

Contents

[Creating the Text Finder Project](#)
[Filling in the Missing Pieces](#)
[Designing the User Interface](#)
[Completing the Header File](#)
[Completing the Source File](#)
[Creating a Resource File](#)
[Compiling and Running Your Program](#)

The Qt Company is hiring!

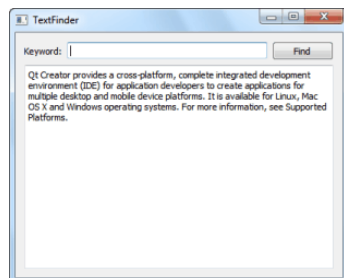
Work with the best
#QtPeople in business.
Check out
www.qt.io/careers

Qt Creator Manual

[Getting Started](#)
[Managing Projects](#)
[Designing User Interfaces](#)
[Coding](#)
[Building and Running](#)
[Testing](#)
[Advanced Use](#)
[Getting Help](#)

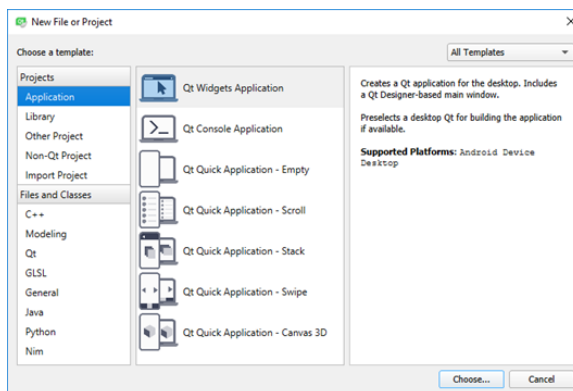
Creating a Qt Widget Based Application

This tutorial describes how to use Qt Creator to create a small Qt application, Text Finder. It is a simplified version of the Qt UI Tools [Text Finder Example](#). The application user interface is constructed from Qt widgets by using Qt Designer. The application logic is written in C++ by using the code editor.

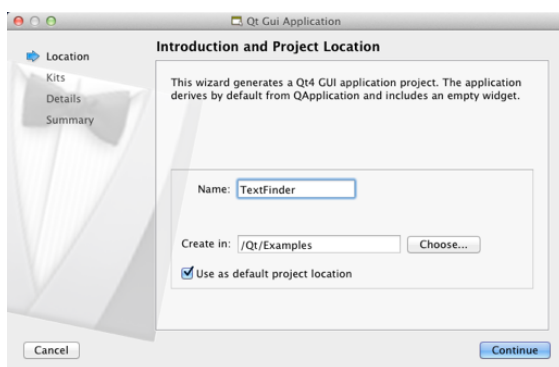


Creating the Text Finder Project

1. Select **File > New File or Project > Application > Qt Widgets Application > Choose**.



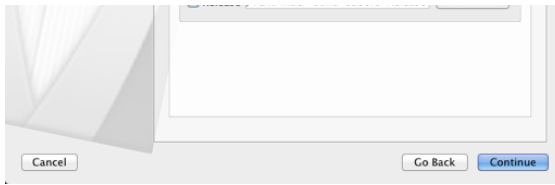
The **Introduction and Project Location** dialog opens.



2. In the **Name** field, type **TextFinder**.
3. In the **Create in** field, enter the path for the project files. For example, **C : \Qt\examples**, and then click **Next** (on Windows and Linux) or **Continue** (on macOS).

The **Kit Selection** dialog opens.

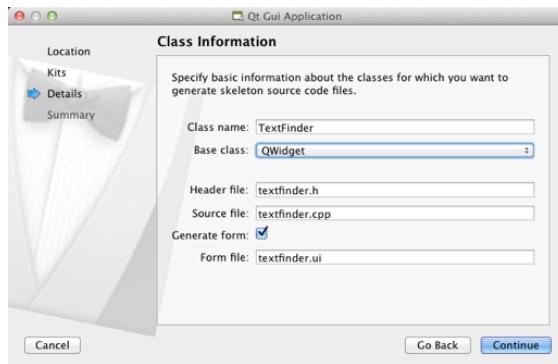




4. Select build and run kits for your project, and click **Next** or **Continue**.

Note: If only one kit is specified in **Tools > Options > Build & Run > Kits** (on Windows and Linux) or in **Qt Creator > Preferences Build & Run > Kits** (on macOS), this dialog is skipped.

The **Class Information** dialog opens.

