برمجة التطبيقات المحمولة
بوساطة Qt

المورد الشامل
Qt C++ framework
Qt Quick
QML

تأليف
المبرمج حسن المرهج

كل شيء حول Qt

يقدم هذا الكتاب كل ما تحتاجه لإتقان Qt C++ framework
حيث يبدأ بالتكلم حول حياة Qt C++ framework و أساسيات بناء
التطبيقات فيها ليصل إلى بناء تطبيقات احترافية موجهة لجميع منصات و نظم التشغيل منها
Win – Linux Embedded – Mac OS X – Symbian – Android- X11 ..
أيضا يتكلم الكتاب عن QML و كيفية تصميم واجهة مستخدم احترافية بوساطتها
و عن تقنية Qt Quick التي تمثل دمج QML و Qt C ++

سيتناول هذا الكتاب ما يلي :

- ✓ حياة Qt C++ framework
- ✓ أساسيات بناء تطبيق
- ✓ استخدام و تطوير كائنات واجهة مستخدم رسومية
- ✓ صفوف متقدمة لبناء تطبيقات إحترافية
- ✓ الرسوميات و معالجتها
- ✓ معالجة الملفات و مستندات XML
- ✓ المسالك المنخفضة المستوى و المرتفعة المستوى
- ✓ برمجة التطبيقات الشبكية و NFC
- ✓ أساسيات OpenGL
- ✓ برمجة تطبيقات قابلة للحمل
- ✓ استخدام برنامج المحاكي لاختبار التطبيق
- ✓ كيفية الحصول على معلومات الجهاز الحامل للتطبيق من مستوى شحن البطارية و حالة SIM الخ
- ✓ استخدام الوسائط المتعددة
- ✓ أساسيات لغة برمجة QML
- ✓ عناصر واجهة المستخدم في QML
- ✓ عناصر الحركة في QML
- ✓ أساسيات تقنية Qt Quick
- ✓ إنشاء مكون QML بوساطة Qt C++
- ✓ Qt Quick في التطبيقات المحمولة
- ✓ تقنية GPS في Qt

الإهداء إلى والِدَيّ حفظهما الله و إخوتي و أصدقائي و إلى كل طالب علم

اللهم صلاتك على خاتم الأنبياء و سيد المرسلين سيدنا محمد الحمد و على آله و صحبه و سلم.

أما بعد توكلت على الله و بدأت أنا العبد الفقير لله

"حسن نور الدين المرهج" بتأليف هذا الكتاب المسمى برمجة التطبيقات المحمولة بوساطة Qt ليكون بإذن الله مفيد لجميع المبرمجين العرب , حيث عانيت من الكثير من المتاعب أثناء تأليفه و لم أتوانى عن تكملة تأليفي له من أجل أن تعم الفائدة فأرجو منكم الدعاء لي و لوالدي الذي وقف جانبي في مسيرتي فله جزيل الأجر و الثواب حفظه الله.

قسم

Qt C++ framework

❖مقدمة Intro

تتنافس شركات البرمجيات في عالم تحكمه الثورة الرقمية ويوصف بعالم الحوسبة المحمولة والذي يعتمد على البرمجيات التي تعمل على المخدم مثل تقنيات Cloud Computing و الأجهزة الذكية Smart Device .

بما أن عالمنا العربي يتأخر في استقطاب و تسويق هذه التقنيات العظيمة الفائدة قررنا تأليف هذا الكتاب الذي يشرح أعظم التقنيات و أكثرها محمولية و انتشارا والتي تدعى C++ Qt Framework .

تم بناء مكتبات Qt في عام 1992 , لغة برمجة Qt (كيوت لفظا) من شركة ترولتيك TrollTech.

إن كثير من البرمجيات العالمية الحالية والموجودة بين أيدينا نستخدمها مكتوبة بوساطة Qt منها :

Google earth , SKYP , Autodesk Maya more …..

و الكثير من البرمجيات التي تمثل الشريان الأبهر للأجهزة الطيبة في المشافي و الكثير من برمجيات الموجودة داخل المركبات الحديثة الخ ...

إن Qt C++ Framework التي سوف تتعرف عليها في فصول هذا الكتاب هي تطوير لمكتبات C++ و هيّأت لتعمل على أي نظام توضع عليه .

بهذا تكون قد أطاحت بشهرة MFC و الكثير من أطر العمل Microsoft و غيرها , حيث أننا اليوم بمساعدة Qt نستطيع كتابة برنامج احترافي ليعمل على الكثير من الأنظمة منها , Win Win CE , Mac OS , Linux Embedded , Linux X11 , Series 40 , Series 60 , S^1 , S^2 , S^3 - Android - iPhone more more ….

نقصد بـ S^ أي نظام Symbian .

قبل أن نعرف Qt كنا نفكر مليّا متسائلين أي لغة سوف نستخدم من أجل كتابة تطبيقنا و هل سوف يعمل هذا التطبيق على نظام واحد فقط كـ Win أم أنه قادر على أن يعمل على أنظمة شركات أخرى ؟

الآن تستطيع كتابة برنامجك مرّة واحدة ليعمل على كافة المنصات .

سوف نستخدم Qt Creator المنتج من شركة نوكيا Nokia في عام 2008 .

شركة Nokia أضافت و طورت الكثير من مكتبات Qt مع إضافة تقنيات جديدة مثل QML لوصف واجهة المستخدم و تقنية Qt Quick والتي هي عبارة عن دمج بين لغة توصيف واجهة المستخدم من البرنامج و إطار عمل Qt C++ , سوف نأتي بالتكلم عنهم بالتفصيل في موضعهم من هذا الكتاب إن شاء الله .

الفصل الأول

الأساسيات Basics

❖ مقدمة Intro :

نبدأ الفصل الأول لإطار عمل Qt بفهم بنية مشروع Qt و كيفية استخدام (الصف) النافذة QWidget و الأدوات المهمة كأداة QLabel و QPushButton و تجميع المشروع باستخدام سطر الأوامر لـ أداة(MinGW) المرفق مع حزمة Qt.

نبدأ المثال الأول ببناء نافذة (QWidget),يوجد فيها (أداة) صندوق تنسيق (تخطيط) الأدوات العمودي (QVBoxLayout) تحتوي على زر (QPushButton) و لوحة لعرض النص (QLabel) تحتوي النص (Hello World!) عند الضغط على الزر (QPushButton) يغلق التطبيق , ثم تجميع المشروع و تنفيذه ولكن دون استخدام محرر Qt Creator لأنّه من الأفضل لتعلم لغة برمجة فهم الرمّازات و الأوامر لبناء و تنفيذ التطبيق التي تضاف من قبل المحرر و المعالجات الذكية حيث تتم تلك الإضافات من وراء الكواليس وتظهر أمامنا في شاشة المحرر جاهزة.

ملاحظة : MingGW مشروع أدوات GNU لـ منصة Win .

الآن دعنا نبدأ بكتابة الرمّاز الخاص بملف المشروع Write source code for file project:

أولاّ ننشأ مجلد باسم المشروع و ليكن Ch1 في قرص C مثلاّ.

ثانياّ ننشأ ملف فارغ حيث يكون امتداده (pro.) (هذا هو الملف الأساسي لبناء مشروع Qt) حيث يحتوي على أسماء ملفات المشروع كافة ويتضمن الصفوف و الملفات المصدرية و واجهات المستخدم حيث تستطيع أن تعتبر ملف المشروع هذا الذي يحمل اللاحقة (pro.) بمثابة حلّ أو حاوي يحوي قائمة الملفات المستخدمة في المشروع ليقوم المفسر بقراءة هذه الملفات كل حسب نوعه أما الأن دعنا ننتقل من الكلام النظري الى أول برنامج لك في Qt.

ننشأ ملف يكون اسمه main بامتداد cpp (main.cpp) حيث سنضع به الرمَاز المصدري الخاص بالتطبيق.

نفتح ملف المشروع (Ch1.pro) بواسطة المفكرة (أو أي محرر نصوص آخر) و نكتب الرمَاز التالي:

SOURCES += main.cpp

حيث أضفنا الملف main.cpp لقائمة ملفات المشروع مثلما ذكرنا مسبقا

❖ فهم نواة التطبيق و إنشاء نافذة (QWidget) Knowing core application and create window (QWidget):

الآن لنعود إلى الرمَاز المصدري الخاص بملف main.cpp بعد ما عرفناه داخل ملف المشروع.

نكتب الرمّاز التالي داخله (هذا الرمّاز يظهر نافذة [QWidget] بعنوان !Hello World):

```
#include <QApplication>
#include <QWidget>
int main(int argc,char* argv[])
{
    QApplication app(argc,argv);
    QWidget* widget= new QWidget();
    widget->setWindowTitle("Hello World!");
    widget->show();

    return app.exec();
 }
```

الآن نبدأ بشرح هذا الرمّاز:

#include <QApplication>

هذا الرمّاز لتضمين ملف أو مكتبة (صف) QApplication ضمن المستند الحالي حيث تستطيع استخدام المناهج و الخصائص و الماكروات و الثوابت العامة التي يحتويها.
تدعى (#include) توجيه ما قبل الترجمة لتضمين (استخدام) ملف ما.
هذا الصف يدير سياق تحكم التطبيق بواجهة المستخدم الرسومية و الإعدادات الرئيسية لتطبيقنا الحالي, وسوف نتكلم عنه لاحقا بتفصيل أكثر في فصول أخرى من الكتاب.

#include <QWidget>

هذا الصف هو الصف الأساسي لجميع كائنات واجهة المستخدم مثل أداة مربع النص (QLineEdit) أو الزر (QPushButton), كل أداة مثل QPushButton أو

QLineEdit ترث من الصف QWidget يرث QWidget الصف QObject الصف الأب لكل الكائنات ,ويرث أيضا الصف الخاص بكائن الرسم QPaintDevice .
و يقوم QWidget باستقبال الأحداث كأحداث الفأرة و لوحة المفاتيح, ويمكن أن يكون أب أي نافذة قائمة بذاتها (Window) أو أن يكون ابن لنافذة ما, وسنرى فيما بعد كيفية استخدامه.

```
int main(int argc, char* argv[]){}
```

الكتلة أو المنهجية الأساسية (أي أول منهجية ينفذها التطبيق عند فتحه) لأي تطبيق مكتوب بلغة c++ وله وسيطين الوسيط الأول من نمط عدد صحيح (int argc) خاص بعدد المحارف الممررة عند فتح التطبيق, أما الوسيط الثاني فهو مؤشر مصفوفة من نمط محرف (char* argv[]) خاص بالنص المدخل عند فتح التطبيق, و المفترض أنك تعلم هذه المعلومات , الآن دعنا نعود للرمّاز المصدري الخاص بالكتلة main.

```
QApplication app(argc, argv);
```

أنشأنا هنا مثيل يدعى app لصف QApplication مع تمرير الوسيطين الخاصين بحفظ النص المدخل عند فتح التطبيق لمعالجتهما, المثيل app يمثل التطبيق ككل (نواة التطبيق الحالي) و سنتكلم عنه بالتفصيل لاحقاً.

```
QWidget* widget = new QWidget();
```

أنشأنا حدث من صف QWidget بالإعتماد على البناء الافتراضي هذا الحدث سيكون النافذة الأب للتطبيق,
وسطاء البناء الافتراضي f,parent :

```
QWidget* parent = 0
```

عندما تكون قيمة الوسيط parent مساوية للقيمة صفر يكون الحدث QWidget() أب.

```
Qt::WindowFlags f =0
```

أي الإعدادات الافتراضية للنافذة وسنتكلم عنه لاحقا.

```
widget->setWindowTitle("Hello World!");
```

استدعينا الإجرائية setWindowTitle التابعة للحدث widget والتي تعدل عنوان النافذة وممرنا لها الوسيط النصي Hello World! ونكون بذلك عدلنا عنوان النافذة .

```
widget->show();
```

أظهرنا كائن النافذة.

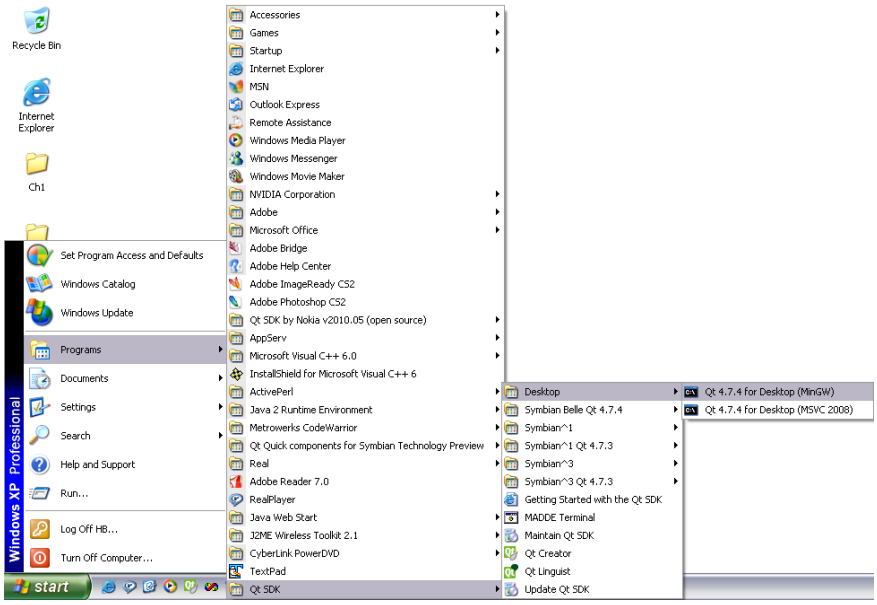
```
return app.exec();
```

حيث تستدعي منهجية exec() من المثيل app (والتي هي منهجية ساكنة [static]) للدخول بحلقة الأحداث الرئيسية (لاستقبال الأحداث كافة كأحداث الفأرة و لوحة المفاتيح...) و لا يتم الخروج من هذه الحلقة حتى استدعاء المنهجية exit() .

ملاحظة هامة : يجب عليك أن تفرق بين مصطلح "حَدَثْ" بفتح الحاء و الدال و الثاء ساكنة و مصطلح "حِدْثْ" بكسر الحاء ثمّ الدال و الثاء ساكنين .

❖ تجميع التطبيق و تنفيذه Compile application and execute it :

نفتح برنامج MinGW الخاص باللغة Qt .
من ابدأ برامج ثم (QtSDK) ثم (Desktop) ثم
Qt 4.7.4 for Desktop (MinGW) كما في الصورة الظاهرة في الشكل (1.1):



الشكل 1.1

ننتقل إلى داخل مجلد المشروع Ch1 الذي يوجد في قرص (C) بواسطة البرنامج (MinGW) بوساطة موجه الأوامر وهذا أمر الانتقال cd c:\Ch1 .

ثم نكتب الأوامر التالية:

```
qmake –o makefile Ch1.pro
```

ثم نضغط على مفتاح Enter

mingw32-make –f makefile.Debug

ثم نضغط على مفتاح Enter

أداة qmake: تساعدنا في تجميع التطبيق لها عدة وظائف شرحها خارج نطاق هذا الكتاب .

من أجل البدء بعملية التجميع نستخدم الوسطاء التالية :

الوسيط output(-o) : لإنشاء الملفات الخاصة بمعلومات التجميع .

(makefile) اسم الملفات التي سينتجها- ينشئها- وهو اسم خياري .

فينتج لدينا الملفات التالية:

makefile

يحتوي على المعلومات الخاصة بإنشاء الملفين makefile.Release و makefile.Debug

makefile.Release

يحتوي على المعلومات الخاصة بالمشروع و الأوامر الخاصة بتجميع المشروع بحيث يخبر المجمع أننا نريد ملف تنفيذي خاص بالنشر.

makefile.Debug

يحتوي على المعلومات الخاصة بالمشروع و الأوامر الخاصة بتجميع المشروع بحيث يخبر المجمع أننا نريد ملف تنفيذي خاص بالتنقيح.

و مجلدين release و debug :

release

ليحتوي على النسخة النهائية من الملف التنفيذي للمشروع الخاص بالنشر.

debug

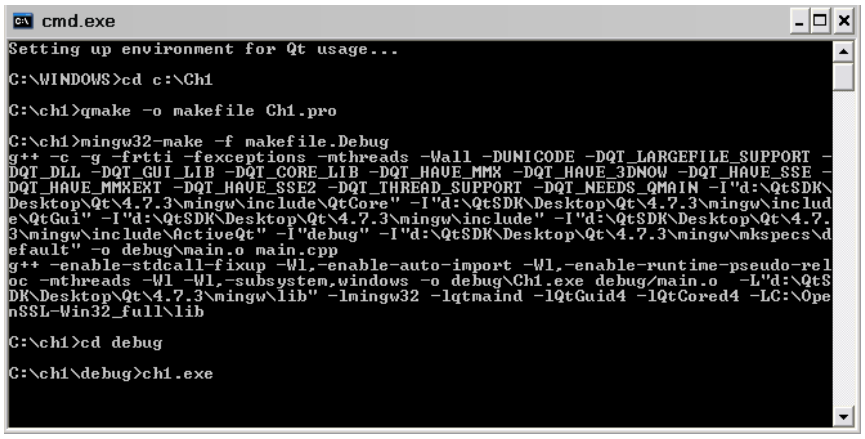
ليحتوي على النسخة النهائية من الملف التنفيذي للمشروع الخاص بالتنقيح.

mingw32-make –f makefile.Debug

هذا الأداة لتجميع المشروع وإنتاج ملف تنفيذي, أما الوسيط (f-) فهو يعني ملف (file) لقراءة الملف الذي يكون مكتوب بعد الوسيط (f-) أي قراءة الملف (makefile.Debug) حيث ينشأ الملف التنفيذي داخل المجلد (debug) باسم المشروع Ch1 و بامتداد exe (Ch1.exe).

باستخدام سطر الأوامر التالي كما في الشكل (1.2) :

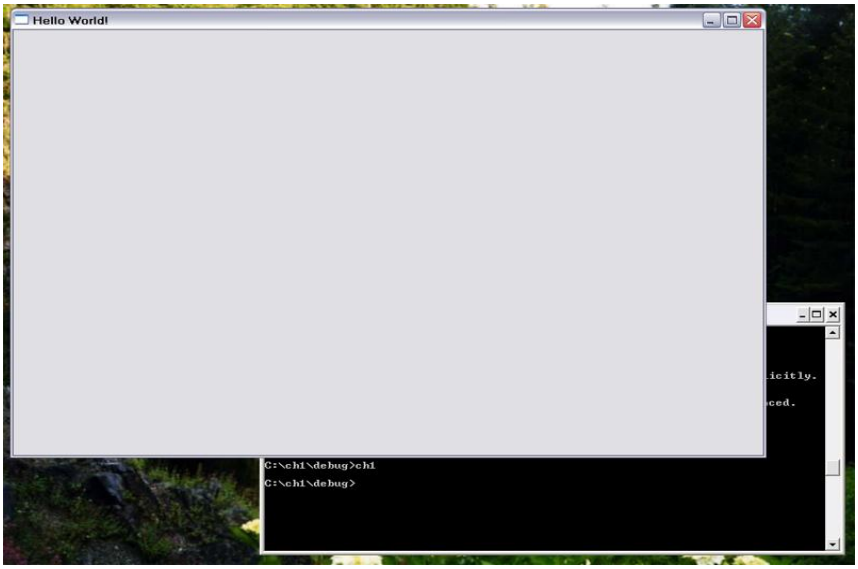
```
cd C:\Ch1
```



الشكل 1.2

لتشغيل الملف التنفيذي أدخل من خلال الأداة MinGW إلى داخل مجلد debug الموجود داخل مجلد Ch1 في قرص (C) واكتب اسم الملف (Ch1.exe) .

يظهر الشكل (1.3) صورة للتطبيق .



الشكل 1.3

❖ بناء تطبيق سطر أوامر :

لبناء تطبيق سطر اوامر يجب أن يكون صف التطبيق الرئيسي "QCoreApplication" بدلا من الصف "QApplication" , حيث الصف "QApplcation" لا يدعم التحكم بشاشة سطر الأوامر , أنشأ مشروع فارغ و سمّه "console" , ثمّ أضف ملف مصدري يدعى "main.cpp" , ضمن ملف المشروع "console.pro" أضف التالي :

```
QT      += core

CONFIG   += console

SOURCES += main.cpp
```

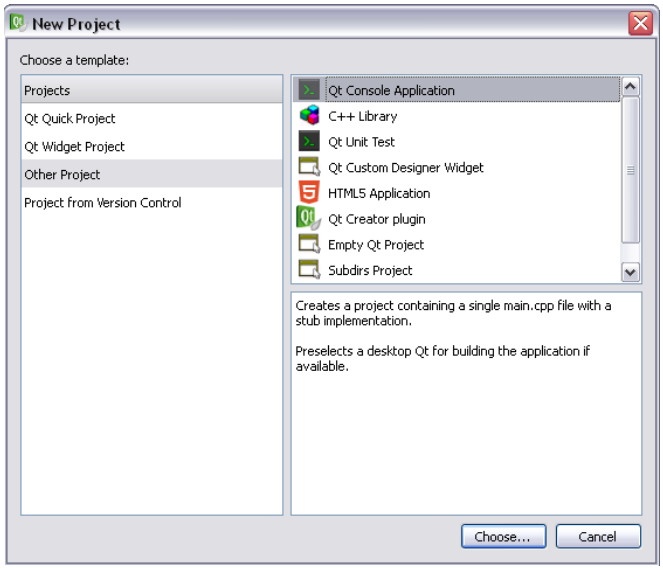
حيث أتحنا استخدام مكاتب الحزمة "core" بأضافة السطر البرمجي "QT += core" , ثمّ اتحنا استخدام جميع إضافات تطبيق سطر الأوامر من خلال إضافة السطر البرمجي "CONFIG += console" , أخيرا أضفنا الملف المصدري "main.cpp" للمشروع , فعند تجميع المشروع سوف يتم قراءة جميع السطور البرمجيّة المضافة لملف المشروع , الآن أهب إلى الملف المصدري و اكتب داخله الرمّاز التالي :

```
#include <QtCore/QCoreApplication>

#include <iostream>
```

```
int main(int argc,char** argv){
   QCoreApplication app(argc,argv);
   std::cout << "Ray Pub" << std::endl ;
   return app.exec();
}
```

ضمّنا الصف "QCoreApplication" و المكتبة "iostream" القياسيّة الخاصة بالتحكم بمجاري الدخل و الخرج , أنشئنا مثيل للصف "QCoreApplication" ليكون هو نواة التطبيق الرئيسية لتنفيذه , ثمّ طبعنا على شاشة سطر الأوامر الجملة التالية "Ray Pub" , أطلقنا تنفيذ التطبيق بوساطة استدعاء المنهج "app.exec()" , جمع التطبيق و اختبر النتيجة .

نستطيع إنشاء مشروع سطر أوامر داخل "Qt Creator" من خلال إختيار مشروع جديد "New Project" ثمّ اختيار القالب "other Project" و تحديد المشروع " Qt Console Application" , انظر الشكل (1.4) :



الشكل 1.4

لننتقل إلى وضع الزر الخاص بالخروج من البرنامج ووضع اللافتة(لوحة النص) التي تحتوي على كلمتي hello world.

❖ **أداة صندوق التنسيق (التخطيط) -العمودية- QVBoxLayout (QVBoxLayout) tool :**

تفيد هذه الأداة لتنسيق الأدوات بشكل عمودي و رصف محتواها

بوضع أداة تحت الأداة الأخرى حيث يتغير البعد بين الأدوات وحجم الأدوات بشكل تلقائي بالتناسب مع حجم النافذة والأداة QVBoxLayout.

وسنرى الفائدة من استخدام هذه الأداة في تحسين مظهر توضع الأدوات على النافذة.

الآن سنضيف الرمّاز المصدري الخاص بالأداة (QVBoxLayout) حيث سننشئها داخل النافذة (الكائن) widget

الآن أضف الرمّاز ذو الخلفية ذات اللون الغامق إلى الملف main.cpp :

```
#include <QApplication>
#include <QWidget>
#include <QVBoxLayout>
int main(int argc,char* argv[])
{
    QApplication app(argc,argv);

    QWidget* widget= new QWidget();
    QVBoxLayout* Vlayout = new QVBoxLayout();

    widget->setWindowTitle("Hello World!");
    widget->setLayout(Vlayout);
    widget->show();

    return app.exec();
}
```

نبدأ بشرح الرمّاز المصدري المضاف.

#include <QVBoxLayout>

ضمّنا الصف الخاص بالأداة (QVBoxLayout)

QVBoxLayout* Vlayout = new QVBoxLayout();

أنشئنا حدث من الأداة (QVBoxLayout) لاستخدامه داخل النافذة.

Widget->setLayout(Vlayout);

وضعنا الأداة (Vlayout) داخل النافذة (الكائن) widget.

❖ إضافة أداتي لوحة النص و الزر (QLabel) و (QPushButton)

: Add QLabel and QPushButton tools

أداة (QLabel) تستخدم لإظهار نص ما داخلها أما أداة (QPushButton) فهو أداة زر يستقبل النقر من الفأرة ليقوم بتنفيذ كود قمنا بوضعه مسبقا.

الآن دعنا نضيف الرمّاز الخاص بهاتين الأداتين داخل الملف main.cpp

أضف الرمّاز ذو الخلفية ذات اللون الغامق:

```
#include <QApplication>
#include <QWidget>
#include <QVBoxLayout>
#include <QLabel>
#include <QPushButton>
int main(int argc,char* argv[])
{
    QApplication app(argc,argv);

    QWidget* widget= new QWidget();
    QVBoxLayout* Vlayout = new QVBoxLayout();
    QLabel* lbl= new QLabel();
    QPushButton* btn= new QPushButton();

    widget->setWindowTitle("Hello World!");
     lbl->setText("Hello World!");
   btn->setText("Click me to E&xit");

    Vlayout->addWidget(lbl);
    Vlayout->addWidget(btn);

    widget->setLayout(Vlayout);
    widget->show();

    return app.exec();
}
```

الآن دعنا نبدأ بشرح الرمّاز المضاف:

#include <QLabel>

#include <QPushButton>

تضمين الصفين (QLabel) صف أداة لوحة النص و (QPushButton) صف أداة الزر.

QLabel* lbl= new QLabel();

أنشأنا حدث من أداة (QLabel) يدعى (lbl).

QPushButton* btn= new QPushButton();

أنشأنا حدث من أداة (QPushButton) يدعى (btn).

lbl->setText("Hello World !");

عرض النص (Hello World!) داخل أداة لوحة النص.

btn->setText("Click me to E&xit");

لتصبح عنوان الزر "Click me to E&xit".

Vlayout->addWidget(lbl);

Vlayout->addWidget(btn);

إضافة الأداتين lbl و btn إلى أداة صندوق حاوية التخطيط العمودية.

الآن جمّع المشروع و نفذه سوف يظهر كما في الشكل (1.4):

الشكل 1.4

❖ الرمّاز الخاص باتصال حدث:

الآن دعنا نضيق الرّمّاز الخاص بحدث النقر على الزر للخروج من التطبيق:

أضف الرمّاز ذو الخلفية ذات اللون الغامق:

```
#include <QApplication>
#include <QWidget>
#include <QVBoxLayout>
```

```
#include <QLabel>
#include <QPushButton>
int main(int argc,char* argv[])
{
    QApplication app(argc,argv);

    QWidget* widget= new QWidget();
    QVBoxLayout* Vlayout = new QVBoxLayout();
    QLabel* lbl = new QLabel();
    QPushButton* btn= new QPushButton();

    widget->setWindowTitle("Hello World!");
    lbl->setText("Hello World!");
    btn->setText("Click me to E&xit");


QObject::connect(btn,SIGNAL(clicked()),widget,SLO
T(close()));

    Vlayout->addWidget(lbl);
    Vlayout->addWidget(btn);

    widget->setLayout(Vlayout);

    widget->show();

    return app.exec();
}
```

نبدأ شرح السطر المضاف:

QObject هو الصف الأساسي لجميع الكائنات مثلما ذكرنا مسبقا.

connect منهجية ساكنة من الصف (المكتبة) QObject لإنشاء اتصال بين إشارة الكائن المرسل و منهجية الكائن المستقبل, أيّ عند قدح حدث ما (إشارة) من الكائن المرسل عندها تنفذ (تستدعى) منهجية (مقبس) الكائن المستقبل.

أمّا وسطائها:

Sender : الكائن المرسل الذي تقدح منه الإشارة (الحدث).

SIGNAL : الإشارة (نوع الحدث الداخلي) الذي وقع على الكائن المرسل, مثل حدث النقر.

Receiver : الكائن المستقبل الذي تقع عليه تنفيذ المنهجية المتصلة بإشارة (بحدث) الكائن المرسل.

SLOT : المقبس (المنهجية) الذي يستدعى عند قدح الإشارة (وقوع الحدث).

هنا في هذا السطر المضاف الكائن المرسل (أيّ الذي تقدح منه الإشارة -الحدث-) هو كائن الزر btn

أما نوع الإشارة (نوع الحدث) هو النقر (()clicked) أضفنا الماكرو (SIGNAL) قبل نوع الحدث

لنخبر مجمع Qt أن الوسيط الذي مرر له هو نوع الإشارة (الحدث الداخلي), أما widget هو الكائن المستقبل

لردة فعل قدح إشارة (الحدث الداخلي) النقر, أما (()close) استدعينا هنا المنهجية ()close لإغلاق التطبيق

أضفنا قبلها الماكرو (SLOT) لنخبر مجمع Qt أن الوسيط الذي داخله هي المنهجية التي سوف تستعدا عند قدح الإشارة (الحدث).

إذا نستنتج أن الأحداث ضمن Qt تعتمد على مبدأ SIGNAL (الإشارة) و SLOT (المقبس) , عند قدح الإشارة (الحدث الداخلي)يتم استدعاء المقبس (المنهجية) المتصل بها.

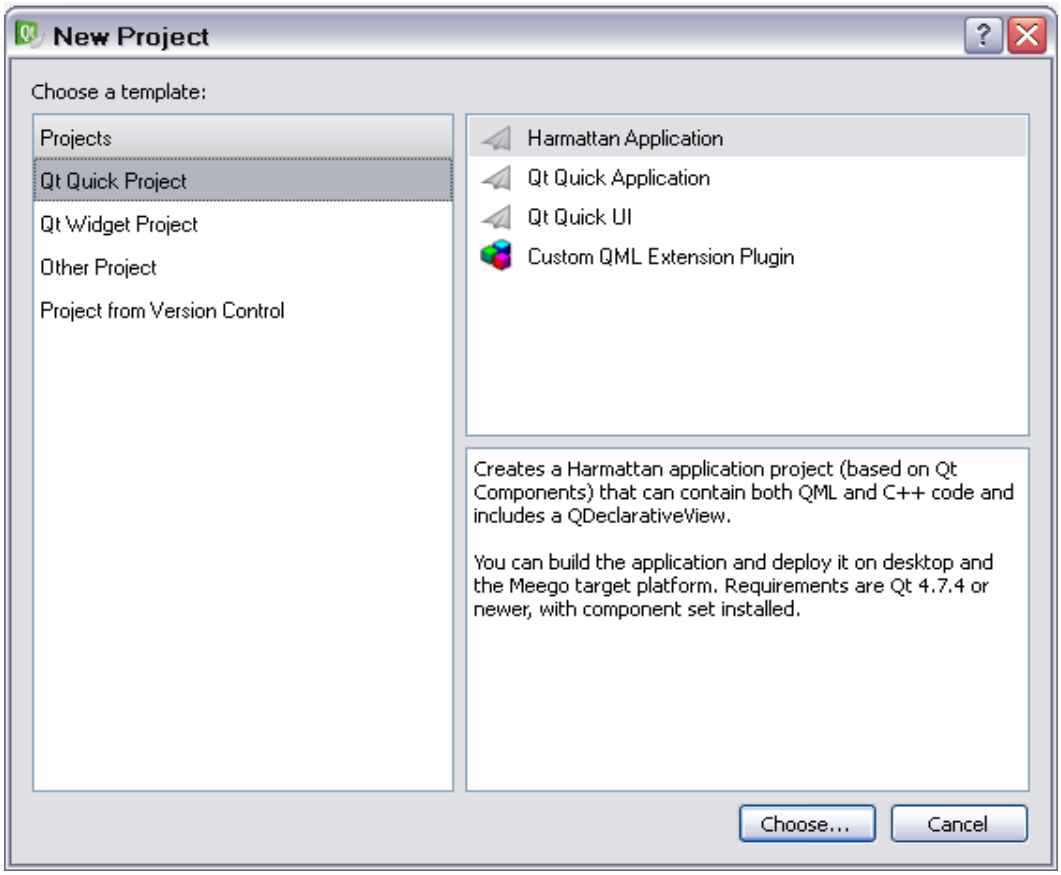
جمع التطبيق ونفذه عند النقر على زر btn سوف يغلق التطبيق.

الآن انهينا هذا المثال .

❖ استخدام Qt Creator:

لمحة عامة عن أنواع المشاريع التي يقدمها :

كما يوضح الشكل (1.5) نأخذ من القائمة File الخيار New File or Project :



الشكل 1.5

نبدأ أولا بالتبويب الأول من قائمة (Projects) وهو (Qt Quick Project).

يحوي على أربع قوالب أو أنواع (المشاريع):

Harmattan Application

هذا الخيار لإنشاء مشروع Qt , QML معا تستدعي ملفات QML عن طريق رمّاز Qt (C++) وهو خاص ببناء تطبيقات لأنظمة سطح المكتب مثل نظام (Win, Mac OS) و لمنصة (MeeGo) وهي عبارة عن نظام Linux مفتوح المصدر للهواتف الذكية.

Qt Quick Application

نفس الخيار السابق لكنه موجه إلى : أنظمة سطح المكتب و أنظمة الجوال كنظام (Symbian).

Qt Quick UI

لإنشاء مشروع QML فقط (لا يحتاج إلى بناء -تجميع-) يعرض على أنظمة سطح المكتب بوساطة برنامج (QML Viewer).

Custom QML Extension Plugin

لإنشاء عنصر (أداة - مكتبة) خاصة بمشاريع (Qt Quick).

التبويب الثاني من قائمة (Projects) وهو (Qt Widget Project).

Qt GUI Application

لإنشاء تطبيق Qt خاص بأنظمة سطح المكتب مثل (Win , Mac OS).

Mobile Qt Application

لإنشاء تطبيق Qt خاص بأجهزة الجوال.

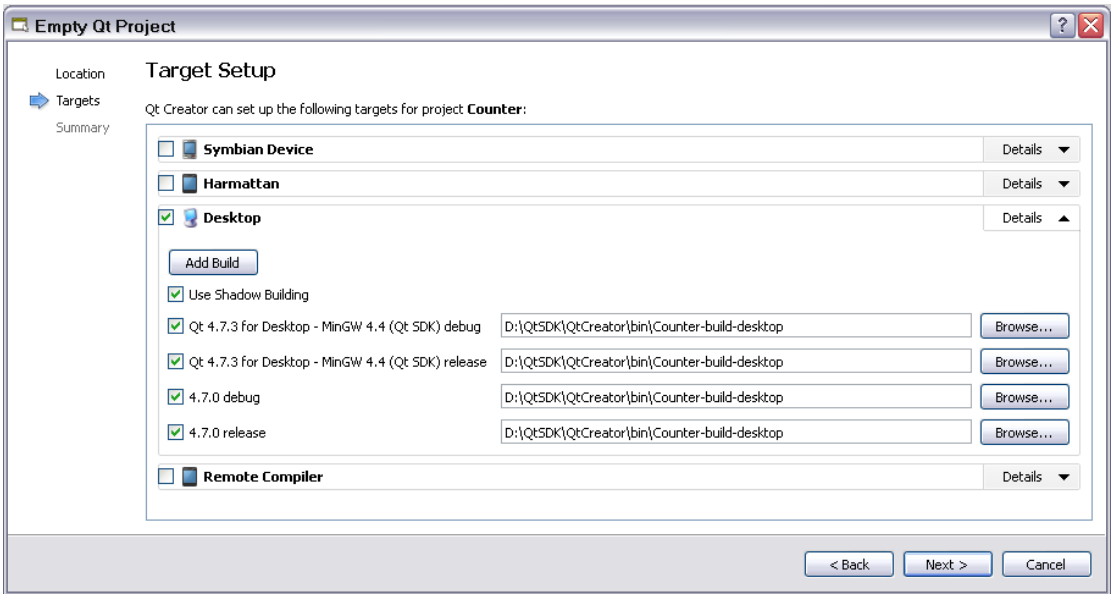
هذه لمحة عامة عن أهم أنواع المشاريع الذي يملكها (Qt Creator من شركة Nokia).

- **بناء تطبيق العداد بواسطة** Qt Creator.

سنبني تطبيق Qt موجه إلى أنظمة سطح المكتب وليكن التطبيق عبارة عن نافذة تحمل العنوان Counter و رمزها شعار Qt ,يوجد فيها زر يحمل العنوان (Add) و لوحة نص تحمل الرقم صفر, عند كل نقرة على الزر تزداد القيمة ضمن لوحة النص بمقدار 1 حتى تصل الرقم 10 فتظهر رسالة يوجد فيها النص التالي (!sorry you can't add 1 to number).

إنشاء التطبيق:

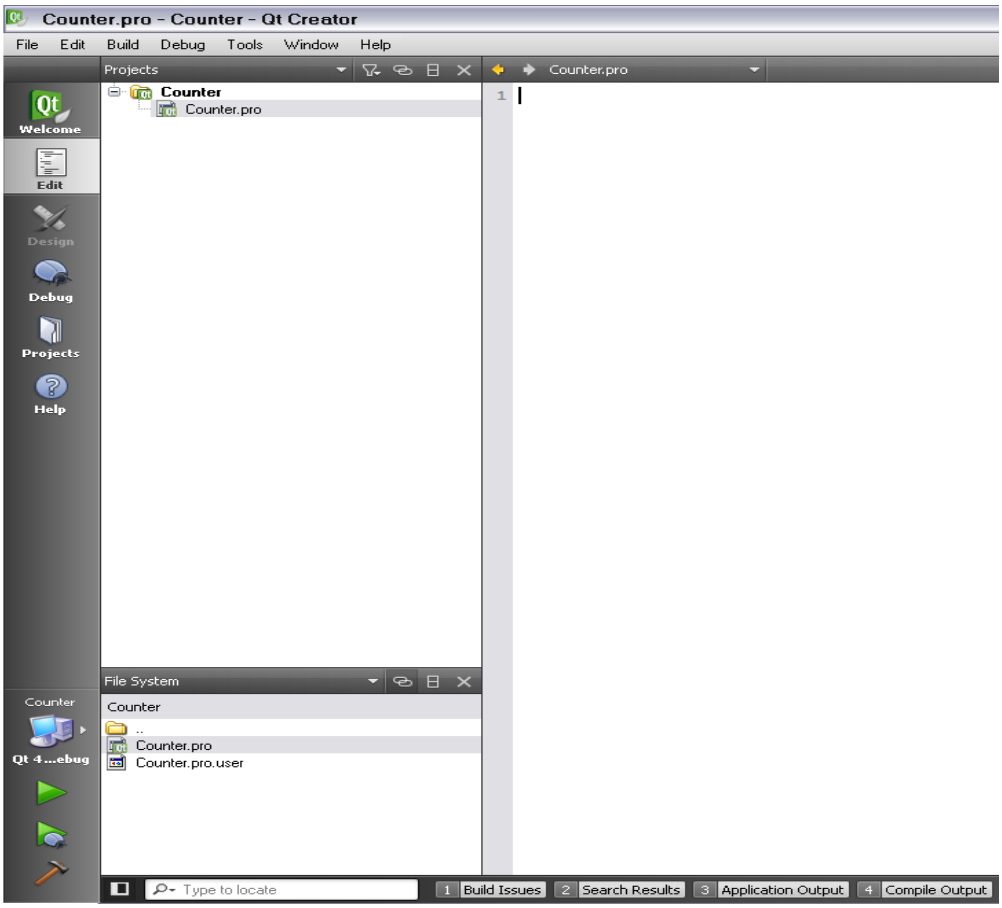
نفتح تطبيق Qt Creator نأخذ من القائمة File الخيار New File or Project ثمّ نأخذ من تبويب َOther Project نختار Empty Qt Project نضع ضمن خانة Name اسم المشروع و ليكن Counter و لندع مسار المشروع المنشأ هو المسار الافتراضي نضغط على زر Next ولنحدد فقط الخيار (Desktop) كما في الشكل (1.6):



الشكل 1.6

هذا التطبيق موجه فقط لأنظمة سطح المكتب ولندع مسارات التجميع المسارات الافتراضية نضغط على زر Next ومن ثمّ زر Finish.

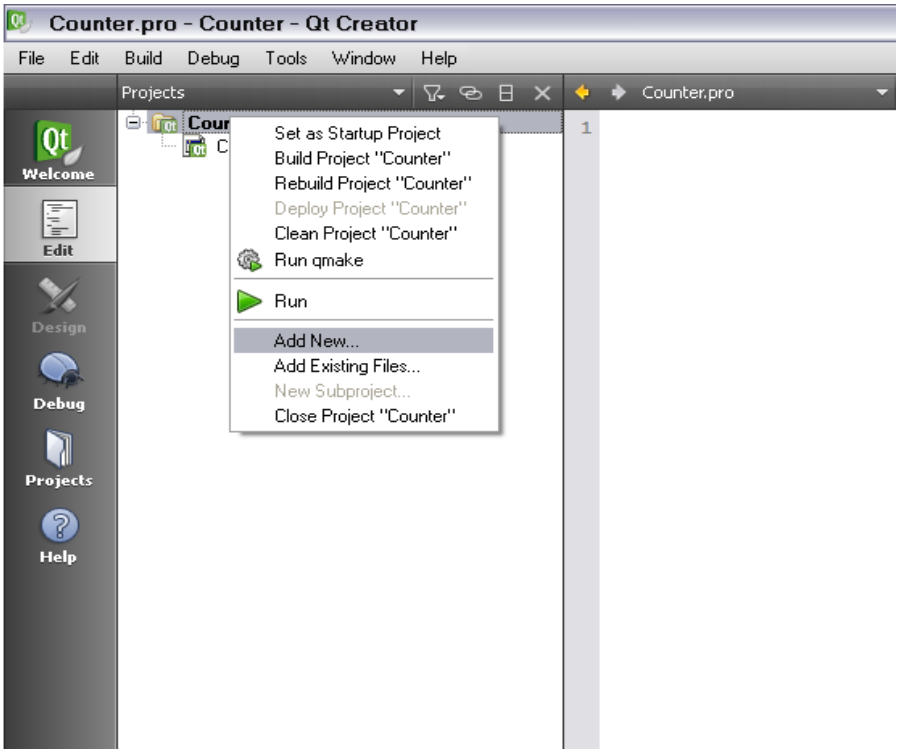
أولا نفتح ملف المشروع Counter.pro من تبويب Projects من شجرة المشروع Counter ننقر نقرة مضاعفة بالفأرة على ملف Counter.pro كما في الشكل (1.7):



الشكل 1.7

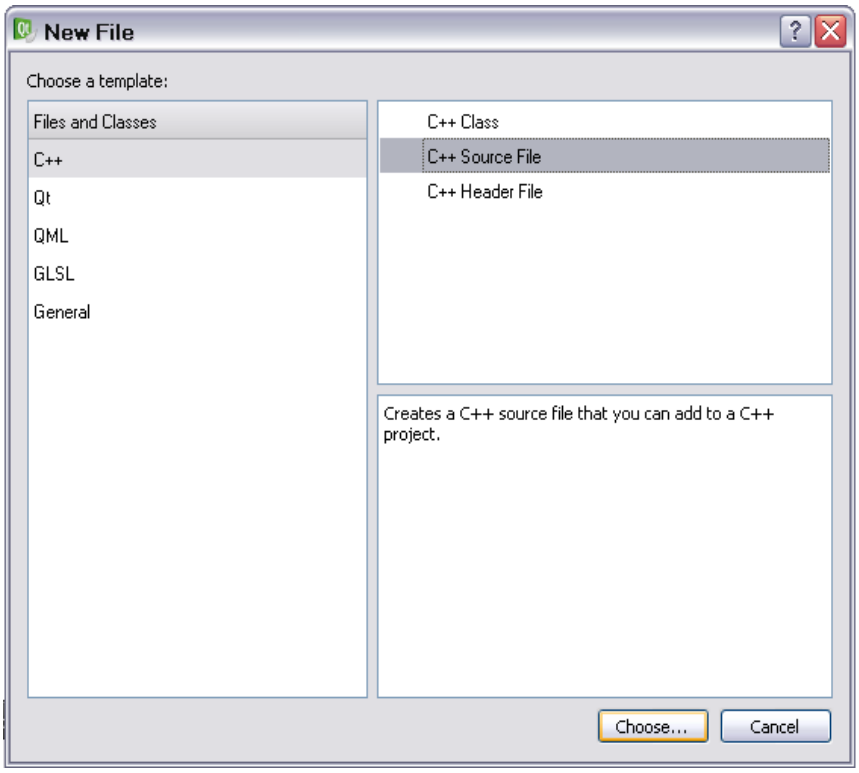
نضيف الملف main.cpp كالتالي:

نضغط بالزر الأيمن للفأرة على المشروع Counter و نختار Add New كما في الشكل (1.8):



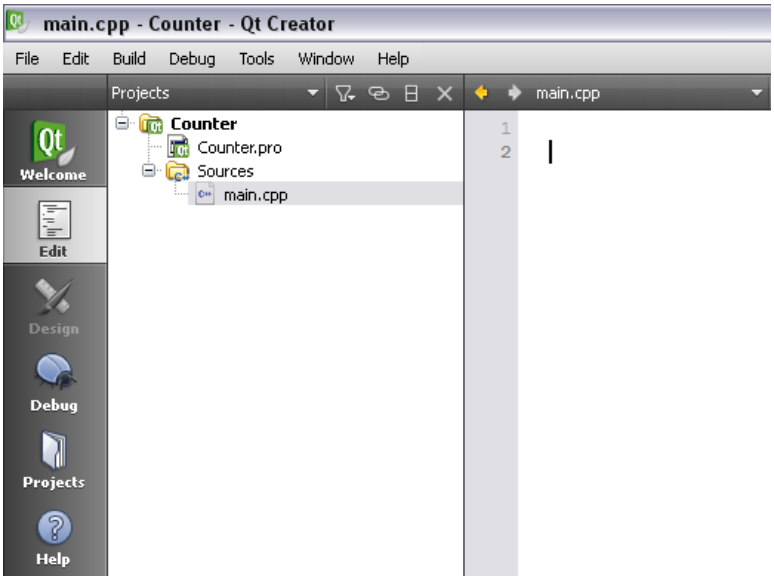
الشكل 1.8

من تبويب C++ نأخذ C++ Source File كما في الشكل (1.9):



الشكل 1.9

ونضغط على زر Choose ثمّ نضع في خانة Name اسم الملف main.cpp ثمّ نضغط Next ثمّ نحدد من خانة Add to project المشروع الذي نريد إضافة الملف عليه ثمّ Finish يظهر لنا الملف كابن للمجلد Sources كما في الشكل (1.10):



الشكل 1.10

هذا الملف سيحوي داخله الرمّاز الخاص بتحميل إعدادات التطبيق الرئيسية و إظهار النافذة الخاصة بالعدّاد سنضع الرمّاز داخل الكتلة main.

الآن دعنا نضيف الصف الخاص بنافذة العدّاد.

نضغط بزر الفأرة الأيمن على المشروع Counter ثم نأخذ Add New ثمّ من تبويب C++ نأخذ C++ Class

نضع الآن ضمن خانة اسم الصف (Class name) Counter_win

أمّا خانة Base class هنا سنحدد اسم الصف (الصف الأساسي) الذي سيرث منه الصف المنشأ أي صفنا الحالي (Counter_win) وهنا سنضع القيمة QWidget لأنّنا نريد إنشاء نافذة من نمط QWidget

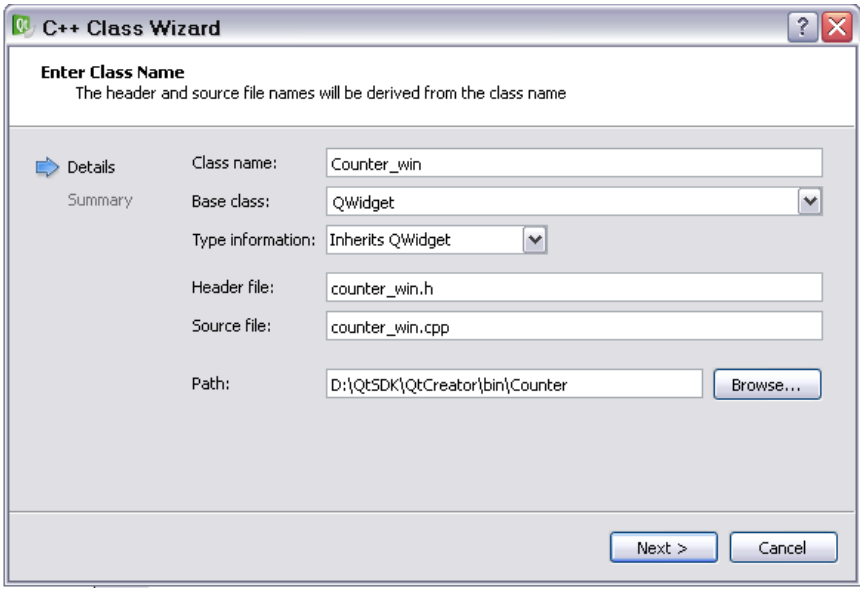
خانة Type Information يجب أن تكون قيمته Inherits QWidget لوراثة أنماط التوابع و وسطائها من الصف الأساسي. هنا سيصبح وسطاء بنّاء الصف (Counter_win) مثل وسطاء بنّاء الصف الأساسي الصف (QWidget).

نبقي الخانات الأخرى على قيمتها الافتراضية كما في الشكل (1.11):

سينشئ ملفين داخل مجلد المشروع (نفس مسار المشروع)

الأول : counter_win.h هذا الملف خاص بتعريف بنية الصف و بتصريح المناهج و المقابس و الإشارات داخل هذا الصف.

الملف الثاني : counter_win.cpp هذا الملف خاص بتحقيق الصف و المناهج و المقابس و الإشارات التي صرحنا عنها داخل الصف Counter_win الموجود داخل الملف الرأسي counter_win.h .



الشكل 1.11

ثمّ نضغط على زر التالي ومن ثمّ إنهاء.

إذا فتحت ملف المشروع Counte.pro ستجد بداخله الرمّاز الخاص بإضافة (تعريف) الملفات التي أضفناها وهي: (main.cpp - counter_win.cpp - counter_win.h).

- **Qt Creator يضيف بشكل تلقائي الرمّاز الخاص بتعريف الملف الذي نضيفه داخل ملف المشروع.**

❖ **الملف الرأسي** counter_win.h :

الآن دعنا نشرح الرمّاز الخاص بالملف الرأسي (counter_win.h):

```
#ifndef COUNTER_WIN_H
#define COUNTER_WIN_H

#include <QWidget>
```

```
class Counter_win : public QWidget
{
    Q_OBJECT
public:
    explicit Counter_win(QWidget *parent = 0);


signals:

public slots:

};

#endif // COUNTER_WIN_H
```

التوجيه #ifndef المكتوب على شكل شرط منطقي هو الذي يحدد تضمين الرمّاز المحصور بين

#define COUNTER_WIN_H و #endif وذلك لعدم تكرار تضمين الرمّاز مرة أخرى عند تضمين الملف الرأسي counter_win.h أكثر من مرة.

#include <QWidget>

ضمّنا الملف (الصف) QWidget.

Class Counter_win : public QWidget

عرّفنا صف اسمه Counter_win مشتق (يرث) من الصف QWidget و public تعني أن جميع المنهجيات و الخصائص العامة ضمن الصف QWidget سوف تبقى عامة.

Q_OBJECT

الماكرو Q_OBJECT خاص بلغة البرمجة Qt يكتب في القسم الخاص ضمن الصف لتفعيل الاتصال بين الكائنات من خلال الإشارات و المقابس (المستقبلات) ضمن هذا الصف بحيث يتاح لك استخدام الأحداث ضمن الصف, ويفعل عدة ميزات أخرى خاصة بنظام هيكلية كائن Qt سوف نذكر بعضها في عدة فصول من هذا الكتاب.

public:

explicit Counter_win(QWidget *parent = 0);

التصريح عن بنّاء الصف بالقسم العام (أيّ ذو مجال رؤية عام) بدايته الكلمة المحجوزة (explicit)

وهذا يعني أنّ البناء لا يطبق عليه أي تحويل ضمني(Implicit Conversion), وله الوسيط QWidget *parent = 0 بقيمته الافتراضية صفر .

أضفنا تصريح الهدّام ~Counter_win

signals:

هذا القسم خاص بالتصريح عن الأحداث الداخليّة الجديدة (إيّ إضافة حدث داخلي جديد خاص بنا) وسنتكلم عنه لاحقا في هذا الفصل .

public slots:

هذا القسم خاص بالتصريح عن المقابس العامة (أيّ المنهجية التي تستدعى عند قدح حدث ما) .

❖ الملف counter_win.cpp :

الرمّاز الخاص بالملف counter_win.cpp ويتضمن الرّمّاز الخاص بتحقيق الصف Counter_win الموجود داخل الملف الرأسي counter_win.h .

```
#include "counter_win.h"

Counter_win::Counter_win(QWidget *parent) :
    QWidget(parent)
{
}
```

ضمّنّا الملف الرأسي counter_win.h .

حققنا البناء الخاص بالصف Counter_win و مررنا وسيط البناء (parent) إلى وسيط البناء للصف الأساسي QWidget , قيمة الوسيط مساوية للصفر على القيمة الافتراضية.

❖ الملف main.cpp :

اكتب داخل الملف main.cpp الرمّاز التالي:

```
#include <QApplication>
```

```
#include <counter_win.h>

int main(int argc, char* argv[])
{
    QApplication app(argc, argv);

    Counter_win counter;
    counter.setWindowTitle("Counter");
    counter.show();

    return app.exec();
}
```

ضمّنّا الملف الرأسي counter_win.h

صرّحنا عن counter أنه مثيل للصف Counter_win

بما أنّ الصف Counter_win يرث الصف QWidget فالصف Counter_win له نفس المناهج العامة للصف QWidget مثل المنهجية setWindowTitle لتغيير قيمة عنوان النافذة و المنهجية show لإظهار النافذة.

جمع التطبيق و نفذه بواسطة زر التشغيل (CTRL + R) المحدد بمربع كما في الشكل(1.12):

الشكل 1.12

بعد تنفيذ التطبيق سوف تظهر نافذة بعنوان Counter.

نضيف التصريح عن الهدّام داخل الملف الرأسي counter_win.h في قسم التصريحات العامة:

```
~Counter_win();
```

نضيف الرمّاز الخاص بإنشاء زر (Add) و لوحة نص التي ستحتوي الرقم المتزايد.

نفتح الملف counter_win.h و نصرح عن الكائنات التالية في قسم التصريح الخاص (private):

```
QVBoxLayout* vLayout;
QPushButton* btn_add;
QLabel* lbl_num;
```

دعنا نضيف هذه الكائنات على النافذة.

نضيف الرمّاز الخاص بإنشاء حدث الكائنات المصرح عنها و إضافتها إلى النافذة

داخل كتلة البنّاء

```
vLayout = new QVBoxLayout();
lbl_num = new QLabel();
btn_add = new QPushButton();
lbl_num->setText(QString::number(0));
btn_add->setText("&Add");
vLayout->addWidget(lbl_num);
vLayout->addWidget(btn_add);
this->setLayout(vLayout);
```

الدالة QString::number() لتحويل الرقم الممرر إلى نص من نمط QString.

في الملف الرأسي counter_win.h في القسم الخاص صرح عن متحول من نمط int (رقم صحيح)

```
int num;
```

المتحول num خاص بالرقم الذي سيجرى عليه عملية الإضافة و عرضه في لوحة النص.

تحقيق الهدّام لحذف الكائنات المحدثة من الذاكرة:

داخل الملف counter_win.cpp اكتب الرمّاز التالي:

```
Counter_win::~Counter_win()
{
    delete vLayout;
    delete btn_add;
    delete lbl_num;
}
```

الآن دعنا ننشأ الحدث الخاص بالنقر على الزر Add لعرض قيمة المتحول num.

الخطوة الأولى : تصريح عن مقبس (منهجية خاصة بالاستدعاء عن وقوع الحدث) داخل الملف الرأسي counter_win.h في القسم العام للتصريح عن المقابس (slots):

```
public slots:
    void Add();
```

نحقق هذه المنهجية داخل الملف counter_win.cpp كالأتي:

```
Counter_win::Add()
{
    num++;
lbl_num->setText(QString::number(num));
}
```

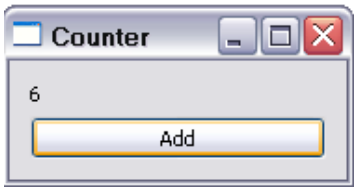
الآن دعنا ننشأ اتصال بين قدح الحدث و منهجية الحدث (Add) التي سوف تستدعى عند قدح الحدث (clicked()) على الزر Add, أضف الرمّاز التالي داخل كتلة البناء في الملف counter_win.cpp:

```
    num = 0;
    connect(btn_add, SIGNAL(clicked()),this ,
SLOT( Add() ));
```

أولا هيّئنا المتحول num (وضعنا قيمته الابتدائية تساوي الصفر)

ثانيا أنشأنا اتصال بين حدث النقر على الزر Add و المنهجية Add() المصرح عنها بأنها مقبس (slot)

الآن نفذ التطبيق سوف يظهر كما في الشكل(1.13):



الشكل 1.13

❖ كيفيّة إنشاء حدث خاص بنا.

سوف ننشأ الحدث above_num() يقدح عندما تكون قيمة الرقم الظاهر في لوحة النص (10)

فيظهر لنا رسالة توضيحية تفيد بعدم قدرتك على الزيادة فوق الرقم 10 وهي:

(sorry you can't add 1 to number!).

❖ **التصريح عن إشارة (signal) أي نوع (نمط) الحدث:**

أولا اذهب إلى الملف الرأسي counter_win.h وأضف داخل قسم الإشارات (signals) هذا الرمّاز:

```
signals:
    void above_num();
```

صرحنا عن إشارة (above_num) تقدح (emit) عندما تصبح قيمة المتحول num أكبر من (10).

أضف أيضا داخل القسم العام الخاص بالتصريح عن المقابس (داخل الملف الرأسي counter_win.h)

هذا الرمّاز:

```
    void show_msg();
```

هذا المقبس الخاص بإظهار الرسالة التي سوف تخبرنا أنه لاتستطيع زيادة قيمة المتحول على قيمة المتحول.

الآن دعنا نحقق المقبس show_msg ,

أكتب الرمّاز التالي داخلة الملف counter_win.cpp بعد تضمين الملف الرأسي QMessageBox داخل الملف counter_win.cpp بواسطة التوجيه

- `#include <QMessageBox>` **:-**

```
void Counter_win::show_msg()
{
QMessageBox msg;
msg.setText("sorry you can't add 1 to number!");
msg.exec();
}
```

msg هو مثيل للصف QMessageBox (الصف الخاص بصندوق الرسالة)

المنهجية setText() لوضع قيمة جسم صندوق الرسالة .

المنهجية exec() للإظهار صندوق الرسالة.

دعنا الآن نضيف الرمّاز الخاص بقدح الحدث (above_num).

عدل رمّاز المقبس (المنهجية) Add الموجود داخل الملف counter_win.cpp:

لتصبح كالأتي:

```
void Counter_win::Add()
{

    if(num == 10)
        emit above_num();
    else
        num++;
    lbl_num->setText(QString::number(num));
}
```

عندما يكون قيمة المتحول (num) يساوي (10) اقذف (اقدح) الحدث (above_num) بواسطة الكلمة المفتاحية emit , وإلّا أضف المتحول num بمقدار واحد ,ثمّ أظهر قيمة المتحول في لوحة النص.

بقي علينا أن نضيف الرمّاز الخاص بإنشاء اتصال بين الإشارة و المقبس .

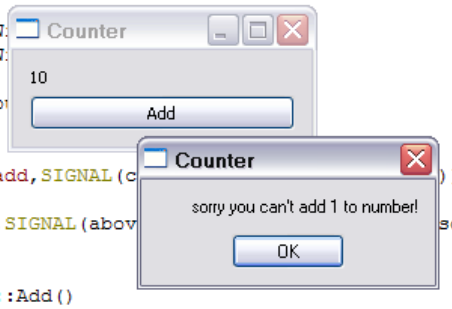
أضف هذا الرمّاز داخل كتلة البناء :

```
    connect(this, SIGNAL( above_num() ), this,
SLOT( show_msg() ));
```

استخدمنا المؤشر this للمرسل و المستقبل لأن الحدث سوف يقدح من الصف (Counter_win) و يستدعي المقبس أيضا من الصف (Counter_win).

