# An OPCUG Product Review

OPCUG

Users helping users
for over 40 years

## Qt Creator – An IDE for C++

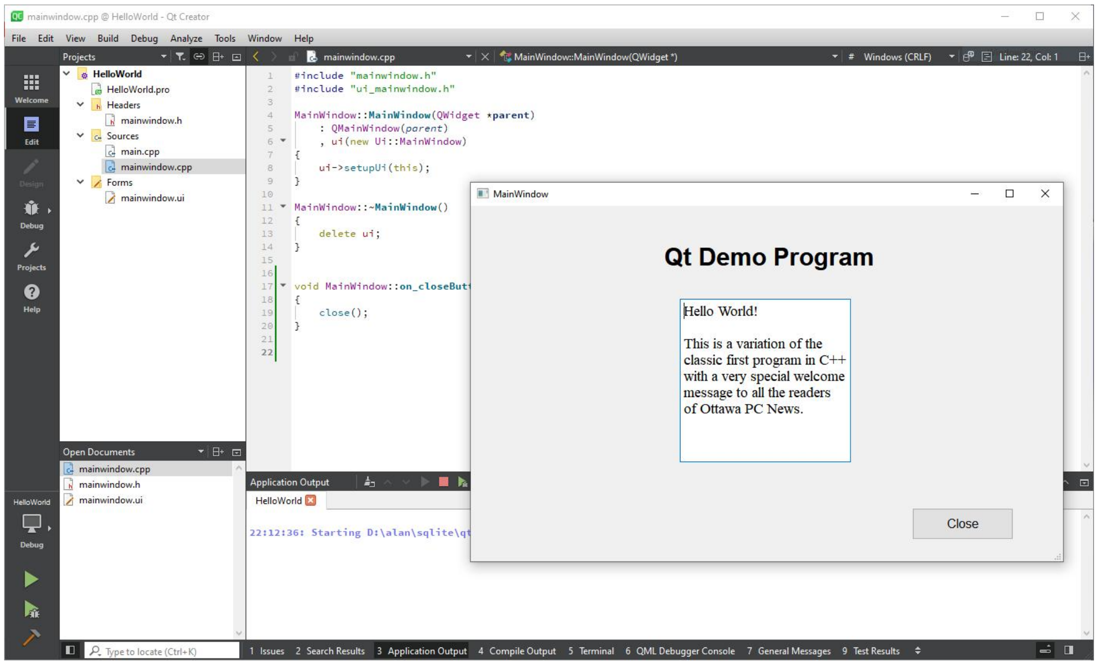
*by Alan German*

As with many programming languages, while it's possible to get started using C++ with just a text editor and a compiler, life is simpler if you use an Integrated Development Environment (IDE). Qt Creator is one such IDE which has the benefit of having an open-source version that provides a simple yet powerful development platform.

Interestingly, Qt is apparently pronounced as "cute" rather than Q-t, although you may only run into this if you watch video tutorials. It's also worth noting that Qt Creator is primarily a commercial product that has its roots in the open- source community. In consequence, there are many on-line help files and video tutorials available that allow non-professional programmers to make use of the open-source version. The software is also cross-platform, with versions for Windows, Linux and macOS. For this article we will focus on the Windows version of Qt Creator.

Downloading the Windows version of Qt for open-source use (https://www.qt.io/download-qt-installer-oss) provides an executable file. Running the installer requires setting up a Qt account. There is no charge for this; however, I was subsequently contacted by E-mail by an agent who was clearly under the impression that my "company" might need information about the GPL and commercial licensing requirements of the software. Perhaps this was because I used my opcug.ca mail address to register. In any case, there are licensing issues that need to be considered if you are planning to distribute software that you develop using the Qt platform.

