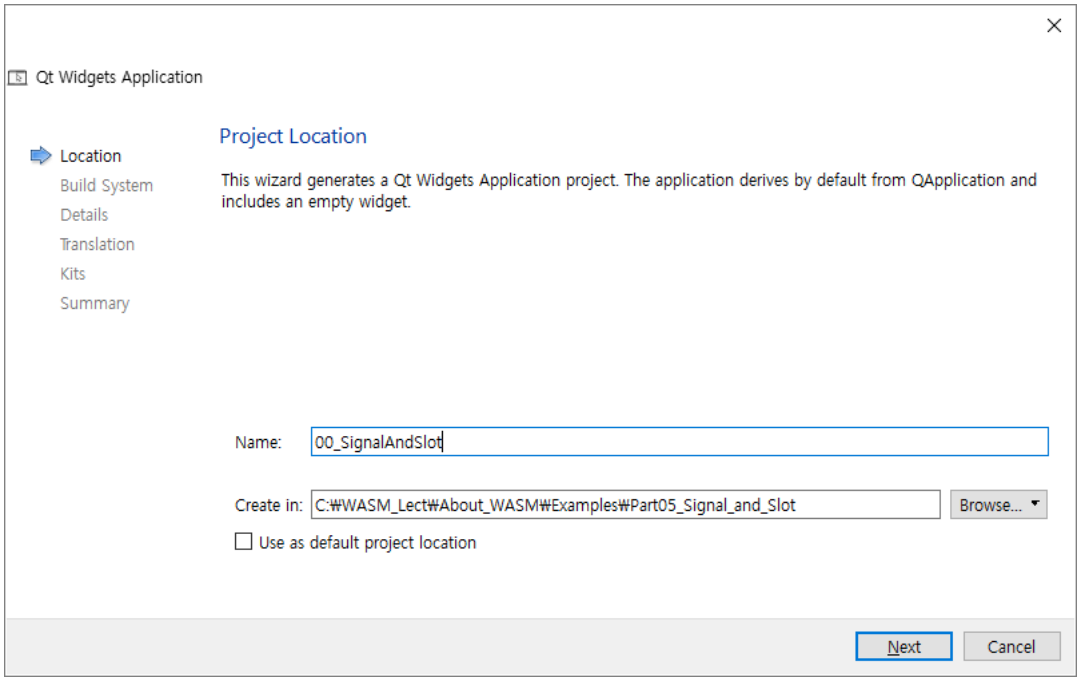
This Slot function displays the value of the QSlider in the second QLineEdit widget. It then fires the sig_textChanged( ) signal declared in this Slot function. This then executes

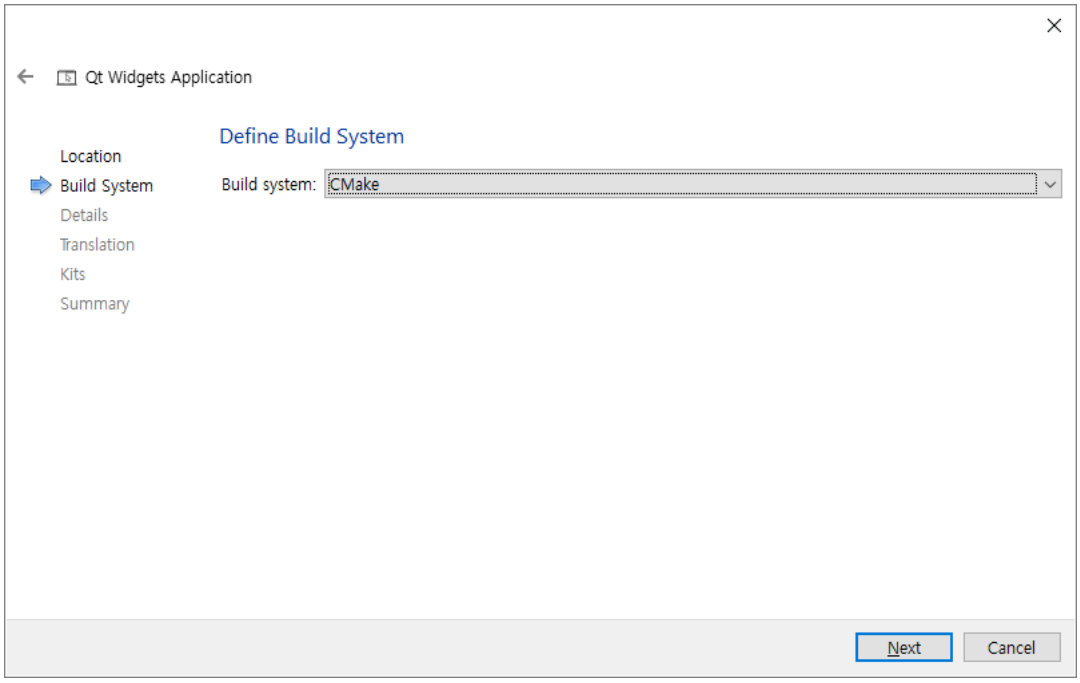
the Slot( ) function associated with this signal, which outputs the value to the third QLineEdit widget.

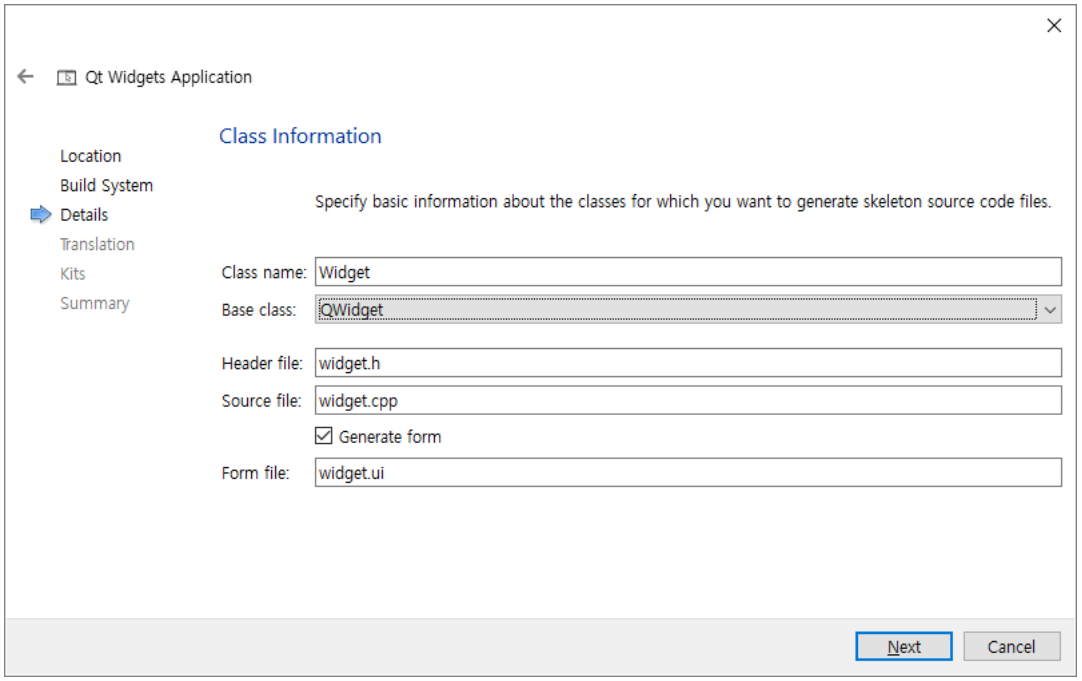


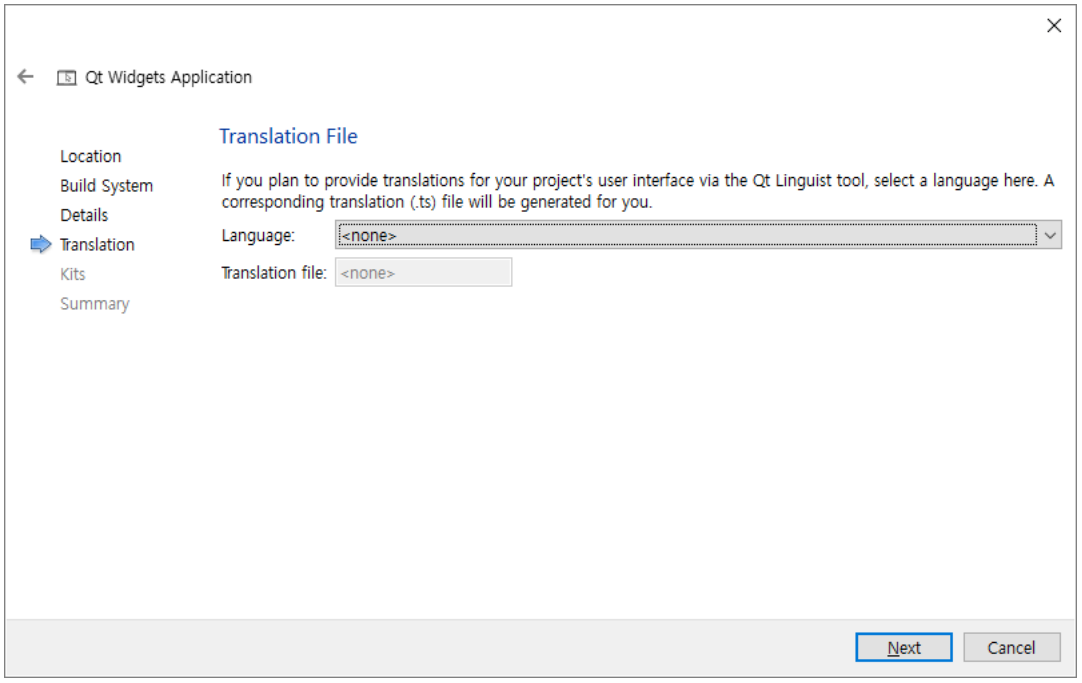
In Qt Creator, create a project as shown in the figure below.

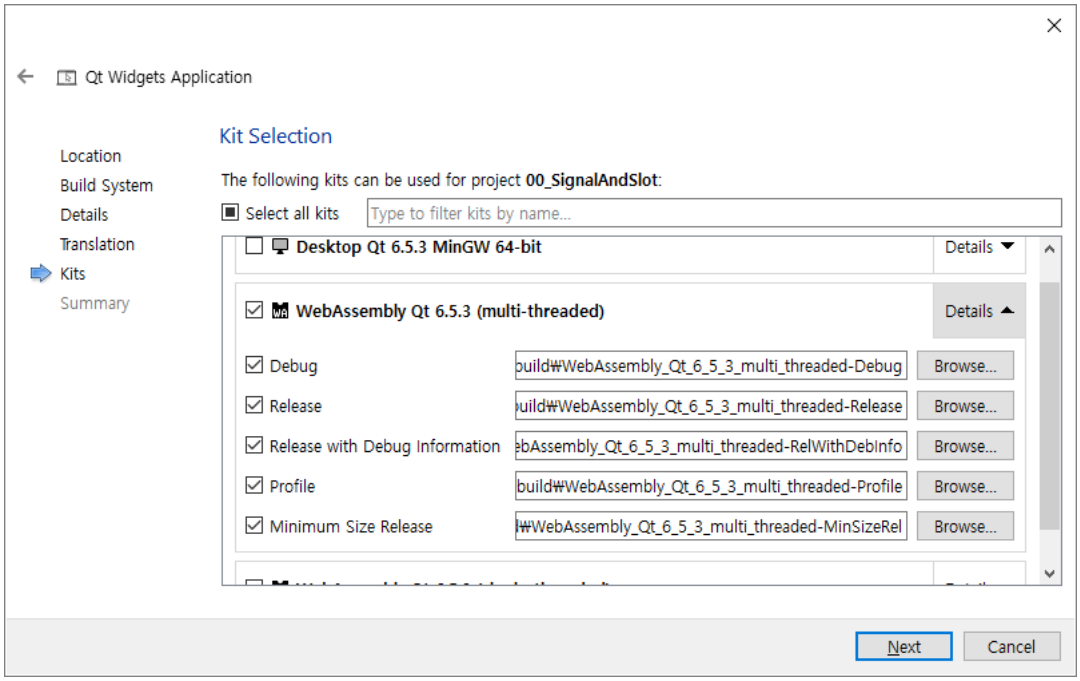



Qt Widgets Application
Location
Build System
Details
Translation
Kits
Summary
Project Location
This wizard generates a Qt Widgets Application project. The application derives by default from QApplication and includes an empty widget.
Name:
00_SignalAndSlot
Create in:
C:₩WASM_Lect₩About_WASM₩Examples₩Part05_Signal_and_Slot
Browse...
Use as default project location
Next
Cancel

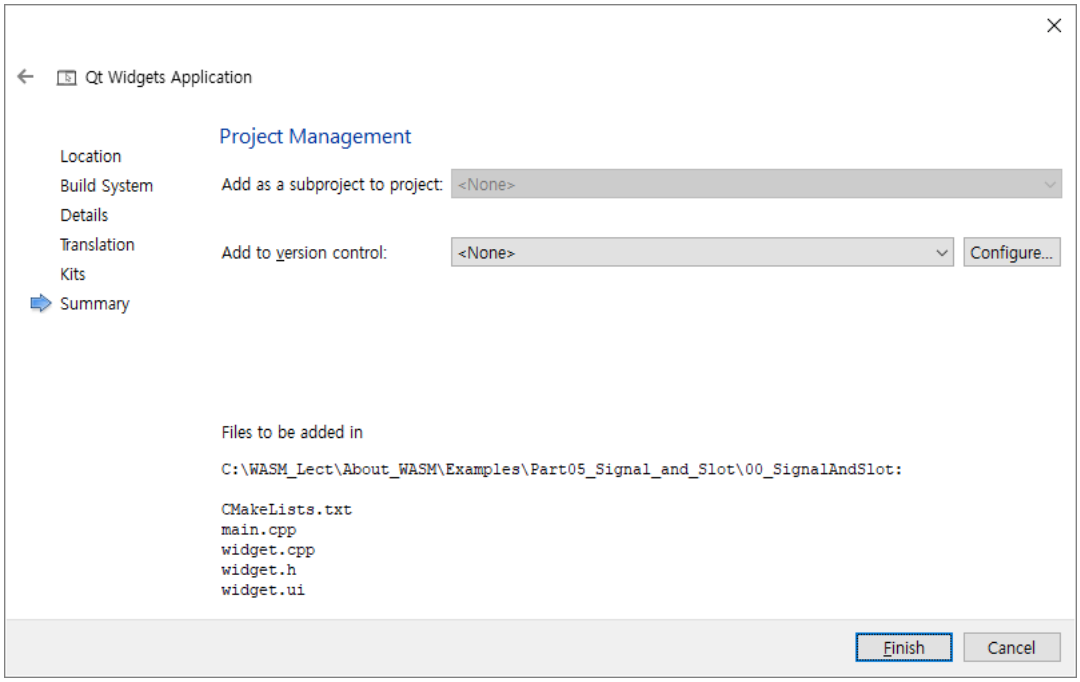

Qt Widgets Application
Location
Build System
Details
Translation
Kits
Summary
Define Build System
Build system:
CMake
Next
Cancel


Qt Widgets Application
Location
Build System
Details
Translation
Kits
Summary
Class Information
Specify basic information about the classes for which you want to generate skeleton source code files.
Class name: Widget
Base class: QWidget
Header file: widget.h
Source file: widget.cpp
Generate form
Form file: widget.ui
Next
Cancel

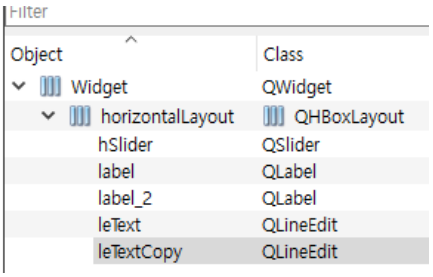

Qt Widgets Application
Location
Build System
Details
Translation
Kits
Summary
Translation File
If you plan to provide translations for your project's user interface via the Qt Linguist tool, select a language here. A corresponding translation (.ts) file will be generated for you.
Language: <none>
Translation file: <none>
Next
Cancel



Select the [WebAssembly Qt 6.x.x (multi-threaded)] item, as shown in the image above.



