

For rapid development of modern UX

# Qt Quick programming

First Edition, Ver 1.5



Qt Quick can implement GUI in QML, functions can be implemented in C++, so you can improve code reusability.

Qt Korea Developer Community  
[www.qt-dev.com](http://www.qt-dev.com)

**Jesus loves you.**

## **Qt Quick programming**

---

**Version** Version 1.5, 2020.10.12

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

---

**Jesus loves you.**

## Preface

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

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

[NIV] Matthew 22:37~39

**Jesus loves you.**

## **Table of Contents**

---

<b>1. What is Qt Quick ?.....</b>	<b>1</b>
<b>2. QML Basic .....</b>	<b>4</b>
<b>2.1. QML Syntax .....</b>	<b>5</b>
<b>2.2. Types.....</b>	<b>16</b>
<b>2.3. Event .....</b>	<b>34</b>
<b>2.4. Dynamic GUI Configuration Using Loader Type .....</b>	<b>49</b>
<b>2.5. Canvas.....</b>	<b>63</b>
<b>2.6. Graphics Effects .....</b>	<b>69</b>
<b>2.7. QML Module Programming.....</b>	<b>93</b>
<b>2.8. How to use JavaScript in QML .....</b>	<b>101</b>
<b>2.9. Dialog.....</b>	<b>112</b>
<b>2.10. Layout.....</b>	<b>119</b>
<b>2.11. Type Positioning.....</b>	<b>126</b>
<b>3. Animation Framework.....</b>	<b>132</b>
<b>3.1. Animation.....</b>	<b>134</b>
<b>3.2. Example using Animation and Easing Curve .....</b>	<b>147</b>
<b>3.3. State and Transition.....</b>	<b>156</b>
<b>3.4. Image Viewer Example .....</b>	<b>164</b>
<b>4. Model / View.....</b>	<b>177</b>
<b>4.1. Data representation using Model / View in QML.....</b>	<b>178</b>
<b>4.2. Implementing Knight in chess .....</b>	<b>190</b>
<b>5. Integration QML and C+.....</b>	<b>196</b>

# **Jesus loves you.**

5.1. Overview.....	197
5.2. Implementing QML Type with C++ .....	211
5.3. Use the <code>QQuickPaintedItem</code> class in QML.....	224
5.4. Scene Graph.....	228
5.5. Mapping Data Variables with Interaction between C++ and QML....	237
5.6. Implement TCP protocol-based chat applications.....	252
6. Qt Quick Controls.....	263
6.1. Qt Quick Controls 1 .....	264
6.2. Qt Quick Controls 1 Styles QML Types.....	288
6.3. Qt Quick Controls 2 .....	292
7. Qt Quick Extras.....	297
7.1. Interactive Controls.....	298
7.2. Non-interactive Controls.....	307
7.3. Extra Style .....	313



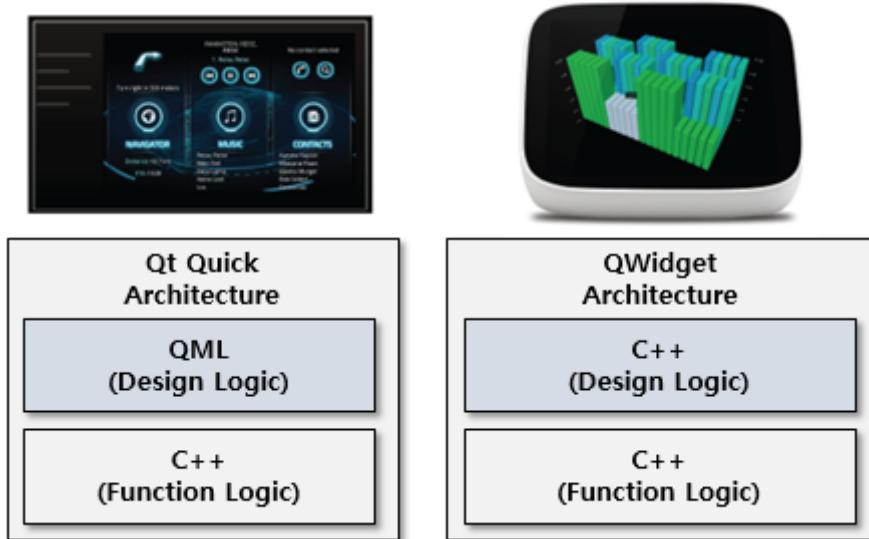
## 1. What is Qt Quick ?

Qt Quick is used to easily design and implement modern GUI interface. Do not use C++ when implementing GUI using Qt Quick. Qt Quick uses the interpreter language called QML. QML stands for Qt Modeling Language.

It is not always necessary to implement a GUI using Qt Quick. You can use C++ or QML when implementing an application using Qt. Whatever you use, there are pros and cons.

If Qt Quick is used, it can be separated into design and functional logics as an advantage. Design logic, for example, means GUI. Functional logics means function. Suppose you click a button on a design logic GUI and the function is executed. On the screen, a button is a design logic, and when you click a button, you can call a function logic.

As such, if design logic is implemented using QML provided by Qt Quick, it is completely separated from the functional logic implemented in C++, and function logic, which is a function implementation module, can be reused even if the design logic GUI is changed.

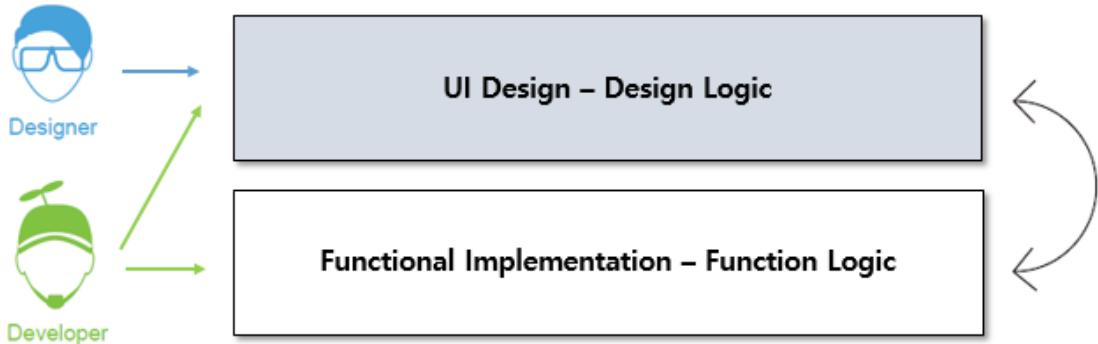


<FIGURE> Differences in Architecture Using Qt Quick and QWidget

And another advantage when using Qt Quick is that it is appropriate to use in an environment such as embedded devices with limited user interfaces. Desktop PCs, for example, can use a variety of user interfaces, including keyboards and mice, for user input. However, it is

## **Jesus loves you.**

appropriate to use Qt Quick in situations where the user interface is limited, such as industrial embedded computers that can only use smartphones or touch.



<FIGURE> Qt Quick Features

Conversely, it may be appropriate to use C++ if the computer or Embedded device in which the application you are trying to develop is in a very limited or low hardware resource, such as CPU, memory, etc. For example, if you need to respond quickly in a limited environment, it is desirable to implement a GUI using C++.

In other words, using QML provided by Qt Quick does not necessarily mean developing good application applications. Please consider whether the application you are trying to develop operates in a computing environment with high hardware specifications such as desktops, or what different user input interfaces are before deciding whether to use Qt Quick. A summary of Qt Quick can be summarized as follows:

- ① 1 Qt Quick is used to easily implement a modern GUI interface.
- ② 2 Qt Quick uses the interpreter language called QML to implement the GUI.
- ③ 3 QML is a procedural language (declarative or interpreter language).

QML, used in Qt Quick, is an interpreter language. Since it is an interpreter language that does not compile like C++ and sequentially interpret it, it is somewhat slower in performance than in C++ in terms of performance. For example, Qt Quick also uses a separate software stack called Qt Declarative, such as using software Stack such as JAVA Virtual Machine when JAVA is operating. An example is HTML in an interpreter language similar to QML. QML is the same interpreter language, as is parsing in a web browser. So far, we have searched about Qt Quick and QML.

## **Jesus loves you.**

If you summarize Qt Quick, the advantage is that if you use Qt Quick to develop an application application, design logic and functional logic are separated, so if you use functional logic other than design logic in a project developed by Qt Quick, you can increase reuse. The disadvantage is that Qt Quick uses design logic as QML instead of C++, so it is slow compared to C++. However, the disadvantage of Qt Quick compared to C++ can be overcome if sufficient hardware resources are provided.

So far, we have found out what the characteristics and benefits of Qt Quick are. Using Qt Quick is not a good thing. Sometimes it's better not to use it. Before using Qt Quick, it is recommended that you use the software you wish to develop after reviewing the features and benefits of Qt Quick as mentioned above.

## 2. QML Basic

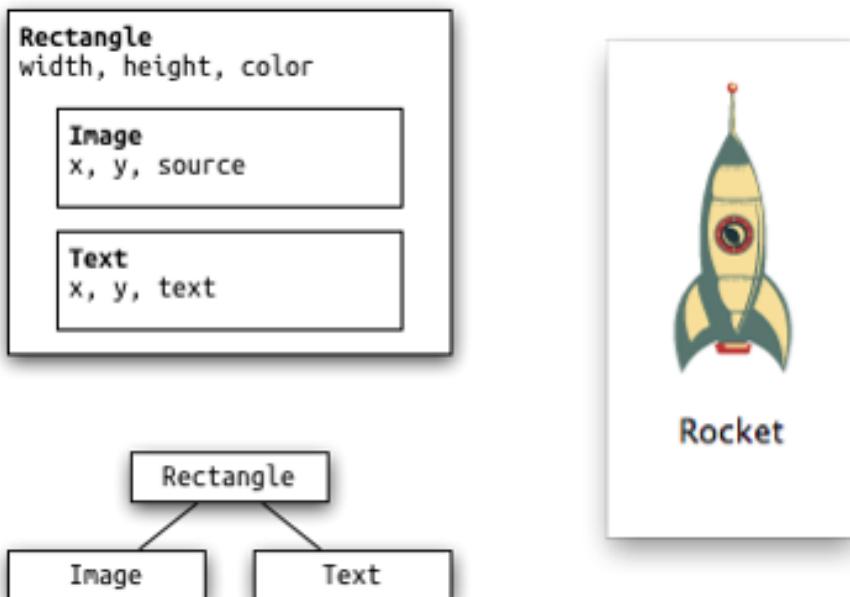
In this chapter, we will look at QML's grammar, data type, QML Element, event processing, components, etc., which are used in Qt Quick, which is essential for implementing GUI. The contents to be discussed in this chapter are as shown in the following table.

<TABLE> What to learn in this chapter

Title	Description
QML Syntax	Description of grammar and structure used in QML
Types	How to use the QML type, for example, to implement a GUI using QWidget on C++
Event	User input events
Loader	Methods for dynamically invoking another QML file or type in QML
Canvas	Methods for drawing within a QML area, such as the QPainter class used in the QWidget class
Graphics Effects	Blending, Masking, Blurring, Effect, etc
QML Module Programming	Modularization to re-use frequently used user-defined types in QML like libraries
GML에서 JavaScript 사용	How to use JavaScript within QML
Dialog	Simple dialogue, color dialogue, file dialogue, font dialogue, etc.
Layout	Layout provided by QML
Type Positioning	The ability to obtain information about a particular type of location among several types.

## 2.1. QML Syntax

QML is similar to HTML structure. And QML has a basic structure in a hierarchy up and down. For example, suppose you print an image within a square area.



<FIGURE> Rocket image and Text Display

In the above figure, if the rocket image to the right is displayed and the text "Rocket" below is displayed in any square area, Rectangle can be declared first and Image and Text can be displayed as shown in the above figure. As such, the structure of displaying images and textures is a top-down hierarchy. In other words, QML has a hierarchy structure, such as HTML, unlike grammar such as C/C++. The following source code is the QML example source code.

```
import QtQuick 2.12
import QtQuick.Window 2.12

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

## Jesus loves you.

```
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'  
}  
}
```

In the QML example above, the uppermost import statement is similar to include in C++ grammar. The following is the Syntax structure of the import statement. ModuleIdentifier specifies the module name and Version.Number indicates the version.

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

Qualifier can be used by the user to specify an alias or Alias to replace the module name. This will be dealt with in detail in the future.

In the QML example source code above, the Window type (also called Element) is the type that creates the most upper window. The items specified in the Window Type sub(internal) are called Properties. In addition, subtypes such as Image and Text may exist in the Window Type in a hierarchical structure.

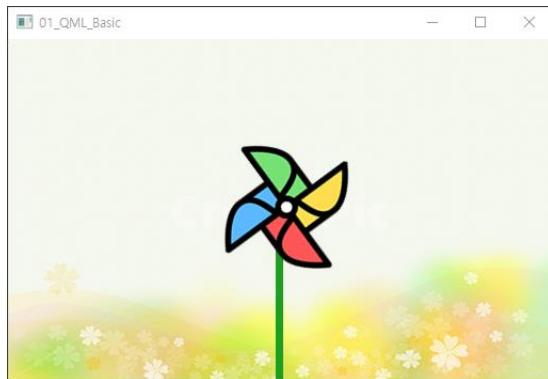
As you can see in the example, you can decide whether to display visible properties in the Window type on the Window type. Id Property specifies the unique name or ID of the type. Width and height are horizontal and vertical. Color Properties can be color specified within the Window Type.

Image types can display image resources within the Window type. Parent.width used within Image means window type width. x and y are the coordinates of the starting position to be displayed in the Image-type Window Type. Source Property specifies the location and filename of the image resource. The value of rocket.y used in the Text Type can be referenced to the y-value of the type Image-type ID Property is rocket.

## Jesus loves you.

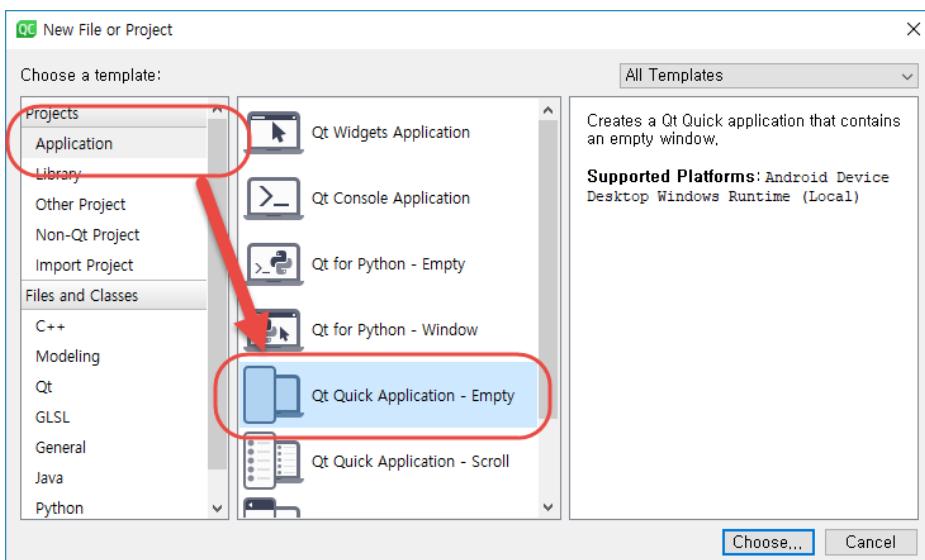
- Example of QML with Image and Mouse

This example rotates the image when the left and right turn keys of the mouse and keyboard are pressed within the Window Type area, and when the image is clicked.



<FIGURE> Example Execution Screen

A mouse click on the screen rotates the pinwheel image as shown in the picture above. And when you click the left turnkey on the keyboard, the pinwheel image rotates to the left, and when you click the right turnkey, it rotates to the right. To develop an application using Qt Quick, create a new project in Qt Creator. Select the [Application] item on the left side of the [New File or Project] dialogue and select the [Qt Quick Application – Empty] item from the central tab as shown in the following figure.

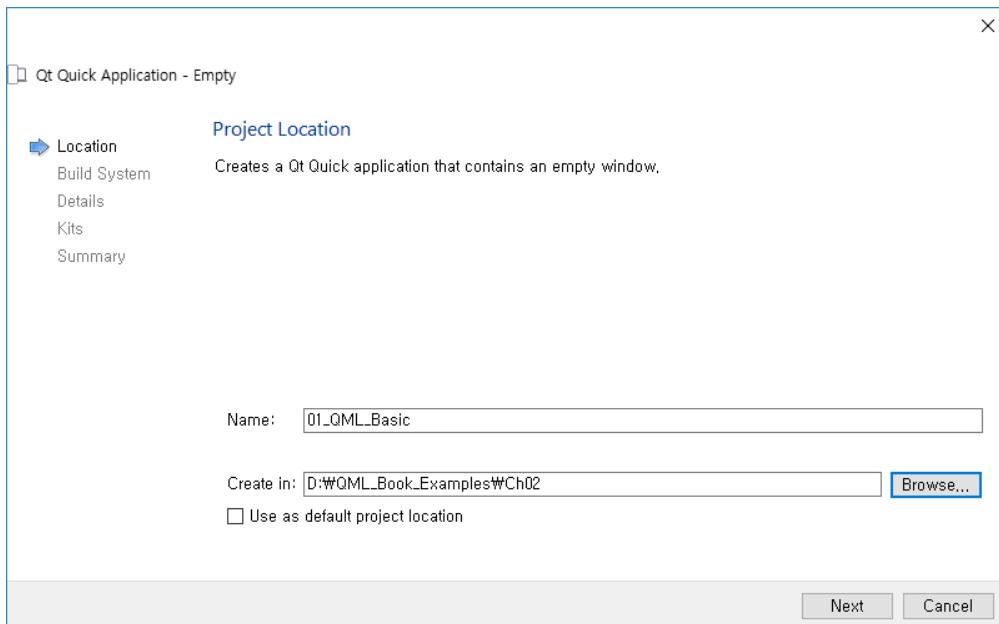


<FIGURE> Project Generation Screen

Select the [Qt Quick Application – Empty] item and click the [Choose] button at the bottom. Select the name you want to assign the project: Enter the name of the project in the [Name]

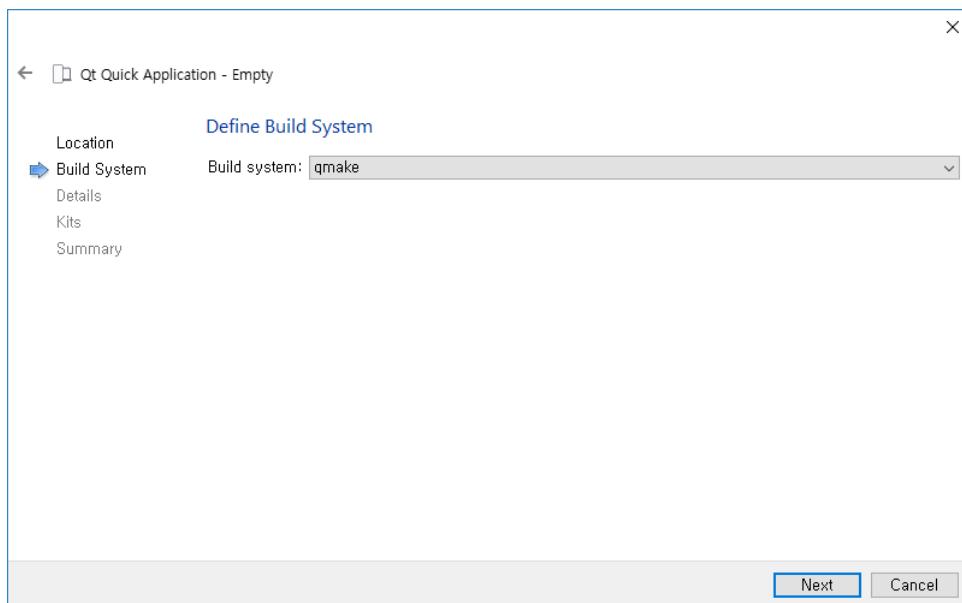
# Jesus loves you.

entry and select the parent directory where the directory created with the project name will be created at the bottom.



<FIGURE> Project Name and Project Parent Directory

Specify the project name and project parent directory and click the [Next] button at the bottom.

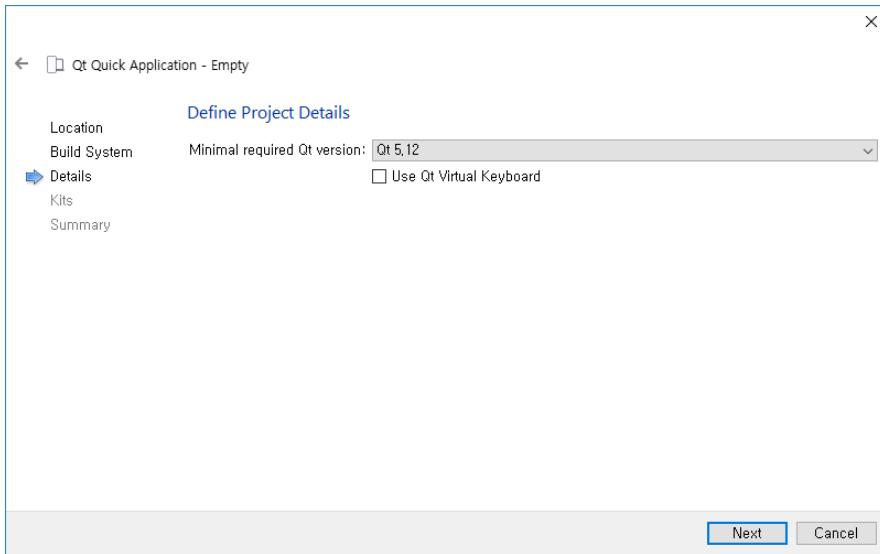


<FIGURE> Build System Settings

The figure above selects the build system. Select qmake and click the [Next] button as you

## Jesus loves you.

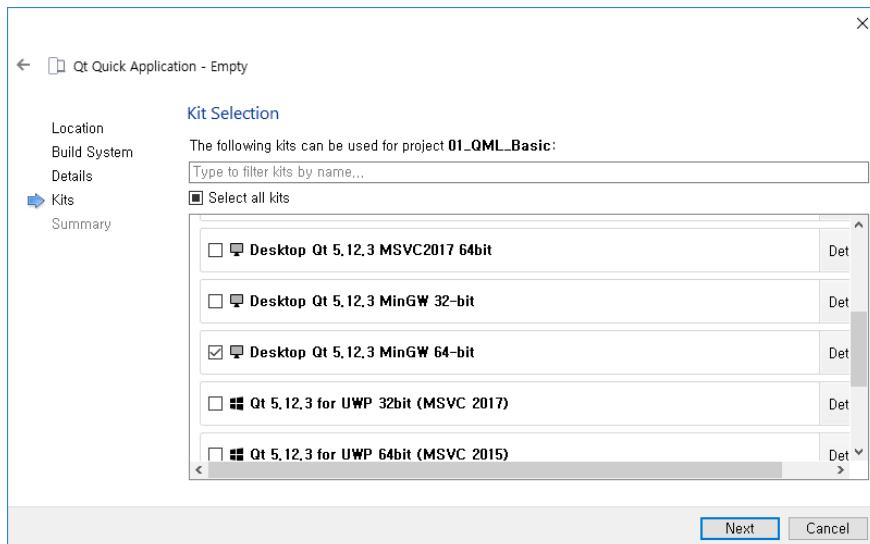
can see in the picture. The following specifies the version of Qt to use in Qt Quick. Select the same version as the currently installed version as shown in the following figure.



<FIGURE> Select Qt Version

And where you select the version, the [Use Qt Virtual Keyboard] check box asks if you want to use the virtual keyboard in the application you are currently implementing. The virtual keyboard provides the same function as the virtual keyboard shown at the bottom of the mobile phone screen when typing text on the phone.

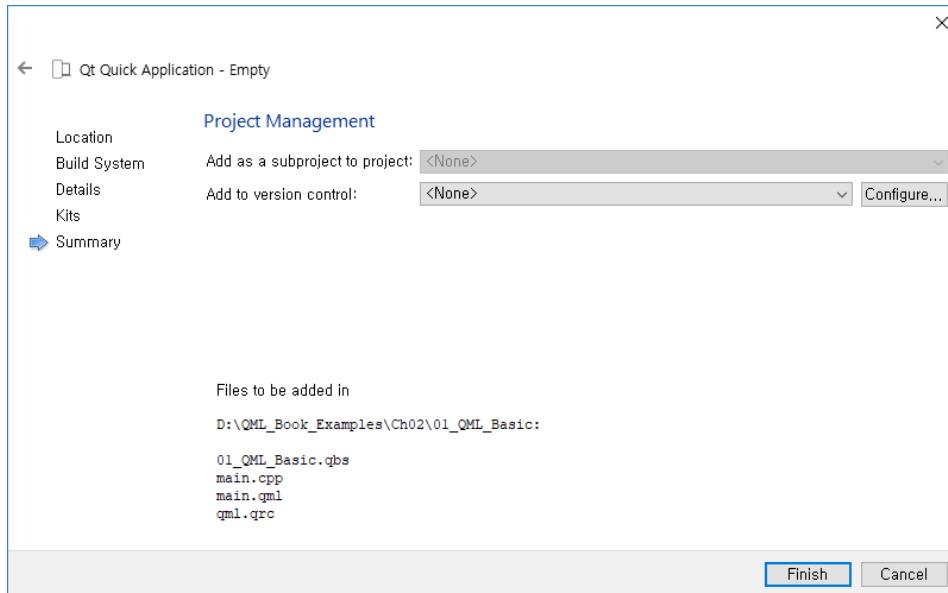
Here, click the [Next] button without checking the [Use Qt Virtual Keyboard] checkbox as shown in the figure above. The next screen is to select the compiler to build the application.



<FIGURE> Select Compiler

## Jesus loves you.

Select the MinGW compiler as shown in the figure above. Then click the [Next] button at the bottom.



<FIGURE> Project Management Version Selection Screen

The above picture is the last screen of the project creation. You can specify one of the versioning systems, such as SVN or Git. Here, select None and click the Finish button. When the project creation is complete, a project file named 01\_QML\_Basic.pro is created. This file specifies the properties of the project. Next, main.cpp and main.qml are created. These two files are source code. Call main.qml from main.cpp. Let's look at the main.cpp source code first.

<Example> Ch02 > 01 > 01\_QML\_Basic

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

int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);

    QQmlApplicationEngine engine;
    const QUrl url(QStringLiteral("qrc:/main.qml"));
}
```

## Jesus loves you.

```
QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,
                 &app, [url](QObject *obj, const QUrl &objUrl) {
    if (!obj && url == objUrl)
        QCoreApplication::exit(-1);
}, Qt::QueuedConnection);

engine.load(url);

return app.exec();
}
```

As seen in the example source code above, the QQmlApplicationEngine class serves to load QML files. You can load the QML file using the load( ) member function at the bottom of the source.

In the source code, the connect( ) function is invoked when a signal from an objectCreated object in the QQmlApplicationEngine class occurs. This signal is called when loading the engine object is complete. If QML loading fails, exit the program. The following example is the main.qml example. It automatically generates basic QML source code. Change this source code as follows.

```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    visible: true
    id: root
    width: 512; height: 320; color: "#D8D8D8"
    property int rotationStep: 45
    BorderImage {
        source: "images/background.png"
    }
    Image {
        id: pole
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.bottom: parent.bottom
        source: "images/stick.png"
    }
    Image {
```

## **Jesus loves you.**

```
id: wheel
anchors.centerIn: parent
source: "images/wheel.png"
Behavior on rotation {
    NumberAnimation {
        duration: 250
    }
}
MouseArea {
    anchors.fill: parent
    onPressed: {
        wheel.rotation += 90
    }
}
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
    }
}
```

Window type is the highest type in QML. Size the horizontal and vertical sizes using width and height. Color Properties are the background color of the Window type.

The BorderImage type specifies the image to be used as a background image. In Image type, id Property is the shape of a pinwheel rod. And the image of id Property being wheel is a pinwheel image. Each image is shown below.

## **Jesus loves you.**



**background.png**



**stick.png**



**wheel.png**

<FIGURE> Resources

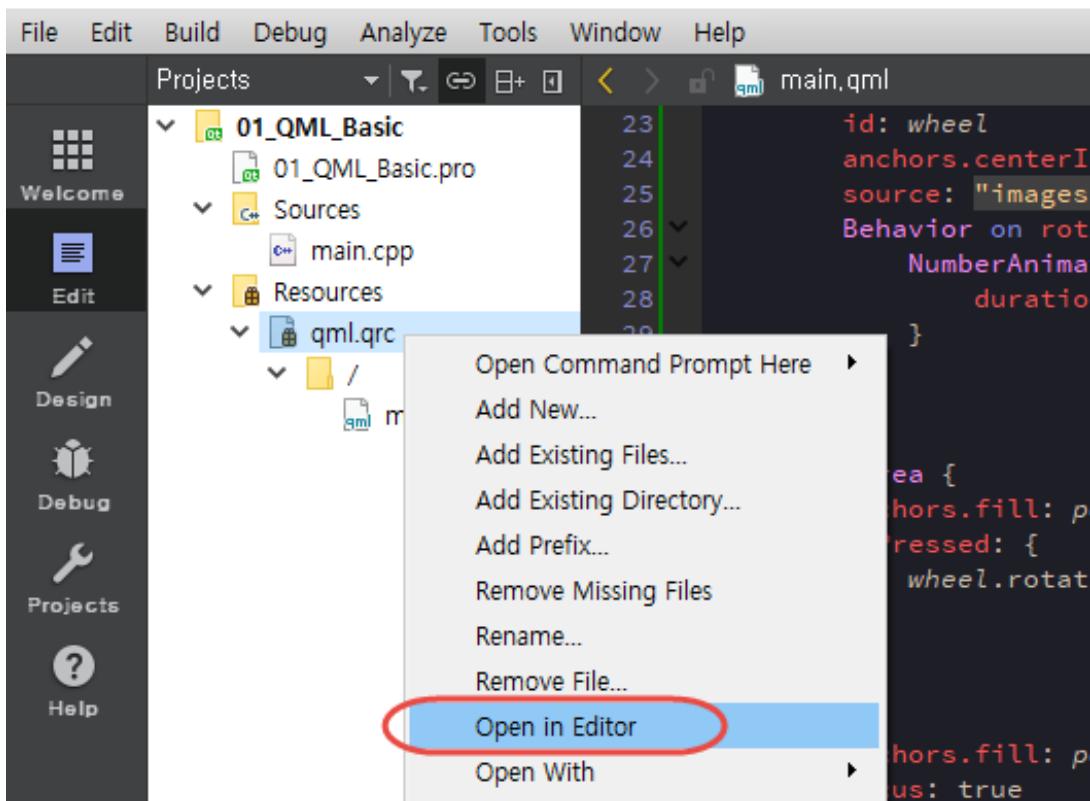
The MouseArea type handles mouse events that occur within the Window area. When the mouse button is clicked, rotate it 90 degrees as defined by the onPressed type of MouseArea type. In 90 degree rotation, the NumberAnimation type declared in the image type of the wheel ID is animation.

The animation is applied when a mouse event occurs in MouseArea and rotates 90 degrees. This means that a 90 degree rotation is not instantaneous, but 250 milliseconds are used to rotate 90 degrees.

The Item Type rotates -90 degrees when the left turnkey is pressed on the keyboard. Rotate 90 degrees when the right direction key is pressed. To add an image to a project, create an image directory in the project subdirectory. Then copy three images. The image file is attached to the example project.

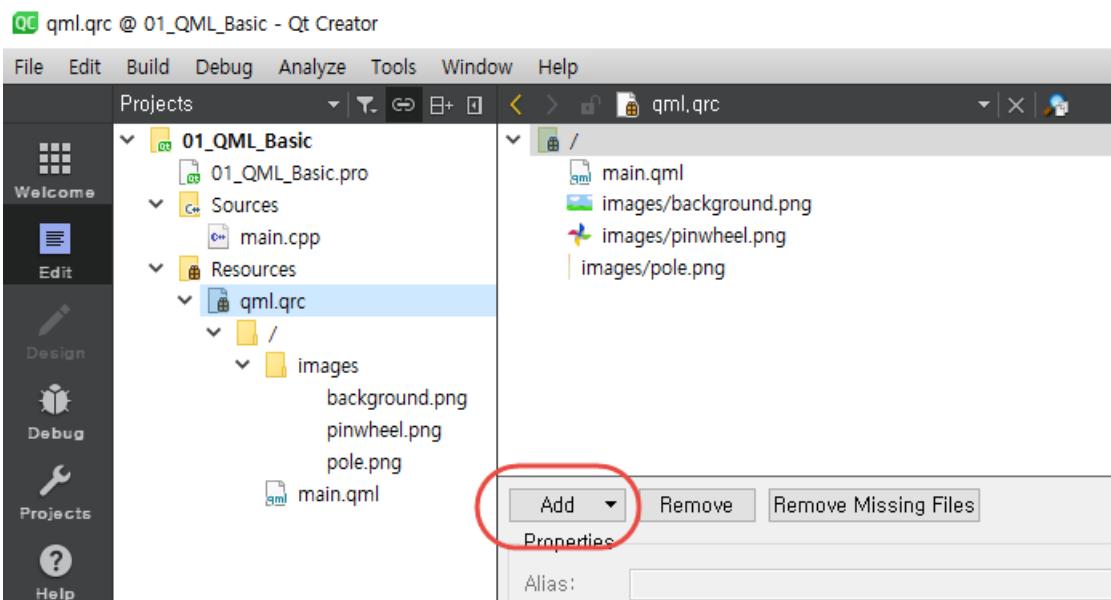
# Jesus loves you.

QC main.qml @ 01\_QML\_Basic - Qt Creator



<FIGURE> Resource Registration Screen

Select the qml.qrc file, then right-click and select [Open in Editor] from the menu. Then register the image as shown in the following figure.



## **Jesus loves you.**

<FIGURE> Resource Registration Screen

Click the [Add] button to register the resource. Then build the source, run it, click the mouse and see if the pinwheel rotates. Let's also check if the pinwheel rotates when the left/right key on the keyboard is pressed.

## 2.2. Types

In QML, Type refers to the object provided to implement the GUI. Window, Image, MouseArea, and Item used in the previous section are called Type in QML. QML provides various types and frequently used types are shown in the following table.

**<TABLE> Frequently Used Type**

Type	Description
Rectangle	Type item in square area
Image	Items for displaying images
BorderImage	Display Background Image in Specified Area
AnimatedImage	Items for displaying items such as moving GIFs
AnimatedSprite	Items for displaying animation using continuous frames
SpriteSequence	Display multiple items using continuous frames
Text	Item to display string
Window	Item to create upper window
Item	user-defined item
Anchors	Provides features such as layout
Screen	Information about areas where items are displayed
Sprite	Specify animation for displayed items
Repeater	Offers multiple created items to be applied to the model
Loader	Display detached items, such as modules or files
Transform	Move Item
Scale	Controlling the size of an item
Rotate	Rotation of Item
Translate	Define the moving properties of an item

# **Jesus loves you.**

- **Rectangle**

Rectangle type provides the function for displaying items in the square area. And the Rectangle type can be used in a superimposed form.

<Example> Ch02 > 01 > 01\_Type\_Basic > rectangle.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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

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

        Rectangle { /
            x: 50; y: 50
            width: 300; height: 150
            color: "lightblue"

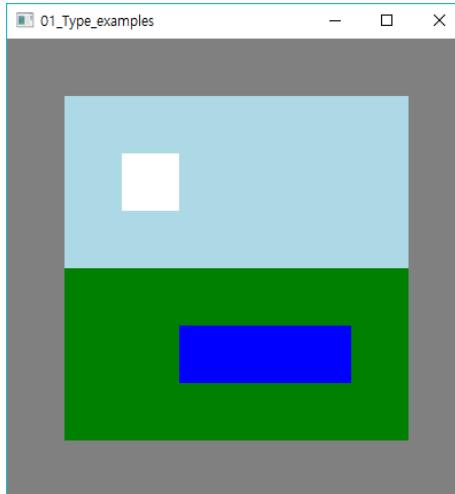
            Rectangle {
                x: 50; y: 50
                width: 50; height: 50
                color: "white"
            }
        }

        Rectangle {
            x: 50; y: 200
            width: 300; height: 150
            color: "green"

            Rectangle {
                x: 100; y: 50
                width: 150; height: 50
                color: "blue"
            }
        }
    }
}
```

## Jesus loves you.

```
    }  
}  
}
```



<FIGURE> Rectangle Example Run Screen

### ● Image

An example is for displaying images using Image Type.

<Example> Ch02 > 02 > 01\_Type\_Examples > image.qml

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
  
Window {  
    visible: true  
    width: 100; height: 100  
  
    Rectangle {  
        width: 100; height: 100  
        color: "white"  
        Image {  
            x: 10; y: 10  
            source: "./images/qtlogo.png"  
        }  
    }  
}
```

## Jesus loves you.



<FIGURE> Image Type Example Execution Screen

- **AnimatedImage**

AnimatedImage Type allows the rendering of moving GIF images on the screen and provides start and stop functions.

<Example> Ch02 > 02 > 01\_Type\_Examples > animatedimage.qml

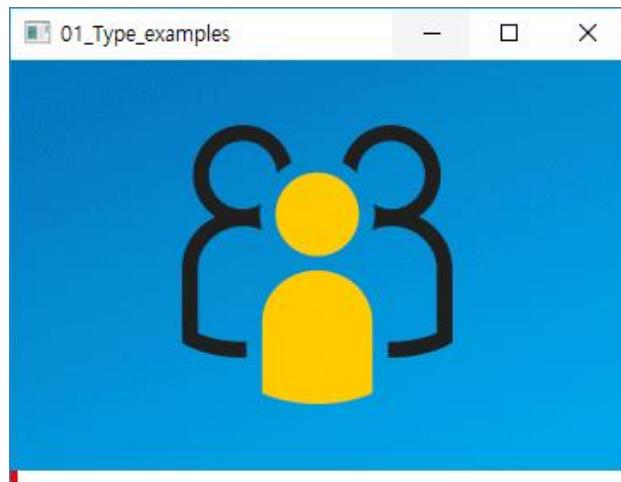
```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    visible: true
    width: animation.width
    height: animation.height + 8

    AnimatedImage {
        id: animation
        source: "images/ani.gif"
    }
    Rectangle
    {
        property int frames: animation.frameCount
        width: 4; height: 8
        x: (animation.width - width) * animation.currentFrame / frames
        y: animation.height
        color: "red"
    }
}
```

## Jesus loves you.

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



<FIGURE> Example Execution Screen

### ● anchors

QML provides anchors, such as sorting widgets using layouts such as QHBoxLayout and QVBoxLayout in the Qt C++ API. The following is an example of using anchors to deploy QML type.

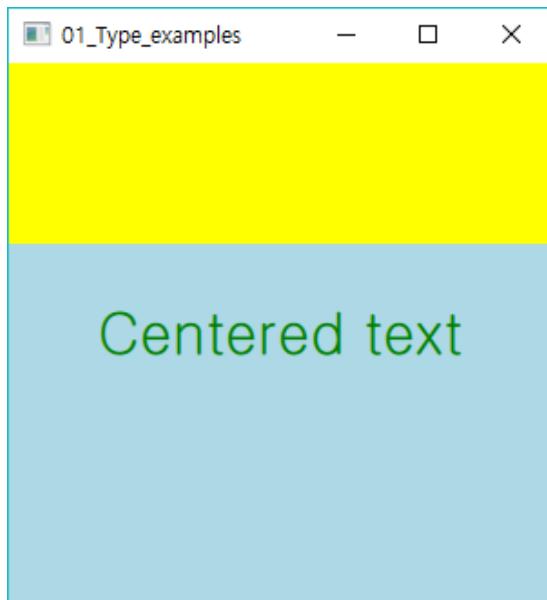
<Example> Ch02 > 02 > 01\_Type\_Examples > anchors.qml

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
  
Window {  
    visible: true; width: 300; height: 300
```

## Jesus loves you.

```
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  
    }  
}  
}  
}
```

The text type was arranged using anchors. In the Text Type area, the anchors.centerIn Property value is specified as reactangle1. Therefore, place the text type in the center ofctangle1.



<FIGURE> Example Execution Screen

# Jesus loves you.

- **Example of deploying multiple types with anchors**

The following QML example source code is an example of placing QML types using anchors.

<Example> Ch02 > 02 > 01\_Type\_Examples > text\_image\_anchors.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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

## Jesus loves you.

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



<FIGURE> Example Execution Screen

### ● Gradient

Gradient provides the ability to specify a range of colors in a specified area. Values in the

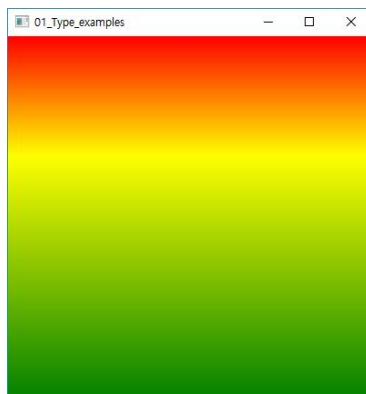
## Jesus loves you.

range can be specified between 0.0 and 1.0. The following example is an example of using Gradient.

<Example> Ch02 > 02 > 01\_Type\_Examples > gradient.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    visible: true; width: 400; height: 400
    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" }
        }
    }
}
```



<FIGURE> Example Execution Screen

### ● SystemPalette

SystemPalette provides access to the Qt application Palette. It provides information about the standard colors used in the application window. The following example is an example of using SystemPalette.

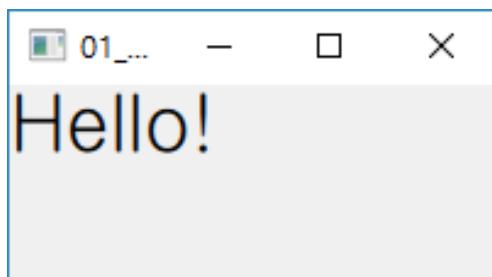
<Example> Ch02 > 02 > 01\_Type\_Examples > systempalette.qml

```
import QtQuick 2.12
```

## Jesus loves you.

```
import QtQuick.Window 2.12

Window {
    visible: true; width: 640; height: 480
    Rectangle {
        width: 640; height: 480
        color: myPalette.window
        SystemPalette {
            id: myPalette
            colorGroup: SystemPalette.Active
        }
        Text {
            id: myText
            anchors.fill: parent
            text: "Hello!";
            font.pixelSize: 32
            color: myPalette.windowText
        }
    }
}
```



<FIGURE> Example Execution Screen

### ● Screen

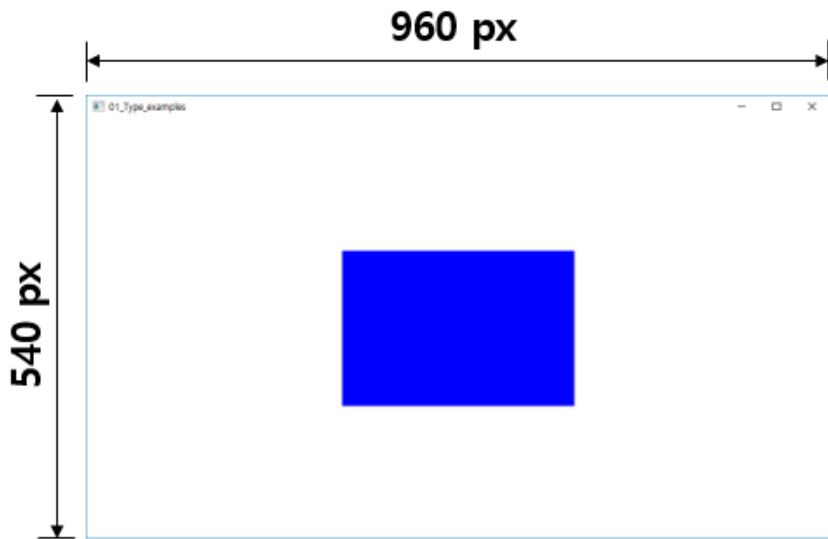
Screen provides information related to the screen where the GUI is loaded. The following example reads the screen resolution information of the system.

<Example> Ch02 > 02 > 01\_Type\_Examples > screen.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
```

## Jesus loves you.

```
Window {  
    visible: true  
  
    // 예) 디스플레이 Resolution: 1920x1080, width = 1920/2, height = 1080/2  
    width : Screen.width / 2;  
    height : Screen.height / 2;  
  
    Rectangle  
    {  
        width : 300  
        height: 200  
        color: "blue"  
        anchors.centerIn: parent  
    }  
}
```



<FIGURE> Screen Example Execution Screen

### ● **FontLoader**

FontLoader can specify which font to use.

<Example> Ch02 > 02 > 01\_Type\_Examples > fontloader.qml

```
import QtQuick 2.12  
import QtQuick.Window 2.12
```

## Jesus loves you.

```
Window {  
    visible: true; width: 400; height: 400  
    Rectangle {  
        FontLoader {  
            id: fixedFont; name: "Courier"  
        }  
        FontLoader {  
            id: webFont  
            source: "http://www.mysite.com/myfont.ttf"  
        }  
        Text {  
            text: "Fixed-size font";  
            font.family: fixedFont.name  
        }  
    }  
}
```

### ● Repeater

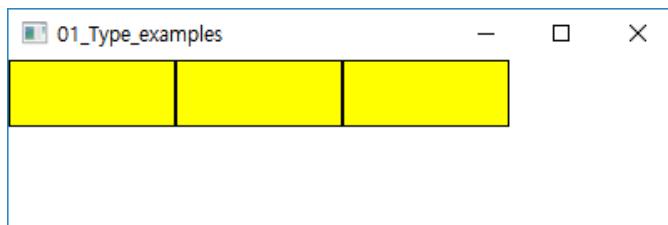
Repeater types can place the same QML type continuously. The following example is an example of using Repeater.

<Example> Ch02 > 02 > 01\_Type\_Examples > repeater.qml

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
  
Window {  
    visible: true; width: 400; height: 100  
    Row {  
        Repeater {  
            model: 3  
            anchors.top: parent.top  
            Rectangle {  
                width: 100; height: 40  
                border.width: 1  
                color: "yellow"  
            }  
        }  
    }  
}
```

## Jesus loves you.

```
}
```



<FIGURE> Example Execution Screen

- **Image**

Image provides the function for displaying images. The following is an example source code for displaying images on the screen using the Image type.

<Example> Ch02 > 02 > 01\_Type\_Examples > image.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    visible: true; width: 300; height: 250
    Rectangle {
        width: 300; height: 250
        color: "white"
        Image {
            x: 10; y: 10
            source: "./images/qtlogo.png"
        }
    }
}
```

## Jesus loves you.



<FIGURE> Example Execution Screen

### ● Transformation 과 Rotation

Transformation and Rotation provide functions such as position movement of QML type and rotation of QML type. The following example changes the angle of the image type to the Y-axis.

<Example> Ch02 > 02 > 01\_Type\_Examples > transformation.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    visible: true; width: 1000; height: 220

    Row { x: 10; y: 10; spacing: 10
        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
            }
        }
    }
}
```

## Jesus loves you.

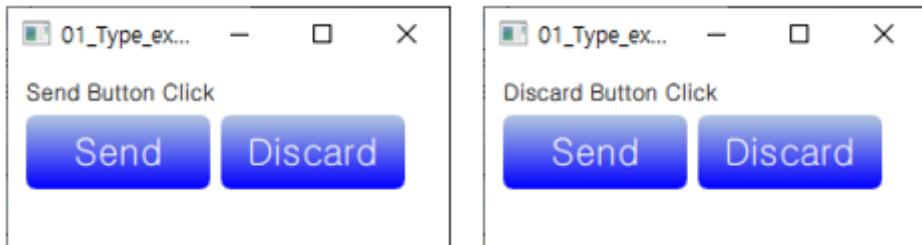
```
Image {  
    source: "images/qtlogo.png"  
    transform: Rotation {  
        origin.x: 30; origin.y: 30  
        axis { x: 0; y: 1; z: 0 } angle: 54  
    }  
}
```



<FIGURE> Transformation and Rotation Example Execution Screen

- **Implementing button functions using Accessible**

This example uses Accessible to create and deploy buttons. The following figure is an example run screen.



<FIGURE> Example execution screen with Accessible

Click the [Send] button to print the "Send Button Click" message on the text type at the top as shown in the figure above. Then click the [Discard] button to output the message "Click the Discard button". Let's take a look at Button.qml among the two QML files we will cover in this example.

<Example> Ch02 > 02 > 01\_Type\_Examples > simple\_accessible.qml

```
import QtQuick 2.12
```

## Jesus loves you.

```
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()
    }
}
```

The above example source code is saved under the name of the Button.qml file. The file name becomes the QML type name when another QML file tries to use the QML type that

## **Jesus loves you.**

implements the button.

Accessible is used to implement user-defined GUI interfaces such as buttons, text input windows, and check boxes. Accessible.role allows you to choose what kind of user-defined GUI interface is. Button is designated here. In addition, you can specify CheckBox, RadioButton, Slider, SpinBox, Dialog, ScrollBar, etc.

