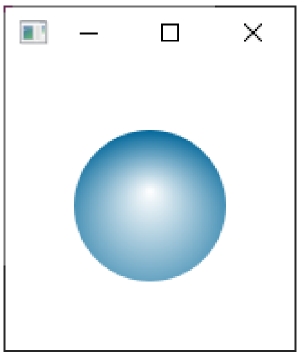
Creating a Mobile Application

This tutorial describes how to use Qt Creator to develop Qt Quick applications for Android and iOS devices when using Qt 6 as the minimum Qt version and CMake as the build system.

We implement a Qt Quick application that accelerates an SVG (Scalable Vector Graphics) image based on the changing accelerometer values.

Note: You must have the Qt Sensors module from Qt 6.2 or later installed to be able to follow this tutorial.



Setting up the Development Environment

To build the application for and run it on a mobile device, you must set up the development environment for the device platform and configure a connection between Qt Creator and the mobile device.

To develop for Android devices, you must install Qt for Android and set up the development environment, as instructed in Connecting Android Devices.

To develop for iOS devices, you must install Xcode and use it to configure a device. For this, you need an Apple developer account and iOS Developer Program certificate that you receive from Apple. For more information, see Connecting iOS Devices.

Creating the Project

1. Select **File** > **New Project** > **Application (Qt)** > **Qt Quick Application**.
2. Select **Choose** to open the **Project Location** dialog.
3. In the **Name** field, enter a name for the application. When naming your own projects, keep in mind that they cannot be easily renamed later.
4. In the **Create in** field, enter the path for the project files. You can move project folders later without problems.
5. Select **Next** (or **Continue** on macOS) to open the **Define Build System** dialog.
6. In the **Build system** field, select CMake as the build system to use for building and running the project.

   Note: If you select qmake, the instructions for configuring the project won't apply.

7. Select **Next** to open the **Define Project Details** dialog.

10. Select **Next** to use the default settings and to open the **Kit Selection** dialog.
11. Select Qt 6.2 or later kits for the platforms that you want to build the application for. To build applications for mobile devices, select kits also for Android and iOS.

    Note: Kits are listed if they have been specified in **Edit** > **Preferences** > **Kits** (on Windows and Linux) or in **Qt Creator** > **Preferences** > **Kits** (on macOS). For more information, see Adding Kits.

12. Select **Next** to open the **Project Management** dialog.
13. Review the project settings, and select **Finish** (or **Done** on macOS) to create the project.

For more information about the settings that you skipped and the other wizard templates available, see Creating Qt Quick Applications.

Adding Images as Resources

The main view of the application displays an SVG bubble image that moves around the screen when you tilt the device.

We use *Bluebubble.svg* in this tutorial, but you can use any other image or component instead.

For the image to appear when you run the application, you must specify it as a resource in the RESOURCES section of *CMakeLists.txt* file that the wizard created for you:

```
qt_add_qml_module(appaccelbubble
    URI accelbubble
    VERSION 1.0
    QML_FILES main.qml
    RESOURCES Bluebubble.svg
)
```

Creating the Accelbubble Main View

We create the main view in the *main.qml* file by adding an Image component with *Bluebubble.svg* as the source:

```
    Image {
        id: bubble
        source: "Bluebubble.svg"
        smooth: true
```

Next, we add custom properties to position the image in respect to the width and height of the main window:

```
        property real centerX: mainWindow.width / 2
        property real centerY: mainWindow.height / 2
        property real bubbleCenter: bubble.width / 2
        x: centerX - bubbleCenter
        y: centerY - bubbleCenter
```

We now want to add code to move the bubble based on Accelerometer sensor values. First, we add the following import statement:

```
import QtSensors
```

Next, we add the Accelerometer component with the necessary properties:

```
    Accelerometer {
        id: accel
        dataRate: 100
        active:true
```

Then, we add the following JavaScript functions that calculate the x and y position of the bubble based on the current Accelerometer values:

```
    function calcPitch(x,y,z) {
        return -Math.atan2(y, Math.hypot(x, z)) * mainWindow.radians_to_degrees;
    }
    function calcRoll(x,y,z) {
        return -Math.atan2(x, Math.hypot(y, z)) * mainWindow.radians_to_degrees;
    }
```

We add the following JavaScript code for `onReadingChanged` signal of Accelerometer component to make the bubble move when the Accelerometer values change:

```
        onReadingChanged: {
            var newX = (bubble.x + calcRoll(accel.reading.x, accel.reading.y, accel.reading.z) * .1)
            var newY = (bubble.y - calcPitch(accel.reading.x, accel.reading.y, accel.reading.z) * .1)

            if (isNaN(newX) || isNaN(newY))
                return;

            if (newX < 0)
                newX = 0

            if (newX > mainWindow.width - bubble.width)
                newX = mainWindow.width - bubble.width

            if (newY < 18)
                newY = 18

            if (newY > mainWindow.height - bubble.height)
                newY = mainWindow.height - bubble.height

                bubble.x = newX
                bubble.y = newY
        }
```

We want to ensure that the position of the bubble is always within the bounds of the screen. If the Accelerometer returns *not a number* (NaN), the value is ignored and the bubble position is not updated.