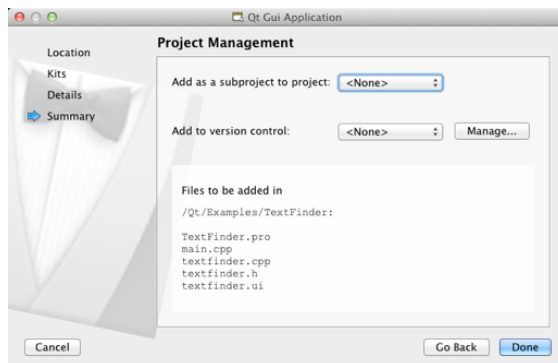


5. In the **Class name** field, type **TextFinder** as the class name.
6. In the **Base class** list, select **QWidget** as the base class type.

Note: The **Header file**, **Source file** and **Form file** fields are automatically updated to match the name of the class.

7. Click **Next** or **Continue**.

The **Project Management** dialog opens.

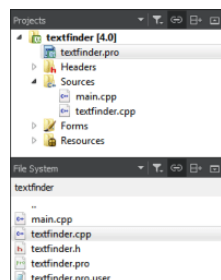


8. Review the project settings, and click **Finish** (on Windows and Linux) or **Done** (on macOS) to create the project.

Note: The project opens in the **Edit** mode, and these instructions are hidden. To return to these instructions, open the **Help** mode.

The **TextFinder** project now contains the following files:

- > textfinder.h
- > textfinder.cpp
- > main.cpp
- > textfinder.ui
- > textfinder.pro

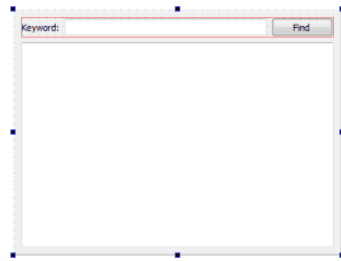


The .h and .cpp files come with the necessary boiler plate code. The .pro file is complete.

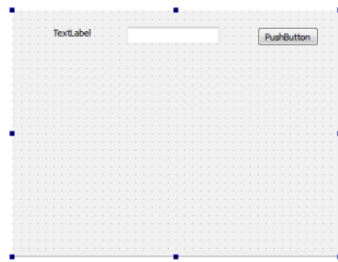
Filling in the Missing Pieces

Begin by designing the user interface and then move on to filling in the missing code. Finally, add the find functionality.

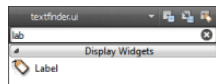
Designing the User Interface



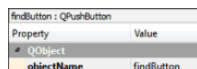
1. In the **Editor** mode, double-click the textfinder.ui file in the **Projects** view to launch the integrated Qt Designer.
2. Drag and drop the following widgets to the form:
 - > **Label (QLabel)**
 - > **Line Edit (QLineEdit)**
 - > **Push Button (QPushButton)**



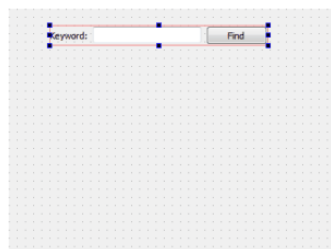
Note: To easily locate the widgets, use the search box at the top of the **Sidebar**. For example, to find the **Label** widget, start typing the word **label**.



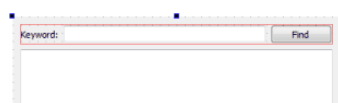
3. Double-click the **Label** widget and enter the text **Keyword**.
4. Double-click the **Push Button** widget and enter the text **Find**.
5. In the **Properties** pane, change the **objectName** to **findButton**.



6. Press **Ctrl+A** (or **Cmd+A**) to select the widgets and click **Lay out Horizontally** (or press **Ctrl+H** on Linux or Windows or **Ctrl+Shift+H** on macOS) to apply a horizontal layout (**QHBoxLayout**).



7. Drag and drop a **Text Edit** widget (**QTextEdit**) to the form.
8. Select the screen area and click **Lay out Vertically** (or press **Ctrl+L**) to apply a vertical layout (**QVBoxLayout**).





Applying the horizontal and vertical layouts ensures that the application UI scales to different screen sizes.

9. To call a find function when users press the **Find** button, you use the Qt signals and slots mechanism. A signal is emitted when a particular event occurs and a slot is a function that is called in response to a particular signal. Qt widgets have predefined signals and slots that you can use directly from Qt Designer. To add a slot for the find function:

- Right-click the **Find** button to open a context-menu.
- Select **Go to Slot > clicked()**, and then select OK.