Accessible.onPressAction specifies an event when a button is clicked. We used signal clicked here. This is the same as the signal used for header files in C++. In other words, it was defined to use the signal clicked. Radius is used to round each corner of the button. Anialiasing is used to display on the screen by applying a Vector. Next, let's look at an example of using Button.qml.

```
import QtQuick 2.12
import QtQuick.Window 2.12
import "content"

Window {
    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 { id : status; width: column.width }

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

## **Jesus loves you.**

```
        }  
    }  
}
```

Place the column types in the order in which they are declared in the column in the type declared in the column. Row types place declared types within Row in the order declared horizontally. And Button type is the name of the user-defined type as defined in Button.qml. The file name becomes the type name.

## 2.3. Event

Qt Quick provides various types of events, such as user input, as shown in the following table.

<TABLE> Events provided by Qt Quick

Type	Description
MouseArea	Process mouse events within a specific QML type
Keys	Additional Properties Provided to Manipulate Key Input
KeyNavigation	Supports key navigation in arrow direction
FocusScope	Keyboard Focus Adjustment
Flickable	Flickable provides a GUI for dragging and dropping items
PinchArea	PinchArea is an invisible item and is commonly used with visible items to provide pinch gesture processing for that item.
MultiPointTouchArea	Handling the activation to operate the multi-touch pointer
Drag	Type for specifying the drag and drop of an item
DropArea	Type for specifying Drag and Drop in a specific area
TextInput	Process user keystrokes
TextEdit	Text Editor to modify text
TouchPoint	Type containing information relating to the coordinates of the touch
PinchEvent	Process events in the Pinch area
WheelEvent	Mouse wheel event
MouseEvent	Mouse Button Event
KeyEvent	key event
DragEvent	Events related to Drag events

# Jesus loves you.

## ● MouseArea

If a mouse event signal is generated on QWidget in a C++-based application, the mouse event can be processed using the props as the associated Slot function is executed. The following is an example source code for processing mouse events in the MouseArea type.

<Example> Ch02 > 03 > 01\_MouseArea

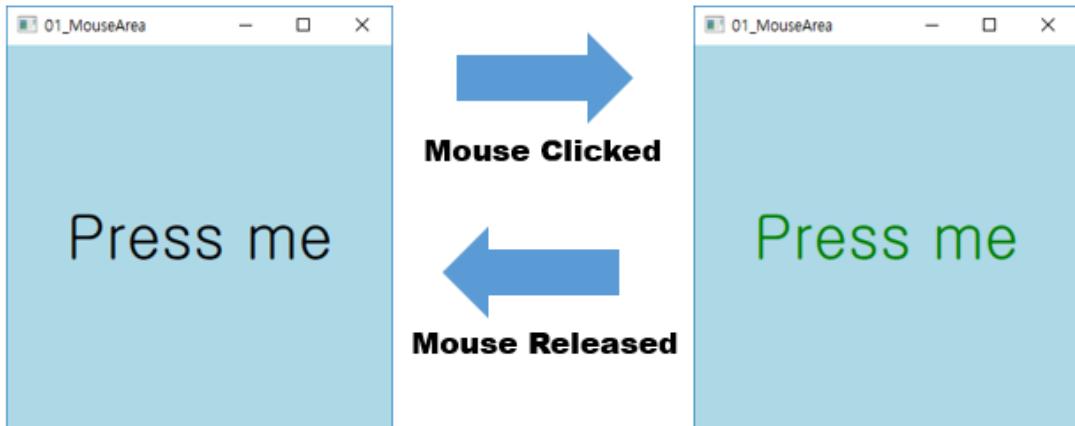
```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    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")
                }
            }
        }
    }
}
```

## Jesus loves you.



<FIGURE> Example Execution Screen

In the example source code above, the anchors.fill sets the area where MouseArea operates. Because Parent is specified, the Text type area is set to MouseArea area here. OnPressed Property occurs when a mouse button is clicked. OnReleased Properties occur when the mouse button is clicked and released.

- **ContainsMouse Properties in MouseArea Area**

ContainsMouse Properties can be used to handle events where the mouse's position enters the Rectangle. To use containsMouse Properties, the hoverEnabled Property must be set to true.

<Example> Ch02 > 03 > 02\_containsMouse

```
import QtQuick 2.12
import QtQuick.Window 2.12

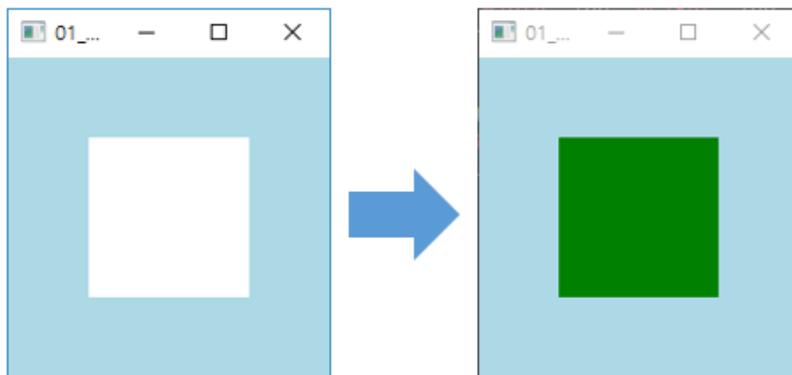
Window {
    visible: true; width: 200; height: 200

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

            MouseArea {
                id: mouseArea
```

## Jesus loves you.

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



<FIGURE> Example Execution Screen

### ● Drag and DropArea

This example is an example of changing the background color of the DropArea area to green when the mouse is drawn inside the Rectangle area and the Rectangle enters the DropArea area.

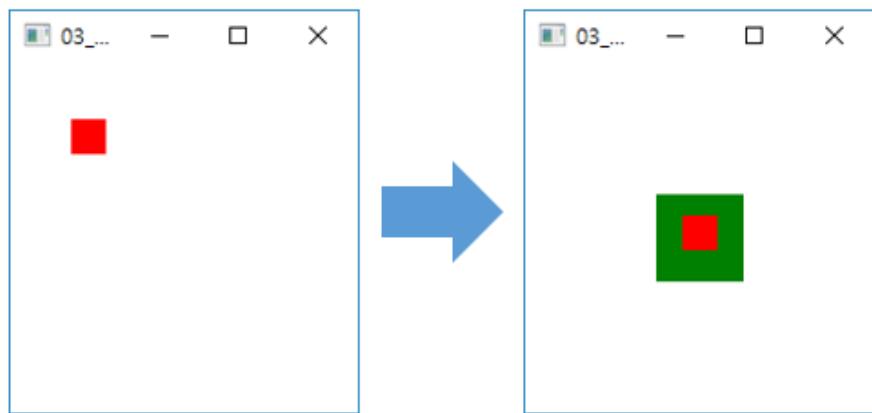
<Example> Ch02 > 03 > 03\_Drag

```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    visible: true; width: 200; height: 200
    DropArea {
        x: 75; y: 75; width: 50; height: 50
        Rectangle {
            anchors.fill: parent
            color: "green"
            visible: parent.containsDrag
        }
    }
    Rectangle {
```

## Jesus loves you.

```
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  
}  
}  
}
```



<FIGURE> Example Execution Screen

### ● Event processing using TextInput type

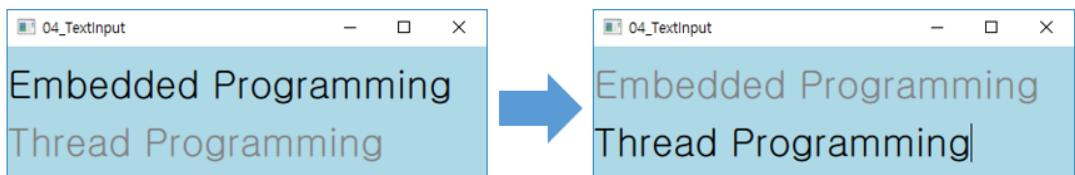
The TextInput type can handle keyboard input events. The following example is an example source code for processing focus events using the TextInput type.

<Example> Ch02 > 03 > 04\_TextInput

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
  
Window {  
    visible: true; width: 400; height: 112
```

## Jesus loves you.

```
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"  
    }  
}  
}
```



<FIGURE> Example Execution Screen

### ● KeyNavigation Type

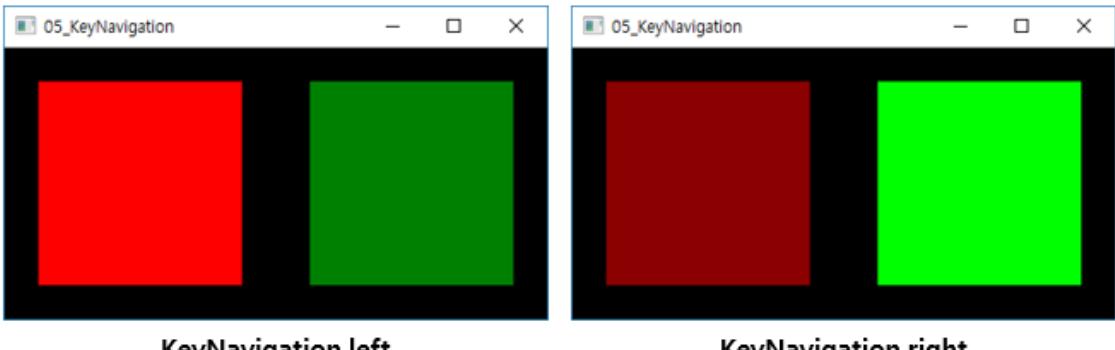
KeyNavigation type can handle up/down/left/right key events for keys. In addition to the orientation, the TAB key event KeyNavigation.tab Property can be used. And KeyNavigation.backtab Properties can be used to handle "Shift + Tab" key events.

<Example> Ch02 > 03 > 05\_KeyNavigation

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
  
Window {  
    visible: true; width: 400; height: 200  
    Rectangle {
```

## Jesus loves you.

```
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
}
}
```



<FIGURE> Example Execution Screen

### ● Keys

Keys events can handle keyboard events. Keys can use onPressed and onReleased Signal Property.

Example: Ch02 > 03 > 06\_Keys

```
import QtQuick 2.12
import QtQuick.Window 2.12

Window
```

## Jesus loves you.

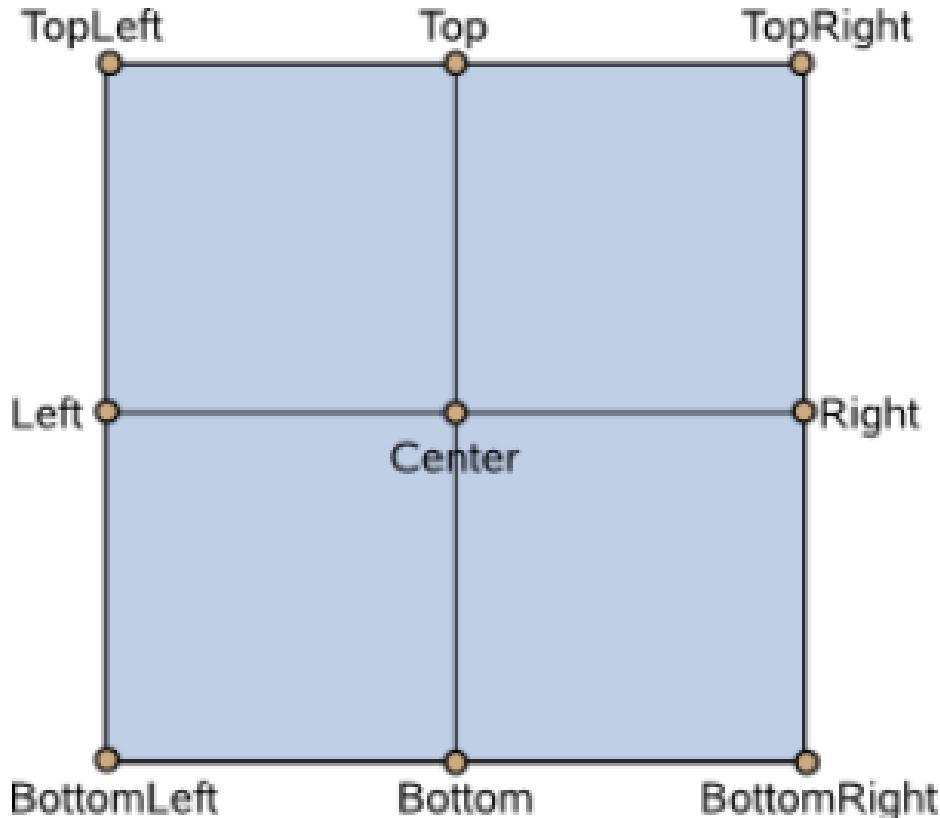
```
{  
    visible: true  
    width: 240; height: 200  
  
    Rectangle {  
        width: 240; height: 200; color: "white"  
  
        Image {  
            id: logo  
            x: 20; y: 20  
            source: "images/qtlogo.png"  
            transformOrigin: Item.Center  
        }  
  
        Keys.onPressed: {  
            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  
    }  
}
```



<FIGURE> Example Execution Screen

In Image type, transformOrigin Properties can select the direction of the center point that rotates the image type image. For example, transformOrigin value Item.The left value causes the turning center point to rotate in the left direction.

**Jesus loves you.**



<FIGURE> Center axis direction that can be set to the transformOrigin value

- **Flickable**

Flickable provides the ability to move types such as Rectangle and Item to Drag and Flicked events on the screen. The following example source code is the Flickable example source code.

<Example> Ch02 > 03 > 07\_Flickable

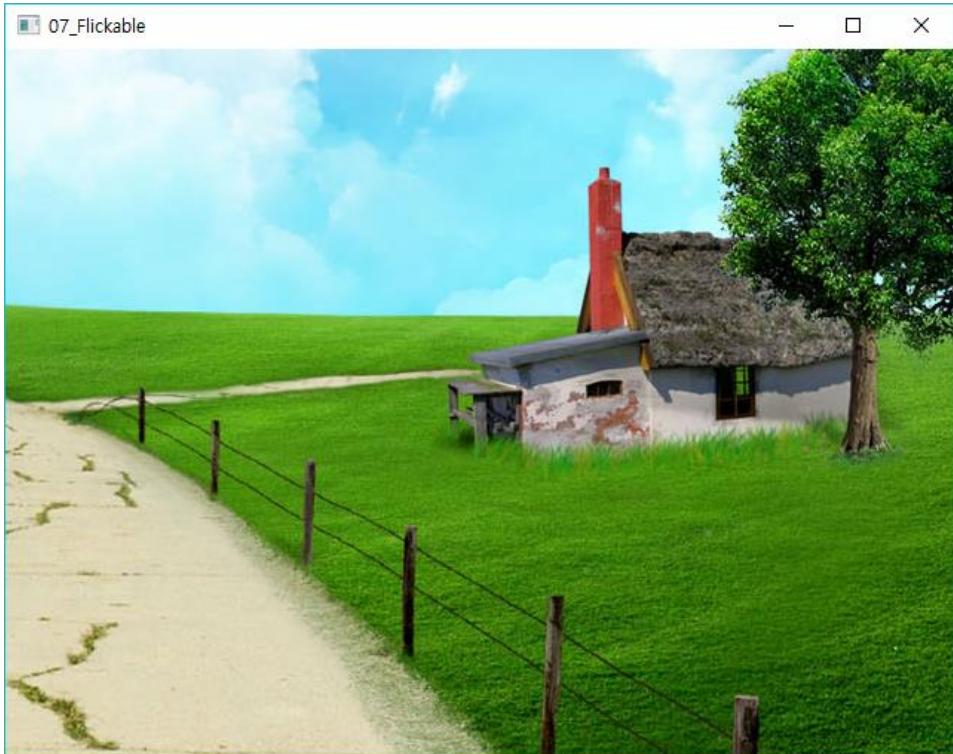
```
import QtQuick 2.12
import QtQuick.Window 2.12

Window
{
    visible: true; width: 640; height: 480
    Rectangle {
        width: 640; height: 480
        Flickable {
```

## **Jesus loves you.**

```
id: view
anchors.fill: parent
contentWidth: picture.width
contentHeight: picture.height
Image {
    id: picture
    source: "images/background.jpg"
}
}
```

The figure below shows an example source code execution screen. Let's try dragging a loaded image on the run screen with the mouse.



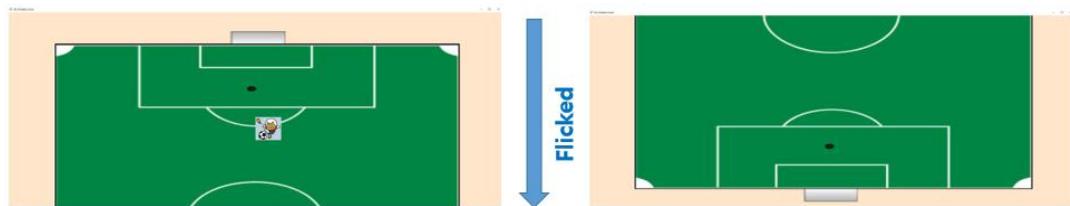
<FIGURE> Flickable Example Execution Screen

- **Flickable Type – Example of processing an image with a Flicked event up and down**

In this example, we will use the Flickable that we discussed earlier to deal with the Flicked event from top to bottom. The image is the image of the soccer field. As shown in the figure

# Jesus loves you.

below, the image is dragged down when a Flicked event occurs.



<FIGURE> Example Execution Screen

<Example> Ch02 > 03 > 08\_Flickable\_Ground

```
import QtQuick 2.12
import QtQuick.Window 2.12

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
        }
    }
}
```

- **Signal and Signal Handler**

## Jesus loves you.

In QML, Signal occurs in type and Signal Handler is connected to Signal to provide the ability to handle events that occur in Signal. For example, as shown in the example source code below, when a signal named clicked occurs, a Signal Handler called onClicker performs an event.

<Example> Ch02 > 03 > 09\_SignalHandler\_exam1

```
import QtQuick 2.12
import QtQuick.Window 2.12

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", mouse.x, mouse.y)
            }
        }
    }
}
```

In the example above, a Signal Handler called onClicked deals with Signal when you click the mouse.

### ● Connections Type

The connections type can be processed within the connections type.

<Example> Ch02 > 03 > 09\_SignalHandler\_exam2

```
import QtQuick 2.12
import QtQuick.Window 2.12

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

## Jesus loves you.

```
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);  
        }  
    }  
}
```

### ● Add Custom Signal

In QML, Syntax for adding user-defined Signals is as follows:

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

When Signal is Emit (evented) in the QML type, the following can be performed to ensure that certain functions associated with Signal are executed:

The following example adds Signal directly to the user. This example has two example source codes. The first is the SquareButton.qml in the directory called content. And it is written as main.qml. Use the user-defined QML type in squareButton.qml in main.qml. First, let's look at the source code for the SquareButton.qml example.

<Example> Ch02 > 03 > 09\_SignalHandler\_exam3

```
import QtQuick 2.12  
  
Rectangle {  
    id: root  
    signal activated(real xPosition, real yPosition)  
    signal deactivated  
  
    MouseArea {
```

## Jesus loves you.

```
anchors.fill: parent
onPressed: { root.activated(mouse.x, mouse.y) }
onReleased: { root.deactivated() }
}
}
```

The following is the main.qml example source code.

```
import QtQuick 2.12
import QtQuick.Window 2.12
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!")
    }
}
```

### ● Signal to Method Connection

The following example source code is an example source code for connecting Signal and Method.

<Example> Ch02 > 03 > 09\_SignalHandler\_exam4

```
import QtQuick 2.12
import QtQuick.Window 2.12

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)
    }
}
```

## **Jesus loves you.**

```
}

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

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

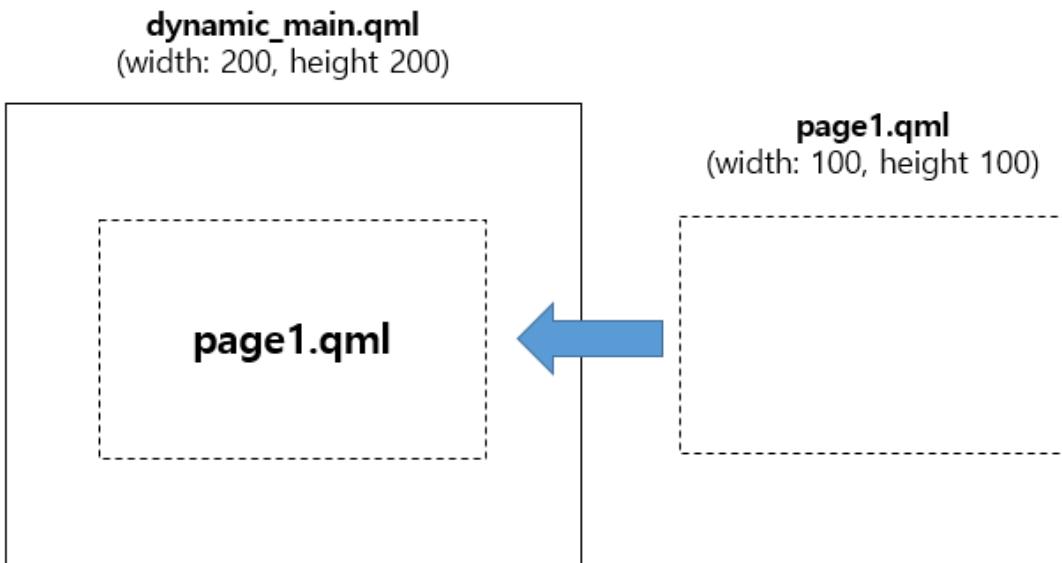
function sendToEmail(person, notice) {
    console.log("Sending to email: " + person + ", " + notice)
}

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

This example source code invokes a method registered in Component.onCompleted when Signal occurs.

## 2.4. Dynamic GUI Configuration Using Loader Type

In this section, let's take a look at the features within QML to dynamically invoke different QML in a specific area. Loader type is provided to load another QML file within QML.



<FIGURE> Loader Usage Example

If you use Loader Type as shown in the figure above, you can dynamically load QML within QML. The following example loads the page1.qml file on the dynamic\_main.qml. First, let's look at the dynamic\_main.qml example source code.

<Example> Ch02 > 04 > 01\_Dynamic\_Basic > dynamic\_main.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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

Rectangle {
```

## **Jesus loves you.**

```
id: myRect
width: 200; height: 100
color: "yellow"
Text {
    anchors.centerIn: parent
    text : "Main QML"; font.bold: true
}

MouseArea {
    anchors.fill: parent
    onClicked:
    {
        pageLoader.source = "page1.qml"
    }
}
}
```

In the example source code above, the page1.qml file can be loaded in the area where the loader type is declared. In Loader type, id is designated as pageLoader.

And when I click on the mouse, id designated the QML file to load using pageLoader in source. Therefore, if you click the mouse within the Rectangle type area of the ID MyRect, the page1.qml file is loaded. The following example source code is page1.qml.

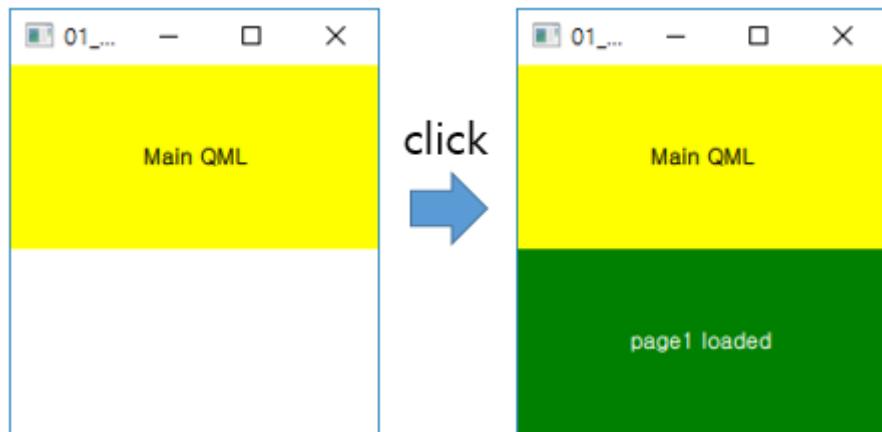
<Example> Ch02 > 04 > 01\_Dynamic\_Basic > page1.qml

```
import QtQuick 2.12

Rectangle {
    width: 200; height: 100; color: "green"

    Text {
        anchors.centerIn: parent
        text: "page1 loaded"
        font.bold: true
        color: "#FFFFFF"
    }
}
```

## Jesus loves you.



<FIGURE> Loader Example Execution Screen

- **Load within Loader Type Area using Component Type**

Component type can specify the type to be loaded in the loader area. The following example source is an example using Component Type.

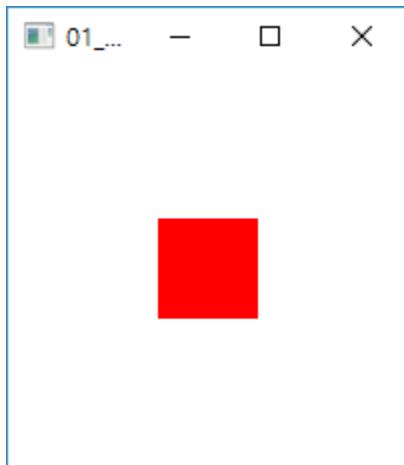
<Example> Ch02 > 04 > 01\_Dynamic\_Basic > dynamic\_component.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    visible: true; width: 200; height: 200
    Loader {
        anchors.centerIn: parent
        sourceComponent: rect
    }

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

## Jesus loves you.



<FIGURE> Example Execution Screen

### ● Events in Loader Type

Connections type can be used to handle events generated by Loader type. The following example shows a signal connected using Connections.

<Example> Ch02 > 04 > 01\_Dynamic\_Basic > application.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

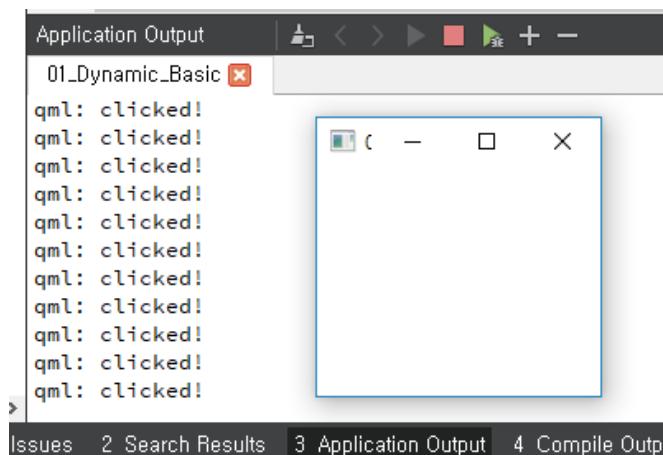
Window {
    visible: true; width: 100; height: 100
    Loader {
        id: myLoader
        source: "myitem.qml"
    }
    Connections {
        target: myLoader.item
        onMessage: console.log(msg)
    }
}
```

Load myitem.qml file from loader type in example source code above. And in connection type, onMessage is a signal registered in myitem.qml. In myitem.qml, the first letter m is changed to M before the message signal for use in the connection. And put the string on before the signal string. The following is the example source code for myitem.qml.

## Jesus loves you.

```
import QtQuick 2.12

Rectangle {
    id: myItem
    signal message(string msg)
    width: 100; height: 100
    MouseArea
    {
        anchors.fill: parent
        onClicked: myItem.message("clicked!")
    }
}
```



<FIGURE> Connection type Example Execution Screen

### ● Key event in Loader area

To handle key events from QML files loaded in Loader type, focus property values must be assigned true. The following example is the key\_main.qml example source code.

<Example> Ch02 > 04 > 01\_Dynamic\_Basic > key\_main.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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

## **Jesus loves you.**

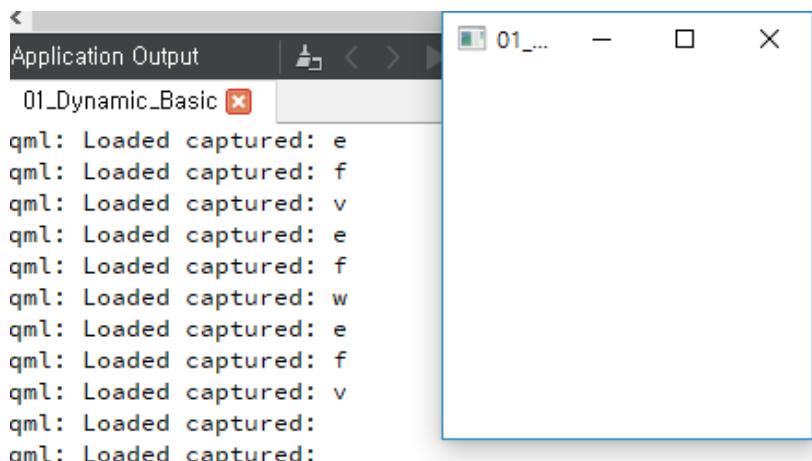
```
    id: loader
    focus: true
    source: "keyreader.qml"
}
}
```

Load the keyreader.qml source file in Loader Type as shown in the example above. The following is an example source code for keyreader.qml.

<Example> Ch02 > 04 > 01\_Dynamic\_Basic > keyreader.qml

```
import QtQuick 2.12

Item {
    Item {
        focus: true
        Keys.onPressed: {
            console.log("Loaded captured:", event.text);
            event.accepted = true;
        }
    }
}
```



<FIGURE> Example Execution Screen

### **● Invoke two or more Loader types**

This example is a call using two Loader types.

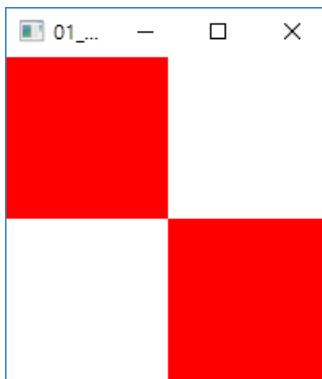
# Jesus loves you.

<Example> Ch02 > 04 > 01\_Dynamic\_Basic > square.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    visible: true; width: 300; height: 300
    Component {
        id: redSquare
        Rectangle { color: "red"; width: 100; height: 100 }
    }

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



<FIGURE> Example Execution Screen

## ● Loader type status property

The status props provided by Loader type provide Loader type status.

<TABLE> Loader type status

Kind	Descripts
Null	The QML source called from the Loader area is inactive or does not exist.
Ready	When the QML source is ready to load
Loading	When the QML source is being modern loaded,

## Jesus loves you.

Error	When an error occurred while loading the QML source.
-------	--

The following is an example source code using status in Loader type.

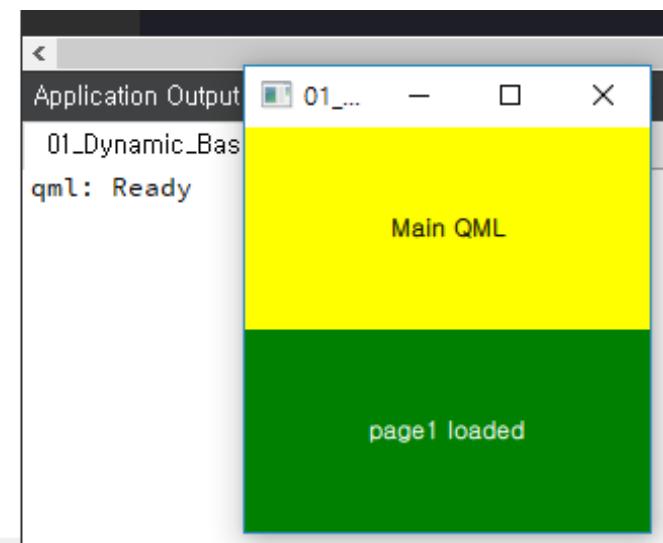
<Example> Ch02 > 04 > 01\_Dynamic\_Basic > status.qml

```
import QtQuick 2.11
import QtQuick.Window 2.12

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

    Loader {
        id: pageLoader
        anchors.top: myRect.bottom
        onStatusChanged: {
            if (pageLoader.status == Loader.Ready)
                console.log('Ready')
        }
    }
    Rectangle {
        id: myRect; width: 200; height: 100; color: "yellow"
        Text {
            anchors.centerIn: parent
            text : "Main QML"; font.bold: true
        }
        MouseArea {
            anchors.fill: parent
            onClicked: pageLoader.source = "page1.qml"
        }
    }
}
```

## Jesus loves you.



<FIGURE> Example Execution Screen

In the figure above, click on the area printed Main QML to load the page1.qml file at the bottom and output the Ready string in the console window.

- **Change the value of a QML file called Loader Type.**

This time, let's look at how to change the value of the declared Property within the QML file called Loader Type. The examples are value\_change.qml and ExComponent.qml. In value\_change.qml, ExComponent.qml is called the Loader type.

<Example> Ch02 > 04 > 01\_Dynamic\_Basic > value\_change.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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" });
    }
}
```

## Jesus loves you.

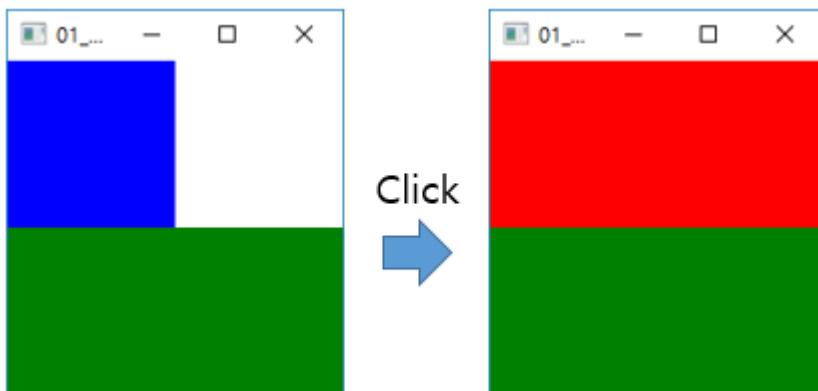
```
Rectangle {  
    anchors.top: squareLoader.bottom  
    width: 200; height: 100  
    color: "green"  
    MouseArea {  
        anchors.fill: parent  
        onClicked: {  
            squareLoader.setSource("ExComponent.qml", {"width": 200})  
        }  
    }  
}
```

Load the ExComponent.qml and the Rectangle at the bottom of the Loader as shown in the example source code above. The following is the ExComponent.qml source code.

<Example> Ch02 > 04 > 01\_Dynamic\_Basic > ExComponent.qml

```
import QtQuick 2.12  
  
Rectangle {  
    id: rect; color: "red"; width: 100; height: 100  
}
```

Load ExComponent.qml from value\_change.qml using Loader. Thus the two Rectangle-type rectangles are loaded as shown in the following figure.



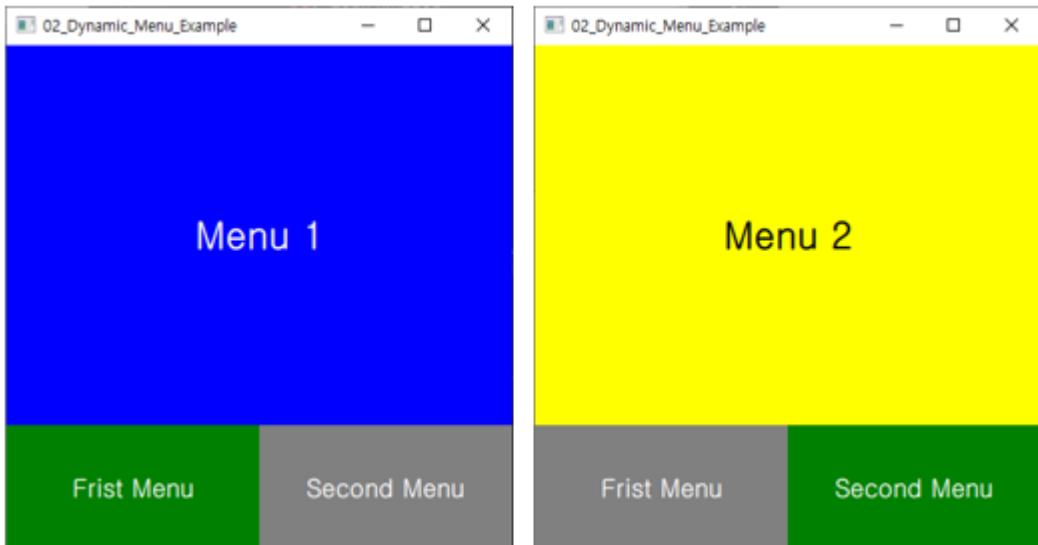
<FIGURE> Example Execution Screen

Click on the Rectangle of the "green" color below, as shown in the figure above, to change the value of the upper Rectangle's width Property from 100 to 200 as seen from the right. The color value is also changed to "red".

# Jesus loves you.

## ● Example of Dynamic QML Loading with Loader Type

This example is an example of changing the QML at the top when you click the first menu at the bottom and the second menu area at the loaded QML. The following is an example run screen.



<FIGURE> Example Execution Screen

Click [second menu] on the bottom right as shown in the figure above to change the top menu to yellow and the bottom right side to green as shown in the figure above. This example consists of three QML files:

<TABLE> Loader 타입의 status 종류

QML	Description
main.qml	Load the menu1.qml and menu2.qml files using loader type.
menu1.qml	"white" colored Rectangle
menu2.qml	"yellow" colored Rectangle

As you can see in the table above, this example consists of three QML files. First, let's look at the source code of the main.qml file.

<Example> Ch02 > 04 > 02\_Dynamic\_Menu\_Example > main.qml

```
import QtQuick 2.0
import QtQuick.Window 2.12
```

## Jesus loves you.

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

    Rectangle {
        id: menu2Button
        anchors.right: parent.right
        anchors.bottom: parent.bottom
        color: "gray"
        width: parent.width/2
        height: 100
        Text {
```

## Jesus loves you.

```
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"; }
    }
]
}
```

The state type at the bottom is State Machine. If Status is behind "menu1", load the file of the Rectangle at the top and change the color to "green". Then, change the QML of the Rectangle on the top of the "menu2" to the menu2.qml file and change the color to "green". The following is the example source code for menu1.qml.

<Example> Ch02 > 04 > 02\_Dynamic\_Menu\_Example > menu1.qml

```
import QtQuick 2.12

Rectangle {
    width: 400; height: 300; color: "blue"
    Text {
        anchors.centerIn: parent
        text: "Menu 1"
```

## **Jesus loves you.**

```
    font.bold: true  
    font.pixelSize: 30  
    color: "white"  
}  
}  
}
```

<Example> Ch02 > 04 > 02\_Dynamic\_Menu\_Example > menu12.qml

```
import QtQuick 2.12  
  
Rectangle {  
    width: 400  
    height: 300  
    color: "yellow"  
  
    Text {  
        anchors.centerIn: parent  
        text: "Menu 2"  
        font.bold: true  
        font.pixelSize: 30  
        color: "black"  
    }  
}
```

## 2.5. Canvas

Canvas provides the ability to draw within the QML area, such as the QPainter class used in the QWidget class. Within the QML area, you can display (Rendering) elements, such as lines, shapes, Gradients, etc., and you can specify the size of the Drawing area using width and height properties.

```
import QtQuick 2.12

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

If QPainter is used in QWidget, onPaint Properties can be used in Canvas, such as update( ) or repaint( ).

```
import QtQuick 2.12
Canvas {
    id: mycanvas; width: 100; height: 200
    onPaint: {
        ...
    }
}
```

- **Draw a simple square within the Canvas type area**

This example is drawing a rectangle in the Canvas area.

<Example> Ch02 > 05 > 01\_Canvas > canvas\_basic.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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

## Jesus loves you.

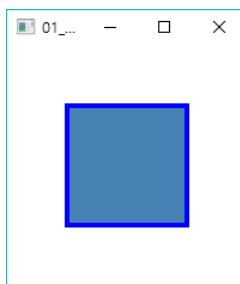
```
Canvas {
    id: root; width: 200; height: 200
    onPaint: {
        var ctx = getContext("2d")

        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()
    }
}
```

In the above source code, onPaint provides the same functionality as the QPainter class used by QWidget. Within onPaint, an Argument named "2d" is specified in the getContext( ) function. In Canvas, only "2d" can be selected. Therefore, it is not possible to specify "3d" in the Canvas area.



<FIGURE> Canvas Example Execution Screen

### ● Rectangular Drawing in Various forms

You can draw various rectangles in the Canvas area, such as the following example source code:

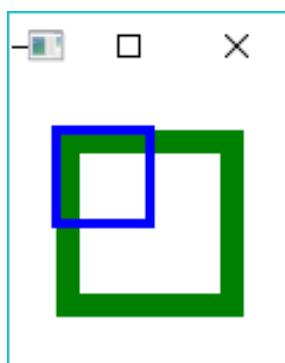
## Jesus loves you.

<Example> Ch02 > 05 > 01\_Canvas > canvas\_rect.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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)
        }
    }
}
```



<FIGURE> Example Execution Screen

### ● Using Gradients in the Canvas Region

<Example> Ch02 > 05 > 01\_Canvas > canvas\_gradients.qml

```
import QtQuick 2.12
```

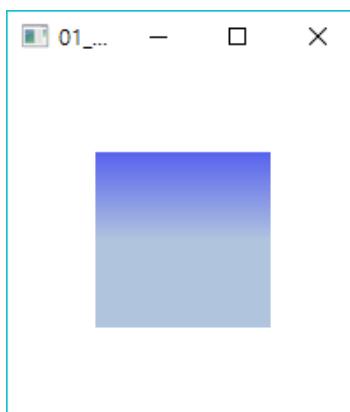
## Jesus loves you.

```
import QtQuick.Window 2.12

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)
        }
    }
}
```



<FIGURE> Gradients Example Execution Screen

- **Drawing with the mouse in the Canvas area**

In this example, let's look at an example of dragging while holding the mouse button down in the Canvas area.

## Jesus loves you.

<Example> Ch02 > 05 > 01\_Canvas > canvas\_mousedrawing.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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, 3, 3)
            }
        }

        MouseArea {

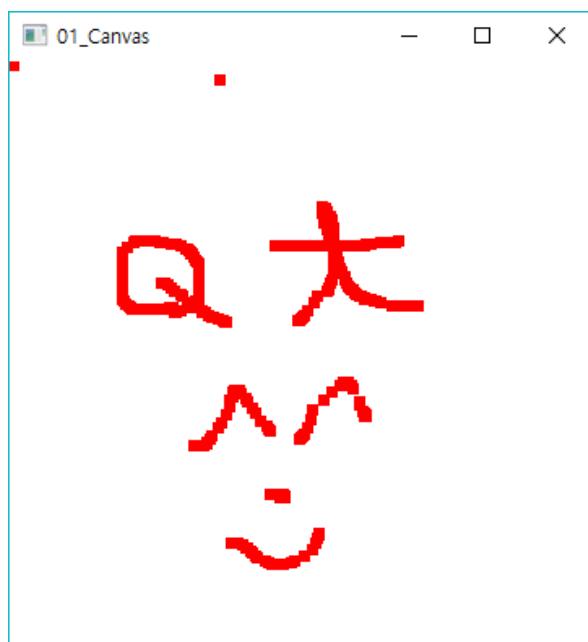
            anchors.fill: parent
            onPressed: {
                myCanvas.xpos = mouseX
                myCanvas.ypos = mouseY
                myCanvas.requestPaint()
            }

            onMouseXChanged: {
                myCanvas.xpos = mouseX
                myCanvas.ypos = mouseY
                myCanvas.requestPaint()
            }

            onMouseYChanged: {
                myCanvas.xpos = mouseX
                myCanvas.ypos = mouseY
            }
        }
    }
}
```

## Jesus loves you.

```
        myCanvas.requestPaint()
    }
}
}
}
```



<FIGURE> Example Execution Screen

## 2.6. Graphics Effects

The effects provided by Qt Quick can use effects such as Blending, Masking, Blurring, Coloring, etc.



mode: normal



mode: addition



mode: average



mode: color



mode: colorBurn



mode: colorDodge

<FIGURE> Blend and Colorize Effect

To use the Graphical Effects provided by Qt Quick in QML, import the following on QML:

```
import QtGraphicalEffects 1.xx
```

Use QtGraphicalEffects and versions as shown above. During "1.xx" above, ".xx" is a subversion. The QtGraphicalEffects module provides a variety of effects as shown in the following table.

<TABLE> QtGraphicalEffects Module provides the type of Effect

Classification	Type	Description
Blend	Blend	Merge two images to process Blend

## Jesus loves you.

Color	BrightnessContrast	contrast with brightness
	ColorOverlay	Changing the color of the source item with Overlay color
	Colorize	Set colors in HSL color space
	Desaturate	chromatic saturation reduction
	GammaAdjust	Adjust the brightness of the original source item
	HueSaturation	Adjust the color of the original item in HSL color space
	LevelAdjust	Adjust the color level in the RGBA color space
Gradient	ConicalGradient	A smooth blend of two or more colors. Colors start at a specified angle and display up to 360 degrees.
	LinearGradient	A smooth blend of two or more colors. Color displays from the specified visibility to the end
	RadialGradient	A smooth blend of two or more colors. Colors are marked from the middle to the edge of the border.
Distortion	Displace	Move pixels of source item according to displacement mapping
DropShadow	DropShadow	Shadow effects on source images
	InnerShadow	fuzzy effect inside source image
Blur	FastBlur	Apply a cloudy effect to one or more source items
	GaussianBlur	Apply higher quality cloudy effect
	MaskedBlur	Use the maskSource to control the intensity of the blur for each pixel so that some parts of the source are more blurred than others.
	RecursiveBlur	Smooth image using recursive feedback loops to blur the source multiple times
Motion Blur	DirectionBlur	Apply Blur to Specific Direction
	RadialBlur	Blur around item center point in circular direction
	ZoomBlur	Apply Blur Based on Center Point
Glow	Glow	Used to produce color and fuzzy effects on the original image and to emphasize dark side areas

## Jesus loves you.

		in the original image
	RectangularGlow	Used to produce color and blur effects on square areas and to emphasize dark face areas
Mask	OpacityMask	Apply the source image as a mask using another item
	ThresholdMask	Apply mask to original item and apply critical area

### ● Blend Effect

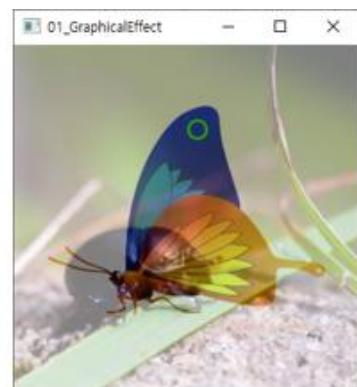
Blend can apply Effect, a mixture of Source images and Foreground Source images. Cache can be used to improve the performance of rendering. To use Cache, you can set the cached property to true.



Source



Foreground Source



Result

<FIGURE> Blend Effect

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect1.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtGraphicalEffects 1.0

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

        Image {
```

## Jesus loves you.

```
id: source; source: "images/source.jpg"
sourceSize: Qt.size(parent.width, parent.height)
smooth: true;
visible: false
}
Image {
    id: butterfly
    source: "images/butterfly.png"
    sourceSize: Qt.size(parent.width, parent.height)
    smooth: true;
    visible: false
}
Blend {
    anchors.fill: source
    source: source
    foregroundSource: butterfly
    mode: "average"
    cached: true
}
}
}
}
```

### ● BrightnessContrast

BrightnessContrast can adjust the Brightness and Contrast of the Source image.



<FIGURE> BrightnessContrast Effect

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect2.qml

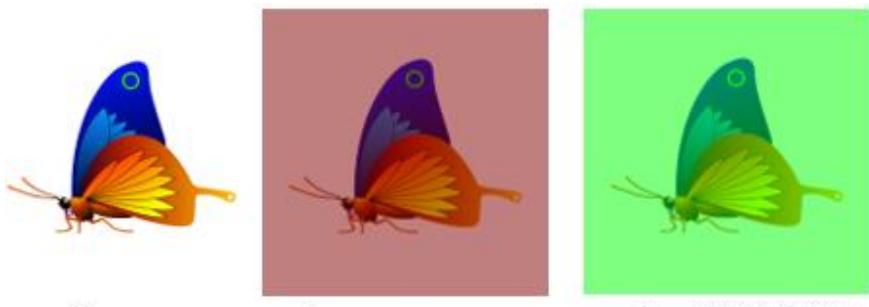
## Jesus loves you.

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtGraphicalEffects 1.0

Window {
    visible: true; width: 300; height: 300
    Item {
        width: parent.width; height: parent.height
        Image {
            id: source
            source: "images/source.jpg"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true
            visible: false
        }
        BrightnessContrast {
            anchors.fill: source
            source: source
            brightness: 0.5; contrast: 0.5
        }
    }
}
```

### ● ColorOverlay

ColorOverlay can set the collide effect on the Source image.



<FIGURE> ColorOverlay Effect

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect3.qml

```
import QtQuick 2.12
```

## Jesus loves you.

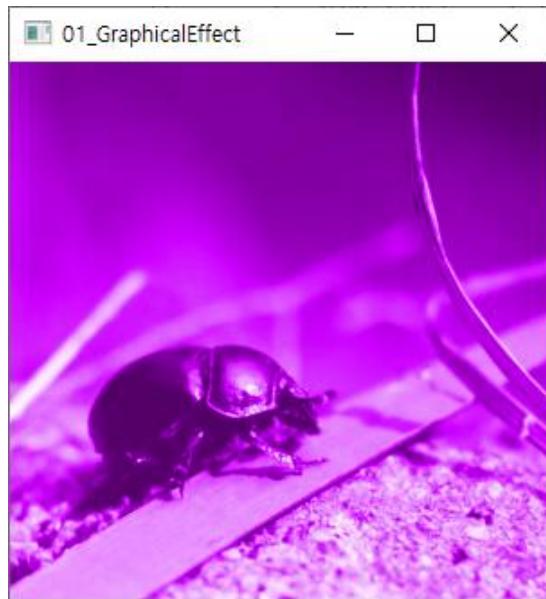
```
import QtQuick.Window 2.12
import QtGraphicalEffects 1.0

Window {
    visible: true; width: 300; height: 300
    Item {
        width: parent.width; height: parent.height
        Image {
            id: lenna
            source: "images/lenna.png"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true
            visible: false
        }
        Image {
            id: butterfly
            source: "images/butterfly.png"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true
            visible: false
        }
        Blend {
            anchors.fill: lenna
            source: lenna
            foregroundSource: butterfly
            mode: "average"
            cached: true
        }
    }
}
```

### ● Colorize

The color can be set in the HSL Color space of the Source image item.

## Jesus loves you.



<FIGURE> Colorize Effect

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect4.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtGraphicalEffects 1.0

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

        Colorize {
            anchors.fill: lenna
            source: lenna
            hue: 0.8
        }
    }
}
```

## Jesus loves you.

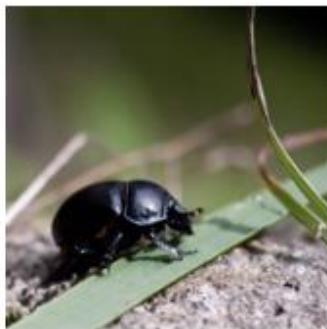
```
    saturation: 1.0  
    lightness: 0.2  
}  
}  
}
```

- **Desaturate**

It provides the ability to reduce saturation of the RGB value of the source image. For example, you can display it in black and white.



Source



desaturation: 0.5



desaturation: 1.0

<FIGURE> Desaturate Effect

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect5.qml

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
import QtGraphicalEffects 1.0  
  
Window {  
    visible: true; width: 300; height: 300  
    Item {  
        width: parent.width  
        height: parent.height  
  
        Image {  
            id: bug  
            source: "images/source.jpg"  
            sourceSize: Qt.size(parent.width, parent.height)  
            smooth: true  
            visible: false  
        }  
    }  
}
```

## Jesus loves you.

```
    }

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

}
```

### ● GammaAdjust

GammaAdjust can apply each pixel according to a predefined curve, such as power-law extension. In other words, effect can be applied using gamma values.



Source

<FIGURE> GammaAdjust Effect

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect6.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtGraphicalEffects 1.0

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

# Jesus loves you.

```
Image {  
    id: src; source: "images/source.jpg"  
    sourceSize: Qt.size(parent.width, parent.height)  
    smooth: true  
    visible: false  
}  
  
GammaAdjust {  
    anchors.fill: src  
    source: src  
    gamma: 2.0  
}  
}  
}
```

## ● HueSaturation

HueSaturation is similar to Collize Effect. The difference between the collize and the hue and the saturation program can be specified.



<FIGURE> HueSaturation Effect

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect7.qml

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
import QtGraphicalEffects 1.0  
  
Window {
```

# Jesus loves you.

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

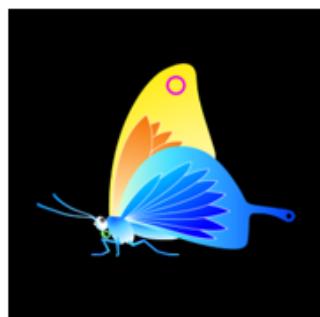
    Image {
        id: src; source: "images/source.jpg"
        sourceSize: Qt.size(parent.width, parent.height)
        smooth: true; visible: false
    }
    HueSaturation {
        anchors.fill: src; source: src
        hue: 0.3
        saturation: 0
        lightness: 0
    }
}
}
```

## ● LevelAdjust

LevelAdjust can apply an effect to separate colors for each color channel in the Source image.