Create a project as shown above and double-click widget.ui in Qt Creator. In the Designer, arrange the GUIs as shown below.



Create the widget.h header file as shown above, and then create the widget.h header file as shown below.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

QT_BEGIN_NAMESPACE
namespace Ui {
class Widget;
}
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

signals:
    void sig_textChanged(QString str);
```

```
private slots:
    void slot_valueChanged(int val);
    void slot_textChanged(QString str);

};
#endif // WIDGET_H
```

Add the sig_textChanged( ) signal as shown in the figure above. This signal is called by the slot_valueChanged( ) function.

Next, add the slot_valueChagned( ) function and the slot_textChanged( ) function.

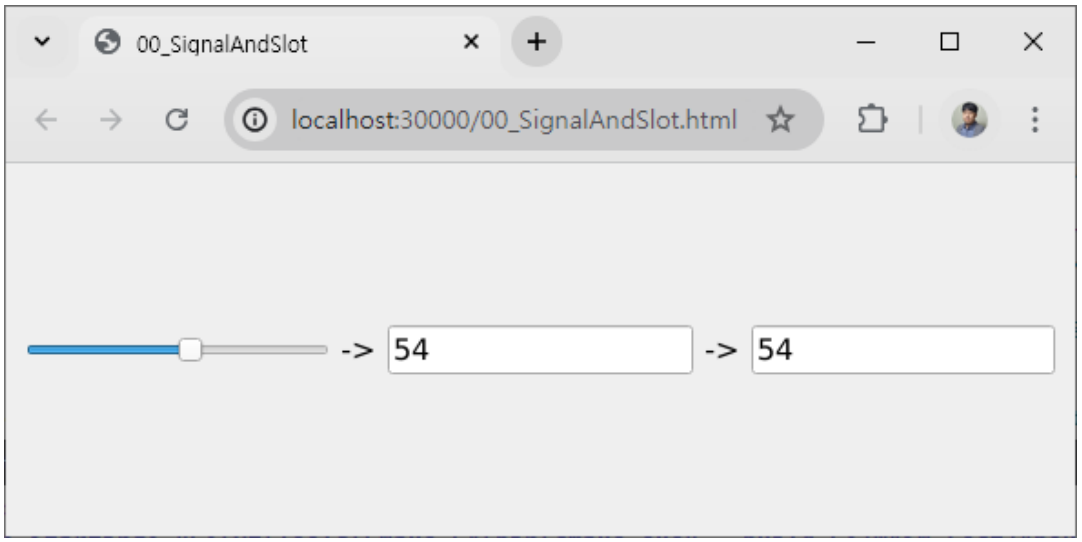
The slot_valueChagned(int val) function is called when the value of the QSlider changes, and it sets the value of the changed QSlider to the value of the first QLineEdit. The sig_textChange( ) signal is then called. This signal function is associated with the slot_textChanged( ) function. So in this function, the same value as the first QLineEdit's value is set in the second QLineEdit.

Next, create the widget.cpp source code file as shown below.

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

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

    connect(ui->hSlider, SIGNAL(valueChanged(int)),
            this, SLOT(slot_valueChanged(int)));

    connect(this, SIGNAL(sig_textChanged(QString)),
            this, SLOT(slot_textChanged(QString)));
```

```
}

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

void Widget::slot_valueChanged(int value)
{
    QString str = QString("%1").arg(value);
    ui->leText->setText(str);

    emit sig_textChanged(str);
}

void Widget::slot_textChanged(QString str)
{
    ui->leTextCopy->setText(str);
}
```

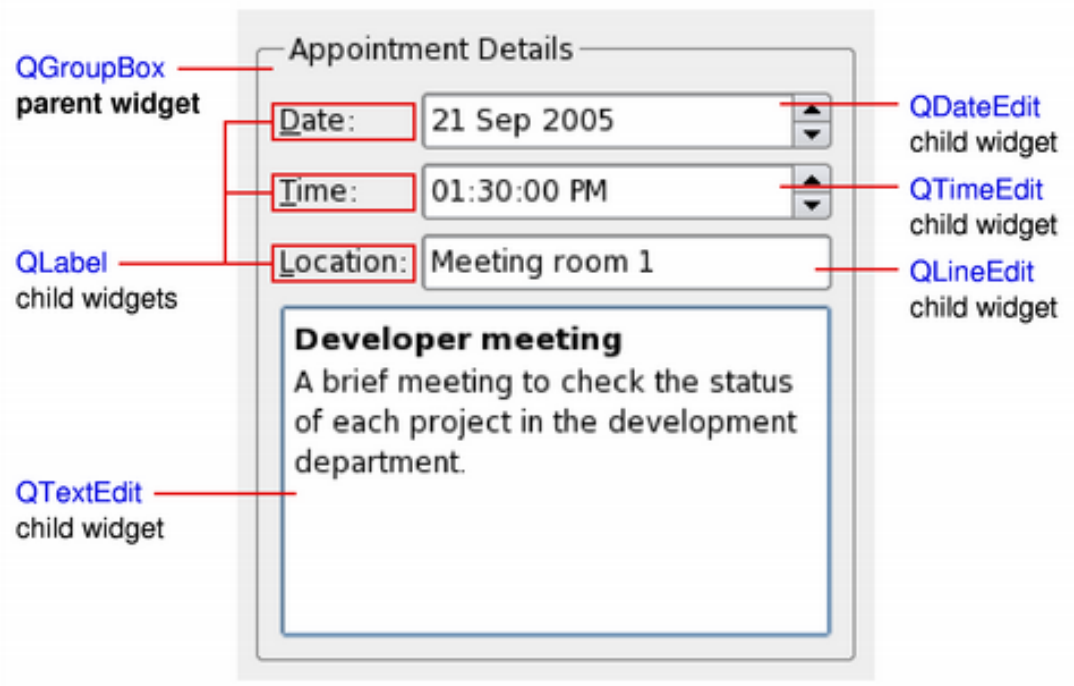
Let's finish writing the source code as above and then build and run it.



Jesus loves you

7. Widget and Layouts

In this chapter, we'll learn about Widgets and Layouts. Widgets are buttons, combo boxes, labels, and so on. And Layout is used to arrange Widgets. When you use a Layout, the size of the Widgets in the Layout changes dynamically when the size of the Window changes.

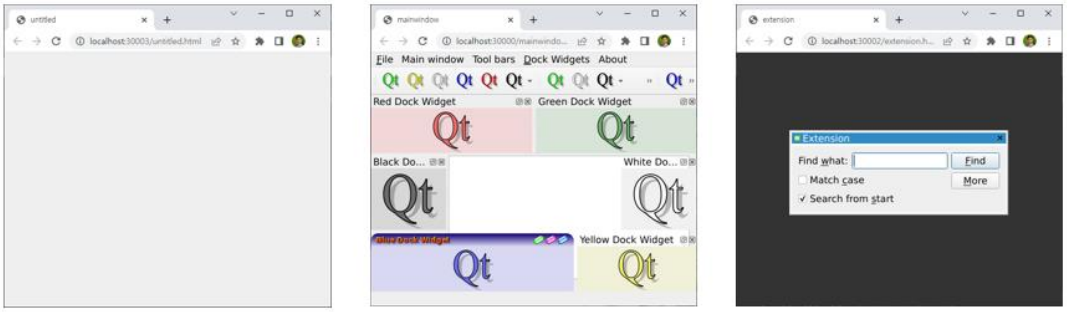


- QWidget, QMainWindow and QDialog

A QWidget can handle GUI events such as mouse, keyboard, etc., draw shapes, lines, etc. like 2D graphics, and render images. All widgets provided by Qt are implemented based on QWidget.

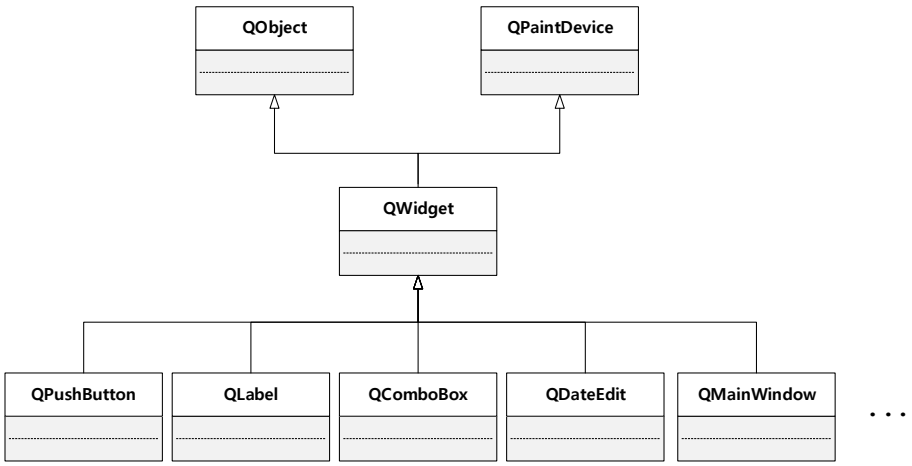
The QMainWindow is divided into the Menu Bar, Tool Bar, Center Widget area, and Status Bar area, and is mainly used in GUI applications on desktop PCs.

QMainWindow is also implemented by inheriting QWidget, and QDialog is also implemented based on QWidget.

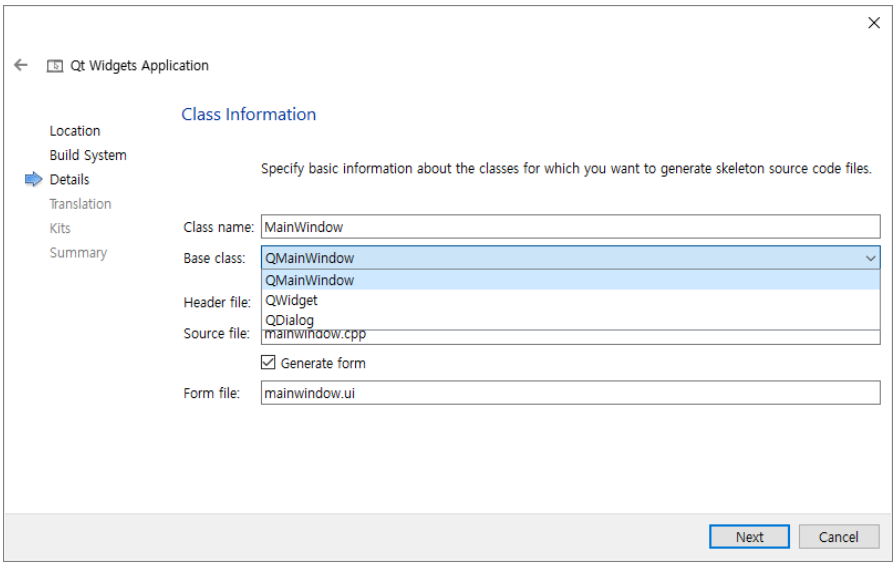


QWidget QMainWindow QDialog

QWidget inherits from QObject and QPaintDevice. QObject implements functions such as Signal and Slot, and QPaintDevice implements functions related to 2D graphics.

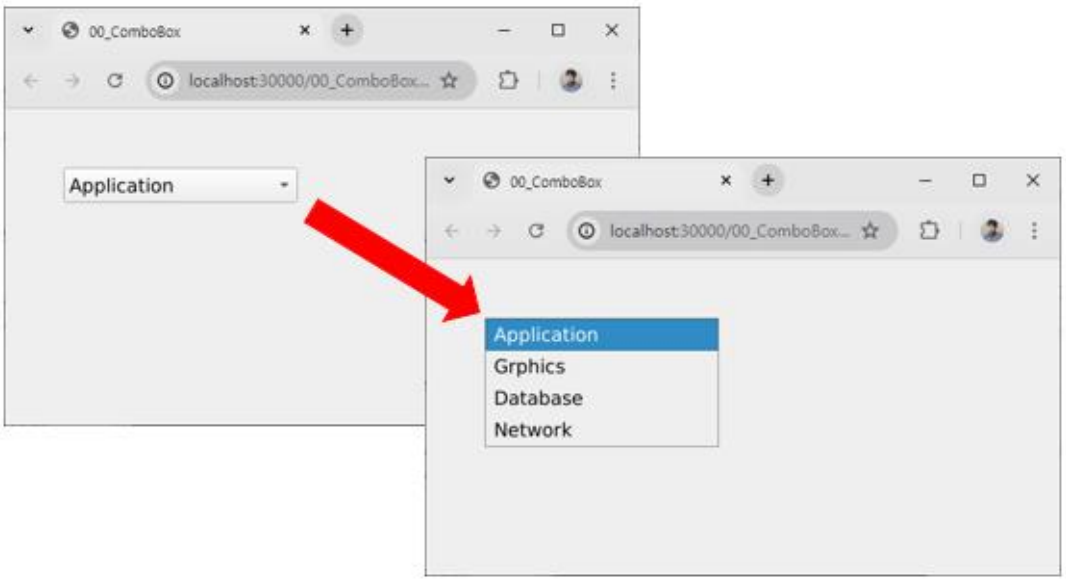


Qt WebAssembly can generate the widgets for the windows you want to create in the following dialog when you first create a project and select a Qt Widget-based application.



- QComboBox

QComboBox provides a GUI interface to select one of the items, as shown in the figure below.