A private slot, `on_findButton_clicked()`, is added to the header file, `textfinder.h` and a private function, `TextFinder::on_findButton_clicked()`, is added to the source file, `textfinder.cpp`.

10. Press **Ctrl+S** (or **Cmd+S**) to save your changes.

For more information about designing forms with Qt Designer, see the [Qt Designer Manual](#).

Completing the Header File

The `textfinder.h` file already has the necessary `#includes`, a constructor, a destructor, and the `Ui` object. You need to add a private function, `loadTextFile()`, to read and display the contents of the input text file in the `QTextEdit`.

1. In the **Projects** pane in the **Edit view**, double-click the `textfinder.h` file to open it for editing.
2. Add a private function to the **private** section, after the `Ui::TextFinder` pointer, as illustrated by the following code snippet:

```
private slots:
    void on_findButton_clicked();

private:
    Ui::TextFinder *ui;
    void loadTextFile();
```

Completing the Source File

Now that the header file is complete, move on to the source file, `textfinder.cpp`.

1. In the **Projects** pane in the **Edit view**, double-click the `textfinder.cpp` file to open it for editing.
2. Add code to load a text file using `QFile`, read it with `QTextStream`, and then display it on `textEdit` with `QTextEdit::setPlainText()`. This is illustrated by the following code snippet:

```
void TextFinder::loadTextFile()
{
    QFile inputFile(":/input.txt");
    inputFile.open(QIODevice::ReadOnly);

    QTextStream in(&inputFile);
    QString line = in.readAll();
    inputFile.close();

    ui->textEdit->setPlainText(line);
    QTextCursor cursor = ui->textEdit->textCursor();
    cursor.movePosition(QTextCursor::Start, QTextCursor::MoveAnchor, 1);
}
```

3. To use `QFile` and `QTextStream`, add the following `#includes` to `textfinder.cpp`:

```
#include <QFile>
#include <QTextStream>
```

4. For the `on_findButton_clicked()` slot, add code to extract the search string and use the `QTextEdit::find()` function to look for the search string within the text file. This is illustrated by the following code snippet:

```
void TextFinder::on_findButton_clicked()
{
    QString searchString = ui->lineEdit->text();
    ui->textEdit->find(searchString, QTextDocument::FindWholeWords);
}
```

5. Once both of these functions are complete, add a line to call `loadTextFile()` in the constructor, as illustrated by the following code snippet:

```
TextFinder::TextFinder(QWidget *parent)
: QWidget(parent), ui(new Ui::TextFinder)
```

```

    connect(parent, Qt::NEW_SLOT, this, &TextFinder::on_findButton_clicked);
{
    ui->setUi(this);
    loadTextFile();
}

```

The `on_findButton_clicked()` slot is called automatically in the uic generated `ui_textfinder.h` file by this line of code:

```

QMetaObject::connectSlotsByName(TextFinder);

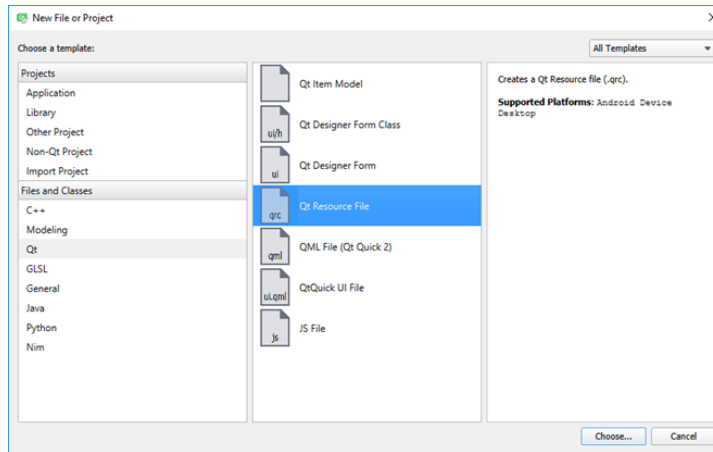
```

Creating a Resource File

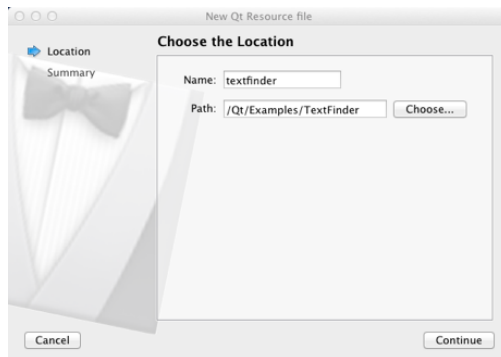
You need a resource file (.qrc) within which you embed the input text file. The input file can be any .txt file with a paragraph of text. Create a text file called `input.txt` and store it in the `textfinder` folder.

