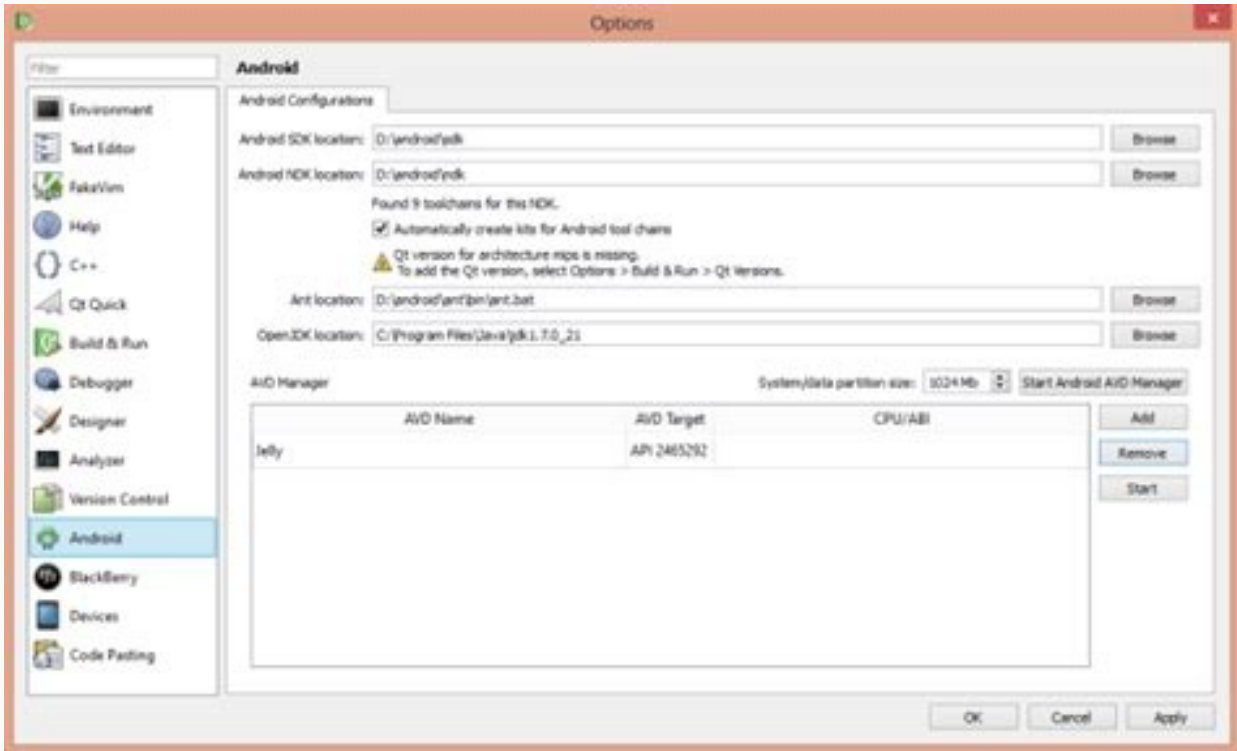
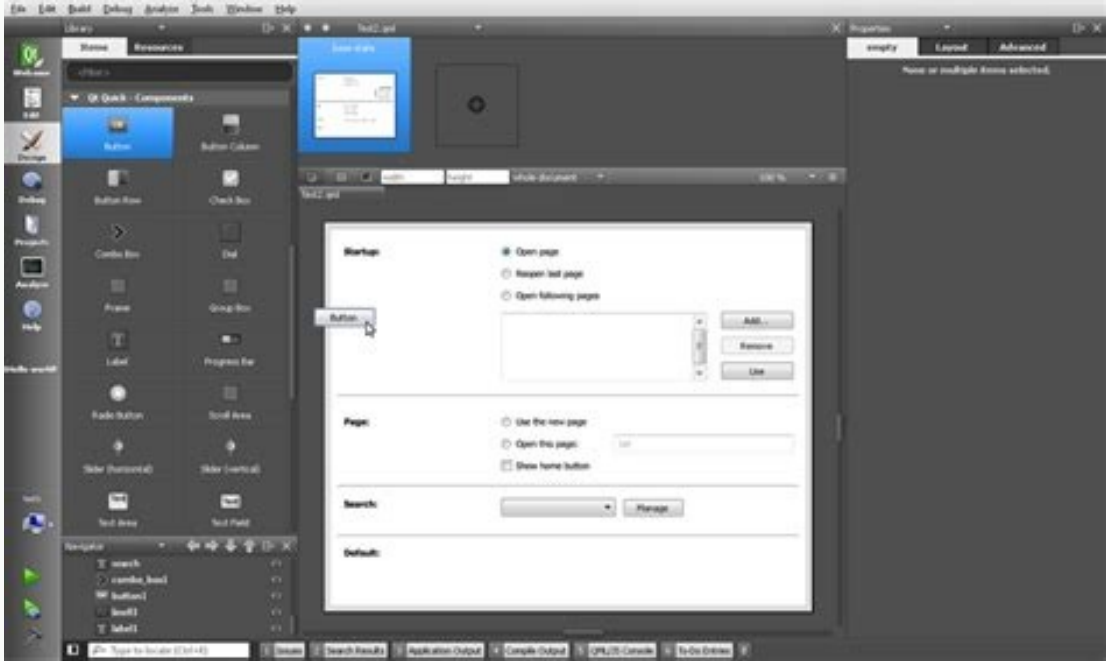


Continue





Qt apps examples. Qt mobile app example. Qt widget application example. Qt console app example.