نفذ التطبيق لترى الشكل (1.14).

```
    lbl_num = new QLabel();
    btn_add = new QPushButton();

    lbl_num->setText(QString::number(0));
    btn_add->setText("&Add");

    vLayout->addW
    vLayout->addW

    this->setLayo

    num = 0;
    connect(btn_add,SIGNAL(c                    ));

    connect(this,SIGNAL(abov                    sg()));
}

void Counter_win::Add()
{
        num++;

    if(num > 10)
```



الشكل 1.14

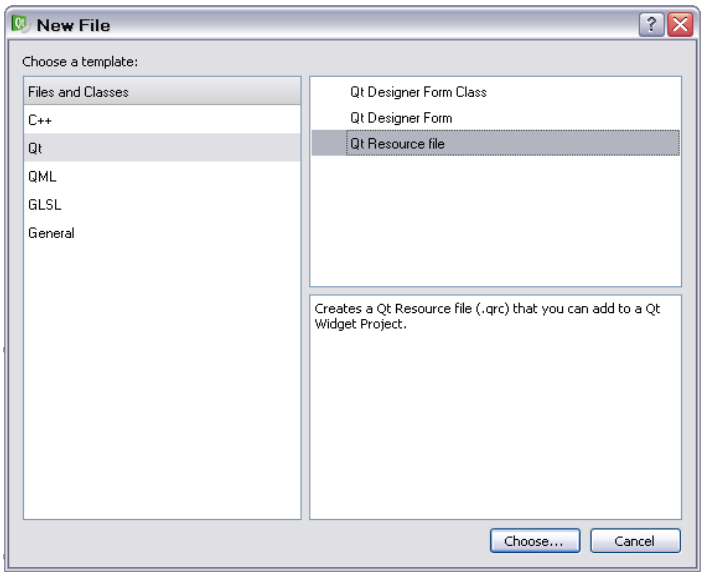
دعنا ننتقل إلى آخر فقرة داخل هذا الفصل و هي استخدام ملف المورد Resource File.

❖ استيراد أيقونة إلى ملف مورد و من ثمّ استخدامها في تطبيقنا .

لإنشاء ملف مورد يجب عليك أن تقوم بالخطوات التالية:

أولا انقر بزر الفأرة الأيمن على المشروع Counter ثم Add New… انقر على التبويب Qt

و حدد الخيار Qt Resource file كما في الشكل (1.15), ثمّ Choose في خانة Name حدد اسم ملف المورد وليكنimgs انقر على زر Next ثمّ انقر على زر Finish.



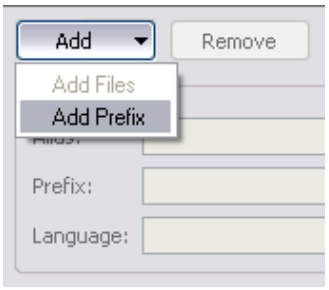
الشكل 1.15

سوف ينشأ ملف باسم imgs.qrc امتداده qrc أي (Qt Resource).

سوف تظهر نافذة لتحرير ملف المورد.

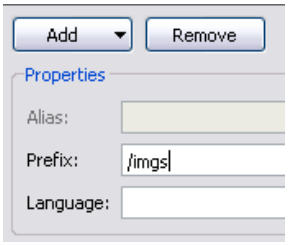
الآن يجب عليك أن تضيف لقب (Prefix) ويفيد اللقب بتقسيم الملفات التي تكون داخل ملف المورد (فهرسة الملفات الموجدة داخل ملف المورد) أيّ كالمجلدات في نظام الوندوز,

انقر على زر Add داخل المحرر واختار Add Prefix كما في الشكل (1.16):



الشكل 1.16

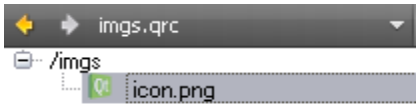
الآن في خانة Prefix ضع اسم اللقب وليكن imgs كما في الشكل (1.17) .



الشكل 1.17

لإضافة الصورة (الأيقونة) إلى ملف المورد داخل اللقب (imgs) من زر Add نأخذ Add Files ثم نحدد الصورة (الأيقونة) التي نريد ونضغط على زر Open لاستيرادها (نستطيع استيراد أيّ ملف) .

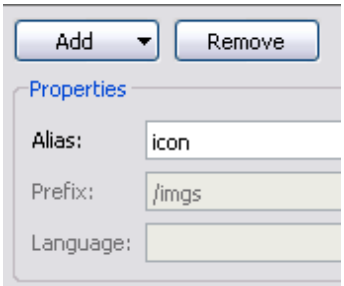
سوف يظهر لنا في نافذة تحرير ملف المصدر الخاص بال Qt الفهرس (اللّقب - Prefix -) و الصورة التي أضفناها تابعة للّقب (imgs) كما في الشكل (1.18):



الشكل 1.18

دعنا الآن نضيف اسم مستعار للصورة التي أضفناها إلى اللّقب (imgs) لاستخدامها برمجيا بواسطة اسمها المستعار (Alias).

ضع قيمة الخانة (Alias) الكلمة (icon) كما في الشكل (1.19), الخانة (Alias) موجود داخل صندوق الخصائص (Properties) الخاص بمحرر ملف المورد.



الشكل 1.19

انتهينا من تحرير ملف المورد .

ملاحظة:

عند إضافة ملف مورد إلى مشروع Qt مبيني بواسطة Qt Creator فأنه سوف يضيف بشكل تلقائي تعرف ملف المورد داخل المشروع كما في الرمّاز التالي:

```
RESOURCES += \
    imgs.qrc
```

دعنا ننتقل إلى الرمّاز الخاص باستيراد الصورة (من ملف مورد الذي أنشأناه سابقا) ووضعه كأيقونة لنافذة التطبيق Counter.

❖ **وضع أيقونة للتطبيق:**

أولا يجب علينا تضمين هذه الملفات الرأسية الخاصة بمعالجة الصورة (icon.png) التي استوردناها إلى ملف المورد (imgs) و تحويلها إلى أيقونة:

```
#include <QPixmap>
#include <QIcon>
```

الصف QPixmap يحتوي على مناهج خاصة لتحضير و عرض الصور.

الصف QIcon يحتوي على مناهج خاصة بمعالجة الأيقونات.

داخل كتلة البناء الخاص بالصف (Counter_win) نضيف الرمّاز التالي:

```
QPixmap img(":/imgs/icon");
QIcon icon(img);
this->setWindowIcon(icon);
```

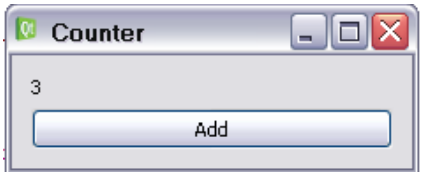
أولا: صرحنا عن مثيل اسمه (img) للصف QPixmap وضعنا قيمة الوسيط الخاص ببنائه مسار الصورة , هنا مسار الصورة (icon.png) داخل ملف المورد اسمها المستعار (icon) الموجودة داخل اللّقب -Prefix- (imgs) حيث يغنينا اسمها المستعار عن كتابة اسمها الحقيقي , أمّا الرمز(/:) في بداية المسار فهو لتحديد مسار الصورة من ملف المورد.

ثانيا: صرحنا عن مثيل اسمه (icon) للصف QIcon وله بناء يحتوي على وسيط من نمط QPixmap

وهنا وضعنا قيمة الوسيط الخاص بالبناء هي img المثيل للصف QPixmap الذي يحوي على صورة الأيقونة.

بقي علينا أن نضيف الأيقونة للنافذة بواسطة المنهجية ()setWindowIcon قيمة الوسيط الممرر يجب أن يكون من نمط QIcon ,وضعنا قيمة الوسيط الممرر لمنهجية ()setWindowIcon المثيل icon.

نفذ التطبيق سوف ترى أن الأيقونة قد وضعت,كما في الشكل (1.20).



الشكل 1.20

• خلاصة الفصل:

قد تعلمنا في هذا الفصل أساسيات لغة Qt كاستخدام الصف QApplication و دخول التطبيق في حلقة الأحداث, وأيضا أساسيات Qt Creator, حيث تعلمنا كيفيّة إنشاء مشروع Qt و بناء نافذة QWidget و إظهارها و تغيير قيمة عنوانها و استخدام لوحة النص QLabel و الزر QPushButton و أداة تنسيق الأدوات QVBoxLayout و كيفيّة إنشاء حدث داخلي(إشارة-Signal) و ربطه بمنهجية (مقبس-Slot) عند قدح الحدث, و استخدام ملف المورد Resource File, و تجميع التطبيق و تنفيذه بواسطة أداة MinGW و أيضا بواسطة Qt Creator.

دعنا ننتقل إلى الفصل التالي (الأدوات - Tools).

الفصل الثاني

كائنات واجهة المستخدم الرسومية و الصفوف الأساسية لتطوير برمجيات احترافية

Graphics User Interface Objects and basic classes for professional software development using Qt framework

- **مقدمة Intro** :

كائنات واجهة المستخدم الرسومية هي كائنات مرئية ذات أشكال مثل مستطيل- دائرة – الخ, تتفاعل مع المستخدم بوساطة أدوات الإدخال كالفأرة – لوحة المفاتيح الخ.. , ايعند قيام المستخدم بفعل ما(كالنقر على الكائن)

سوف يقدح هذا الكائن إشارة (حدث داخلي - Signal) وبالتالي ينفذ أمر ما قد هيئناه مسبقا (المنهجية التي تحوي الأمر تدعى مقبس - Slot) .

في عصرنا الحالي أي تطبيق يحوي على كائنات واجهة المستخدم إن كان تطبيق موجة لأنظمة سطح المكتب أو لأنظمة الأجهزة الذكية , ككائن الزر أو كائن إدخال النص أو كائن عرض التاريخ الخ..

Qt Creator يستخدم البرنامج Qt Designer لإنشاء الواجهات الذي **يتضمن صندوق أدوات يحتوي على كائنات المستخدم الرسومية ويساعد نا Qt Designer بتصميم واجهات التطبيق.**

سوف نتكلم في هذا الفصل عن الكائنات الموجودة داخل صندوق أدواتQt Creator وأهم خصائصها و أحدائها وعن إنشاء قوائم بوساطة الصف QMenu و الصف QAction و كيفيّة استخدام رمّاز CSS لتنسيق و تحسين مظهر أدوات واجهة المستخدم الرسومية و سنتكلم عن صف المتمم QCompleter و التحقق من صحة نص كائنات الإدخال بواسطة الكائن QValidatorوتفاعلها مع كائن التعابير القياسية. QRegExp

❖التطبيق الأول لإنشاء واجهة مستخدم باستخدام Qt Creator

❖ First application for create user interface using Qt Creator

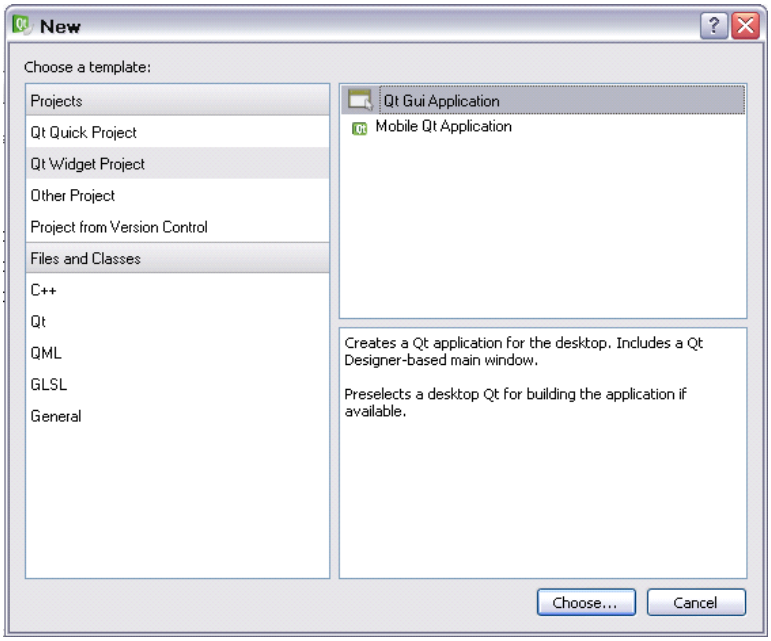
سننشأ واجهة مستخدم (QWidget) باستخدام المعالج الذكي الخاص بإنشاء مشروع جديد ضمن

Qt Creator.

من قائمة File نختار New File or Project (Ctrl + N) من قائمة Projects نحدد التبويب

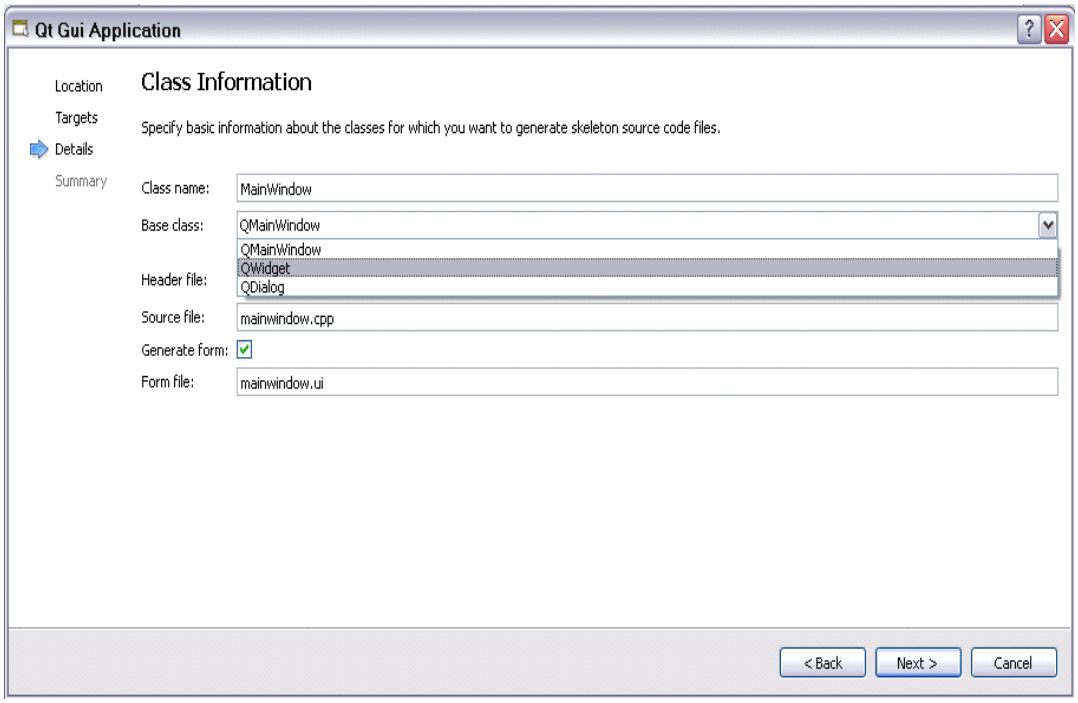
Qt Widget Project ثم نحدد على Qt Gui Application وننقر على زر Choose

كما في الشكل:(2.1)



الشكل2.1

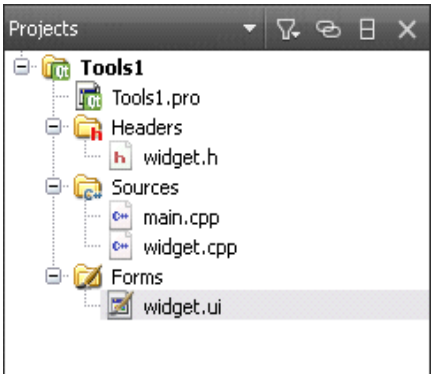
اسم المشروع Tools1 ثم ننقر على الزر التالي و من ثم التالي في صفحة Class Information نضع قيمة الخانة Bass Class على الخيار QWidget كما في الشكل (2.2) ثم التالي ومن ثمّ إنهاء.



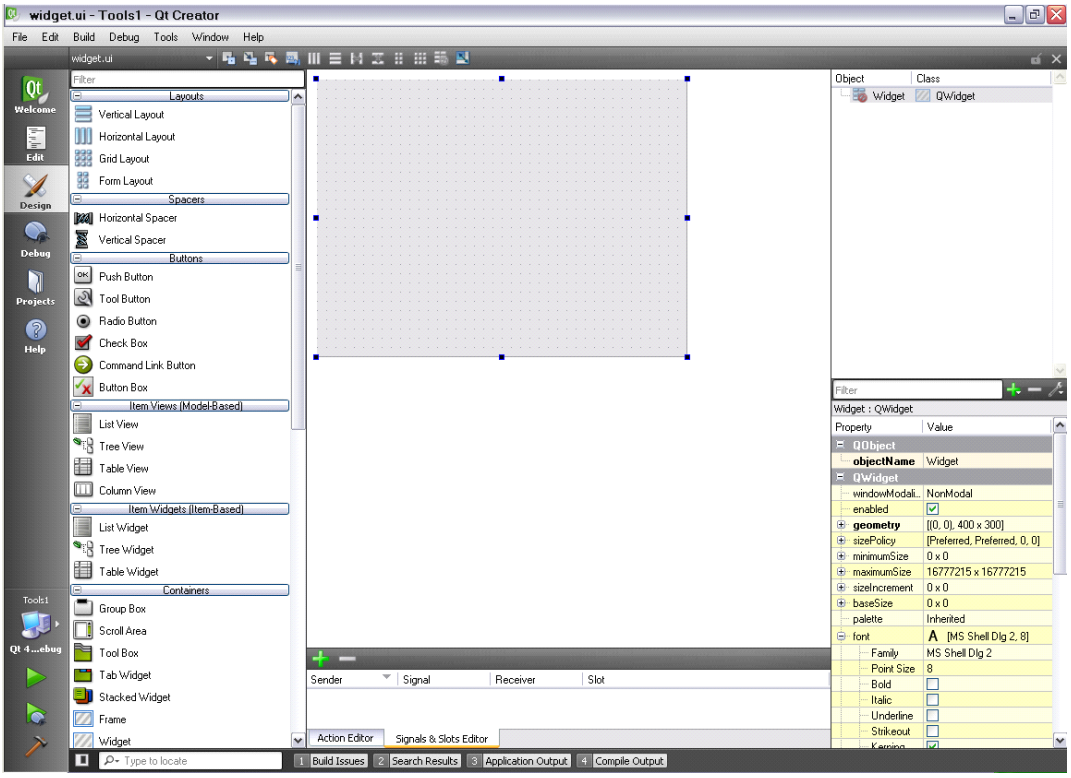
الشكل2.2

ستجد أنه أضاف إلى شجرة المشروع الفهرس Forms كما في الشكل (2.3)والذي يحتوي على

واجهات المستخدم المرئية تجد داخله الملف, widget.ui افتح هذا الملف ستجد أنّ Qt Creatorسينقلك إلى واجهة تصميم مرئية و على الجهة اليسرى داخل Qt Creator صندوق الأدوات و على الجهة اليمنى تبويب الخصائص كما في الشكل.(2.4)



الشكل3.2



الشكل2.4

نفذ التطبيق بواسطة زر Run (Ctrl + R) ستجد قد ظهرت نافذة تحمل العنوان.Widget

إذا فتحت ملف widget.ui بواسطة المفكرة أو محرر نصوص آخر ستجد بداخله رمّاز (XML) خاص بتوصيف الواجهة و الأدوات الموجودة ضمنه ,هذا الملف يأخذه المترجم UIC كدخل و يكون الخرج ملف الترويسة.ui_widget.h

***ملاحظة*:سوف نتكلم عن كيفية تفسير و تجميع مشروع. Qt في ملحق هذا الكتاب.**

داخل ملف widget.cpp ستجد هذا الرمّاز:

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);
}
Widget::~Widget()
{
    delete ui;
```

```
}
```

سوف يقوم البنّاء بتنصيب واجهة المستخدم المنشأة (widget.ui) ضمن هذا الصف ,لاستخدامها داخل المشروع أما الهدّام لتدمير الكائن ui الكائن الخاص بواجهة المستخدم.

أما ملف الترويسة ui_widget.h سوف تجده داخل المجلد Tools1-build-desktop الذي يحوي الملفات الخاصة بتجميع المشروع , الملف ui_widget.h يحوي رمّاز فضاء التسمية ui (namespace)والذي يحتوي على الصف Widget المشتق من الصف Ui_Widget حيث المنهجية setupUi موجودة داخله ,المنهجية setupUi الخاصة بتنصيب واجهة المستخدم (widget.ui) على الكائن widget الممرر كوسيط لها.

ملف التروسية ui_widget.h:

```
/********************************************************************************
** Form generated from reading UI file 'widget.ui'
**
** Created: Wed 7. Sep 22:20:26 2011
**      by: Qt User Interface Compiler version 4.7.3
**
** WARNING! All changes made in this file will be lost when recompiling UI file!
********************************************************************************/
#ifndef UI_WIDGET_H
#define UI_WIDGET_H
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QHeaderView>
#include <QtGui/QWidget>
QT_BEGIN_NAMESPACE
class Ui_Widget
{
public:
    void setupUi(QWidget *Widget)
    {
        if (Widget->objectName().isEmpty())
            Widget->setObjectName(QString::fromUtf8("Widget"));
```

```
        Widget->resize(400, 300);
        retranslateUi(Widget);
        QMetaObject::connectSlotsByName(Widget);
    } // setupUi
    void retranslateUi(QWidget *Widget)
    {
        Widget-
>setWindowTitle(QApplication::translate("Widget",
"Widget", 0, QApplication::UnicodeUTF8));
    } // retranslateUi
};
namespace Ui {
    class Widget: public Ui_Widget {};
} // namespace Ui
QT_END_NAMESPACE
#endif // UI_WIDGET_H
```

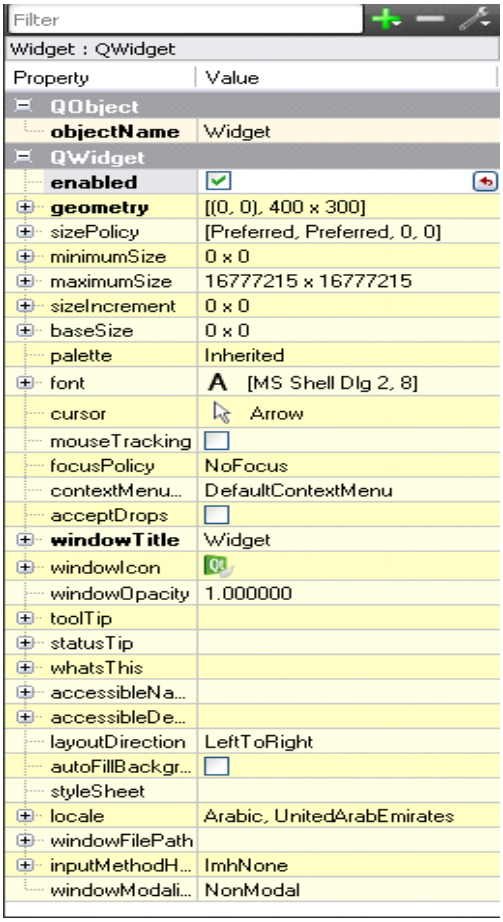
ui_widget.h

❖خصائص واجهة المستخدم (ui)

Properties UI:

سنتكلم عن أهم خصائص واجهة المستخدم , الكائن.QWidget

اذهب إلى واجهة المستخدم widget.ui ستجد صندوق الخصائص كما في الشكل.(2.5)



الشكل2.5

ستجد التبويب QObject الصف الأساس للصف QWidget والتبويب QWidget والذي هو الصف الأساس لجميع كائنات واجهة المستخدم جميع خصائص الصف QWidget ستجدها ذاتها لجميع أدوات واجهة المستخدم لأنها موروثة من الصف QWidget بالإضافة إلى خصائص أخرى خاصة بالأداة ذاتها .

نبدأ بالخاصيّة enabled خاصيّة التفعيل بما أنّ جميع أدوات واجهة المستخدم موروثة من QWidget

فسيأثر تغيير قيمة هذه الخاصيّة على جميع الأدوات الموجودة داخل هذه الواجهة.

الخاصيّة geometry لتغيير توضع و أبعاد النافذة , لها أربع وسطاء:

X , Y , Width , Height

الخاصيّة sizePolicy لتناسب حجم الأداة مع الحجم الحالي للكائن الحاوي أو النافذة الأب, و لها أربع وسطاء:

- **Horizontal Policy**

سبع قيم له:

Fixed , Minimum , Maximum , Preferred , Minimum Expanding, Expanding , Ignored .

التعداد QSizePolicy::Policy :

QSizePolicy::Fixed	حجم الأداة داخل أداة التخطيط ثابت
QSizePolicy::Minimum	أصغر حجم ممكن للأداة هو حجمها المثالي أو أكبر
QSizePolicy::Maximum	أكبر حجم للأداة هو حجمها المثالي أو أصغر
QSizePolicy::Preferred	الحجم المثالي هو الحجم المستخدم وممكن أن تكبر أو تصغر الأداة بتغيير حجم النافذة
QSizePolicy::MinimumExpanding	الحجم المثالي هو الأفضل و لكن ممكن أن يزداد
QSizePolicy::Expanding	الحجم المثالي هو الحجم المستخدم ولكن لديه ميول للزيادة
QSizePolicy::Ignored	تجاهل الحجم المثالي للأداة أي قدرة الأداة على زيادة حجمها و نقصانه بحرية

- Vertical Policy

وله نفس قيم الوسيط السابق.

- Horizontal Stretch
- Vertical Stretch

الخاصيّة minimumSize أصغر قيمة يأخذها الكائن لحجمه وله وسيطين:

- Width
- Height

الخاصيّة maximumSize أكبر قيمة يأخذها الكائن لحجمه وله وسيطين:

- Width
- Height

الخاصيّة sizeIncrement مقدار زيادة الحجم بالتناسب مع ,baseSize وله وسيطين:

- Width

- **Height**

الخاصيّة baseSize الحجم الأساسي للكائن , وله وسيطين:

- **Width**
- **Height**

الخاصيّة palette تغيير ألوان تصميم الكائنات.widgets

الخاصيّة font لتغيير مظهر الخط.

الخاصيّة cursor لتغيير شكل المؤشر.

الخاصيّة mouseTracking عند تفعيلها يستجيب الكائن إلى حدث تحريك مؤشر الفأرة عليه من دون الضغط على أي زر للفأرة و إذا كانت غير مفعلة فلا يستجيب إلا عند الضغط على زر من الفأرة و تحريك مؤشرها معا.

الخاصيّة focusPolicy لتحديد كيفيّة أسلوب التركيز على الكائن وله خمس وسطاء, التعداد Qt::FocusPolicy :

لا تركيز له	**Qt::NoFocus**
التركيز على الكائن بواسطة المفتاحTab	**Qt::TabFocus**
التركيز على الكائن بواسطة النقر فقط	**Qt::ClickFocus**
التركيز على الكائن بواسطة مفتاح Tab و النقر و الأسهم	**Qt::StrongFocus**
التركيز على الكائن بواسطة عجلة الفأرة	**Qt::WheelFocus**

الخاصيّة contextMenuPolicy للتحكم بكيفيّة إظهار القائمة المنبثقة للكائن ,وله خمس وسطاء ,التعداد Qt::ContextMenuPolicy :

منع إظهار القائمة المنبثقة	**Qt::NoContextMenu**
إظهار القائمة المنبثقة بالشكل الافتراضي أيّ بواسطة تحقيق الحدث contextMenuEvent(QContextMenuEvent) تظهر القائمة المنبثقة بواسطة الضغط على زر الفأرة الأيمن	**Qt::DefaultContextMenu**
كل (Actions) المضافة للكائن (widget) ستظهر ضمن القائمة المنبثقة	**Qt::ActionsContextMenu**
الكائن (widget) سيقذف الحدث -signal customContextMenuRequested(QPoint)	**Qt::CustomContextMenu**

يتم إظهار النافذة المنبثقة بواسطة هذا الحدث.	
عدم إظهار القائمة المنبثقة بواسطة تحقيق الحدث contextMenuEvent(QContextMenuEvent) و إظهارها بواسطة كل من الحدث (mousePressEvent)و (mouseReleaseEvent)	Qt::PreventContextMenu

الخاصيّة acceptDrops عند تفعيلها نهيئ الكائن widget لاستقبال أحداث الإفلات.

الخاصيّة windowTitle تأخذ قيمة نصيّة , قيمة عنوان النافذة.

الخاصيّة windowIcon مسار أيقونة النافذة.

الخاصيّة windowOpacity قيمة الشفافيّة للكائن.

الخاصيّة toolTip تأخذ قيمة نصية تظهر ك تلميح للكائن.

الخاصيّة statusTip تأخذ قيمة نصية لإظهارها في شريط الحالة عند حركة الفأرة على الكائن widget .

الخاصيّة whatsThis تأخذ قيمة نصيّة , تظهر صندوق يحتوي على معلومات نصية , تعمل فقط مع النوافذ التي تحتوي على زر مساعدة" ؟ "بجانب زر إغلاق.

الخاصيّة accessibleName تأخذ قيمة نصية, مثيل لاسم الكائن ,لا تؤثر على سلوك الكائن widgetإنما فقط للتعريف.

الخاصيّة accessibleDescription تأخذ قيمة نصية , وصف الكائن ,لا تؤثر على سلوك الكائن widget إنما فقط للتعريف .

الخاصيّة layoutDirection تحدد اتجاه الكائنات وله ثلاث وسطاء , التعداد Qt::LayoutDirection :

اتجاه الكائنات من اليسار إلى اليمين	Qt::LeftToRight
اتجاه الكائنات من اليمين إلى اليسار	Qt::RightToLeft
اتجاه الكائنات نفس اتجاه الصف الأب أيّ يأخذ الاتجاه بشكل تلقائي من widget الأساسي أو صف التطبيق	Qt::LayoutDirectionAuto

الخاصيّة autoFillBackground إذا كانت مفعلة هذه الخاصية سوف يملأ لون الخلفية اللون الافتراضي له المأخوذ من الخاصية palette-QPalette::Window- قبل استدعاء حدث الرسم paintEvent

الخاصيّة styleSheet يكتب داخلها سكربت CSS لتغيير مظهر widgets الكائنات.

الخاصيّة locale لتهيئة إعدادات المكان اللغة و الدولة حيث ينسق الأرقام و العملة الخ ... بشكل تلقائي

حسب اللغة و الدولة المختارة , وله وسيطين:

اللغة المستخدمة	Language
الدولة – تنسيق العملة و التاريخ الخ .... حسب الدولة المختارة-	Country

الخاصيّة inputMethodHints لتحديد صيغة الإدخال , كإدخال الأرقام فقط أو إدخال كافة المحارف بشكلها الكبير الخ ...قيمتها الافتراضية ,ImhNone وله خمسة عشر وسيط كالوسيط

ImhDigitsOnlyإذا كان مفعّل فالسماحية لإدخال محرف رقم ,أيّ من الرقم 0 إلى الرقم9.

الخاصيّة windowModality لتخصيص نمط حظر النوافذ الأخرى الظاهرة التابعة للتطبيق ذاته من الإدخال عند إظهار هذه النافذة وله ثلاث وسطاء , التعداد Qt::WindowModality :

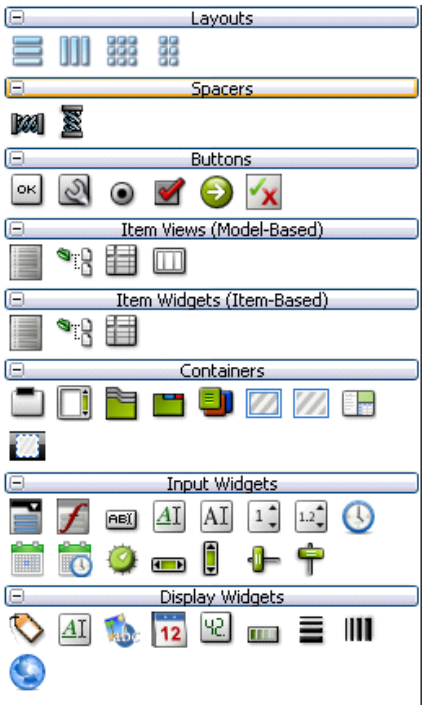
عدم حظر النوافذ الأخرى من الإدخال إن كان من النوافذ الآباء أو الأبناء أو الأخوة	Qt::NonModal
لا تحظر من الإدخال ,تبقى هذه النافذة فوق النافذة الأب	Qt::WindowModal
حظر على مستوى التطبيق ,سوف تحظر من الإدخال جميع النوافذ الظاهرة عند إظهار هذه النافذة	Qt::ApplcationModal

انتهينا الآن من خصائص الكائن QWidget

دعنا ننتقل إلى صندوق الأدوات.

- صندوق الأدواتTools Box :

سوف نتكلم الآن عن الأدوات) الكائنات المرئيّة (الموجودة داخل صندوق الأدوات الشكل (2.6).



الشكل2.6

يتضمن صندوق الأدوات ثمانية تبويبات , كل تبويب يحتوي على مجموعة من الأدوات المرئيّة (كائنات واجهة المستخدم) ويدل اسم التبويب على عمل الأدوات التي يحتويها ,انظر إلى الجدول:

اسم التبويب	عمل الكائنات التي يحتويها
Layouts	يحتوي على الأدوات الخاصة بالتخطيط تنسيق توضع الكائنات
Spacers	يحتوي على الأدوات الخاصة بتنسيق مقدار التباعد بين الكائناتwidgets
Buttons	يحتوي على كائنات الأزرار بأنواعها
Item Views (Model-Based)	يحتوي على الكائنات الخاصة بإظهار البيانات على شكل قائمة جدول – قائمة شجرية الخ(تضع البيانات بداخله على شكل نموذج) أي كائن مشتق من كائناتmodel
Item Widgets (Item-Based)	يحتوي على الكائنات الخاصة بإظهار البيانات على شكل قائمة مثل جدول – قائمة شجرية الخ.. تضع البيانات بداخله على شكل نموذج (model) أو بإضافة عمود و صف و خليّة مباشرة ,لأن

كائناته موروثة من الصف view و مطورة عنه.	
يحتوي على الكائنات الخاصة باحتواء الكائنات و تنسيقها ضمن مجموعات.	Containers
يحتوي على الكائنات الخاصة بالإدخال كأداة شريط الانزلاق و أداة إدخال النص الخ..	Input Widgets
يحتوي على الكائنات الخاصة بالعرض كأداة اللافتة	Display Widgets

نبدأ الآن بشرح أهم مناهج و خصائص الأدوات (كائنات واجهة المستخدم) .

❖الأدوات التابعة لتبويب التخطيط (Layouts) :

verticalLayout (QVBoxLayout) :كائن لتنسيق توضع الأدوات التي تكون داخله بشكل عمودي

ضمن عمود واحد , له خصائص لتعديل مسافة الهوامش كخاصيّة (layoutLeftMargin) و خصائص لتعديل مسافة تباعد الكائنات (widgets) عن بعضها, خاصيّة (layoutSpacing).

horizontalLayout (QHBoxLayout) : نفس عمل الكائن السابق و لكن ينسق توضع الأدوات بشكل أفقي

ضمن صف واحد.

gridLayout (QGridLayout) : نفس عمل الكائن السابق و لكن ينسق توضع الأدوات بشكل شبكيّ ضمن صفوف وأعمدة , تستطيع وضع عدة كائنات بشكل أفقيّ و عموديّ,وله عدة خصائص أخرى غير موجودة بالكائن السابق كخاصيّة (layoutRowMinimumHeight)لتقييد أصغر قيمة ارتفاع يأخذها الصف المضمن داخلها.

formLayout (QFormLayout) :كائن لتنسيق توضع الأدوات التي تكون داخله بشكل شبكي لكن يحتوي فقط على عمودان ,تستطيع إدراج صفوف العدد الذي تريده) كل صف يحتوي على عمودين– حقلين

أهم مناهجه :

QFormLayout::addRow(const QString & label, QWidget* widget)

لإضافة صف يحتوي الحقل الأول منه على لافتة نص و الحقل الثاني كائن (widget) ككائن إدخال النص

(QLineEdit) .

```
QFormLayout* frmLayout = new QFormLayout();

QLineEdit* fName = new QLineEdit();

frmLayout->addRow(“First Name” , fName);
```

QFormLayout::insertRow(int index, const QString &label, QWidget* widget)

نفس عمل المنهج السابق ولكن تستطيع هنا تحديد رقم توضع الصف المدرج.

```
QFormLayout* frmLayout = new QFormLayout();

QLineEdit* fName = new QLineEdit();

frmLayout->insertRow (2 ,“First Name” , fName);
```

- **الأدوات التابعة لتبويب تنسيق التباعد بين الكائنات (Spacers)** :

horizontalSpacer , verticalSpacer كائن لتنسيق مسافة التباعد بين الكائنات , أهم خصائصه

orientation تنسيق التباعد بشكل أفقيّ أو عموديّ ,وله وسيطين , Vertical : Horizontal .

خصائص الارتفاع و العرض , Width , Height وخاصيّة sizeType تأمين حجم كائن Spacerعند تغيير حجم الكائن الحاوي ,وله سبع وسطاء قد تمّ ذكرهم مسبقا عند شرح الخاصيّة.sizePolicy

- **الأدوات التابعة لتبويب الأزرار (Buttons)** :

Push Button (QPushButton) كائن زر عادي يرث الصف QAbstractButton , أهم خصائصQAbstractButton :

الخاصيّة text النص الظاهر على الزر,

الخاصيّة icon لوضع شعار داخل الزر إما من ملف خارجي أو من ملف مورد.

الخاصيّة iconSize لتعديل حجم الشعار داخل الزر.

الخاصيّة shortcut لوضع اختصار مفاتيح من لوحة المفاتيح, لتقدح حدث النقر على الزر.

الخاصيّة checkable عند تفعيل هذه الخاصيّة يصبح الزر على شكل - زر تحديد - أيّ check Button.

الخاصيّة checked تفعيل التحديد أو عدم تفعيله.

الخاصيّة autoExclusive تحديد فردي , أيّ عند تفعيل هذه الخاصية مع الخاصية checkableسوف تكون رد فعل الزر عند النقر عليه أنه سوف يلغي بشكل تلقائي تحديد جميع الأزرار الموجودين ضمن نفس الحاوية الموجود فيها هذا الزر . كما في التعامل مع QRadioButton.

أهم خصائص: QPushButton

الخاصيّة default عند تفعيل هذه الخاصيّة سوف يصبح الزر الافتراضي أيّ عند الضغط على مفتاحEnter

سوف يقدح حدث النقر لدى الزر.

الخاصيّة flat عند تفعيل هذه الخاصيّة سوف يأخذ الزر شكلا مسطح.flat

Tool Button (QToolButton)كائن زر استخدامه غالبا داخل شريط الأدوات , يأخذ شكل زر خاص بفتح نافذة اختيار ملف أو نافذة حوار ما ...يرث من الصف QAbstractButton ,أهم خصائصه:

popupModeأسلوب إظهار القائمة المرتبطة بهذا الزر , وله ثلاث وسطاء, التعداد QToolButton::ToolButtonPopupMode :

تظهر القائمة بعد الضغط و الإمساك بتأخير زمني	**QToolButton::DelayedPopup**
يظهر مؤشر على جانب الزر , عند الضغط عليه تظهر القائمة مباشرة	**QToolButton::MenuButtonPopup**

تظهر القائمة بعد الضغط مباشرة من دون تأخير زمني	QToolButton::InstantPopup

toolButtonStyleلتعديل توضع النص بجانب الشعار,أو لعرض النص فقط أو الشعار فقط , له خمس وسطاء, التعداد Qt::ToolButtonStyle :

لإظهار شعار فقط من دون إظهار نص الزر	Qt::ToolButtonIconOnly
لإظهار نص فقط من دون إظهار الشعار	Qt::ToolButtonTextOnly
لإظهار النص بجانب الشعار	Qt::ToolButtonTextBesideIcon
لإظهار النص تحت الشعار	Qt::ToolButtonTextUnderIcon
حيث يأخذ تغيرات المنهجيةStyle	Qt::ToolButtonFollowStyle

الخاصيّة autoRaise عند تفعيلها سيظهر الزر من دون انتفاخ في سطحه أيّ سيصبح شكله Flat.

الخاصيّة arrowType لوضع شعار سهم داخل الزر , له خمس وسطاء, التعداد Qt::ArrowType :

من دون شعار سهم,سيحتوي الشعار المدرج بالخاصيّةIcon	Qt::NoArrow
سيظهر شعار سهم باتجاه الأعلى,ويختفي الشعار المدرج بالخاصيّةIcon	Qt::UpArrow
سيظهر شعار سهم باتجاه الأسفل,ويختفي الشعار المدرج بالخاصيّةIcon	Qt::DownArrow
سيظهر شعار سهم باتجاه اليسار,ويختفي الشعار المدرج بالخاصيّةIcon	Qt::LeftArrow
سيظهر شعار سهم باتجاه اليمين,ويختفي الشعار المدرج بالخاصيّةIcon	Qt::RightArrow

Radio Button (QRadioButton) كائن زر الاختيار الفردي.

Check Box (QCheckBox) كائن زر الاختيار المتعدد, مشتق من الصف QAbstractButton, له خاصيّة tristate عند تفعيلها يصبح الكائن ذو ثلاث حالات) حالة عدم اختيار – حالة اختيار جزئي – حالة اختيار .(له حدث يدعى

QCheckBox::stateChanged(int)

يقدح الحدث عندما تغير حالة التحديد للزر , له وسيط يحدد قيمة حالة الاختيار الحاليّة , التعداد **Qt::CheckState** :

حالة عدم تحديد	**0**	**Qt::Unchecked**
حالة تحديد جزئيّ	**1**	**Qt::PartiallyChecked**
حالة تحديد	**2**	**Qt::Checked**

Command Link Button (QCommandLinkButton) زر ارتباط , مشتق من الصف , QPushButton يأخذ سمة ويندوز فيستا ,له خاصية description ليضع وصف للزر بالخط الصغير بأسفله.

Button Box (QDialogButtonBox) صندوق يحتوي على مجموعة من الأزرار القياسية , تستطيع إضافة و حذف أزرار من خلال اختيار الأزرار , بواسطة الخاصيّة standardButtons, وله خاصيّة تحدد الاتجاه orientation لها وسيطين أفقي – عمودي.

الأدوات التابعة لتبويب عرض البيانات على شكل قائمة (Model-Based):

List View (QListView) كائن لإظهار البيانات على شكل قائمة , يمكن إدراج بيانات نصيّة أو صور شعارات -أيقونات لكن بعد تهيئتها و إدراجها داخل صف نموذج (Model) .لإدراج البيانات يجب أن نحقق ثلاث مراحل,هي:

المرحلة الأولى تحقيق أي كائن من صف النموذج (Model) أو استخدام كائن محقق كالكائن QStandardItemModel .

المرحلة الثانية إدخال البيانات داخل الصف المحقق من صف النموذج.

المرحلة الثالثة إعطاء قيمة الخاصيّة QListView::setModel للكائن الكائن QListView هي الصف الذي حققناه سابقا.

البنية الهرمية للصف: QListView

QObject → QWidget → QFrame →QAbstractScrollArea →QAbstractItemView → QListView

Tree View (QTreeView)كائن لإظهار البيانات على شكل قائمة شجرية ,يمكن إدراج بيانات نصيّة أو صور - شعارات لكن بعد تهيئتها و إدراجها داخل صف نموذج , (Model) لإدراج البيانات يجب أن نحقق الثلاث مراحل المذكورين مسبقا.

البنية الهرمية للصف:QTreeView

QObject → QWidget → QFrame →QAbstractScrollArea →QAbstractItemView →QTreeView →Header

Table View (QTableView) كائن لإظهار البيانات على شكل جدول يمكن إدراج البيانات داخله بعد تحقيق الثلاث المراحل المذكورة مسبقا.

البنية الهرمية للصف:QTableView

QObject → QWidget → QFrame →QAbstractScrollArea →QAbstractItemView →QTableView →Header

Column View (QColumnView)كائن لإظهار البيانات على شكل أعمدة , يمكن إدراج : البيانات داخله بعد تحقيق الثلاث المراحل المذكورة مسبقا.

البنية الهرمية للصف:QColumnView

QObject → QWidget → QFrame →QAbstractScrollArea →QAbstractItemView →QColumnView

مثال عن استخدام الكائن QTableView :

```
QStandardItemModel* model = new QStandardItemModel(3,3);

model->setHeaderData(0,Qt::Horizontal,"First Name");

model->setHeaderData(1,Qt::Horizontal,"Last Name");

model->setHeaderData(2,Qt::Horizontal,"Birthday");

model->setData(model->index(0,0),"Hasan");

model->setData(model->index(0,1),"Morhej");

model->setData(model->index(0,2),"1989/7/17");

model->setData(model->index(1,0),"Mouhammad");
```

```
model->setData(model->index(1,1),"Ali");
model->setData(model->index(1,2),"1988/8/9");
model->setData(model->index(2,0),"Bouchra");
model->setData(model->index(2,1),"Mansour");
model->setData(model->index(2,2),"1991/7/31");
tableView->setModel(model);
```

- **الأدوات التابعة لتبويب عرض البيانات على شكل قائمة (Item-Based):**

List Widget (QListWidget) كائن لإظهار البيانات على شكل قائمة , يمكن إدخال البيانات عن طريق تحقيق صف نموذج (Model) أو عن طريق مناهج تابعة للكائن QListWidgetأهمها:

```
QListWidget::addItem(QListWidgetItem* item);
```

لإضافة عنصر إلى الكائن QListWidget من نمط QListWidgetItem

الكائن QListWidgetItem عنصر قائمة, نستخدمه إذا كنّا نريد تنسيق الخط و إضافة شعار إلى النص...

```
QListWidget::addItem(const QString &label);
```

إضافة عنصر نصي إلى القائمة.

```
QListWidget::addItems(const QStringList &labels);
```

لإضافة عناصر نصيّة إلى القائمة باستخدام الكائن QStringList نكتب :

```
QStringList  Strs;
Strs << “Name1” << “Name2” << “Name3” ;
QListWidget* listWidget= new QListWidget();
listWidget ->addItems(Strs);
```

الكائن QStringList يستخدم لإضافة قائمة من النصوص.

QListWidget::insertItem(QListWidgetItem* item, int index);

QListWidget::insertItem(const QString &label, int index);

QListWidget::insertItems(const QStringList &labels, int index);

لإدراج عنصر مع تحديد رقم الصف الذي سينحشر فيه العنصر.

Int QListWidget::count();

يرجع عدد الصفوفالعناصر الموجودة فيه.

Int QListWidget::currentRow() const

يرجع رقم الصف عنصر المحدد.

QListWidgetItem* QListWidget::currentItem() const

يرجع عنصر القائمة المحدد.

QListWidget::setCurrentRow(int row);

لتحديد صف عنصر من خلال رقم الصف الممرر.

QListWidget::sortItems(Qt::SortOrder order = Qt::AscendingOrder);

ترتيب العناصر , إذا تركت وسيط المنهجية خالي فسيأخذ الوسيط الإفتراضي , أي سوف يرتب العناصر حسب الترتيب الأبجدي للأحرف , أمّا القيمة الثانية Qt::DescendingOrder فسوف يرتب العناصر بعكس الترتيب الأبجدي للأحرف.

QList<QListWidgetItem*> QListWidget::findItems(const QString &text, Qt::MatchFlags flags) const

بحث ضمن العناصر الموجودة داخل الكائن عن العنصر الممر للوسيط , text تعيد هذه المنهجية القيمة على شكل كائن قائمة QList العناصر التي داخله تكون من نمط مؤشر عنصر قائمة*QListWidgetItem

أمّا الوسيط flags فهو العلم الذي يحدد أسلوب البحث

الأعلام المتاحة لطريقة عمل البحث, التعداد **Qt::MatchFlag** :

البحث عن المحتوى بدقة أيّ يجب أن يكون	**Qt::MatchExactly**

المحتوى كامل (المحتوى المدخل هو ذاته محتوى العنصر) مع مراعاة الأحرف الكبيرة و الصغير	
البحث عن النص بدقة أيّ يجب أن يكون النص كامل) (النص المدخل هو ذاته نص العنصر) مع عدم مراعاة الأحرف الكبيرة و الصغير	Qt::MatchFixedString
البحث عن أيّ عنصر يحتوي جزء من النص المدخل	Qt::MatchContains
البحث عن أيّ عنصر يحتوي في بدايته النص المدخل مع عدم مراعاة الأحرف الكبيرة و الصغير	Qt::MatchStartsWith
البحث عن أيّ عنصر يحتوي في نهايته النص المدخل مع عدم مراعاة الأحرف الكبيرة و الصغير	Qt::MatchEndsWith
البحث عن النص بدقة أيّ يجب أن يكون النص كامل (النص المدخل هو ذاته نص العنصر) مع مراعاة الأحرف الكبيرة و الصغير	Qt::MatchCaseSensitive
البحث عن النص الذي يكون تعبيره القياسيّ هو ذاته التعبير القياسيّ المدخل (النص المدخل) , كالتعبير القياسيّ الخاص بعنوان البريد الإلكتروني	Qt::MatchRegExp
البحث عن النص الذي يتطابق تركيبه مع المحارف البديلة التي في الجدول ,انظر الجدول:	Qt::MatchWildcard

يجب أن يتطابق نص البحث مع بداية النص المدخل	c
كل رمز ? يدل على محرف واحد غير معروف أي نعوض المحرف الذي لا نعلم ماذا يكون(محرف مجهول) بإشارة استفهام	?
يجب أن يتطابق امتداد النص المدخل مع نص البحث كالبحث عن أسماء	*

	الملفات النصية في قائمة ما (*.txt) سيبحث بشكل تلقائي عن النهاية التي كون نصها (.txt)	
	المحارف التي يحتويها النص المدخل يجب أن تكون متطابقة مع المحارف التي تكون داخل الأقواس المربعة , مثال نريد البحث عن النص الذي يحتوي المحارف من المحرف a بشكله الصغير إلى المحرف h , نكتب: [a-h] , و من أجل البحث عن الأرقام نكتب [0-9] :	[...]
Qt::MatchWrap	البحث عن جميع العناصر المطابق نصها نص البحث , إذا كانت المشيرة الخاصة في البحث محدد على العنصر ذو الموقع الرابع فرضا سوف يبدأ البحث من هذا العنصر إلى أن يصل إليه مجددا أي عندما يبدأ البحث من العنصر الرابع ويصل إلى العنصر الأخير سوف يرجع إلى العنصر الأول و يبحث إلى أن يصل إلى العنصر الرابع	
Qt::MatchRecursive	يبحث بتسلسل هرمي مع مروره بكافة العناصر	

رمّاز خاص بالبحث عن عنصر ضمن قائمة يحتوي على النص: Qt

أنشأ مشروع Qt Gui Application ضع الصف الأساسي , QWidget أولا يجب أن نضيف الرمّاز الخاص بإضافة ملفات الترويسة:

```
#include <QList>
#include <QListWidget>
#include <QListWidgetItem>
#include <QLayout>
#include <QIcon>
#include <QPixmap>
```

ثانيا:

اكتب داخل بناء النافذة الرمّاز التالي:

```
QListWidget*  lstW = new QListWidget();
QList<QListWidgetItem*>  list;
QLayout*  layoutW = this->layout();
QListWidgetItem* itemIcon = new QListWidgetItem();
QStringList lstStr;
lstStr << “Q” << “TrollTech” << “QtAr” << “arabQt” << ”Qt” << “Nokia” << “T”;
lstW->addItems(lstStr);
itemIcon.setIcon(QIcon(QPixmap(“:/icons/qt-logo.ico”)));
lstW(itemIcon);
layoutW->addWidget(lstW);
list = lstW->findItems(“Qt”, Qt::MatchContains);
if(!list.empty())
{
```

```
//do anything
If (list.count() > 1)
this.setWindowTitle( list.takeAt(1)->text() );
}
```

المتحول list سيحتوي على جميع العناصر التي تتضمن داخلها النص. Qt

أمّا الإشارات الأحداث التابعة للكائن:QListWidget

```
void currentItemChanged ( QListWidgetItem * current,
QListWidgetItem * previous )
```

يقدح عند تغيير التحديد ,أمّ الوسيطين:

الوسيط الأول current فهو العنصر الذي حدد حاليا.

الوسيط الثاني previous فهو العنصر الذي كان محددا سابقا.

```
void currentRowChanged ( int currentRow )
```

يقدح عند تغيير التحديد عن الصف , الوسيط currentRow فهو رقم الصف المحدد حاليا.

```
void currentTextChanged ( const QString & currentText )
```

يقدح عند تغيير النص للعنصر المحدد , الوسيط currentText النص الجديد للعنصر.

```
void itemActivated ( QListWidgetItem * item )
```

يقدح عند تفعيل العنصر , الوسيط item العنصر الذي تم تفعيله .

```
void itemChanged ( QListWidgetItem * item )
```

يقدح عند تغيير خصائص عنصر , الوسيط item كائن العنصر الذي قد غيّرت خصائصه.

```
void itemClicked ( QListWidgetItem * item )
```

يقدح عند النقر على عنصر ما , الوسيط item العنصر الذي نقر عليه.