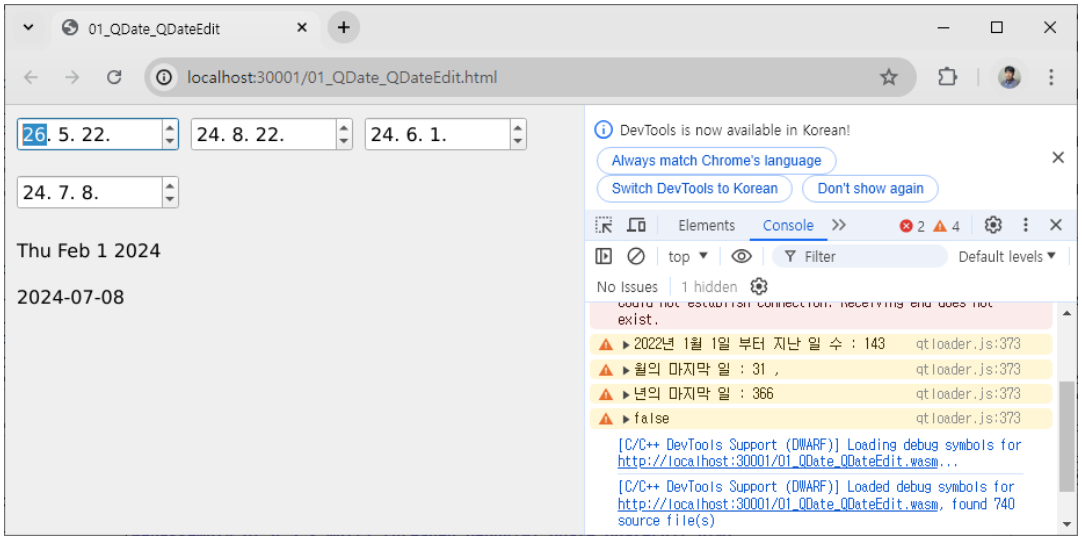



```
#include "widget.h"
#include <QComboBox>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    QComboBox *combo = new QComboBox(this);
    combo->setGeometry(50, 50, 200, 30);

    combo->addItem("Application");
    combo->addItem("Grphics");
    combo->addItem("Database");
    combo->addItem("Network");
}
```

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

- QDate and QDateEdit

QDate provides functions for handling dates. You can use the QDateEdit widget to display objects of class QDate on the GUI.

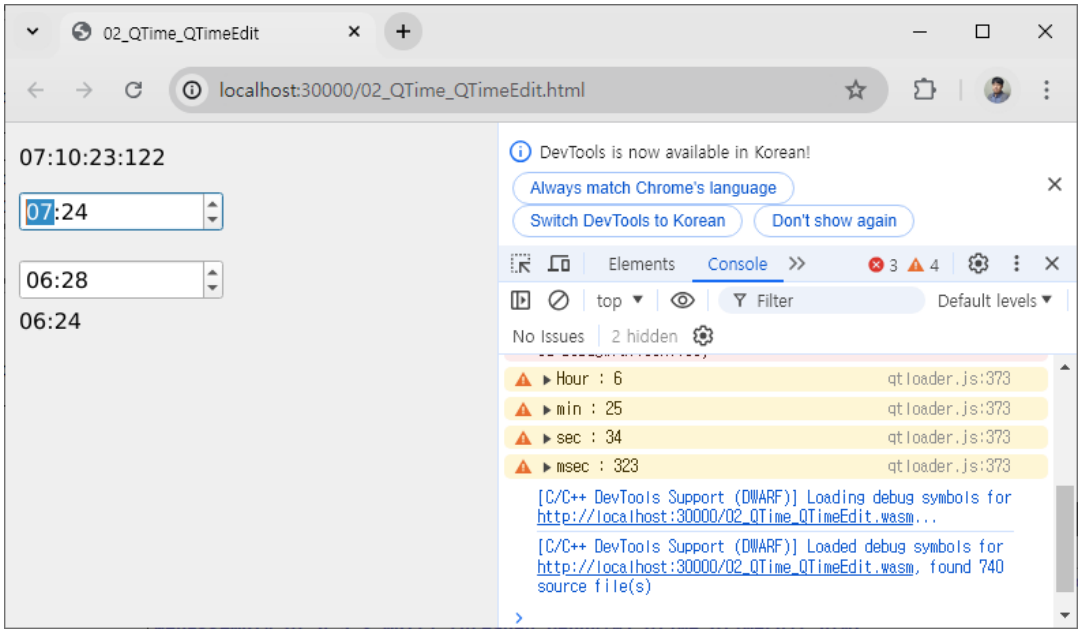



```
QDateEdit *dateEdit[4];
QLabel *lbl[6];

QDate dt1 = QDate(2024, 5, 22);
QDate dt2 = QDate::currentDate();

dateEdit[0] = new QDateEdit(dt1.addYears(2), this);
dateEdit[0]->setGeometry(10, 10, 140, 30);

dateEdit[1] = new QDateEdit(dt1.addMonths(3), this);
dateEdit[1]->setGeometry(160, 10, 140, 30);
```

```
dateEdit[2] = new QDateEdit(dt1.addDays(10), this);
dateEdit[2]->setGeometry(310, 10, 140, 30);

dateEdit[3] = new QDateEdit(dt2, this);
dateEdit[3]->setGeometry(10, 60, 140, 30);

if(QDate::isValid(2024, 2, 41))
    qDebug("true");
else
    qDebug("false");

QDate dt3 = QDate(2024, 2, 1);
QDate dt4 = QDate::currentDate();

lbl[0] = new QLabel(dt3.toString(), this);
lbl[0]->setGeometry(10,110, 150, 30);
lbl[1] = new QLabel(dt4.toString("yyyy-MM-dd"), this);
lbl[1]->setGeometry(10,150, 150, 30);
```

- QTime and QTimeEdit

QTime provides the ability to handle time. QTimeEdit provides the ability to use QTime to display it in a GUI.

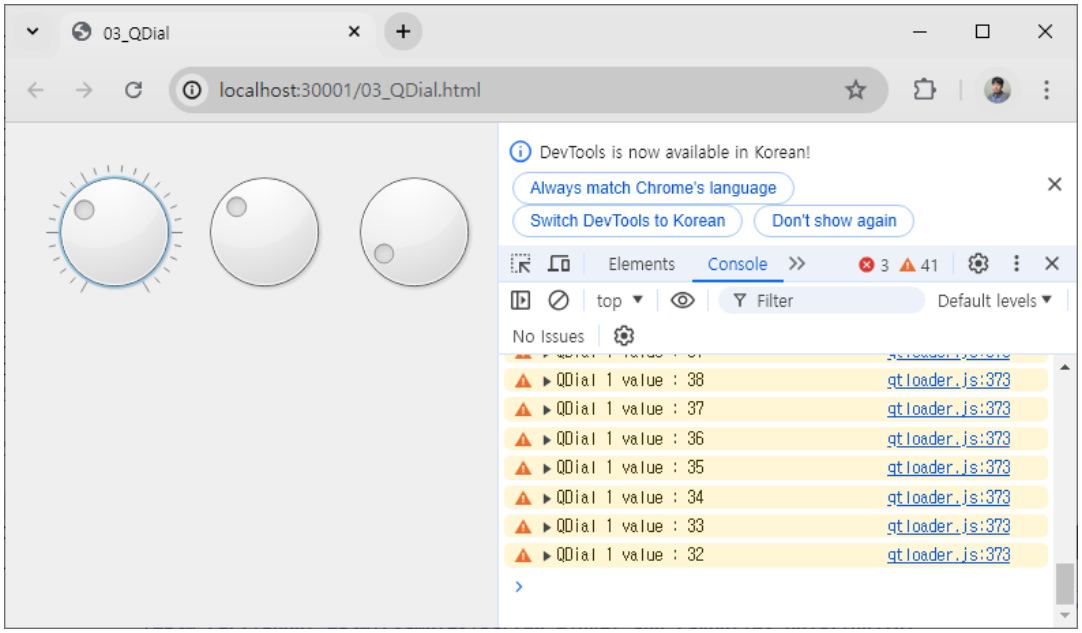



```
QTime ti0 = QTime(7, 10, 23, 122);

QLabel *lbl_toString;
Lbl_toString = new QLabel(ti0.toString("hh:mm:ss:zzz"), this);
lbl_toString->setGeometry(10, 10, 150, 30);

// Hour,Min,Sec,Millisecond
QTime ti1 = QTime(6, 24, 55, 432);
QTimeEdit *qte[2];
qte[0] = new QTimeEdit(ti1, this);
qte[0]->setDisplayFormat("hh:mm");
qte[0]->setGeometry(10, 50, 150, 30);

qte[1] = new QTimeEdit(ti1.addSecs(200), this);
qte[1]->setDisplayFormat("hh:mm");
qte[1]->setGeometry(10, 100, 150, 30);

QLabel *lbl_fromString = new QLabel(ti1.toString("hh:mm"), this);
```

```
lbl_fromString->setGeometry(10, 130, 150, 30);
QTime ti5 = QTime(6, 25, 34, 323);

qDebug("Hour : %d", ti5.hour());
qDebug("min : %d", ti5.minute());
qDebug("sec : %d", ti5.second());
qDebug("msec : %d", ti5.msec());
```

- QDial

QDial provides the user with a Dial-like GUI interface.

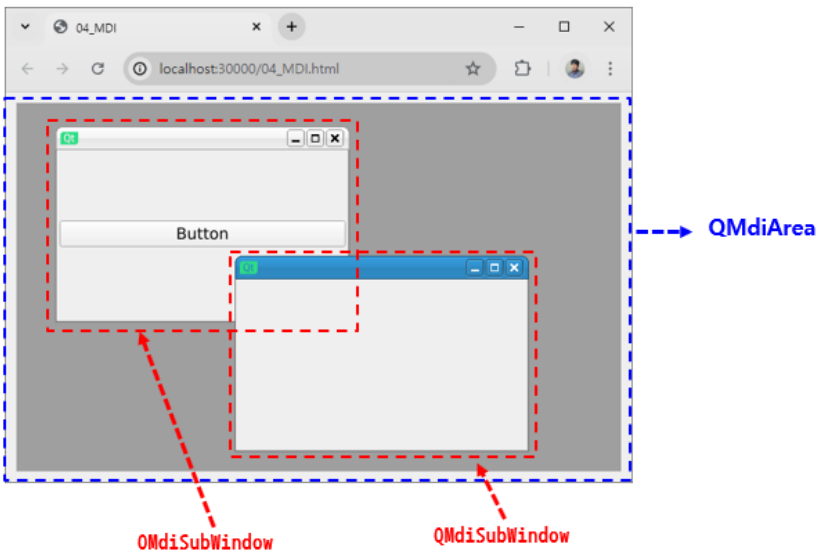



```
#include "widget.h"
Widget::Widget(QWidget *parent) : QWidget(parent)
{
    int xpos = 30;
    for(int i = 0 ; i < 3 ; i++, xpos += 110)
    {
        m_dial[i] = new QDial(this);
```

```
            m_dial[i]->setRange(0, 100);
            m_dial[i]->setGeometry(xpos, 30, 100, 100);
        }
        m_dial[0]->setNotchesVisible(true);
        connect(m_dial[0], SIGNAL(valueChanged(int)),
                    this,          SLOT(changedData()));
}
void Widget::changedData()
{
        qDebug("QDial 1 value : %d", m_dial[0]->value());
}
Widget::~Widget()
{
}
```

- QMdiArea

QMdiArea can create multiple windows within a window pane. Therefore, you can create and use sub-windows within a window pane.



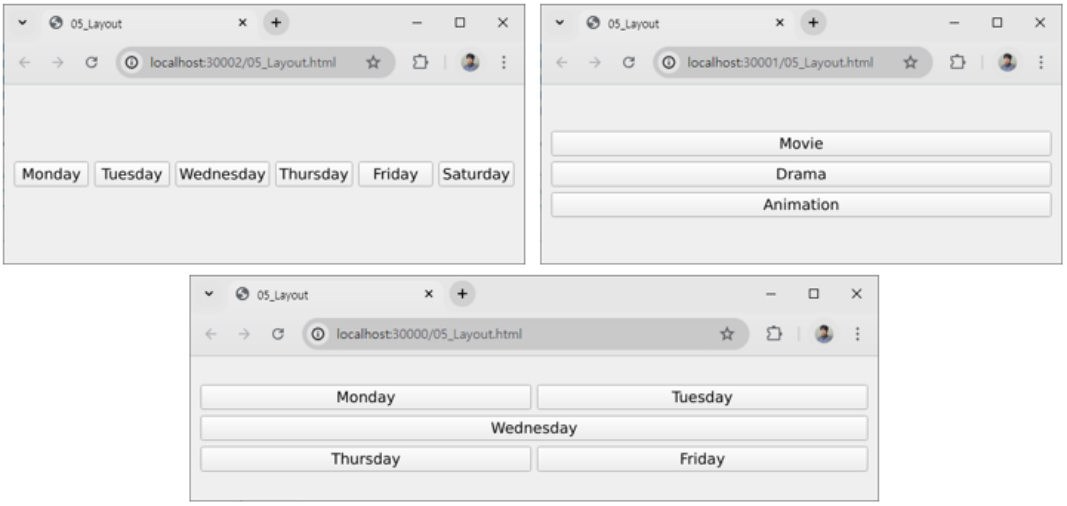

```
#include "widget.h"
```

```
#include <QMdiArea>
#include <QMdiSubWindow>
#include <QPushButton>
#include <QHBoxLayout>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    setWindowTitle(QString::fromUtf8("MDI Example"));
    setFixedSize(600,400);

    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);

    QHBoxLayout *hboxLayout = new QHBoxLayout();
    hboxLayout->addWidget(area);

    setLayout(hboxLayout);
}
```

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

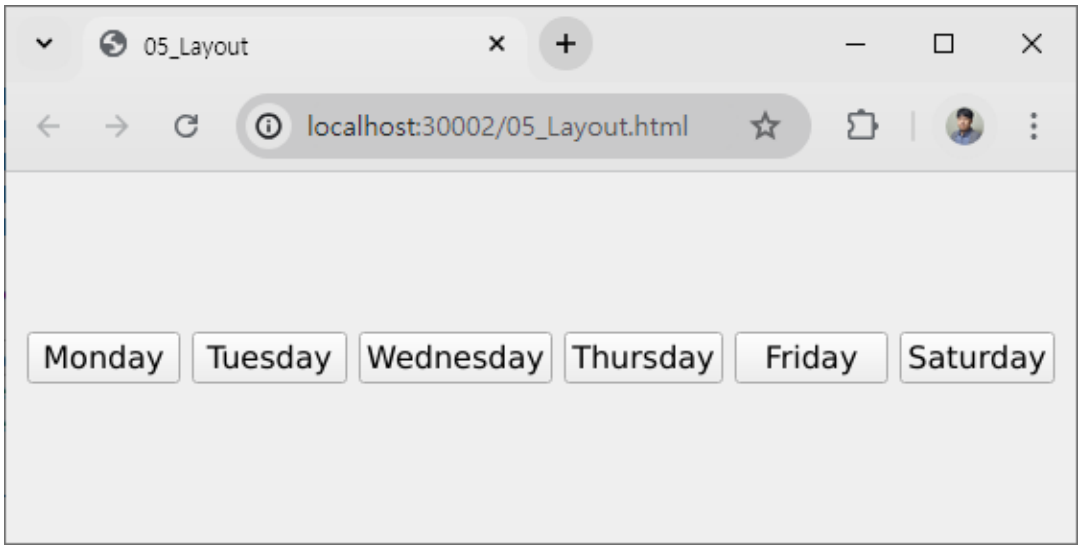
- Favorite Layout

Qt provides a variety of layouts. In this section, we will discuss the most commonly used layouts: QHBoxLayout, QVBoxLayout, and QGridLayout.



