Readme for using QT to build a limited GUI for ROS

Michaloski, John L. (Fed)

11/17/2016 5:13:00 PM

RcsGuiReadme.docx

Qt Creator IDE is a cross-platform integrated development environment (IDE) tailored to the needs of Qt developers. It includes an advanced C++ code editor, integrated GUI layout and forms designer, project and build management tools, integrated, context-sensitive help system, visual debugger, rapid code navigation tools, and supports multiple platforms. This document presents the background/notes to build a QT C++ project that interfaces to the Robot Operating System (ROS) to read robot poses and joint values. By no means is the document a complete, but may save you time if you are just beginning. The internet has a tendency to scatter tidbits of information everywhere, and with repeated searches including dry holes, obsolete or unrelated searches, you will find what you need.

This implementation provides a QT GUI using a UI and NOT USING CATKIN. The necessary libraries for boost, Eigen and ROS are manually entered QT qmake file.

QT is a C++ compile with graphical interface that is portable across many platforms. For background on QT, the web site <https://wiki.qt.io/Qt_for_Beginners> has a nice tutorial for beginners.

## INSTALL QTCREATOR on UBUNTU

For the latest QTCreator IDE, QT has a tutorial for installation of Qt 5.7.0 to Ubuntu 12.10. It may be used also for newer versions of Qt and Ubuntu. <https://wiki.qt.io/Install_Qt_5_on_Ubuntu>

For generic QT on Ubuntu the following worked:

> sudo apt-get install qtcreator

This installation should include QT UI designer.

## RUNNING QTCREATOR in Conjunction with ROS

## Running QtCreator

ROS uses environment variables to define the version, master IP, and other ROS parameters that are generally ROS dependent as well as package workspace dependent. If you are building a ROS workspace, you can launch QTCreator with full ROS environment set

> source devel/setup.bash

> qtcreator (inherit env variables?)

## How to put generated files (e.g. object files) into a separate folder when using Qt/qmake?

<http://stackoverflow.com/questions/3440387/how-to-put-generated-files-e-g-object-files-into-a-separate-folder-when-using#23007090>

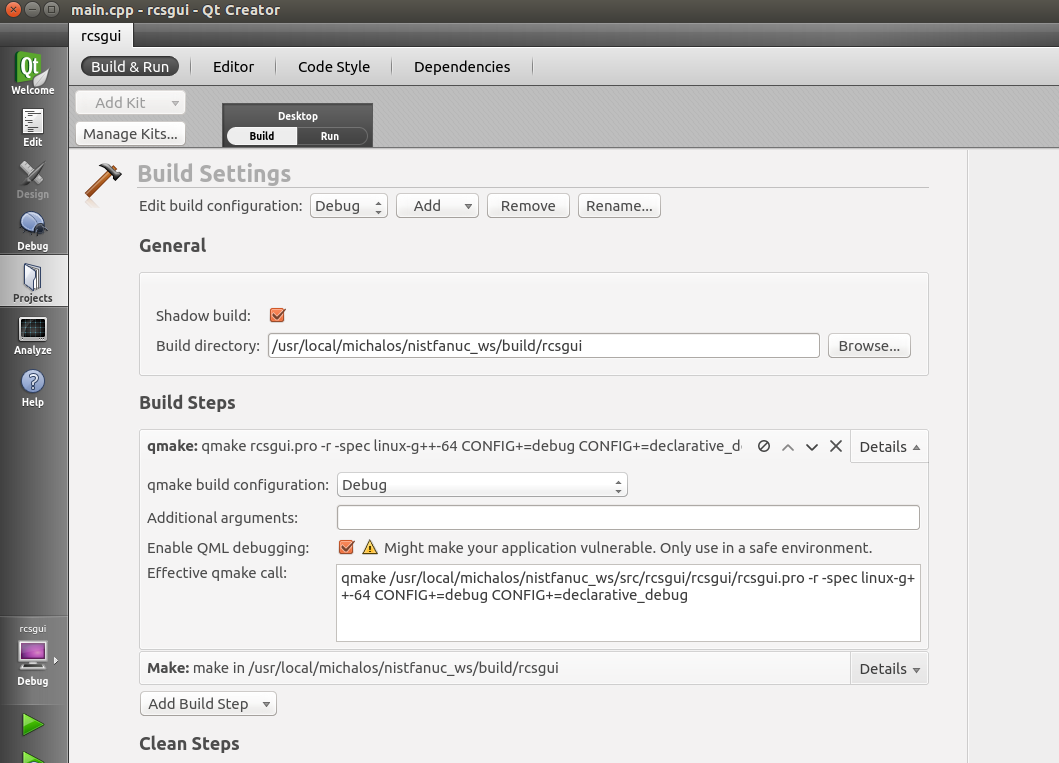
You can add the following lines to your \*.pro file:

DESTDIR=../../devel/rcsgui #Target file directory

OBJECTS\_DIR=generated\_files #Intermediate object files directory

MOC\_DIR=generated\_files #Intermediate moc files directory

But you must first change the location of the Build directory by hitting the "Browse" button and the above are relative to this directory as shown in the figure below:



## Editing the Qmake.pro file:

### Add ROS functionality to QT Creator project

The headers and libraries of the ROS depends on the version, so for the indigo version this is the main ROS header path:

/opt/ros/indigo/include/ros/ros.h

Thus, ROS headers and library must be included:

unix{

LIBS += -L/opt/ros/indigo/lib -lroscpp -lroslib -lrosconsole -lrostime -ltf -ltf2

LIBS += -lrosconsole\_bridge -lrosconsole\_log4cxx -lrosconsole\_backend\_interface

}

The list of ROS libraries grew as the needs of the project grew. Only basic ROS was used (no moveit), and only ROS package distributions were used for now. It would not be hard to add private workspace package libraries and include files.

### Adding Boost functionality

Add boost library: add to xxxx.pro qmake file (you must open qmake.pro file in the project source listing):

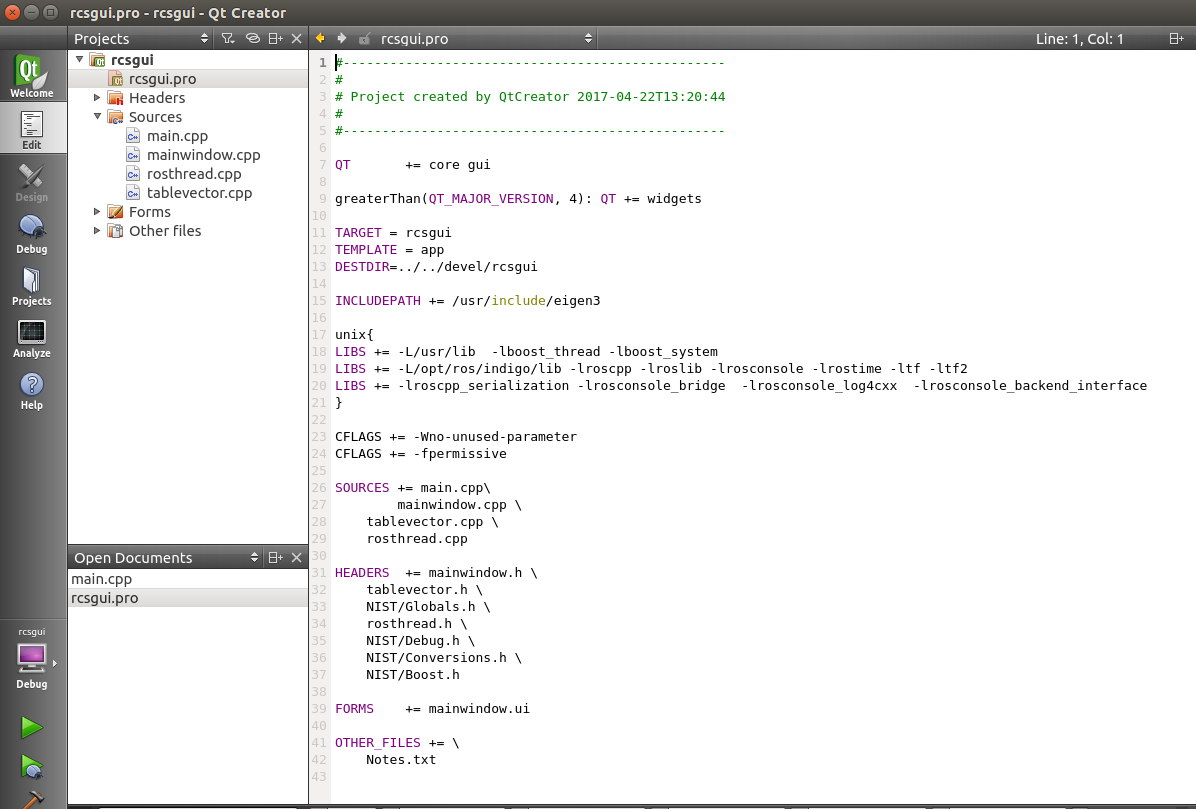
unix{

LIBS += -L/usr/lib -lboost\_thread -lboost\_system

}

### Adding Eigen Headers (include all compiled code):

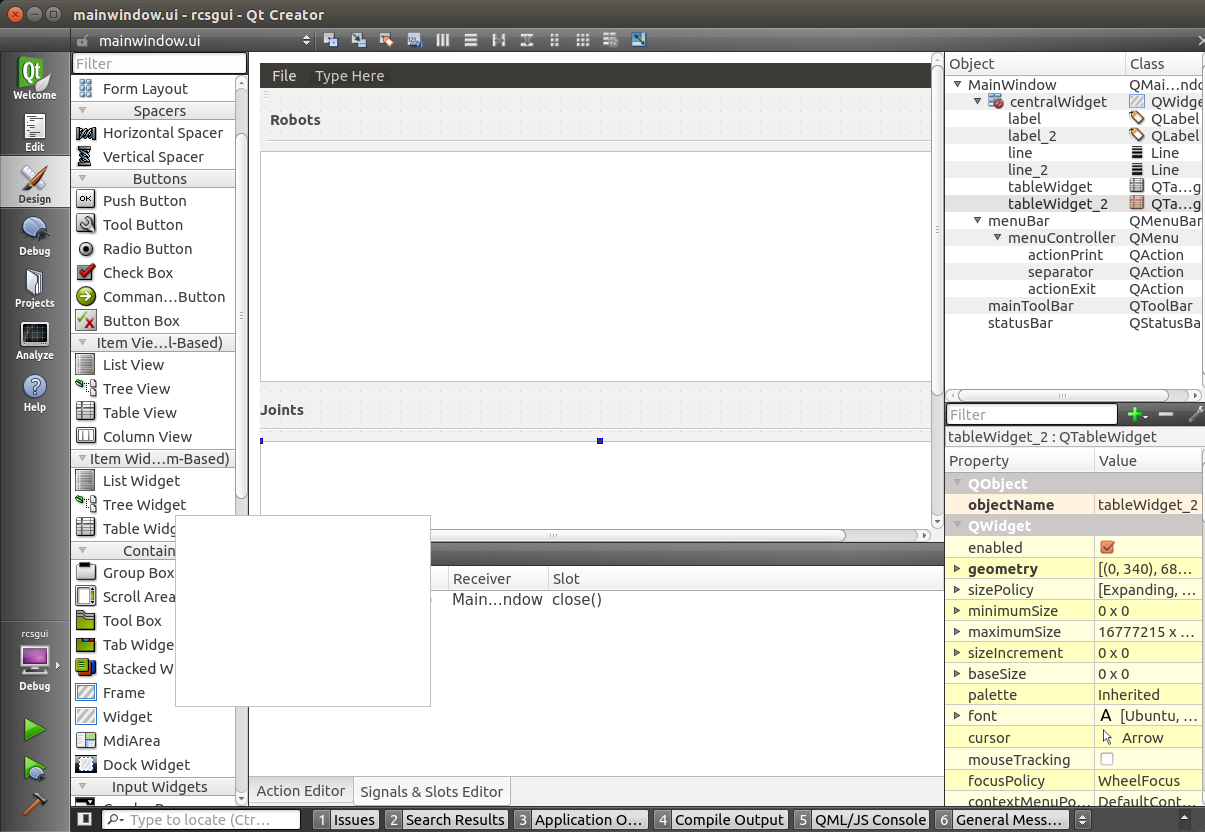
INCLUDEPATH += /usr/include/eigen3



## QTCREATOR GUI

### QT Table Widget

<http://toto-share.com/2011/11/qt-qtablewidget-tutorial/>



## Quitting a QT Application

From: http://stackoverflow.com/questions/8026101/correct-way-to-quit-a-qt-program#8026335

QApplication is derived from QCoreApplication and thereby inherits quit() which is a public slot of QCoreApplication, so there is no difference between QApplication::quit() and QCoreApplication::quit().