To add a resource file:

1. Select **File > New File or Project > Qt > Qt Resource File > Choose**.

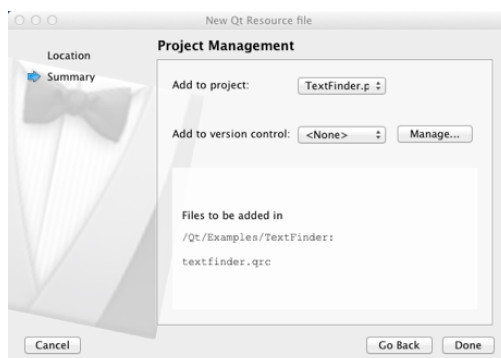


The **Choose the Location** dialog opens.



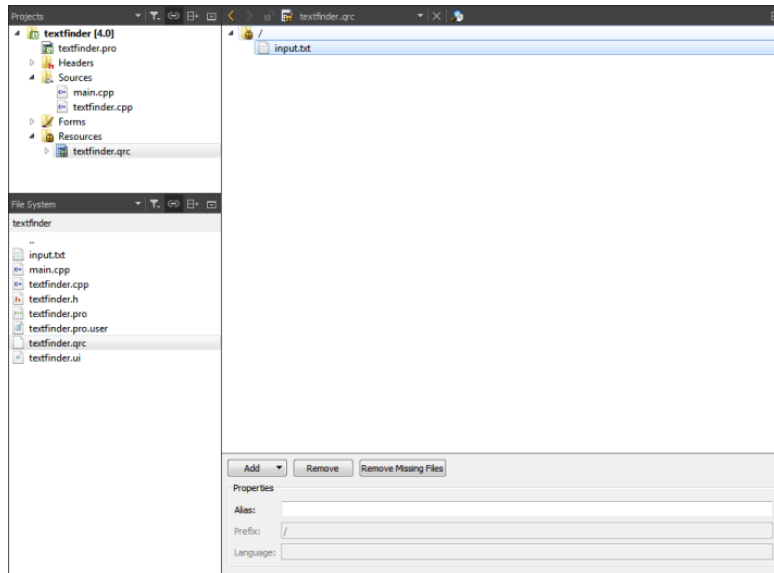
2. In the **Name** field, enter `textfinder`.
3. In the **Path** field, enter `C:\Qt\examples\TextFinder`, and click **Next** or **Continue**.

The **Project Management** dialog opens.



4. In the **Add to project** field, select `TextFinder.p` and click **Finish** or **Done** to open the file in the code editor.

4. In the **Add to project** field, select **textfinder.pro** and click **Finish** or **Done** to open the file in the code editor.
5. Select **Add > Add Prefix**.
6. In the **Prefix** field, replace the default prefix with a slash (/).
7. Select **Add > Add Files**, to locate and add input.txt.



Compiling and Running Your Program

Now that you have all the necessary files, click the ► button to compile and run your program.

[< Creating a Qt Quick Application](#)

[Creating a Mobile Application >](#)

© 2017 The Qt Company Ltd. Documentation contributions included herein are the copyrights of their respective owners. The documentation provided herein is licensed under the terms of the [GNU Free Documentation License version 1.3](#) as published by the Free Software Foundation. Qt and respective logos are trademarks of The Qt Company Ltd in Finland and/or other countries worldwide. All other trademarks are property of their respective owners.

Download

Start for Free
Qt for Application Development
Qt for Device Creation
Qt Open Source
Terms & Conditions
Licensing FAQ

Product

Qt in Use
Qt for Application Development
Qt for Device Creation
Commercial Features
Qt Creator IDE
Qt Quick

Services

Technology Evaluation
Proof of Concept
Design & Implementation
Productization
Qt Training
Partner Network

Developers

Documentation
Examples & Tutorials
Development Tools
Wiki
Forums
Contribute to Qt

About us

Training & Events
Resource Center
News
Careers
Locations
Contact Us