- QHBoxLayout

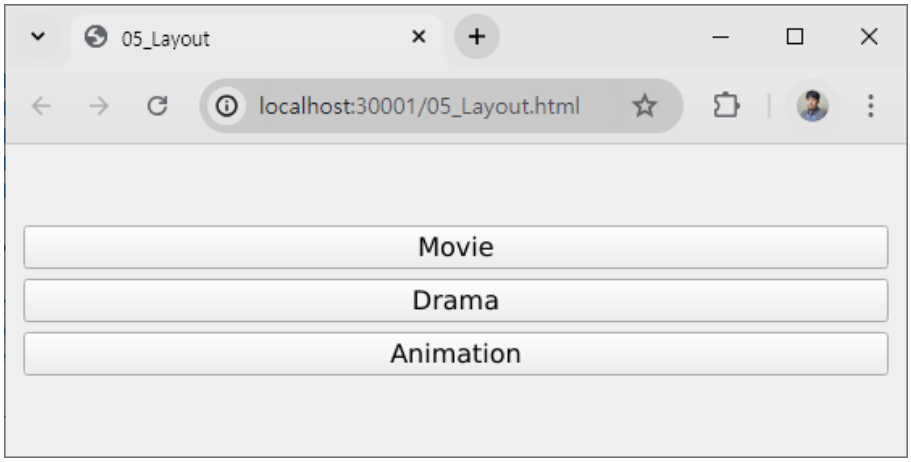
Position the Widgets in horizontal orientation.



```
...
QHBoxLayout *hboxLayout = new QHBoxLayout(this);
QPushButton *btn[6];
Qstring btnStr[6] = {"Monday", "Tuesday", "Wednesday", "Thursday",
                     "Friday", "Saturday"};

for( int i = 0 ; i < 6 ; i++ )
{
    btn[i] = new QPushButton(btnStr[i]);
    hboxLayout->addWidget(btn[i]);
}
setLayout(hboxLayout);
...
```

- QVBoxLayout

QVBoxLayout arranges the Widgets in a new orientation.

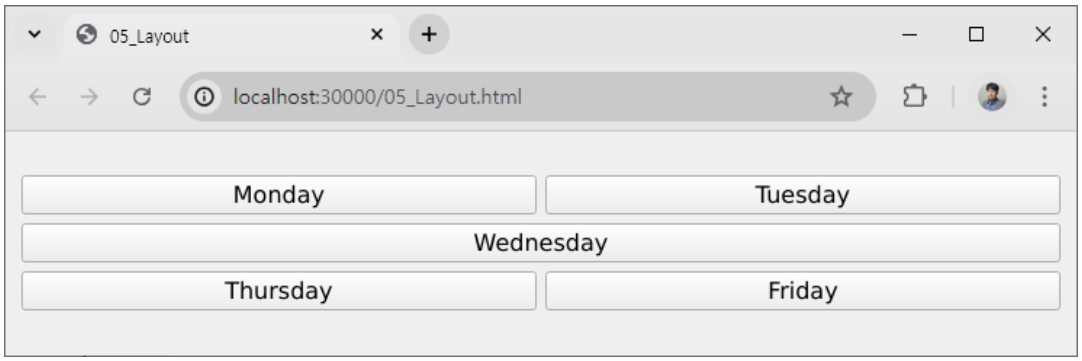



```
...

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

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

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

- QGridLayout

This Layout arranges Widgets like a Cell.

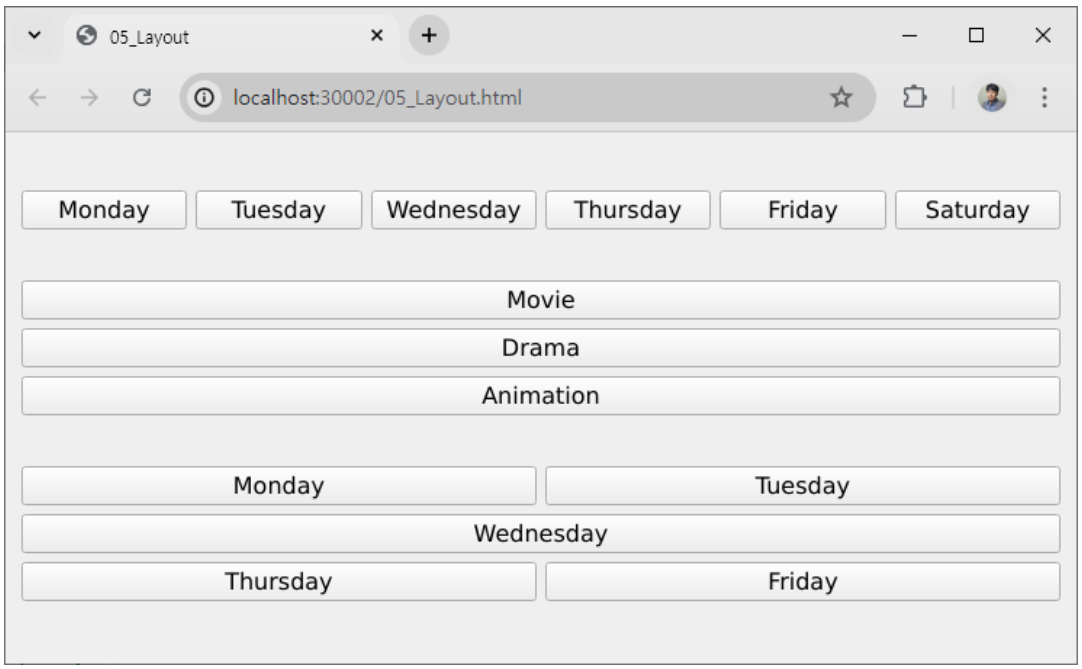



```
...

QGridLayout *gridLayout = new QGridLayout();
QPushButton *gbtn[5];
QString gbtnStr[6] = {"Monday", "Tuesday", "Wednesday",
                      "Thursday", "Friday", "Saturday"};
for(int i = 0 ; i < 5 ; i++)
{
    gbtn[i] = new QPushButton(gbtnStr[i]);
}
```

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

QVBoxLayout *defaultLayout = new QVBoxLayout();

defaultLayout->addLayout(gridLayout);
setLayout(defaultLayout);

...
```

- Using Layouts as a nested structure

When implementing GUIs, we use different layouts. So, let's create the following GUI using a nested layout as an example.



```
#include "widget.h"
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QGridLayout>
#include <QPushButton>

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

    QString btnStr[6] = {"Monday", "Tuesday", "Wednesday",
                         "Thursday", "Friday", "Saturday"};

    for(int i = 0 ; i < 6 ; i++)
    {
        btn[i] = new QPushButton(btnStr[i]);
        hboxLayout->addWidget(btn[i]);
    }

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

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

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

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

    QString gbtnStr[6] = {"Monday", "Tuesday", "Wednesday",
                          "Thursday", "Friday", "Saturday"};


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

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

    QVBoxLayout *defaultLayout = new QVBoxLayout();

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

    setLayout(defaultLayout);
}

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

8. Basic data types and useful types

When you declare the type of a variable when implementing your source code, use a type such as int or float. You can use any variable type used in C/C++. Qt also provides more descriptive variable type names, as shown in the table below.

Type	Size (Bit)	Description
bool	8	true / false
qint8	8	signed char
qint16	16	signed short
qint32	32	signed int
qint64	64	long long int
quint8	8	unsigned char
quint16	16	unsigned short
quint32	32	unsigned int
quint64	64	unsigned long long int
float	32	floating point number
double	64	floating point number
const char*	32	문자열 상수를 가리키는 포인터, 마지막에 0은 제외

There are many more useful types available. You can find more information by searching for the keyword "Qt Type Declarations".

Qt provides a number of template functions to easily handle data types. For example, it provides the qAbs( ) function to get absolute values. To find a value between a maximum and minimum value, qBound( ) is provided. The qMax( ) and qMin( ) functions are provided to find the maximum and minimum values.

- Finding absolute values

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

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

- Finding a value between a minimum and maximum value

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

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

- Functions 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
```

- Find the maximum value

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

- Rounding

```
qreal valueA = 2.3;
qreal valueB = 2.7;
```

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

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

In addition to classes that deal with simple strings, Qt provides a variety of classes that support data streams, multibyte character types

Class name	Description
QBitArray	Provides bitwise data manipulation and operators for AND, OR, and NOT.
QByteArray	1 Byte array type
Qstring	Stores a string, using a QChar (16-bit Unicode character) to store a single string.
QVariant	Useful when dealing with undefined types such as the void type.
QPoint	QPointF is used to store X, Y values QPointF is used to store decimal points.
QRect	Used to store X, Y, WIDTH, and HEIGHT values. QRectF is used to store decimal points.
QRegExp	Classes for handling normalized expressions
QRegion	For checking if certain regions overlap, for example, union regions, intersection regions, etc. in a set.
QSize	Classes for storing width and height
QVariant	Classes for union types that can be stored as void types.
QVector2D	Classes for representing vectors or vertices in two-dimensional space.
QVector3D	Classes that can take X, Y, and Z coordinates

QVector4D	For representing vectors or vertices in four-dimensional space

- QBitArray

QBitArray is used for the purpose of handling data in bits. To declare 200 bits, you can use it like this

```
QBitArray ba(200);
```

To dynamically declare the size, you can use the resize( ) member function as shown below.

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

To change the value of a specific bit, you can use the setBit( ) function.

```
QBitArray ba(3);
ba.setBit(0, true);
ba.setBit(1, false);
ba.setBit(2, true);
```

When you want to use a speaker, you can use it like this

- QByteArray

Class QByteArray provides an array of bytes (8-bit). Class QByteArray provides append( ), prepend( ), insert( ), replace( ), and remove( ) members for handling arrays the append( ), prepend( ), insert( ), replace( ), and remove( ) member functions

functions to handle arrays.

```
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
```

- QByteArray

It is stored in 1Byte units (8bits). It can be stored as a string, hex, etc. It also provides member functions to dynamically add, modify, and delete sizes.

```
QByteArray ba("Hello");
```

To dynamically resize, you can use the resize( ) function.

```
QByteArray ba;
ba.resize(5);
ba[0] = 0x3c;
ba[1] = 0xb8;
ba[2] = 0x64;
ba[3] = 0x18;
ba[4] = 0xca;
```

To access an array at a specific address, use the at( ) member function.

```
QByteArray ba("aAHeLLo, ");

for (int i = 0; i < ba.size(); ++i)
{
    if (ba.at(i) >= 'a' && ba.at(i) <= 'f')
            qDebug() << "Found character in range [a-f] : " << ba.at(i) ;
}
```

To paste before the currently stored string, you can use the prepend( ) function. To

append after a string, use the append( ) function. To replace a specific character in a string, you can use the replace( ) function.

```
QByteArray x("and");

x.prepend("rock ");             // x == "rock and"
x.append(" roll");              // x == "rock and roll"
x.replace(5, 3, "&");           // x == "rock & roll"
```

To find a specific string among the strings stored in a QByteArray, you can use the following method.

```
QByteArray ba("abc <b>bold</b>, def <b>bold</b>");
qsizetype j = 0;
while ((j = ba.indexOf("<b>", j)) != -1)
{
    qDebug() << "<b> Position : " << j;
    ++j;
}
```

To convert to char *, you can use the data( ) function as follows.

```
QByteArray ba("Hello");

char *data = ba.data();

while (*data) {
     qDebug() << "[" << *data << "]";
     ++data;
}

//[ H ]
//[ e ]
//[ l ]
```

```
//[ l ]
//[ o ]
```

To use the fill( ) member function, you can use it as follows.

```
QByteArray ba("Istambul");

ba.fill('o');
ba.fill('X', 2);

// [Result] ba == "oooooooo",   // ba == "XX"
```

The first( ) function allows you to extract a specified number of characters as shown below.

```
QByteArray x("Pineapple");
QByteArray y = x.first(4);

// y ==  "Pine
```

To decode Base64-formatted data, you can use the following method.

```
QByteArray text = QByteArray::fromBase64("UXQgaXMgZ3JlYXQh");
qDebug() << "String: " << text.data();
```

It can convert data stored in Hex. Provides the ability to convert encoded data entered in URL/URI style.

```
QByteArray text = QByteArray::fromHex("517420697320677265617421");
qDebug() <<  "String : " << text.data();

QByteArray text = QByteArray::fromPercentEncoding("Qt%20is%20great%33");
qDebug() <<  "String : " << text.data();

// String : Qt is great!
// String : Qt is great!
```

Provides the ability to convert and store strings into types such as int, float, long, etc.

```
QByteArray str("FF");
bool ok;

int hex = str.toInt(&ok, 16);
int dec = str.toInt(&ok, 10);

// hex == 255, ok == true
// dec == 0, ok == false
```

- QByteArrayMatcher

This class is provided for finding matching byte array patterns in a byte array.

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

QByteArrayMatcher matcher(y);

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

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

- Qchar

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

This class is provided to support US-ASCII/Latin-1 encoded strings.

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

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

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

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

- QLocale

This class is used to convert to different language character sets.

```
QLocale egyptian(QLocale::Arabic, QLocale::Egypt);
```

```
QString s1 = egyptian.toString(1.571429E+07, 'e');
QString s2 = egyptian.toString(10);
double d = egyptian.toDouble(s1);
int i = egyptian.toInt(s2);
```

- QString

Class QString supports Unicode strings and provides the ability to store 16-bit QChars.

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

QString provides the ability to replace string constants, such as const char *, using the fromUtf8() function.

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

Provides the ability to store a QChar at 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 use the if statement to make the following comparisons

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

If you want to know the position of a specific string you are looking for, you can use the indexOf() function to find the position of the string you are looking for.

```
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;
}
```

QStrings can be used with certain types using the arg( ) function.

```
int value;
QByteArray word;
QString sentence;

value = 14;
word    = QByteArray("Qt");
sentence = QString("Hello World");

QString status = QString("%1, %2, %3").arg(value).arg(word).arg(sentence);
qDebug() << status;
```

You can use the chop( ) function to remove the ₩r₩n as shown below.

```
QString str("LOGOUT₩r₩n");
str.chop(2);
```

- Comparing QStrings and using Regular Expressions

```
int a = QString::compare("aUtO", "AuTo", Qt::CaseInsensitive);   // a == 0
int b = QString::compare("aUtO", "AuTo", Qt::CaseSensitive);       // b > 0
int c = QString::compare("AuTo", "AuTo", Qt::CaseSensitive);       // c == 0
```

For example, to check if the string "Peter Pan" contains "peter", you can use it as follows

```
QString str = "Peter Pan";
bool ret = str.contains("peter", Qt::CaseInsensitive);
```

If you also need to be case sensitive, you can use the following

```
QString str = "Peter Pan";
bool ret = str.contains("peter", Qt::CaseSensitive);
```

To use regular expressions, you can use them as follows

```
QString str = "banana and panama";
str.count(QRegularExpression("a[nm]a"));
```

To check if the end of a string contains a specific string, it can be used like this

```
QString str = "Bananas";
str.endsWith("anas");
str.endsWith("pple");

// anas: true, pple: false
```

To remove whitespace from a QString, you can use the trimmed( ) function.

```
QString str = "  lots₩t of₩nwhitespace₩r₩n ";
str = str.trimmed();
```

- QVariant

You can use an untyped type.

```
int original = 123;
QVariant v(original);

int x = v.toInt();
qDebug() << "x : " << x;

QString xStr = v.toString();
qDebug() << "xStr : " << xStr;
qDebug() << "v type: " << v.typeName();
```

- QPoint

Used to store the X and Y values of an Integer value.