A mobile application designed for use on devices with the Google Android platform. Android apps are available from the Google Play Store (formerly known as Android Market), Amazon Appstore, and various Android app websites, and apps can run on Android smartphones, tablets, Google TV, and other devices. As with Apple and its apps in the Apple App Store, Google encourages developers to develop their own Android apps. While many Android apps are free to download, users can also purchase premium apps, with the latter's revenue split between Google (30%) and the software developer (70%). In addition, some Android apps use the freemium business model, whereby the app developer can monetize free apps using Google's in-app billing feature. Porting to Android Deploying Android Applications © 2022 Qt Company Ltd. The documentation contained herein is the property of their respective owners. The documentation provided here is licensed under the GNU Free Documentation License version 1.3, published by the Free Software Foundation. Qt and its logo - The Qt Company Ltd. trademarks in Finland and/or other countries of the world. All other trademarks are the property of their respective owners. This guide explains how to deploy cross-platform applications built with Qt (Qt Widgets or Qt Quick/QML) to Android mobile phones or tablets. This article uses Qt Creator running on Windows as the deployment tool. Android is a mobile operating system based on a modified version of the Linux kernel and others... This tutorial describes how to use Qt Creator to create Qt Quick applications for Android and Android devices, on iOS using Qt 6 as the minimum version of Qt and CMake as the build system. We are implementing a Qt Quick application that accelerates an SVG (Scalable Vector Graphics) image based on changing accelerometer values. Note:A mobile application developed for use on devices with the Google Android platform. Android apps are available on Google Play Store (formerly known as Android Market), Amazon Appstore, and various Android app websites, and apps run on Android smartphones, tablets, Google TV, and other devices. As with Apple and its Apple App Store apps, Google encourages developers to create their own Android apps. While many Android apps are free to download, users can also purchase premium apps, with revenue from these apps split between Google (30%) and software developers (70%). In addition, some Android apps follow a freemium business model, where the app developer can earn revenue from free apps through the Google in-app billing feature. Porting to Android Deploying Android apps © 2022 The Qt Company Ltd. The documentation contained herein is copyrighted by their respective owners. The documentation provided here is licensed under the terms of the GNU Free Documentation License version 1.3 as published by the Free Software Foundation. Qt and its logo are trademarks of The Qt Company Ltd. in Finland and/or other countries around the world. All other trademarks are the property of their respective owners. This tutorial explains how to deploy cross-platform applications (Qt Widgets or Qt Quick/QML) built with Qt to Android mobile phones or tablets. This article uses Qt Creator running on a Windows operating system as the deployment tool. Android is a mobile operating system based on a modified version of the Linux kernel and more... This tutorial describes how to use Qt Creator to create Qt Quick apps for Android and Android iOS devices with Qt 6 as the minimum version of Qt and CMake as the build system. We implement a Qt Quick application that accelerates an SVG (Scalable Vector Graphics) image based on changing accelerometer values. Note:You must have the Qt Sensors module installed from Qt 6.2 or later to follow this tutorial. Setting up the development environment To build an application and run it on a mobile device, you need to set up a development environment for the device's platform and establish a connection between Qt Creator and the mobile device. To develop for Android devices, you must install Qt for Android and set up your development environment as described in Connecting Android Devices. To develop for iOS devices, you need to install Xcode and use it to set up the device. To do this, you need an Apple developer account and a certificate for the iOS Developer Program, which you can obtain from Apple. For more information, see Connecting an iOS device. Create a project Choose File > New Project > Application (Qt) > Quick Qt Application. Click Choose to open the Project Location dialog box. In the Name field, enter a name for the application. When naming your own projects, remember that you can't easily rename them later. In the Create in field, enter the path to the project files. You can easily move project folders later. Choose Next (or Continue on macOS) to open the Detect Build System dialog box. In the Build System field, select CMake as the build system you want to use to build and run the project. Note. If you decide to use qmake, the project setup instructions do not apply. Click Next to open the Define Project Details dialog box. In the Minimum required Qt version field, select Qt 