Source



minimumOutput: "#00ffff"  
maximumOutput: "#ff000000"



minimumInput: "#00000070"  
maximumInput: "#ffffff"  
minimumOutput: "#000000"  
maximumOutput: "#ffffff"

<FIGURE> LevelAdjust Example Execution Screen

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect8.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
```

# **Jesus loves you.**

```
import QtGraphicalEffects 1.0

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
        }

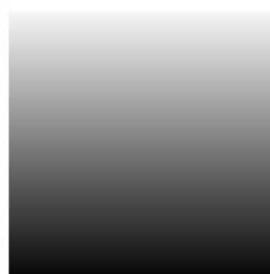
        LevelAdjust {
            anchors.fill: butterfly
            source: butterfly
            minimumInput: "#00000070"
            maximumInput: "#ffffff"
            minimumOutput: "#000000"
            maximumOutput: "#ffffff"
        }
    }
}
```

## ● **Gradient**

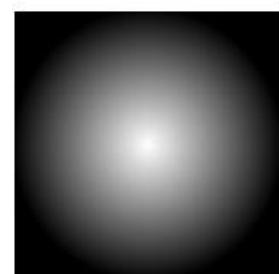
Gradient provides an effect that seamlessly blends and displays two or more colors. The LogicalGradient starts at a specified angle and displays it up to 360 degrees. LinearGradient displays the color from the specified point in time to the ending point. RadialGradient displays from the middle to the border end point.



ConicalGradient



LinearGradient



RadialGradient

# **Jesus loves you.**

<FIGURE> Gradient Effect

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect\_Gradient.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtGraphicalEffects 1.0

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" }
            }
        }
    }
}
```

# Jesus loves you.

```
    }  
}
```

## ● Displace

The Display provides the ability to move pixels of the source image item according to the displacement mapping.



Source

Displacement Source

Result

<FIGURE> Displace Effect

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect9.qml

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
import QtGraphicalEffects 1.0  
  
Window {  
    visible: true; width: 300; height: 300  
    Item {  
        width: parent.width; height: parent.height  
  
        Image {  
            id: src  
            source: "images/source.jpg"  
            sourceSize: Qt.size(parent.width, parent.height)  
            smooth: true; visible: false  
        }  
  
        Rectangle {  
            id: displacement
```

## Jesus loves you.

```
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: src
    source: src
    displacementSource: displacement
    displacement: 0.1
}
}
}
```

### ● DropShadow

DropShadow provides the ability to specify shadow effects on Source images. Radius Properties can specify the degree of smoothness of the shadow. The larger the value of this Property, the more blurred and distorted the Edges portion of the shadow. Color Properties can also be used to specify the color of the shadow.



Source



color: "#000000"  
radius: 8.0  
samples: 16  
horizontalOffset: 0  
verticalOffset: 20  
spread: 0



color: "#0000ff"  
radius: 8.0  
samples: 16  
horizontalOffset: 0  
verticalOffset: 20  
spread: 0



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

<FIGURE> DropShadow Effect

## Jesus loves you.

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect10.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtGraphicalEffects 1.0

Window {
    visible: true
    width: 300
    height: 300

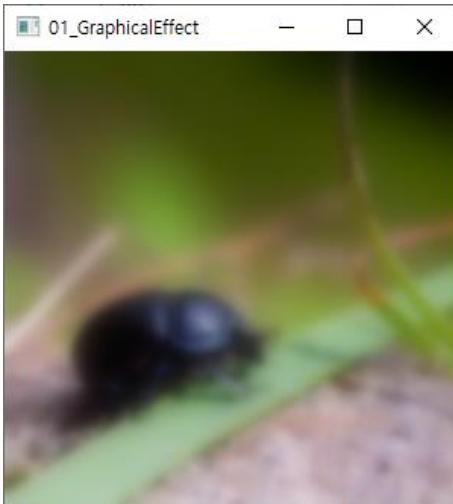
    Item {
        width: parent.width; height: parent.height
        Image {
            width: 300; height: 300
            id: butterfly
            source: "images/ball.png"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true
            visible: false
        }

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

### ● FastBlur

FastBlur can apply a blur on the source image. Provides a lower Blur Effect than GaussianBlur but faster rendering speed than GaussianBlur.

## Jesus loves you.



<FIGURE> FastBlur Effect

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect11.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtGraphicalEffects 1.0

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

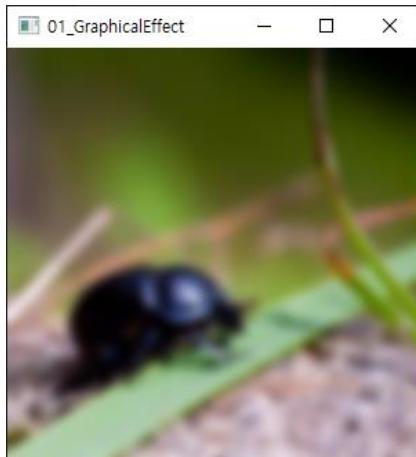
        Image {
            id: src; source: "images/source.jpg"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true
            visible: false
        }

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

# Jesus loves you.

## ● GaussianBlur

It guarantees higher quality than FastBlur but less rendering speed than FastBlur.



<FIGURE> GaussianBlur Effect

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect12.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtGraphicalEffects 1.0

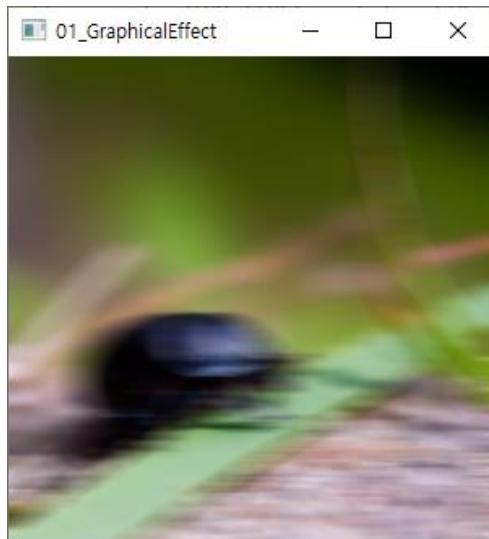
Window {
    visible: true; width: 300; height: 300
    Item {
        width: parent.width; height: parent.height
        Image {
            id: src
            source: "images/source.jpg"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true
            visible: false
        }
        GaussianBlur {
            anchors.fill: src
            source: src
            deviation: 4
            radius: 8
            samples: 16
        }
    }
}
```

## **Jesus loves you.**

```
    }  
}  
}
```

- **DirectionBlur**

DirectionBlur can apply Blur Effect in certain directions.



<FIGURE> DirectionBlur Effect

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect13.qml

## **Jesus loves you.**

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtGraphicalEffects 1.0

Window {
    visible: true; width: 300; height: 300
    Item {
        width: parent.width; height: parent.height
        Image {
            id: src; source: "images/source.jpg"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true; visible: false
        }
        DirectionalBlur {
            anchors.fill: src; source: src
            angle: 90
            length: 32
            samples: 24
        }
    }
}
```

### ● RadialBlur

Blur Effect, such as the one rotated to the center of the Source image, can be applied.



<FIGURE> RadialBlur Effect

## **Jesus loves you.**

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect14.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtGraphicalEffects 1.0

Window {
    visible: true
    width: 300
    height: 300

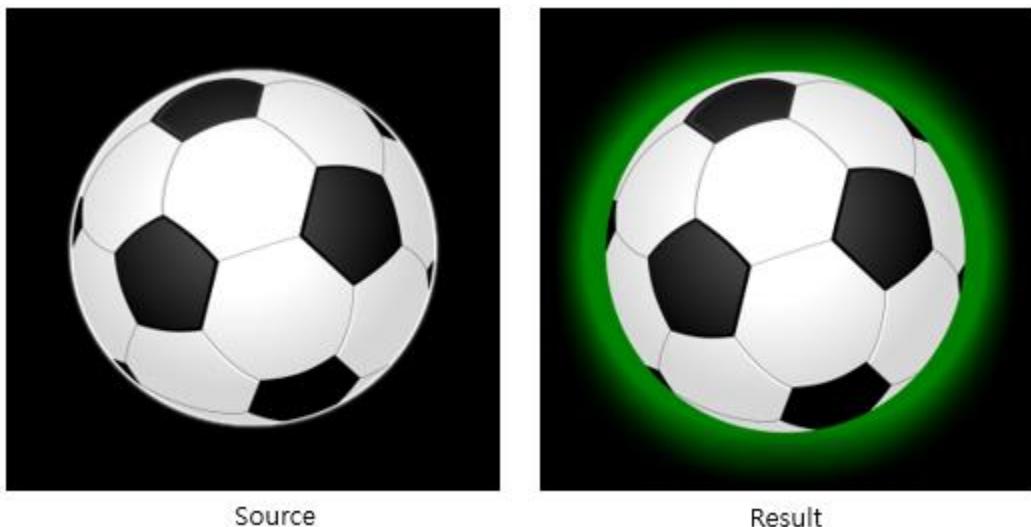
    Item {
        width: parent.width
        height: parent.height

        Image {
            id: src; source: "images/source.jpg"
            sourceSize: Qt.size(parent.width, parent.height)
            smooth: true; visible: false
        }
        RadialBlur {
            anchors.fill: src
            source: src
            samples: 24
            angle: 30
        }
    }
}
```

### ● **Glow**

Used to produce color and blurred effects on the Source image and to emphasize dark cotton areas.

## Jesus loves you.



<FIGURE> Glow Effect

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect15.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtGraphicalEffects 1.0

Window {
    visible: true; width: 300; height: 300
    Rectangle {
        anchors.fill: parent
        color: "black"
    }
    Image {
        id: butterfly
        source: "images/ball.png"
        sourceSize: Qt.size(parent.width, parent.height)
        smooth: true
        visible: false
    }
    Glow {
        anchors.fill: butterfly
        radius: 30
        samples: 16
        color: "green"
        source: butterfly
    }
}
```

## **Jesus loves you.**

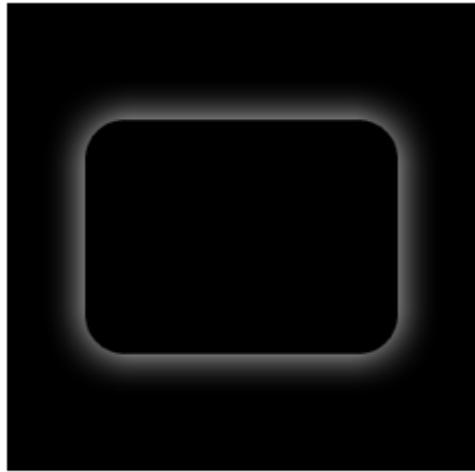
```
    }  
}
```

### ● **RectangularGlow**

Color the Rectangle area and create a blur. Then, it provides glow effect in the form of locating Rectangle items.



Source



Result

<FIGURE> RectangularGlow

<Example> Ch02 > 06 > 01\_GraphicalEffect > effect16.qml

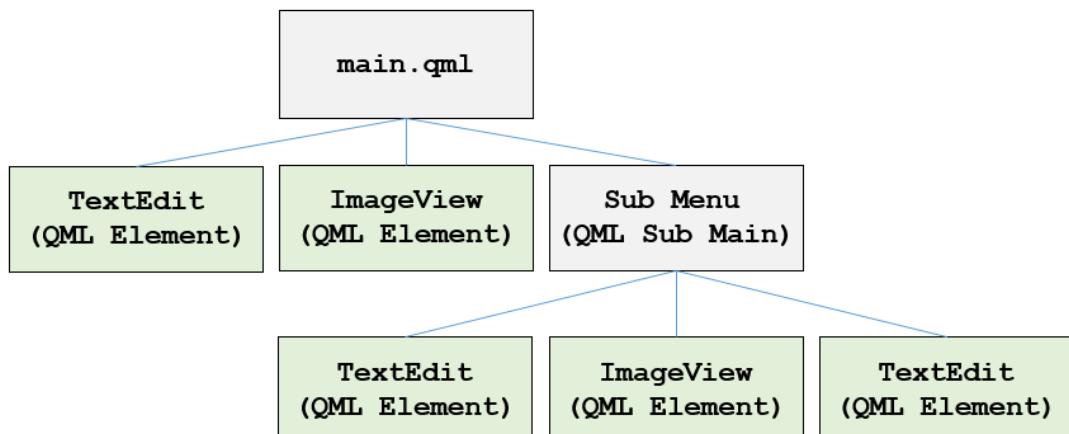
```
import QtQuick 2.12  
import QtQuick.Window 2.12  
import QtGraphicalEffects 1.0  
  
Window {  
    visible: true; width: 300; height: 300  
    Rectangle {  
        id: background; anchors.fill: parent; color: "black"  
    }  
    RectangularGlow {  
        id: effect  
        anchors.fill: rect  
        glowRadius: 10  
        spread: 0.2  
        color: "white"  
    }  
}
```

## **Jesus loves you.**

```
    cornerRadius: rect.radius + glowRadius
}
Rectangle {
    id: rect
    color: "black"
    anchors.centerIn: parent
    width: Math.round(parent.width / 1.5)
    height: Math.round(parent.height / 2)
    radius: 25
}
}
```

## 2.7. QML Module Programming

A library of frequently used classes in C++. This provides modularity to re-use frequently used user-defined types in QML like libraries.



<FIGURE> QML Module Structure

- **Custom type**

A user-defined type can be created in a separate QML file and imported from another QML. For example, suppose you have the main.qml and LineEdit.qml QML source code files. For calling and using LineEdit.qml QML from main.qml, the filename "LineEdit" is used as the name of the module that can be called, except for the file extension. The following example calls the LineEdit.qml file from the main.qml.

<Example> Ch02 > 07 > 01\_Module\_Exam > main.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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

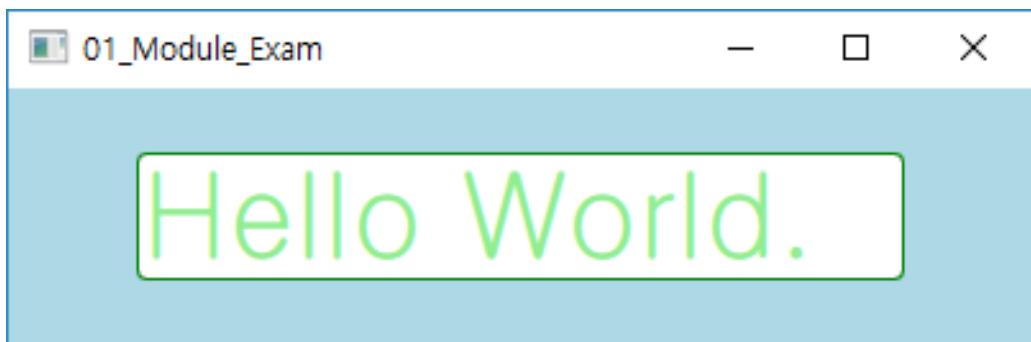
    LineEdit
{
```

## Jesus loves you.

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

<Example> Ch02 > 07 > 01\_Module\_Exam > LineEdit.qml

```
import QtQuick 2.12  
  
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  
    }  
}
```



<FIGURE> Example Execution Screen

### ● Custom Property

In addition to those used in the basic QML type, you can define those used directly within the QML type.

## Jesus loves you.

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

The following are examples of user-defined Property use.

```
property string message: "qt-dev.com"
property int num: 123
property real numfloat: 123.456
property bool boolcondi: true
property url address: "http://www.qt-dev.com"
```

The following example sources are examples of using user-defined properties.

<Example> Ch02 > 07 > 01\_Module\_Exam > new\_main.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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

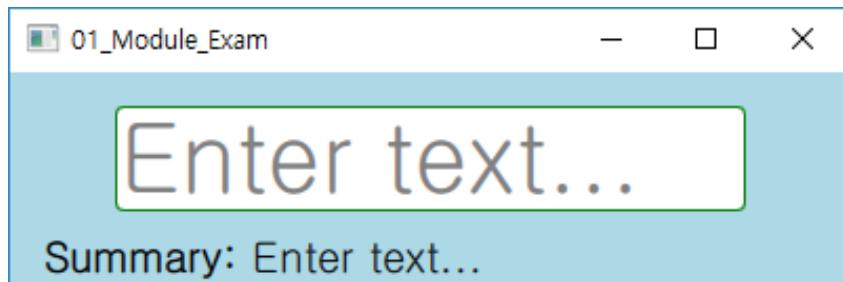
    NewLineEdit {
        id: lineEdit
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.top: parent.top
        anchors.topMargin: 16
        width: 300; height: 50
    }
    Text {
        anchors.top: lineEdit.bottom
        anchors.topMargin: 12
        anchors.left: parent.left
        anchors.leftMargin: 16
        font.pixelSize: 20
        text: "<b>Summary:</b> " + lineEdit.text
    }
}
```

<Example> Ch02 > 07 > 01\_Module\_Exam > NewLineEdit.qml

```
import QtQuick 2.12
```

## Jesus loves you.

```
Rectangle {  
    border.color: "green"; color: "white"  
    radius: 4; smooth: true  
  
    TextInput {  
        id: TextInput; anchors.fill: parent  
        anchors.margins: 2  
        text: "Enter text..."  
        color: focus ? "black" : "gray"  
        font.pixelSize: parent.height - 4  
    }  
    property string text: TextInput.text  
}
```



<FIGURE> Example Execution Screen

### ● Custom Signal

The following example is Syntax structure for signal use.

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

- ✓ Example for defining a user-defined

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

## Jesus loves you.

```
        parent.myChecked(true);
    }
}

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

- ✓ Slot Example Connected to Signal

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

Where you define Signal as shown in the example above, you can define the signal name and the Arguments to deliver in the signal. If there is no Argument, you only need to specify the signal name. In Slot, the first alphabet before the signal name is then changed to uppercase. And you have to put "on" before the signal name.

Signal : `myChecked(bool checkValue)`



Slot : `onMyChecked`

<FIGURE> Signal and slot

Now let's use a user-defined signal to implement CheckBox. The QML file is an example of using the checked\_main.qml file by invoking NewCheckBox.qml at the bottom of the content directory.

<Example> Ch02 > 07 > 01\_Module\_Exam > checked\_main.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
import "content"

Window {
    id: root
    visible: true; width: 250; height: 100
    color: "lightblue"

    NewCheckBox {
        anchors.horizontalCenter: parent.horizontalCenter
```

## Jesus loves you.

```
anchors.verticalCenter: parent.verticalCenter

onMyChecked: checkValue ? root.color = "red"
              : root.color = "lightblue"
}

}
```

<Example> Ch02 > 07 > 01\_Module\_Exam > content > NewCheckBox.qml

```
import QtQuick 2.12

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"
        }
    ]
}
```

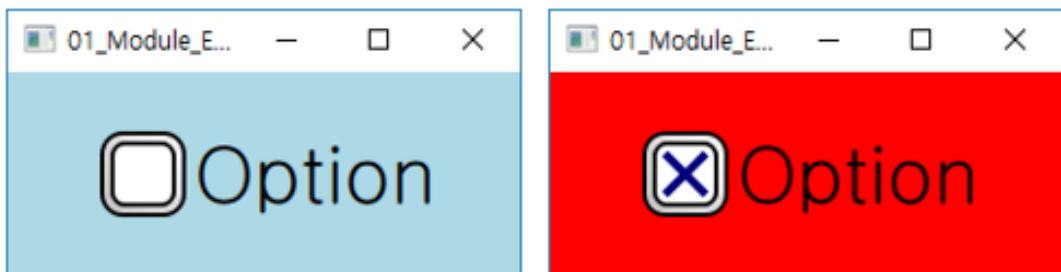
## Jesus loves you.

```
PropertyChanges {
    target: checkImage;
    source: "../images/unchecked.svg"
}
]

state: "unchecked"

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

signal myChecked(bool checkValue)
}
```



<FIGURE> 사용자 정의 Signal 예제 실행 화면

### ● Alias

Alias provides an alias function to replace the module name used in QML. The following example is an example of using Alias.

<Example> Ch02 > 07 > 01\_Module\_Exam > alias.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
```

## **Jesus loves you.**

```
import "content" as MyContent

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

The official QML module provided by Qt Quick can also use Alias as shown in the following example.

<Example> Ch02 > 07 > 01\_Module\_Exam > myqt.qml

```
import QtQuick 2.12 as MyQt
import QtQuick.Window 2.12

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. How to use JavaScript in QML

Qt Quick provides the following methods for using JavaScript within QML:

- ✓ Property Binding
- ✓ Call JavaScript Method Using Signal Handler
- ✓ Use JavaScript function in QML type
- ✓ Import JavaScript files to use

QML has Syntax, such as JSON, XML, or HTML. That's why it doesn't provide structured Syntax such as JavaScript, C++. QML, for example, does not provide Syntax such as if statement, function, variable substitution, etc.

Therefore, for structured programming, JavaScript can be used in QML. You can also use files written in JavaScript within QML. In this section, let's look at the four methods mentioned above.

### ● Property Binding

Property Binding can assign a value to a Property called color using the three-paragraph operator grammar as shown in the following example.

```
import QtQuick 2.12
import QtQuick.Window 2.12

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

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

## Jesus loves you.

The following example QML source code is an example of using the JavaScript functions. You can invoke the JavaScript function using the binding provided by QML.

```
import QtQuick 2.12
import QtQuick.Window 2.12

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"
        } );
    }
}
```

### ● Call JavaScript Method Using Signal Handler

MouseArea was used to handle events when mouse button was clicked in QML. Called a predefined Slot, such as onClicked , provided by MousArea, and used as follows.

```
...
Rectangle {
    id: relay

    signal messageReceived(string person, string notice)

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

## Jesus loves you.

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

...
...
```

### ● QML 파일에서 JavaScript 함수 사용

To use the JavaScript function in the QML type, you can invoke the JavaScript function as shown in the example below.

```
import QtQuick 2.12
import QtQuick.Window 2.12

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()
```

## **Jesus loves you.**

```
{  
    return colorButton.width / 2;  
}  
}
```

The following example is an example source code with a factor in the JavaScript function.

```
...  
Item {  
    width: 200; height: 200  
  
    MouseArea {  
        anchors.fill: parent  
        onClicked: label.moveTo(mouse.x, mouse.y)  
    }  
  
    Text {  
        id: label  
        function moveTo(newX, newY) {  
            label.x = newX;  
            label.y = newY;  
        }  
  
        text: "Move me!"  
    }  
}
```

The following example calls the JavaScript function from MouseArea and returns the function results.

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

# Jesus loves you.

```
    onClicked: console.log(factorial(10))
}
}
```

## ● JavaScript Import

QML allows you to invoke JavaScript files (files with `.js` extension) that have a JavaScript function defined.

```
import QtQuick 2.12
import QtQuick.Window 2.12
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)
    }
}
```

When a mouse button click event occurs, as seen in the example source code above, you can invoke a function defined in the JavaScript file called `script.js` as shown below.

```
// script.js

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

Like libraries, JavaScript can also be defined and used as Library using Pragma keywords. The following example uses the Pragma keyword.

## **Jesus loves you.**

```
import QtQuick 2.12
import QtQuick.Window 2.12
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
            + " is: " + MyLib.factorial(input)
    }
}
```

The following example source code is the script.js source code.

```
.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()
{
```

## Jesus loves you.

```
    return factorialCount;  
}
```

Qt for invoking JavaScript files in QML .include( ) can be used as shown in the example below.

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
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))  
        }  
    }  
}
```

```
// 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)
```

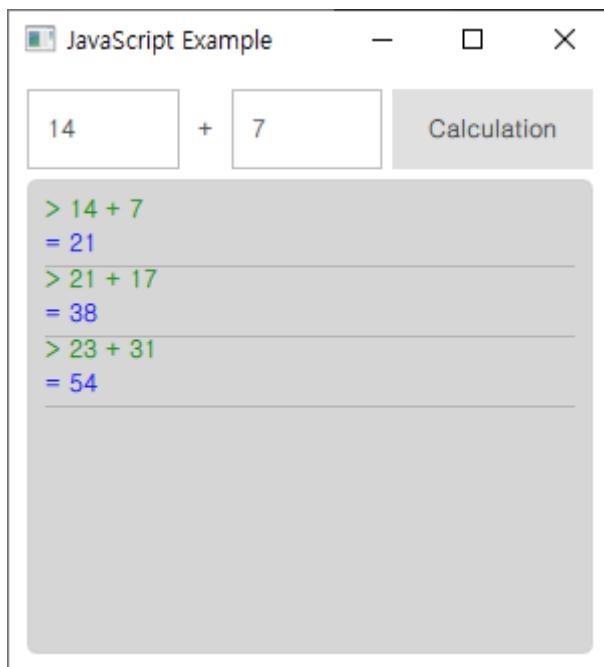
## Jesus loves you.

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

Qt for invoking the example JavaScript file above, facrotyl.js from script.js. You can invoke JavaScript files using the include( ) function.

- **Example of using JavaScript in QML**

This example is for invoking and using JavaScript functions in QML.



<FIGURE> JavaScript Example Execution Screen

If you enter "," between two values and two values, as shown in the figure above, the results of the sum of the two values are printed as shown at the bottom of the example run screen. The following is the QML example source code.

<Example> Ch02 > 08 > 01\_JavaScript\_Example > main.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtQuick.Controls 2.5
import QtQuick.Layouts 1.3
import "js/jslib.js" as JUtils
```

## Jesus loves you.

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

    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: " 계산 "
                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)
            }
        }
    }
}
```

## Jesus loves you.

```
}

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
            }
            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);
}
```

## Jesus loves you.

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

The following example is the JavaScript source code.

<Example> Ch02 > 08 > 01\_JavaScript\_Example > js > jslib.js

```
.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
    }

    data.expression = src1 + ' + ' + src2;
    data.result = result;

    return data;
}
```

## 2.9. Dialog

Qt Quick offers a variety of dialogue, including simple dialogue, color dialogue, file dialogue and font dialogue. To use the dialogue in QtQuick, import the Dialogs as follows.

```
import QtQuick.Dialogs x.x
```

### ● Dialog Type

Qt Quick provides Dialog type to use dialogue in QML. The following example is using Dialog.

<Example> Ch02 > 09 > 01\_Dialog > main.qml

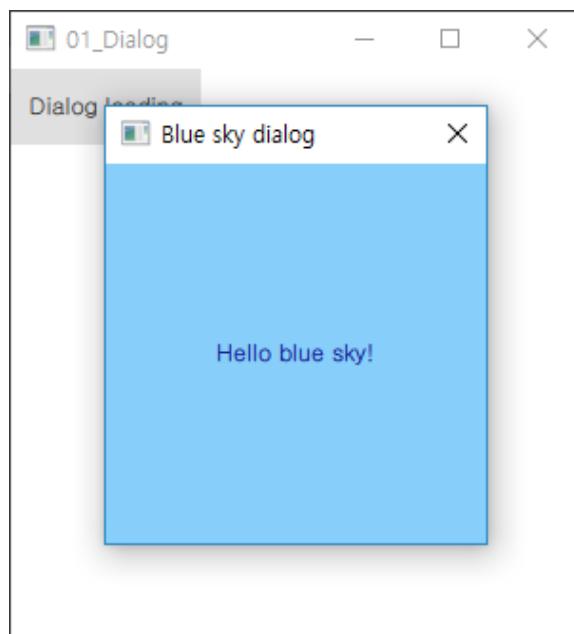
```
import QtQuick 2.12
import QtQuick.Controls 2.5
import QtQuick.Dialogs 1.2

ApplicationWindow {
    width: 300; height: 300; visible: true
    Button {
        text: "Dialog loading"
        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
            }
        }
    }
}
```

# Jesus loves you.

```
}
```



<FIGURE> Example Execution Screen

## ● ColorDialog

The ColorDialog type provides a dialogue for users to select a color.

<Example> Ch02 > 09 > 01\_Dialog > colordialog.qml

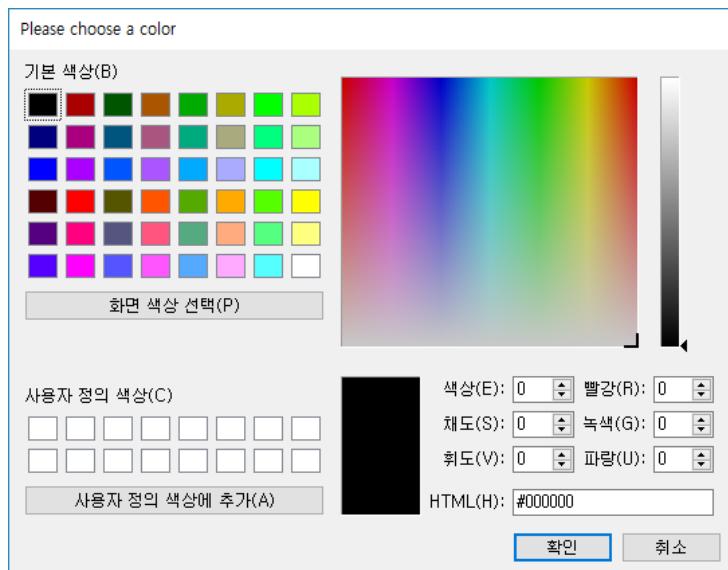
```
import QtQuick 2.12
import QtQuick.Controls 2.5
import QtQuick.Dialogs 1.2

ApplicationWindow {
    width: 300; height: 300; visible: true
    Button {
        text: "ColorDialog loading";
        onClicked: { colorDialog.visible = true }
    }

    ColorDialog {
        id: colorDialog
        title: "Please choose a color"
        onAccepted: {
```

# Jesus loves you.

```
        console.log("You chose: " + colorDialog.color)
        Qt.quit()
    }
    onRejected: {
        console.log("Canceled")
        Qt.quit()
    }
    Component.onCompleted: visible = false
}
}
```



<FIGURE> ColorDialog 예제 실행 화면

## ● FileDialog

The FileDialog type provides users with a dialogue function to select files.

<Example> Ch02 > 09 > 01\_Dialog > filedialog.qml

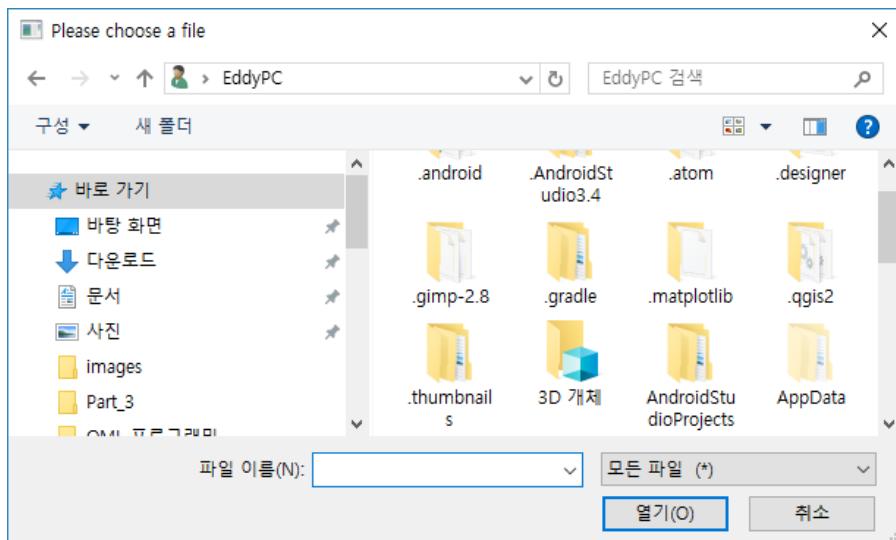
```
import QtQuick 2.12
import QtQuick.Controls 2.5
import QtQuick.Dialogs 1.2

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

## Jesus loves you.

```
Button {
    text: "FileDialog loading"
    onClicked: { fileDialog.visible = true }
}

FileDialog {
    id: fileDialog
    title: "Please choose a file"
    folder: shortcuts.home
    onAccepted: {
        console.log("You chose: " + fileDialog.fileUrls)
        Qt.quit()
    }
    onRejected: {
        console.log("Canceled")
        Qt.quit()
    }
    Component.onCompleted: visible = false
}
}
```



<FIGURE> FileDialog 예제 실행 화면

### ● FontDialog

The FontDialog type provides users with a dialogue function to select fonts.

## Jesus loves you.

<Example> Ch02 > 09 > 01\_Dialog > fontdialog.qml

```
import QtQuick 2.12
import QtQuick.Controls 2.5
import QtQuick.Dialogs 1.2

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

    Button {
        text: "FontDialog loading";
        onClicked: {
            fontDialog.visible = true
        }
    }

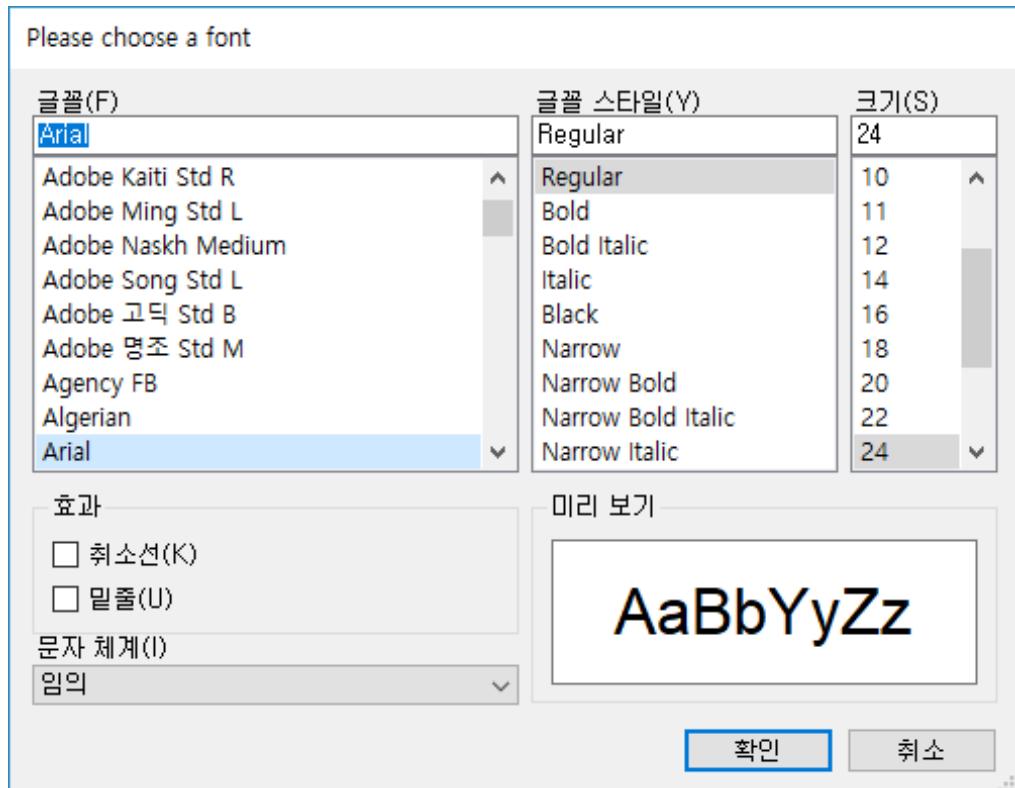
    FontDialog {
        id: fontDialog
        title: "Please choose a font"
        font: Qt.font({ family: "Arial", pointSize: 24, weight: Font.Normal })

        onAccepted: {
            console.log("You chose: " + fontDialog.font)
            Qt.quit()
        }

        onRejected: {
            console.log("Canceled")
            Qt.quit()
        }

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

## Jesus loves you.



<FIGURE> FontDialog Example Execution Screen

### ● **MessageDialog Type**

MessageDialog is used to communicate information to users.

<Example> Ch02 > 09 > 01\_Dialog > messagedialog.qml

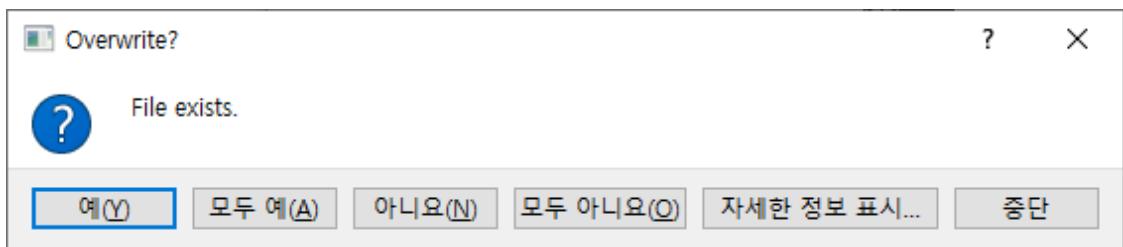
```
import QtQuick 2.12
import QtQuick.Controls 2.5
import QtQuick.Dialogs 1.2

ApplicationWindow {
    width: 300; height: 300; visible: true
    Button { text: "MessageDialog loading";
              onClicked: { messageDialog.visible = true } }
}

MessageDialog {
    title: "Overwrite?"
    icon: StandardIcon.Question
```

## Jesus loves you.

```
text: "File exists."
detailedText: "Do you want to overwrite the existing file? "
standardButtons: StandardButton.Yes | StandardButton.YesToAll |
    StandardButton.No | StandardButton.NoToAll | StandardButton.Abort
Component.onCompleted: visible = true
onYes: console.log("copied")
onNo: console.log("didn't copy")
onRejected: console.log("aborted")
}
}
```



<FIGURE> MessageBox Example Execution Screen

The `messageDialog` type icon Property provides the following icons.



StandardIcon.Question



StandardIcon.Information



StandardIcon.Warning



StandardIcon.Critical

<FIGURE> ICONS

## 2.10. Layout

To use the layout, import it to the QML file as follows:

```
import QtQuick.Layouts X.X
```

Qt Quick provides the following types of layouts:

<TABLE> Layouts

Layout	Description
RowLayout	Place Type Horizontal on One Row
ColumnLayout	Place Type in Vertical Orientation
GridLayout	Place Type as Cell
Layout	GridLayout, RowLayout, or ColumnLayout.

- RowLayout

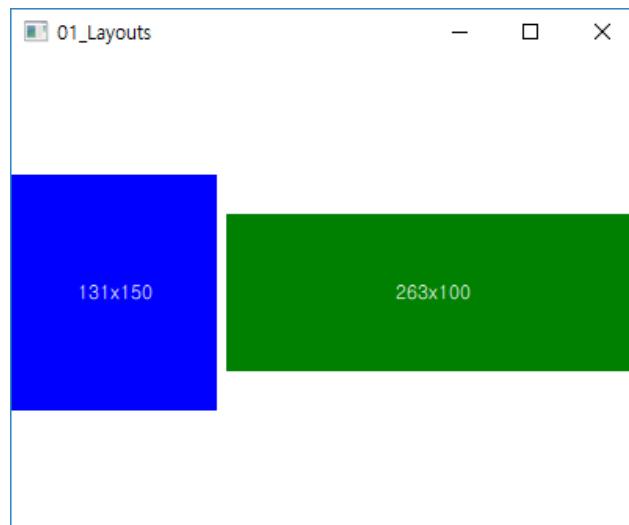
<Example> Ch02 > 10 > 01\_Layouts > rowlayout.qml

```
import QtQuick 2.12
import QtQuick.Controls 2.5
import QtQuick.Layouts 1.2

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
        }
    }
}
```

## Jesus loves you.

```
Text {  
    anchors.centerIn: parent; color: "white"  
    text: parent.width + 'x' + parent.height  
}  
}  
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  
    }  
}  
}  
}  
}
```



<FIGURE> Example Execution Screen

- ColumnLayout

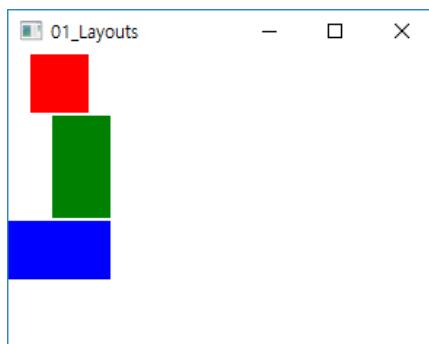
<Example> Ch02 > 10 > 01\_Layouts > columnlayout.qml

```
import QtQuick 2.12  
import QtQuick.Controls 2.5
```

## Jesus loves you.

```
import QtQuick.Layouts 1.2

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
        }
    }
}
```



<FIGURE> Example Execution Screen

- **GridLayout**

## Jesus loves you.

<Example> Ch02 > 10 > 01\_Layouts > gridlayout.qml

```
import QtQuick 2.12
import QtQuick.Controls 2.5
import QtQuick.Layouts 1.2

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

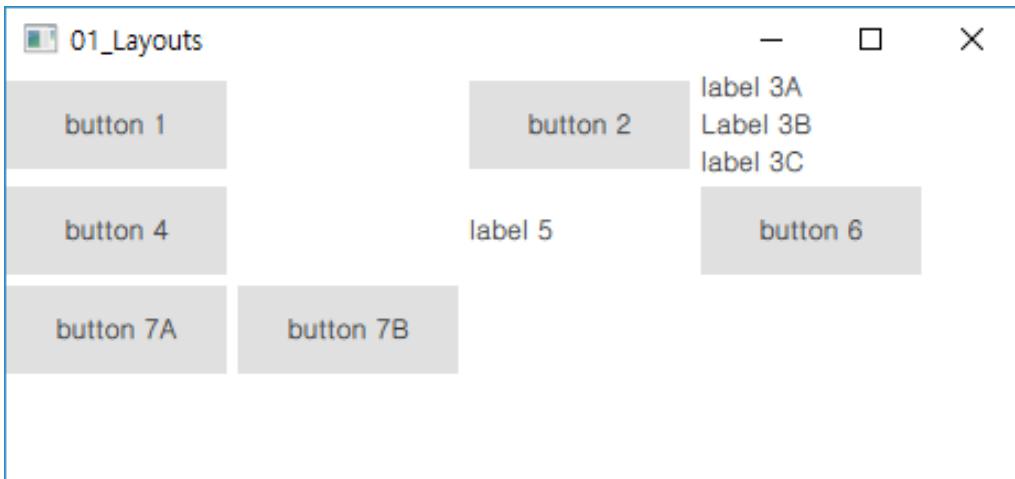
    GridLayout
    {
        columns: 3
        Button { text: "button 1"}
        Button { text: "button 2"}

        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"}
        }
    }
}
```

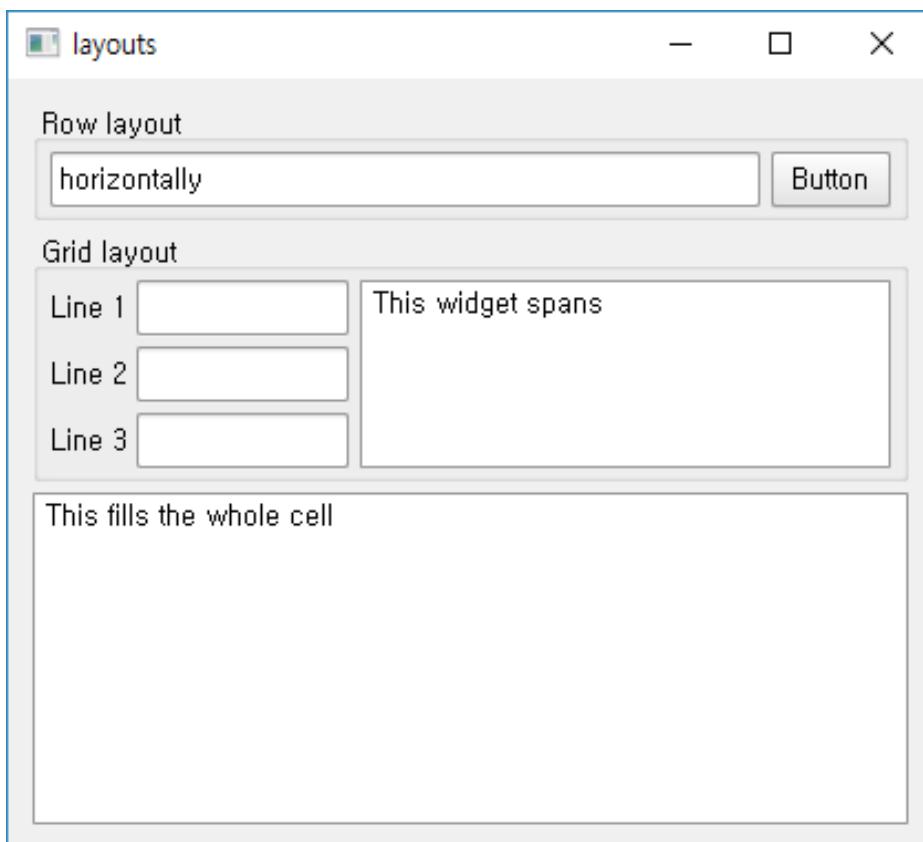
# Jesus loves you.



<FIGURE> GridLayout Example Execution Screen

## ● Layout example

In this example, let's use the Layout provided by Qt Quick to implement a widget that places the type as shown in the figure below.



<FIGURE> Example Execution Screen

## Jesus loves you.

In this example, we used ColumnLayout, RowLayout, GridLayout, and Layout as a nested layout structure. The following is an example source code.

<Example> Ch02 > 10 > 02\_Layout\_Example

```
import QtQuick 2.12
import QtQuick.Controls 2.5
import QtQuick.Layouts 1.2

ApplicationWindow
{
    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
            Layout.fillWidth: true
        }
    }
}
```

## Jesus loves you.

```
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
        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

Let's take a look at how to obtain information about a particular type of location among the different types. For example, suppose there are more than 100 Rectangle types in QML. If more than 100 types of location information meet certain conditions, the QML provides a positioner. Therefore, in this section, let's look at how to use the Positioner to determine a specific type of location information.

### ● Positioner

Positioner items can be used to know specific types of location information in child types such as Column, Row, and Grid. It can be determined by the index property which type is the first one and isFirstIt can be determined by the index property. If the first type isFirstItem Property returns true or false. The following is an example of using a Positioner.

<Example> Ch02 > 11 > 01\_Type\_Position

```
import QtQuick 2.12
import QtQuick.Window 2.12

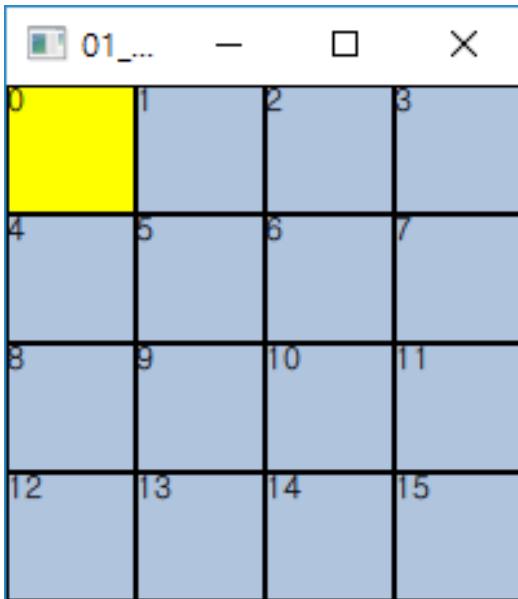
Window {
    visible: true;
    width: 200
    height: 200
    Grid {
        Repeater {
            model: 16
            Rectangle {
                id: rect
                width: 50; height: 50
                border.width: 1
                color: Positioner.isFirstItem ? "yellow" : "lightsteelblue"
                Text {
                    text: rect.Positioner.index
                }
            }
        }
    }
}
```

## Jesus loves you.

```
}
```

```
}
```

```
}
```



<FIGURE> Example Execution Screen

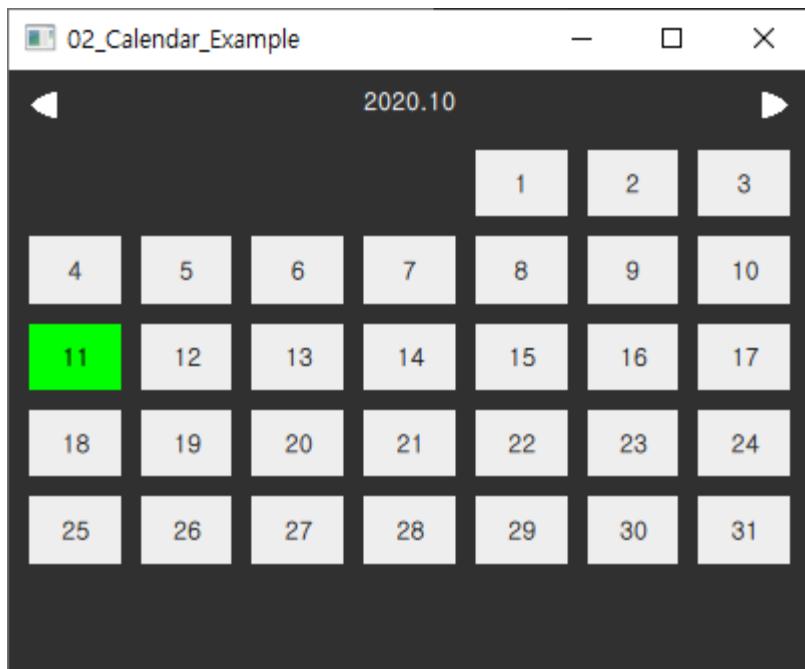
### ● Calendar

In this example, let's use Grid and Repeater to implement the calendar. To implement a calendar, you will use a class provided by JavaScript for calculating dates.

```
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 returns the number of days of the current month. FirstDay prints the number of days of the week for the year, month, and day. The Getday( ) function returns one of the num values of the day (0-6). For example, return two side on Tuesday. The following is the execution screen of an example in which the calendar function has been completed.

## Jesus loves you.



<FIGURE> Calendar Example Execution Screen

<Example> Ch02 > 11 > 01\_Type\_Position

```
import QtQuick 2.12
import QtQuick.Window 2.12

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
```

## Jesus loves you.

```
height: childrenRect.height

Image {
    source: "./images/left.png"
    anchors.left: parent.left
    anchors.leftMargin: 10
    MouseArea {
        anchors.fill: parent
        onClicked: showDate =
            new Date(showDate.getFullYear(), showDate.getMonth(), 0)
    }
}
Text {
    color: "white"
    text: Qt.formatDateTime(showDate, "yyyy.MM")
    font.bold: true
    anchors.horizontalCenter: parent.horizontalCenter
}

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

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

function isToday(index) {
    if (today.getFullYear() != showDate.getFullYear())
        return false;
    if (today.getMonth() != showDate.getMonth())
        return false;

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

## **Jesus loves you.**

```
}

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

        Text {
```

## Jesus loves you.

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

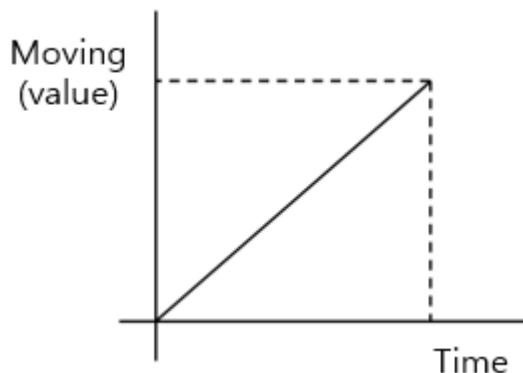
### 3. Animation Framework

To use animations such as soft screen transitions when switching screens, Qt Quick provides an Animation Framework. Animated elements are available on the Windows screen, and types placed on the GUI can also use each animation element.

