

# QML Programming



Second Edition, Ver 2.0



Qt 개발자 커뮤니티

[www.qt-dev.com](http://www.qt-dev.com)

Qt Framework는 QML 언어로 GUI를 구현하고 기능은 C++ 언어로 구현해  
소스코드 재 사용성을 높일 수 있습니다.

예수님은 당신을 사랑합니다.

## Qt 6 QML 프로그래밍

---

**Release version** 2.0 (2024.03.01)

**Homepage URL** [www.qt-dev.com](http://www.qt-dev.com)

---

예수님은 당신을 사랑합니다.

## 머리말

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

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

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

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

[새번역성경] 마태복음 22:37~39

## Table of Contents

---

1. What is Qt Quick and QML.....	1
2. QML Basic .....	4
2.1. QML 기초 문법 .....	5
2.2. Types .....	15
2.3. Event.....	32
2.4. Loader type 을 이용한 Dynamic UI 구성 .....	47
2.5. Canvas .....	60
2.6. Graphics Effects.....	66
2.7. Module programming.....	87
2.8. QML에서 JavaScript 사용 .....	95
2.9. Dialog .....	106
2.10. Layout .....	112
2.11. Type Positioning .....	119
2.12. Qt Quick Controls .....	125
3. Animation Framework.....	143
3.1. Animation .....	145
3.2. Animation 과 Easing curve 를 이용한 예제 구현 .....	158
3.3. State and Transition .....	167
3.4. Image Viewer 구현.....	175
4. Model and View .....	187
4.1. Model/View 를 이용한 데이터 표현 .....	188
4.2. Chess Knight .....	200

예수님은 당신을 사랑합니다.

<b>5. Integration QML and C++.....</b>	<b>206</b>
<b>5.1. Overview .....</b>	<b>207</b>
<b>5.2. C++로 QML 타입 구현.....</b>	<b>220</b>
<b>5.3. QML에서 QQuickPaintedItem 클래스 사용 .....</b>	<b>232</b>
<b>5.4. Scene Graph .....</b>	<b>236</b>
<b>5.5. C++과 QML 간의 Interaction 과 변수 매핑 .....</b>	<b>246</b>
<b>5.6. TCP 프로토콜 기반 채팅 어플리케이션 구현.....</b>	<b>261</b>

예수님은 당신을 사랑합니다.

# QML Programming



Second Edition, Ver 2.0



Qt 개발자 커뮤니티

[www.qt-dev.com](http://www.qt-dev.com)

Qt Framework는 QML 언어로 GUI를 구현하고 기능은 C++ 언어로 구현해  
소스코드 재 사용성을 높일 수 있습니다.

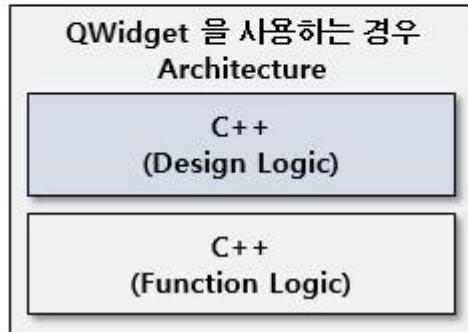
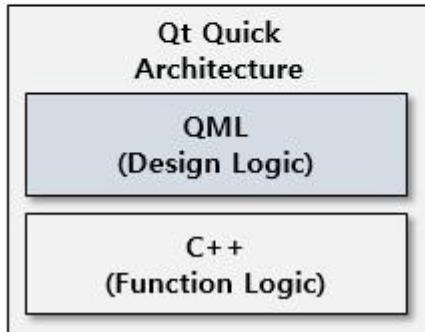
## 1. What is Qt Quick and QML

Qt Quick 은 현대적인 GUI인터페이스를 쉽게 설계 및 구현하기 위한 목적으로 사용된다. Qt Quick을 이용해 GUI를 구현할 때 C++을 사용하지 않는다. Qt Quick 은 QML이라는 인터프리터 언어를 사용한다. QML은 Qt Modeling Language 의 약자이다.

항상 Qt Quick 을 이용해 GUI를 구현해야 하는 것은 아니다. Qt를 이용해 응용 어플리케이션 구현 시 C++ 을 이용 할 수 도 있고 QML 을 사용할 수 있다. 어떤 것을 사용하든지 장단점은 존재한다.

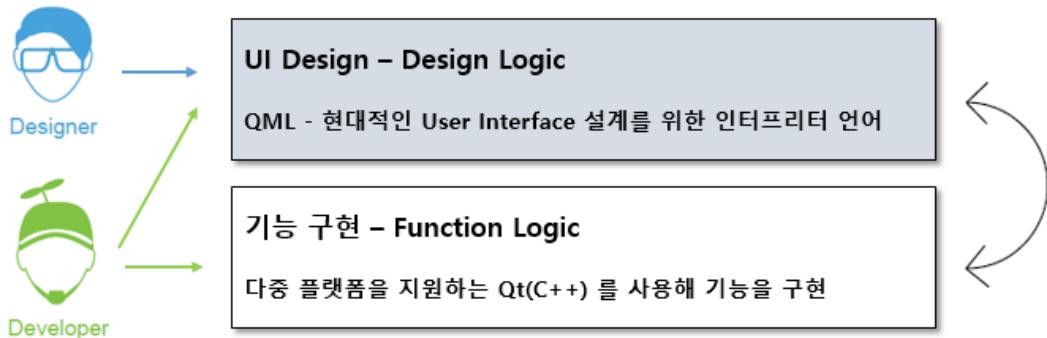
Qt Quick 을 사용할 경우 장점으로 디자인로직과 기능로직으로 분리할 수 있다. 예를 들어 디자인로직은 GUI를 뜻한다. 기능로직은 기능을 뜻한다. 디자인로직인 GUI상에서 어떤 버튼을 클릭 했을 때 기능이 실행된다고 가정해 보자. 화면상에서 버튼은 디자인로직이며 버튼을 클릭했을 때 어떤 기능이 동작하는 것을 기능로직 이라고 할 수 있다.

이와 같이 Qt Quick 에서 제공하는 QML을 이용해 디자인로직을 구현하면 C++로 구현된 기능로직과 완전히 분리됨으로써 디자인로직인 GUI가 변경되어도 기능 구현 모듈인 기능로직은 재사용 할 수 있다는 장점이 있다.



그리고 Qt Quick 을 사용하는 경우 또다른 장점으로 사용자 인터페이스가 제한된 임베디드 디바이스와 같은 환경에서 사용하는 것이 적합하다. 예를 들어 Desktop PC는 사

용자의 입력이 키보드, 마우스 등 다양한 사용자 인터페이스를 사용할 수 있다. 하지만 스마트폰 또는 터치만 사용 가능한 산업용 임베디드 컴퓨터 등과 같이 사용자 인터페이스가 제한된 상황에서는 Qt Quick 을 사용하는 것이 적합하다.



반대로 개발하려는 어플리케이션이 동작하는 컴퓨터 또는 Embedded 장치가 CPU, 메모리 등과 같은 하드웨어 리소스가 낮거나 매우 제한적인 환경이라면 C++을 사용하는 것이 적합할 수 있다. 예를 들어 제한된 환경에서 빠르게 반응해야 한다면 GUI를 C++을 이용해 구현하는 것이 바람직하다.

즉 Qt Quick 에서 제공하는 QML 을 사용해야만 꼭 좋은 응용 어플리케이션 개발하는 것은 아니다. 개발하려는 응용 어플리케이션이 데스크탑과 같은 하드웨어 사양이 높은 컴퓨팅 환경에서 동작하는지, 그렇지 않은지 또는 여러가지 사용자 입력 인터페이스는 무엇인지 고려 후 Qt Quick 을 사용할지 결정하기 바란다. Qt Quick 을 요약하면 다음과 같은 사항으로 요약할 수 있다.

- ① Qt Quick 은 현대적인 GUI 인터페이스를 쉽게 구현하기 위해 사용한다.
- ② Qt Quick 은 GUI를 구현하는데 QML 이라는 인터프리터 언어를 사용한다.
- ③ QML 은 절차적 언어(Declarative 또는 인터프리터 언어) 이다.

Qt Quick 에서 사용하는 QML은 인터프리터 언어이다. C++처럼 컴파일 되지 않고 순차적인 해석을 하는 인터프리터 언어이므로 C++과 비교 했을 때 성능 측면에서는 C++로 GUI를 구현하는 것보다는 성능상 다소 느린다. 예를 들어 JAVA가 동작할 때 JAVA Virtual Machine 과 같은 소프트웨어 Stack 을 사용하는 것과 같이 Qt Quick 도 별도의 Qt Declarative 라는 Stack 을 사용한다는 특징이 있다. QML과 비슷한 인터프리터 언어로 HTML을 예로 들 수 있다. 웹 브라우저에서 Parsing 되는 인터프리터 언어와 같이 QML도 이와 동일한 인터프리터 언어이다. 지금까지 Qt Quick과 QML에 대해서 알아보

예수님은 당신을 사랑합니다.

았다.

Qt Quick 을 요약 해 설명 하면 장점으로 Qt Quick 을 사용해 응용 어플리케이션을 개발하면 디자인로직과 기능로직이 분리 되기 때문에 Qt Quick 으로 개발한 프로젝트에서 디자인로직을 제외한 기능로직을 활용할 경우 재사용 성을 높일 수 있는 장점이 있다. 단점으로 Qt Quick 으로 디자인로직을 C++ 이 아닌 QML 이라는 인터프리터 언어를 사용하기 때문에 C++과 비교해 느린 편이다. 하지만 충분한 하드웨어 리소스가 제공된다면 C++에 비해 Qt Quick 이 느리다는 단점은 극복할 수 있다.

지금까지 Qt Quick 의 특징과 장점이 무엇인지 알아보았다. Qt Quick 을 사용한다고 해서 좋은 것이 아니다. 오히려 사용하지 않는 것이 좋을 때가 있다. Qt Quick 을 사용하기 전 개발하고자 하는 소프트웨어가 위에서 언급한 Qt Quick 의 특징 및 장점에 부합하는지 검토 후 사용하는 것을 권장한다.

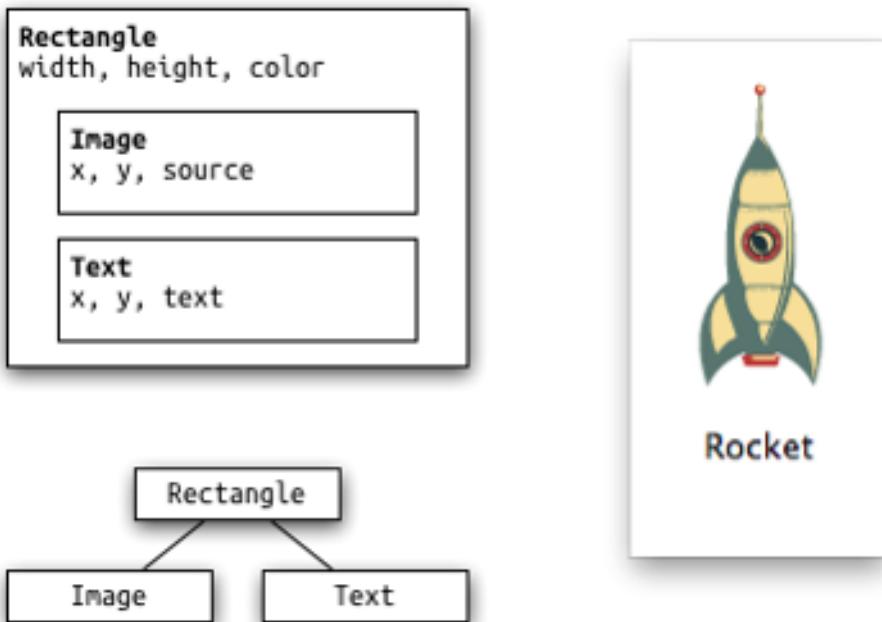
## 2. QML Basic

이번 장에서는 Qt Quick에서 사용하는 QML의 문법, 데이터 타입, QML Element, 이벤트 처리, Components 등 QML을 이용해 GUI를 구현하는데 있어서 꼭 필요한 내용을 살펴볼 것이다. 이번 장에서 다룰 내용은 다음 표에서 보는 것과 같다.

제목	내용 및 설명
QML 기초 문법	QML에서 사용하는 문법과 구조에 대한 설명
Types	예를 들어 C++에서 QWidget을 이용해 GUI를 구현하는 것과 같이 QML 타입을 제공
Event	사용자 입력 이벤트 처리
Loader 타입을 이용한 동적 GUI 구성	QML에서 또 다른 QML 파일 또는 타입을 동적으로 호출하기 위한 기능 제공
Canvas	QWidget 클래스에서 사용했던 QPainter 클래스와 같이 QML 영역 내에서 Drawing을 하기 위한 기능을 제공
Graphics Effects	Blending, Masking, Blurring 등과 같은 Effect를 사용
QML Module Programming	QML에서 자주 사용하는 사용자 정의 타입을 라이브리처럼 재 사용할 수 있도록 모듈화 기능을 제공
QML에서 JavaScript 사용	QML 내에서 JavaScript 사용
Dialog	단순한ダイアル로그, 컬러ダイアル로그, 파일ダイアル로그, 폰트ダイ얼로그 등 다양한ダイ얼로그를 제공
Layout	Column Layout, GridLayout 등 QML에서 제공하는 Layout.
Type Positioning	개의 타입 중에서 특정 위치의 타입의 정보를 얻기 위한 기능 제공

## 2.1. QML 기초 문법

QML은 HTML 구조와 흡사하다. 그리고 QML은 기본적 구조는 상하 계층 구조로 되어 있다. 예를 들어 어떤 사각형 영역 안에 이미지를 출력한다고 가정해보자.



위의 그림에서 우측의 로켓 이미지를 표시하고 아래에 “Rocket”이라는 텍스트 어떤 사각형 영역에 표시한다고 한다면 위의 그림에서 보는 것과 같이 Rectangle(사각형) 을 먼저 선언하고 Image와 Text를 표시할 수 있다. 이와 같이 이미지와 텍스트를 표시하는 구조가 상하 계층구조이다. 즉 QML은 C/C++ 등과 같은 문법과 다르게 HTML과 같이 상하 계층(Hierarchy Element) 구조로 되어 있다. 다음 소스코드는 QML 예제 소스코드이다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    id: root
    width: 120
    height: 240
    color: "#D8D8D8"
```

```
Image {  
    id: rocket  
    x: (parent.width - width) / 2  
    y: 40  
    source: 'assets/rocket.png'  
}  
  
Text {  
    y: rocket.y + rocket.height + 20  
    width: root.width  
    horizontalAlignment: Text.AlignHCenter  
    text: 'Rocket'  
}  
}
```

위의 QML 예제에서 가장 상단의 import 문은 C++ 문법에서 "#include<...>" 문과 유사하다. 다음은 import 문의 Syntax 구조이다. <ModuleIdentifier>은 모듈 명을 명시하며 <Version.Number> 는 버전을 표시한다.

```
import <ModuleIdentifier> <Version.Number> [as <Qualifier>]
```

<Qualifier> 는 모듈 이름을 대신할 가명 또는 별명(Alias)을 사용자가 명시할 수 있다. 이는 차후 자세히 다룰 것이다. 만약 버전 정보를 명시 하지 않으면 최신 버전을 사용한다.

위의 QML 예제 소스코드에서 Window 타입(또는 Element 라고도 함)은 가장 상위 윈도우를 생성하는 타입이다.

Window 타입 하위(내부)에 명시된 항목들은 프로퍼티(Property)라고 한다. 그리고 Window 타입 안에 내부에 속한 Image 와 Text 와 같은 하위 타입이 계층 구조로 존재할 수 있다.

예제에서 보는 것과 같이 Window 타입에서 visible 프로퍼티는 화면상에 표시할지 결정할 수 있다. Id 프로퍼티는 타입의 고유한 이름 또는 아이디를 명시한다. width 와 height는 가로와 세로 크기 이다. Color 프로퍼티는 Window 타입 내에 컬러를 지정할 수 있다.

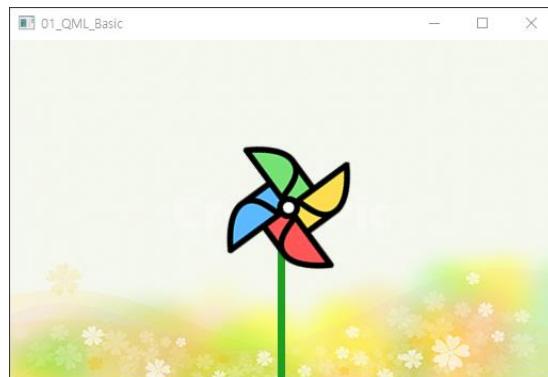
Image 타입은 Window 타입 내에서 이미지 리소스를 표시할 수 있다. Image 내에서 사용한 parent.width 는 Window 타입의 width를 의미한다. x 와 y는 Image 타입의 Window 타입에 표시할 시작 위치의 좌표이다. source 프로퍼티는 이미지 리소스의 위

예수님은 당신을 사랑합니다.

치 및 파일명을 지정한다. Text 타입에서 사용한 rocket.y 같은 Image 타입의 아이디 프로퍼티가 rocket 인 타입의 y 값을 참조할 수 있다.

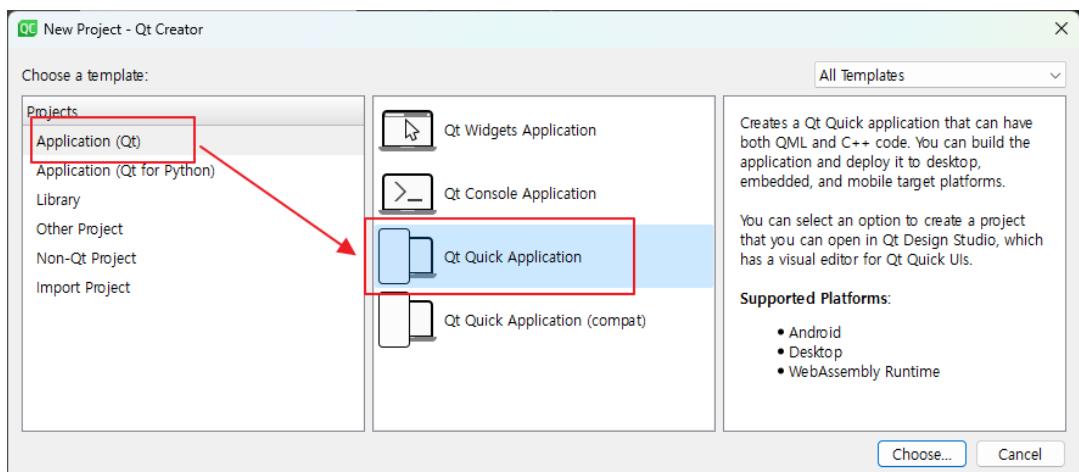
- 이미지와 마우스를 이용한 QML 예제

이번 예제는 Window 타입 영역 안에 마우스와 키보드의 좌측 방향키와 우측 방향키를 눌렀을 때 이미지를 클릭했을 때 이미지를 회전하는 예제이다.

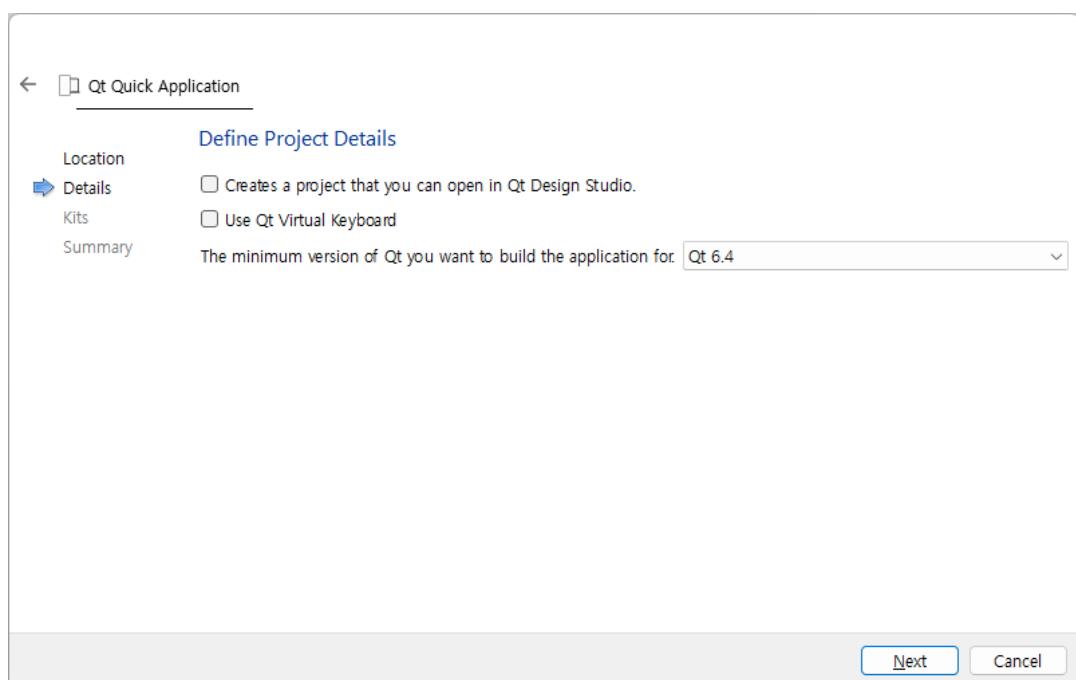
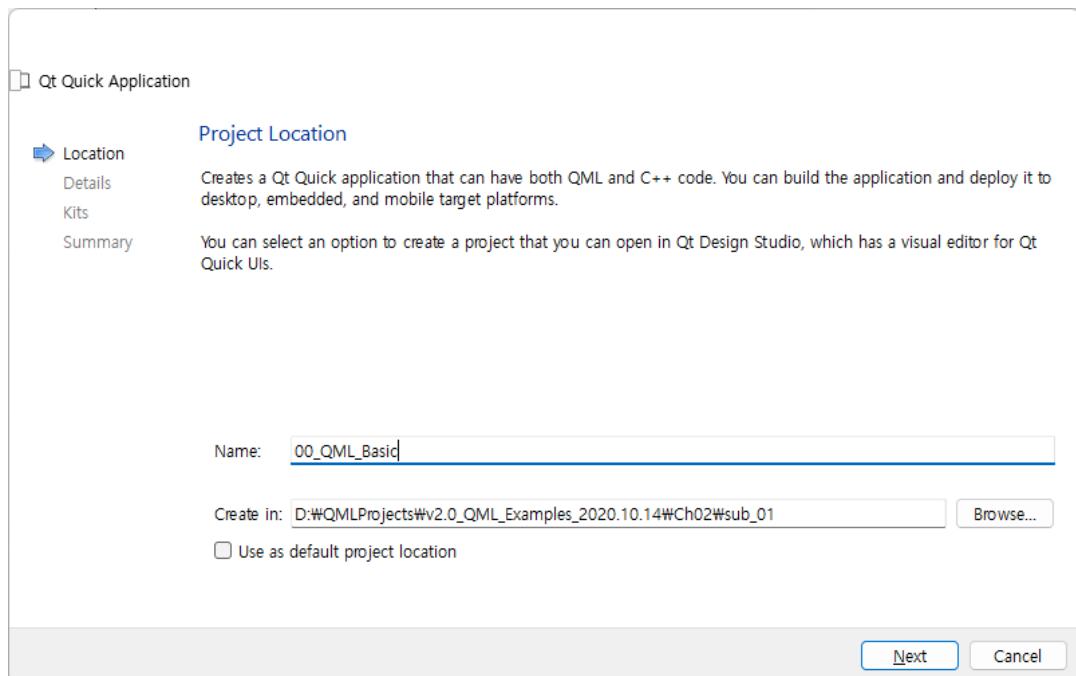


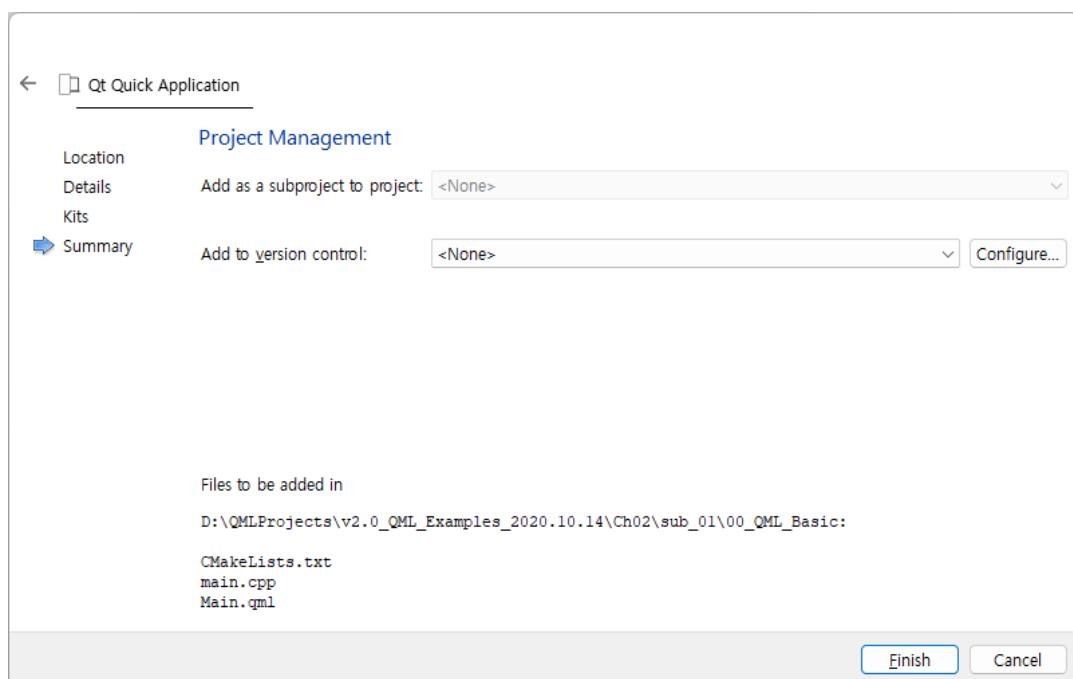
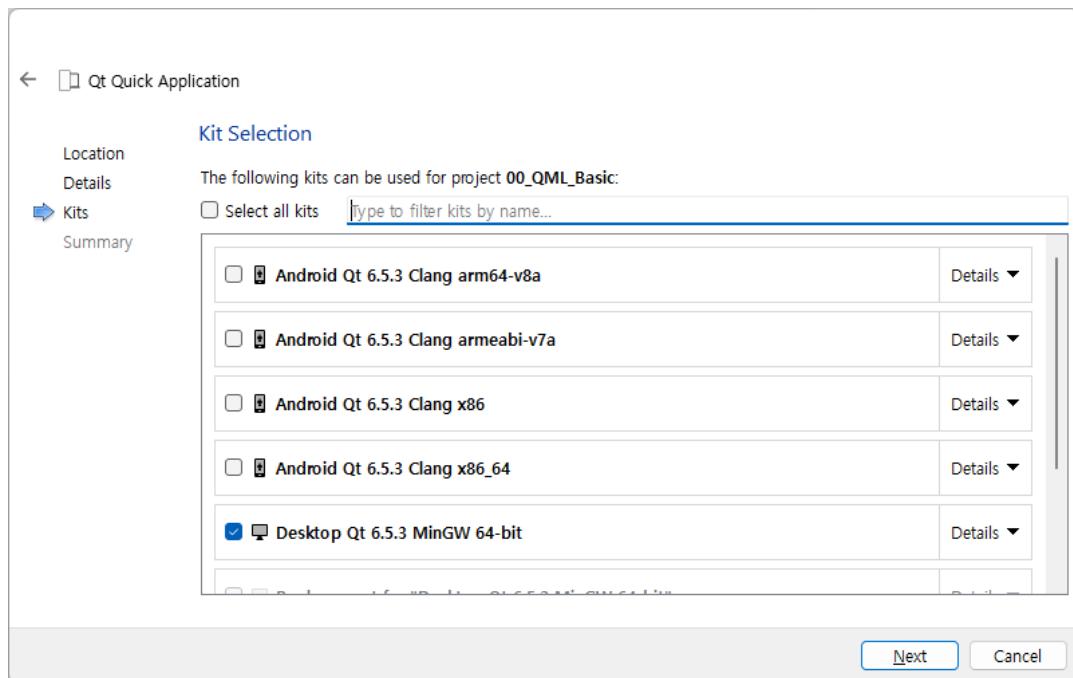
위의 그림에서 보는 것과 같이 화면상에서 마우스를 클릭하면 바람개비 이미지가 회전 한다. 그리고 키보드의 좌측 방향키를 클릭하면 바람개비 이미지가 좌측으로 회전하고 우측 방향키를 클릭하면 우측으로 회전한다. Qt Quick 을 이용해 응용 어플리케이션을 개발하기 위해서 Qt Creator 에서 새로운 프로젝트를 생성한다.

새로운 프로젝트를 생성하기 위해서는 [File] 메뉴 하위에 있는 [New Project]를 클릭한다. 그리고 [Qt Quick Application] 을 선택한다.



다음으로 Name 항목에 프로젝트 이름을 입력한다.





위의 그림에서 보는 것과 같이 다이얼로그 창에서 [Finish] 버튼을 클릭하면 프로젝트 생성이 완료된다. 여기서 한가지 궁금한 점이 생기는 독자가 있을 것이다.

프로젝트 빌드 툴로써 CMake 를 사용할 것인지 qmake를 사용할 것인지 궁금할 것이

예수님은 당신을 사랑합니다.

다. 그 이유로 Qt Creator 10.x 버전부터는 Qt Quick 관련(QML) 프로젝트 생성 시, qmake 를 선택할 수 없다. 따라서 프로젝트 생성 시 무조건 CMake를 사용한다. 하지만 수동으로 qmake 를 사용하거나 기존의 qmake 로 작성한 경우 불러와 사용할 수 있다. 따라서 프로젝트 생성 시 빌드 툴로써 CMake를 사용해야 한다. 프로젝트 생성이 완료되었으면 main.cpp를 먼저 살펴보도록 하자.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    QQmlApplicationEngine engine;
    const QUrl url(u"qrc:/00_QML_Basic/Main.qml"_qs);
    QObject::connect(&engine, &QQmlApplicationEngine::objectCreationFailed,
                     &app, []() { QCoreApplication::exit(-1); },
                     Qt::QueuedConnection);
    engine.load(url);

    return app.exec();
}
```

위의 예제 소스코드에서 보는 것과 같이 QQmlApplicationEngine 클래스는 QML파일을 로딩하는 역할을 한다. 소스 하단의 load() 멤버 함수를 이용해 QML파일을 로딩 할 수 있다. 다음으로 main.qml 파일을 아래와 같이 작성한다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    id: root
    width: 512; height: 320; color: "#D8D8D8"

    property int rotationStep: 90

    BorderImage {
```

예수님은 당신을 사랑합니다.

```
source: "images/background.png"
}

Image {
    id: pole
    anchors.horizontalCenter: parent.horizontalCenter
    anchors.bottom: parent.bottom
    source: "images/stick.png"
}

Image {
    id: wheel
    anchors.centerIn: parent
    source: "images/wheel.png"
    Behavior on rotation {
        NumberAnimation {
            duration: 250
        }
    }
}

MouseArea {
    anchors.fill: parent
    onPressed: {
        wheel.rotation += rotationStep
    }
}

Item {
    anchors.fill: parent
    focus: true
    Keys.onLeftPressed: {
        console.log("move left")
        wheel.rotation -= root.rotationStep
    }

    Keys.onRightPressed: {
        console.log("move right")
        wheel.rotation += root.rotationStep
    }
}
```

예수님은 당신을 사랑합니다.

```
}
```

```
}
```

```
}
```

Window 타입은 QML에서 가장 상위의 타입이다. 가로와 세로 크기를 width 와 height를 사용해 크기를 지정한다. color 프로퍼티는 Window 타입의 배경 색이다.

BorderImage 타입은 배경 이미지로 사용할 이미지를 지정한다. Image 타입에서 id 프로퍼티가 pole 은 바람개비 막대기 모양이다. 그리고 id 프로퍼티가 wheel 인 Image 는 바람개비 이미지이다. 각각의 이미지는 다음과 같다.



**background.png**



**stick.png**



**wheel.png**

MouseArea 타입은 Window 영역 안에서 발생하는 마우스 이벤트를 처리한다. 마우스 버튼이 클릭하면 MouseArea 타입의 onPressed 타입에서 정의한 데로 90도 rotation 한다. 90도 rotation 할 때, wheel 아이디의 Image 타입에서 선언한 NumberAnimation 타입은 애니메이션이다.

이 애니메이션은 MouseArea에서 마우스 이벤트가 발생하고 90 도 회전하게 되면 이 애니메이션이 적용된다. 따라서 90도 회전할 때 순간적으로 90도 회전이 이루어지는 것이 아니라 90도 회전하는데 250 밀리초를 사용한다는 의미이다.

Item 타입은 키보드에서 좌측 방향키를 눌렀을 때 -90도 회전시킨다. 우측 방향키를 눌렀을 때 90도 회전한다. 이미지를 프로젝트에 추가하기 위해서 프로젝트 하위 딕토리에 image 딕토리를 만든다. 그리고 3개의 이미지를 복사한다. 이미지 파일은 예제 프로젝트에 첨부하였다.

이미지를 프로젝트 딕토리에 복사한 후, CMakeLists.txt 파일을 열어서 아래와 같이 추가한다.

```
...
```

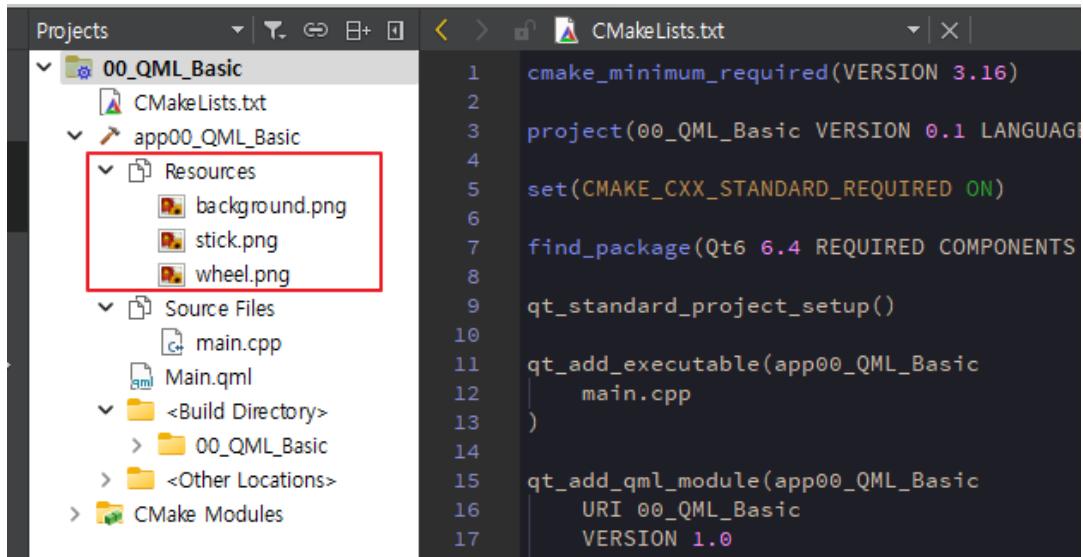
```
qt_add_executable(app00_QML_Basic
```

```
main.cpp
```

예수님은 당신을 사랑합니다.

```
)  
  
qt_add_qml_module(app00_QML_Basic  
    URI 00_QML_Basic  
    VERSION 1.0  
    QML_FILES Main.qml  
    RESOURCES  
        "images/background.png"  
        "images/stick.png"  
        "images/wheel.png"  
)  
...
```

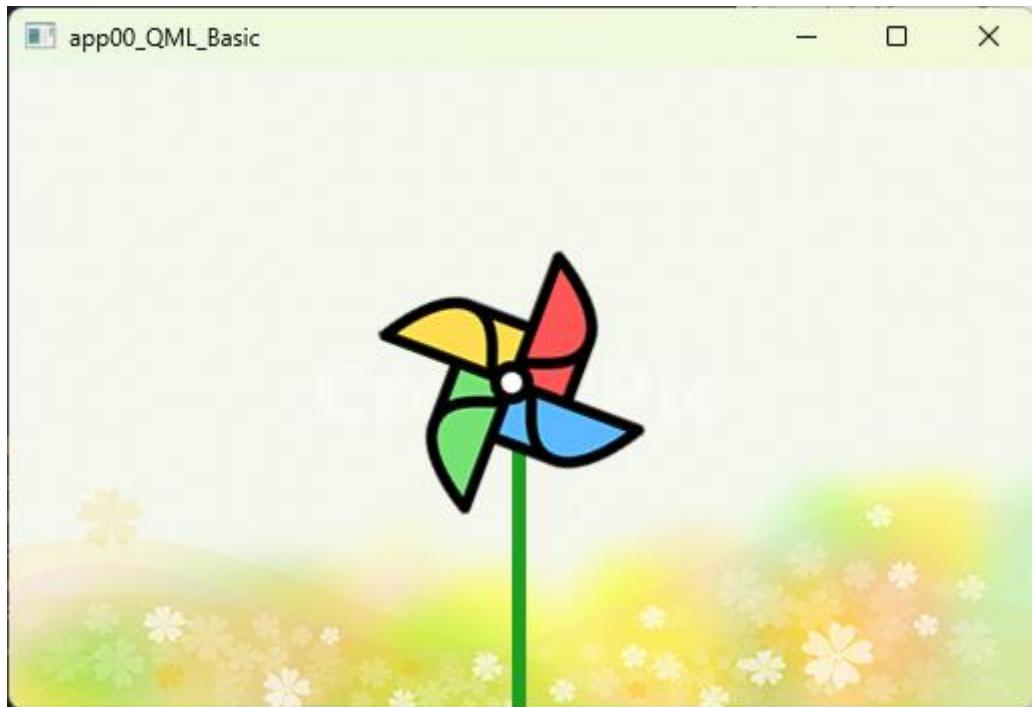
위와 같이 추가하고 저장을 누르면 프로젝트 목록에 Resources 항목이 아래와 같이 등록된 것을 확인할 수 있다.



다음으로 소스를 빌드해 실행한 다음 마우스를 클릭해보고 바람개비가 회전하는지 확인해 보자.

그리고 키보드의 좌/우 방향키를 눌렀을 때 바람개비가 회전하는지도 확인해 보자.

예수님은 당신을 사랑합니다.



이 예제의 소스코드는 00\_QML\_Basic 디렉토리를 참조하면 된다.

## 2.2. Types

QML에서 Type이란 GUI를 구현하기 위해 제공하는 오브젝트를 말한다. 이전 절에서 사용했던 Window, Image, MouseArea, Item 등을 QML에서 Type이라고 한다. QML에서는 다양한 타입을 제공하며 자주 사용되는 타입은 다음 표와 같다.

Type 명	설명
Rectangle	사각형 영역의 타입의 아이템
Image	이미지를 표시하기 위한 아이템
BorderImage	지정한 영역에서 Background 이미지를 표시
AnimatedImage	움직이는 GIF와 같은 아이템을 표시하기 위한 아이템
AnimatedSprite	연속적인 프레임을 사용해 애니메이션을 표시하기 위한 아이템
SpriteSequence	여러 아이템을 연속적인 프레임을 사용해 표시
Text	문자열을 표시하기 위한 아이템
Window	상위 윈도우를 생성하기 위한 아이템
Item	사용자 정의 아이템
Anchors	레이아웃과 같은 기능을 제공
Screen	아이템이 표시된 영역에 대한 정보
Sprite	표시된 아이템의 애니메이션 지정
Repeater	생성한 여러 아이템을 모델에 적용하기 위해 제공
Loader	모듈 혹은 파일과 같이 분리된 아이템을 표시
Transform	아이템의 이동
Scale	아이템의 크기 제어
Rotate	아이템의 회전
Translate	아이템의 이동 속성을 정의

예수님은 당신을 사랑합니다.

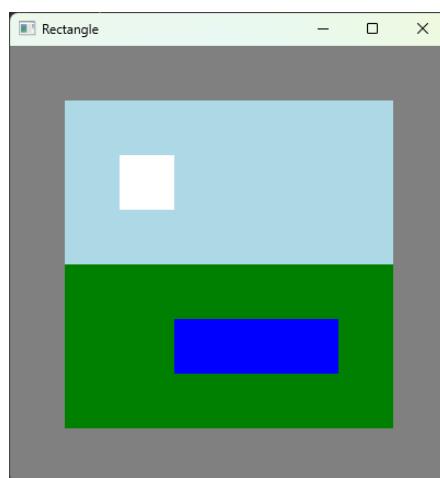
- Rectangle

Rectangle 타입은 사각형 영역에 아이템을 표시하기 위한 기능을 제공한다. 그리고 Rectangle 타입은 중첩된 형태로 사용할 수 있다.

```
import QtQuick
import QtQuick.Window

Window {
    title: "Repeater"
    visible: true
    width: 400; height: 100

    Row {
        Repeater {
            model: 3
            anchors.top: parent.top
            Rectangle {
                width: 100; height: 40
                border.width: 1
                color: "yellow"
            }
        }
    }
}
```



- **Image**

Image Type 을 사용해 이미지를 표시하기 위한 예제이다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: 300; height: 250
    title: "Image type"

    Rectangle
    {
        width: 300; height: 250
        color: "white"

        Image {
            x: 10; y: 10
            source: "images/qtlogo.png"
        }
    }
}
```



- AnimatedImage

AnimatedImage 타입은 움직이는 GIF 이미지를 화면에 랜더링 할 수 있으며 시작, 정지 기능을 제공한다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: ani.width
    height: ani.height + 8
    title: "AnimatedImage"

    AnimatedImage {
        id: ani
        source: "images/ani.gif"
    }

    Rectangle {
        property int frames: ani.frameCount
        width: 4
        height: 8
        x: (ani.width - width) * ani.currentFrame / frames
        y: ani.height
        color: "red"
    }

    MouseArea {
        anchors.fill: parent
        onClicked: {
            if( ani.paused === true )
                ani.paused = false
            else
                ani.paused = true
        }
    }
}
```

예수님은 당신을 사랑합니다.



- anchors

Qt C++ API에서 QHBoxLayout, QVBoxLayout 등과 같은 레이아웃을 사용해 위젯을 정렬하는 것과 같이 QML에서는 anchors를 제공한다. 다음은 anchors를 사용해 QML 타입을 배치한 예제이다.

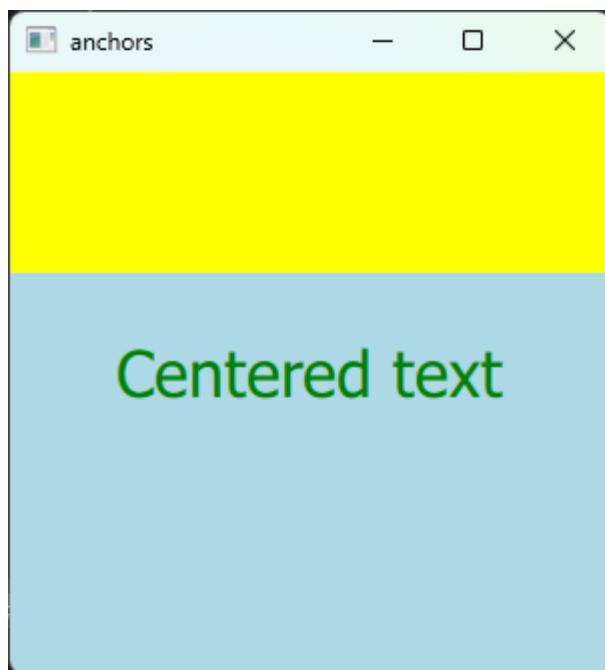
```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: 300; height: 300
    title: "anchors"

    Rectangle {
        width: 300
        height: 300
        color: "lightblue"
        id: rectangle1

        Rectangle {
            id: subRect
            width: 300
            height: 100
            color: "yellow"
        }
    }
}
```

```
Text {  
    text: "Centered text";  
    color: "green"  
    font.family: "Helvetica";  
    font.pixelSize: 32  
    anchors.top: subRect.bottom  
    anchors.centerIn: rectangle1  
}  
}  
}
```



- anchors 를 이용한 여러 개의 Type 배치 예제

다음 QML 예제소스코드는 anchors 를 이용해 QML 타입(Type)들을 배치하는 예제 이다.

```
import QtQuick  
import QtQuick.Window  
  
Window {  
    visible: true
```

예수님은 당신을 사랑합니다.

```
width: 400; height: 200
title: "Text anchors"

Rectangle {
    width: 400; height: 200

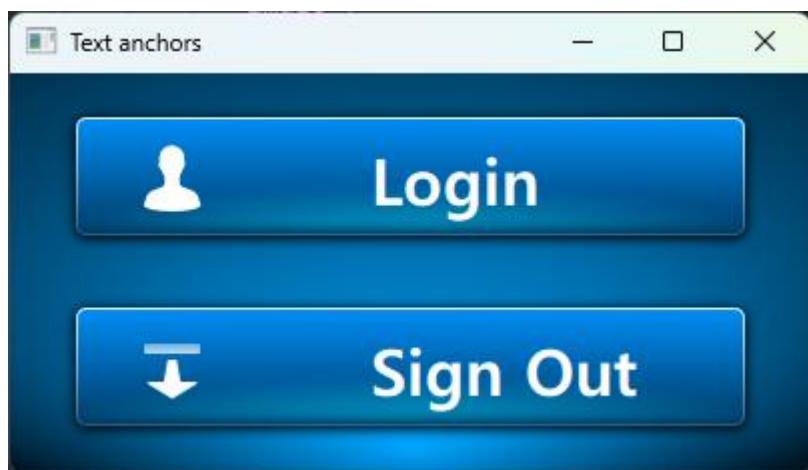
    Image {
        source: "./images/bluebackground.png"
    }

    BorderImage {
        source: "./images/bluebutton.png"
        border { left: 13; top: 13; right: 13; bottom: 13 }
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.top: parent.top
        anchors.topMargin: 15
        width: 350; height: 75
        Image {
            anchors.left: parent.left
            anchors.leftMargin: 40
            anchors.verticalCenter: parent.verticalCenter
            source: "./images/login.png"
        }
        Text {
            anchors.left: parent.horizontalCenter
            anchors.leftMargin: -20
            anchors.verticalCenter: parent.verticalCenter
            text: "Login"
            font.bold: true
            color:"white"
            font.pixelSize: 32
        }
    }
}

BorderImage {
    source: "./images/bluebutton.png"
    border { left: 13; top: 13; right: 13; bottom: 13 }
    anchors.horizontalCenter: parent.horizontalCenter
    anchors.bottom: parent.bottom
    anchors.bottomMargin: 15
    width: 350; height: 75
    Image {
```

예수님은 당신을 사랑합니다.

```
anchors.left: parent.left
anchors.leftMargin: 40
anchors.verticalCenter: parent.verticalCenter
source: "./images/signout.png"
}
Text {
    anchors.left: parent.horizontalCenter
    anchors.leftMargin: -20
    anchors.verticalCenter: parent.verticalCenter
    text: "Sign Out"
    font.bold: true
    color:"white"
    font.pixelSize: 32
}
}
}
}
```



- Gradient

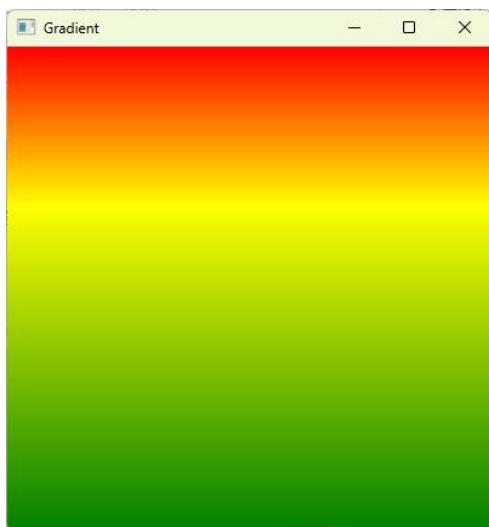
Gradient 는 지정한 영역에 색상의 범위를 지정할 수 있는 기능을 제공한다. 범위의 값은 0.0 ~ 1.0 사이의 값을 지정할 수 있다. 다음 예제는 Gradient를 사용한 예제이다.