```
void itemDoubleClicked ( QListWidgetItem * item )
```

كالحدث السابق لكن يقدح بالنقر المضاعف.

void itemEntered (QListWidgetItem * item)

يقدح عند تحريك مؤشر الفأرة فوق عنصر ما أو الضغط عليه , و الوسيط item هو العنصر الذي حرك مؤشر الفأرة فوقه , يقدح هذا الحدث فقط عند إعطاء قيمة true للخاصيّة mouseTrackingللكائن.QListWidget

void itemPressed (QListWidgetItem * item)

يقدح عند الضغط بزر الفأرة على عنصر من عناصر ,QListWidget و الوسيط Item العنصر الذي ضغط عليه.

void itemSelectionChanged ()

يقدح عند تغيير تحديد العنصر.

يوجد أيضا مقبس يدعى clear خاص بالكائن QListWidget لحذف جميع العناصر.

البنية الهرمية للصف: QListWidget

QObject → QWidget → QFrame →QAbstractScrollArea
→QAbstractItemView → QListView →QListWidget

انتهينا من أهم الخصائص و الأحداث و المقابس التابعة للكائن. QListWidget

Tree Widget (QTreeWidget) كائن لإظهار البيانات على شكل شجريّ , يمكن إدخال البيانات عن طريق تحقيق صف نموذج (Model) أو عن طريق مناهج تابعة للكائن QTreeWidgetأهمها:

void addTopLevelItem (QTreeWidgetItem * item)

لإضافة عنصر شجري.

void addTopLevelItems (const QList<QTreeWidgetItem *> & items)

لإضافة عناصر شجريّة.

int columnCount () const

يرجع عدد الأعمدة.

int currentColumn () const

يرجع رقم العمود المحدد.

currentItem () const QTreeWidgetItem *

يرجع العنصر المحدد.

void editItem (QTreeWidgetItem * item, int column = 0)

تعديل قيمة عنصر شجريّ.

findItems (const QString & text, QList<QTreeWidgetItem *>
Qt::MatchFlags flags, int column = 0) const

بحث عن العناصر المماثلة لقيمة الوسيط, text يعيد العناصر التي قد وجدها على شكل قائمة تحوي عناصر من نمط مؤشر لكائن عنصر شجريّ.

headerItem () const QTreeWidgetItem *

يرجع عنصر رأس القائمة الشجريّة (شريط العنوان)

Int indexOfTopLevelItem (QTreeWidgetItem * item) const

يرجع رقم صف العنصر الممرر الموجود ضمن القائمة الشجرية.

void insertTopLevelItem (int index, QTreeWidgetItem * item)

إدراج عنصر قائمة شجريّة حيث عنوان الصف (رقم الصف) هو. index

void insertTopLevelItems (int index, const QList<QTreeWidgetItem *>
& items)

كالمنهجيّة السابقة لكن هنا يقوم بإدراج عناصر شجريّة.

QTreeWidgetItem * itemAt (const QPoint & p) const

يرجع عنصر قائمة شجريّة ذو الموقع الممرر للوسيط.p&

QTreeWidgetItem * itemAt (int x, int y) const

يرجع عنصر قائمة شجريّة ذو الموقع الممرر على المحور الأفقيّ (x) و على المحور العموديّ.(y)

QWidget * itemWidget (QTreeWidgetItem * item, int column) const

يرجع الكائن widget من العنصر item الموجود ضمن القائمة الشجريّة داخل العمود column .

حيث يبحث في كائن القائمة الشجريّة داخل العمود ذو الرقم قيمة الوسيط الممرر column لمثيل العنصر , item عندما يجده يعيد مؤشر للكائن.* QWidget

void removeItemWidget (QTreeWidgetItem * item, int column)

يحذف عنصر قائمة فقط بتمرير مثيل العنصر item و رقم العمود.

selectedItems () const QList<QTreeWidgetItem *>

يرجع العناصر المحددة.

void setColumnCount (int columns)

لإدراج أعمدة بعدد قيمة الوسيط.columns

void setCurrentItem (QTreeWidgetItem * item)

للتحديد على العنصر المثيل للعنصر.item

void setHeaderItem (QTreeWidgetItem * item)

إعطاء قيمة رأس القائمة الشجريّة(شريط العنوان) من خلال الوسيط. item

void setHeaderLabel (const QString & label)

إعطاء قيمة رأس القائمة الشجريّة (شريط العنوان) من خلال الوسيط.label

void setHeaderLabels (const QStringList & labels)

إعطاء قيمة رأس القائمة الشجريّة (شريط العنوان) من خلال الوسيط , labels تستخدم هذه المنهجيّة عند وجود أكثر من عمود.

void setItemWidget (QTreeWidgetItem * item, int column, QWidget * widget)

لإضافة كائن QWidget كالزر أو أداة إدخال نص الخ ...

void sortItems (int column, Qt::SortOrder order)

لترتيب العناصر داخل العمود.column

takeTopLevelItem (int index) QTreeWidgetItem *

يرجع مؤشر عنصر قائمة شجريّة ذو الفهرس index مع إزالة العنصر من القائمة الشجريّة.

topLevelItem (int index) const QTreeWidgetItem *

يرجع مؤشر عنصر قائمة شجريّة ذو الفهرس index من دون حذف العنصر المرجع.

Int topLevelItemCount () const

يرجع عدد العناصر الموجودة داخل القائمة الشجريّة.

أهم المقابس:(Slots)

void clear();

حذف جميع العناصر.

void expandItem(const QTreeWidgetItem* item)

توسيع عقدة العنصر. item

void collapseItem(const QTreeWidgetItem* item)

ضم عقدة العنصر.item

أهم الأحداث:(Signals)

void currentItemChanged (QTreeWidgetItem * current,
QTreeWidgetItem * previous)

يقدح عند تغيير التحديد عن عنصر ما إلى عنصر آخر , الوسيط current هو العنصر الحالي , أمّا الوسيط previous هو العنصر الذي كان محدد مسبقا.

void itemActivated (QTreeWidgetItem * item, int column)

يقدح عند تفعيل خليّة عنصر قائمة شجريّة ,الوسيط item هو العنصر المفعّل داخله الخليّة أمّا الوسيط column رقم عمود الخليّة المفعّلة.

void itemChanged (QTreeWidgetItem * item, int column)

يقدح عند تغيير قيمة خليّة .

void itemClicked (QTreeWidgetItem * item, int column)

يقدح عند النقر على خليّة.

void itemCollapsed (QTreeWidgetItem * item)

يقدح عند ضمّ عنصر.

void itemDoubleClicked (QTreeWidgetItem * item, int column)

يقدح عند النقر المزدوج على خليّة.

void itemEntered (QTreeWidgetItem * item, int column)

يقدح عند دخول مؤشر الفأرة فوق خليّة , هذا الحدث يعمل فقط عند تفعيل الخاصيّة mouseTracking.

void itemExpanded (QTreeWidgetItem * item)

يقدح عند توسيع عنصر.

void itemPressed (QTreeWidgetItem * item, int column)

يقدح عند الضغط على خليّة.

void itemSelectionChanged ()

يقدح عند تغيير التحديد.

البينية الهرميّة للكائن QTreeWidget :

QObject → QWidget → QFrame →QAbstractScrollArea →QAbstractItemView →QTreeView →QTreeWidget →Header

الآن سوف نكتب رمّاز بسيط لإنشاء قائمة شجريّة QListWidget تحتوي على عمودين الأول اسم ألبوم الموسيقى و الثاني تعليقات , اسم الألبوم يحتوي على عقدة عند توسيعها سوف تظهر أسما الأغاني ,ويظهر في العمود الثاني التعليقات .

ضمّن أولا الملفات الرأسيّة التالية:

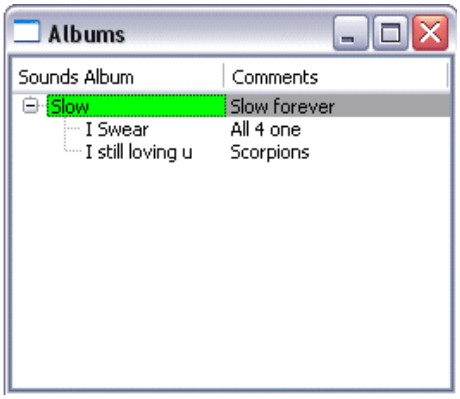
```
#include <QList>

#include <QTreeWidget>
```

```
#include <QTreeWidgetItem>
```

اكتب الرمّاز التالي داخل الكتلة الرئيسيّة للتطبيق:

```
 QTreeWidget* tre= new QTreeWidget () ;
QStringList lstHeader ;
lstHeader << "Sounds Album" << "Comments" ;
tre->setHeaderLabels(lstHeader);
QTreeWidgetItem* snds= new QTreeWidgetItem();
QTreeWidgetItem* chSnds1= new QTreeWidgetItem();
QTreeWidgetItem* chSnds2= new QTreeWidgetItem();
QList<QTreeWidgetItem*> lstSnds;
QList<QTreeWidgetItem*> lstalbm;
snds->setText(0,"Slow");
snds->setText(1,"Slow forever");
chSnds1->setText(0,"I Swear");
chSnds1->setText(1,"All 4 one");
lstSnds.append(chSnds1);
chSnds2->setText(0,"I still loving u");
chSnds2->setText(1,"Scorpions");
lstSnds.append(chSnds2);
snds->addChildren(lstSnds);
lstalbm.append(snds);
snds->setBackgroundColor(0,Qt::green);
snds->setBackgroundColor(1,Qt::gray);
tre->addTopLevelItems(lstalbm);
tre->setWindowTitle("Albums");
tre->expandAll();
tre->show();
```

عندما تنفذه سوف يظهر كما في الشكل: (2.7)



الشكل2.7

ملاحظة **:ستجد داخل القرص المرفق هذا المثال لكن مع بعض الإضافة كإظهار صندوق نص يظهر عند النقر المضاعف بزر الفأرة يحوي نص الخلية التي قد نقر عليها.**

انتهينا الآن من كائن. QTreeWidget

Table Widget (QTableWidget) كائن لإظهار البيانات على شكل جدول يمكن إدخال البيانات عن طريق تحقيق صف نموذج (Model) أو عن طريق مناهج تابعة للكائن QTableWidgetأهمها:

QWidget * cellWidget (int row, int column) const

ترجع مؤشر لكائن *QWidget الموجود داخل الموقع(الصف row و العمودcolumn).

int column (const QTableWidgetItem * item) const

يرجع رقم العمود لذي يحتوي العنصر الممرر.

int columnCount () const

يرجع عدد الأعمدة.

int currentColumn () const

يرجع رقم العمود (الحالي) المحدد.

QTableWidgetItem * currentItem () const

يرجع مؤشر العنصر المحدد.

int currentRow () const

يرجع رقم الصف (الحالي) المحدد.

QList<QTableWidgetItem *> findItems (const QString & text, Qt::MatchFlags flags) const

بحث عن العناصر المماثلة لقيمة الوسيط, text يعيد العناصر التي قد وجدها على شكل قائمة تحوي عناصر من نمط مؤشر لكائن عنصر جدول.

QTableWidgetItem * horizontalHeaderItem (int column) const

يرجع مؤشر من نمط QTableWidgetItem لعنصر العنوان الرأسيّ الأفقي للعمود ذو الموقع column.

QTableWidgetItem * item (int row, int column) const

يرجع مؤشر من نمط QTableWidgetItem للعنصر ذو الموقع (column - row) حيث تقاطع الصف row مع العمود column هو مؤشر كائن العنصر المعاد.

QTableWidgetItem * itemAt (const QPoint & point) const

يرجع مؤشر من نمط QTableWidgetItem للعنصر ذو الموقع (point) العنصر التي تكون النقطة داخل حدوده.

QTableWidgetItem * itemAt (int ax, int ay) const

يرجع مؤشر من نمط QTableWidgetItem للعنصر ذو الموقع(ax - ay) العنصر التي تكون نقطة التقاطع المحور الأفقي ax مع المحور العمودي ay داخل حدوده.

void removeCellWidget (int row, int column)

حذف الكائن QWidget الموجود داخل الخليّة ذو لموقع تقاطع الصف rowمع العمود column.

int row (const QTableWidgetItem * item) const

يرجع رقم الصف الذي يحتوي العنصر الممرر.

int rowCount () const

يرجع عدد الصفوف.

QList<QTableWidgetItem *> selectedItems ()

يرجع قائمة تحوي على العناصر المحددة من نمط مؤشر كائن عنصر جدول (خليّة)

void setCellWidget (int row, int column, QWidget * widget)

لوضع كائن QWidget داخل الخليّة التي يكون موقعها تقاطع الصف row مع العمود column.

void setColumnCount (int columns)

يصبح عدد أعمدة الجدول نفس قيمة الوسيط.column

void setCurrentCell (int row, int column)

للتركيز التحديدي على الخليّة التي يكون موقعها تقاطع.(row-column)

void setHorizontalHeaderItem (int column, QTableWidgetItem * item)

لوضع كائن عنصر جدول للعنوان الرأسي التابع للعمود ذو الموقع.column

void setHorizontalHeaderLabels (const QStringList & labels)

لإعطاء قيم نصيّة للعناوين الرأسيّة الأفقيّة.

void setItem (int row, int column, QTableWidgetItem * item)

لوضع عنصر في الخليّة التي يكون موقعها تقاطع row مع.column

void setRowCount (int rows)

يصبح عدد صفوف الجدول نفس قيمة الوسيط.rows

void setVerticalHeaderItem (int row, QTableWidgetItem * item)

لوضع كائن عنصر جدول للعنوان الرأسي التابع للصف ذو الموقع.row

void setVerticalHeaderLabels (const QStringList & labels)

لإعطاء قيم نصيّة للعناوين الرأسيّة العمودية.

void sortItems (int column, Qt::SortOrder order = Qt::AscendingOrder)

ترتيب كافة العناصر التي تكون داخل العمود ذو الموقع.column

QTableWidgetItem * takeItem (int row, int column)

نسخ العنصر من الخلية التي يكون موقعها تقاطع row مع column ومن ثمّ إزالة العنصر من الجدول.

QTableWidgetItem * verticalHeaderItem (int row) const

يرجع مؤشر كائن عنصر جدول (*QTableWidgetItem) للعنوان الرأسي التابع للصف ذو العنوان.row

أهم المقابس (Slots) للكائن: QTableWidget

void clear ()

حذف العناصر جميعها.

void insertColumn (int column)

إضافة عمود في الموقع.column

void insertRow (int row)

إضافة صف في الموقع.row

void removeColumn (int column)

إزالة عمود ذو الموقع.column

void removeRow (int row)

إزالة صف ذو الموقع.row

أهم الأحداث (Signals) للكائن:QTableWidget

void cellActivated (int row, int column)

يقدح عند تفعيل خليّة.

void cellChanged (int row, int column)

يقدح عند تغير قيمة خليّة.

void cellClicked (int row, int column)

يقدح عند النقر على خليّة.

void cellDoubleClicked (int row, int column)

يقدح عند النقر المضاعف على خليّة.

void cellEntered (int row, int column)

يقدح عند دخول مؤشر الفأرة حدود خليّة , مفعّل فقط بتفعيل الخاصيّة.mouseTracking

void cellPressed (int row, int column)

يقدح عند الضغط على خليّة.

void currentCellChanged (int currentRow, int currentColumn, int previousRow, int previousColumn)

يقدح عند تغيير التحديد من خليّة إلى أخرى.

قد شرحنا هذه الأحداث مسبقا:

void itemActivated (QTableWidgetItem * item)

void itemChanged (QTableWidgetItem * item)

void itemClicked (QTableWidgetItem * item)

void itemDoubleClicked (QTableWidgetItem * item)

void itemEntered (QTableWidgetItem * item)

void itemPressed (QTableWidgetItem * item)

void itemSelectionChanged ()

البنيّة الهرميّة للكائن: QTableWidget

QObject → QWidget → QFrame →QAbstractScrollArea
→QAbstractItemView →QTableView → QTableWidget →Header

رمّاز بسيط لإنشاء جدول يحتوي على بيانات شخصيّة:

أولا أضف ملفات الترويسة التالية:

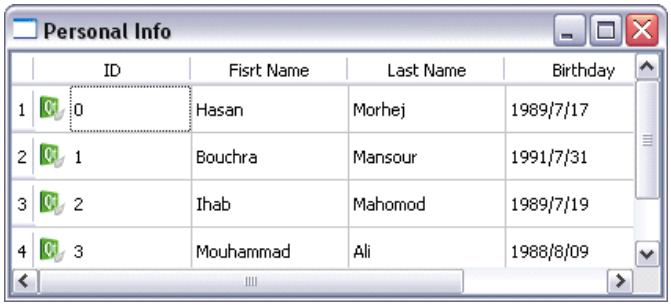
```
#include <QTableWidget>

#include <QTableWidgetItem>

#include <QIcon>

#include <QPixmap>
```

ثمّ اكتب الرمّاز التالي:

```
  QTableWidget  tble;
   tble.setColumnCount(4);
   tble.setRowCount(4);
   QStringList headerlbls;
   headerlbls << "ID" << "First Name" << "Last Name" << "Birthday" ;
   tble.setHorizontalHeaderLabels(headerlbls);
   headerlbls.clear();
   headerlbls << "1" << "2" << "3" << "4" ;
   tble.setVerticalHeaderLabels(headerlbls);

   tble.setItem(0,0,new
QTableWidgetItem(QIcon(QPixmap(":/icons/qt.ico")),"0"));
   tble.setItem(0,1,new QTableWidgetItem("Hasan"));
   tble.setItem(0,2,new QTableWidgetItem("Morhej"));
   tble.setItem(0,3,new QTableWidgetItem("1989/7/17"));

   tble.setItem(1,0,new
QTableWidgetItem(QIcon(QPixmap(":/icons/qt.ico")),"1"));
   tble.setItem(1,1,new QTableWidgetItem("Bouchra"));
   tble.setItem(1,2,new QTableWidgetItem("Mansour"));
   tble.setItem(1,3,new QTableWidgetItem("1991/7/31"));

   tble.setItem(2,0,new
QTableWidgetItem(QIcon(QPixmap(":/icons/qt.ico")),"2"));
   tble.setItem(2,1,new QTableWidgetItem("Ihab"));
```

```
    tble.setItem(2,2,new QTableWidgetItem("Mahomod"));
    tble.setItem(2,3,new QTableWidgetItem("1989/7/19"));

    tble.setItem(3,0,new
QTableWidgetItem(QIcon(QPixmap(":/icons/qt.ico")),"3"));
    tble.setItem(3,1,new QTableWidgetItem("Mouhammad"));
    tble.setItem(3,2,new QTableWidgetItem("Ali"));
    tble.setItem(3,3,new QTableWidgetItem("1988/8/09"));
    tble.setWindowTitle("Personal Info");
    tble.show();
```

عند تنقيّ التطبيق سيظهر كما في الشكل:(2.8)



الشكل2.8

ملاحظة **:ستجد داخل القرص المرفق هذا المثال باسم (tableWidget) مع بعض الإضافة.**

انتهينا الآن من كائن. QTableWidget

- **الأدوات التابعة لتبويب الكائنات الحاوية(Containers):**

Group Box (QGroupBox) كائن صندوق يحوي مجموعةأدوات بتنسيق معين .

أهم خصائصه:

```
void setTitle(const QString &title);
```

لوضع عنوان للصندوق.

void setAlignment(Qt::Alignment);

أعلام الوسيطQt::Alignment , التعداد **Qt::AlignmentFlag** :

إزاحة العنوان إلى اليسار	**Qt::AlignLeft**
إزاحة العنوان إلى اليمين	**Qt::AlignRight**
إزاحة العنوان إلى الوسط	**Qt::AlignHCenter**

بنيته الهرميّة:

QObject →QWidget →QGroupBox

Scroll Area (QScrollArea) كائن صندوق يحتوي كائنات ويظهر منزلقة في حال كان حجم الكائن المضمن أكبر من الحاوي Scroll Area.

رمّاز بسيط لإضافة لافتة نص QLabel داخل أداة , QScrollArea عرض QLabel أكبر مقاسا من عرض QScrollArea لنرى كيف تظهر المنزلقة الأفقيّة:

```
   QScrollArea* scrollA = new QScrollArea(this);
   QLabel* lbl= new QLabel("A B C D E F G H I J K L M N O P Q R S T U V
W X Y Z");
   scrollA->setWidget(lbl);
scrollA->setGeometry(0,0,200,50);
scrollA->setBackgroundRole(QPalette::Light);
```

البنية الهرمية للكائن QScrollArea :

QObject → QWidget → QFrame →QAbstractScrollArea →QScrollArea

Tool Box (QToolBox) كائن صندوق أدوات , أهم المنهجيّات:

int addItem (QWidget * widget, const QIcon & iconSet, const QString & text)

إضافة تبويب جديد إلى أسفل الكائن QToolBox يحوي الكائن widget و يكون شعار التبويب iconSetو نص التبويب.text

int addItem (QWidget * w, const QString & text)

إضافة تبويب جديد إلى أسفل الكائن QToolBox يحوي الكائن widget و نص التبويب.text

int count () const

يرجع عدد التبويبات.

int currentIndex () const

يرجع فهرس رقم التبويب المحدد.

QWidget * currentWidget () const

يرجع مؤشر للكائن المحدد أو صفر إذا لم يكن أي كائن محدد.

int indexOf (QWidget * widget) const

يرجع فهرس مثيل الكائن الممرر للوسيط , widget* إذا لم يجد مثيل له يعيد القيمة. -1

int insertItem (int index, QWidget * widget, const QIcon & icon, const QString & text)

إضافة تبويب جديد إلى الموقع Index للكائن QToolBox يحوي الكائن widget و يكون شعار التبويب iconSet و نص التبويب.text

int insertItem (int index, QWidget * widget, const QString & text)

إضافة تبويب جديد إلى الموقع Index للكائن QToolBox يحوي الكائن widget و نص التبويب.text

bool isItemEnabled (int index) const

يرجع القيمة true إذا كان الموقع الممرر للتبويب التابع للكائن QToolBox مفعّل.

QIcon itemIcon (int index) const

يرجع شعار التبويب ذو العنوان.index

QString itemText (int index) const

يرجع نص التبويب ذو العنوان.index

QString itemToolTip (int index) const

يرجع نص تلميح التبويب ذو العنوان.index

void removeItem (int index)

إزالة التبويب ذو العنوان index .

void setItemEnabled (int index, bool enabled)

تفعيل أو عدم تفعيل التبويب ذو العنوان Index .

void setItemIcon (int index, const QIcon & icon)

وضع شعار للتبويب ذو العنوان Index .

void setItemText (int index, const QString & text)

وضع نص للتبويب ذو العنوان index .

void setItemToolTip (int index, const QString & toolTip)

وضع نص تلميح للتبويب ذو العنوان index .

QWidget * widget (int index) const

يرجع مؤشر للكائن (التبويب) ذو العنوان index .

الحدث currentChange(int index) يقدح عند تغيير التركيز عن التبويب .

رمّاز بسيط لإنشاء كان QToolBox يحتوي على تبويبان يحتوي كل واحد منهما على زر و لافتة نص.

```
 QToolBox* toolbx= new QToolBox(this) ;
QVBoxLayout* layoutTab1= new QVBoxLayout();
layoutTab1->addWidget(new QLabel("Label-Test-1"));
layoutTab1->addWidget(new QPushButton("Btn-Test-1"));
toolbx->addItem(new QLabel(""),"Tab1");
toolbx->widget(0)->setLayout(layoutTab1);
toolbx->setGeometry(0,0,100,200);
QVBoxLayout* layoutTab2= new QVBoxLayout();
layoutTab2->addWidget(new QLabel("Label-Test-2"));
layoutTab2->addWidget(new QPushButton("Btn-Test-2"));
toolbx->addItem(new QLabel(""),"Tab2");
toolbx->widget(1)->setLayout(layoutTab2);
```

البنية الهرمية للكائن QToolBox :

QObject → QWidget → QFrame → QToolBox

Tab Widget (QTabWidget) كائن متعدد الصفحات , كل صفحة تحتوي على مجموعة من الكائنات , تحدد الصفحة الذي تريد من خلال التبويب الظاهر على طرفها ,أهم مناهجه:

int addTab (QWidget * page, const QString & label)

إضافة صفحة جديدة بعد آخر تبويب موجود ,يكون نص تبويبها , label يرجع هذا المنهج رقم التبويب) موقع الصفحة.(

int addTab (QWidget * page, const QIcon & icon, const QString & label)

إضافة صفحة جديدة بعد آخر تبويب موجود ,يكون نص تبويبها, label و شعار تبويبها ,icon يرجع هذا المنهج رقم التبويب .

void clear ()

يزيل جميع الصفحات الموجودة.

int count () const

عدد صفحات الكائن QTabWidget .

int currentIndex () const

يرجع موقع الصفحة المحددة.

bool isTabEnabled (int index) const

يرجع true إذا كان التبويب ذو الموقع Index مفعّل.

void removeTab (int index)

إزالة العنصر ذو العنوان index .

void setTabIcon (int index, const QIcon & icon)

وضع شعار للتبويب ذو العنوان index .

void setTabPosition (TabPosition)

لتغيير توجه (توضع) التبويبات , الوسيط TabPosition , التعداد QTabWidget::TabPosition :

QTabWidget::North	تتوضع التبويبات من الجهة العليا
QTabWidget::South	تتوضع التبويبات من الجهة السفلى
QTabWidget::West	تتوضع التبويبات من الجهة اليسرى
QTabWidget::East	تتوضع التبويبات من الجهة اليمنى

void setTabShape (TabShape s)

لتغيير شكل التبويبات , الوسيط TabShape , التعداد QTabWidget::TabShape :

QTabWidget::Rounded	إظهار التبويبات بشكل مصقول
QTabWidget::Triangular	إظهار التبويبات بشكل مثلثيّ

void setTabText (int index, const QString & label)

لوضع نص للتبويب ذو العنوان index .

void setTabToolTip (int index, const QString & tip)

لوضع نص تلميح للتبويب ذو العنوان index .

void setTabsClosable (bool closeable)

تفعيل أو عدم تفعيل زر الإغلاق للتبويبات.

void setUsesScrollButtons (bool useButtons)

تفعيل أو عدم تفعيل أزرار المنزلقة.

أهم الأحداث:(Signals)

void currentChanged(int index);

يقدح عند تغيير التركيز عن تبويب.

void tabCloseRequest(int index);

يقدح عند نقر زر إغلاق التبويب.

رمّاز بسيط لإنشاء كائن , QTabWidget يحتوي على تبويبان يوجد داخلهما لافتات و كائنات إدخال نصوص:

ضمّن ملفات الترويسة التالية:

```
#include <QTabWidget>
#include <QFrame>
#include <QLabel>
#include <QPushButton>
#include <QVBoxLayout>
#include <QLineEdit>
```

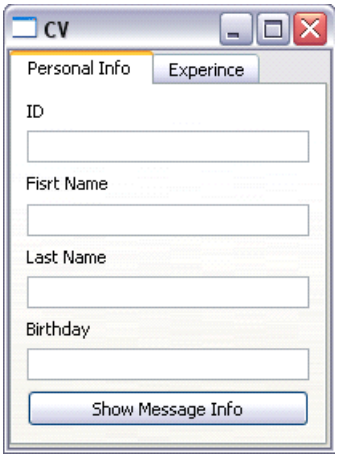
الآن أضف بالكتلة الرئيسية الرمّاز التالي:

```
 QTabWidget* tabWidget= new QTabWidget();
QFrame* pg1= new QFrame();
QFrame* pg2= new QFrame();
QVBoxLayout* layPg1= new QVBoxLayout();
QVBoxLayout* layPg2= new QVBoxLayout();
pg1->setLayout(layPg1);
pg2->setLayout(layPg2);
layPg1->addWidget(new QLabel("ID"));
layPg1->addWidget(new QLineEdit(""));
layPg1->addWidget(new QLabel("Fisrt Name"));
layPg1->addWidget(new QLineEdit(""));
layPg1->addWidget(new QLabel("Last Name"));
layPg1->addWidget(new QLineEdit(""));
layPg1->addWidget(new QLabel("Birthday"));
layPg1->addWidget(new QLineEdit(""));
layPg1->addWidget(new QPushButton("Show Message Info"));
tabWidget->addTab(pg1,"Personal Info");
layPg2->addWidget(new QLabel("ID"));
layPg2->addWidget(new QLineEdit(""));
layPg2->addWidget(new QLabel("Fisrt Year"));
layPg2->addWidget(new QLineEdit(""));
layPg2->addWidget(new QLabel("Last Year"));
```

```
layPg2->addWidget(new QLineEdit(""));
layPg2->addWidget(new QLabel("Company Name"));
layPg2->addWidget(new QLineEdit(""));
layPg2->addWidget(new QPushButton("Show Message Info"));
tabWidget->addTab(pg2,"Experince");
tabWidget->setWindowTitle("CV");
tabWidget->show();
```

في هذا الرمّاز استخدمنا لإضافة الأدوات على الصفحة الكائن QFrame و الكائن QVBoxLayout , نستطيع إضافة الأدوات بواسطة استخدام الكائن ,QWidget سنرى كيف في المثال التالي الخاص بالأداة.QStackedWidget

عند تنفيذ التطبيق سيظهر كما في الشكل:(2.9)



الشكل2.9

البنية الهرمية للكائن QTabWidget :

QObject → QWidget → QTabWidget

Stacked Widget (QStackedWidget) كائن تكدس الأدوات , يوضع مجموعة من الأدوات widgets ضمن إطارات(صفحات) ,حيث تكون الإطارات مخفيّة , وعند إظهار إطار سيظهر مع محتوياته

أهم مناهجه:

int addWidget (QWidget * widget)

إضافة كائن widget بعد آخر موقع متاح ,يرجع موقع الكائن المضاف.

int count () const

يرجع عدد الكائنات .

int currentIndex () const

يرجع موقع الكائن المحدد حاليا.

QWidget * currentWidget () const

يرجع مؤشر الكائن المحدد حاليا.

int indexOf (QWidget * widget) const

يرجع موقع الكائن المثيل للوسيط. *widget

int insertWidget (int index, QWidget * widget)

إدراج كائن widget عند العنوان index .

void removeWidget (QWidget * widget)

إزالة كائن widget المثيل للوسيط. *widget

QWidget * widget (int index) const

يرجع مؤشر كائن widget ذو العنوان index

له حدثان: (Signals)

void currentChanged(int index);

void widgetRemoved (int index);

رمّاز بسيط لإنشاء كائن QStackedWidget يحوي إطارين:

```
stW= new QStackedWidget () ;
 QWidget* pi= new QWidget();
```

```
QWidget* exp= new QWidget();
QVBoxLayout* layPi= new QVBoxLayout();
QVBoxLayout* layexp= new QVBoxLayout();
layPi->addWidget(new QLabel("ID"));
layPi->addWidget(new QLineEdit(""));
layPi->addWidget(new QLabel("Fisrt Name"));
layPi->addWidget(new QLineEdit(""));
layPi->addWidget(new QLabel("Last Name"));
layPi->addWidget(new QLineEdit(""));
layPi->addWidget(new QPushButton("Show Message Info"));
layexp->addWidget(new QLabel("Fisrt Year"));
layexp->addWidget(new QLineEdit(""));
layexp->addWidget(new QLabel("Last Year"));
layexp->addWidget(new QLineEdit(""));
layexp->addWidget(new QLabel("Company Name"));
layexp->addWidget(new QLineEdit(""));
layexp->addWidget(new QPushButton("Show Message Info"));
pi->setLayout(layPi);
exp->setLayout(layexp);
stW->addWidget(pi);
stW->addWidget(exp);
stW->setWindowTitle("CV");
stW->setCurrentIndex(0);
QPushButton* nxt= new QPushButton("Next");
QPushButton* prev= new QPushButton("Prev");
layPi->addWidget(nxt);
layexp->addWidget(prev);
connect(nxt,SIGNAL(clicked()),this,SLOT(nxtPg()));
connect(prev,SIGNAL(clicked()),this,SLOT(prevPg()));
stW->show();
```

عرّف المقابس التالية في قسم التصريحات الخاص بها:

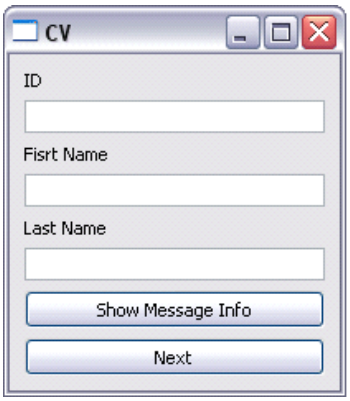
```
private slots:
   void nxtPg();
   void prevPg();
```

حقق هذه المقابس:

```
void MainWindow::nxtPg(){
```

```
stW->setCurrentIndex(1);
}
void MainWindow::prevPg(){
stW->setCurrentIndex(0);
}
```

عند تنفيذ التطبيق سيظهر كما في الشكل:(2.10)



الشكل2.10

اضغط على زر Next ستجده قد انتقل إلى الإطار التالي (الصفحة ذو الرقم1).

ملاحظة:ستجد داخل القرص المرفق المثال كاملا.

البنية الهرمية للكائن QStackedWidget :

QObject → QWidget → QFrame → QStackedWidget

Frame (QFrame) كائن الإطار, يستخدم لاحتواء مجموعة من الأدوات. widgets

البنية الهرميّة للكائن:QFrame

QObject → QWidget → QFrame

Widget (QWidget) الكائن الأساسيّ(الأب)لجميع كائنات واجهة المستخدم , يرث الصف QObject و الصف.QPaintDevice

MdiArea (QMdiArea) كائن منطقة (Multiple Documents Interface Area) الواجهة متعددة المستندات

مخصص ليتيح عرض أكثر من نافذة- مستند- داخله

ضمن مساحة الكائن QMdiArea بحيث نستطيع تحرير المستند , أهم مناهجه:

QMdiSubWindow * activeSubWindow () const

يرجع مؤشر كائن النافذة الابنة الفرعية النشطة حاليا.

QMdiSubWindow * addSubWindow (QWidget * widget,
Qt::WindowFlags windowFlags = 0)

إضافة نافذة فرعيّة , يرجع مؤشر الكائن المضاف إيّ النافذة الفرعيّة المضافة .

QBrush background () const

يرجع كائن الفرشاة لخلفية.QMdiArea

QMdiSubWindow * currentSubWindow () const

يرجع مؤشر كائن النافذة الفرعية النشطة.

void removeSubWindow (QWidget * widget)

إزالة المستند من الكائن QMdiArea الممرر للوسيط.widget*

void setBackground (const QBrush & background)

لوضع قيمة الفرشاة للخلفيّة.

void setTabPosition (QTabWidget::TabPosition position)

لتغيير توجه (توضع) التبويبات , الوسيط TabPosition , التعداد
QTabWidget::TabPosition :

QTabWidget::North	تتوضع التبويبات من الجهة العليا
QTabWidget::South	تتوضع التبويبات من الجهة السفلى
QTabWidget::West	تتوضع التبويبات من الجهة اليسرى
QTabWidget::East	تتوضع التبويبات من الجهة اليمنى

void setTabShape (QTabWidget::TabShape shape)

لتغيير شكل التبويبات , الوسيط TabShape , التعداد QTabWidget::TabShape :

QTabWidget::Rounded	إظهار التبويبات بشكل مصقول
QTabWidget::Triangular	إظهار التبويبات بشكل مثلثيّ

void setViewMode (ViewMode mode)

لإظهار النوافذ الأبناء(المستندات الفرعيّة) إمّا ضمن تبويبات أو نوافذ فرعية دون احتوائها ضمن تبويبات ,الوسيطViewMode , التعداد QMdiArea::ViewMode :

لإظهار المستندات الفرعيّة ضمن تبويبات	**QMdiArea::TabbedView**
لإظهار المستندات الفرعيّة دون احتوائها ضمن تبويبات	**QMdiArea::SubWindowView**

QList<QMdiSubWindow *> subWindowList (WindowOrder order = CreationOrder) const

يرجع قائمة تحتوي على مؤشرات النوافذ الفرعية المحتواة ضمن الكائن.QMdiArea

QTabWidget::TabPosition tabPosition () const

يرجع توجه التبويبات.

QTabWidget::TabShape tabShape () const

يرجع شكل التبويب.

ViewMode viewMode () const

يرجع نمط إظهار النوافذ الفرعيّة.

أحداث الكائن QMdiArea :

الحدث:

void subWindowActivated (QMdiSubWindow * window)

يقدح عند تنشيط نافذة فرعية والوسيط window * هو النافذة التي نشطت حاليا.

أهم المقابس للكائن:QMdiArea

void activateNextSubWindow ()

لتعيينالتركيزعلى النافذة الفرعيّة التالية.

void activatePreviousSubWindow ()

لتعيين التركيز على النافذة الفرعيّة السابقة.

void cascadeSubWindows ()

لترتيب توضع النوافذ الفرعيّة بشكل متدرج.

void tileSubWindows ()

لترتيب توضع النوافذ الفرعيّة بشكل ممدد بحيث تكون جميع النوافذ الأبناء مقسمة على كامل مساحة الكائن.QMdiArea

void closeActiveSubWindow ()

إغلاق النافذة الفرعيّة النشطة.

void closeAllSubWindows ()

إغلاق جميع النوافذ الفرعيّة.

void setActiveSubWindow (QMdiSubWindow * window)

تنشيط النافذة الفرعيّة.window*

البنية الهرمية للكائن QMdiArea :

QObject → QWidget → QFrame →QAbstractScrollArea → QMdiArea

سنضع في نهاية الفصل مثال عن استخدم الواجهة متعددة المستندات QMdiArea ستجده أيضا داخل القرص المرفق.

Dock Widget (QDockWidget) : كائن لرصف الكائنات , صندوق تستطيع إضافة كائنات بداخله , ورصف الصندوق على الجهة اليمنى , السفلى , العليا , أو اليسارية للنافذة , و يمكن أن يكون كائن قائم بذاته ليس داخل النافذة أو أن يكون داخل النافذة , أهم مناهجه:

Qt::DockWidgetAreas allowedAreas () const

تعيد الجهات المسموح للكائن QDockWidget رصفها عليه.

```
If ( dockWidget->allowedAreas() ==
```

(Qt::LeftDockWidgetArea | Qt::TopDockWidgetArea))

// if allow areas left and top do anything u want …

التعدادQt::DockWidgetArea :

Qt::LeftDockWidgetArea	يسمح الرصف للجهة اليساريّة
Qt::RightDockWidgetArea	يسمح الرصف للجهة اليمينيّة
Qt::TopDockWidgetArea	يسمح الرصف للجهة العليا
Qt::BottonDockWidgetArea	يسمح الرصف للجهة السفلى
Qt::AllDockWidgetAreas	يسمح الرصف للجهات كلها
Qt::NoDockWidgetArea	الرصف على داخل النافذة غير مسموح

DockWidgetFeatures features () const

يرجع الميزات المنشطة للكائن.QDockWidget

التعدادQDockWidget::DockWidgetFeature :

QDockWidget::DockWidgetClosable	تفعيل زر الإغلاق
QDockWidget::DockWidgetMovable	السماح بتحريك الكائن QDockWidget
QDockWidget::DockWidgetFloatable	فصل الكائن عن النافذة بحيث يصبح خارج النافذة
QDockWidget::DockWidgetVerticalTitleBar	شريط عنوان الكائن QDockWidgetيتوضع دائما على الجهة اليسرى
QDockWidget::AllDockWidgetFeatures	تفعيل الأربع ميزات السابقات
QDockWidget::NoDockWidgetFeatures	عكس الميزة السابقة ,تعطيل كل الميزات المتاحة للكائن

bool isAreaAllowed (Qt::DockWidgetArea area) const

ترجع true إذا كان الجهة الممررة للوسيط area مسموح للكائن QDockWidget الرصف عليها وإلا ترجع. false

bool isFloating () const

ترجع true إذا كان الكائنQDockWidget عائم فوق النافذة أيّ ليس محتوى ضمن النافذة و إلا يرجع.false

void setAllowedAreas (Qt::DockWidgetAreas areas)

لتضع الجهات المسموح الرصف عليها.

void setFeatures (DockWidgetFeatures features)

لتضع الميزات التي تريدها للكائن.QDockWidget

void setFloating (bool floating)

إذا مررت القيمة true للوسيط يصبح الكائن في حال العوم فوق النافذة , و إذا مررت false يصبح الكائن محتوى ضمن النافذة.

void setTitleBarWidget (QWidget * widget)

لتضع كائن widget لشريط العنوان للكائن QDockWidget بحيث يستحوذ الكائن widget على شريط العنوان.

void setWidget (QWidget * widget)

لوضع كائن widget داخل الكائن.QDockWidget

QWidget * titleBarWidget () const

يرجع مؤشر كائن المستحوذ على شريط العنوان للكائن.QDockWidget

QWidget * widget () const

يرجع مؤشر الكائن المحتوى ضمن الكائن.QDockWidget

الأحداث (Signals) للكائن QDockWidget :

void allowedAreasChanged (Qt::DockWidgetAreas allowedAreas)

يقدح عند تغيير الجهات المسموح الرصف عليها ,الوسيط allowedAreas الجهات المسموح الرصف عليها بعد التعديل.

void dockLocationChanged (Qt::DockWidgetArea area)

يقدح عند تغيير جهة رصف الكائن QDockWidget الوسيط area هو جهة الرصف الحاليّة.

void featuresChanged (QDockWidget::DockWidgetFeatures features)

يقدح عند تغيير الميزات للكائن ,QDockWidget الوسيط features الميزات المفعّلة حاليا بعد التعديل.

void topLevelChanged (bool topLevel)

يقدح عند تغيير حالة خاصيّة العوم ,عندما يكون الكائن في حالة العوم يصبح قيمة الوسيط topLevelمساوية.true

void visibilityChanged (bool visible)

يقدح عند إظهار الكائن أو إخفاءه , عند الإظهار الوسيط visible تصبح قيمته true و عند الإخفاء تصبح.false

البنية الهرميّة للكائن:QDockWidget

QObject → QWidget →QDockWidget

- الأدوات التابعة لتبويب كائنات الإدخال (Input Widgets):

Combo Box (QComboBox) كائن زر مع قائمة منسدلة, تنسدل القائمة بأسفل الكائن عند النقر أو الضغط عليه أو استدعاء المنهجية ,(QComboBox::showPopup أهم مناهجه:

void addItem (const QString & text, const QVariant & userData = QVariant())

لإضافة عنصر , الوسيط userData لإضافة بيانات اختياريّة دون إظهارها في الكائن QComboBoxلكنها سوف تكون مرافقة للعنصر المضاف.

void addItem (const QIcon & icon, const QString & text, const QVariant & userData = QVariant())

لإضافة عنصر مع شعار بجانبه ,الوسيط userData لإضافة بيانات اختياريّة دون إظهارها في الكائن QComboBox لكنها سوف تكون مرافقة للعنصر المضاف.

void addItems (const QStringList & texts)

لإضافة عناصر نصيّة.

QCompleter * completer () const

يرجع مؤشر كائن المتمم ,QCompleter سوق نتكلم عنه لاحقا في الفصل الحالي.

int count () const

يرجع عدد العناصر الموجودة ضمن الكائن.QComboBox

int currentIndex () const

يرجع موقع العنصر الحالي.

QString currentText () const

يرجع نص العنصر الحالي.

int findText (const QString & text, Qt::MatchFlags flags = static_cast<Qt::MatchFlags> (Qt::MatchExactly | Qt::MatchCaseSensitive)) const

يرجع موقع العنصر الذي قد وجده ضمن الكائن QComboBox و إلا يرجع.1-

bool hasFrame () const

يرجع true إذا كان الكائن ضمن إطار و إلا.false

virtual void hidePopup ()

يخفي القائمة المنسدلة إذا كانت ظاهرة.

QSize iconSize () const

يرجع حجم شعار العناصر.

void insertItem (int index, const QString & text, const QVariant & userData = QVariant())

لإضافة عنصر في الموقع ,index الوسيط userData لإضافة بيانات اختياريّة دون إظهارها في الكائن QComboBox لكنها سوف تكون مرافقة للعنصر المضاف.

void insertItem (int index, const QIcon & icon, const QString & text, const QVariant & userData = QVariant())

لإضافة عنصر مع شعار بجانبه في الموقع ,index الوسيط userData لإضافة بيانات اختياريّة دون إظهارها في الكائن QComboBox لكنها سوف تكون مرافقة للعنصر المضاف.

void insertItems (int index, const QStringList & list)

لإضافة عناصر نصيّة , تضاف هذه العناصر من الموقع.index

bool isEditable () const

يرجع true إذا كان الكائن في حال التحرير.

QIcon itemIcon (int index) const

يرجع شعار العنصر ذو الموقع.index

QString itemText (int index) const

يرجع نص العنصر ذو الموقع.index

QLineEdit * lineEdit () const

يرجع مؤشر الكائن QLineEdit الموجود بجانب زر انسدال القائمة , هذه المنهجيّة مفعّلة في حال كانت الخاصيّة editable مساوية.true

int maxCount () const

يرجع أكبر قيمة لعدد العناصر التي يستطيع احتوائها الكائن.QComboBox

int maxVisibleItems () const

يرجع أكبر قيمة لعدد العناصر التي يستطيع إظهارها الكائن QComboBox من دون إظهار المنزلقة.

void removeItem (int index)

إزالة عنصر من قائمة الكائن QComboBox ذو الموقع.index

void setCompleter (QCompleter * completer)

لوضع متمم للكائن.QComboBox

void setEditable (bool editable)

إذا كانت قيمة الوسيط true سوف تفعّل خاصيّة التحرير أيّ يسمح بالكتابة ضمن الجزء العلوي للكائن QComboBox و إذا كانت false سوف يلغى تفعيل هذه الخاصيّة.

void setFrame (bool)

إذا كان الوسيط true سوف يظهر على حواف الكائن إطار و إذا كان false سوف يلغى إظهار الإطار.

void setIconSize (const QSize & size)

وضع حجم شعار العنصر.

void setItemIcon (int index, const QIcon & icon)

إضافة شعار للعنصر ذو الموقع.index

void setItemText (int index, const QString & text)

وضع نص للعنصر ذو الموقع.index

void setLineEdit (QLineEdit * edit)

وضع أداة إدخال نص اختيارية للكائن.QComboBox

void setMaxCount (int max)

وضع أكبر قيمة لعدد العناصر التي يستطيع احتوائها الكائن.QComboBox

void setMaxVisibleItems (int maxItems)

وضع أكبر قيمة لعدد العناصر التي يستطيع إظهارها الكائن QComboBox من دون إظهار المنزلقة.

void setValidator (const QValidator * validator)

وضع كائن التحقق من شرعية النص المدخل و تقييده.QValidator

virtual void showPopup ()

إظهار القائمة المنسدلة للكائن.QComboBox

const QValidator * validator () const

إرجاع مؤشر كائن.QValidator

مقابس (Slots) الكائن:QComboBox

void clear ()

إزالة جميع العناصر من الكائن.QComboBox

void clearEditText ()

مسح نص العنصر الذي يكون في حالة التحرير.

void setCurrentIndex (int index)

وضع التركيز على العنصر ذو الموقع.index

void setEditText (const QString & text)

وضع قيمة النص text للعنصر الذي يكون في حالة التحرير.

أهم الأحداث (Signals) للكائن:QComboBox

void activated (int index)

يقدح عند اختيار عنصر ما , و الوسيط index هو موقع العنصر.

void activated (const QString & text)

يقدح عند اختيار عنصر ما , و الوسيط text هو نص العنصر.

void currentIndexChanged (int index)

يقدح عند تغيير التركيز عن عنصر ما , و الوسيط index هو موقع العنصر الحالي.

void currentIndexChanged (const QString & text)

يقدح عند تغيير التركيز عن عنصر ما , و الوسيط text هو نص العنصر الحالي.

void editTextChanged (const QString & text)

يقدح عند تغيير نص العنصر الذي يكون في حالة تحرير ,الوسيط text و نص العنصر المحرر.

void highlighted (int index)

يقدح عند التحديد على عنصر ما من القائمة المنسدلة دون إختياره , الوسيط index موقع العنصر المحدد.

void highlighted (const QString & text)

يقدح عند التحديد على عنصر ما من القائمة المنسدلة دون إختياره . الوسيط text نص العنصر المحدد.

البنية الهرمية للكائن:QComboBox

QObject → QWidget → QComboBox

Font Combo Box (QFontComboBox) : كائن QComboBox لكنه يحتوي على جميع أنواع الخطوط الموجودة ضمن النظام الذي يحمل التطبيق.

أهم مناهجه:

QFont currentFont () const

يرجع نوع الخط المركز عليه حاليا.

أهم مقابسه:(Slots)

void setCurrentFont (const QFont & font)

ليصبح التركيز على الخط ذو النوع.font

أهم الأحدث:(Signals)

void currentFontChanged (const QFont & font)

يقدح عند تغيير التركيز عن نوع الخط الحالي.

البنية الهرمية للكائن:QFontComboBox

QObject → QWidget → QComboBox → QFontComboBox

Line Edit (QLineEdit) كائن إدخال نص سطري ,أهم مناهجه:

Qt::Alignment alignment () const

يرجع نوع إزاحة النص.

void backspace ()

منهجية تستدعي الضغط على مفتاح Backspace أيّ يحذف المحرف الموجود خلف موقع المشيرة ,و يحذف النص المحدد في حال تحديد أكثر من محرف .

QCompleter * completer () const

يرجع مؤشر لكائن المتمم QCompleter الخاص بالكائن ,QLineEdit سنتكلم بالتفصيل عن الكائن QCompleter في الصفحات القادمة من هذا الفصل.

QMenu * createStandardContextMenu ()

يرجع مؤشر كائن قائمة , نستطيع بواسطته إضافة قائمة منبثقة خاصة بالكائن QLineEdit تظهر عند النقر بزر الفأرة الأيمن على الكائن.QLineEdit

void cursorBackward (bool mark, int steps = 1)

تحديد المحارف التي تكون خلف موقع المشيرة بحيث تكون عددها steps و قيمة mark مساويةtrue

و إلا إذا كانت قيمة mark تساوي false سيلغي تحديد المحارف.

void cursorForward (bool mark, int steps = 1)

تحديد المحارف التي تكون أمام موقع المشيرة بحيث تكون عددها steps و قيمة mark مساويةtrue

و إلا إذا كانت قيمة mark تساوي false سيلغي تحديد المحارف.

int cursorPosition () const

يرجع موقع المشيرة الحالي.

int cursorPositionAt (const QPoint & pos)

يضع موقع المشيرة في النقطة.pos

void cursorWordBackward (bool mark)

التحديد من موقع المشيرة الحالي إلى بداية الكلمة إذا كانت قيمة mark تساوي true و إلا إذا كان false إلغاء التحديد.

void cursorWordForward (bool mark)

التحديد من موقع المشيرة الحالي إلى نهاية الكلمة إذا كانت قيمة mark تساوي true و إلا إذا كان false إلغاء التحديد.

void del ()

منهجية تستدعي الضغط على مفتاح Delete أيّ تحذف المحرف الذي يكون أمام موقع المشيرة الحالي , و إذا كان النص محدد يحذفه.

void deselect ()

إلغاء التحديد.

QString displayText () const

يرجع قيمة نصيّة بالنص الظاهر كما هو أيّ إذا كانت الخاصيّة echoMode قيمتها Passwordأو PasswordEchoOnEdit يرجع النص قناع كلمة السر و طول النص المعاد عدد المحارف الموجودة داخل كائن QLineEdit و إذا كانت قيمته NoEcho يعيد نص فارغ و إذا كانت قيمته Normal يعيد النص كما هو أي كأنك استدعيت المنهجيّة.text

bool dragEnabled () const

يرجع true إذا كان السحب قد نشّط.

EchoMode echoMode () const

يرجع نمط إظهار المحارف , وهذا جدول يحتوي على الأنماط المتاحة و كل عمل كل واحد منهم,التعداد QLineEdit::EchoMode :

يسمح للمستخدم بإدخال محارف و إظهارها بشكلها الإفتراضي	QLineEdit::Normal
يسمح للمستخدم بإدخال محارف لكن دون إظهارها	QLineEdit::NoEcho
يسمح للمستخدم بإدخال محارف و إظهارها على شكل نجم أو دوائر أيّ على شكل قناع محارف كلمة المرور (قناع كلمة مرور حسب	QLineEdit::Password

النظام الذي يعمل عليه التطبيق)	
مثل عمل السابق لكن فقط عند إدخال المحارف تظهر كما هي	QLineEdit::PasswordEchoOnEdit

void end (bool mark)

إذا كانت قيمة الوسيط mark تساوي true سوف تنتقل المشيرة من موقعها الحالي إلى نهاية النص مع تحديد المحارف التي تقع بين الموقع الحالي للمشيرة و نهاية النص و إلا إذا كانت قيمة الوسيط mark تساوي false سوف تنتقل المشيرة إلى نهاية النص لكن دون تحديد المحارف.

void getTextMargins (int * left, int * top, int * right, int * bottom) const

يرجع لكل وسيط ممرر قيمة الهامش:

الهامش الأيسر	* left
الهامش العلوي	* top
الهامش الأيمن	* right
الهامش السفلي	* bottom

bool hasFrame () const

يرجع true إذا كان الإطار ظاهر على حواف الكائن.QLineEdit

bool hasSelectedText () const

يرجع true إذا كان جزء أو كامل النص قد حدّد و إلا يرجع.false

void home (bool mark)

نفس عمل المنهج end لكن يكون انتقال المشيرة إلى أوّل النص.

void setInputMask (const QString & inputMask)

يضع نص قناع الإدخال , قناع الإدخال يتكون من إحدى المحارف الموجودة في الجدول التالي ,كل محرف له وظيفة معيّة , وستصبح قيمة الخاصيّة maxLength عدد المحارف الموجودة داخل خاصيّة قناع الإدخال , inputMask قناع الإدخال يقيد عدد محارف الإدخال و مجموعات

المحارف التي يستطيع المستخدم إدخالها ,إذا كانت خاصيّة قناع الإدخال فارغة لن تعمل وظيفة تقييد الإدخال.

A	السماح فقط بإدخال المحارف الأبجديّة اللغوية - الأحرف الكبيرة و الصغيرة
a	السماح فقط بإدخال المحارف الأبجديّة اللغوية ,و السماح بإدخال محرف الفراغ- الأحرف الكبيرة و الصغيرة
N	السماح فقط بإدخال المحارف الأبجديّة اللغوية و الرقميّة - الأحرف الكبيرة و الصغيرة
n	السماح فقط بإدخال المحارف الأبجديّة اللغوية و الرقميّة ,و السماح بإدخال محرف الفراغ -الأحرف الكبيرة و الصغيرة
X	السماح بإدخال أيّ محرف
x	السماح بإدخال أيّ محرف
9	السماح فقط بإدخال الأبجديّة الرقميّة(9-0).
0	السماح فقط بإدخال الأبجديّة الرقميّة , و السماح بإدخال محرف الفراغ(9-0).
D	السماح فقط بإدخال الأبجديّة الرقميّة لكن من الرقم 1 إلى الرقم(9-1) 9
d	السماح فقط بإدخال الأبجديّة الرقميّة , لكن من الرقم 1 إلى الرقم, (9-1) 9 و السماح بإدخال محرف الفراغ.
#	السماح فقط بإدخال الأبجديّة الرقميّة, (9-0) , و السماح بإدخال محرف الفراغ و الإشارتين -,+
H	السماح فقط بإدخال المحارف الستة عشريّة
h	السماح فقط بإدخال المحارف الستة عشريّة ,و السماح بإدخال محرف الفراغ
B	السماح فقط بإدخال المحارف الثنائيّة
b	السماح فقط بإدخال المحارف الثنائيّة ,و السماح بإدخال محرف الفراغ
>	ستقوم بتحويل كافة المحارف التي تكون بعدها إلى أحرف كبيرة
<	ستقوم بتحويل كافة المحارف التي تكون بعدها إلى أحرف صغيرة
!	إلغاء عمليات التحويل على المحارف الموجودين بعد هذا الرمز ,! كعملية تحويل المحارف إلى محارف كبيرة , الرمز<
\	وضع المحرف الذي يليه كمحرف أساسيّ أي لا يحذف ولا يعدل

QString inputMask () const

يرجع نص قناع الإدخال.

void insert (const QString & newText)

إدخال نص من موقع المشيرة الحالي , و إذا كان نص ما محدد سوف يحذف و يضع النص الجديد مكانه.

bool isModified () const

يرجع القيمة true إذا غير محتوى النص الإفتراضي للكائن , QLineEdit أما إذا لم يغير فيرجع.false

bool isReadOnly () const

يرجع true إذا كان الكائن QLineEdit للقراءة فقط.

bool isRedoAvailable () const

يرجع true إذا تراجع المستخدم عن نص ما بعد ما تمّ إدخاله ,أيّ استخدم الأمر تراجع.Undo

bool isUndoAvailable () const

يرجع true إذا كان المستخدم قد عدل على النص الموجود.

int maxLength () const

يرجع طول النص.

QString placeholderText () const

يرجع النص الباهت الذي يظهر في كائن QLineEdit عندما يكون نصّه فارغ و غير مركز عليه.

QString selectedText () const

يرجع النص المحدد.

int selectionStart () const

يرجع موقع أول محرف من النص المحدد.

void setAlignment (Qt::Alignment flag)

يعدل إزاحة النص.

void setCompleter (QCompleter * c)

يضع المتمم الخاص بالكائن.QLineEdit

void setCursorPosition (int)

يغير موقع المشيرة.

void setDragEnabled (bool b)

لتفعيل أو تعطيل ميزة السحب.

void setEchoMode (EchoMode)

يضع نمط إظهار المحارف.

void setFrame (bool)

لإظهار أو إخفاء الإطار الموجود حول الكائن.QLineEdit

void setMaxLength (int)

لوضع أكبر عدد محارف يستطيع إدخاله المستخدم.

void setModified (bool)

لوضع حالة تغيير النص المدخل يأخذ وسيطها قيمة true أو.false

void setPlaceholderText (const QString &)

لوضع نص باهت للكائن.QLineEdit

void setReadOnly (bool)

لتفعيل أو تعطيل خاصية للقراءة فقط.

void setSelection (int start, int length)

لتحديد النص , يبدأ تحديده من المحرف ذو الموقع start بطول) عدد المحارف.length (

void setTextMargins (int left, int top, int right, int bottom)

لتعديل قيمة الهوامش للكائن.QLineEdit

void setTextMargins (const QMargins & margins)

لتعديل قيمة الهوامش للكائن QLineEdit لكن بحيث يكون نمط الوسيط الكائن.QMargins

void setValidator (const QValidator * v)

لوضع كائن التأكد من صحة الإدخال ,QValidator سنتكلم عنه بالتفصيل في الصفحات القادمة من هذا الفصل.

QString text () const

يرجع النص الموجود داخل الكائن.QLineEdit

QMargins textMargins () const

يرجع قيمة الهوامش على شكل كائن.QMargins

const QValidator * validator () const

يرجع مؤشر كائن.QValidator

مقابس (Slots) الكائن.QLineEdit

void clear ()

عند استدعاء هذا المقبس سوف يتم إزالة كامل النص المحتوى ضمن الكائن.QLineEdit

void copy () const

عند استدعاء هذا المقبس سوف يتم نسخ النص المحدد.

void cut ()

عند استدعاء هذا المقبس سوف يتم قص النص المحدد.

void paste ()

عند استدعاء هذا المقبس سوف يتم لصق النص.

void redo ()

عند استدعاء هذا المقبس سوف يتم التراجع عن آخر عملية تراجع قمت بها.

void selectAll ()

عند استدعاء هذا المقبس سوف يتم تحديد كامل النص.

void setText (const QString &text)

وضع نص للكائن QLineEdit حيث تكون قيمته هي الوسيط.text

void undo ()

عند استدعاء هذا المقبس سوف يتم التراجع عن آخر عملية قام بها المستخدم لنص الكائن QLineEdit.

أحداث (Signals) الكائن QLineEdit :

void cursorPositionChanged (int old, int new)

يقدح عند تغيير موقع المشيرة , الوسيط old هو موقع المشيرة قبل نقلها) تغيير موقعها , (الوسيط new هو موقع المشيرة بعد نقلها (تغيير موقعها).

void editingFinished ()

يقدح عند الانتهاء من تحرير النص الموجود داخل الكائن QLineEdit أيّ عند فقد التركيز للكائن أو ضغط مفتاحEnter عندما تكون قد وضعت قيمة للخاصيّة inputMask أو Validator

لن يقدح بالضغط على مفتاح Enter إلا بتحقيق شروط شرعيّة الإدخال.

void returnPressed ()

يقدح عند الضغط على مفتاح,Enter عندما تكون قد وضعت قيمة للخاصيّة inputMask أو Validator

لن يقدح هذا الحدث إلّا بتحقيق شروط شرعيّة الإدخال و من ثمّ الضغط على مفتاح.Enter

void selectionChanged ()

يقدح عند تحديد نص.

void textChanged (const QString & text)

يقدح عند تغيير النص الموجود داخل الكائن.QLineEdit

void textEdited (const QString & text)

يقدح عندما يكون كائن QLineEdit في حالة تحرير النص الموجود داخله.

البنية الهرمية للكائن QLineEdit :

QObject → QWidget →QLineEdit

Text Edit (QTextEdit) كائن لإظهار و تحرير نص غنيّ (Rich Text) و نص بسيط (Plain Text),

يستخدم لتنسيق النص الغنيّ وسوم.HTML-Style

أهم مناهجه:

bool acceptRichText () const

يرجع القيمة true إذا كان مسموح لكتابة أو لصق خط غنيّ - خط منسق.

Qt::Alignment alignment () const

يرجع الإزاحة الخاصة بالسطر النصيّ التي تكون المشيرة داخله.

bool canPaste () const

يرجع true إذا كان في الحافظة بيانات نسخت عليها و هذه البيانات مسموح لها أن تلصق في الكائن.QTextEdit

QMenu * createStandardContextMenu ()

يرجع مؤشر كائن قائمة , نستطيع بواسطته إضافة قائمة منبثقة خاصة بالكائن QLineEdit تظهر عند النقر بزر الفأرة الأيمن على الكائن.QTextEdit

QMenu * createStandardContextMenu (const QPoint & position)

يرجع مؤشر كائن قائمة , نستطيع بواسطته إضافة قائمة منبثقة خاصة بالكائن QLineEdit تظهر عند النقر بزر الفأرة الأيمن على الكائن ,QTextEdit و الوسيط position ليحدد موقع إظهار القائمة.

QTextCharFormat currentCharFormat () const

يرجع تنسيق النص التي تكون المشيرة موجودة بين محارفه .

QFont currentFont () const

يرجع كائن الخط الخاص بالنص التي تكون المشيرة موجودة بين محارفه حاليا .

QTextDocument * document () const

يرجع مؤشر مستند نص الكائن QTextEdit مع تنسيقه.

QString documentTitle () const

يرجع نص عنوان المستند.

bool find (const QString & exp, QTextDocument::FindFlags options = 0)

ترجع true إذا وجد النص داخل الكائن , QTextEdit يبدأ البحث من موقع المشيرة الحالي إلى نهاية النص ,عندما يجد النص يحدد النص داخل الكائن.QTextEdit

QString fontFamily () const

يرجع اسم خط النص التي تكون المشيرة موجودة بين محارفه حاليا..

Bool fontItalic () const

يرجع true إذا كان النص الحالي ذو تنسيق مائل.

qreal fontPointSize () const

يرجع حجم الخط للنص الحالي.

bool fontUnderline () const

يرجع true إذا كان النص الحالي ذو تنسيق خط سفلي.

int fontWeight () const

يرجع عرض الخط للنص الحالي.

bool isReadOnly () const

يرجع true إذا كان الكائن QTextEdit للقراءة فقط

void mergeCurrentCharFormat (const QTextCharFormat & modifier)

دمج التنسيق modifier مع تنسيق المحرف الحالي.

void setAcceptRichText (bool accept)

إذا مررت للوسيط القيمة true سيقبل الكائن QTextEdit تنسيق النص الغنيّ ,و إذا مررت false لن يقبل إلا النص البسيط.

void setCurrentCharFormat (const QTextCharFormat & format)

وضع تنسيق للنص المحدد و من موقع المشيرة الحالي.

void setCursorWidth (int width)

تعديل عرض المشيرة بمقياس.pix

void setDocument (QTextDocument * document)

يضع مستند كائن QtextDocument للكائن.QTextEdit

void setDocumentTitle (const QString & title)

وضع عنوان للمستند.

void setReadOnly (bool ro)

تعديل خاصيّة للقراءة فقط.

void setTabChangesFocus (bool b)

إذا مررت للوسيط القيمة true سيعمل مفتاح tab بنقل التركيز عن الكائن QTextEdit و إذا مررت للوسيط القيمة false سوف يعمل مفتاح tab على وضع مسافة جدوليّة ضمن النص من موقع المشيرة الحالي,القيمة الافتراضية للخاصيّة tabChangesFocus هي.false

void setTabStopWidth (int width)

وضع طول المسافة الجدوليّة , القيمة الافتراضية 80 بكسل.

void setTextInteractionFlags (Qt::TextInteractionFlags flags)

تحدد هذه المنهجيّة كيفية استجابة الكائن QTextEdit للأحداث المدخلة من قبل المستخدم , مثال تحديد النص بواسطة الفأرة كحدث النقر و مسك زر الفأرة و سحبها على النص الذي نريد تحديده ,انظر إلى الجدول , التعداد Qt::TextInteractionFlag :

عدم تفاعل الكائن مع أي حدث من المستخدم	**Qt::NoTextInteraction**
يستطيع المستخدم فقط تحديد النص بواسطة الفأرة و نسخ نص	**Qt::TextSelectableByMouse**

Qt::TextSelectableByKeyboard	يستطيع المستخدم فقط تحديد النص بواسطة لوحة المفاتيح و نسخ نص
Qt::LinksAccessibleByMouse	يستطيع المستخدم تفعيل الروابط الموجودة ضمن النص فقط بواسطة النقر بالفأرة عليها
Qt::LinksAccessibleByKeyboard	يستطيع المستخدم فتح الروابط الموجودة ضمن النص فقط بواسطة لوحة المفاتيح(التحديد على الرابط بواسطة مفتاح Tabو تفعيل الرابط بالضغط على مفتاح Enter)
Qt::TextEditable	يستجيب الكائن لكافة أنواع إدخال المستخدم الخاصة بتحرير النص فقط.
Qt::TextEditorInteraction	Qt::TextSelectableByMouse | Qt::TextSelectableByKeyboard | Qt::TextEditable
Qt::TextBrowserInteraction	Qt::TextSelectableByMouse | Qt::LinksAccessibleByMouse | Qt::LinksAccessibleByKeyboard

int tabStopWidth () const

يرجع قيمة المسافة الجدوليّة بالبكسل.

QColor textBackgroundColor () const

يرجع لون خلفيّة النص الحالي.

QColor textColor () const

يرجع لون النص الحالي.

QString toHtml () const

يرجع نص مع تنسيق بصيغة HTML للكائن. QTextEdit

QString toPlainText () const

يرجع نص من دون تنسيق .

المقابس (Slots) للكائن:QTextEdit

void append (const QString & text)

مقبس إضافة النص.

void clear ()

مقبس مسح كامل النص.

void copy ()

مقبس نسخ النص المحدد إلى الحافظة.

void cut ()

مقبس قص النص المحدد إلى الحافظة.

void insertHtml (const QString & text)

مقبس إضافة النص بصيغة.HTML

void insertPlainText (const QString & text)

مقبس إضافة نص من دون تنسيق.

void paste ()

لصق النص.

void redo ()

تراجع عن آخر خطوة تراجع.

void selectAll ()

مقبس تحديد كامل النص.

void setHtml (const QString & text)

مقبس وضع نص بصيغة.HTML

void setPlainText (const QString & text)

مقبس وضع نص بسيط.

void setText (const QString & text)

مقبس وضع نص.

void setTextBackgroundColor (const QColor & c)

مقبس وضع لون لخلفية النص الحالي.

void setTextColor (const QColor & c)

مقبس وضع لون للنص الحالي.

void undo ()

تراجع.

void zoomIn (int range = 1)

تكبير حجم خط كامل النص , لا يستطيع هذا المقبس تكبير حجم الصورة.

void zoomOut (int range = 1)

تصغير حجم خط كامل النص , لا يستطيع هذا المقبس تصغير حجم الصورة.

أحداث (Signals) الكائن: QTextEdit

void copyAvailable (bool yes)

يقدح عندما يحدد نص ما أو يلغى تحديد نص.

void currentCharFormatChanged (const QTextCharFormat & f)

يقدح عند تغيير تنسيق النص الحالي.

void cursorPositionChanged ()

يقدح عند تغيير موقع المشيرة.

void redoAvailable (bool available)

void selectionChanged ()

void textChanged ()

void undoAvailable (bool available)

شرحنا الأربع أحداث الأخير في قسم الكائن.QLineEdit

البنية الهرمية للكائن: QTextEdit

QObject → QWidget → QFrame →QAbstractScrollArea → QTextEdit

Plain Text Edit (QPlainTextEdit) كائن لتحرير و عرض نص بسيط (Plain Text), لا يستطيع هذا الكائن التعامل مع النص ذو التنسيق الغنيّ.

يوجد العديد من المناهج و المقابس و الإشارات المشتركة مع الكائن QTextEdit و الكائن QLineEdit, لذا سنورد هنا فقط شرح المناهج و المقابس و الأحداث الغير مذكورة مسبقا.

أهم مناهجه:

bool backgroundVisible () const

int blockCount () const

يرجع عدد الأسطر.

bool canPaste () const

يرجع true إذا كان يوجد بيانات في الحافظة جاهزة للصق.

QMenu * createStandardContextMenu ()

QTextCharFormat currentCharFormat () const

QTextCursor cursorForPosition (const QPoint & pos) const

int cursorWidth () const

QTextDocument * document () const

QString documentTitle () const

void ensureCursorVisible ()

bool find (const QString & exp, QTextDocument::FindFlags options = 0)

bool isReadOnly () const

bool isUndoRedoEnabled () const

int maximumBlockCount () const

يرجع قيمة عدد الأسطر المسموح إضافتها إلى الكائن.QPlainTextEdit

void mergeCurrentCharFormat (const QTextCharFormat & modifier)

void print (QPrinter * printer) const

void setBackgroundVisible (bool visible)

void setCurrentCharFormat (const QTextCharFormat & format)

void setCursorWidth (int width)

void setDocument (QTextDocument * document)

void setDocumentTitle (const QString & title)

void setMaximumBlockCount (int maximum)

وضع عدد الأسطر المسموح إضافتها للكائن.QPlainTextEdit

void setReadOnly (bool ro)

void setTabChangesFocus (bool b)

void setTabStopWidth (int width)

void setTextInteractionFlags (Qt::TextInteractionFlags flags)

void setUndoRedoEnabled (bool enable)

void setWordWrapMode (QTextOption::WrapMode policy)

bool tabChangesFocus () const

int tabStopWidth () const

Qt::TextInteractionFlags textInteractionFlags () const

QString toPlainText () const

مقابس (Slots) الكائن:QPlainTextEdit

void appendHtml (const QString & html)

مقبس لإضافة نص بصيغ.HTML

void appendPlainText (const QString & text)

مقبس لإضافة نص بسيط.

void clear ()

مقبس مسح كامل النص الموجود داخل الكائن.QPlainTextEdit

void copy ()

void cut ()

void insertPlainText (const QString & text)

void paste ()

void redo ()

void selectAll ()

void setPlainText (const QString & text)

void undo ()

أحداث (Signals) الكائن:QPlainTextEdit

void blockCountChanged (int newBlockCount)

يقدح عندما يزيد أو ينقص عدد الأسطر.

void copyAvailable (bool yes)

void cursorPositionChanged ()

void modificationChanged (bool changed)

void redoAvailable (bool available)

void selectionChanged ()

void textChanged ()

void undoAvailable (bool available)

void updateRequest (const QRect & rect, int dy)

البنية الهرميّة للكائن:QPlainTextEdit

QObject → QWidget → QFrame →QAbstractScrollArea → QPlainTextEdit

Spin Box (QSpinBox) : **صندوق الدوران , يستخدم لتضع فيه قيم من نمط صحيح integerو تكون هذه القيم ضمن مجال معين فرضا من 1 إلى. 12**

أهم مناهجه:

QString cleanText () const

يرجع قيمة صندوق الدوران من دون البادئة أو اللاحقة.

int maximum () const

ترجع أكبر قيمة في مجال صندوق الدوران.

int minimum () const

ترجع أصغر قيمة في مجال صندوق الدوران.

QString prefix () const

ترجع البادئة قيمة نصيّة تظهر بالجانب الخلفي لقيمة صندوق الدوران .

void setMaximum (int max)

void setMinimum (int min)

void setPrefix (const QString & prefix)

تضع البادئة.

void setRange (int minimum, int maximum)

تضع مجال صندوق الدوران أصغر قيمة و أكبر قيمة

void setSingleStep (int val)

قيمة مقدار التزايد و التناقص.

void setSuffix (const QString & suffix)

تضع اللاحقة.

int singleStep () const

QString suffix () const

ترجع اللاحقة قيمة نصيّة تظهر بالجانب الأمامي لقيمة صندوق الدوران .

int value () const

ترجع قيمة صندوق الدوران.

مقابس (Slots) الكائن QSpinBox :

void setValue (int val)

مقبس لوضع قيمة صندوق الدوران الحالية.

الأحداث الداخليّة (Signals) للكائن QSpinBox :

void valueChanged (int i)

void valueChanged (const QString & text)

يقدح عند تغيير قيمة الكائن.QSpinBox

البنية الهرمية للكائن QSpinBox :

QObject → QWidget → QFrame →QAbstractSpinBox → QSpinBox

Double Spin Box (QDoubleSpinBox) :شبيه الكائن السابق لكنه يأخذ قيم من نمط Double

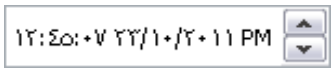
)بفاصلة عائمة , (له نفس منهجيات و مقابس و أحداث الكائن QSpinBox ما عدا منهجية decimals

تستخدم لوضع عدد الخانات الرقميّة بعد الفاصلة.

البنية الهرمية للكائن QDoubleSpinBox :

QObject → QWidget → QFrame →QAbstractSpinBox → QDoubleSpinBox

Date/Time Edit (QDateTimeEdit) : كائن لتحرير وإظهار التاريخ و الوقت بتنسيق ما , تستطيع تعديل الوقت و التاريخ باستخدام سهم الأعلى و سهم الأسفل ضمن لوحة المفاتيح أو باستخدام الفأرة بالنقر على سهم الأعلى أو سهم الأسفل.



أهم مناهجه:

bool calendarPopup () const

ترجع true إذا كان يستطيع إظهار لائحة التقويم , القيمة الافتراضية هي false أي لا يستطيع إظهار لائحة التقويم ,عندما تقوم قيمة المنهجية true سيظهر بدل من سهمين الأعلى و الأسفل سهم واحد متجه نحو الأسفل عند النقر عليه ستظهر لائحة التقويم.

QCalendarWidget * calendarWidget () const

يرجع مؤشر لائحة التقويم.

void clearMaximumDate ()

يعيد أكبر قيمة للتاريخ إلى قيمته الافتراضية , القيمة الافتراضية هي. 31\12\7999

void clearMaximumDateTime ()

يعيد أكبر قيمة للتاريخ و الوقت إلى قيمته الافتراضية , القيمة الافتراضية هي 31\12\7999 23:59:59 –و 999 ميلي ثانية.

void clearMaximumTime ()

يعيد أكبر قيمة للوقت إلى قيمته الافتراضية , القيمة الافتراضية هي 23:59:59 و 999 ميلي ثانية.

void clearMinimumDate ()

يعيد أصغر قيمة للتاريخ إلى قيمته الافتراضية , القيمة الافتراضية هي. 14\09\1752

void clearMinimumDateTime ()

يعيد أصغر قيمة للتاريخ و الوقت إلى قيمته الافتراضية , القيمة الافتراضية هي 14\09\1752 00:00:00 –و 0 ميلي ثانية.

void clearMinimumTime ()

يعيد أصغر قيمة للوقت إلى قيمته الافتراضية , القيمة الافتراضية هي 00:00:00 و 0 ميلي ثانية.

Section currentSection () const

يرجع القسم المحدد الحالي من الكائن , QDateTimeEdit انظر إلى الجدول , التعداد QDateTimeEdit::Section :

لا تحديد	**QDateTimeEdit::NoSection**
قسم الصباح \ المساء	**QDateTimeEdit::AmPmSection**
قسم الميلي ثانية	**QDateTimeEdit::MSecSection**
قسم الثوان	**QDateTimeEdit::SecondSection**
قسم الدقائق	**QDateTimeEdit::MinuteSection**
قسم الساعات	**QDateTimeEdit::HourSection**
قسم الأيام	**QDateTimeEdit::DaySection**
قسم الأشهر	**QDateTimeEdit::MonthSection**
قسم السنين	**QDateTimeEdit::YearSection**

int currentSectionIndex () const

تعيد رقم القسم المحدد , إذا كان تنسيق التاريخ yyyy/MM/dd انظر الجدول:

قسم السنة	**0**
قسم الشهر	**1**
قسم اليوم	**2**
قسم الساعة	**3**
قسم الدقائق	**4**
قسم الثوان	**5**
قسم الصباح \ المساء	**6**

QDate date () const

يرجع كائن التاريخ الخاص بالكائن.QDateTimeEdit

QDateTime dateTime () const

يرجع كائن الوقت و التاريخ الخاص بالكائن.QDateTimeEdit

QString displayFormat () const

يرجع نص تنسيق العرض , انظر الجدول:

23/10/2011	**dd/MM/yyyy**

MMM-d-yy	Oct-23-11
MMMM\d\yy	October\23\11

نستطيع وضع أي رمز نريد كفاصل بين كل من اليوم و الشهر و السنة.

QDate maximumDate () const

QDateTime maximumDateTime () const

QTime maximumTime () const

QDate minimumDate () const

QDateTime minimumDateTime () const

QTime minimumTime () const

Section sectionAt (int index) const

يرجع القسم ذو العنوان index.

int sectionCount () const

يرجع عدد الأقسام.

QString sectionText (Section section) const

يرجع نص القسم المحدد بواسطة الوسيط section.

void setCalendarPopup (bool enable)

لتفعيل أو إلغاء تفعيل إظهار لائحة التقويم.

void setCalendarWidget (QCalendarWidget * calendarWidget)

وضع لائحة تقويم.

void setCurrentSection (Section section)

تغيير تركيز القسم إلى القسم section.

void setCurrentSectionIndex (int index)

تغيير تركيز القسم إلى القسم ذو الموقع index.

void setDateRange (const QDate & min, const QDate & max)

وضع مجال للتاريخ , أصغر قيمة للتاريخ نستطيع إظهارها , و أكبر قيمة.

void setDateTimeRange (const QDateTime & min, const QDateTime & max)

void setDisplayFormat (const QString & format)

void setMaximumDate (const QDate & max)

void setMaximumDateTime (const QDateTime & dt)

void setMaximumTime (const QTime & max)

void setMinimumDate (const QDate & min)

void setMinimumDateTime (const QDateTime & dt)

void setMinimumTime (const QTime & min)

void setSelectedSection (Section section)

تحديد القسم.section

void setTimeRange (const QTime & min, const QTime & max)

void setTimeSpec (Qt::TimeSpec spec)

لتخصيص الوقت على نظام التوقيت المحلي (Qt::LocalTime) أو توقيت غرينتش العالمي (Qt::UTC).

QTime time () const

Qt::TimeSpec timeSpec () const

مقابس (Slots) الكائن QDateTimeEdit :

void setDate (const QDate & date)

مقبس لوضع قيمة التاريخ.

void setDateTime (const QDateTime & dateTime)

مقبس لوضع قيمة التاريخ و الوقت.

void setTime (const QTime & time)

مقبس لوضع قيمة الوقت.

أحداث (Signals) الكائن QDateTimeEdit :

void dateChanged (const QDate & date)

يقدح عندما يغير المستخدم التاريخ للكائن.QDateTimeEdit

void dateTimeChanged (const QDateTime & datetime)

يقدح عندما يغير المستخدم التاريخ أو الوقت للكائن.QDateTimeEdit

void timeChanged (const QTime & time)

يقدح عندما يغير المستخدم الوقت للكائن.QDateTimeEdit

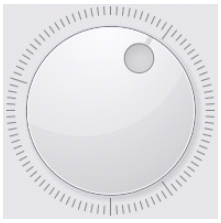
البنية الهرمية للكائن QDateTimeEdit :

QObject → QWidget → QFrame →QAbstractSpinBox →QDateTimeEdit

Time Edit (QTimeEdit): كائن لعرض و تحرير الوقت ,الصف الأساسي له هو QDateTimeEdit.

Date Edit (QDateEdit) كائن لعرض و تحرير التاريخ ,الصف الأساسي له هو QDateTimeEdit.

Dial (QDial) : كائن قرص مدرج ضمن مجال معين , مشتق من الصف QAbstractSliderالخاص بالتعامل مع القيم الصحيحة التي تقع ضمن مجال رقمي معين , يحوي هذا الصف المنهجيات الأساسية و المشتركة لجميع كائنات المنزلقة-QDial) QScrollBar-QSlider).



مناهج الصف QAbstractSlider

bool hasTracking () const

يرجع true إذا كانت الخاصية tracking مفعّلة , عندما تكون الخاصية Tracking مفعّلة سوف يقدح الحدث ()valueChanged عندما يكون المستخدم بسحب المنزلقة , وإذا كانت الخاصية tracking غير مفعّلة سوف لن يقدح الحدث ()valueChanged إلا عندما يترك المستخدم سحب المنزلقة.

bool invertedAppearance () const

يعكس مظهر المنزلقة من اليسار إلى اليمين و بالعكس.

bool invertedControls () const

يعكس عمل مفاتيح و أزرار التحكم بالمنزلقة , مثال يصبح عمل مفتاح السهم الأعلى للوحة المفاتيح إنقاص قيمة المنزلقة.

int maximum () const

أكبر قيمة للمنزلقة.

int minimum () const

أصغر قيمة للمنزلقة.

Qt::Orientation orientation () const

اتجاه المنزلقة.

int pageStep () const

قيمة تزايد أو تناقص المنزلقة بالضغط على مفتاح.(page Up - page Down)

void setInvertedAppearance (bool)

void setInvertedControls (bool)

void setMaximum (int)

void setMinimum (int)

void setPageStep (int)

void setRange (int min, int max)

void setSingleStep (int)

قيمة تزايد أو تناقص المنزلقة بالخطوة الواحدة .

void setSliderPosition (int pos)

تغيير موقع المنزلقة إلى الموقع.pos

void setTracking (bool enable)

int singleStep () const

int sliderPosition () const

int value () const

يرجع قيمة المنزلقة.

مقابس (Slots) الصف:QAbstractSlider

void setOrientation (Qt::Orientation)

void setValue (int)

أحداث (Signals) الصف: QAbstractSlider

void actionTriggered (int action)

يقدح عندما يقوم المستخدم بأي فعل على كائن المنزلقة من خلال الفأرة أو لوحة المفاتيح ,مثال عندما يقوم المستخدم بضغط مفتاح سهم الأعلى ,بالنسبة للوسيط action انظر الجدول , التعداد QAbstractSlider::SliderAction :

QAbstractSlider::SliderNoAction	**0**	**لا فعل من المستخدم الحالة الافتراضية**
QAbstractSlider::SliderSingleStepAdd	**1**	**عندما يقوم المستخدم بالضغط على مفتاح سهم الأعلى**

عندما يقوم المستخدم بالضغط على مفتاح سهم الأسفل	2	QAbstractSlider::SliderSingleStepSub
عندما يقوم المستخدم بالضغط على مفتاحPage Up	3	QAbstractSlider::SliderPageStepAdd
عندما يقوم المستخدم بالضغط على مفتاح Page Down	4	QAbstractSlider::SliderPageStepSub
عند نقل المنزلقة إلى أصغر قيمة لها	5	QAbstractSlider::SliderToMinimum
عند نقل المنزلقة إلى أكبر قيمة لها	6	QAbstractSlider::SliderToMaximum
عند تحريك المنزلقة برمجيا أو بواسطة الفأرة	7	QAbstractSlider::SliderMove

void rangeChanged (int min, int max)

يقدح عندما يغير قيمة مجال المنزلقة.

void sliderMoved (int value)

يقدح عند نقل المنزلقة إلى الموقع ذو القيمة.value

void sliderPressed ()

يقدح عند الضغط على المنزلقة بزر الفأرة.

void sliderReleased ()

يقدح عند إفلات المنزلقة من زر الفأرة.

void valueChanged (int value)

يقدح عندما يغير قيمة المنزلقة.

مناهج الكائن: QDial :

int notchSize () const

يرجع عدد السنن , يرتبط عدد السنن بقيمة الخاصية singleStep عندما تكون قيمة الخاصية singleStepمساوية 2 سوف تكون السنن موزعة على محيط القرص و كل درجتين يوجد سنّة واحدة , أي المسافة التي تقع بين سنتين تساوي القيمة.2

qreal notchTarget () const

يرجع مسافة التباعد بين سنن الدرجات التي تكون حوالي قرص الكائن , QDial يقاس الحجم بالبكسل , الحجم يقبل الفاصلة.

bool notchesVisible () const

يرجع true إذا كانت سنن الدرجات ظاهرة و إلا يرجع.false

void setNotchTarget (double target)

وضع قيمة مسافة التباعد بين سنن الدرجات.

bool wrapping () const

يرجع true إذا كانت منزلقة الكائن QDial تقبل تحريكها دورة كاملة.

مقابس (Slots) الكائن: QDial

void setNotchesVisible (bool visible)

void setWrapping (bool on)

أحداث (Signals) الكائن QDial نفس أحداث الصف.QAbstractSlider

البنية الهرمية للكائن: QDial

QObject → QWidget →QAbstractSlider → QDial

كائنات المنزلقة:

QScrollBar

شريط الانزلاق , يمكن أن يكون مظهره عمودي أو أفقي.

البنية الهرمية:

QObject → QWidget →QAbstractSlider

QSlider 

البنية الهرمية للكائن:QSlider

QObject → QWidget →QAbstractSlider → QSlider

- **الأدوات التابعة لتبويب كائنات العرض (Display Widgets):**

Label (QLabel) : كائن اللافتة ,يستخدم لعرض نص أو صورة ما.

مناهج الكائن QLabel :

Qt::Alignment alignment () const

bool hasScaledContents () const

يرجع true إذا كانت الصورة التي يحتويها ممتدة على كامل مساحة الكائن ,QLabel وإلا ترجع , false القيمة الافتراضية.false

bool hasSelectedText () const

ترجع true إذا كان نص محدد داخل. QLabel

int margin () const

يرجع قيمة مسافة الهامش , يرتبط الهامش في كائن QLabel بخاصية الإزاحة , مثال إذا كانت الإزاحة إلى اليمين و الأعلى سوف يضع هامش يميني و علوي.

QMovie * movie () const

يرجع مؤشر كائن QMovie الخاص بتشغيل الصور المتحركة ك صور.(GIF)

bool openExternalLinks () const

يرجع true إذا كان متاح فتح الروابط ,الموجودة داخل الكائن.QLabel

const QPicture * picture () const

يرجع مؤشر كائن الصورة الموجودة داخل كائن QLabel و إذا كان لا يوجد داخله صورة يرجع 0.

const QPixmap * pixmap () const

يرجع مؤشر كائن QPixmap الخاص بتحضير و عرض الصور ,و إذا كان لا يوجد داخل الكائن QLabel كائن QPixmap يرجع0.

QString selectedText () const

يرجع النص المحدد , في حال لا يوجد نص محدد يرجع كائن QString فارغ.

int selectionStart () const

يرجع موقع أول محرف من النص المحدد.

void setAlignment (Qt::Alignment)

void setMargin (int)

void setOpenExternalLinks (bool open)

void setScaledContents (bool)

void setSelection (int start, int length)

void setTextFormat (Qt::TextFormat)

وضع تنسيق لنص كائن اللافتة , نص بسيط أو غني, القيمة الافتراضية تنسيق النص بشكل تلقائي أيّ حسب تنسيق المستخدم لخط النص.

void setTextInteractionFlags (Qt::TextInteractionFlags flags)

QString text () const

Qt::TextFormat textFormat () const

Qt::TextInteractionFlags textInteractionFlags () const

مقابس (Slots) الكائن: QLabel

void clear ()

void setMovie (QMovie * movie)

void setNum (int num)

عند استدعاءه يصبح قيمة الكائن QLabel الرقم صحيح.num

void setNum (double num)

عند استدعاءه يصبح قيمة الكائن QLabel الرقم المضاعف) ذو فاصلة عائمة (-Double-num.

void setPicture (const QPicture & picture)

void setPixmap (const QPixmap &)

void setText (const QString &)

أحداث (Signals) الكائن QLabel

void linkActivated (const QString & link)

يقدح عند تفعيل الرابط الموجود داخل الكائن , QLabel الوسيط link يحتوي على عنوان الرابط.

void linkHovered (const QString & link)

يقدح عند تحريك مؤشر الفأرة فوق رابط موجود داخل الكائن , QLabel الوسيط link يحتوي على عنوان الرابط.

البنية الهرمية للكائنQLabel

QObject → QWidget →QFrame → QLabel

Text Browser (QTextBrowser): كائن يستخدم لفتح مستندات تحوي نص غنيّ مع حفظ خصائص كل مستند تمّ فتحه) موقعه-عنوانه ,(...يعرض نص غني صيغته ,HTML اشتق من الكائن QTextEdit بنمط للقراءة فقط , مع إضافة عدة مناهج جديدة كمنهجية الانتقال للأمام (Forward) أو للخلف (Backward) أو الانتقال لأول صفحة قد فتحت , (home)أهم مناهجه:

int backwardHistoryCount () const

يرجع عدد الصفحات المزارة مسبقا و التي يستطيع المستخدم الانتقال إليها بواسطة منهجية الانتقال للخلف.

void clearHistory ()

مسح الصفحات المزارة.

int forwardHistoryCount () const

يرجع عدد الصفحات المزارة مسبقا و التي يستطيع المستخدم الانتقال إليها بواسطة منهجية الانتقال للأمام.

QString historyTitle (int i) const

يرجع عنوان الصفحة المزارة ذو العنوان i ضمن الصفحات المزارة مسبقا.

QUrl historyUrl (int i) const

يرجع رابط الصفحة المزارة ذو العنوان i ضمن الصفحات المزارة مسبقا.

bool isBackwardAvailable () const

يرجع true إن كان متاحا الانتقال للخلف من الصفحات المزارة مسبقا.

bool isForwardAvailable () const

يرجع true إن كان متاحا الانتقال للأمام من الصفحات المزارة مسبقا.

bool openExternalLinks () const

يرجع true إذا كان متاحا فتح الروابط ذو الارتباطات الخارجية التي يكون نص رابطها الرئيسي غير نص الرابط الرئيسي الحالي.

bool openLinks () const

يرجع true إن كان متاحا فتح الروابط ضمن الصفحة الحالية.

QStringList searchPaths () const

يرجع كائن قائمة نصية ,يستخدم هذا المنهج لحفظ مسارات الصور و الملفات التي نريد أن نتعامل معها ضمن الكائن QTextBrowser كإنشاء صفحة مساعدة بصيغة HTML تحتوي على صور و تنسيق CSS ما نكون قد حفظنا مواقع هذه الملفات باستخدام المنهجية:

setSearchPaths (const QStringList & paths)

void setOpenExternalLinks (bool open)

void setOpenLinks (bool open)

void setSearchPaths (const QStringList & paths)

QUrl source () const

يعيد مصدر(موقع)المستند الحالي , بنمط. url

مقابس (Slots) الكائن: QTextBrowser

virtual void backward ()

الانتقال إلى الخلف.

virtual void forward ()

الانتقال إلى الأمام.

virtual void home ()

الانتقال إلى أول صفحة قد تم فتحها.

virtual void reload ()

إعادة تحميل المستند الحالي.

virtual void setSource (const QUrl & name)

وضع عنوان المستند الذي نريد فتحه - عرضه .

textBrowser->setSource (QUrl("file:///C:/help/index.htm"));

أحداث (Signals) الكائن QTextBrowser :

void anchorClicked (const QUrl & link)

يقدح عند الضغط على رابط بمفتاح Enter أو عند النقر بالفأرة على الرابط , الوسيط link يحتوي على موقع الرابط الذي قد وقع الحدث عليه.

void backwardAvailable (bool available)

يقدح عند عرض مستند آخر , الوسيط available يكون true إذا كان متاح الانتقال إلى الخلف , أما عندما يكون غير متاح الانتقال إلى الخلف ستكون قيمته. false

void forwardAvailable (bool available)

يقدح عند عرض مستند آخر , الوسيط available يكون true إذا كان متاح الانتقال إلى الأمام ,أما عندما يكون غير متاح الانتقال إلى الأمام ستكون قيمته. false

void highlighted (const QUrl & link)

يقدح عندما يحرك المستخدم مؤشر الفأرة فوق الرابط , الوسيط link هو عنوان الرابط.

void highlighted (const QString & link)

يقدح عندما يحرك المستخدم مؤشر الفأرة فوق الرابط , الوسيط link هو عنوان الرابط بصيغة نصية.

void historyChanged ()

يقدح عند إضافة أو حذف مواصفات مستند ما ضمن المحفوظات.

void sourceChanged (const QUrl & src)

يقدح عند تغير موقع المستند.

البنية الهرمية للكائن QTextBrowser :

QObject → QWidget → QFrame →QAbstractScrollArea → QTextEdit →QTextBrowser

Graphics View (QGraphicsView) :كائن تحضير و عرض المشاهد الرسوميّة

سوف نشرح عن هذا الكائن بالتفصيل ضمن فصل برمجة الرسوميّات.Graphics

البنية الهرمية للكائن QGraphicsView

QObject → QWidget → QFrame →QAbstractScrollArea → QGraphicsView

Calendar Widget (QCalendarWidget) : كائن عرض لائحة التاريخ , نستطيع من خلاله أن نحصل على التاريخ المحدد عليه أو تاريخنا الحالي , ونستطيع أيضا تحرير التاريخ بشكل كامل.

مناهج الكائن QCalendarWidget

QMap<QDate, QTextCharFormat> dateTextFormat () const

يرجع قيمة من نمط مجموعة مفهرسة (مزدوجة) مفتاح – قيمة QMap<T , T> النمط الأول فيها هو المفتاح (يعتبر عنوان القيمة) النمط الثاني هو القيمة.

المفتاح في هذه المنهجية هو كائن تاريخ أما القيمة هي كائن تنسيق النص (QTextCharFormat).

يرجع لنا هذا المنهج جميع تنسيقات النصية للتواريخ المنسقة مسبقا.