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

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

QPoint can use the rx() and ry() functions as shown below.

```
QPoint p(1, 2);
p.rx()--;    // p becomes (0, 2)

QPoint p(1, 2);
p.ry()++;    // p becomes (1, 3)
```

You can use the following operators

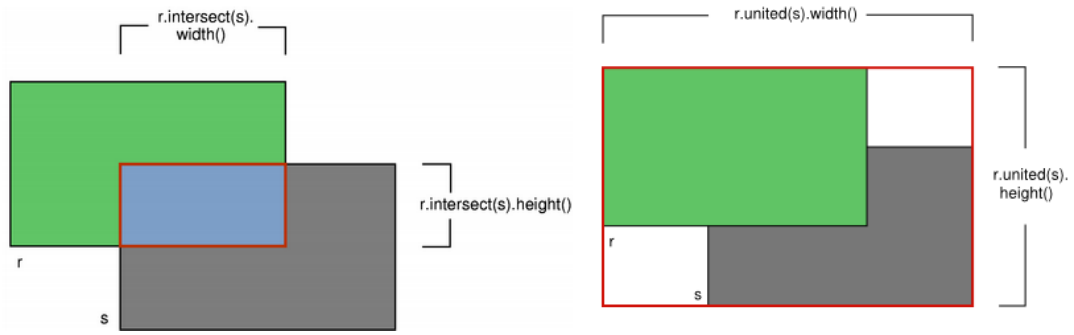
```
QPoint p( 3, 7);
QPoint q(-1, 4);

p -= q;      // p becomes (4, 3)
```

- QRect

QRect can be useful for storing X, Y, WIDTH, and HEIGHT values.

```
QRect r1(100, 200, 11, 16);
QRect r2(QPoint(100, 200), QSize(11, 16));
```

And QRect can use the intersect( ) and united( ) functions to find the intersection and union regions.



9. Useful Template classes Qt provide

The Template classes provided by Qt are easier to use and safer than the Containers provided by STL. They are also lightweight. Therefore, Qt's Template classes are easier to use than C++'s Templates.

- QMap

QMap provides fast key-based searching, with key and value values stored as pairs.

```
QMap<QString, int> map;
```

To add keys and values to a QMap, you can use the following methods.

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

In addition to the above methods, you can also use the insert( ) function to add keys and values.

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

To refer to a value mapped to a specific Key value, you can use the following

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

You can use the contains( ) function to check if a particular Key value exists.

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

- QHash

Class QHash provides a hash table-based Dictionary. The data is stored as a pair of key

and value. It provides the ability to quickly search for the data you want to find by key and value. QHash provides very similar functionality to QMap, but its internal algorithm is faster than QMap.

```
QHash<QString, int> hash;

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

You can use the insert( ) function as a way to store a key and value in a QHash as a pair, and the value( ) member function to get the value of the value.

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

- QPair

The QPair class can store two items as a pair. The QPair class can be used by declaring it as follows, as shown in the example below.

```
QPair<QString, double> pair;

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

- QList

QList<T> provides fast index-based access and is also very fast to delete stored data. QList is an index-based class, which is more convenient to use than the Iterator-based one in QLinkedList.

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

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

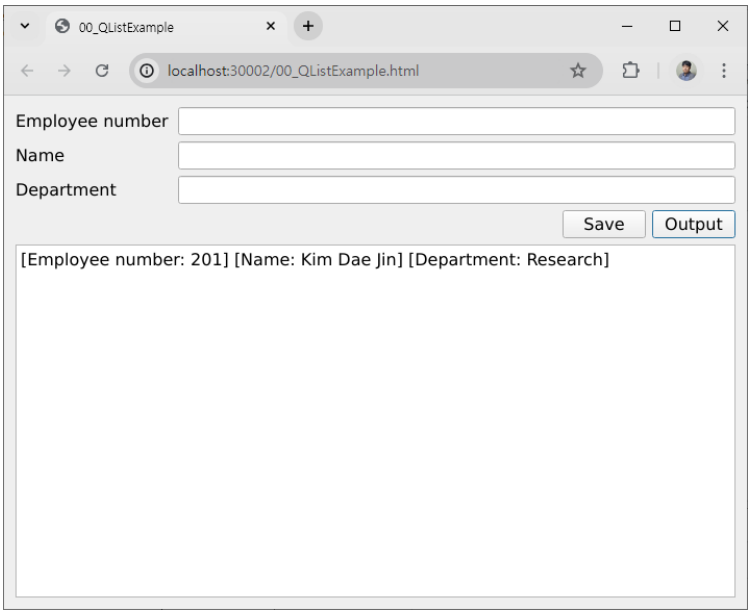
```
if (list[0] == "Bob")
        list[0] = "Robert";
```

QList makes it easy to retrieve a stored location in the list using the at( ) function.

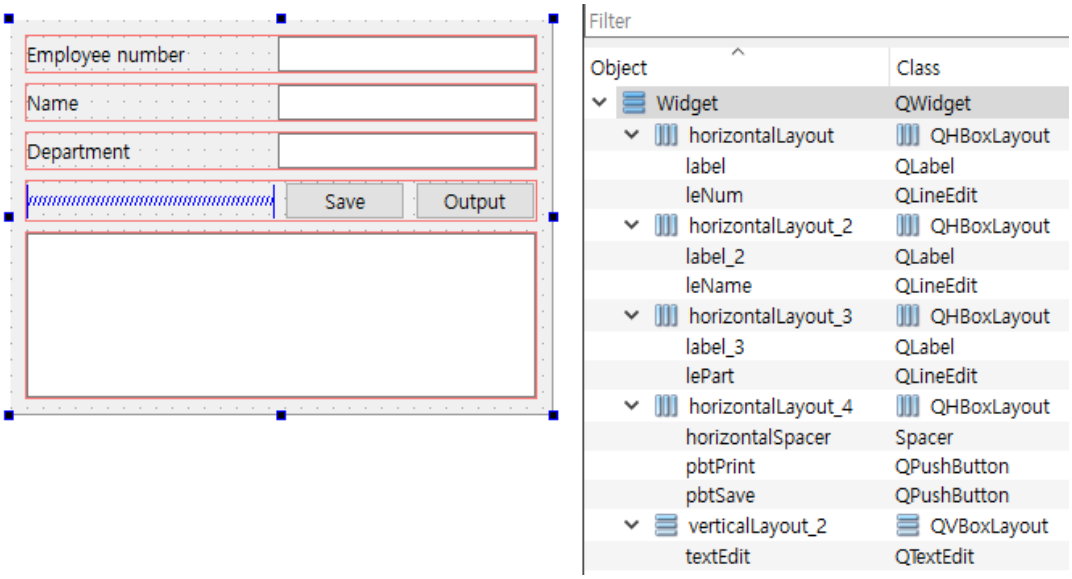
```
for (int i = 0 ; i < list.size() ; ++i) {
      if (list.at(i) == "Jane")
           cout << "Found Jane at position " << i << endl;
}
```

- Example with QList and Structure

This time, let's use QList and Structure to write the following example.



Create a Widget-based project, then double-click the Widget.ui file to place the GUI Widgets as shown below.



Next, open the widget.h header file and write the source code like below.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

QT_BEGIN_NAMESPACE
namespace Ui {
class Widget;
}
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

```
private:
    Ui::Widget *ui;

    typedef struct _tEmployee {
        int num;
        QString name;
        QString part;
    } tEmployee;

    QList<tEmployee> m_employeeList;

private slots:
    void slot_pbtSave();
    void slot_pbtPrint();

};
#endif // WIDGET_H
```

Next, create the widget.cpp source code file as shown below.

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

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

    connect(ui->pbtSave, SIGNAL(pressed()), this, SLOT(slot_pbtSave()));
    connect(ui->pbtPrint, SIGNAL(pressed()), this, SLOT(slot_pbtPrint()));
}

void Widget::slot_pbtSave()
```

```
{
    int num = ui->leNum->text().toInt();
    QString name = ui->leName->text();
    QString part = ui->lePart->text();

    tEmployee employee;
    employee.num = num;
    employee.name = name;
    employee.part = part;

    m_employeeList.append(employee);

    ui->leNum->clear();
    ui->leName->clear();
    ui->lePart->clear();
}

void Widget::slot_pbtPrint()
{
    ui->textEdit->clear();

    for( qsizetype i = 0 ; i < m_employeeList.size() ; i++ )
    {
        int num = m_employeeList.at(i).num;
        QString name = m_employeeList.at(i).name;
        QString part = m_employeeList.at(i).part;

        QString str;
        str = QString("[Employee number: %1] [Name: %2] [Department: %3]")
                .arg(num).arg(name, part);

        ui->textEdit->append(str);
```

```
    }
}

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

10. 2D Graphics 2D Graphics using the QPainter class

In this section, we will learn how to use the QPainter class to render 2D graphical elements on a QWidget.

To render 2D graphic elements on a QWidget, you need to use the paintEvent( ) virtual function.

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
class Widget : public QWidget
{
    Q_OBJECT
public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

protected:
    virtual void paintEvent(QPaintEvent *event);
};
#endif // WIDGET_H
```

Declare the paintEvent( ) function as shown in the example header file above, and use it in your source code file as shown below.

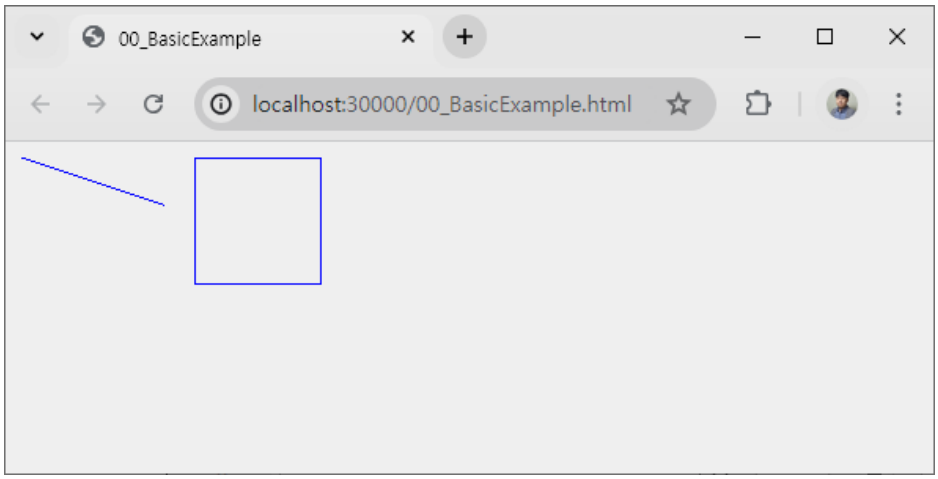
```
#include "widget.h"
#include <QPainter>

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

```
}

void Widget::paintEvent(QPaintEvent *event)
{
    Q_UNUSED(event)
    QPainter painter;
    painter.begin(this);
    painter.setPen(Qt::blue);
    painter.drawLine(10, 10, 100, 40);
    painter.drawRect(120, 10, 80, 80);
    painter.end();
}
...
```

The begin( ) function of the QPainter class means to start drawing. And end( ) means to finish drawing.

The drawLine( ) function draws a line. The first and second arguments are the X and Y coordinates. The third and fourth are the X2, Y2 coordinates.

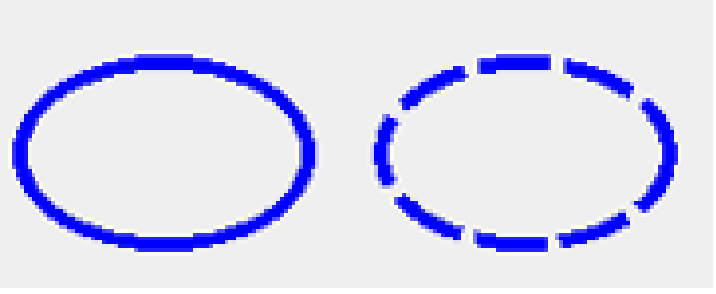
The drawRect( ) function draws a rectangle. The first and second arguments are X, Y, and the third is WIDTH. The last argument is HEIGHT. If you run the above source code, you can see the following image on the web browser.



- Line Style

To define the style of the line in the QPainter region, you can use the setPen( ) function of the QPainter class.

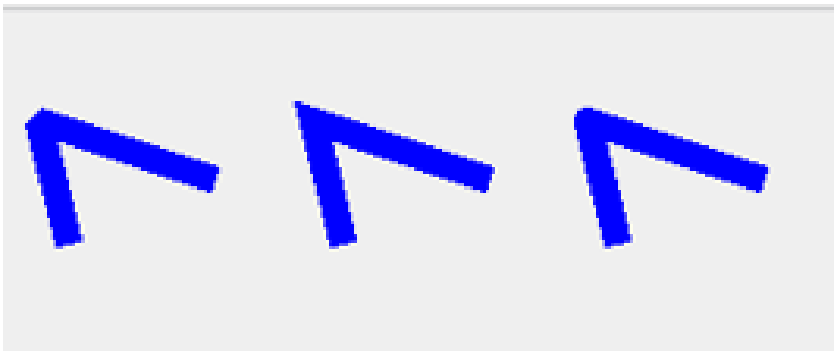
```
...
void Widget::paintEvent(QPaintEvent *event)
{
    QPainter painter;
    painter.begin(this);

    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);
    painter.end();
}
...
```



- Line Join style

To use different styles for the corners when lines are joined, you can use QPen's setJoinStyle( ) function.

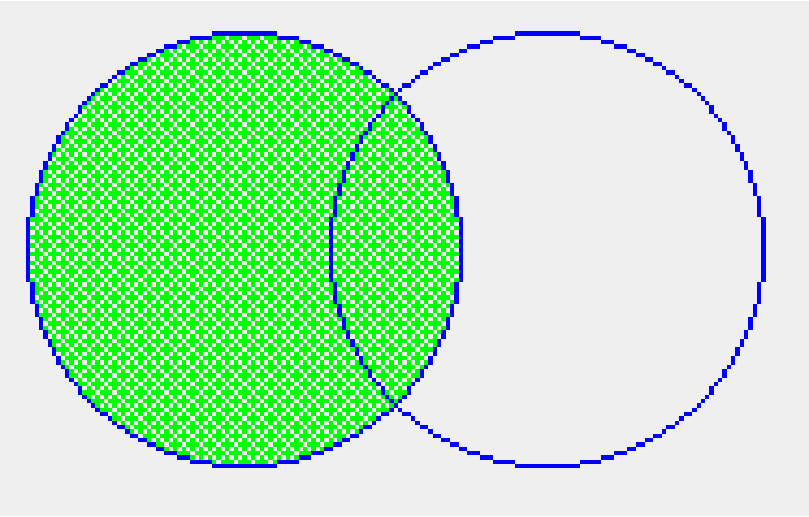
```
void Widget::paintEvent(QPaintEvent *event)
{
    QPainter painter;
    painter.begin(this);
    QPen pen(Qt::blue);
    pen.setWidth(10);
    QPointF p1[3] = {QPointF(30.0, 80.0),QPointF(20.0, 40.0), QPointF(80.0, 60.0) };
    pen.setJoinStyle(Qt::BevelJoin);
    painter.setPen(pen);
    painter.drawPolyline(p1, 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);
    painter.end();
}
```



- QPainter's setBrush( )

QPainter's setBrush( ) function allows you to use a specific Brush inside a circle or rectangle shape.