```
import QtQuick
import QtQuick.Window

Window {
```

```
visible: true
width: 400; height: 400
title: "Gradient"

Rectangle {
    width: 400; height: 400
    gradient: Gradient {
        GradientStop { position: 0.0; color: "red" }
        GradientStop { position: 0.33; color: "yellow" }
        GradientStop { position: 1.0; color: "green" }
    }
}
}
```



- SystemPalette

SystemPalette는 Qt 어플리케이션 Palette에 접근하기 위한 기능을 제공한다. 즉 응용 어플리케이션 윈도우에서 사용되는 표준 색상에 관한 정보를 제공한다. 다음 예제는 SystemPalette를 사용한 예제이다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: 250; height: 100
```

예수님은 당신을 사랑합니다.

```
title: "SystemPalette"

Rectangle {
    width: 200; height: 80
    color: myPalette.window

    SystemPalette {
        id: myPalette
        colorGroup: SystemPalette.Active
    }

    Text {
        id: myText
        anchors.fill: parent
        text: "Hello!";
        font.pixelSize: 32
        color: myPalette.windowText
    }
}
}
```



### ● Screen

Screen은 GUI가 로딩된 스크린에 관련된 정보를 제공한다. 다음은 시스템의 스크린 Resolution 정보를 읽어오는 예제이다.

```
import QtQuick 2.12
import QtQuick.Window 2.12

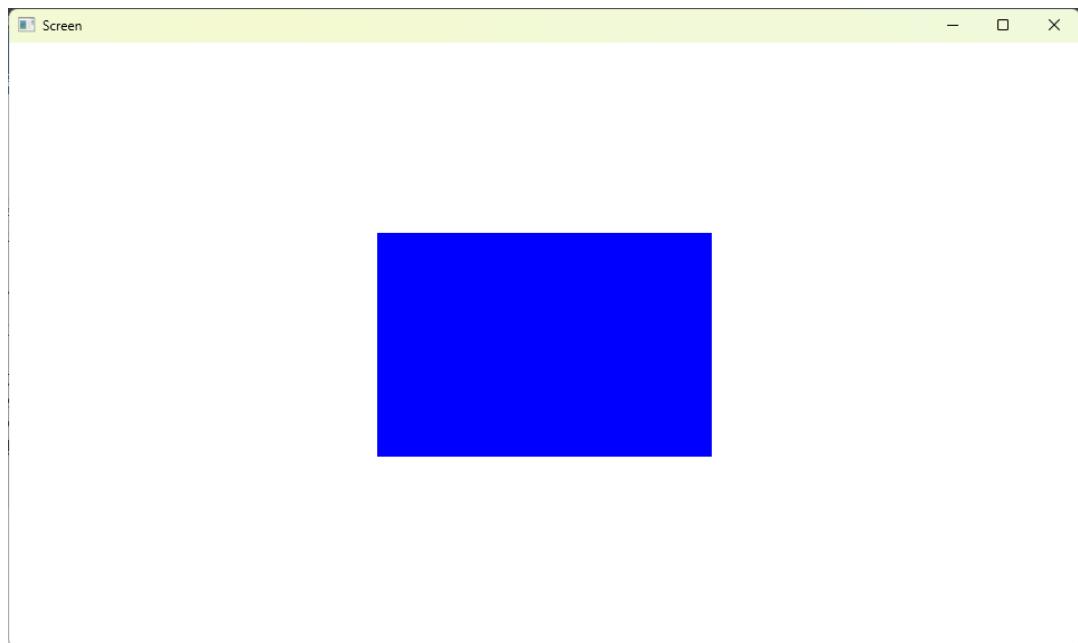
Window {
    title: "Screen"
    visible: true
```

예수님은 당신을 사랑합니다.

```
width : Screen.width / 2;
height : Screen.height / 2;

Rectangle {
    width : 300
    height: 200
    color: "blue"

    anchors.centerIn: parent
}
}
```



### ● FontLoader

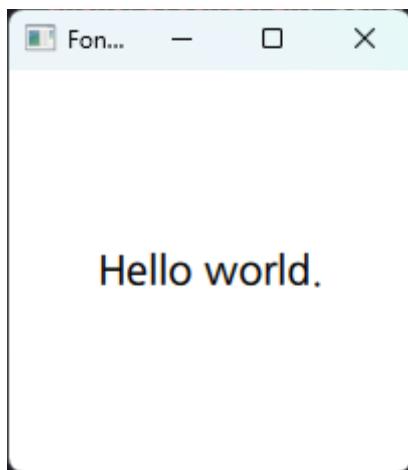
FontLoader 는 사용하고자 하는 폰트를 지정할 수 있다.

```
import QtQuick
import QtQuick.Window

Window {
    title: "FontLoader"
    visible: true
    width: 200; height: 200
```

예수님은 당신을 사랑합니다.

```
Rectangle {  
    width: parent.width  
    height: parent.height  
  
    FontLoader {  
        id: myFont  
        source: "NanumGothic.ttf"  
    }  
  
    Text {  
        text: "Hello world."  
        font.family: myFont.name  
        font.bold: true  
        font.pixelSize: 20  
        anchors.centerIn: parent  
    }  
}  
}
```

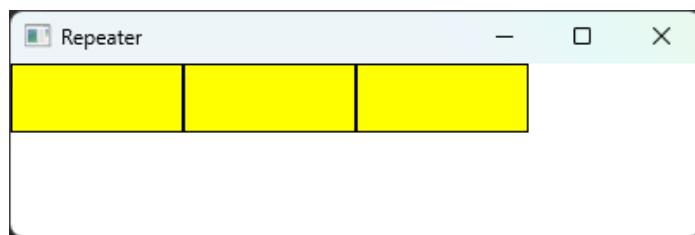


- Repeater

Repeater 타입은 동일한 QML 타입을 연속으로 배치할 수 있다. 다음 예제는 Repeater를 사용한 예제이다.

```
import QtQuick  
import QtQuick.Window
```

```
Window {  
    title: "Repeater"  
    visible: true  
    width: 400; height: 100  
  
    Row {  
        Repeater {  
            model: 3  
            anchors.top: parent.top  
            Rectangle {  
                width: 100; height: 40  
                border.width: 1  
                color: "yellow"  
            }  
        }  
    }  
}
```



### ● Transformation 과 Rotation

Image 는 이미지를 표시하기 위한 기능을 제공한다. 다음은 Image 타입을 사용해 화면에 이미지를 표시하는 예제 소스코드이다.

```
import QtQuick  
import QtQuick.Window  
  
Window {  
    title: "transformation"  
    visible: true; width: 1000; height: 220  
  
    Row {  
        x: 10; y: 10; spacing: 10
```

예수님은 당신을 사랑합니다.

```
Image {
    source: "images/qtlogo.png"
}

Image {
    source: "images/qtlogo.png"
    transform: Rotation {
        origin.x: 30; origin.y: 30
        axis { x: 0; y: 1; z: 0 } angle: 18
    }
}

Image {
    source: "images/qtlogo.png"
    transform: Rotation {
        origin.x: 30; origin.y: 30
        axis { x: 0; y: 1; z: 0 } angle: 36
    }
}

Image {
    source: "images/qtlogo.png"
    transform: Rotation {
        origin.x: 30; origin.y: 30
        axis { x: 0; y: 1; z: 0 } angle: 54
    }
}
}
```

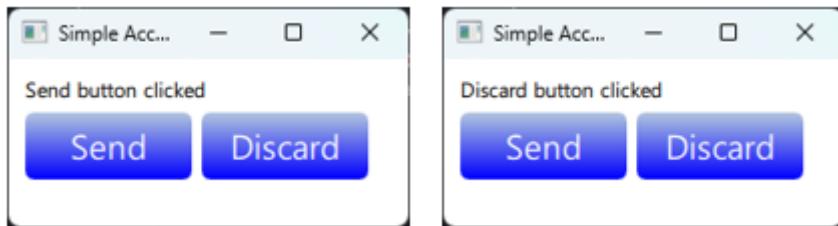


- Accessible 을 이용한 버튼 기능 구현

이번 예제는 Accessible 을 이용해 버튼을 만들고 배치하는 예제이다. 다음 그림은 예

예수님은 당신을 사랑합니다.

제 실행 화면 이다.



위의 그림에서 보는 것과 같이 [Send] 버튼을 클릭하면 상단의 Text 태입에 "Send 버튼 클릭함" 메시지를 출력한다. 그리고 [Discard] 버튼을 클릭하면 "Discard 버튼을 클릭함" 메시지를 출력한다. 이번 예제에서 다룰 두개의 QML파일 중 Button.qml을 먼저 살펴보도록 하자.

```
import QtQuick

Rectangle {
    id: button

    property alias text : buttonText.text
    Accessible.name: text
    Accessible.description: "This button does " + text
    Accessible.role: Accessible.Button
    Accessible.onPressAction: {
        button.clicked()
    }

    signal clicked

    width: buttonText.width + 20
    height: 50
    gradient: Gradient {
        GradientStop { position: 0.0; color: "lightsteelblue" }
        GradientStop { position: 1.0;
            color: button.focus ? "red" : "blue" }
    }

    radius: 5
    antialiasing: true
    Text {
        id: buttonText
        text: parent.description
        anchors.centerIn: parent
```

```
    font.pixelSize: parent.height * .5
    color: "white"
    styleColor: "black"
}
MouseArea {
    id: mouseArea
    anchors.fill: parent
    onClicked: parent.clicked()
}
}
```

위의 예제 소스코드는 Button.qml 파일 명으로 저장하였다. 파일명은 다른 QML파일에서 버튼을 구현한 QML타입을 사용하려 할 때 파일명이 QML 타입 명이 된다.

Accessible 은 버튼, Text입력창, Check 박스 등의 사용자 정의 GUI 인터페이스를 구현하는데 사용된다. Accessible.role 은 사용자 정의 GUI 인터페이스가 어떤 종류인지 선택 할 수 있다. 여기서는 Button 을 지정하였다. 이외에도 CheckBox, RadioButton, Slider, SpinBox, Dial, ScrollBar 등을 지정할 수 있다.

Accessible.onPressAction 는 버튼을 클릭 했을 때 이벤트를 지정한다. 여기서 signal clicked 를 사용하였다. 이는 C++에서 헤더 파일에 사용했던 signal 과 동일하다. 즉 clicked 라는 시그널을 사용하겠다고 정의하였다. radius 는 버튼의 각 모서리를 둥글게 하기 위해 사용한다. antialiasing 은 백터(Vector)를 적용해 화면에 표시하기 위해서 사용한다. 다음은 Button.qml을 사용할 예제를 살펴 보도록 하자.

```
import QtQuick
import QtQuick.Window
import "content"

Window {
    title: "Simple Accessible"
    visible: true
    id: window; width: 240; height: 100
    color: "white"

    Column {
        id: column; spacing: 6
        anchors.margins: 10
        anchors.fill: parent
        width: parent.width

        Text {
```

예수님은 당신을 사랑합니다.

```
    id : status
    width: column.width
}

Row {
    spacing: 6
    Button {
        id: sendButton
        width: 100; height: 40; text: "Send"
        onClicked: {
            status.text = "Send button clicked"
        }
    }
    Button { id: discardButton
        width: 100; height: 40; text: "Discard"
        onClicked: {
            status.text = "Discard button clicked"
        }
    }
}
}
```

Column 타입을 Column 내에 선언한 타입을 세로 방향으로 선언한 순서대로 배치한다. Row 타입은 Row 내에 선언한 타입을 가로 방향으로 선언한 순서대로 배치한다. 그리고 Button 타입은 Button.qml에서 정의한 사용자 정의 타입의 이름이다. 파일명이 타입 명이 된다.

## 2.3. Event

Qt Quick 은 사용자 입력과 같은 이벤트를 처리하기 위해서 다음 표에서 보는 것과 같은 다양한 타입을 제공한다.

Type 명	설명
MouseArea	특정 QML 타입 내에서 마우스 이벤트 처리
Keys	키 입력을 조작하기 위해 제공하는 추가 프로퍼티
KeyNavigation	화살표 방향의 키 네비게이션을 지원
FocusScope	키보드 포커스 조정
Flickable	Flickable은 항목을 끌어서 놓을 수 있는 GUI를 제공
PinchArea	PinchArea는 보이지 않는 항목이며 일반적으로 해당 항목에 핀치 제스처 처리를 제공하기 위해 보이는 항목과 함께 사용
MultiPointTouchArea	멀티 터치 포인터를 조작하기 위한 활성화 처리
Drag	아이템의 Drag와 Drop을 지정하기 위한 타입
DropArea	특정 영역에서 Drag와 Drop 을 지정하기 위한 타입
TextInput	사용자의 키 입력을 처리
TextEdit	텍스트를 수정할 수 있는 Text Editor
TouchPoint	터치의 좌표와 관련한 정보를 포함하는 타입
PinchEvent	Pinch 영역에서 이벤트 처리
WheelEvent	마우스 wheel 이벤트
MouseEvent	마우스 버튼 이벤트
KeyEvent	키 이벤트
DragEvent	Drag 이벤트와 관련한 이벤트

- MouseArea

C++ 기반의 어플리케이션에서 QWidget 상에서 마우스 이벤트 시그널이 발생하면 연

예수님은 당신을 사랑합니다.

결된 Slot 함수가 실행되는 것과 같이 프로퍼티를 사용해 마우스 이벤트를 처리할 수 있다. 다음은 MouseArea 타입에서 마우스 이벤트를 처리하는 예제 소스코드이다.

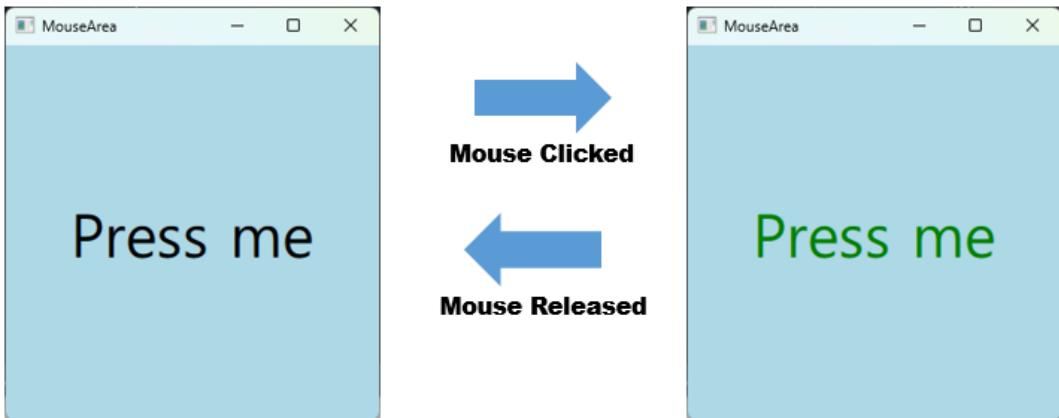
```
import QtQuick
import QtQuick.Window

Window {
    title: "MouseArea"
    visible: true
    width: 300
    height: 300

    Rectangle {
        width: parent.width
        height: parent.height
        color: "lightblue"

        Text {
            anchors.horizontalCenter: parent.horizontalCenter
            anchors.verticalCenter: parent.verticalCenter
            text: "Press me"; font.pixelSize: 48

            MouseArea {
                anchors.fill: parent
                onPressed: {
                    parent.color = "green"
                    console.log("Press")
                }
                onReleased: {
                    parent.color = "black"
                    console.log("Release")
                }
            }
        }
    }
}
```



위의 예제소스코드에서 anchors.fill은 MouseArea가 동작할 영역을 설정한다. Parent를 지정하였기 때문에 여기서는 Text 타입의 영역이 MouseArea 영역으로 설정된다.

onPressed 프로퍼티는 마우스 버튼 클릭 시 발생한다. onReleased 프로퍼티는 마우스 버튼 클릭 후 해제(Released) 시 발생한다.

이 예제의 소스코드는 00\_MouseArea 디렉토리를 참조하면 된다.

- MouseArea 영역에서 containsMouse 프로퍼티

마우스의 위치가 Rectangle 안에 들어오는 이벤트를 처리하기 위해서 containsMouse 프로퍼티를 사용할 수 있다.

containsMouse 프로퍼티를 사용하기 위해서는 hoverEnabled 프로퍼티를 true로 설정해야 한다.

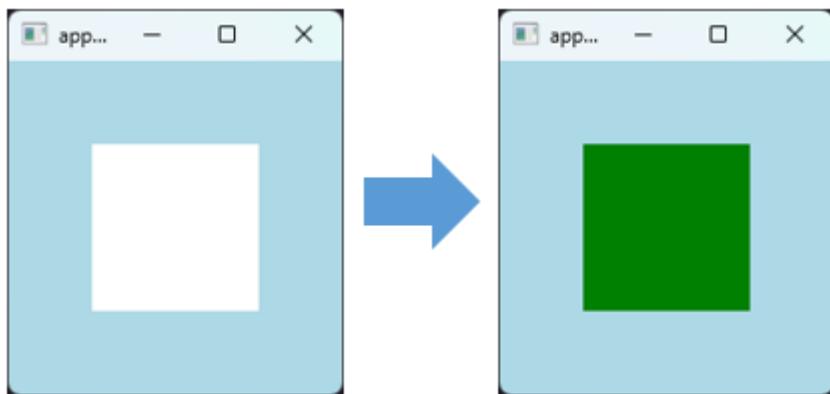
```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: 200
    height: 200

    Rectangle {
        width: 200; height: 200; color: "lightblue"
        Rectangle {
            x: 50; y: 50; width: 100; height: 100
            color: mouseArea.containsMouse ? "green" : "white"

            MouseArea {
```

```
        id: mouseArea  
        anchors.fill: parent;  
        hoverEnabled: true  
    }  
}  
}  
}
```



이 예제의 디렉토리는 01\_containsMouse 디렉토리를 참조하면 된다.

- Drag 와 DropArea

이번 예제는 Rectangle 영역 안에서 마우스를 Drag 해 DropArea 영역 안에 Rectangle 이 들어오면 DropArea 영역의 배경 Color 를 green 으로 변경하는 예제이다.

```
import QtQuick  
import QtQuick.Window  
  
Window {  
    title: "Drag"  
    visible: true  
    width: 200  
    height: 200  
  
    DropArea {  
        x: 75; y: 75  
        width: 50; height: 50  
        Rectangle {  
            anchors.fill: parent  
            color: "white"  
        }  
        onEntered: parent.color = "#008000"  
        onExited: parent.color = "white"  
        onDropped: parent.color = "white"  
    }  
}
```

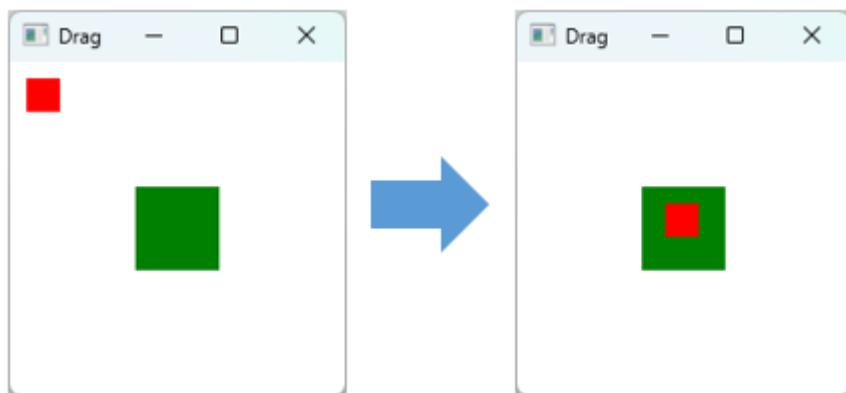
예수님은 당신을 사랑합니다.

```
        color: "green"
    }
}

Rectangle {
    x: 10; y: 10
    width: 20; height: 20
    color: "red"

    Drag.active: dragArea.drag.active
    Drag.hotSpot.x: 10
    Drag.hotSpot.y: 10

    MouseArea {
        id: dragArea
        anchors.fill: parent
        drag.target: parent
    }
}
}
```



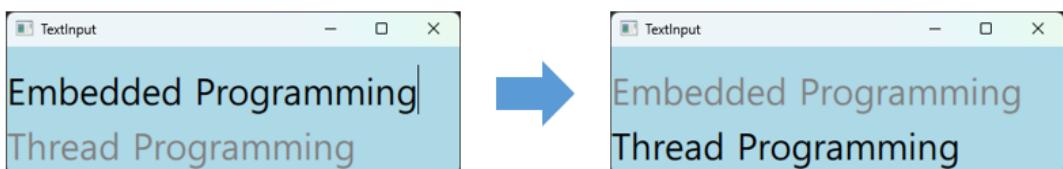
이 예제의 소스코드는 02\_drag 디렉토리를 참조하면 된다.

- TextInput 타입을 이용한 이벤트 처리

TextInput 타입은 키보드 입력 이벤트를 처리할 수 있다. 다음 예제는 TextInput 타입을 이용해 포커스 이벤트를 처리하는 예제소스코드이다.

```
import QtQuick
import QtQuick.Window
```

```
Window {  
    title: "TextInput"  
    visible: true  
    width: 400; height: 112  
  
    Rectangle {  
        width: 400; height: 112; color: "lightblue"  
        TextInput {  
            anchors.left: parent.left; y: 16  
            anchors.right: parent.right  
  
            text: "Embedded Programming";  
            font.pixelSize: 32  
            color: focus ? "black" : "gray"  
            focus: true  
        }  
  
        TextInput {  
            anchors.left: parent.left; y: 64  
            anchors.right: parent.right  
  
            text: "Thread Programming";  
            font.pixelSize: 32  
            color: focus ? "black" : "gray"  
        }  
    }  
}
```



이 예제의 디렉토리는 03\_TextInput 디렉토리를 참조하면 된다.

- KeyNavigation 타입

KeyNavigation 타입은 키의 상/하/좌/우 방향 키 이벤트를 처리할 수 있다. 방향 이외에도 TAB키 이벤트인 KeyNavigation.tab 프로퍼티를 사용할 수 있다. 그리고 "Shift +

예수님은 당신을 사랑합니다.

"Tab" 키 이벤트를 처리하기 위해서 KeyNavigation.backtab 프로퍼티를 사용할 수 있다.

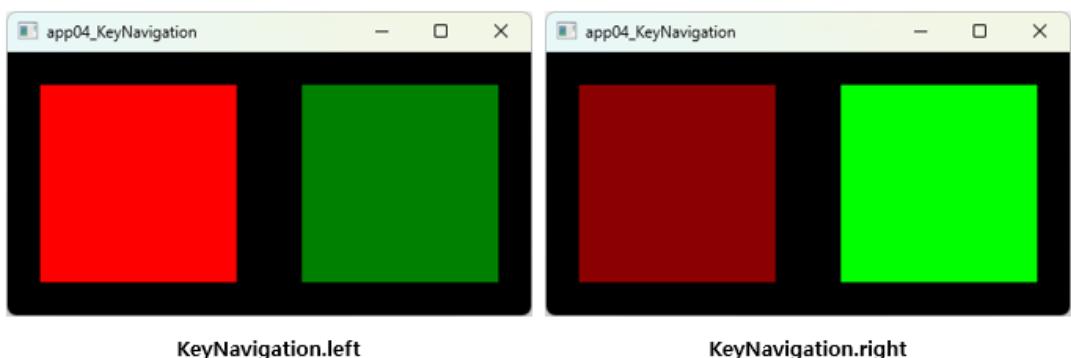
```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: 400; height: 200

    Rectangle {
        width: 400; height: 200; color: "black"

        Rectangle {
            id: leftRect
            x: 25; y: 25; width: 150; height: 150
            color: focus ? "red" : "darkred"
            KeyNavigation.right: rightRect
            focus: true
        }

        Rectangle {
            id: rightRect
            x: 225; y: 25; width: 150; height: 150
            color: focus ? "#00ff00" : "green"
            KeyNavigation.left: leftRect
        }
    }
}
```



이 예제의 소스코드는 04\_KeyNavigation 디렉토리를 참조하면 된다.

- Keys

Keys 이벤트는 키보드 이벤트를 처리할 수 있다. Keys 는 onPressed 와 onReleased 시 그널 프로퍼티를 사용하 수 있다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: 240; height: 200

    Rectangle {
        width: 230; height: 190; color: "white"

        Image {
            id: logo
            x: 30; y: 30
            source: "images/qtlogo.png"
            transformOrigin: Item.Center
            //transformOrigin: Item.Left
        }

        Keys.onPressed: (event)=> {
            if (event.key === Qt.Key_Left) {
                logo.rotation = (logo.rotation - 10) % 360
            } else if (event.key === Qt.Key_Right) {
                logo.rotation = (logo.rotation + 10) % 360
            }
        }

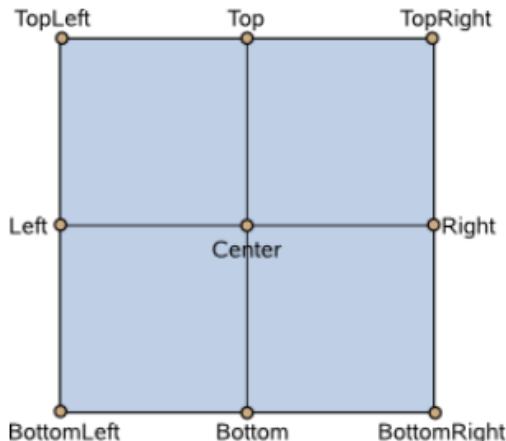
        focus: true
    }
}
```



Image 태입에서 transformOrigin 프로퍼티는 Image 태입의 이미지를 회전 시키는 중

예수님은 당신을 사랑합니다.

심점의 방향을 선택할 수 있다. 예를 들어 transformOrigin 값으로 Item.Left 값을 사용하면 회전 중심점이 Left 방향으로 회전하게 된다.



이 예제의 소스코드는 05\_Keys 디렉토리를 참조하면 된다.

- Flickable

Flickable 은 화면상에서 Rectangle, Item 등 탑입을 Drag 와 Flicked 이벤트로 이동할 수 있는 기능을 제공한다. 다음 예제 소스코드는 Flickable 예제 소스코드이다.

```
import QtQuick
import QtQuick.Window

Window
{
    title: "Flickable"
    visible: true
    width: 640; height: 480

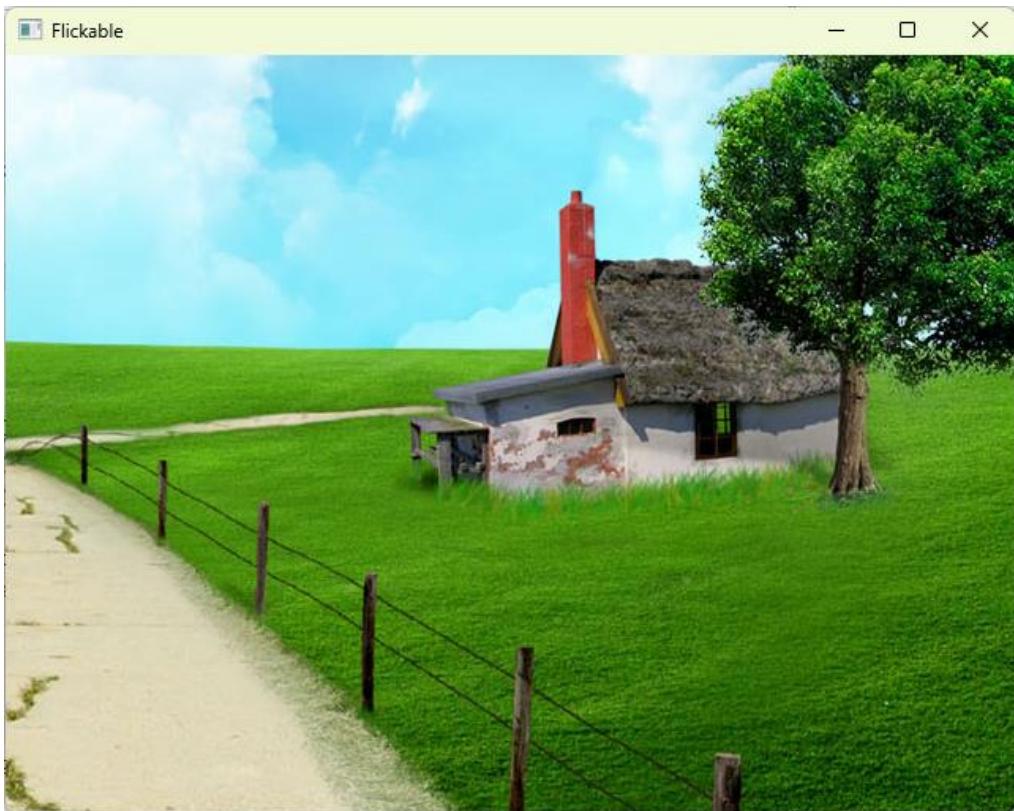
    Rectangle {
        width: 640
        height: 480

        Flickable {
            id: view
            anchors.fill: parent
            contentWidth: picture.width
            contentHeight: picture.height
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
Image {  
    id: picture  
    source: "images/background.jpg"  
}  
}  
}  
}
```

아래 그림은 예제 소스코드 실행 화면이다. 실행 화면에서 로딩된 이미지를 마우스로 Drag 해보도록 하자.

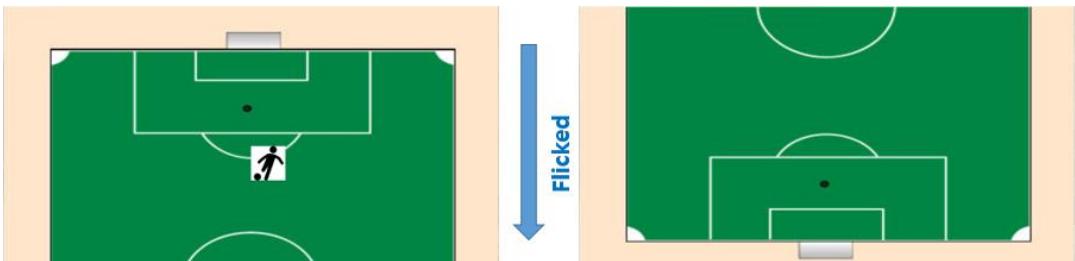


이 예제의 소스코드는 06\_Flickable 디렉토리를 참조하면 된다.

- 이미지를 상하로 Flicked 이벤트 처리 예제

이번 예제는 앞에서 다루었던 Flickable 을 이용해 이미지를 상하 좌우로 Flicked 이벤트를 처리하는 예제에 대해서 다루어 보도록 한다. 이미지는 축구장 이미지이다. 아래 그림에서 보는 것과 같이 아래로 Flicked 이벤트가 발생하면 아래로 이미지가 Drag 된다.

예수님은 당신을 사랑합니다.



```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: Screen.width
    height: Screen.height

    Flickable {
        width : Screen.width
        height : Screen.height
        contentWidth: Screen.width
        contentHeight: Screen.height * 2
        interactive: true // event on/off

        Image {
            id: ground
            anchors.fill: parent
            source : "images/ground.jpg"
            sourceSize.width: Screen.width
            sourceSize.height: Screen.height * 2
        }

        Image {
            id: player
            source : "images/player.png"
            x: Screen.width / 2
            y: Screen.height / 2
        }
    }
}
```

이 예제의 소스코드는 07\_Flickable\_Ground 디렉토리를 참조하면 된다.

- Signal 과 Signal Handler

QML에서 Signal은 타입에서 발생하고 Signal Handler는 Signal과 연결되어 Signal이 발생한 이벤트를 처리하기 위한 기능을 제공한다.

예를 들어 아래 예제 소스코드에서 보는 것과 같이 clicked라는 Signal이 발생하면 onClicked라는 Signal Handler가 이벤트를 수행한다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true; width: 360; height: 360
    Rectangle {
        id: rect
        width: 360; height: 360; color: "blue"

        MouseArea {
            anchors.fill: parent
            onClicked: {
                rect.color = Qt.rgba(Math.random(),
                                     Math.random(),
                                     Math.random(), 1);

                console.log("Clicked mouse at", mouseX, mouseY)
            }
        }
    }
}
```

이 예제의 소스코드는 08\_SignalHandler\_exam1 디렉토리를 참조하면 된다.

- Connections 타입

Connections 타입은 Signal Handler를 Connections 타입 내에서 처리할 수 있다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true; width: 360; height: 360
```

```
Rectangle {  
    id: rect  
    width: 360; height: 360; color: "blue"  
  
    MouseArea {  
        id: mouseArea  
        anchors.fill: parent  
    }  
  
    Connections {  
        target: mouseArea  
        onClicked: {  
            rect.color = Qt.rgba(Math.random(),  
                               Math.random(),  
                               Math.random(), 1);  
        }  
    }  
}
```

이 예제의 소스코드는 08\_SignalHandler\_exam2 디렉토리를 참조하면 된다.

- 사용자 정의 Signal 추가

QML에서는 사용자가 정의한 Signal 을 추가하기 위한 Syntax 는 다음과 같다.

```
signal <name>[([<type> <parameter name>[, ...]])]
```

QML 타입에서 Signal 이 Emit(이벤트 발생) 되면 Signal 과 연결된 특정 함수가 실행 되게 하기 위해서 다음과 같이 실행 할 수 있다.

다음 예제는 Signal을 사용자가 직접 추가한 예제이다. 이번 예제는 2개의 예제 소스코드로 되어 있다.

첫 번째는 content 라는 디렉토리 안에 SquareButton.qml 존재한다. 그리고 main.qml로 되어 있다. SquareButton.qml 에서 사용자가 정의한 QML 타입을 main.qml 에서 사용한다. 먼저 SquareButton.qml 예제 소스코드를 살펴보자.

```
import QtQuick  
  
Rectangle {  
    id: root  
    signal activated(real xPosition, real yPosition)
```

예수님은 당신을 사랑합니다.

```
signal deactivated
width: 100; height: 100

MouseArea {
    anchors.fill: parent
    onPressed: root.activated(mouseX, mouseY)
    onReleased: {
        root.deactivated()
    }

}
}
```

다음은 main.qml 예제 소스코드 이다.

```
import QtQuick
import QtQuick.Window
import "content"

Window {
    visible: true; width: 360; height: 360
    SquareButton
    {
        width: 360; height: 360
        onActivated: console.log("Activated at " + xPosition + "," + yPosition)
        onDeactivated: console.log("Deactivated!")
    }
}
```

이 예제의 소스코드는 08\_SignalHandler\_exam3 디렉토리를 참조하면 된다.

- Signal 과 Method(또는 함수) 연결

다음 예제 코드는 Signal 과 Method를 연결하기 위한 예제 소스코드 이다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true; width: 360; height: 360

    id: relay
    signal messageReceived(string person, string notice)
```

```
Component.onCompleted: {
    relay.messageReceived.connect(sendToPost)
    relay.messageReceived.connect(sendToTelegraph)
    relay.messageReceived.connect(sendToEmail)
}

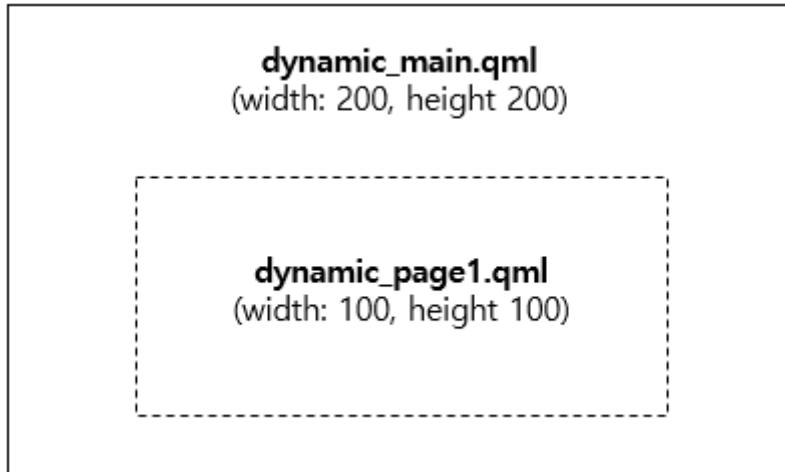
function sendToPost(person, notice) {
    console.log("Sending to post: " + person + ", " + notice)
}
function sendToTelegraph(person, notice) {
    console.log("Sending to message: " + person + ", " + notice)
}
function sendToEmail(person, notice) {
    console.log("Sending to email: " + person + ", " + notice)
}

MouseArea {
    anchors.fill: parent
    onClicked: {
        relay.messageReceived("Tom", "Happy Birthday")
    }
}
```

이 예제 소스코드는 Signal 이 발생하면 Component.onCompleted 에 등록된 Method 가 호출된다. 이 예제의 소스코드는 08\_SignalHandler\_exam4 디렉토리를 참조하면 된다.

## 2.4. Loader type 을 이용한 Dynamic UI 구성

이번 절에서는 QML 내에서 특정 영역에 다른 QML을 동적으로 호출하기 위한 기능을 살펴보도록 하자. QML내에 또 다른 QML 파일을 로딩 하기 위해서 Loader 타입을 제공한다.



위의 그림에서 보는 것과 같이 Loader 타입을 사용하면 QML 내에 QML을 동적으로 로딩 할 수 있다. 다음 예제는 `dynamic_main.qml` 상에서 `dynamic_page1.qml` 파일을 로딩하는 예제이다. 먼저 `dynamic_main.qml` 예제 소스코드를 살펴보도록 하자.

```
import QtQuick
import QtQuick.Window

Window {
    width: 200; height: 200; visible: true
    Loader {
        id: pageLoader
        anchors.top: myRect.bottom
    }

    Rectangle {
        id: myRect
        width: 200; height: 100
        color: "yellow"
        Text {
            anchors.centerIn: parent
            text: "Dynamic UI Loading Example"
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
        text : "Main QML"; font.bold: true
    }

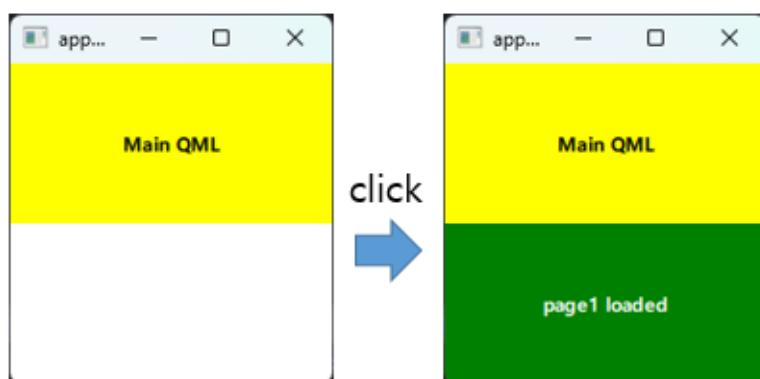
MouseArea {
    anchors.fill: parent
    onClicked: {
        pageLoader.source = "dynamic_page1.qml"
    }
}
}
```

위의 예제 소스코드에서 Loader 타입이 선언된 영역에서 page1.qml 파일을 로딩할 수 있다. Loader 타입에서 id 를 pageLoader 로 지정하였다.

그리고 마우스를 클릭하면 id 가 pageLoader 인 source 프로퍼티를 이용해 로딩할 QML 파일을 지정 하였다. 따라서 MyRect 라는 아이디의 Rectangle 타입 영역 내에서 마우스를 클릭하면 page1.qml 파일이 로딩된다. 다음은 dynamic\_page1.qml 이다.

```
import QtQuick

Rectangle {
    width: 200; height: 100; color: "green"
    Text {
        anchors.centerIn: parent
        text: "page1 loaded"
        font.bold: true;
        color: "#FFFFFF"
    }
}
```



예수님은 당신을 사랑합니다.

- Component 타입을 이용해 Loader 타입 영역 내에 로딩

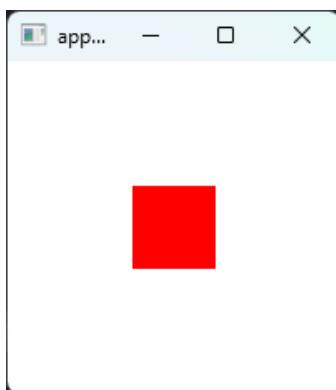
Component 타입은 Loader 영역에서 로딩되는 탑입을 지정할 수 있다. 다음 예제 소스는 Component 타입을 사용한 예제이다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true; width: 200; height: 200

    Loader {
        anchors.centerIn: parent
        sourceComponent: rect
    }

    Component {
        id: rect
        Rectangle {
            width: 50
            height: 50
            color: "red"
        }
    }
}
```



- Loader 타입에서 이벤트 처리

Loader 타입에서 발생한 이벤트를 처리하기 위해서 Connections 타입을 사용할 수 있다. 다음 예제는 Connections를 이용해 시그널을 연결한 예제이다.

예수님은 당신을 사랑합니다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: 100
    height: 100

    Loader {
        id: myLoader
        source: "application_myitem.qml"
    }

    Connections {
        target: myLoader.item
        function onMessage(msg) {
            console.log(msg)
        }
    }
}
```

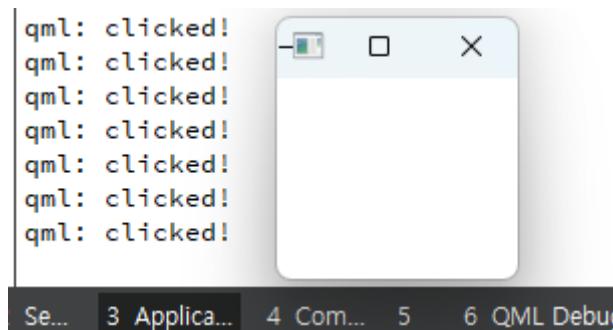
위의 예제 소스코드에서 Loader 타입에서 application\_myitem.qml 파일을 로딩 한다. 그리고 Connection 타입에서 onMessage 라는 프로퍼티는 application\_myitem.qml 에서 등록한 시그널이다.

application\_myitem.qml 에서 message 라는 시그널을 Connections type 에서 사용하기 위해서 message 시그널 이름 앞에 첫 번째 철자 m 을 M으로 바꾼다. 그리고 시그널 문자열 앞에 on 이라는 문자열을 붙인다. 다음은 application\_myitem.qml 예제 소스코드 이다.

```
import QtQuick

Rectangle {
    id: myItem
    signal message(string msg)
    width: 100; height: 100

    MouseArea {
        anchors.fill: parent
        onClicked: myItem.message("clicked!")
    }
}
```



- Loader 영역에서 Key 이벤트 처리

Loader 타입에서 로딩되는 QML 파일로부터 발생한 Key 이벤트를 처리하기 위해서 focus 프로퍼티 값을 true 할당해야 한다. 다음 예제는 key\_main.qml 예제 소스 코드이다.

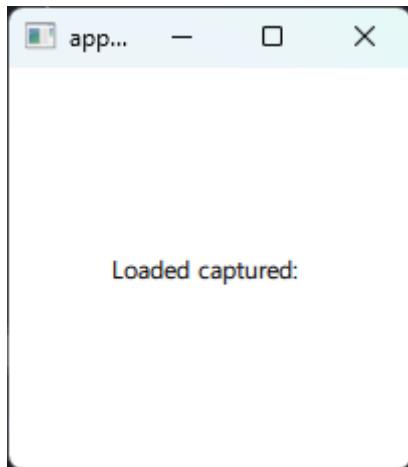
```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: 200
    height: 200

    Loader {
        id: loader
        focus: true
        source: "key_keyreader.qml"
        anchors.centerIn: parent
    }
}
```

위에 예제에서 보는것과 같이 Loader 타입에서는 key\_keyreader.qml 소스파일을 로딩 한다. 다음은 key\_keyreader.qml 예제 소스코드이다.

예수님은 당신을 사랑합니다.



- 2개 이상의 Loader 호출

이번 예제는 2개의 Loader 타입을 이용해 호출하는 예제이다.

```
import QtQuick
import QtQuick.Window

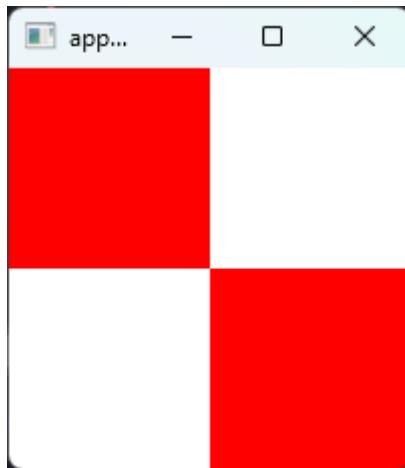
Window {
    visible: true
    width: 200
    height: 200

    Component {
        id: redSquare
        Rectangle {
            color: "red"; width: 100; height: 100
        }
    }

    Loader {
        sourceComponent: redSquare
    }

    Loader {
        sourceComponent: redSquare; x: 100; y: 100
    }
}
```

예수님은 당신을 사랑합니다.



- Loader 타입의 status 프로퍼티

Loader 타입에서 제공하는 status 프로퍼티는 Loader 타입의 상태를 제공한다.

종류	설명
Null	Loader 영역에서 호출하는 QML소스가 비 활성화 된 상태 또는 존재하지 않는 경우
Ready	QML 소스가 로딩할 준비가 되었을 때
Loading	QML 소스가 현재 로딩 되고 있을 때
Error	QML 소스가 로딩 중에 에러가 발생 했을 때.

다음은 Loader 타입에서 상태를 사용한 예제 소스코드 이다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: 200; height: 200

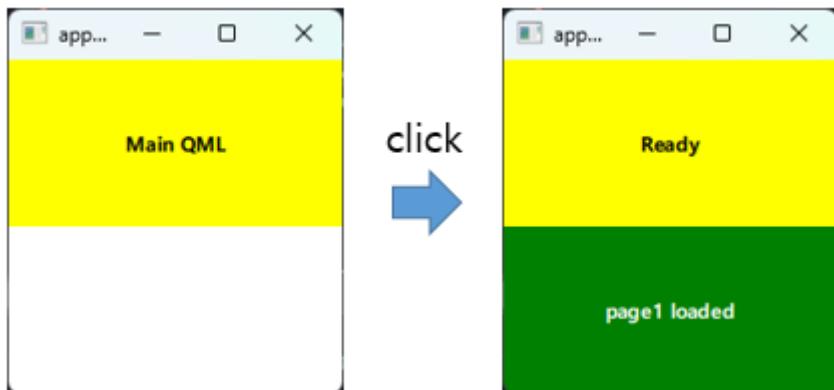
    Loader {
        id: pageLoader
        anchors.top: myRect.bottom
        onStatusChanged: {
            if (pageLoader.status == Loader.Ready)
                myText.text = "Ready"
        }
    }
}
```

```
}

Rectangle {
    id: myRect
    width: 200; height: 100;
    color: "yellow"

    Text {
        id: myText
        anchors.centerIn: parent
        text : "Main QML"; font.bold: true
    }

    MouseArea {
        anchors.fill: parent
        onClicked: pageLoader.source = "dynamic_page1.qml"
    }
}
}
```



위의 그림에서 Main QML 이라고 출력한 영역을 클릭하면 page1.qml 파일이 하단에 로딩되고 콘솔 창에 Ready 문자열을 출력한다.

- Loader 타입으로 불러온 QML 파일의 프로퍼티의 값을 변경

이번에는 Loader 타입으로 불러온 QML 파일 내에 선언한 프로퍼티의 값을 변경하는 방법에 대해서 살펴보도록 하자. 이번 예제는 value\_change.qml 과 ExComponent.qml로 되어 있다. value\_change.qml 에서 ExComponent.qml을 Loader 타입으로 불러오는 예이다. 먼저 value\_change.qml 파일의 예제 소스코드 살펴보자.

예수님은 당신을 사랑합니다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: 200; height: 200

    Loader {
        id: squareLoader
        onLoaded: {
            console.log("Width : " + squareLoader.item.width);
        }
    }

    Component.onCompleted: {
        squareLoader.setSource("ExComponent.qml", { "color": "blue" });
    }
}

Rectangle {
    anchors.top: squareLoader.bottom
    width: 200; height: 100
    color: "green"
    MouseArea {
        anchors.fill: parent
        onClicked: {
            squareLoader.setSource("ExComponent.qml", {"width": 200})
        }
    }
}
```

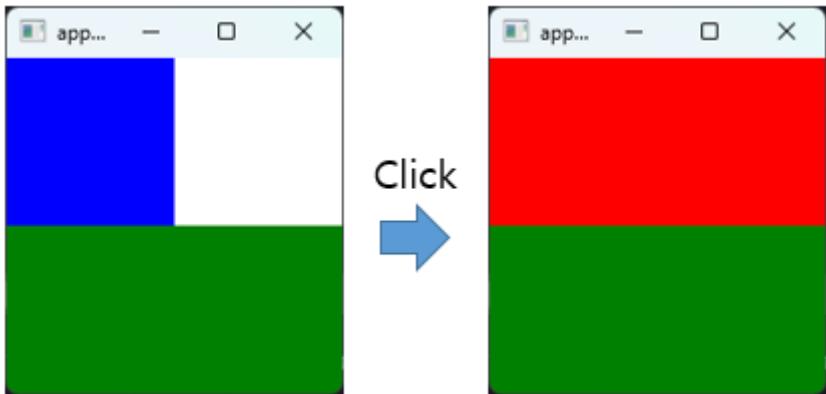
위의 예제 소스코드에서 보는 것과 같이 Loader 에서 불러온 ExComponent.qml 과 하단의 Rectangle을 로딩한다. 다음은 ExComponent.qml 소스코드 이다.

```
import QtQuick

Rectangle {
    id: rect
    color: "red"
    width: 100
    height: 100
}
```

예수님은 당신을 사랑합니다.

value\_change.qml 에서 ExComponent.qml을 Loader 를 이용해 불러온다. 따라서 2개의 Rectangle 타입의 사각형이 다음 그림에서 보는 것과 같이 로딩된다.