```
QMap<QDate ,QTextCharFormat> mapFrmt = calendarWidget->dateTextFormat();

QList<QDate> lstDate =  mapFrmt.values();

QDate date;

date.setDate(2012, 12, 12);

QString  fontF = mapFrmt.value( date ).fontFamily();

//QString  fontF  = lstDate.at(0). fontFamily();
```

QTextCharFormat dateTextFormat (const QDate & date) const

يرجع تنسيق النص ذو التاريخ.date

Qt::DayOfWeek firstDayOfWeek () const

يرجع قيمة أول يوم في الأسبوع(اليوم الذي يعرض في أول عمود من لوحة التاريخ),

محتوى التعداد , Qt::DayOfWeek انظر الجدول:

يوم الاثنين	**Qt::Monday**
يوم الثلاثاء	**Qt::Tuesday**
يوم الأربعاء	**Qt::Wednesday**
يوم الخميس	**Qt::Thursday**
يوم الجمعة	**Qt::Friday**
يوم السبت	**Qt::Saturday**
يوم الأحد	**Qt::Sunday**

QTextCharFormat headerTextFormat () const

يرجع تنسيق نص العنوان الرأسي الأفقي و العمودي للكائن.QCalendarWidget

HorizontalHeaderFormat horizontalHeaderFormat () const

يرجع تنسيق عنوان الرأسي الأفقي (نمط إظهار أسماء أيام الأسبوع)

التعداد QCalendarWidget::HorizontalHeaderFormat انظر الجدول الخاص به:

يظهر أول محرف من اسم اليوم	**QCalendarWidget::SingleLetterDayNames**
يظهر الاسم المختصر لليوم	**QCalendarWidget::ShortDayNames**
يظهر اسم اليوم كامل	**QCalendarWidget::LongDayNames**
عدم إظهار العنوان الرأسي الأفقي الذي يحوي أسماء أيام الأسبوع	**QCalendarWidget::NoHorizontalHeader**

bool isGridVisible () const

يرجع true إذا كان حدود خلايا الجدول ظاهرة.

bool isNavigationBarVisible () const

يرجع true إذا كان الشريط الخاص بإظهار السنة و الشهر (الشريط العلوي للكائن QcalendarWidget) ظاهر , و إذا كان في حالة عدم إظهار يرجع.false

QDate maximumDate () const

QDate minimumDate () const

int monthShown () const

يرجع رقم الشهر المحدد عليه حاليا ضمن الكائن.QCalendarWidget

QDate selectedDate () const

يرجع التاريخ المحدد عليه حاليا.

SelectionMode selectionMode () const

يرجع نمط التحديد , انظر الجدول , التعداد QCalendarWidget::SelectionMode :

QCalendarWidget::NoSelection	**لا يستطيع المستخدم تحديد تاريخ**
QCalendarWidget::SingleSelection	**يستطيع المستخدم تحديد تاريخ وحيد فقط**

void setDateTextFormat (const QDate & date, const QTextCharFormat & format)

وضع تنسيق نص للتاريخ.date

void setFirstDayOfWeek (Qt::DayOfWeek dayOfWeek)

اختيار أول يوم للأسبوع(ما اسم اليوم الذي سيبدأ به الأسبوع)

void setHeaderTextFormat (const QTextCharFormat & format)

وضع تنسيق نص للعنوان الرأسي الأفقي و العمودي.

void setHorizontalHeaderFormat (HorizontalHeaderFormat format)

void setMaximumDate (const QDate & date)

void setMinimumDate (const QDate & date)

void setSelectionMode (SelectionMode mode)

void setVerticalHeaderFormat (VerticalHeaderFormat format)

وضع تنسيق العنوان الرأسي العمودي (إظهار رقم الأسبوع من السنة أو عدم إظهاره) , انظر للجدول الخاص بالتعدادQCalendarWidget::VerticalHeaderFormat :

إظهار رقم الأسبوع	**QCalendarWidget::ISOWeekNumbers**
عدم إظهار العنوان الرأسي العمودي	**QCalendarWidget::NoVerticalHeader**

void setWeekdayTextFormat (Qt::DayOfWeek dayOfWeek, const QTextCharFormat & format)

وضع تنسيق نصي لعمود اليوم.dayOfWeek

VerticalHeaderFormat verticalHeaderFormat () const

QTextCharFormat weekdayTextFormat (Qt::DayOfWeek dayOfWeek) const

int yearShown () const

يرجع رقم صحيح قيمته هي السنة المحددة ضمن الكائن. QCalendarWidget

مقابس (Slots) الكائن QCalendarWidget :

void setCurrentPage (int year, int month)

الانتقال إلى الصفحة ذو السنة التي قيمتها year و الشهر الذي قيمته.month

void setDateRange (const QDate & min, const QDate & max)

void setGridVisible (bool show)

void setNavigationBarVisible (bool visible)

void setSelectedDate (const QDate & date)

لتحديد التاريخ المساوي للوسيط.date

void showNextMonth ()

إظهار صفحة الشهر التي تلي صفحة الشهر المحددة الحالية.

void showNextYear ()

إظهار صفحة السنة التي تلي صفحة السنة المحددة الحالية.

void showPreviousMonth ()

void showPreviousYear ()

void showSelectedDate ()

يظهر الصفحة التي حدد فيها التاريخ , يمكن أن نتصفح الأشهر و السنوات لكن من دون تحديد تاريخ ما , هذا المقبس يظهر الصفحة التي حدد فيها التاريخ.

void showToday ()

يظهر الصفحة التي حدد فيها تاريخ اليوم ,يمكن أن نتصفح الأشهر و السنوات لكن من دون تحديد تاريخ ما , هذا المقبس يظهر الصفحة التي حدد فيها تاريخ اليوم.

أحداث (Signals) الكائن QCalendarWidget

void activated (const QDate & date)

يقدح عند تفعيل تاريخ ما.

void clicked (const QDate & date)

يقدح عند النقر على يوم ما من القسم الخاص بعرض أيام الشهر ,الوسيط date يحوي التاريخ المحدد حاليا.

void currentPageChanged (int year, int month)

يقدح عند تغيير صفحة التاريخ المعروضة حاليا.

void selectionChanged ()

يقدح عند تغيير تاريخ اليوم.

البنية الهرمية للكائن QCalendarWidget

QObject → QWidget → QCalendarWidget

LCD Number (QLCDNumber) كائن إظهار الأعداد بنمط عرض , LCD تستطيع اختيار نظام الأعداد الذي تود إظهاره في الكائن QLCDNumber (Hex , Dec , Oct , Bin).

مناهج الكائن QLCDNumber

bool checkOverflow (double num) const

اختبار العدد الممرر للوسيط num من نمط Double إذا كان مسموح عرضه ضمن الكائن QLCDNumber , يرجع true في حال كان غير مسموح عرض الرقم (عدد خانات الرقم أكبر من عدد خانات الرقم للخاصية digitCount).

bool checkOverflow (int num) const

نفس عمل المنهجية السابقة باستثناء قيمة الوسيط الممرر من نمط.int

int digitCount () const

عدد خانات الرقمية التي يستطيع الكائن عرضها

int intValue () const

يرجع قيمة الكائن QLCDNumber من نمط.int

Mode mode () const

يرجع نمط نظام الأعداد المستخدم للكائن QLCDNumber (أعداد ثنائية , أعداد ثمانية ,أعداد عشرية ,أعداد ست عشرية) .

التعداد QLCDNumber::Mode :

QLCDNumber::Hex	0	نظام الأعداد ست عشري
QLCDNumber::Dec	1	نظام الأعداد عشري
QLCDNumber::Oct	2	نظام الأعداد ثماني
QLCDNumber::Bin	3	نظام الأعداد ثنائي

SegmentStyle segmentStyle () const

ترجع مظهر الكائن الخارجي , انظر الجدول , التعداد QLCDNumber::SegmentStyle :

يظهر العدد مع بروز و يكون لون خط العدد نفس لون الخلفية	**QLCDNumber::Outline**
يظهر العدد مع بروز و يكون لون خط العدد نفس لون الخط في النظام لون النص الإفتراضي (يمكن تغيير بواسطة الصف QPalette	**QLCDNumber::Filled**
يظهر العدد من دون بروز و يكون لون خط العدد نفس لون الخط في النظام يمكن تغيير بواسطة الصفQPalette	**QLCDNumber::Flat**

void setDigitCount (int numDigits)

إعطاء قيمة عدد الخانات الرقمية التي يستطيع الكائن QLCDNumber إظهارها.

void setMode (Mode)

إعطاء قيمة نظام الأعداد للكائن.QLCDNumber

void setSegmentStyle (SegmentStyle)

تغيير شكل المظهر الخارجي للكائن.QLCDNumber

bool smallDecimalPoint () const

يرجع قيمة من نمط bool , true إذا كانت الفاصلة لا تأخذ حجم عدد false,إذا كانت الفاصلة تأخذ حجم عدد ,أنظر شرح المقبس setSmallDecimalPoint (bool)

double value () const

يرجع قيمة الكائن QLCDNumber بنمط.Double

مقابس (Slots) الكائن QLCDNumber

void display (const QString & str)

لإظهار النص الممرر للوسيط , str تستخدم لإظهار أعداد من نظام ستة عشري , عشري , ثماني , ثنائي.

void display (double num)

لإظهار رقم من نمط Double الممرر للوسيط. num

void display (int num)

لإظهار رقم من نمط int الممرر للوسيط. num

void setBinMode ()

وضع نمط نظام الأعداد على النظام الثنائي.

void setDecMode ()

وضع نمط نظام الأعداد على النظام العشري.

void setHexMode ()

وضع نمط نظام الأعداد على النظام الست عشري.

void setOctMode ()

وضع نمط نظام الأعداد على النظام الثماني.

void setSmallDecimalPoint (bool)

إذا مررنا true لوسيط المنهج سوف تظهر الفاصلة إن كانت موجودة بين محرفين أي تأخذ فقط حجم الفاصلة (نقطة) أما إذا مررت false سوف تأخذ الفاصلة حجم عدد) إذا كانت الخاصية digitCountقيمتها 4 سوف تنقص قيمتها بمقدار واحد أي تصبح قيمتها 3 .

أحداث (Signals) الكائن QLCDNumber

void overflow ()

يقدح عند إعطاء قيمة للكائن QLCDNumber أكبر من القيمة المسموح إظهارها

البنية الهرمية للكائن: QLCDNumber

QObject → QWidget → QFrame → QLCDNumber

Progress Bar (QProgressBar) : كائن شريط التقدم , غنيّ عن التعريف ,يستخدم في إظهار مقدار عمل حالة معية كتحميل ملف من الإنترنت يظهر لنا حالة تقدم التحميل ,انظر الصورة:

أهم مناهجه:

Qt::Alignment alignment () const

يرجع إزاحة الكائن.QProgressBar

QString format () const

يرجع تنسيق نص حالة التقدم , التنسيق الإفتراضي , %p% إذا أردنا تعديل التنسيق نعدل فقط الرمز الأخير, مثال.%p km

bool invertedAppearance ()

إذا كانت القيمة المعادة true يعني أن الكائن معكوس المظهر (إذا كان اتجاه مظهر حالة التقدم من اليسار إلى اليمين سيصبح من اليمين إلى اليسار) , قيمته الافتراضية.false

bool isTextVisible () const

يرجع true في حال كان نص حالة التقدم ظاهرة , يرجع false إذا كان غير ظاهر نص حالة التقدم.

int maximum () const

يرجع أكبر قيمة يستطيع شريط التقدم وصول قيمة حالة تقدمه لها.

int minimum () const

يرجع نقطة البداية لشريط حالة التقدم (أصغر قيمة له).

Qt::Orientation orientation () const

يرجع اتجاه كائن.QProgressBar

void setAlignment (Qt::Alignment alignment)

void setFormat (const QString & format)

void setInvertedAppearance (bool invert)

void setTextDirection (QProgressBar::Direction textDirection)

وضع اتجاه نص حالة التقدم - QProgressBar::TopToBottom,
QProgressBar::BottomToTop, لا يعمل على نظام Win.

void setTextVisible (bool visible)

virtual QString text () const

QProgressBar::Direction textDirection ()

QProgressBar::Direction textDirection () const

int value () const

يرجع قيمة حالة التقدم بنمط رقم صحيح int.

مقابس QProgressBar (Slots) الكائن:

void reset ()

إعادة حالة الكائن لوضعه الإفتراضي إظهار حالة تقدمه على القيمة الصغرى و عدم إظهار نص حالة التقدم إلا عندما تزداد قيمة تقدمه.

void setMaximum (int maximum)

void setMinimum (int minimum)

void setOrientation (Qt::Orientation)

void setRange (int minimum, int maximum)

void setValue (int value)

إعطاء قيمة التقدم يجب أن تكون القيمة بين أكبر قيمة لحالة التقدم الكائن QProgressBar و أصغر قيمة لحالة تقدمه .

أحداث (Signals) الكائن: QProgressBar

void valueChanged (int value)

يقدح عند تغيير قيمة حالة التقدم , الوسيط value هو قيمة التقدم الحالية.

البنية الهرمية للكائن:QProgressBar

QObject → QWidget → QProgressBar

Line كائن الخط ,كائن يرسم شكل خط على النافذة.

له خاصية orientationتأخذ قيمة أفقي أو عمودي.(Horizontal - Vertical)

البنية الهرمية للكائن: Line

QObject → QWidget → QFrame → Line

QWebView Web View كائن تصفح الويب :

يعرض صفحات الويب , نستطيع تصفح موقع ويب بواسطة الكائن QWebView التابع لحزمة webkitمن شركة نوكيا الخاص باللغة Qt عن طريق تمرير عنوان الموقع للكائن QWebView . سوف نتكلم بالتفصيل عن هذا الكائن في فصل برمجة الشبكات.

إذا كنا نريد استخدام الكائن QWebView أو أي كائن تابع لحزمة webkit يجب إضافة الحزمة للتطبيق لكي نستطيع استخدام صفوف هذه الحزمة , نستطيع إضافتها بواسطة الرمّاز التالي:

QT += webkit

نكتب هذه الرمّاز داخل ملف المشروع. appName.pro

أهم مناهج الكائن: QWebView

bool findText (const QString & subString, QWebPage::FindFlags options = 0)

يرجع القيمة true إذا وجد النص الممرر للوسيط subString ضمن صفحة الويب الحالية ويحدد على النص في الصفحة في حال إيجاده , وفي حال عدم إيجاد النص يرجع القيمة . false
أما بالنسبة للوسيط options فانظر إلى الجدول الخاص بالتعدادQWebPage::FindFlag :

QWebPage::FindBackward	بدء البحث من أخر صفحة الويب إلى أعلى صفحة الويب
QWebPage::FindCaseSensitively	البحث مع مراعاة حالة الأحرف (كبيرة -صغيرة)
QWebPage::FindWrapsAroundDocument	البحث من بداية صفحة الويب إلى نهايتها عند الوصول إلى نهاية الصفحة يضع مؤشر البحث في بداية صفحة الويب
QWebPage::HighlightAllOccurrences	تحديد كافة النصوص المشابهة لنص البحث(subString)

QWebHistory * history () const

يرجع مؤشر كائن محفوظات الويب (صفحات الويب المزارة مسبقا).

QIcon icon () const

يرجع كائن شعار صفحة الويب إن وجد و إلا يرجع. 0

void load (const QUrl & url)

تحميل صفحة ويب url , عنوان صفحة الويب من نمط: QUrl

```
webView->load( QUrl ( “http://www.google.com” ) );
```

QWebPage * page () const

يرجع مؤشر كائن صفحة الويب المعروضة ضمن الكائن QWebView حاليا.

QAction * pageAction (QWebPage::WebAction action) const

يرجع مؤشر الكائن QAction كائن قائمة الأفعال – الأوامر- سنتكلم عنه داخل الصفحات القادمة من هذا الفصل , تنفيذ أمر للكائن QWebView كأمر إعادة التحميل أو النسخ أو .... سنتكلم عنه بالتفصيل في فصل الشبكات.

QString selectedText () const

يرجع النص المحدد ضمن صفحة الويب الحالية.

void setPage (QWebPage * page)

لوضع كائن صفحة ويب بواسطة إعطاء قيمة الوسيط page مؤشر كائن الصفحة.

void setZoomFactor (qreal factor)

لتكبير أو تصغير صفحة الويب المعروضة بواسطة تمرير قيمة مقدار التكبير أو التصغير للوسيط factor (الوسيط factor من نمط حقيقي)

QWebSettings * settings () const

يرجع مؤشر كائن الإعدادات , هذا الكائن لحفظ جميع الإعدادات الخاصة بالصفحة من تنسيق خط و خلفية الصفحة إلى إعدادات cache الخاصة بالصفحة المحملة.

QString title () const

يرجع نص عنوان صفحة الويب المعروضة حاليا.

QUrl url () const

يرجع عنوان موقع الصفحة المعروضة حاليا.

qreal zoomFactor () const

يرجع قيمة مقدار التكبير أو التصغير.

void setHtml (const QString & html, const QUrl & baseUrl = QUrl (())

وضع رمّاز HTML وعرض تفسيره ضمن الكائن. QWebView

أهم مقابس (Slots) الكائن QWebView

void back ()

الرجوع إلى الصفحة السابقة.

void forward ()

الإنتقال إلى الأمام صفحة.

void print (QPrinter * printer) const

طباعة صفحة الويب المعروضة , تستطيع طباعتها بواسطة تمرير مؤشر كائن الطباعة QPrinter* .

void reload ()

إعادة تحميل الصفحة المفتوحة.

void stop ()

إيقاف تحميل الصفحة في حالة تحميل صفحة ما.

أحداث (Signals) الكائن QWebView

void iconChanged ()

يقدح عند تغيير أيقونة موقع الويب.

void linkClicked (const QUrl & url)

يقدح عند النقر على رابط ما ضمن الصفحة الحالية , الوسيط url هو عنوان الرابط.

void loadFinished (bool ok)

يقدح عند الانتهاء من تحميل صفحة ويب ,الوسيط ok تكون قيمته true إذا قد حملت الصفحة من دون أي خطأ و إلا ستكون قيمته.false

void loadProgress (int progress)

يقدح عند إنهاء تحميل أي عنصر ضمن صفحة الويب التي تكون قيد التحميل , الوسيط progressمن نمط صحيح يأخذ القيمة التي يكون مجالها 100 – 0 لعرض قيمة حالة التحميل.

void loadStarted ()

يقدح عند بداية تحميل صفحة ويب.

void statusBarMessage (const QString & text)

يقدح عند تغيير نص شريط الحالة بواسطة صفحة الويب.

void titleChanged (const QString & title)

يقدح عند تغيير نص عنوان الصفحة بواسطة صفحة الويب.

void urlChanged (const QUrl & url)

يقدح عند تغيير عنوان الموقع , الوسيط url يحمل قيمة عنوان الموقع الجديد.

البنية الهرمية للكائن: QWebView

QObject → QWidget → QWebView

انتهينا الآن من شرح الكائنات الموجودة داخل صندوق الأدوات, آسف !أعلم أنه يوجد بعض الملل في شرح كائنات صندوق الأدوات لكننا كنا مجبرين بذلك من أجل الاختصار , إذا كنا نريد شرح الأدوات بغير هذه الطريقة لكنا قد ألّفنا كتاب خاص بشرح كائنات صندوق الأدوات ضمن Qt .

❖ الحافظة :

من أجل الوصول إلى حافظة النظام و نسخ بيانات إليها و لصقها فيها يتوجب استخدام الصف "QClibboard" , من أجل استخدام الصف "QClipboard" يتوجب أن نسند لكائنه المؤشر العام لحافظة التطبيق وذلك باستخدام المنهج الساكن :

QClipboard* QApplication::clipboard()

و من أجل اختبار نوع البيانات الموجودة ضمن الحافظة يتوجب أن نسند لمؤشر الكائن "QMimeData" مؤشر لـ لبيانات الحافظة , وذلك كالآتي :

const QMimeData* mimeData = clipboard->mimeData()

الصف "QMimeData" خاص باحتواء بيانات و تحليلها لمعرفة معلومات عن أنماطها , يملك هذا الصف عدة مناهج مفيدة لمعرفة أنواع البيانات كـ :

bool QMimeData::hasText () const

يرجع "true" في حال كان نوع البيانات المسند له نص بسيط

bool QMimeData::hasHtml () const

يرجع "true" في حال كان نوع البيانات المسند له تنسيق "html"

bool QMimeData::hasImage () const

يرجع "true" في حال كان نوع البينات المسند له صورة

مثال توضيحيّ , يحوي هذا التطبيق على زر يدعى "paste" خاص بلصق محتوى حافظة النظام ضمن كائن اللافتة "QLabel" , أنشأ تطبيق "Qt Gui Application" مشتق صفه الأساس من الصف "QMainWindow" , ضع كائن لافتة و كائن زر على واجهة المستخدم , ثمّ ضمّن ملفات الترويسة التالية ضمن الملف "mainwindow.h" :

```
#include <QClipboard>
#include <QMimeData>
```

الآن صرّح عن مؤشر لكائن الحافظة و مؤشر آخر ثابت لحاوية أنواع البيانات :

```
QClipboard *clipboard ;
const QMimeData *mimeData ;
```

ضمن بناء الصف "mainwindow" أضف الرمّاز التالي :

```
clipboard = QApplication::clipboard();
mimeData = clipboard->mimeData();
```

ضمن معالج حدث النقر على الزر أصف الرمّاز الخاص بـ لصق محتوة الحافظة بعد معرفة توع بياناتها :

```
if (mimeData->hasHtml()){
   ui->label->setText(mimeData->html());
   ui->label->setTextFormat(Qt::RichText);
   }
else if(mimeData->hasText()){
   ui->label->setText(mimeData->text());
   ui->label->setTextFormat(Qt::PlainText);
}
```

```
else if (mimeData->hasImage()){
  ui->label
  ->setPixmap(qvariant_cast<QPixmap>(mimeData
  ->imageData()));
}
```

المنهج "qvariant_cast" يستخدم من اجل التحويل بين أنماط "Qt" المعرّفة , نفذ التطبيق و اختبر النتيجة , من أجل نسخ بيانات إلى حافظة النظام يتوجب أن أحد المناهج التالي :

void QClipboard::setImage (const QImage & image, Mode mode = Clipboard)

من أجل نسخ صورة إلى الحافظة

void QClipboard::setPixmap (const QPixmap & pixmap, Mode mode = Clipboard)

من اجل نسخ صورة من نمط "QPixmap" إلى الحافظة

void QClipboard::setText (const QString & text, Mode mode = Clipboard)

من اجل نسخ نص إلى الحافظة

void QClipboard::setMimeData (QMimeData * src, Mode mode = Clipboard)

من أجل نسخ بيانات من نمط "QMimeData"

• كائن القائمة (QMenu) و كائن الفعل (QAction) :

سنتكلم في هذه الفقرة عن كائن القائمة و إنشاء عناصر لها بواسطة الكائن QAction بحيث كل عنصر ينفذ أمر ما ,و وضع أيقونة لعنصر قائمة و الاستجابة لأحداث القائمة (النقر بزر

الفأرة و تحريك مؤشر الفأرة فوق عنصر قائمة و تعيين التركيز على عنصر) و كيفية استخدامها في QMenuBar شريط قائمة , و قائمة السياق Context Menu التي تستجيب للحدث QContextMenuEvent الذي يقدح عند النقر بزر الفأرة الأيمن أو زر القائمة الموجود داخل لوحة المفاتيح فتظهر القائمة بالموقع الذي نحدده ,و القائمة المنبثقة Popup Menu قائمة اختيارية ذات مستقبل (Slot) خاص بنا يقدح مستجيبا للحدث (Signal) customContextMenuRequested(QPoint) . و سنتكلم أيضا عن شريط الأدوات و إضافة أداة بواسطة الكائن QAction و عن شريط الحالة و كيفية استخدامه.

الكائن QAction كائن الأوامر (الأفعال) يحوي أوامر نعينها نحن , كل QAction له نص و أيقونة و اختصار,

يقدح الأمر (الفعل) إما من خلال النقر عليه أو من خلال استدعائه أو الضغط على الأمر بمفتاح Enterأو الضغط على اختصاره , يظهر بشكل مرئي إن كان قد أضفته لقائمة أو شريط أدوات أو الخ , ..نستطيع إضافة الكائن QAction إلى جميع كائنات واجهات المستخدم و نستطيع إظهاره كقائمة سياق بواسطة الخاصيّة contextMenuPolicy تضع قيمتها Qt::ActionsContextMenuحيث يأخذ الكائن widget بشكل تلقائي جميع الأفعال المضافة له و بالتالي تظهر قائمة السياق على الكائن عند النقر بزر الفأرة الأيمن على الكائن widgetو سنرى بعد قليل رمّاز لإنشاء أفعال و ربطها بكائن و كيفية إظهار قائمة سياق.

يوجد أيضا كائن يدعى QActionGroup يستخدم لاحتواء مجموعة من الأفعال داخله (قائمة أفعال - مجموعة من الكائنQAction).

كائن القائمة QMenu يستخدم لإنشاء قائمة تحتوي على عناصر تنفذ أفعال ما.

مناهج الكائن QMenu

QAction * actionAt (const QPoint & pt) const

يرجع مؤشر كائن عنصر القائمة (كائن الفعلQAction) ذو الموقع.pt

QRect actionGeometry (QAction * act) const

يرجع كائن مستطيل QRect يحوي على موقع و أبعاد عنصر القائمة (الفعل). *act

QAction * activeAction () const

يرجع مؤشر كائن الفعل المركز عليه حاليا.

QAction * addAction (const QString & text)

إضافة فعل يحوي فقط على نص , و يرجع مؤشر كائن الفعل المضاف.

QAction * addAction (const QIcon & icon, const QString & text)

إضافة فعل يحوي على نص و شعار ,و يرجع مؤشر كائن الفعل المضاف.

QAction * addAction (const QString & text, const QObject * receiver, const char * member, const QKeySequence & shortcut = 0)

إضافة فعل يحوي على نص و أمر (رد فعل) , الوسيط *receiver هو مؤشر لكائن المستقبل لرد الفعل (أي الذي سيقع عليه الفعل) و أما الوسيط *member من نمط مؤشر محرف تضع هنا منهجية المقبس(المستقبل Slot) الذي يوجد فيه الرمّاز الخاص بتنفيذ أمر ما ,و يرجع هذا المنهج مؤشر كائن الفعل المضاف.

QAction * addAction (const QIcon & icon, const QString & text, const QObject * receiver, const char * member, const QKeySequence & shortcut = 0)

إضافة فعل يحوي على نص و شعار و أمر (ردة فعل) , الوسيط *receiver هو مؤشر لكائن المستقبل لردة الفعل (أي الذي سيقع عليه الفعل) و أما الوسيط *member من نمط مؤشر محرف تضع هنا منهجية المقبس(المستقبلSlot-) الذي يوجد فيه الرمّاز الخاص بتنفيذ أمر ما , و يرجع هذا المنهج مؤشر كائن الفعل المضاف.

void addAction (QAction * action)

إضافة مؤشر لكائن فعل منشأ مسبقا.

QAction * addMenu (QMenu * menu)

لإضافة قائمة فرعية للقائمة الرئيسية ,يضيف القائمة الممرر مؤشرها للوسيط , و إرجاع مؤشر كائن الفعل (العنصر المضاف)

QMenu * addMenu (const QString & title)

إضافة قائمة فرعية النص الظاهر لها هو , title يرجع مؤشر لكائن القائمة المضاف.

QMenu * addMenu (const QIcon & icon, const QString & title)

إضافة قائمة فرعية النص الظاهر لها هو title تحوي شعار ,icon يرجع مؤشر لكائن القائمة المضاف.

QAction * addSeparator ()

إضافة فاصل بين عناصر القائمة , يرجع مؤشر لكائن الفعل (العنصر المضاف)

void clear ()

مسح جميع عناصر القائمة.

QAction * exec ()

إظهار القائمة.

QIcon icon () const

يرجع شعار القائمة.

QAction * insertMenu (QAction * before, QMenu * menu)

إضافة قائمة فرعية قبل الفعل , *before يرجع مؤشر فعل القائمة (عنصر القائمة الفرعية)

QAction * insertSeparator (QAction * before)

إضافة فاصل قبل الفعل , *before يرجع مؤشر فعل العنصر المضاف.

bool isEmpty () const

يرجع true إذا كان كائن القائمة خالي , يرجع false في حال العكس.

QAction * menuAction () const

يرجع مؤشر كائن القائمة (عنصر القائمة الرئيسي – رأس القائمة – ك قائمة ملف تحتها تنسدل فتح و حفظ.. الخ..)

void popup (const QPoint & p, QAction * atAction = 0)

عرض القائمة في الموقع. p

void setActiveAction (QAction * act)

تحديد التركيز على العنصر.*act

void setIcon (const QIcon & icon)

وضع شعار للقائمة.

void setTitle (const QString & title)

تغيير عنوان القائمة.

QString title () const

يرجع نص العنوان القائمة.

الكائن QMenuBar : شريط القائمة يستخدم لاحتواء قوائم . كقائمة ملف و تحرير و عرض الخ ..في نفس الموقع الأفقي لهم , دعنا نكتب رمّاز لإنشاء شريط قائمة و إضافة قائمة Fileو قائمة , Help قائمة File تحوي العناصر (الأفعال) Open و , Exit و القائمة Help تحوي العنصر (الفعل) , About

اكتب الرمّاز التالي:

```
QMenuBar* menB = new QMenuBar(this);
QMenu* menFile= new QMenu("&File");
QMenu* menHelp= new QMenu("&Help");
QAction* actExit= new QAction(0);
QAction* actAbout= new QAction(0);
actExit->setShortcut( QKeySequence(“Ctrl+X”) );
actExit->setText("E&xit");
actAbout->setText("&About");
menFile->addAction("Open");
menFile->addSeparator();
menFile->addAction(actExit);
menHelp->addAction(actAbout);
menB->addMenu(menFile);
menB->addMenu(menHelp);
connect(actExit,SIGNAL(triggered()),this,SLOT(close()));
connect(actAbout,SIGNAL(triggered()),this,SLOT(show_MSG()));
connect(actAbout,SIGNAL(hovered()),this,SLOT(msgStatus()));
```

لا تنسى إضافة ملفات الترويسة التالية :

```
#include <QMenuBar>

#include <QMenu>

#include <QAction>
```

إذا لاحظت الحدث ()hovered يقدح عند التركيز على عنصر إن كان بواسطة الفأرة أو لوحة المفاتيح.

أما الحدث ()triggered يقدح عند النقر على عنصر قائمة أو الضغط بمفتاح Enter عليه أو الضغط على اختصار عنصر القائمة من لوحة المفاتيح.

أضفنا اختصار لعنصر القائمة Exit بواسطة الرمّاز

```
actExit->setShortcut( QKeySequence("Ctrl+X") );
```

المنهج setShortcut(QKeySequence) يمرر وسيط الكائن الخاص بأزرار الاختصار التابعة للوحة المفاتيح , مررنا للكائن QKeySequence النص Ctrl+X أي عند الضغط على مفتاح التحكم Control و مفتاح المحرف X ينفذ أمر الخروج.

لننتقل إلى قائمة السياق. (Context Menu)

ذكرنا سابقا عن قائمة السياق إنها تظهر عند الضغط بزر الفأرة الأيمن أو زر القائمة من لوحة المفاتيح ولها حدث خاص بها يدعى

```
contextMenuEvent(QContextMenuEvent *event)
```

و يحدد موقع إظهار القائمة بواسطة المنهجية event->globalPos() التي ترجع مؤشر لكائن النقطة.

الحدث contextMenuEvent معرف بأنه virtual و يعرف في قسم protected داخل ملف الترويسة,

الحدث contextMenuEvent موجود داخل الصف, QWidget يقبل إعادة التحقيق , يجب إعادة تحقيقه لتنفيذ الأوامر الخاصة بنا ,له الوسيط , QContextMenuEvent يحوي منهج يدعى pos() يرجع كائن نقطة قيمته موقع مؤشر الفأرة بالنسبة للكائن widget و يحوي منهج globalPos() يرجع كائن نقطة قيمته موقع مؤشر الفأرة بالنسبة للشاشة.

دعنا نكتب رمّاز بسيط لإنشاء قائمة سياق , اتبع الخطوات التالية:

أولا : ننشئ مشروع Qt Gui Application حيث يكون الصف الأساسي. QWidget

ثانيا : أضف ضمن ملف الترويسة widget.h الكائن الخاص بوسيط حدث قائمة السياق :

```
#include < QContextEventMenu>
```

ثالثا : عرف في ملف الترويسة ضمن القسم المحمي protected الحدث contextMenuEventكالتالي

```
protected:
void contextMenuEvent(QContextMenuEvent *  event);
```

رابعا : نحقق تعريفه داخل ملف التحقيق:

```
void  widget::contextMenuEvent(QContextMenuEvent *  event)
{
QMenu menu;
menu.addAction(“&Copy”);
menu.addAction(“&Paste”);
menu.addSeparator();
menu.addAction(“E&xit”);
menu.exec( event.globalPos() );
}
```

الآن عند النقر بزر الفأرة الأيمن أو لواسطة زر القائمة من لوحة المفاتيح سوف تظهر القائمة , يوجد داخلها ثلاث عناصر و فاصل .

نستطيع أيضا إظهار قائمة السياق بواسطة الحدث mousePressEvent(QMouseEvent*)و الحدث mouseReleaseEvent(QMouseEvent*)لكن بعد أن تعدل الخاصية contextMenuPolicyللكائن widget لتصبح قيمتها , Qt::PreventContextMenu
انظر الرمّاز:

```
void  widget::mousePressEvent(QMouseEvent *   event)

{

   if( event->button()  &  Qt::RightButton )  {

   QMenu  menu(this);

   menu.addAction(actExit);

   menu.exec( event->globalPos() );

}

}
```

الحدث mousePressEvent موجود داخل الكائن QWidget و هو معرف بأنه محمي (protected)و افتراضي ,, (virtual) يستدعى عند النقر بأي زر من أزرار الفأرة.

انتهينا الآن من قائمة السياق لننتقل إلى النوع الأخير من القوائم وهو القائمة المنبثقة popup Menu.

ذكرنا سابقا أن القائمة المنبثقة هي قائمة اختيارية ضمن Qt (ذو مستقبل خاص بنا) لنرى الرمّاز التالي:

أضف رمّاز اتصال الإشارة بالمستقبل ببناء النافذة

```
connect(this,SIGNAL(customContextMenuRequested(QPoint)),this,SLO
T(customContextMenuRequested(QPoint)));
```

– Slot customContextMenuRequested(const QPoint &pos)عرّف المستقبل ضمن ملف الترويسة داخل القسم , Slot ثم حقق المستقبل بكتابة الرمّاز التالي:

```
void widget::customContextMenuRequested(const QPoint &pos)
{
   QMenu men;
   men.addAction("E&xit");
   connect( men.actions().at(0) ,SIGNAL(triggered()),this,SLOT(close()));
```

```
    men.addSeparator();
    men.exec( mapToGlobal (pos));
}
```

المنهج mapToGlobal(const QPoint &pos) يمرر لوسيطه مرجع كائن نقطة بالنسبة للكائن widget ليرجع كائن نقطة تحوي موقع النقطة الممررة للوسيط ولكن بالنسبة للشاشة Screen .

انتهينا من جميع أنواع القوائم لننتقل إلى شريط الأدوات و شريط الحالة.

شريط الأدوات QToolBar : يستخدم لاحتواء أوامر - أدوات تحكم تكون اختصار لما تحويه القوائم أو لتنفيذ أمر ما بحيث يكون الأمر كثير الاستخدام وبالتالي يكون سريع الوصول إليه من قبل المستخدم , تستطيع الإضافة عليه كائنات واجهة مستخدم رسومية ككائن الزر أو كائن إدخال النص بواسطة المنهجية QToolBar::addWidget(QWidget*) و تستطيع إضافة أفعال مباشر عليه بواسطة المنهجية , QToolBar::addAction(QAction*) له عدة خصائص أهمها

بتمرير true يعطي السماحية للكائن بالحركة	QToolBar::setMovable(bool)
بتمرير true يعطي السماحية للكائن بالعوم فوق الكائن widget(يصبح منفصل عن الكائن الحاوي له)	QToolBar::setFloating(bool)
الجهات المسموحة للكائن بالرصف عليها	QToolBar::setAllowedAreas(Qt::ToolBarAreas)
إضافة فاصل بين الأوامر و الكائناتwidgets	QToolBar::addSeparator()

رمّاز بسيط لإنشاء كائن QToolBar يوجد عليه أمر About و أمر Exit و كائن QLineEdit :

```
QToolBar*  toolbar = new QToolBar( this );

toolbar->addAction( "&About" , this , SLOT(show_msg()) );

toolbar->addAction( "E&xit" , this , SLOT(close()) );

toolbar->addSeparator();
```

toolbar->addWidget(new QLineEdit(“Hello”));

الكائن QToolBar مشتق من الصف.QWidget

شريط الحالة QStatusBar شريط أفقي يظهر معلومات (تلميحات خاصة بكائن ما أو توجيها بسيطة) يمكن الإضافة عليه كائن , widget أهم مستقبل (Slot)

QStatusBar::showMessage(const QString &info , int timeout=0)

يظهر رسالة داخله مدة إظهار تلك الرسالة النصية هي. timeout

المنهج ()QString currentMessage يرجع قيمة رسالة المعلومات الحالية (المعروضة ضمن شريك الحالة)

انتهينا من شرح كائن شريط الأدوات و كائن شريط الحالة , لنأخذ فاصل بالتكلم عن تنسيق و تجميل كائنات واجهة المستخدم الرسومية بواسطة المنهجية setStyleSheet(const QString &css) .

- **تحسين مظهر كائنات واجهة المستخدم الرسومية من خلال تنسيق خطّاط CSS** :

لدى Qt قدرة على التعامل مع كائنات واجهة المستخدم الرسومية و كأنها عناصر صفحة ويب HTMLهنا سنستفاد من هذه الميزة باستخدام خطّاط CSS لتنسيق كائنات Qt الرسومية و أيضا لديها إمكانية معالجة و تنفيذ خطّاطات جافا سكريبت Java Script سنتكلم عنه لاحقا , أما الآن دعنا نرى كيف تتعامل Qt مع خطّاط. CSS

المنهج setStyleSheet موجود هذا المنهج داخل الكائن QWidget أي جميع كائنات واجهة المستخدم تحوي هذا المنهج لأنها جميعا ترث من الكائن , QWidget باختصار نمرر للمنهج setStyleSheet قيمة نصية تحوي الخطّاط , CSS خطّاط CSS القياسي لكن مع مراعاة عدة أمور:

أولا العناصر هنا هي أسماء الأدوات , مثال نريد كتابة خطّاط CSS خاص بكائن الزر QPushButtonيجب أن نكتب اسم العنصر

QPushButton {//here write css script for QPushButton Object }

نستطيع إضافة خطّاط CSS يتفاعل مع أحداث المستخدم من تحريك مؤشر الفأرة فوق كائن ما إلى الضغط على كائن ما...

عندما نكتب خطّاط CSS لكائن ما مثال كائن الزر فإن الخطاط سوف يؤثر فقط على كائن الزر الذي قد كتب فيه الخطّاط أي داخل منهجيته , setStyleSheet أمّا إذا أردنا أن نؤثر على كافة كائنات الأزرار الموجودة داخل النافذة (الحاوي) الحالي نستطيع من خلال كتابة خطّاط CSS داخل المنهجية setStyleSheet التابعة للمؤشر this أي الصف الحالي.