Running Qt Creator produces a typical multi-window programming environment as shown in the screenshot. There is a directory tree on the left side allowing navigation through a project's files and folders. The main window is an editor where code can be reviewed and modified. Note the tabs along the bottom edge of the window. Selecting *3. Application Output* is necessary to display a program's output when the code is compiled and run. There are also various program controls down the left edge of the window, including the green triangle that can be used to run a program, and a regular File-Edit-View menu across the top edge of the window.



```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}


void MainWindow::on_closeBut
{
    close();
}

```

```
22:12:36: Starting D:\alan\sqlite\qt
```

Qt Demo Program

Hello World!

This is a variation of the classic first program in C++ with a very special welcome message to all the readers of Ottawa PC News.

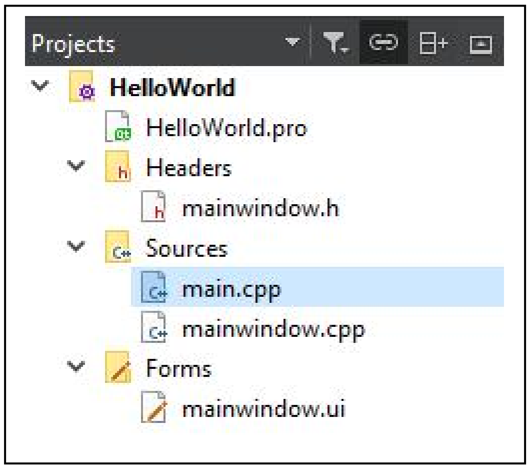
C++ and Qt Creator make it fairly easy to develop software with a GUI interface. The screen shot shows the output of a demonstration program which is a variation of the classic hello-world application. Note the main elements, the program's title, a text box, and a control button that can be used to close the running program.

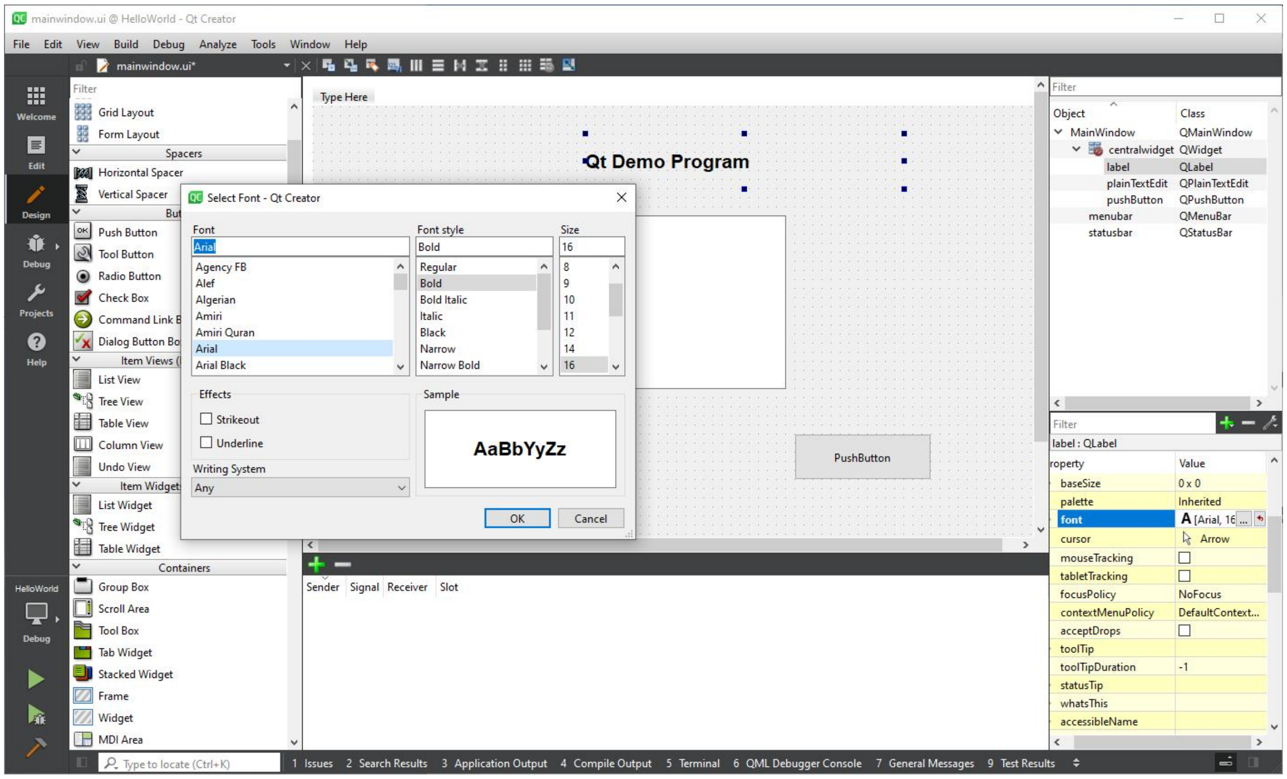
To develop this program, we select *Create Project* from Qt's main screen. This brings up a selection window where we can choose between various types of project, each of which has a number of subsidiary selections. We will choose *Application (Qt)* and a *Qt Widgets Application*. This will create a main window (in which we can arrange various design elements), C++ source code files, and the header files required for the project.