위의 그림에서 보는 것과 같이 아래 "green" Color 의 Rectangle을 클릭하면 우측에서 보는 것과 같이 상단의 Rectangle의 width 프로퍼티의 값이 100에서 200으로 변경된다. 또한 Color 값이 "red"로 변경된다.

- Loader 타입을 이용한 동적 QML 로딩 예제

이번 예제는 로딩된 QML에서 하단의 첫 번째 메뉴와 두 번째 메뉴 영역을 클릭 시 상단의 QML을 변경하는 예제이다. 다음은 예제 실행 화면이다.



위의 그림에서 보는 것과 같이 하단 우측의 [두 번째 메뉴]를 클릭하면 그림 우측과 같이 상단 메뉴가 Yellow 색으로 변경되고 하단의 우측이 녹색으로 변경된다. 이번 예제

예수님은 당신을 사랑합니다.

는 다음과 같이 3개의 QML 파일로 구성된다.

QML 파일 명	설명
main.qml	menu1.qml 과 menu2.qml 파일을 Loader 타입을 이용해 불러온다.
menu1.qml	"white" 컬러의 Rectangle
menu2.qml	"yellow" 컬러의 Rectangle

위의 표에서 보는 것과 같이 이번 예제는 3개의 QML 파일로 구성되어 있다. 먼저 main.qml 파일의 소스코드부터 살펴보자.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true; width: 400; height: 400

    Rectangle {
        id: root
        width: 400; height: 400

        Loader {
            id: myLoader
            anchors.left: parent.left; anchors.right: parent.right
            anchors.top: parent.top; anchors.bottom: menu1Button.top
            onLoaded: { source: "menu1.qml" }
        }

        Rectangle {
            id: menu1Button
            anchors.left: parent.left; anchors.bottom: parent.bottom
            color: "gray"
            width: parent.width/2
            height: 100

            Text {
                anchors.centerIn: parent;
                text: "First Menu";
                font.bold: true; font.pixelSize: 20; color: "white"
            }
            MouseArea {
                anchors.fill: parent
```

예수님은 당신을 사랑합니다.

```
        onClicked: root.state = "menu1";
    }
}

Rectangle {
    id: menu2Button

    anchors.right: parent.right
    anchors.bottom: parent.bottom

    color: "gray"

    width: parent.width/2
    height: 100

    Text {
        anchors.centerIn: parent
        text: "Second Menu";
        font.bold: true; font.pixelSize: 20; color: "white"
    }

    MouseArea {
        anchors.fill: parent
        onClicked: root.state = "menu2";
    }
}

state: "menu1"

states: [
    State {
        name: "menu1"
        PropertyChanges { target: menu1Button; color: "green"; }
        PropertyChanges { target: myLoader; source: "menu1.qml"; }
    },
    State {
        name: "menu2"
        PropertyChanges { target: menu2Button; color: "green"; }
        PropertyChanges { target: myLoader; source: "menu2.qml"; }
    }
]
```

예수님은 당신을 사랑합니다.

}

하단의 State 타입은 State Machine 으로 현재 Status 가 "menu1" 이면 상단의 Rectangle 의 QML 을 menu1.qml QML 파일을 로딩하고 Color 를 "green" 으로 변경 한다.

그리고 Status 가 "menu2" 이면 상단의 Rectangle 의 QML 을 menu2.qml 파일로 변경하고 Color 를 "green" 으로 변경한다. 다음은 menu1.qml 예제 소스코드 이다.

```
import QtQuick

Rectangle {
    width: 400; height: 300; color: "blue"
    Text {
        anchors.centerIn: parent
        text: "Menu 1"
        font.bold: true
        font.pixelSize: 30
        color: "white"
    }
}
```

다음으로 menu1.qml 예제 소스코드를 아래와 같이 작성한다.

```
import QtQuick

Rectangle {
    width: 400; height: 300; color: "yellow"
    Text {
        anchors.centerIn: parent
        text: "Menu 2"
        font.bold: true
        font.pixelSize: 30
        color: "black"
    }
}
```

이 예제의 소스코드는 01\_Dynamic\_Menu\_Example 디렉토리를 참조하면 된다.

## 2.5. Canvas

Canvas 는 QWidget 클래스에서 사용했던 QPainter 클래스와 같이 QML 영역 내에서 Drawing을 하기 위한 기능을 제공한다. QML 영역 내에서 선, 도형, Gradients 등과 같은 요소 Drawing 과 Image를 표시(Rendering)할 수 있으며 width 와 height 프로퍼티를 사용해 Drawing 영역의 크기를 지정할 수 있다.

```
import QtQuick

Canvas {
    id: mycanvas
    width: 100
    height: 200
}
```

QWidget 에서 QPainter를 사용할 경우 update( ) 또는 repaint( )와 같이 Canvas 에서는 onPaint 프로퍼티를 사용할 수 있다.

```
import QtQuick
Canvas {
    id: mycanvas; width: 100; height: 200
    onPaint: {
        ...
    }
}
```

- Canvas 타입 영역 내에 간단한 사각형 그리기

이번 예제는 Canvas영역에 사각형을 그리는 예제이다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: 200; height: 200

    Canvas {
        id: root
```

예수님은 당신을 사랑합니다.

```
width: 200; height: 200

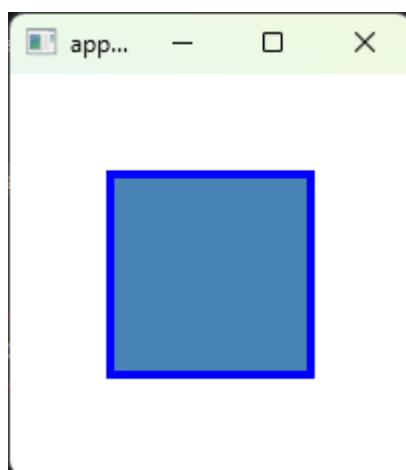
onPaint:
{
    var ctx = getContext("2d")

    ctx.lineWidth = 4
    ctx.strokeStyle = "blue"
    ctx.fillStyle = "steelblue"

    ctx.beginPath()
    ctx.moveTo(50,50)
    ctx.lineTo(150,50)
    ctx.lineTo(150,150)
    ctx.lineTo(50,150)
    ctx.closePath()

    ctx.fill()
    ctx.stroke()
}
}
```

위의 소스코드에서 onPaint 는 QWidget 에서 사용하는 QPainter 클래스와 동일한 기능을 제공한다. onPaint 내에 getContext( ) 함수에 “2d” 라는 Argument를 지정하였다. Canvas 에서는 “2d” 만 선택할 수 있다. 따라서 Canvas 영역에서 “3d” 는 지정할 수 없다.



- 다양한 형태의 사각형 Drawing

Canvas 영역에 다음 예제 소스코드와 같이 다양한 사각형을 Drawing 할 수 있다.

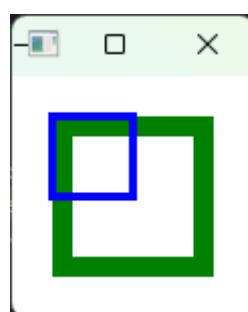
```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    width: 120
    height: 120

    Canvas {
        id: root
        width: 120
        height: 120

        onPaint: {
            var ctx = getContext("2d")
            ctx.fillStyle = 'green'
            ctx.strokeStyle = "blue"
            ctx.lineWidth = 4

            ctx.fillRect(20, 20, 80, 80)
            ctx.clearRect(30,30, 60, 60)
            ctx.strokeRect(20,20, 40, 40)
        }
    }
}
```



예수님은 당신을 사랑합니다.

- Canvas 영역에 Gradients 사용

```
import QtQuick
import QtQuick.Window

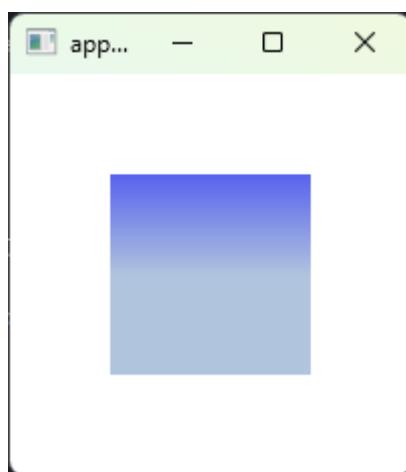
Window {
    visible: true; width: 200; height: 200

    Canvas {
        id: root
        width: 200; height: 200

        onPaint: {
            var ctx = getContext("2d")

            var gradient = ctx.createLinearGradient(100,0,100,200)
            gradient.addColorStop(0, "blue")
            gradient.addColorStop(0.5, "lightsteelblue")

            ctx.fillStyle = gradient
            ctx.fillRect(50,50,100,100)
        }
    }
}
```



- Canvas 영역에 마우스로 Drawing

이번 예제는 Canvas 영역에서 마우스 버튼을 누른 상태에서 드래그해 Drawing 하는

예수님은 당신을 사랑합니다.

예제를 살펴보도록 하자.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true; width: 360; height: 360

    Rectangle {
        id: root
        width: 360
        height: 360

        Canvas {
            id: myCanvas
            anchors.fill: parent

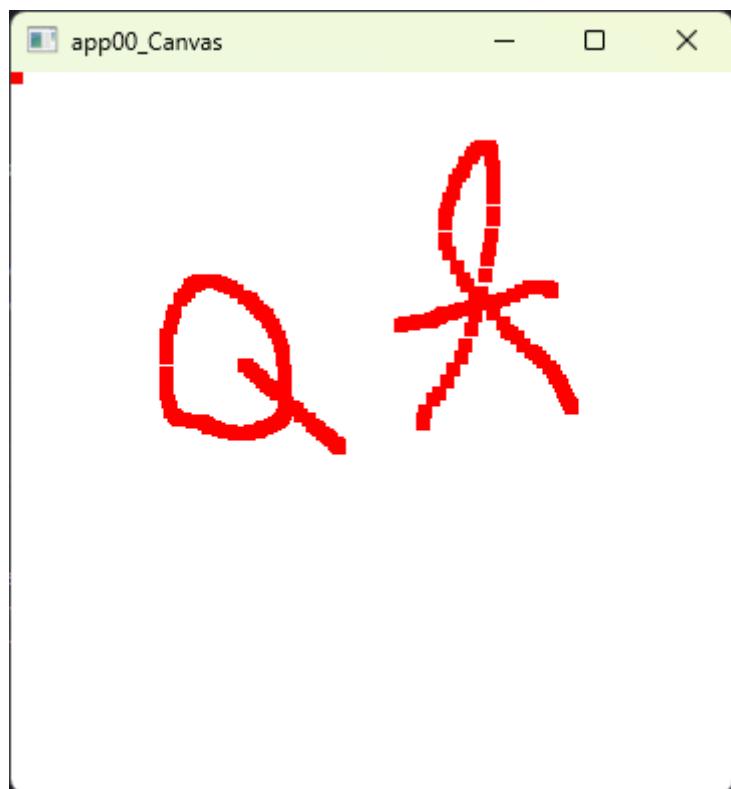
            property int xpos
            property int ypos

            onPaint: {
                var ctx = getContext('2d')
                ctx.fillStyle = "red"
                ctx.fillRect(myCanvas.xpos-1, myCanvas.ypos-1, 7, 7)
            }

            MouseArea{
                anchors.fill: parent
                onPressed: {
                    myCanvas.xpos = mouseX
                    myCanvas.ypos = mouseY
                    myCanvas.requestPaint()
                }
                onMouseXChanged: {
                    myCanvas.xpos = mouseX
                    myCanvas.ypos = mouseY
                    myCanvas.requestPaint()
                }
                onMouseYChanged: {
                    myCanvas.xpos = mouseX
                    myCanvas.ypos = mouseY
                    myCanvas.requestPaint()
                }
            }
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
        }  
    }  
}  
}
```



## 2.6. Graphics Effects

Qt에서는 Effects로 Blending, Masking, Blurring, Coloring 등과 같은 Effect를 사용할 수 있다.

### ● Blend Effect



Source



Foreground Source

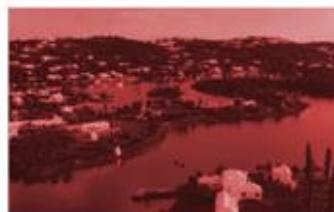


Result

### ● Colorize Effect



Source



hue: 0.0  
saturation: 0.5  
lightness: -0.2



hue: 0.8  
saturation: 1.0  
lightness: 0

Qt Quick에서 제공하는 Graphic Effects를 QML에서 사용하기 위해서 QML상에 다음과 같이 import 해야한다.

```
import Qt5Compat.GraphicalEffects
```

QtGraphicalEffects 모듈은 다음 표에서 보는 것과 같이 다양한 Effects를 제공한다.

분류	타입	설명
Blend	Blend	두 개의 이미지를 Merge 해 Blend 처리
Color	BrightnessContrast	밝기와 대조
	ColorOverlay	Overlay컬러를 적용한 소스 아이템의 컬러를 변경
	Colorize	HSL 컬러 공간에 색상 설정

	Desaturate	색상의 채도 감소
	GammaAdjust	원본 소스 아이템의 밝기 조절
	HueSaturation	HSL 컬러 공간에 원본 아이템 컬러를 조절
	LevelAdjust	RGBA 컬러 공간에 색상 레벨 조정
Gradient	ConicalGradient	두 개 이상의 색상으로 매끄럽게 혼합. 색상은 지정된 각도에서 시작하여 360도 까지 표시.
	LinearGradient	두 개 이상의 색상으로 매끄럽게 혼합. 색상은 지정된 시정부터 끝점까지 표시
	RadialGradient	두 개 이상의 색상으로 매끄럽게 혼합. 색상은 중간에서 시작하여 테두리 끝점까지 표시
Distortion	Displace	변위 매핑(displacement map)에 따라 소스 아이템의 픽셀을 이동
DropShadow	DropShadow	소스 이미지에 그림자 효과
	InnerShadow	소스 이미지 안쪽에 흐릿한 효과
Blur	FastBlur	하나 이상의 소스 항목에 흐림 효과를 적용
	GaussianBlur	더 높은 품질의 흐림 효과를 적용
	MaskedBlur	소스의 일부 부분이 다른 부분보다 흐려 지도록 maskSource를 사용하여 각 픽셀에 대해 흐림 강도를 제어
	RecursiveBlur	재귀 피드백 루프를 사용하여 소스를 여러 번 흐리게 해 이미지를 부드럽게 처리
Motion Blur	DirectionBlur	특정 방향에 Blur 적용
	RadialBlur	아이템 중심점을 주변에 원의 방향으로 Blur 적용
	ZoomBlur	중심점을 기준으로 Blur 적용
Glow	Glow	원본 이미지에 색상과 흐릿한 효과를 생성하고 원본 이미지에서 어두운 면 영역을 강조하기 위해 사용
	RectangularGlow	사각형 영역에 색상과 흐릿한 효과를 생성

		고 어두운 면 영역을 강조하기 위해 사용
Mask	OpacityMask	다른 아이템을 사용해 소스 이미지를 마스크 적용
	ThresholdMask	원본 아이템에 마스크를 적용하고 임계 영역을 적용

- Blend Effect

Blend 는 Source 이미지와 Foreground Source 이미지를 혼합한 Effect 를 적용할 수 있다. 렌더링의 성능을 향상시키기 위해 Cache를 사용할 수 있다. Cache 를 사용하기 위해서 cached 프로퍼티를 true 로 설정하면 된다.



Source



Foreground Source



Result

```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true; width: 300; height: 200
    Item {
        width: parent.width
        height: parent.height

        Image {
            id: bgc
            source: "images/background.png"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true
            visible: false
        }

        Image {
            id: butterfly
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
        source: "images/butterfly.png"
        sourceSize: Qt.size(parent.width, parent.height)
        smooth: true
        visible: false
    }

    Blend {
        anchors.fill: bgc
        source: bgc
        foregroundSource: butterfly
        mode: "average"
        cached: true
    }
}
}
```

### ● BrightnessContrast

BrightnessContrast 는 Source 이미지의 Brightness 와 Contrast를 조정할 수 있다.



Source



brightness: 0.5  
contrast: 0.5

```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true; width: 300; height: 200

    Item {
        width: parent.width
        height: parent.height

        Image {
            id: bgc
            source: "images/background.png"
```

예수님은 당신을 사랑합니다.

```
sourceSize: Qt.size(parent.width, parent.height)
smooth: true
visible: false
}

BrightnessContrast {
    anchors.fill: bgc
    source: bgc
    brightness: 0.5
    contrast: 0.5
}
}
}
```

- ColorOverlay

ColorOverlay 는 Source 이미지 상에 Colorize 효과를 설정할 수 있다.



Source



color: "#8000ff00"

```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true; width: 300; height: 300

    Image {
        id: bug
        source: "images/butterfly.png"
        sourceSize: Qt.size(parent.width, parent.height)
        smooth: true
        visible: false
    }
}
```

예수님은 당신을 사랑합니다.

```
ColorOverlay {  
    anchors.fill: bug  
    source: bug  
    color: "#8000ff00"  
}  
}
```

- Colorize

Colorize Effect는 Source 이미지 아이템의 HSL Color 공간에 Color를 설정할 수 있다.



Source

hue: 0.0  
saturation: 0.5  
lightness: -0.2

```
import QtQuick  
import QtQuick.Window  
import Qt5Compat.GraphicalEffects  
  
Window {  
    visible: true; width: 300; height: 200  
    Item {  
        width: parent.width  
        height: parent.height  
  
        Image {  
            id: bgc  
            source: "images/background.png"  
            sourceSize: Qt.size(parent.width, parent.height)  
            smooth: true  
            visible: false  
        }  
  
        Colorize {  
            anchors.fill: bgc
```

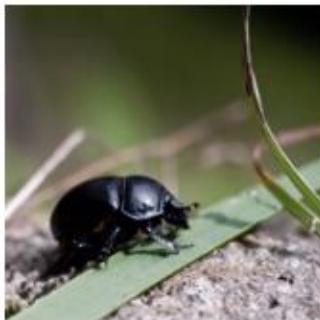
```
        source: bgc
        hue: 0.0
        saturation: 0.5
        lightness: -0.2
    }
}
}
```

- Desaturate

소스 이미지의 RGB값의 포화도(desaturation)를 저하시키는 기능 제공한다. 예를 들어 흑백으로 표시하는 것과 같이 표시할 수 있다.



Source



desaturation: 0.5



desaturation: 1.0

```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true; width: 300; height: 300
    Item {
        width: parent.width
        height: parent.height

        Image {
            id: bug
            source: "images/bug.jpg"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true
            visible: false
        }
    }
}
```

```
        Desaturate {  
            anchors.fill: bug  
            source: bug  
            desaturation: 1.0  
        }  
    }  
}
```

- GammaAdjust

GammaAdjust 는 power-law expression 과 같이 미리 정의된 Curve에 따라 각 픽셀을 적용 할 수 있다. 즉 감마 값을 이용해 Effect를 적용할 수 있다.



Source

Gamma: 2.0

```
import QtQuick  
import QtQuick.Window  
import Qt5Compat.GraphicalEffects  
  
Window {  
    visible: true; width: 300; height: 200  
    Item {  
        width: parent.width  
        height: parent.height  
  
        Image {  
            id: bgc; source: "images/background.png"  
            sourceSize: Qt.size(parent.width, parent.height)  
            smooth: true  
            visible: false  
        }  
  
        GammaAdjust {  
            anchors.fill: bgc  
            source: bgc  
            gamma: 2.0  
        }  
    }  
}
```

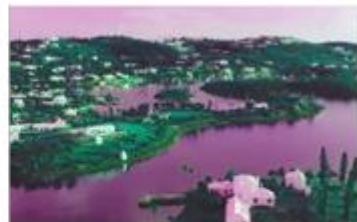
```
        }  
    }  
}
```

- HueSaturation

HueSaturation은 Colorize Effect와 비슷하다. Colorize와 차이점으로 hue 와 saturation 프로퍼티를 지정할 수 있다.



Source

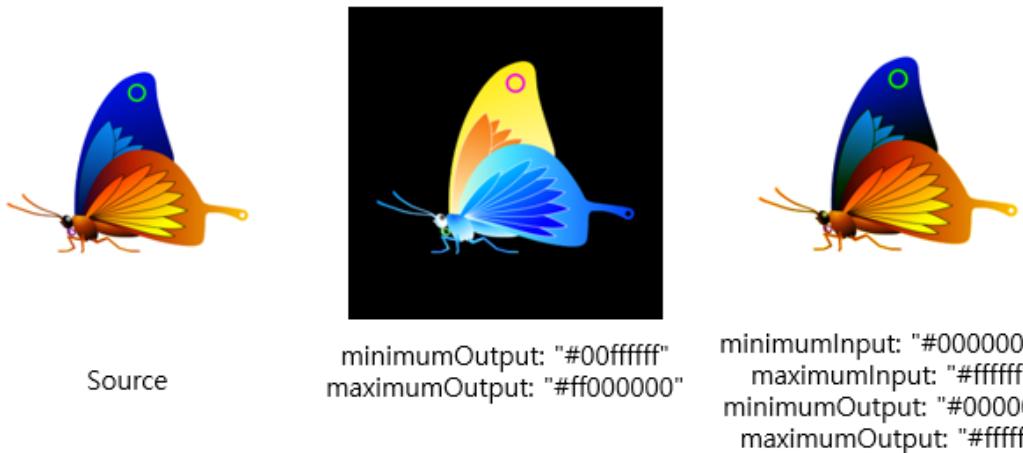


hue: 0.3  
saturation: 0  
lightness: 0

```
import QtQuick  
import QtQuick.Window  
import Qt5Compat.GraphicalEffects  
  
Window {  
    visible: true; width: 300; height: 200  
    Item {  
        width: parent.width  
        height: parent.height  
  
        Image {  
            id: bgc; source: "images/background.png"  
            sourceSize: Qt.size(parent.width, parent.height)  
            smooth: true; visible: false  
        }  
        HueSaturation {  
            anchors.fill: bgc; source: bgc  
            hue: 0.3  
            saturation: 0  
            lightness: 0  
        }  
    }  
}
```

- LevelAdjust

LevelAdjust 는 Source 이미지의 각 Color 채널을 위해 색상을 분리하기 위한 Effect를 적용할 수 있다.



```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true; width: 300; height: 300
    Item {
        width: parent.width
        height: parent.height

        Image {
            id: butterfly; source: "images/butterfly.png"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true; visible: false
        }

        LevelAdjust {
            anchors.fill: butterfly
            source: butterfly
            minimumInput: "#00000070"
            maximumInput: "#ffffff"
            minimumOutput: "#000000"
            maximumOutput: "#ffffff"
        }
    }
}
```

```
}
```

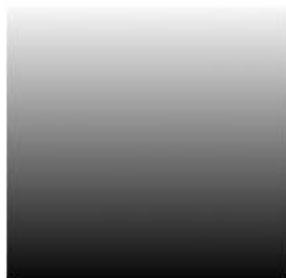
- Gradient

Gradient는 두 개 이상의 색상으로 매끄럽게 혼합해 표시하는 Effect를 제공 한다. ConicalGradient 는 지정된 각도에서 시작하여 360도 까지 표시한다.

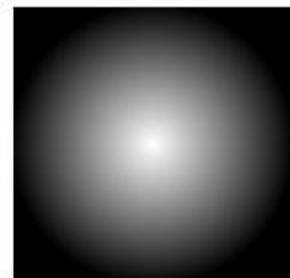
LinearGradient 는 색상을 지정된 시점부터 끝나는 지점까지 표시한다. RadialGradient는 중간에서 시작하여 테두리 끝점까지 표시한다.



ConicalGradient



LinearGradient



RadialGradient

```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true; width: 600; height: 200

    Row
    {
        ConicalGradient {
            width: 200; height: 200
            angle: 0.0
            gradient: Gradient {
                GradientStop { position: 0.0; color: "white" }
                GradientStop { position: 1.0; color: "black" }
            }
        }

        LinearGradient {
            width: 200; height: 200
            start: Qt.point(0, 0)
```

```
end: Qt.point(0, 300)
gradient: Gradient {
    GradientStop { position: 0.0; color: "white" }
    GradientStop { position: 1.0; color: "black" }
}
}

RadialGradient {
    width: 200; height: 200
    gradient: Gradient {
        GradientStop { position: 0.0; color: "white" }
        GradientStop { position: 0.5; color: "black" }
    }
}
}
```

- Displace

Displace 는 변위 매핑(displacement map)에 따라 소스 이미지 아이템의 픽셀을 이동시키는 기능을 제공한다.



```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true
    width: 300; height: 200
    Item {
        width: parent.width
        height: parent.height

        Image {
```

예수님은 당신을 사랑합니다.

```
id: bgc
source: "images/background.png"
sourceSize: Qt.size(parent.width, parent.height)
smooth: true
visible: false
}

Rectangle {
    id: displacement
    color: Qt.rgba(0.5, 0.5, 1.0, 1.0)
    anchors.fill: parent
    visible: false

    Image {
        anchors.centerIn: parent
        source: "images/glass_normal.png"
        sourceSize: Qt.size(parent.width/2, parent.height/2)
        smooth: true
    }
}

Displace {
    anchors.fill: bgc
    source: bgc
    displacementSource: displacement
    displacement: 0.1
}
}
```

### ● DropShadow

DropShadow 는 Source 이미지에 그림자 Effect 를 지정하기 위한 기능을 제공한다.  
radius 프로퍼티는 그림자의 부드러움 정도를 지정할 수 있다.

이 프로퍼티의 값이 클수록 그림자의 Edges 부분이 흐릿하고 일그러지게 표시할 수 있다. 또한 그림자의 컬러를 지정하기 위해 color 프로퍼티를 사용할 수 있다.

예수님은 당신을 사랑합니다.



Source



color: "#aa000000"  
radius: 8.0  
samples: 16  
horizontalOffset: 0  
verticalOffset: 20  
spread: 0

```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true; width: 300; height: 300
    Item {
        width: parent.width
        height: parent.height

        Image {
            width: 300; height: 300
            id: ball
            source: "images/ball.png"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true
            visible: false
        }

        DropShadow {
            anchors.fill: ball
            radius: 8.0
            samples: 16
            horizontalOffset: 0
            verticalOffset: 20
            spread: 0
            source: ball
            color: "#aa000000"
        }
    }
}
```

}

- FastBlur

FastBlur 는 Source 이미지 상에 흐림 효과를 적용할 수 있다. GaussianBlur 보다 낮은 Blur Effect를 제공하지만 랜더링 속도는 GaussianBlur 보다 빠르다.



Source



radius: 32

```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true
    width: 300; height: 200
    Item {
        width: parent.width
        height: parent.height

        Image {
            id: bgc; source: "images/background.png"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true
            visible: false
        }

        FastBlur {
            anchors.fill: bgc
            source: bgc
            radius: 32
        }
    }
}
```

- GaussianBlur

FastBlur 보다 높은 Quality를 보장하지만 FastBlur 보다 랜더링 속도는 느리다.



Source



deviation: 4  
radius: 8  
samples: 16

```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true
    width: 300; height: 200
    Item {
        width: parent.width
        height: parent.height

        Image {
            id: bgc
            source: "images/background.png"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true
            visible: false
        }

        GaussianBlur {
            anchors.fill: bgc
            source: bgc
            deviation: 4
            radius: 8
            samples: 16
        }
    }
}
```

- DirectionBlur

DirectionBlur 는 특정 방향으로 Blur Effect를 적용할 수 있다.



Source



angle: 90  
length: 32  
samples: 24

```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true
    width: 300
    height: 200

    Item {
        width: parent.width
        height: parent.height

        Image {
            id: bgc; source: "images/background.png"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true
            visible: false
        }

        DirectionalBlur {
            anchors.fill: bgc; source: bgc
            angle: 90
            length: 32
            samples: 24
        }
    }
}
```

예수님은 당신을 사랑합니다.

- RadialBlur

Source 이미지의 중심으로 회전한 것과 같은 Blur Effect를 적용할 수 있다.



Source

samples: 24  
angle: 30

```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true; width: 300; height: 200

    Item {
        width: parent.width
        height: parent.height

        Image {
            id: bgc; source: "images/background.png"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true; visible: false
        }

        RadialBlur {
            anchors.fill: bgc
            source: bgc
            samples: 24
            angle: 30
        }
    }
}
```

- Glow

Source 이미지에 색상과 흐릿한 효과를 생성하고 어두운 면 영역을 강조하기 위해서 사용한다.



Source



Result

```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true; width: 300; height: 300
    Rectangle {
        anchors.fill: parent
        color: "black"
    }

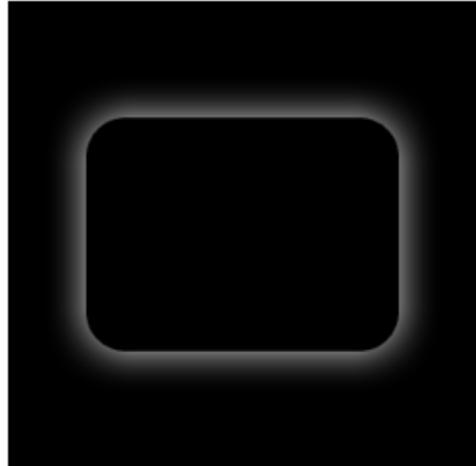
    Image {
        id: ball
        source: "images/ball.png"
        sourceSize: Qt.size(parent.width, parent.height)
        smooth: true
        visible: false
    }
    Glow {
        anchors.fill: ball
        radius: 30
        samples: 16
        color: "green"
        source: ball
    }
}
```

- **RectangularGlow**

Rectangle 영역에 색상을 지정하고 Blur를 생성한다. 그리고 Rectangle 아이템을 위치시키는 형태로 Glow Effect를 제공한다.



Source



Result

```
import QtQuick
import QtQuick.Window
import Qt5Compat.GraphicalEffects

Window {
    visible: true; width: 300; height: 300

    Rectangle {
        id: background; anchors.fill: parent; color: "black"
    }

    RectangularGlow {
        id: effect
        anchors.fill: rect
        glowRadius: 10
        spread: 0.2
        color: "white"
        cornerRadius: rect.radius + glowRadius
    }

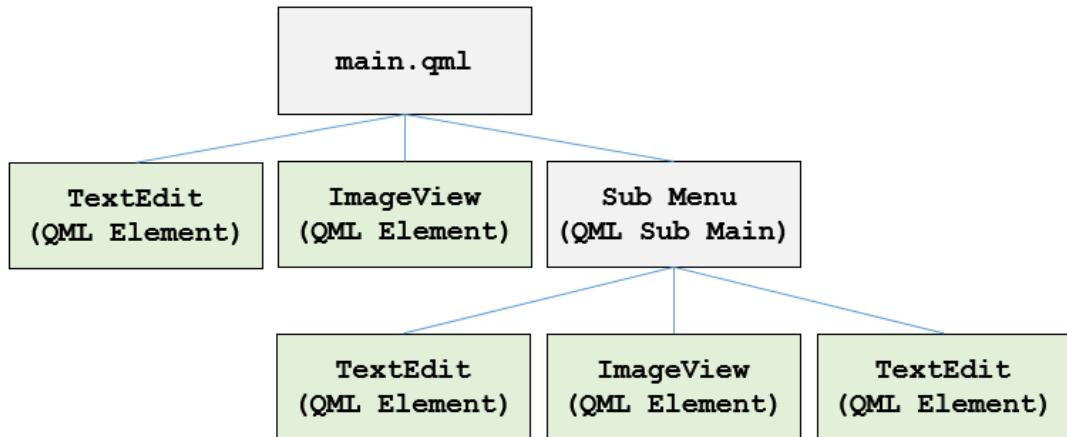
    Rectangle {
        id: rect
        color: "black"
        anchors.centerIn: parent
    }
}
```

예수님은 당신을 사랑합니다.

```
width: Math.round(parent.width / 1.5)
height: Math.round(parent.height / 2)
radius: 25
}
}
```

## 2.7. Module programming

C++에서 자주 사용되는 클래스를 라이브러리로 만들어 사용하는 것처럼 QML에서 자주 사용하는 사용자 정의 타입을 라이브러리처럼 재 사용할 수 있도록 아래 그림에서 보는 것과 같이 모듈화가 가능하다.



- 사용자 정의 타입

사용자가 정의한 타입을 별도의 QML파일로 작성해 다른 QML에서 불러올 수 있다. 예를 들어 main.qml 과 LineEdit.qml QML 소스코드 파일이 있다고 가정해 보자. main.qml에서 LineEdit.qml QML을 호출해 사용하기 위해 파일의 확장자를 제외한 파일명인 "LineEdit" 명이 호출할 수 있는 모듈 명의 이름으로 사용된다. 다음 예제는 main.qml에서 LineEdit.qml 파일을 호출하는 예제이다.

아래 예제 QML 소스코드는 Main.qml 이다.

```

import QtQuick
import QtQuick.Window

Window {
    visible: true; width: 400; height: 100
    color: "lightblue"

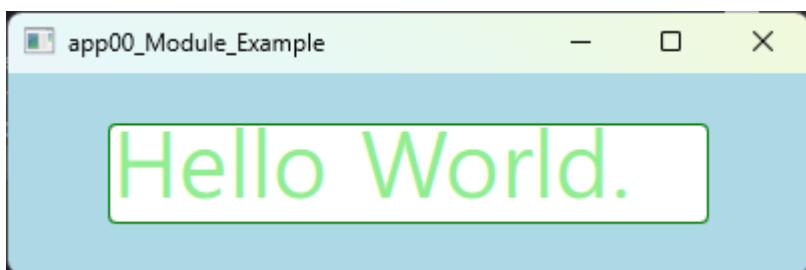
    LineEdit
    {
        anchors.horizontalCenter: parent.horizontalCenter
    }
}
  
```

예수님은 당신을 사랑합니다.

```
anchors.verticalCenter: parent.verticalCenter  
width: 300; height: 50  
}  
}
```

다음 예제는 LineEdit.qml 이다.

```
import QtQuick  
  
Rectangle  
{  
    border.color: "green";  
    color: "white"; radius: 4; smooth: true  
  
    TextInput {  
        anchors.fill: parent  
        anchors.margins: 2  
        text: "Hello World."  
        color: focus ? "black" : "lightgreen"  
        font.pixelSize: parent.height - 4  
    }  
}
```



### ● 사용자 정의 Property

기본적으로 제공되는 QML 타입에서 사용하는 프로퍼티 외에 직접 QML 타입 내에서 사용하는 프로퍼티를 정의 할 수 있다.

```
Syntax: property <type> <name>[: <value>]
```

다음은 사용자 정의 Property 사용 예이다.

```
property string message: "qt-dev.com"  
property int num: 123
```

예수님은 당신을 사랑합니다.

```
property real numfloat: 123.456
property bool boolcondi: true
property url address: "http://www.qt-dev.com"
```

다음 예제 소스는 사용자 정의 프로퍼티를 사용한 예제이다. 그리고 아래 예제의 소스 코드는 new\_main.qml 이다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true; width: 400; height: 120
    color: "lightblue"

    NewLineEdit {
        id: lineEdit
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.top: parent.top
        anchors.topMargin: 16
        width: 300; height: 50
    }

    Text {
        anchors.top: lineEdit.bottom
        anchors.topMargin: 12
        anchors.left: parent.left
        anchors.leftMargin: 16
        font.pixelSize: 20
        text: "<b>Summary:</b> " + lineEdit.text
    }
}
```

아래 예제 소스코드는 NewLineEdit.qml 이다.

```
import QtQuick

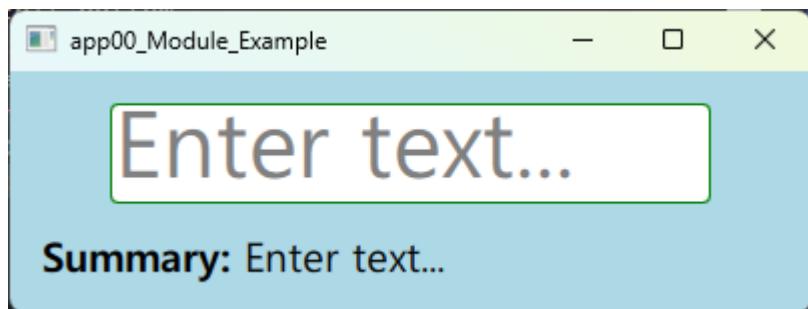
Rectangle {
    border.color: "green"
    color: "white"
    radius: 4; smooth: true

    TextInput {
        id: textView
```

예수님은 당신을 사랑합니다.

```
anchors.fill: parent
anchors.margins: 2
text: "Enter text..."
color: focus ? "black" : "gray"
font.pixelSize: parent.height - 4
}

property string text: textInput.text
}
```



- 사용자 정의 시그널(Signal)

Qt/C++에서 사용자 정의 Signal 과 Slot처럼 QML에서 사용자 정의 Signal과 Slot을 사용할 수 있도록 signal 키워드를 제공한다. 다음 예제는 signal 사용하기 위한 Syntax 구조이다.

```
Signal syntax: signal <name>[(<type> <value>, ...)]
```

- ✓ 사용자 정의 시그널을 정의 하기 위한 예제

```
MouseArea {
    anchors.fill: checkImage
    onClicked: if (parent.state === "checked") {
        parent.state = "unchecked";
        parent.myChecked(false);
    } else {
        parent.state = "checked";
        parent.myChecked(true);
    }
}
```

예수님은 당신을 사랑합니다.

```
...  
signal myChecked(bool checkValue)
```

- ✓ 시그널과 연결된 Slot 예

```
...  
onMyChecked: checkValue ? parent.color = "red": parent.color = "lightblue"  
...
```

위의 예에서 보는 것과 같이 Signal 을 정의하는 곳에서는 시그널 이름과 시그널에서 전달할 Arguments(함수 인자)를 정의하면 된다. 만약 Argument 가 없다면 시그널 이름만 명시하면 된다. 그런 다음 Slot 에서는 시그널 이름 앞에 첫 번째 알파벳은 대문자로 변경한다. 그리고 시그널 이름 앞에 “on” 붙여야 한다.

Signal : myChecked(bool checkValue)	→	Slot : onMyChecked
-------------------------------------	---	--------------------

이번에는 사용자 정의 시그널을 사용해 CheckBox를 구현해 보도록 하자. QML 파일은 checked\_main.qml 파일이 “content” 디렉토리 하위에 있는 NewCheckBox.qml을 호출해 사용하는 예제이다.

checked\_main.qml 파일 먼저 작성해 보도록 하자.

```
import QtQuick  
import QtQuick.Window  
import "content"  
  
Window {  
    id: root  
    visible: true; width: 250; height: 100  
    color: "lightblue"  
  
    NewCheckBox {  
        anchors.horizontalCenter: parent.horizontalCenter  
        anchors.verticalCenter: parent.verticalCenter  
  
        onMyChecked: checkValue ? root.color = "red"  
                      : root.color = "lightblue"  
    }  
}
```

이번에는 content 라는 디렉토리를 만들고 그 안에 NewCheckBox.qml 파일을 아래와

예수님은 당신을 사랑합니다.

같이 작성한다.

```
import QtQuick

Item {
    width: textItem.width + checkImage.width

    Image {
        id: checkImage
        anchors.left: parent.left
        anchors.verticalCenter: parent.verticalCenter
    }

    Text {
        id: textItem
        anchors.left: checkImage.right
        anchors.leftMargin: 4
        anchors.verticalCenter: checkImage.verticalCenter
        text: "Option"
        font.pixelSize: checkImage.height - 4
    }

    states: [
        State {
            name: "checked"
            PropertyChanges {
                target: checkImage;
                source: "../images/checked.svg"
            }
        },
        State {
            name: "unchecked"
            PropertyChanges {
                target: checkImage;
                source: "../images/unchecked.svg"
            }
        }
    ]
}

state: "unchecked"

MouseArea {
    anchors.fill: checkImage
```

예수님은 당신을 사랑합니다.

```
onClicked: if (parent.state === "checked") {  
    parent.state = "unchecked";  
    parent.myChecked(false);  
} else {  
    parent.state = "checked";  
    parent.myChecked(true);  
}  
  
signal myChecked(bool checkValue)  
}
```



- Alias

Alias는 QML에서 사용하는 모듈 명을 대신 사용할 가명 기능을 제공한다. 다음 예제는 Alias를 사용한 예제이다.

```
import QtQuick  
import QtQuick.Window  
import "content" as MyContent  
  
Window {  
    visible: true; width: 250; height: 100; color: "lightblue"  
  
    MyContent.NewCheckBox {  
        anchors.horizontalCenter: parent.horizontalCenter  
        anchors.verticalCenter: parent.verticalCenter  
    }  
}
```

Qt Quick에서 제공하는 공식 QML 모듈에서도 다음 예제와 같이 Alias를 사용할 수 있다.

```
import QtQuick as MyQt
```

예수님은 당신을 사랑합니다.

```
import QtQuick.Window

MyQt.Window {
    visible: true; width: 250; height: 100; color: "lightblue"

    MyQt.Rectangle {
        width: 250; height: 100; color: "lightblue"
        MyQt.Text {
            anchors.centerIn: parent
            text: "Hello Qt!"; font.pixelSize: 32
        }
    }
}
```

## 2.8. QML에서 JavaScript 사용

QML은 JSON, XML 또는 HTML과 같은 Syntax를 가지고 있다. 그렇기 때문에 JavaScript, C++과 같은 구조적인 Syntax를 제공하지 못한다. 예를 들어 QML은 If 문, 함수, 변수 치환 등과 같은 Syntax를 제공하지 않는다.

따라서 구조적인 프로그래밍을 QML에서 가능하게 하기 위해서 JavaScript를 QML과 함께 사용할 수 있다. 또한 QML내에서 JavaScript 파일을 모듈처럼 사용할 수 있다. 이번 장에서는 JavaScript를 QML에서 사용하는 방법에 대해서 알아보도록 하자.

- Property Binding

Property Binding은 다음 예제에서 보는 것과 같이 3항 연산자 문법을 이용해 color라는 프로퍼티(Property)에 값을 할당 할 수 있다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    title: qsTr("Hello World")

    id: colorbutton;
    width: 200;
    height: 80;
    color: mousearea.pressed ? "steelblue" : "lightsteelblue"
    MouseArea {
        id: mousearea; anchors.fill: parent
    }
}
```

다음 예제 QML 소스코드에서는 JavaScript 함수를 사용한 예이다. QML에서 제공하는 binding을 이용해 함수를 JavaScript 함수를 호출할 수 있다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true
```

```
id: colorbutton
width: 200; height: 80; color: "red"

MouseArea
{
    id: mousearea; anchors.fill: parent
}

Component.onCompleted:
{
    color = Qt.binding( function() {
        return mousearea.pressed ? "steelblue" : "lightsteelblue"
    } );
}
}
```

- Signal Handler 를 사용해 JavaScript Method 호출

QML에서 마우스 버튼 클릭 시 이벤트를 처리하기 위해서 MouseArea를 사용하였다. MouseArea에서 제공하는 onClicked 와 같이 미리 정의된 Slot을 호출해 아래와 같이 사용했다.

```
...
Rectangle {
    id: relay

    signal messageReceived(string person, string notice)

    Component.onCompleted: {
        relay.messageReceived.connect(sendToPost)
        relay.messageReceived.connect(sendToTelegraph)
        relay.messageReceived.connect(sendToEmail)
        relay.messageReceived("Tom", "Happy Birthday")
    }

    function sendToPost(person, notice) {
        console.log("Sending to post: " + person + ", " + notice)
    }
    function sendToTelegraph(person, notice) {
        console.log("Sending to telegraph: " + person + ", " + notice)
    }
    function sendToEmail(person, notice) {
```

예수님은 당신을 사랑합니다.

```
        console.log("Sending to email: " + person + ", " + notice)
    }
}
```

...

- QML 탑입에서 JavaScript 함수 사용

QML 탑입에서 JavaScript 함수를 사용하기 위해서 아래 예제와 같이 JavaScript 함수를 호출해 사용할 수 있다.

```
import QtQuick
import QtQuick.Window

Window {
    visible: true
    title: qsTr("Hello World")

    id: colorButton;

    width: 200;
    height: calculateHeight();

    color: mousearea.pressed ? "steelblue" : "lightsteelblue"

    MouseArea {
        id: mousearea; anchors.fill: parent
    }

    function calculateHeight()
    {
        return colorButton.width / 2;
    }
}
```

다음 예제는 JavaScript 함수의 인자를 사용한 예제 소스코드이다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 200; height: 200; visible: true
```

예수님은 당신을 사랑합니다.

```
MouseArea {  
    anchors.fill: parent  
    onClicked: label.moveTo(mouseX, mouseY)  
}  
  
Text {  
    id: label  
    function moveTo(newX, newY) {  
        label.x = newX;  
        label.y = newY;  
    }  
  
    text: "Move me!"  
}  
}
```

다음 예제는 MouseArea에서 JavaScript 함수를 호출하고 함수 결과를 리턴 하는 예제이다.

```
...  
Item {  
    function factorial(a) {  
        a = parseInt(a);  
        if (a <= 0)  
            return 1;  
        else  
            return a * factorial(a - 1);  
    }  
    MouseArea {  
        anchors.fill: parent  
        onClicked: console.log(factorial(10))  
    }  
}
```

### ● JavaScript 파일을 Import 해 사용

QML에서는 JavaScript 함수가 정의되어 있는 JavaScript 파일 (확장자가 .js인 파일)을 호출할 수 있다.

```
import QtQuick  
import QtQuick.Window
```

예수님은 당신을 사랑합니다.

```
import "script.js" as MyScript

Window {
    visible: true
    title: qsTr("Hello World")

    id: item; width: 200; height: 200

    MouseArea {
        id: mouseArea; anchors.fill: parent
    }
    Component.onCompleted: {
        mouseArea.clicked.connect(MyScript.jsFunction)
    }
}
```

위의 예제 소스코드에서 보는 것과 같이 마우스 버튼의 클릭 이벤트가 발생하면 아래와 같이 script.js 라는 JavaScript 파일에서 정의된 함수를 호출할 수 있다.

```
// script.js

function jsFunction()
{
    console.log("Called JavaScript function!")
}
```

라이브러리와 같이 JavaScript에서도 Pragma 키워드를 사용해 Library 와 같이 정의해 사용할 수 있다. 다음 예제는 pragma 키워드를 사용한 예제이다.

```
import QtQuick
import QtQuick.Window
import "script.js" as MyLib

Window {
    visible: true
    id: item; width: 500; height: 200

    Text {
        width: parent.width
        height: parent.height
        property int input: 17

        text: "The Number of " + input
    }
}
```

예수님은 당신을 사랑합니다.

```
        + " is: " + MyLib.factorial(input)
    }
}
```

다음 예제 소스코드는 script.js 소스코드이다.