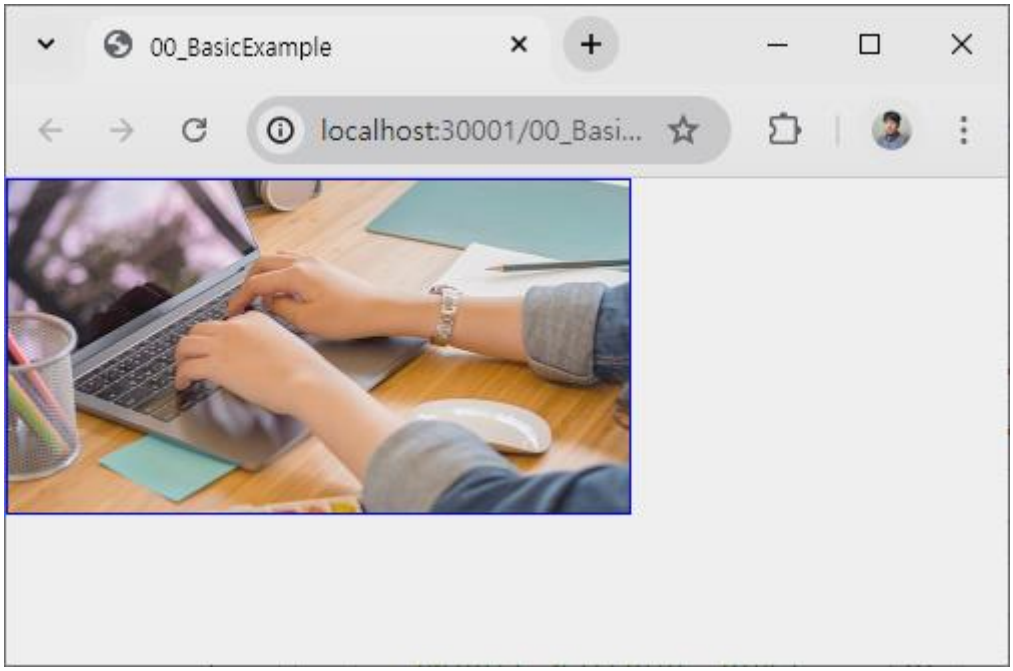
```
void Widget::paintEvent(QPaintEvent *event)
{
    QPainter painter;
    painter.begin(this);

    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);

    painter.end();
}
```



- QPixmap

The QPixmap class can be used to display images on a QWidget. To display images, you can use classes such as QImage.

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

	QPixmap pixmap(":/res/image.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);

	painter.end();
}
```

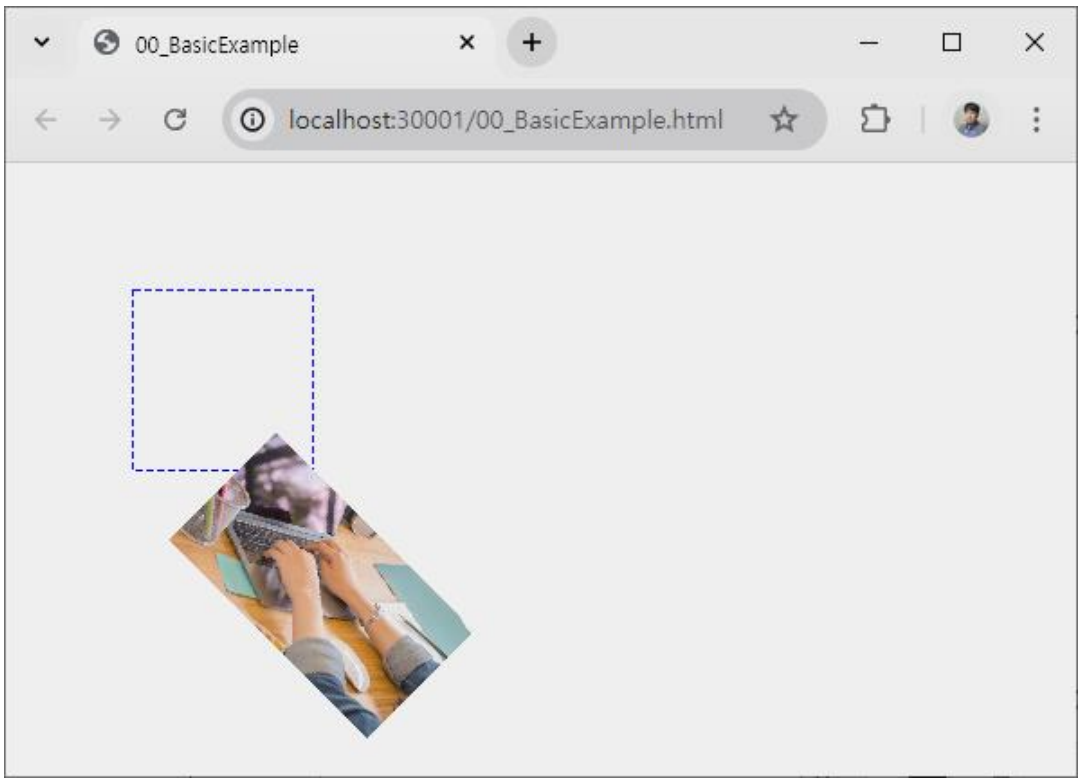


- QTransform

This class allows you to pan, zoom, rotate, and use perspective techniques.

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

    QImage image(":/images/image.png");

    painter.setPen(QPen(Qt::blue, 1, Qt::DashLine));
    painter.drawRect(70, 70, 100, 100);

    QTransform transform;
    transform.translate(150, 150);
    transform.rotate(45);
    transform.scale(0.5, 0.5);
```

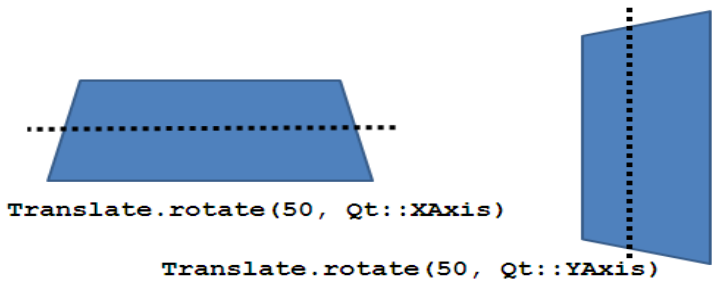
```
    painter.setTransform(transform);
    painter.drawImage(0, 0, image);

    painter.end();
}
```

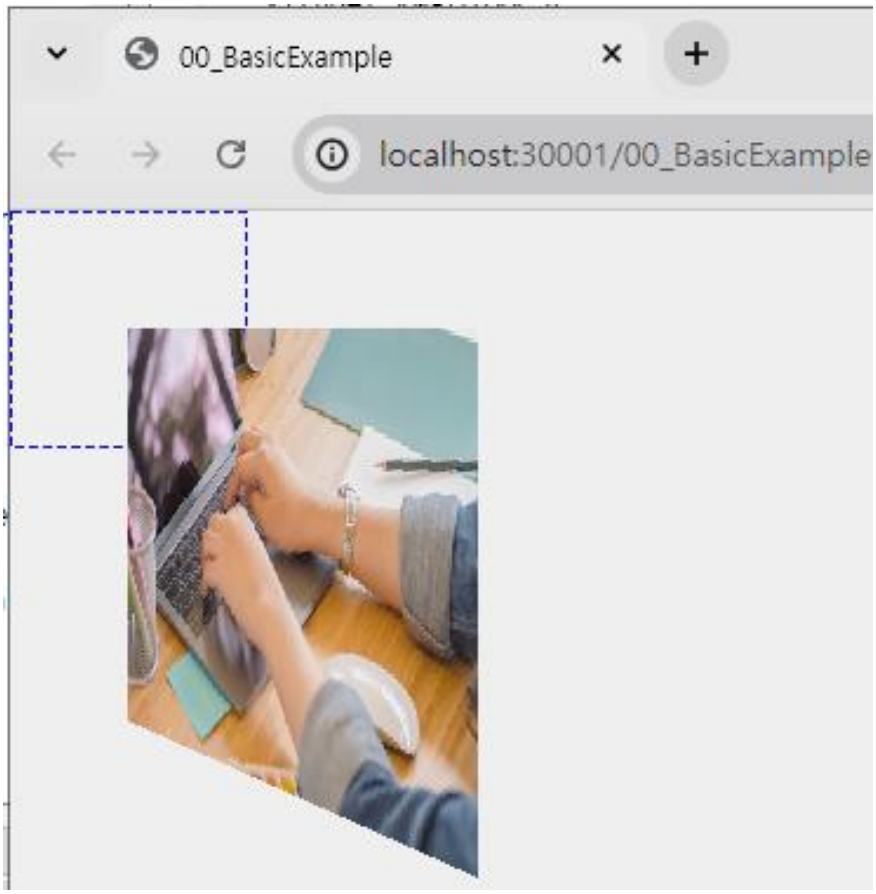


The rotate( ) function of the QTransform class allows you to use the perspective representation of the X-axis, Z-axis, or Z-axis.

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

    QImage image(":/images/image.png");
```

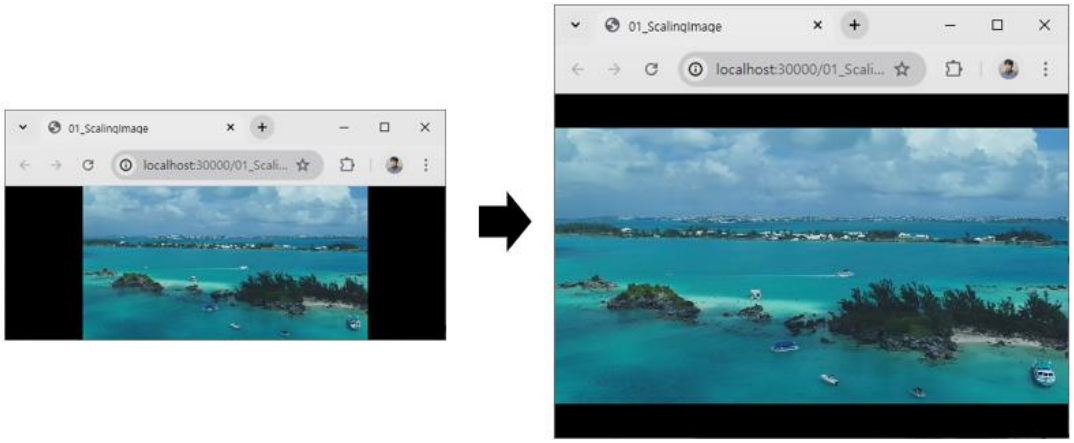
```
    painter.setPen(QPen(Qt::blue, 1, Qt::DashLine));
    painter.drawRect(0, 0, 100, 100);

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

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

    painter.end();
}
```



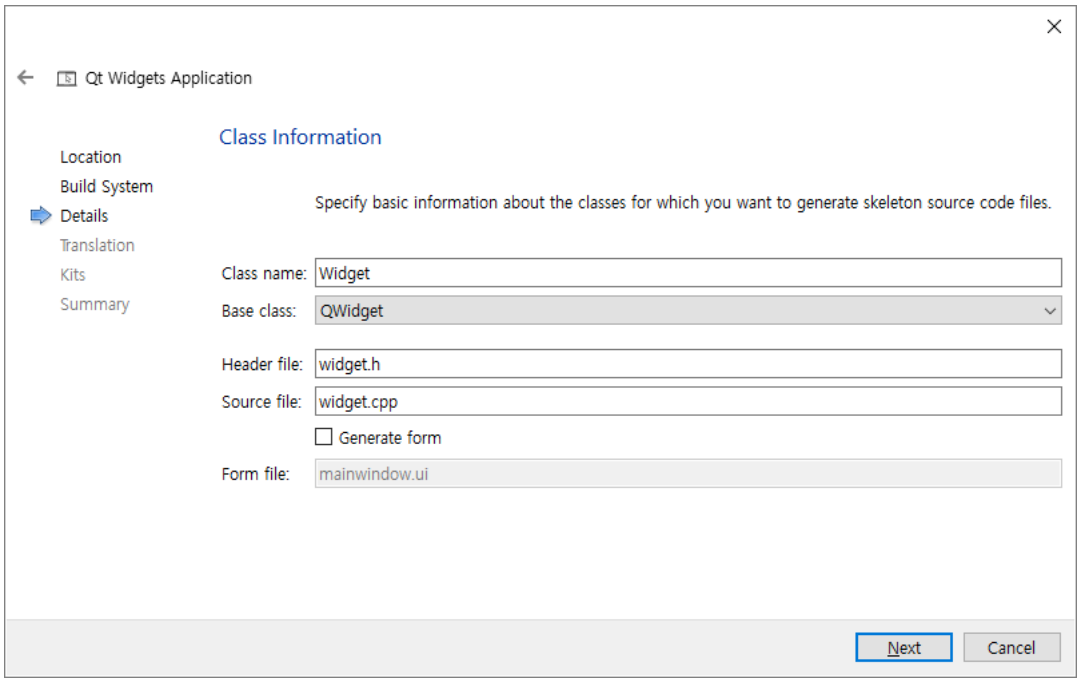


- Image Scaling

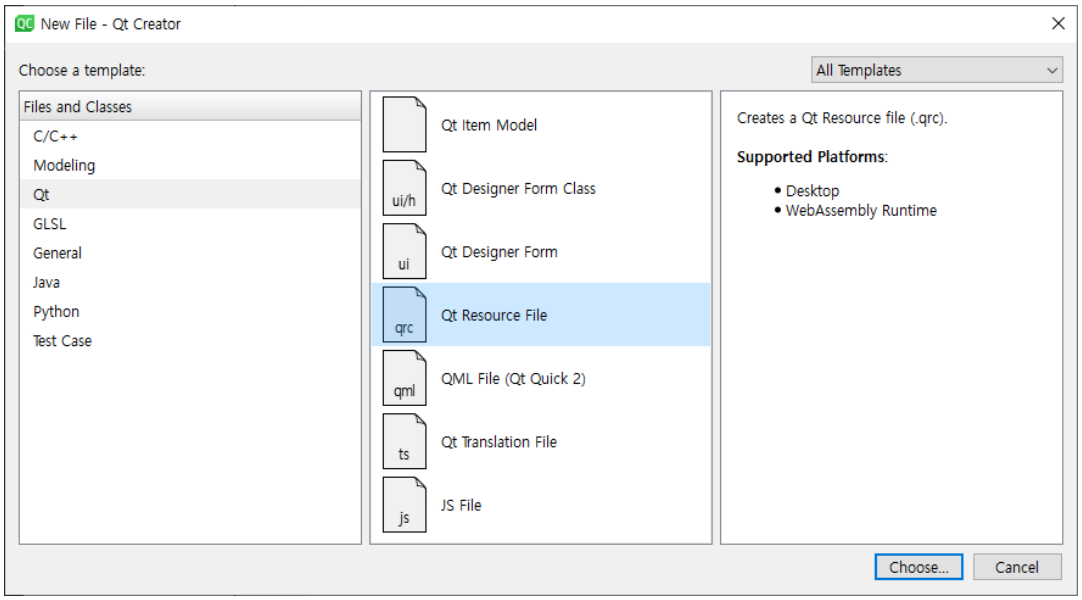
In this example, we'll use the QPainter class to create an example that scales an image based on the size of the web browser.



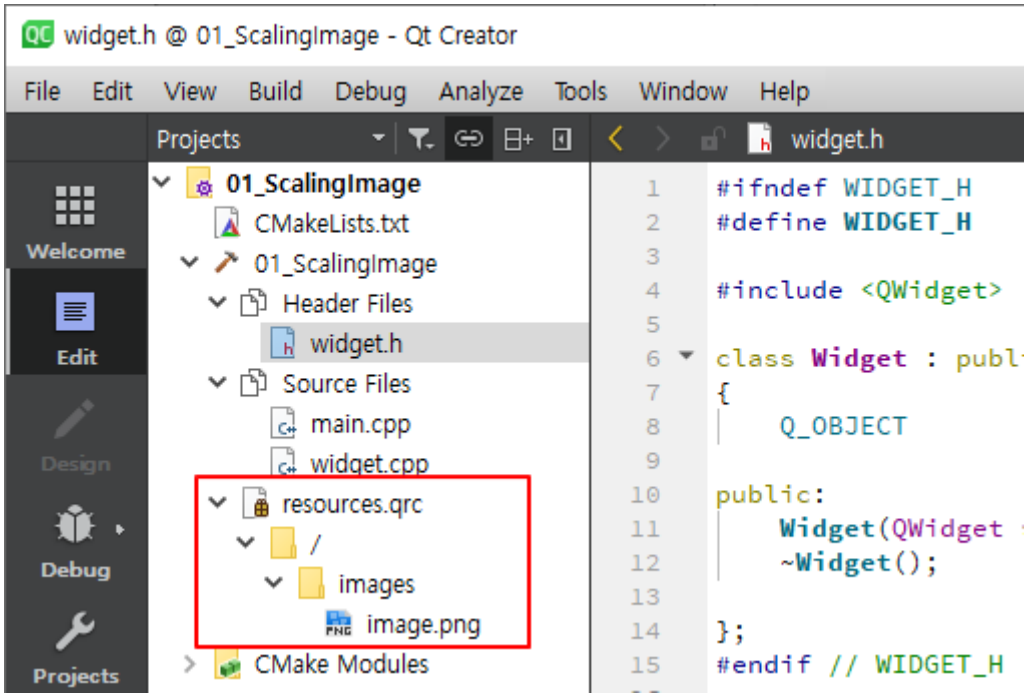
Create a project in the Qt Creator window. In the project creation dialogue, uncheck the [Generation form] checkbox, as shown in the figure below.



Once the project is created, add the RESOURCE file.



Add the Qt Resource File and add the Scaling image. You can use the image from the examples directory, or you can use a different image.



Next, declare the paintEvent( ) function in the widget.h header file as shown below.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

class Widget : public QWidget
{
    Q_OBJECT

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

protected:
    virtual void paintEvent(QPaintEvent *event);

};
```

```
#endif // WIDGET_H
```