Animation elements such as transparency, movement, zoom, etc. can be used as animation elements. For example, if an application requires a window with a screen width and a vertical size of 600 x 400 pixels to run, it can be used as an animation element to increase the window size from 100 x 100 to 600 x 400 over the specified time period.

At the same time, transparency allows animation effects to be used to change the transparency from 0.0 to 1.0 over a specified period of time. 0.0 is completely transparent and 1.0 is completely transparent.

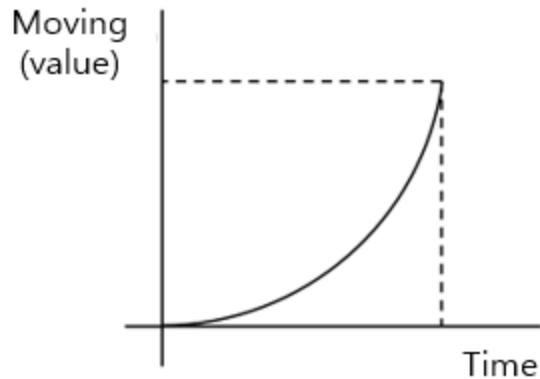
In addition, Easing Curves can be used for the time value of the animation. For example, suppose you specified 1.0 seconds for the x and y positions of the GUI buttons to move and move from positions of 100, to coordinates of 200 and 200. It will then travel at a constant speed to move from the x and y starting coordinates to the destination coordinates.



<FIGURE> moving and time value graph

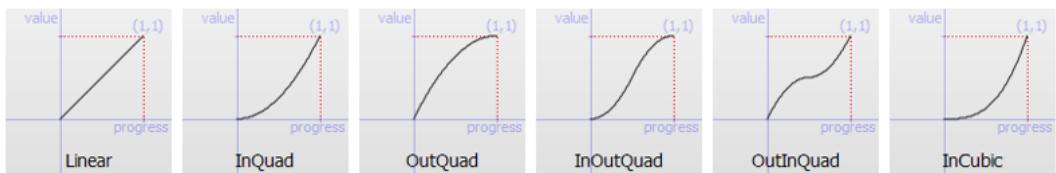
As shown in the figure above, the y-axis of movement (value) is a graph showing that the x- and y-position coordinates of the GUI buttons take a constant 1.0 second to move from 100, 100 to 200 and 200 values. In other words, the time increases linearly with the moving value of the movement. However, using Easing Curves for time, you can change the moving value to the moving value as shown in the following figure.

**Jesus loves you.**



<FIGURE> Graphs with Easing Curve

The values of time moving to the final x and y coordinates can be applied as shown in the figure above. In addition, there are approximately 46 different Easing Curves available for Easing Curve.



<FIGURE> Various Easing Curve

And Qt Quick can use a technique called State Transition for the Animation element. State Machine can take an example of an ON/OFF switch. Not only can ON/OFF be used to change one value, but also a number of options can be used together. For example, if you want the fluorescent light on the front door of the house to be turned on together with the fluorescent light in the living room, you can use a technique called State Transition to turn it on or off together. In other words, in some situations, a state transition technique may be used, in which different values are applied simultaneously. In this chapter, let's take a look at what we have described so far.

## 3.1. Animation

Qt Quick can use Animation in the same type as Rectangle, Item, Image, etc. And Qt Quick provides the following animation types.

<TABLE> Animations

Type	Description
NumberAnimation	Animation using numbers, such as transparent values such as coordinate X, Y, or Opacity.
PropertyAnimation	Application of Animation to Properties such as RGB, Width, and Height
RotationAnimation	Apply Rotation Animation
PauseAnimation	Function to stop Animation
SmoothedAnimation	Function to use gravity acceleration weights
SpringAnimation	Offered for use with animations, such as springs
Behavior	Animation that occurs when a certain value changes, e.g., width, and animation when a certain value changes.
ScriptAction	provided for the purpose of executing a specific method or script when an animation is executed
PropertyAction	Provided to change the value of Element's Properties during Animation operation

Qt Quick은 Animation 을 그룹화 할 수 있는 종류로 다음과 같은 타입을 제공한다.

<TABLE> Animation 종류

타입	설명
SequentialAnimation	Runs multiple animations sequentially
ParrellelAnimation	Running multiple animations in parallel

## Jesus loves you.

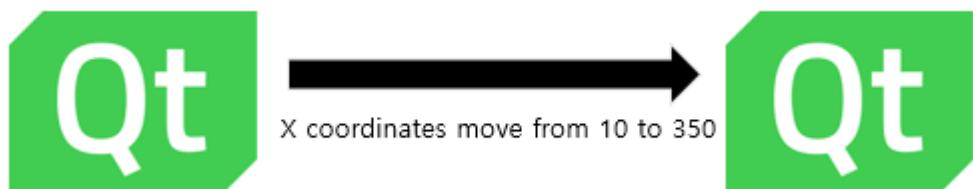
- **NumberAnimation**

NumberAnimation can be used to apply animation using numbers. For example, the image type specified by the X and Y coordinates changes the X coordinate from 10 to 250 values for 2,000 milliseconds. If you do not specify a value for the duration proportion, the default specifies a value of 250 milliseconds.

<Example> Ch03 > 01 > 01\_Animation\_Example > numberanimation.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    width: 630; height: 430; visible: true
    Image {
        source: "images/qtlogo.png"
        x: 10; y: 20;
        NumberAnimation on x {
            from: 10; to: 350
            duration: 2000
        }
    }
}
```



<FIGURE> Moving image

- **PropertyAnimation**

PropertyAnimation can be applied to type Properties. The following example is an example of using the image-type width and heightproperty values as the values in the animation.

<Example> Ch03 > 01 > 01\_Animation\_Example > propertyanimation.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
```

## Jesus loves you.

```
Window {  
    width: 230; height: 230; visible: true  
    Image {  
        id: proAni  
        source: "images/qtlogo.png"  
        x: 50; y: 40; width: 50; height: 50;  
    }  
    PropertyAnimation {  
        target: proAni  
        properties: "width, height"  
        from: 0; to: 100; duration: 1000  
        running: true  
    }  
}
```



<FIGURE> PropertyAnimation Example Execution Screen

### ● RotationAnimation

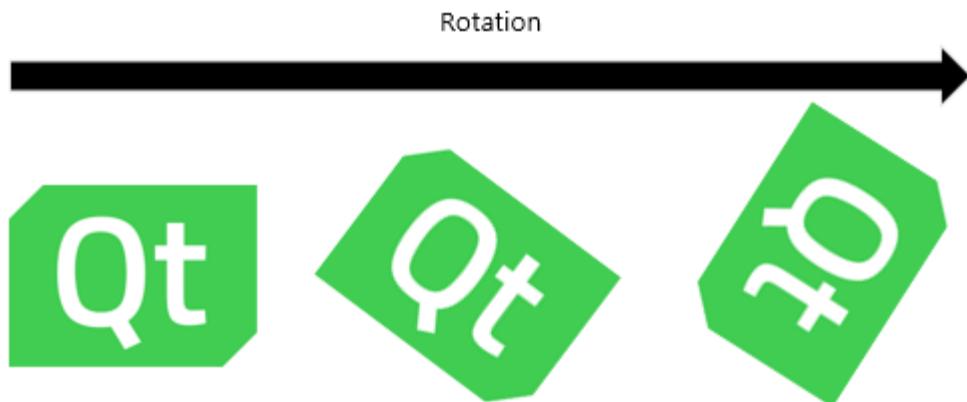
The RotationAnimation type can apply the start and last values to animation. The starting and ending values are the values of the angle. Therefore, from Property is the starting angle and to Property is the last angle.

<Example> Ch03 > 01 > 01\_Animation\_Example > rotationanimation.qml

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
  
Window {
```

## Jesus loves you.

```
width: 330; height: 330; visible: true
Image {
    id: rotAni
    source: "images/qtlogo.png"
    anchors.centerIn: parent
    smooth: true
    RotationAnimation on rotation {
        from: 45; to: 315
        direction: RotationAnimation.Shortest
        duration: 3000
    }
}
```



<FIGURE> RotationAnimation Example Execution Screen

In the example above, the direction Properties of RotationAnimation can be specified whether to rotate clockwise or counterclockwise. The following constants can be used as the directional property value:

<TABLE> direction property constants

Constant	설명
RotationAnimation.Numerical	Using the Linear interpolation algorithm. clockwise direction
RotationAnimation.Clockwise	Rotate clockwise
RotationAnimation.Counter-clockwise	a counterclockwise rotation

## Jesus loves you.

RotationAnimation.Shortest	STEP turns 20 degrees. Rotation direction is half-clockwise.
----------------------------	--

### ● Animation event

An event can occur when Animation is started, when Animation is stopped, or when the Property value of Animation is changed.

<Example> Ch03 > 01 > 01\_Animation\_Example > ani\_event\_complex.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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

            onStarted : console.log("started")
            onStopped : console.log("stopped")
        }
        onScaleChanged: {
            if(scale > 0.5) {
                scaleAni.complete()
            }
            console.log("scale : " + scale);
        }
    }
}
```

### ● Behavior

Behavior is used for the purpose of operating when the value of the type of propulsion is changed.

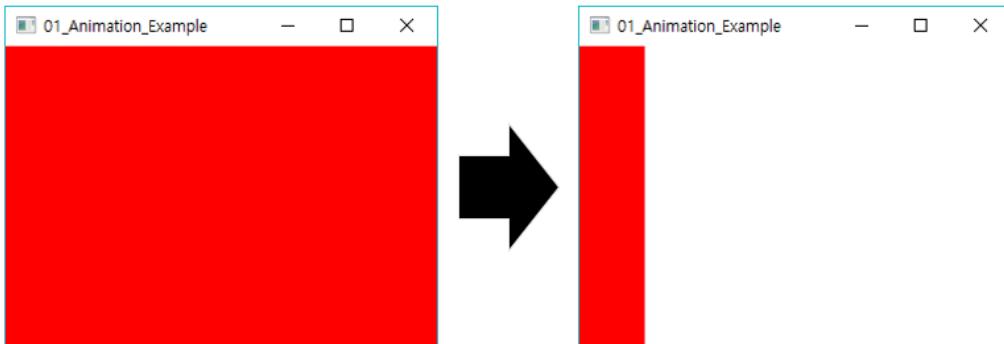
## Jesus loves you.

<Example> Ch03 > 01 > 01\_Animation\_Example > behaivor.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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
        }
    }
}
```

In the example above, if the mouse click to change the Rectangle type width value to 50, the width value changes from 330 to 50, and Animation can be applied.



<FIGURE> Behaivor Example Execution Screen

- SmoothedAnimation

The SmoothedAnimation type provides the ability to process animation using the value of the Velocity Property. The following example shows Rectangle moving according to the directional keys of the keyboard.

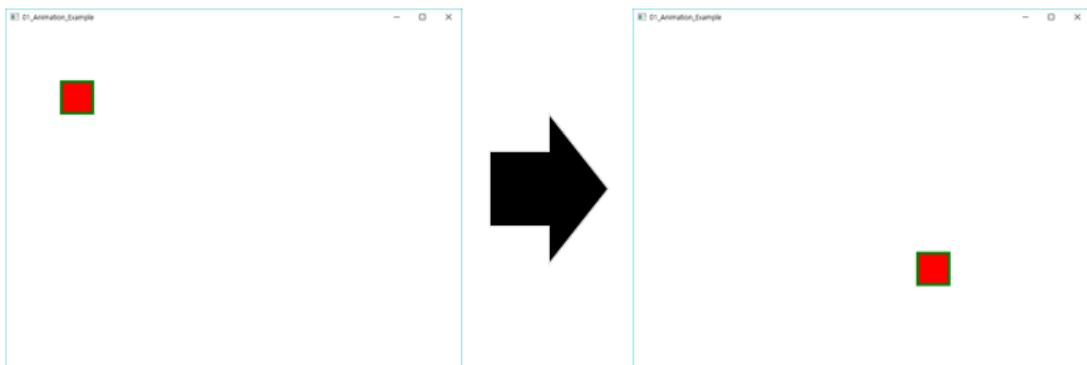
<Example> Ch03 > 01 > 01\_Animation\_Example > smoothedanimation.qml

```
import QtQuick 2.12
```

# Jesus loves you.

```
import QtQuick.Window 2.12

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
    }
}
```



<FIGURE> SmoothedAnimation Example Execution Screen

# Jesus loves you.

- SpringAnimation

The SpringAnimation type can use an animation such as a spring. Spring Properties can reflect the effect of spring elasticity whether or not the spring is elastic. For example, if you stretch and release a spring, you can apply a propulsion effect, such as shrinking quickly. Therefore, the spring property value is large and the elasticity or cohesiveness is high.

Damping props can be used for effects such as braking. The larger the value, the faster the brakes. The value can be set between 0.0 and 1.0.

<Example> Ch03 > 01 > 01\_Animation\_Example > springanimation.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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 = mouse.x - rect.width/2
            rect.y = mouse.y - rect.height/2
        }
    }
}
```

- SequentialAnimation

SequentialAnimation Type provides the function for running multiple animations in order.

<Example> Ch03 > 01 > 01\_Animation\_Example > sequentialanimation.qml

```
import QtQuick 2.12
```

## Jesus loves you.

```
import QtQuick.Window 2.12

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 provides the ability to perform all animations simultaneously. For example, in the previous SequentialAnimation, Animation is run in order, and ParallelAnimation is run simultaneously (parallel).

<Example> Ch03 > 01 > 01\_Animation\_Example > parallelanimation.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

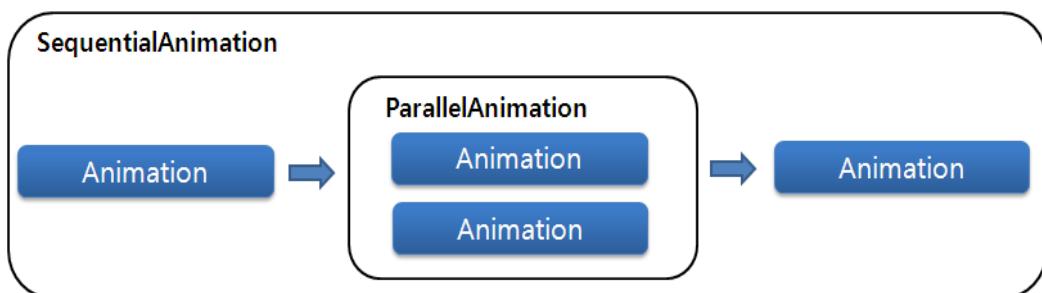
Window {
    width: 300; height: 200; visible: true
    Image {
        id: aniPara
        anchors.centerIn: parent
        source: "images/qtlogo.png"
    }
}
```

# Jesus loves you.

```
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  
}  
}
```

- **Nested examples of SequentialAnimation and ParallelAnimation**

SequentialAnimation and ParallelAnimation can be performed with overlaid structures. For example, an overlaid structure may be performed as shown in the figure below.



<FIGURE> SequentialAnimation and ParallelAnimation are overlaid structures

The following examples illustrate the use of SequentialAnimation and ParallelAnimation in an overlaid structure as shown in the figure above.

<Example> Ch03 > 01 > 01\_Animation\_Example > sequential-animation-nested.qml

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
  
Window {  
    width: 300; height: 200; visible: true  
    Image {  
        id: qtlogo; anchors.centerIn: parent  
        source: "./images/qtlogo.png"
```

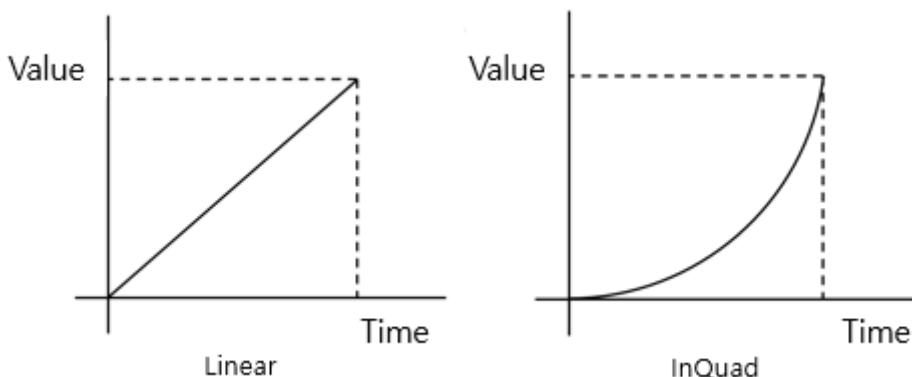
## Jesus loves you.

```
}

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

The Easing Curve can apply various Curves without the execution of Animation in a linear pattern until the beginning and end of Animation, such as moving, transparent, zooming, and shrinking coordinates. For example, you can change the rate of path progression from the starting to the last coordinate of the type to the following pattern.



<FIGURE> Easing Curve

The following example source code is an example of using OutExpo of the Easing curve.

# Jesus loves you.

<Example> Ch03 > 01 > 01\_Animation\_Example > easingcurve.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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
        }
    }
}
```

In the example above, the easing.type Property can specify one of the easing curves to use. You can see the difference between not using the Easing curve and not using it.

## ● PauseAnimation

PauseAnimation can pause animation for a period of time specified by the duration property value between animations executed sequentially in SequentialAnimation.



<FIGURE> PauseAnimation

The following example uses the PauseAnimation type.

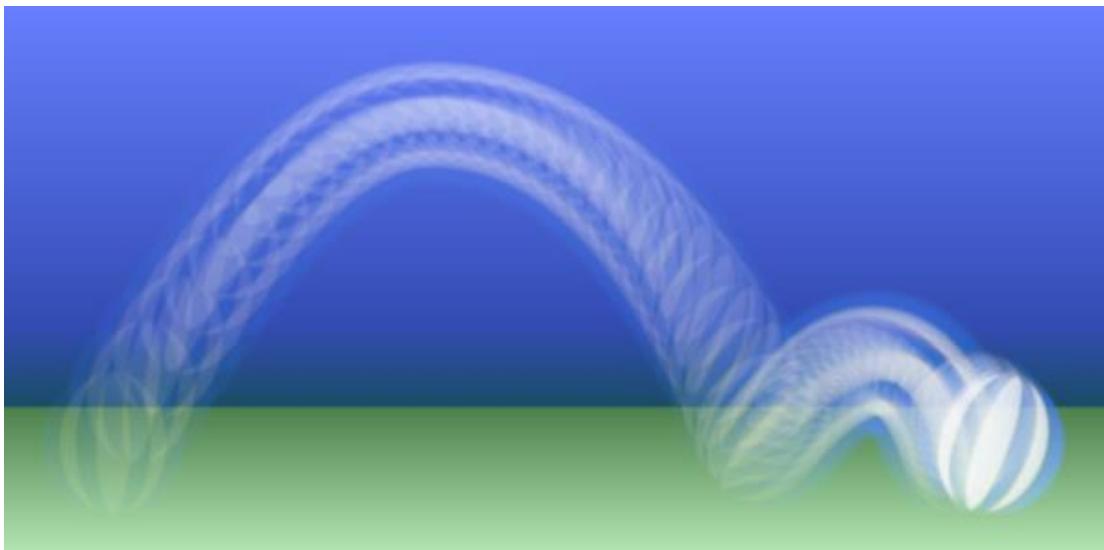
<Example> Ch03 > 01 > 01\_Animation\_Example > pauseanimation.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
```

## **Jesus loves you.**

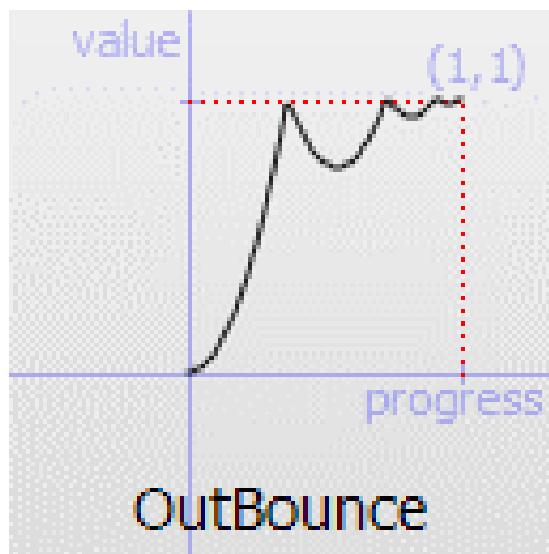
```
Window {
    width: 300; height: 200; visible: true
    Image {
        id: logo
        anchors.centerIn: parent; source: "images/qtlogo.png"
    }
    SequentialAnimation {
        NumberAnimation {
            target: logo; properties: "scale"
            from: 0.0; to: 1.0; duration: 1000
        }
        PauseAnimation {
            duration: 1000
        }
        NumberAnimation {
            target: logo; properties: "scale"
            from: 1.0; to: 0.0; duration: 1000
        }
        running: true
    }
}
```

## 3.2. Example using Animation and Easing Curve



<FIGURE> Ball Bouncing example

Let's make an example of the ball splashing on the floor as shown in the picture above. First, let's use the image type to display the ball image and apply an animation that bounces the ball from top to bottom as shown in the following example. Use the OutBounce of the Easing Curve to apply a bubbly animation when the ball image falls to the floor.



<FIGURE> OutBounce

## Jesus loves you.

<Example> Ch03 > 02 > 01\_Ball\_Bounding\_Example > bouncing-ball1.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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
        }
    }
}
```

When written and executed as above, the ball image will fall from the bottom and behave as if it were bouncing. Let's add one more NumberAnimation this time.

<Example> Ch03 > 02 > 01\_Ball\_Bounding\_Example > bouncing-ball2.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

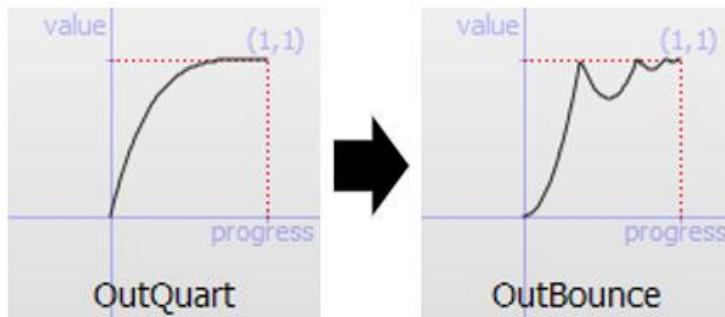
Window {
    width: 600; height: 300; visible: true
    Image {
        source: "images/ball.png"
        anchors.horizontalCenter: parent.horizontalCenter

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

## Jesus loves you.

```
        easing.type: "OutBounce"  
        duration: 1000  
    }  
}  
}  
}  
}
```

The added animation has an animation that seems to throw the ball image up. Therefore, the animation is performed sequentially as shown in the figure below.



<FIGURE> OutQuart 와 OutBounce

So far the ball image has moved on the Y axis. This time, let's apply animation on the X axis. To move to the X-axis, add NumberAnimation as follows.

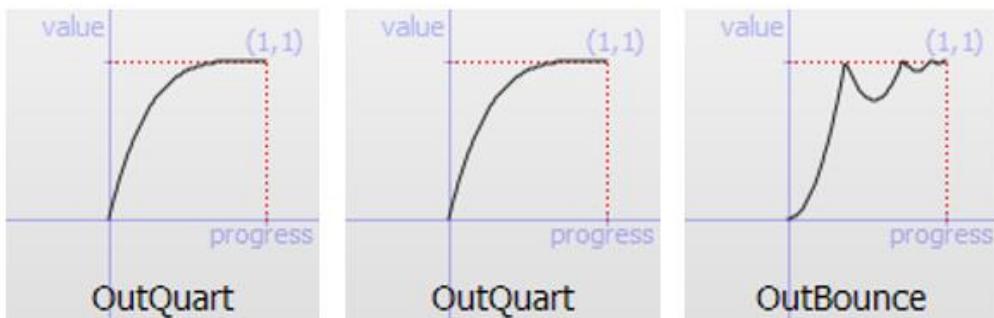
<Example> Ch03 > 02 > 01\_Ball\_Bounding\_Example > bouncing-ball3.qml

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
  
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 {  
                easing.type: "OutBounce"  
                duration: 1000  
            }  
        }  
    }  
}
```

## Jesus loves you.

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

As shown above, adding NumberAnimation moves the ball image from left to right. If you run the example, the ball image moves from left to right. And at the same time, it's like the ball bouncing. Animation works, for example, when we throw a ball, it bounces on the floor. In this example, three animations were used as shown in the following figure.



<FIGURE> Easing curve

Next, let's add RotationAnimation. In RotationAnimation, the ball image rotates 360 degrees clockwise.

<Example> Ch03 > 02 > 01\_Ball\_Bounding\_Example > bouncing-ball4.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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

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

## **Jesus loves you.**

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

Next, try using Animation ParallelAnimation and SequentialAnimation.

<Example> Ch03 > 02 > 01\_Ball\_Bounding\_Example > bouncing-ball5.qml

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
  
Window {  
    width: 600; height: 300; visible: true  
  
    Image {  
        id: ball  
        source: "images/ball.png"
```

## **Jesus loves you.**

```
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
        }
    }
}
```

## Jesus loves you.

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

Lastly, let's use Gradient as the background color to paint the background color. Let's add Rectangle and Gradient types to the code written as shown in the following example.

<Example> Ch03 > 02 > 01\_Ball\_Bounding\_Example > complete-bouncing-ball.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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) }
        }
    }
}
```

## **Jesus loves you.**

```
}
```

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

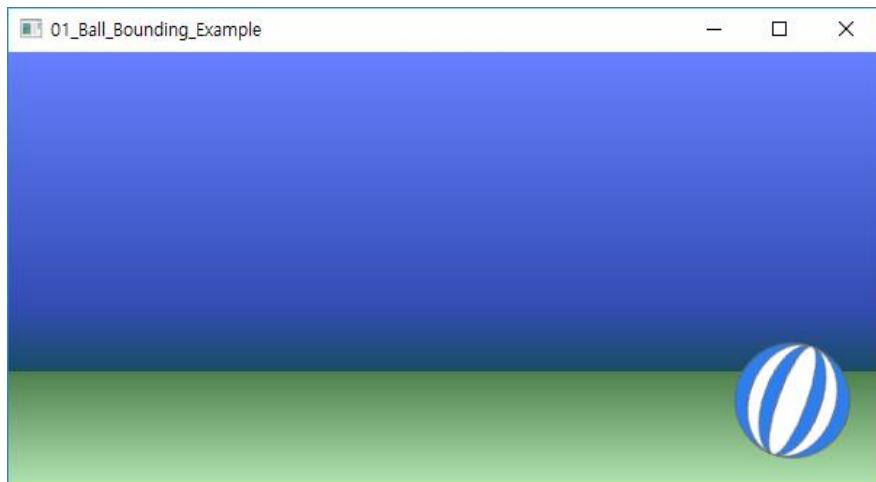
        NumberAnimation {
            target: ball
            property: "x"
            from: 20; to: 500
            easing.type: "OutQuad"
            duration: 1250
        }
        SequentialAnimation {
            NumberAnimation {
                target: ball
                property: "y"
                from: 200; to: 20
                easing.type: "OutQuad"
                duration: 250
            }
            NumberAnimation {
                target: ball
                property: "y"
                from: 20; to: 200
                easing.type: "OutBounce"
                duration: 1000
            }
        }
    }
}
```

```
SequentialAnimation {
```

## Jesus loves you.

```
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  
}  
}  
}  
}
```

If you write the source code as shown above, you can check that the animation is applied, such as the ball image bouncing. Animation works by executing the example you created and clicking on the ball image with the mouse.

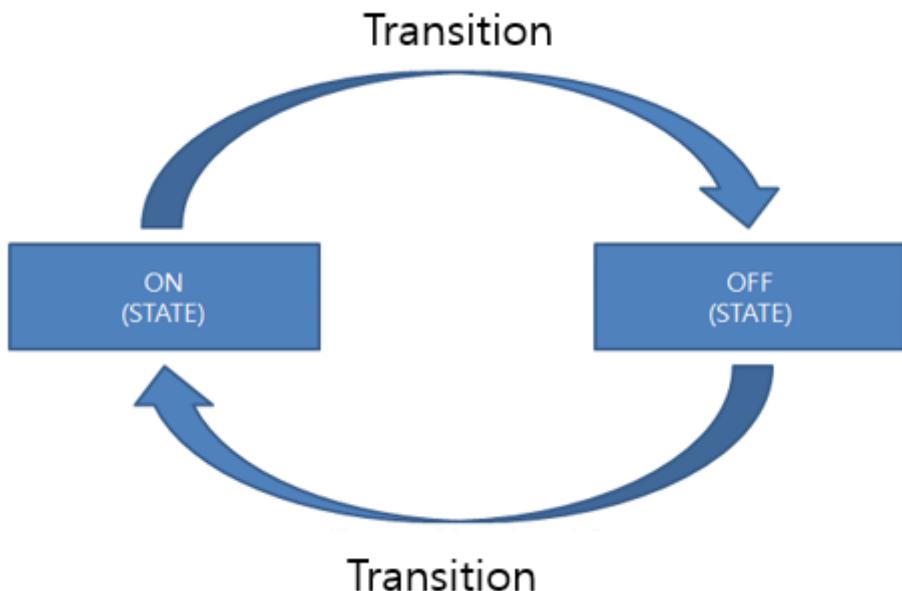


<FIGURE> Example Execution Screen

## 3.3. State and Transition

State and Transition can apply State Machine techniques to QML types. For example, suppose you have an ON/OFF switch. The ON or OFF state is defined as State. And the change of state from ON to OFF or OFF to ON is defined as the transition.

In QML, a state is a state of a certain type, and a change in the properties of a profferty caused by an event can be defined as a transition.



<FIGURE> State and Transition

### ● State Example

For example, if the Rectangle type color is Red, it is called State. Transition can be defined as the change of Rectangle type color from Red to Black. The following example illustrates the function of changing the color of Rectangle using State.

<Example> Ch03 > 03 > 01\_StateTransition > state.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    width: 150
```

## Jesus loves you.

```
height: 250
visible: true

Rectangle {
    width: parent.width; height: parent.height

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

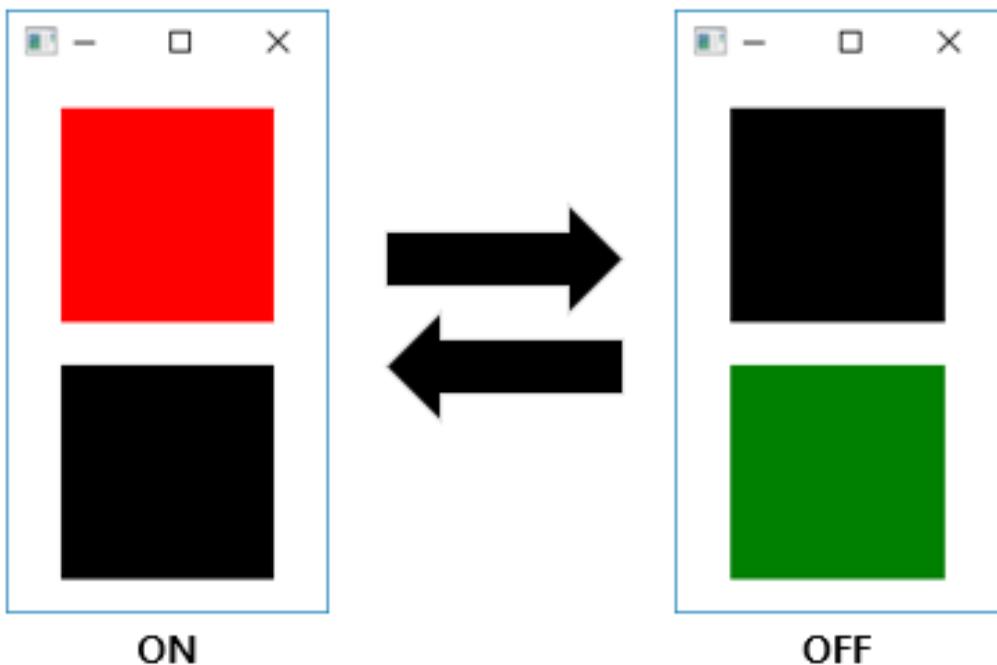
    Rectangle {
        id: offElement; x: 25; y: 135; width: 100; height: 100
    }

    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"
}
}
```

## Jesus loves you.



<FIGURE> Example Execution Screen

- **State and Transition example**

Let's use State and Transition to implement this example. The image is rotated when the example is executed, as shown in the figure below. State has two states, "up" and "down" and the image rotates 180 degrees when the transition from "up" to "down" is performed. Conversely, the transition from "down" to "up" rotates the image 180 degrees.

<Example> Ch03 > 03 > 01\_StateTransition > transition-rotate.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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"
```

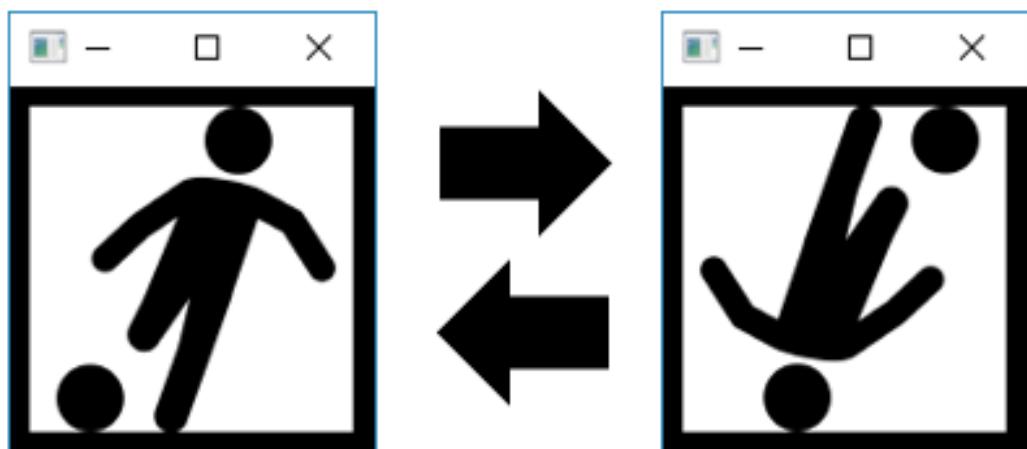
## Jesus loves you.

```
}

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"
}
}
```



<FIGURE> Example Execution Screen

# Jesus loves you.

- **State and when**

When the state is executed under certain conditions. When Properties can be used as shown in the following example.

<Example> Ch03 > 03 > 01\_StateTransition > state-when.qml

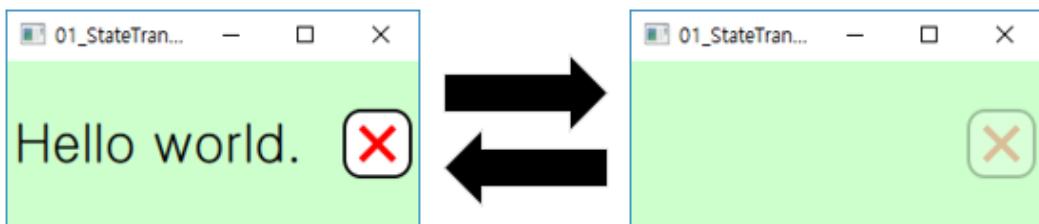
```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    width: 150; height: 250; visible: true
    Rectangle {
        width: 150; height: 250; color: "#ccffcc"
        TextInput {
            id: textField
            text: "Hello world."
            font.pointSize: 24
            anchors.left: parent.left
            anchors.leftMargin: 4
            anchors.verticalCenter: parent.verticalCenter
        }
        Image {
            id: closeButton
            source: "./images/close.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 != ""
            PropertyChanges { target: closeButton; opacity: 1.0 }
        },
        State {
    }
```

## Jesus loves you.

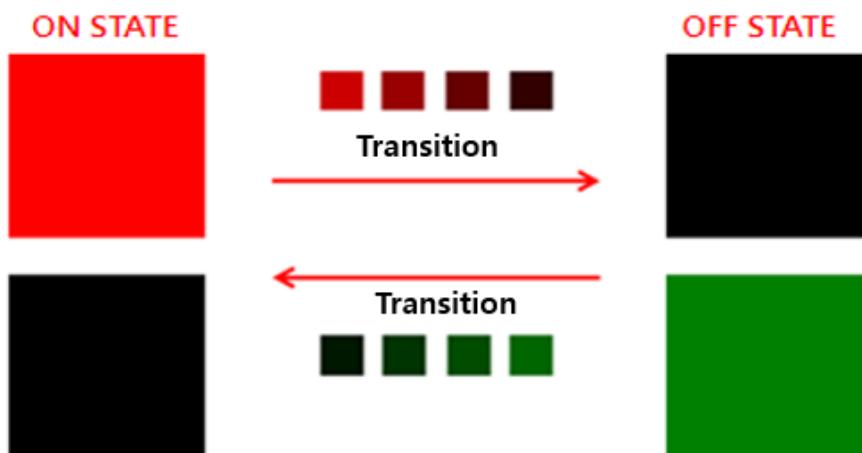
```
name: "without text"
when: textField.text == ""
    PropertyChanges { target: closeButton; opacity: 0.25 }
    PropertyChanges { target: textField; focus: true }
}
]
}
}
```



<FIGURE> Example Execution Screen

### ● Using Animation in Transition

In this example, let's try PropertyAnimation when Transition is performed. If PropertyAnimation's Properties are set to color Properties and Duration Property is set to 1000, animation effects such as clear color values during 1000 milliseconds can be used. The first transition was specified to change from "off" to "on" state, and the second transition was specified to change from "on" to "off" state.



<FIGURE> Transition

## Jesus loves you.

<Example> Ch03 > 03 > 01\_StateTransition > transition.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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

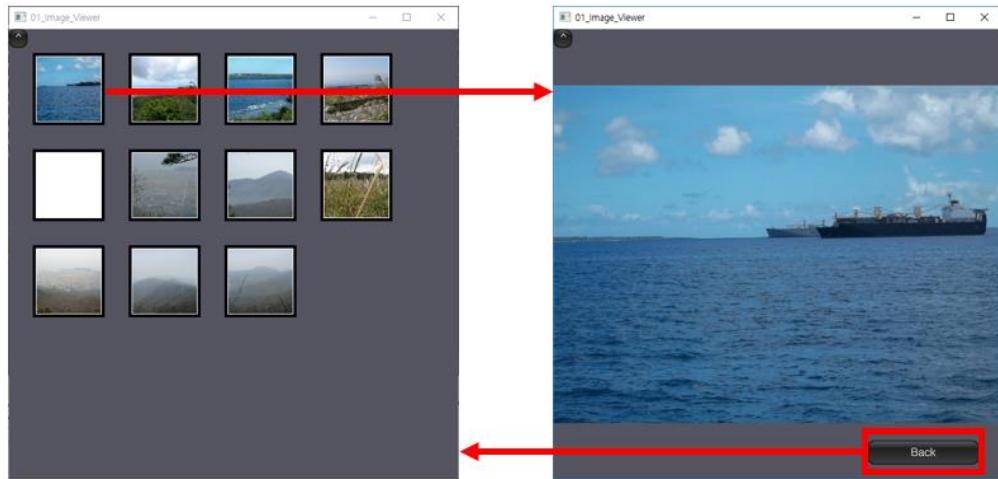
    Rectangle {
        width: 150; height: 250
        Rectangle {
            id: onElement
            x: 25; y: 15; width: 100; height: 100
        }
        Rectangle {
            id: offElement
            x: 25; y: 135; width: 100; height: 100
        }
        states: [
            State {
                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: [
            Transition {
                from: "off"; to: "on"
                PropertyAnimation {
```

## **Jesus loves you.**

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

## 3.4. Image Viewer Example

In this example, let's try to implement Image Viewer. Image Viewer is a screen that views Thumbnail images (reduced state) on the main screen. Click on the image you want to see larger, and you can see the image larger as shown in the picture below.



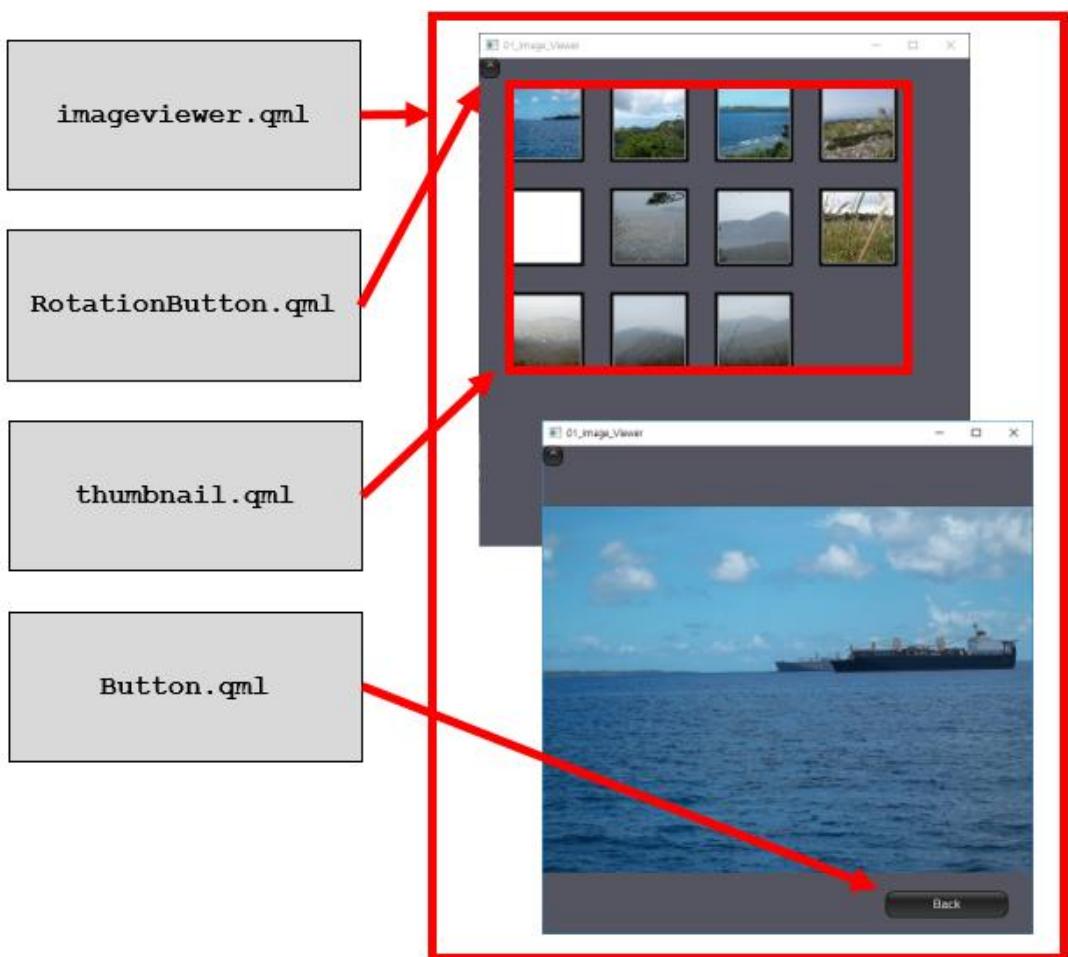
<FIGURE> Example Execution Screen

If you look at the top left of the screen, you can see that the image is displayed. This image can rotate the screen 90 degrees. For example, if we look at a cell phone from portrait to landscape, it provides the same function as the screen rotates 90 degrees to left or right. As such, this example is an Image Viewer example that provides similar functionality. The Image Viewer example consists of a total of four QML source files, as shown in the following figure.

<TABLE> QML Source code files

QML 소스 파일 명	설명
ImageViewer.qml	Main QML source code with QML type placed.
RotationButton.qml	Rotate the screen -90 degrees, click again to rotate 0 degrees.
Button.qml	Button, click to return to the previous screen.
Thumbnail.qml	User-defined type called Thumbnail

## Jesus loves you.



<FIGURE> Image Viewer QML source files

In `ImageViewer.qml`, call and use `Thumbnail`, `RotationButton`, and `Button`. The following example is the `ImageViewer.qml` example source code.

<Example> Ch03 > 04 > 01\_StateTransition > `ImageViewer.qml`

```
import QtQuick 2.12
import QtQuick.Window 2.12
import "module"

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

    Item {
        id: screen; width: 600; height: 600
```

## Jesus loves you.

```
property int animDuration: 500

RotationButton {
    id: rotationButton
    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"
    }
}
```

## Jesus loves you.

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

## Jesus loves you.

```
Image {  
    id: picture  
    z: 2  
    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" }  
    }  
]
```

## **Jesus loves you.**

```
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 {
```

## Jesus loves you.

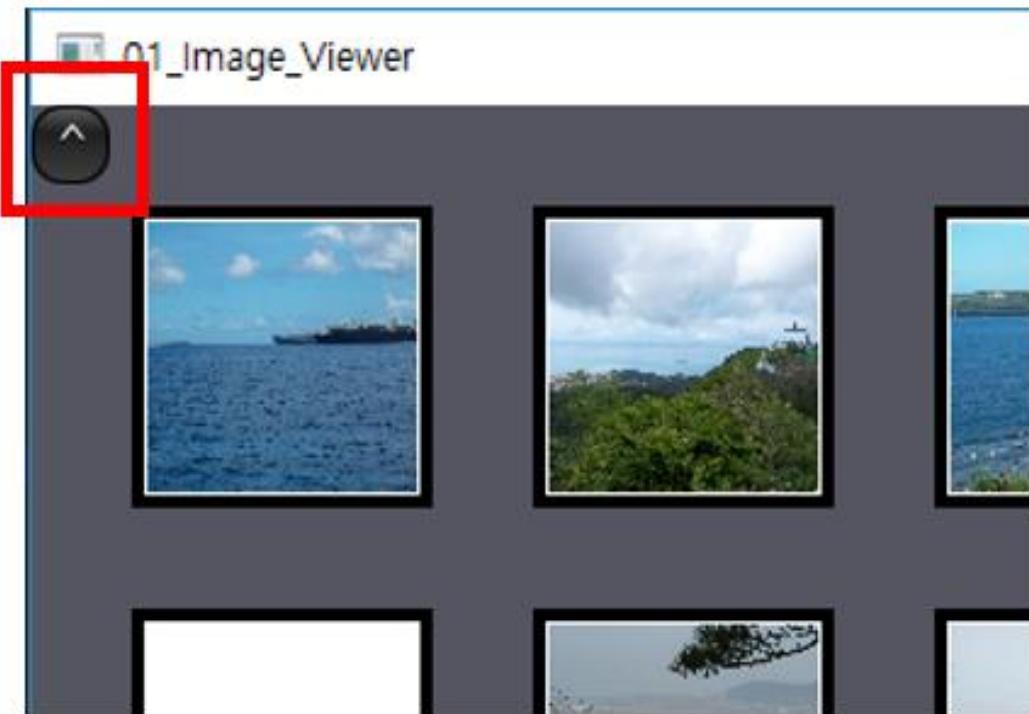
```
        target: picture
        properties: "x"; to: 2 * screen.width;
        easing.type: Easing.InOutQuad
    }
    NumberAnimation {
        target: backButton
        properties: "y";
        to: background.height + 3 * backButton.height
        duration: screen.animDuration * 2
        easing.type: 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
    }
}
]
}
}
```

The RotationButton.qml source file is located at the top left. Click this button to rotate -90 degrees. Click this button with -90 degrees of rotation to rotate it to 0 degrees, the original state.

## Jesus loves you.



<FIGURE> RotationButton position

The following example is the `RotationButton.qml` example source code.

<Example> Ch03 > 04 > 01\_StateTransition > module > `RotationButton.qml`

```
import QtQuick 2.12

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
    }
}
```

## Jesus loves you.

```
onClicked: {
    container.angle = (container.angle == 0 ? -90 : 0)
}

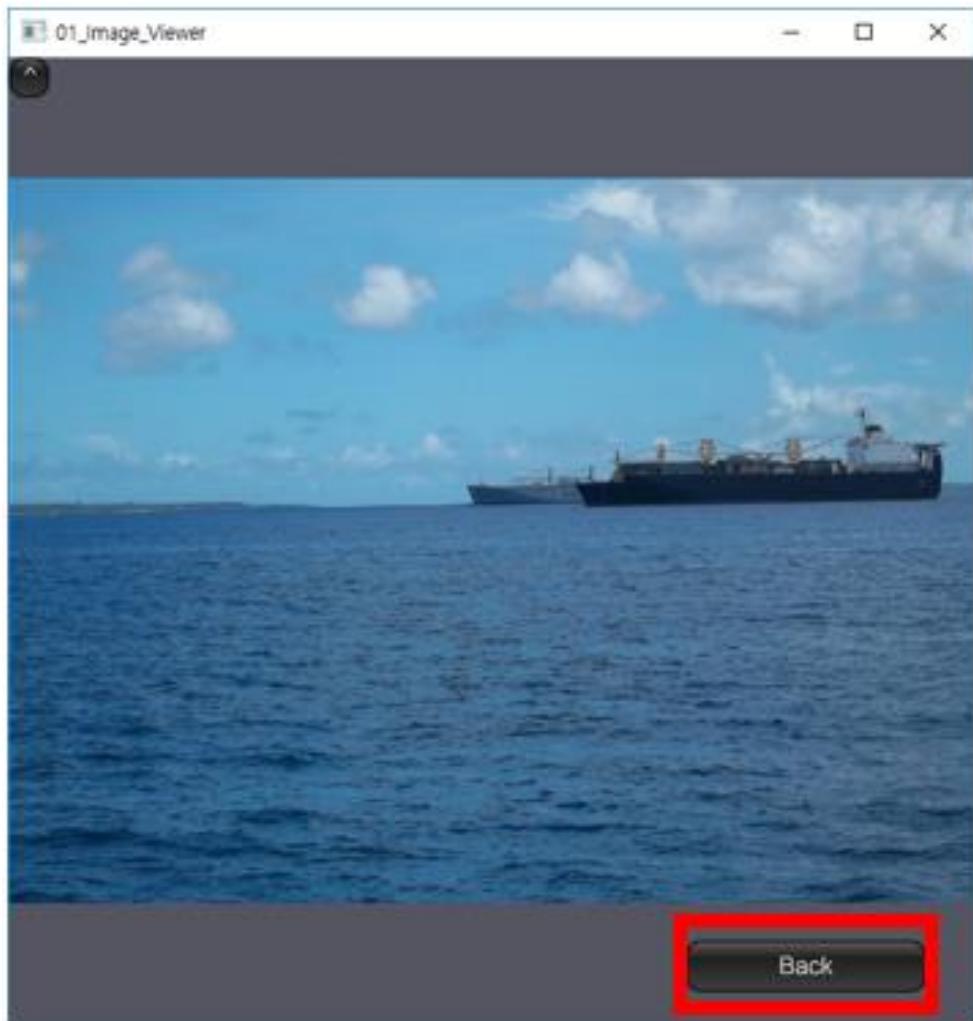
states: [
    State {
        name: "normal"
        when: container.angle == 0
    },
    State {
        name: "rotated"
        when: container.angle == -90
    }
]

state: "normal"

transitions: [
    Transition {
        from: "normal"; to: "rotated"
        NumberAnimation {
            targets: button
            properties: "rotation"; to: -90
            duration: 200
        }
    },
    Transition {
        from: "rotated"; to: "normal"
        NumberAnimation {
            targets: button
            properties: "rotation"; to: 0
            duration: 200
        }
    }
]
```

Button.qml reverts from image larger view screen to Thumbnail.qml screen.