```
.pragma library

var factorialCount = 0;
function factorial(a)
{
    a = parseInt(a);

    // factorial recursion
    if (a > 0)
        return a * factorial(a - 1);

    // shared state
    factorialCount += 1;

    // recursion base-case.
    return 1;
}

function factorialCallCount()
{
    return factorialCount;
}
```

QML에서는 JavaScript 파일을 호출하기 위해서 Qt.include( )를 아래 예제에서 보는 것과 같이 사용할 수 있다.

```
import QtQuick
import QtQuick.Window
import "script.js" as MyScript

Window {
    visible: true
    width: 100; height: 100
    MouseArea {
        anchors.fill: parent
        onClicked: {
            MyScript.showCalculations(10)
            console.log("Call from QML:", MyScript.factorial(10))
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
        }
    }
}
```

```
// script.js

Qt.include("factorial.js")

function showCalculations(value)
{
    console.log("Call factorial() from script.js:",
               factorial(value));
}
```

```
// factorial.js

function factorial(a)
{
    a = parseInt(a);

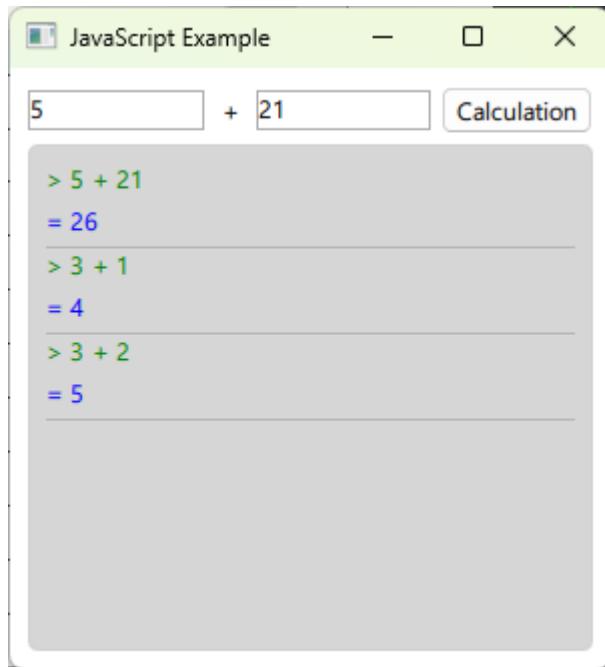
    if (a <= 0)
        return 1;

    else
        return a * factorial(a - 1);
}
```

위의 예제 JavaScript 파일인 facrotyal.js 를 script.js 에서 호출하기 위해서 Qt.include( ) 함수를 사용해 JavaScript 파일을 호출 할 수 있다.

### ● QML에서 JavaScript 사용 예제

이번 예제는 QML에서 자바스크립트 함수를 호출해 사용하기 위한 예제이다.



위의 그림에서 보는 것과 같이 2개의 값을 입력하고 [Calculation] 버튼을 클릭하면 2개의 값을 합한 결과를 위의 그림에서 보는 것과 같이 보여준다. Main.qml 파일을 아래와 같이 작성한다.

```
import QtQuick
import QtQuick.Window
import QtQuick.Controls
import QtQuick.Layouts

import "js/jslib.js" as JUtils

Window {
    id: root; visible: true; width: 300; height: 300
    title: qsTr("JavaScript Example")

    property string val: ""

    ColumnLayout {
        anchors.fill: parent; anchors.margins: 9
        RowLayout {
            Layout.fillWidth: true
            TextField {
                id: input1;
                Layout.fillWidth: true;
```

```
        focus: true
    }
    Label {
        text: " +
    }
    TextField {
        id: input2;
        Layout.fillWidth: true;
        focus: true
    }
    Button {
        text: " Calculation "
        onClicked: {
            root.val = input1.text.trim() + "," + input2.text.trim();
            root.jsCall(val);
        }
    }
}
Item {
    Layout.fillWidth: true; Layout.fillHeight: true
    Rectangle {
        anchors.fill: parent
        color: '#333'; opacity: 0.2; radius: 5
        border.color: Qt.darker(color)

    }

    ScrollView {
        id: scrollView
        anchors.fill: parent; anchors.margins: 9
        ListView {
            id: resultView
            model: ListModel {
                id: outputModel
            }
            delegate: ColumnLayout {
                width: ListView.view.width
                Label {
                    Layout.fillWidth: true
                    color: 'green'
                    text: "> " + model.expression
                }
            }
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
Label {
    Layout.fillWidth: true
    color: 'blue'
    text: "= " + model.result
}
Rectangle {
    height: 1
    Layout.fillWidth: true
    color: '#333'
    opacity: 0.2
}
}
}
}

function jsCall(exp) {
    var data = JUtils.addCall(exp);
    console.log(data);

    outputModel.insert(0, data)
}
}
```

다음 예제는 위의 QML에서 호출한 JavaScript 파일의 소스코드이다.

```
.pragma library

function addCall(msg)
{
    var str = msg.toString();
    var res = str.split(",");
    var src1 = parseInt(res[0]);
    var src2 = parseInt(res[1]);
    var result = src1 + src2;

    var data = {
        expression : result
    }
}
```

예수님은 당신을 사랑합니다.

```
data.expression = src1 + ' + ' + src2;  
data.result = result;  
  
    return data;  
}
```

위의 예제 소스코드는 00\_JavaScript\_Example 디렉토리를 참조하면 된다.

## 2.9. Dialog

QML에서 다이얼로그를 사용하기 위해서는 다음과 같이 Dialogs를 import 해야 한다.

```
import QtQuick.Dialogs
```

### ● Dialog 타입

Qt Quick은 QML에서 다이얼로그를 사용하기 위해서 Dialog 타입을 제공한다. 다음 예제는 Dialog를 사용한 예제이다.

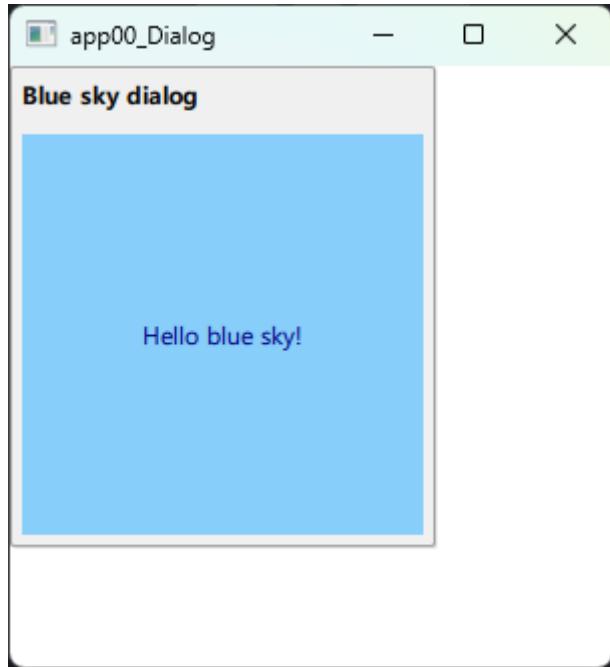
```
import QtQuick
import QtQuick.Controls
import QtQuick.Dialogs

Window {
    width: 300; height: 300; visible: true

    Button {
        text: "Dialog loading"
        anchors.centerIn: parent
        onClicked: { myDial.visible = true }
    }

    Dialog {
        id: myDial; visible: false
        title: "Blue sky dialog"

        contentItem: Rectangle {
            color: "lightskyblue"
            implicitWidth: 200; implicitHeight: 200
            Text {
                text: "Hello blue sky!"; color: "navy"
                anchors.centerIn: parent
            }
        }
    }
}
```



### ● ColorDialog 타입

ColorDialog 타입은 사용자가 Color를 선택할 수 있는ダイ얼로그를 제공한다.

```
import QtQuick
import QtQuick.Controls
import QtQuick.Dialogs

Window {
    width: 500; height: 400; visible: true
    Button {
        text: "ColorDialog loading";
        anchors.centerIn: parent
        onClicked: {
            colorDialog.visible = true
        }
    }

    ColorDialog {
        id: colorDialog
        title: "Please choose a color"
        onAccepted: {
            console.log("You chose: " + colorDialog.selectedColor)
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
onRejected: {
    console.log("Canceled")
}

Component.onCompleted: visible = false
}

}
```



### ● FileDialog 타입

FileDialog 타입은 사용자에게 파일을 선택할 수 있는 다이얼로그 기능을 제공한다.

```
import QtQuick
import QtQuick.Controls
import QtQuick.Dialogs
```

예수님은 당신을 사랑합니다.

```
Window {
    width: 300; height: 300; visible: true

    Button {
        text: "FileDialog loading"
        anchors.centerIn: parent
        onClicked: {
            fileDialog.visible = true
        }
    }

    FileDialog
    {
        id: fileDialog
        title: "Please choose a file"
        onAccepted: {
            console.log("You chose: " + fileDialog.selectedFile)
        }
        onRejected: {
            console.log("Canceled")
        }

        Component.onCompleted: visible = false
    }
}
```

### ● **FontDialog** 타입

FontDialog 타입은 사용자에게 폰트를 선택할 수 있는ダイ얼로그 기능을 제공한다.

```
import QtQuick
import QtQuick.Controls
import QtQuick.Dialogs

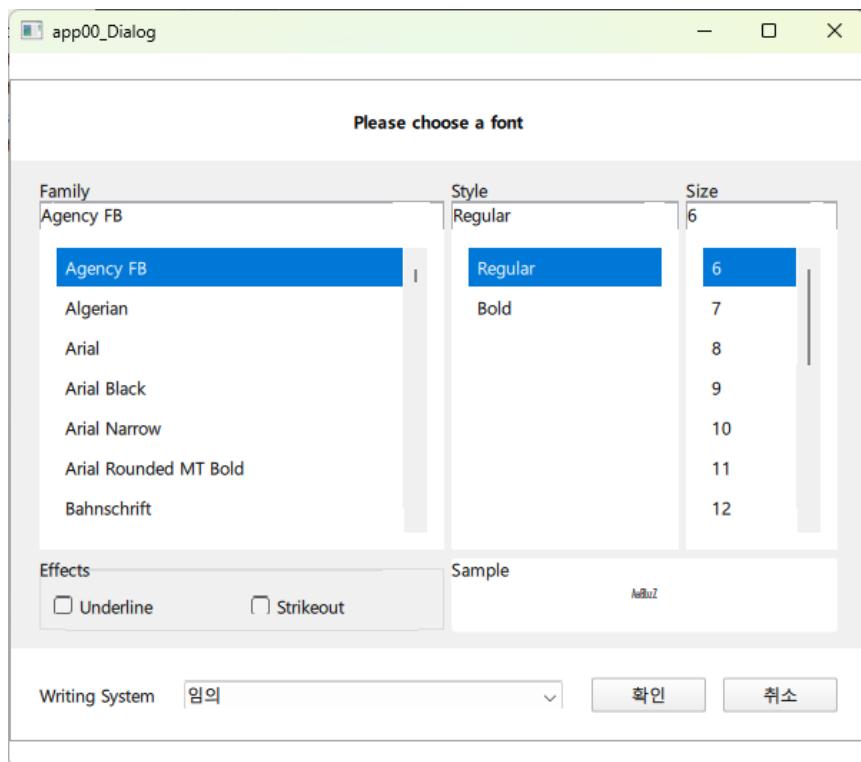
Window {
    width: 600; height: 500; visible: true
    Button {
        text: "FontDialog loading";
        anchors.centerIn: parent
        onClicked: {
            fontDialog.visible = true
        }
    }
```

예수님은 당신을 사랑합니다.

```
}

FontDialog {
    id: fontDialog
    title: "Please choose a font"

    onAccepted: {
        console.log("You chose: " + fontDialog.selectedFont)
    }
    onRejected: {
        console.log("Canceled")
    }
    Component.onCompleted: visible = true
}
}
```



예수님은 당신을 사랑합니다.

### ● **MessageDialog 타입**

MessageDialog 는 사용자에게 정보를 전달하기 위한 목적으로 사용한다.

```
import QtQuick
import QtQuick.Controls
import QtQuick.Dialogs

Window {
    width: 500; height: 400; visible: true

    Button {
        text: "MessageDialog loading";
        anchors.centerIn: parent
        onClicked: {
            dialog.visible = true
        }
    }

    MessageDialog {
        id: dialog
        text: qsTr("The document has been modified.")
        informativeText: qsTr("Do you want to save your changes?")
        buttons: MessageDialog.Ok | MessageDialog.Cancel

        onButtonClicked: function (button, role) {
            switch (button) {
                case MessageDialog.Ok:
                    document.save()
                    break;
            }
        }
    }
}
```

## 2.10. Layout

QML에서 레이아웃을 사용하기 위해서는 다음과 같이 QML 파일에 import 해야 한다.

```
import QtQuick.Layouts
```

Qt 제공하는 레이아웃의 종류는 다음과 같다.

Layout	설명
RowLayout	하나의 Row 상에서 타입을 수평으로 배치
ColumnLayout	수직 방향으로 타입을 배치
GridLayout	셀과 같이 타입을 배치
Layout	GridLayout, RowLayout, ColumnLayout 중 하나를 지정해 사용 가능.

- RowLayout

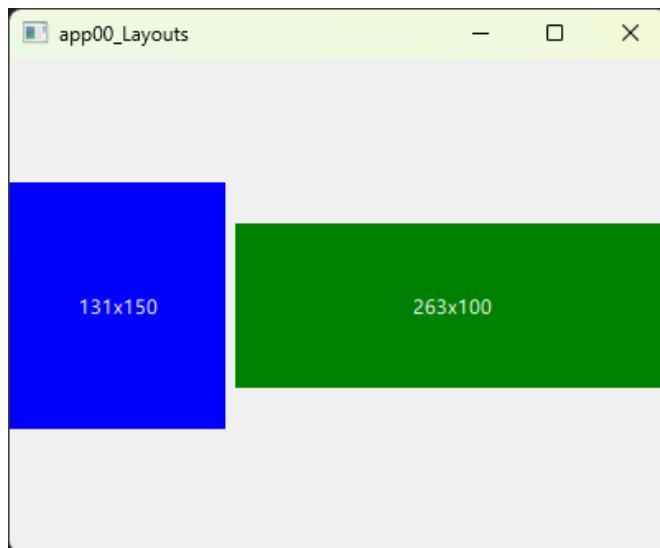
```
import QtQuick
import QtQuick.Controls
import QtQuick.Layouts

ApplicationWindow {
    width: 400; height: 300; visible: true

    RowLayout {
        id: layout
        anchors.fill: parent
        spacing: 6
        Rectangle {
            color: 'blue'
            Layout.fillWidth: true
            Layout.minimumWidth: 50
            Layout.preferredWidth: 100
            Layout.maximumWidth: 300
            Layout.minimumHeight: 150
            Text {
                anchors.centerIn: parent; color: "white"
                text: parent.width + 'x' + parent.height
            }
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
    }
    Rectangle {
        color: 'green'
        Layout.fillWidth: true
        Layout.minimumWidth: 100
        Layout.preferredWidth: 200
        Layout.preferredHeight: 100
        Text {
            anchors.centerIn: parent; color: "white"
            text: parent.width + 'x' + parent.height
        }
    }
}
```



- ColumnLayout

```
import QtQuick
import QtQuick.Controls
import QtQuick.Layouts

ApplicationWindow {
    width: 400; height: 300; visible: true

    ColumnLayout{
        spacing: 2
```

```
Rectangle {  
    Layout.alignment: Qt.AlignCenter  
    color: "red"  
    Layout.preferredWidth: 40  
    Layout.preferredHeight: 40  
}  
  
Rectangle {  
    Layout.alignment: Qt.AlignRight  
    color: "green"  
    Layout.preferredWidth: 40  
    Layout.preferredHeight: 70  
}  
  
Rectangle {  
    Layout.alignment: Qt.AlignBottom  
    Layout.fillHeight: true  
    color: "blue"  
    Layout.preferredWidth: 70  
    Layout.preferredHeight: 40  
}  
}  
}
```



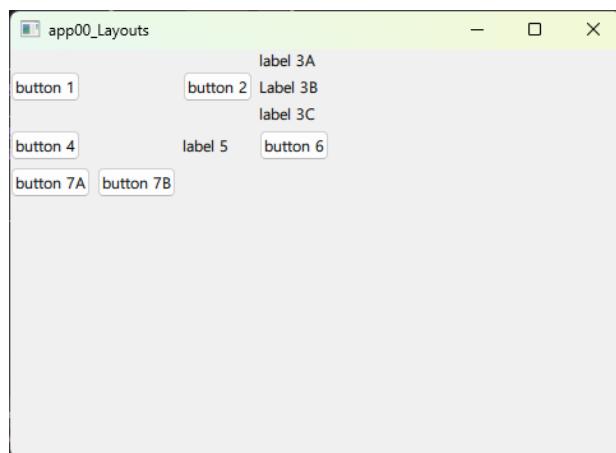
예수님은 당신을 사랑합니다.

- GridLayout

```
import QtQuick
import QtQuick.Controls
import QtQuick.Layouts

Window {
    id: myWindow; width: 480; height: 320; visible: true

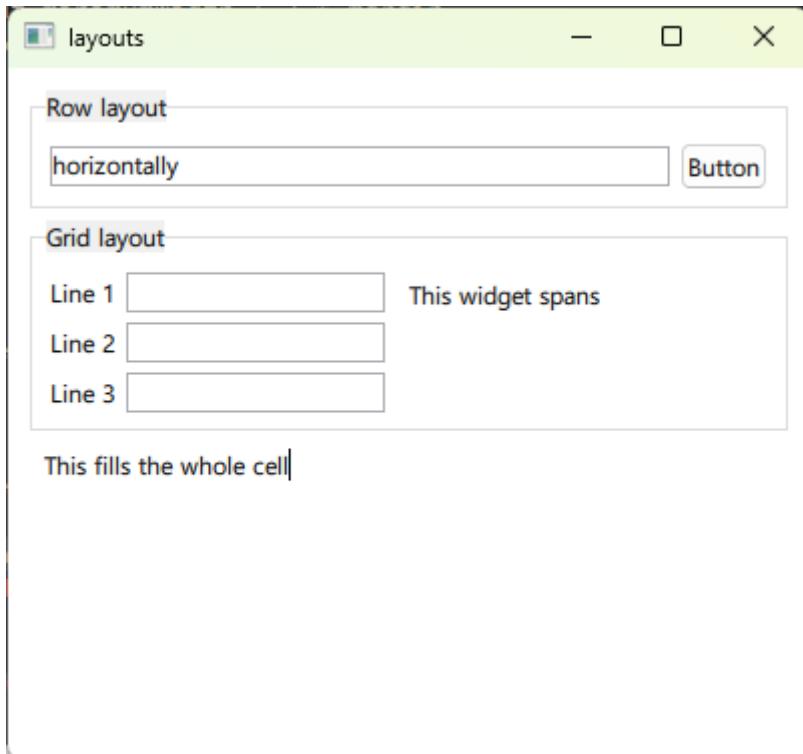
    GridLayout{
        columns: 3
        Button { text: "button 1"}
        Button { text: "button 2"}
        ColumnLayout{
            Label { text: "label 3A"}
            Label { text: "Label 3B"}
            Label { text: "label 3C"}
        }
        Button { text: "button 4"}
        Label{ text: "label 5"}
        Button { text: "button 6"}
        RowLayout{
            Button { text: "button 7A"}
            Button { text: "button 7B"}
        }
    }
}
```



예수님은 당신을 사랑합니다.

- Layout 예제

이번 예제는 Qt Quick에서 제공하는 Layout을 이용해 아래 그림에서 보는 것과 같이 타입을 배치하는 위젯을 구현해 보도록 하자.



이번 예제에서는 중첩된 Layout 구조로 ColumnLayout, RowLayout, GridLayout 그리고 Layout을 사용해 타입 배치하였다. 다음은 예제 소스코드이다.

```
import QtQuick
import QtQuick.Controls
import QtQuick.Layouts

Window
{
    width: 400; height: 300
    visible: true
    title: "layouts"
    property int margin: 11
    minimumWidth: mainLayout.Layout.minimumWidth + 2 * margin
    minimumHeight: mainLayout.Layout.minimumHeight + 2 * margin

    ColumnLayout {
        id: mainLayout
```

```
anchors.fill: parent
anchors.margins: margin
GroupBox {
    id: rowBoxr
    title: "Row layout"
    Layout.fillWidth: true

    RowLayout {
        id: rowLayout
        anchors.fill: parent
        TextField {
            text: "horizontally"
            Layout.fillWidth: true
        }
        Button {
            text: "Button"
        }
    }
}

GroupBox {
    id: gridView
    title: "Grid layout"
    Layout.fillWidth: true

    GridLayout {
        id: gridLayout
        rows: 3
        flow: GridLayout.TopToBottom
        anchors.fill: parent

        Label { text: "Line 1" }
        Label { text: "Line 2" }
        Label { text: "Line 3" }

        TextField { }
        TextField { }
        TextField { }

        TextArea {
            text: "This widget spans";
            Layout.rowSpan: 3
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
        Layout.fillHeight: true
        Layout.fillWidth: true
    }
}
TextArea {
    id: t3
    text: "This fills the whole cell"
    Layout.minimumHeight: 30
    Layout.fillHeight: true
    Layout.fillWidth: true
}
}
```

## 2.11. Type Positioning

이번 장에서는 여러 개의 타입 중에서 특정 위치의 타입의 정보를 얻기 위한 방법에 대해서 살펴보자.

예를 들어 QML에서 100개 이상의 Rectangle 타입이 있다고 가정해 보자. 만약 100개 이상의 타입 중에서 특정 조건에 부합하는 타입의 위치 정보를 얻기 위해서 QML에서 Positioner를 제공한다.

따라서 이번 장에서는 Positioner를 이용해 특정 타입의 위치 정보를 알아내는 방법에 대해서 살펴보도록 하자.

### ● Positioner

Positioner 아이템은 Column, Row, Grid 와 같은 Child 타입에서 특정 타입의 위치 정보를 알 수 있다. 특정 타입이 몇 번째 타입인지 index 프로퍼티로 확인할 수 있으며 isFirstItem 프로퍼티는 첫 번째 타입인지 그렇지 않은지 알 수 있다. 만약 첫 번째 타입이면 isFirstItem 프로퍼티는 true를 리턴 하고 그렇지 않으면 false를 리턴 한다. 다음은 Positioner를 사용한 예제이다.

```
import QtQuick
import QtQuick.Window

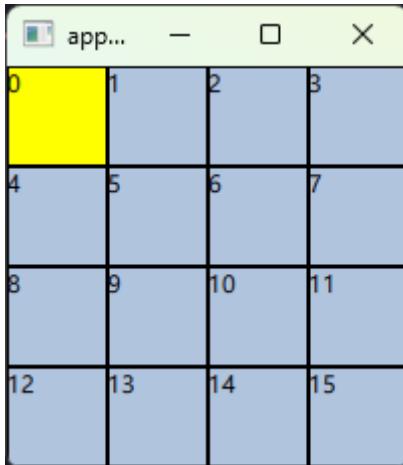
Window {
    visible: true; width: 200; height: 200

    Grid {
        Repeater {
            model: 16

            Rectangle {
                id: rect
                width: 50; height: 50
                border.width: 1
                color: Positioner.isFirstItem ? "yellow" : "lightsteelblue"

                Text { text: rect.Positioner.index }
            }
        }
    }
}
```

```
}
```



### ● Calendar 예제

이번 예제에서는 Grid 와 Repeater 를 사용해 달력을 구현해보자. 달력을 구현하기 위해서 날짜를 계산하기 위한 라이브러리를 JavaScript 에서 제공하는 클래스를 사용할 것이다.

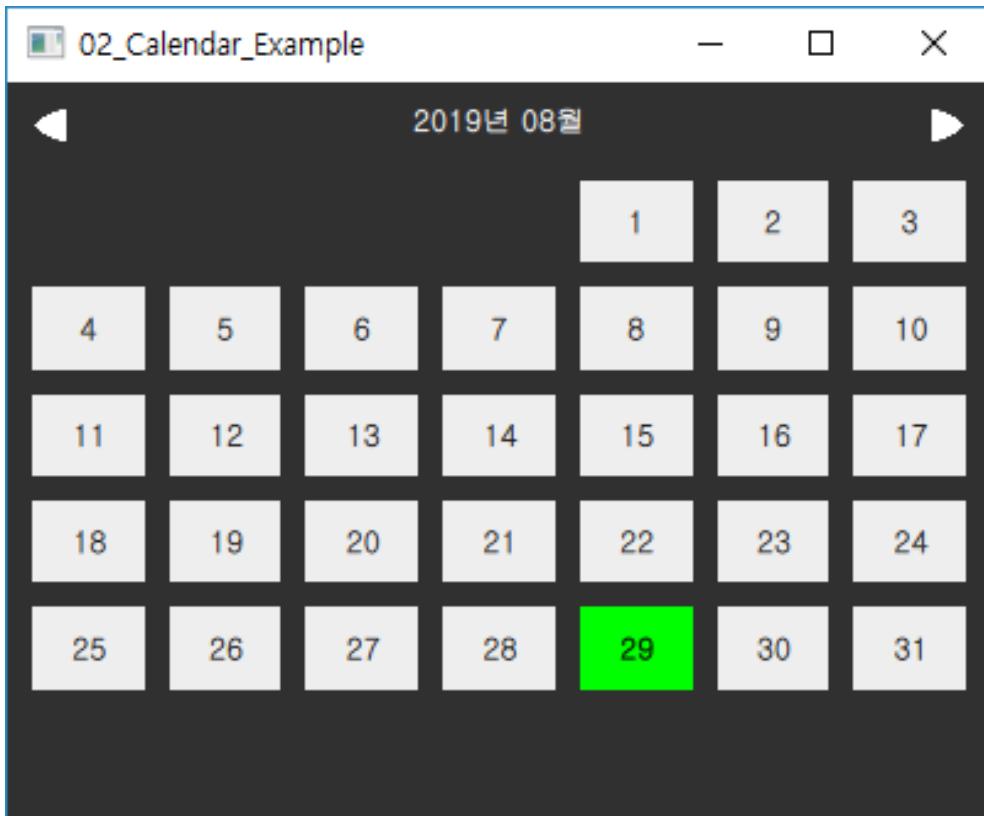
```
property date today: new Date()
property date showDate: new Date()

property int daysInMonth:
new Date(showDate.getFullYear(), showDate.getMonth() + 1, 0).getDate()

property int firstDay:
new Date(showDate.getFullYear(), showDate.getMonth(), 1).getDay()
```

daysInMonth 는 현재 월의 일수를 리턴 한다. firstDay 는 해당 년, 월, 일의 요일의 num 값을 출력한다.

Getday() 함수는 요일의 num 의 값(0~6) 중에서 하나를 리턴 한다. 예를 들어 화요일이면 2를 리턴 한다. 다음은 달력 기능이 완성된 예제의 실행 화면이다.



```
import QtQuick
import QtQuick.Window

Window {
    id: calendar; width: 400; height: 300; visible: true; color: "#303030"

    property date today: new Date()
    property date showDate: new Date()
    property int daysInMonth:
        new Date(showDate.getFullYear(), showDate.getMonth() + 1, 0).getDate()

    property int firstDay:
        new Date(showDate.getFullYear(), showDate.getMonth(), 1).getDay()

    Item {
        id: title
        anchors.top: parent.top
        anchors.topMargin: 10
        width: parent.width
```

예수님은 당신을 사랑합니다.

```
height: childrenRect.height

Image {
    source: "images/left.png"
    anchors.left: parent.left
    anchors.leftMargin: 10

    MouseArea {
        anchors.fill: parent
        onClicked: {
            var tYear = showDate.getFullYear()
            var tMonth = showDate.getMonth()
            showDate = new Date(tYear, tMonth, 0)
        }
    }
}

Text {
    color: "white"
    text: Qt.formatDateTime(showDate, "MM, yyyy")
    font.bold: true
    anchors.horizontalCenter: parent.horizontalCenter
}

Image {
    source: "images/right.png"
    anchors.right: parent.right
    anchors.rightMargin: 10

    MouseArea {
        anchors.fill: parent
        onClicked: {
            var tYear = showDate.getFullYear()
            var tMonth = showDate.getMonth()
            showDate = new Date(tYear, tMonth + 1, 1)
        }
    }
}

function isToday(index) {
    if (today.getFullYear() != showDate.getFullYear())

```

예수님은 당신을 사랑합니다.

```
        return false;
    if (today.getMonth() != showDate.getMonth())
        return false;

    return (index === today.getDate() - 1)
}

Item {
    id: dateLabels
    anchors.top: title.bottom
    anchors.left: parent.left
    anchors.right: parent.right
    anchors.margins: 10

    height: calendar.height - title.height - 20 - title.anchors.topMargin
    property int rows: 6

    Item {
        id: dateGrid
        width: parent.width
        anchors.top: parent.top
        anchors.topMargin: 5
        anchors.bottom: parent.bottom

        Grid {
            columns: 7
            rows: dateLabels.rows
            spacing: 10

            Repeater {
                model: firstDay + daysInMonth

                Rectangle {
                    color: {
                        if (index < firstDay)
                            calendar.color;
                        else
                            isToday(index - firstDay) ? "#00ff00": "#eeeeee";
                    }
                    width: (calendar.width - 20 - 60)/7
                    height: (dateGrid.height -
                        (dateLabels.rows - 1)*10)/dateLabels.rows
                }
            }
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
Text {  
    anchors.centerIn: parent  
    text: index + 1 - firstDay  
    opacity: (index < firstDay) ? 0 : 1  
    font.bold: isToday(index - firstDay)  
}  
}  
}  
}  
}  
}
```

이 예제의 소스코드는 01\_Calendar\_Example 디렉토리를 참조하면 된다.

## 2.12. Qt Quick Controls

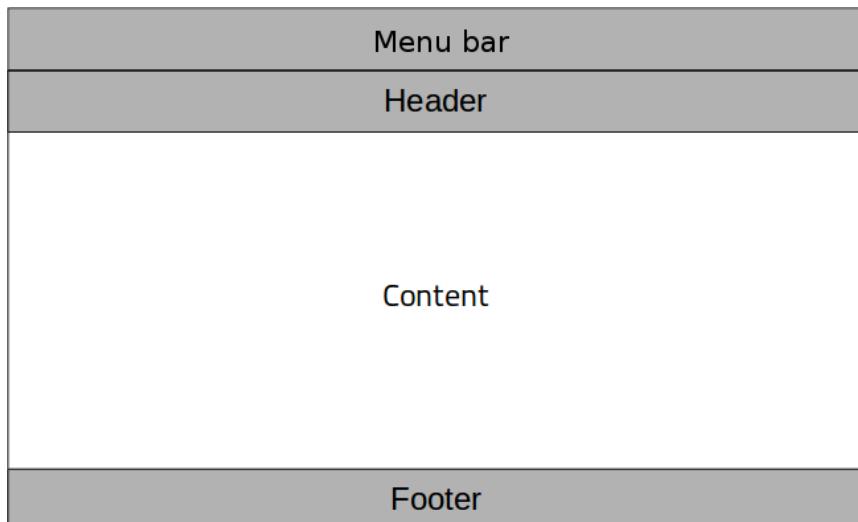
Qt Quick Controls 는 QML에서 GUI 구현에 필요한 기본적인 UI Controls를 제공한다. 예를 들어 Button, CheckBox, Tab, Combo, SpinBox 등 GUI 구현에 필요한 Type들을 제공한다.

Qt Quick Controls 는 Qt 5.7 에서 Release 되었다. 따라서 Qt Quick Controls 는 Qt 5.7 이상에서 사용할 수 있다. 그리고 Qt 5.X.X 버전을 사용하는 것과 Qt 6.5.X 버전과는 다소 다르다. 여기서는 Qt 6.5.X 이상 버전을 중심으로 설명하겠다. 만약 Qt 5.15.X 버전에서 Qt Quick Controls 를 참조하고자 한다면 이 문서의 2.0 이하 버전의 문서를 참조하면 된다.

그리고 여기서 설명하는 Controls는 자주 사용하는 Controls 들만을 언급하였다. 따라서 더 많은 Controls에 대해서는 Qt Assistant 또는 공식 도움말을 참조하면 된다.

- Application Window

메인 윈도우(가장 상위 Element) 의 MenuBar, Action, StatusBar, ToolBar 등과 기본적인 윈도우 GUI 를 사용하기 위해 제공하는 타입이다.



```
import QtQuick.Controls
```

예수님은 당신을 사랑합니다.

```
ApplicationWindow {  
    visible: true  
  
    menuBar: MenuBar {  
        // ...  
    }  
  
    header:ToolBar {  
        // ...  
    }  
  
    footer: RowLayout{  
        // ...  
    }  
  
    StackView {  
        anchors.fill: parent  
    }  
}
```

- Navigation 과 Views

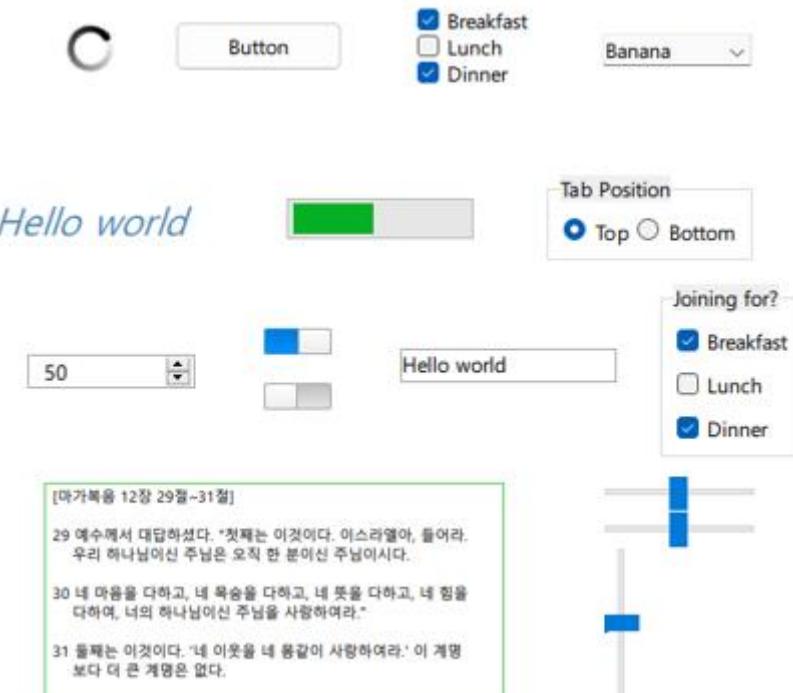
ScrollView, SplitView, TableView, TreeView 등과 같은 Navigation 과 View 기능의 타입을 제공한다.

QML타입	설명
ScrollView	QML 타입 내에 스크롤을 제공하는 View
SplitView	QML 타입 사이를 Splitter를 사용해 Drag 기능을 제공
StackView	Stack 처럼 여러 화면이 구성되어 UI를 구성할 때 사용하는 기능
TableView	TABLE 형태의 리스트 UI기능을 제공하는 아이템
TreeView	디렉토리 구조와 같은 트리 형태의 UI 기능을 제공하는 아이템

- Controls

Button, ComboBox, GroupBox, ProgressBar, CheckBox, Label, SpinBox 등과 같은 타입들을 제공한다.

예수님은 당신을 사랑합니다.



Controls 와 관련된 예제 소스코드는 00\_Controls 디렉토리를 참조하면 된다.

### ● Menus

Menu, MenuItem 그리고 MenuSeparator QML 타입을 제공한다.

QML타입	설명
Menu	팝업 메뉴, Context 메뉴
MenuItem	메뉴 또는 메뉴바에 추가하기 위한 아이템
MenuSeparator	메뉴 아이템 사이를 구분하기 위한 UI 기능 제공

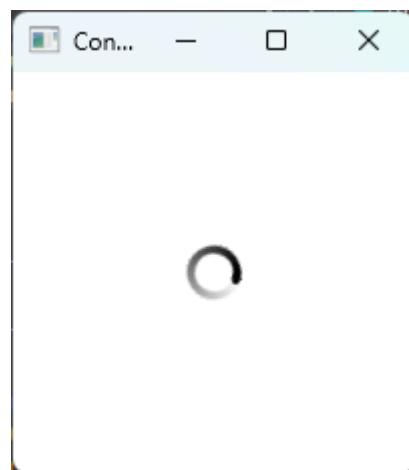
### ● BusyIndicator

ProgressBar 와 유사하다. 불확실한 진행률을 표시할 때 사용할 수 있다. 또한 백그라운드 활동을 표시하는 데 사용할 수 있다.

```
import QtQuick  
import QtQuick.Window  
import QtQuick.Controls
```

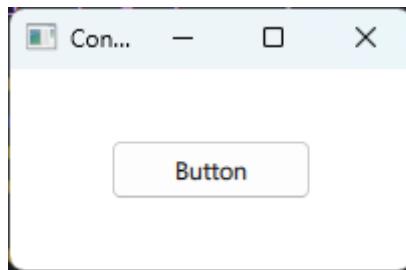
예수님은 당신을 사랑합니다.

```
Window {  
    width: 300; height: 300; visible: true  
    title: qsTr("Controls")  
  
    BusyIndicator {  
        anchors.centerIn: parent  
        running: true  
    }  
}
```



- Button

```
import QtQuick  
import QtQuick.Window  
import QtQuick.Controls  
  
Window {  
    width: 200; height: 200; visible: true  
    title: qsTr("Controls")  
  
    Button {  
        width: 100; height: 30  
        anchors.centerIn: parent  
        text: "Button"  
    }  
}
```



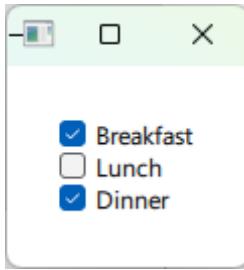
- CheckBox

```
import QtQuick
import QtQuick.Window
import QtQuick.Controls

Window {
    width: 100; height: 100; visible: true
    title: qsTr("Controls")

    Column {
        anchors.centerIn: parent

        CheckBox {
            text: qsTr("Breakfast")
            checked: true
        }
        CheckBox {
            text: qsTr("Lunch")
        }
        CheckBox {
            text: qsTr("Dinner")
            checked: true
        }
    }
}
```



- ComboBox

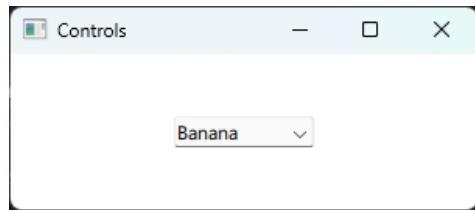
```
import QtQuick
import QtQuick.Window
import QtQuick.Controls

Window {
    width: 300; height: 100; visible: true;
    title: qsTr("Controls")

    ComboBox {
        anchors.centerIn: parent

        editable: true
        model: ListModel {
            id: model
            ListElement { text: "Banana" }
            ListElement { text: "Apple" }
            ListElement { text: "Coconut" }
        }
        onAccepted: {
            // Add only when no item with the same name exists
            if (find(currentText) === -1) {
                model.append({text: editText})
            }
        }
    }
}
```

예수님은 당신을 사랑합니다.



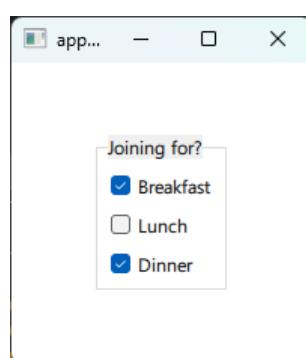
- **GroupBox**

```
import QtQuick
import QtQuick.Window
import QtQuick.Controls

Window {
    width: 200; height: 200
    visible: true

    GroupBox {
        anchors.centerIn: parent
        title: "Joining for?"

        Column {
            spacing: 10
            CheckBox { text: "Breakfast"; checked: true }
            CheckBox { text: "Lunch"; checked: false }
            CheckBox { text: "Dinner"; checked: true }
        }
    }
}
```

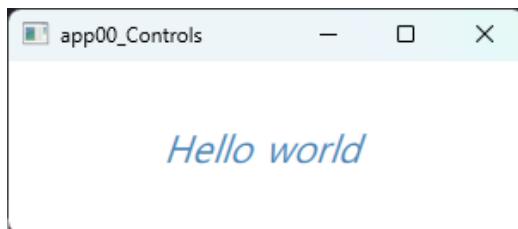


- Label

```
import QtQuick
import QtQuick.Window
import QtQuick.Controls

Window {
    width: 300
    height: 100
    visible: true

    Label {
        anchors.centerIn: parent
        text: "Hello world"
        font.pixelSize: 22
        font.italic: true
        color: "steelblue"
    }
}
```



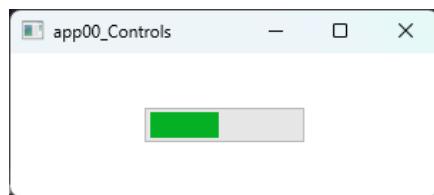
- ProgressBar

```
import QtQuick
import QtQuick.Window
import QtQuick.Controls

Window {
    width: 300; height: 300; visible: true
    Column {
        anchors.centerIn: parent
        spacing: 10
        ProgressBar {
```

예수님은 당신을 사랑합니다.

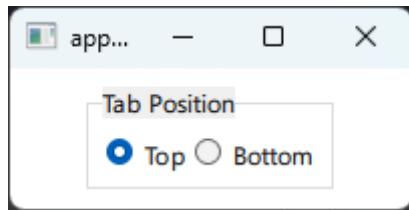
```
        value: 0.5
    }
}
}
```



### ● RadioButton

```
import QtQuick
import QtQuick.Window
import QtQuick.Controls
import QtQuick.Layouts

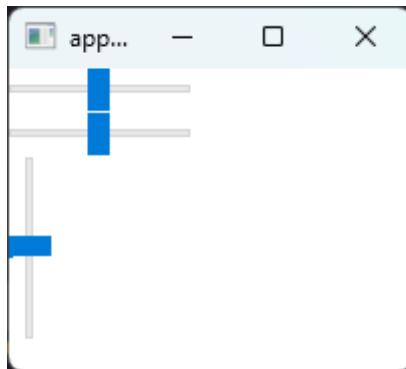
Window {
    width: 200; height: 70; visible: true
    GroupBox {
        title: "Tab Position"
        anchors.centerIn: parent
        RowLayout {
            RadioButton {
                text: "Top"
                checked: true
            }
            RadioButton {
                text: "Bottom"
            }
        }
    }
}
```



- Slider

```
import QtQuick
import QtQuick.Window
import QtQuick.Controls

Window {
    width: 400; height: 260; visible: true
    Column{
        Label { id: label }
        Slider{
            id: slider1; to: 100; from: 0
            value: 50; onValueChanged: label.text = "slider1: " + parseInt(value)
        }
        Slider{
            id: slider2; to: 100; from: 0
            value: 50; stepSize: 10
            onValueChanged: label.text = "slider2: " + parseInt(value)
        }
        Slider{
            id: slider3; to: 100; from: 0
            value: 50; stepSize: 25
            orientation: Qt.Vertical
            onValueChanged: label.text = "slider3: " + parseInt(value)
        }
    }
}
```



- **SpinBox**

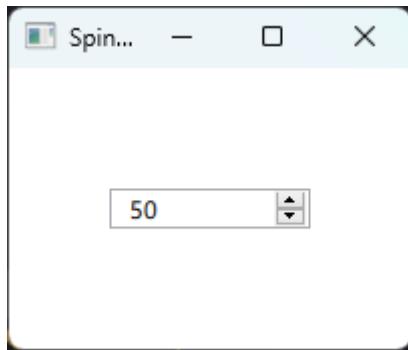
```
import QtQuick
import QtQuick.Window
import QtQuick.Controls

Window {
    title: qsTr("SpinBox")
    width: 200; height: 140; visible: true

    Column{
        anchors.centerIn: parent
        spacing: 5

        Label {
            id: label
        }

        SpinBox {
            id: spinbox1; width: 100
            from: 100; to: 0; value: 50
            onValueChanged: label.text = "spinbox1: " + value
        }
    }
}
```



- Switch

```
import QtQuick
import QtQuick.Window
import QtQuick.Controls

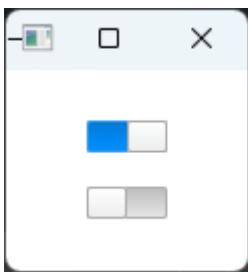
Window {
    width: 100
    height: 100
    visible: true

    Column {
        anchors.centerIn: parent
        spacing: 5
        Switch {
            id: first
            checked: true
            onClicked: {
                if(checked)
                    console.log("checked true");
                else
                    console.log("checked false")
            }
        }

        Switch {
            checked: false
        }
    }
}
```

예수님은 당신을 사랑합니다.

}



- TextArea

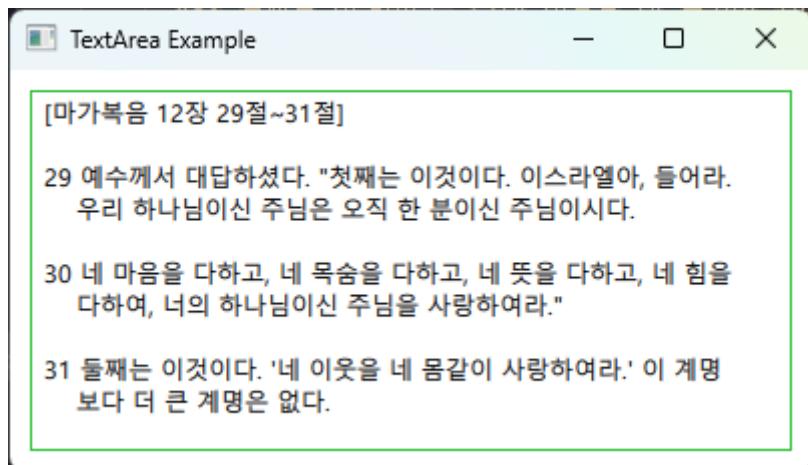
```
import QtQuick
import QtQuick.Window
import QtQuick.Controls

Window {
    width: 400; height: 200; visible: true
    title: "TextArea Example"

    TextArea {
        anchors.centerIn: parent
        background: Rectangle {
            border.color: "#21be2b"
        }

        width: 380; height: 180
        text:
            "[마가복음 12장 29절~31절] \n\n" +
            "29 예수께서 대답하셨다. \"첫째는 이것이다. 이스라엘아, 들어라.\n" +
            "    우리 하나님이신 주님은 오직 한 분이신 주님이시다.\n\n" +
            "30 네 마음을 다하고, 네 목숨을 다하고, 네 뜻을 다하고, 네 힘을\n" +
            "    다하여, 너의 하나님이신 주님을 사랑하여라.\" \n\n" +
            "31 둘째는 이것이다. \'네 이웃을 네 몸같이 사랑하여라.\' 이 계명\n" +
            "    보다 더 큰 계명은 없다."
    }
}
```

예수님은 당신을 사랑합니다.

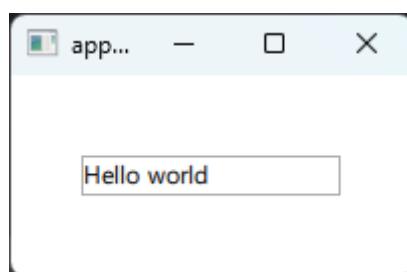


- TextField

```
import QtQuick
import QtQuick.Window
import QtQuick.Controls

Window {
    width: 200
    height: 100
    visible: true

    TextField {
        anchors.centerIn: parent
    }
}
```



예수님은 당신을 사랑합니다.

- Button 과 Action 예제

이번에 다룰 예제는 Button 타입에서 발생한 이벤트를 Action 에서 처리하는 예제이다.