Next, open the widget.cpp source file and write the source code as shown below.

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

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

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/image.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();
}

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

As shown in the source code above, in order to scale the image to fit the size of the web browser, we need to get the size of the web browser. After getting the size of the web browser, we can scale the image using the scaled( ) function provided by the QPixmap class.

11. Implement chroma key image processing

In this chapter, we'll implement a simple Chroma key application using QPainter, which we covered in the previous chapter.

Chroma key is a method of filtering out certain colors and replacing them with pixels from another image. For example, you've probably seen many videos where the background of a weather newscast is replaced with another image or video.

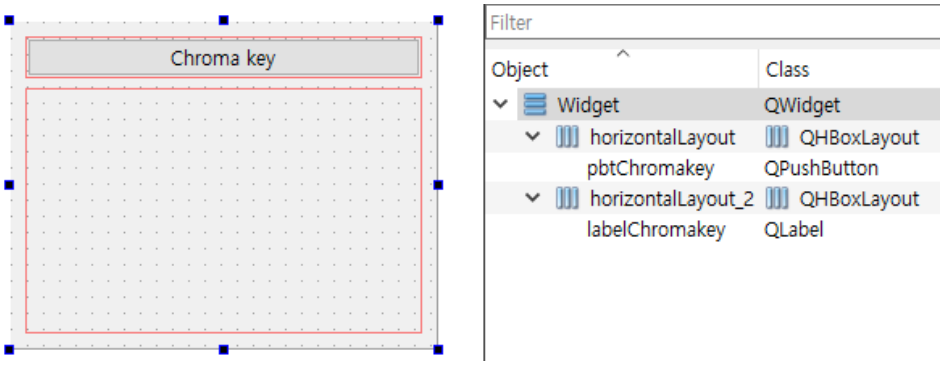
In this example, we're going to replace the background of a photo image format with a the background of a photo image format.

SOURCE IMAGE TARGET IMAGE RESULT IMAGE

In the image above, the background is green in the SOURCE image. We want to replace the background colour in the SOURCE image with the TARGET image.

To do this, we need a way to separate the people in the SOURCE image from the background. The background of the SOURCE image is green, so to separate the people from the background, we need to find the Threshold value of green, which is the background colour of the SOURCE image, to separate the people from the green background.

Then, change the separated background to the TARGET image to get the RESULT image. Create a new project based on QWidget and place the widget on the GUI as follows.



Place the QPushButton and QLabel as shown in the image above. Then, write the following code in the widget.h header file.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

QT_BEGIN_NAMESPACE
namespace Ui {
class Widget;
}
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
    Ui::Widget *ui;

    QImage m_sourceImg;
```

```
    QImage m_targetImg;
    QImage m_resultImg;

    int m_imgWidth;
    int m_imgHeight;
    int m_imgSize;

    void chromakeyProcess(QImage& sourceImage,
                          QImage& targetImage,
                          QImage& resultImage);
private slots:
    void slotChromakey();
};
#endif // WIDGET_H
```

The first argument to the chromakeyProcess( ) function is the SOURCE IMAGE to do the Chroma keying on. The second image is the background image to replace the green color with. The third argument will store the result.

The m_imgWidth is the width of the image, m_imgHeight is the height, and m_imgSize is the size of the image. Note that the imageSize value is not the size of the file, but rather the product of the image's width and height. Each pixel is then multiplied by its RGBA value again.

```
WIDTH * HEIGHT * RGBA(4) = Image Size
```

The image size is in pixels. One of the things to note here is that the size of the SORUCE image, the size of the TARGET image, and the size of the RESULT image must all be the same. Here is the widget.cpp source code file

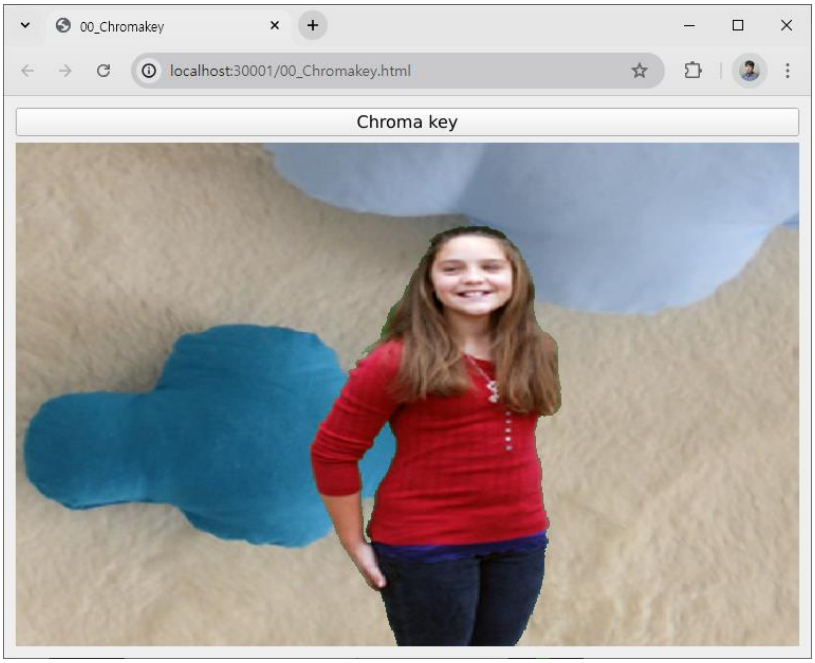
```
#include "widget.h"
#include "./ui_widget.h"

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

```
    ui->setupUi(this);
    setWindowTitle("Chromakey Example");

    connect(ui->pbtChromakey, SIGNAL(clicked()),
            this, SLOT(slotChromakey()));

    m_sourceImg = QImage(":/images/source.png");
    m_targetImg = QImage(":/images/target.png");

    m_imgWidth  = m_sourceImg.width();
    m_imgHeight = m_sourceImg.height();
    m_imgSize   = m_imgWidth * m_imgHeight * 4;

    m_resultImg = QImage(m_imgWidth, m_imgHeight, QImage::Format_RGB32);
}

void Widget::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 < m_imgSize ; 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];
            }
            else
            {
                 pResultData[inc+2] = pSourceData[inc+2];
                 pResultData[inc+1] = pSourceData[inc+1];
                 pResultData[inc] = pSourceData[inc];
            }
       }
}

void Widget::slotChromakey()
{
      chromakeyProcess(m_sourceImg, m_targetImg, m_resultImg);

      int width     = ui->labelChromakey->width();
      int height    = ui->labelChromakey->height();

      QPixmap drawPixmap = QPixmap::fromImage(m_resultImg)
                                        .scaled(width, height);

      ui->labelChromakey->setPixmap(drawPixmap);
```

```
}

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

We used QImage as an argument to the chromakeyProcess( ) member function. The reason we used the QImage class, despite the fact that there are many other classes such as QPixmap, is that it gives us direct access to the RGBA value of each pixel in the image, where R is Red, G is Green, B is Blue, and A is Alpha, which is the transparent value. Let's run the example and click the RESULT button to see the results.



12. Timer

In this chapter, we will implement a Timer using the QTimer class provided by Qt. The QTimer class can call the Slot function repeatedly based on a specified time.

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

timer->start(1000);
```

As shown in the example source code above, we declare an object of class QTimer, and we need to connect the Signal and Slot functions using the connect( ) function in order to call it repeatedly at the specified time.

In the connect( ) function, the first argument specifies a QTimer object. The second specifies a Signal. The timeout( ) Signal will call the update( ) Slot function specified in the fourth argument after the specified time has elapsed.

The start( ) member function of class QTimer in the last line calls the Slot function specified by the connect( ) function repeatedly after the time specified by the first argument has elapsed. The first argument of this function has a unit of milliseconds. The QTimer class can stop the timer by using the stop( ) member function. If you want the timer to be called only once, rather than repeatedly, you can use the singleShot( ) member function. The singleShot( ) member function can be used as shown in the following example source code.

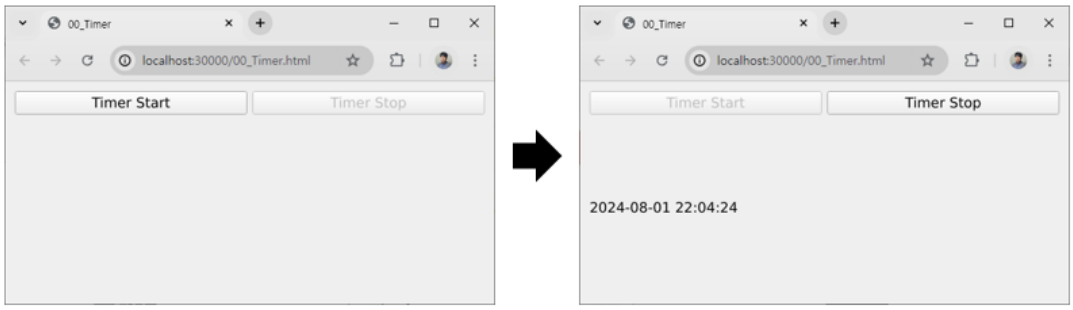
```
QTimer::singleShot(200, this, SLOT(updateCaption()));
```

The first argument to the singleShot( ) member function is the elapsed time, in milliseconds. The third argument specifies the Slot function to be called after the first argument's time has elapsed.

- Examples using the QTimer class

In the following example, the Slot function is called at one second intervals. As shown in the following figure, clicking the [Timer Start] button starts the timer, and clicking the

[Timer Stop] button stops the timer.



In the QTimer example run screen, the current time is an example of getting the current time from the system and outputting it to the QLabel widget at intervals of 1000 milliseconds (1 second) specified by QTimer. The following example source code is the widget.h source code.

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QTimer>

QT_BEGIN_NAMESPACE
namespace Ui {
class Widget;
}
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

private:
```

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

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

#endif // WIDGET_H
```

startPressed( ) is a Slot function that is called when the [Timer Start] button is clicked. The stopPressed( ) function is a Slot function that is called when the [Timer Stop] button is clicked. The elapsedTime( ) function is called when the time specified by the QTimer has elapsed. The following example shows the widget.cpp source code.

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

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

    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);
}

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("yyyy-MM-dd hh:mm:ss");

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

Widget::~Widget()
{
```

```
    delete ui;
}
```

13. Thread Programming

Qt provides the QThread class to support Threads. When using Threads with the QThread class, synchronisation is sometimes required in certain situations. Synchronisation is necessary because variables referenced by multiple users in a threaded environment can refer to incorrect values for other users when one user changes the value of the variable. Therefore, synchronisation is provided to prevent other users from referencing or changing a variable from the start to the end of the source code of a particular function that runs as a thread, thereby preventing them from referencing the incorrect value of the variable before it was changed. Qt provides the QMutex class to support synchronisation. The following example source code is part of a class that inherits from and implements the QThread class.