We give the project the name HelloWorld and browse for a folder (D:\HelloWorld) in which to store the project files. In the remaining screens we can leave the project parameters at their defaults so that we employ *qmake* to build an executable program, use *MainWindow* as the class, don't enable translation, and use the *MinGW 64-bit kit* (Minimal GNU compiler for Windows) as the compiler. All of these components are part of the installed version of Qt Creator and are handled automatically by the software.

The project now consists of a number of files and folders which are displayed in the directory tree. If we run the program code at this point, the GUI output window is displayed but it is blank. We need to add some content to the *mainwindow.ui* form.

This is easily done just by double-clicking on the name of the form. This opens the *Design* window where we can add design elements, such as a *Label*, a *Plain Text Edit* box, and a *Push Button*, to the main window area. Various attributes for each of the design elements can be changed using the options available in the right sidebar of the design window. So, we can add "Qt Demo Program" to the label as our program's title, enter the "Hello World!" text into the Plain Text Edit box, and change the text on the Push Button to read "Close".


Projects
- HelloWorld
  - HelloWorld.pro
  - Headers
    - mainwindow.h
  - Sources
    - main.cpp
    - mainwindow.cpp
  - Forms
    - mainwindow.ui



Select Font - Qt Creator

| Font | Font style | Size |
|---|---|---|
| Arial | Bold | 16 |
| Agency FB | Regular | 8 |
| Alef | Bold | 9 |
| Algerian | Bold Italic | 10 |
| Amiri | Italic | 11 |
| Amiri Quran | Black | 12 |
| Arial | Narrow | 14 |
| Arial Black | Narrow Bold | 16 |

Effects: ☐ Strikeout ☐ Underline
Writing System: Any
Sample: AaBbYyZz
OK Cancel

| Object | Class |
|---|---|
| MainWindow | QMainWindow |
| centralwidget | QWidget |
| label | QLabel |
| plainTextEdit | QPlainTextEdit |
| pushButton | QPushButton |
| menubar | QMenuBar |
| statusbar | QStatusBar |

label : QLabel

| roperty | Value |
|---|---|
| baseSize | 0 x 0 |
| palette | Inherited |
| font | A [Arial, 16... |
| cursor | Arrow |
| mouseTracking | ☐ |
| tabletTracking | ☐ |
| focusPolicy | NoFocus |
| contextMenuPolicy | DefaultContext... |
| acceptDrops | ☐ |
| toolTip | |
| toolTipDuration | -1 |
| statusTip | |
| whatsThis | |
| accessibleName | |

Note that simply labelling the *Push Button* as "Close" doesn't do anything to have this action occur when the button is pressed while the program is running. To achieve this end, we need to take note of the content of the lower-central portion of the design window where we see *Sender* | *Signal* | *Receiver* | *Slot* displayed.

C++ is an object-oriented language and uses the concept of one object sending a message to a second object, with the second object then taking some form of action. The process of signals and slots is a cornerstone of Qt and is the basis for its event handling. In the present case, the user clicking a control button is an event which creates a signal. The receiver of the signal is a slot, which takes the form of a section of C++ code that will instruct the program to stop.