## Jesus loves you.



<FIGURE> Button position

The following example is the Button.qml example source code.

<Example> Ch03 > 04 > 01\_StateTransition > module > Button.qml

```
import QtQuick 2.12

Item {
    id: container
    signal clicked
    property string text

    Rectangle {
        id: background
```

## **Jesus loves you.**

```
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"
    }
}

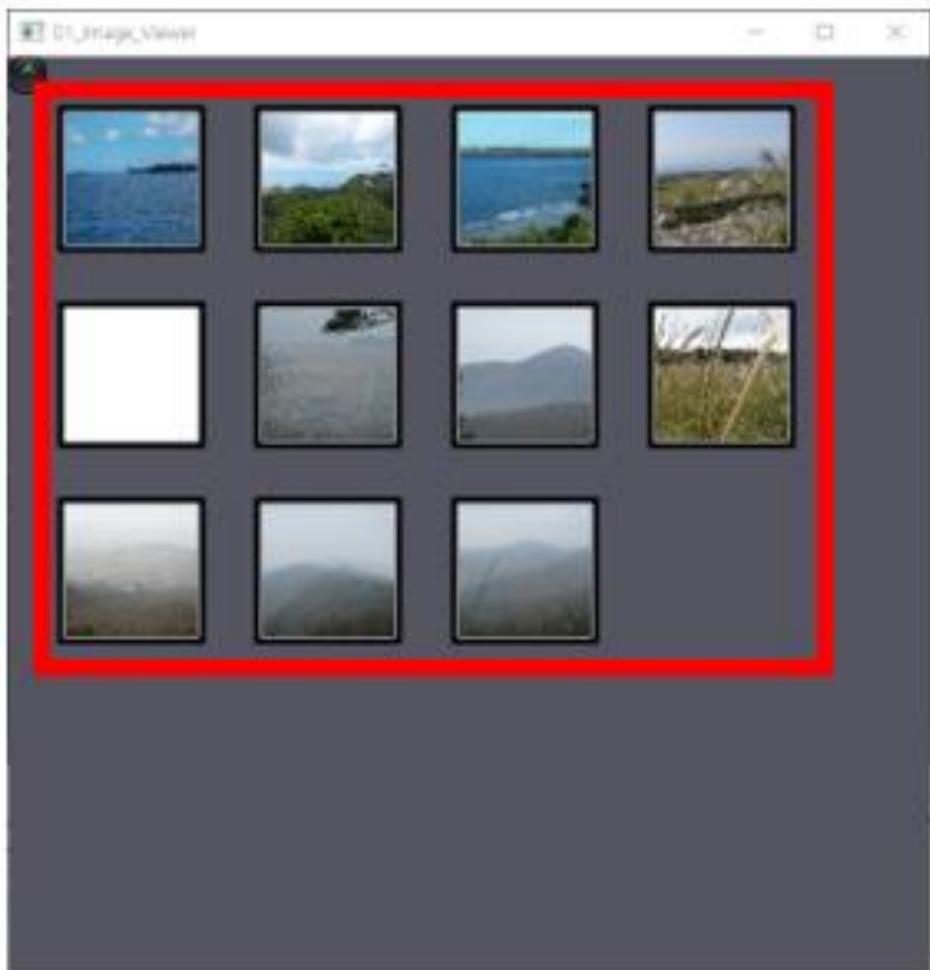
Text {
    color: "white"
    anchors.centerIn: background;
    font.bold: true;
    font.pixelSize: 15
    text: container.text
    style: Text.Raised
    styleColor: "black"
}

MouseArea {
    id: mouseRegion
    anchors.fill: parent
    onClicked: container.clicked()
}

}
```

## **Jesus loves you.**

Thumbnail.qml source file is a function to show several images on one screen by reducing the size of the original image and the view in the picture below.



<FIGURE> Thumbnail position

The following example source code is Thumbnail.qml source.

<Example> Ch03 > 04 > 01\_StateTransition > module > Thumbnail.qml

```
import QtQuick 2.12

Item {
    id: container
    signal clicked

    property int column
    property int row
```

## Jesus loves you.

```
property string image

width : 96; height : 96

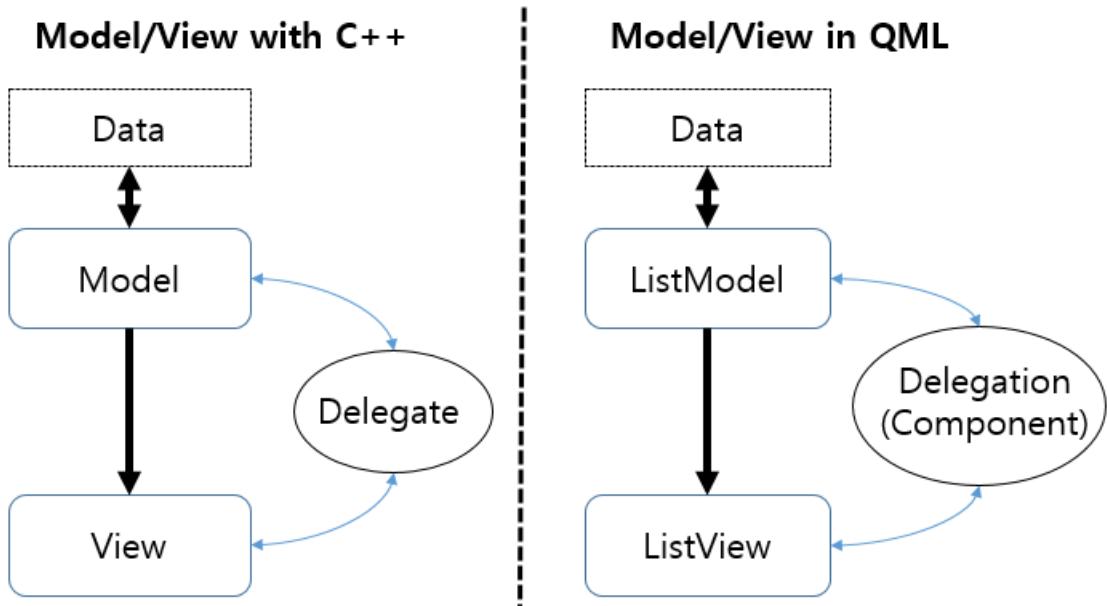
x: 32 + column * (width + 32)
y: 32 + row * (width + 32)

Rectangle { color: "black" ; 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();
}
}
```

## 4. Model / View

Qt/C++ provides features such as Model/View. The following figure compares the Model/View method used in C++ with the Model/View method used in QML.



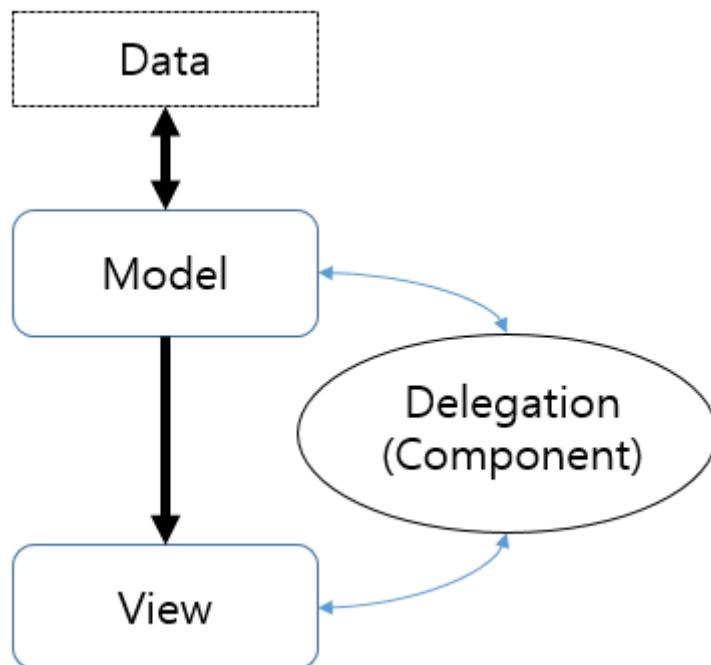
<FIGURE> Comparison of Model / View

As you can see in the figure above, the C++ method and QML method have the same concept and use method of use. In this chapter, let's look at how to use Model /View and learn more about how to use it through simple application examples.

## 4.1. Data representation using Model / View in QML

Use Model / View to express large amounts of data to users in QML, such as using tables to represent large amounts of data, in the same form as tables or in the same form as lists. Model can be viewed as a container for data. And View is like a tool, such as a table or a list, that shows users the containers that contain the data.

For example, in Model / View concepts, data is not inserted directly into View when inserted. Also, when data is modified, the data shown in View is not modified. Insert/modify/delete data is done in Model. View is connected to the Model to show the data contained in the Model and to communicate user events to the Model.



<FIGURE> Model / View Architecture

### ● **ListModel and ListView**

The ListModel provides the same functionality as the Container that holds the data you want to display in the ListView. A ListModel may have more than one ListElement. And use the component between ListModel and ListView. Component can specify the form or style in

## Jesus loves you.

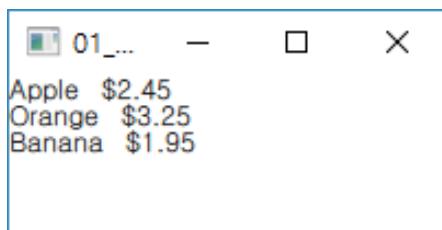
which data is represented in the ListView. The following are examples of using ListModel, ListView, and Component.

<Example> Ch04 > 01 > 01\_Presenting\_Data > listmodel\_exam\_1.qml

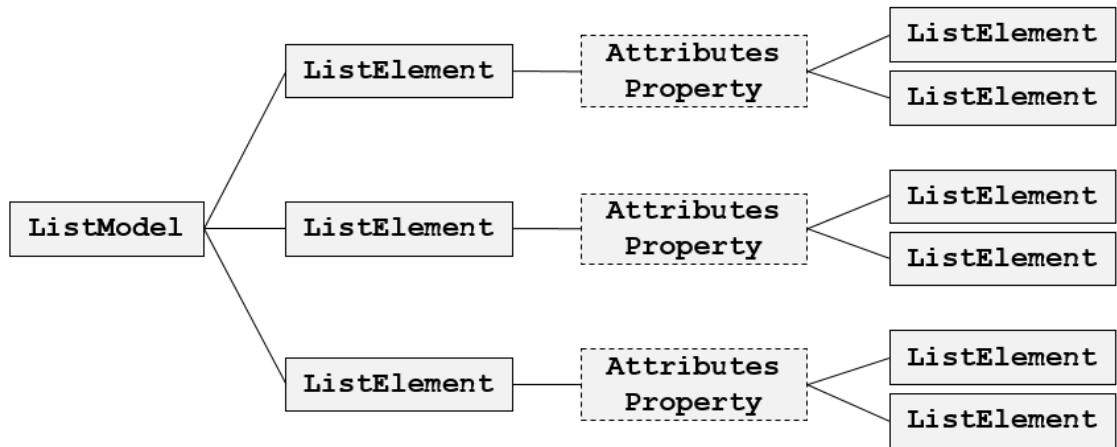
```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    width: 200; height: 200; 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
    }
}
```

As shown in the example above, the ListModel may use one or more ListElement to add data. In addition, the ListElement can be placed below it as shown in the following figure.



<FIGURE> Example Execution Screen



<FIGURE> ListModel and ListElement

## ● Use of attributes Property of ListElement

The following example is an example of using attributes to use ListElement in the lower part of ListElement.

<Example> Ch04 > 01 > 01\_Presenting\_Data > listmodel\_exam\_2.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

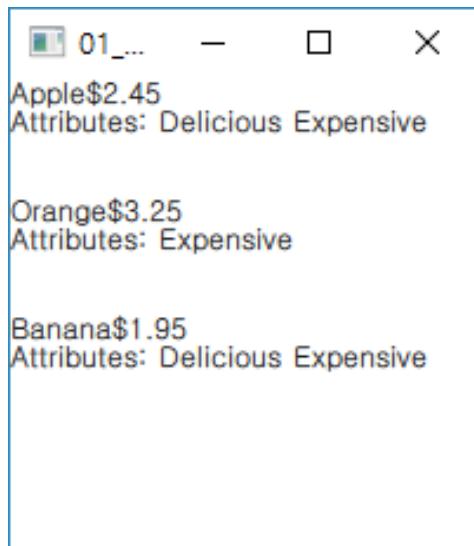
Window {
    width: 200
    height: 200
    visible: true

    ListModel
    {
        id: fruitModel
        ListElement {
            name: "Apple"; cost: 2.45
            attributes: [
                ListElement { description: "Delicious" },
                ListElement { description: "Expensive" }
            ]
        }
    }
}
```

## Jesus loves you.

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

## Jesus loves you.



<FIGURE> Example Execution Screen

### ● Header and footer Property of ListView

In the ListView, header properties can be used directly by specifying the top style. Footer can specify the style at the bottom. The following example is an example source code with headers and footers.

<Example> Ch04 > 01 > 01\_Presenting\_Data > list-view-header-footer.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    width: 400; height: 200; visible: true; color: "white"
    id: root

    ListModel {
        id: nameModel
        ListElement { name: "Alice"; }
        ListElement { name: "Bob"; }
        ListElement { name: "Jane"; }
        ListElement { name: "Victor"; }
        ListElement { name: "Wendy"; }
    }
}
```

## Jesus loves you.

```
Component {
    id: nameDelegate
    Text {
        text: name;
        font.pixelSize: 24
        anchors.left: parent.left
        anchors.leftMargin: 2
    }
}

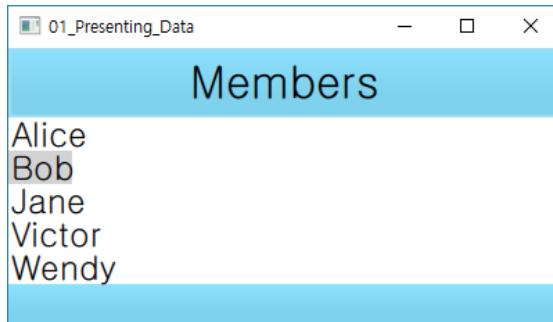
Component {
    id: bannercomponent
    Rectangle {
        id: banner
        width: root.width; height: 50
        gradient: clubcolors
        border {color: "#9EDDF2"; width: 2}
        Text {
            anchors.centerIn: parent
            text: "Members"
            font.pixelSize: 32
       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

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 {
```

## Jesus loves you.

```
width: parent.width; height: 30;  
gradient: clubcolors  
}  
  
highlight: Rectangle {  
    color: "lightgray"  
}  
  
// 키보드 이벤트로 메뉴를  
// 이동하기 위한 프로퍼티  
focus: true  
}  
}
```



<FIGURE> Example Execution Screen

### ● GridView

GridView can display ListElement in Grid style. The following example uses GridView.

<Example> Ch04 > 01 > 01\_Presenting\_Data > grid\_exam\_1.qml

```
import QtQuick 2.12  
import QtQuick.Window 2.12  
  
Window {  
    width: 300  
    height: 200  
    visible: true  
  
    ListModel {  
        id: gridModel
```

## Jesus loves you.

```
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" }

}

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: 80
    model: gridModel
    delegate: contactDelegate
    highlight: Rectangle {
        color: "lightsteelblue"; radius: 5
    }

    focus: true
}
}
```

## Jesus loves you.



<FIGURE> Example Execution Screen

As you can see in the picture above, when the example is executed, you can see the reverse area moving by moving to the keyboard orientation key.

### ● Using Animation in GridView

Animation can be used to move the directional key of the keyboard. The following is an example of using Animation in GridView.

<Example> Ch04 > 01 > 01\_Presenting\_Data > grid\_exam\_ani.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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" }
    }
    Component {
        id: highlight
        Rectangle {
```

## Jesus loves you.

```
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

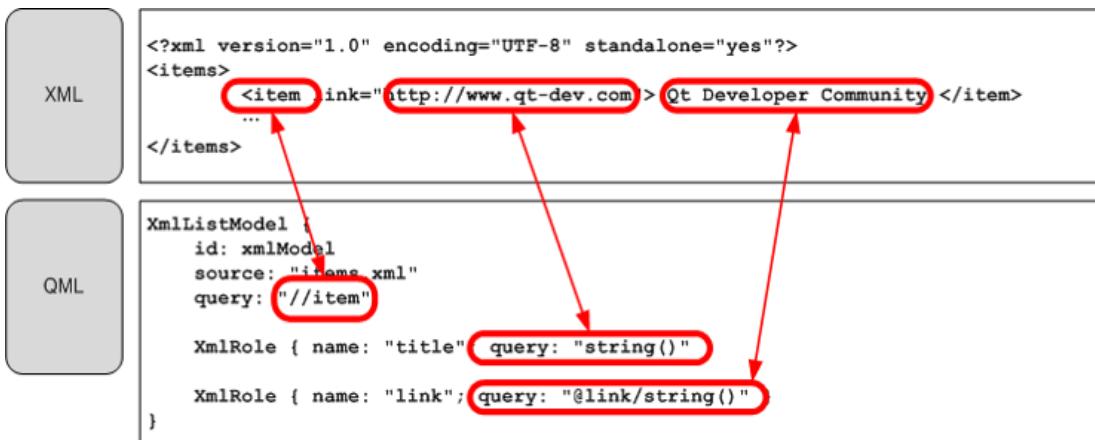
    model: gridModel
    delegate: Column {
        Image {
            width: 60; height: 50; source: frame;
            anchors.horizontalCenter: parent.horizontalCenter
        }
        Text {
            text: name;
            anchors.horizontalCenter: parent.horizontalCenter
        }
    }
}

highlight: highlight
highlightFollowsCurrentItem: true // 현재 아이템 Size가 일치하도록 설정
focus: true
}
}
```

### ● XML Model

QML provides an `XmlListModel` to express XML data. The `XmlListModel` can remotely import XML data by specifying an xml file using `source` properties or specifying an Internet URL.

## Jesus loves you.



<FIGURE> XML Model 사용방법

As you can see in the figure above, you can use query property to specify the identity portion of XML to query the data for that XML. The XmlRole can map data from the XmlListModel using xPath. The following example reads and displays data from XML files.

<Example> Ch04 > 01 > 01\_Presenting\_Data > xml\_exam.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtQuick.XmlListModel 2.0

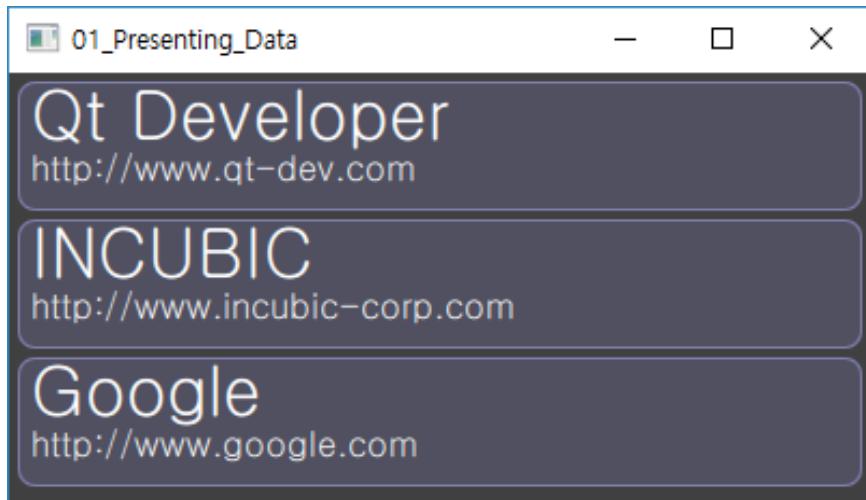
Window {
    width: 400; height: 200
    visible: true
    color: "#404040"

    XmlListModel {
        id: xmlModel
        source: "item.xml"
        query: "//item"
        XmlRole { name: "title"; query: "string()" }
        XmlRole { name: "link"; query: "@link/string()" }
    }

    Component {
        id: xmlDelegate
        Item {
            width: parent.width; height: 64
            Rectangle {
```

## Jesus loves you.

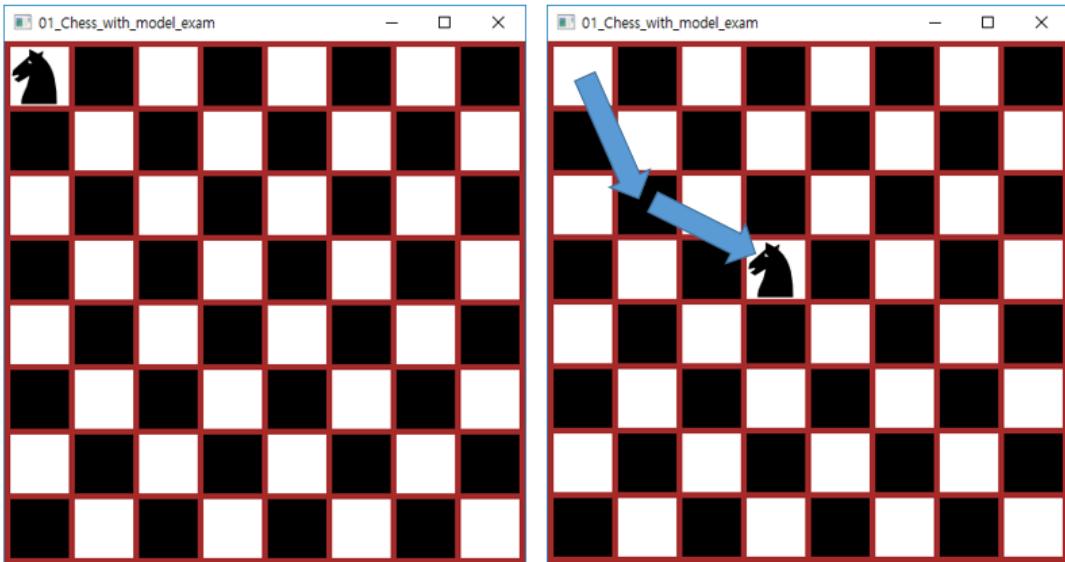
```
width: Math.max(childrenRect.width + 16, parent.width)
height: 60; clip: true
color: "#505060"; border.color: "#8080b0"; radius: 8
Column {
    Text { x: 6; color: "white"; font.pixelSize: 32; text: title }
    Text { x: 6; color: "white"; font.pixelSize: 16; text: link }
}
}
}
}
ListView {
    anchors.fill: parent; anchors.margins: 4; model: xmlModel
    delegate: xmlDelegate
}
}
```



<FIGURE> Example Execution Screen

## 4.2. Implementing Knight in chess

In this section, let's use Repeater and Mode/View to implement chess. The following figure allows Knight to move according to the rules of movement in chess.



<FIGURE> Example Execution Screen

- [Step 1] Implementing a chessboard

To implement the chessboard shape, 8 squares by 8 squares are implemented using repeater. The following is an example source code that embodies the shape of a chessboard.

<Example> Ch04 > 02 > 01\_Chess\_with\_model\_exam > chess-board1.qml

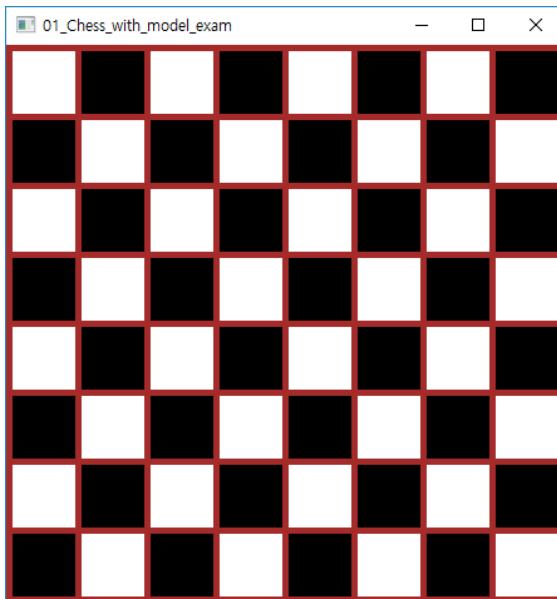
```
import QtQuick 2.12
import QtQuick.Window 2.12

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
```

## Jesus loves you.

```
Rectangle {  
    width: 50; height: 50  
    color: {  
        var row = Math.floor(index / 8);  
        var column = index % 8  
        console.log("row, column : " + row + "," + column)  
        if ((row + column) % 2 == 1) // Even-numbered 1, otherwise 0  
            "black";  
        else  
            "white";  
    }  
}  
}  
}  
}
```



<FIGURE> Example Execution Screen

- [Stage 2] Implementing Knight

This time, let's place Knight on the chessboard. Image type was used to place Knight on the chessboard. The following example source code is an example source code with Knight placed.

<Example> Ch04 > 02 > 01\_Chess\_with\_model\_exam > chess-board2.qml

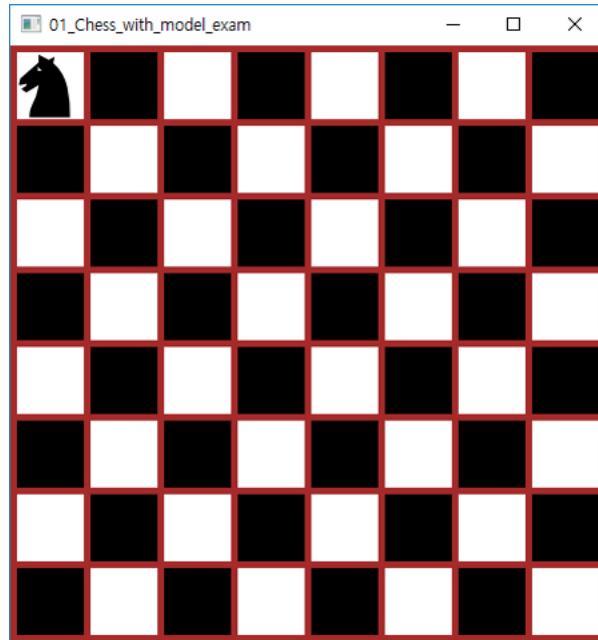
## **Jesus loves you.**

```
import QtQuick 2.12
import QtQuick.Window 2.12

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
    }
}
```

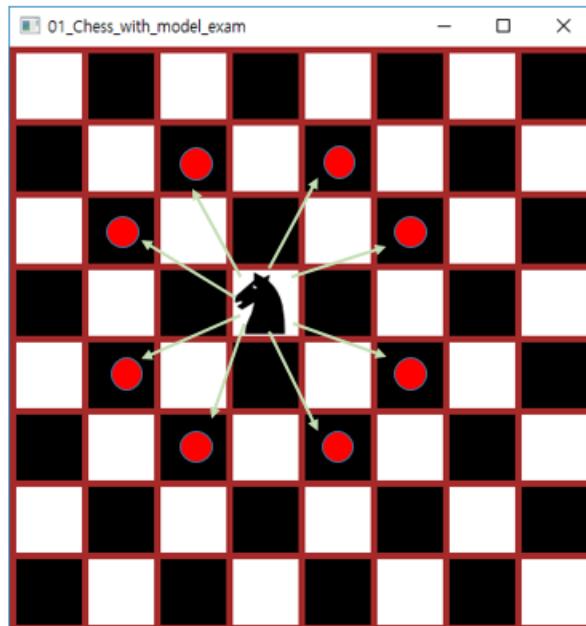
**Jesus loves you.**



<FIGURE> Example Execution Screen

- **[Step 3] Move Knight**

At this stage, let's implement the ability to move the deployed Knight using a mouse. As shown in the following figure, Knight can move forward 2 bays, side 1 bays, side 1 bays, and front 2 bays with movable rules.



<FIGURE> Knight Move Rule

## Jesus loves you.

- Calculate the value required to implement the rules and functions that Knight can move
  - cx, cy is the location of the knight
  - x and y are the current location of the Knight
  - Forward 2 bays, side 1 bays, side 1 bays, forward 2 bays.
  - $(x-cx) = 2$ ,  $(y-cy) = 1$  or  $(x-cx) = 1$ ,  $(y-cy) = 2$
  - Negative values may result, so only positive values should be shown in the above calculation expression.

Depending on the calculation of the values required for the above implementation, it can be implemented as follows to determine whether a position is possible for Knight to move:

```
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;
}
```

The following example source code is the full example source code with the Knight movement rule of chess.

<Example> Ch04 > 02 > 01\_Chess\_with\_model\_exam > chess-board3.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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)
                        color: "#0000ff"
                    else
                        color: "#ffff00"
                }
            }
        }
    }
}
```

## Jesus loves you.

```
        "black";
    else
        "white";
}
MouseArea {
    anchors.fill: parent
    onClicked: {
        var x = index % 8;
        var y = Math.floor(index/8);

        console.log("x, y :" + x + "," + y);
        // When Knight is two bays ahead, x,y = 2, 1
        // Or when Knight is one step forward, x,y = 1, 2
        console.log(Math.abs(x - knight.cx) + "," +
                    Math.abs(y - knight.cy))

        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
}
```

## **5. Integration QML and C++**

---

In this chapter, we will learn how to communicate data and events between C++ and QML.

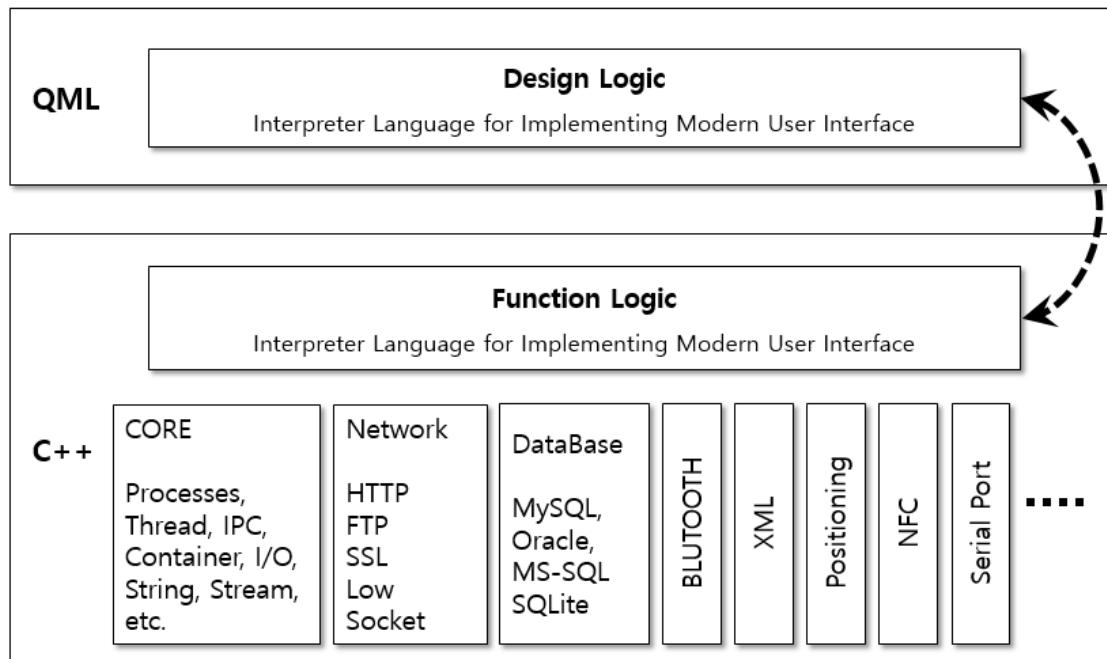
Suppose the interaction between C++ and QML is a dynamic display of data in QML. For example, in a chat program, when a new user accesses, the newly accessed user's information should be displayed on a GUI implemented in QML, then C++ should forward the newly accessed information to QML.

In this case, the information should be communicated from C++ to QML or from QML to C++. We will also look at how a member function of any particular class of C++ should call when an event occurs that clicks a specific button on QML.

Therefore, this chapter will explore how to communicate data and events between QML and C++.

## 5.1. Overview

The interaction between QML and C++ is the method for communicating data communication and events between QML and Qt, as shown in the figure below. It implements Design Logic (GUI) with QML and Function Logic with C++ to provide functions for interaction between QML and C++.



<FIGURE> Interaction Structure between QML and C++

Qt Meta-Object System is provided for data communication between Design Logic written in QML and Business Logic written in C++. To communicate QML-type Properties with C++, use `Q_PROPERTY( )` as follows: Use `Q_INVOKABLE` to invoke a member function of the C++ class in QML. When an event occurs in QML, the SLOT function defined by the accessor public slot can be executed in C++.

- **Implementing functions for communicating QML-type Properties with C++**

```
Q_PROPERTY(...)
```

# Jesus loves you.

- Call member function of C++ class from QML type

```
Q_INVOKABLE
```

- Invoke C++ Slot function when an event occurs in QML

```
public slots:  
    void refresh()
```

For the delivery of data communication events between QML and C++, there are four possible cases:

- ① Invoke C++ function from QML type to get result value

```
Rectangle  
{  
    width : 300; height: 300  
    Text {  
        text: ( To recall a function that returns the QString implemented in C++ )  
        ...  
    }  
    ...  
}
```

- ② Invoke a C++ function to get a return value from a QML type

```
Text {  
    ...  
    Component.onCompleted:  
    {  
        msg.author = "Hello" // To call author (QString str) function in C++ code  
    }  
}
```

- ③ Invoke Slot function implemented by C++ when an event occurs in QML

```
MouseArea {  
    anchors.fill: parent
```

## Jesus loves you.

```
onClicked:  
{  
    var str = "Who are you ?"  
    var result = msg.postMessage(str) // Public slots in C++ code  
    ...  
}
```

### ④ invoking a member function of a C++ class in QML

```
MouseArea  
{  
    anchors.fill: parent  
    onClicked:  
    {  
        ...  
        msg.refresh(); // Calling the defined function of the public in C++ code  
    }  
}
```

### ● Syntax of Q\_PROPERTY

Q\_PROPERTY is used in C++ and is used to implement functions for communicating QML-type Properties with C++. Syntax of Q\_PROPERTY is as follows.

```
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] )
```

## Jesus loves you.

For example, suppose you used Q\_PROPERTY as follows:

```
Q_PROPERTY(QString author READ author WRITE setAuthor NOTIFY authorChanged)
```

READ specifies the name of the function that returns the QString value of author from C++. WRITE assigns a value for the QString value of the author variable. For example, the setAuthor (QString author) function defines a function that sets the value to the first factor value. NOTIFY specifies the SIGNAL function. At C++, Q\_PROPERTY is declared as follows:

```
#include <QObject>

class Message : public QObject
{
    Q_OBJECT
    Q_PROPERTY( QString author
                READ author
                WRITE setAuthor
                NOTIFY authorChanged)
    ...
}
```

### ● **Q\_INVOKABLE**

Q\_INVOKABLE is used as a method for invoking the member function declared as Accessor Public in C++ from QML. The following is an example source code using Q\_INVOKABLE.

```
#include <QObject>

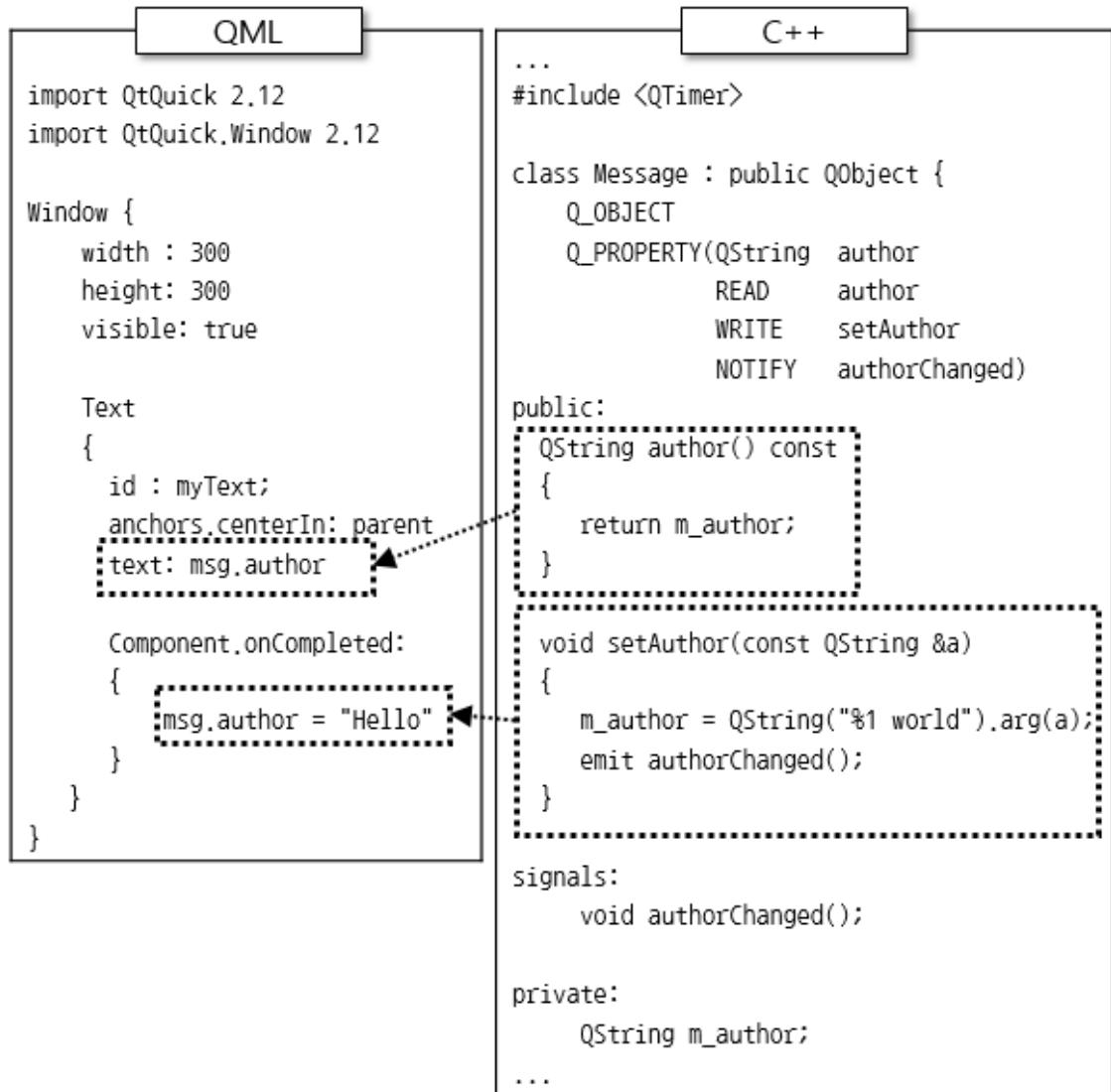
class Message : public QObject
{
    Q_OBJECT
    ...

public:
    Q_INVOKABLE bool postMessage(const QString &msg)
    {
        ...
        return true;
    }
}
```

## Jesus loves you.

- Recall the value of a specific Property of QML type (Exporting)

In order to recall the value of a particular property of the QML type, a member function of the C++ class can be called when defined using the `Q_PROPERTY()` provided by the `QObject` class. For example, to assign a text property value of text type among QML types, you can use:



<FIGURE> Invoke the value of a specific property of QML type

As seen in the above source code, the `QQuickView` class can be used to invoke a member function of the C++ `Message` class from QML. For example, the `QQuickView` class can be used as follows:

## Jesus loves you.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQuickView>
#include <QQmlContext>
#include <qqml.h>
#include "message.h"

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

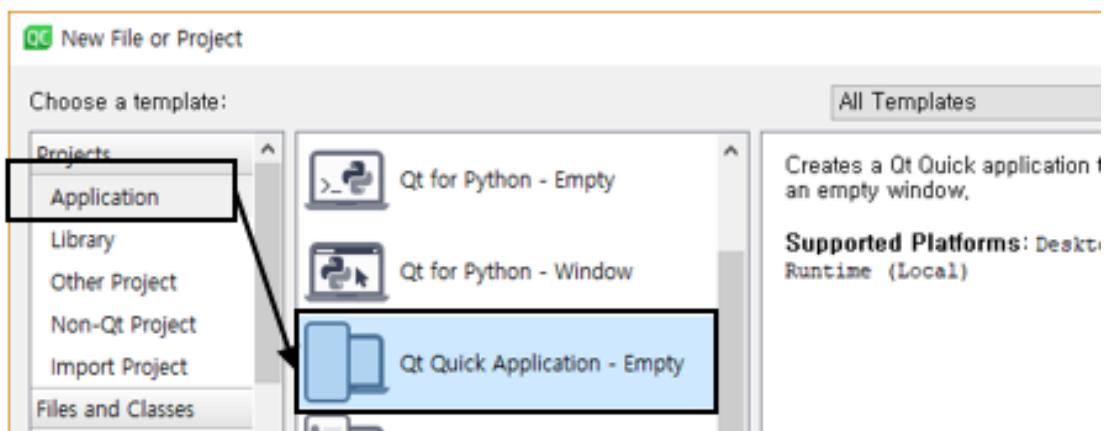
    QQuickView viewer;
    Message msg;

    viewer.engine()->rootContext()->setContextProperty("msg", &msg);
    viewer.setSource(QUrl( "qrc:///main.qml" ) );

    return app.exec();
}
```

- **Example for interaction between QML and C++**

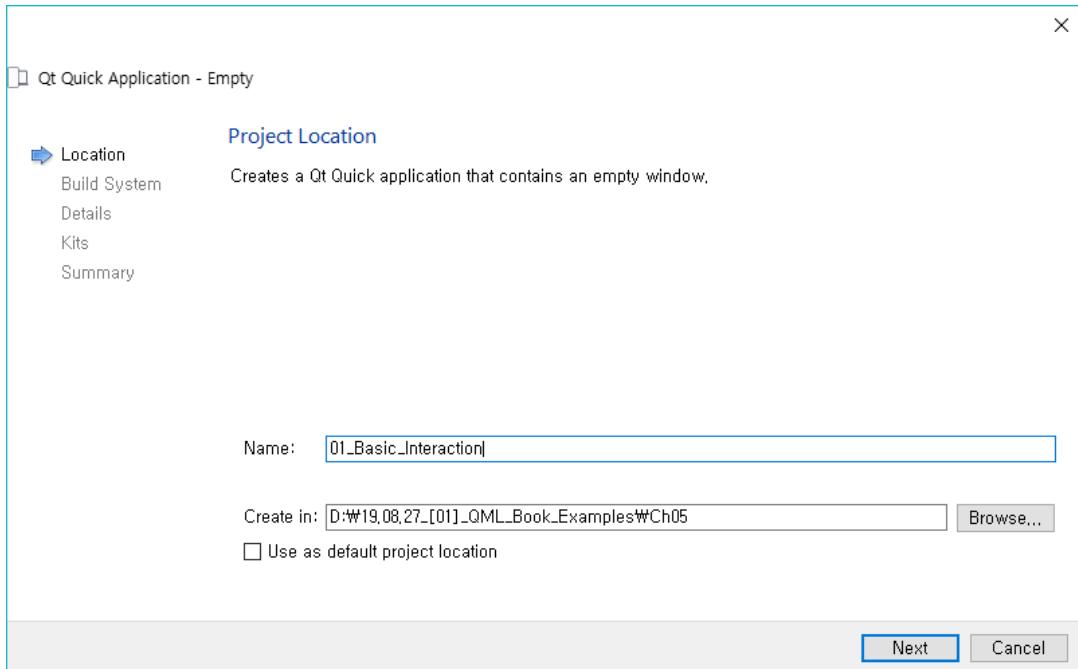
In this example, let's look at an example of the interaction between QML and C++. As described in the example source code above, let's set the value of the text type text property in QML to the return value of the autor( ) member function of the C++ class. Create a project as shown in the following figure.



<FIGURE> New File or Project

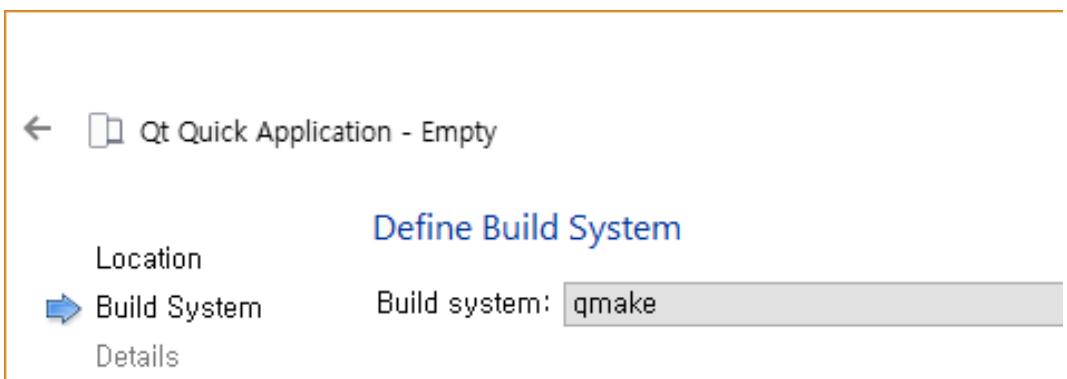
## Jesus loves you.

As you can see in the picture above, select Application > [Qt Quick Application - Empty] and select [Choose...] at the bottom.] Click the button.



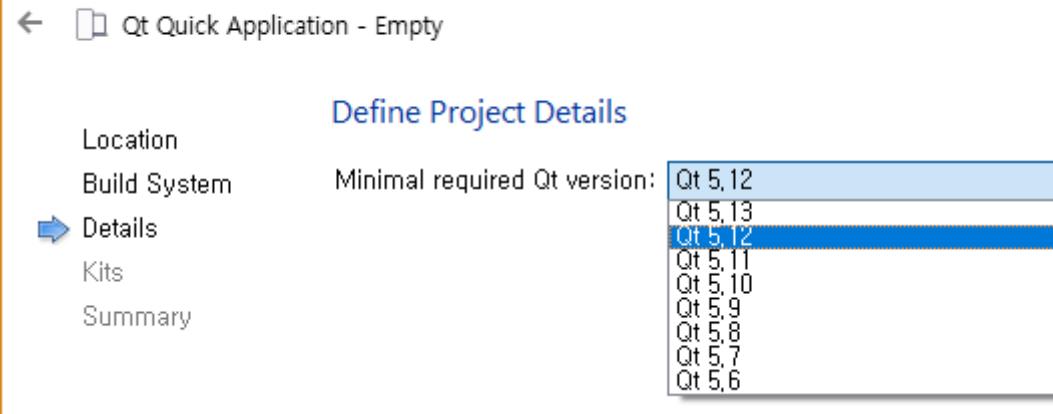
<FIGURE> Set Project Name and Project Location

Specify the project name and the directory in which the project will be located, as shown in the figure above. However, be careful not to use Hangul in the directory where the project name and project will be created. When using Hangeul in directory names, it may not be implemented.



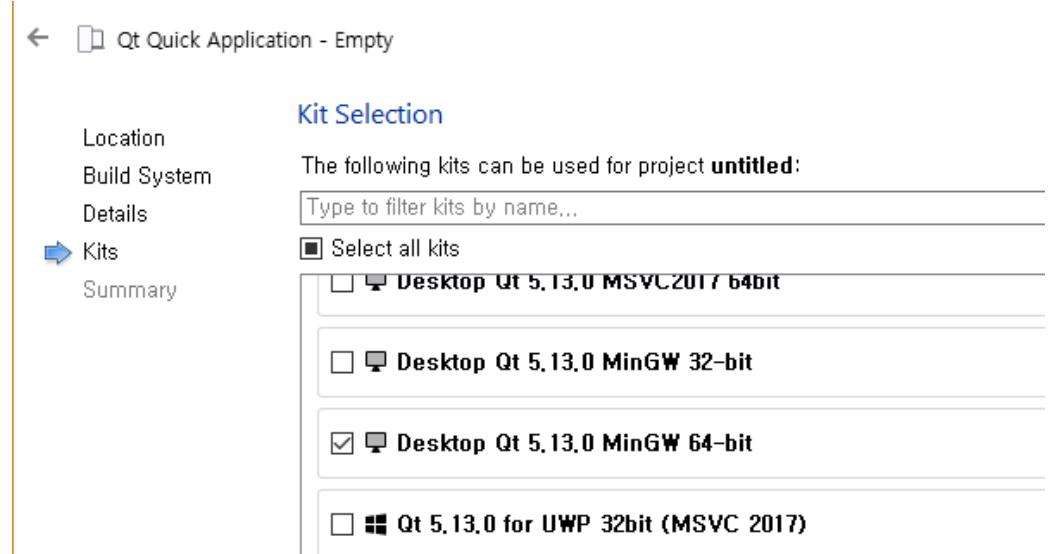
<FIGURE> Select Build System

As shown in the figure above, select qmake as the build system and click the [Next] button at the bottom to move on.



<FIGURE> QML version to use

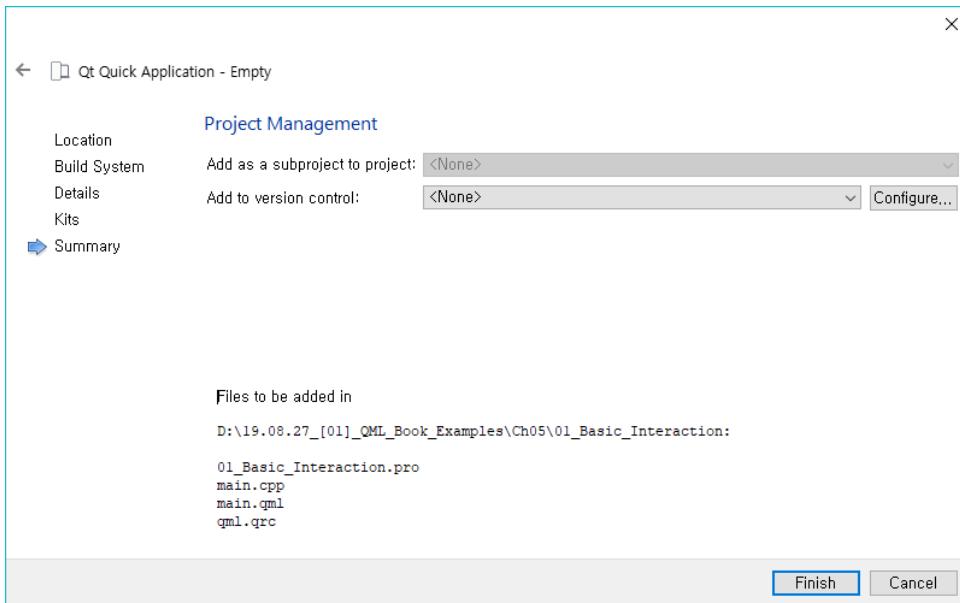
The figure above selects the minimum version to be used in QML. You may use the installed version of Qt.