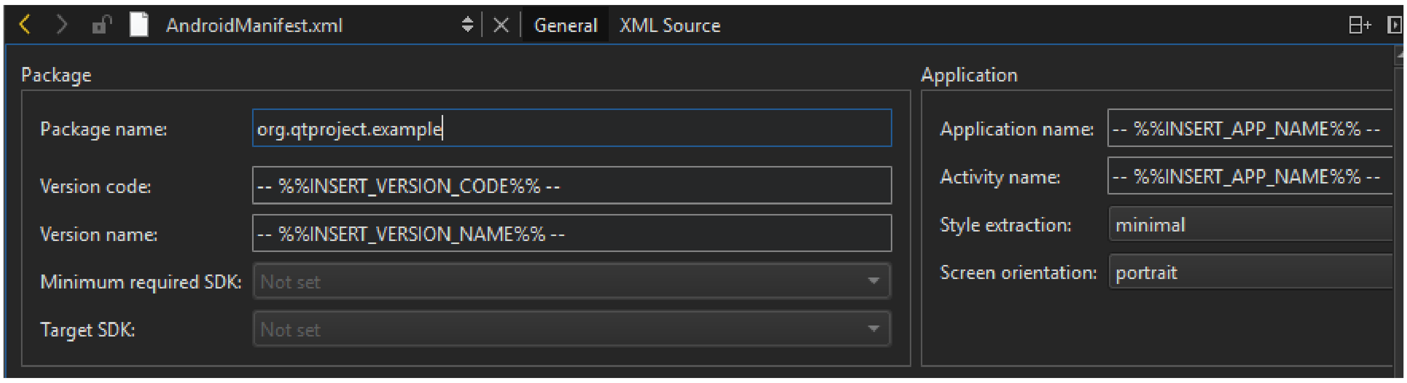
We add SmoothedAnimation behavior on the `x` and `y` properties of the bubble to make its movement look smoother.

```
        Behavior on y {
            SmoothedAnimation {
                easing.type: Easing.Linear
                duration: 100
            SmoothedAnimation {
                easing.type: Easing.Linear
                duration: 100
            }
        }
```

Locking Device Orientation

The device display is rotated by default when the device orientation changes between portrait and landscape. For this example, it would be better for the screen orientation to be fixed.

To lock the orientation to portrait or landscape on Android, specify it in an *AndroidManifest.xml* that you can generate in Qt Creator. For more information, see Editing Manifest Files.



Package

Package name:	org.qtproject.example
Version code:	-- %%INSERT_VERSION_CODE%% --
Version name:	-- %%INSERT_VERSION_NAME%% --
Minimum required SDK:	Not set
Target SDK:	Not set

Application

Application name:	-- %%INSERT_APP_NAME%% --
Activity name:	-- %%INSERT_APP_NAME%% --
Style extraction:	minimal
Screen orientation:	portrait

To generate and use a manifest file, you must specify the Android package source directory, `QT_ANDROID_PACKAGE_SOURCE_DIR` in the *CMakeLists.txt* file:

```
set_property(TARGET appaccelbubble APPEND PROPERTY
    QT_ANDROID_PACKAGE_SOURCE_DIR ${CMAKE_CURRENT_SOURCE_DIR}/android
)
```

Because our CMake version is older than 3.19, we must add a manual finalization step to the `qt_add_executable` function:

```
qt_add_executable(appaccelbubble
    main.cpp
    MANUAL_FINALIZATION
)
```

We also need to add the `qt_finalize_executable` function:

```
qt_finalize_executable(appaccelbubble)
```

On iOS, you can lock the device orientation in an *Info.plist* file that you specify in the *CMakeLists.txt* file as the value of the `MACOSX_BUNDLE_INFO_PLIST` variable:

```
set_target_properties(appaccelbubble PROPERTIES
    MACOSX_BUNDLE_GUI_IDENTIFIER my.example.com
    MACOSX_BUNDLE_BUNDLE_VERSION ${PROJECT_VERSION}
    WIN32_EXECUTABLE TRUE
)
```

Adding Dependencies

You must tell the build system which Qt modules your application needs by specifying dependencies in the project file. Select **Projects** to update the CMake configuration with the following Qt module information: `Sensors`, `Svg`, `Xml`.

The *CMakeLists.txt* file should contain the following entries that tell CMake to look up the Qt installation and import the Qt Sensors, Qt SVG, and Qt XML modules needed by the application:

```
find_package(Qt6 6.2 COMPONENTS Quick Sensors Svg Xml REQUIRED)
```

You also need to add the Qt modules to the list of target link libraries. `target_link_libraries` tells CMake that the accelbubble executable uses the Qt Sensors, Qt SVG, and Qt XML modules by referencing the targets imported by the `find_package()` call above. This adds the necessary arguments to the linker and makes sure that the appropriate include directories and compiler definitions are passed to the C++ compiler.

```
target_link_libraries(appaccelbubble
    PRIVATE Qt6::Quick Qt6::Sensors Qt6::Svg Qt6::Xml)
```

After adding the dependencies, select **Build** > **Run CMake** to apply configuration changes.

For more information about the CMakeLists.txt file, see Getting started with CMake.

Running the Application

The application is complete and ready to be deployed to a device:

1. Enable *USB Debugging* on the Android device or *developer mode* on the iOS device.
2. Connect the device to the development PC.

   If you are using a device running Android v4.2.2, it should prompt you to verify the connection to allow USB debugging from the PC it is connected to. To avoid such prompts every time you connect the device, select the **Always allow from this computer** check box, and then select **OK**.
3. To run the application on the device, press **Ctrl+R**.

Files:

- accelbubble/Bluebubble.svg
- accelbubble/CMakeLists.txt
- accelbubble/main.qml

‹ Creating a Qt Widget Based Application Managing Projects ›