```
#include <QThread>
#include <QMutex>
#include <QDateTime>

class MyThread : public QThread
{
    Q_OBJECT
    void run() override {
        while(!m_threadStop)
        {
            m_mutex.lock();
            ...
                m_mutex.unlock();
            ...
                sleep(1);
        }
    }
public:
```

```
    MyThread(int n);
private:
    bool m_threadStop;
    QMutex m_mutex;
};
...
```

The example above is an implementation of the MyThread class using QThread, as shown in the source code. The run( ) function is a virtual member function inherited from QThread, and you can implement the functions you want to run in the thread in this function.

Because the run( ) function is an internally implemented function, calling the start( ) function provided by QThread from outside the class will automatically call the run( ) function.

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

Sometimes you may want to declare the run( ) function as Public and call the run( ) function externally. In this case, the run( ) function does not behave as it would in a Thread.

This means that it behaves like a normal function, not a thread, so you should be careful not to call the run( ) function directly. The QMutex class used by the run( ) function is for synchronisation.

As shown in the example source code above, the lock( ) member function provided by the QMutex class declares that synchronisation has begun, and the unlock( ) member function declares that this is the last leg of synchronisation, and variables declared within this leg are inaccessible until the external class calls the unlock( ) member function.

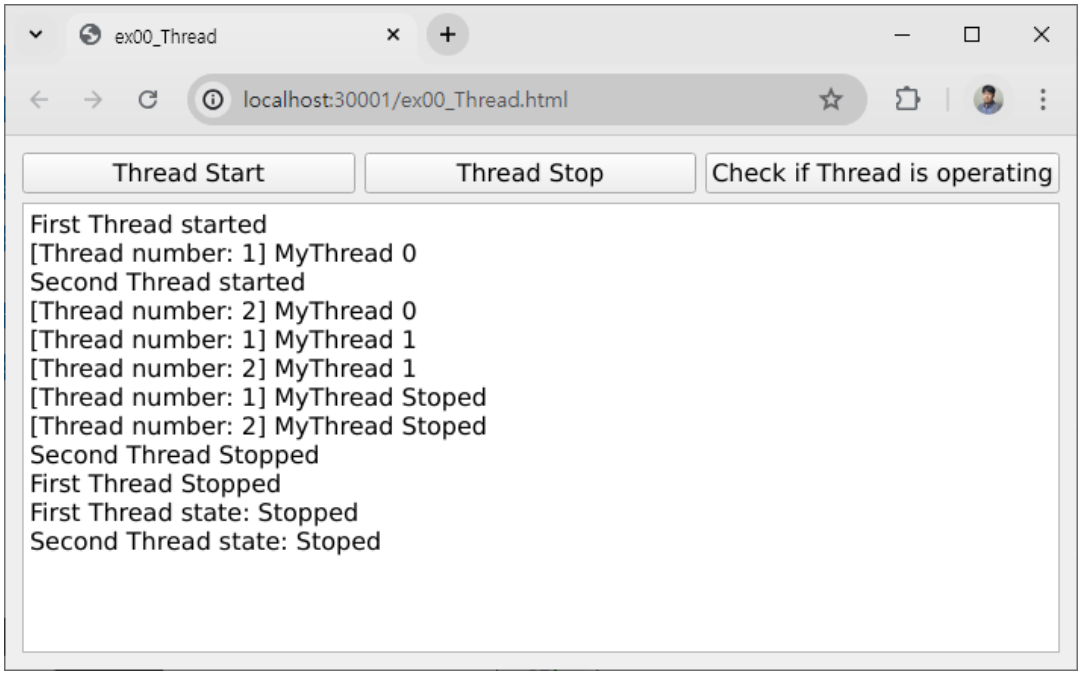
For example, suppose a network application has 10 current connections and stores the value of 10 in an int public variable called users. If the number of current connections increases to 11 because of a new connection, the value of the users variable is incremented by one, and if there is a process to check the current connections at the same time, the wrong current connection information could be given to other connections.

So, in this case, if you use synchronisation to increment the value of the users variable by 1 between lock( ) and unlock( ) in a section of source code that increments the value of the users variable, the users variable is treated as waiting in a queue to prevent it from being changed or referenced.

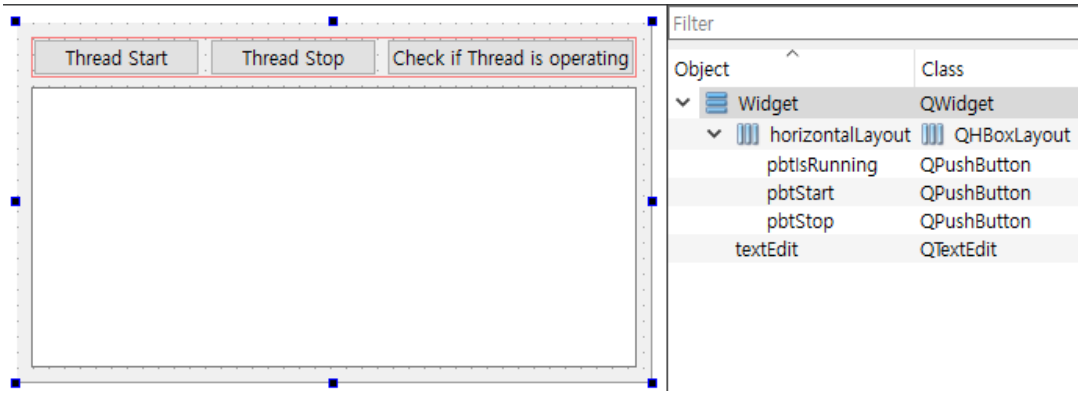
The QMutex class is useful if you need to synchronise, but if you use it too often it can block and cause the program to freeze, so it is recommended that you only use it when necessary.

- A simple example using the QThread class

In this example, when you click the [Start Thread] button on the web browser, two threads are started and a message is displayed as shown in the figure below.



Click [Thread Stop] button to stop the two threads. Click the [Check if Thread is operating] button to display a message to check if the two threads are operating. Create a project based on Qt Widget and place the widgets in widget.ui as shown in the figure below.



Next, add a class to your project. Name the class MyThread. Once the class is created, create the MyThread.h header file as shown below.

```
#ifndef MYTHREAD_H
#define MYTHREAD_H

#include <QThread>
#include <QMutex>

class MyThread : public QThread
{
    Q_OBJECT
public:
    MyThread(int num);
    QString threadMsg();

private:
    bool    m_threadStop;
    int     m_number;
    QMutex mutex;

public:
    void stop();

signals:
```

```
    void sig_threadMsg(QString);

protected:
    virtual void run();
};


#endif // MYTHREAD_H
```

Next, in your MyThread.cpp source code file, write the following

```
#include "MyThread.h"

MyThread::MyThread(int num)
{
    m_number = num;
}

void MyThread::stop()
{
    m_threadStop = true;

    QString str = QString("[Thread number: %1] MyThread Stoped")
                                    .arg(m_number);
    emit sig_threadMsg(str);
}

void MyThread::run()
{
    m_threadStop = false;
    int i = 0;

    while(!m_threadStop)
    {
```

```
            mutex.lock();

            QString str = QString("[Thread number: %1] MyThread %2")
                                        .arg(m_number).arg(i);
            i++;
            emit sig_threadMsg(str);

            mutex.unlock();

            sleep(1);
        }
}
```

Next, add the following source code to your widget.h header file.

```
#ifndef WIDGET_H
#define WIDGET_H

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

QT_BEGIN_NAMESPACE
namespace Ui {
class Widget;
}
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

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

```
private:
    Ui::Widget *ui;

    MyThread *m_thread1;
    MyThread *m_thread2;

private slots:
    void slot_pbtStart();
    void slot_pbtStop();
    void slot_isRunning();

    void slot_threadMsg(QString);

    void slot_thread1_started();
    void slot_thread1_finished();
    void slot_thread2_started();
    void slot_thread2_finished();
};
#endif // WIDGET_H
```

Next, in your widget.cpp source code file, write the following

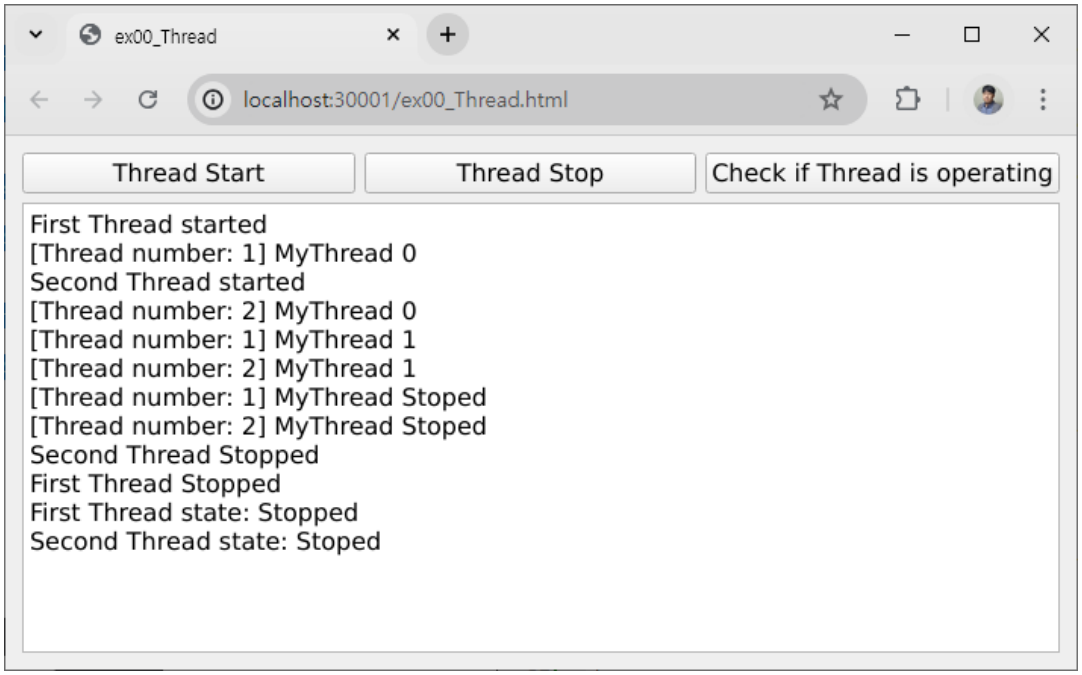
```
#include "widget.h"
#include "./ui_widget.h"
#include <QFontDatabase>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    QFontDatabase::addApplicationFont(":/NanumGothic.ttf");

    ui->setupUi(this);
```

```
    connect(ui->pbtStart, SIGNAL(pressed()),
            this, SLOT(slot_pbtStart()));
    connect(ui->pbtStop, SIGNAL(pressed()),
            this, SLOT(slot_pbtStop()));
    connect(ui->pbtIsRunning, SIGNAL(pressed()),
            this, SLOT(slot_isRunning()));


    m_thread1 = new MyThread(1);
    m_thread2 = new MyThread(2);

    connect(m_thread1, SIGNAL(sig_threadMsg(QString)),
            this, SLOT(slot_threadMsg(QString)));
    connect(m_thread2, SIGNAL(sig_threadMsg(QString)),
            this, SLOT(slot_threadMsg(QString)));

    connect(m_thread1, SIGNAL(started()),
            this, SLOT(slot_thread1_started()));
    connect(m_thread1, SIGNAL(finished()),
            this, SLOT(slot_thread1_finished()));

    connect(m_thread2, SIGNAL(started()),
            this, SLOT(slot_thread2_started()));
    connect(m_thread2, SIGNAL(finished()),
            this, SLOT(slot_thread2_finished()));
}

void Widget::slot_pbtStart()
{
    ui->textEdit->clear();

    m_thread1->start();
```

```
    m_thread2->start();
}

void Widget::slot_pbtStop()
{
    m_thread1->stop();
    m_thread2->stop();
}

void Widget::slot_isRunning()
{
    if(m_thread1->isRunning())
        ui->textEdit->append("First Thread state: Running");
    else
        ui->textEdit->append("First Thread state: Stopped");

    if(m_thread2->isRunning())
        ui->textEdit->append("Second Thread state: Running");
    else
        ui->textEdit->append("Second Thread state: Stoped");
}

void Widget::slot_threadMsg(QString msg)
{
    ui->textEdit->append(msg);
}

void Widget::slot_thread1_started()
{
    ui->textEdit->append("First Thread started");
}
```

```
void Widget::slot_thread1_finished()
{
	ui->textEdit->append("First Thread Stopped");
}

void Widget::slot_thread2_started()
{
	ui->textEdit->append("Second Thread started");
}

void Widget::slot_thread2_finished()
{
	ui->textEdit->append("Second Thread Stopped");
}

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

If you write and build it like above, and then run it, you can see it running like below in your web browser.



- Example Threads that satisfy Reentrancy and Thread-Safety

Reentrancy means that when two or more threads are running, regardless of the order in which the threads are executed, after one thread is executed, the next thread is executed as if it had been executed.

Thread-Safety refers to the use of mechanisms such as mutexes to ensure reliability when accessing shared memory regions, such as Static or Heap memory regions, in situations where two or more Threads are operating. The following source code is an example of creating two Threads and implementing a Thread without considering Reentrancy and Thread-Safety.

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

class Producer : public QThread
{
    Q_OBJECT
    void run() override {
```

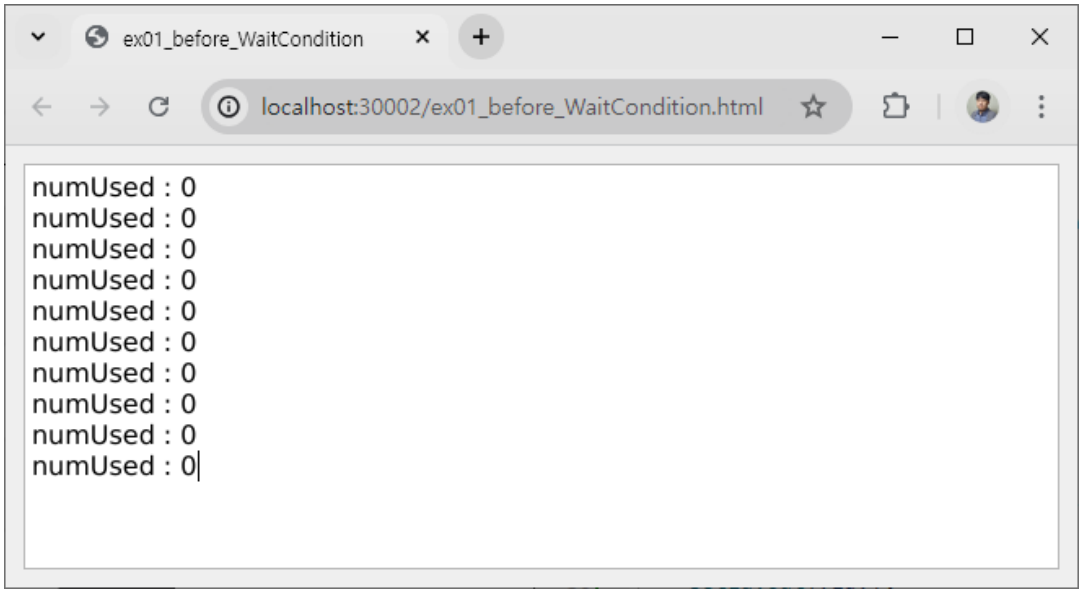
```
            for(int i = 0 ; i < 10 ; i++) {
                sleep(1);
                ++numUsed;
            }
        }

public:
        Producer() {}
};

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

Implement the Producer and Consumer classes as shown in the example source code above, and write the following in main.cpp

```
#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();
}
```

In the example source code above, we declare two implemented Thread class objects and call the start( ) member function provided by QThread to start a Thread, as shown in the following figure.



As shown in the example run screen, the run( ) function of the Consumer class will be executed first, followed by the run( ) function of the Producer class. The following example shows an implementation of a Thread 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);
            mutex.unlock();
            qDebug("[Consumer] numUsed : %d", numUsed);
        }
    }
public:
```

```
    Consumer() { }
};


#endif // MYTHREAD_H
```

As shown in the example run screen, the run( ) function of the Consumer class will be executed first, followed by the run( ) function of the Producer class. The following example shows an implementation of a Thread that satisfies Reentrancy and Thread-Safety. Let's modify the Consumer and Producer classes implemented in the previous example as follows.

The Producer and Consumer threads are activated when the value of numUsed changes, i.e., the Consumer is waiting by the wait( ) member function of the QWaitCondition class. The Producer class increments the value of numUsed by 1 and wakes up the Consumer class by using the wakeAll( ) member function of the QWaitCondition class.

In this way, it is easy to implement threads that satisfy Reentrancy and Thread-Safety on more than one thread.