6.2. Select Next to open the Translation File dialog box. Click Next to use the default settings and open the Specify Selection dialog box. Select Qt 6.2 or higher for the platforms you want to build the application for. If you want to create applications for mobile devices, you can also choose from Android and iOS packages. Note. The list shows the preferences you specify in Edit > Preferences > Preferences (on Windows and Linux) or Qt Creator > Preferences > Preferences (on macOS). For more information, see Adding Kits. Select NextProject management window. Review the project settings and select Done (or Done on macOS) to create the project. For more information on omitted options and other available wizard templates, see Creating Quick Qt Applications. Adding images as assets The main application window displays a bubble SVG image that moves around the screen as you tilt your device. We use Bluebubble.svg for this tutorial, but you can use any other image or component instead. In order for an image to be displayed at application startup, you must specify it as a resource in the RESOURCES section of the CMakeLists.txt file that the wizard created for you: .sv Creating the main source of the Accelbubble view: Image { id : bubble source: "Bluebubble.svg" smooth: true Then we add a custom property to set the image relative to the width and height of the main window: real centerX property: mainWindow.height / 2 real centerY property: mainWindow.height / 2 real bubbleCenter property: bubble.width / 2 x: centerX - bubbleCenter y: centerY - bubbleCenter Now we want to add code to move the bubble based on accelerometer sensor value. First we add the following import command: Then we add the Accelerometer component with the required properties: Accelerometer { id: accel dataRate: 100 active:true Then we add the following JavaScript functions that calculate the x and y coordinates 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 added the following JavaScript code to the onReadingChanged signal of the Accelerometer component so that the bubble moves when the accelerometerchange: 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)) 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 make sure that the position of the bubble is always within the limits of the screen. If the accelerometer returns a non-number (NaN) value, the value is ignored and the bubble's position is not updated. We'll add a SmoothedAnimation behavior to the bubble's x and y properties to make its movement appear smoother. Lock device orientation By default, the device display rotates when the device orientation changes from portrait to landscape. In this example, it would be better if the screen orientation was fixed. To lock portrait or landscape orientation on an Android device, specify it in the AndroidManifest.xml file that you can generate in Qt Creator. For more information, see Editing Manifest Files. To generate and use the manifest file, you need to specify the android package source directory QT\_ANDROID\_PACKAGE\_SOURCE\_DIR in CMakeLists.txt: set (property(TARGET appaccelbubble APPEND PROPERTY QT\_ANDROID\_PACKAGE\_SOURCE\_DIR because our android version must be older than C.I.CMAKE\_CURRENT\_S.qt.add\_executable(statement one Add completion action: (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 orientation of the device in the Info.plist file as the value of the MACOSX\_BUNDLE\_INFO\_PLIST variable: set (target\_properties (appaccelbubble PROPERTIESmy.example.com macosx\_bundle\_version \${project\_version} macosx\_bundle\_short\_version\_string \${project\_version.major}) \$(project\_version minor)macosx\_bunle.info.plist "\${(cmake current source dir)/info.plist" macosx\_specifying dependencies in the project file. Select Projects and update the CMake configuration with the following information about the Qt module: Sensors, Svg, Xml. The CMakeLists.txt file must contain the following entries that instruct CMake to search for the Qt installation and import the Qt sensors, Qt SVG, and Qt XML modules required by the application: find\_package(Qt6 6.2 COMPONENTS Quick Sensors REQUIRED Svg Xml) Qt file modules must be added to the target list link libraries. target\_link\_libraries tells CMake that the executable accelbubble uses the Qt Sensors, Qt SVG, and Qt XML modules by referencing the targets imported by calling the find\_package() method above. It adds the necessary arguments to the linker and ensures 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 the configuration changes. For more information about the CMakeLists.txt file, see Introduction to CMake. Launching the application You can now deploy the application to your device: Enable USB debugging on an Android device or developer mode on an iOS device. Connect the device to the development computer. If you're running Android 4.2.2, you need to verify the connection to allow USB debugging from your computer. To avoid such prompts every time you connect a device, select the Always allow from this computer check box, and then select OK. Press Ctrl+R to launch the app on your device. Files: accelbubble/Bluebubble.svgaccelbubble/main.qml accelbubble/main.qml



