[How to execute a cmd command using QProcess?](https://stackoverflow.com/questions/21596104/how-to-execute-a-cmd-command-using-qprocess)

QProcess::startDetached will take the first parameter as the command to execute and the following parameters, delimited by a space, will be interpreted as separate arguments to the command.

Therefore, in this case: -

QProcess::startDetached("cmd /c net stop \"MyService\"");

The function sees **cmd** as the command and passes /c, net, stop and "MyService" as arguments to cmd. However, other than /c, the others are parsed separately and are not valid arguments.

What you need to do is use quotes around the "net stop \"MyService\" to pass it as a single argument, so that would give you: -

QProcess::startDetached("cmd /c \"net stop \"MyService\"\"");

Alternatively, using the string list you could use: -

QProcess::startDetached("cmd", QStringList() << "/c" << "net stop \"MyService\"");

# [Qt - reading from a text file](https://stackoverflow.com/questions/2612103/qt-reading-from-a-text-file)

You have to replace string line

QString line = in.readLine();

into while:

QFile file("/home/hamad/lesson11.txt");

if(!file.open(QIODevice::ReadOnly)) {

QMessageBox::information(0, "error", file.errorString());

}

QTextStream in(&file);

while(!in.atEnd()) {

QString line = in.readLine();

QStringList fields = line.split(",");

model->appendRow(fields);

}

file.close();

# [QT C++ remove file with \* (name contains)](https://stackoverflow.com/questions/52133207/qt-c-remove-file-with-name-contains)

The main task is to filter the files so we can use QDir with the nameFilter as shown below:

QDir dir("/path/of/directory", {"test\*.txt"});

for(const QString & filename: dir.entryList()){

dir.remove(filename);

}

Or use QDirIterator:

QDirIterator it("/path/of/directory", {"test\*.txt"});

while (it.hasNext())

QFile(it.next()).remove();

//QDir().remove(it.next());

QMainWindow – widget window title

QIcon icon("src/Me.png");

setWindowIcon(icon);

setWindowTitle(QStringLiteral("主窗口"));

QWidget

QIcon icon("src/Me.png");

setWindowIcon(icon);

setWindowTitle(QStringLiteral("XX窗口"));

QApplication

QApplication a(argc, argv);

QIcon icon("src/Me.png");

# [Get the Highlighted/Selected text](https://stackoverflow.com/questions/5379120/get-the-highlighted-selected-text)

Getting the text the user has selected is relatively simple. There's no benefit to be gained by involving jQuery since you need nothing other than the window and document objects.

function getSelectionText() {

var text = "";

if (window.getSelection) {

text = window.getSelection().toString();

} else if (document.selection && document.selection.type != "Control") {

text = document.selection.createRange().text;

}

return text;

}

If you're interested in an implementation that will also deal with selections in <textarea> and texty <input> elements, you could use the following. Since it's now 2016 I'm omitting the code required for IE <= 8 support but I've posted stuff for that in many places on SO.

function getSelectionText() {

var text = "";

var activeEl = document.activeElement;

var activeElTagName = activeEl ? activeEl.tagName.toLowerCase() : null;

if (

(activeElTagName == "textarea") || (activeElTagName == "input" &&

/^(?:text|search|password|tel|url)$/i.test(activeEl.type)) &&

(typeof activeEl.selectionStart == "number")

) {

text = activeEl.value.slice(activeEl.selectionStart, activeEl.selectionEnd);

} else if (window.getSelection) {

text = window.getSelection().toString();

}

return text;

}

document.onmouseup = document.onkeyup = document.onselectionchange = function() {

document.getElementById("sel").value = getSelectionText();

};

Selection:

<br>

<textarea id="sel" rows="3" cols="50"></textarea>

<p>Please select some text.</p>

<input value="Some text in a text input">

<br>

<input type="search" value="Some text in a search input">

<br>

<input type="tel" value="4872349749823">

<br>

<textarea>Some text in a textarea</textarea>

# [QtWebEngine - synchronously execute JavaScript to read function result](https://stackoverflow.com/questions/45330481/qtwebengine-synchronously-execute-javascript-to-read-function-result)

The callback is asynchronous because the JavaScript execution occurs not only in another thread but in another process. So there is no way to make it fully synchronous.

