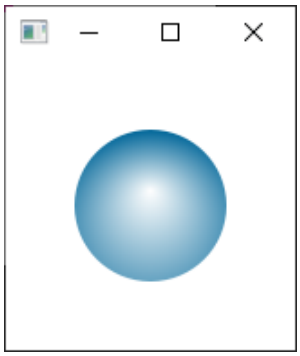
创建移动应用程序

本教程介绍如何使用 Qt Creator 为安卓和 iOS 设备开发 Qt 快速应用程序，同时使用 Qt 6 作为最低 Qt 版本，使用 CMake 作为构建系统。

我们实现了一个Qt Quick应用程序，该应用程序根据不断变化的加速度计值加速SVG（可缩放矢量图形）图像。

> **注意：** 您必须安装 Qt 6.2 或更高版本中的 Qt 传感器模块才能学习本教程。



设置开发环境

要为移动设备构建应用程序并在移动设备上运行该应用程序，您必须为设备平台设置开发环境，并配置Qt Creator与移动设备之间的连接。

要针对安卓设备进行开发，您必须按照连接安卓设备中的说明安装安卓版 Qt 并设置开发环境。

要针对 iOS 设备进行开发，必须安装 Xcode 并使用它来配置设备。为此，您需要一个苹果开发者帐户和iOS开发者计划证书，该证书是从苹果收到的。有关更多信息，请参阅连接 iOS 设备。

创建项目

1. 选择"**文件**>**新项目**>**应用程序 （Qt）** > **Qt 快速应用程序**"。
2. 选择"**选择**"以打开"**项目位置**"对话框。
3. 在"**名称**"字段中，输入应用程序的名称。在命名您自己的项目时，请记住，以后无法轻松重命名它们。
4. 在"**创建位置**"字段中，输入项目文件的路径。您可以稍后移动项目文件夹而不会出现问题。
5. 选择"**下一步**"（或在 macOS 上**选择**"**继续**"）以打开"**定义生成系统**"对话框。
6. 在"**生成系统**"字段中，选择"CMake"作为用于生成和运行项目的生成系统。

   > **注意：** 如果选择 qmake，则配置项目的说明将不适用。

7. 选择"**下一步**"以打开"**定义项目详细信息**"对话框。
8. 在"**所需的最低 Qt 版本**"字段中，选择"Qt 6.2"。
9. 选择"**下一步**"以打开"**翻译文件**"对话框。

的工具包。

> **注意：** 如果在**"编辑>首选项">工具包"**（在 Windows 和 Linux 上）或**"Qt 创建者**>首选项">**工具包"**（在 macOS 上）中指定了工具包，则会列出**这些工具包。** 有关详细信息，请参阅添加工具包。

12. 选择"**下一步**"以打开"**项目管理**"对话框。
13. 查看项目设置，然后选择**"完成"**（或在 macOS 上**选择"完成**"）以创建项目。

有关跳过的设置和其他可用向导模板的更多信息，请参阅创建 Qt 快速应用程序。

将图像添加为资源

应用程序的主视图显示一个 SVG 气泡图像，当您倾斜设备时，该图像在屏幕上移动。

我们在本教程中使用 *bluebubble.svg*，但您可以改用任何其他图像或组件。

若要在运行应用程序时显示该图像，必须在向导为您创建的 *CMakeLists.txt*文件中将其指定为资源：RESOURCES

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

创建增量存根主视图

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:

```
        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 signal of Accelerometer component to make the bubble move when the Accelerometer values change:onReadingChanged

```
    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.

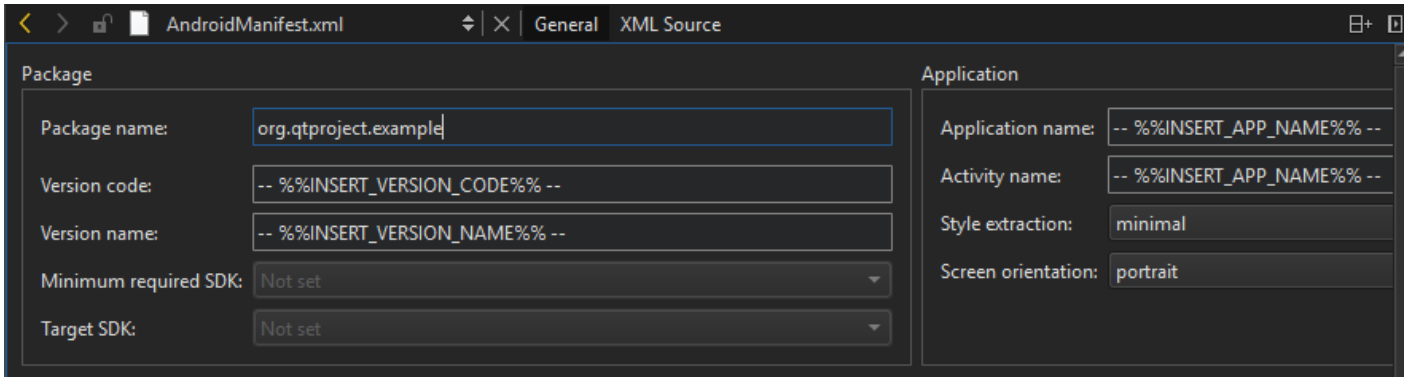
We add SmoothedAnimation behavior on the and properties of the bubble to make its movement look smoother.xy

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



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

```
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 function:`qt_add_executable`

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

We also need to add the function:`qt_finalize_executable`

```
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 variable:`MACOSX_BUNDLE_INFO_PLIST`

```
set_target_properties(appaccelbubble PROPERTIES
    MACOSX_BUNDLE_GUI_IDENTIFIER my.example.com
    MACOSX_BUNDLE_BUNDLE_VERSION ${PROJECT_VERSION}
    MACOSX_BUNDLE_SHORT_VERSION_STRING ${PROJECT_VERSION_MAJOR}.${PROJECT_VERSION_MINOR}
    MACOSX_BUNDLE_INFO_PLIST "${CMAKE_CURRENT_SOURCE_DIR}/Info.plist"
    MACOSX_BUNDLE TRUE
    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: , , .SensorsSvgXml

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. tells CMake that the accelbubble executable uses the Qt Sensors, Qt SVG, and Qt XML modules by referencing the targets imported by the 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_librariesfind_package()

```
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 ›

Qt DOCUMENTATION

Qt The Qt Company

Contact Us

Company

About Us
Investors
Newsroom
Careers
Office Locations

Licensing

Terms & Conditions
Open Source
FAQ

Support

Support Services
Professional Services
Partners
Training

For Customers

Support Center
Downloads
Qt Login
Contact Us
Customer Success

Community

Contribute to Qt
Forum
Wiki
Downloads
Marketplace

Feedback Sign In