```
import QtQuick
import QtQuick.Controls
import QtQuick.Window
import QtQuick.Layouts

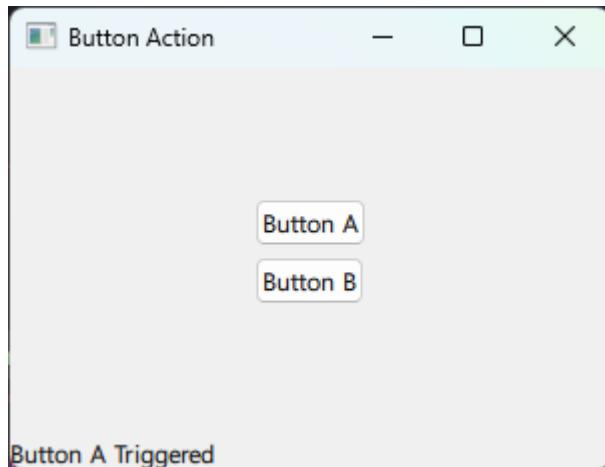
ApplicationWindow {
    title: qsTr("Button Action")
    width: 300; height: 200; visible: true

    ColumnLayout{
        anchors.centerIn: parent
        Button { action: actionBarA }
        Button { action: actionBarB }
    }

    Action{
        id: actionBar
        text: "Button A"
        onTriggered: statusLabel.text = "Button A Triggered "
    }

    Action{
        id: actionBarB
        text: "Button B"
        checkable: true
        onCheckedChanged:
            statusLabel.text = "Button B checked: " + checked
    }

    footer: Label {
        id: statusLabel; text: ""
    }
}
```



이 예제의 소스코드는 01\_Button\_Action 디렉토리를 참조하면 된다.

- ApplicationWindow 타입을 이용한 예제

이번 예제에서는 ApplicationWindow, MenuBar,ToolBar 그리고 StatusBar 타입을 이용해 윈도우 GUI 화면을 구현한 예제이다.

```
import QtQuick
import QtQuick.Controls
import QtQuick.Window
import QtQuick.Layouts

ApplicationWindow {
    id: myWindow; width: 480; height: 320
    title: "Applicatoin Example"
    visible: true

    menuBar: MenuBar {
        Menu {
            title: qsTr("File")
            MenuItem {
                text: qsTr("Exit")
                onTriggered: Qt.quit();
            }
            MenuItem {
                text: "File Item2"
                onTriggered: statusBar.text = "File Item2 Click"
            }
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
        }
    }
}

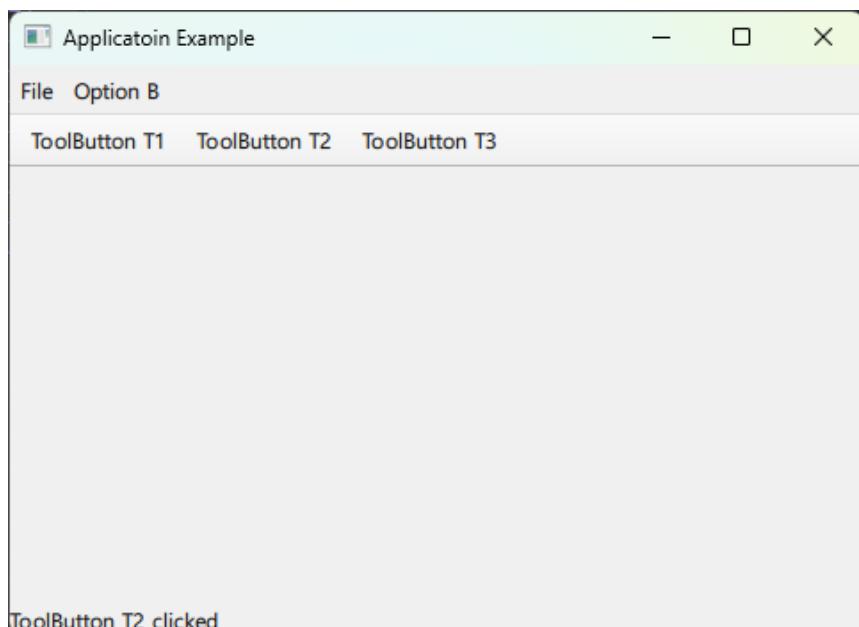
Menu {
    title: "Option B"
    MenuItem {
        text: "Opt B1"
        onTriggered: statusLabel.text = "Opt B1 Click"
    }
    MenuItem {
        text: "Opt B2"; checkable: true
        onCheckedChanged:
            statusLabel.text = "Opt B2 Checked - " + checked
    }
    MenuItem {
        text: "Opt B3"; checkable: true
        onCheckedChanged:
            statusLabel.text = "Opt B3 Checked - " + checked
    }
}

header:ToolBar{
    RowLayout{
        ToolButton{
            text: "ToolButton T1"
            onClicked: statusLabel.text = "ToolButton T1 clicked"
        }
        ToolButton{
            text: "ToolButton T2"
            onClicked: statusLabel.text = "ToolButton T2 clicked"
        }
        ToolButton{
            text: "ToolButton T3"
            onClicked: statusLabel.text = "ToolButton T3 clicked"
        }
    }
}

footer: RowLayout{
```

예수님은 당신을 사랑합니다.

```
Label{  
    id: statusLabel  
    text: "Status Bar"  
}  
}  
}
```



이 예제의 소스코드는 02\_ApplicationWindow 디렉토리를 참조하면 된다.

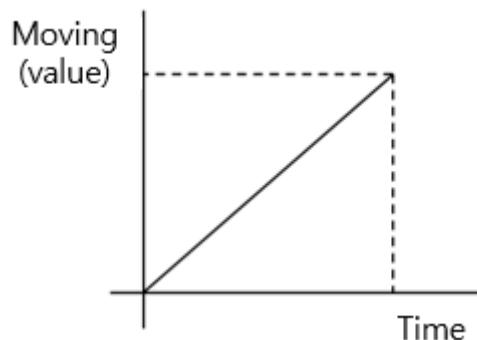
### 3. Animation Framework

화면 전환 시 부드러운 화면 전환과 같은 애니메이션을 사용하기 위해서 Animation Framework 를 제공한다.

애니메이션 요소로 투명, 이동, 확대/축소 등과 같은 애니메이션 요소를 사용할 수 있다. 예를 들어 어플리케이션 실행 시 화면 가로와 세로 크기가  $600 \times 400$  픽셀 크기의 윈도우 화면이 있다고 가정해 보자. 처음 크기는  $100 \times 100$  크기에서 지정한 시간 동안  $600 \times 400$  크기로 윈도우 창의 크기가 커지게 할 수 있다.

또한 동시에 투명(Opacity) 을 사용해 Animation 효과를 추가할 수 있다. 0.0 은 완전한 투명 상태이고 1.0 은 완전한 투명 상태이다. 지정된 시간 동안 0.0 에서 1.0 까지 값이 변경되도록 Animation 처리가 가능하다.

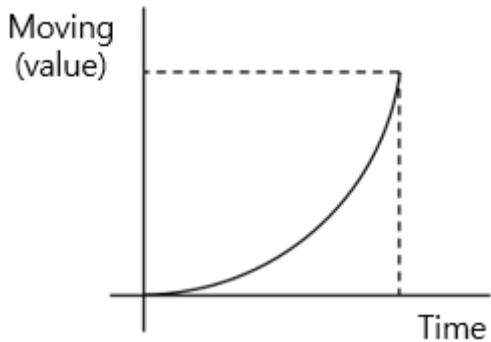
이외에도 애니메이션의 시간 값에 Easing Curves 를 사용할 수 있다. 예를 들어 1.0 초 동안 GUI 버튼의 X, Y 위치가 100, 100 의 위치에서 200, 200 의 좌표로 이동하고 이동하는 시간을 1.0 초로 지정했다고 가정해보자. 이때 X, Y 시작 좌표부터 목적지 좌표까지 이동하는데 일정한 속도로 이동할 것이다.



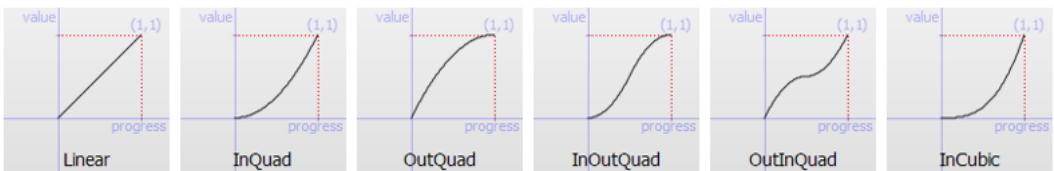
위의 그림에서 보는 것과 같이 이동(값) 이라는 Y 축은 GUI 버튼의 X, Y 위치 좌표가 100, 100 에서 200, 200 값으로 이동하는 데 1.0 초 시간이 일정하게 소요되는 것을 나타내는 그래프이다.

즉 이동 값에 따라 시간이 일정하게(Linear) 증가한다. 하지만 시간에 Easing Curves 를 사용하면 이동 값에 시간의 값을 다음 그림에서 보는 것과 같이 변경할 수 있다.

예수님은 당신을 사랑합니다.



최종 X, Y 좌표까지 이동하는 시간의 값을 위의 그림에서 보는 것과 같이 적용이 가능하다. 이외에도 Easing Curve 는 약 46 가지의 다양한 Easing Curve 를 적용할 수 있다.



<그림> 다양한 Easing Curve 적용

그리고 QML 은 Animation 요소에 State Transition 이라는 기법을 사용할 수 있다. 예로 ON/OFF 스위치를 예를 들 수 있다. 한가지 값을 변경하기 위해 ON/OFF 를 사용할 수 있을 뿐만 아니라 여러가지 옵션을 함께 사용할 수 있다.

즉 어떤 상황이 되면 그 상황에 따라 여러가지 값이 동시에 적용되는 기법인 State Transition 기법을 사용할 수 있다. 이번 장에서는 지금까지 설명한 내용에 대해서 살펴보도록 하자.

### 3.1. Animation

Qt에서는 아래 표에서 보는 것과 같이 자주 사용하는 Animation 타입입니다.

타입	설명
NumberAnimation	좌표 X, Y 혹은 Opacity와 같은 투명 값 등과 같이 숫자를 이용한 Animation을 적용
PropertyAnimation	RGB, width, height등의 프로퍼티에 Animation을 적용
RotationAnimation	회전 Animation을 적용
PauseAnimation	Animation 을 정지 시키기 위한 기능
SmoothedAnimation	중력 가속도 가중치를 사용하기 위한 기능
SpringAnimation	스프링과 같은 Animation을 사용하기 위한 목적으로 제공
Behavior	특정 값이 변화 되면 발생하는 Animation, 예로 width 가 변경되면 Animation 실행
ScriptAction	Animation 실행 될 때 특정 Method 또는 Script를 실행하기 위한 목적으로 제공
PropertyAction	Animation 동작하는 동안 Element 의 프로퍼티 값을 변경하기 위해 제공

다음은 Animation 을 그룹화 할 수 있는 타입을 제공한다.

타입	설명
SequentialAnimation	여러 개의 Animation 을 순차적으로 실행
ParallelAnimation	여러 개의 Animation 을 병렬로 실행

- NumberAnimation

NumberAnimation 은 숫자를 이용해 Animation 을 적용할 수 있다. 예를 들어 X, Y 좌표로 지정한 Image 타입을 2,000 Millisecond 동안 X 좌표가 10에서 250 값으로 변경한다. 만약 duration 프로퍼티의 값을 지정하지 않으면 디폴트로 250 millisecond 값이 지정된다.

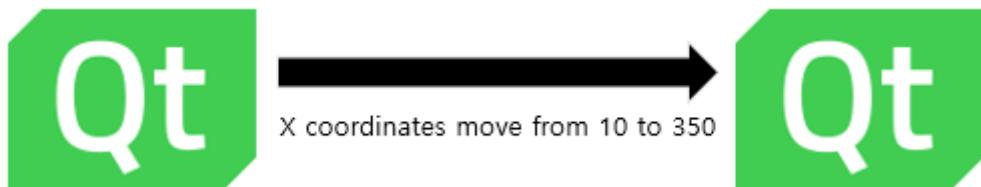
예수님은 당신을 사랑합니다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 630; height: 230; visible: true

    Image {
        source: "images/qtlogo.png"
        x: 10; y: 20;

        NumberAnimation on x
        {
            from: 10; to: 350
            duration: 2000
        }
    }
}
```



- PropertyAnimation

PropertyAnimation은 타입의 프로퍼티 상에 Animation을 적용할 수 있다. 다음 예제는 Image 타입의 width 와 height 프로퍼티의 값을 애니메이션의 값으로 사용한 예제이다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 230; height: 230; visible: true

    Image {
        id: proAni
        source: "images/qtlogo.png"
        x: 50; y: 40;
```

예수님은 당신을 사랑합니다.

```
width: 50; height: 50;  
}  
  
PropertyAnimation {  
    target: proAni  
    properties: "width, height"  
    from: 0; to: 100; duration: 1000  
    running: true  
    //loops : Animation.Infinite  
}  
}
```

magnifying width and height



- RotationAnimation

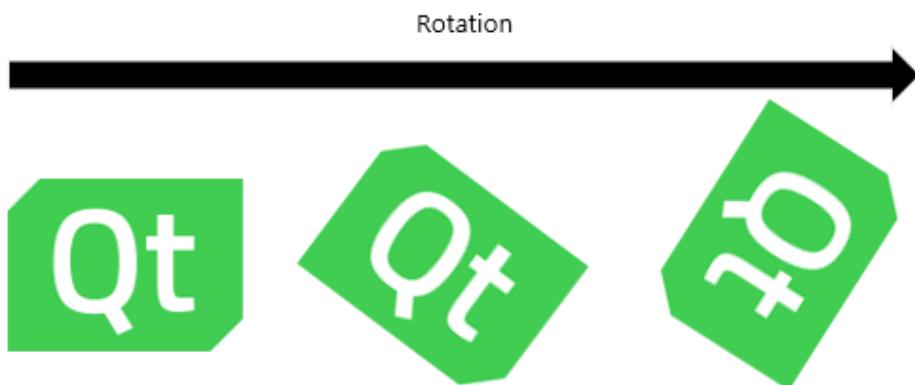
RotationAnimation 타입은 시작과 마지막 값을 애니메이션에 적용할 수 있다. 시작과 마지막 값은 각도의 값이 된다. 따라서 from 프로퍼티는 시작 각도이며 to 프로퍼티는 마지막 각도이다.

```
import QtQuick  
import QtQuick.Window  
  
Window {  
    width: 330; height: 330; visible: true  
  
    Image {  
        id: rotAni  
        source: "images/qtlogo.png"  
        anchors.centerIn: parent
```

예수님은 당신을 사랑합니다.

```
smooth: true

    RotationAnimation on rotation {
        from: 45; to: 315
        direction: RotationAnimation.Shortest
        duration: 3000
    }
}
```



위의 예제에서 `RotationAnimation` 의 `direction` 프로퍼티는 시계방향 또는 반 시계 방향으로 회전할지 지정할 수 있다. 다음은 `direction` 프로퍼티 값으로 사용할 수 있는 상수이다.

상수	설명
<code>RotationAnimation.Numerical</code>	Linear 보간 알고리즘 사용. 방향은 시계 방향
<code>RotationAnimation.Clockwise</code>	시계 방향으로 회전
<code>RotationAnimation.Counterclockwise</code>	반 시계 방향으로 회전
<code>RotationAnimation.Shortest</code>	STEP 은 20도씩 회전. 회전 방향은 반 시계 방향.

- Animation 의 이벤트

Animation 이 시작되는 순간, Animation 이 중지 되었을 때 또는 Animation 의 프로퍼티 값이 변경되었을 때 이벤트를 발생할 수 있다.

예수님은 당신을 사랑합니다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 330; height: 230; visible: true
    Image {
        source: "images/qtlogo.png"
        x: 10; y: 20;
        id : logo

        NumberAnimation on scale
        {
            id : scaleAni
            from: 0.1; to: 1.0
            duration: 2000
            running: true

            onStart : console.log("started")
            onStop : console.log("stopped")
        }

        onScaleChanged: {
            if(scale > 0.5) {
                scaleAni.complete()
            }
        }
    }
}
```

- Behavior

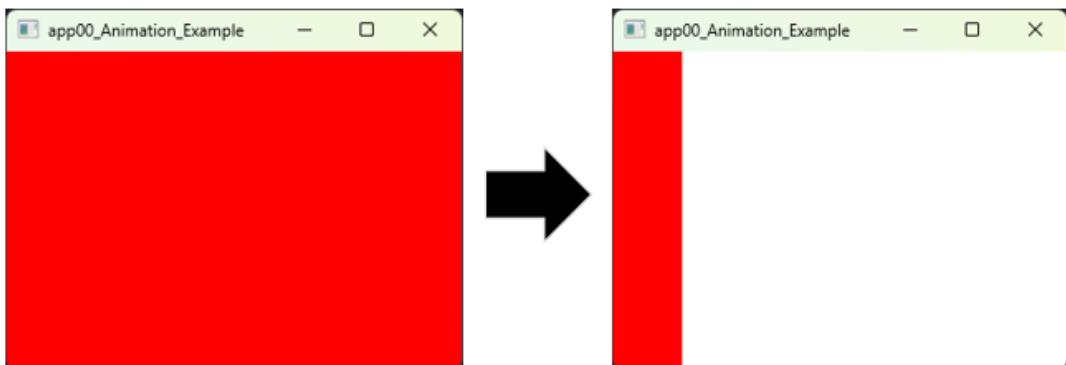
Behavior 는 타입의 프로퍼티의 값이 변경 되었을 때 동작 시키기 위한 목적으로 사용한다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 330; height: 230; visible: true
```

```
Rectangle {  
    width: 330; height: 230;  
    id: rect  
    color: "red"  
  
    Behavior on width {  
        NumberAnimation {  
            duration: 1000  
        }  
    }  
  
    MouseArea {  
        anchors.fill: parent  
        onClicked: rect.width = 50  
    }  
}  
}
```

위의 예제는 마우스를 클릭하면 Rectangle 타입의 width 값이 50 으로 변경되면 width 값이 현재 값이 330 에서 50 으로 변경되는데 Animation 을 적용할 수 있다.



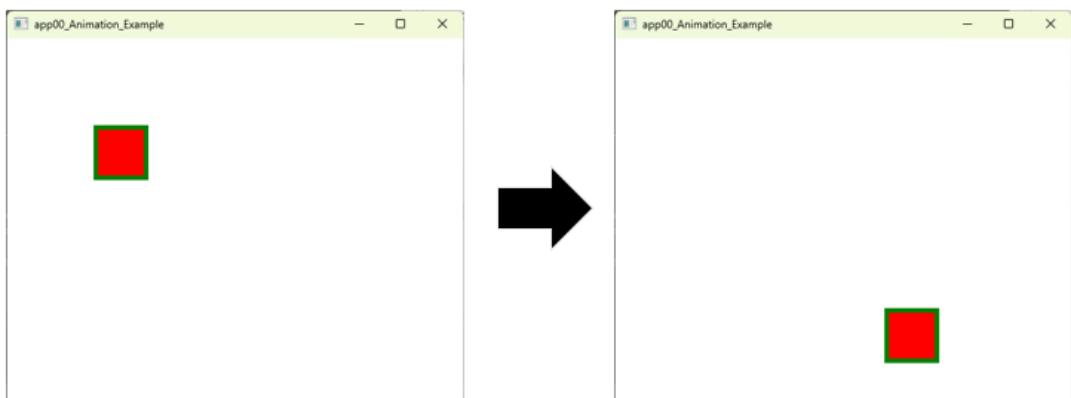
- SmoothedAnimation

SmoothedAnimation 타입은 velocity 프로퍼티 값을 사용해 Animation 을 처리할 수 있는 기능을 제공한다. 다음 예제는 키보드의 방향키에 따라 Rectangle 이 이동하는 예제이다.

```
import QtQuick  
import QtQuick.Window
```

예수님은 당신을 사랑합니다.

```
Window {  
    width: 800; height: 600; visible: true  
  
    Rectangle {  
        width: 800; height: 600  
        Rectangle {  
            width: 60; height: 60  
            x: rect1.x - 5; y: rect1.y - 5  
            color: "green"  
  
            Behavior on x { SmoothedAnimation { velocity: 200 } }  
            Behavior on y { SmoothedAnimation { velocity: 200 } }  
        }  
  
        Rectangle {  
            id: rect1; width: 50; height: 50; color: "red"  
        }  
  
        focus: true  
        Keys.onRightPressed: rect1.x = rect1.x + 100  
        Keys.onLeftPressed: rect1.x = rect1.x - 100  
        Keys.onUpPressed: rect1.y = rect1.y - 100  
        Keys.onDownPressed: rect1.y = rect1.y + 100  
    }  
}
```



- SpringAnimation

SpringAnimation 타입은 스프링과 같은 Animation 을 사용할 수 있다. spring 프로퍼티는 스프링의 탄력이 좋은지 그렇지 않은지 효과를 반영할 수 있다. 예를 들어 스프링을 늘렸다가 놓으면 빨리 줄어드는 것과 같은 프로퍼티 효과를 적용할 수 있다. 따라서 spring 프로퍼티 값이 클 수록 탄력성 또는 응집성이 높다.

damping 프로퍼티는 제동(brake) 과 같은 효과를 위해 사용할 수 있다. 값이 클 수록 제동이 빨리 걸린다. 값은 0.0에서 1.0 사이의 값을 설정할 수 있다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 300; height: 200; visible: true

    Rectangle {
        id: rect
        width: 50; height: 50
        color: "red"

        Behavior on x {
            SpringAnimation {
                spring: 14; damping: 0.2
            }
        }

        Behavior on y {
            SpringAnimation {
                spring: 14; damping: 0.2
            }
        }
    }

    MouseArea {
        anchors.fill: parent
        onClicked: {
            rect.x = mouseX - rect.width/2
            rect.y = mouseY - rect.height/2
        }
    }
}
```

```
    }  
}
```

- SequentialAnimation

SequentialAnimation 타입은 여러 개의 Animation 을 순서대로 실행하기 위한 기능을 제공한다.

```
import QtQuick  
import QtQuick.Window  
  
Window {  
    width: 300; height: 200; visible: true  
  
    Image {  
        id: aniSeq  
        anchors.centerIn: parent  
        source: "images/qtlogo.png"  
    }  
  
    SequentialAnimation {  
        NumberAnimation {  
            target: aniSeq; properties: "scale"  
            from: 1.0; to: 0.5; duration: 1000  
        }  
        NumberAnimation {  
            target: aniSeq; properties: "opacity"  
            from: 1.0; to: 0.5; duration: 1000  
        }  
        running: true  
    }  
}
```

- ParallelAnimation

ParallelAnimation 은 사용한 Animation 을 모두 동시에 수행할 수 있는 기능을 제공한다.

예수님은 당신을 사랑합니다.

예를 들어 이전 SequentialAnimation 에서는 Animation 을 순서대로 실행 하고 ParallelAnimation 은 Animation 들을 동시(병렬)에 실행 한다.

```
import QtQuick
import QtQuick.Window

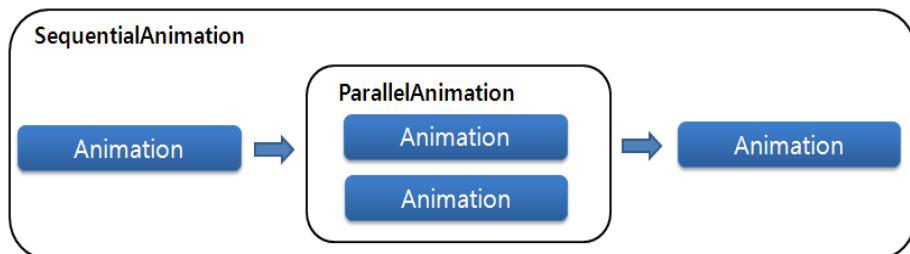
Window {
    width: 300; height: 200; visible: true

    Image {
        id: aniPara
        anchors.centerIn: parent
        source: "images/qtlogo.png"
    }

    ParallelAnimation {
        NumberAnimation {
            target: aniPara; properties: "scale"
            from: 1.0; to: 0.5; duration: 1000
        }
        NumberAnimation {
            target: aniPara; properties: "opacity"
            from: 1.0; to: 0.5; duration: 1000
        }
        running: true
    }
}
```

- SequentialAnimation 과 ParallelAnimation 의 중첩된 예제

SequentialAnimation 과 ParallelAnimation은 중첩된 구조로 수행할 수 있다. 예를 들어 아래 그림과 같이 중첩된 구조로 수행할 수 있다.



예수님은 당신을 사랑합니다.

다음 예제는 SequentialAnimation 과 ParallelAnimation을 위의 그림에서 보는 것과 같이 중첩된 구조로 사용하는 예제이다.

```
import QtQuick
import QtQuick.Window

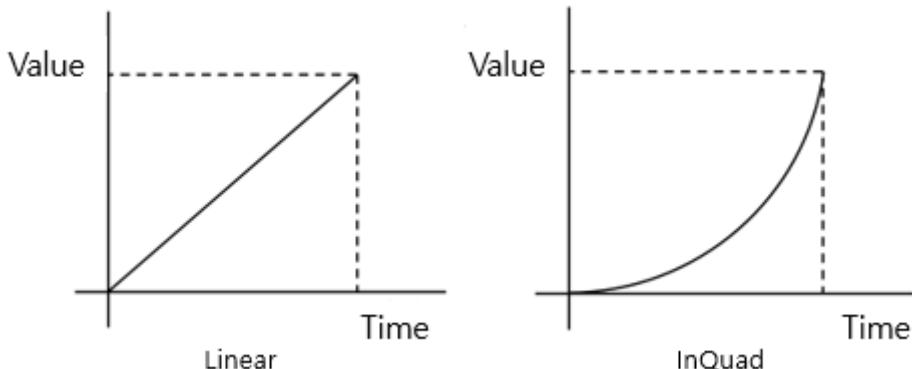
Window {
    width: 300; height: 200; visible: true

    Image {
        id: qtlogo; anchors.centerIn: parent
        source: "./images/qtlogo.png"
    }

    SequentialAnimation {
        NumberAnimation {
            target: qtlogo; properties: "scale"
            from: 1.0; to: 0.5; duration: 1000
        }
        SequentialAnimation {
            NumberAnimation {
                target: qtlogo; properties: "rotation"
                from: 0.0; to: 360.0; duration: 1000
            }
            NumberAnimation {
                target: qtlogo; properties: "opacity"
                from: 1.0; to: 0.0; duration: 1000
            }
        }
        running: true
    }
}
```

### ● Easing Curve

Easing Curve 는 좌표의 이동, 투명, 확대 및 축소 등과 같이 Animation 의 시작과 마지막 시간까지 Linear한 패턴으로 Animation 이 실행되지 않고 다양한 Curve 를 적용 할 수 있다. 예를 들어 타입의 시작 좌표부터 마지막 좌표까지 이동하는 경로 진행 속도를 다음과 같은 패턴으로 변경할 수 있다.



다음 예제 소스코드는 Easing curve 의 OutExpo를 사용한 예제이다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 300; height: 200; visible: true
    Image {
        id: easCur
        anchors.centerIn: parent
        source: "images/qtlogo.png"

        NumberAnimation {
            target: easCur; properties: "scale"
            from: 0.1; to: 1.0; duration: 1000
            easing.type: "OutExpo"
            running: true
        }
    }
}
```

위의 예제에서 easing.type 프로퍼티는 사용할 Easing curve 중 하나를 지정할 수 있다.

Easing curve 를 사용하지 않을 때와 사용했을 때의 차이점을 비교해보면 어떤 차이점이 있는지 이해할 수 있다.

- PauseAnimation

PauseAnimation 은 SequentialAnimation 에서 순차 대로 실행되는 Animation 사이에 duration 프로퍼티 값이 지정한 시간 동안 Animation 을 일시정지 할 수 있다.

예수님은 당신을 사랑합니다.

NumberAnimation

PauseAnimation

NumberAnimation

일시 정지 duration: 1000  
millisecond

다음 예제는 PauseAnimation 타입을 사용한 예제이다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 300; height: 200; visible: true

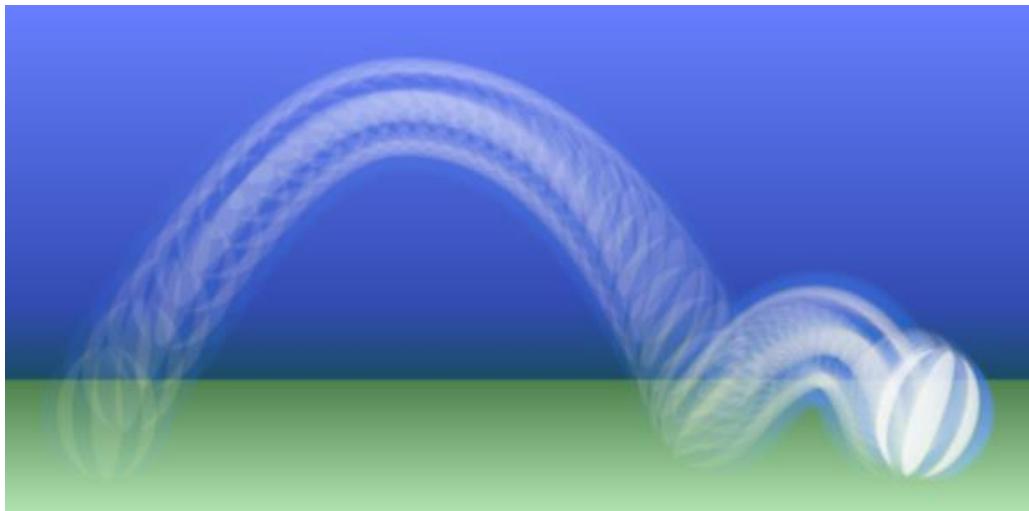
    Image {
        id: logo
        anchors.centerIn: parent; source: "images/qtlogo.png"
    }

    SequentialAnimation {
        NumberAnimation {
            target: logo; properties: "scale"
            from: 0.0; to: 1.0; duration: 1000
        }
        PauseAnimation {
            duration: 1000
        }
        NumberAnimation {
            target: logo; properties: "scale"
            from: 1.0; to: 0.0; duration: 1000
        }
        running: true
    }
}
```

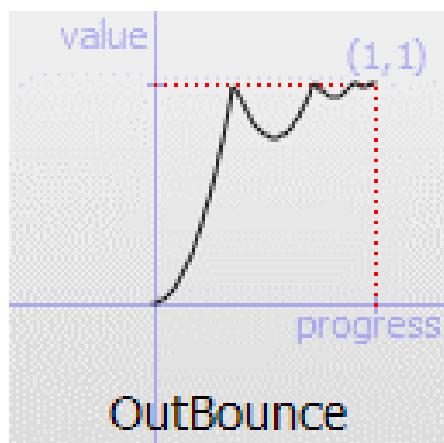
이번 장에서 다른 예제 소스코드는 00\_Animation\_Example 디렉토리를 참조하면 된다.

### 3.2. Animation 과 Easing curve 를 이용한 예제 구현

이번 절에서는 Animation 과 Easing curve를 이용한 예제를 작성해 보자.



위의 그림에서 보는 것과 같이 Ball 이 바닥에 튕며 이동하는 예제를 작성해 보도록 하자. 먼저 Ball 이미지를 표시하기 위해 Image 타입을 사용하고 Ball 이미지가 다음 예제와 같이 위에서 아래로 Ball 떨어지며 튕는 Animation 을 적용해 보자. Ball 이미지가 바닥에 떨어질 때 퉁퉁 튕는 Animation 을 적용하기 위해 Easing curve의 OutBounce 를 사용해 보자.



```
import QtQuick  
import QtQuick.Window
```

```
Window {  
    width: 600; height: 300; visible: true  
  
    Image {  
        source: "images/ball.png"  
        anchors.horizontalCenter: parent.horizontalCenter  
  
        NumberAnimation on y  
        {  
            from: 20; to: 200  
            easing.type: "OutBounce"  
            duration: 1000  
        }  
    }  
}
```

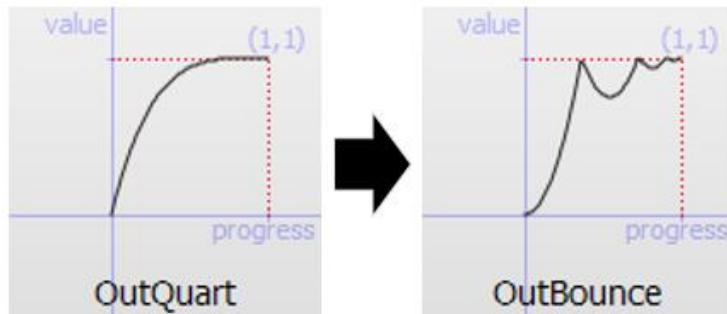
위와 같이 작성하고 실행하면 Ball 이미지가 아래에서 떨어지며 통통 튀는 듯한 것과 같이 동작할 것이다. 이번에는 NumberAnimation 을 하나 더 추가 시켜 보자.

```
import QtQuick  
import QtQuick.Window  
  
Window {  
    width: 600; height: 300; visible: true  
  
    Image {  
        source: "images/ball.png"  
        anchors.horizontalCenter: parent.horizontalCenter  
  
        SequentialAnimation on y {  
            NumberAnimation {  
                from: 200; to: 20  
                easing.type: "OutQuad"  
                duration: 250  
            }  
            NumberAnimation {  
                from: 20; to: 200  
                easing.type: "OutBounce"  
                duration: 1000  
            }  
        }  
    }  
}
```

```

        }
    }
}
}
```

추가한 Animation 은 Ball 이미지가 위로 던져지는 듯한 Animation 을 적용하였다. 따라서 아래 그림에서 보는 것과 같이 Animation 이 순차적으로 수행된다.



지금까지 Ball 이미지가 Y축으로 움직였다. 이번에는 X축으로 Animation 을 적용 시켜보자. X축으로 이동하기 위해서 다음과 같이 NumberAnimation 을 추가한다.

```

import QtQuick
import QtQuick.Window

Window {
    width: 600; height: 300; visible: true

    Image {
        source: "images/ball.png"

        NumberAnimation on x {
            from: 20; to: 500
            easing.type: "OutQuad"
            duration: 1250
        }

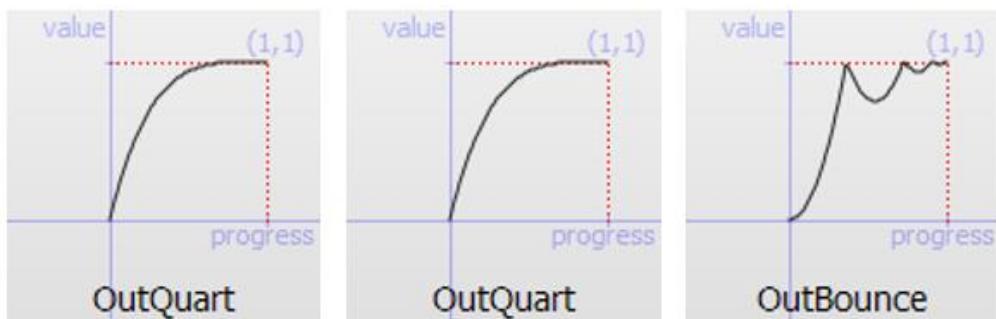
        SequentialAnimation on y {
            NumberAnimation {
                from: 200; to: 20
                easing.type: "OutQuad"
                duration: 250
            }
        }
    }
}
```

```

NumberAnimation {
    from: 20; to: 200
    easing.type: "OutBounce"
    duration: 1000
}
}
}
}

```

위와 같이 NumberAnimation 을 추가하면 Ball 이미지가 좌측에서 우측 방향으로 이동한다. 예제를 실행 시켜보면 Ball 이미지가 좌측에서 우측으로 이동한다. 그리고 동시에 볼이 통통 튀기는 것과 같다. 예를 들어 우리 Ball 을 던지면 바닥에 통통 튀는 것과 같이 Animation 이 동작한다. 이번 예제에서는 다음 그림에서 보는 것과 같이 3개의 Animation 을 사용하였다.



다음은 RotationAnimation 을 추가해 보자. RotationAnimation 에서는 시계방향으로 Ball 이미지가 360도 회전한다.

```

import QtQuick
import QtQuick.Window

Window {
    width: 600; height: 300; visible: true

    Image {
        source: "images/ball.png"

        NumberAnimation on x {
            from: 20; to: 500
            easing.type: "OutQuad"
            duration: 1250
        }
    }
}

```

예수님은 당신을 사랑합니다.

```
}

SequentialAnimation on y {
    NumberAnimation {
        from: 200; to: 20
        easing.type: "OutQuad"
        duration: 250
    }
    NumberAnimation {
        from: 20; to: 200
        easing.type: "OutBounce"
        duration: 1000
    }
}

RotationAnimation on rotation {
    from: 0; to: 360
    direction: RotationAnimation.Clockwise
    duration: 1000
}
}
```

다음으로 Animation 을 ParallelAnimation 과 SequentialAnimation 을 사용해 보자.

```
import QtQuick
import QtQuick.Window

Window {
    width: 600; height: 300; visible: true

    Image {
        id: ball
        source: "images/ball.png"
        x: 20; y: 200
        smooth: true
        MouseArea {
            anchors.fill: parent
            onClicked: ballAnimation.running = true
        }
    }
}
```

```
ParallelAnimation {
    id: ballAnimation

    NumberAnimation {
        target: ball
        property: "x"
        from: 20; to: 500
        easing.type: "OutQuad"
        duration: 1250
    }

    SequentialAnimation {
        NumberAnimation {
            target: ball
            property: "y"
            from: 200; to: 20
            easing.type: "OutQuad"
            duration: 250
        }
        NumberAnimation {
            target: ball
            property: "y"
            from: 20; to: 200
            easing.type: "OutBounce"
            duration: 1000
        }
    }
}

SequentialAnimation {
    RotationAnimation {
        target: ball
        property: "rotation"
        from: 0; to: 360
        direction: RotationAnimation.Clockwise
        duration: 1000
    }
    RotationAnimation {
        target: ball
```

예수님은 당신을 사랑합니다.

```
        property: "rotation"
        from: 360; to: 380
        direction: RotationAnimation.Clockwise
        duration: 250
    }
}
}
}
}
```

마지막으로 배경색으로 Gradient를 사용해 배경색을 칠해보자. 다음 예제와 같이 작성된 코드에 Rectangle 과 Gradient 타입을 추가해 보자.

```
import QtQuick
import QtQuick.Window

Window {
    width: 600; height: 300; visible: true

    Rectangle {
        x: 0; y:0
        width: parent.width; height: 220
        gradient: Gradient {
            GradientStop { position: 0.0; color: Qt.rgba(0.4,0.5,1.0,1) }
            GradientStop { position: 0.8; color: Qt.rgba(0.2,0.3,0.7,1) }
            GradientStop { position: 1.0; color: Qt.rgba(0.1,0.3,0.4,1) }
        }
    }
    Rectangle {
        y: 220
        width: parent.width; height: 80
        gradient: Gradient {
            GradientStop { position: 1.0; color: Qt.rgba(0.7,0.9,0.7,1) }
            GradientStop { position: 0.0; color: Qt.rgba(0.3,0.5,0.3,1) }
        }
    }

    Image {
        id: ball
        source: "images/ball.png"
```

예수님은 당신을 사랑합니다.

```
x: 20; y: 200
smooth: true

MouseArea {
    anchors.fill: parent
    onClicked: ballAnimation.running = true
}

ParallelAnimation {
    id: ballAnimation

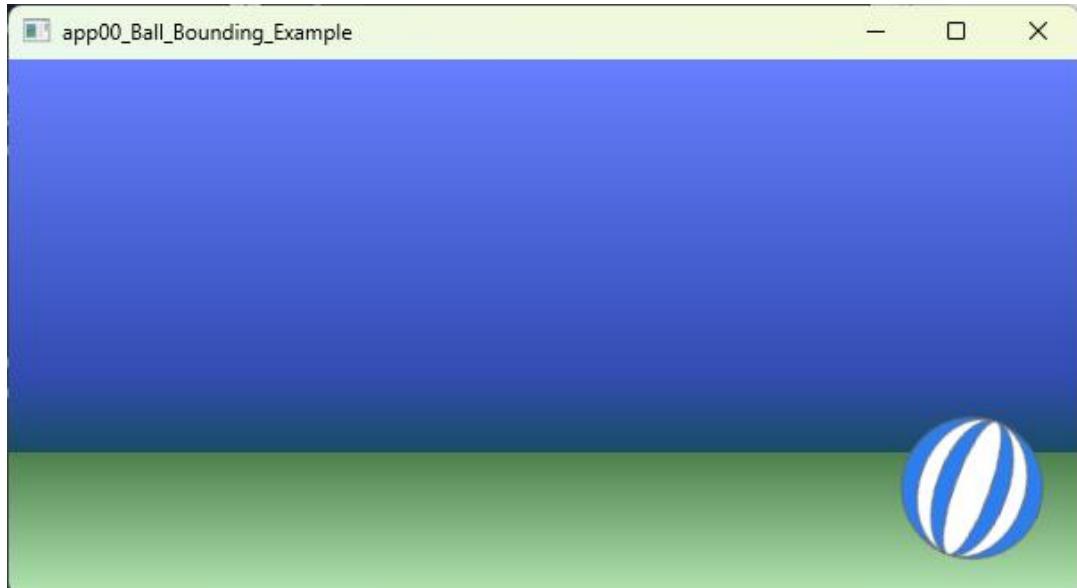
    NumberAnimation {
        target: ball
        property: "x"
        from: 20; to: 500
        easing.type: "OutQuad"
        duration: 1250
    }

    SequentialAnimation {
        NumberAnimation {
            target: ball
            property: "y"
            from: 200; to: 20
            easing.type: "OutQuad"
            duration: 250
        }
        NumberAnimation {
            target: ball
            property: "y"
            from: 20; to: 200
            easing.type: "OutBounce"
            duration: 1000
        }
    }
}

SequentialAnimation {
    RotationAnimation {
        target: ball
```

```
        property: "rotation"
        from: 0; to: 360
        direction: RotationAnimation.Clockwise
        duration: 1000
    }
    RotationAnimation {
        target: ball
        property: "rotation"
        from: 360; to: 380
        direction: RotationAnimation.Clockwise
        duration: 250
    }
}
}
}
}
```

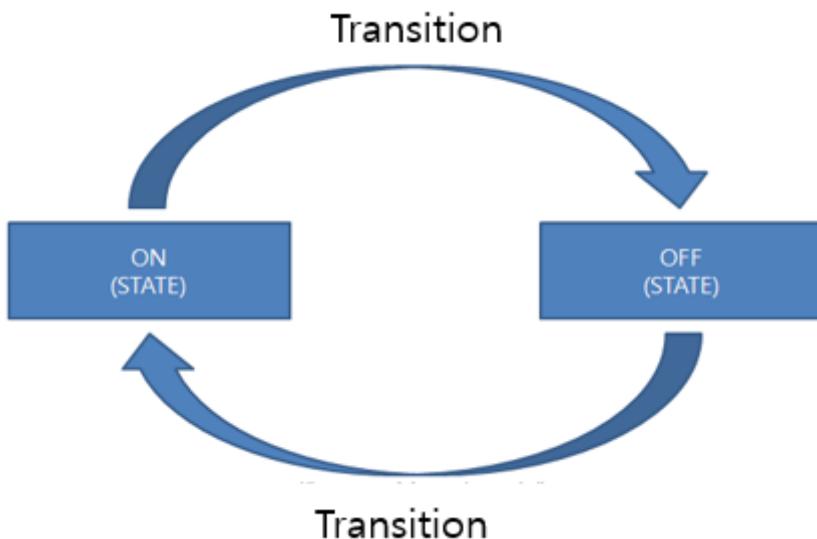
위와 같이 소스코드를 작성하면 Ball 이미지가 Bouncing 되는것과 같은 Animation을 적용한 것을 확인할 수 있다. 작성한 예제를 실행 시키고 Ball 이미지를 마우스로 클릭하면 Animation 이 동작된다.



### 3.3. State and Transition

State 와 Transition 은 QML의 타입에 State Machine 기법을 적용할 수 있다. 예를 들어 ON/OFF 스위치가 있다고 가정해 보자. ON 또는 OFF 상태를 State라고 정의한다. 그리고 ON 에서 OFF 또는 OFF 에서 ON 으로 State 가 바뀌는 것을 행동, 즉 Transition 이라고 정의한다.

QML에서 State 는 특정 타입의 상태이고 어떤 이벤트에 의해 QML 타입이 Animation 에 의해 이동, 프로퍼티 속성이 변경되는 것을 Transition 이라고 정의할 수 있다.



- State를 이용한 예제

예를 들어 Rectangle 타입의 Color 가 Red 인 것을 State 라고 할 수 있다. Rectangle 타입의 Color 가 Red 에서 Black 으로 바뀌는 것을 Transition 으로 정의 할 수 있다. 다음 예제는 Rectangle 의 Color 가 변경되는 기능을 State를 이용해 구현한 예제이다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 150; height: 250; visible: true

    Rectangle {
```

```
width: parent.width; height: parent.height

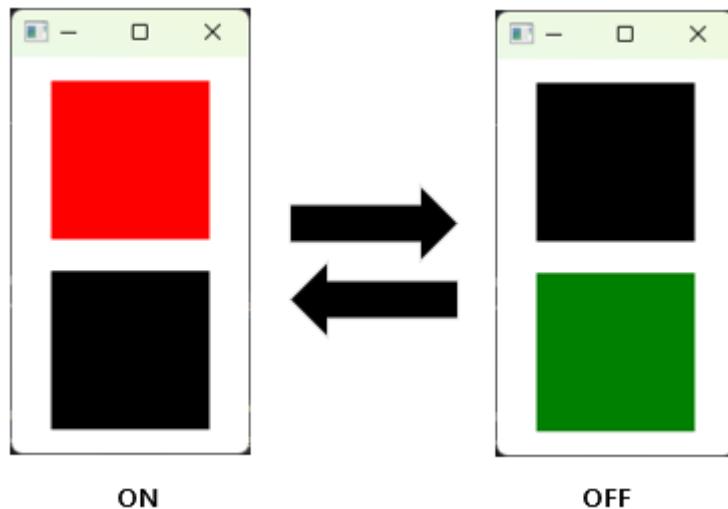
Rectangle {
    id: onElement
    x: 25; y: 15; width: 100; height: 100
}

Rectangle {
    id: offElement
    x: 25; y: 135; width: 100; height: 100
}

states: [
    State {
        name: "on"
        PropertyChanges { target: onElement; color: "red" }
        PropertyChanges { target: offElement; color: "black" }
    },
    State {
        name: "off"
        PropertyChanges { target: onElement; color: "black" }
        PropertyChanges { target: offElement; color: "green" }
    }
]

state: "on"

MouseArea
{
    anchors.fill: parent
    onClicked: parent.state === "on" ?
        parent.state = "off" : parent.state = "on"
}
}
```



- State 와 Transition 을 이용한 예제

이번 예제는 State 와 Transition 을 사용해 예제를 구현해 보도록 하자. 아래 그림에서 보는 것과 같이 예제를 실행하면 해당 이미지가 Rotation 된다.

State 는 "up" 과 "down" 2개의 State 가 존재하면 "up" 에서 "down" 로 Transition 이 수행되면 이미지가 180도 회전한다. 반대로 "down" 에서 "up" 로 Transition 이 수행되면 이미지가 180도 회전한다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 150; height: 150; visible: true

    Rectangle {
        width: 150; height: 150; color: "black"

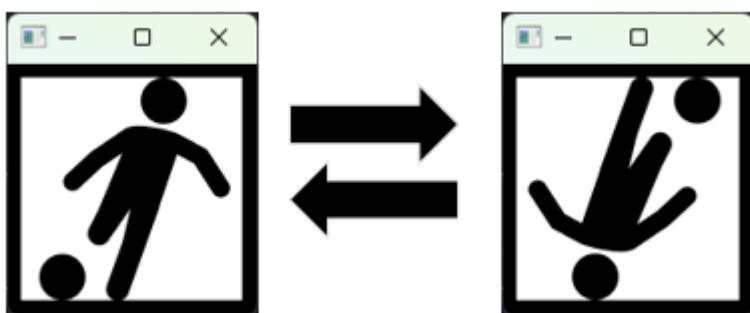
        Image {
            id: player
            anchors.horizontalCenter: parent.horizontalCenter
            anchors.verticalCenter: parent.verticalCenter
            source: "images/player.png"
        }
    }

    states: [
```

```
State {
    name: "up"
    PropertyChanges { target: player; rotation: 0 }
},
State {
    name: "down"
    PropertyChanges { target: player; rotation: 180 }
}
]

state: "up"

transitions: [
    Transition {
        from: "*"; to: "*"
        PropertyAnimation {
            target: player
            properties: "rotation"; duration: 1000
        }
    }
]
MouseArea {
    anchors.fill: parent
    onClicked: parent.state === "up" ?
        parent.state = "down" : parent.state = "up"
}
}
```



## 예수님은 당신을 사랑합니다.

- State 와 when

State 의 when 프로퍼티는 특정 조건일 때 실행된다. 다음 예제에서 보는 것과 같이 when 프로퍼티를 사용할 수 있다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 250; height: 100; visible: true

    Rectangle {
        width: 250; height: 100; color: "#ccffcc"

        TextInput {
            id: textField
            text: "Hello world."
            font.pointSize: 24
            anchors.left: parent.left
            anchors.leftMargin: 4
            anchors.verticalCenter: parent.verticalCenter
        }

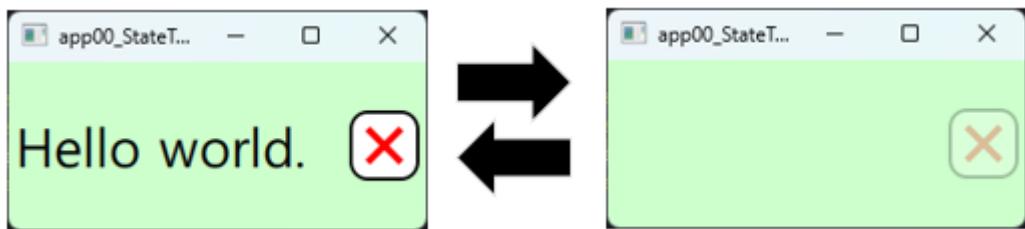
        Image {
            id: clearButton
            source: "images/clear.svg"
            anchors.right: parent.right
            anchors.rightMargin: 4
            anchors.verticalCenter: textField.verticalCenter

            MouseArea {
                anchors.fill: parent
                onClicked: textField.text = ""
            }
        }
    }

    states: [
        State {
            name: "with text"
            when: textField.text !== ""
        }
    ]
}
```

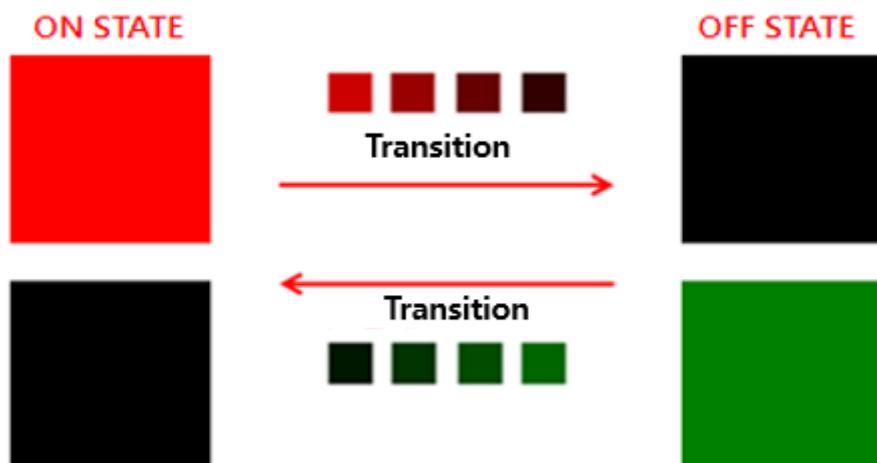
예수님은 당신을 사랑합니다.