<FIGURE> compiler selection

The above steps are the screens that select the compiler to build the created project. Here you select the MinGW 64-bit version.

# Jesus loves you.



<FIGURE> 프로젝트 Management 선택 화면

The above step is the final step, and you can select the source code versioning system. For example, you can choose whether to use SVN or Git. Here, select the None item and click the [Finish] button at the bottom as shown in the figure above. When the project creation is complete, the header and source code of the message class are written as follows, as shown in the following example.

<Example> Ch05 > 01 > 01\_Basic\_Interaction > message.h

```
#ifndef MESSAGE_H
#define MESSAGE_H

#include <QObject>
#include <QQmlProperty>
#include <QDebug>
#include < QTimer>

class Message : public QObject {
    Q_OBJECT
    Q_PROPERTY(QString author READ author WRITE setAuthor NOTIFY authorChanged)

public:
    void setAuthor(const QString &a) {
        m_author = QString("%1 world.").arg(a);
    }

    QString author() const {
        return m_author;
    }
};

#endif // MESSAGE_H
```

## **Jesus loves you.**

```
    emit authorChanged();
}

QString author() const {
    return m_author;
}

signals:
    void authorChanged();

private:
    QString m_author;
};

#endif // MESSAGE_H
```

<Example> Ch05 > 01 > 01\_Basic\_Interaction > message.cpp

```
#include "message.h"

//Message::Message()
//{
//}
```

Message.cpp is a source code implementation unit that requires the creation of function implementations, but is implemented in the header for convenience in understanding the source code. Complete the main.qml file as follows:

<Example> Ch05 > 01 > 01\_Basic\_Interaction > main.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

Window {
    width : 300; height: 300; visible: true
    Text {
        id : myText;
        anchors.centerIn: parent
        text: msg.author
        Component.onCompleted: {
            msg.author = "Hello"
```

## Jesus loves you.

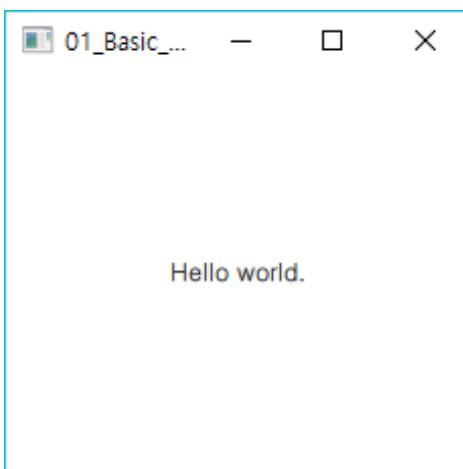
```
    myText.text = msg.author  
}  
}  
}
```

Next, to enable communication between QML and C++, write the main.cpp file as follows.

<Example> Ch05 > 01 > 01\_Basic\_Interaction > main.cpp

```
#include <QGuiApplication>  
#include <QQmlApplicationEngine>  
#include <QQuickView>  
#include <QQmlContext>  
#include <qqml.h>  
#include "message.h"  
  
int main(int argc, char *argv[])  
{  
    QGuiApplication app(argc, argv);  
    QQuickView viewer;  
    Message msg;  
    viewer.engine()->rootContext()->setContextProperty("msg", &msg);  
    viewer.setSource(QUrl( "qrc:///main.qml" ) );  
  
    return app.exec();  
}
```

If you have written the source code as above, let's build and run the project.

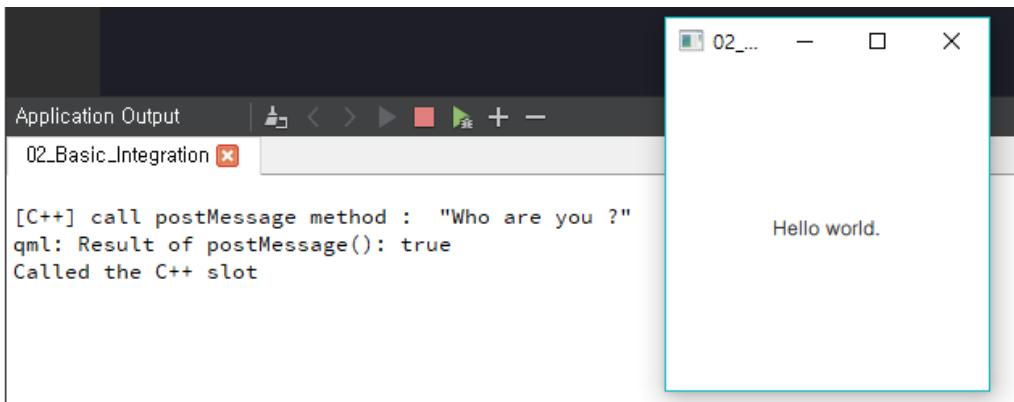


<FIGURE> Example Execution Screen

# Jesus loves you.

- **Example of using the Q\_INVOKABLE function**

In this example, let's create an example of performing the Slot function defined by Q\_INVOKABLE in the C++ Message class when you click the mouse in QML. Click on the screen to print the message in the console debugging window as shown in the following figure.



<FIGURE> Example Execution Screen

<Example> Ch05 > 01 > 02\_Basic\_Interaction > main.qml

```
import QtQuick 2.12
import QtQuick.Window 2.12

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: {
            var str = "Who are you ?"
            var result = msg.postMessage(str)
            console.log("Result of postMessage():", result);
            msg.refresh();
        }
    }
}
```

## Jesus loves you.

```
    }
}
}
```

<Example> Ch05 > 01 > 02\_Basic\_Interaction > message.h

```
#ifndef MESSAGE_H
#define MESSAGE_H

#include <QObject>
#include <QQmlProperty>
#include <QDebug>

class Message : public QObject {
    Q_OBJECT
    Q_PROPERTY(QString author READ author WRITE setAuthor NOTIFY authorChanged)
public:
    Message() {
        qDebug() << "Message() Construction";
    }
    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:
    QString m_author;

public slots:
    void refresh() {
```

## Jesus loves you.

```
    qDebug() << "Called the C++ slot";
}

};

#endif // MESSAGE_H
```

<Example> Ch05 > 01 > 02\_Basic\_Interaction > message.cpp

```
#include "message.h"

//Message::Message()
//{ }
```

<Example> Ch05 > 01 > 02\_Basic\_Interaction > main.cpp

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQQuickView>
#include <QQmlContext>
#include <qqml.h>
#include "message.h"

int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);
    QQQuickView viewer;
    Message msg;

    viewer.engine()->rootContext()->setContextProperty("msg", &msg);
    viewer.setSource( QUrl( "qrc:///main.qml" ) );

    return app.exec();
}
```

Let's write the source code and run the example as shown in the example above. In QML, you can see that the function defined by `Q_INVOKABLE` of C++ is performed in `onClicked` Property of `MouseArea` type.

## 5.2. Implementing QML Type with C++

In this section, let's implement the QML type using C++. To use the QML type implemented in C++ in QML, the `qmlRegisterType( )` function can be used and two methods can be used as follows.

```
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);
```

The following is an example source code for how to use the `qmlRegisterType( )` function. The first factor specifies the name to import when QML import. You can specify the type name to use in the Major version for the old one, the Minor version for the third, and the QML for the last four.

```
qmlRegisterType<Message>( "Message", 1, 0, "Msg" );
```

In order to define the QML module called Message as a C++ class, it can be implemented as follows.

```
class Message : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString author READ author WRITE setAuthor NOTIFY authorChanged)
    ...
}
```

The following example is available in QML, assuming that the QML module called Message is implemented as shown in the following example.

```
import QtQuick 2.4
import Message 1.0

Rectangle {
    width : 300; height: 300
```

## Jesus loves you.

```
Msg {  
    ...  
}  
...
```

The use of the second factor when using the qmlRegisterType( ) function is used to specify the revision. The following are examples of using the Revision version.

```
qmlRegisterType<Message, 1>("Message", 1, 1, "Msg")
```

If the revision is used as shown above, the revision information should be added on Q\_PROPERTY.

```
class Message : public QObject  
{  
    Q_OBJECT  
    Q_PROPERTY(QString author READ author WRITE setAuthor  
              NOTIFY authorChanged REVISION 1)  
signals:  
    Q_REVISION(1) void authorChanged();  
...
```

It can be implemented in a form where subtype exists inside the newly implemented QML type in QML.

```
MessageBoard  
{  
    Message { author: "Eddy" }  
    Message { author: "Candy" }  
}
```

In addition to the above, it can be used in the following ways:

```
MessageBoard  
{  
    messages: [  
        Message { author: "Eddy" },  
        Message { author: "Candy" }  
    ]  
}
```

You can use the Q\_CLASSINFO( ) macro and the QQmlListProperty( ) function for use as shown in the two QML examples above.

## Jesus loves you.

```
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;
};
```

- Example of a custom QML type implementation using C++ classes

In this example, let's implement the new type to be used in QML. The name of the module to be implemented uses the module name Message. And three seconds after running the program, the following Slot function is called by the QTimer defined in the Message class.

```
void timerTimeout()
{
    emit newMessagePosted("I am a boy");
}
```

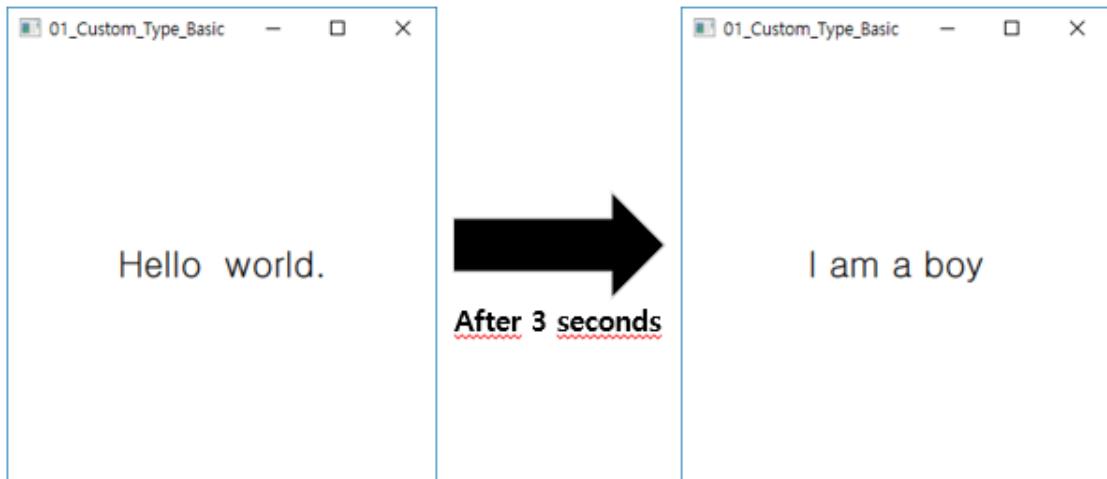
When the above Slot function is called, the onNewMessagePosted QML Slot Property of Msg type is called as follows. The following are part of the example QML:

```
...
import Message 1.0

Rectangle {
    ...
    Text {
        ...
        Msg {
            id: myMsg
            onNewMessagePosted: {
                console.log("[QML] New Message received: ", subject);
                myText.font.pixelSize = 25
                myText.text = subject;
            }
        }
    }
}
```

## Jesus loves you.

```
        }  
    }  
}
```



<FIGURE> Example Execution Screen

The "Hello world" string output from the window changes to "I am a boy" as shown in the figure above. The following source code is the header file source code for the Message class.

<Example> Ch05 > 02 > 01\_Custom\_type\_Basic > message.h

```
#ifndef MESSAGE_H  
#define MESSAGE_H  
  
#include <QObject>  
#include <QQmlProperty>  
#include <QDebug>  
#include <QTimer>  
  
class Message : public QObject  
{  
    Q_OBJECT  
    Q_PROPERTY(QString author READ author WRITE setAuthor NOTIFY authorChanged)  
  
public:  
    Message() {  
        qDebug() << "Message() Construction";  
        QTimer::singleShot(3000, this, SLOT(timerTimeout()));  
    }
```

## Jesus loves you.

```
}

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
```

<Example> Ch05 > 02 > 01\_Custom\_type\_Basic > message.cpp

```
#include "message.h"
```

## **Jesus loves you.**

```
//Message::Message()  
//{  
//}
```

<Example> Ch05 > 02 > 01\_Custom\_type\_Basic > main.qml

```
import QtQuick 2.12  
import Message 1.0  
  
Rectangle {  
    width : 300; height: 300  
    Text {  
        id : myText;  
        anchors.centerIn: parent  
        text: myMsg.author  
        font.pixelSize: 25  
  
        Component.onCompleted: {  
            myMsg.author = "Hello "  
        }  
  
        Msg {  
            id: myMsg  
  
            onNewMessagePosted: {  
                console.log("Message received: ", subject);  
                myText.font.pixelSize = 25  
                myText.text = subject;  
            }  
        }  
    }  
}
```

<Example> Ch05 > 02 > 01\_Custom\_type\_Basic > main.cpp

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

## Jesus loves you.

```
#include <QQmlContext>
#include <qqml.h>
#include "message.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    qmlRegisterType<Message>( "Message", 1, 0, "Msg" );

    QQQuickView viewer;
    Message msg;
    viewer.engine()->rootContext()->setContextProperty("msg", &msg);

    viewer.setSource( QUrl( "qrc:///main.qml" ) );
    viewer.show();

    return app.exec();
}
```

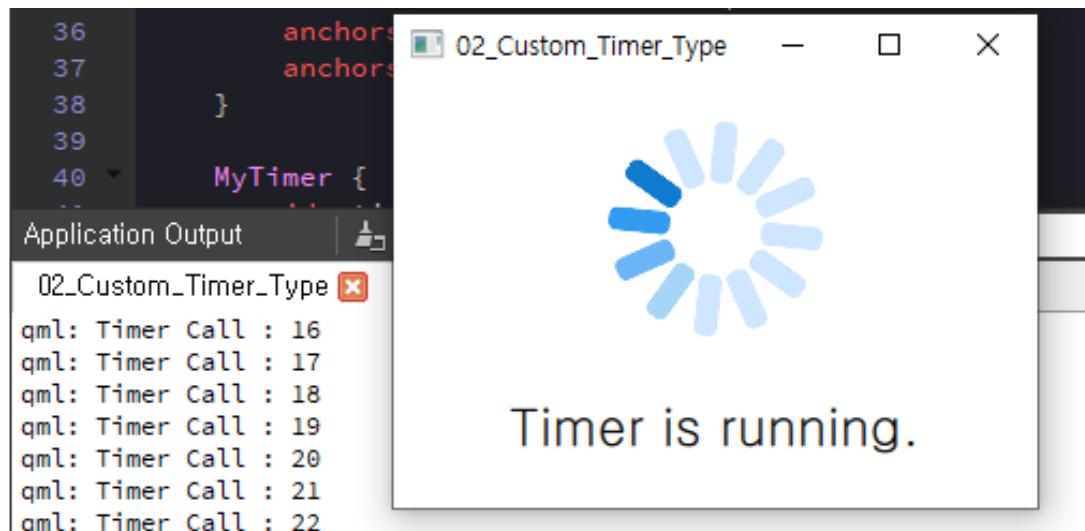
- **Timer Type Implementation**



<FIGURE> Example Execution Screen

As shown in the figure above, the timer is stopped when QML loads. Click in the QML area to activate the timer. Then print the debugging message from the debugging window of the Qt Creator IDE Tool as shown below.

**Jesus loves you.**



<FIGURE> Application Output Debugging Message

<Example> Ch05 > 02 > 02\_Custom\_Timer\_Type > main.cpp

```
#include "timer.h"

#include <QGuiApplication>
#include <QQuickView>
#include <qqml.h>

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

    qmlRegisterType<Timer>("MyCustomTimer", 1, 0, "MyTimer");
    QQuickView viewer;

    viewer.setSource( QUrl( "qrc:///main.qml" ) );
    viewer.show();

    return app.exec();
}
```

In the example source code above, "MyCustomTimer," the first factor in the `qmlRegisterType()` function, is used in QML as the name of the QML module to import. And the second and third factors are version information and the last fourth factor, MyTimer, is the name of the type used in QML. The following example is the QML source code.

## Jesus loves you.

<Example> Ch05 > 02 > 02\_Custom\_Timer\_Type > main.qml

```
import QtQuick 2.12
import MyCustomTimer 1.0

Rectangle {
    width: 300
    height: 200
    property int timerCnt: 0

    Image {
        id: loadImage
        source: "qrc:/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 stop state."
        font.pixelSize: 24
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.top: loadImage.bottom
        anchors.topMargin: 30
    }

    MyTimer {
        id: timer
    }
}
```

## Jesus loves you.

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

In the QML source code above, onTimeout is a Signal Property. This Property occurs when timeout( ) signals are called in the C++ Timer class. In the C++ Timer class, the start( ) and stop( ) functions of the Slot functions are mapped to the timer.start( ) and timer.stop( ) functions in the QML source code above. For example, if timer.start( ) is called in QML, the start() member function in the C++ Timer class is called. The following example is the header file for the Source Code Timer class.

<Example> Ch05 > 02 > 02\_Custom\_Timer\_Type > timer.h

```
#ifndef TIMER_H
#define TIMER_H
#include <QObject>

class QTimer;

class Timer : public QObject
{
```

## Jesus loves you.

```
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 = 0 );
    void setInterval( int msec );
    int interval();
    bool isActive();

public slots:
    void start();
    void stop();

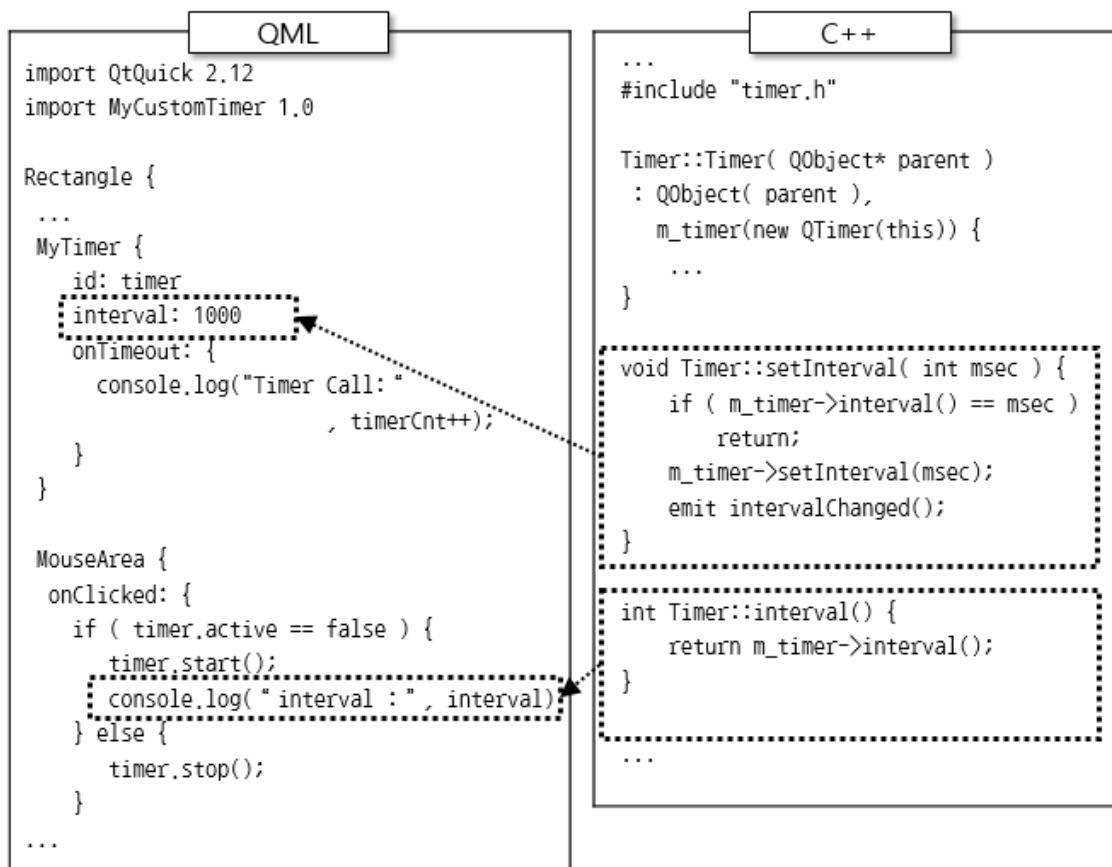
signals:
    void timeout();
    void intervalChanged();
    void activeChanged();

private:
    QTimer* m_timer;
};

#endif // TIMER_H
```

In the QML source code above, 1000 values were assigned to the interval property, when the setInterval( ) member function is called from the C++ Timer class. The interval( ) member function is called from the Timer class when the interval value is read from QML.

## Jesus loves you.



<FIGURE> Mapping between QML and Timer Classes

<Example> Ch05 > 02 > 02\_Custom\_Timer\_Type > timer.cpp

```

#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 )
{

```

## Jesus loves you.

```
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();
}

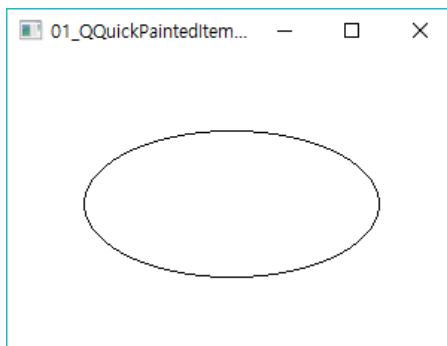
void Timer::start()
{
    if ( m_timer->isActive() )
        return;
    m_timer->start();
    emit activeChanged();
}

void Timer::stop()
{
    if ( !m_timer->isActive() )
        return;
    m_timer->stop();
    emit activeChanged();
}
```

## 5.3. Use the QQuickPaintedItem class in QML

The QQuickPaintItem class provides the same functionality as the QPainter class. The difference is used by the QPainter class to display 2D graphic elements (lines, lines, circles, curves, gradients, image indicators) within the GUI window area implemented by the QWidget class. However, the QQuickPaintItem class can display 2D graphic elements within the QML-type area. Therefore, 2D graphic results drawn into the QQuickPaintItem class can be displayed within the QML-type area.

QML	C++
<pre>import QtQuick 2.12 import Shapes 1.0  Item {     width: 300     height: 200      Ellipse {         x: 50;         y: 50         width: 200         height: 100     } }</pre>	<pre>#include &lt;QQuickPaintedItem&gt;  class EllipseItem : public QQuickPaintedItem {     Q_OBJECT  public:     EllipseItem( QQuickItem *parent = 0 );     void paint(QPainter *painter); };</pre>



<FIGURE> Example Execution Screen

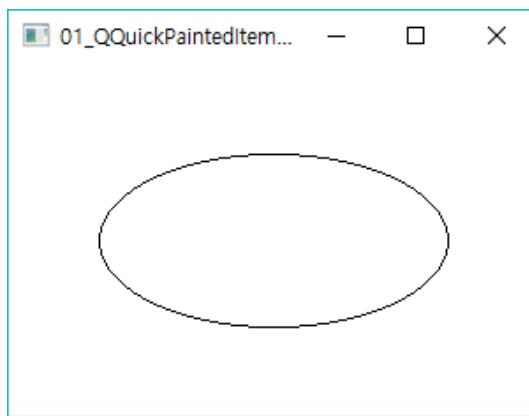
## **Jesus loves you.**

The result of drawing an ellipse from the paint( ) member function of the EllipseItem class on the left example source code to display the ellipse type, such as the example run screen, is displayed in the Ellipse type area of QML.

The QQuickPaintItem class inherits from the QQuickItem class. The QQuickItem class is used as a parent class to implement all graphic display-related QML types provided by Qt Quick. The QQuickItem class can handle events such as keyboards, touches, and mice in addition to those related to graphic display.

### **● Example using the QQuickPaintedItem class**

In this example, let's try to implement the example we saw in the example above. In the QQuickPaintItem class, you can display the results of drawing an ellipse, a 2D graphic element, in the QML area.



<FIGURE> Example Execution Screen

The Qt C++ class QQuickPaintItem class will be used to draw an ellipse, as shown in the example run screen above.

<Example> Ch05 > 03 > 01\_QQuickPaintedItem\_Example > ellipseitem.h

```
#ifndef ELLIPSEITEM_H
#define ELLIPSEITEM_H

#include <QQuickPaintedItem>

class EllipseItem : public QQuickPaintedItem
{
    Q_OBJECT
```

## Jesus loves you.

```
public:  
    EllipseItem(QQuickItem *parent = nullptr);  
    void paint(QPainter *painter);  
};  
  
#endif
```

As you can see in the example source code above, it is a member function that implements a source code that draws an ellipse in which the paint( ) function is a 2D graph element.

<Example> Ch05 > 03 > 01\_QQuickPaintedItem\_Example > ellipseItem.cpp

```
#include <QtGui>  
#include "ellipseitem.h"  
#include <QDebug>  
  
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);  
}
```

<Example> Ch05 > 03 > 01\_QQuickPaintedItem\_Example > ellipse.qml

```
import QtQuick 2.12  
import Shapes 1.0  
  
Item {  
    width: 300; height: 200
```

## **Jesus loves you.**

```
Ellipse {  
    x: 50; y: 50  
    width: 200; height: 100  
}  
}
```

The following example source code is the main.cpp source code. Main.cpp provides the ability to connect EllipseItem classes from QML to Ellipse QML type using qmlRegisterType( ).

<Example> Ch05 > 03 > 01\_QQuickPaintedItem\_Example > main.cpp

```
#include <QGuiApplication>  
#include <QQuickView>  
#include "ellipseitem.h"  
  
int main(int argc, char *argv[]){  
    QGuiApplication app(argc, argv);  
    qmlRegisterType<EllipseItem>("Shapes", 1, 0, "Ellipse");  
  
    QQuickView view;  
    view.setSource(QUrl("qrc:///ellipse.qml"));  
    view.show();  
  
    return app.exec();  
}
```

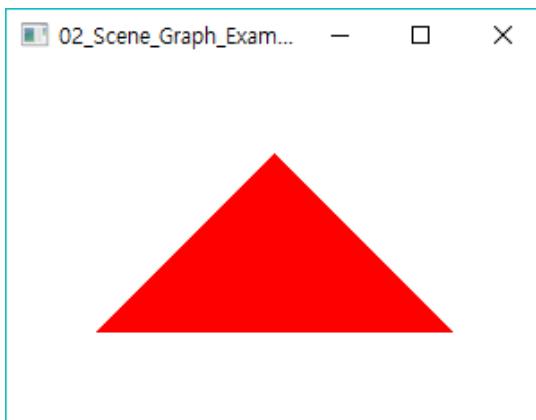
## 5.4. Scene Graph

Scene Graph provides the same functionality as QPainter provided by Qt C++. To use Scene Graph, you can use the QSGNode class that is implemented by inheriting the QQuickItem class. This class provides the same functionality as the QGraphicsItem class provided by the Qt Graphics View Framework.

The Scene Graph class uses OpenGL ES or OpenGL by default. In addition, the QSGNode class offers classes such as QSGGeometry, QSGM Material (Texture), etc.

### ● Triangle Example Using Scene Graph Classes

In this example, use the QSGNode, QSGGeometry, and QSGFlatColorMaterial classes to illustrate the 2D graph element Triangle.



<FIGURE> Example Execution Screen

The QSGNode class is a class that manages all graph units, Node, in Scene Graph. QSGGeometry class is a class that stores geometry information to be rendered on Scene Graph. The QSGFlatColorMaterial class is a class used to express Color within Scene Graph.

<Example> Ch05 > 04 > 01\_Scene\_Graph\_Example > triangleitem.h

```
#ifndef TRIANGLEITEM_H
#define TRIANGLEITEM_H

#include <QQuickItem>
#include <QSGGeometry>
```

## Jesus loves you.

```
#include <QSGFlatColorMaterial>

class TriangleItem : public QQuickItem
{
    Q_OBJECT

public:
    TriangleItem(QQuickItem *parent = 0);

protected:
    QSGNode *updatePaintNode(QSGNode *node, UpdatePaintNodeData *data);

private:
    QSGGeometry m_geometry;
    QSGFlatColorMaterial m_material;
};

#endif
```

<Example> Ch05 > 04 > 01\_Scene\_Graph\_Example > triangleitem.cpp

```
#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) {
        node = new QSGGeometryNode();
    }

    QSGGeometry::Point2D *v = m_geometry.vertexDataAsPoint2D();
```

## Jesus loves you.

```
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;
}
```

The ItemHasContents ENUM value specified in the setFlag() function is the Flag ENUM value provided by QQuickItem. The m\_material object of the QSGFlatColorMaterial class included in Scene Graph can be colour coded using the setColor( ) member function. Next is the QML source code.

<Example> Ch05 > 04 > 01\_Scene\_Graph\_Example > triangletest.qml

```
import QtQuick 2.12
import Shapes 1.0

Item
{
    width: 300; height: 200
    Triangle {
        x: 50; y: 50
        width: 200; height: 100
    }
}
```

Rendering results from the Triangle QML type, as shown in the QML example source code above, rendered in the TriangleItem class. Next is the main.cpp source code.

```
#include <QGuiApplication>
#include <QQQuickView>
```

# **Jesus loves you.**

```
#include "triangleitem.h"

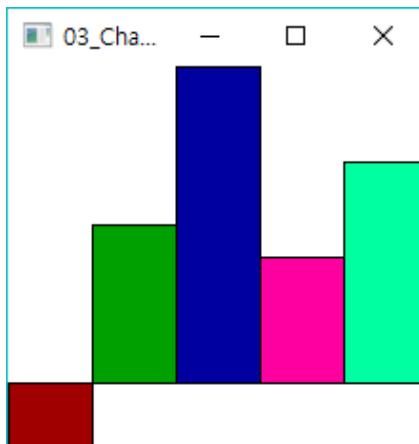
int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);
    qmlRegisterType<TriangleItem>("Shapes", 1, 0, "Triangle");

    QQuickView view;
    view.setSource(QUrl("qrc:///triangletest.qml"));
    view.show();

    return app.exec();
}
```

- **Example of implementing a bar chart**

In this example, let's try to implement a Bar chart graph as shown in the following figure.



<FIGURE> Example Execution Screen

Two QML types have been implemented to display a bar chart as shown in the figure above. The Chart type acts like a Container type. Provides a Container function for displaying five bar graphs within a single chart, for example, as shown in the figure above. And the bar type is an element of the actual bar graph. For example, as shown in the figure above, five bar types were used because they were five bars.

<Example> Ch05 > 04 > 02\_ChartItem\_Example > chart.qml

```
import QtQuick 2.12
```

## Jesus loves you.

```
import Shapes 1.0

Chart {
    width: 220; height: 200
    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 }
    ]
}
```

As shown in the example above, Chart type is implemented as ChartItem class and Bar type as BarItem class. BarItem defines the properties of a bar graph, and drawing actual bar graph elements is done in the ChartItem class.

<Example> Ch05 > 04 > 02\_ChartItem\_Example > main.cpp

```
#include <QGuiApplication>
#include <QQuickView>
#include "baritem.h"
#include "chartitem.h"

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

    qmlRegisterType<ChartItem>("Shapes", 1, 0, "Chart");
    qmlRegisterType<BarItem>("Shapes", 1, 0, "Bar");

    QQuickView view;
    view.setSource(QUrl("qrc:///chart.qml"));
    view.show();

    return app.exec();
}
```

<Example> Ch05 > 04 > 02\_ChartItem\_Example > chartitem.h

## Jesus loves you.

```
#ifndef CHARTITEM_H
#define CHARTITEM_H

#include <QQuickPaintedItem>

class BarItem;
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
```

<Example> Ch05 > 04 > 02\_ChartItem\_Example > chartitem.cpp

```
#include <QtGui>
#include "baritem.h"
#include "chartitem.h"

ChartItem::ChartItem(QQuickItem *parent) : QQuickPaintedItem(parent)
{
}

void ChartItem::paint(QPainter *painter)
{
    if (m_bars.count() == 0)
        return;
```

## Jesus loves you.

```
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());
    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();

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)
```

## Jesus loves you.

```
{  
    ChartItem *chart = qobject_cast<ChartItem *>(list->object);  
    if (chart) {  
        bar->setParent(chart);  
        chart->m_bars.append(bar);  
        chart->barsChanged();  
    }  
}
```

<Example> Ch05 > 04 > 02\_ChartItem\_Example > baritem.h

```
#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
```

## Jesus loves you.

<Example> Ch05 > 04 > 02\_ChartItem\_Example > baritem.cpp

```
#include "baritem.h"

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();
    }
}
```

## 5.5. Mapping Data Variables with Interaction between C++ and QML

In this section, let's learn about the interaction and variable mapping between Qt C++ and QML. The interaction refers to the handling of the type of QML at C++, the access or setting of the property value of the QML type, and the connection of SIGNAL within QML. So let's talk about how Qt C++ and QML communicate data and communicate events.

<TABLE> The contents to be covered in this section

분류	다를 내용
Interaction between C++ and QML	Interaction between C++ and QML using QQuickView and QQmlComponent classes provided by C++
	How to access a Property or Object within the QML Type using C++
	How to WRITE/READ QML Type Properties Using C++
	To invoke a method () written in QML from C++
	How to use the connection() function in C++ to connect to the Signal in QML
Data exchange and variable mapping between C++ and QML	Data exchange and variable type mapping between C++ and QML
	Data exchange between JavaScript arrays used in QML and QVariantList used in C++
	Exchange data objects (or classes) between JavaScript arrays used in QML and QVariantList used in C++
	How to use the Q_ENUMS() macro to leverage the Enumeration Type

- ✓ Interaction between C++ and QML using QQuickView and QQmlComponent classes provided by C++

QQuickView and QQmlComponent classes provide the same functionality for interaction with QML. Use for the same purpose, with slight differences in the method used. The following is how to interact between QML using the QQuickView class.

## **Jesus loves you.**

```
...
QQuickView view;

view.setSource(QUrl::fromLocalFile("MyItem.qml"));
view.show();

QObject *object = view.rootObject();
...
```

The following is an example source code for interacting with QML using the QQmlComponent class.

```
...
QQmlEngine engine;

QQmlComponent component(&engine, QUrl::fromLocalFile("MyItem.qml"));
QObject *object = component.create();
...
```

- ✓ How to access a Property or Object within the QML Type using C++

To change the QML-type profferty using the QObject used in the example above, the following setProperty( ) member function of the QObject class or QQmlProperty can be used.

```
object->setProperty("width", 500);
QQmlProperty(object, "width").write(500);
```

In addition to the above methods, objects in the QObject class can be transformed into the QQuickItem class to access QML-type properties as shown in the following example sources.

```
QQuickItem *item = qobject_cast<QQuickItem*>(object);
item->setWidth(500);
```

- ✓ How to WRITE/READ QML Type Properties Using C++

To access the QML type at C++, you can access the QML type's Properties using the findChild( ) member function of the QObject class. The following is the QML example source code.

```
import QtQuick 2.12
```

## **Jesus loves you.**

```
Item {  
    width: 100; height: 100  
  
    Rectangle {  
        anchors.fill: parent  
        objectName: "rect"  
    }  
}
```

For access to the Rectangle-type color Properties in the above QML example source code, you can change the color Property value at C++ as follows:

```
QObject *rect = object->findChild<QObject*>("rect");  
  
if (rect)  
    rect->setProperty("color", "red");
```

The following example source code is a method for referencing the value of a profferty of the QML type.

```
import QtQuick 2.12  
  
Item {  
    property int someNumber: 100  
}
```

The someNumber Properties above can be used as shown in the following C++ example source code, as a way to refer to C++.

```
QQmlEngine engine;  
  
QQmlComponent component(&engine, "MyItem.qml");  
QObject *object = component.create();  
  
qDebug() << "Property value:"  
    << QQmlProperty::read(object, "someNumber").toInt();  
  
QQmlProperty::write(object, "someNumber", 5000);  
  
qDebug() << "Property value:"
```

## Jesus loves you.

```
<< object->property("someNumber").toInt();  
  
object->setProperty("someNumber", 100);
```

- ✓ To invoke a method ( ) written in QML from C++

C++에서 QVariant 클래스를 이용해 QML에서 구현된 Method를 호출하기 위해서 QMetaObject 클래스에서 제공하는 invokeMethod( ) 멤버 함수를 이용해 호출할 수 있다. 다음은 QML에 구현된 Method 함수 예제 소스코드이다.

```
import QtQuick 2.12  
  
Item {  
    function myQmlFunction(msg) {  
        console.log("Got message:", msg)  
        return "some return value"  
    }  
}  
...
```

MyQmlFunction( ) implemented in QML above can be used as shown in the following example to call C++.

```
QQmlEngine engine;  
  
QQmlComponent component(&engine, "MyItem.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));  
  
qDebug() << "QML function returned:" << returnedValue.toString();  
...
```

- ✓ How to use the connection() function in C++ to connect to the Signal in QML

## Jesus loves you.

```
import QtQuick 2.12

Item {
    id: item
    width: 100; height: 100
    signal qmlSignal(string msg)

    MouseArea {
        anchors.fill: parent
        onClicked: {
            item.qmlSignal("Hello from QML")
        }
    }
}
```

The following example is an example source code for linking Signal within QML to the Slot function provided by C++ above.

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

public slots:
    void cppSlot(const QString &msg) {
        qDebug() << "Called the C++ slot with message: " << msg;
    }
};

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

    QQmlView view(QUrl::fromLocalFile("MyItem.qml"));
    QObject *item = view.rootObject();

    MyClass myClass;
    QObject::connect(item,           SIGNAL(qmlSignal(QString)), &myClass,
                     SLOT(cppSlot(QString)));
}
```

## Jesus loves you.

```
view.show();
return app.exec();
}
```

If the Argument value of the Signal function used in QML is to use the Variant type, the var type shall be used in QML as follows: The following is the QML example source code.

```
import QtQuick 2.12

Item {
    id: item
    width: 100; height: 100

    signal qmlSignal(var anObject)

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

In the QML above, the Argument of the Signal function was used as the var type. In order to map the variant type at C++, the QVariant class shall be used as follows:

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

public slots:
    void cppSlot(const QVariant &v)
    {
        QQuickItem *item = qobject_cast<QQuickItem*>(v.value<QObject*>());
        qDebug() << "Item size:" << item->width() << item->height();
    }
};

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

## Jesus loves you.

```
QQuickView view(QUrl::fromLocalFile("MyItem.qml"));
QObject *item = view.rootObject();
MyClass myClass;
QObject::connect(item, SIGNAL(qmlSignal(QVariant)),
                 &myClass, SLOT(cppSlot(QVariant)));
view.show();
return app.exec();
}
```

- ✓ Data exchange and variable type mapping between C++ and QML

Data can be passed through a Property, Method parameter, Method Return value, or Signal parameter value as a way to exchange data between C++ and QML. Basically, the process of converting to QML type in order to communicate the data type in C++ is automatically converted to a variable type that maps to what is seen in the following table.

<TABLE> 변수 타입 자동 매팅

Qt C++ variable	QML variable
bool	bool
unsigned int or 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

## Jesus loves you.

QVector2D	vector2d
QVector3D	vector3d
QVector4D	vector4d

Classes such as QColor, QFont, and QQuaterion used in Qt C++ can be used in QML as shown in the following example.

```
Item {  
    Image { sourceSize: "100x200" }  
    Image { sourceSize: Qt.size(100, 200) }  
}
```

- ✓ Exchange of data between JavaScript arrays in QML and QVariantList in C++

To convert data type from JavaScript Array and C++ to QVariantList used in QML, the QML engine automatically maps and converts the type. The following is the QML example source code.

```
...  
Item  
{  
    function readValues(anArray, anObject)  
    {  
        for (var i=0; i < anArray.length; i++)  
            console.log("Array item:", anArray[i])  
  
        for (var prop in anObject) {  
            console.log("Object item:",  
                      prop, "=", anObject[prop])  
        }  
    }  
}
```

To convert anArray object from C++ to QVariantList, as shown in the QML example source code above, you can use it as follows:

```
...  
QQuickView view(QUrl::fromLocalFile("MyItem.qml"));  
QVariantList list;
```

## Jesus loves you.

```
list << 10 << QColor(Qt::green) << "bottles";

QVariantMap map;
map.insert("language", "QML");
map.insert("released", QDate(2015, 9, 21));

QMetaObject::invokeMethod(view.rootObject(),
                         "readValues",
                         Q_ARG(QVariant, QVariant::fromValue(list)),
                         Q_ARG(QVariant, QVariant::fromValue(map)));
...

/* Result
Array item: 10
Array item: #00ff00
Array item: bottles
Object item: language = QML
Object item: released = Tue Sep 21 2021 00:00:00 GMT+1000 (EST)
*/
```

- ✓ **Exchange data objects (or classes) between JavaScript Array and QVariantList used by C++ in QML**

The conversion to the Date class of JavaScript used in QML and the QDateTime used in C++ can use the QVariant class as follows: The following is the QML example source code.

```
...
Item
{
    function readDate(dt)
    {
        console.log("The given date is:" , dt.toUTCString());
        return new Date();
    }
}
```

To use the above date classes in C++, you can use the QDateTime class as shown in the following example.

```
QQQuickView view(QUrl::fromLocalFile("MyItem.qml"));
```

## Jesus loves you.

```
QDateTime dateTime = QDateTime::currentDateTime();

QDateTime retValue;
QMetaObject::invokeMethod(view.rootObject(), "readDate",
                           Q_RETURN_ARG(QVariant, retValue),
                           Q_ARG(QVariant, QVariant::fromValue(dateTime)) );

qDebug() << "Value returned from readDate(): " << retValue;
```

Sequence-type array data such as JavaScript arrays using QML can be used in C++ with classes such as QList, or QStringList.

### ✓ How to use the Q\_ENUMS( ) macro to leverage the Enumeration Type

In order to use C++ ENUM values for the properties within the QML type, the Q\_ENUMS( ) macro provided by the Qt Meta-object system can be used. The following example is the QML example source code.

```
...
Message {
    onStatusChanged: {
        if (status == Message.Ready)
            console.log("Message is loaded!")
    }
}
...
```

In the QML example source code above, Message.Ready is the ENUM value provided by C++. To use the Enum value defined in Qt C++, the Q\_ENUMS( ) macro must be used as seen in the following example source code.

```
...
class Message : public QObject
{
    Q_OBJECT
    Q_ENUMS(Status)
    Q_PROPERTY(Status status READ status NOTIFY statusChanged)

public:
```

## Jesus loves you.

```
enum Status {
    Ready,
    Loading,
    Error
};

Status status() const;

signals:
    void statusChanged();
};
```

### ✓ Example of data exchange between C++ and QML using Model and View

In this example, let's learn how to use the values of the QStringList class in Qt C++ in QML.

<Example> Ch05 > 05 > 01\_stringlistmodel > main.cpp

```
#include <QGuiApplication>
#include <QStringList>
#include <qqmlengine.h>
#include <qqmlcontext.h>
#include <qqml.h>
#include <QtQuick/qquickitem.h>
#include <QtQuick/qquickview.h>

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");

    QQuickView view;

    QQmlContext *ctxt = view.rootContext();
    ctxt->setContextProperty("MyModel", QVariant::fromValue(dataList));
```

## Jesus loves you.

```
view.setSource(QUrl("qrc:view.qml"));
view.show();

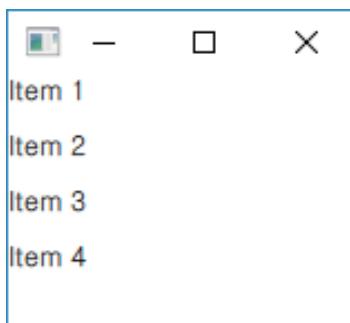
return app.exec();
}
```

As shown in the example above, we used the model/View provided by QML to use the data array stored in the QStringList class in QML. The following is an example of QML.

<Example> Ch05 > 05 > 01\_stringlistmodel > view.qml

```
import QtQuick 2.12

ListView {
    width: 100; height: 100
    model: MyModel
    delegate: Rectangle {
        height: 25; width: 100
        Text { text: modelData }
    }
}
```



<FIGURE> Example Execution Screen

### ✓ Exchange of C++ classes using Model and View

In the previous example, string data was passed to the QML in the QList class. In this example, let's deal with an example of passing classes to QML. First, let's write a C++ class that we want to pass on to QML.

<Example> Ch05 > 05 > 02\_objectlistmodel > dataobject.h

```
#ifndef DATAOBJECT_H
#define DATAOBJECT_H
```

## Jesus loves you.

```
#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:
    DataObject(QObject *parent=0);
    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
```

<Example> Ch05 > 05 > 02\_objectlistmodel > dataobject.cpp

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

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

DataObject::DataObject( const QString &name, const QString &color, QObject *parent )
    : QObject(parent), m_name(name), m_color(color)
{
}

QString DataObject::name() const {
```

## **Jesus loves you.**

```
    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();
    }
}
```

The color( ) member function of the DataObject class returns the m\_color value stored in this class from QML. Therefore, the member function of the DataObject class can be used in QML.

<Example> Ch05 > 05 > 02\_objectlistmodel > view.qml

```
import QtQuick 2.12

ListView {
    width: 100; height: 100
    model: MyModel
    delegate: Rectangle {
        height: 25; width: 100
        color: model.modelData.color
        Text { text: name }
    }
}
```

Next, let's learn how to save objects in the DataObject class to the QList and forward values to the QML using the QQmlContext class.

## Jesus loves you.

<Example> Ch05 > 05 > 02\_objectlistmodel > main.cpp

```
#include <QGuiApplication>
#include <qqmlengine.h>
#include <qqmlcontext.h>
#include <qqml.h>
#include <QtQuick/qquickitem.h>
#include <QtQuick/qquickview.h>
#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"));

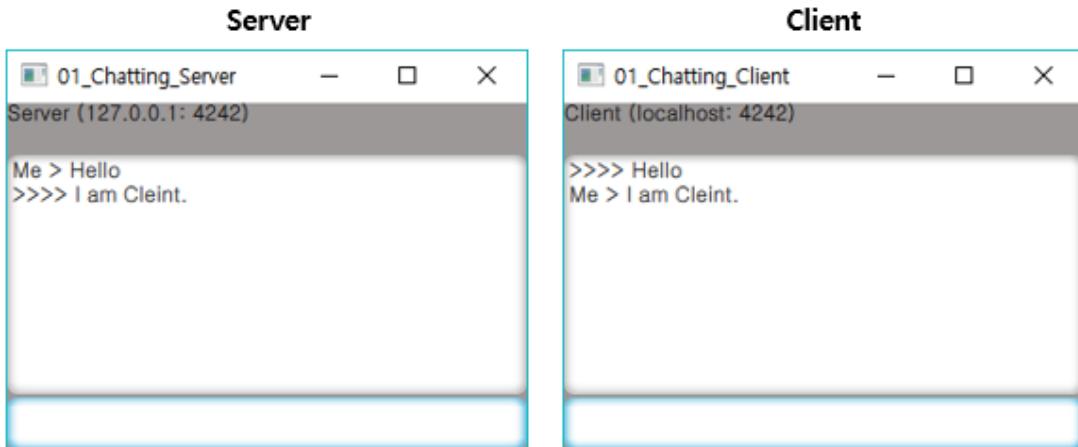
    QQuickView view;
    view.setResizeMode(QQuickView::SizeRootObjectToView);
    QQmlContext *ctxt = view.rootContext();
    ctxt->setContextProperty("MyModel", QVariant::fromValue(dataList));

    view.setSource(QUrl("qrc:view.qml"));
    view.show();

    return app.exec();
}
```

## 5.6. Implement TCP protocol-based chat applications

In this section, let's develop a TCP protocol-based chat application using C++ and QML. Examples of applications you implement are divided into Server and Client.



<FIGURE> Server and Client Example Run screen

Two examples of Server and Client consist of the following source code files:

<TABLE> Server Source Code Files

Kind	File name	Description
C++	main.cpp	Program Start Point
	tcpconnection.h tcpconnection.cpp	TcpConnection class enables QML to send or receive messages over the network
QML	Server.qml	Call and set QML type implemented in ChatWindow.qml
	ChatWindow.qml	Configure the GUI with QML. Configure the screen for entering the message in the Title, chat content window, and bottom, as you can see on the example screen of the execution above.
	Display.qml	Implement chat content window QML type
	LineEdit.qml	Macy Input QML Type Implementation

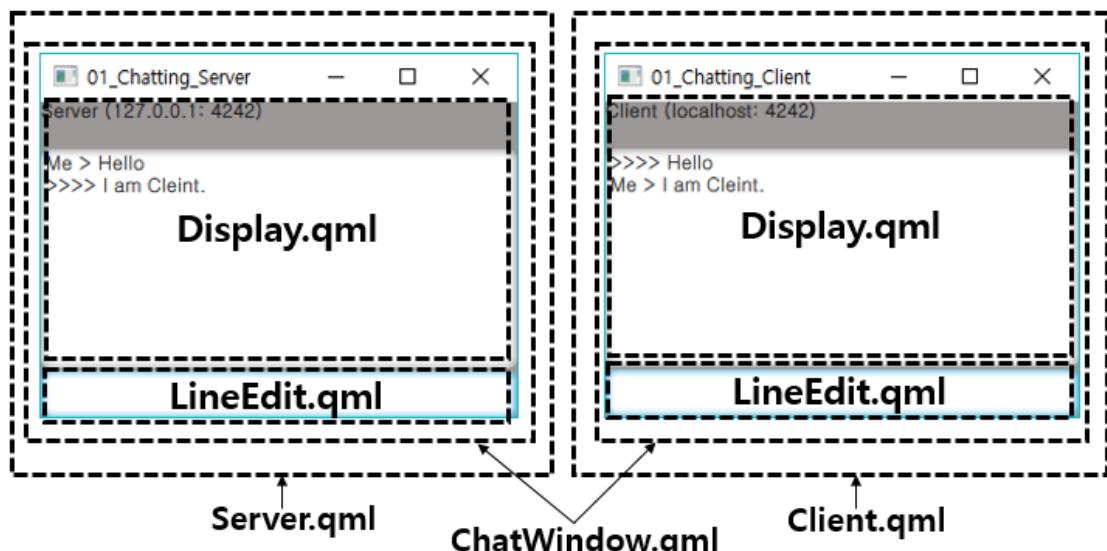
# Jesus loves you.

<TABLE> Client Source Code Files

Kind	File name	Description
C++	main.cpp	Program Start Point
	tcpconnection.h tcpconnection.cpp	TcpConnection class enables QML to send or receive messages over the network
QML	Client.qml	Call and set QML type implemented in ChatWindow.qml
	ChatWindow.qml	Configure the screen for entering messages in the Title, chat content window, and bottom, as you can see on the Run Example screen above the configuration in QML.
	Display.qml	Implement chat content window QML type
	LineEdit.qml	Macy Input QML Type Implementation

The two examples, Server and Client, use the same TcpConnection class. And there is no difference except that the server example uses Server.qml and the client example uses Client.qml.

Therefore, only Server.qml and Client.qml are different, and both Server and Client have the same source code for ChatWindow.qml, Display.qml, and LineEdit.qml. Then call Server.qml from the main.cpp in the Server example and Client.qml in the Client example.



## Jesus loves you.

<FIGURE> Server and Client source files

<Example> Ch05 > 06 > 01\_Chatting\_Server > main.cpp

```
#include <QApplication>
#include <QQQuickView>
#include <QUrl>
#include <q.qml.h>
#include "tcpconnection.h"

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

    qmlRegisterType<TcpConnection>("TCP", 1, 0, "TcpConnection");
    QQQuickView view;
    view.setSource(QUrl("qrc:///Server.qml"));
    view.show();

    return app.exec();
}
```

As shown in the example above, the Server example calls Server.qml. In the Client example, the Client.qml is called as part of the example source code below.

