***General***

***String:***

QString a { "love" };

// Example to print the string into a QTextEdit control…

ui->textEdit->setPlainText(a);

***QString***:

Get a C like char\* String

QString tmp = “This is a test of QString”;

tmp.toStdString().c\_str(); 🡪 This gives a const char\*, and can be displayed with qInfo

Convert a number into a QString

int i = 42;

QString s = QString::number(i);

***Print to Console:***

#include<QDebug>

qInfo( "C Style Info Message" );

***Note***: Stdio’s printf and C++’s cout don’t appear to work in a Qt application

***Set Application Icon:***

1 - Create an .rc file (i.e myApp.rc) with the content:

IDI\_ICON1 ICON DISCARDABLE "myappico.ico"

2- Update your Qt project file (\*.pro) with the line:

RC\_FILE = myapp.rc

<https://doc.qt.io/archives/qt-4.8/appicon.html>

***Controls***

***Find a Control in a screen:***

The main class, either of type QWidget or QMainWindow has a ***findChild*** function

Example:

Class named ***Calculator***, as a PushButton named “***pushButtonTest***”

QString controlName ("pushButtonTest");

QPushButton \*button = Calculator::findChild<QPushButton \*>(controlName);

***QMenu and QAction***

Main menu are handled by QMenu, while all the clickable item in a menu, requires a QAction

***Note***: Qt Designer can create menus and action, but connecting Signal (menu) to Slot (mother class normally) is problematic (can’t see the slots in Qt Designer)

Solution: build the Menus programmatically, inside the Mother class.

***MainPanel.h***

class MainPanel : public QMainWindow

{

private slots:

void **newFile** (); 🡪 this is used to connect signal to a slot

(will execute our code )

private:

Ui::Calculator \*ui;

QMenu \*fileMenu; 🡪 will just contain the sub menu item

QAction \*newAct; 🡪 clickable item New, must be a QAction (not a QMenu)

***MainPanel.cpp***

MainPanel :: MainPanel (QWidget \*parent) : QMainWindow(*parent*)

, ui(new Ui::Calculator)

{

ui->setupUi(this);

newAct = new QAction(tr("&New"), this);

newAct->setShortcuts(QKeySequence::New);

newAct->setStatusTip(tr("Create a new file"));

connect(newAct, &QAction::triggered, this, &Calculator::newFile);

}

void MainPanel ::**newFile**()

{

qInfo( "newFile is triggered!" );

}

***Signals and Slots***

An event is a ***Signal***

A Event Handler is a ***Slot*** (a private method of a main class)

***Connect a Signal to a Control***

There are few ways to connect signals & slot:

1 - Use the Qt’s SIGNAL and SLOT macro

connect( Widget\_Emiter,

SIGNAL( Events\_Name \*\*1)),

This \*\*2,

SLOT( private\_Slot\_Handler\_Method\_name\*\*3) );

\*\*1 All Qt Widget comes with default events like  [pressed](https://doc.qt.io/qt-5/qabstractbutton.html#pressed)(),released() ,

Check the exact events of the widget you use for more events

Note: This is a method, so put the parentheses!

\*\*2 Here, the main class (QWidget of QMainClass) is the one who will receive and handle the signal

\*\*3 Make sure to add a private slot method in the main class

2 – Refer the direct reference to object’s method (with the Class:MethodName)

connect( producerTasks , &SignalProducerTask::completed ,

this, &MainWindow::threadCompletedSlot );

Here, producerTasks is a pointer to an SignalProducerTask object, which has a completed signal.

And this is pointer to a MainWindow, which has a threadCompletedSlot slot

Example: Calculator

***Calculator.h***

…

private slots:

void **clickHandler**();

…

***Calculator.cpp (***constructor)

Calculator::**Calculator**(QWidget \*parent): QMainWindow(*parent*)

, ui(new Ui::Calculator)

{

QString controlName ("pushButtonTest");

QPushButton \*button = Calculator::findChild<QPushButton \*>(controlName);

connect(button,SIGNAL(released()),this,SLOT(clickHandler()) );

}

The main class, either of type QWidget

***Auto Signal***

A class can send signal to itself, by using one of it slot method.

Here, an example of Clock class, that has an heartbeat using signal/slot

void ClockThread::***run***()

{

QTimer timer;

connect(&timer,

SIGNAL(timeout()),

this,

SLOT(timerHit()),

Qt::DirectConnection);

timer.setInterval(1000);

timer.start(); // puts one event in the threads event queue

exec();

timer.stop();

}

void ClockThread::**timerHit**()

{

//Do something here, like a heartbeat

}

***Thread and Timer***

Here is a useful Clock class, that can be used

***Note***: According to Qt documentation, we should use a QRunnable instead, and lunch it from a straight (not subclassed) QThread using the method ***movetothread***

***clockthread.h***

#include <QThread>

#include <QDateTime>

#include <QTimer>

class **ClockThread** : public QThread

{

public:

**ClockThread**();

Q\_OBJECT

signals:

void **sendTime**(QString time); 🡪This goes out to the mother

class, to interact with the mother

class using this Clockthread

private:

void ***run***();

int currentCount;

private slots:

void **timerHit**(); 🡪 This is used internally by the ClockThread

class, you do what you want with it like a

heartbeat for example

};

Clockthread.cpp

ClockThread::**ClockThread**() {}

void ClockThread::***run***()

{

QTimer timer;

connect(&timer, SIGNAL(timeout()), this,

SLOT(timerHit()),

Qt::DirectConnection);

timer.setInterval(1000);

currentCount = 0;

timer.start(); // puts one event in the threads event queue

exec();

timer.stop();

}

void ClockThread::**timerHit**()

{

QString newTime= QDateTime::currentDateTime().toString("ddd MMMM d yy, hh:mm:ss");

if(m\_lastTime != newTime ){

m\_lastTime = newTime;

emit sendTime(newTime) ;

}

if ( ++currentCount >= 5 ){ this->quit(); }

}