```
PropertyChanges { target: closeButton; opacity: 1.0 }
},
State {
    name: "without text"
    when: textField.text === ""
        PropertyChanges { target: closeButton; opacity: 0.25 }
        PropertyChanges { target: textField; focus: true }
    }
]
}
}
```



- Transition에서 Animation 사용

이번 예제에서는 Transition 이 수행 될 때 PropertyAnimation 을 사용해 보도록 하자. PropertyAnimation 의 프로퍼티는 color 프로퍼티로 설정하고 duration 프로퍼티를 1000으로 설정하면 1000 millisecond 동안 color값이 선명해지는 것과 같은 애니메이션 효과를 사용할 수 있다. 첫 번째 Transition은 “off”에서 “on” state 로 변경되도록 지정하고 두 번째 Transition은 “on”에서 “off” state 로 변경되도록 지정하였다.



```
import QtQuick
import QtQuick.Window

Window {
    width: 150; height: 250; visible: true

    Rectangle {
        width: 150; height: 250
        Rectangle {
            id: onElement; x: 25; y: 15; width: 100; height: 100
        }

        Rectangle {
            id: offElement; x: 25; y: 135; width: 100; height: 100
        }

        states: [
            State {
                name: "on"
                PropertyChanges { target: onElement; color: "red" }
                PropertyChanges { target: offElement; color: "black" }
            },
            State {
                name: "off"
                PropertyChanges { target: onElement; color: "black" }
                PropertyChanges { target: offElement; color: "green" }
            }
        ]
    }

    state: "on"

    MouseArea {
        anchors.fill: parent
        onClicked: parent.state === "on" ?
            parent.state = "off" : parent.state = "on"
    }

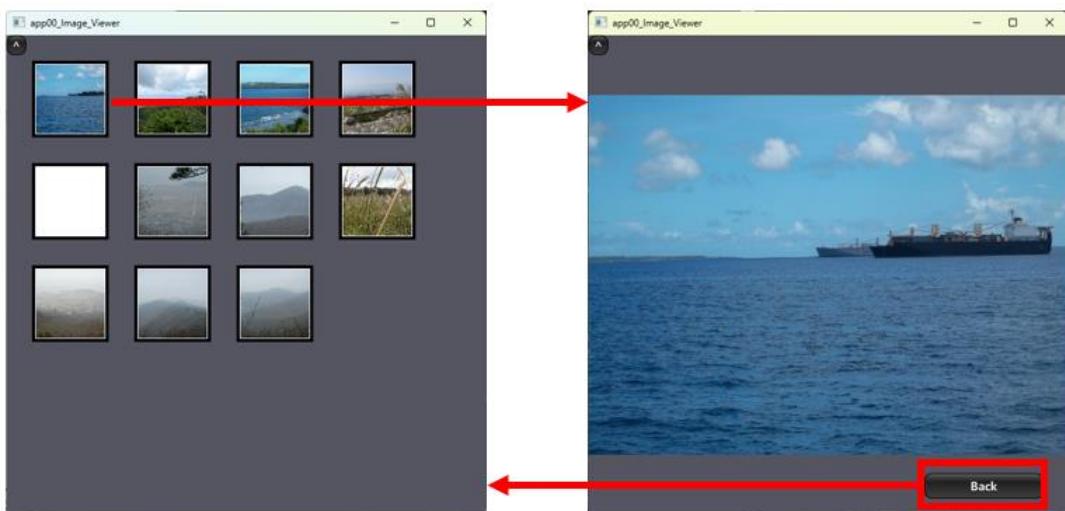
    transitions: [
```

예수님은 당신을 사랑합니다.

```
Transition {
    from: "off"; to: "on"
    PropertyAnimation {
        target: onElement
        properties: "color"
        duration: 1000
    }
},
Transition {
    from: "on"; to: "off"
    PropertyAnimation {
        target: offElement
        properties: "color"
        duration: 1000
    }
}
]
```

### 3.4. Image Viewer 구현

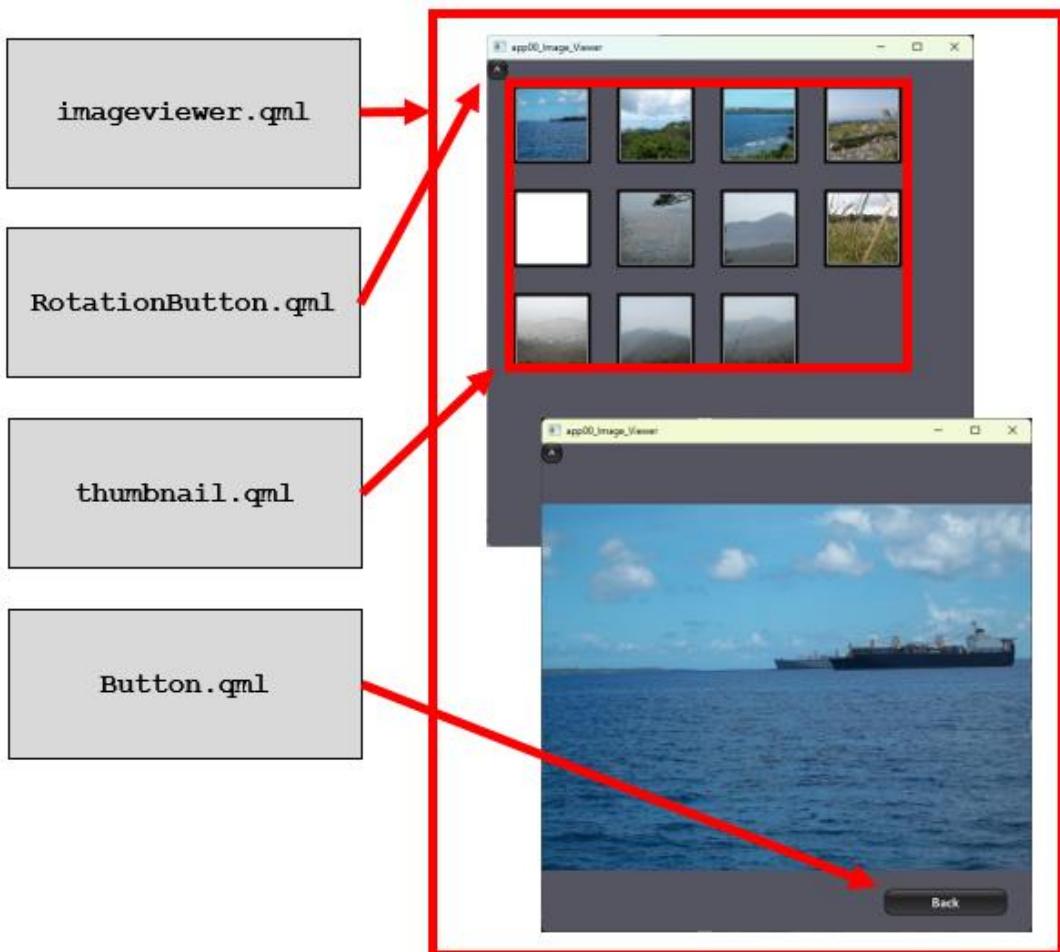
이번 예제는 Image Viewer를 구현해 보도록 하자. Image Viewer 는 메인 화면에서 Thumbnail 이미지(축소해 놓은 상태)를 보는 화면이다. 크게 보기자를 원하는 이미지를 클릭하면 아래 그림에서 보는 것과 같이 이미지를 크게 볼 수 있다.



화면 상단 좌측에 보면 이미지가 표시된 것을 확인할 수 있다. 이 이미지는 화면을 90도 회전할 수 있다. 예를 들어 우리가 핸드폰을 세로방향에서 가로 방향으로 보면 화면이 좌측 또는 우측으로 90도 회전하는 것과 같은 기능을 제공한다.

이와 같이 이번 예제에서도 이와 비슷한 기능을 제공하는 Image Viewer 예제이다. Image Viewer 예제는 다음 그림에서 보는 것과 같이 모두 4개의 QML 소스 파일로 구성된다.

QML 소스 파일 명	설명
ImageViewer.qml	QML 태입을 배치한 메인 QML 소스코드.
RotationButton.qml	화면을 -90도 회전, 다시 클릭하면 0도로 회전하는 기능
Button.qml	버튼, 클릭 시 이전화면으로 돌아감.
Thumbnail.qml	Thumbnail이라는 사용자 정의 타입



`ImageViewer.qml` 에서는 `Thumbnail`, `RotationButton`, `Button` 을 호출해 사용한다. 다음 예제는 `ImageViewer.qml` 예제 소스코드이다.

```
import QtQuick
import QtQuick.Window
import "module"

Window {
    width: 600; height: 600; visible: true

    Item {
        id: screen; width: 600; height: 600
        property int animDuration: 500

        RotationButton {
            id: rotationButton
```

예수님은 당신을 사랑합니다.

```
duration: screen.animDuration
z: 100
anchors.top: screen.top
anchors.left: screen.left
}

Rectangle {
    id: background
    anchors.centerIn: parent; color: "#555560";
    width: rotationButton.angle == 0 ? parent.width : parent.height
    height: rotationButton.angle == 0 ? parent.height : parent.width
    rotation: rotationButton.angle

    Behavior on rotation {
        RotationAnimation {
            duration: screen.animDuration
            easing.type: Easing.InOutQuad
        }
    }
}

Item {
    id: grid
    anchors.fill: parent

    function displayPicture(path) {
        picture.source = path
        screen.state = "displayPicture"
    }

    Thumbnail {
        column: 0; row: 0; image: "images/101.JPG"
        onClicked: parent.displayPicture(image)
    }
    Thumbnail {
        column: 1; row: 0; image: "images/102.JPG"
        onClicked: parent.displayPicture(image)
    }
    Thumbnail {
        column: 2; row: 0; image: "images/103.JPG"
```

예수님은 당신을 사랑합니다.

```
        onClicked: parent.displayPicture(image)
    }
Thumbnail {
    column: 3; row: 0; image: "images/104.JPG"
    onClicked: parent.displayPicture(image)
}
Thumbnail {
    column: 0; row: 1; image: "images/105.JPG"
    onClicked: parent.displayPicture(image)
}
Thumbnail {
    column: 1; row: 1; image: "images/106.JPG"
    onClicked: parent.displayPicture(image)
}
Thumbnail {
    column: 2; row: 1; image: "images/107.JPG"
    onClicked: parent.displayPicture(image)
}
Thumbnail {
    column: 3; row: 1; image: "images/108.JPG"
    onClicked: parent.displayPicture(image)
}
Thumbnail {
    column: 0; row: 2; image: "images/109.JPG"
    onClicked: parent.displayPicture(image)
}
Thumbnail {
    column: 1; row: 2; image: "images/110.JPG"
    onClicked: parent.displayPicture(image)
}
Thumbnail {
    column: 2; row: 2; image: "images/111.JPG"
    onClicked: parent.displayPicture(image)
}
}

Image {
    id: picture
    z: 2
```

예수님은 당신을 사랑합니다.

```
x: 2 * parent.width; y: 0
width: parent.width; height: parent.height
smooth: true
fillMode: Image.PreserveAspectFit
}

Button {
    id: backButton
    width: 150
    height: 32

    x: parent.width - width - 30
    y: parent.height + 3 * height
    z: 5

    text: "Back"
    onClicked: screen.state = "displayGrid"
    visible: false
}
}

state: "displayGrid"

states: [
    State {
        name: "displayGrid"
        PropertyChanges { target: background; color: "#555560" }
    },
    State {
        name: "displayPictures"
        PropertyChanges { target: background; color: "black" }
    }
]

transitions: [
    Transition {
        from: "displayGrid"; to: "displayPicture"
        PropertyAnimation {
            target: backButton; properties: "visible"; to: true
        }
    }
]
```

```
        }
        NumberAnimation {
            target: grid
            properties: "scale"; to: 0.5
        }
        NumberAnimation {
            target: grid
            property: "opacity"; to: 0.0
            duration: screen.animDuration
            easing.type: Easing.InOutQuad
        }
        NumberAnimation {
            target: picture
            properties: "x"; to: 0
            duration: screen.animDuration
            easing.type: Easing.InOutQuad
        }
        NumberAnimation {
            target: backButton
            properties: "y"
            to: background.height - backButton.height - 20
            duration: screen.animDuration * 2
            easing.type: Easing.OutBounce
        }
    },
    Transition {
        from: "displayPicture"; to: "displayGrid"
        SequentialAnimation {
            ParallelAnimation {
                NumberAnimation {
                    target: picture
                    properties: "x"; to: 2 * screen.width;
                    easing.type: Easing.InOutQuad
                }
                NumberAnimation {
                    target: backButton
                    properties: "y"
                    to: background.height + 3 * backButton.height
                    duration: screen.animDuration * 2
                }
            }
        }
    }
},
```

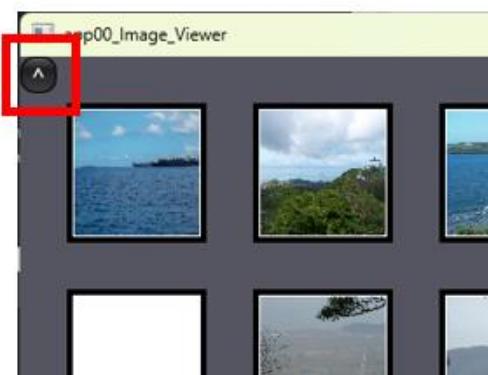
예수님은 당신을 사랑합니다.

```
        easing.type: Easing.InBack
    }
}

PauseAnimation { duration: screen.animDuration / 2 }

ParallelAnimation {
    NumberAnimation {
        target: grid
        properties: "scale"; to: 1.0
        duration: screen.animDuration
        easing.type: Easing.InOutQuad
    }
    NumberAnimation {
        target: grid
        properties: "opacity"; to: 1.0
        duration: screen.animDuration
        easing.type: Easing.InOutQuad
    }
}
]
}
}
```

RotationButton.qml 소스 파일은 상단 좌측에 위치한다. 이 버튼을 클릭하면 -90 도 회전 한다. -90 도 회전한 상태에서 이 버튼을 클릭하면 원래 상태인 0 도로 회전 한다.



다음 예제는 RotationButton.qml 예제 소스 코드이다.

```
import QtQuick
```

예수님은 당신을 사랑합니다.

```
Item {
    id: container; width: 50; height: 50

    property int angle: 0
    property int duration: 250

    Button {
        id: button
        text: "^"
        width: 25; height: 25;
        x: 0; y: 0
        onClicked: {
            container.angle = (container.angle == 0 ? -90 : 0)
        }
    }

    states: [
        State {
            name: "normal"
            when: container.angle == 0
        },
        State {
            name: "rotated"
            when: container.angle == -90
        }
    ]
}

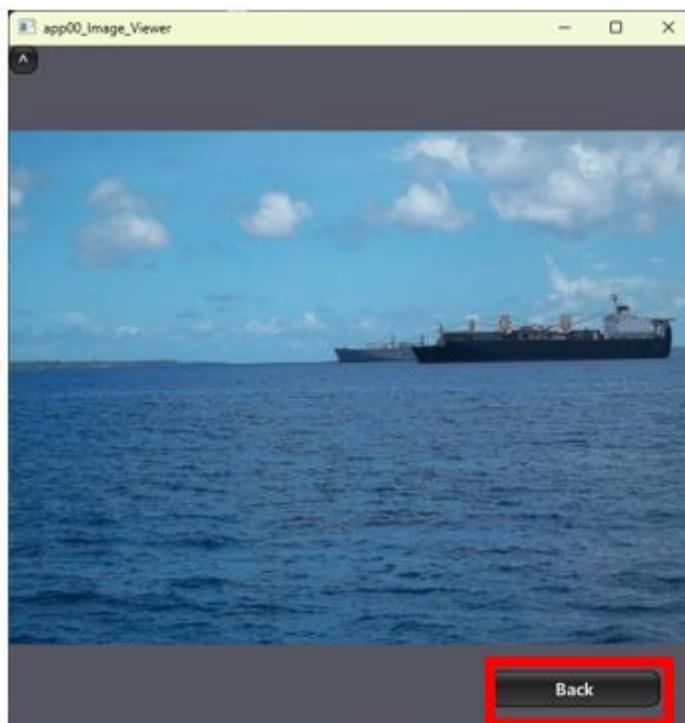
state: "normal"

transitions: [
    Transition {
        from: "normal"; to: "rotated"
        NumberAnimation {
            targets: button
            properties: "rotation"; to: -90
            duration: 200
        }
    },
    Transition {
        from: "rotated"; to: "normal"
    }
]
```

예수님은 당신을 사랑합니다.

```
NumberAnimation {
    targets: button
    properties: "rotation"; to: 0
    duration: 200
}
]
}
}
```

Button.qml 은 이미지 크게 보기 화면에서 Thumbnail.qml 화면으로 되돌린다.



다음 예제는 Button.qml 예제 소스코드 이다.

```
import QtQuick

Item {
    id: container

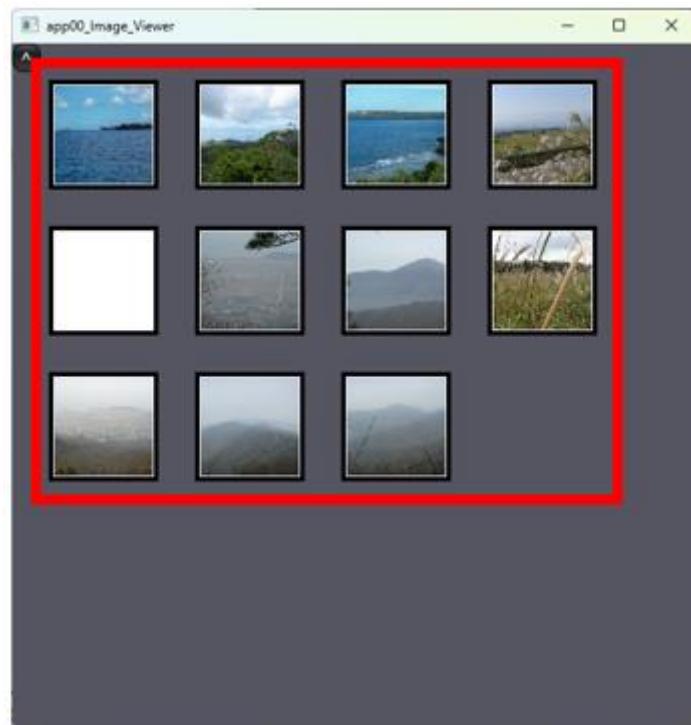
    signal clicked

    property string text
```

```
Rectangle {  
    id: background  
    anchors.fill: parent  
    border.color: mouseRegion.pressed ? "gray" : "black"  
    smooth: true  
    radius: 10  
    gradient: Gradient {  
        GradientStop { position: 0.0; color: "#606060" }  
        GradientStop { position: 0.33; color: "#202020" }  
        GradientStop { position: 1.0; color: "#404040" }  
    }  
}  
Text {  
    color: "white"  
    anchors.centerIn: background;  
    font.bold: true;  
    font.pixelSize: 15  
    text: container.text; style: Text.Raised; styleColor: "black"  
}  
MouseArea {  
    id: mouseRegion  
    anchors.fill: parent  
    onClicked: container.clicked()  
}  
}
```

Thumbnail.qml 소스 파일은 아래 그림에서 보는 것과 원래 이미지 크기를 축소해 여러 개의 이미지를 한 화면에 보여주기 위한 기능이다.

예수님은 당신을 사랑합니다.



다음 예제 소스코드는 Thumbnail.qml 소스이다.

```
import QtQuick

Item {
    id: container

    signal clicked

    property int column
    property int row
    property string image

    width : 96
    height : 96

    x: 32 + column * (width + 32)
    y: 32 + row * (width + 32)

    Rectangle {
        color: "black"
```

예수님은 당신을 사랑합니다.

```
anchors.fill: parent
}

Rectangle {
    x: 4
    y: 4
    width: parent.width - 8
    height: parent.height - 8
    color: "white"
    smooth : true
}

Image {
    x: 5; y: 5
    width: parent.width - 10
    height: parent.height - 10

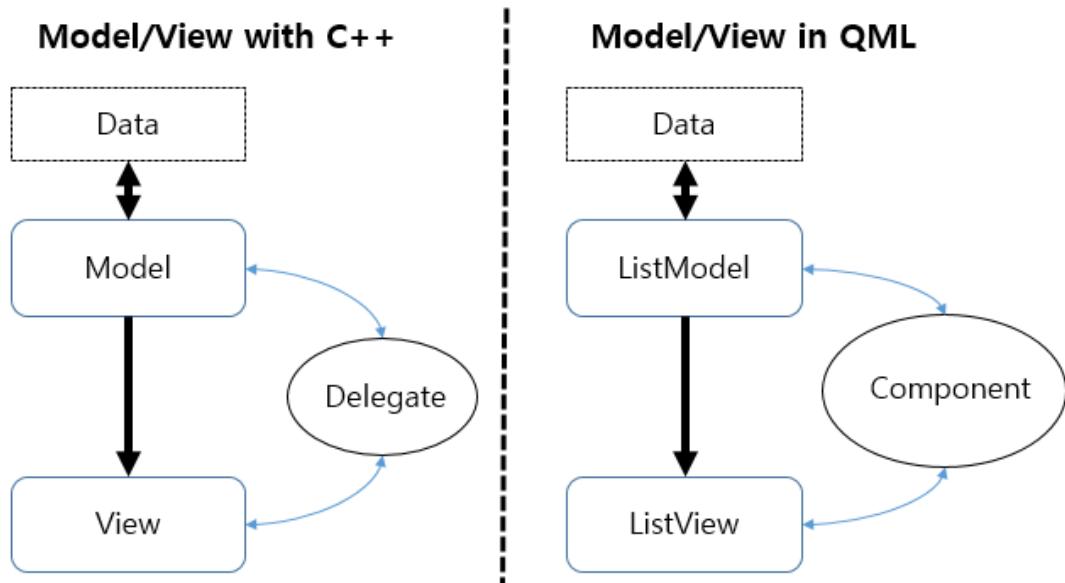
    source: "../" + container.image
    fillMode: Image.PreserveAspectCrop
}

MouseArea {
    anchors.fill: parent
    onClicked: parent.clicked()
}
}
```

이번 장에서 다룬 예제는 00\_Image\_Viewer 디렉토리를 참조하면 된다.

## 4. Model and View

Qt/C++에서 Model/View를 사용한 것과 같이 QML에서도 Model/View를 사용한다. 다음 그림은 C++에서 사용하던 Model/View 방식과 QML에서 방식의 Model / View 방식을 비교한 그림이다.



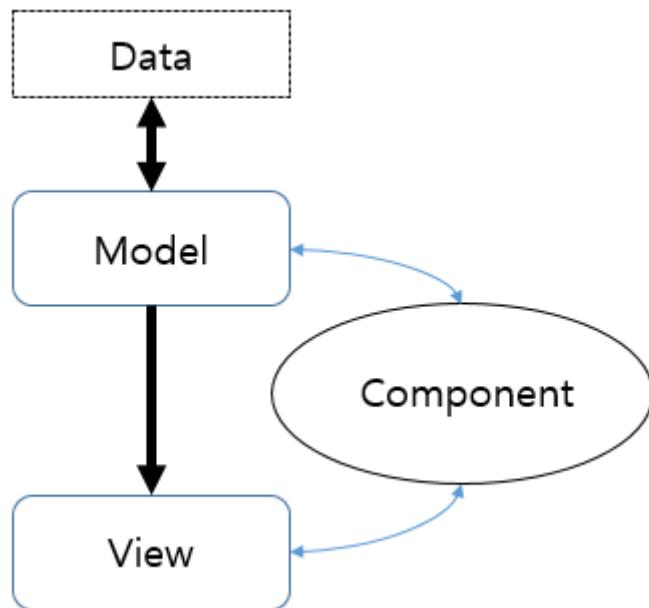
위의 그림에서 보는 것과 같이 C++ 방식과 QML 방식은 개념 및 사용 방법이 동일하다. 이번 장에서는 Model /View를 사용하는 방법에 대해서 살펴본 후 간단한 응용 예제를 통해 사용방법에 대해서 자세히 알아 보도록 하자.

## 4.1. Model/View 를 이용한 데이터 표현

많은 양의 데이터를 표현하기 위해서 표를 사용하는 것과 같이 QML에서 사용자에게 표와 같은 형태 또는 리스트 형태로 많은 양의 데이터를 표현하는 데 Model / View를 사용한다.

Model은 데이터를 담는 하나의 컨테이너로 보아도 무방하다. 그리고 View는 데이터가 담겨있는 컨테이너를 사용자에게 보여주는 표 또는 리스트와 같은 도구와 같다.

예를 들어 Model / View 개념에서는 데이터를 삽입할 때 View에 직접 삽입하지 않는다. 또한 데이터 수정 시 View에 표시된 데이터를 수정하지 않는다. 데이터의 삽입/수정/삭제는 Model에서 이루어 진다. View는 Model과 연결되어 Model에 담겨 있는 데이터를 보여주는 역할과 사용자 이벤트를 Model로 전달하는 역할을 한다.



- **ListModel 과 ListView**

ListModel은 ListView에 표시하고자 하는 데이터를 담는 Container와 같은 기능을 제공한다. ListModel은 한 개 이상의 ListElement를 가질 수 있다. 그리고 ListModel과 ListView 사이에서 Component를 사용한다. Component는 ListView에 데이터를 표현하는 형태 또는 스타일을 지정할 수 있다. 다음은 ListModel, ListView 그리고

예수님은 당신을 사랑합니다.

Component 를 사용한 예제 이다.

```
import QtQuick
import QtQuick.Window

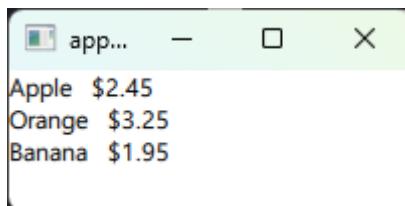
Window {
    width: 200; height: 70; visible: true

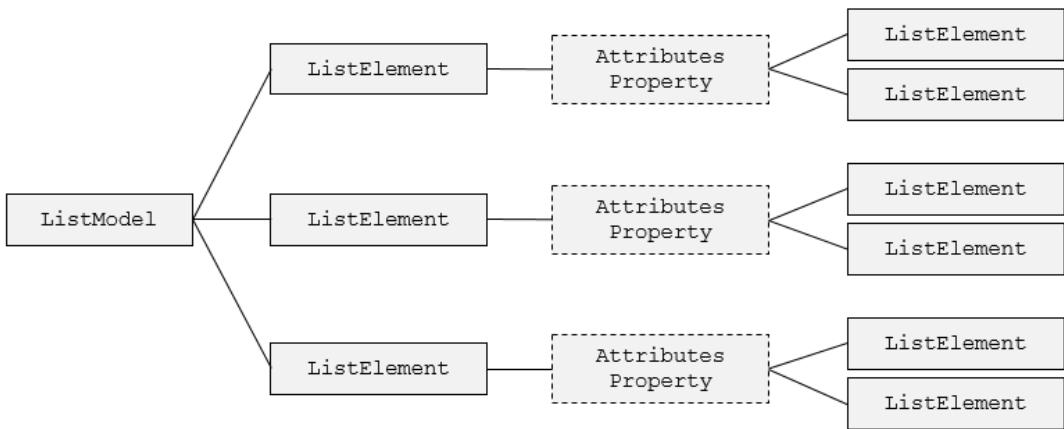
    ListModel {
        id: fruitModel
        ListElement { name: "Apple"; cost: 2.45 }
        ListElement { name: "Orange"; cost: 3.25 }
        ListElement { name: "Banana"; cost: 1.95 }
    }

    Component {
        id: fruitDelegate
        Row {
            spacing: 10
            Text { text: name }
            Text { text: '$' + cost }
        }
    }

    ListView {
        anchors.fill: parent
        model: fruitModel
        delegate: fruitDelegate
    }
}
```

위의 예제 에서 보는 것과 같이 ListModel 은 데이터를 추가 하기 위해서 한 개 이상의 ListElement를 사용할 수 있다. 또한 다음 그림에서 보는 것과 같이 ListElement 는 하위에 ListElement를 배치 할 수 있다.





- ListElement 의 attributes 프로퍼티의 사용

다음 예제는 ListElement 의 하위에 ListElement 를 사용하기 위해서 attributes 를 사용한 예제이다.

```

import QtQuick
import QtQuick.Window

Window {
    width: 200; height: 200; visible: true

    ListModel {
        id: fruitModel

        ListElement {
            name: "Apple"; cost: 2.45
            attributes: [
                ListElement { description: "Delicious" },
                ListElement { description: "Expensive" }
            ]
        }
        ListElement {
            name: "Orange"; cost: 3.25
            attributes: [
                ListElement { description: "Expensive" }
            ]
        }
        ListElement {
    
```

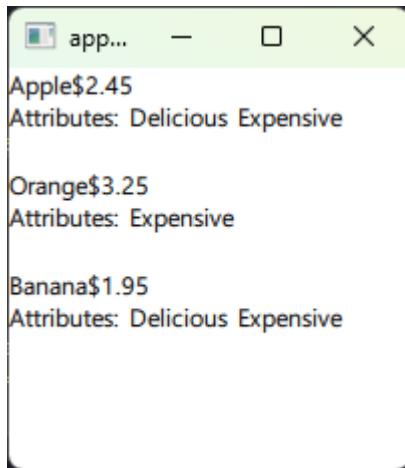
예수님은 당신을 사랑합니다.

```
name: "Banana"; cost: 1.95
attributes: [
    ListElement { description: "Delicious" },
    ListElement { description: "Expensive" }
]
}
}

Component {
    id: fruitDelegate
    Item {
        width: 200; height: 50
        Text { id: nameField; text: name }
        Text { text: '$' + cost; anchors.left: nameField.right }
        Row {
            anchors.top: nameField.bottom
            spacing: 5
            Text { text: "Attributes:" }
            Repeater {
                model: attributes
                Text { text: description }
            }
        }
    }
}

ListView {
    anchors.fill: parent
    model: fruitModel
    delegate: fruitDelegate
}
}
```

예수님은 당신을 사랑합니다.



- ListView 의 header 와 footer 프로퍼티

ListView 에서 header 프로퍼티는 상단의 스타일을 직접 지정해 사용할 수 있다. footer 는 하단의 스타일을 지정할 수 있다. 다음 예제는 header 와 footer 를 사용한 예제 소스코드 이다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 400; height: 240; visible: true; color: "white"
    id: root

    ListModel {
        id: nameModel
        ListElement { name: "Alice"; }
        ListElement { name: "Bob"; }
        ListElement { name: "Jane"; }
        ListElement { name: "Victor"; }
        ListElement { name: "Wendy"; }
    }
    Component {
        id: nameDelegate
        Text {
            text: name;
            font.pixelSize: 24
            anchors.left: parent.left
        }
    }
}
```

예수님은 당신을 사랑합니다.

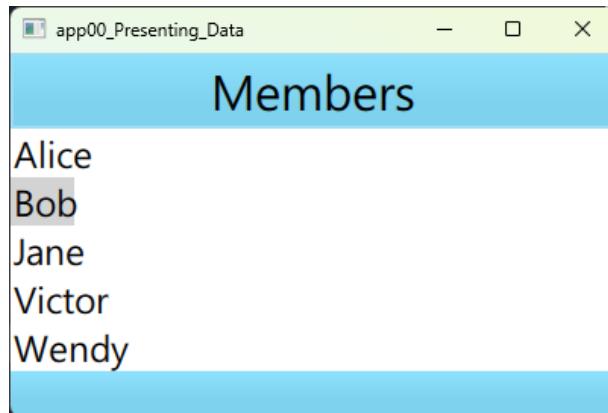
```
        anchors.leftMargin: 2
    }
}

Component {
    id: bannercomponent
    Rectangle {
        id: banner
        width: root.width; height: 50
        gradient: clubcolors
        border {color: "#9EDDF2"; width: 2}
        Text {
            anchors.centerIn: parent
            text: "Members"
            font.pixelSize: 32
        }
    }
}

Gradient {
    id: clubcolors
    GradientStop { position: 0.0; color: "#8EE2FE"}
    GradientStop { position: 0.66; color: "#7ED2EE"}
}

ListView {
    anchors.fill: parent
    clip: true
    model: nameModel
    delegate: nameDelegate
    header: bannercomponent
    footer: Rectangle {
        width: parent.width; height: 30;
        gradient: clubcolors
    }
    highlight: Rectangle {
        color: "lightgray"
    }
    focus: true // Properties for moving menus with keyboard events
```

```
}
```



- GridView

GridView 는 ListElement 를 Grid 스타일로 표시할 수 있다. 다음 예제는 GridView 를 사용한 예제이다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 300; height: 200; visible: true

    ListModel {
        id: gridModel
        ListElement {
            name: "Picture 1"; frame: "images/101.JPG"
        }
        ListElement {
            name: "Picture 2"; frame: "images/102.JPG"
        }
        ListElement {
            name: "Picture 3"; frame: "images/103.JPG"
        }
        ListElement {
            name: "Picture 4"; frame: "images/104.JPG"
        }
    }
}
```

예수님은 당신을 사랑합니다.

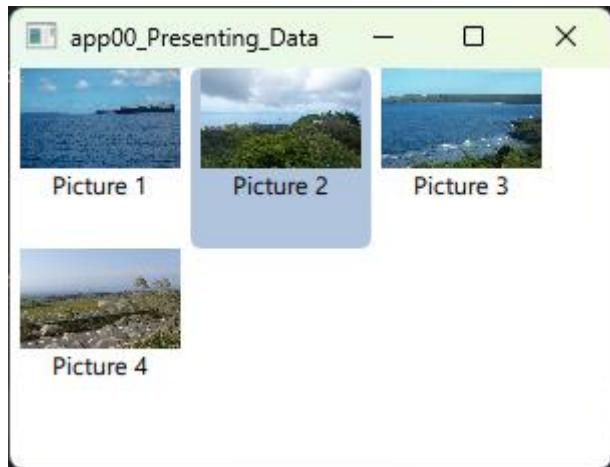
```
        }

    }

Component {
    id: contactDelegate
    Item {
        width: grid.cellWidth; height: grid.cellHeight
        Column {
            anchors.fill: parent
            Image {
                width: 80; height: 50
                source: frame;
                anchors.horizontalCenter: parent.horizontalCenter
            }
            Text {
                text: name;
                anchors.horizontalCenter: parent.horizontalCenter
            }
        }
    }
}

GridView {
    id: grid
    anchors.fill: parent
    cellWidth: 90; cellHeight: 90

    model: gridModel
    delegate: contactDelegate
    highlight:
        Rectangle {
            color: "lightsteelblue"; radius: 5
        }
    focus: true
}
}
```



위의 그림에서 보는 것과 같이 예제가 실행 되면 키보드 방향키로 이동하면 역상 된 영역이 움직이는 것을 확인할 수 있다.

- GridView에서 Animation 사용

키보드의 방향키를 움직일 때 Animation 을 사용할 수 있다. 다음은 GridView 에 Animation 을 사용한 예제이다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 300; height: 200; visible: true

    ListModel {
        id: gridModel
        ListElement {
            name: "Picture 1"; frame: "images/101.JPG"
        }
        ListElement {
            name: "Picture 2"; frame: "images/102.JPG"
        }
        ListElement {
            name: "Picture 3"; frame: "images/103.JPG"
        }
        ListElement {
```

예수님은 당신을 사랑합니다.

```
        name: "Picture 4"; frame: "images/104.JPG"
    }
}

Component {
    id: highlight
    Rectangle {
        width: view.cellWidth; height: view.cellHeight
        color: "lightsteelblue"; radius: 5
        x: view.currentItem.x
        y: view.currentItem.y
        Behavior on x { SpringAnimation { spring: 3; damping: 0.2 } }
        Behavior on y { SpringAnimation { spring: 3; damping: 0.2 } }
    }
}

GridView {
    id: view
    width: 300; height: 200
    cellWidth: 80; cellHeight: 80

    model: gridModel
    delegate: Column {
        Image {
            width: 60; height: 50; source: frame;
            anchors.horizontalCenter: parent.horizontalCenter
        }
        Text {
            text: name;
            anchors.horizontalCenter: parent.horizontalCenter
        }
    }
    highlight: highlight

    // Set the current item's size to match
    highlightFollowsCurrentItem: true

    focus: true
}
```

}

## ● XML Model

QML에서는 XML 데이터를 표현하기 위해서 XmlListModel 을 제공한다. XmlListModel 은 source 프로퍼티를 사용해 xml 파일을 지정하거나 인터넷 URL 을 지정해 원격으로 XML 데이터를 가져올 수 있다. 다음 예제는 XML 파일로부터 데이터를 읽어와 표시하는 예제 이다.

```
import QtQuick
import QtQuick.Window
import QtQml.XmlListModel

Window {
    width: 400; height: 200; visible: true; color: "#404040"

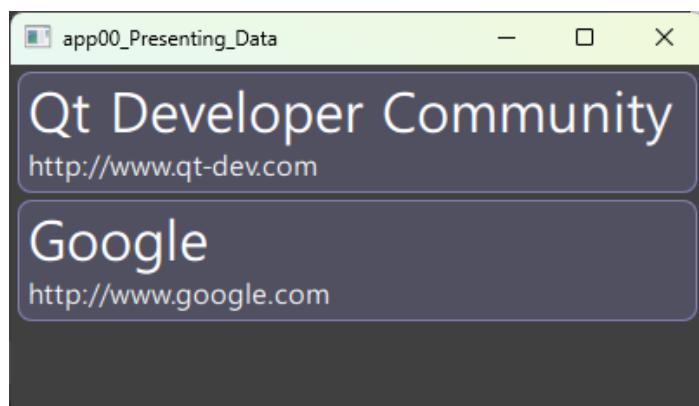
    XmlListModel {
        id: xmlModel
        source: "item.xml"
        query: "/items/item"

        XmlListModelRole { name: "title"; attributeName: "title" }
        XmlListModelRole { name: "link"; attributeName: "link" }
    }

    Component {
        id: xmlDelegate
        Item {
            width: parent.width; height: 74
            Rectangle {
                width: Math.max(childrenRect.width + 16, parent.width)
                height: 70; clip: true
                color: "#505060"; border.color: "#8080b0"; radius: 8
                Column {
                    Text {
                        x: 6; color: "white"
                        font.pixelSize: 32; text: title
                    }
                    Text {
```

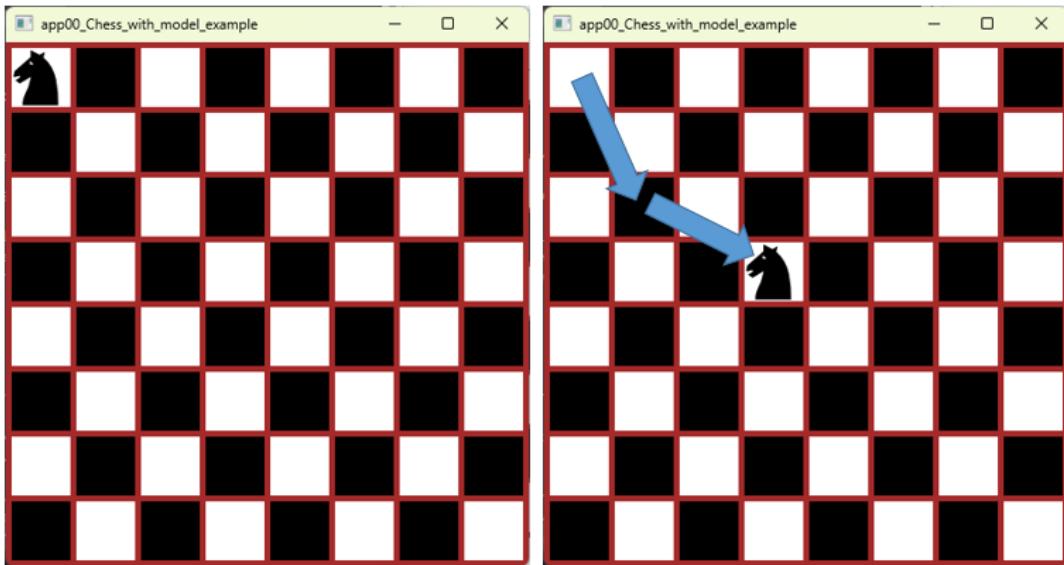
예수님은 당신을 사랑합니다.

```
        x: 6; color: "white"  
        font.pixelSize: 16; text: link  
    }  
}  
}  
}  
}  
  
ListView {  
    anchors.fill: parent; anchors.margins: 4; model: xmlModel  
    delegate: xmlDelegate  
}  
}
```



## 4.2. Chess Knight

이번 장에서는 Repeater 와 Mode/View를 이용해 체스를 구현해 보도록 하자. 다음 그림은 Knight 가 체스에서 이동 규칙에 따라 이동할 수 있다.



### ● [1 단계] 체스판 구현

체스판 모양을 구현하기 위해서 가로 8 칸 세로 8 칸을 Repeater 를 이용해 구현한다. 다음은 체스판 모양을 구현한 예제 소스코드이다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 445; height: 445; color: "brown"; visible: true

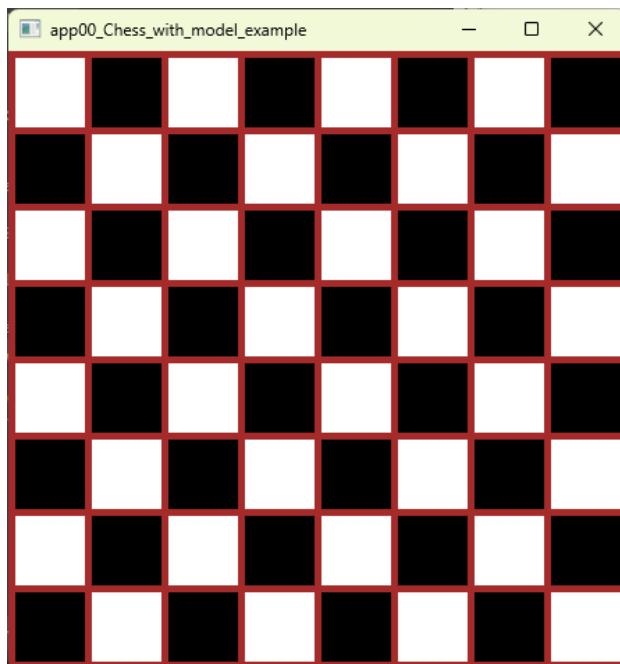
    Grid {
        x: 5; y: 5
        rows: 8; columns: 8; spacing: 5

        Repeater {
            model: parent.rows * parent.columns
```

예수님은 당신을 사랑합니다.

```
Rectangle {
    width: 50; height: 50
    color: {
        var row = Math.floor(index / 8);
        var column = index % 8

        // Even 1, otherwise 0
        if ((row + column) % 2 == 1)
            "black";
        else
            "white";
    }
}
}
```



### ● [2 단계] Knight 구현

이번에는 체스판 위에 Knight 를 배치해 보자. 체스판 위에 Knight 를 배치하기 위해서 Image 타입을 사용하였다. 다음 예제 소스코드는 Knight 를 배치한 예제 소스코드 이다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 445; height: 445; color: "brown"; visible: true

    Grid {
        x: 5; y: 5
        rows: 8; columns: 8; spacing: 5

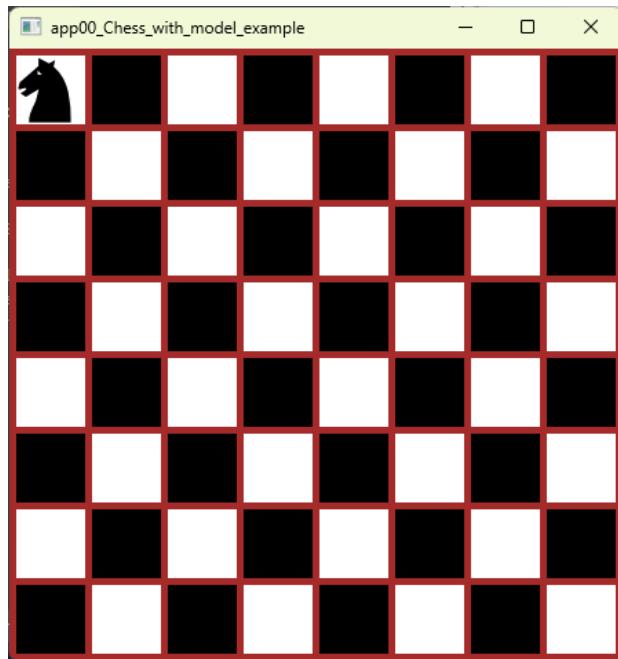
        Repeater {
            model: parent.rows * parent.columns
            Rectangle {
                width: 50; height: 50
                color: {
                    var row = Math.floor(index / 8);
                    var column = index % 8
                    if ((row + column) % 2 == 1)
                        "black";
                    else
                        "white";
                }
            }
        }
    }

    Image {
        id: knight
        property int cx
        property int cy

        source: "./images/knight.png"

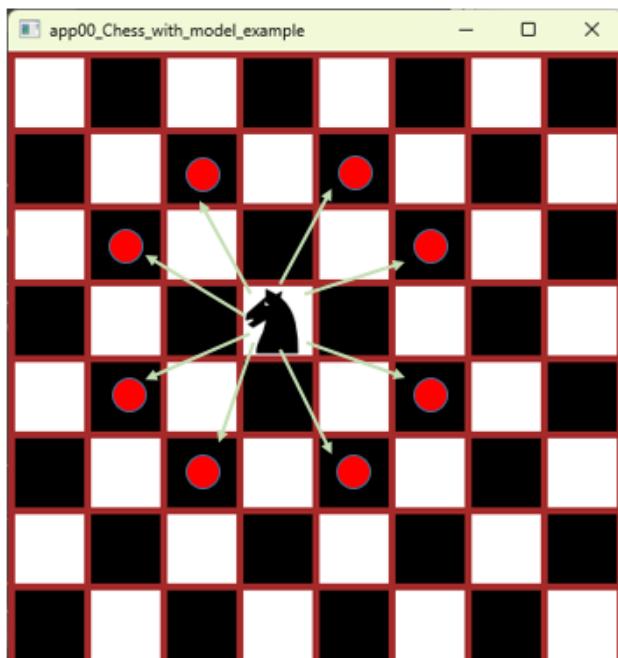
        x: 5 + 55 * cx
        y: 5 + 55 * cy
    }
}
```

예수님은 당신을 사랑합니다.