```
this->setStyleSheet("QPushButton{color : red ;}");
```

أما من أجل كائن رز واحد نعدل قيمة المنهجية setStyleSheet الخاصة بكائن الزر بشكل مباشر:

```
pushButton->setStyleSheet("QPushButton{color : red ;}");
```

وتستطيع أيضا كتابته بالصيغة التالية:

```
pushButton->setStyleSheet("color : red ;");
```

أما من أجل جميع كائنات الزر الموجودة داخل التطبيق نستخدم مؤشر كائن التطبيق:qApp

```
qApp->setStyleSheet("QPushButton{color : red ;}");
```

كيف نستطيع أن نتفاعل مع حدث ما يقدح من Qt و تنفيذ خطّاط CSS عند قدحه أو تنفيذ خطّاطا عندما تكون خاصيّة ما مفعّلة , لديها الإمكانية بواسطة كتابة الحدث بعد اسم الكائن لكن بوجود محرف النقطتين العمودية (:) بينهما (بين اسم الكائن و حدثه) , انظر الرمّاز لتغيير لون خط النص (اللون الأمامي للكائن) الكائن QLineEdit عن تحريك مؤشر الفأرة فوقه و يعود إلى لونه الإفتراضي عند خروج مؤشر الفأرة من حدود الكائن: QLineEdit

```
lineEdit->setStyleSheet("QLineEdit:hover{color : blue ;}");
```

سنعرض الآن أهم الأحداث المتفاعلة مع خطّاط , CSS انظر الجدول:

حدث تحريك مؤشر الفأرة فوق الكائن	**ObjectName:hover**
حدث النقر بزر الفأرة على الكائن	**ObjectName:pressed**
حدث نقل التركيز إلى الكائن	**ObjectName:focus**
حدث تفعيل الكائن	**ObjectName:active**
حدث وضع علامة اختيار للكائن	**ObjectName:checked**
حدث إزالة علامة الاختيار للكائن	**ObjectName:unchecked**

إذا أردنا أن نصل إلى كائن موجود ضمن كائن من خلال خطّاط CSS نكتب بعد اسم الكائن محرفي نقطتين فوقيتين (::) ثم اسم الكائن الداخلي , انظر الجدول يعرض أهم الكائنات الداخلية:

كائن السهم العلوي (كالسهم في كائن QscrollBar)	ObjectName::up-arrow
كائن السهم السفلي	ObjectName::down-arrow
كائن السهم اليميني	ObjectName::right-arrow
كائن السهم اليساري	ObjectName::left-arrow
كائن الزر العلوي (كالزر الموجود في الكائن QScrollBar- QspinBox)	ObjectName::up-button
كائن الزر السفلي	ObjectName::down-button
كائن الزر اليميني	ObjectName::right-button
كائن الزر اليساري	ObjectName::left-button
كائن شريط التبويب	ObjectName::tab-bar
كائن اللون الأمامي شريط التقدم	ObjectName::chunk
كائن زر الإغلاق (كزر إغلاق الكائن QdockWidget)	ObjectName::close-button
كائن التبويب الموجود داخل شريط التبويب للصفQTabWidget	ObjectName::tab

أما إذا كنا نريد أن ننفذ خطّاط CSS للتأثير بالكائن عندما تكون خاصيّة ما له ذو قيمة معينة (حالة شرطية) فنكتب التالي:

```
lineEdit->setStyleSheet("QLineEdit[readOnly='true']{border-color: red blue green black;}");
```

سينفذ الخطّاط عندما تكون خاصية للقراءة فقط مفعّلة لكائن. QLineEdit

نكتب من أجل اختبار شرطي لخاصيّة ما اسم الكائن يليه الرمز [] بداخله شرط الاختبار.

ننصح أن تستخدم ملف مورد يكون موجود بداخله ملف خطّاط CSS لجميع كائنات المستخدم الرسومية و استيراده بواسطة رمّاز لقراءة ملف حيث يكون هذا الملف المقروء هو ملف خطّاط CSS , الخطّاط الذي قرءناه من الملف نضعه قيمة للمنهج setStyleSheet التابع لمؤشر التطبيق. qApp

الآن بعد أن انتهينا من تفاعل شكل الكائنات الرسومية من خلال خطّاط CSS سوف ننتقل للتكلم عن كائن مفيد جدا يدعى المتمم. QCompleter

- المتمم QCompleter :

يستخدم المتمم QCompleter لتكملة نص ما عند كتابة جزء منه , أي عند كتابة محارف معينة داخل كائن إدخال ما سوف يبحث المتمم داخل مصفوفة من الكلمات عن كلمة موجود داخلها هذه المحارف ثم تهيئتها في الذاكرة و من ثم عرضها على الشاشة إما بطريقة قائمة أو بطريقة إظهار الكلمة التي قد وجدها مع تحديد نصها داخل كائن الإدخال , مثال على المتمم عندما تريد أن تبحث في موقع الغوغل google.com عن موضوع ما , اكتب محرف (أي محرف) على فرض محرف الألف سوف يظهر قائمة منبثقة من كائن الإدخال يوجد داخله مجموعة كلمات على شكل عناصر قائمة تبدأ بحرف الألف (طبعا سوف يظهر الكلمات الأكثر توقعا لدى الغوغل لأنه يستخدم الذكاء الصنعي بالبحث عن الكلمات الأكثر استخدام من قبل المستخدمين) في الغوغل يستخدم تقنية أجاكس Ajax لإظهار قائمة المتمم أما هنا نحن نستخدم صف المتمم من مكتبات. Qt QCompleter

عند استخدام المتمم يجب علينا تضمين ملف الترويسة الخاصة به:

```
#include <QCompleter>
```

أهم مناهج الكائن QCompleter :

```
QCompleter ( const QStringList & list, QObject * parent = 0 )
```

بناء الكائن , يحوي وسيط list تضاف من خلاله مصفوفة من الكلمات للبحث ضمنها عن الكلمة المشابهة للمحارف المدخلة بواسطة كائن الإدخال المرتبط به كائن المتمم.

```
Qt::CaseSensitivity  caseSensitivity () const
```

يرجع حالة حساسية الأحرف (تحسس الأحرف الكبيرة من الصغيرة أو عدم تحسس الأحرف) المدخلة لمقارنتها بالكلمات الموجودة داخل كائن المتمم , انظر الجدول , التعداد Qt::CaseSensitivity :

عدم تحسس الأحرف	Qt::CaseInsensitive
تحسس الأحرف الكبيرة من الأحرف الصغيرة	Qt::CaseSensitive

```
int  completionCount () const
```

يرجع عدد الكلمات التي تكون داخل المصفوفة التي تحوي كلمات للكائنQCompleter المهيئة لتتمة المحارف المدخلة بعد مطابقة التشابه.

CompletionMode completionMode () const

يرجع نمط إظهار الكلمة (الكلمات) المشابهة للأحرف المدخلة إلى المتمم , انظر الجدول , التعداد QCompleter::CompletionMode :

إظهار الكلمات المشابهة للمحارف المدخلة الموجودة داخل مصفوفة الكلمات التابعة لكائن المتمم على شكل قائمة منبثقة	**QCompleter::PopupCompletion**
إظهار الكلمة المشابهة للمحارف المدخلة الموجودة داخل مصفوفة الكلمات التابعة لكائن المتمم على شكل كلمة واحدة في نفس كائن الإدخال المرتبط المتمم به مع تحديد محارفها الغير مكتوبة بعد.	**QCompleter::InlineCompletion**
إظهار جميع الكلمات (مصفوفة الكلمات) الموجودة داخل كائن المتمم على شكل قائمة منبثقة	**QCompleter::UnfilteredPopupCompletion**

QAbstractItemModel * completionModel () const

يرجع مؤشر الكائن QAbstractItemModel الخاص بتهيئة و إضافة البيانات من ثم إضافتها إلى كائنات العرض ,هذا المنهج لقراءة مصفوفة الكلمات ووضعها في كائن QAbstractItemModelثم إرجاع مؤشره.

QString currentCompletion () const

يرجع نص الكلمة الحالية (التي تماثلت المحارف الأولى لها مع المحارف المدخلة)

int maxVisibleItems () const

يرجع أكبر قيمة لعدد العناصر التي يستطيع إظهارها الكائن QCompleter من دون إظهار المنزلقة.

QAbstractItemModel * model () const

يرجع مؤشر الكائن QAbstractItemModel الخاص بتهيئة و إضافة البيانات من ثم إضافتها إلى كائنات العرض ,أي يعمل المنهج على قراءة مصفوفة الكلمات ووضعها في كائن QAbstractItemModelثم إرجاع مؤشره.

void setCaseSensitivity (Qt::CaseSensitivity caseSensitivity)

وضع حالة تحسس للأحرف الكبيرة من الأحرف الصغيرة.

void setCompletionMode (CompletionMode mode)

وضع نمط إظهار الكلمة (الكلمات) المشابهة للأحرف المدخلة إلى المتمم.

void setMaxVisibleItems (int maxItems)

وضع أكبر قيمة لعدد العناصر التي يستطيع إظهارها الكائن QCompleter من دون إظهار المنزلقة.

void setModel (QAbstractItemModel * model)

لوضع نموذج يحوي مصفوفة من الكلمات للكائن , QCompleter النموذج هو كائن QAbstractItemModel .

أهم أحداث الكائن QCompleter

void activated (const QString & text)

يقدح عند اختيار عنصر من القائمة التي تظهر (تنبثق) من الكائن الرسومي المرتبط به كائن المتمم.

void highlighted (const QString & text)

يقدح عند تعيين التركيز (تحديد) عنصر من القائمة التي تنبثق من الكائن الرسومي المرتبط به كائن المتمم.

مثال بسيط عن كائن المتمم , نافذة يوجد فيها أداة إدخال النص QLineEdit عند إدخال محرف أو أكثر سوف يقارنه بقائمة من أسماء الحيوانات الموجودة في مصفوفة الكلمات الخاصة بالمتمم عند تشابه المحارف المدخلة مع أي من أسماء الحيوانات سوف تظهر قائمة المتمم تكون عناصرها أسماء الحيوانات المشابهة محارفها الأولى مع محارف المدخلة في الكائن , QLineEditعند تحديد على عنصر ما من قائمة المتمم سوف يظهر نص العنصر المحدد في

عنوان النافذة و عند اختيار عنصر ما سوف يظهر نص العنصر المختار داخل صندوق رسالة مباشرة بعد اختياره ,انظر الرمّاز التالي:

```
#include <widget.h>
#include “ui_widget.h”

#include <QCompleter>
#include <QMessageBox>

widget::Widget(QWidget *parent) :
   QWidget(parent),
   ui(new Ui::Widget)
{
   ui->setupUi(this);

   QStringList  array_words;
   array_words << "Fox" << "horse" << "Monkey" << "Lion" << "tiger"
<< "Penguin"    << "Fish" << "dog" << "donkey" << "whale" << "wolf" <<
"dinosaur" << "dolphin" ;
   QCompleter *  com= new QCompleter(array_words);
   com->setMaxVisibleItems(3);
   com->setCompletionMode(QCompleter::PopupCompletion);
   com->setCaseSensitivity(Qt::CaseSensitive);
   ui->lineEdit->setCompleter(com);

connect(com,SIGNAL(activated(QString)),this,SLOT(on_activated(QStri
ng)));

connect(com,SIGNAL(highlighted(QString)),this,SLOT(on_highlighted(Q
String)));

}

void  widget::on_activated (const QString &txt){
   QMessageBox   msg;
   msg.setText(txt);
   msg.exec();
}

void  widget::on_highlighted(const QString &txt){
```

```
this->setWindowTitle(txt);
}
```

سوف تجد في القرص المرفق المثال كامل مع بعض الإضافات , لننتقل الآن إلى شرح الكائن QValidator(كائن التأكد من صحة النص المدخل).

- **تنفيذ رمّاز JS بوساطة Qt** :

أتاحة Qt استدعاء خطّاط JS و تنفيذه و الحصول على نتيجته من خلال تطوير الحزمة QtScript و التي مهمتها تنفيذ خطّاطات خارجية بوساطة رمّاز Qt C++ .

في حال أردنا استخدام أحد مكاتب الحزمة QtScript يتوجب حزم هذه المكتبة عند تجميع التطبيق وذلك من خلال إضافة السطر البرمجي التالي داخل ملف المشروع :

```
QT += script
```

الصفوف التي سوف نستخدمها من اجل تنفيذ خطّاط خارجي هي :

1- QScriptEngine : يمثل بيئة خطّاط Qt
2- QScriptValue : يحوي أنماط بيانات خطّاط Qt , مهمته احتواء قيمة خطّاط Qt
3- QScriptValueList : يحوي أنماط بيانات خطّاط Qt , مهمته احتواء مجموعة قيم خطّاط Qt

مثال إنشاء منهج بوساطة خطّاط , مهمته إرجاع مربع العدد الممرر لوسيطه , انشأ مشروع سطر أوامر و سمّه "exec_script" , ثمّ اذهب إلى ملف المشروع و أضف السطر البرمجي :

```
QT += script
```

داخل الملف "main.cpp" ضمّن المكتبة "QtScript" :

```
#include <QtScript>
```

و ضمّن ملف التروسية "مجرى الدخل و الخرج" :

```
#include <iostream>
```

اكتب داخل الكتلة "main" الرمّاز التالي :

```
QScriptEngine engine;

engine.globalObject().setProperty("words", "result is ");
```

```
QScriptValue func_sqr = engine.evaluate(
"(function sqr(num) { return words + num * num ; })" ) ;
int num = 0 ;
std::cin >> num ;
QScriptValueList args;
args <<  num ;

QScriptValue val_func = func_sqr.call(QScriptValue(), args);
std::cout << val_func.toString().toAscii().data() ;
```

أولا صرّحنا عن مثيل للصف "QScriptEngine" يدعى "engine" مهمته تجسيم بيئة الرماز الخطي , ثمّ أنشئنا متحول داخل بيئة الخطاط يدعى "words" يحتوي على النص " result is" و ذلك خلال استدعاء المنهج "setProperty" من الكائنات العامة لبيئة الرمّاز الخطي "globalObject()" , من أجل احتواء نتيجة تنفيذ الإجراء الخطي أنشئنا مثيل للصف "QScriptValue" يدعى "func_sqr" , كتبنا المنهج الخطي "sqr" داخل بيئة محرك الخطّاط بوساطة استدعاء المنهج "evaluate" التابع للصف "QScriptEngine" و تمرير المنهج الخطي لوسيطه , لاحظ كيف حصلنا على قيمة المتحول "words" من خلال كتابته فقط , المنهج الخطي "sqr" له وسيط وحيد , ويتوجب علينا تمرير قيمة لهذا الوسيط ليرجع تربيع هذه القيمة الممررة , لذا يتوجب علينا استخدم الصف "QScriptValueList" , استدعينا المنهج "call" من أجل تنفيذ المنهج الخطي "sqr" و وضعنا قيمة ناتجه داخل المثيل "val_func" , أخيرا حوّلنا النتيجة إلى نمط نصي ثمّ طبعناها على شاشة سطر الأوامر , نفذ التطبيق و اختبر النتيجة .

• كائن التأكد من صحة النص المدخل QValidator :

التأكد من صحة النص المدخل تعني إن أيّ كائن إدخال ككائن QLineEdit لن يسمح بإدخال أي محرف غير قانوني (أي أنه لن يسمح بإدخال أي محرف غير قياسي بالنسبة لكائن صحة النص المدخل المتفاعل معه).

الكائن QValidator هو صف مجرد , يحتوي المناهج الرئيسية الخاصة باختبار النص المدخل يرثه الصف QIntValidator و الصف QDoubleValidator و الصف QRegExpValidator :

يجب أن يكون قيمة النص المدخل رقم صحيح (Integer)ضمن المجال المعطى للكائن QintValidator	QintValidator
يجب أن يكون قيمة النص المدخل رقم مضاعف (Double)ضمن المجال المعطى للكائنQdoubleValidator	QdoubleValidator
يجب أن يكون قيمة النص المدخل مطابق للتعبير القياسي المعطى للكائن QRegExpValidatorكعنوان بريد إلكتروني أو موقع ويب	QregExpValidator

مثال بسيط لربط كائن QIntValidator بكائن الإدخال: QLineEdit

```
#include <QtGui/QApplication>
#include <QLineEdit>
#include <QIntValidator>
#include <QValidator>

int  main(int argc, char *argv[])
{
   QApplication   app(argc, argv);
   QValidator*  val = new  QIntValidator(2000,3000,qApp);
   //QIntValidator*  val = new  QIntValidator(qApp);
   //val->setRange(2000,3000);
   QLineEdit*  ln_edit = new QLineEdit();
   ln_edit->setValidator(val);
   ln_edit->show();
   return  app.exec();
}
```

يوجد منهج للكائن QValidator يدعى:

```
virtual State  validate ( QString & input, int & pos ) const = 0
```

يعمل على التحقق من صحة النص المدخل , input و الوسيط pos موقع المشيرة ضمن نص الوسيط المدخل , يرجع المنهج validate قيمة من التعداد State التابع للكائن QValidator انظر الجدول , التعداد QValidator::State :

النص المدخل ليس من نمط شرط كائن التحقق	0	QValidator::Invalid

(كإدخال محارف أبجدية لكائن تحقق من صحة إدخال رقم صحيحQintValidator)		
النص المدخل من نمط شرط كائن التحقق لكن ليس ضمن مجال التحقق أو لم يطبق عليه شرط التعبير القياسيّ	1	QValidator::Intermediate
نجاح التحقق من صحة النص المدخل	2	QValidator::Acceptable

سوف نتحدث الآن عن كائن التحقق من صحة التعبير القياسي والذي يدعى QRegExpValidator .

- الكائن QRegExpValidator :

الصف QRegExpValidator مشتق من الصف , QValidator يستخدم الكائن QRegExpValidatorللتحقق من صحة إدخال النص بحيث يتطابق تعبيره القياسي مع تعبير القياسي للكائن , نضع التعبير القياسي للكائن QRegExpValidator بواسطة الكائن QRegExp .

مثال عن استخدام الكائن QRegExpValidator مع الكائن, QRegExp

هذا الرمّاز لن يسمح بإدخال إلا الأرقام التابعة لنظام العد الثنائيّ أي واحد , صفر: (0,1)

```
QLineEdit* lineEdit=  new  QLineEdit();

QRegExp  regExp("[0-1]*");

QValidator*  val= new QRegExpValidator( regExp , this );

lineEdit->setValidator( val );

lineEdit->show();
```

نتذكر الحدث returnPressed (Signal) التابع لكائن الإدخال QLineEdit يقدح عند الضغط على مفتاح Enter بشرط أن يكون متحقق صحة النص المدخل , لنعدل الرمّاز السابق كما تشاهد:

```
QLineEdit* lineEdit=  new  QLineEdit();

QRegExp  regExp("[0-1]{2,4}");
```

```
QValidator*  val= new QRegExpValidator( regExp , this );

lineEdit->setValidator( val );

QObject::connect(lineEdit , SIGNAL(returnPressed() ) , this, SLOT(
anySlot() ) );

lineEdit->show();
```

سوف يقدح الحدث returnPressed عندما يكون عدد الأرقام التي تكون داخل أداة QLineEditرقمين أو ثلاث أو أربع ويجب أن يكونوا الأرقام من تابعين لأرقام نظام العد الثنائي 0و , 1 من ثمّ الضغط على مفتاح. Enter

- **الكائن QRegExp** :

كائن التعبير القياسي QRegExp مفيد في حال كنّا نريد مقارنة نص فرعي داخل نص ما إذا كان التعبير القياسي للنص يتطابق مع التعبير القياسي الذي قد وضعناه للكائن , QRegExp مثال عنوان بريد إلكتروني , موقع ويب , الخ...

يستطيع الكائن QRegExp التفاعل مع كائن التحقق QRegExpValidator للتحقق من صحة إدخال نص.

الحالات التي ينبغي علينا أن نستخدم الكائن: QRegExp

أولا : في حال كنا نريد تحقق من صحة إدخال نص, بحيث النص المدخل يحتوي على أرقام و محارف أبجدية و رموز

ثانيا : في حال كنا نريد البحث عن عناوين بريد الإلكتروني , مواقع ويب , صناديق بريد.

ثالثا : في حال كنا نريد البحث ومن ثمّ إستبدال النص الذي قد وجد , مثال نريد أن نبحث عن جميع عناوين البريد الإلكتروني الذي ينتهي امتداده com و استبداله بـ. sy

• نص التعبير القياسيّ:

سنتكلم عن أهم نصوص (تعابير) الكائن. QRegExp

التعبير [...] مقارنة المحارف الموجودة داخل الأقواس المربعة بالنص المدخل للكائن QRegExp.

التعبير [1-9] المقارنة من الرقم 1 إلى الرقم 9 بالنص المدخل للكائن , QRegExp لأن إشارة " – "ترمز إلى التسلسل , أيّ (من إلى) , مثال وضعنا المحرف A أولا ثم رمز – ثم محرف Dبين الأقواس المربعة [A-E] كأننا كتبنا , [ABCDE] كما التعبير [1-9] يعوض : [123456789] .

التعبير {n,m} عدد المحارف المسموحة للإدخال من العدد n إلى العدد n , m أصغر عدد محارف مسموح إدخاله m , أكبر عدد محارف مسموح إدخاله.

مثال b{2,4} : يقصد مسموح بإدخال محرف b من محرفين b إلى أربع محارف.

مثال آخر {4,6}[8-0] يسمح بإدخال الأرقام من 0 إلى الرقم 8 لكن يجب أن يكون عدد الأرقام بين الأربع أرقام إلى ستة أرقام.

التعبير { ,n} عدد المحارف المسموحة للإدخال من العدد n إلى غير حد n , أصغر عدد محارف مسموح إدخاله.

التعبير {m,} عدد المحارف المسموحة للإدخال من صفر عدد) من دون أي محرف مدخل (إلى عدد محارف مساوي قيمة) m أكبر عدد محارف مسموح إدخاله. (

التعبير ^ عندما يكون موقعه في بداية نص التعبير يعني أنه يجب مطابقة بداية النص المدخل مع بداية التعبير القياسي , أي التعبير الموجود بعد الرمز ^ مباشرة.

مثال:

```
QRegExp  regExp ( "^[0-9]" );
```

أما في حال أخرى يأتي التعبير ^ بمعنى نفي , وسيسمح بأي نص مدخل , مثال لا نريد نص مدخل يحتوي على الأرقام بين 0 إلى 9 و أي محرف آخر نريده.

مثال:

```
QRegExp  regExp ( "[^0-9]" );
```

التعبير $ عندما يكون موقعه في نهاية نص التعبير يعني أنه يحب مطابقة نهاية النص المدخل مع نهاية التعبير القياسي , أي التعبير الموجود قبل الرمز $ مباشرة.

التعبير * عندما يكون موقعه بعد تعبير ما سوف يسمح بتكرار النص المدخل و الذي يطابق التعبير ,وسيسمح أيضا بعدم إدخال أي محرف.

جرب هذا الرمّاز أولا:

```
QRegExp  regExp ( “(b)” );
```

ثمّ أضف الرمز * كما ترى:

```
QRegExp  regExp ( “(b*)” );
```

ولا حظ الفرق بينهم.

التعبير + عندما يكون موقعه بعد تعبير ما سوف يسمح بتكرار النص المدخل و الذي يطابق التعبير ,ولكن يجب أن يدخل محرف أو أكثر.

جرب هذا الرمّاز أولا:

```
QRegExp  regExp ( “(b)” );
```

ثمّ أضف الرمز + كما ترى:

```
QRegExp  regExp ( “(b+)” );
```

التعبير ? عندما يكون موقعه بعد تعبير ما يعني أن النص المدخل سيتطابق مع التعبير في حال كان موجود ذاته(حالة تطابق النص المدخل مع التعبير القياسي الموجود خلف الرمز ؟) أو غير موجود النص المطابق للتعبير . مثال:

```
QRegExp  regExp ( “(ABC?)“);
```

محرف C أنت حر في إدخاله.

التعبير | تعبير منطقي بمعنى" أو , "OR" , "مثال:

```
QRegExp  regExp ( “(Ahmad|Mouhammad|Ihab)” );
```

سوف يسمح بإدخال اسم واحد فقط من هذه الأسماء الثلاثة.

التعبير \ نستخدمه في حال كنا نريد أن يكون النص المدخل رمز معرف من جهة التعبير القياسي كرمز , ? لا نستطيع كتابته ضمن التعبير مفرد كالرمّاز:

```
QRegExp  regExp ( “?“);
```

بل نضع خلفه الرمز: \

```
QRegExp  regExp ( “\?“);
```

جدول لبعض التعابير القياسية الأكثر استخداما:

الشرح	التعبير
تطابق عدد فقط	\d
تطابق أي محرف ما عدا أن يكون النص المدخل عدد	\D
تطابق فراغ فقط	\s
تطابق أي محرف ما عدا محرف الفراغ Space	\S
تطابق محارف الأبجدية و الأرقام و الرمز_	\w
عكس التعبير السابق	\W
حد الكلمة) نهايتها أو بدايتها(	\b
ليس للكلمة حد	\B
تعبير النقطة :أي محرف حتى لو كان سطر جديد	.

مثال عن التعبير القياسي الخاص بتطابق العدد:

```
QRegExp  regExp ( "\\d+");
```

يجب أن يكون النص المدخل عدد واحد أو أكثر (رقم)لكي يحصل التطابق بين النص المدخل و التعبير.

المنهج setPattern (const QString &pattern) لا يرجع قيمة, مهمته وضع نص التعبير القياسي.

```
QRegExp  regExp;

regExp.setPattern("<h1>.*</h1>");
```

المنهج QString pattern() const يرجع نص التعبير القياسي.

المنهج pos يعيد قيمة من نمط صحيح قيمته موقع المشيرة الحالي ضمن النص الذي يقارن بالتعبير القياسي أي موقع آخر محرف قارنه كائن التعبير القياسي ضمن النص المدخل.

المنهج

int QRegExp::indexIn(const QString &str, int offset = 0 , CaretMode caretMode = CaretAtZero)

يعمل هذا المنهج على مقارنة النص str بالتعبير القياسي للكائن , QRegExp يرجع قيمة صحيحة تكون قيمتها 1- إذا لم يكن أي نص فرعي داخل النص str مطابق للتعبير القياسي , و إلا يرجع موقع أول نص فرعي مطابق للتعبير القياسي , أما بالنسبة للوسيط offset هو موقع المشيرة في النص str لمطابقة التعبير القياسي موقع أول محرف يبدأ منه مطابقة النص بالتعبير القياسي ,الوسيط caretMode من نمط تعداد CaretMode و قيمته الافتراضية CaretAtZero , انظر الجدول, التعدادQRegExp::CaretMode :

QRegExp::CaretAtZero	**0**	**علامة الإقحام (^) سوف يكون موقعها أول النص (الوسيطstr) الذي سوف يقارن بالتعبير القياسي**
QRegExp::CaretAtOffset	**1**	**علامة الإقحام (^) سوف يكون موقعها قيمة الإزاحة (الوسيطoffset) لنص الوسيطstr الذي سوف يقارن بالتعبير القياسي**
QRegExp::CaretWontMatch	**2**	**يجب أن يكون كامل النص متطابق مع التعبير القياسي**

مثال عن المنهج: indexIn

```
QRegExp regExp;
regExp.setPattern("^([^a-zA-Z_-\\s]+)\\s([^a-zA-Z_-\\s]+)\\s([^a-zA-Z_-\\s]+)$");

if( regExp.indexIn ("11 22 33") != -1){
QString  str=regExp.cap(2);//will be value it equal 22
}
QStringList list=regExp.capturedTexts();//all captured texts
int nCount=regExp.captureCount();      //captured words count
int len=regExp.matchedLength();       //length characters of captured texts
```

انتهينا من كائن التعبير القياسي QRegExp وبه يكون انتهاء هذا الفصل.

لا عفوا قد نسيت ذكري المسبق أني سوف أضع مثال في آخر الفصل لتوضيح طريقة استخدام الكائن QMdiArea ,لنبدأ..

قبل أن نكتب الرمّاز الخاص بالمثال MdiArea سنتكلم عن الكائن QMainWindow لأننا سوف نستخدمه في مثال MdiArea نظرا لتهيئته بشكل جيد للتعامل مع تطبيقات المتعددة الوثائق .

- **الكائن QMainWindow : قد تمّ إنشاء هذا الصف ليكون نافذة التطبيق الرئيسية لأنه بنية مهيأ لاحتواء مباشر لشريط القائمة و شريط الأدوات و شريط الحالة و الشريط الخاص برصف الأدوات (أيّ إنه مضمن هذه الكائنات داخله) , تستطيع أن تضع كائن مرئي (Widget) في كامل القسم الأوسط منه بوساطة المنهجية setCentralWidget , سنرى كيفية كتابة رمّاز متعدد المستندات بالتفاعل مع الكائن QMainWindow بالمثال القادم.**

❖ المثال MdiArea :

سوف ننشئ تطبيق يحتوي على وثائق (مستندات) متعددة , يكون نافذة التطبيق الرئيسية مشتق من الكائن QMainWindow يحتوي على قائمة File الحاوية للعنصر New و العنصر Close Sub Window و Exit .

أولا : أنشأ مشروع Qt Gui Application سمه MdiArea في صفحة Class Information ضمن خانة Base Class دعها على قيمتها الافتراضية وهي QMainWindow .

ثانيا - ضمن ملفات الترويسة التالية داخل الملف mainwindow.h :

```
#include <QMenu>
#include <QAction>
#include <QMdiArea>
#include <QMdiSubWindow>
#include <QTextEdit>
#include <QPixmap>
#include <QIcon>
```

ثالثا - صرح عن المتحولات التالية في القسم الخاص داخل نفس الملف السابق :

```
QMenu*  file;
QMdiArea*  mdiArea_widget;
```

```
QMdiSubWindow*  mdi_Sub_Win;
QTextEdit*  txtEdit;
```

رابعا - عرف المقابس التالية ضمن نفس الملف :

```
private slots:
  void  close_sub_win();
  void  new_sub_win();
  void  show_msg(QMdiSubWindow*);
```

المقبس close_sub_win لإغلاق النافذة الابن النشطة .

المقبس new_sub_win لإضافة مستند (نافذة ابن) جديدة .

المقبس show_msg لإظهار عنوان النافذة الابن (المستند) النشطة في شريط الحالة , الوسيط *QMdiSubWindow هو مؤشر لكائن النافذة الابن التي نود إظهار عنوانها.

خامسا – أضف ملف مورد للمشروع يدعى icons يحوي على الفهرس tools يوجد داخله ثلاث شعارات الشعار الأول يدعى new و يرمز إلى عنصر القائمة new و الشعار الثاني يدعى close يرمز إلى عنصر القائمة close sub window و الشعار الثالث و الأخير يدعى exit و يرمز إلى عنصر القائمة Exit .

سادسا – اكتب داخل ملف mainwindow.cpp الرمّاز الخاص بإضافة القائمة File و عناصرها

new – close sub window – exit وذلك في كتلة البناء تحت سطر الرمّاز -ui >setupUi(this);

وهو هذا :

```
file = this->menuBar()->addMenu("&File");
file->addAction(QIcon(QPixmap(":/tools/new")),"&New");
file->addSeparator();
file->addAction(QIcon(QPixmap(":/tools/close")),"&Close Sub
Window");
file->addSeparator();
file->addAction(QIcon(QPixmap(":/tools/exit")),"E&xit");
```

بما أننا استخدمنا النافذة الأساسية للتطبيق ترث الصف QMainWindow فهي تحوي شريط قائمة مهيأ مسبقا من قبلها لذلك أخذنا مؤشر كائن شريط القائمة التابع للنافذة

الأساسية mainwindow للمتحول file , من ثمّ أضفنا العناصر الثلاث يوجد بين كل عنصر و عنصر آخر خط فاصل .

سابعا – أضف الرمّاز الخاص بإضافة عناصر لشريط الأدوات مع ربطها بالمستقبلات التي تقابلها, اكتب الرمّاز تحت سابقه :

```
  ui->mainToolBar->addAction(QIcon(QPixmap(":/tools/new")),
"New",this,SLOT(new_sub_win()));
  ui->mainToolBar->addAction(QIcon(QPixmap(":/tools/close")),
"Close Sub Window",this,SLOT(close_sub_win()));
  ui->mainToolBar-
>addAction(QIcon(QPixmap(":/tools/exit")),"Exit",this,SLOT(close()));
```

ثامنا – اربط الحدث و المستقبل للعناصر الثلاث السابقة (جديد – إغلاق النافذة النشطة - خروج) بواسطة الرمّاز التالي :

```
  connect(file->actions().at(4),SIGNAL(triggered()),this,SLOT(close()));
  connect(file-
>actions().at(0),SIGNAL(triggered()),this,SLOT(new_sub_win()));
  connect(file-
>actions().at(2),SIGNAL(triggered()),this,SLOT(close_sub_win()));
```

تاسعا – ننشئ حدث الكائن QMdiArea و نضعه في وسط نافذة التطبيق بحيث يأخذ كامل الحيز الأوسط للنافذة , اكتب الرمّاز التالي :

```
  mdiArea_widget = new QMdiArea();
  this->setCentralWidget(mdiArea_widget);
```

عاشرا – ربط حدث تنشيط النافذة الابن بالمقبس الخاص بإظهار عنوان النافذة الابن النشطة في شريط الحالة :

```
connect(mdiArea_widget,SIGNAL(subWindowActivated(QMdiSubWind
ow*)),this,SLOT(show_msg(QMdiSubWindow*)));
```

الخطوة الحادية عشر – لنحقق المقبس الخاص بإضافة مستند (نافذة ابن) جديد :

```
void MainWindow::new_sub_win(){
  mdi_Sub_Win = new QMdiSubWindow();
  txtEdit = new QTextEdit();
```

```
   mdiArea_widget->addSubWindow(mdi_Sub_Win);
   mdi_Sub_Win->setGeometry(0,0,400,400);
   mdi_Sub_Win->setWidget(txtEdit);
   mdi_Sub_Win->setWindowTitle(
tr("Document %1").arg(QString::number(mdiArea_widget-
>subWindowList().length())));
   mdi_Sub_Win->show();
   mdiArea_widget->cascadeSubWindows();
}
```

أولا نأخذ حدث للنافذة الابن , ثانيا لأخذ حدث من كائن صندوق النص , ثمّ نضيق كائن النافذة الابن على الكائن الحاوي للنوافذ الأبناء **mdiArea_widget** , نهيئ حجم النافذة الابن مع موقعها بواسطة التابع **setGeometry** نضيف للنافذة الابن كائن صندوق النص و نضع عنوانها **Document %1** أي قيمة الوسيط الأول بعد هذا النص سوف يكون بدل من الرمز **1%** وهنا الوسيط قيمته هي العدد الكلي للنوافذ الأبناء المنشأة مسبقا طبعا هيئنا عنوان النافذة الابن بواسطة المنهج **tr** التابع للصف الأب **QObject** الخاص بتنسيق النص و ترجمته إلى إصدار نص المصدر الحالي (المحلي), ثمّ نظهر النافذة الابن بواسطة المنهجية **show** و أخرا نرتب توضع النوافذ الأبناء بشكل متدرج المنهج **cascadeSubWindows** .

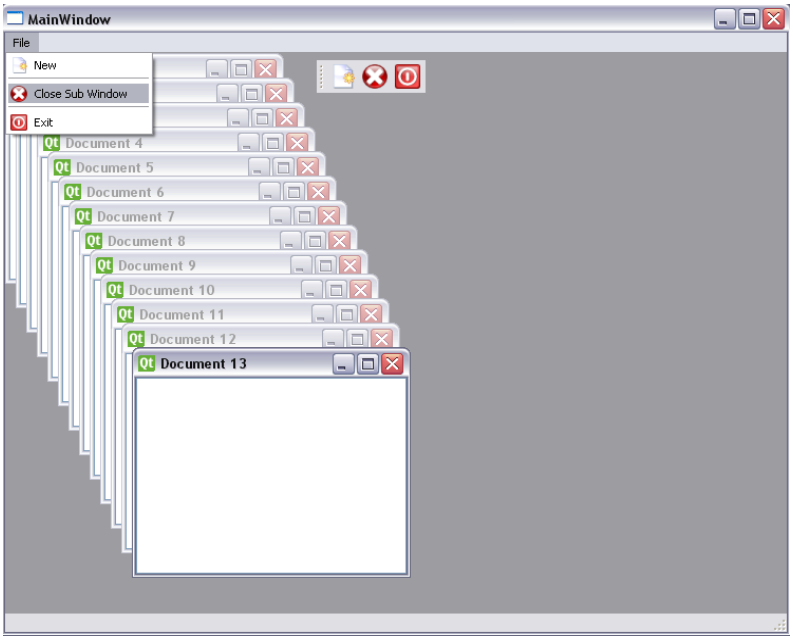
الخطوة الثانية عشر – تحقيق مقبس إغلاق النافذة المنشطة :

```
void MainWindow::close_sub_win(){
mdiArea_widget->closeActiveSubWindow();
}
```

الخطوة الثالثة عشر و الأخيرة – نحقق مقبس إظهار عنوان النافذة الابن المنشطة بشريط الحالة :

```
void MainWindow::show_msg(QMdiSubWindow * sub_win)
{
ui->statusBar->showMessage(sub_win->windowTitle());
}
```

نفذ التطبيق سوف تراه كما في الشكل (2.11) :



الشكل 2.11

انتهينا من مثال التطبيق متعدد الوثائق , سوف تجده داخل القرص المرفق .

قد تعلمنا في هذا الفصل مواضيع ذات أهمية كبيرة في أي تطبيق مبني في Qt إن كان موجه لأنظمة سطح المكتب

(Window - Mac OS - Linux) أو لأنظمة الأجهزة الذكية

(Series 40-60 - Symbian – Maemo / MeeGo – Win CE) و أجهزة Android من شركة Google , الخ...

• خلاصة الفصل:

نستطيع الآن العمل مع كافة كائنات واجهة المستخدم الرسومية من كائنات الإدخال , و كائنات العرض و كائنات الأزرار و التخطيط الخ ...بالإضافة إلى العمل مع التطبيقات المتعددة المستندات و كائن المتمم QCompleter و كائن التحقق من صحة إدخال النص QValidator و كائن التعابير القياسية. QRegExp

الفصل الثالث

برمجة رسوميات في Qt

Programming Graphics By Qt

مقدمة Intro :

إن عالم البرمجيات من دون معالجة الرسوميات الساكنة و المتحركة خالي من المتعة , لذا قررت أن أضع هذا الفصل الذي يتكلم عن أكثر الصفوف المستخدمة لإنشاء و معالجة و تحريك الرسوميات في Qt .

نتكلم في هذا الفصل عن برمجة الرسوميات كرسم مستطيل, خط, منحني دائرة ... وسوف نتكلم عن كيفية تلوينها و استخدام صف الفرشاة (إعطاء شكل لمادة التعبئة) و كيفية تعبئة الكائنات بألوان متدرجة (Gradient)

نذكر أيضا الكائن QGraphicsView الذي يعرض المشاهد الرسومية QGraphicsScene ,بعد إضافة هذه الكائنات له يقوم بتصييرها Render وثم يعرضها.

الكائن QGraphicsScene يحوي بدوره عناصر رسومية QGraphicsItem يمكن أن تكون هذه العناصر إما صورة أو منحني أو خط , قطع ناقص الخ...

يحوي أيضا أساسيات الطباعة و المعاينة قبل الطباعة .

بعد شرح كائنات الطباعة سوف نتكلم عن برمجة كائنات رسومية متحركة غير ساكنة ,نتكلم عن صف شريط الزمن QTimeLine , و نشرح العمل مع الصف QPropertyAnimation وصفوف المجموعات كصف QSequentialAnimationGroup وعدة صفوف سوف نذكرها في مكانها , أما الآن دعنا نخوض في أعماق برمجة الرسوميات في Qt .

برمجة الرسوميات الساكنة منها و المتحركة في Qt سهلة بالنسبة لمقدراتها .

❖ أساسيات الرسم في Qt :

نبدأ أول فقرة في هذا الفصل بالتكلم عن الكائن المنخفض المستوى الخاص برسم الأشكال البسيطة و المعقدة مباشرة على أي كائنات رسومية widgets أو أجهزة رسومية وهو الصف QPainter الذي يرث QPaintDevice .

حتى نستطيع الرسم بواسطة الصف QPainter على أي كائن تابع لواجهة المستخدم الرسومية يجب أن نعيد تحقيق المنهج المحمي و الإفتراضي paintEvent الخاص بحدث الرسم على الكائن widget .

يحوي صف QPainter مناهج خاصة به لرسم أي نوع من الأشكال من خط و مستطيل و منحنيات و تستطيع من خلاله استيراد صور بالاستعانة بكائنات أخرى كالكائن QPixmap ومن ثمَ الرسم داخلها و حفظها .

دعنا نرى مثال بسيط لرسم خط على الكائن widget مباشرة من خلال إعادة تحقيق المنهج paintEvent :

أولا أضف ملف الترويسة QPaintEvent الخاص بوسيط حدث الرسم
paintEvent(QPaintEvent *) والذي يفيدنا من أجل تحديد المساحة المهيئة للرسم من خلال المنهج rect أو من خلال المنهج region التابعين للصف QPaintEvent :

```
#include <QPaintEvent>
```

الآن عرّف حدث الرسم في القسم العام على أنه افتراضي كالتالي :

```
public :

  virtual void paintEvent(QPaintEvent *);
```

أضف الرمّاز الخاص بتضمين ملف الترويسة QPainter :

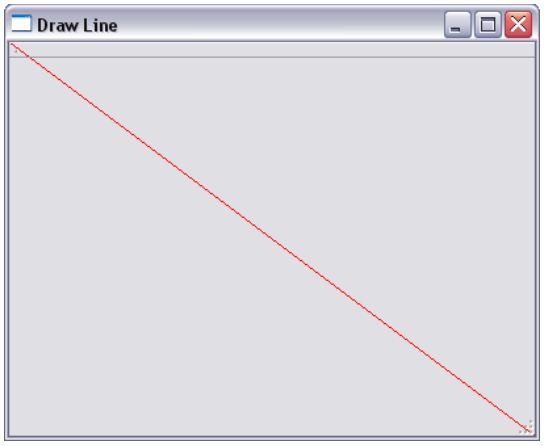
```
#include <QPainter>
```

اذهب إلى قسم التحقيق و حققه بكتابة الرمّاز التالي :

```
void MainWindow::paintEvent(QPaintEvent *prt)

{
```

```
   QPainter painter(this);
   painter.setPen(Qt::red);
   painter.drawLine(0,0,prt->rect().width(),prt->rect().height());
}
```

عند تنفيذ التطبيق سوف ترى كالشكل (3.1) :



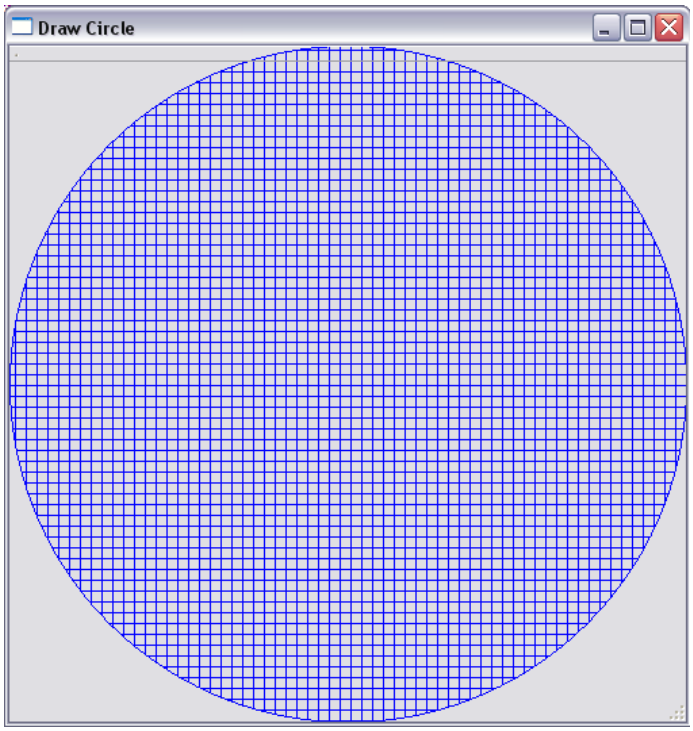
الشكل 3.1

لرسم دائرة عدّل الرمّاز السابق الخاص بالمنهجية paintEvent ليصبح كالآتي :

```
void MainWindow::paintEvent(QPaintEvent *prt)
{
   QPainter painter(this);
));255,0    painter.setPen(qRgb(0,
),Qt:: CrossPattern));255,0    painter.setBrush(QBrush(QColor(0,
   painter.drawEllipse(0,0,prt->rect().width(),prt->rect().height());
}
```

أخذنا مثيل للصف QPainter وممرنا إلى بنّاءه المؤشر الذاتي this أي جهاز الرسم سوف يكون النافذة الحالية, ثمّ وضعنا قيمة لون القلم (الحد) اللون الأزرق بوساطة المنهج qRgb الذي يمرر له اللون الأحمر و الأخضر و الأزرق بقيمة البايت (Byte) أي 255 أقصى حد لها وبدوره يرجع اللون الحاصل بعد مزج الألوان , ثمّ وضعنا قيمة الفرشاة (لون و شكل التعبئة) اللون أزرق و شكل التعبئة على القيمة Qt:: CrossPattern والتي سوف نضع جدول في

هذا الفصل خاص بثوابت شكل تعبئة الفرشاة وجدول آخر خاص بثوابت نمط رسم القلم (حدّ القلم) نتكلم عنه في مكانه ,ثم رسمنا الدائرة بوساطة المنهج drawEllipse الخاص برسم قطع , يجب أن تكون نافذة التطبيق على شكل مربع (العرض = الارتفاع) و إلا سيرسم على النافذة شكل بيضاوي , نفذ التطبيق سوف ترى الشكل (3.2):



الشكل 3.2

بالنسبة لمناهج الرسم الخاصة بالكائن QPainter دائما يبدأ اسمها بـ draw , مثال المنهج الخاص برسم منحني يكون :

QPainter::drawArc(int x, int y, int w, int h , int a , int alen)

يوجد عدة مناهج رسم للكائن QPainter أهمها :

يستخدم لرسم خط	drawLine
يستخدم لرسم مجموعة من الخطوط	drawLines
يستخدم لرسم قطع ناقص	drawEllipse
يستخدم لرسم مضلع مغلقة حدوده	drawPolygon
يستخدم لرسم مضلع ممكن أن تكون حدوده غير مغلقة	drawPolyline

يستخدم لرسم صورة محتواة ضمن كائن QPixmap	drawPixmap
يستخدم لرسم وتر	drawChord
يستخدم لرسم منحني	drawArc
يستخدم لرسم مستطيل أو مربع زواياه منحنية (مدورة)	drawRoundedRect
يستخدم لرسم (طباعة) نص	drawText
يستخدم لرسم صورة محتواة ضمن كائن QPicture	drawPicture
يستخدم لرسم نقطة	drawPoint
يستخدم لرسم نقاط	drawPoints

يوجد مقبس لكائنات widgets يدعى repaint يستدعي المنهج paintEvent لتحديث و إعادة رسم الكائنات.

لنبدأ بالتكلم عن أهم الكائنات الرسومية و أدواتها في Qt .

❖ كائن عرض المشهد الرسومي QGraphicsView :

يستخدم هذا الكائن مثلما ذكرنا مسبقا أنه يعرض المشاهد الرسومية بعد تصييرها Render .

تستطيع التصيير بواسطة OpenGL عن طريق المنهج setViewPort(QGLWidget*) .

المنهج setViewPort فهو لعرض كائن widget ضمن منفذ المشهد .

بوابة المشهد هي الجزء من المشهد المعروض حاليا , عندما يكون المشهد أكبر مقاسا من كائن عرض المشهد الحاوي له سوف يظهر بشكل تلقائي شريط انزلاق لكي تستطيع بواسطته مشاهدة الجزء الغير ظاهر من المشهد .

إحداثيات مساقط المنظور الإفتراضية هي 0,0 أي أول نقطة للمشهد الزاوية العليا اليسارية من كائن عرض المشهد , لكن سوف تتغير بالتناسب مع تغيير قيمة شريط الانزلاق .

يضاف كائن المشهد لكائن عرض المشهد بوساطة المنهج :

QGraphicsView::setScene(QGraphicsScene*)

❖ كائن المشهد QGraphicsScene:

يستخدم كائن المشهد QGraphicsScene لاحتواء مجموعة كبيرة من العناصر الرسومية QGraphicsItem

يضاف هذا الكائن إلى كائن عرض المشهد QGraphicsView فيعرض جميع الكائنات الرسومية الموجود داخله.

يسمح لك كائن المشهد بإضافة عناصر رسومية QGraphicsItem بوساطة المنهج addItem(QGraphicsItem*) وتستطيع أيضا إضافة عناصر رسومية عليه بشكل مباشر بوساطة المناهج addLine أو addPixmap , addText الخ ..

يمكّنك كائن المشهد من تكبير و تصغير المشهد (Zoom in-Zoom out) بوساطة المنهج scale الذي يفيد بتحويل المقياس.

يمكن أن تعيد كائن رسومي ما QGraphicsItem يكون موجود داخل كائن المشهد بوساطة إحداثيات موقع الكائن الرسومي المراد .

مثال بسيط لرسم نص على مشهد بشكل مباشر ومن ثمّ عرضه داخل كائن العرض , انظر الرمّاز :

```
QGraphicsView  view;

QGraphicsScene  scene;

scene.addText(QObject::tr("Hello \nQt"));

view.setScene(&scene);

view.show();
```

أخذنا مثيل لكائن عرض المشهد QGraphicsView يدعى view و مثيل آخر لكائن المشهد scene, ثمّ أضفنا نص (طبعنا نص) داخل المشهد بوساطة المنهج addText ثم وضعنا المشهد داخل كائن عرضه أي الكائن view و أظهرنا كائن العرض بوساطة المستقبل show .

لا تنسى إضافة ملفات الترويسة QGraphicsView , QGraphicsScene .

نفذ التطبيق سوف تظهر نافذة موجود داخلها النص Hello وفي السطر الذي يليه النص Qt .

لنرى كيفية تكبير تصغير المشهد المعروض , فقط عدل الرمّاز السابق ليصبح كالتالي :

```
QGraphicsView view;

QGraphicsScene scene;

QMatrix  m;

m.scale(0.1,0.1);

scene.addText(QObject::tr("Hello \nQt",QFont("Arial",36)));

view.setScene(&scene);

view.setMatrix(m);

view.show();
```

الكائن m هو مثيل للصف QMatrix الذي يفيد بتحويل نظام الإحداثيات , أما المنهج scale (تحويل المقياس) يقس نظام الإحداثيات ثمّ يعدله بعد ضربه بالقيمة المدخل أي كل قيمة 1 من وحدة المقاس (الإفتراضية pixel) سوف تضرب بالعدد المدخل للمنهج scale, له وسيطين الوسيط الأول sx لتعديل مقاس المحور الأفقي , أما الوسيط الثاني sy لتعديل مقاس المحور العمودي .

عدل قيمة وسطاء المنهج scale و اختير النتيجة .

لننتقل إلى كائنات العناصر الرسومية QGraphicsItem .

❖ كائنات العناصر الرسومية QGraphicsItem :

هي كائنات رسومية من نص إلى مستطيل أو خط أو منحني الخ... لكي تعرض تضاف لكائن المشهد ,

تستطيع وضع تأثيرات (عناصر تأثير - QGraphicsEffect) لهذه الكائنات من تأثير ضبابي إلى تأثير الشفافية الخ.. سوف يأتي شرحها بعد قليل .

الصف الأب (الأساس) لجميع كائنات العناصر الرسومية هو QGraphicsItem .

يوجد عدة عناصر رسومية أهمها :

كائن النص QGraphicsTextItem .

كائن الخط QGraphicsLineItem .

كائن المستطيل QGraphicsRectItem .

كائن Pixmap QGraphicsPixmapItem .

كائن القطع QGraphicsEllipseItem .

كائن عنصر رسومي متحرك QGraphicsItemAnimation .

كائن عنصر عارض الويب QGraphicsWebView .

كائن عنصر المضلع QGraphicsPolygonItem

كائن احتواء مجموعة من العناصر الرسومية QGraphicsItemGroup .

لنأخذ مثال بسيط لإنشاء عنصر مستطيل QGraphicsRectItem ثمّ عرضه , انظر الرمّاز :

```
QGraphicsView view;
QGraphicsScene scene;
rectItem; QGraphicsRectItem
rectItem.setRect(0,0,200,300);
rectItem.setBrush(QBrush(Qt::red));
scene.addItem(&rectItem);
view.setScene(&scene);
view.setGeometry(200,200,500,500);
view.show();
```

استخدمنا المنهج addItem لإضافة عنصر المستطيل.

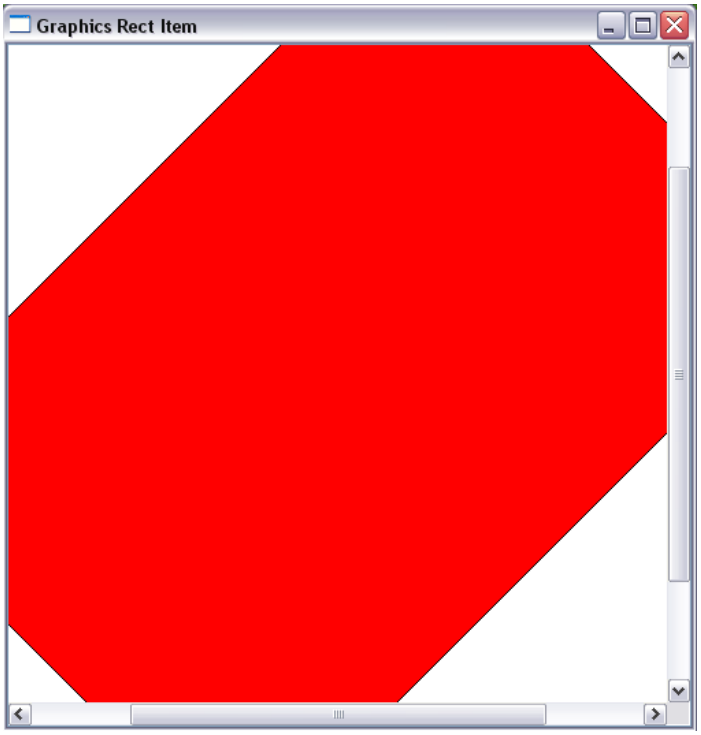
عند تنفيذ هذا المثال سوف يظهر مستطيل ارتفاعه 300 بكسل و عرضه 200 بكسل ولون خلفيته هو أحمر .

إذا أردنا أن ندور المستطيل بزاوية 45 درجة و تكبير عنصر المستطيل (تحويل نظام الإحداثيات – تحويل المقاس) ,عدل الرمّاز السابق كالتالي:

```
QGraphicsView view;

QGraphicsScene scene;

QGraphicsRectItem rectItem;

rectItem.setRect(0,0,200,300);

rectItem.setBrush(QBrush(Qt::red));

rectItem.rotate(360/8);

rectItem.scale(2,2);

scene.addItem(&rectItem);

view.setScene(&scene);

view.setGeometry(200,200,500,500);

view.show();
```

دوّرنا المستطيل بمقدار زاوية 45 درجة بوساطة المنهج rotate , و حولنا مقاس نظام الإحداثيات بوساطة المنهج scale , هذه المناهج تابعة لكائن العنصر الرسومي QGraphicsItem .

عند تنفيذ التطبيق سوف ترى مقدار الدوران على المستطيل واضح و سوف ترى المستطيل أكبر بقيمة الضعف عن ذي قبل , لاحظ أيضا ظهور الأشرطة المنزلقة (العمودية و الأفقية) لأن المشهد أصبح أكبر من كائن عرض المشهد , انظر الشكل (3.3) :



الشكل 3.3

❖ **إعادة مؤشر عنصر كائن رسومي موجود داخل مشهد ما :**

ذكرنا مسبقا في فقرة كائن المشهد أننا نستطيع أخذ مؤشر لكائن موجود داخل كائن المشهد بوساطة المنهج itemAt التابع لكائن المشهد لنرى مثال عن ذلك , نفس المثال السابق عدله ليصبح رمّازه كالتالي :

```
QGraphicsView view;

QGraphicsScene scene;

QGraphicsRectItem rectItem;

rectItem.setRect(0,0,200,300);

rectItem.setBrush(QBrush(Qt::red));

rectItem.rotate(90/2);

rectItem.scale(2,2);

scene.addItem(&rectItem);
```

```
view.setScene(&scene);

view.setGeometry(200,200,500,500);

view.show();


QGraphicsItem *item;

item=scene.itemAt(100,100);

//QGraphicsRectItem *rect=
qgraphicsitem_cast<QGraphicsRectItem*>(item);

//rect->scale(0.1,0.1);

item->scale(0.1,0.1);
```

صرحنا عن مؤشر عنصر رسومي QGraphicsItem اسمه item ثمّ استخدمنا المنهج itemAt ومررنا الوسيطين x , y , سوف يرجع مؤشر كائن العنصر الرسومي الذي وقعت فيه نقطة تقاطع الوسيطان x ,y , ثمّ صغرنا مقاس وحدة الإحداثيات إلى العشر بوساطة منهج التقييس scale .

انظر إلى سطرين التعليق داخل الرمّاز سو ترى أننا صرحنا عن مؤشر كائن عنصر رسومي اسمه rect ووضعنا قيمته قيمة مؤشر العنصر الرسومي الذي حصلنا عليه مسبقا , هنا حولنا كائن العنصر الرسومي المجرد (الأب) إلى كائن عنصر مستطيل بوساطة المنهج qgraphicsitem_cast الموجود داخل ملف الترويسة qgraphicsitem.h , وصغرنا مقاس وحدة الإحداثيات إلى العشر بوساطة منهج التقييس scale .

المنهج qgraphicsitem_cast يفيدنا للتحويل بين أنماط كائنات العناصر الرسومية.

نفذ التطبيق سوف تلاحظ أنّ المستطيل قد صغر حجمه إلى عشر ما كان عليه .

لننتقل إلى فقرة إضافة تأثيرات للعناصر رسومية .

- **عرض صورة متحركة :**

لعرض صورة متحركة "GIF" يتوجب علينا استيرادها بوساطة الصف "QMovie" من ثمّ عرضها , انظر الرمّاز التالي :

```
QMovie* movie = new QMovie(“pic.gif”);