void ranJavaScript()

{

emit notifyRanJavaScript();

}

QString get()

{

QString result;

QEventLoop loop;

QObject::connect(this, SIGNAL(notifyRanJavaScript()), &loop, SLOT(quit()));

view->page()->runJavaScript("test();", [this](const QVariant &v)

{

result = v.toString();

this.ranJavaScript();

});

loop.exec();

return result;

}

# QWinTaskbarButton Class

<https://doc.qt.io/qt-5/qwintaskbarbutton.html#details>

<https://doc.qt.io/qt-5/qwintaskbarprogress.html>

# [rightclick event in Qt to open a context menu](https://stackoverflow.com/questions/24254006/rightclick-event-in-qt-to-open-a-context-menu)

customContextMenuRequested is emitted when the widget's contextMenuPolicy is Qt::CustomContextMenu, and the user has requested a context menu on the widget. So in the constructor of your widget you can call setContextMenuPolicy and connect customContextMenuRequested to a slot to make a custom context menu.

In the constructor of plotspace :

this->setContextMenuPolicy(Qt::CustomContextMenu);

connect(this, SIGNAL(customContextMenuRequested(const QPoint &)),

this, SLOT(ShowContextMenu(const QPoint &)));

ShowContextMenu slot should be a class member of plotspacelike :

void plotspace::ShowContextMenu(const QPoint &pos)

{

QMenu contextMenu(tr("Context menu"), this);

QAction action1("Remove Data Point", this);

connect(&action1, SIGNAL(triggered()), this, SLOT(removeDataPoint()));

contextMenu.addAction(&action1);

contextMenu.exec(mapToGlobal(pos));

}

## [**Right-click context menus with Qt**](http://www.setnode.com/blog/right-click-context-menus-with-qt/)

— richardwb on Thursday, July 23, 2009 @ 20:47

Getting right-clicks to popup a context menu is pretty straightforward in Qt. There are just a couple of things to watch out for...

First, there are several ways to tell Qt you want a context menu. One approach is to subclass the widget and override the QWidget::contextMenuEvent()event handler. However, I think the easiest approach is to call setContextMenuPolicy(Qt::CustomContextMenu) on the widget you want, and then connect the customContextMenuRequested() signal to the appropriate slot:

*// myWidget is any QWidget-derived class*

myWidget->setContextMenuPolicy(Qt::CustomContextMenu);

connect(myWidget, SIGNAL(customContextMenuRequested(const QPoint&)),

this, SLOT(ShowContextMenu(const QPoint&)));

Next, note that the const QPoint& pos parameter in the customContextMenuRequested() signal is [normally in widget coordinates](http://doc.trolltech.com/4.5/qwidget.html#customContextMenuRequested). However, also note that classes which inherit from QAbstractScrollArea[1](http://www.setnode.com/blog/right-click-context-menus-with-qt/#fn:viewportclasses) instead use the coordinates of their viewport(). Either way, you will need to map these coordinates to global coordinates using mapToGlobal().

Finally, simply either popup() or exec() your QMenu. Remember that popup() is non-blocking, so if you wish to use that, make sure your QMenu is created on the heap (or some other way of guaranteeing that the QMenu's lifetime outlasts the scope of the function)!

void MyClass::ShowContextMenu(const QPoint& pos) *// this is a slot*

{

*// for most widgets*

QPoint globalPos = myWidget->mapToGlobal(pos);

*// for QAbstractScrollArea and derived classes you would use:*

*// QPoint globalPos = myWidget->viewport()->mapToGlobal(pos);*

QMenu myMenu;

myMenu.addAction("Menu Item 1");

*// ...*

QAction\* selectedItem = myMenu.exec(globalPos);

if (selectedItem)

{

*// something was chosen, do stuff*

}

else

{

*// nothing was chosen*

}

}

# [How to get the mouse position on the screen in Qt?](https://stackoverflow.com/questions/19523375/how-to-get-the-mouse-position-on-the-screen-in-qt)

As the docs state: -

QCursor::pos()

Returns the position of the cursor (hot spot) of the primary screen in global screen coordinates.

If you have multiple screens, you can use: -

QPoint QCursor::pos(const QScreen \* screen)

Note that QScreen is for Qt 5