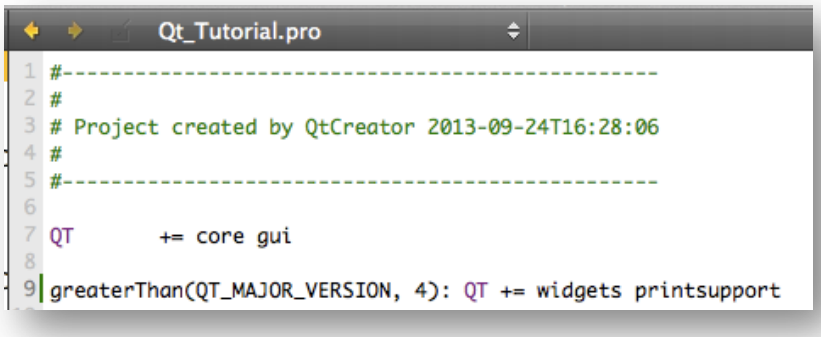
QT VISUALIZATION

Doug Maunder & Ashton Jackson

INSTALLATION & SETUP

1. Download Qt Creator: http://qt-project.org/downloads.
2. Download the most recent version of QCustomPlot.tar.gz:
   a. http://www.qcustomplot.com/index.php/download
3. In Qt Creator, select *New File or Project* under the File menu.
4. Click *Applications, Qt Gui Application,* and *Choose.* Name your project, and click *Continue* three times to keep the default settings. Click *Done.*
5. Right click on your main project file in Qt Creator's Edit mode, and select *Add Existing Files.* Select qcustomplot.h and qcustomplot.cpp, and click *Open.*
6. Open your .pro file. In the 9th line, add the word "printsupport" after "widget" as seen below, if it's not already there.



7. Open your mainwindow.ui file form the Forms folder. Drag and drop the Widget icon from the Containers menu onto the gray grid.
8. Right click within the blue box that appears, click *Promote To…,* and enter the promoted class name "QCustomPlot".
9. Click *Add* and *Promote.* You are now ready to begin using QCustomPlot for data visualization.

Notes: Windows users may find it beneficial to install Visual Studio, as that is the default compiler kit for Qt Creator. Changing compilers is also possible, but seems to introduce a few interesting complications.

Mac users will need to enter "cache()" at the top of their main .pro file to overcome an inherent bug in Qt Creator.

PLOTTING FUNCTIONS – THE BASICS

1. Inside mainwindow.cpp add "setGeometry(400, 250, 542, 390);" to define sufficient workspace.
2. Either in this parent function or in a new user defined function begin by initializing your axis using "QVector<type>".
3. Assign the generated/collected data points to the newly created axis variables. This can generally be done via standard C++ loops.
4. With the data assigned it can be sent to a custom plot. For now a basic plot will be created using the following code:

```
1.          // create graph and assign data to it:
2.          customPlot->addGraph();
3.          customPlot->graph(0)->setData(x, y);
4.          // give the axes some labels:
5.          customPlot->xAxis->setLabel("Time");
6.          customPlot->yAxis->setLabel("Sensor Read");
7.          // set axes ranges, so we see all data:
8.          customPlot->xAxis->setRange(XX, XX);
9.          customPlot->yAxis->setRange(XX, XX);
```

   Make sure to set an appropriate range (denoted by XX in lines 8 and 9).

5. Any additional functions that you might include need to be added to mainwindow.h (the header file) before compiling.

CUSTOMIZING GRAPHICS

The **look of the graph** is characterized by many factors, all of which can be modified. Here are the most important ones:

- **Line style**: Call QCPGraph::setLineStyle.
- **Line pen**: All pens the QPainter-framework provides are available, e.g. solid, dashed, dotted, different widths, colors, transparency, etc. Set the configured pen via QCPGraph::setPen.
- **Scatter symbol**: Call QCPGraph::setScatterStyle to change the look of the scatter point symbols. If you don't want any scatter symbols to show at each data point, use QCPScatterStyle::ssNone.
- **Fills under graph or between two graphs**: All brushes the QPainter-framework provides can be used in graph fills: solid, various patterns, textures, gradients, colors, transparency, etc. Set the configured brush via QCPGraph::setBrush.

The **look of the axis** can be modified by changing the pens they are painted with and the fonts their labels use. Here's a quick summary of the most important properties: setBasePen, setTickPen, setTickLength, setSubTickLength,setSubTickPen, set TickLabelFont, setLabelFont, setTickLabelPadding, setLabelPadding. You can reverse an axis (e.g. make the values decrease instead of increase from left to right) with setRangeReversed.