QLabel  lbl;

lbl.setMovie(movie);

lbl.show();

movie->start();
```

استخدمنا المنهج "()start" لبدأ تشغيل حركة الصورة .

❖ إضافة تأثيرات للعناصر رسومية - الكائن QGraphicsEffect :

التأثيرات على العناصر الرسومية تعني تصيير render العنصر الرسومي مع التأثير الرسومي الموضوع , كإضافة تأثير الظل أو تأثير الضباب على عنصر رسومي ودمجه معه بوساطة التصيير render .

طبعا سوف يتم تصيير فقط منفذ العرض viewport أي المقطع المرئي (المشاهد) من المشهد .

الصف الأب لكائنات التأثير الرسومي يدعى QGraphicsEffect وهذه الكائنات (الصفوف) هي :

كائن التأثير الضبابي	QgraphicsBlurEffect
كائن تأثير الظل	QgraphicsDropShadowEffect
كائن تأثير التظليل أي تظليل الكائن الرسومي بلون ما	QgraphicsColorizeEffect
كائن تأثير الشفافيّة	QgraphicsOpacityEffect

تستطيع إنشاء تأثير رسومي خاص بك من خلال إنشاء صف يرث الصف الأب QGraphicsEffect ة من ثمّ تحقيقه (أي إعادة تحقيق مناهجه).

سوف نضع أربع أمثلة , لكل تأثير رسومي مثال :

- **المثال الأول عن التأثير الرسومي الضبابي QGraphicsBlurEffect .**

سوف نصرح عن كائن رسومي من نمط QGraphicsPixmapItem يحوي داخله صورة ثمّ نضع عليه تأثير ضبابي QGraphicsBlurEffect.

انظر الرمّاز :

أضف الترويسات التالية :

```
#include <QGraphicsScene>
#include <QGraphicsPixmapItem>
#include <QGraphicsBlurEffect>
#include <QGraphicsView>
```

اكتب الرمّاز التالي :

```
QGraphicsView  view;
QGraphicsScene* scene= new QGraphicsScene(0,0,600,600);
QGraphicsPixmapItem* pixmap= new QGraphicsPixmapItem();
pixmap->setPixmap(QPixmap(":/png/qt"));
QGraphicsBlurEffect* blurEffect= new QGraphicsBlurEffect;
blurEffect->setBlurHints(QGraphicsBlurEffect::QualityHint);
blurEffect->setBlurRadius(10);
pixmap->setGraphicsEffect(blurEffect);
scene->addItem(pixmap);
view.setScene(scene);
view.show();
```

أنشأنا مثيل لكائن عرض المشهد يدعى view و صرحنا عن مؤشر لكائن المشهد يدعى scene يبدأ بالرسم داخله من نقطة الإحداثيات 0,0 بعرض و ارتفاع 600 بكسل ,و آخر لكائن العنصر الرسومي QGraphicsPixmapItem يدعى pixmap ثمّ وضعنا صورة ضمنه , و صرحنا عن مؤشر لكائن التأثير الضبابي QGraphicsBlurEffect يدعى blurEffect وضعنا للمنهج (setBlurHints(BlurHints القيمة

QGraphicsBlurEffect::QualityHint , هذا المنهج مفيد في تحديد كيفية تطبيق التأثير على العنصر المراد تأثيره , وله ثلاث أنماط تابعة للتعداد QGraphicsBlurEffect::BlurHint, انظر الجدول :

لتطبيق التأثير بأفضل أداء , أسرع أداء مع دقة منخفضة	**QGraphicsBlurEffect::PerformanceHint**
لتطبيق التأثير بأفضل دقة , أبطأ أداء مع دقة عالية	**QGraphicsBlurEffect::QualityHint**
لتطبيق التأثير بنمط الرسوم المتحركة , أي ليضع التأثير على العنصر الرسومي وهو في حال الحركة	**QGraphicsBlurEffect::AnimationHint**

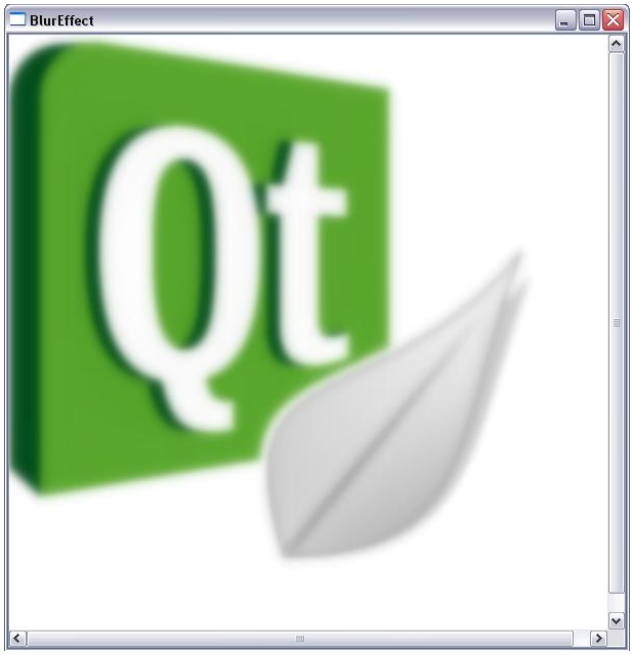
أما التابع setBlurRadius(qreal) فهو مقدار درجة الضبابية (التأثير الضبابي) على الكائن الرسومي و القيمة الإفتراضية 5 بكسل.

ملاحظة :

جميع كائنات التأثير الرسومي لا تتأثر بتغيير قيمة المنهج للعنصر الرسومي scale لأن قيمة وسيط المنهج setBlurRadius تعطى إلى نظام الإحداثيات المستخدم و بدوره يحوله إلى نظام الإحداثيات المستخدم .

ثمّ أضفنا للعنصر pixmap التأثير الضبابي , و أضفنا العنصر الرسومي pixmap إلى كائن المشهد scene و عرضناه داخل كائن عرض المشهد view .

نفذ التطبيق سوف ترى الشكل (3.4) :



الشكل 3.4

- **المثال الثاني تأثير الظل QGraphicsDropShadowEffect.**

عدل الرمّاز السابق ليصبح كالأتي :

```
QGraphicsView view;
QGraphicsScene* scene= new QGraphicsScene(0,0,600,600);
QGraphicsPixmapItem* pixmap= new QGraphicsPixmapItem();
pixmap->setPixmap(QPixmap(":/png/qt"));
QGraphicsDropShadowEffect* shadowEffect= new
QGraphicsDropShadowEffect;
shadowEffect->setColor(QColor(qRgb(255,0,0)));
shadowEffect->setOffset(10,10);
shadowEffect->setBlurRadius(10);
pixmap->setGraphicsEffect(shadowEffect);
scene->addItem(pixmap);
```

```
view.setScene(scene);

view.show();
```

صرحنا عن مؤشر لكائن تأثير الظل QGraphicsDropShadowEffect يدعى shadowEffect , وضعنا لون الظل الأحمر , ثمّ عدلنا إزاحة الظل لتصبح 10 على كل من المحور الأفقي و العمودي بوساطة المنهج setOffset القيمة الافتراضية 8 بكسل إي سيظهر ظل بلون أحمر تحت حد الرسم بمقدار 10 بكسل على المحورين الأفقي و العمودي , مقدار التأثير 10 بكسل .

نفذ التطبيق سوف يظهر كما في الشكل (3.5) :

الشكل 3.5

- المثال الثالث تأثير التظليل QGraphicsColorizeEffect .

عدل الرمّاز السابق ليصبح كالأتي :

```
QGraphicsView view;
```

```
    QGraphicsScene* scene= new QGraphicsScene(0,0,600,600);
    QGraphicsPixmapItem* pixmap= new QGraphicsPixmapItem();
    pixmap->setPixmap(QPixmap(":/png/qt"));
    QGraphicsColorizeEffect* colorizeEffect= new
QGraphicsColorizeEffect;
,0, 255)));0    colorizeEffect->setColor(QColor(qRgb(
    colorizeEffect->setStrength(0.5);
    pixmap->setGraphicsEffect(colorizeEffect);
    scene->addItem(pixmap);
    view.setScene(scene);
    view.show();
```

صرحنا عن مؤشر لكائن تأثير التظليل QGraphicsColorizeEffect , وضعنا لون التظليل أزرق , ثمّ وضعنا مقدار قوة التظليل 0.5 بوساطة المنهج setStrength(qreal) أصغر قيمة لهذا المنهج هي 0.0 و أكبر قيمة هي 1.0 .

نفذ التطبيق سوف ترى كما في الشكل (3.6) :

الشكل 3.6

- **المثال الرابع تأثير الشفافيّة QGraphicsOpacityEffect .**

عدل الرمّاز السابق ليصبح كالأتي :

```
QGraphicsView view;

QGraphicsScene* scene= new QGraphicsScene(0,0,600,600);

QGraphicsPixmapItem* pixmap= new QGraphicsPixmapItem();

pixmap->setPixmap(QPixmap(":/png/qt"));

QGraphicsOpacityEffect * opacityEffect= new
QGraphicsOpacityEffect ;

opacityEffect->setOpacity(0.5);

pixmap->setGraphicsEffect(opacityEffect);

scene->addItem(pixmap);

view.setScene(scene);

view.show();
```

المنهج setOpacity(qreal) لتضع مقدار الشفافية , قيمة الشفافيّة يجب أن تكون ضمن المجال "0.0" و "1.0" .

يوجد منهج آخر هام للكائن QGraphicsOpacityEffect يدعى setOpacityMask(QBrush) يستخدم من أجل نمط تدرج الشفافية لأي عنصر رسومي .

نفذ التطبيق سوف ترى كما في الشكل (3.7) :

الشكل 3.7

انتهينا فقرة كائنات التأثير , لننتقل إلى فقرة كائن القلم QPen .

❖ كائن القلم QPen :

كائن القلم QPen هو كائن الرسم التأسيسي للعنصر الرسومي أي القلم هو حد العنصر الرسومي من مستطيل , مضلع , قطع الخ ..

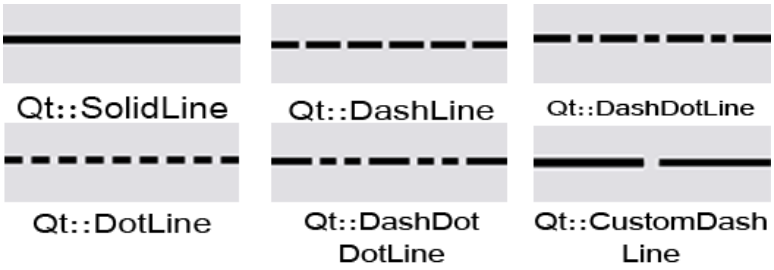
عرضه الإفتراضي 1 بكسل و لونه افتراضيا أسود .

نستطيع تغير عرضه و لونه من خلال المنهجان التاليان :

لتغيير عرضه استخدم المنهج QPen::setWidth(int) .

لتغيير لونه استخدم المنهج QPen::setColor(QColor) .

يوجد منهج يدعى QPen::setStyle(Qt::PenStyle) , يستخدم لوضع نمط حد القلم أي كخط منقط أو خط عبارة عن شحط (dashes) متتالية أو خط مصمت الخ .. انظر جدول التعداد Qt::PenStyle نمط مظهر القلم :



مثال بسيط لكيفيّة استخدام كائن القلم , في المثال سوف نرسم بوساطة الكائن QPolygon بحيث تكون حدوده هي مثيل كائن القلم المصرح عنه , انظر الرمّاز :

```
QPolygon pol;
pol.append(QPoint(10,10));
pol.append(QPoint(100,150));
pol.append(QPoint(150,75));
pol.append(QPoint(200,150));
pol.append(QPoint(300,10));
pol.append(QPoint(10,10));
QPen pen;
pen.setWidth(3);
pen.setColor(qRgb(0,0,255));
pen.setStyle(Qt::DashDotLine);
QGraphicsPolygonItem polygonItem;
polygonItem.setPen(pen);
QGraphicsView* view = new QGraphicsView();
QGraphicsScene* scene = new QGraphicsScene(0,0,400,400);
scene->setBackgroundBrush(QColor(qRgb(15,150,0)));
polygonItem.setPolygon(pol);
scene->addItem(&polygonItem);
```

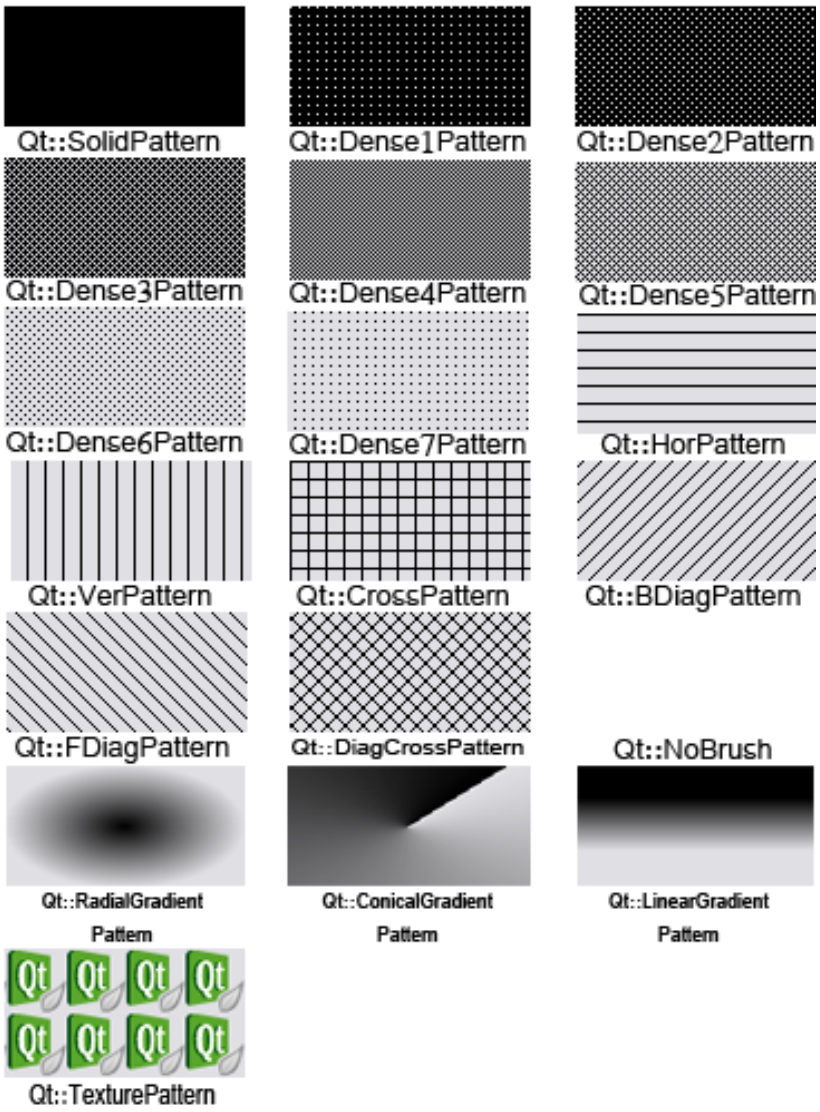
```
view->setScene(scene);

view->setGeometry(200,200,500,500);

view->show();
```

صرحنا عن مثيل للكائن QPolygon والذي سوف يحوي نقاط رؤوس المضلع , و صرحنا عن مثيل لكائن عنصر المضلع الرسومي QGraphicsPolygonItem , صرحنا أيضا عن مثيل لكائن القلم QPen وضعنا قيمة عرضه هي 3 بكسل و لونه أزرق و نمط حد القلم هي Qt::DashDotLine.

نفذ التطبيق , انظر الشكل (3.8) .

❖ كائن الفرشاة QBrush :

يستخدم كائن الفرشاة لملئ مساحة ما ضمن الأشكال الرسومية يمكن أن تكون عبارة عن لون أو صورة ما أو ألوان متدرجة , لها أنماط لشكل التعبئة , انظر الجدول :



- **مثال لاستخدام كائن الفرشاة .**

سوف تكتب رمّاز لرسم مستطيل يأخذ مساحة النافذة بالكامل يكون نمط حد القلم له Qt::NoPen أي لا حد له ونمط الفرشاة Qt::SolidPattern أي يمسح كامل مساحة المستطيل باللون الذي نضعه لكائن الفرشاة , نستخدم لرسم المستطيل الحدث paintEvent .

الرمّاز :

```
void Widget::paintEvent(QPaintEvent *paintEvent)

{

   QPainter painter(this);
```

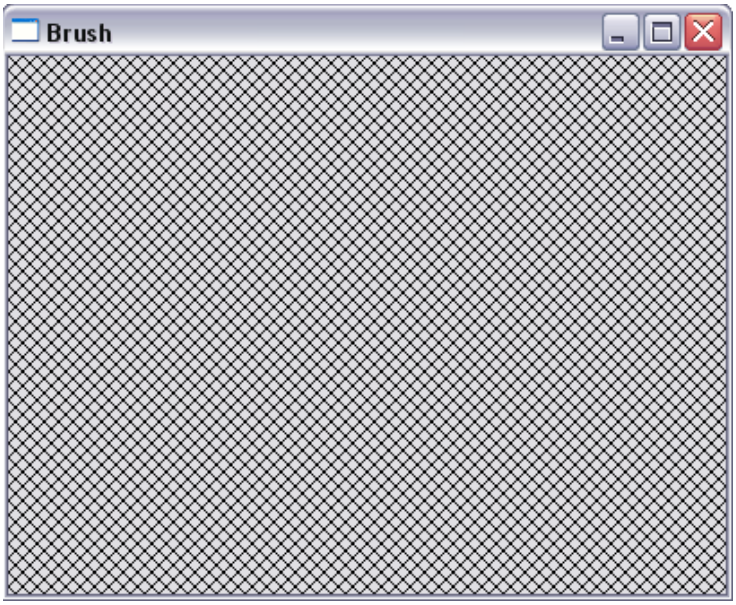
```
    QBrush brush;
    brush.setStyle(Qt::SolidPattern);
    brush.setColor(Qt::black);
    painter.setBrush(brush);
    painter.setPen(Qt::NoPen);
    painter.drawRect(0,0,paintEvent->rect().width(),paintEvent-
>rect().height());
}
```

مثلما تلاحظ أن نمط شكل التعبئة الخاصة بمثيل كائن الفرشاة هو Qt::SolidPattern أي نمط مصمت , عند تنفيذ الرمّاز سوف ترى أن النافذة بالكامل لونه خلفيتها سوداء .

الآن عدّل نمط شكل التعبئة إلى Qt::DiagCrossPattern أي السطر البرمجي الخاص بوضع نمط شكل التعبئة سيصبح :

```
brush.setStyle(Qt::DiagCrossPattern);
```

نفذ التطبيق سوف ترى كما في الشكل (3.8) :



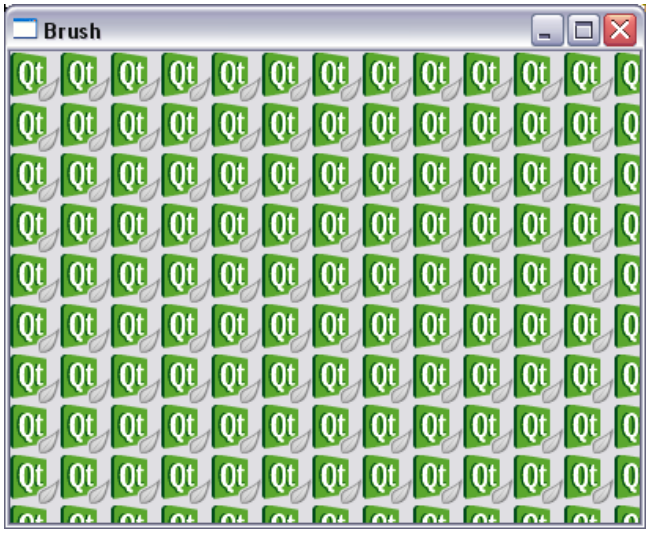
الشكل 3.8

ليكن نمط تعبئة الفرشاة عبارة عن صورة نحن نختارها أي النمط Qt::TexturePattern .

عدل الرمّاز السابق ليصبح كالآتي :

```
void Widget::paintEvent(QPaintEvent *paintEvent)
{
   QPainter painter(this);
   QBrush brush;
   brush.setStyle(Qt::TexturePattern);
   brush.setTexture(QPixmap(":/imgs/qt"));
   painter.setBrush(brush);
   painter.setPen(Qt::NoPen);
   painter.drawRect(0,0,paintEvent->rect().width(),paintEvent-
>rect().height());
}
```

وضعنا نمط شكل التعبئة Qt::TexturePattern ثمّ حددنا الصورة التي نريدها كنمط تعبئة للفرشاة بوساطة المنهج setTexture(const QPixmap &pixmap) التابع لكائن الفرشاة QBrush , نفذ التطبيق سوف ترى كما في الشكل (3.9) :



الشكل 3.9

سوف تجد هذا المثال بالقرص المرفق .

- أنماط اللون المتدرج لدى شكل التعبئة لكائن للفرشاة :

يوجد أنماط تعبئة يكون عبارة عن لون أو أكثر متدرج (gradient) , بوجد ثلاث أشكال لها وهم :

1 - Qt::RadialGradientPattern ذو نمط تدرج نصف قطري – شعاعي .

2 - Qt::ConicalGradientPattern ذو نمط تدرج مخروطي .

3 - Qt::LinearGradientPattern ذو نمط تدرج خطي .

يقابل كل نمط من هؤلاء الأنماط كائن من Qt يعطى له قيم التدرج و خصائصه التي نريد من ثمّ نضعه داخل كائن الفرشاة وبالتالي يقوم كائن الفرشاة بمسح التعبئة بلون متدرج (نفس كائن التدرج الذي هيئناه ووضعناه لكائن الفرشاة).

وكائنات تدرج اللون هي :

كائن اللون المتدرج	نمط شكل التعبئة الذي يقابله	وصفه

كائن تدرج شعاعي الشكل	Qt::RadialGradientPattern	QRadialGradient
كائن تدرج مخروطي الشكل	Qt::ConicalGradientPattern	QConicalGradient
كائن تدرج خطي الشكل	Qt::LinearGradientPattern	QLinearGradient

الصف الأب لكائنات التدرج هو QGradient .

مثال لاستخدام كائنات التدرج :

سوف تكتب رمّاز لرسم مستطيل يأخذ مساحة النافذة بالكامل يكون شكل التعبئة عبارة عن لون متدرج بين الأسود من الجهة العليا و اللون الأبيض من الجهة السفلى .

الرمّاز :

```
void Widget::paintEvent(QPaintEvent *paintEvent)
{
   QPainter painter(this);
paintEvent->rect().height()); , , 00   QLinearGradient gradient(0,
   gradient.setColorAt(0, Qt::black);
   gradient.setColorAt(1,Qt::white);
   QBrush brush(gradient);
   painter.setBrush(brush);
   painter.drawRect(0,0,paintEvent->rect().width(),paintEvent-
>rect().height());
}
```

صرحنا عن مثيل للكائن QLinearGradient (تدرج اللون بشكل خطي) يدعى gradient وضعنا في بنائه قيمة موضع البداية و قيمة موضع النهاية لمنطقة التدرج التي نريدها على المحورين الأفقي و العمودي , هنا نريد التدرج بشكل عمودي على محور Y لذا وضعنا قيم موضع البداية و النهاية للتدرج :

نقطة البداية هي x = 0 , y = 0 .

نقطة التوقف x = 0 , y مساوي لارتفاع النافذة .

أي ستصبح منطقة التدرج من بداية النافذة من الأعلى إلى نهايتها من الأسفل .

ووضعنا لون التدرج الأول هو الأسود ولون التدرج الآخر هو الأبيض بوساطة المنهج setColorAt صيغته العامة هي :

setColorAt (qreal position, const QColor & color)

ينشأ هذا المنهج نقطة توقف حيث يكون موقعها هو قيمة الوسيط position يقبل القيم التي تكون ضمن المجال 0.0 إلى 1.0 , و لون المساحة الموجودة من موقع آخر نقطة توقف إلى نقطة التوقف الحالية هو قيمة الوسيط color .

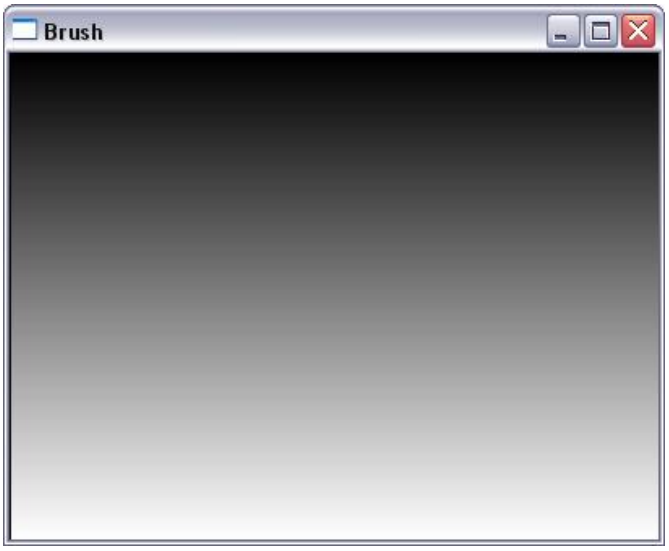
وفي الرمّاز وضعنا موقع التدرج الأول هو 0 ولونه أسود , و موقع التدرج الثاني هو 1 و لونه الأبيض .

أيّ سوف يبدأ التدرج من أعلى النافذة (المساحة المعطاة لبنائه) باللون الأسود مزامنا مسحه لخلفية النافذة (المساحة المعطاة) مزجه مع اللون الأبيض حتى يصل إلى نهاية النافذة يكون قد أصبح لونه أبيض مصمت .

لنعد إلى الرمّاز صرحنا عن مثيل لكائن الفرشاة يدعى brush وضعنا لبنائه الكائن gradient ووضعنا قيمة الفرشاة لكائن الرسم painter المثيل brush ثمّ رسمنا مستطيل يأخذ كامل مساحة النافذة بوساطة السطر البرمجي التالي :

```
painter.drawRect(0,0,paintEvent->rect().width(),paintEvent-
>rect().height());
```

عند نفذ التطبيق سوف ترى الشكل (3.10) :



الشكل 3.10

ملاحظة :

نستطيع أن نعطي أكثر من نقطتي توقف لكائن التدرج , ضف إلى الرمّاز السابق السطر البرمجي التالي و نفذ التطبيق وانظر إلى التدرج المعطى :

```
gradient.setColorAt(0, Qt::black);
gradient.setColorAt(0.5, Qt::red);
gradient.setColorAt(1,Qt::white);
```

لرسم ترج لوني بشكل شعاعي غير الرمّاز السابق ليصبح :

```
void Widget::paintEvent(QPaintEvent *paintEvent)
{
  QPainter  painter(this);
  QRadialGradient  gradient(paintEvent->rect().width()/2,
paintEvent->rect().height()/2,paintEvent->rect().height()/2);
   gradient.setColorAt(0, Qt::black);
   gradient.setColorAt(0.5, Qt::red);
   gradient.setColorAt(1,Qt::white);
```

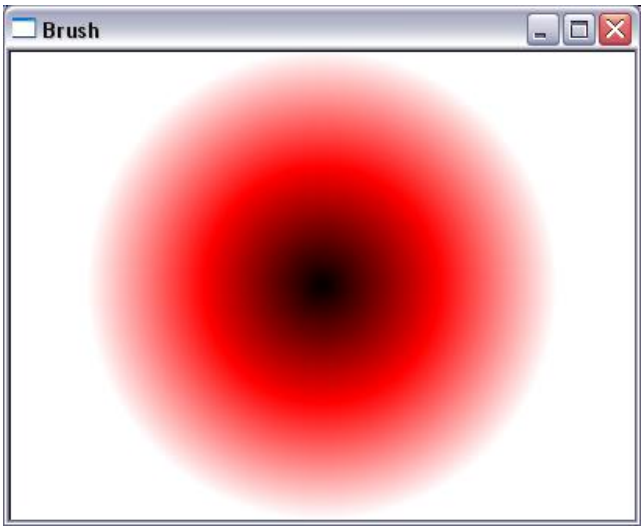
```
   QBrush brush(gradient);

   painter.setBrush(brush);

   painter.drawRect(0,0,paintEvent->rect().width(),paintEvent-
>rect().height());

}
```

صرحنا عن مثيل للكائن QRadialGradient يدعى gradient مررنا لوسطاء بنائه القيم التالية :

الوسيط الأول هو مركز التدرج مررنا له قيمة موقع النقطة التي تقع في منتصف النافذة .

الوسيط الثاني هو نصف قطر التدرج والذي مركزه ذاته مركز التدرج مررنا له قيمة نصف ارتفاع النافذة .

نفذ التطبيق سوف ترى الشكل (3.11) :



الشكل 3.11

أما بالنسبة لكيفيّة استخدام كائن التدرج المخروطي الشكل QConicalGradient عدل السطر البرمجي التالي الموجود داخل الرمّاز السابق :

```
   QRadialGradient gradient(paintEvent->rect().width()/2,

paintEvent->rect().height()/2,paintEvent->rect().height()/2);
```

ليصبح كالتالي :

```
QConicalGradient gradient(paintEvent->rect().width()/2,

paintEvent->rect().height()/2,90);
```

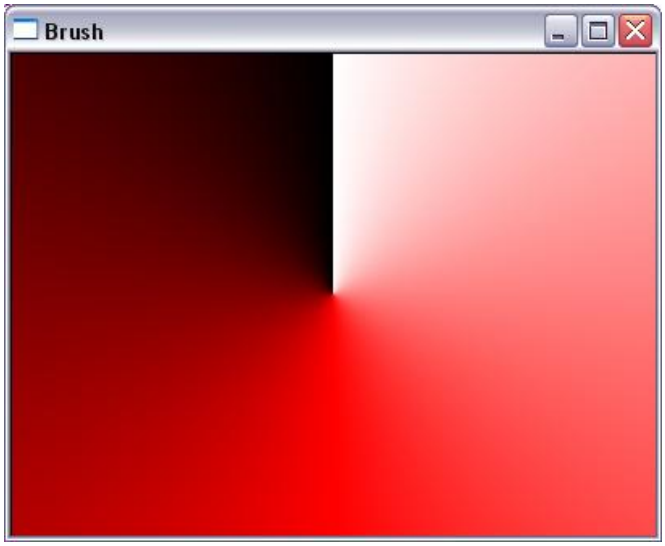
صرخنا عن مثيل للكائن QConicalGradient يدعى gradient مررنا بوسطاء بنائه القيم التالية :

الوسيط الأول هو cx أي مركز المحور الأفقي للشكل المخروطي المتدرج اللون .

الوسيط الثاني cy أي مركز المحور العمودي .

الوسيط الثالث هو قيمة زاوية بداية المسح مقدر بالدرجة .

بعد التعديل نفذ التطبيق لترى الشكل (3.12) :



الشكل 3.12

- **نمط انتشار كائن التدرج :**

لاحظت عندما كتبنا رمّاز لكائن التدرج الشعاعي الشكل أنه يوجد مساحة خالية حول دائرة التدرج من دون تعبئة بأي تدرج , إذا كنا نريد أن تعبئ هذه المساحة بالتدرج ذاته (الموضوع حاليا) فنستطيع ذلك بوساطة المنهج setSpread(Spread) التابع للكائن QGradient ,

الوسيط Spread هو عبارة عن تعداد يوجد داخله ثلاث قيم كل قيمة تدل على نمط انتشار التعبئة للتدرج , التعداد QGradient::Spread :

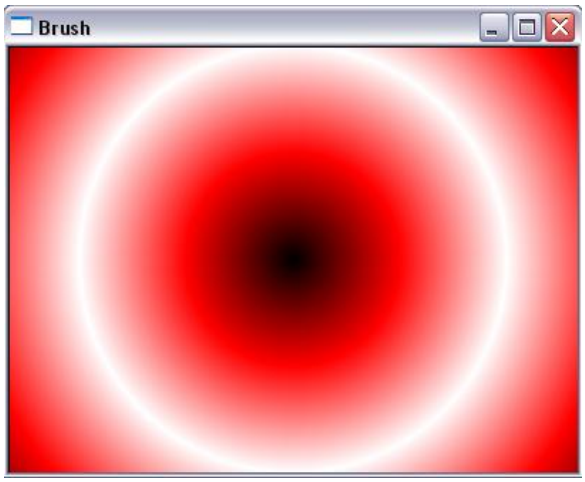
QGradient::PadSpread	**النمط الإفتراضي , سوف يملئ المساحة المعطاة بالتدرج اللوني وترك المساحة الباقية من الكائن الذي نرسم عليه على لونها الإفتراضي**
QGradient::RepeatSpread	**سوف يعيد رسم نفس التدرج والتي تكون بدايته هي نهاية المساحة المعطاة**
QGradient::ReflectSpread	**سوف يعيد رسم عكس التدرج والتي تكون بدايته هي نهاية المساحة المعطاة**

انظر الرّماز التالي :

```
void Widget::paintEvent(QPaintEvent *paintEvent)
{
  QPainter painter(this);
   QRadialGradient gradient(paintEvent->rect().width()/2,
 paintEvent->rect().height()/2,paintEvent->rect().height()/2);
   gradient.setColorAt(0, Qt::black);
   gradient.setColorAt(0.5, Qt::red);
   gradient.setColorAt(1,Qt::white);
   gradient.setSpread(QGradient::ReflectSpread);
   //gradient.setSpread(QGradient::RepeatSpread);
   //gradient.setSpread(QGradient::PadSpread);
   QBrush brush(gradient);
```

```
    painter.setBrush(brush);
    painter.drawRect(0,0,paintEvent->rect().width(),paintEvent-
>rect().height());
}
```

وضعنا هنا نمط انتشار التدرج بشكل عكسي , انظر الشكل (3.13) :



الشكل 3.13

إلى هنا نكون أنهينا فقرة كائن الفرشاة , لننتقل إلى التكلم عن الطباعة و كائناتها .

❖ الطباعة :

في أغلب التطبيقات التي نستخدمها يوجد خيارات لطباعة بيانات ما إن كان نص أو صورة الخ ..

لذا لمن المهم التكلم عن كيفية برمجة تطبيق يوجد بداخله أوامر للطباعة .

إن Qt يوجد بداخلها صفوف تستخدم من أجل الطباعة و معاينة الصفحة التي نود أن نطبعها قبل الطباعة و في تغيير إعدادات الطباعة بشكل يلبي مطلبنا الخ ...

لندخل في صلب الموضوع .

الصف QPrinter هو الصف الأساسي لجهاز الطباعة الخاص بتهيئة البيانات ومن ثمّ إعطاءها كخرج للطابعة .

مثال لإنشاء تطبيق يحتوي على كائن QTextEdit و أسفله زر للطباعة عند النقر عليه تظهر نافذة الحوار الخاصة بالطباعة لطباعة النص الغني الموجود داخل الكائن QTextEdit :

بعد إنشاء مشروع Qt Gui Application الصف الأساس للنافذة هو QMainWindow أضف كائن textEdit من صندوق الأدوات و أسفله ضع كائن pushButton , لإنشاء مقبس متصل بحدث النقر من خلال Qt Creator نفذ الآتي اضغط على كائن الزر بزر الفأرة الأيمن و اختار

go to slot... سوف تظهر نافذة بعنوان Go to slot بداخلها قائمة تحوي على أحداث الزر مكتوب أعلاها Select signal اختر منها ()clicked أي حدث النقر عند الضغط على زر Ok سوف تأخذك مباشرة لداخل رمّاز حدث النقر (المقبس المتصل مع حدث النقر) .

اكتب داخله الرمّاز التالي :

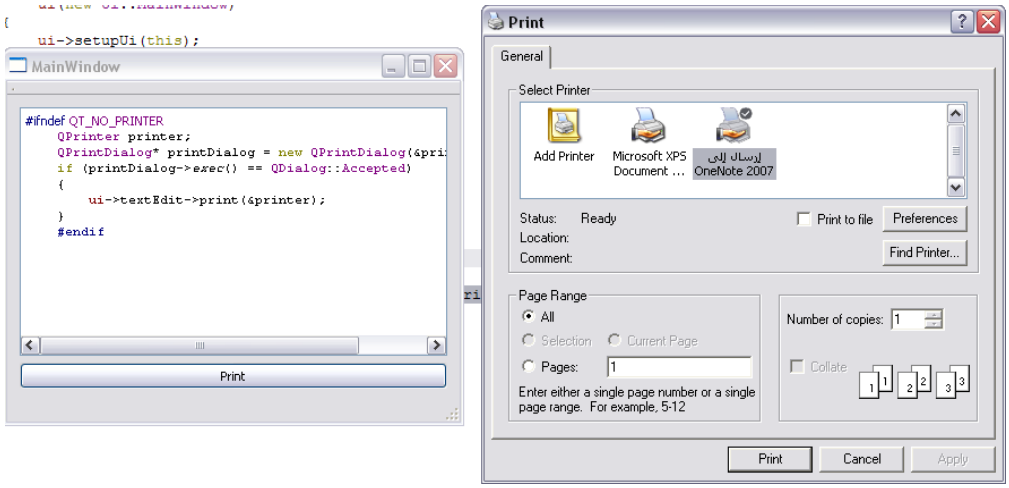
```
#ifndef QT_NO_PRINTER

QPrinter printer;

QPrintDialog* printDialog = new QPrintDialog(&printer,this);

if (printDialog->exec() == QDialog::Accepted)

{

   ui->textEdit->print(&printer);

}

#endif
```

نبدأ بالتوجيه ما قبل الترجمة الذي يختبر إذا لم يكن معرف الصف الذي يليه , هنا سوف يقوم باختبار إذا لم يكن النظام الحامل للتطبيق معرف داخله كائنات الطباعة (كبعض أنظمة الخاصة ببعض الأجهزة الذكية) سوف لن ينفذ الرمّاز الذي يليه الموجود بين التوجيهان ifndef# و التوجيه endif# .

أخذنا مثيل للصف QPrinter يدعى printer و مؤشر لكائن QPrintDialog الخاص بإظهار نافذة حوار تحوي أجهزة الطابعة المتاح استخدامها (المتصلة بالجهاز الذي يعمل عليه التطبيق) و خصائص و إعدادات الصفحات المهيئة للطباعة مررنا لبناء هذا الكائن مرجع المثيل printer (الجهاز الرسومي) و مررنا أيضا المؤشر الذاتي this أي الصف الأب له هو

الصف الحالي (mainwindow) , ثمّ أظهرنا نافذة خصائص الطباعة باستخدام المنهج exec التابع للكائن QPrintDialog و اختبرنا إن ضغط على زر موافق بعد تعديل الإعدادات التي يود تغييرها , داخل جسم الشرط استخدمنا المنهج print التابع لكائن النص QTextEdit لرسم النص الموجود داخله على مثيل كائن الطابعة printer لإظهاره كخرج على جهاز الطابعة.

عند تنفيذ التطبيق و الضغط على زر الطباعة (print) سوف ترى الشكل (3.14):



الشكل 3.14

لرسم على الصفحة التي نود طباعتها أشكال هندسية أو صورة أو الخ.. بشكل مباشر يجب علينا أن نستخدم الصف QPainter بعد أن نمرر لوسيطه الجهاز الرسومي و هو هنا مثيل الصف QPrinter , لنرى كيفية ذلك من خلال المثال التالي.

لنرسم دائرة باللون أحمر على الصفحة التي نود طباعتها باستخدام الصف QPainter , عدل الرمّاز السابق الموجود داخل مستقبل حدث النقر ليصبح كالآتي :

```
#ifndef QT_NO_PRINTER

QPrinter printer;

QPrintDialog* printDialog = new QPrintDialog(&printer,this);

if (printDialog->exec() == QDialog::Accepted)

{

   QPainter painter(&printer);

   painter.setPen(Qt::NoPen);
```

```
    painter.setBrush(QBrush(Qt::red));
    painter.drawEllipse(0,0,500,500);
  }
  #endif
```

صرحنا عن مثيل للصف QPainter يدعى painter و مررنا لوسيطه المثيل printer وهو الجهاز الرسومي الذي سوف يكتب عليه بوساطة المثيل painter , ثمّ وضعنا القلم و الفرشاة بالون الأحمر و رسمنا قطع ناقص (هنا دائرة) .

نفذ التطبيق واختبر النتيجة التي سوف تظهر معك ضمن الصفحة المطبوعة .

نستطيع رسم كائن من كائنات واجه المستخدم (widgets) على صفحة الطباعة بوساطة المنهج QWidget::render , لكن سوف لن يظهر على صفحة الطباعة إلا الجزء المعروض (المشاهد) من الكائن QWidget لنرى المثال التالي :

```
  #ifndef QT_NO_PRINTER
    QPrinter printer;
    QPrintDialog* printDialog = new QPrintDialog(&printer,this);
    if (printDialog->exec() == QDialog::Accepted)
    {
    QPainter painter(&printer);
    QGraphicsScene *scene = new QGraphicsScene(0,0,400,400);
    scene->addPixmap(QPixmap::grabWindow(qApp->desktop()-
>winId()));
    ui->graphicsView->setScene(scene);
    ui->graphicsView->render(&painter);
    }
  #endif
```

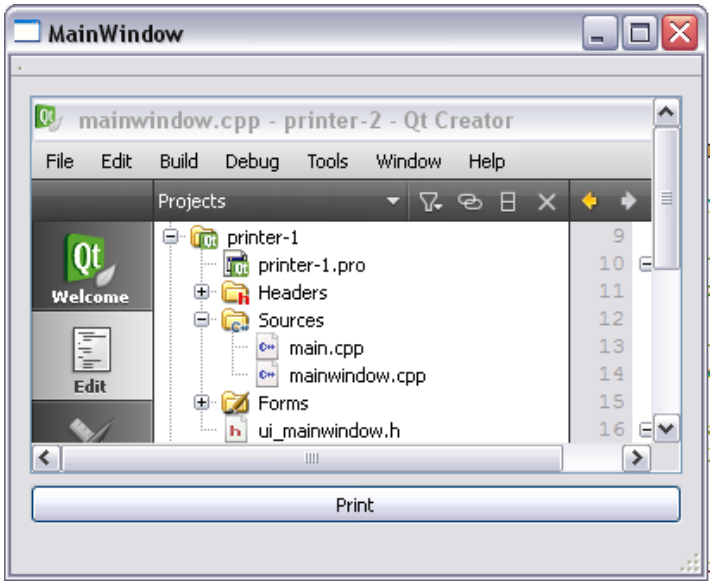
قبل تنفيذه أضف ملف الترويسة الخاص بجلب مواصفات و إعدادات سطح المكتب لنظام التشغيل :

```
#include <QDesktopWidget>
```

قد التقطنا صورة لسطح المكتب لديك بوساطة المنهج QPixmap::grabWindow والذي مررنا لوسيطه مقبض سطح المكتب بوساطة المنهج QDesktopWidget::winId .

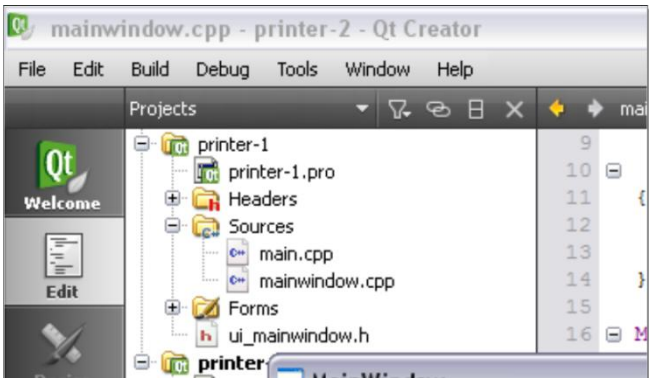
استخدمنا المنهج render لرسم الجزء المشاهد من الكائن بعد أن مررنا لوسيطه مرجع كائن الرسم painter الذي هو بدوره يعطي خرجه لجهاز الطباعة printer .

عند التنفيذ سوف يظهر الشكل (3.15) :



الشكل 3.15

عند الطباعة سوف يطبع فقط الجزء المشاهد من صورة سطح المكتب لديك الملتقطة و الموضوعة ضمن الكائن graphicsView , سوف يكون الخرج كالشكل (3.16) :



الشكل 3.16

ملاحظة : جميع كائنات المستخدم الرسومية يوجد لديها المنهج render لأنه تابع أصلا للكائن QWidget والتي تورثه جميع كائنات المستخدم الرسومية .

يوجد عدة مناهج مفيدة تابعة للصف QPrinter كإنشاء صفحة جديدة و تحديد حجم الصفحة وتغيير مساحة نافذة العرض الخ.. سوف نتكلم عنها في هذه الفقرة .

المنهج الخاص بتغير نمط الألوان للصفحة المهيئة للطباعة وهو ذو نمطين : إما أبيض و أسود إما ملون .

```
void QPrinter::setColorMode(ColorMode)
```

التعداد QPrinter::ColorMode :

صفحة ملونة	QPrinter::Color
صفحة متدرجة ألوانها بين الأبيض و الأسود	QPrinter::GrayScale

المنهج الخاص بتغيير حجم صفحة الطباعة , تستطيع أن تضع الأحجام القياسية مثل A1,A2,A3,A4 الخ..

أو حجم مخصص

```
void QPrinter::setPagerSize(PagerSize)

void QPrinter::setPagerSize(const QSizeF &PagerSize, Unit unit )
```

الوسيط PagerSize من نمط QSizeF الحجم (الارتفاع + العرض) يأخذ القيمة float .

الوسيط unit وحدة القياس انظر الجدول , التعداد QPrinter::Unit :

```
QPrinter::Millimeter

QPrinter::Point

QPrinter::Inch

QPrinter::Pica

QPrinter::Didot

QPrinter::Cicero
```

QPrinter::DevicePixel

تحديد الهامش للصفحة المهيئة للطباعة :

void QPrinter::setPageMargins (qreal left, qreal top, qreal right, qreal bottom, Unit unit)

إنشاء صفحة جديدة تضاف إلى سلسلة الصفحات المهيئين للطباعة :

bool QPrinter::newPage ()

لإنشاء خرج على ملف pdf انظر الرمّاز التالي:

```
QPrinter printer;
printer.setOutputFormat(QPrinter::PdfFormat);
printer.setOutputFileName("table.pdf");
QPainter painter;
painter.begin(&printer);
QGraphicsScene *scene = new QGraphicsScene(0,0,400,400);
scene->addPixmap(QPixmap::grabWidget(ui->tableWidget));
ui->graphicsView->setScene(scene);
ui->graphicsView->render(&painter);
painter.end();
```

سوف يكون خرج الصفحة المهيئة للطباعة على ملف من نوع pdf وليس على جهاز الطابعة مباشرة .

القيمة الإفتراضية لتنسيق الخرج QPrinter::NativeFormat أي إظهار الخرج على جهاز الطابعة مباشرة.

❖ معاينة الصفحة قبل الطباعة :

الآن إذا كنا نريد أن نعاين الصفحة المهيئة للطباعة قبل طباعتها يتوجب علينا أن نستخدم الصف QPrintPreviewDialog الخاص بإظهار تطبيق فرعي يوجد بداخله الصفحات المراد طباعتها والتي قد هيئناها مسبقا لنرى كيفية استخدامه .

عدّل الرمّاز الموجود داخل مقبس حدث النقر للزر ليصبح كالآتي :

```
QPrinter printer;

QPrintPreviewDialog  printPreviewDialog(&printer, this);

printPreviewDialog.setGeometry(0,30,qApp->desktop()->screen()->width(),

qApp->desktop()->screen()->height()-60);


connect(&printPreviewDialog,SIGNAL(paintRequested(QPrinter*)),this,

SLOT(print(QPrinter*)));

printPreviewDialog.exec();
```

صرحنا عن مثيل لكائن المعاينة QPrintPreviewDialog يدعى printPreviewDialog مررنا لبنائه الوسيط الأول مرجع كائن الطابعة printer و المؤشر الذاتي this أي الأب للكائن printPreviewDialog , ثمّ غيرنا حجمه و موقعه لتصبح نافذة التطبيق الفرعي الخاص بالمعاينة على ملئ الشاشة , ربطنا الحدث paintRequest(QPrinter*) بالمستقبل print(QPrinter*) , ثمّ أمرنا كائن المعاينة بإظهار نافذته .

الحدث paintRequest(QPrinter*) يقدح عند استدعاء التنفيذ لكائن المعاينة أي المنهج QPrintPreviewDialog::exec() , سوف ينفذ أولا الرمّاز الموجود داخل المستقبل paintRequest(QPrinter*) و من ثمّ يظهر النافذة بعد قراءة البيانات الموجودة داخل كائن الطابعة QPrinter بعد الكتابة عليه داخل رمّاز المستقبل paintRequest(QPrinter*) .

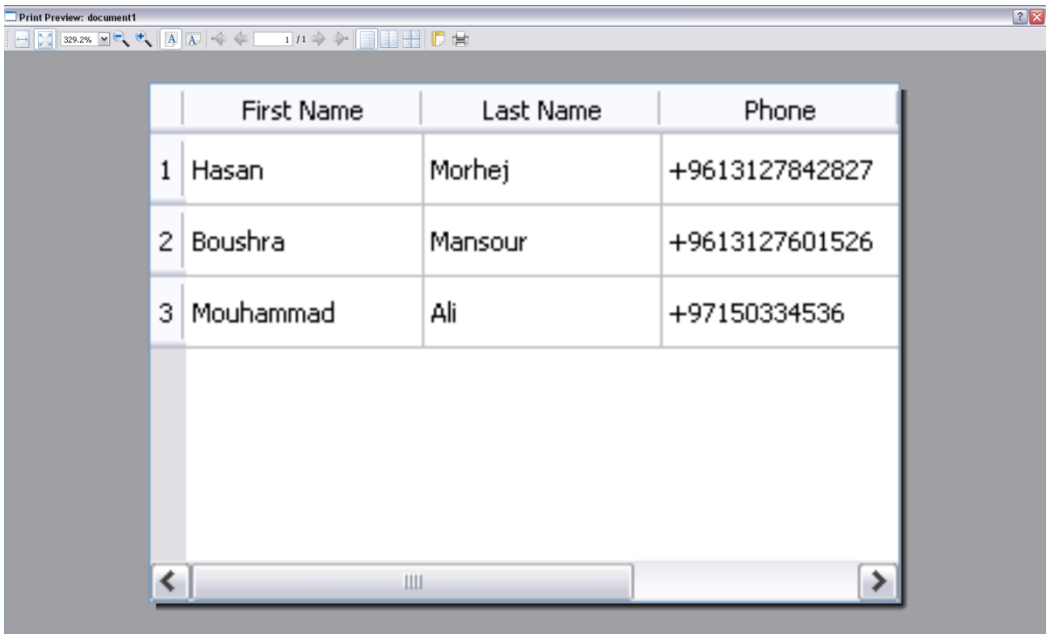
أضف إلى ملف الترويسة الخاص بالتطبيق mainwindow.h المستقبل print(QPrinter*) :

```
private slots:

void print(QPrinter*);
```

حقق المستقبل print داخل الملف mainwindow.cpp :

```
void MainWindow::print(QPrinter *printer)
{
  #ifndef QT_NO_PRINTER
     ui->tableWidget->render(printer);
   #endif
}
```

عند التنفيذ سوف تظهر الصفحة المهيئة للطباعة داخل التطبيق الفرعي الخاص بالمعاينة كالشكل (3.18) :



الشكل 3.18

انتهت فقرة معاينة الصفحة قبل الطباعة .

لننتقل إلى التكلم عن كائنات Animation .

❖ برمجة كائنات رسومية متحركة في Qt :

تكلمنا في الفقرات السابقة عن برمجة رسوميات ثابتة - ساكنة غير متحركة كرسم خط , دائرة , مضلع الخ ..

سوف نتحدث هنا عن برمجة رسوميات متحركة ككائن رسومي دائري الشكل يتجه نحو هدف معين بحركة تتسارع مع مرور الوقت , هنا نتحدث عن نوعين لبرمجة الرسوميات المتحركة النوع الأول هو عبارة عن شريط زمن يوجد بداخله إطارات سيرها يتزامن مع الوقت , النوع الثاني نستطيع تحريك الكائن الرسومي من خلال إعطاء الكائن QPropertyAnimation قيمة بدائية و قيمة نهائية للخاصية التي نريد أن نعدل قيمها للكائن الرسومي المراد إحداث حركة له , أي مهمة الكائن QPropertyAnimation هي تغيير قيمة خاصية ما للكائن الرسومي المتصل به بالتزامن مع المدة الموضوعة له.

• الإطارات المتزامنة مع الوقت بشريط الزمن :

إذا كنت ممن عمل مسبقا على برامج التصميم أو المونتاج سوف تلاحظ أن شريط الزمن الموجود داخلها هو نفس عمل شريط الزمن الذي سوف نتكلم عنه .

شريط الزمن هو عبارة عن سلسلة من الإطارات المتتالية ضمن وقت معين , يتزامن المرور على هذه الإطارات بمرور الوقت ضمن شريط الزمن لينتج حركة رسومية ما (فيلم) , فيQt يدعى الصف الأساسي لشريط الزمن QTimeLine حيث نستطيع إنشاء إطارات بداخله تتالى ضمن مدة ما و بالتفاعل مع الكائن QGraphicsItemAnimation ينتج كائن رسومي متحرك .

مثال استخدام كائن شريط الزمن QTimeLine و تفاعله مع الكائن QGraphicsItemAnimation العنصر الرسومي المتحرك , يوجد لدينا عنصر رسومي مستطيل الشكل نريد تحريكه بشكل أفقي من أقصى الجهة اليسرى للنافذة إلى أقصى الجهة اليمنى , أنشئ مشروع فارغ emptyProject أضف عليه ملف يدعى main.cpp و اكتب بداخله الرمّاز الخاص بالكتلة main :

```
#include <QtGui/QApplication>

#include "timeLine.h"

int main(int argc, char *argv[])
```

```
{
   QApplication app(argc, argv);
   timeLine  timeLine;
   timeLine.show();
   return app.exec();
}
```

أنشئ ملف ترويسة اسمه timeLine.h يحوي صف يدعى timeLine , سوف نكتب داخل هذا الصف الرمّاز الخاص بإنشاء مشهد رسومي والذي يحوي بدوره المستطيل المتحرك, انظر الرمّاز :

```
#include <QGraphicsItem>
#include <QGraphicsItemAnimation>
#include <QTimeLine>
#include <QGraphicsScene>
#include <QGraphicsView>

class timeLine : public QGraphicsView
{
public :
   timeLine(QGraphicsScene *scene = 0) : QGraphicsView(scene)
   {
      QGraphicsItem *rect = new QGraphicsRectItem(0,0,40,40);

      QTimeLine *timer = new QTimeLine(4000);
```

```
    timer->setFrameRange(0, 2000);
    timer->setCurveShape(QTimeLine::CosineCurve);
    timer->setDirection(QTimeLine::Forward);
    timer->setLoopCount(0);
    QGraphicsItemAnimation *animationItem = new
QGraphicsItemAnimation;
    animationItem->setItem(rect);
    animationItem->setTimeLine(timer);

    for (int i = 0; i <= 210 ; ++i){
       animationItem->setPosAt(i / 210.0, QPointF(i, 0));
  }

    QGraphicsScene *scene1 = new QGraphicsScene();
    scene1->setSceneRect(0, 0, 250, 250);
    scene1->addItem(rect);
    setScene(scene1);
    timer->start();
  }
};
```

أنشئنا صف يدعى timeLine مشتق من الصف QGraphicsView داخل بنائه صرحنا عن مؤشر لكائن عنصر رسومي مستطيل الشكل QGraphicsRectItem حجمه 40 . 40 يدعى rect, و صرحنا عن مؤشر لكائن شريط الزمن QTimeLine يدعى timer وضعنا المدة 4000 ميلي ثانية لوسيط بنائه أي شريط الزمن يحوي المشهد الرسومي المتحرك مدته 4 ثوان , ثمّ أضفنا 2000 إطار داخله يبدأ توالي الإطارات من الإطار الأول إلى الإطار الأخير

أي الإطار 2000 هكذا سوف يكون مدة بقاء الإطار المشاهد (الإطار الواحد) 2 ميلي ثانية بالاعتماد على مدة شريط الزمن , وضعنا قيمة شكل منحني الحركة QTimeLine::CosineCurve يوجد عدة أنماط لمنحني الحركة انظر الجدول , التعداد QTimeLine::CurveShape :

تبدأ الحركة ببطء ثمّ يتزايد سرعتها	QTimeLine::EaseInCurve
تبدأ الحركة بسرعة معتدلة ثمّ تنتهي ببطء	QTimeLine::EaseOutCurve
تبدأ الحركة ببطء ثمّ تصبح السرعة معتدلة ثمّ تعود تتحرك ببطء	QTimeLine::EaseInOutCurve
تتزايد السرعة بشكل تدريجي	QTimeLine::LinearCurve
انعطاف الحركة بشكل جيبي	QTimeLine::SineCurve
انعطاف الحركة بشكل تجيبي	QTimeLine::CosineCurve

أما بالنسبة لاتجاه القراءة في شريط الزمن فوضعنا يقرأ من آخر إطار إلى أول إطار أي إتجاه القراءة في شريط الزمن بشكل خلفي , يوجد نوعين إتجاه انظر الجدول , التعداد QTimeLine::Direction :

اتجاه قراءة شريط الزمن من أول إطار إلى آخر إطار	QTimeLine::Backward
اتجاه قراءة شريط الزمن من آخر إطار إلى أول إطار	QTimeLine::Forward

إعادة قراءة شريط الزمن لا نهائيّة إي تكرار لا نهائي لأننا وضعنا قيمة وسيط المنهج setLoopCount(0) الصفر فبهذا قلنا له كرر إعادة عرض الإطارات بشكل دائم.

صرحنا عن مؤشر لكائن QGraphicsItemAnimation يدعى animation وضعنا داخله الكائن rect و شريط الزمن timer

```
QGraphicsItemAnimation *animationItem = new
QGraphicsItemAnimation;

animationItem->setItem(rect);
```

```
animationItem->setTimeLine(timer);
```

الآن يجب أن نغير موقع الكائن الرسومي rect بوساطة الرمّاز :

```
for (int i = 0; i <= 210 ; ++i){

   animationItem->setPosAt(i / 210.0, QPointF(i, 0));

 }
```

هنا حلقة تكرار بقيمة 210 دورات لأننا نريد أن يمضي كائن المستطيل بالحركة من بداية المشهد إلى نهايته, عرض المشهد 250 بكسل و عرض كائن المستطيل 40 بكسل إذا طرحناهم يصدر 210 , المنهج setPosAt(qreal step, QPointF &point) يستخدم لتغيير موقع الكائن الرسومي, الوسيط الأول step من نمط عدد حقيقي يأخذ فقط قيم المجال 0.0 إلى 1.0 وهو عبارة عن رقم الخطوة ضمن الحركة أما الوسيط الثاني point هو عبارة عن الموقع الحالي للخطوة الحالية , بما أننا نريد الحركة بشكل أفقي أي على محور X غيرنا قيمة الموقع على المحور الأفقي فقط QPointF(i, 0)

صرحنا عن مؤشر لكائن مشهد يدعى scene1 يبدأ من المحور الأفقي الأيسر و العمودي الأعلى (النقطة 0.0) بحجم يمتد حتى 250 بكسل عرضا و طولا , ثمّ أضفنا الكائن rect إلى المشهد و وضعنا المشهد لكائن العرض الرسومي أي الصف الحالي (الموجودين داخله) :

```
setScene(scene1);
```

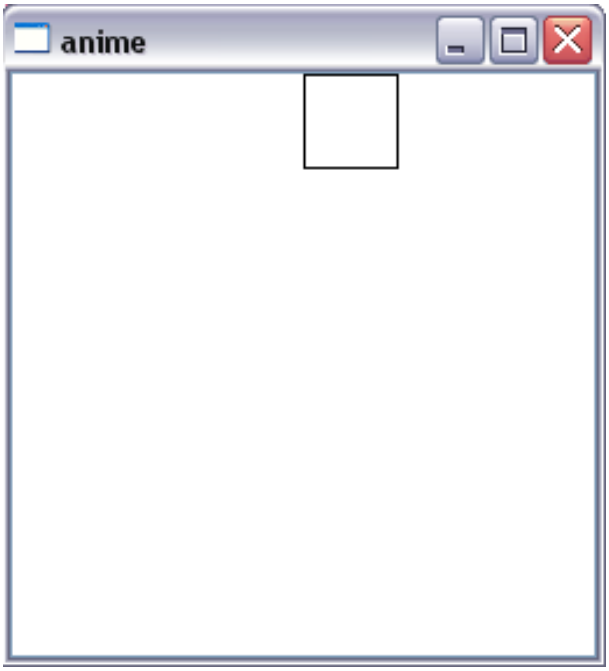
بقي أن نجعل الحركة تعمل بوساطة المستقبل start :

```
timer->start();
```

مسبقا صرحنا ضمن الكتلة الرئيسية للتطبيق main عن الصف timeLine و أظهرنا نافذة عرض الرسوميات التي تحوي المشهد الرسومي المتحرك :