## Adding a C++ Class to your Project

Right click on project name, and then "Add New"

File->Add New -> C++ Class

## QThreads

http://blog.debao.me/2013/08/how-to-use-qthread-in-the-right-way-part-1/

## Catch Qt QMainWindow at Close (Hitting X Close Button)

Of note, the QT GUI must run in the main thread, and the ROS code ran as a background Qthread. So when the user clicked either File->Exit or the Close App (X) button, both had to terminate and waith till the ROS thread was done. The following code seems to work.

void MainWindow::closeEvent(QCloseEvent \*event)

{

// This was the code to perform on shutdown

// Want background ROS thread to

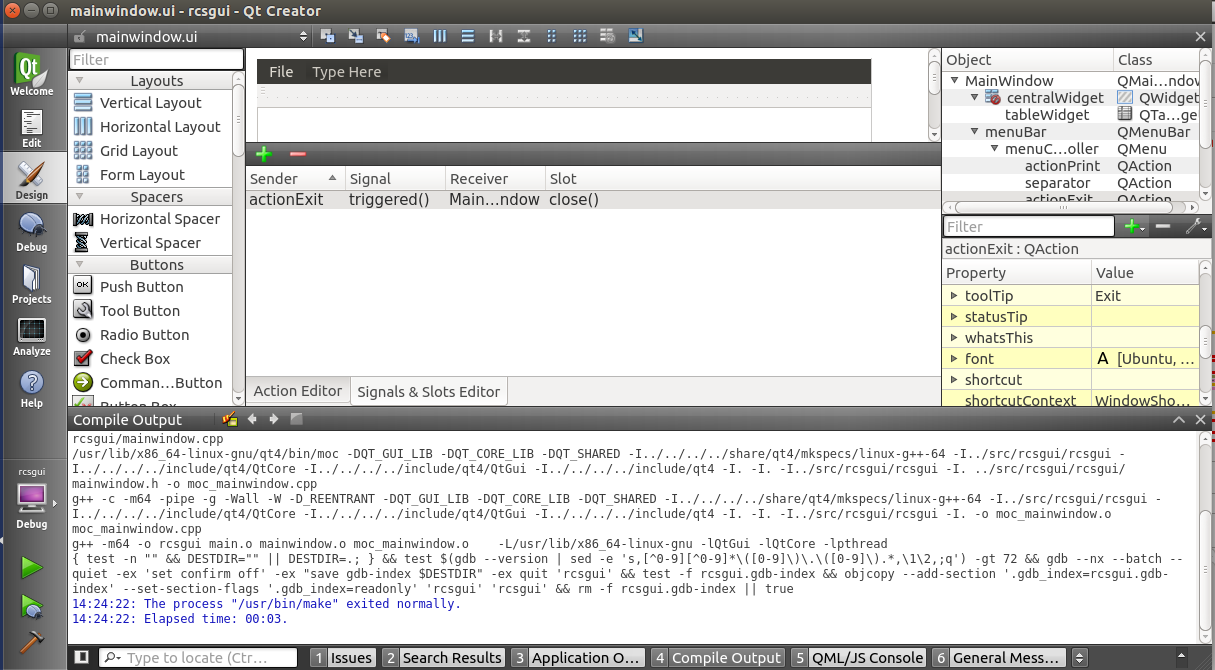
rosThread.abort=true;

rosThread.wait(1000);

usleep(10000);

}

This was done by going into the UI editor (Design button) and going into the "Signals & Slots" Editor and adding an actionExit (corresponding to Clickin X on the main window) and calling the Slot "close()" defined in the MainWindow.h header and code shown above.



## QString

QT has its own implementation for many objects: string, settings, gui, etc.

### How to convert QString to std::string

QString qs;

// do things

std::cout << qs.toStdString() << std::endl;

### Convert std::string to QString

There's a QString function called fromUtf8 that takes a const char\*:

QString str = QString::fromUtf8(content.c\_str());

## Qt Ini file

Use the QSettings construct to read an ini file. QT uses "/" to allow for a class hierarchy of section and parameter names. Thus, "Section/Parameter" will give you a section and the parameter key value under it.

Below is code to read the keys from under a section.

QSettings settings("test.ini", QSettings::IniFormat);

settings.beginGroup("TAG1");

const QStringList childKeys = settings.childKeys();

QStringList values;

foreach (const QString &childKey, childKeys)

values << settings.value(childKey).toString();

settings.endGroup();

## How to read a value using QSetting if the value contains comma character

Comma character is treated as list separator by QSettings. INI values with commas are parsed as string lists. You can convert a string list back to original string as follows:

std::vector<std::string> ParseQVariant(QVariant & value)

{

std::vector<std::string> values;

if (value.type() == QVariant::StringList) {

QStringList list(value.toStringList());

foreach( QString str, list) {

values.push\_back(str.toStdString());

}

} else {

values.push\_back( value.toString().toStdString());

}

return values;

}

## Using QT to debug existing exe application

<http://doc.qt.io/qtcreator/creator-debugger-operating-modes.html>

**Start External** to start and debug processes without a proper Qt Creator project setup, either locally or on a remote machine.