```
...
view.setSource(QUrl("qrc:///Client.qml"));
...
```

The TcpConnection class is a class that implements the TcpConnection QML type in QML. In this class, the QTcpServer class, for servers, is used to implement the ability to send/receive messages needed for TCP protocol-based network chat.

In the case of the Client, the QTcpSocket class enables the sending/receiving of messages required for TCP protocol-based network chat. Therefore, the TcpConnection class provides network setup in QML and the ability of the server and the Client to send/receive messages. The following example is the header tcpconnection.h source code for the TcpConnection class.

<Example> Ch05 > 06 > 01\_Chatting\_Server > tcpconnection.h

```
#ifndef TCPCONNECTIONELEMENT_H
```

## Jesus loves you.

```
#define TCPCONNECTIONELEMENT_H

#include <QObject>

class QTcpServer;
class QTcpSocket;
class TcpConnection : public QObject
{
    Q_OBJECT

    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_ENUMS(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();
```

## **Jesus loves you.**

```
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
```

<Example> Ch05 > 06 > 01\_Chatting\_Server > tcpconnection.cpp

```
#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;
}
```

## Jesus loves you.

```
void TcpConnection::setPort(int port)
{
    if (m_port != port) {
        m_port = port;
        emit portChanged();
    }
}

QString TcpConnection::hostName() const
{
    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;
```

## Jesus loves you.

```
m_tcpServer->listen( QHostAddress(mHostName), m_port );
connect( m_tcpServer, SIGNAL( newConnection() ),
          this, SLOT( slotConnection() ) );
}

else {
    m_tcpSocket = new QTcpSocket(this);
    m_tcpSocket->connectToHost( mHostName, m_port );
    connect( m_tcpSocket, SIGNAL(readyRead()), this, SLOT(receivedData()) );
}
}

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 );
}
```

The sendData( ) member function provides the ability to send messages. port( ) returns the current network port number. setPort( ) can set the network port number.

The initialize( ) member function uses the QTcpServer class to implement functions for server functions if the server is a server. In this case, use the QTcpClient class.

The slotConnection( ) Slot function is a slot function called when a new contact event occurs. This Slot function handles readyRead( ) signal events using the connection( ) function so that the receivedData( ) Slot function can be invoked when a new user receives a message. ReadyRead( ) is the signal that occurs when a message is received from the network. Let's take a look at the example of Server.qml used by Server.

<Example> Ch05 > 06 > 01\_Chatting\_Server > Server.qml

```
import QtQuick 2.12
import TCP 1.0

ChatWindow {
```

## Jesus loves you.

```
width: 300; height: 200  
type : TcpConnection.Server  
port : 4242  
}
```

In the above server.qml , the type property value is TcpConnection.Server was used.  
TcpConnection in Client.qmlUse the Client.

<Example> Ch05 > 06 > 01\_Chatting\_Server > ChatWindow.qml

```
import QtQuick 2.12  
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  
    }  
}
```

## Jesus loves you.

```
Display {
    id: output
    height:300
    anchors.bottomMargin: 2
    anchors {
        top : title.bottom
        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: {
        output.text += "Me > " + text + "\n"
        tcpConnection.sendData(text)
    }
}

function titleText() {
    return (tcpConnection.type == TcpConnection.Server ? "Server" : "Client")
        + " (" + hostName + ":" + port + ")"
}
}
```

ChatWindow.qml forms the screen of the GUI. Server and Client use the same source code.  
The following is an example source code for Display.qml.

<Example> Ch05 > 06 > 01\_Chatting\_Server > Display.qml

```
import QtQuick 2.12
```

## Jesus loves you.

```
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  
        verticalTileMode: BorderImage.Stretch  
        source: "./images/textoutput.png"  
    }  
  
    TextEdit {  
        id: output  
        anchors { margins: 3; fill: parent }  
        selectionColor: "transparent"  
    }  
}
```

The above Display.qml displays chat content. For example, all messages that other remote users send or I send messages are output. The following is an example source code for LineEdit.qml, which provides the function of the message entry window.

<Example> Ch05 > 06 > 01\_Chatting\_Server > LineEdit.qml

```
import QtQuick 2.12  
  
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 }  
    }  
}
```

## **Jesus loves you.**

```
horizontalTileMode: BorderImage.Stretch
verticalTileMode: BorderImage.Stretch
source: "./images/textinput.png"
}

TextInput {
    id : entryField
    anchors { margins: 3; fill: parent }
    focus: true

    Keys.onReturnPressed : {
        textEntered(text)
        text = ""
    }
}
}
```

## **6. Qt Quick Controls**

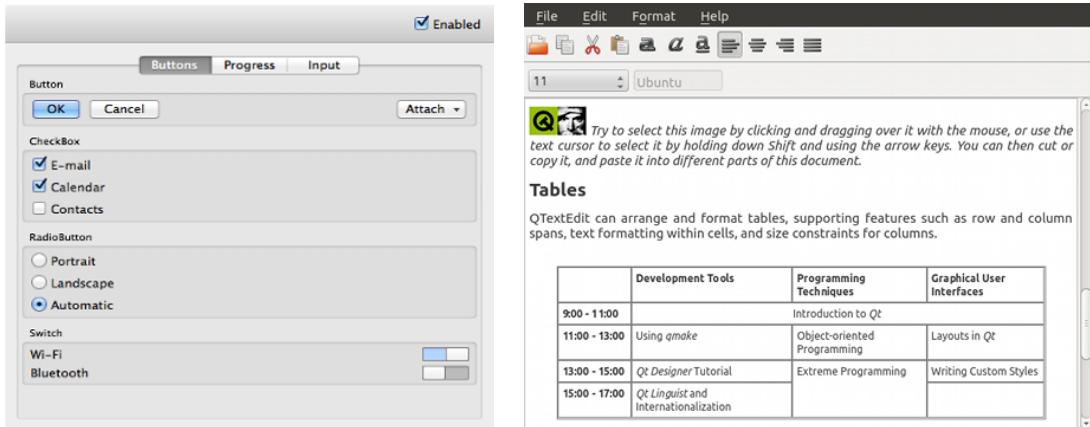
---

Qt Quick Controls provides a variety of UI Controls (Button, CheckBox, Tab, Combo, SpinBox, etc.) that make it easier to implement GUI in QML. The controls provided by Qt Quick provide version 1.x and version 2.x. Version 1.0 of Qt Quick Controls was released from version 5.1 of Qt.

Qt Quick Controls 2.0 version has been officially released from Qt 5.7. Therefore, to use Qt Quick Controls version 2.0 or higher, you must use Qt 5.7 or later. And from version Qt 5.13, version 1 of Qt Quick Controls is no longer included. Therefore, version 1 of Qt Quick Controls is available only in versions Qt 5.1 to 5.12.

## 6.1. Qt Quick Controls 1

Qt Quick Controls version 1.x has been officially released from version Qt 5.1. Therefore, version 1.x of Qt Quick Controls is not available under version 5.1 of Qt. Qt Quick Controls 1 provides UI Controls as shown below.



<FIGURE> Windows GUI Screen Implemented with Controls

To use version 1.x of Qt Quick Controls, the QML must import as follows:

```
import QtQuick.Controls 1.2
```

To use Qt Quick Controls, the highest QML type should be used as shown in the following example.

```
import QtQuick.Controls 1.2

ApplicationWindow {
    title: "My Application"; width: 640; height: 480; visible: true

    Button {
        text: "Push Me"
        anchors.centerIn: parent
    }
}
```

The Qt Quick Controls 1.x version provides various GUI QML types. The various QML types can be classified into four categories:

# Jesus loves you.

<TABLE> Qt Quick Controls 1 Classification

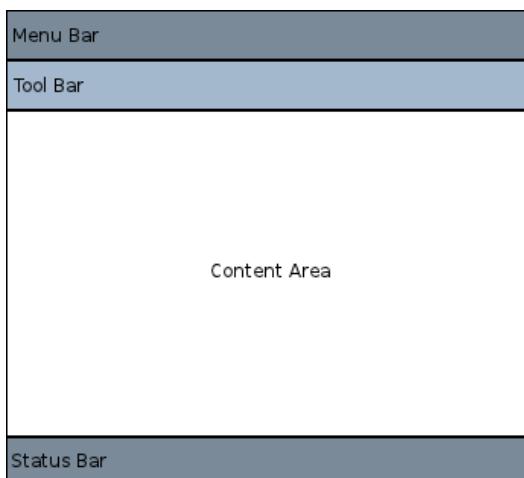
Classification	Description
ApplicationWindow	Main Window, Top Type
Navigation 과 Views	Types such as ScrollView, SplitView, TabView, etc.
Controls	Types such as Button, ComboBox, GroupBox, Tab, Table, etc.
Menus	Menu, MenuItem and MenuSeparator Type

## ✓ Application Window

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

<TABLE> Application Window

QML Type	Description
Action	Provide to link events when you click Menu
ApplicationWindow	Top Window of Application Window
MenuBar	Horizontal Orientation Menu
StatusBar	UI for displaying status information, features for placing at the bottom
ToolBar	A type placed at the bottom of a menu, such as ToolButton.



<FIGURE> Component Location on the GUI

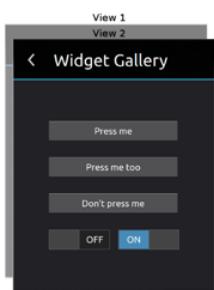
# Jesus loves you.

## ✓ Navigation 과 Views

<TABLE> Navigation 과 Views

QML타입	설명
ScrollView	View provides scrolling within the QML type
SplitView	Splitter between QML types to provide drag functionality
StackView	Features used when configuring UI with multiple screens, such as Stack
TabView	UI that provides tab functionality
TableView	Item providing list UI function in TABLE form
TreeView	Items that provide tree-shaped UI functions such as directory structure

StackView



TabView



TableView

Title	Author
A Masterpiece	Gabriel
Brilliance	Jens
Outstanding	Frederik

TreeView



<FIGURE> Navigation and Views

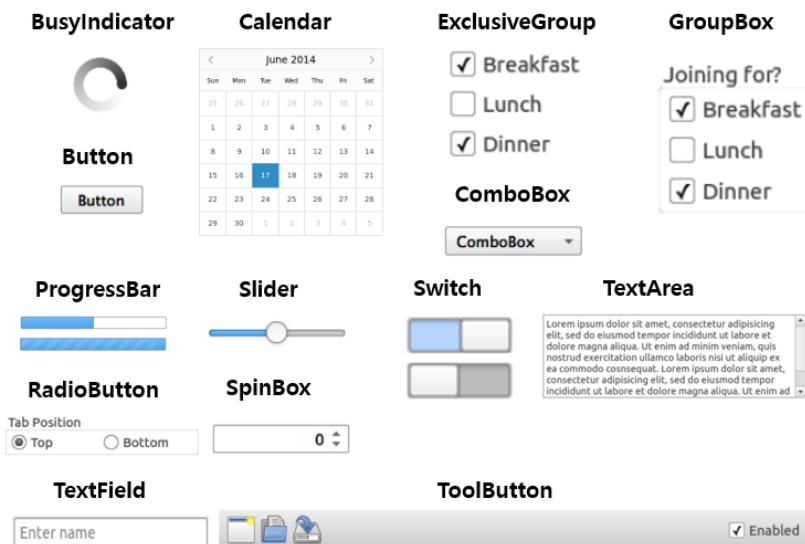
## ✓ Controls

<TABLE> Controls

QML타입	설명
BusyIndicator	A temporary wait control, such as displaying a loading image.
Button	button containing text
Calendar	Function to select a date to show that calendar
CheckBox	check box containing text
ComboBox	Control showing a list of Drop-Down styles
ExclusiveGroup	UI that allows you to check one or more of multiple items

# Jesus loves you.

GroupBox	Provides group frame UI functionality
Label	Provides text display UI functionality
ProgressBar	UI for displaying progress
RadioButton	A radio button that allows you to select one of several.
Slider	Horizontal or Vertical Slider Function UI
SpinBox	UI with up/down buttons to select one of the items
TextArea	UI that allows you to display TEXT of multiple lines, such as text boxes.
TextField	Provides Single Line UI function for editing textures
ToolButton	UI provided to place multiple images as buttons



<FIGURE> Controls UI

## ✓ Menus

<TABLE> Menus

QML	Description
Menu	popup menu, context menu
MenuItem	Items for adding to a menu or menu bar
MenuSeparator	Distinguish between menu items

## Jesus loves you.

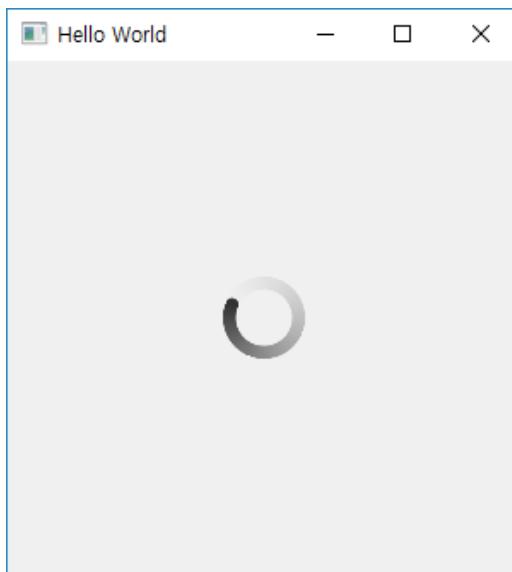
### ✓ **BusyIndicator**

<Example> Ch06 > 01 > 01\_Controls > busyindicator.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.4
import QtQuick.Window 2.2

ApplicationWindow {
    title: qsTr("Hello World"); width: 300; height: 300; visible: true

    BusyIndicator {
        anchors.centerIn: parent
        running: true
    }
}
```



<FIGURE> Example Execution Screen

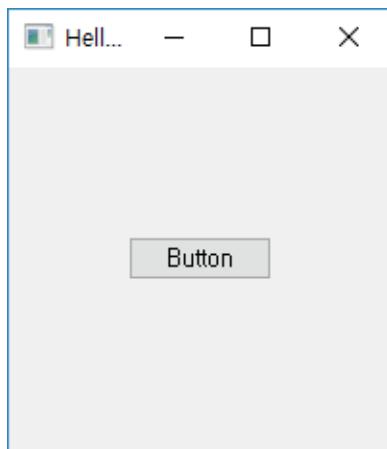
### ✓ **Button**

<Example> Ch06 > 01 > 01\_Controls > button.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.4
import QtQuick.Window 2.2
```

## Jesus loves you.

```
ApplicationWindow {  
    title: qsTr("Hello World")  
    width: 200; height: 200; visible: true  
  
    Button {  
        anchors.centerIn: parent  
        text: "Button"  
        tooltip: "Description"  
    }  
}
```



<FIGURE> Example Execution Screen

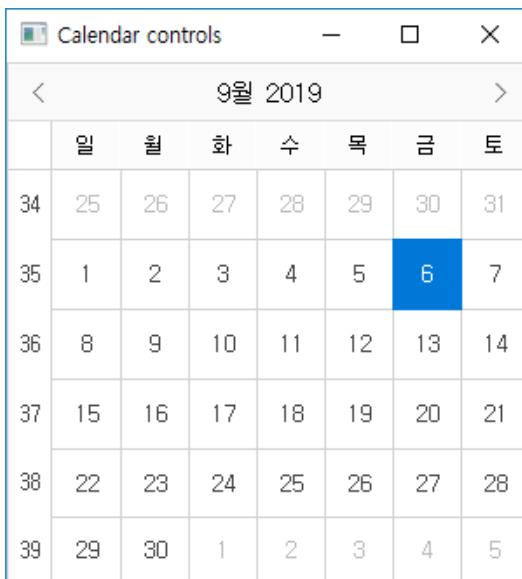
### ✓ Calendar

<Example> Ch06 > 01 > 01\_Controls > calendar.qml

```
import QtQuick 2.4  
import QtQuick.Controls 1.4  
import QtQuick.Window 2.2  
  
ApplicationWindow {  
    title: qsTr("Calendar controls")  
    width: 300; height: 300; visible: true  
    Calendar {  
        width: parent.width  
        height: parent.height  
        weekNumbersVisible: true
```

## Jesus loves you.

```
}
```



<FIGURE> Example Execution Screen

### ✓ CheckBox

<Example> Ch06 > 01 > 01\_Controls > checkbox.qml

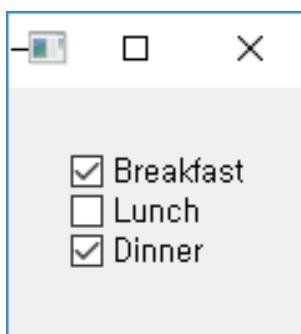
```
import QtQuick 2.4
import QtQuick.Controls 1.4
import QtQuick.Window 2.2

ApplicationWindow
{
    title: qsTr("Checkbox controls"); width: 100; height: 100; visible: true
    Column {
        anchors.centerIn: parent

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

## Jesus loves you.

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



<FIGURE> Example Execution Screen

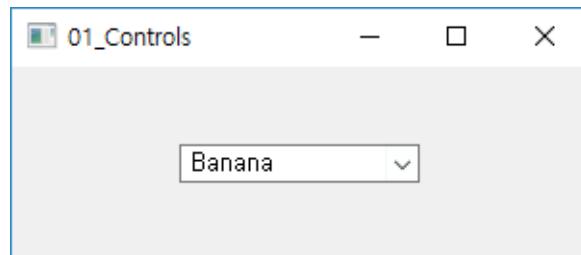
### ✓ ComboBox

<Example> Ch06 > 01 > 01\_Controls > combobox.qml

```
import QtQuick 2.4  
import QtQuick.Controls 1.4  
import QtQuick.Window 2.2  
  
ApplicationWindow {  
    width: 300; height: 100; visible: true  
    ComboBox {  
        anchors.centerIn: parent  
        editable: true  
        model: ListModel {  
            id: model  
            ListElement { text: "Banana"; color: "Yellow" }  
            ListElement { text: "Apple"; color: "Green" }  
            ListElement { text: "Coconut"; color: "Brown" }  
        }  
        onAccepted: {  
            if (find(currentText) === -1) {  
                model.append({text: editText})  
                currentIndex = find(editText)  
            }  
        }  
    }  
}
```

## Jesus loves you.

```
        }
    }
}
```



<FIGURE> Example Execution Screen

### ✓ **GroupBox**

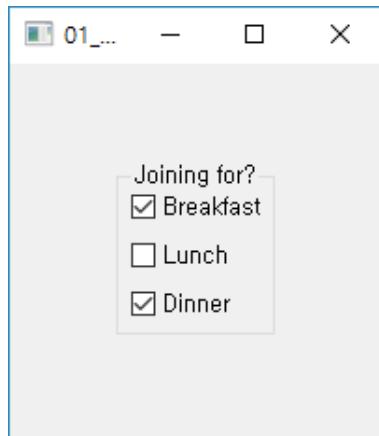
<Example> Ch06 > 01 > 01\_Controls > groupbox.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.4
import QtQuick.Window 2.2

ApplicationWindow
{
    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 }
        }
    }
}
```

## Jesus loves you.



<FIGURE> Example Execution Screen

### ✓ ExclusiveGroup

CheckBox has multiple choices. However, using ExclusiveGroup allows you to select only one CheckBox that belongs to the ExclusiveGroup like Radio.

<Example> Ch06 > 01 > 01\_Controls > exclusivegroup.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.4
import QtQuick.Window 2.2

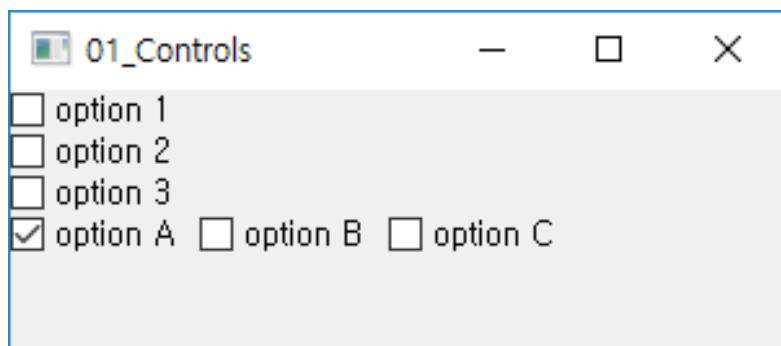
ApplicationWindow {
    width: 300; height: 100; visible: true

    ExclusiveGroup { id: exclusiveGroup1 }
    ExclusiveGroup { id: exclusiveGroup2 }

    Column{
        Column {
            CheckBox { text: "option 1"; exclusiveGroup: exclusiveGroup1 }
            CheckBox { text: "option 2"; exclusiveGroup: exclusiveGroup1 }
            CheckBox { text: "option 3"; exclusiveGroup: exclusiveGroup1 }
        }
        Row {
            spacing: 10
            CheckBox {
                text: "option A"; checked: true; exclusiveGroup: exclusiveGroup2
            }
        }
    }
}
```

## Jesus loves you.

```
CheckBox {  
    text: "option B"; exclusiveGroup: exclusiveGroup2  
}  
CheckBox {  
    text: "option C"; exclusiveGroup: exclusiveGroup2  
}  
}  
}  
}
```



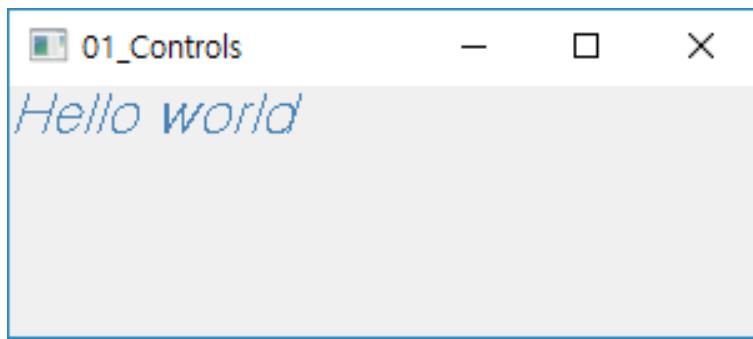
<FIGURE> Example Execution Screen

### ✓ Label

<Example> Ch06 > 01 > 01\_Controls > label.qml

```
import QtQuick 2.4  
import QtQuick.Controls 1.4  
import QtQuick.Window 2.2  
  
ApplicationWindow {  
    width: 300; height: 100; visible: true  
  
    Label {  
        text: "Hello world"  
        font.pixelSize: 22  
        font.italic: true  
        color: "steelblue"  
    }  
}
```

**Jesus loves you.**



<FIGURE> Example Execution Screen

✓ **ProgressBar**

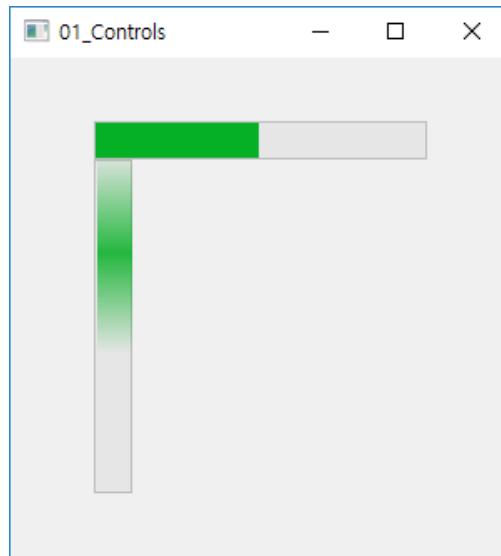
<Example> Ch06 > 01 > 01\_Controls > progressbar.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.4
import QtQuick.Window 2.2

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

    Column {
        anchors.centerIn: parent
        ProgressBar {
            value: 0.5
        }
        ProgressBar {
            indeterminate: true
            orientation: Qt.Vertical
        }
    }
}
```

## Jesus loves you.



<FIGURE> Example Execution Screen

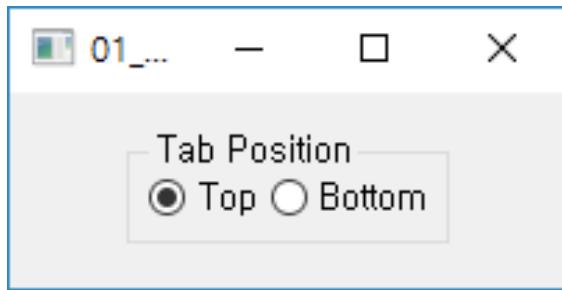
### ✓ RadioButton

<Example> Ch06 > 01 > 01\_Controls > radiobutton.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.4
import QtQuick.Window 2.2
import QtQuick.Layouts 1.1

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

## Jesus loves you.



<FIGURE> Example Execution Screen

### ✓ Slider

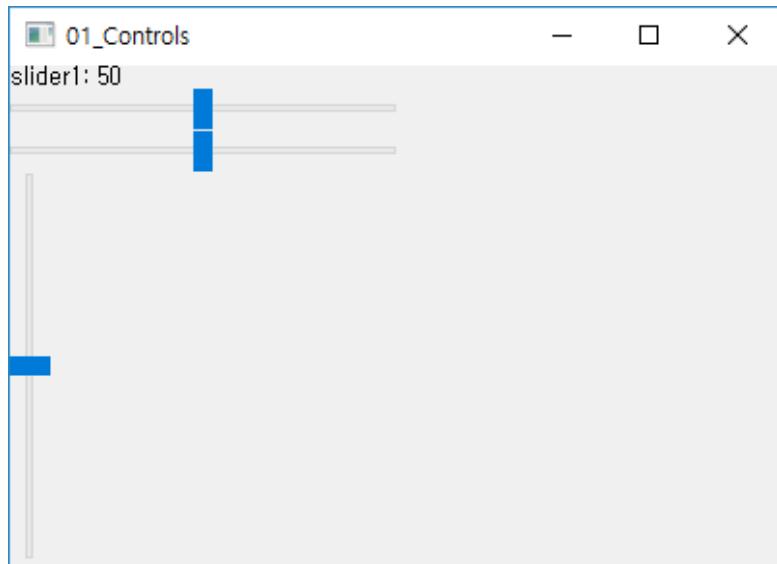
<Example> Ch06 > 01 > 01\_Controls > slider\_event.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.4
import QtQuick.Window 2.2

ApplicationWindow {
    width: 400; height: 350; visible: true

    Column{
        Label { id: label }
        Slider {
            id: slider1; maximumValue: 100; minimumValue: 0
            value: 50; onValueChanged: label.text = "slider1: " + value
        }
        Slider {
            id: slider2; maximumValue: 100; minimumValue: 0;
            value: 50; stepSize: 10
            onValueChanged: label.text = "slider2: " + value
        }
        Slider {
            id: slider3; maximumValue: 100; minimumValue: 0
            value: 50; stepSize: 25
            orientation: Qt.Vertical
            onValueChanged: label.text = "slider3: " + value
        }
    }
}
```

## Jesus loves you.



<FIGURE> Example Execution Screen

### ✓ **SpinBox**

<Example> Ch06 > 01 > 01\_Controls > spinbox\_event.qml

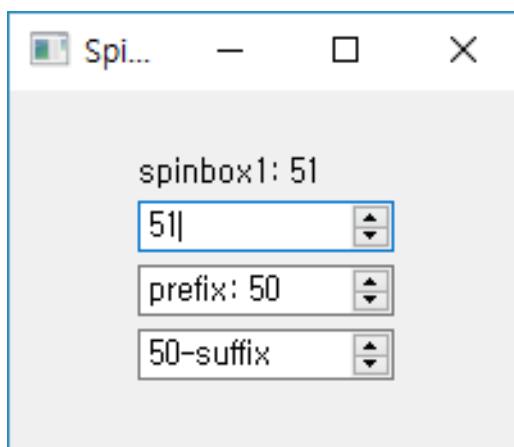
```
import QtQuick 2.4
import QtQuick.Controls 1.4
import QtQuick.Window 2.2

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

    Column{
        anchors.centerIn: parent
        spacing: 5
        Label {
            id: label
        }
        SpinBox{
            id: spinbox1; width: 100
            maximumValue: 100; minimumValue: 0; value: 50
            onValueChanged: label.text = "spinbox1: " + value
        }
        SpinBox{
            id: spinbox2; width: 100
        }
    }
}
```

## Jesus loves you.

```
maximumValue: 100; minimumValue: 0;  
value: 50; stepSize: 5; prefix: "prefix: "  
onValueChanged: label.text = "spinbox2: " + value  
}  
SpinBox{  
    id: spinbox3; width: 100;  
    maximumValue: 100; minimumValue: 0; value: 50; stepSize: 10  
    suffix: "-suffix"  
    onValueChanged: label.text = "spinbox3: " + value  
}  
}  
}
```



<FIGURE> Example Execution Screen

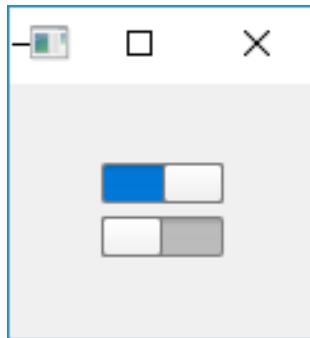
### ✓ Switch

<Example> Ch06 > 01 > 01\_Controls > switch.qml

```
import QtQuick 2.4  
import QtQuick.Controls 1.4  
import QtQuick.Window 2.2  
  
ApplicationWindow {  
    width: 100; height: 100  
    visible: true  
    Column {  
        anchors.centerIn: parent  
        spacing: 5
```

# Jesus loves you.

```
Switch {  
    id: first  
    checked: true  
    onClicked: {  
        if(checked === true)  
            console.log("first checked true");  
        else  
            console.log("first checked false")  
    }  
}  
Switch { checked: false }  
}  
}
```



<FIGURE> Example Execution Screen

## ✓ TextArea

<Example> Ch06 > 01 > 01\_Controls > textarea.qml

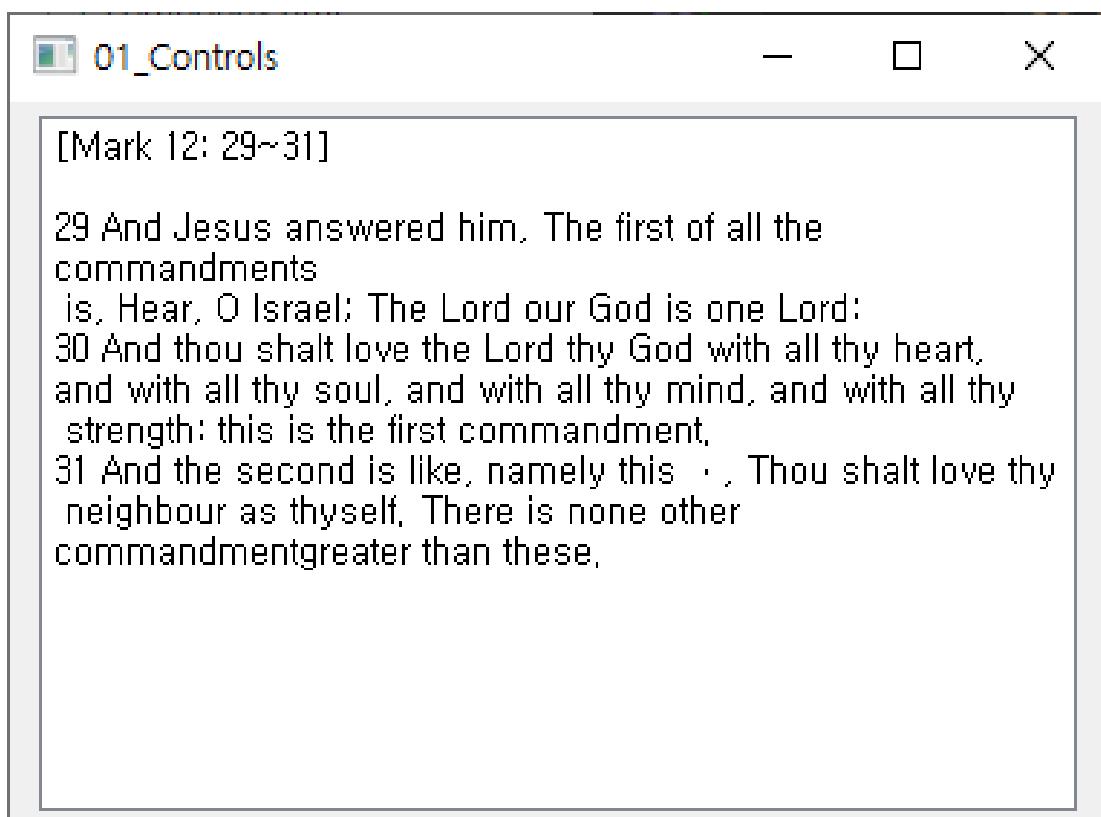
```
import QtQuick 2.4  
import QtQuick.Controls 1.4  
import QtQuick.Window 2.2  
  
ApplicationWindow {  
    width: 380  
    height: 250  
    visible: true  
  
    TextArea {  
        anchors.centerIn: parent
```

## Jesus loves you.

```
width: 360
height: 240

text:
    "[Mark 12: 29~31] \n\n" +
    "29 And Jesus answered him, The first of all the commandments \n" +
    " is, Hear, O Israel; The Lord our God is one Lord:\n" +
    "30 And thou shalt love the Lord thy God with all thy heart, \n" +
    "and with all thy soul, and with all thy mind, and with all thy\n" +
    " strength: this is the first commandment. \n" +
    "31 And the second is like, namely this · , Thou shalt love thy \n" +
    " neighbour as thyself. There is none other commandment" +
    "greater than these."
}
```

```
}
```



<FIGURE> TextArea Example Execution Screen

# Jesus loves you.

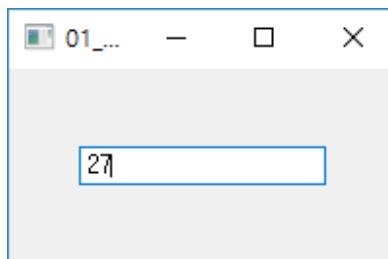
## ✓ **TextField**

<Example> Ch06 > 01 > 01\_Controls > textfield.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.4
import QtQuick.Window 2.2

ApplicationWindow {
    width: 200
    height: 100
    visible: true

    TextField {
        validator: IntValidator {bottom: 11; top: 31;}
        focus: true
        anchors.centerIn: parent
    }
}
```



<FIGURE> Example Execution Screen

## ✓ **ToolButton**

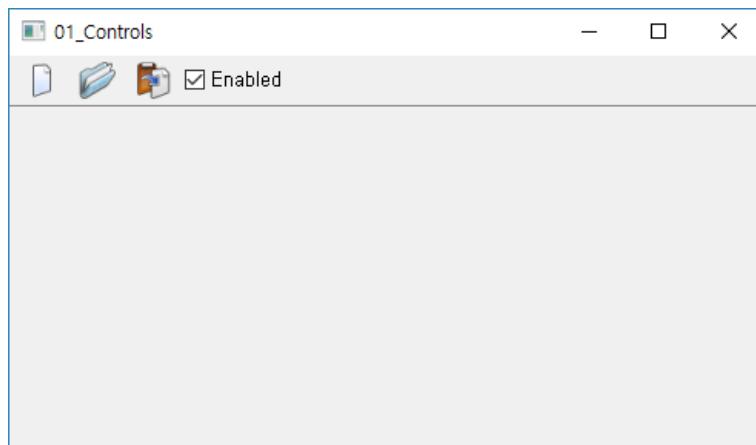
<Example> Ch06 > 01 > 01\_Controls > toolbutton.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.4
import QtQuick.Window 2.2
import QtQuick.Layouts 1.1

ApplicationWindow {
    width: 200; height: 100; visible: true
    ToolBar {
        RowLayout {
```

## Jesus loves you.

```
ToolBar { iconSource: "new.png" }
ToolBar { iconSource: "open.png" }
ToolBar { iconSource: "save-as.png" }
CheckBox {
    text: "Enabled"
    checked: true
}
}
```



<FIGURE> Example Execution Screen

### ✓ Example of Button and Action

An example of the handling of events that occur in Button in Action. Click the button on the example run screen to print the text on the StatusBar type.

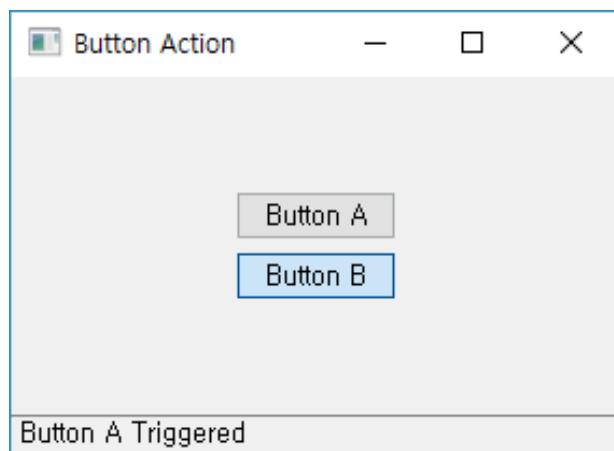
<Example> Ch06 > 01 > 02\_Button\_Action > main.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Window 2.2
import QtQuick.Layouts 1.1

ApplicationWindow {
    title: qsTr("Button Action")
    width: 380; height: 320; visible: true
```

## Jesus loves you.

```
ColumnLayout {  
    anchors.centerIn: parent  
    Button { action: actionButtonA }  
    Button { action: actionButtonB }  
}  
Action {  
    id: actionButtonA; text: "Button A"  
    onTriggered: statusBar.text = "Button A Triggered "  
}  
Action {  
    id: actionButtonB; text: "Button B"  
    checkable: true  
    onCheckedChanged:  
        statusBar.text = "Button B checked: " + checked  
}  
statusBar: StatusBar {  
    Label { id: statusBar; text: "Status Bar" }  
}  
}
```



<FIGURE> Example Execution Screen

### ✓ ApplicationWindow Example

In this example, a window GUI screen is implemented using ApplicationWindow, MenuBar,ToolBar and StatusBar types.

<Example> Ch06 > 01 > 03\_ApplicationWindow > main.qml

## **Jesus loves you.**

```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Window 2.2
import QtQuick.Layouts 1.1

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

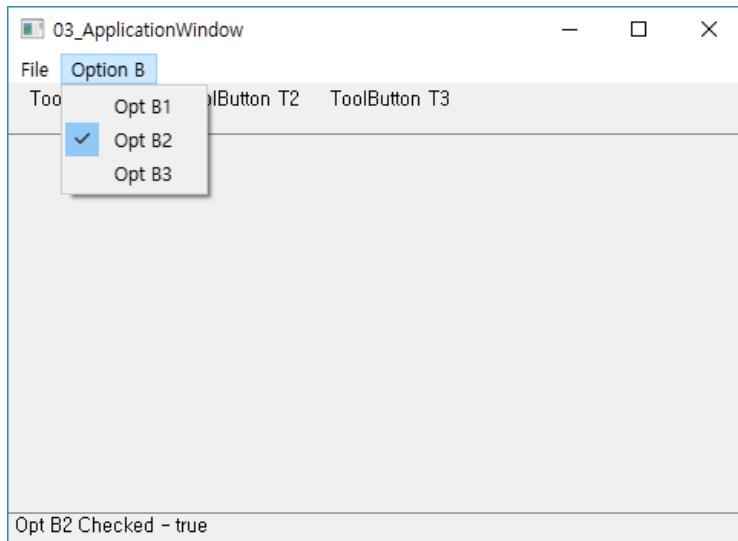
    menuBar: MenuBar {
        Menu {
            title: qsTr("File")
            MenuItem {
                text: qsTr("Exit")
                onTriggered: Qt.quit();
            }
            MenuItem {
                text: "File Item2"
                onTriggered: statusLabel.text = "File Item2 Click"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

## Jesus loves you.

```
toolBar: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"
        }
    }
}

statusBar:StatusBar {
    RowLayout {
        Label {
            id: statusLabel; text: "Status Bar"
        }
    }
}
```

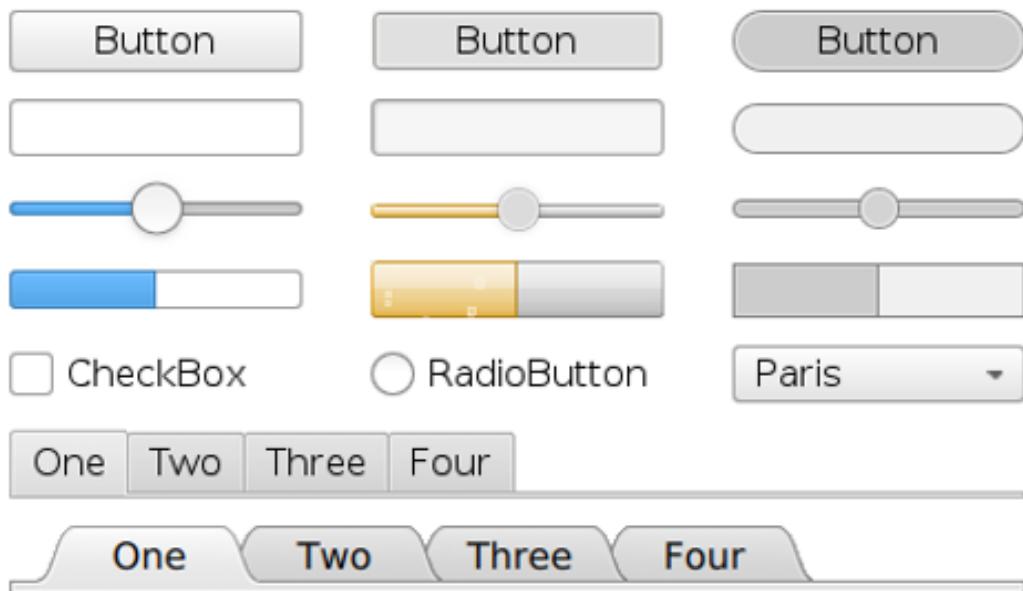
# Jesus loves you.



<FIGURE> Example Execution Screen

## 6.2. Qt Quick Controls 1 Styles QML Types

Other designs can be applied to Controls QML types using Controls Styles.



<FIGURE> Controls Styles

For example, you can apply different styles in addition to those that are provided by default, such as Button, as shown in the figure above.

**Tumbler basic style**



**Tumbler Flat style**

23	58
24	59
01	00
02	01
03	02

<FIGURE> Flat Style

The above figure is an example of the application of the Flat Style to the Tumbler QML type. To apply Style to the Controls QML type, the Import statement must be used as follows.

## Jesus loves you.

```
import QtQuick.Controls.Styles 1.4
```

The following example is an example source code with style applied to the Button QML Controls type.

<Example> Ch06 > 02 > 01\_Style > buttonstyle.qml

```
import QtQuick 2.12
import QtQuick.Controls 1.4
import QtQuick.Controls.Styles 1.4
import QtQuick.Layouts 1.1

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

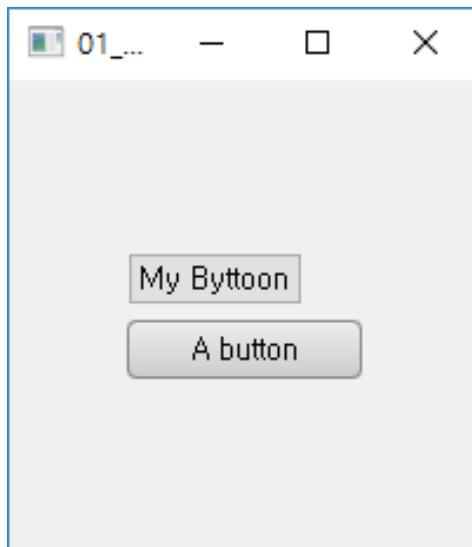
    ColumnLayout {
        anchors.centerIn: parent; spacing: 6

        Button { text: "My Byttoon" }
        Button {
            text: "A button"
            style: ButtonStyle {
                background: Rectangle {
                    implicitWidth: 100
                    implicitHeight: 25
                    border.width: control.activeFocus ? 2 : 1
                    border.color: "#888"
                    radius: 4

                    gradient: Gradient {
                        GradientStop {
                            position: 0
                            color: control.pressed ? "#ccc" : "#eee"
                        }
                        GradientStop {
                            position: 1
                            color: control.pressed ? "#aaa" : "#ccc"
                        }
                    }
                }
            }
        }
    }
}
```

## Jesus loves you.

```
        }
    }
}
```



<FIGURE> Example Execution Screen

The following example applies TabViewStyle to TabView.

<Example> Ch06 > 02 > 01\_Style > tabviewstyle.qml

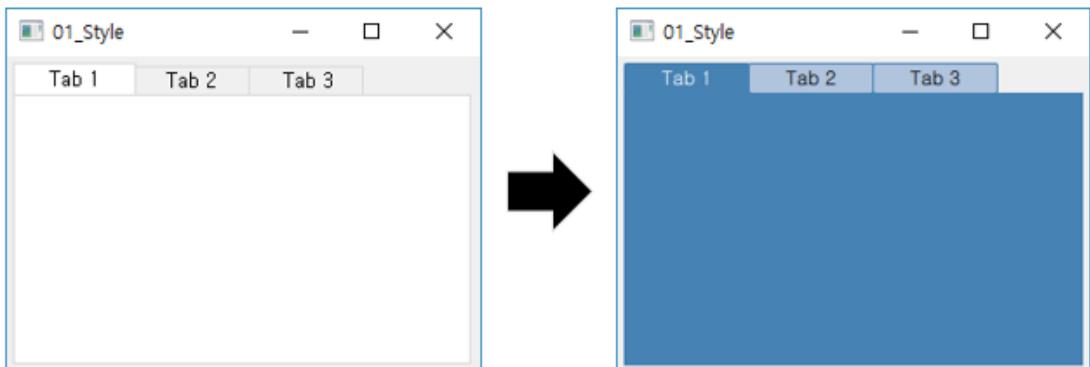
```
import QtQuick 2.12
import QtQuick.Controls 1.4
import QtQuick.Controls.Styles 1.4

ApplicationWindow {
    width: 300; height: 200; visible: true
    TabView {
        id: frame; anchors.fill: parent
        anchors.margins: 4
        Tab { title: "Tab 1" }
        Tab { title: "Tab 2" }
        Tab { title: "Tab 3" }

        style: TabViewStyle {
            frameOverlap: 1
        }
    }
}
```

## Jesus loves you.

```
tab: Rectangle {  
    color: styleData.selected ? "steelblue" :"lightsteelblue"  
    border.color: "steelblue"  
    implicitWidth: Math.max(text.width + 4, 80)  
    implicitHeight: 20  
    radius: 2  
    Text {  
        id: text; anchors.centerIn: parent  
        text: styleData.title  
        color: styleData.selected ? "white" : "black"  
    }  
}  
frame: Rectangle { color: "steelblue" }  
}  
}  
}
```



<FIGURE> QML Controls type

## 6.3. Qt Quick Controls 2

Qt Quick Controls 2.0 version has been officially released from Qt 5.7. Therefore, to use Qt Quick Controls version 2.0 or higher, you must use Qt 5.7 or higher.

<TABLE> Version Matching

Qt	QtQuick	<b>QtQuick.Controls, QtQuick.Controls.Material, QtQuick.Controls.Universal, QtQuick.Templates</b>	Qt.labs.calendar, Qt.labs.platform
5.7	2.7	<b>2.0</b>	1.0
5.8	2.8	<b>2.1</b>	1.0
5.9	2.9	<b>2.2</b>	1.0
5.10	2.10	<b>2.3</b>	1.0
5.11	2.11	<b>2.4</b>	1.0
5.12	2.12	<b>2.12</b>	1.0

To use version 2.x of Qt Quick Controls, you must import from QML as follows:

```
import QtQuick.Controls 2.12
```

In C++ the associated classes should include:

```
#include <QtQuickControls2>
```

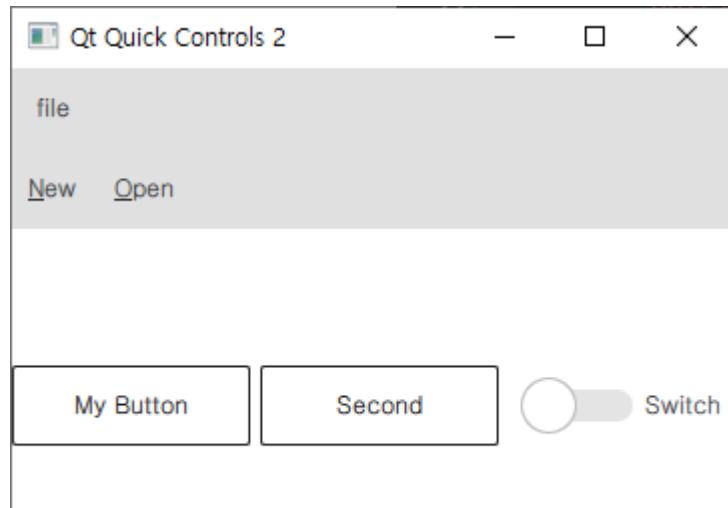
In addition, the project file (.pro) should include the following quickcontrols2.

```
QT += quickcontrols2
```

### ✓ Example using Qt Quick Controls 2

In this example, let's use Qt Quick Controls 2 to implement the GUI feature as shown in the following figure.

## Jesus loves you.



<FIGURE> Example Execution Screen

As you can see in the picture above, the top of the GUI is the menu. The [file] menu is submenu [New] and [Open]. Click [New] to print the message " New" on the central label as shown in the figure above, and click on the [Open] menu to print the message "Open".

The bottom of the menu is the Toolbar menu. Toolbar's menu item is the same as MenuBar's item. The following is an example source code that implements the GUI as shown in the figure above.

<Example> Ch06 > 03 > 01\_Controls2 > Basic.qml

```
import QtQuick 2.12
import QtQuick.Controls 2.12
import QtQuick.Layouts 1.12

ApplicationWindow
{
    id: window
    width: 360; height: 220; visible: true
    title: "Qt Quick Controls 2"

    Action {
        id: actionNew
        text: qsTr("&New")
        shortcut: StandardKey.New
        onTriggered: area.text = "New"
    }
}
```

## **Jesus loves you.**

```
Action {  
    id: actionOpen  
    text: qsTr("&Open")  
    shortcut: StandardKey.Open  
    onTriggered: area.text = "Open"  
}  
menuBar: MenuBar {  
    Menu {  
        title: "file"  
        MenuItem { action: actionNew }  
        MenuItem { action: actionOpen }  
    }  
}  
header: ToolBar {  
    RowLayout {  
        ToolButton { action: actionNew }  
        ToolButton { action: actionOpen }  
    }  
}  
Rectangle {  
    anchors.fill: parent  
    ColumnLayout {  
        id: colLayout  
        width: parent.width; height: parent.height  
        Label {  
            id: area  
            width: parent.width  
            height: 200  
        }  
        RowLayout {  
            id: rowLayout  
            Button {  
                text: "My Button"  
                Layout.fillWidth: true  
            }  
            Button {  
                id: button  
                text: "Second"  
                highlighted: true  
            }  
        }  
    }  
}
```

## Jesus loves you.

```
        Layout.fillWidth: true
    }
    Switch {
        id: element
        x: 95
        y: 165
        text: qsTr("Switch")
    }
} // RowLayout
} // ColumnLayout
} // Rectangle
}
```

The following example source code is a type example source code customized to the Button QML type.