```
timeLine timeLine;

timeLine.show();
```

نفذ التطبيق و اختبر النتيجة , عند التنفيذ سوف يظهر كالشكل (3.19) :



الشكل 3.19

أهم مناهج الصف QGraphicsItemAnimation الخاصة بتعديل الموقع و شكل الكائن الرسومي :

تغيير موقع الكائن	**void setPosAt (qreal step, const QPointF & point)**
تدوير الكائن بزاوية مقدارها قيمة الوسيط angle	**void setRotationAt (qreal step, qreal angle)**
إعادة تقييس حجم الكائن على المحورين الأفقي و العمودي	**void setScaleAt (qreal step, qreal sx, qreal sy)**
تحريك الكائن بشكل شفرتي المقص على المحورين الأفقي و العمودي	**void setShearAt (qreal step, qreal sh, qreal sv)**

أما بالنسبة للصف QTimeLine يوجد له حدث يقدح عند تغيير الإطارات أي عند كل تغيير إطار يقدح هذا الحدث :

void frameChanged (int frame)

أهم المستقبلات للصف QTimeLine :

void resume ()

void setPaused (bool paused)

void start ()

void stop ()

انتهينا من أول طريقة لتحريك الكائنات الرسومية في Qt والتي هي عبارة عن تتالي إطارات بالتزامن مع مدة شريط الزمن , لننتقل إلى الطريقة الثانية باستخدام الصف QPropertyAnimation.

- **الصف QPropertyAnimation :**

الصف QPropertyAnimation يستخدم لتحريك الكائنات الرسومية أو كائنات واجهة المستخدم, يرث الصف QVariantAnimation عملية تحريك الكائن تتم بوساطة تغيير قيم خصائصه المرئيّة كالموقع بمدة زمنيّة محددة , مثال لتحريك كائن ما يجب تحديد نقطة البداية (موقع الكائن الأولي) و نقطة النهاية (آخر نقطة المراد تحريك الكائن له) بالتالي سوف يقوم بتغيير قيم موقع الكائن تدريجيا (قيم المجال المحصور من النقطة الأولى للحركة إلى النقطة الأخيرة) , يوجد بداخل الصف QPropertyAnimation خصائص لحفظ منحني سير الحركة و المدة الخ...

مثال : لننشأ زر QPushButton يتحرك على المحور الأفقي من الجهة اليسرى إلى اليمنى , انظر الرمّاز :

```
   QPushButton* btn = new QPushButton(QObject::tr("Qt
Animation.."));

   btn->setGeometry(QRect(0,0,300,300));

   btn->show();

   QPropertyAnimation* anime = new
QPropertyAnimation(btn,"geometry");

   anime->setDuration(10000);

   anime->setStartValue(QRectF(0,(qApp->desktop()->height()/2) -
(btn->height()/2) ,300,300));
```

```
anime->setEndValue(QRectF(qApp->desktop()->width()-300,
(qApp->desktop()->height()/2) - (btn->height()/2) ,300,300));
anime->setEasingCurve(QEasingCurve:: SineCurve);
anime->setLoopCount(-1);
anime->start();
```

صرحنا مؤشر لكائن QPushButton يدعى btn حددنا موقعه و حجمه بوساطة المنهج setGeometry(QRect)) و أظهرناه, ثمّ صرحنا مؤشر لكائن QPropertyAnimation يدعى anime وضعنا قيمة وسيطه الأول ضمن البناء مؤشر كائن الزر btn والذي هو كائن الهدف المراد تغيير خاصيته بتدرج ما و الوسيط الثاني الخاصية الديناميكية المراد تغيير قيمها, المدة التي يجب أن تتغير فيها قيم الخاصية المعطاة هي 10000 ميلي ثانية (10 ثوان) , القيمة البدائية للخاصية geometry : الموقع على المحور الأفقي 0 و المحور العمودي في منتصف الشاشة ارتفاعا أما الحجم فهو 300 بكسل عرضا و طولا ,

القيمة النهائية للخاصية geometry : الموقع على المحور الأفقي هو أقصى الحد الأيمن للشاشة أما على المحور العمودي فهو منتصف الشاشة ارتفاعا , الحجم 300 بكسل طولا و عرضا .

بالنسبة لمنحني سير الحركة وضعنا القيمة QEasingCurve:: SineCurve ,

وإعادة تكرار الحركة دائم لأن القيمة -1 تعني هذا , ثم بدأنا الحركة بوساطة المستقبل start() .

بالنسبة للقيمة البدائية setStartValue و القيمة النهائية setEndValue وسيطها من نوع QVariant و يدل هذين المنهجين عل تغيير قيمة الخاصية (هنا geometry) تدريجيا من القيمة البدائية المعطاة حتى القيمة النهائية.

نفذ التطبيق و اختبر النتيجة .

جدول قيم التعداد QEasingCurve::Type منحني سير الحركة :

QEasingCurve::Linear	حركة مستقيمة بسرعة ثابتة , الدالة للمنحني t
QEasingCurve::InQuad	حركة مستقيمة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني t^2

QEasingCurve::OutQuad	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني t^2
QEasingCurve::InOutQuad	حركة مستقيمة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تعود تتباطأ كلما اقتربت من نقطة النهاية , الدالة للمنحني t^2
QEasingCurve::OutInQuad	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تتزايد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني t^2
QEasingCurve::InCubic	حركة مستقيمة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني t^3
QEasingCurve::OutCubic	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني t^3
QEasingCurve::InOutCubic	حركة مستقيمة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تعود تتباطأ كلما اقتربت من نقطة النهاية , الدالة للمنحني t^3
QEasingCurve::OutInCubic	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تتزايد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني t^3
QEasingCurve::InQuart	حركة مستقيمة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني t^4

QEasingCurve::OutQuart	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني t^4
QEasingCurve::InOutQuart	حركة مستقيمة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تعود تتباطأ كلما اقتربت من نقطة النهاية , الدالة للمنحني t^4
QEasingCurve::OutInQuart	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تتزايد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني t^4
QEasingCurve::InQuint	حركة مستقيمة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني t^5
QEasingCurve::OutQuint	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني t^5
QEasingCurve::InOutQuint	حركة مستقيمة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تعود تتباطأ كلما اقتربت من نقطة النهاية , الدالة للمنحني t^5
QEasingCurve::OutInQuint	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تتزايد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني t^5
QEasingCurve::InSine	حركة مستقيمة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني sin(t)

QEasingCurve::OutSine	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني sin(t)
QEasingCurve::InOutSine	حركة مستقيمة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تعود تتباطأ كلما اقتربت من نقطة النهاية , الدالة للمنحني sin(t)
QEasingCurve::OutInSine	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تتزايد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني sin(t)
QEasingCurve::InExpo	حركة مستقيمة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني 2^t منحني أسّي
QEasingCurve::OutExpo	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني 2^t منحني أسّي
QEasingCurve::InOutExpo	حركة مستقيمة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تعود تتباطأ كلما اقتربت من نقطة النهاية , الدالة للمنحني 2^t منحني أسّي
QEasingCurve::OutInExpo	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تتزايد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني 2^t منحني أسّي
QEasingCurve::InCirc	حركة مستقيمة سرعتها تدريجية تبدأ الحركة

	ببطء و تتصاعد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني sqrt(1-t^2) منحني دائري
QEasingCurve::OutCirc	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني sqrt(1-t^2) منحني دائري
QEasingCurve::InOutCirc	حركة مستقيمة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تعود تتباطأ كلما اقتربت من نقطة النهاية , الدالة للمنحني sqrt(1-t^2) منحني دائري
QEasingCurve::OutInCirc	حركة مستقيمة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تتزايد سرعتها كلما اقتربت من نقطة النهاية , الدالة للمنحني sqrt(1-t^2) منحني دائري
QEasingCurve::InElastic	حركة متموجة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من نقطة النهاية
QEasingCurve::OutElastic	حركة متموجة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من نقطة النهاية
QEasingCurve::InOutElastic	حركة متموجة سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تعود تتباطأ كلما اقتربت من نقطة النهاية
QEasingCurve::OutInElastic	حركة متموجة سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تتزايد سرعتها كلما اقتربت من نقطة النهاية

QEasingCurve::InBack	حركة متموجة بشكل خلفي سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من نقطة النهاية
QEasingCurve::OutBack	حركة متموجة بشكل خلفي سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من نقطة النهاية
QEasingCurve::InOutBack	حركة متموجة بشكل خلفي سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تعود تتباطأ كلما اقتربت من نقطة النهاية
QEasingCurve::OutInBack	حركة متموجة بشكل خلفي سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تتزايد سرعتها كلما اقتربت من نقطة النهاية
QEasingCurve::InBounce	حركة متموجة على شكل ارتداد سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من نقطة النهاية
QEasingCurve::OutBounce	حركة متموجة على شكل ارتداد سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من نقطة النهاية
QEasingCurve::InOutBounce	حركة متموجة على شكل ارتداد سرعتها تدريجية تبدأ الحركة ببطء و تتصاعد سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تعود تتباطأ كلما اقتربت من نقطة النهاية
QEasingCurve::OutInBounce	حركة متموجة على شكل ارتداد سرعتها تدريجية تبدأ الحركة سريعة و تتنازل سرعتها كلما اقتربت من منتصف المسافة ثمّ بعد المنتصف تتزايد سرعتها كلما اقتربت من نقطة النهاية
QEasingCurve::Custom	وضع منحني اختياري بوساطة المنهج

• صفوف المجموعة لكائنات التحريك :

إذا كنا نريد احتواء مجموعة من كائنات التحريك كالصف QPropertyAnimation و بدأ عملها ضمن تسلسل معين و فاصل زمني ما هو الحل .

إن Qt يوجد بداخلها صفوف مجموعات خاصة باحتواء كائنات التحريك .

الصف QParallelAnimationGroup و الصف QSequentialAnimationGroup و اللذان يرثان الصف QAnimationGroup .

الصف QSequentialAnimationGroup مهيأ لاحتواء مجموعة من كائنات التحريك لكنه يبدأ تنفيذ عملها بتتالي أي واحدة تلوا الأخرى ويستطيع وضع فاصل زمني بين الحركة و الأخرى.

الصف QParallelAnimationGroup مهيأ لاحتواء مجموعة من كائنات التحريك لكنه يبدأ تنفيذ عملها بتوازي أي يبدأ تنفيذ عمل جميع كائنات الحركة مع بعضهم البعض .

مثال :

لننشأ تطبيق موجود داخله كائن يحوي صورة معروض ضمن كائن المشهد Scene يتحرك الكائن على الخط الأفقي مسافة معينة مرّة واحدة و من ثمّ يدور حول ذاته دورة كاملة ,هنا سوف ننشأ صف للكائن الذي سوف يحوي الصورة و سنستخدم الصف QSequentialAnimationGroup للتتالي الحركات .

أولا انشأ مشروع فارغ حيث يكون اسمه groupAnimation أضف ملف ترويسة يدعى pixObj هذا الملف سوف يحوي الرمّاز الخاص بصف الكائن الذي يحوي الصورة التي نريد تحريكها ,إنّ الصف pixObj موروث من الصف QGraphicsObject والذي بدوره موروث من الصف QObject و الصف QGraphicsItem, يحوي الصف pixObj كائن QPixmap يرسم الصورة المحتواة ضمن هذا الكائن داخله بوساطة المنهج الإفتراضي paint , لكن لماذا أنشئنا الصف pixObj إذا كانت مهمته فقط رسم صورة موجودة ضمن كائن QPixmap ؟

إن الصف QPropertyAnimation لا يقبل أن يعالج خاصيّة أيّ كائن إلا إذا كان يرث الصف QObject و الكائن QPixmap يرث الصف QPaintDevice لذا أنشئنا صف يرث صف الكائن الرسومي المشتق من الصف QObject و رسمنا داخله الصورة .

أضف ملفات الترويسة التالية داخل الملف pixObj.h :

```
#include <QPainter>

#include <QGraphicsObject>
```

ثمّ أنشأ الصف بكتابة الرمّاز التالي :

```
class pixObj : public QGraphicsObject

{

   Q_OBJECT

private:

   QPixmap pPix;


public:

   pixObj(const QPixmap &pixmap) : QGraphicsObject(), pPix(pixmap)
{}


   QPixmap pixmap(){return pPix;}



   void paint(QPainter *painter, const QStyleOptionGraphicsItem *sogi,
QWidget *widget)

   {
```

```
    QImage image;
    image = pPix.toImage();

            for (int j=0; j <= image.size().width()-1 ; j++)
            {
          image.setPixel(j,0,qRgb(0,255,0));
          image.setPixel(j,image.size().height()-1,qRgb(0,255,0));
            }

            for (int j=0; j <= image.size().height()-1 ; j++)
            {
          image.setPixel(0,j,qRgb(0,255,0));
          image.setPixel(image.size().width()-1,j,qRgb(0,255,0));
            }


    pPix.convertFromImage(image);
    painter->drawPixmap(QPointF(), pPix);

  }

boundingRect() const    QRectF
  {
```

```
    return QRectF( QPoint(), pPix.size() );
  }
};
```

الصف pixObj مشتق من الصف QGraphicsObject , قد فعّل ميزات نظام هيكلية كائن Qt باستخدام الماكرو Q_OBJECT كميزات الخصائص الديناميكية و ميزات اتصال الكائنات مع بعضها باستخدام تقنية الإشارة و المستقبل (Signal-Slot) , صرح عن المثيل pPix من نمط QPixmap في القسم الخاص من الصف, هذا المثيل سوف نستخدمه لاحتواء الصورة المراد رسمها , بناء الصف pixObj له وسيط من نمط QPixmap سوف تضع قيمة هذا الوسيط ضمن المثيل pPix :

```
public:
  pixObj(const QPixmap &pixmap) : QGraphicsObject(), pPix(pixmap)
{}
```

المنهج pixmap يعيد المثيل pPix .

المنهج الإفتراضي و المهيمن عليه paint هو المسؤول عن رسم الصورة في الكائن pixObj , نريد أن نكتب داخله رمّاز لرسم الصورة المحتواة ضمن المثيل pPix ورسم إطار بلون أخضر حول الصورة وذلك من خلال المنهج setPixel التابع للكائن QImage :

```
 QImage image;
    image = pPix.toImage();

        for (int j=0; j <= image.size().width()-1 ; j++)
        {
      image.setPixel(j,0,qRgb(0,255,0));
      image.setPixel(j,image.size().height()-1,qRgb(0,255,0));
        }

        for (int j=0; j <= image.size().height()-1 ; j++)
```

```
        {
        image.setPixel(0,j,qRgb(0,255,0));
        image.setPixel(image.size().width()-1,j,qRgb(0,255,0));
        }
    pPix.convertFromImage(image);
    painter->drawPixmap(QPointF(), pPix);
```

المثيل image من نمط QImage سوف يحوي الصورة الموجودة ضمن المثيل pPix من خلال المنهج ()toImage للكائن QPixmap ,

ضمن حلقة التكرار الأولى رسم الحد الأعلى و الحد الأسفل للصورة أما الحلقة الثانية رسم الحد الأيسر و الحد الأيمن بلون أخضر .

قد تمّ رسم الحد من خلال الرسم المباشر بالبكسل على الصورة وهذه ميزة الكائن QImage لذلك قد استخدمناه.

عدنا ووضعنا الصورة المعدلة ضمن المثيل pPix من خلال المنهج

convertFromImage(QImage &image) التابع للكائن QPixmap , تمّ رسم الصورة على الكائن pixObj من خلال المنهج drawPixmap . وضعنا قيمة وسيطه الأول ()QPointF أي سوف يرسمها من النقطة 0.0 من محور الإحداثيات .

بالنسبة للمنهج الإفتراضي المهيمن عليه boundingRect مهمته إرجاع موقع و حجم الصورة المرسومة و الذي سوف يستخدمه الكائن الرسومي pixObj (الصف الحالي) لمعرفة موقع و حجم المساحة المراد إعادة الرسم داخلها .

انتهينا من كتابة رمّاز الصف pixObj .

أضف ملف يدعى main.cpp سوف يحوي رمّاز الكتلة الخاصة بانطلاق التطبيق و إضافة المشاهد الرسومية المتتالية .

اكتب الرمّاز التالي داخل الملف main.cpp :

```
#include <QtGui/QApplication>
```

```
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QPropertyAnimation>
#include <QSequentialAnimationGroup>

#include "pixObj.h"

int main(int argc, char **argv)
{
   QApplication app(argc, argv);
   pixObj* pix = new pixObj(QPixmap(":/icons/qt"));

   QPropertyAnimation* anime1 = new
QPropertyAnimation(pix,"pos");
    anime1->setDuration(2000);
    anime1->setStartValue(QPoint(0,0));
    anime1->setEndValue(QPoint(200,0));

   QPropertyAnimation* anime2 = new
QPropertyAnimation(pix,"rotation");
    anime2->setDuration(2000);
    anime2->setStartValue(qreal(1));
    anime2->setEndValue(qreal(360));
```

```
   QSequentialAnimationGroup* seq = new
QSequentialAnimationGroup();
    seq->addAnimation(anime1);
    seq->addPause(5000);
    seq->addAnimation(anime2);
    seq->setLoopCount(-1);
    seq->start();


       QGraphicsScene *scene = new QGraphicsScene();
       scene->setSceneRect(0, 0, pix->pixmap().size().width(), pix-
>pixmap().height());
       scene->addItem(pix);
       QGraphicsView view;
       view.setGeometry(QRect(100,100,500,500));
       view.setScene(scene);
       view.show();
   return app.exec();
}
```

أضفنا ملفات الترويسة التالية :

```
#include <QtGui/QApplication>
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QPropertyAnimation>
```

```
#include <QSequentialAnimationGroup>

#include "pixObj.h"
```

ضمن الكتلة main صرحنا عن حدث للصف pixObj يدعى pix و مررنا لوسيط بنائه كائن QPixmap يحوي صورة و التي سوف يقع عليها فعل التحريك .

صرحنا عن حدث للصف QPropertyAnimation يدعى anime1 الخاص بتحريك الكائن pix على المحور الأفقي و حدث آخر يدعى anime2 الخاص بتحريك الكائن pix بطريقة دورانيّ .

صرحنا عن حدث من نمط QSequentialAnimationGroup يدعى seq والذي سوف يحوي anime1 و anime2 و سوف ينظم تنفيذ عملهما بحيث تعمل الحركة الأولى anime1 و بعد انتهائها يقف مؤقتا لمدة 5 ثوان و من ثمّ ينفذ الحركة الثانية anime2 , سوف يكرر هذه العملية إلى ما لا نهاية من خلال المنهج :

```
seq->setLoopCount(-1);
```

أضفنا الحركات من خلال المنهج :

```
seq->addAnimation(anime1);

);2seq->addAnimation(anime
```

أما الإيقاف المؤقت :

```
seq->addPause(5000);
```

بدأ تنفيذ العمل :

```
seq->start();
```

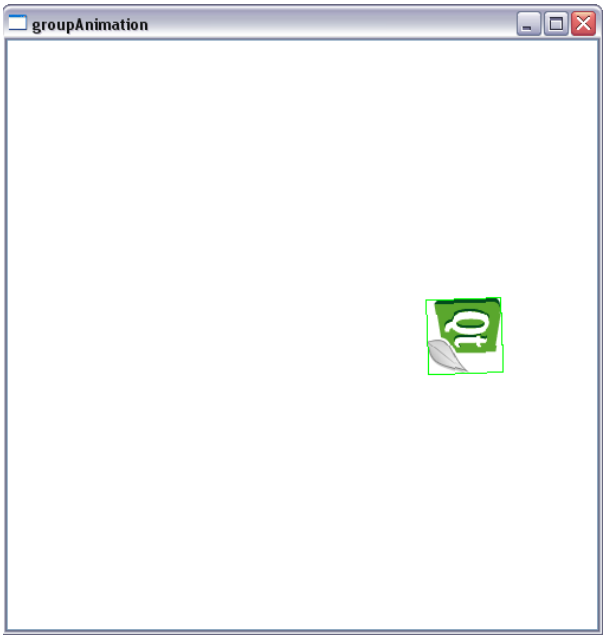
لإظهار الكائن الرسومي pix يجب أن نضعه داخل كائن مشهد رسومي انظر الرمّاز :

```
    QGraphicsScene *scene = new QGraphicsScene();

    scene->setSceneRect(0, 0, pix->pixmap().size().width(), pix-
>pixmap().height());

    scene->addItem(pix);
```

وضعنا المشهد الرسومي scene داخل كائن QGraphicsView :

```
QGraphicsView view;
view.setGeometry(QRect(100,100,500,500));
view.setScene(scene);
view.show();
```

نفذ التطبيق سوف يظهر كما في الشكل (3.20) :



الشكل 3.20

إذا كنا نريد أن تعمل جميع الحركات مع بعضها يجب علينا استخدام الصف QParallelAnimationGroup بدلا من الصف QSequentialAnimationGroup , اذهب إلى الرمّاز الموجود داخل الكتلة main السابقة و استبدل الرمّاز :

```
QSequentialAnimationGroup* seq = new
QSequentialAnimationGroup();
seq->addAnimation(anime1);
seq->addPause(5000);
seq->addAnimation(anime2);
seq->setLoopCount(-1);
```

```
seq->start();
```

بالرمّاز التالي :

```
QParallelAnimationGroup* par = new QParallelAnimationGroup();

par->addAnimation(anime1);

par->addAnimation(anime2);

par->start();
```

الآن نفذ التطبيق سوف تجد أن الحركة الدورانية للكائن pix مع مسيره على المحور الأفقي قد نفذوا على التوازي (مع بعضهم البعض) .

يوجد عدة خصائص ديناميكية للكائنات يستطيع الصف QPropertyAnimation التعامل معها أهمها :

تغيير الموقع	Pos
تغيير الحجم و الموقع	Geometry
تغيير قيمة الدوران	Rotation
تغيير قيمة الشفافية	Opacity

إذا كنا نريد إنشاء خاصية ديناميكية يجب علينا أن نستخدم الماكرو Q_PROPERTY الخاص بلغة Qt .

- **إنشاء خاصية ديناميكية :**

الخاصية الديناميكية تابعة لنظام هيكلة كائن Qt أي ضمن الميزات التي يفعلها الماكرو Q_OBJECT , تستطيع استخدامها فقط ضمن الصفوف المشتقة من الصف QObject .

دعنا ننشأ خاصية تدعى zoom إلى الصف pixObj السابق و التي تكون مهمتها تقريب و تبعيد الصورة .

اكتب الرمّاز التالي داخل الصف pixObj السابق تحت الماكرو Q_OBJECT :

```
Q_PROPERTY(int zoom READ zoom WRITE setZoom FINAL)
```

قد تمّ تعريف خاصية ديناميكية باسم zoom من نمط int , المنهج الخاص بقراءة قيمتها zoom و المنهج الخاص بوضع قيمتها setZoom و سوف لن يسمح بتجاوز أو هيمنة هذه الخاصية من خلال الصفة FINAL .

الماكرو Q_PROPERTY خاص بتعريف خاصية ديناميكية .

اكتب في القسم العام للصف pixObj المنهجين الخاصين بقراءة قيمة الخاصية و وضع قيمتها :

```
void setZoom(int z){

   this->setScale((qreal(z)/10.0));

}

int  zoom() const  {  return (  this->scale()*10.0);  }
```

المنهج setZoom لتقريب أو تبعيد مشهد الصورة له وسيط من نمط int يحدد مقدار تقريب أو تبعيد الصورة .

المنهج zoom يرجع قيمة من نمط int تكون مقدار التباعد للصورة .

القيمة الإفتراضية هي 10 .

داخل الكتلة main استبدل الرمّاز الخاص بسير الكائن pix على المحور الأفقي بالرمّاز الخاص بتقريب الصورة من خلال استخدام الخاصية zoom التي عرفناها :

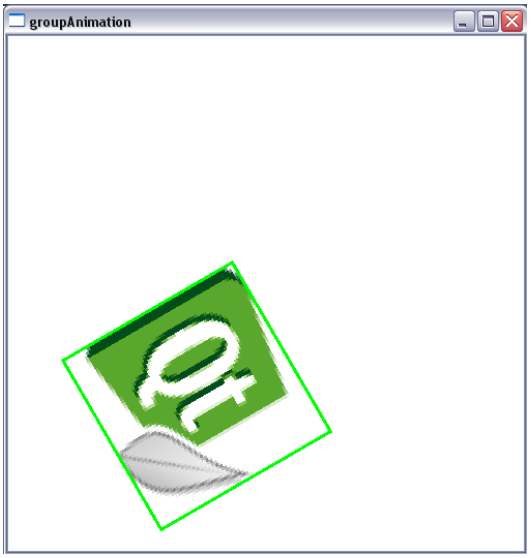
```
QPropertyAnimation* anime = new QPropertyAnimation(pix,"pos");

anime1->setDuration(2000);

anime1->setStartValue(QPoint(0 , 0));

anime1->setEndValue(QPoint(200 , 0));
```

استبدله بالرمّاز :

```
QPropertyAnimation* anime1 = new
QPropertyAnimation(pix,"zoom");

anime1->setDuration(2000);
```

```
anime1->setStartValue(10);

anime1->setEndValue(30);
```

نفذ التطبيق سوف يظهر كما في الشكل (3.21) :



الشكل 3.21

نهاية فقرة الخاصية الديناميكية في Qt .

• خلاصة الفصل:

قد أنهينا هذا الفصل الذي تكلمنا بداخله عن استخدام الكائنات الرسومية الساكنة و المتحركة و وكائنات الطباعة و المعاينة قبل الطباعة و كائنات المجموعات و بعض الميزات المتقدمة في Qt مثل الخاصية الديناميكية .

لننتقل إلى الفصل الخاص بالتكلم عن استخدام صفوف معالجة الملفات و البيانات الموجودة داخلها و صفوف المجاري Streams .

الفصل الرابع

معالجة الملفات

Processing of the files

مقدمة Intro :

كل إنسان يمتلك ذكريات في ذاكرته (المذكرة) التي هي جزء من عقله , فالمذكرة تحوي مشاهد إما صورية أو سمعية أو حسية تم حفظها فيها يجلبها الإنسان لتتركب و تتراءى أمامه بشكل غير ملموس من جديد إراديا أو لا إراديا*سبحان الخالق* .

و في عالم التطبيقات البرمجية يوجد ما يشبه ذلك إلى حد ما فدائما يوجد بيانات نصية أو ثنائية الخ.. إمّا خام أو قد تمّ معالجتها يحتاج التطبيق استيرداها ليكمل عمله الذي تمّ طلبه من قبل عميل ما .

في إطار عمل Qt تمّ إنشاء عدة صفوف ذات أداء عالي , تملك واجهة غنية سهلة الاستخدام للوصول إلى الملفات و معالجتها .

من أهم جزء لمعالجة الملفات بعد الوصول إليها هي استخدام المجاري Streams من أجل المعالجة, فالمجاري تتيح الوصول إلى كتل البيانات و معالجتها بشكل برمجي غير آبهة بالموضع الفيزيائي لهذه الكتل , و باستطاعتها سلسلة أي كائن (صف) و تهيئته للحفظ و من ثمّ استرداده لاحقا عند الحاجة إليه, كإرساله عبر الشبكة أو عرضه أو تنفيذه .

يحوي هذا الفصل "معالجة الملفات" أغلب التقنيات المتاحة في إطار عمل Qt للوصول إلى الملفات و المجلدات و معالجتها و كيفية استخدام المجاري و التقنية الأفضل لحفظ الكائنات و التي هي عملية السلسلة , و قراءة و كتابة مستندات XML, و كيفية مراقبة ملفات و مجلدات محددة و التعرف على استخدام نوافذ الحوار الخاصة بفتح و حفظ و تحديد ملف أو مجلد .

تتم جميع عمليات الوصول إلى الملفات و معالجة كتلها البيانيّة باستقلالية عن نظام التشغيل.

حقا إنّ Qt C++ Framework لمذهلة .

❖قسم المجاري .

❖ مصفوفة البت و مصفوفة البايت :

في أي رمّاز مصدري لتطبيق ما نحتاج لحفظ و إرسال و استقبال بيانات , بالتالي يتوجب أن نتعامل معها بدقة وتسلسل معين , تتيح لنا المصفوفات هذه الميزات طبعا يحدد نمط المصفوفة حسب نوع البيانات المراد معالجتها,غالبا نستخدم مصفوفة بنمط بايت لأنّ باستطاعتها استيعاب أي نمط من البيانات, و نحتاج في كثير من الأحيان لحفظ بيانات منطقية true = 1 , false = 0 من أجل بعض الإعدادات أو شيء آخر .

يوجد ضمن مكتبات Qt صفوف مجهزة لحفظ و معالجة مصفوفة من نمط بت (bit) و من نمط بايت (byte) وهذين الصفين يدعوا بـ : QBitArray – QByteArray .

- **الصف QBitArray :**

يستخدم في تخزين مصفوفة من البتات و الوصول إلى أي عنصر فيها و تغيير قيمته والتي هي عبارة عن 0 أو 1 (true,false) و إجراء العمليات المنطقة على عناصره :

A &
N
D

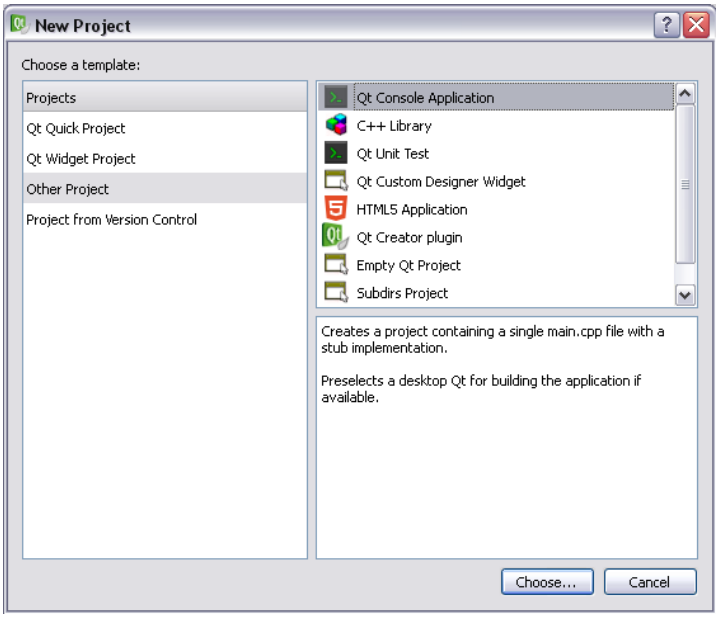
O |
R

X ^
O
R

N ~
O
T

مثال بسيط لاستخدام الصف QBitArray :

أولا أنشأ تطبيق Qt Console Application (تطبيق سطر أوامر) من New Project اختبر تبويب Other Project ثمّ Qt Console Application كما في الشكل (4.1) وليكن اسم المشروع BitArray .



الشكل 4.1

ثانيا ضمّن ملف الترويسة QBitArray و iostream داخل الملف main.cpp :

```
#include <QBitArray>

#include <iostream>
```

ثالثا تعريف الصف cout و endl التابعان لفضاء التسمية std داخل الملف main.cpp لنستطيع استخدامهم داخله وذلك من خلال التعليمة using , اكتب الرمّاز التالي تحت تضمين ملفات الترويسة :

```
using std::cout;

using std::endl;
```

رابعا اكتب الماكرو التالي والذي يدعى nl الذي سوف يتم استدعائه عندما نريد الانتقال إلى سطر جديد ضمن شاشة سطر الأوامر Console :

```
#ifndef nl
#define nl cout << endl ;
#endif
```

خامسا اكتب بعد السطر البرمجي :

```
QCoreApplication a(argc, argv);
```

الرمّاز الخاص بإنشاء مثيلان (arBit1 – arBit2) لصف مصفوفة البت QBitArray , حيث سيتم وضع لهما قيم عشوائية, و سوف تظهر نتيجة حاصل العملية المنطقية AND عليهما ضمن شاشة سطر الأوامر :

```
  QBitArray arBit1(4,true);
  arBit1.fill(false,1,3);
  arBit1.resize(8);
  arBit1.toggleBit(6);
  arBit1.setBit(2,true);
  QBitArray arBit2(4,true);
  arBit2.fill(false,0,2);
  arBit2.resize(8);
  arBit2.truncate(6);
  for(int i=0; i <= arBit1.size() -1 ;i++)
     cout << arBit1[i] << " ";
  nl
  for(int i=0; i <= arBit2.size() -1 ;i++)
     cout << arBit2[i] << " ";
  nl
  arBit1 &= arBit2 ;
```

```
for(int i=0; i <= arBit1.size() -1 ;i++)

    cout << arBit1[i] << " ";
```

صرحنا عن مثيل للصف QBitArray يدعى arBit1 يوجد داخله أربع خلايا (بتات) قيمة كل واحدة مساوية 1 (true) , ثمّ جعلنا قيمة الخلية الثانية و الثالثة قيمتهما 0 (false) , أعدنا تحجيم مصفوفة البتات إلى 8 خلايا من خلال المنهج resize(8) أي سوف يضاف 4 خلايا جدد إلى نهاية مصفوفة البت arBit1 قيمتهم 0 (false) , عكسنا قيمة الخلية التي يكون موقعها 6 , ثمّ وضعنا قيمة الخلية الثانية (ذي الموقع 2) 1 (true) .

صرحنا عن مثيل ثاني للصف QBitArray يدعى arBit2 يوجد داخله أربع خلايا (بتات) قيمة كل واحدة مساوية 1 (true) , ثمّ جعلنا قيمة الخلية الأولى و الثانية قيمتهما 0 (false) , أعدنا تحجيم مصفوفة البتات إلى 8 خلايا ثمّ محونا جميع الخلايا الموجودة بعد الخلية ذي الموقع 6 من خلال المنهج truncate(6) .

حلقة التكرار الأولى سوف تطبع جميع قيم الخلايا الموجودين داخل المصفوفة arBit1 , ثمّ استدعينا الماكرو nl للانتقال إلى سطر جديد. حلقة التكرار الثانية لطباعة جميع قيم المصفوفة arBit2 .

نفذنا العملية المنطقة AND(&) من خلال السطر البرمجي :

```
arBit1 &= arBit2 ;
```

نتيجة العملية AND أصبحت داخل المصفوفة arBit1 , أظهرنا النتيجة بوساطة حلقة التكرار الأخير من هذا الرمّاز .

نفذ التطبيق سوف ترى كما في الشكل (4.2) :

الشكل 4.2

تستطيع إجراء عملية منطقية غير AND(&) من خلال تبديل الرمز (&) برمز آخر خاص بعملية منطقية أخرى.

نذكر أهم مناهج و عوامل (معاملات) الصف QBitArray :

يرجع قيمة البت ذي الموقع المساوي لقيمة الوسيط i	**bool at (int i) const**
يزيل جميع الخلايا للكائن	**void clear ()**
يسند القيمة 0 للخلية ذات الموقع المساوي للوسيط i	**void clearBit (int i)**
يعيد عدد الخلايا (عدد البتات) الموجودة داخل المصفوفة	**int count () const**
يعيد عدد الخلايا المساوية قيمتهم لقيمة الوسيط on	**int count (bool on) const**
يعيد قيمة الخلية (البت) ذات الموقع المساوي للوسيط i	**bool testBit (int i) cons**
عامل إزاحة, يعمل كدخل للكائن QBitArray من كائن مجرى بيانات QDataStream ,	**>>**

ينفذ عملية فك السلسلة من مجرى البيانات ثمّ
يعطي قيمتها للكائن QBitArray

>> عامل إزاحة, يعمل كخرج من الكائن
QBitArray على كائن مجرى بيانات
QDataStream , ينفذ عملية السلسلة
لبيانات الكائن QBitArray من ثمّ يضعها
داخل كائن مجرى البيانات

انتهينا من التكلم عن الصف QBitArray (صف مصفوفة من نمط بت) , لننتقل إلى التكلم عن الصف QByteArray (صف مصفوفة من نمط بايت - byte) .

- الصف QByteArray :

يستخدم في تخزين مصفوفة من البايتات و الوصول إلى أي عنصر فيها و تغيير قيمته و معالجة البيانات التي يحويها , بإمكانه تحويل البيانات المضمنة داخله إلى ترميز ستة عشري, و استخدام التشفير المئوي الذي يستخدم في عناوين صفحات الويب , و اختبار محتواه بالمقارنة ببيانات أخرى و استبدال بيانات داخله ببيانات أخرى .

مثال بسيط لكيفية استخدام الصف QByteArray يحوي أكثر و أهم مناهجه التي تستخدم في رمّازات التطبيقات.

أنشأ تطبيق Qt Console Application يدعى ByteArray , اذهب إلى الملف main.cpp واكتب الرمّاز التالي :

```
#include <QtCore/QCoreApplication>

#include <iostream>

#include <QByteArray>

using std::cout;

using std::endl;

#if !defined(nl)

#define nl cout << endl ;
```

```
#endif
int main(int argc, char *argv[])
{
   QCoreApplication a(argc, argv);

   QByteArray byteArray("Hello\nQt for Arab");
   cout << byteArray.data(); nl nl
   for (int i=0; i <= byteArray.size()-1 ; i++)
      cout << byteArray[i] << " " ; nl nl
   byteArray.replace("for","4");
   cout << byteArray.data() ;  nl nl

   if(byteArray.contains("Qt"))
     cout << "Qt" ; nl nl

   cout << "Encode byte array to hex Encode :" << endl <<
byteArray.toHex().data() ; nl nl
   cout << "Decode hex Encode :" << endl <<
QByteArray::fromHex(byteArray.toHex()).data() ; nl nl

   byteArray.append("\nHasan");
   cout << byteArray.data();nl nl
   byteArray.prepend("Qt C++ Framework\n");
   cout << byteArray.data(); nl nl
```

```
    byteArray.resize(16);
    cout << byteArray.data();


    return a.exec();
}
```

صرحنا عن مثيل يدعى byteArray من نمط QByteArray (صف مصفوفة بايت - byte) يوجد بداخله النص:

Hello

Qt for Arab

حيث كل محرف من هذا النص يحجز خانة (byte) بالترتيب من هذه المصفوفة .

طبعنا النص على الشاشة بوساطة استدعاء cout الذي مررنا له قيمة المصفوفة بوساطة المنهج data التابع للصف QByteArray و التي تكون مهمته إرجاع مؤشر محرف على بيانات المصفوفة كاملة .

من ثمّ طبعناه من جديد على شاشة سطر الأوامر ولكن يحوي بين كل محرف و محرف فراغ ,تمّ ذلك كالآتي : داخل حلقة التكرار استخدمنا المعامل [] (معامل الأقواس المربعة) للصف QByteArray الذي يستخدم من أجل إرجاع قيمة المحرف الذي عنوانه الوسيط الممرر له ,هنا استدعينا هذا المعامل لطباعة المحرف تلوا الآخر من خلال استدعائه ضمن حلقة التكرار و تمرير المتحول i الخاص بهذه الحلقة لهذا المعامل .

استبدلنا النص for بـ 4 بوساطة المنهج replace للصف QByteArray :

```
QByteArray &  replace ( const char * before, const char * after )
```

اختبرنا أن النص المحتوى ضمن المصفوفة يحوي النص Qt ليطبع النص Qt على شاشة سطر الأوامر .

شفرنا بيانات المصفوفة إلى تشفير ستة عشري بوساطة المنهج ()toHex ثمّ فككنا تشفيره بوساطة المنهج ()fromHex و أظهرنا البيانات المشفرة على شاشة سطر الأوامر و من ثمّ أظهرناها بعد فك تشفيرها .

```
QByteArray  toHex () const
```

QByteArray fromHex (const QByteArray & hexEncoded)

أضفنا بيانات من نمط نص و هي "Hasan" إلى نهاية المصفوفة بوساطة المنهج append ثم أضفنا بيانات إلى بداية المصفوفة بوساطة المنهج prepend :

QByteArray & append (const char * str)

QByteArray & prepend (const char * str)

عند إضافة البيانات إلى بداية المصفوفة سوف يعاد تحجيم المصفوفة كالتالي :

عدد الخلايا الموجودة في المصفوفة بالجمع مع عدد بايتات البيانات التي سوف تضاف إليها .

سوف يصبح موقع أول خلية للبيانات الموجودة مسبقا بعد موقع آخر خلية للبيانات المضافة .

أعدنا تحجيم المصفوفة بوساطة المنهج resize(int) :

void resize (int size)

عند تحجيم المصفوفة إلى حجم أصغر مما كانت عليه سوف تزال جميع الخلايا الموجودة بعد موقع المساوي لحجم المصفوفة الجديدة أي قيمة الوسيط size الممرر إلى المنهج resize .

عند تنفيذ التطبيق سوف يظهر الشكل (4.3):

الشكل 4.3

يوجد منهج يدعى remove تابع للصف QByteArray يستخدم لإزالة الخلايا :

QByteArray & QByteArray::remove (int pos, int len)

سوف يزيل الخلايا التي يكون موقعها بين المجال قيمة الوسيط pos و قيمة الوسيط pos بجمعه مع قيمة الوسيط len .

انتهيا من شرح الصف QByteArray والذي سوف نستخدمه كثيرا في مواقع عدة من هذا الكتاب نظرا لفائدته و سهولة استخدامه .

لننتقل إلى التكلم عن صف جهاز الدخل و الخرج الأساسي في Qt وهو QIODevice .

- **الصف QIODevice :**

الصف QIODevice يتّسم بصفة التجريد و يكون الأب لجميع صفوف أجهزة الدخل و الخرج في Qt , يرث الصف QObject يحوي تعريف لمناهج القراءة و الكتابة من و إلى كتل البيانات إما بوصول تسلسلي أو عشوائي , أهم الصفوف التي ترثه :

يحوي هذا الصف المناهج العامة لجميع أنواع المقابس , يورث منه الصفين :	**QAbstractSocket**
QTcpSocket	
QUdpSocket	
يدعم الوصول لكتل البيانات و معالجتها الموجودة داخل الصف QByteArray بوساطة المناهج الموصوفة بالواجهة التابعة للصف QIODevice.	**QBuffer**
الصف الخاص بالوصول إلى الملفات و معالجتها .	**QFile**
الصف الخاص بالتعامل مع المقبس المحلي .	**QLocalSocket**
لاحتواء معلومات الترويسة و البيانات التي أرسلت إلى مخدم ما كطلب .	**QNetworkReply**
لتنفيذ برامج خارجية و إجراء اتصال معهم .	**QProcess**

تعيد الصفوف التي ترث QIODevice تحقيق مناهجه بشكل يلاءم عملها .

عندما تريد الوصول إلى كتلة بيانات ما يجب عليك أن تضع نمط فتح هذه الكتلة أي تهيئتها عندما تقوم بفتحها بوساطة جهاز دخل/خرج للقراءة أو الكتابة الخ.. .

انظر الجدول الخاص بالتعداد نمط فتح الكتلة QIODevice::OpenModeFlag :

عدم فتح الجهاز الخاص بالوصول إلى كتلة البيانات المراد معالجتها	QIODevice::NotOpen
فتح الجهاز الخاص بالوصول إلى كتلة البيانات المراد معالجتها من أجل القراءة منها	QIODevice::ReadOnly
فتح الجهاز الخاص بالوصول إلى كتلة البيانات المراد معالجتها من أجل الكتابة عليها	QIODevice::WriteOnly
فتح الجهاز الخاص بالوصول إلى كتلة البيانات المراد معالجتها من أجل القراءة و الكتابة منها و عليها	QIODevice::ReadWrite
فتح الجهاز الخاص بالوصول إلى كتلة البيانات المراد معالجتها من أجل الإضافة عليها	QIODevice::Append
عندما يتم فتح الجهاز بنمط كتابة أو إضافة بيانات مع هذا النمط سوف يتم مسح جميع البيانات الموجودة داخل كتلة البيانات المتصلة بجهاز الدخل و الخرج	QIODevice::Truncate
عندما تقرا نهاية السطر من كتلة بيانات ما سوف يترجم إلى "\n" وعندما تكتب نهاية السطر لكتلة بيانات سوف يترجم إلى التشفير المحلي مثال في Win سوف تترجم نهاية السطر إلى "\r\n"	QIODevice::Text
تهمل الذواكر المؤقتة و القياسية في جهاز الدخل و الخرج , تستخدم عندما نريد	QIODevice::Unbuffered

- **الصف QBuffer :**

الصف QBuffer يرث الصف QIODevice .

يتيح لك الصف QBuffer معالجة مصفوفة البايت QByteArray من خلال المناهج المعرفة داخل الصف QIODevice بعد إعادة تحقيقها داخله, انظر الرمّاز التالي الذي يحوي كيفيّة استخدامه :

```
QByteArray byteArray("Hasan");

QBuffer buffer(&byteArray);

buffer.open(QIODevice::WriteOnly);

buffer.seek(3);

buffer.write(" Al-Morhej", 10);

buffer.close();

std::cout << byteArray.data() ;

// std::cout << buffer.data().data() ;
```

صرحنا عن مثيل لمصفوفة بايت يدعى byteArray و وضعنا قيمته النص Hasan , ثمّ صرحنا عن مثيل للصف QBuffer يدعى buffer و مررنا لوسيطه مرجع المثيل byteArray , أي سيعالج البيانات المحتواة داخل مصفوفة البايت byteArray ,فتحنا كتلة البيانات byteArray بنمط للكتابة فقط QIODevice::WriteOnly من خلال المنهج open التابع للصف QBuffer و نقلنا موقع المؤشر داخل كتلة البيانات إلى البايت الثالث من خلال المنهج seek من أجل كتابة بيانات من موقع المؤشر الحالي, أضفنا النص Al-Morhej باستخدام المنهج write , ثمّ أغلقنا معالجة كتلة البيانات بوساطة المنهج close , طبعنا النص على شاشة سطر الأوامر من خلال المنهج data الذي يعيد مؤشر محرف لكامل عناصر المصفوفة .

***ملاحظة*: المناهج open , seek , write , close تابعة بالأساس للصف QIODevice و معاد تحقيقها داخل الصف QBuffer.**

- **الصف QDataStream :**

تمّ إنشاء الصف QDataStream ليتيح للمبرمج احتواء سلسلة من البيانات الثنائيّة و معالجتها بواسطته, صف مجرى البيانات QDataStream يشفر بياناته الثنائية التي يحتويها باستقلالية عن نظام التشغيل المحمول عليه التطبيق, يستطيع الصف QDataStream سلسلة أي نوع من البيانات , int , char , QVariant , QChar , qint32 الخ ...

ولا يمكنه القراءة و الكتابة إلا على مصفوفة بايت QByteArray أو على أي صف من الصفوف الموروثة من جهاز الدخل و الخرج الأساسي في Qt الصف QIODevice .

عندما يتم تحويل الكائن إلى سلسلة من البيانات الثنائيّة نستطيع تقسيم السلسلة إلى عدة كتل من البيانات من دون فقدان أي من بياناتها وسوف تتم عملية الكتابة و القراءة على هذه الكتل بشكل سريع و ذات وصول معنون لذلك يجب أن نستخدمه في عدة حالات مثال عندما نريد إرسال بيانات عبر الشبكة نود أن نحول هذه البيانات إلى بيانات ثنائيّة و ثمّ تقسيمها إلى عدة كتل من ثمّ إرسالها دفعة تلوى الأخرى.

سنكتب رمّاز خاص بسلسلة بيانات نصية و رقمية من خلال الصف QDataStream و كتابة هذه البيانات بعد سلسلتها على مصفوفة بايت و من ثمّ فك سلسلتها و قراءتها من مصفوفة البايت و طباعتها على شاشة سطر الأوامر .

انظر الرمّاز التالي :

```
QByteArray* byteArray = new QByteArray;

QDataStream out(byteArray,QIODevice::WriteOnly);

out.setVersion(QDataStream::Qt_4_7);

out << (char*)("Hasan Al-Morhej\nLearning Qt");

out << (int)2012;

QDataStream in(byteArray,QIODevice::ReadOnly);

in.setVersion(QDataStream::Qt_4_7);

char* str;

int year;
```

```
in >> str >> year;

std::cout  <<  str << " " << year << std::endl ;
```

صرحنا عن حدث من الصف QByteArray يدعى byteArray و مثيل للصف QDataStream يدعى out يستخدم لكتابة البيانات ,مررنا لوسطاء بناءه الوسيط الأول الكائن byteArray لتكتب البيانات التي سوف تضاف للمجرى out عليه أما الوسيط الثاني نمط فتح المجرى للكتابة فقط , استخدمنا المعامل >> لكتابة البيانات على المجرى وبدوره بعد سلسلة تلك البيانات يكتبها على الكائن byteArray , أضفنا بيانات من نمط مؤشر محرف و من نمط رقم صحيح إلى الكائن out , ثمّ صرحنا عن مثيل آخر لصف مجرى البيانات يدعى in يستخدم لقراءة البيانات مررنا لوسطاء بناءه الوسيط الأول الكائن byteArray لنقرأ البيانات منه وكتابتها داخل المجرى in لفك سلسلتها أما الوسيط الثاني نمط فتح المجرى للقراءة فقط , استخدمنا المعامل << لقراءة البيانات من المجرى بعد فك سلسلتها و كتابتها على المثيل str و المثيل year , ثمّ طبعناها على شاشة سطر الأوامر .

لنتحدث الآن عن معنى الإصدار "version" في سلسلة كتل البيانات :

بمرور الوقت يتم تطوير مكتبات Qt من إضافة صفوف جديدة و تطوير آلية تخزين كتل البيانات ضمن كائن معين ككائن "QPixmap" الذي يستطيع احتواء صورة , عند تطوير كائن سوف تتغير بنيته الداخلية "كتلة بيانات" لذا يتوجب أن يعدل آلية سلسلة هذا الكائن ضمن مجرى البيانات ب تعديل الصف "QDataStream" بحيث يتلاءم مع هرمية البنية الداخلية للكائن الذي قد تمّ تطويره أو إنشاءه , و مع كل تطوير لكائن مجرى البيانات يكون قد تم إصدار نسخة جديدة منه , و هذا ما يدعى بالإصدار في الصف "QDataStream" , نستطيع تحديد إصدار المجرى كالآتي :

```
void QDataStream::setVersion(int v)
```

يتوجب علينا تحديد الإصدار عند عملية السلسة و عند عملية فك السلسة .

أما الإصدارات المتاحة في Qt هي :

- QDataStream::Qt_1_0
- QDataStream::Qt_2_0
- QDataStream::Qt_2_1
- QDataStream::Qt_3_0
- QDataStream::Qt_3_1
- QDataStream::Qt_3_3
- QDataStream::Qt_4_0
- QDataStream::Qt_4_1

- QDataStream::Qt_4_2
- QDataStream::Qt_4_3
- QDataStream::Qt_4_4
- QDataStream::Qt_4_5
- QDataStream::Qt_4_6
- QDataStream::Qt_4_7

• إنشاء صف يقبل السلسلة :

أنشأ تطبيق سطر أوامر و سمه "serialize" , ثمّ أنشأ صف يدعى "myClass4ser" و الذي سوف تتم سلسلته , سوف تكون مهمة هذا الصف حفظ بيانات الاسم الأول و الشهرة و رقم الهاتف و استيرادهم متى أردنا , الرمّاز الخاص بحفظ البيانات الشخصيّة و استيرادها :

أولا ضمن ملف ترويسة الصف "myClass4ser.h" اكتب التالي :

```
#ifndef MYCLASS4SER_H

#define MYCLASS4SER_H

#include <QMetaType>


class myClass4ser

{

public:

   myClass4ser();

   ~myClass4ser();


   void setfName(QString);

   void setlName(QString);

   void setphoneNum(int);
```

```
    QString getfName();
    QString getlName();
    int getphoneNum();

private:
    QString m_fName;
    QString m_lName;
    int m_phoneNum;
};
QDataStream &operator<<(QDataStream &, const myClass4ser &);
QDataStream &operator>>(QDataStream &, myClass4ser &);


Q_DECLARE_METATYPE(myClass4ser)
#endif // MYCLASS4SER_H
```

مثلما تلاحظ أننا عرفنا المناهج الخاصة بالعمليتين "<<" و ">>" بنمط صف مجرى بيانات , وهين المنهجين عاميين "أي خارج الصف (myClss4ser)", والذي سوف نحققهما في ملف تحقيق الصف "myClass4ser.cpp" , ثمّ كتبنا الماكرو :

```
Q_DECLARE_METATYPE(myClass4ser)
```

و الذي مهمته تسجيل الصف "myClass4ser" ضمن الأنماط المعرّفة في Qt , حيث يتم استيراد هذا الماكرو من المكتبة "QMetaType" , سوف نتكلم عنه بتفصيل أكبر ضمن فصل المسلك .

الآن داخل ملف التحقيق "myClass4ser.cpp" اكتب الآتي :

```
#include "myclass4ser.h"
```

```
myClass4ser::myClass4ser()
{
    m_fName = "";
    m_lName = "";
    m_phoneNum = 0;
}

myClass4ser::~myClass4ser(){}

void myClass4ser::setfName(QString fName){
m_fName = fName ;
}

void myClass4ser::setlName(QString lName){
m_lName = lName ;
}

void myClass4ser::setphoneNum(int phoneNumber){
m_phoneNum = phoneNumber ;
}

QString myClass4ser::getfName(){
```

```
return m_fName;
}
QString myClass4ser::getlName(){
return m_lName;
}

int myClass4ser::getphoneNum(){
return m_phoneNum;
}

QDataStream &operator<<(QDataStream &dataStream, const
myClass4ser &myClass){
   myClass4ser myObj = myClass ;
   dataStream << myObj.getfName();
   dataStream << myObj.getlName();
   dataStream << myObj.getphoneNum();
   return dataStream;
}
QDataStream &operator>>(QDataStream &dataStream, myClass4ser
&myClass){
   QString fName;
   QString lName;
   int phoneNumber;
```

```
    dataStream >> fName ;
    dataStream >> lName ;
    dataStream >> phoneNumber ;

    myClass.setfName(fName);
    myClass.setlName(lName);
    myClass.setphoneNum(phoneNumber);

    return dataStream;
}
```

حققنا المنهجين العامين العمليتين "<<" و ">>" الخاصين بتهيئة الصف "myClass4ser" لعملية سلسلته ضمن كائن مجرى بيانات , داخل الكتلة "main" للتطبيق , اكتب الرمّاز التالي :

```
#include <QtCore/QCoreApplication>
#include "myclass4ser.h"
#include <iostream>
#include <string>
#define ln << std::endl <<
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    qRegisterMetaType<myClass4ser>("myClass4ser");
```

```
    std::string fName;
    std::string lName;
    int phoneNumber;
    std::cout << "PLZ Enter the first name , last name and ur phone
number :\n"  ;
    std::cin >> fName >> lName >> phoneNumber ;
    myClass4ser obj4saveData;
    obj4saveData.setfName(fName.data());
    obj4saveData.setlName(lName.data());
    obj4saveData.setphoneNum(phoneNumber);

    QByteArray byteArray;
    QDataStream write(&byteArray,QIODevice::WriteOnly);
    write.setVersion(QDataStream::Qt_4_7);
    write << obj4saveData ;

    myClass4ser obj4loadData;
    QDataStream read(&byteArray,QIODevice::ReadOnly);
    read.setVersion(QDataStream::Qt_4_7);
    read >> obj4loadData ;
    std::cout << obj4loadData.getfName().toAscii().data()
              ln  obj4loadData.getlName().toAscii().data()
              ln obj4loadData.getphoneNum();
```

```
    return a.exec();

}
```

هنا سجلنا أولا الصف "myClass4ser" ***نتكلم عنه لا حقل في فصل المسالك*** **, ثمّ تم تخزين البيانات الشخصية المدخلة على مثيل الصف "myClass4ser" من ثمّ أخضعنا هذا المثيل لعملية السلسة , بعدها تم فك عملية سلسة البيانات الموجودة داخل مصفوفة البايت "byteArray" , و طبعناها على شاشة سطر الأوامر , نفذ التطبيق و اختبر النتيجة .**

الصف QTextStream :

صف مجرى النص يمتلك هذا الصف مناهج تفيد في معالجة النصوص ضمن مجرى, بإمكانه أن يستقبل البيانات النصية من الصف QIODevice , QByteArray , QString , تم إنشاء الصف QTextStream لكتابة و قراءة نصوص كاملة نصوص سطريه كلمات أرقام الخ... بطريقة سهلة من خلال استخدام معاملاته .

مثال لقراءة بيانات من شاشة سطر الأوامر و من ثمّ طباعتها على الشاشة بعد إضافة ترقيم لكل سطر مدخل , عدد السطور المسموح إدخالها هي ستة سطور.

انظر الرمّاز :

```
#include <QtCore/QCoreApplication>

#include <iostream>

#include <QTextStream>

#include <QFile>

int main(int argc, char *argv[])

{

    QCoreApplication a(argc, argv);

    QTextStream stream(stdin);

    QString line;

    QString text;

    int i = 0;
```

```
  while (!(i == 6)){
    line = stream.readLine();
    i++;
    (text +=QString::number(i)+ "." + line +"\n");
  }
  std::cout << text.toAscii().data() ;
  return a.exec();
}
```

صرحنا عم مثيل للصف QTextStream يدعى stream مررنا لوسيط بناءه الماكرو stdin الموجود داخل الملف الرأسي iostream , الماكرو stdin سيقوم بقراءة جميع البيانات المدخلة من شاشة سطر الأوامر فلذا سوف تكتب جميع البيانات المدخلة من شاشة سطر الأوامر على مجرى النص stream بشكل مباشر .

مثال آخر :

```
  QTextStream streamIn("0B1111 017 15 0XF");
  int BNum, ONum, DNum, HNum;
  streamIn  >> bin >> BNum >> oct >> ONum >> dec >> DNum >> hex
>> HNum ;
  std::cout << BNum << endl << ONum  << endl << DNum << endl <<
HNum ;
```

هنا مررنا لوسيط المثيل streamIn أربعة أرقام ,الرقم الأول ينمط ثنائي الثاني ثماني الثالث عشري أما الرابع ست عشري , كتبنا قيمة كل رقم بالنمط العشري على المتحولات الظاهرة لك داخل الرمّاز ثمّ طبعناها على شاشة سطر الأوامر , تم تحويل الأرقام إلى النمط العشري عند كتابتها على المتحولات الخاصة لحفظ الأرقام من المجرى بوساطة المناهج و التي تدعى هنا كمصطلح معاملات (bin-oct-dec-hex) الموجودة داخل الصف QTextStream, نفذ التطبيق و اختبر النتيجة .

لتحويل نمط ترميز النص يتوجب علينا استخدام المنهج setCodec() :

```
void QTextStream::setCodec ( const char * codecName )
```

انظر النص البرمجي التالي :

```
QTextStream stream("Hasan Al-Morhej");

stream.setCodec("UTF-16");
```

حولنا نمط ترميز النص إلى الترميز ذي النوع UFT-16 من خلال المنهج setCodec(const char*) بسهولة .

❖أنواع ترميز النصوص المدعومة في Qt :

- **Apple Roman**
- **Big5**
- **Big5-HKSCS**
- **CP949**
- **EUC-JP**
- **EUC-KR**
- **GB18030-0**
- **IBM 850**
- **IBM 866**
- **IBM 874**
- **ISO 2022-JP**
- **ISO 8859-1 to 10**
- **ISO 8859-13 to 16**
- **Iscii-Bng, Dev, Gjr, Knd, Mlm, Ori, Pnj, Tlg, and Tml**
- **JIS X 0201**
- **JIS X 0208**
- **KOI8-R**
- **KOI8-U**
- **MuleLao-1**
- **ROMAN8**
- **Shift-JIS**
- **TIS-620**
- **TSCII**
- **UTF-8**
- **UTF-16**
- **UTF-16BE**

- **UTF-16LE**
- **UTF-32**
- **UTF-32BE**
- **UTF-32LE**
- **Windows-1250 to 1258**
- **WINSAMI2**

أنهينا الحديث عن معالجة المجاري , لننتقل إلى القسم الثاني من هذا الفصل و الذي يدعى "قسم معالجة الملفات و المجلدات (الأدلة)" .

❖قسم معالجة الملفات و المجلدات (الأدلة) .

بعد أن أنهينا التكلم عن المصفوفات و المجاري في Qt التي نستخدمها لمعالجة كتل البيانات بغض النظر عن الموضع الفيزيائي لها , سوف نبدأ بشرح كيفية الحفظ الفيزيائيّ لتلك البيانات التي تمّ معالجتها .

• الصف QSettings :

أنشأ هذا الصف لحفظ إعدادات التطبيق باستقلالية عن نظام التشغيل , تستطيع من خلاله إنشاء مجموعة مفاتيح كل منها تحوي مجموعة أخرى و قيم عديدة , إذا لم يتم تحديد مسار ملف التي حفظت الإعدادات داخله سوف يقوم بشكل تلقائي بحفظ ملف الإعدادات ضمن المسار الإفتراضي , تحفظ عادة بيانات الإعدادات إما بتنسيق NativeFormat التنسيق الإفتراضي أو التنسيق IniFormat و تستطيع أيضا إنشاء تنسيق خاص بك من خلال المنهج registerFormat يجب عندها أن تضع التنسيق InvalidFormat للكائن QSettings, انظر الجدول الخاص بالتعداد QSettings::Format:

التنسيق الإفتراضي , تحفظ بيانات الإعدادات ضمن إما ملف أو مسجل النظام حسب نظام التشغيل الحامل للتطبيق , في نظام Win تحفظ البيانات داخل مسجل النظام Registry , في نظام Mac OS تحفظ البيانات داخل ملف plist. , أما في نظام Unix داخل ملف Ini	**QSettings::NativeFormat**
بغض النظر عن نظام التشغيل تحفظ الإعدادات داخل ملف Ini	**QSettings::IniFormat**

نستخدم هذا التنسيق عندما ننشأ تنسيق خاص بنا و نريد وضعه كتنسيق حالي للكائن QSettings الذي نستخدمه	QSettings::InvalidFormat

نستطيع تحديد مسار الملف الذي سوف تحفظ بيانات الإعدادات داخله من خلال المنهج :

void QSettings::setPath (Format format, Scope scope, const QString & path)

مثلما تلاحظ يوجد وسيط يدعى Scope يستخدم من أجل مجال الرؤية للإعدادات , انظر الجدول للتعداد QSettings::Scope :

الإفتراضي , مجال الرؤية محدد فقط للمستخدم الحالي	QSettings::UserScope
مجال الرؤية محدد من اجل جميع المستخدمين على نظام التشغيل	QSettings::SystemScope

في حال لم نحدد مسار الملف و كان أي تنسيق ماعدا QSettings::NativeFormat فعندها سوف يحفظ بشكل تلقائي ضمن المسار المحدد في Qt حسب نظام التشغيل , انظر الجدول :

المسار	مجال الرؤية	التنسيق	المنصة
%APPDATA%	UserScope	IniFormat	Windows
%COMMON_APPDATA%	SystemScope		
$HOME/.config	UserScope	NativeFormat,IniFormat	Unix
/etc/xdg	SystemScope		
$HOME/.config	UserScope	IniFormat	Mac OS X
/etc/xdg	SystemScope		

أما إذا كان التنسيق QSettings::NativeFormat فعندها يتم حفظ بيانات الإعدادات في المسارات الظاهرة في الجدول :

المسار	مجال الرؤية	التنسيق	المنصة
HKEY_CURRENT_USER\Software HKEY_LOCAL_MACHINE\Software	UserScope SystemScope	NativeFormat	Windows
$HOME/.config /etc/xdg	UserScope SystemScope	NativeFormat	Unix
$HOME/Library/Preferences /Library/Preferences	UserScope SystemScope	NativeFormat	Mac OS X

عندما نريد أن نستخدم الصف QSettings يتوجب علينا أولا أن نضع إما اسم الشركة و التطبيق ليكون كمفتاح رئيسي أو مسار و اسم المفتاح أو الملف الذي نريد حفظ الإعدادات فيه , انظر الرمّاز :