To create the signal/slot process, we right-click on the *Push Button*, select *Go to slot* in the pop-up menu, and choose the signal labelled *QAbstract Button clicked()*. This takes us to the program editing window with the file *mainwindow.cpp* already loaded and a code snippet for *on_pushButton_clicked* displayed. The code that we wish to have activated is quite simple - *close();* - which we add between the curly brackets, making the completed function:

```
void MainWindow::on_pushButton_clicked()
{
      close();
}
```

The result of this section of code is fairly evident. When, in the main program window, the user clicks the Close button, control is transferred to the *on_pushButton_clicked* function and the close command cause the program to terminate.

Our Hello World program is a very simple example of GUI programming using C++ and Qt Creator. However, there are a number of features of the software which may not be intuitive and are worthy of note. In particular, Qt Creator has two build modes, *Debug* and *Release*. Essentially the debug mode creates a larger executable file since this version contains many hooks that are used in the debugging process. When a program has been thoroughly debugged, it can be built as a release version which is considerably smaller.

By default, Qt creates multiple folders to separate the initial project files and the output files. For example, our simple program creates the folder *D:\HelloWorld\HelloWorld* which contains the main project file, *HelloWorld.pro*, the C++ source code files, and the associated headers. In addition, the folder *D:\HelloWorld\build-HelloWorld-MinGW64-bit-Debug* contains various make, object, and executable files.

The initial debugging process is very useful, especially for beginners in C++ programming. For example, if we had missed entering the final semi-colon in the close(); command, this line in the editor would be followed by a line, highlighted in red, indicating *Expected ';' after expression (fix available)*. This error message is preceded by a small lightbulb icon. Clicking on this icon applies the fix and inserts the required semi-colon at the end of the command. However, the system is much more powerful than this as the *Debug* menu option allows the insertion of breakpoints in the code. When the running program hits a breakpoint, execution pauses, and a number of parameters are displayed. For example, this can include the values of certain variables, such as counters and computed results, allowing the programmer to verify their validity at a specific point in the program.

To re-open a project, we can use the *Open Project* button on Qt Creator's welcome screen. This requires browsing to the project folder and opening the file *HelloWorld.pro*. Alternatively, selecting the *Projects* button on the welcome screen displays a list of recently used projects in which, for our purposes, the HelloWorld project will be the first entry.

The Windows version of Qt Creator has a set of tutorials built in. For example, selecting *Tutorials - Creating a Qt Widget Based Application* from the welcome screen provides assistance on building an application very similar to our HelloWorld program. In addition, the *Examples* button provides access to a number of projects, including a calculator, a document viewer, and a media player. And, as noted previously, there are many similar resources available on-line.

Running the debug version of our Hello World program produces an executable file, HelloWorld.exe, with a size of 1.6 MB. In contrast, the release version of the program, produced using the same source code, is a mere 28 KB. However, the availability of a smaller executable is not the end of the release story. Attempting to run this stand-alone program on another computer will almost certainly fail with a number of errors messages to the effect that various files are unavailable, e.g. *Qt6Core.dll was not found*. The problem is that the installed version of Qt Creator has the required dynamic link libraries (DLL) available whereas other users' computers may not. This is where consideration needs to be given to how to package a release version of the newly-developed program and the consequent licensing implications. However, I leave such considerations to would-be software developers/distributors.

As a learning tool, and even as a development platform for applications to be used locally, Qt Creator offers a comprehensive and powerful programming environment. The software has its quirks, and C++ itself may exhibit a fairly steep learning curve for users unfamiliar with object-oriented programing. Nevertheless, the open-source version of the software is free to use and there is a wide variety of assistance readily available online. So, if you want to become a C++ programmer, Qt Creator is a good way to get started.

## Bottom Line

Qt Creator (Open Source and Commercial)
Version 6.5.2
Qt Group, Helsinki, Finland
https://www.qt.io/product/development-tools