<Example> Ch06 > 03 > 01\_Controls2 > Button.qml

```
import QtQuick 2.12
import QtQuick.Templates 2.5 as T

T.Button {
    id: control
    implicitWidth: Math.max(background ? background.implicitWidth : 0,
                           contentItem.implicitWidth + leftPadding + rightPadding)
    implicitHeight: Math.max(background ? background.implicitHeight : 0,
                           contentItem.implicitHeight + topPadding + bottomPadding)
    leftPadding: 4
    rightPadding: 4

    background: Rectangle {
        id: buttonBackground
        implicitWidth: 100
        implicitHeight: 40
        opacity: enabled ? 1 : 0.3
        border.color: "#222222"; border.width: 1
        radius: 2

        states: [
            State {
```

## Jesus loves you.

```
        name: "normal"
        when: !control.down
        PropertyChanges { target: buttonBackground }
    },
    State {
        name: "down"
        when: control.down
        PropertyChanges {
            target: buttonBackground
            border.color: "#dddddd"
        }
    }
]

}

contentItem: Text {
    id: textItem
    text: control.text; font: control.font
    opacity: enabled ? 1.0 : 0.3
    color: "#000000"
    horizontalAlignment: Text.AlignHCenter
    verticalAlignment: Text.AlignVCenter
    elide: Text.ElideRight
    states: [
        State {
            name: "normal"; when: !control.down
        },
        State {
            name: "down"
            when: control.down
            PropertyChanges {
                target: textItem; color: "#b2b1b1"
            }
        }
    ]
}
}
```

## 7. Qt Quick Extras

The Qt Quick Extras module consists of Interactive Control, which receives input from the user, and Non-interactive Control, which does not receive input from the user. For example, Interactive Control is an interactive control that requires an event to be generated from a user, such as Button.

In addition, it is called Non-interactive Control that users provide for the purpose of delivering information without the need for an event to occur. For example, for QML types that display the vehicle's instrument cluster UI, the QML type that does not require input from the user is classified as Non-interactive Controls. The following are classified as Interactive Controls and Non-interactive Controls.

<TABLE> Interactive Controls

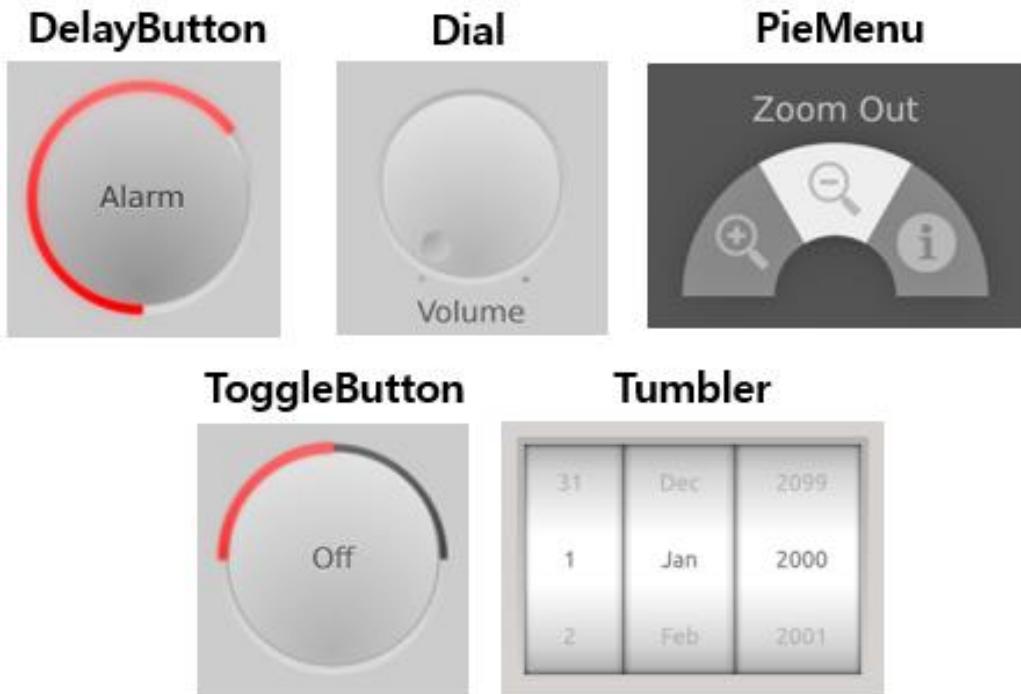
<b>QML</b>	<b>Description</b>
DelayButton	QML Type for long processing after clicking a button
Dial	QML type in circular form, such as Dial
PieMenu	GUI with menu items arranged along curves
ToggleButton	button such as ON/OFF
Tumbler	Wheel Spin Type

<TABLE> Non-interactive Controls

<b>QML</b>	<b>Description</b>
CircularGauge	GUI for setting values along the dial curve
Gauge	Type for specifying a range to indicate a Gauge
StatusIndicator	A type to indicate a particular situation. For example, Active or Inactive

## 7.1. Interactive Controls

Among Extras, Interactive Controls consists of a QML type to receive input from users and provides a GUI as shown in the following figure.



<FIGURE> Interactive Controls

### ✓ **DelayButton**

The onActivated signal is generated after the specified time value of the delay Property has been passed with the button pressed. The following is an example of DelayButton.

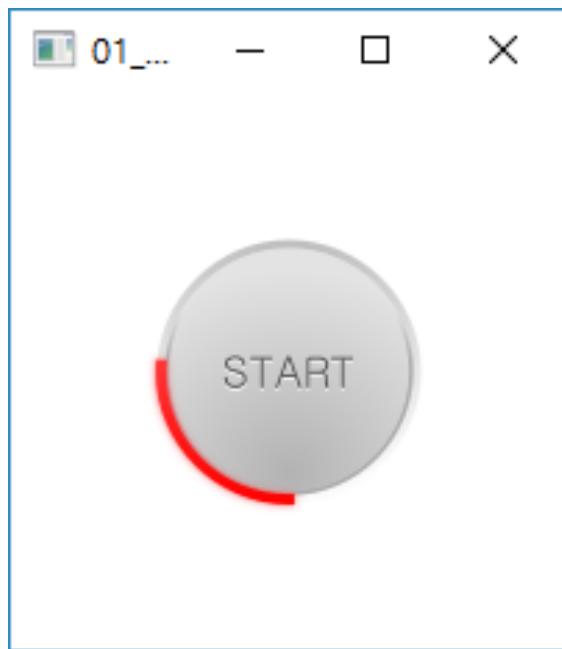
<Example> Ch07 > 01\_Extra > delaybutton.qml

```
import QtQuick 2.12
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Layouts 1.1

ApplicationWindow
```

## Jesus loves you.

```
{  
    width: 200  
    height: 200  
    visible: true  
  
    DelayButton  
    {  
        id: myDelay  
        delay: 3000 // milliseconds  
        text: progress < 1 ? "START" : "STOP"  
        anchors.centerIn: parent  
  
        onActivated: {  
            console.log("DelayButton Actived (Complete)");  
        }  
    }  
}
```



<FIGURE> Example Execution Screen

### ✓ Dial

Dial provides a GUI where you can change the value by dragging the scale on the GUI.

# Jesus loves you.

<Example> Ch07 > 01\_Extra > dial.qml

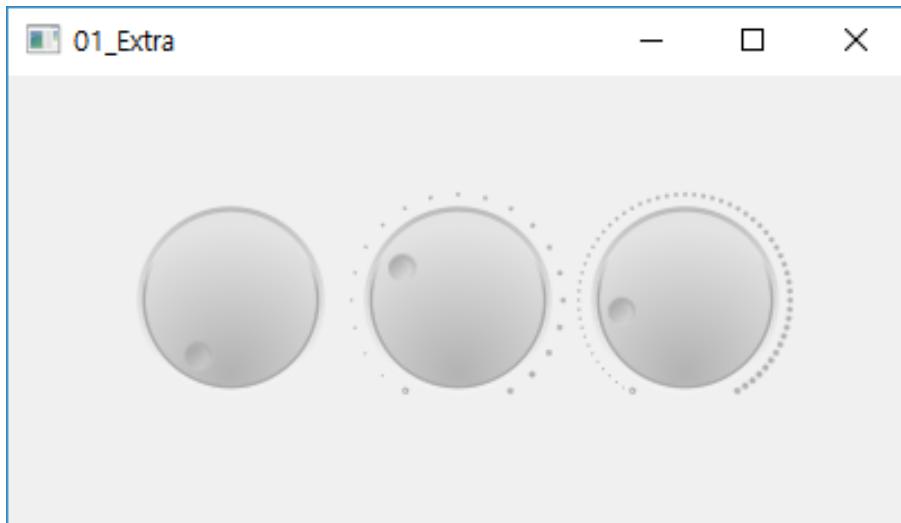
```
import QtQuick 2.12
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Layouts 1.1

ApplicationWindow
{
    width: 400
    height: 200
    visible: true

    RowLayout {
        anchors.centerIn: parent
        Dial {
            id: myDial1
            minimumValue: 0; maximumValue: 100
            tickmarksVisible: false
            stepSize: 20
            onValueChanged: {
                console.log('value:' + value)
            }
        }
        Dial {
            width: 200; height: 200
            id: myDial2
            minimumValue: 0; maximumValue: 20; stepSize: 2
            onValueChanged: {
                console.log('value:' + value)
            }
        }
        Dial {
            width: 200; height: 200
            id: myDial3
            minimumValue: 0; maximumValue: 60; stepSize: 2
            onValueChanged: {
                console.log('value:' + value)
            }
        }
    }
}
```

## Jesus loves you.

```
}
```



<FIGURE> Example Execution Screen

### ✓ PieMenu

PieMenu offers a fan-shaped menu GUI. You can select one of the items listed in a fan shape. Click the mouse button on the GUI and PieMenu appears.

<Example> Ch07 > 01\_Extra > piemenu.qml

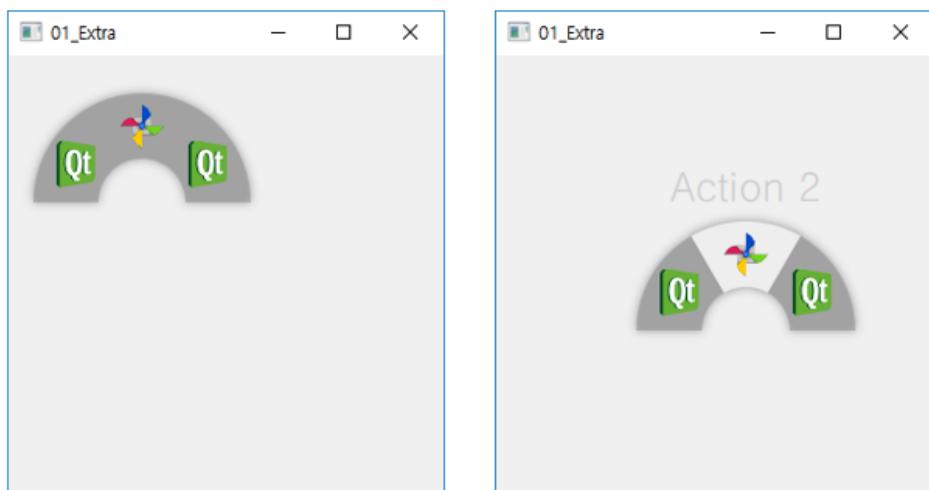
```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Layouts 1.1

ApplicationWindow
{
    width: 300; height: 300; visible: true
    MouseArea {
        anchors.fill: parent
        onClicked: { pieMenu.popup(mouseX, mouseY); }
    }

    PieMenu {
        id: pieMenu
```

## Jesus loves you.

```
triggerMode: TriggerMode.TriggerOnClick
MenuItem {
    text: "Action 1"
    iconSource: "images/qtlogo.png"
    onTriggered: {
        console.log('Action 1 click')
    }
}
MenuItem {
    text: "Action 2"
    iconSource: "images/pinwheel.png"
    onTriggered: {
        console.log('Action 2 click')
    }
}
MenuItem {
    text: "Action 3"
    iconSource: "images/qtlogo.png"
    onTriggered: {
        console.log('Action 3 click')
    }
}
```



<FIGURE> Example Execution Screen

# Jesus loves you.

## ✓ **ToggleButton**

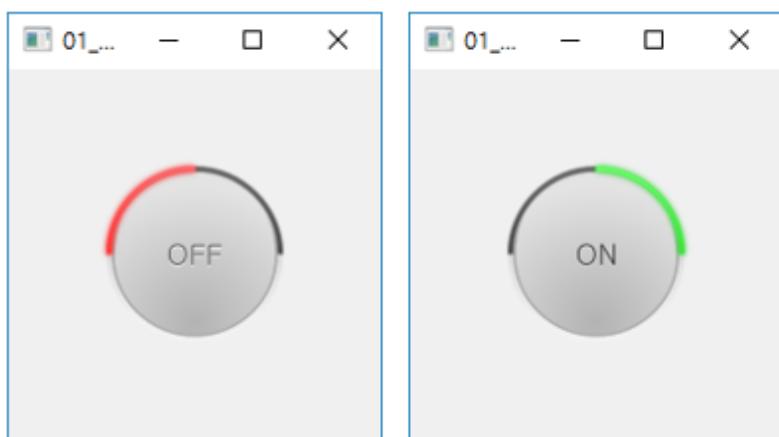
ToggleButton is used for the purpose of displaying one of two things, such as ON/OFF. The following is an example source code for ToggleButton.

<Example> Ch07 > 01\_Extra > togglebutton.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Layouts 1.1

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

    ToggleButton {
        id: myBtn
        text: "OFF"
        anchors.centerIn: parent
        onClicked: {
            if(checked)
                myBtn.text = "ON"
            else
                myBtn.text = "OFF"
        }
    }
}
```



<FIGURE> Example Execution Screen

## Jesus loves you.

### ✓ Tumbler

Tumbler is a GUI that provides Wheel Spin. The following example is an example of using the Tumbler type.

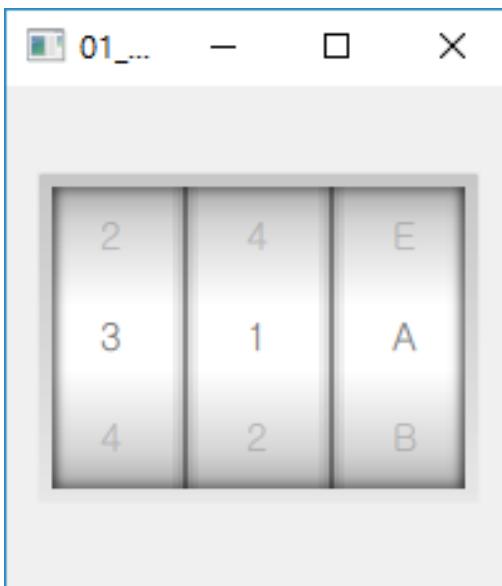
<Example> Ch07 > 01\_Extra > tumbler.qml

```
import QtQuick 2.12
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Layouts 1.1

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

    Tumbler {
        anchors.centerIn: parent
        TumblerColumn
        {
            id: myCol1
            model: 5
            onCurrentIndexChanged: {
                console.log("cur:" + myCol1.currentIndex);
            }
        }
        TumblerColumn {
            model: [1, 2, 3, 4]
        }
        TumblerColumn {
            model: ["A", "B", "C", "D", "E"]
        }
    }
}
```

## Jesus loves you.



<FIGURE> Example Execution Screen

### ✓ Tumbler 와 ListModel

This example is an example source code that uses data used by Wheel Spin in Tumbler as a ListModel.

<Example> Ch07 > 01\_Extra > tumbler\_model.qml

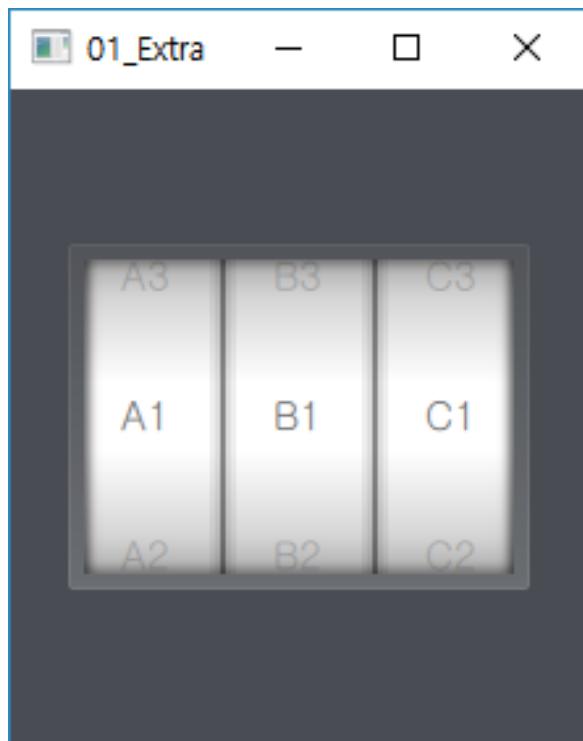
```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Layouts 1.1

ApplicationWindow
{
    width: 220
    height: 250
    visible: true

    Rectangle
    {
        anchors.centerIn: parent
        width: parent.width
        height: parent.height;
        color: "#494d53"
```

## Jesus loves you.

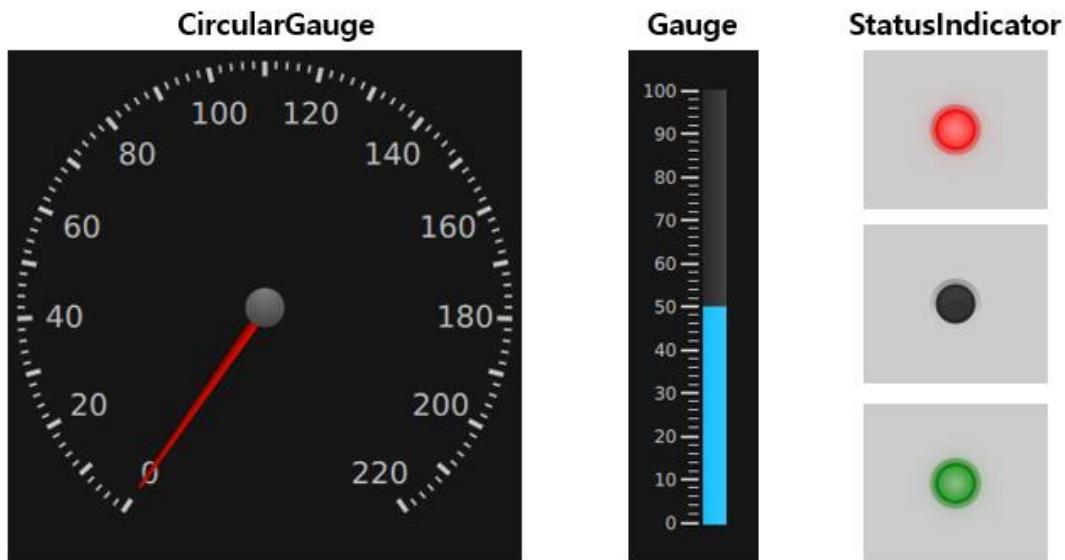
```
ListModel {  
    id: listModel  
    ListElement { foo: "A1"; bar: "B1"; baz: "C1" }  
    ListElement { foo: "A2"; bar: "B2"; baz: "C2" }  
    ListElement { foo: "A3"; bar: "B3"; baz: "C3" }  
}  
  
Tumbler {  
    anchors.centerIn: parent  
    TumblerColumn { model: listModel; role: "foo" }  
    TumblerColumn { model: listModel; role: "bar" }  
    TumblerColumn { model: listModel; role: "baz" }  
}  
}  
}
```



<FIGURE> Example Execution Screen

## 7.2. Non-interactive Controls

Non-interactive Controls are QML types provided by Extras modules and are types that do not receive input from users.



<FIGURE> Non-interactive Controls

### ✓ CircularGauge

The CircularGauge QML type provides a GUI, such as an automotive cluster (instrument cluster). The following example is the CircularGauge type example source code.

<Example> Ch07 > 02\_Extra\_Non\_Interactive > circulargauge.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Layouts 1.1

ApplicationWindow
{
    width: 250
    height: 250
```

## **Jesus loves you.**

```
visible: true
color: "black"

CircularGauge
{
    id: myCircular
    value: accelerating ? maximumValue : 0

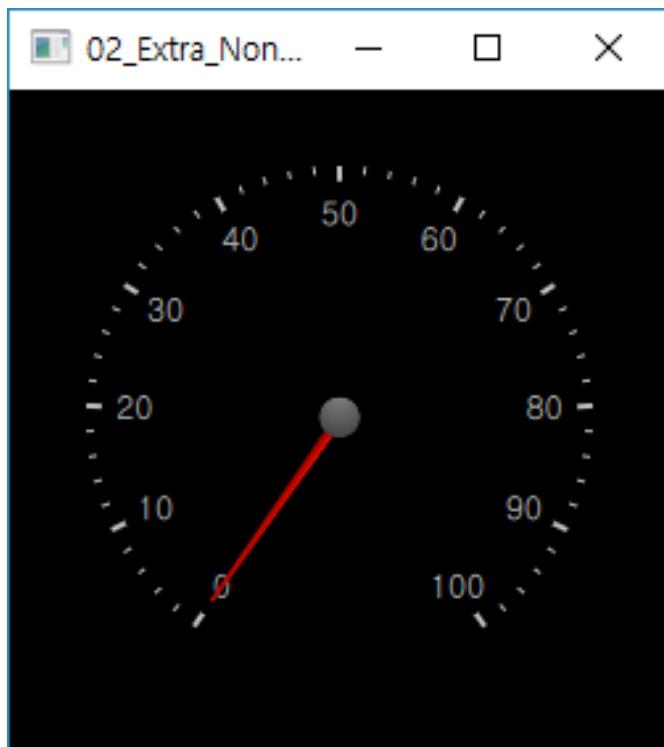
    anchors.centerIn: parent
    property bool accelerating: false

    Keys.onSpacePressed:
    {
        accelerating = true
        console.log("value:" + value.toPrecision(2))
    }

    Keys.onReleased:
    {
        if (event.key === Qt.Key_Space)
        {
            accelerating = false;
            event.accepted = true;
        }
    }
}

Component.onCompleted: forceActiveFocus()

Behavior on value
{
    NumberAnimation { duration: 1000 }
}
}
```



<FIGURE> Example Execution Screen

✓ **Gauge**

Gauge is the type to specify the range to display the current unit.

<Example> Ch07 > 02\_Extra\_Non\_Interactive > gauge.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Layouts 1.1
```

```
ApplicationWindow
{
```

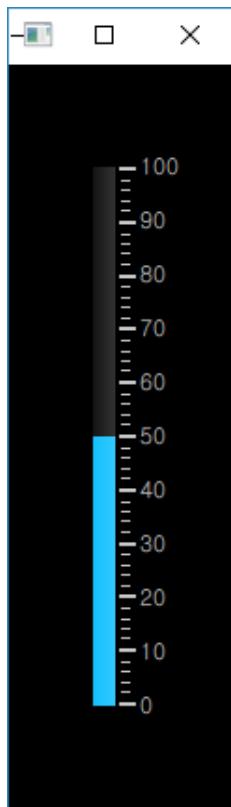
```
    width: 100
    height: 400
    visible: true
    color: "black"
```

```

    Gauge
    {
```

## Jesus loves you.

```
anchors.centerIn: parent
width: 30
height: 300
minimumValue: 0
value: 50
maximumValue: 100
tickmarkAlignment: Qt.AlignRight
}
}
```



<FIGURE> Example Execution Screen

The example above is a vertically oriented Gauge. The following is an example source code with Gauge positioned horizontally.

<Example> Ch07 > 02\_Extra\_Non\_Interactive > gauge2.qml

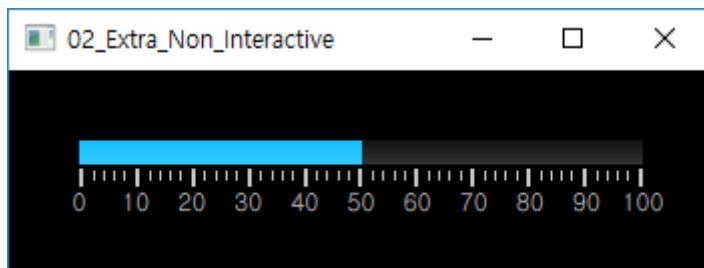
```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
```

## Jesus loves you.

```
import QtQuick.Layouts 1.1

ApplicationWindow {
    width: 350; height: 100; visible: true; color: "black"

    Gauge {
        width: 300; height: 30;
        minimumValue: 0
        value: 50
        maximumValue: 100
        //tickmarkAlignment: Qt.AlignTop
        tickmarkAlignment: Qt.AlignBottom
        orientation: Qt.Horizontal
        anchors.centerIn: parent
    }
}
```



<FIGURE> Example Execution Screen

### ✓ StatusIndicator

The StatusIndicator provides a GUI for displaying specific states. The following is the StatusIndicator example source code.

<Example> Ch07 > 02\_Extra\_Non\_Interactive > statusindicator.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Layouts 1.1

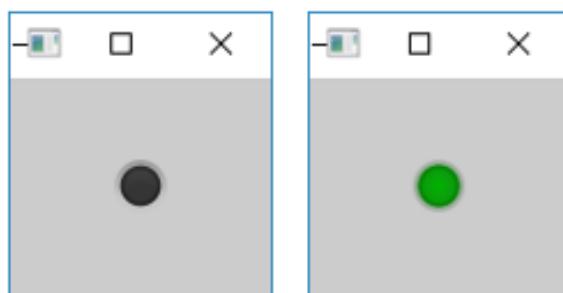
ApplicationWindow {
```

## Jesus loves you.

```
visible: true

Rectangle {
    width: parent.width
    height: parent.height
    color: "#cccccc"

    MouseArea {
        anchors.fill: parent
        onClicked: {
            if(myStatus.active == true)
                myStatus.active = false
            else
                myStatus.active = true
        }
    }
    StatusIndicator {
        id: myStatus
        anchors.centerIn: parent
        color: "green"
    }
}
```



<FIGURE> Example Execution Screen

## 7.3. Extra Style

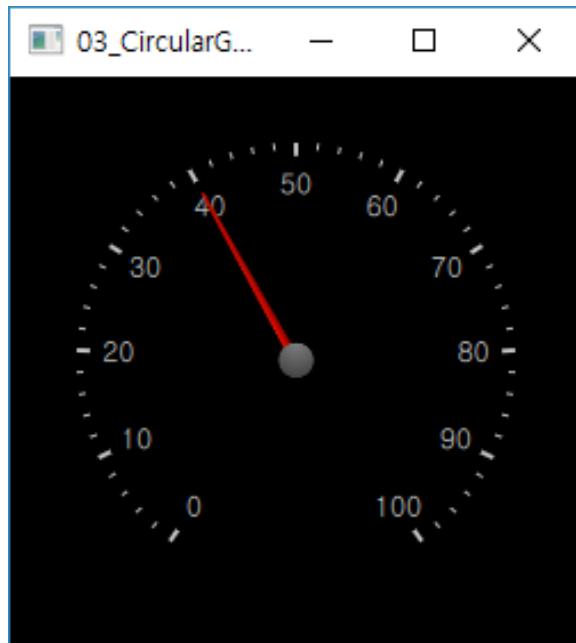
Like the style used in Qt Quick Controls, you can use the style in Extras. Similar to Controls, to use Style in Extra, you must import as follows.

```
import QtQuick.Controls.Styles 1.4
```

In this section, try using Style in CircularGauge type.

✓ [Step 1] CircularGauge Type

Using the CircularGauge type, complete the example as shown in the following figure.



<FIGURE> Example Execution Screen

<Example> Ch07 > 03\_CircularGage\_Style > basic\_circulargauge.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Controls.Styles 1.4
```

```
ApplicationWindow {
```

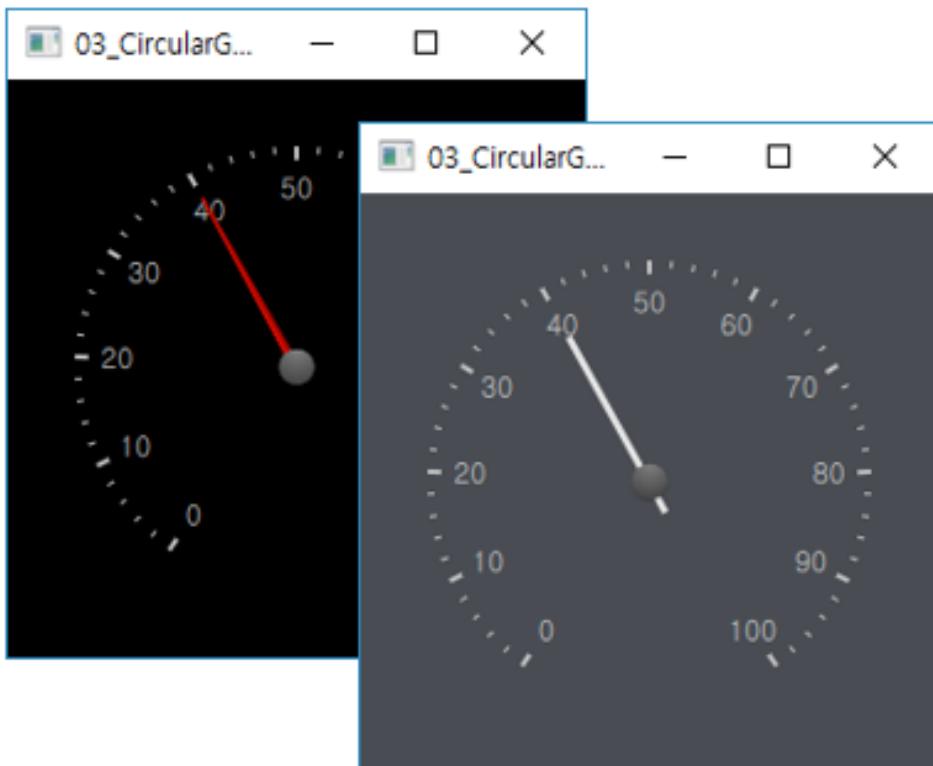
## Jesus loves you.

```
width: 250; height: 250; visible: true; color: "black"

CircularGauge {
    id: myCircular; value: 40
    anchors.centerIn: parent
}
}
```

### ✓ [Stage 2] Change Needle Style

[Stage 2] Change Needle Style Next, let's change the Needle style, which points to numbers as shown in the picture below, as seen in the next picture.



<FIGURE> Example Execution Screen

<Example> Ch07 > 03\_CircularGage\_Style > circulargauge\_style1.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Controls.Styles 1.4
```

## Jesus loves you.

```
ApplicationWindow
{
    width: 250
    height: 250
    visible: true
    color: "#494d53"

    CircularGauge
    {
        id: gauge
        value: 40
        anchors.centerIn: parent

        style: CircularGaugeStyle
        {
            id: style
            needle: Rectangle
            {
                y: outerRadius * 0.15

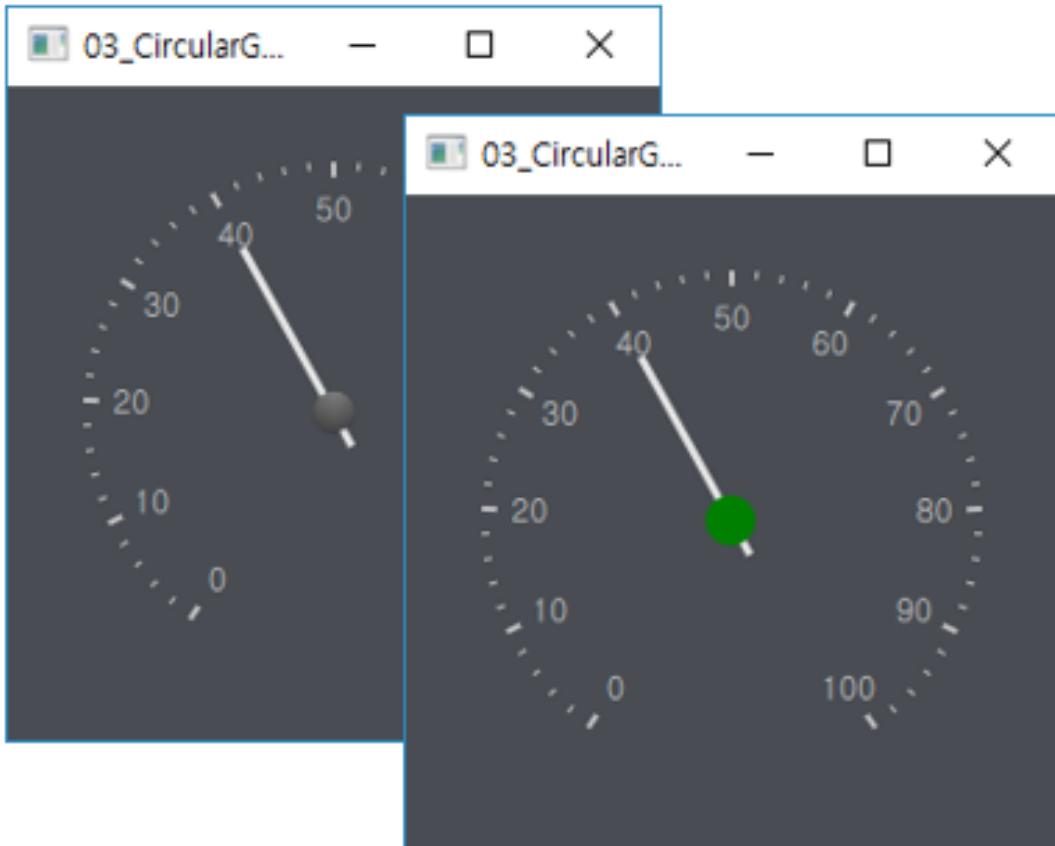
                implicitWidth: outerRadius * 0.03
                implicitHeight: outerRadius * 0.9

                antialiasing: true
                color: "#e5e5e5"
            }
        }
    }
}
```

- ✓ [Step 3] Change the center circle of the needle style to the green circle

Next, let's change the center circle of the needle as shown in the figure below.

**Jesus loves you.**



<FIGURE> Example Execution Screen

<Example> Ch07 > 03\_CircularGage\_Style > circulargauge\_style2.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Controls.Styles 1.4

ApplicationWindow
{
    width: 250
    height: 250
    visible: true
    color: "#494d53"

    CircularGauge
    {
        id: gauge
```

## Jesus loves you.

```
value: 40
anchors.centerIn: parent

style: CircularGaugeStyle
{
    id: style

    needle: Rectangle
    {
        y: outerRadius * 0.15

        implicitWidth: outerRadius * 0.03
        implicitHeight: outerRadius * 0.9

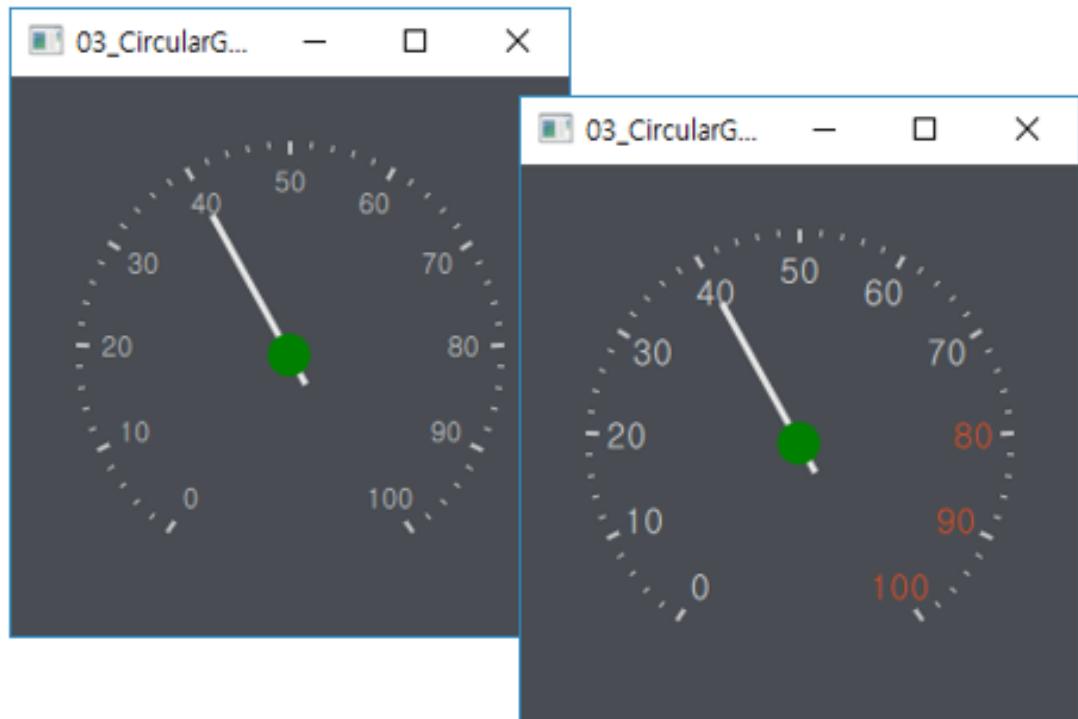
        antialiasing: true
        color: "#e5e5e5"
    }
    foreground: Item
    {
        Rectangle
        {
            width: outerRadius * 0.2
            height: width
            radius: width / 2
            color: "green"

            anchors.centerIn: parent
        }
    }
}
}
```

### ✓ [Step 4] Mark the number 80 in a different color

This time, let's change the value of the numerical scale from 80 to a different color.

**Jesus loves you.**



<FIGURE> Example Execution Screen

<Example> Ch07 > 03\_CircularGage\_Style > circulargauge\_style3.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Controls.Styles 1.4

ApplicationWindow {
    width: 250
    height: 250
    visible: true
    color: "#494d53"

    CircularGauge
    {
        id: gauge
        value: 40
        anchors.centerIn: parent
        style: CircularGaugeStyle {
            id: style
```

## Jesus loves you.

```
needle: Rectangle
{
    y: outerRadius * 0.15

    implicitWidth: outerRadius * 0.03
    implicitHeight: outerRadius * 0.9

    antialiasing: true
    color: "#e5e5e5"
}

foreground: Item
{
    Rectangle
    {
        width: outerRadius * 0.2
        height: width
        radius: width / 2
        color: "green"
        anchors.centerIn: parent
    }
}

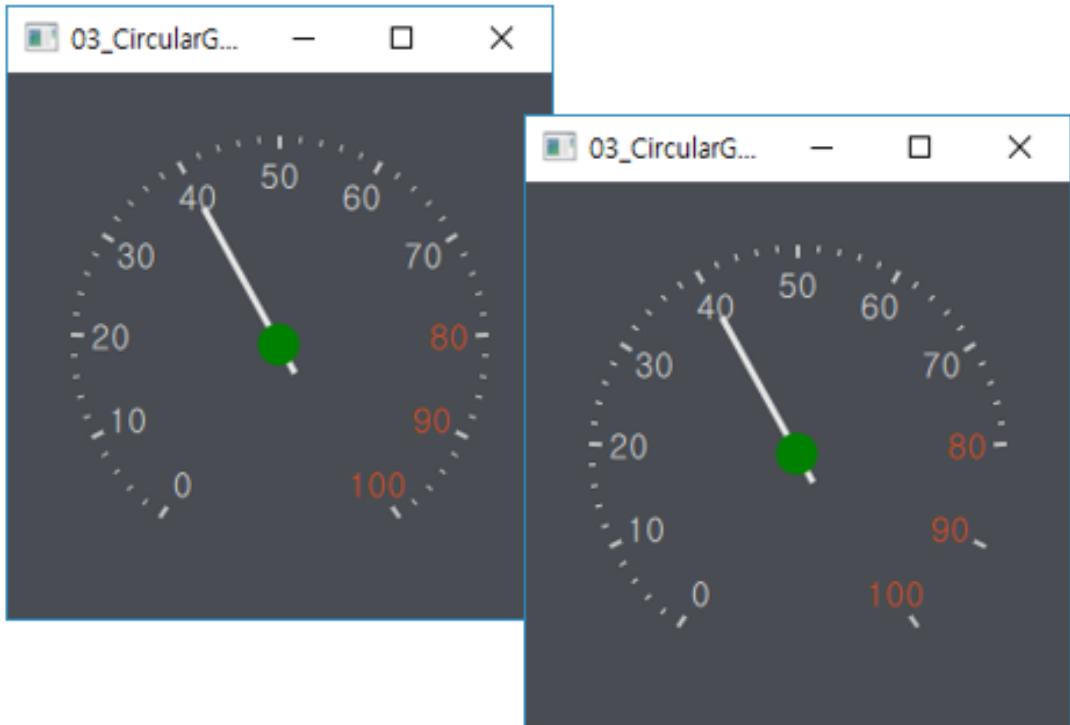
tickmarkLabel: Text
{
    font.pixelSize: 15
    text: styleData.value

    color: styleData.value >= 80 ? "#e34c22" : "#e5e5e5"
    antialiasing: true
}
}
```

### ✓ [Stage 5] Do not display tick marks from 80

This time, let's find out how not to mark the scale from above 80, as shown in the picture below.

## Jesus loves you.



<FIGURE> Example Execution Screen

<Example> Ch07 > 03\_CircularGage\_Style > circulargauge\_style4.qml

```
import QtQuick 2.4
import QtQuick.Controls 1.3
import QtQuick.Extras 1.4
import QtQuick.Controls.Styles 1.4

ApplicationWindow
{
    width: 250
    height: 250
    visible: true
    color: "#494d53"

    CircularGauge
    {
        id: gauge
        value: 40
        anchors.centerIn: parent
        style: CircularGaugeStyle
```

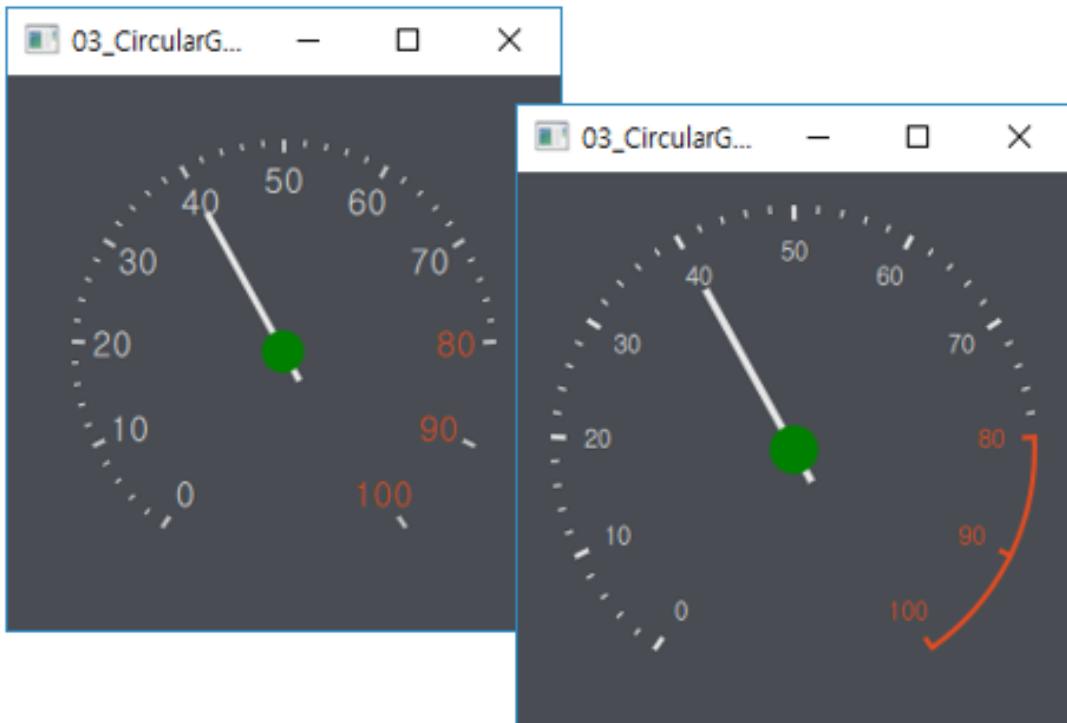
## Jesus loves you.

```
{  
    id: style  
    needle: Rectangle {  
        y: outerRadius * 0.15  
        implicitWidth: outerRadius * 0.03  
        implicitHeight: outerRadius * 0.9  
        antialiasing: true  
        color: "#e5e5e5"  
    }  
    foreground: Item {  
        Rectangle {  
            width: outerRadius * 0.2  
            height: width  
            radius: width / 2  
            color: "green"  
            anchors.centerIn: parent  
        }  
    }  
  
    tickmarkLabel: Text {  
        font.pixelSize: 15  
        text: styleData.value  
        color: styleData.value >= 80 ? "#e34c22" : "#e5e5e5"  
        antialiasing: true  
    }  
  
    minorTickmark: Rectangle {  
        visible: styleData.value < 80  
        implicitWidth: outerRadius * 0.01  
        antialiasing: true  
        implicitHeight: outerRadius * 0.03  
        color: "#e5e5e5"  
    }  
}  
}  
}
```

## Jesus loves you.

- ✓ [Step 6] Mark the tick as a line that connects.

This time, let's learn how to mark a tick with a line that leads to where it was located.



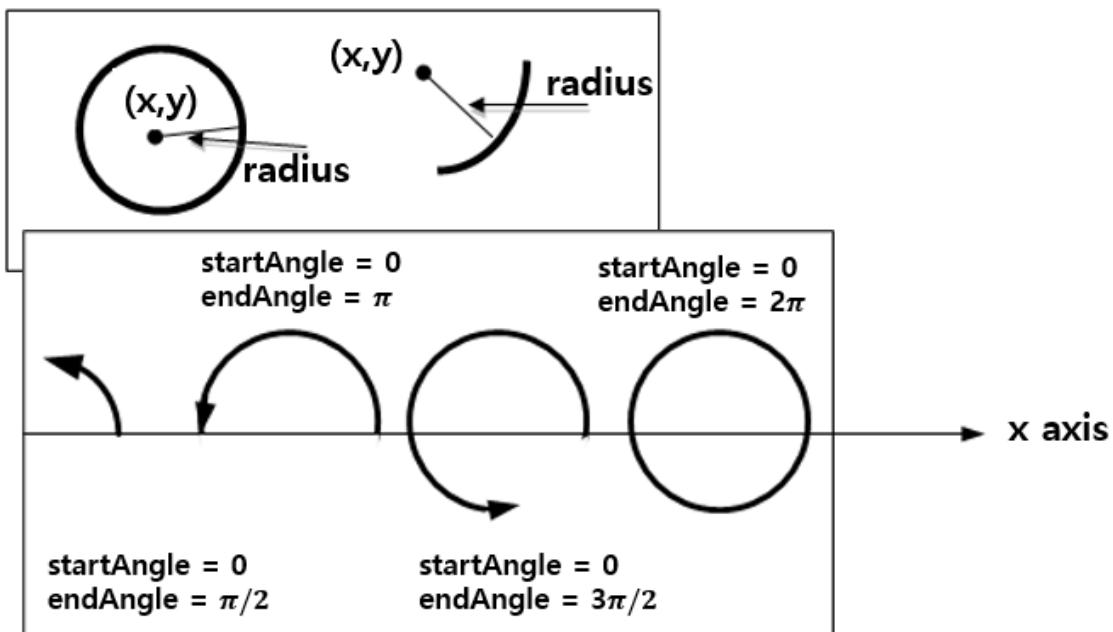
<FIGURE> 예제 실행 화면

Canvas was used to mark the lines connected to the grid starting at 80 or higher as shown in the figure above. The arc function was then used to draw the line from a specific angle to an angle from which the line ended.

```
object arc( real x,  
            real y,  
            real radius,  
            real startAngle,  
            real endAngle,  
            bool anticlockwise )
```

The first, second x, and y are the central coordinates. The third radius is the radius. The fourth factor determines the angle of initiation, the fifth the last angle, and the sixth factor determines whether it is clockwise or not.

**Jesus loves you.**



<FIGURE> arc function factors

<Example> Ch07 > 03\_CircularGage\_Style > circulargauge\_style5.qml

```
import QtQuick 2.0
import QtQuick.Controls 1.4
import QtQuick.Controls.Styles 1.4
import QtQuick.Extras 1.4
import QtQuick.Extras.Private 1.0

ApplicationWindow {
    width: 250
    height: 250
    visible: true
    color: "#494d53"

    CircularGauge {
        value: 40
        id: gauge
        width: parent.width-30; height: parent.height-30
        anchors.centerIn: parent
        style: CircularGaugeStyle {
            id: style
        }
    }
}
```

## Jesus loves you.

```
function degreesToRadians(degrees) {
    return degrees * (Math.PI / 180);
}

background: Canvas {
    onPaint: {
        var ctx = getContext("2d");
        ctx.reset();
        ctx.beginPath();
        ctx.strokeStyle = "#e34c22";
        ctx.lineWidth = outerRadius * 0.02;
        ctx.arc(outerRadius,
                outerRadius,
                outerRadius - ctx.lineWidth / 2,
                degreesToRadians(valueToAngle(80) - 90),
                degreesToRadians(valueToAngle(100) - 90));
        ctx.stroke();
    }
}

tickmark: Rectangle {
    visible: styleData.value < 80 || styleData.value % 10 == 0
    implicitWidth: outerRadius * 0.02
    antialiasing: true
    implicitHeight: outerRadius * 0.06
    color: styleData.value >= 80 ? "#e34c22" : "#e5e5e5"
}

minorTickmark: Rectangle {
    visible: styleData.value < 80
    implicitWidth: outerRadius * 0.01
    antialiasing: true
    implicitHeight: outerRadius * 0.03
    color: "#e5e5e5"
}

tickmarkLabel: Text {
    font.pixelSize: Math.max(6, outerRadius * 0.1)
    text: styleData.value
```

## Jesus loves you.

```
color: styleData.value >= 80 ? "#e34c22" : "#e5e5e5"
antialiasing: true
}

needle: Rectangle {
    y: outerRadius * 0.15
    implicitWidth: outerRadius * 0.03
    implicitHeight: outerRadius * 0.9
    antialiasing: true
    color: "#e5e5e5"
}

foreground: Item {
    Rectangle {
        width: outerRadius * 0.2
        height: width
        radius: width / 2
        color: "green"
        anchors.centerIn: parent
    }
}
}
}
```