- [3 단계] Knight 이동

이번 단계에서는 배치한 Knight 를 마우스를 이용해 이동하는 기능을 구현해 보도록 하자. 다음 그림에서 보는 것과 같이 Knight 는 이동할 수 있는 규칙으로 앞으로 2 칸, 옆으로 1 칸 또는 옆으로 1 칸, 앞으로 2 칸을 이동할 수 있다.



- Knight 가 이동할 수 있는 규칙 및 기능 구현에 필요 값 계산

- cx, cy 는 Knight 의 위치
- x, y 는 현재 Knight 의 위치
- 앞으로 2칸, 옆으로 1칸 또는 옆으로 1칸, 앞으로 2칸으로 이동 가능
- $(x-cx) = 2, (y-cy) = 1$  or  $(x-cx) = 1, (y-cy) = 2$
- 음수 값이 나올 수 있으므로 위의 계산 식에 양수 값만 나오도록 해야 함.

위의 구현에 필요한 값 계산에 따라 다음과 같이 Knight 가 이동할 위치가 가능한지 판단하기 위해 다음과 같이 구현할 수 있다.

```
if ((Math.abs(x - knight.cx) == 1 && Math.abs(y - knight.cy) == 2) ||
    (Math.abs(x - knight.cx) == 2 && Math.abs(y - knight.cy) == 1)) {
    knight.cx = x;
    knight.cy = y;
}
```

다음 예제 소스코드는 체스의 Knight 이동 규칙을 적용한 전체 예제 소스코드이다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 445; height: 445; color: "brown"; visible: true

    Grid {
        x: 5; y: 5
        rows: 8; columns: 8; spacing: 5

        Repeater {
            model: parent.rows * parent.columns
            Rectangle {
                width: 50; height: 50
                color: {
                    var row = Math.floor(index / 8);
                    var column = index % 8
                    if ((row + column) % 2 == 1)
                        "black";
                    else
                        "white";
                }
            }
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
        "white";
    }
MouseArea {
    anchors.fill: parent
    onClicked: {
        var x = index % 8;
        var y = Math.floor(index/8);

        // Knight moves x, y = 2, 1 or x, y = 1, 2
        if ((Math.abs(x - knight.cx) == 1 &&
            Math.abs(y - knight.cy) == 2) ||
            (Math.abs(x - knight.cx) == 2 &&
            Math.abs(y - knight.cy) == 1)) {

            knight.cx = x;
            knight.cy = y;
        }
    }
}
}

Image {
    id: knight
    property int cx
    property int cy

    source: "./images/knight.png"

    x: 5 + 55 * cx
    y: 5 + 55 * cy
}
}
```

이번 장에서 다른 예제는 00\_Chess\_with\_model\_example 디렉토리를 참조하면 된다.

## 5. Integration QML and C++

이번 장에서는 C++ 과 QML간의 데이터 전달과 이벤트를 전달하는 방법에 대해서 알아볼 것이다.

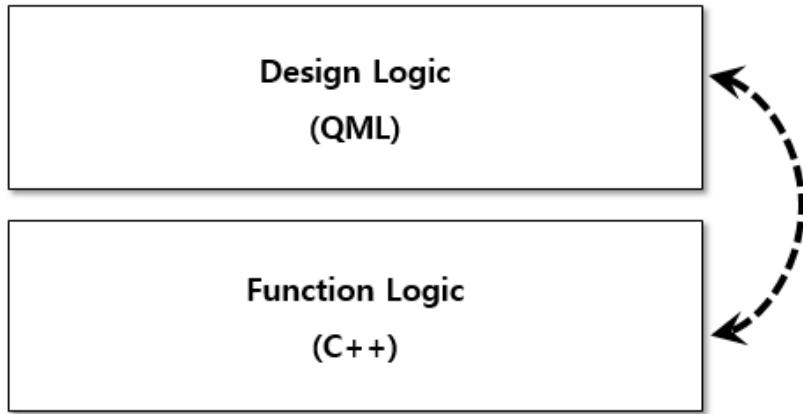
C++과 QML 간의 Interaction 이란 QML 에서 동적으로 데이터를 표시해야 한다고 가정해 보자. 예를 들어 채팅 프로그램에서 새로운 사용자가 접속하면 새로 접속한 사용자의 정보를 QML로 구현된 GUI 상에 표시해야 할 경우 C++에서 새로 접속한 정보를 QML로 전달해야 한다.

이런 경우에 C++ 에서 QML 로 또는 QML 에서 C++로 정보를 전달해야 한다. 또한 QML 상에서 어떤 특정 버튼을 클릭하는 이벤트가 발생했을 때 C++의 어떤 특정 클래스의 멤버 함수가 호출해야 하는 방법에 대해서도 살펴볼 것이다.

따라서 이번 장에서는 QML과 C++간의 데이터 통신 및 이벤트를 전달하는 방법에 대해서 살펴볼 것이다.

## 5.1. Overview

QML 과 C++ 간의 Interaction 이란 아래 그림에서와 같이 QML과 Qt간의 데이터 통신 및 이벤트를 전달하기 위한 방법을 말한다. QML로 Design Logic (GUI)를 구현하고 C++로 Function Logic 을 구현해 QML과 C++간의 Interaction을 위한 기능을 제공한다.



QML로 작성한 Design Logic과 C++로 작성한 Function Logic 간의 데이터 통신을 위해 Qt Meta-Object System을 제공한다.

QML 타입의 프로퍼티를 C++과 통신하기 위해서 다음과 같이 `Q_PROPERTY()`를 사용한다. QML에서 C++ 클래스의 멤버 함수를 호출하기 위해서 `Q_INVOKABLE`을 사용한다. 그리고 QML에서 이벤트가 발생하면 C++에서 접근자 public slot으로 정의한 SLOT 함수를 실행 할 수 있다.

- **QML 타입의 프로퍼티를 C++과 통신하기 위한 기능 구현**

```
Q_PROPERTY(...)
```

- **QML 타입에서 C++ 클래스의 멤버 함수 호출**

```
Q_INVOKABLE
```

- **QML에서 이벤트가 발생하면 C++ Slot 함수 호출**

```
public slots:  
    void refresh()
```

예수님은 당신을 사랑합니다.

QML 과 C++간의 데이터 통신 이벤트 전달을 위한 경우로 다음과 같이 4 가지로 정리할 수 있다.

### ① QML 태입에서 C++ 함수를 호출해 결과 값을 가져오기

```
Rectangle
{
    width : 300; height: 300
    Text {
        text: ( C++ 코드에 구현된 QString을 리턴 하는 함수 호출 )
        ...
    }
    ...
}
```

### ② QML 태입에서 특정 값을 C++ 함수를 호출해 리턴 값을 가져오기

```
Text {
    ...
    Component.onCompleted:
    {
        msg.author = "Hello" // C++ 코드에서 author(QString str) 함수 호출
    }
}
```

### ③ QML 에서 이벤트가 발생하면 C++ 에서 구현한 Slot 함수를 호출하기

```
MouseArea {
    anchors.fill: parent

    onClicked:
    {
        var str = "Who are you ?"
        var result = msg.postMessage(str) // C++ 코드에서 public slots
                                         // 접근자에서 정의한 SLOT함수 호출
        ...
    }
}
```

예수님은 당신을 사랑합니다.

#### ④ QML에서 C++ 클래스의 멤버 함수를 호출하기

```
MouseArea
{
    anchors.fill: parent
    onClicked:
    {
        ...
        msg.refresh(); // C++ 코드에서 public의 정의한 함수 호출
    }
}
```

#### ● Q\_PROPERTY의 Syntax

Q\_PROPERTY는 C++에서 사용하며 QML 타입의 프로퍼티를 C++과 통신하기 위한 기능 구현 시 사용한다. Q\_PROPERTY의 Syntax는 다음과 같다.

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

예를 들어 다음과 같이 Q\_PROPERTY를 사용하였다고 가정해 보자.

```
Q_PROPERTY(QString author READ author WRITE setAuthor NOTIFY authorChanged)
```

READ는 C++에서 QString 값인 author 값을 리턴 하는 함수의 이름을 지정한다. WRITE는 QString 값인 author 변수의 값을 할당한다. 예로 setAuthor(QString author) 함수가 첫 번째 인자 값으로 값을 설정하는 함수를 정의한다. NOTIFY는 SIGNAL 함수를 명시한다. C++에서 Q\_PROPERTY의 선언하는 위치는 다음과 같다.

```
#include <QObject>

class Message : public QObject
{
    Q_OBJECT
    Q_PROPERTY( QString author
                READ author
                WRITE setAuthor
                NOTIFY authorChanged)
    ...
}
```

- **Q\_INVOKABLE**

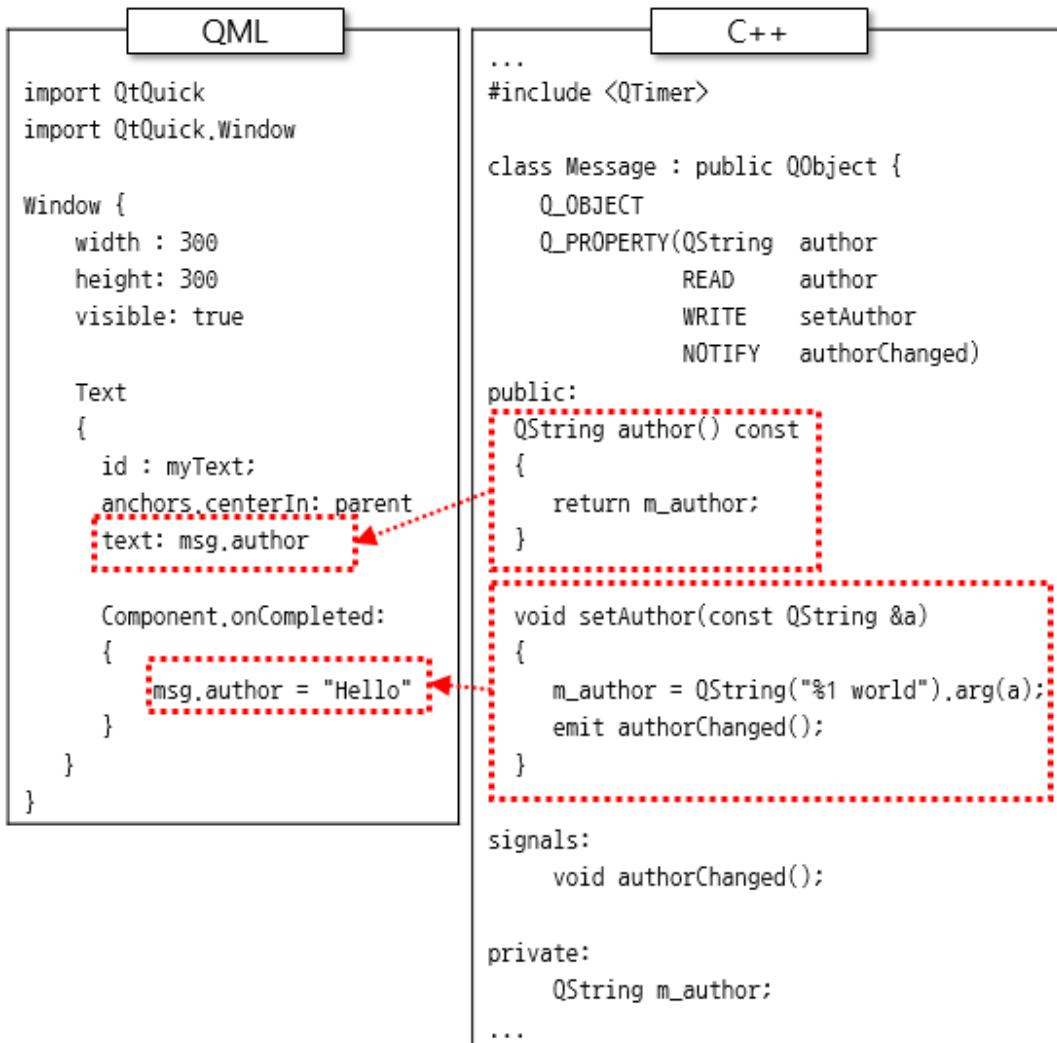
`Q_INVOKABLE` 은 QML에서 C++에서 접근자 `public`으로 선언한 멤버 함수를 호출하기 위한 방법으로 사용한다. 다음은 `Q_INVOKABLE`을 사용한 예제 소스코드이다.

```
#include <QObject>

class Message : public QObject
{
    Q_OBJECT
    ...
public:
    Q_INVOKABLE bool postMessage(const QString &msg)
    {
        ...
        return true;
    }
}
```

- QML 탑입의 특정 프로퍼티의 값을 호출(Exporting)

QML 탑입의 특정 프로퍼티의 값을 호출(Exporting) 하기 위해서 `QObject` 클래스에서 제공하는 `Q_PROPERTY()`를 이용해 정의하면 C++ 클래스의 멤버 함수를 호출 할 수 있다. 예를 들어 QML 탑입 중 `Text` 탑입의 `text` 프로퍼티 값을 할당하기 위해서 다음과 같이 사용할 수 있다.



위의 소스코드에서 보는 것과 같이 C++ Message 클래스의 멤버 함수를 QML에서 호출하기 위해서 QQuickView 클래스를 사용하면 된다. QQuickView 클래스를 예로 다음과 같이 사용할 수 있다.

```

#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlContext>
#include "message.h"

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

```

```
QQmlApplicationEngine engine;

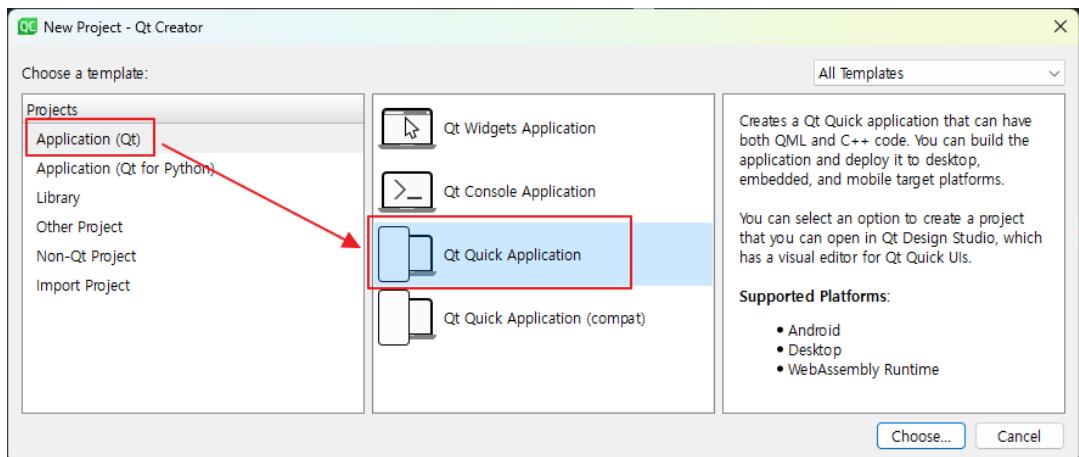
Message msg;
engine.rootContext()->setContextProperty("msg", &msg);

const QUrl url("qrc:/00_Basic_Interaction/Main.qml");
engine.load(url);

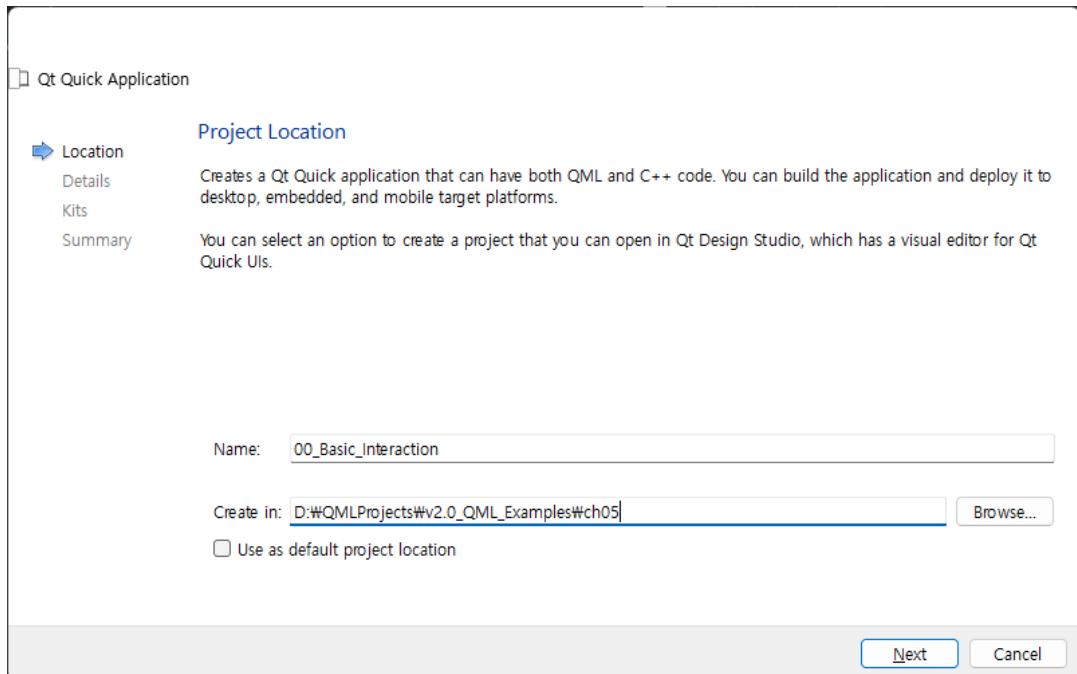
return app.exec();
}
```

### ● QML 과 C++ 간의 Interaction 하기 위한 예제

이번 예제에서는 QML 과 C++ 간의 Interaction 을 원한 예제를 살펴보도록 하자. 위의 예제 소스코드에서 설명했던 것과 같이 QML 에서 Text 타입의 text 프로퍼티의 값을 C++ 클래스의 author() 멤버 함수의 리턴 값을 설정해 보자. 다음 그림에서 보는 것과 같이 프로젝트를 생성한다.



위의 그림에서 보는 것과 같이 [Qt Quick Application]을 선택하고 하단의 [Choose...] 버튼을 클릭한다.



위의 그림에서 보는 것과 같이 프로젝트명과 프로젝트가 위치할 디렉토리를 지정한다. 프로젝트 생성이 완료되면 Message class 를 프로젝트에 추가한다. 그리고 message.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef MESSAGE_H
#define MESSAGE_H

#include <QObject>

class Message : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString author
               READ author
               WRITE setAuthor
               NOTIFY authorChanged)

public:
    explicit Message(QObject *parent = nullptr);

    void setAuthor(const QString &a)
    {
        m_author = QString("%1 world.").arg(a);
    }
}
```

예수님은 당신을 사랑합니다.

```
    emit authorChanged();

}

QString author() const
{
    return m_author;
}

signals:
void authorChanged();

private:
QString m_author;

};

#endif // MESSAGE_H
```

다음으로 message.cpp 소스코드 파일에는 따로 작성하지 않아도 된다. 따라서 아래와 같이 생성자 함수만 존재하는 것을 확인하면 된다.

```
#include "message.h"

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

다음으로 Main.qml 파일을 아래와 같이 작성한다.

```
import QtQuick
import QtQuick.Window

Window {
    width : 300; height: 300; visible: true
    Text {
        id : myText;
        anchors.centerIn: parent
        text: msg.author
        font.pixelSize: 20
    }
}
```

예수님은 당신을 사랑합니다.

```
Component.onCompleted: {
    msg.author = "Hello"
    myText.text = msg.author
}
}
```

다음으로 QML 과 C++ 간의 통신을 가능하기 위해서 main.cpp 파일을 다음과 같이 작성한다.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlContext>
#include "message.h"

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    QQmlApplicationEngine engine;

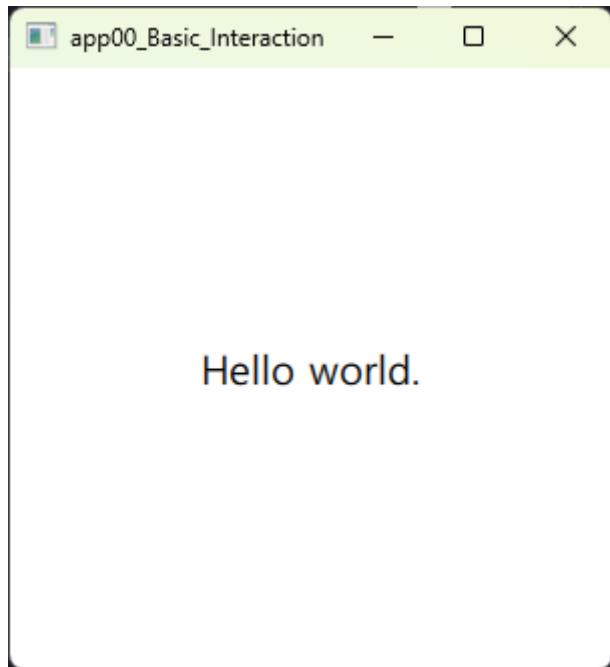
    Message msg;
    engine.rootContext()->setContextProperty("msg", &msg);

    const QUrl url("qrc:/00_Basic_Interaction/Main.qml");
    engine.load(url);

    return app.exec();
}
```

위와 같이 소스코드를 작성하였다면 프로젝트를 빌드하고 실행해 보도록 하자.

예수님은 당신을 사랑합니다.



이 예제의 소스코드는 00\_Basic\_Interaction 디렉토리를 참조하면 된다.

### ● Q\_INVOKABLE 함수 사용 예제

이번 예제에서는 QML에서 마우스를 클릭하면 C++ Message 클래스의 Q\_INVOKABLE로 정의한 Slot 함수를 실행하는 예제를 작성해 보자. 화면을 클릭하면 다음 그림에서 보는 것과 같이 콘솔 디버깅 창에 메시지를 출력한다.

The screenshot shows a Qt development environment. On the left, the project tree for "01\_Basic\_Interaction" is visible, containing files like CMakeLists.txt, app01\_Basic\_Interaction, Header Files (message.h), Source Files (main.cpp, message.cpp), and Main.qml. The Main.qml file is selected in the tree. In the center, the code editor shows the following QML and C++ code:

```
13     msg.author = "Hello"
14
15 }
16
17 MouseArea {
18     anchors.fill: parent
19     onClicked: {
20         var str = "Who are you ?"
21         var result = msg.postMessage(str)
22
23         console.log("Result of postMessage() : " + result)
24
25         msg.refresh();
26     }
27 }
28 }
```

To the right, a terminal window titled "app01\_Basic\_Interaction" shows the output "Hello world.". At the bottom, the application's command-line output is shown in the "Application Output" tab:

```
13:56:21: Starting D:\QMLProjects\v2.0_QML_Examples\ch05\sub_01\buil...
Debug\app01_Basic_Interaction.exe...
[C++]
call postMessage method : "Who are you ?"
qml: Result of postMessage(): true
Called the C++ slot
```

예수님은 당신을 사랑합니다.

이번 프로젝트는 이전 프로젝트와 동일한 방법으로 프로젝트를 생성한다. 그런 다음 Message class 를 추가한다. Message 클래스를 추가했다면 message.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef MESSAGE_H
#define MESSAGE_H

#include <QObject>
#include <QDebug>

class Message : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString author
               READ author
               WRITE setAuthor
               NOTIFY authorChanged)

public:
    explicit Message(QObject *parent = nullptr);
    Q_INVOKABLE bool postMessage(const QString &msg) {
        qDebug() << "[C++] call postMessage method : "
              << msg;

        return true;
    }

    void setAuthor(const QString &a) {
        m_author = QString("%1 world.").arg(a);
        emit authorChanged();
    }

    QString author() const {
        return m_author;
    }

signals:
    void authorChanged();

private:
```

예수님은 당신을 사랑합니다.

```
QString m_author;

public slots:
    void refresh() {
        qDebug() << "Called the C++ slot";
    }
};

#endif // MESSAGE_H
```

다음으로 message.cpp 소스코드에는 생성자 함수만 존재한다. 따라서 아래와 같이 되어있는지 확인한다.

```
#include "message.h"

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

다음으로 Main.qml 파일을 아래와 같이 작성한다.

```
import QtQuick
import QtQuick.Window

Window {
    width : 200; height: 200; visible: true

    Text {
        id : myText;
        anchors.centerIn: parent
        text: msg.author

        Component.onCompleted: {
            msg.author = "Hello"
        }
    }

    MouseArea {
        anchors.fill: parent
        onClicked: {
```

예수님은 당신을 사랑합니다.

```
var str = "Who are you ?"
var result = msg.postMessage(str)
console.log("Result of postMessage():", result);

msg.refresh();
}
}
}
```

다음으로 main.cpp 파일을 열어서 아래와 같이 작성한다.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlContext>
#include "message.h"

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    QQmlApplicationEngine engine;
    Message msg;

    engine.rootContext()->setContextProperty("msg", &msg);

    const QUrl url("qrc:/01_Basic_Interaction/Main.qml");
    engine.load(url);

    return app.exec();
}
```

위의 예제와 같이 소스코드를 작성하고 예제를 실행해 보도록 하자. QML에서 MouseArea 태입의 onClicked 프로퍼티에서 C++의 Q\_INVOKABLE로 정의한 함수를 실행한 것을 확인할 수 있다.

이 예제의 예제소스코드는 01\_Basic\_Interaction 디렉토리를 참조하면 된다.

## 5.2. C++로 QML 타입 구현

이번 장에서는 C++을 이용해 QML 타입을 구현해 보자. C++로 구현한 QML 타입을 QML에서 사용하기 위해서는 `qmlRegisterType()` 함수를 사용할 수 있으며 아래와 같이 2 가지 방식을 사용할 수 있다.

```
template<typename T>
int qmlRegisterType(const char *uri, int versionMajor, int versionMinor,
                    const char *qmlName);

template<typename T, int metaObjectRevision>
int qmlRegisterType(const char *uri, int versionMajor, int versionMinor,
                    const char *qmlName);
```

다음은 `qmlRegisterType()` 함수의 사용 방법의 예제 소스코드이다. 첫 번째 인자는 QML import 시 import 할 이름을 명시한다. 두 번째는 Major 버전, 세 번째는 Minor 버전 그리고 마지막 네 번째는 QML에서 사용할 타입 이름을 명시하면 된다.

```
qmlRegisterType<Message>("Message", 1, 0, "Msg");
```

그리고 `Message`라는 QML 모듈을 C++ 클래스로 정의하기 위해서 다음과 같이 구현할 수 있다.

```
class Message : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString author READ author WRITE setAuthor NOTIFY authorChanged)
    ...
}
```

다음 예제는 위와 같이 `Message`라는 QML 모듈을 구현하였다고 가정하면 다음 예제에서 보는 것과 같이 QML에서 사용할 수 있다.

```
import QtQuick
import Message 1.0

Window {
    width : 300; height: 300
    ...
    Msg {
```

예수님은 당신을 사랑합니다.

```
    ...
}
```

다음은 qmlRegisterType( ) 함수를 사용할 때 두 번째 인자를 사용하는 것은 Revision 버전을 명시 하기 위해서 사용한다. 다음은 Revision 버전을 사용한 예제이다.

```
qmlRegisterType<Message, 1>( " Message", 1, 1, " Msg")
```

위와 같이 Revision 버전을 사용하였다면 Q\_PROPERTY 상에서 Revision 정보를 추가해야 한다.

```
class Message : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString author READ author WRITE setAuthor
               NOTIFY authorChanged REVISION 1)
signals:
    Q_REVISION(1) void authorChanged();
...
```

QML에서 새로 구현한 QML 타입의 내부에 하위 타입이 존재하는 형태로 구현할 수 있다.

```
MessageBoard
{
    Message { author: "Eddy" }
    Message { author: "Candy" }
}
```

위와 같은 방식과 더불어 다음과 같은 방식으로도 사용할 수 있다.

```
MessageBoard
{
    messages: [
        Message { author: "Eddy" },
        Message { author: "Candy" }
    ]
}
```

위의 2 가지 QML 예제에서 보는 것과 같이 사용하기 위해서 Q\_CLASSINFO() 매크로를 사용하고 QQmlListProperty( ) 함수를 사용할 수 있다.

예수님은 당신을 사랑합니다.

```
class MessageBoard : public QObject {
    Q_OBJECT
    Q_PROPERTY(QQmlListProperty<Message> messages READ messages)
    Q_CLASSINFO("DefaultProperty", "messages")

public:
    QQmlListProperty<Message> messages() const;

private:
    QList<Message *> messages;
};
```

- C++ 클래스를 이용해 커스텀 QML 타입 구현 예제

이번 예제는 QML에서 사용할 새로운 타입을 직접 구현해 보자. 구현할 모듈의 이름은 Message라는 모듈 이름을 사용한다. 그리고 프로그램을 실행하고 3초 후에 Message 클래스에서 정의한 QTimer에 의해 다음과 같은 Slot 함수가 호출되도록 하는 예제이다.

```
void timerTimeout()
{
    emit newMessagePosted("I am a boy");
}
```

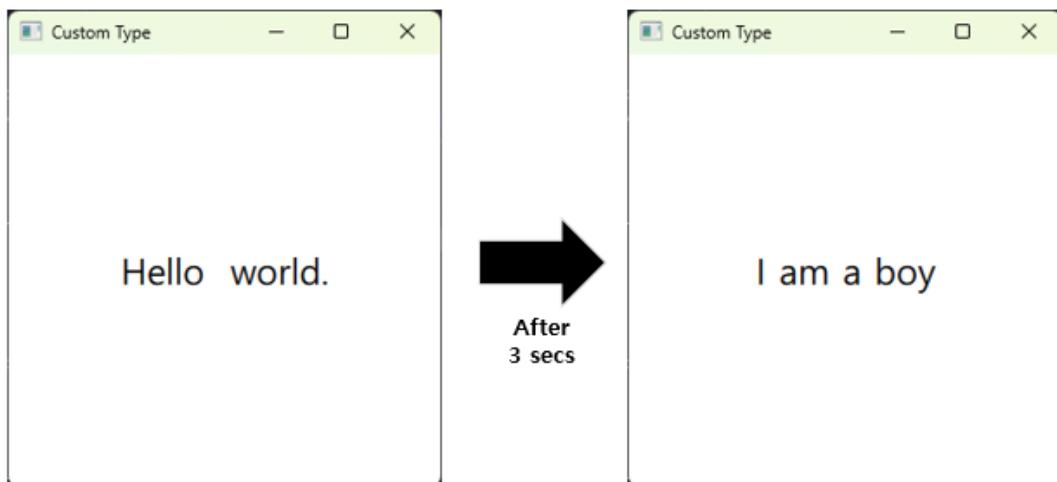
위의 Slot 함수가 호출되면 아래와 같이 Msg 타입의 onNewMessagePosted라는 QML의 Slot 프로퍼티가 호출된다. 다음은 예제 QML의 일부이다.

```
...
import Message 1.0

Window {
    ...
    Text {
        ...
        Msg {
            id: myMsg
            onNewMessagePosted: {
                console.log("[QML] New Message received: ", subject);
                myText.fontSize = 25
            }
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
    myText.text = subject;
}
}
}
```



위의 그림에서 보는 것과 같이 윈도우에서 출력된 "Hello world" 문자열이 "I am a boy"로 변경된다. 다음 소스코드는 Message 클래스의 헤더 파일 소스코드이다.

```
#ifndef MESSAGE_H
#define MESSAGE_H

#include <QObject>
#include <QTimer>
#include <QDebug>

class Message : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString author
               READ author
               WRITE setAuthor
               NOTIFY authorChanged)

public:
    explicit Message(QObject *parent = nullptr) {
        QTimer::singleShot(3000, this, SLOT(timerTimeout()));
    }
}
```

예수님은 당신을 사랑합니다.

```
Q_INVOKABLE bool postMessage(const QString &msg) {
    qDebug() << "[C++ Layer] call postMessage method : "
        << msg;
    return true;
}

void setAuthor(const QString &a) {
    m_author = QString("%1 world.").arg(a);
    emit authorChanged();
}

QString author() const {
    return m_author;
}

signals:
    void authorChanged();
    void newMessagePosted(const QString &subject);

private:
    QString m_author;

public slots:
    void refresh() {
        qDebug() << "Called the C++ slot";
    }

    void timerTimeout() {
        emit newMessagePosted("I am a boy");
    }
};

#endif // MESSAGE_H
```

다음은 Main.qml 을 아래와 같이 작성한다.

```
import QtQuick
import QtQuick.Controls
```

예수님은 당신을 사랑합니다.

```
import Message 1.0

Window {
    width : 300; height: 300; visible: true
    title: "Custom Type"

    Text {
        id : myText;
        anchors.centerIn: parent
        text: myMsg.author
        font.pixelSize: 25
        Component.onCompleted: {
            myMsg.author = "Hello "
        }
    }

    Msg {
        id: myMsg
        onNewMessagePosted: function(subject) {
            console.log("Message received: ", subject);
            myText.font.pixelSize = 25
            myText.text = subject;
        }
    }
}
```

다음으로 main.cpp 소스코드 파일을 아래와 같이 작성한다.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "message.h"

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    qmlRegisterType<Message>("Message", 1, 0, "Msg");

    QQmlApplicationEngine engine;
```

예수님은 당신을 사랑합니다.

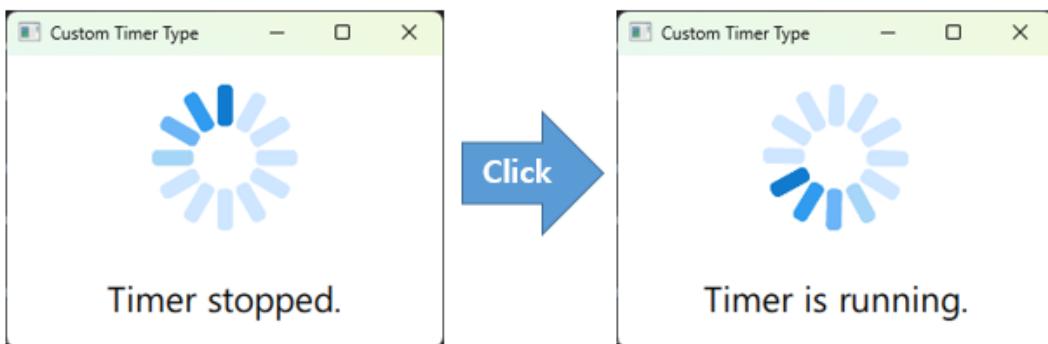
```
const QUrl url("qrc:/00_Custom_Type_Basic/Main.qml");
QObject::connect(&engine, &QQmlApplicationEngine::objectCreationFailed,
    &app, []() { QCoreApplication::exit(-1); },
    Qt::QueuedConnection);
engine.load(url);

return app.exec();
}
```

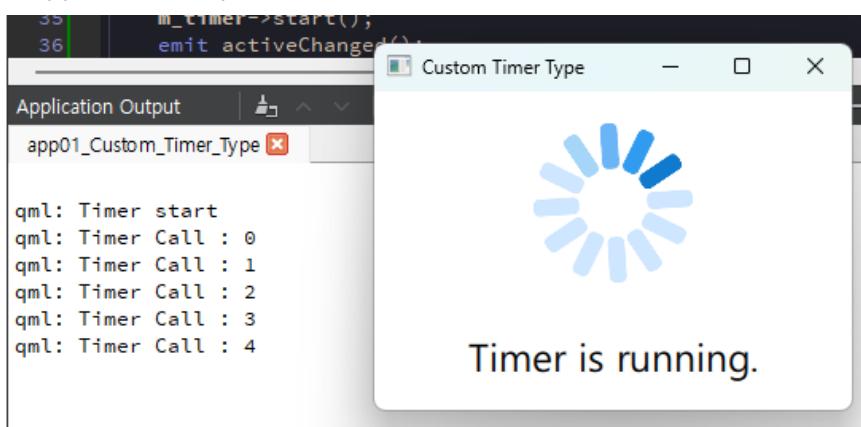
이 예제의 소스코드는 00\_Custom\_Type\_Basic 디렉토리를 참조하면 된다.

- 타이머 탑입 구현 예제

이번 예제는 QML에서 동작하는 타이머를 구현한 예제이다. 다음 그림은 예제 실행 화면이다.



위의 그림에서 보는 것과 같이 QML이 로딩되면 타이머가 정지된 상태이다. QML 영역을 클릭하면 타이머가 동작한다. 그리고 아래 그림과 같이 Qt Creator IDE 툴의 디버깅 창(Application Output)에서 디버깅 메시지를 출력한다.



예수님은 당신을 사랑합니다.

프로젝트에서 사용한 이미지는 예제 소스코드 디렉토리에 있다. 예제 소스코드 디렉토리는 01\_Custom\_Timer\_Type 디렉토리 안에 images라는 디렉토리에 있다. 그리고 Timer라는 클래스를 프로젝트에 추가한 다음 timer.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef TIMER_H
#define TIMER_H

#include <QObject>
#include <QTimer>

class Timer : public QObject
{
    Q_OBJECT
    Q_PROPERTY(int interval
               READ interval
               WRITE setInterval
               NOTIFY intervalChanged)

    Q_PROPERTY(bool active
               READ isActive
               NOTIFY activeChanged)

public:
    explicit Timer( QObject* parent = nullptr );

    void setInterval( int msec );
    int interval();

    bool isActive();

public slots:
    void start();
    void stop();

signals:
    void timeout();
    void intervalChanged();
    void activeChanged();
```

```
private:  
    QTimer* m_timer;  
};  
  
#endif // TIMER_H
```

다음은 timer.cpp 소스코드 파일이다. 아래에서 보는 것과 같이 소스코드를 작성한다.

```
#include <QTimer>  
#include <QDebug>  
#include "timer.h"  
  
Timer::Timer( QObject* parent ) : QObject( parent ),  
    m_timer( new QTimer( this ) )  
{  
    connect( m_timer, SIGNAL(timeout()), this, SIGNAL(timeout()) );  
}  
  
void Timer::setInterval( int msec )  
{  
    if ( m_timer->interval() == msec )  
        return;  
    m_timer->setInterval(msec);  
    emit intervalChanged();  
}  
  
int Timer::interval()  
{  
    return m_timer->interval();  
}  
  
bool Timer::isActive()  
{  
    return m_timer->isActive();  
}  
  
void Timer::start()  
{  
    if ( m_timer->isActive() )
```

예수님은 당신을 사랑합니다.

```
    return;
    m_timer->start();
    emit activeChanged();
}

void Timer::stop()
{
    if ( !m_timer->isActive() )
        return;
    m_timer->stop();
    emit activeChanged();
}
```

다음으로 main.cpp 파일을 아래와 같이 작성한다.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "timer.h"

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    qmlRegisterType<Timer>("MyCustomTimer", 1, 0, "MyTimer");

    QQmlApplicationEngine engine;

    const QUrl url("qrc:/01_Custom_Timer_Type/Main.qml");
    QObject::connect(&engine, &QQmlApplicationEngine::objectCreationFailed,
                     &app, []() { QCoreApplication::exit(-1); },
                     Qt::QueuedConnection);
    engine.load(url);

    return app.exec();
}
```

위의 예제 소스코드에서 qmlRegisterType( ) 함수의 첫 번째 인자인 “MyCustomTimer”는 QML에서 import 할 QML 모듈 이름으로 사용된다. 그리고 두 번째와 세 번째 인자는 버전 정보 그리고 마지막 네 번째 인자인 “MyTimer”는 QML에서 사용하는 타입의 이름이다. 다음은 Main.qml 파일이다. 아래에서 보는 것과 같이 작성한다.

```
import QtQuick
import QtQuick.Controls
import MyCustomTimer 1.0

Window {
    width: 300; height: 200; visible: true
    title: "Custom Timer Type"

    property int timerCnt: 0

    Image {
        id: loadImage
        source: "images/loading.png"
        width: 100; height: 100
        anchors.top: parent.top
        anchors.topMargin: 20
        anchors.horizontalCenter: parent.horizontalCenter
    }

    PropertyAnimation {
        id: loadAni
        target: loadImage
        loops: Animation.Infinite
        from: 0;
        to: 360
        property: "rotation"
        duration: 2000
        running: false
    }

    Text {
        text: timer.active ? "Timer is running." : "Timer stopped."

        font.pixelSize: 24
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.top: loadImage.bottom
        anchors.topMargin: 30
    }
}
```

```
MyTimer {
    id: timer
    interval: 1000
    onTimeout: {
        console.log( "Timer Call : ", timerCnt++);
    }
}

MouseArea {
    anchors.fill: parent

    onClicked: {
        if ( timer.active == false ) {
            console.log( "Timer start" );
            timer.start();
            loadAni.start();
        } else {
            console.log( "Timer stop" );
            timer.stop();
            loadAni.stop();
        }
    }
}
```

위의 QML 소스코드에서 `onTimeout` 은 Signal 프로퍼티이다. 이 프로퍼티는 C++ `Timer` 클래스에서 `timeout()` 시그널이 호출되면 발생한다.

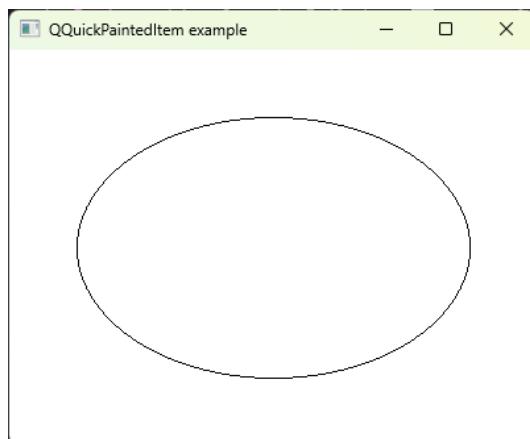
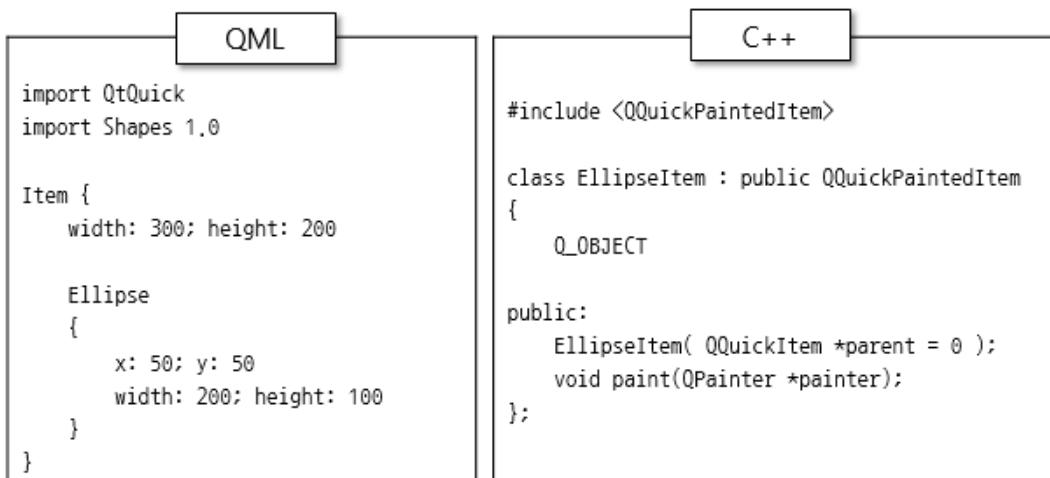
그리고 C++ `Timer` 클래스에서 Slot 함수 중 `start()` 와 `stop()` 함수는 위의 QML 소스 코드에서 `timer.start()` 와 `timer.stop()` 함수와 매핑 된다.

예를 들어 QML 에서 `timer.start()` 가 호출되면 C++ `Timer` 클래스에서 `start()` 멤버 함수가 호출된다.

### 5.3. QML에서 QQuickPaintedItem 클래스 사용

`QQuickPaintedItem` 클래스는 `QPainter` 클래스와 같은 기능을 제공한다. 차이점은 `QPainter` 클래스는 `QWidget` 클래스로 구현한 GUI 윈도우 영역 내에서 2D 그래픽 요소 (선, 라인, 원, 곡선, 그라디언트, 이미지 표시등)를 표시하기 위한 목적으로 사용한다.

하지만 `QQuickPaintedItem` 클래스는 QML 타입 영역 내에서 2D 그래픽 요소를 표시할 수 있다. 따라서 `QQuickPaintedItem` 클래스로 드로잉한 2D 그래픽 결과를 QML 타입 영역 내에 표시할 수 있다.



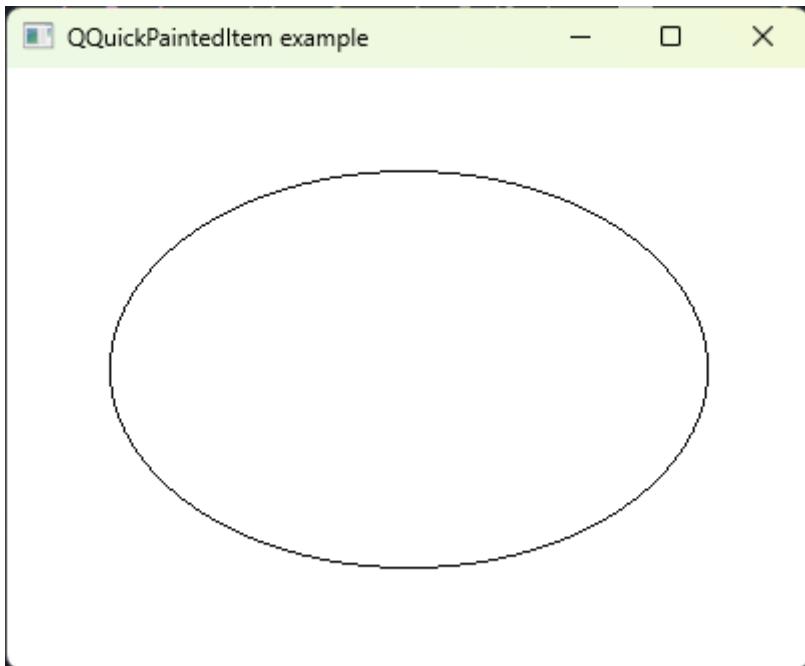
위의 그림은 예제 실행 화면이다. 예제 실행 화면과 같은 타원을 표시하기 위해 좌측 예제 소스코드의 `EllipseItem` 클래스의 `paint()` 멤버 함수에서 타원을 드로잉한 결과가 QML의 `Ellipse` 타입 영역에 표시된다.

예수님은 당신을 사랑합니다.

QQuickPaintedItem 클래스는 QQuickItem 클래스로부터 상속 받는다. QQuickItem 클래스는 Qt Quick에서 제공하는 모든 그래픽 표시 관련 QML 타입을 구현하는 대부분의 클래스로 사용된다. QQuickItem 클래스는 그래픽 표시 관련된 것 이외에도 키보드, 터치, 마우스 등 이벤트를 처리 할 수 있다.

- **QQuickPaintedItem** 클래스를 이용한 예제

이번 예제에서는 위의 예에서 살펴본 예제를 구현해 보도록 하자. QQuickPaintedItem 클래스에서 2D 그래픽 요소인 타원을 드로잉한 결과를 QML 영역에 표시할 수 있다.



위의 예제 실행 화면에서 보는 것과 같이 타원을 그리기 위해서 Qt C++ 클래스인 QQuickPaintedItem 클래스를 사용할 것이다.

프로젝트 생성 후 프로젝트 상에 EllipseItem 클래스를 추가한다. 그리고 ellipseitem.h 헤더 파일을 열어서 아래와 같이 작성한다.

```
#ifndef ELLIPSEITEM_H
#define ELLIPSEITEM_H

#include <QQuickPaintedItem>

class EllipseItem : public QQuickPaintedItem
{
```

예수님은 당신을 사랑합니다.

```
Q_OBJECT
public:
    EllipseItem(QQuickItem *parent = nullptr);
    void paint(QPainter *painter);
};

#endif
```

위의 예제 소스코드에서 보는 것과 같이 `paint()` 함수가 2D 그래프 요소인 타원을 그리는 소스코드가 구현된 멤버 함수이다.

다음으로 `ellipseitem.cpp` 소스코드 파일을 열어서 아래와 같이 작성한다.

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

EllipseItem::EllipseItem(QQuickItem *parent)
    : QQuickPaintedItem(parent)
{
}

void EllipseItem::paint(QPainter *painter)
{
    const qreal halfPenWidth = qMax(painter->pen().width() / 2.0, 1.0);

    QRectF rect = boundingRect();

    rect.adjust(halfPenWidth,
                halfPenWidth,
                -halfPenWidth,
                -halfPenWidth);

    painter->drawEllipse(rect);
}
```

다음으로 `Main.qml` 파일을 열어서 아래와 같이 작성한다.

```
import QtQuick
import QtQuick.Window
import Shapes 1.0
```

예수님은 당신을 사랑합니다.

```
Window {  
    width: 400  
    height: 300  
    visible: true  
    title: "QQuickPaintedItem example"  
  
    Ellipse {  
        anchors.centerIn: parent  
        width: 300  
        height: 200  
    }  
}
```

다음 예제 소스코드는 main.cpp 소스코드이다. main.cpp 에서는 qmlRegisterType( ) 을 이용해 EllipseItem 클래스를 QML에서 Ellipse QML 타입으로 사용할 수 있도록 연결하는 기능을 제공한다.

```
#include <QGuiApplication>  
#include <QQmlApplicationEngine>  
#include "ellipseitem.h"  
  
int main(int argc, char *argv[]){  
    QGuiApplication app(argc, argv);  
  
    qmlRegisterType<EllipseItem>("Shapes", 1, 0, "Ellipse");  
  
    QQmlApplicationEngine engine;  
    const QUrl url("qrc:/00_QQuickPaintedItem_Example/Main.qml");  
    QObject::connect(&engine, &QQmlApplicationEngine::objectCreationFailed,  
                    &app, []() { QCoreApplication::exit(-1); },  
                    Qt::QueuedConnection);  
    engine.load(url);  
  
    return app.exec();  
}
```

이 예제의 소스코드는 00\_QQuickPaintedItem\_Example 디렉토리를 참조하면 된다.

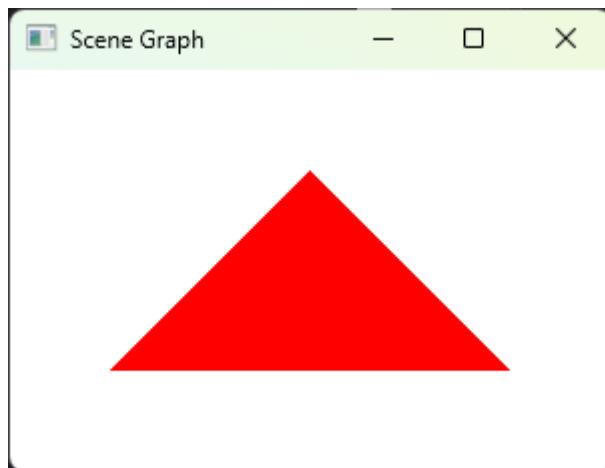
## 5.4. Scene Graph

Scene Graph 는 Qt C++ 에서 제공하는 QPainter 와 동일한 기능을 제공한다. Scene Graph 를 사용하기 위해서는 QQuickItem 클래스를 상속받아 구현된 QSGNode 클래스를 사용할 수 있다. 이 클래스는 Qt Graphics View Framework 에서 제공하는 QGraphicsItem 클래스와 동일한 기능을 제공한다.

Scene Graph 클래스는 기본적으로 OpenGL ES 2.0 또는 OpenGL 2.0 을 사용한다. 그리고 QSGNode 클래스는 이외에도 QSGGeometry, QSGMaterial(Texture) 등과 같은 클래스를 제공한다.

- Scene Graph 클래스를 이용한 Triangle 예제

이번 예제에서는 QSGNode, QSGGeometry 그리고 QSGFlatColorMaterial 클래스를 이용해 2D 그래프 요소인 Triangle 을 표시하는 예제를 다루어 보자.



QSGNode 클래스는 Scene Graph 에서 모든 그래프 단위인 Node 를 관리하는 클래스이다. QSGGeometry 클래스는 Scene Graph 상의 렌더링 할 Geometry 정보를 저장하는 클래스이다. QSGFlatColorMaterial 클래스는 Scene Graph 내에 Color 를 표현하기 위한 사용하는 클래스이다.

새로운 프로젝트를 생성한 다음 TriangleItem 클래스를 추가한다. 그리고 아래와 같이 triangleitem.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef TRIANGLEITEM_H
```

예수님은 당신을 사랑합니다.

```
#define TRIANGLEITEM_H

#include <QQuickItem>
#include <QSGGeometry>
#include <QSGFlatColorMaterial>

class TriangleItem : public QQuickItem
{
    Q_OBJECT
    QML_ELEMENT
public:
    TriangleItem(QQuickItem *parent = nullptr);

protected:
    QSGNode *updatePaintNode(QSGNode *node, UpdatePaintNodeData *data);

private:
    QSGGeometry m_geometry;
    QSGFlatColorMaterial m_material;
};

#endif // TRIANGLEITEM_H
```

다음으로 triangleitem.cpp 소스코드 파일을 아래와 같이 작성한다.

```
#include "triangleitem.h"
#include <QSGGeometryNode>

TriangleItem::TriangleItem(QQuickItem *parent)
    : QQuickItem(parent),
      m_geometry(QSGGeometry::defaultAttributes_Point2D(), 3)
{
    setFlag(ItemHasContents);
    m_material.setColor(Qt::red);
}

QSGNode *TriangleItem::updatePaintNode(QSGNode *n, UpdatePaintNodeData *)
{
    QSGGeometryNode *node = static_cast<QSGGeometryNode *>(n);
    if (!node) {
```

예수님은 당신을 사랑합니다.

```
node = new QSGGeometryNode();
}

QSGGeometry::Point2D *v = m_geometry.vertexDataAsPoint2D();
const QRectF rect = boundingRect();
v[0].x = (float)rect.left();
v[0].y = (float)rect.bottom();

v[1].x = (float)rect.left() + (float)rect.width()/2;
v[1].y = (float)rect.top();

v[2].x = (float)rect.right();
v[2].y = (float)rect.bottom();

node->setGeometry(&m_geometry);
node->setMaterial(&m_material);

return node;
}
```

setFlag() 함수에서 지정한 ItemHasContents ENUM 값은 QQuickItem에서 제공하는 Flag ENUM값이다.

Scene Graph에 포함된 QSGFlatColorMaterial 클래스의 m\_material 오브젝트는 setColor( ) 멤버 함수를 이용해 색상을 지정할 수 있다. 다음으로 main.cpp 파일을 열어서 아래와 같이 작성한다.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "triangleitem.h"

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    qmlRegisterType<TriangleItem>("Shapes", 1, 0, "Triangle");

    QQmlApplicationEngine engine;
    const QUrl url("qrc:/00_Scene_Graph_Example/Main.qml");
    QObject::connect(&engine, &QQmlApplicationEngine::objectCreationFailed,
```

예수님은 당신을 사랑합니다.

```
&app, []() { QApplication::exit(-1); },
Qt::QueuedConnection);
engine.load(url);

return app.exec();
}
```

다음은 QML 소스코드이다. 아래와 같이 소스코드를 작성한다.

```
import QtQuick
import QtQuick.Window
import Shapes 1.0

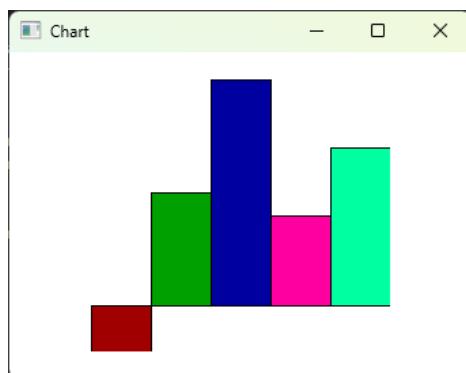
Window {
    width: 300; height: 200; visible: true
    title: "Scene Graph"

    Triangle {
        anchors.centerIn: parent
        width: 200; height: 100
    }
}
```

이 예제의 소스코드는 00\_Scene\_Graph\_Example 디렉토리를 참조하면 된다.

### ● Bar 차트 구현 예제

이번 예제에서는 다음 그림에서 보는 것과 같이 Bar(막대) 차트 그래프를 구현해 보도록 하자.



위의 그림에서 보는 것과 같이 Bar 차트를 표시하기 위해서 2 개 QML 타입을 구현하였다. Chart 타입은 Container 와 같은 역할을 한다. 예를 들어 위의 그림에서

예수님은 당신을 사랑합니다.

보는 것과 같이 5 개의 막대 그래프를 한 개의 차트 안에 표시하기 위한 Container 기능을 제공한다. 그리고 Bar 타입은 실제 막대 그래프의 요소이다. 예를 들어 위의 그림에서 보는 것과 같이 5 개의 막대 이므로 5 개의 Bar 타입을 사용하였다.

새로운 프로젝트를 생성한 다음 프로젝트 상에 BarItem 클래스를 생성한다. 그리고 baritem.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef BARITEM_H
#define BARITEM_H
#include <QColor>
#include <QObject>

class BarItem : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QColor color READ color WRITE setColor NOTIFY colorChanged)
    Q_PROPERTY(qreal value READ value WRITE setValue NOTIFY valueChanged)

public:
    BarItem(QObject *parent = 0);
    QColor color() const;
    void setColor(const QColor &newColor);
    qreal value() const;
    void setValue(qreal newValue);

signals:
    void colorChanged();
    void valueChanged();

private:
    QColor m_color;
    qreal m_value;
};

#endif
```

다음으로 baritem.cpp 소스코드 파일을 아래와 같이 작성한다.

```
#include "baritem.h"
```

예수님은 당신을 사랑합니다.

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

QColor BarItem::color() const
{
    return m_color;
}

void BarItem::setColor(const QColor &newColor)
{
    if (m_color != newColor) {
        m_color = newColor;
        emit colorChanged();
    }
}

qreal BarItem::value() const
{
    return m_value;
}

void BarItem::setValue(qreal newValue)
{
    if (m_value != newValue) {
        m_value = newValue;
        emit valueChanged();
    }
}
```

다음으로 ChartItem 클래스를 추가한다. 그리고 chartitem.h 헤더 파일을 아래와 같이 작성한다.

```
#ifndef CHARTITEM_H
#define CHARTITEM_H

#include <QQQuickPaintedItem>

class BarItem;
```

예수님은 당신을 사랑합니다.

```
class ChartItem : public QQuickPaintedItem
{
    Q_OBJECT
    Q_PROPERTY(QQmlListProperty<BarItem> bars READ bars NOTIFY barsChanged)

public:
    ChartItem(QQuickItem *parent = nullptr);
    void paint(QPainter *painter);
    QQmlListProperty<BarItem> bars();

signals:
    void barsChanged();

private:
    static void append_bar(QQmlListProperty<BarItem> *list, BarItem *bar);
    QList<BarItem*> m_bars;
};

#endif
```

다음으로 chartitem.cpp 소스코드 파일을 아래와 같이 작성한다.

```
#include <QPainter>
#include "baritem.h"
#include "chartitem.h"

ChartItem::ChartItem(QQuickItem *parent)
    : QQuickPaintedItem(parent)
{
}

void ChartItem::paint(QPainter *painter)
{
    if (m_bars.count() == 0)
        return;

    qreal minimum = m_bars[0]->value();
    qreal maximum = minimum;

    for (int i = 1; i < m_bars.count(); ++i) {
        minimum = qMin(minimum, m_bars[i]->value());
```

예수님은 당신을 사랑합니다.

```
maximum = qMax(maximum, m_bars[i]->value());
}

if (maximum == minimum)
    return;

painter->save();

const QRectF rect = boundingRect();

qreal scale = rect.height()/(maximum - minimum);
qreal barWidth = rect.width()/m_bars.count();

qDebug() << scale;
qDebug() << barWidth;

for (int i = 0; i < m_bars.count(); ++i) {
    BarItem *bar = m_bars[i];
    qreal barEdge1 = scale * (maximum - bar->value());
    qreal barEdge2 = scale * maximum;
    QRectF barRect(rect.x() + i * barWidth,
                   rect.y() + qMin(barEdge1, barEdge2),
                   barWidth, qAbs(barEdge1 - barEdge2));

    painter->setBrush(bar->color());
    painter->drawRect(barRect);
}

painter->restore();
}

QQmlListProperty<BarItem> ChartItem::bars()
{
    return QQmlListProperty<BarItem>(this, &m_bars);
}

void ChartItem::append_bar(QQmlListProperty<BarItem> *list, BarItem *bar)
{
    ChartItem *chart = qobject_cast<ChartItem *>(list->object);
```

예수님은 당신을 사랑합니다.

```
if (chart) {
    bar->setParent(chart);
    chart->m_bars.append(bar);
}
}
```

다음으로 main.cpp 파일을 열어서 아래와 같이 작성한다.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "chartitem.h"
#include "baritem.h"

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    qmlRegisterType<ChartItem>("Shapes", 1, 0, "Chart");
    qmlRegisterType<BarItem>("Shapes", 1, 0, "Bar");

    QQmlApplicationEngine engine;
    const QUrl url(u"qrc:/01_ChartItem_Example/Main.qml");
    QObject::connect(&engine, &QQmlApplicationEngine::objectCreationFailed,
                     &app, []() { QCoreApplication::exit(-1); },
                     Qt::QueuedConnection);
    engine.load(url);

    return app.exec();
}
```

다음으로 Main.qml 을 아래와 같이 작성한다.

```
import QtQuick
import QtQuick.Window
import Shapes 1.0

Window {
    width: 340
    height: 240
    visible: true
    title: qsTr("Chart")
```

```
Chart {  
    width: 220; height: 200  
    anchors.centerIn: parent  
    bars: [  
        Bar { color: "#a00000"; value: -20 },  
        Bar { color: "#00a000"; value: 50 },  
        Bar { color: "#0000a0"; value: 100 },  
        Bar { color: "#ff00a0"; value: 40 },  
        Bar { color: "#00ffa0"; value: 70 }  
    ]  
}  
}
```

위의 예제에서 보는 것과 같이 Chart 타입은 ChartItem 클래스, Bar 타입은 BarItem 클래스로 구현하였다.

BarItem 은 막대 그래프의 속성을 정의하였고 실제 막대 그래프 요소를 드로잉하는 것은 ChartItem 클래스에서 드로잉한다.

이 예제의 소스코드는 01\_ChartItem 디렉토리를 참조하면 된다.

## 5.5. C++과 QML 간의 Interaction 과 변수 매핑

여기서 말하는 Interaction이란 C++에서 QML의 타입을 핸들링, QML타입의 프로퍼티 값 접근 또는 설정, QML 내의 SIGNAL과 연결 등을 말한다.

따라서 이번 장에서는 Qt C++과 QML 간의 Interaction과 변수 매핑에 대해서 알아보도록 하자.

- ✓ C++에서 QQmlComponent 클래스를 이용해 C++과 QML간의 Interaction QQmlComponent 클래스는 QML과 Interaction하기 위한 기능을 제공한다. 아래와 같은 QML 파일을 작성한다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 300; height: 300; visible: true
    title: qsTr("Interaction Example")

    Rectangle {
        id: myRect
        anchors.centerIn: parent
        width: 200; height: 100
        color: "#0000FF"
    }
}
```

위와 같은 QML 파일을 있다고 가정해 보자 위의 QML 타입을 접근하기 위해서 QQmlComponent 클래스를 이용해야 한다.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlComponent>
#include <QQmlProperty>
#include <QDebug>

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

```
{  
    QGuiApplication app(argc, argv);  
  
    QQmlApplicationEngine engine;  
    QQmlComponent component(&engine, "qrc:/00_Interaction/Main.qml");  
    QObject *object = component.create();  
  
    qDebug() << "Width : " << QQmlProperty(object, "width").write(400);  
    return app.exec();  
}
```

- ✓ C++을 이용해 QML 타입 내의 프로퍼티 또는 Object를 접근 하는 방법

위의 예제에서 사용했던 QObject 를 이용해 QML 타입의 프로퍼티를 변경하기 위해서 다음과 같은 QObject 클래스의 setProperty( ) 멤버 함수 또는 QQmlProperty 를 사용 할 수 있다.

```
object->setProperty("width", 500);  
QQmlProperty(object, "width").write(500);
```

- ✓ C++을 이용해 QML 타입 프로퍼티를 WRITE/READ 하는 방법

C++ 에서 QML 타입을 접근하기 위해서 QObject 클래스의 findChild( ) 멤버 함수를 이용해 QML타입의 프로퍼티를 접근할 수 있다. 다음은 QML 예제 소스코드 이다.

```
import QtQuick  
import QtQuick.Window  
  
Window {  
    width: 200; height: 200; visible: true  
    title: qsTr("Interaction Example")  
  
    Rectangle {  
        id: myRect  
        width: 200; height: 100  
        color: "#0000FF"  
    }  
  
    Rectangle {
```

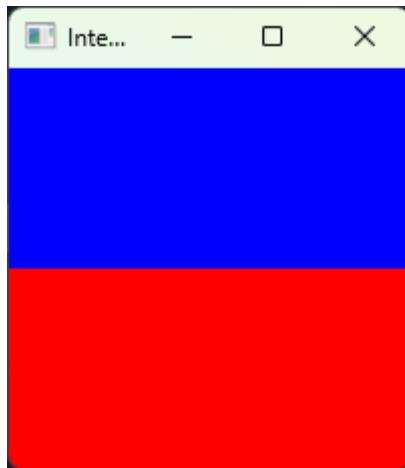
예수님은 당신을 사랑합니다.

```
    objectName: "secondRect"  
    anchors.left: myRect.left  
    anchors.top: myRect.bottom  
    width: 200; height: 100  
    color: "#00FF00"  
}  
}  
}
```

위의 QML 예제 소스코드에 Rectangle 타입의 color 프로퍼티에 접근하기 위해서 다음과 같이 C++에서 color 프로퍼티 값을 변경할 수 있다.

```
#include <QGuiApplication>  
#include <QQmlApplicationEngine>  
#include <QQmlComponent>  
#include <QQmlProperty>  
  
#include <QDebug>  
  
int main(int argc, char *argv[])  
{  
    QGuiApplication app(argc, argv);  
  
    QQmlApplicationEngine engine;  
  
    QQmlComponent component(&engine, "qrc:/00_Interaction/Main.qml");  
    QObject *object = component.create();  
    QObject *rect = object->findChild<QObject*>("secondRect");  
    rect->setProperty("color", "#FF0000");  
  
    return app.exec();  
}
```

예수님은 당신을 사랑합니다.



다음 예제 소스코드는 QML 타입의 프로퍼티 값을 참조하기 위한 방법이다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 200; height: 200; visible: true
    title: qsTr("Interaction Example")

    property int someNumber: 100
}
```

위의 someNumber 프로퍼티를 C++에서 참조하기 위한 방법으로 QObject 클래스에서 제공하는 property( ) 멤버 함수를 다음 C++ 예제 소스코드에서 보는 것과 같이 사용 할 수 있다.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlComponent>
#include <QQmlProperty>
#include <QDebug>

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    QQmlApplicationEngine engine;
```

```
QQmlComponent component(&engine, "qrc:/00_Interaction/Main.qml");
QObject *object = component.create();

qDebug() << "Property value:"
    << QQmlProperty::read(object, "someNumber").toInt();

QQmlProperty::write(object, "someNumber", 5000);

qDebug() << "Property value:"
    << object->property("someNumber").toInt();

return app.exec();
}
```

- ✓ QML로 작성된 Method( ) 를 C++에서 호출하기 위한 방법

C++에서 QVariant 클래스를 이용해 QML에서 구현된 Method를 호출하기 위해서 QMetaObject 클래스에서 제공하는 invokeMethod( ) 멤버 함수를 이용해 호출할 수 있다. 다음은 QML 에 구현된 Method 함수 예제 소스코드이다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 200; height: 200; visible: true
    title: qsTr("Interaction Example")

    function myQmlFunction(msg) {
        console.log("Got message:", msg)
        return "some return value"
    }
}
```

위의 QML 에서 구현된 myQmlFunction( ) 를 C++에서 호출하기 위해서 다음 예제에서 보는 것과 같이 사용할 수 있다.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
```

예수님은 당신을 사랑합니다.

```
#include <QQmlComponent>
#include <QQmlProperty>

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    QQmlApplicationEngine engine;

    QQmlComponent component(&engine, "qrc:/00_Interaction/Main.qml");
    QObject *object = component.create();

    QVariant returnedValue;
    QVariant msg = "Hello from C++";
    QMetaObject::invokeMethod(object,
        "myQmlFunction",
        Q_RETURN_ARG(QVariant, returnedValue),
        Q_ARG(QVariant, msg) );
    return app.exec();
}
```

✓ C++에서 connection() 함수를 사용해 QML의 Signal과 연결하는 방법

C++에서 Signal 과 Slot을 연결하기 위한 connection( ) 함수를 이용해 QML 내의 Signal을 연결하기 위해서 다음 QML 예제에서 보는 것과 같이 signal 을 등록해야 한다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 200; height: 200; visible: true
    title: qsTr("Signal Example")

    id: item
    signal qmlSignal(string msg)

    MouseArea {
        anchors.fill: parent
```

예수님은 당신을 사랑합니다.

```
    onClicked: {
        item.qmlSignal("Hello from QML")
    }
}
```

다음 예제는 위의 QML 내의 Signal 을 C++ 에서 제공하는 Slot 함수와 연결하기 위한 예제 소스코드 이다.

아래에서 보는 것과 같이 C++ 클래스를 생성 후 아래와 같이 헤더 파일을 작성한다.

```
#ifndef MYCLASS_H
#define MYCLASS_H

#include <QObject>
#include <QDebug>

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

public slots:
    void cppSlot(const QString &msg) {
        qDebug() << "Called the C++ slot with message:"
              << msg;
    }
};

#endif // MYCLASS_H
```

다음으로 소스코드 파일을 작성한다.

```
#include "myclass.h"

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

다음으로 main.cpp를 아래와 같이 작성한다.

예수님은 당신을 사랑합니다.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlComponent>
#include <QDebug>
#include "myclass.h"

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

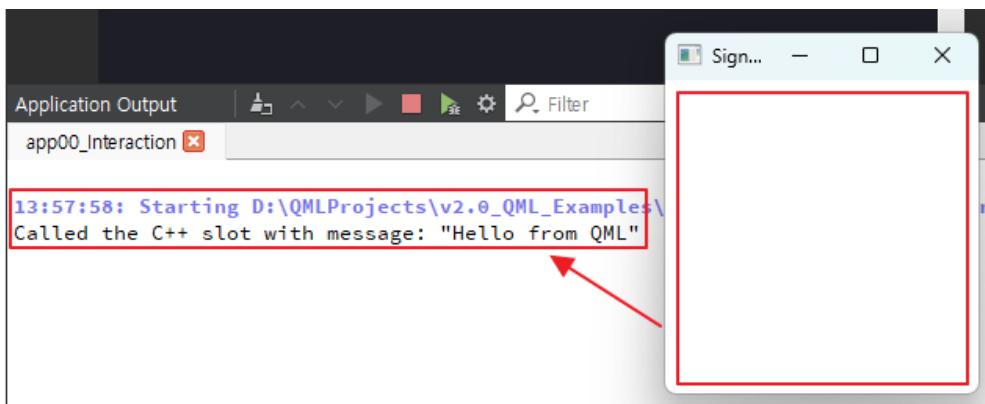
    QQmlApplicationEngine engine;

    QQmlComponent component(&engine, "qrc:/00_Interaction/Main.qml");
    QObject *object = component.create();

    MyClass myClass;
    QObject::connect(object, SIGNAL(qmlSignal(QString)),
                     &myClass, SLOT(cppSlot(QString)));

    return app.exec();
}
```

QML 영역을 클릭하면 아래 그림에서 보는 것과 같이 메시지를 확인할 수 있다.



✓ C++과 QML간의 데이터 교환 및 변수 타입 매핑

C++로부터 QML 간의 데이터 교환을 위한 위해서 아래와 같은 타입으로 변환할 수 있다.

Qt C++ 변수	QML 변수 타입
bool	bool
unsigned int 또는 int	int
double	double
float, qreal	real
QString	string
QColor	color
QFont	font
QDate	date
QTime	time
QPoint, QPointF	point
QSize, QSizeF	size
QRect, QRectF	rect
QMatrix4x4	matrix4x4
QQuaternion	quaternion
QVector2D	vector2d
QVector3D	vector3d
QVector4D	vector4d

Qt C++에서 사용했던 QColor, QFont, QQuaternion 등과 같은 클래스들을 QML에서 사용하기 위해서 다음 예제에서 보는 것과 같이 사용할 수 있다.

```
Item {
    Image { sourceSize: "100x200" }
    Image { sourceSize: Qt.size(100, 200) }
}
```

- Model 과 View 를 이용한 C++ 과 QML간 데이터 교환 예제

이번 예제는 Qt C++에서 QStringList 클래스의 값을 QML에서 사용하는 방법에 대해서 알아보도록 하자.

예수님은 당신을 사랑합니다.

프로젝트 생성 후 아래와 같이 QML을 작성한다.

```
import QtQuick
import QtQuick.Window

Window {
    width: 300; height: 200; visible: true
    title: qsTr("String List Model")

    ListView {
        anchors.centerIn: parent
        width: 100; height: 100

        model: MyModel
        delegate: Rectangle
        {
            height: 25
            width: 100
            Text { text: modelData }
        }
    }
}
```

다음으로 main.cpp 파일을 열어서 아래와 같이 작성한다.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlContext>

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    QStringList dataList;
    dataList.append("Item 1");
    dataList.append("Item 2");
    dataList.append("Item 3");
    dataList.append("Item 4");
```

예수님은 당신을 사랑합니다.

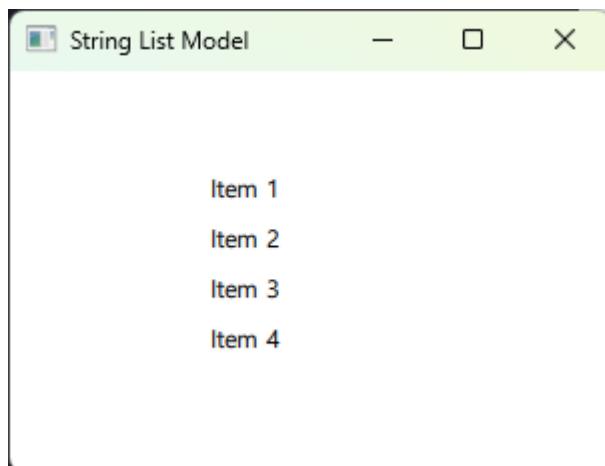
```
QQmlApplicationEngine engine;

QQmlContext *ctx = engine.rootContext();
ctx->setContextProperty("MyModel", QVariant::fromValue(dataList));

const QUrl url("qrc:/00_StringListModel/Main.qml");
engine.load(url);

return app.exec();
}
```

위의 예제에서 보는 것과 같이 QStringList 클래스에 저장된 데이터 배열을 QML에서 사용하기 위해서는 QML에서 제공하는 Model/View를 사용하였다. 다음은 QML 예제이다.



위의 예제 소스코드는 00\_StringListModel 디렉토리를 참조하면 된다.

- Model 과 View 를 이용한 C++ 클래스 교환

이전 예제에서는 QList 클래스에 문자열 데이터를 QML에게 전달하였다. 이번 예제에서는 클래스를 QML에게 전달하는 예제를 다루어 보도록 하자. 먼저 QML로 전달하려는 C++클래스를 작성해 보자.

프로젝트 생성 후 DataObject 클래스를 생성한다. 그리고 dataobject.h 헤더 파일을 작성한다.

```
#ifndef DATAOBJECT_H
#define DATAOBJECT_H
```

예수님은 당신을 사랑합니다.

```
#include <QObject>

class DataObject : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString name READ name WRITE setName NOTIFY nameChanged)
    Q_PROPERTY(QString color READ color WRITE setColor NOTIFY colorChanged)
public:
    explicit DataObject(QObject *parent = nullptr);
    DataObject(const QString &name,
               const QString &color,
               QObject *parent=0);

    QString name() const;
    void setName(const QString &name);
    QString color() const;
    void setColor(const QString &color);

signals:
    void nameChanged();
    void colorChanged();

private:
    QString m_name;
    QString m_color;
};

#endif // DATAOBJECT_H
```

다음으로 dataobject.cpp 파일을 아래와 같이 작성한다.

```
#include "dataobject.h"

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

DataObject::DataObject(const QString &name,
                      const QString &color,
```

예수님은 당신을 사랑합니다.

```
QObject *parent )  
: QObject(parent),  
m_name(name),  
m_color(color)  
{  
}  
  
QString DataObject::name() const {  
    return m_name;  
}  
void DataObject::setName(const QString &name)  
{  
    if (name != m_name) {  
        m_name = name;  
        emit nameChanged();  
    }  
}  
QString DataObject::color() const {  
    return m_color;  
}  
  
void DataObject::setColor(const QString &color)  
{  
    if (color != m_color) {  
        m_color = color;  
        emit colorChanged();  
    }  
}
```

DataObject 클래스의 color( ) 멤버 함수는 QML에서 이 클래스에 저장된 m\_color 값을 리턴 한다. 따라서 DataObject 클래스의 멤버 함수를 QML에서 사용할 수 있다.

다음으로 Main.qml 파일을 아래와 같이 작성한다.

```
import QtQuick  
import QtQuick.Window  
  
Window {  
    width: 300; height: 200; visible: true  
    title: qsTr("Obect List Model")
```

```
ListView {
    anchors.centerIn: parent
    width: 100; height: 100
    model: MyModel

    delegate: Rectangle
    {
        height: 25
        width: 100
        color: model.modelData.color

        Text {
            text: name
        }
    }
}
```

다음은 DataObject 클래스의 오브젝트를 QList에 저장하고 QQmlContext 클래스를 이용해 QML로 값을 전달하기 위한 방법에 대해서 알아보자. main.cpp 파일을 열어서 아래와 같이 작성한다.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlContext>
#include "dataobject.h"

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    QList<QObject*> dataList;
    dataList.append(new DataObject("Item 1", "red"));
    dataList.append(new DataObject("Item 2", "green"));
    dataList.append(new DataObject("Item 3", "blue"));
    dataList.append(new DataObject("Item 4", "yellow"));

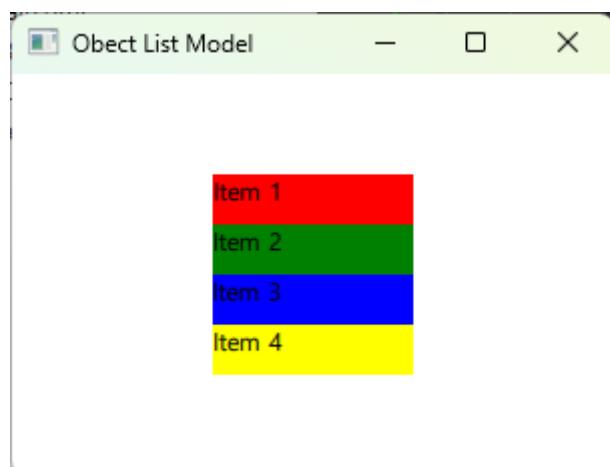
    QQmlApplicationEngine engine;
```

예수님은 당신을 사랑합니다.

```
QQmlContext *ctx = engine.rootContext();
ctx->setContextProperty("MyModel", QVariant::fromValue(dataList));

const QUrl url("qrc:/01_ObjectListModel/Main.qml");
engine.load(url);

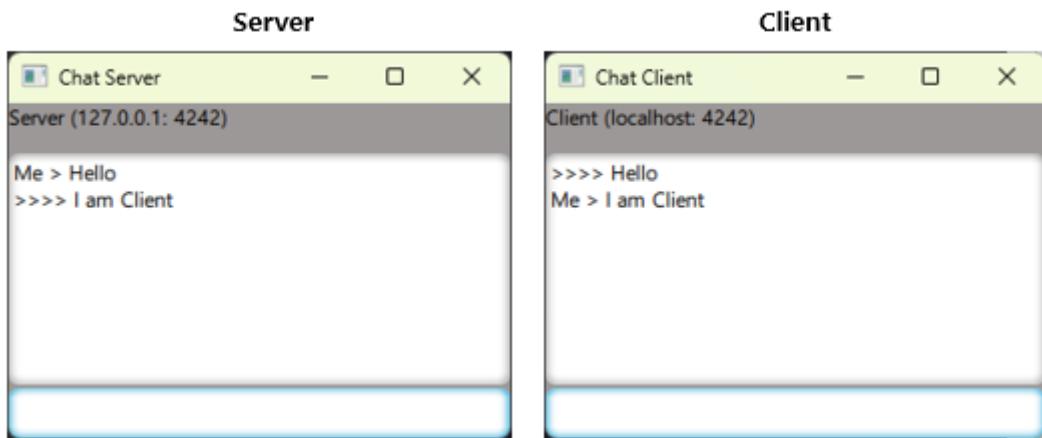
return app.exec();
}
```



이 예제의 소스코드는 01\_ObjectListModel 디렉토리를 참조하면 된다.

## 5.6. TCP 프로토콜 기반 채팅 어플리케이션 구현

이번 장에서는 C++과 QML 을 이용해 TCP 프로토콜 기반 채팅 응용 어플리케이션을 개발해 보도록 하자. 구현하는 응용 예제는 Server 와 Client 로 나눈다.



Server 와 Client 의 두 개의 예제는 다음과 같은 소스코드 파일로 구성된다. 아래 표는 Server 예제의 C++ 과 QML 소스코드 파일이다.

종류	파일명	설명
C++	main.cpp	프로그램 시작점
	tcpconnection.h	TcpConnection 클래스로 QML에서 네트워크를 경유
	tcpconnection.cpp	해 메시지를 전송 또는 수신 하는 기능이 구현
QML	Server.qml	ChatWindow.qml 에서 구현된 QML 타입을 호출해 설정
	ChatWindow.qml	QML로 GUI를 구성. 위의 실행 예제 화면에서 보는 것과 같이 Title, 채팅 내용 창 그리고 하단의 메시지 입력을 위한 화면 구성
	Display.qml	채팅 내용 창 QML 타입 구현
	LineEdit.qml	메시지 입력 QML 타입 구현

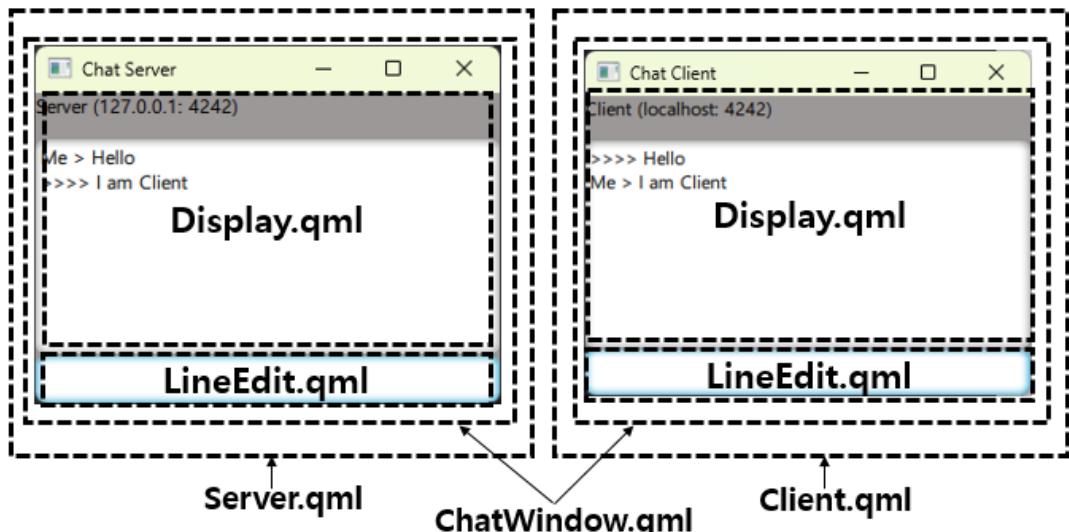
아래 표는 Client 예제의 C++ 과 QML 소스코드 파일이다.

종류	파일명	설명
----	-----	----

C++	main.cpp	프로그램 시작점
	tcpconnection.h tcpconnection.cpp	TcpConnection 클래스로 QML에서 네트워크를 경유해 메시지를 전송 또는 수신 기능 구현
QML	Client.qml	ChatWindow.qml 에서 구현된 QML 타입을 호출해 설정
	ChatWindow.qml	QML로 GUI를 구성 위의 실행 예제 화면에서 보는 것과 같이 Title, 채팅 내용 창 그리고 하단의 메시지 입력을 위한 화면 구성
	Display.qml	채팅 내용 창 QML 타입 구현
	LineEdit.qml	메이시 입력 QML 타입 구현

Server 와 Client 두 개의 예제는 동일한 TcpConnection 클래스를 사용한다. 그리고 서버 예제는 Server.qml을 사용하고 클라이언트 예제는 Client.qml을 사용하는 것 외에는 차이점이 없다.

따라서 Server.qml 과 Client.qml 만 다르고 ChatWindow.qml, Display.qml 그리고 LineEdit.qml 소스코드를 Server 와 Client 모두 동일 하다. 그리고 다른 차이점으로 Server 예제에서 main.cpp 에서 Server.qml을 호출하고 Client 예제에서는 Client.qml을 호출한다.



아래 main.cpp 는 00\_Chatting\_Server 의 main.cpp 이다.

<그림> Server 와 Client 의 소스파일 구성

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "tcpconnection.h"

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    qmlRegisterType<TcpConnection>("TCP", 1, 0, "TcpConnection");

    QQmlApplicationEngine engine;
    const QUrl url("qrc:/00_Chatting_Server/Server.qml");
    engine.load(url);

    return app.exec();
}
```

위의 예제에서 보는 것과 같이 Server 예제에서는 Server.qml 을 호출한다.

TcpConnection 클래스는 QML에서 TcpConnection QML 타입을 구현한 클래스이다. 이 클래스에서는 서버인 경우 QTcpServer 클래스를 이용해 TCP 프로토콜 기반 네트워크 채팅에 필요한 메시지 송/수신 기능이 구현되어 있다.

Client 인 경우에는 QTcpSocket 클래스를 이용해 TCP 프로토콜 기반 네트워크 채팅에 필요한 메시지 송/수신 기능이 구현되어 있다. 따라서 TcpConnection 클래스는 QML에서 네트워크의 설정과 Server 와 Client 가 메시지 송/수신 기능을 제공한다. 다음 예제는 TcpConnection 클래스의 헤더 tcpconnection.h 소스코드이다.

<예제> Ch05 > 06 > 01\_Chatting\_Server > tcpconnection.h

```
#ifndef TCPCONNECTIONELEMENT_H
#define TCPCONNECTIONELEMENT_H

#include <QObject>

class QTcpServer;
class QTcpSocket;
class TcpConnection : public QObject
{
    Q_OBJECT
```

예수님은 당신을 사랑합니다.

```
Q_PROPERTY(int port READ port WRITE setPort NOTIFY portChanged)

Q_PROPERTY(QString hostname READ hostname WRITE setHostName
           NOTIFY hostNameChanged)

Q_PROPERTY(ConnectionType type READ connectionType WRITE setConnectionType
           NOTIFY connectionTypeChanged)

public:
    enum ConnectionType { Server, Client };
    Q_ENUM(ConnectionType)

    TcpConnection(QObject *parent = nullptr);
    void setConnectionType(ConnectionType connectionType);
    ConnectionType connectionType() const;

    void setPort(int port);
    int port() const;

    void setHostName(const QString &hostName);
    QString hostName() const;

public slots:
    void initialize();
    void sendData(const QString &data);

signals:
    void dataReceived( const QString &data );
    void portChanged();
    void hostNameChanged();
    void connectionTypeChanged();

private slots:
    void receivedData();
    void slotConnection();

private:
    int m_port;
    QString m_hostName;
    ConnectionType m_connectionType;
```

```
QTcpServer *m_tcpServer;
QTcpSocket *m_tcpSocket;
};

#endif
```

다음은 tcpconnection.cpp 소스코드 파일이다.

```
#include "tcpconnection.h"
#include <QHostAddress>

#include <QTcpServer>
#include <QTcpSocket>

TcpConnection::TcpConnection(QObject *parent)
    : QObject(parent), m_hostName("127.0.0.1")
{
}

void TcpConnection::sendData(const QString &data)
{
    m_tcpSocket->write( data.toUtf8() + "\n" );
}

int TcpConnection::port() const
{
    return m_port;
}

void TcpConnection::setPort(int port)
{
    if (m_port != port) {
        m_port = port;
        emit portChanged();
    }
}

QString TcpConnection::hostName() const
{
```

예수님은 당신을 사랑합니다.

```
    return m_hostName;
}

void TcpConnection::setHostName(const QString &hostName)
{
    if (m_hostName != hostName) {
        m_hostName = hostName;
        emit hostNameChanged();
    }
}

TcpConnection::ConnectionType TcpConnection::connectionType() const
{
    return m_connectionType;
}

void TcpConnection::setConnectionType(ConnectionType connectionType)
{
    if (m_connectionType != connectionType) {
        m_connectionType = connectionType;
        emit connectionTypeChanged();
    }
}

void TcpConnection::initialize()
{
    if ( m_connectionType == Server ) {
        m_tcpServer = new QTcpServer;
        m_tcpServer->listen( QHostAddress(m_hostName), m_port );
        connect( m_tcpServer, SIGNAL( newConnection() ),
                 this, SLOT( slotConnection() ) );
    }
    else {
        m_tcpSocket = new QTcpSocket(this);
        m_tcpSocket->connectToHost( m_hostName, m_port );
        connect( m_tcpSocket, SIGNAL(readyRead()), this, SLOT(receivedData()) );
    }
}
```

예수님은 당신을 사랑합니다.

```
void TcpConnection::slotConnection()
{
    m_tcpSocket = m_tcpServer->nextPendingConnection();
    connect( m_tcpSocket, SIGNAL(readyRead()), this, SLOT(receivedData()) );
}

void TcpConnection::receivedData()
{
    const QString txt = QString::fromUtf8(m_tcpSocket->readAll());
    emit dataReceived( txt );
}
```

sendData( ) 멤버 함수는 메시지를 전송하는 기능을 제공한다. port( ) 는 현재 네트워크 포트 번호를 리턴 한다. setPort( ) 는 네트워크 포트 번호를 설정할 수 있다.

initialize( ) 멤버 함수는 만약 Server 인 경우 QTcpServer 클래스를 이용해 Server 기능을 위한 기능을 구현한다. Client 이 경우 QTcpClient 클래스를 사용한다.

slotConnection( ) Slot 함수는 새로운 접속자 이벤트가 발생하면 호출되는 Slot 함수이다. 이 Slot 함수에서는 새로운 사용자가 메시지를 수신하면 receivedData() Slot 함수가 호출될 수 있도록 connection() 함수를 이용해 readyRead() 시그널 이벤트를 처리한다. readyRead( ) 네트워크로부터 메시지 수신 시 발생하는 Signal 이다. 다음은 Server 에서 사용하는 Server.qml 예제를 살펴 보도록 하자.

```
import QtQuick
import QtQuick.Window
import TCP 1.0

Window {
    width: 300; height: 200; visible: true
    title: "Chat Server"

    ChatWindow {
        width: 300
        height: 200
        type : TcpConnection.Server
        port : 4242
    }
}
```

위의 Server.qml 에서는 type 프로퍼티 값으로 TcpConnection.Server 를 사용하였다.

예수님은 당신을 사랑합니다.

Client.qml 에서는 TcpConnection.Client 를 사용한다. 다음은 ChatWindow.qml 파일이다.

```
import QtQuick
import TCP 1.0

Item {
    property alias type : tcpConnection.type
    property alias port : tcpConnection.port
    property alias hostName : tcpConnection.hostName

    TcpConnection {
        id : tcpConnection
        onDataReceived : {
            output.text += ">>>> " + data
        }
    }

    Component.onCompleted: tcpConnection.initialize()

    Rectangle {
        color: "#9c9898"; border.width: 0;
        border.color: "#000000"; anchors.fill: parent
    }

    Text {
        id : title
        text: titleText()
        anchors {
            top : parent.top
            left : parent.left
            right : parent.right
        }
        height : 30
    }

    Display {
        id: output
        height:300
        anchors.bottomMargin: 2
        anchors {
            top : title.bottom
        }
    }
}
```

예수님은 당신을 사랑합니다.

```
        left : parent.left
        right : parent.right
        bottom : entryRect.top
    }
}

LineEdit {
    id: entryRect
    x: 0; y: 83; width: 100; height: 30
    focus: true
    anchors {
        bottom : parent.bottom
        left : parent.left
        right : parent.right
    }
    onTextEntered: function(text) {
        output.text += "Me > " + text + "\n"
        tcpConnection.sendData(text)
    }
}

function titleText() {
    return (tcpConnection.type == TcpConnection.Server ? "Server" : "Client")
        + " (" + hostName + ":" + port + ")"
}
}
```

ChatWindow.qml 은 GUI 의 화면을 구성한다. Server 와 Client가 동일한 소스코드를 사용한다. 다음은 Display.qml 예제 소스코드 이다.

```
import QtQuick

Rectangle {
    property alias text: output.text
    color: "transparent"
    border.color: "transparent"
    BorderImage {
        anchors.fill: parent
        border { left: 5; top: 5; right: 5; bottom: 6 }
        horizontalTileMode: BorderImage.Stretch
```

예수님은 당신을 사랑합니다.

```
    verticalTileMode: BorderImage.Stretch  
    source: "./images/textoutput.png"  
}  
  
TextEdit {  
    id: output  
    anchors { margins: 3; fill: parent }  
    selectionColor: "transparent"  
}  
}
```

위의 Display.qml 은 채팅 내용이 출력된다. 예를 들어 원격의 다른 사용자가 메시지를 보내거나 내가 메시지를 전송하는 모든 메시지가 출력된다. 다음은 메시지 입력 창 기능을 제공하는 LineEdit.qml 예제 소스코드이다.

```
import QtQuick  
  
Rectangle {  
    signal textEntered(string text)  
  
    height: entryField.height+6  
    color: "transparent"  
    border.color: "transparent"  
    anchors {  
        rightMargin: 0; leftMargin: 0; bottomMargin: 0  
    }  
  
    BorderImage {  
        anchors.fill: parent  
        border { left: 5; top: 5; right: 5; bottom: 6 }  
        horizontalTileMode: BorderImage.Stretch  
        verticalTileMode: BorderImage.Stretch  
        source: "./images/textinput.png"  
    }  
    TextInput {  
        id : entryField  
        anchors { margins: 3; fill: parent }  
        focus: true  
  
        Keys.onReturnPressed : {
```

예수님은 당신을 사랑합니다.

```
    textEntered(text)
    text = ""
}
}
}
```

서버 예제는 00\_Chatting\_Server 디렉토리를 참조하면 된다. 그리고 클라이언트 예제는 00\_Chatting\_Client 디렉토리를 참조하면 된다.