```
QSettings  settings("My Company","My App");

settings.setValue("user name","hasan al-morhej");
```

ملاحظة: بيانات المفتاح من نمط QVariant .

إذا كنا نريد إنشاء مفتاح فرعي ثمّ معرف القيمة user name فقط أضف الرمز "/" قبل اسم معرف القيمة :

```
settings.setValue("sec/user name","hasan al-morhej");
```

للوصول إلى مفتاح ما في Win داخل مسجل النظام يتوجب علينا فقط وضع مسار المفتاح :

```
settings.value("HKEY_CURRENT_USER\\. . .
```

يجب علينا قبل استخدام الوصول إلى مفتاح موجود في مسجل النظام Registry لنظام تشغيل Win أن نختبر أن النظام الحالي هو MS Win لأننا إذا نفذنا تطبيق على نظام Mac OS أو

أي نظام آخر غير MS Win و كان يحوي وصول إلى مسجل نظام Win فسوف يسبب ذلك لنا الكثير من المشاكل , نستطيع اختباره بوساطة الماكرو التالي :

```
#ifdef Q_OS_WIN

  //do anything

#endif
```

لإنشاء مجموعة من المفاتيح و القراءة منها و الكتابة عليها يتوجب علينا أن نستخدم المنهج beginGroup و المنهج endGroup, انظر الرمّاز :

```
  QSettings settings("My Company","My App");

  settings.beginGroup("security");

  settings.setValue("user name","hasan al-morhej");

  settings.setValue("password","hasan al-morhej");

  settings.endGroup();
```

اسم المجموعة هنا (المفتاح الرئيسي لما تحته من المفاتيح) security .

للكتابة أو القراءة من مجموعة داخل مجموعة أخرى ببساطة بعد استدعاء المنهج beginGroup للمجموعة الأولى نستدعيه مرة ثانية للمجموعة الثانية و نستدعي أيضا المنهج endGroup لكل استدعاء beginGroup.

لحذف مفتاح نستخدم المنهج :

```
void QSettings::remove ( const QString & key )
```

لكتابة مصفوفة من البيانات نستخدم المنهج :

```
void QSettings::beginWriteArray ( const QString & prefix, int size = -1 )
```

و المنهج :

```
void QSettings::setArrayIndex ( int i )
```

لوضع الدليل الحالي للمصفوفة , يتوجب وضعه بعد المنهج beginWriteArray .

عند الانتهاء من استخدام المصفوفة نستدعي المنهج :

```
void QSettings::endArray ()
```

أما لقراءة مصفوفة من ملف إعدادات نستخدم المنهج :

```
int QSettings::beginReadArray ( const QString & prefix )
```

يعيد هذا المنهج حجم المصفوفة التي يحويها , نستخدم معه المنهج setArrayIndex و المنهج endArray.

لكتابة مصفوفة :

```
settings.beginWriteArray ("array");
for (int i = 0; i <= size ; i++)
{
setArrayIndex(i);
settings.setValue("val1",QVariant());
settings.setValue("val2",QVariant());
}
settings.endArray();
```

لقراءتها :

```
settings.beginReadArray ("array");
for (int i = 0; i <= size ; i++)
{
setArrayIndex(i);
QVariant var1 = settings.value("val1");
QVariant var2 = settings.value("val2");
}
settings.endArray();
```

انتهينا من شرح الصف QSettings الخاص بحفظ إعدادات التطبيق, لننتقل للتكلم عن الصف QDir الخاص بالوصول إلى المجلدات .

- **الصف QDir :**

نستخدمه للوصول إلى المجلدات و محتوياتها , يمتلك العديد من المناهج لمعالجة المجلدات و الملفات من إنشاء و حذف و قراءة محتوى مجلد و بعض المناهج الساكنة التي تعيد مسار المجلدات الرئيسية في النظام كمجلد الملفات المؤقتة و جذر النظام و الخ...

الصف QDir بسيط الاستخدام , لنرى ذلك من خلال رمّاز لإنشاء مجلد بعد أن يختبر إن كان موجود أو لا :

```
QDir dir;

dir.setPath("C:");

if(!dir.exists("hasan al-morhej"))

   dir.mkdir("hasan al-morhej");
```

لحذف مجلد ما:

```
dir.rmdir("c:/ hasan al-morhej");
```

لإنشاء مسار كامل نستخدم المنهج mkPath :

```
dir.mkpath("c:/dir1/dir2/dir3");
```

لحذف ملف :

```
dir.remove("c:/dir1/dir2/dir3/data.txt");
```

لإعادة تسمية ملف أو مجلد نستخدم المنهج rename :

```
QDir dir;

dir.setPath("C:");

dir.rename("dir1","cd");
```

لإعادة عدد الملفات و المجلدات التي يحويها الدليل الحالي نستخدم المنهج :

```
uint QDir::count () const
```

إذا كنا نريد استرجاع مصفوفة من أسماء الملفات التي تكون داخل مجلد ما والتي صفتها للقراءة فقط و يكون امتدادها txt نستطيع فعل ذلك بوساطة المنهج entryList لنرى ذلك :

```
QDir dir;

dir.setPath("C:/Text");

dir.setNameFilters(QStringList() << "*.txt");

QStringList strFiles=dir.entryList(QDir::AllEntries, QDir::Name);

for (int i=0 ; i <= strFiles.count()-1 ; i++)

std::cout <<strFiles.at(i).toAscii().data() << std::endl;
```

صرحنا عن مثيل للصف QDir يدعى dir دليله الحالي هو "C:/Text" المجلد Text داخل القرص C , نفذنا عامل التصفية لكل الملفات النصية من خلال المنهج setNameFilters يستخدم هذا المنهج الرموز البديلة لتصفية الملفات Wildcard (راجع الفصل الثاني الجدول الخاص بالتعداد Qt::MatchFlags) وضعنا كافة أسماء الملفات التي يكون امتداده txt. داخل المثيل strFiles من خلال المنهج entryList ثمّ طبعنا هذه الأسماء على شاشة سطر الأوامر.

بالنسبة للمنهج entryList الذي يعيد مصفوفة نصية بأسماء الملفات له وسيطين , الوسيط الأول التعداد QDir::Filter الخاص بالتصفية حسب الصفات الموضحة بالجدول الخاص به, أما الوسيط الثاني التعداد QDir::SortFlag الخاص بترتيب الملفات حسب الحجم الاسم الخ..

جدول التعداد QDir::Filter :

إعادة قائمة من المجلدات المطابقة لعامل التصفية	QDir::Dirs
إعادة قائمة من المجلدات من دون الأخذ بعين الاعتبار عامل التصفية	QDir::AllDirs
إعادة قائمة من الملفات المطابقة لعامل التصفية	QDir::Files
إعادة قائمة من سواقات الأقراص (لا يعمل تحت منصة Unix)	QDir::Drives
عدم إعادة الملفات و المجلدات التي تحوي رموز ارتباطات (كملف اختصار لتطبيق أو	QDir::NoSymLinks

موقع ويب)	
عدم إعادة الملفات و المجلدات التي تحوي محارف رموز الاختصار "." أو ".." , هذه النقط تعتبر من المحارف الخاصة ضمن نظام التشغيل و ليس النقطة الاعتيادية كنقطة فاصل الامتداد عن اسم الملف	QDir::NoDotAndDotDot
عدم إعادة الملفات و المجلدات التي تحوي المحرف الخاص"."	QDir::NoDot
عدم إعادة الملفات و المجلدات التي تحوي المحرف الخاص".."	QDir::NoDotDot
QDir::Dirs | QDir::Files | QDir::Drives	QDir::AllEntries
إعادة قائمة من المجلدات و الملفات التي تسمح الوصول إليها من أجل القراءة منها	QDir::Readable
إعادة قائمة من المجلدات و الملفات التي تسمح الوصول إليها من أجل الكتابة عليها	QDir::Writable
إعادة قائمة من المجلدات و الملفات التي تسمح الوصول إليها من أجل التنفيذ	QDir::Executable
إعادة قائمة من المجلدات و الملفات التي قد تم التعديل عليها مسبقا (لا يعمل تحت منصة Unix)	QDir::Modified
إعادة قائمة من المجلدات و الملفات المخفية	QDir::Hidden
إعادة قائمة من مجلدات و ملفات النظام	QDir::System
سوف يكون عامل التصفية حساس لحالة الأحرف	QDir::CaseSensitive

أما التعداد QDir::SortFlag الخاص بترتيب الملفات ضمن القائمة المعادة الخاصة بالكائن QDir ,انظر الجدول:

الترتيب حسب الاسم	QDir::Name
الترتيب حسب آخر وقت عدل فيه الملف أو المجلد	QDir::Time
الترتيب حسب الحجم	QDir::Size
الترتيب حسب النوع (حسب الامتداد)	QDir::Type
عدم الترتيب	QDir::Unsorted
عدم الترتيب (الحالة الإفتراضية)	QDir::NoSort
ضع المجلدات أولا للترتيب ثمّ الملفات	QDir::DirsFirst
ضع الملفات أولا للترتيب ثمّ المجلدات	QDir::DirsLast
الترتيب حسب عكس الترتيب الموضوع	QDir::Reversed
الترتيب من دون الأخذ بعين الاعتبار حالة الأحرف كبيرة أم صغيرة	QDir::IgnoreCase
الترتيب بشكل ملائم باستخدام إعدادات الموقع الحالي للملفات و المجلدات المطلوب ترتيبها	QDir::LocaleAware

ذكرنا مسبقا أنه بعض الصفات الخاصة بتصفية الملفات و المجلدات لا تعمل تحت منصة Unix لذا يجب علينا أن نختبر نظام التشغيل الذي يعمل عليه التطبيق حاليا لكي نضع الصفات المناسبة للنظام الحالي , انظر الجدول الخاص بماكروات أشهر الأنظمة المعرفة :

- Q_OS_UNIX
- Q_OS_UNIXWARE
- Q_OS_LINUX
- Q_OS_MAC
- Q_OS_SYMBIAN
- Q_OS_MSDOS
- Q_OS_WIN

- Q_OS_WIN32
- Q_OS_WINCE

إذا كنا نريد معرفة المسار الحقيقي لمجلد أو ملف اختصار يجب علينا استخدام المنهج :

```
QString QDir::canonicalPath () const
```

بعد أن أنهينا هذه الفقرة جرب أن تنشأ تطبيق بسيط للبحث عن الملفات باستخدام الصف QDir .

انتهينا من شرح الصف QDir الخاص بالوصول إلى المجلدات و الملفات .

• الصف QDirIterator :

يستخدم هذا الصف لتخزين مسارات الملفات و المجلدات المحتواة داخل مسار محدد ضمن مكرر Iterator .

مثال لعرض جميع مسارات الملفات و المجلدات الفرعية و محتواها الموجودة داخل المسار "C:/Windows" على شاشة سطر الأوامر :

```
QDirIterator dirIterator("C:/Windows",QDirIterator::Subdirectories);

while (dirIterator.hasNext())

    std::cout << dirIterator.next().toAscii().data() << std::endl;
```

صرحنا عن مثيل للصف QDirIterator المسار الرئيسي له هو جذر النظام Win (أي المسار الذي سوف يعرض محتوياته) و مررنا لوسيطه الثاني الخاص بتحديد كيفية البحث عن محتويات المسار المحدد أي مستوى رؤيته للملفات و المجلدات, هنا مررنا العنصر QDirIterator::Subdirectories من التعداد QDirIterator::IteratorFlag بوساطته سوف يعرض جميع المجلدات الفرعية الموجودة داخل المسار المحدد و يعرض محتواها أيضا إن كان مجلد أو ملف .

سوف تبقى عملية التكرار طالما أن المثيل dirIterator يملك داخله مسارات ملفات و مجلدات تمّ ذلك بوساطة المنهج hasNext() ,أمّا المنهج next() يرجع قيمة نصية تحوي المسار الذي قد تمّ الوصول إليه حاليا داخل المكرر و من ثمّ يقوم بإزالتها منه .

التعداد QDirIterator::IteratorFlag :

سوف لن يعرض إلا أسماء المجلدات و الملفات الموجودة داخل المسار المحدد دون عرض محتوى المجلدات الفرعية	QDirIterator::NoIteratorFlags
سوف يتم عرض جميع محتويات المجلد الحالي و المجلدات الفرعية	QDirIterator:: Subdirectories
يرفق مع العنصر Subdirectories حيث عندما يجد ملف اختصار لمجلد يعرض محتويات هذا المجلد	QDirIterator:: FollowSymlinks

لننتقل للتكلم عن الصف الخاص بالوصول إلى ملف و معالجته QFile .

- الصف QFile :

يستخدم هذا الصف للوصول إلى الملفات و القراءة منها و الكتابة عليها , يمتلك مناهج سهلة الاستخدام لتنفيذ هذه العمليات , ذكرنا مسبقا أن الصف QFile يرث الصف QIODevice , يمتلك الصف QFile المقدرة على قراءة و كتابة البيانات النصية و الثنائية من و على ملف و معالجة هذه البيانات باستخدام الصف QTextStream و الصف QDataStream .

لإنشاء ملف و الكتابة عليه انظر الرمّاز التالي :

```
QFile file("c:/test.txt");

file.open(QFile::WriteOnly);

file.write("hasan al-morhej ");

file.close();
```

لقراءة ملف :

```
QFile file("c:/test.txt");

file.open(QFile::ReadOnly);

std::cout << file.readAll().data();

file.close();
```

للكتابة على ملف بوساطة الصف QTextStream انظر الرمّاز :

```
QFile file("c:/test.txt");
if(file.open(QIODevice::WriteOnly | QIODevice::Text))
{
   QTextStream stream(&file);
   stream << "test test test ..." ;
   file.close();
}
```

أما للقراءة من ملف بوساطة الصف QTextStream , انظر الرمّاز التالي :

```
QString strLine;
QFile file("c:/test.txt");
if(file.open(QIODevice::ReadOnly | QIODevice::Text))
{
   QTextStream stream(&file);
   while(!stream.atEnd()){
      strLine = stream.readLine() ;
      std::cout << strLine.toAscii().data() << std::endl;
   }
   file.close();
}
```

إذا أردنا إضافة محارف بموقع محدد من ملف يجب علينا أولا استخدام المنهج seek للانتقال إلى الموقع الذي نريد ثمّ استخدام المنهج putChar لكتابة المحرف :

```
QFile file("c:/test.txt");
```

```
if(file.open(QFile::WriteOnly))
{
file.seek(10);
file.putChar('Q');
file.putChar('t');
file.close();
}
```

لإعطاء صلاحية لملف تم الوصول إليه يجب أن نستخدم المنهج :

```
bool QFile::setPermissions ( Permissions  permissions )
```

ولقراءة الصلاحية نستخدم المنهج :

```
Permissions QFile::permissions () const
```

مثال :

```
QFile file("c:/test.txt");
if(file.open(QFile::WriteOnly)){
file.setPermissions(QFile::WriteOwner);
//do anything..
file.close();
}
```

جدول يحوي عناصر التعداد QFile::Permission :

صلاحية القراءة من قبل مالك الملف	QFile::ReadOwner
صلاحية الكتابة من قبل مالك الملف	QFile::WriteOwner
صلاحية التنفيذ من قبل مالك الملف	QFile::ExeOwner
صلاحية القراءة من قبل المستخدم	QFile::ReadUser

صلاحية الكتابة من قبل المستخدم	QFile::WriteUser
صلاحية التنفيذ من قبل المستخدم	QFile::ExeUser
صلاحية القراءة من قبل مجموعة	QFile::ReadGroup
صلاحية الكتابة من قبل مجموعة	QFile::WriteGroup
صلاحية التنفيذ من قبل مجموعة	QFile::ExeGroup
صلاحية القراءة من قبل أي شخص	QFile::ReadOther
صلاحية الكتابة من قبل أي شخص	QFile::WriteOther
صلاحية التنفيذ من قبل أي شخص	QFile::ExeOther

إذا حصل خطأ ما عند الوصول إلى ملف أو القراءة منه أو الكتابة عليه نستطيع تحديد ماهية الخطأ الذي حصل باستخدام المنهج QFile::error() الذي يعيد الخطأ على شكل عنصر من عناصر التعداد QFile::FileError :

لم يحصل أي خطأ	QFile::NoError
حصل خطأ عند القراءة من الملف	QFile::ReadError
حصل خطأ عند الكتابة إلى الملف	QFile::WriteError
حصل خطأ قاتل	QFile::FatalError
حصل خطأ نتيجة خطأ بتركيب الملف	QFile::ResourceError
لم يستطيع فتح الملف	QFile::OpenError
تم إحباط العملية التي تتم حاليا للملف	QFile::AbortError
انتهى الوقت الواجب فيه إتمام العملية على الملف	QFile::TimeOutError
خطأ غير معروف	QFile::UnspecifiedError
عدم الاستطاعة على حذف الملف	QFile::RemoveError
عدم الاستطاعة على إعادة تسمية الملف	QFile::RenameError

عدم الاستطاعة على تغيير الموقع ضمن الملف	QFile::PositionError
عدم الاستطاعة على إعادة تحجيم الملف	QFile::ResizeError
عدم الاستطاعة على تغيير الصلاحية للملف	QFile::PermissionsError
عدم الاستطاعة على نسخ الملف	QFile::CopyError

لحذف ملف نستخدم المنهج :

```
bool  QFile::remove ()
```

لإعادة تسمية الملف نستخدم المنهج :

```
bool  QFile::rename ( const QString & newName )
```

لإعادة تحجيم الملف نستخدم المنهج :

```
bool  QFile::resize ( qint64 sz )
```

نكون الآن انتهينا من الفقرة التي تحوي شرح الصف QFile , لننتقل للتكلم عن الصف الخاص بجلب معلومات ملف ما الصف QFileInfo .

- **الصف QFileInfo :**

لمعرفة معلومات ملف كتاريخ الإنشاء و الحجم و المسار الحقيقي له و الصلاحيات الخ... ما علينا إلا استخدام الصف QFileInfo الخاص بجلب معلومات ملف محدد و الذي يمتلك واجهة (مناهج) سهلة الاستخدام .

انظر الرمّاز التالي الذي يحوي كيفيّة استخدام أغلب مناهج الصف QFileInfo :

```
#include <QtCore/QCoreApplication>

#include <QFileInfo>
```

```
#include <iostream>

#include <QDateTime>

void printDateAndTime(char*, QDateTime);

int main(int argc, char *argv[])

{

   QCoreApplication a(argc, argv);


   QFileInfo fileInfo("c:/test.txt");

   std::cout << "Base Name is " << fileInfo.baseName().toAscii().data()
<< std::endl ;

   std::cout << "File Name is " << fileInfo.fileName().toAscii().data() <<
std::endl ;

   std::cout << "File path is " << fileInfo.filePath().toAscii().data() <<
std::endl ;

   std::cout << "Dir path is " << fileInfo.absolutePath().toAscii().data()
<< std::endl ;

   std::cout << "Size is " <<
QString::number(fileInfo.size()).toAscii().data()

   << " bytes" << std::endl ;

   std::cout << "Writable is " << (fileInfo.isWritable() ==
true?"true":"false")

   << std::endl ;

   std::cout << "Readable is " << (fileInfo.isReadable() ==
true?"true":"false")

   << std::endl ;
```

```
   std::cout << "Executable is " << (fileInfo.isExecutable() ==
true?"true":"false")

   << std::endl ;

   std::cout << "Hidden is " << (fileInfo.isHidden() ==
true?"true":"false") << std::endl ;


   printDateAndTime("Created Date is ", fileInfo.created());

   printDateAndTime("Last Modified Date is ", fileInfo.lastModified());

   printDateAndTime("Last Read Date is ", fileInfo.lastRead());


   return a.exec();

}


void printDateAndTime(char* str,QDateTime dateTime)

{

   std::cout << str << dateTime.date().day() << "/" <<
dateTime.date().month()

   << "/" << dateTime.date().year()

   << " " << dateTime.time().toString().toAscii().data() << std::endl ;

                                                                    }
```

أنهينا الفقرة التي تتكلم عن الصف الخاص بجلب معلومات ملف محدد QFileInfo ,لننتقل الآن للتكلم عن الصف QFileSystemModel الخاص بجلب نموذج البيانات لنظام الملفات المحلي .

- **الصف QFileSystemModel :**

مثلما ذكرنا أن هذا الصف خاص بجلب نموذج البيانات لنظام الملفات المحلي يرث الصف QAbstractItemModel , هذا الصف يهيئ لنا نموذج شجري يحوي المجلدات و الملفات و معلوماتها التي تكون تحت الجذر (المجلد الرئيسي) الذي نضعه نحن , يمتلك عدة أحداث داخلية أهمها :

يقدح هذا الحدث عند الانتهاء من تحميل ملفات المجلد المحدد , له وسيط وحيد يحوي مسار المجلد الذي قد تم تحميله	void directoryLoaded (const QString & path)
يقدح هذا الحدث عند تغيير اسم ملف , له وسيطين الأول مسار الملف الذي قد تمّ تغيير اسمه , أمّا الثاني اسمه القديم , الوسيط الأخير الاسم الجديد له	void fileRenamed (const QString & path, const QString & oldName, const QString & newName)
يقدح عند تغيير مسار الجذر (المجلد , السواقة الرئيسي) , أمّا الوسيط newPath فهو مسار الجذر الجديد	void rootPathChanged (const QString & newPath)

الآن لنكتب رمّاز يعرض لنا قائمة شجرية تحوي المجلدات و الملفات التي تكون تحت ظلال الجذر قرص C ويتغير عنوان النافذة عند كل تحميل لمجلد , انظر الرمّاز :

```
QFileSystemModel *fileSysModel = new QFileSystemModel();

QTreeView *treeView = new QTreeView();


treeView->setModel(fileSysModel);

treeView->setRootIndex(fileSysModel->setRootPath("C:"));

treeView->show();
```

```
QObject::connect(fileSysModel,SIGNAL(directoryLoaded(QString)),tree
View
```

```
, SLOT(setWindowTitle(QString)));
```

أولا صرحنا عن الحدث fileSysModel من الصف QFileSystemModel و عن حدث آخر يدعى treeView من الصف QTreeView , وضعنا نموذج القائمة الشجرية هو الحدث fileSysModel نموذج من بيانات المجلدات و الملفات , ثمّ وضعنا الدليل الرئيسي لنموذج القائمة الشجرية هو مسار الجذر لنموذج نظام الملفات و المجلدات والذي يكون قرص C , أظهرنا نموذج القائمة الشجرية بعد تهيئتها لتعرض الجذر قرص C بوساطة المنهج ()show , أما الحدث directoryLoaded التابع للحِدثْ fileSysModel الذي يقدح عندما يتم الانتهاء من تحميل مجلد ربطناه مع المقبس setWindowTitle التابع للحِدثْ treeView , بالتالي سوف يتم تغيير عنوان النافذة إلى مسار المجلد الذي تم تحميله من قبل fileSysModel.

ملاحظة : إذا أردنا أن نضع الجذر الرئيسي هو جهاز الكمبيوتر يتوجب علينا أن نترك نص المسار الرئيسي خال:

```
treeView->setRootIndex(fileSysModel->setRootPath(""));
```

- **قراءة و كتابة مستندات XML :**

مستندات XML كثيرة الإستخدام و ذلك لـ استقلاليتها عن نظم التشغيل و سهولة حفظ بيانات فيها و استردادها و صغر الحجم الذي يحجزه مستند XML , سنوضح هنا كيفيّة كتابة مستند XML و قراءته من دون التوسع بالشرح .

تحوي Qt على مجموعة مكاتب خاصة بالوصول و معالجة مستندات XML , عندما نود استخدام أحد هذه الصفوف داخل مشروعنا , يتوجب أن نضيف السطر البرمجي التالي داخل ملف المشروع :

```
QT += xml
```

و ذلك لإخبار المجمع بأننا نريد حزم حزمة "QtXml" ضمن التطبيق .

في هذه الفقرة سوف نستخدم الصف "QXmlStreamReader" لقراءة مستندات "XML" و الصف "QXmlStreamWriter" لإنشاء و كتابة مستند "XML" .يتوجب علينا أن نهيئ كتلة بيانات من أجل الكتابة من خلال استخدام أحد صفوف جهاز الدخل و الخرج "QIODevice" , ثمّ نصرح عن مثيل / حدث للصف "QXmlStreamWriter" , و تحديد

إصدار و تشفير المستند , من ثمّ إنشاء العنصر المراد و إضافة قيم أو عناصر داخله , و ذلك كالآتي :

```
QFile file("profile.xml");

file.open(QIODevice::WriteOnly);

QXmlStreamWriter* writer = new QXmlStreamWriter(&file);

writer->writeStartDocument("1.0");

writer->setCodec("UTF-8");

writer->writeEndDocument();

writer->writeStartElement("profile");

writer->writeTextElement("fname","Hasan");

writer->writeTextElement("lname","Al Morheij");

writer->writeTextElement("phone","+963999999999");

.

.

.

writer->writeEndElement();
```

استخدمنا الصف "QFile" من أجل إنشاء ملف XML و تهيئته من أجل الكتابة , ثمّ صرحنا عن حدث للصف "QXmlStreamWriter" و مررنا لوسيط بنّاءه جهاز الدخل و الخرج المحدد "file" من أجل كتابة مستند الـ "XML" داخله , بدأنا بإنشاء مستند "XML" من خلال استخدام المنهج "writeStartDocument" الذي حددنا من خلاله إصدار مستند "XML" , ثمّ حددنا الترميز الخاص بالمستند "UTF-8" بوساطة المنهج "setCodec" , أنهينا تعريف تروسية مستند "XML" من خلال استدعاء المنهج "writeEndDocument" , ستصبح تروسية تعريف المستند كالآتي :

```
<?xml version="1.0" encoding="UTF-8"?>
```

بعدها أنشأنا عنصر يدعى "profile" داخله عدة عناصر أبناء خاصة باحتواء البيانات الشخصيّة للمستخدم , أخير استدعينا المنهج "writeEndElement" من أجل إغلاق

العنصر "profile" ضمن مستند "XML" , سيبدو شكل رمّاز "XML" بعد تنفيذ الرمّاز كالآتي :

```
<?xml version="1.0" encoding="UTF-8"?>
<profile>
<fname>Hasan</fname>
<lname>Al Morheij</lname>
<phone>+963999999999</phone >
</profile>
```

من أجل قراءة مستند "XML" نستخدم الصف "QXmlStreamReader" , لنقرأء مستند "XML" السابق :

```
QString list;
QFile file("profile.xml");
file.open(QIODevice::ReadOnly);
QXmlStreamReader* reader = new QXmlStreamReader(&file);

reader->readNextStartElement();

if(reader->name() == "profile"){
      list.append("<"+reader->name().toString()+">\n");

while(!reader->atEnd()){
   reader->readNextStartElement();

   if(reader->isStartElement())
```

```
        list.append("<"+reader->name().toString()+">");


    list.append(reader->readElementText());


    if(reader->isEndElement())
        list.append("</"+reader->name().toString()+">\n");
    }


}
```

أولا حددنا الملف "profile.xml" من أجل قراءة و تفسير محتواه , و ذلك من خلال استخدام الصف "QFile" , و أنشئنا حدث للصف "QXmlStreamReader" يدعى "reader" , استدعينا المنهج "readNextStartElement" من أجل الإنتقال لأول عنصر موجود ضمن المستند , ثمّ اختبرنا أن اسم العنصر الحالي هو "profile" لـ يقراءة جميع محتوياته "عناصره الأبناء" , و ذلك من خلال إنشاء حلقة تكرار يتم الخروج منها بانتهاء قراءة جميع محتوى العنصر الحالي "profile" , استدعينا المنهج "atEnd" الذي يخبرنا بأنه تم الإنتهاء من قراءة جميع محتويات "العناصر الأبناء" للعنصر الحالي "profile" , بما أننا نريد قراءة العناصر الأبناء للعنصر "profile" يتوجب علينا استدعاء المنهج "readNextStartElement" لـ يتم قراءة العناصر واحد تلوا الآخر على التسلسل , تركيبة عنصر "XML" هي بادئة العنصر و قيمة و نهاية العنصر :

<startElemetnt> value </endElement>

نقوم بالحصول على اسم العنصر من خلال استدعاء المنهج "name" و ذلك بعد اختبار أنّ الوسم الحالي هو عنصر , المنهج "isStartElement" يرجع "true" في حال كان الوسم الحالي "مكان المؤشر ضمن مستند XML" هو بادئة لعنصر "XML" ,أما المنهج "isEntElement" يرجع "true" في حال كان الوسم الحالي من المستند هو نهاية عنصر "XML" , استخدمنا المنهج "readElementText" من أجل الحصول على قيمة العنصر الحالي "مكان المؤشر الحالي ضمن المستند" .

- الصف **QFileSystemWatcher** :

يستخدم الصف QFileSystemWatcher لمراقبة التعديلات التي تطرأ على المجلدات (الأدلة) و الملفات كالإضافة و الحذف و إعادة التسمية الخ...

مثال بسيط لاستخدام الصف QFileSystemWatcher :

```
QListWidget listWidget;

QFileSystemWatcher* fileSysWatcher = new
QFileSystemWatcher(QStringList()

<< "C:/" << "C:/Text" << "C:/Text/txt.txt");


listWidget.addItems(QStringList() << "C:/" << "C:/Text" <<
"C:/Text/txt.txt");

listWidget.show();


QObject::connect(fileSysWatcher , SIGNAL(fileChanged(QString)) ,

&listWidget ,SLOT(setWindowTitle(QString))) ;

QObject::connect(fileSysWatcher ,
SIGNAL(directoryChanged(QString)) ,

&listWidget ,SLOT(setWindowTitle(QString))) ;
```

صرحنا عن مثيل للصف QListWidget يدعى listWidget و مثيل للصف QFileSystemWatcher يدعى fileSysWatcher مررنا لوسيطه قائمة نصية تحوي القرص C و المجلد Text و الملف txt.txt لمراقبتها , ثمّ أضفنا على كائن القائمة المسارات المراقبة , ربطنا الحدث الداخلي fileChanged(QString) و الحدث directoryChanged(QString) التابعين للحدث fileSysWatcher مع كائن القائمة listWidget بالمقبس setWindowTitle(QString), حيث سوف يتم قدح هذه الأحداث عند أي تغيير يطرأ على المجلدات و الملفات المضافة للكائن fileSysWatcher .

يمتلك عدة مناهج أهمها :

المنهج الخاص بإضافة مسار جديد من أجل المراقبة:

void QFileSystemWatcher::addPath (const QString & path)

المنهج الخاص بإضافة مسارات جديدة من أجل المراقبة:

void QFileSystemWatcher::addPaths (const QStringList & paths)

إعادة قائمة نصية تحوي مسارات المجلدات التي تتم مراقبتها :

QStringList directories () const

إعادة قائمة نصية تحوي مسارات الملفات التي تتم مراقبتها :

QStringList files () const

لإزالة مسار مجلد و ملف من فعل المراقبة :

void removePath (const QString & path)

لإزالة مسارات مجلدات و ملفات من فعل المراقبة :

void removePaths (const QStringList & paths)

- **الصف QProcess :**

يستخدم لتنفيذ برامج خارجية و الاتصال معها :

```
QProcess process;

process.start("notepad",QStringList() << "c:/Text/txt.txt");
```

يستخدم المنهج ()start لتنفيذ تطبيق ما, هنا مررنا لوسيطه الأول مسار و اسم التطبيق "notepad" و لوسيطه الثاني الوسطاء الخارجية الذي نود إدخالها إلى التطبيق ليعالجها هنا مسار ملف نصي لتقرأه المفكرة , نفذ التطبيق و اختبر النتيجة .

إذا أردنا أن نجلب جميع مسارات بيئة النظام المحلي فما علينا سوى استدعاء المنهج ()systemEnvironment التابع للصف QProcess ليعيد قائمة نصية QStringList تحوي جميع مسارات بيئة النظام , انظر الرمّاز التالي :

```
QProcess process;
```

```
for (int i=0; i <= process.systemEnvironment().length() -1 ; i++ )

std::cout << process.systemEnvironment().at(i).toAscii().data()
<<std::endl

<< std::endl ;
```

- **الصف QFileDialog :**

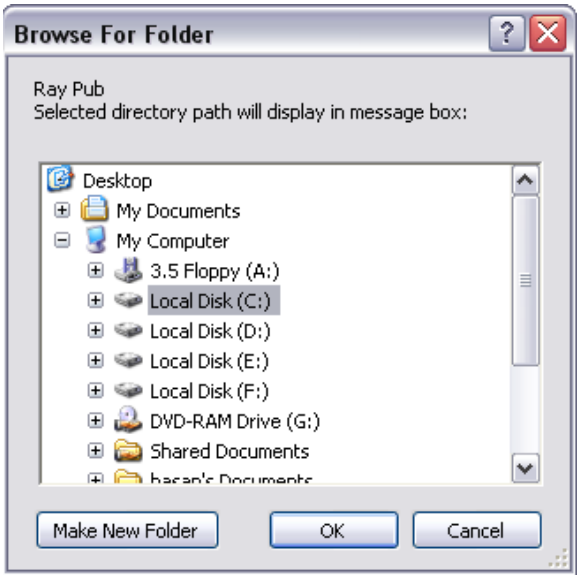
يستخدم هذا الصف من أجل تحديد مجلد , مجلدات أو ملف , ملفات من خلال نافذة حوار سهلة الاستخدام , يرث الصف QDialog , يحوي عدة مناهج ساكنة لإظهار نافذة حوار من أجل حفظ ملف أو فتحه أو اختيار مجلد, نستطيع تصفية أنواع الملفات الذي نريد إظهارها ضمن إطار مستكشف الملفات لنافذة الحوار .

لعرض نافذة حوار لتحديد مجلد نستطيع استخدام المنهج الستاتيكي getExistingDirectory() الموجود ضمن الصف QFileDialog , انظر الرمّاز التالي :

```
,QObject::tr("hasan 0    QString str = QFileDialog::getExistingDirectory(
al-morhej\ nSelected directory path will display in message
box:"),"C:/");

    if(QMessageBox::information(0,"path",str) == QMessageBox::Ok)

    return 0;
```

استدعينا هنا المنهج الستاتيكي getExistingDirectory() من الصف QFileDialog لعرض نافذة حوار من أجل تحديد مجلد , يعيد هذا المنهج مسار المجلد المحدد بنمط نصي QString,مررنا لوسيطه الأول 0 أي الصف الأب له هو الصف الحالي أما الوسيط الثاني وضعنا النص الموضح في الرمّاز ليعرض فوق إطار مستكشف المجلدات , أمّا الوسيط الثالث يحدد المسار الذي سوف يحدده تلقائيّا عند فتح نافذة الحوار الخاص باستكشاف المجلدات (الأدلة) وهنا القرص C.

عند التنفيذ سوف تظهر نافذة كالشكل (4.4) :



الشكل 4.4

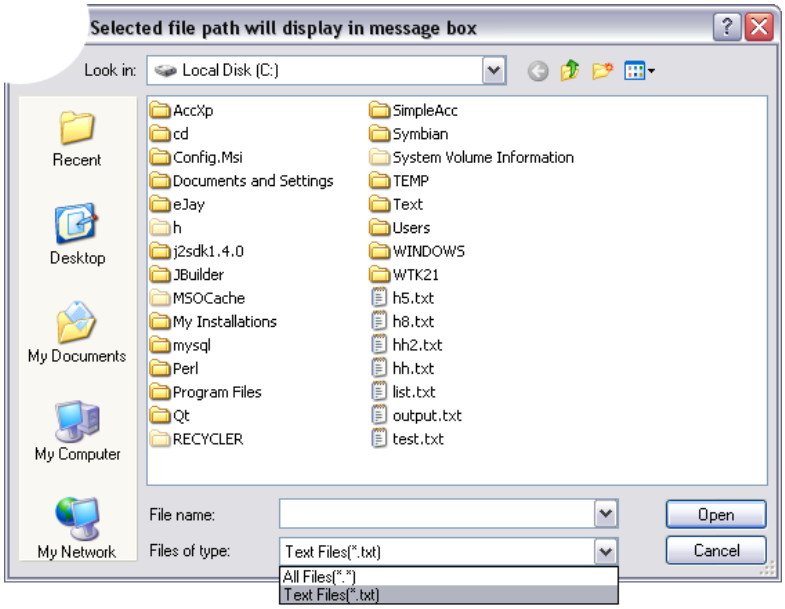
لتحديد ملف من خلال نافذة حوار خاص به يجب علينا استدعاء المنهج الساكن getOpenFileName() التابع للصف QFileDialog , انظر الرمّاز التالي :

```
QString str = QFileDialog::getOpenFileName(0,QObject::tr("Hasan Al-Morhej:Selected file path will display in message box"),"C:/",QObject::tr("All Files(*.*);;Text Files(*.txt)"),0,0);

if(QMessageBox::information(0,"path",str) == QMessageBox::Ok)

    return 0;
```

بالنسبة للوسيط الخاص بالتصفية (تصفية أنواع الملفات) يفصل بين كل نوع و نوع بفاصلتين منقوطتين متتاليتين :

```
All Files(*.*);;Text Files(*.txt)
```

عند التنفيذ سوف ترى الشكل (4.5) :



الشكل 4.5

لتحديد مجموعة من الملفات نستدعي المنهج الساكن getOpenFileNames() انظر الرمّاز التالي :

```
   QStringList str =
QFileDialog::getOpenFileNames(0,QObject::tr("Hasan Al-
Morhej:Selected files path will display in message
box"),"C:/",QObject::tr("All Files(*.*);;Text Files(*.txt)"),0,0);

   QString strforMsg;

   for(int i=0; i <= str.length() - 1 ; i++) strforMsg += str.at(i) + "\n";

   if(QMessageBox::information(0,"path",strforMsg) ==
QMessageBox::Ok)

      return 0;
```

لإظهار نافذة حوار خاصة بحفظ ملف نستدعي المنهج الساكن getSaveFileName() من الصف QFileDialog , انظر الرمّاز التالي :

```
   QString str = QFileDialog::getSaveFileName(0,QObject::tr("Hasan Al-
Morhej:Save File Dialog"),"C:/",QObject::tr("Text File(*.txt);;In
Out(*.io)"),0,0);
```

```
if(QMessageBox::information(0,"path",str) == QMessageBox::Ok)
   return 0;
```

سنكتب الآن رمّاز يحوي على أهم مناهج الصف QFileDialog من تغيير لوحة نص ما ضمن نافذة الحوار إلى تحديد أنواع الملفات و تحديد كيفية قبول الملف الخ... :

```
QFileDialog fileDialog;
QStringList strList;
QString str;
fileDialog.setAcceptMode(QFileDialog::AcceptOpen);
fileDialog.setLabelText(QFileDialog::FileType,QObject::tr("Select any type u want"));
fileDialog.setLabelText(QFileDialog::Reject,QObject::tr("Close the dialog"));
fileDialog.setWindowTitle(QObject::tr("File Dialog"));
fileDialog.setFileMode(QFileDialog::ExistingFile);
fileDialog.setNameFilters(QStringList() << QObject::tr("Text Files(*.txt)") <<
QObject::tr("Input Output Files(*.io)"));
fileDialog.setViewMode(QFileDialog::Detail);
fileDialog.setStyleSheet("color:blue;");
if(fileDialog.exec())
    if(fileDialog.acceptMode() ==  QFileDialog::AcceptOpen)
strList = fileDialog.selectedFiles();
for(int i = 0 ; i <= strList.length() - 1 ; i++) str += strList.at(i) + "\n" ;
```

```
if(QMessageBox::information(0,QObject::tr("path"),str) ==
QMessageBox::Ok)

    return 0;
```

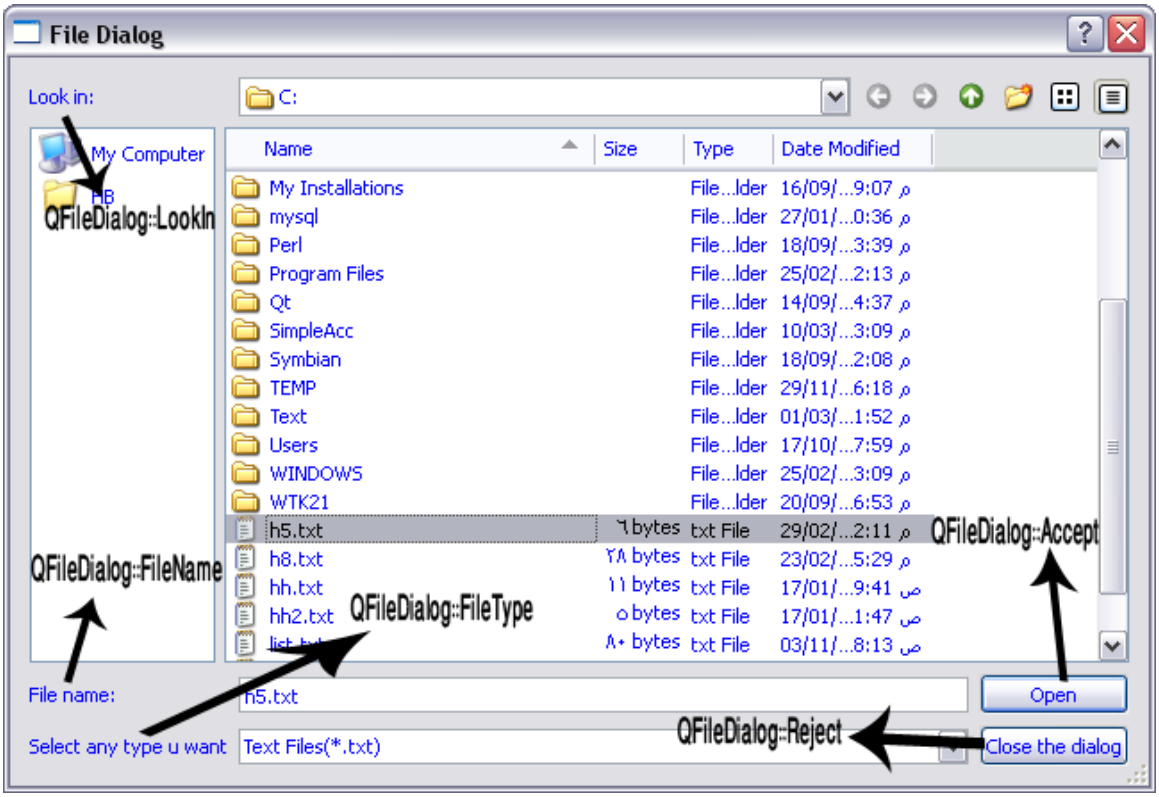
أولا المنهج setAcceptMode يحدد نمط نافذة الحوار إما نمط نافذة حوار من أجل الحفظ أو من أجل الفتح , وسيطه يأخذ نمط التعداد QFileDialog::AcceptMode :

إظهار نافذة الحوار بنمط فتح ملف	**QFileDialog::AcceptOpen**
إظهار نافذة الحوار بنمط حفظ ملف	**QFileDialog::AcceptSave**

ثانيا المنهج setLabelText لتغيير قيمة لوحات النصوص ضمن نافذة الحوار وسيطه يأخذ نمط التعداد QFileDialog::DialogLabel :

لوحة النص الخاصة بالقائمة الرئيسية للمجلدات	**QFileDialog::LookIn**
لوحة النص الخاصة بتحديد اسم الملف	**QFileDialog::FileName**
لوحة النص الخاصة بتحديد نوع الملف	**QFileDialog::FileType**
لوحة النص الخاصة بزر الموافقة على التحديد	**QFileDialog::Accept**
لوحة النص الخاصة بزر الإلغاء	**QFileDialog::Reject**

انظر الشكل (4.6) الخاص بالتعداد QFileDialog::DialogLabel :



الشكل 4.6

ثالثا المنهج setFileMode يحدد ماهية الأنماط المسموحة للملفات و المجلدات إن كان عند العرض أو الضغط على زر الموافقة وسيطه من نمط التعداد QFileDialog::FileMode :

سوف يعرض أي ملف و يسمح باختياره	**QFileDialog::AnyFile**
سوف يعرض فقط الملفات المحدد نمطها و يسمح بتحديد ملف واحد	**QFileDialog::ExistingFile**
سوف لن يسمح باختيار إلّا المجلدات (الأدلة)	**QFileDialog::Directory**
سوف يعرض فقط الملفات المحدد نمطها و يسمح بتحديد عدة ملفات	**QFileDialog::ExistingFiles**
سوف لن يسمح بعرض و اختيار إلّا المجلدات (الأدلة)	**QFileDialog::DirectoryOnly**

رابعا المنهج setNameFilters لتحديد الأنماط التي نريد (امتداد الملفات) .

خامسا المنهج setViewMode نمط عرض شعارات و معلومات المجلدات و الملفات داخل المستكشف (عارض نموذج الملفات و المجلدات) وسيطه الوحيد من نمط التعداد QFileDialog::ViewMode :

عرض الشعارات و المعلومات بنمط تفصيلي	QFileDialog::Detail
عرض الشعارات و المعلومات بنمط قائمة	QFileDialog::List

سادسا و أخيرا استخدمنا المنهج ()exec لتنفيذ و إظهار نافذة الحوار .

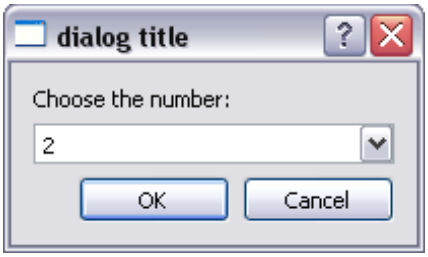
الصف QFileDialog يمتلك خمسة أحداث داخلية و هي :

يقدح عند تغيير المسار الحالي	void currentChanged (const QString & path)
يقدح عند فتح مجلد (الدخول إليه لاستكشافه)	void directoryEntered (const QString & directory)
يقدح عند اختيار ملف	void fileSelected (const QString & file)
يقدح عند اختيار أكثر من ملف	void filesSelected (const QStringList & selected)
يقدح عند تحديد نمط (امتداد) ملف	void filterSelected (const QString & filter)

أنهينا شرح أهم مناهج و أحداث صف نافذة الحوار الخاصة بتحديد ملف أو مجلد الصف QFileDialog .

تكلمنا مسبقا عن صف نافذة الحوار الخاصة بتحديد الطابعة و تعيين إعداداتها الصف QPrintDialog , يوجد أيضا في Qt صف خاص بنافذة حوار تحديد اللون وهو الصف QColorDialog و صف آخر لتحديد الخط الصف QFontDialog , تمّ إنشاء هذان الصفان ليسهلان عمليات تحديد اللون و الخط , يمتلكان واجهة سهلة الاستخدام و مجهزة بشكل

مناسب لعملها , ويوجد أيضا الصف QInputDialog الذي هو عبارة عن نافذة حوار مخصصة لإدخال البيانات إن كانت نصية أو اختيارية أو رقمية الخ.. يحوي زرين الأول موافق (OK) و الثاني إلغاء أمر (Cancel) ولا فتة نصية تدل على ما يتوجب عليك إدخاله و صندوق الإدخال, انظر الشكل (4.7) :



الشكل 4.7

• خلاصة الفصل :

تم التحدث في هذا الفصل عن مصفوفة البت و البايت و المجاري منها مجرى النص و مجرى البيانات و طريقة معالجتها و سلسلتها لكتل البيانات , و جهاز الدخل و الخرج الأساسي في Qt والذي يدعى QIODevice, وأيضا تكلمنا عن طريقة الوصول إلى الملفات و القراءة منها و الكتابة عليها و كتابة و قراءة مستندات XML , و تحديد ملف أو مجلد من خلال نافذة حوار خاصة و طريقة مراقبة ملف أو مجلد معين .

الآن لننتقل إلى الفصل الذي يتكلم عن أهم تقنية لأي تطبيق يحوي معالجة لعدة مهام في نفس الوقت و التي هي المسالك Threads , طبعا جميع التطبيقات الحالية تحوي معالجة لمهام عدة تتخاطب في ما بينها بالوقت ذاته .

الفصل الخامس

المسالك

The Threads

مقدمة Intro :

لشيء جميل سماع مقطوعة موسيقية هادئة للتأمل , الأجمل أن تكون أنت العازف فتصدر لحن ينطق حالتك الراهنة من إلهامك , هل تعلم أن تعلم العزف على آلة موسيقية ما يوجب عليك أن تتعلم كيف تؤدي الفعل الخاص بفصل الحواس , ماذا يعني فصل الحواس ؟

فصل الحواس لكأن تستطيع السمع أو التكلم و أنت تعزف على آلة موسيقية ما, كالعزف على آلة العود يجب على العازف أن تكون يده اليسار على الزند ليضغط بأصبعه على وتر ما من أجل صدور صوت العلامة التي يريدها ولكن هذا لا يكفي, يتوجب عليه أيضا أن يضرب بالريشة الموجودة في يده اليمين على الوتر المراد صدور صوته , تلاحظ أنه استخدم يده اليسار بطريقة مختلفة عن يده اليمين و هو يسمع اللحن الصادر من آلة العود , ذكرنا ثلاثة أشياء يقوم بفعلها العازف بنفس الوقت , تمّ تحليل و صدور هذه الأوامر الثلاث من العقل ,حيث يبعث رسالة تحوي فعل إلى الجزء المراد صدور هذا الفعل من خلاله,*سبحان الله الذي وهب للإنسان عقل*.

كما أنّ المبرمج خلال تحليل و كتابته لتطبيق ما يفكر بسبع أفكار على التوازي وهذا ما يدعى بتعدد المسالك .

تعدد المسالك هو تنفيذ عدة أفعال بنفس الوقت كما لاحظت منذ قليل كيف يستخدمها العازف و المبرمج.

يتكلم هذا الفصل عن كيفية عمل مسلك و كيفيّة إنشائه بطرق عدة و المزامنة بين المسالك من خلال منع التشارك و دعمه, و استخدام مناهج عالية المستوى لتنفيذ منهج ضمن مسلك منفصل من دون أن يكون هذا المنهج ضمن صف مسلك مستقل (ثانوي), لنبدأ في هذا الفصل الممتع و الصعب في آن واحد.

❖ كيفية عمل مسلك :

عندما تنشأ مسلك يتم تجسيمه داخل المساحة الذاكرية المحجوزة للتطبيق بمكدس يحوي مناهج "سلسلة تعليمات" المسلك , حيث يتم تنفيذ المسالك من حيثنا على التوازي و من حيث المعالج على التسلسل ,

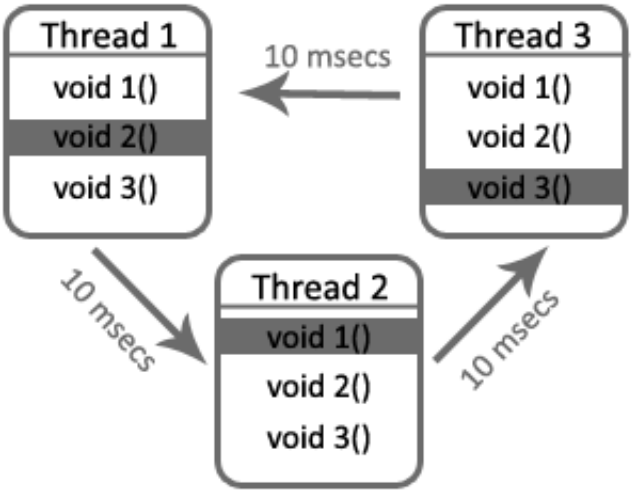
هنا لا نتكلم عن البرمجة التفرعية أو ما يدعى حاليا تغدد المهام .

لنفترض وجود ثلاث مسالك كل مسلك يحوي ثلاث مناهج و قد تمّ قدح تنفيذ هذه المسالك وكل مسلك بدوره قدح تنفيذ المناهج الثلاث التي يحويها , سوف تكون طريقة عمل هذه المسالك كالآتي :

أولا إنشاء ثلاث مكدسات في المساحة الذاكرية للتطبيق , حيث لكل مسلك مكدس يحوي مناهجه .

ثانيا يتم اختيار المسلك الذي سوف ينفذ أولا من قبل المعالج, عند إختياره لمسلك و تنفيذ منهج داخله سوف يتم الإنتقال إلى المسلك الآخر بعد مدة زمنية قصيرة تصل إلى 20 ملي- ثا تختلف هذه السرعة باختلاف سرعة المعالج ,و من ثمّ تنفيذ منهج داخله حيث تتم عملية الإنتقال بين المسالك قبل تنفيذ عمل المسلك بشكل كلي و في زمن يتجاوز 10 ملي ثانية "تزداد المدة و تنقص حسب سرعة المعالج" , عند العودة لمتابعة عمل منهج ضمن مسلك سيعاود عمل المنهج من آخر نقطة انتقال من هذا المسلك لمسلك آخر , نقطة الإنتقال هي النقطة التي تم وقوف عمل منهج موجود ضمن مسلك من أجل الإنتقال إلى مسلك آخر لتنفيذه.

انظر الشكل (5.1) يحوي ثلاث مكدسات تم تجسيمها للمسالك الثلاث و كل مكدس يحوي مناهج ثلاث يوجد خلفية مصمتة للمنهج الذي يعمل حاليا أي سلسلة التعليمات التي تنفذ بعد أن تمّ استدعائها من المنهج الرئيسي للمسلك (عادة يكون المنهج ()run) و الذي سوف يوضع عليه نقطة انتقال, مثلما تلاحظ يوجد سهم يشير إلى الإنتقال للمسلك الذي يلي المسلك الحالي ومدتها 10 ملي- ثا, أي كل 10 ملي- ثا سوف يتم انتقال التحكم إلى مسلك آخر لتكملة تنفيذ عمله .



الشكل 5.1

ملاحظة **: في الشكل (5.1) مناهج المسالك ()void 1 الخ.. نقصد فيها سلسلة التعليمات التي تمّ استدعائها ضمن المنهج الرئيسي للمسلك (()void run) .**

ملاحظة**: عند إقلاع تطبيق سوف يتم فتح مسلك بشكل تلقائي لهذا التطبيق ويدعى المسلك الرئيسي و كل المسالك عداه ضمن التطبيق تدعى مسالك ثانوية.**

❖ متى نحتاج استخدام مسلك :

نحتاج استخدام مسلك مستقل عند وجود معالجة بيانات في التطبيق تأخذ وقت أو في حالات الانتظار لالتقاط إشارة أو فعل ما لا نعلم متى يتم تنفيذه , في التطبيقات الشبكية مثلا يجب أن ننصت إلى منفذ محدد (مراقبته) من أجل معرفة العملاء الذين أرسلوا طلب إلى هذا المنفذ وهذا لا يتم إلا بوساطة مسلك مستقل و إلا إذا استخدمنا المسلك الرئيسي للتطبيق سوف يقف عمل التطبيق نهائيا حتى نوقف عملية المراقبة للمنفذ المراد .

مثال آخر في حال كنا نستخدم الرسم المباشر على صورة ما يتوجب علينا أيضا استخدام مسلك مستقل من أجل أن تتم عملية الرسم من دون أن يقف التطبيق أي يصبح في حالة تجميد حتى تنتهي عملية الرسم .

❖ كيفيّة إنشاء مسلك :

- **مثلما ذكرنا أنه يوجد نوعين من المسالك :**

مسلك رئيسيّ للتطبيق ككل , حيث يتم إنشاءه بشكل تلقائي عند تنفيذ التطبيق , والذي يدخل التطبيق في حلقة تكرار لا نهائيّة لاستقبال الأحداث وتنفيذ أوامر الكتلة الرئيسية من خرج و دخل و معالجة للبيانات , يتم استدعاء المسلك الرئيسيّ بوساطة المنهج "thread" التابع لكائن التطبيق الرئيسيّ و الذي يعيد مؤشره :

QApplication::thread()

QCoreApplication::thread()

نستطيع الوصول له بوساطة استدعاء المنهج الساكن "instance()" و الذي يعيد مؤشر لكائن التطبيق :

```
QApplication::instance()->thread()->terminate();
```

```
QCoreApplication::instance()->thread()->terminate();
```

النوع الثاني من المسالك هو مسلك ثانوي, مستقل عن المسلك الرئيسي للتطبيق , سنتكلم في فقرات هذا الفصل عن كيفية استخدامه بالتفصيل .

- **لنبدأ بتعلم كيفيّة يتم إنشاء مسلك ثانوي:**

يتوجب عليك أن تنشأ صف مشتق من الصف QThread , يحوي هذا الصف منهج محمي و افتراضي يدعى run و هو نقطة إقلاع المسلك , يجب عليك عند اشتقاق صف من QThread أن تعيد تحقيق هذا المنهج , لأنه سوف يتم استدعاء المنهج run() عند تنفيذ المسلك .

مثال : نريد إنشاء تطبيق سطر أوامر يطبع على الشاشة الوقت الحالي كل خمس ثوان بوساطة استخدام مسلك مستقل .

أنشأ تطبيق console , ثمّ أضف صف thread1 مشتق من الصف QThread , أضف داخل الملف

"thread1.h" ملف الترويسة الخاصة بالوقت "QTime" و الملف "iostream" , اكتب ضمن القسم المحمي (protected) المنهج run() , اذهب إلى ملف التحقيق "thread1.cpp" وحقق المنهج run() واكتب داخله الرمّاز التالي :

```
void thread1::run(){
```

```
forever{
    std::cout <<
QTime::currentTime().toString().toAscii().data()
    << std::endl;
    QThread::sleep(5);
}
}
```

استخدمنا المنهج الساكن sleep(int secs) من الصف QThread الذي يأخذ وسيط من نمط عدد صحيح لعدد الثوان التي سوف يكون فيها المسلك الحالي في حالة ثبات (نوم) , داخل الملف "main.cpp" اكتب الرمّاز التالي :

```
thread1* thr1 = new thread1();
thr1->start();
```

لا تنسى استدعاء الصف "thread1" أعلى الملف "main.cpp" .

نفذ التطبيق و اختبر النتيجة .

يوجد عدة مناهج لإخضاع المسلك في حالة ثبات لمدة زمنية وهم :

مدة الثبات تقدر بالملي ثانية `void msleep (unsigned long msecs )`

مدة الثبات تقدر بالثانية `void sleep (unsigned long secs )`

مدة الثبات تقدر بالمايكرو ثانية `void usleep (unsigned long usecs )`

المنهج الخاص بوضع أولوية التنفيذ لمسلك :

```
void QThread::setPriority ( Priority priority )
```

التعداد QThread::Priority :

إعطاء الأولوية للمسلك الذي تم إنشاءه حاليا في حال كان لا يوجد مسالك تعمل	QThread::IdlePriority
الأولوية للمسلك الذي تم إنشاءه حاليا أقل من LowPriority	QThread::LowestPriority
الأولوية للمسلك الذي تم إنشاءه حاليا أقل من NormalPriority	QThread::LowPriority
إعطاء الأولوية الإفتراضية من نظام التشغيل للمسلك الذي تم إنشاءه حاليا	QThread::NormalPriority
الأولوية للمسلك الذي تم إنشاءه حاليا أكثر من NormalPriority	QThread::HighPriority
الأولوية للمسلك الذي تم إنشاءه حاليا أكثر منHighPriority	QThread::HighestPriority
إعطاء الأولوية للمسلك الذي تم إنشاءه حاليا قدر الإمكان	QThread::TimeCriticalPriority
استخدام نفس أولوية المسلك الحالي, وهي الأولوية الإفتراضية	QThread::InheritPriority

❖ إنشاء مسلك بوساطة تحقيق الواجهة QRunnable :

نستطيع إنشاء مسلك من خلال تحقيق الصف QRunnable , يحوي هذا الصف على منهج محمي و افتراضي يدعى ()run والتي يكون نقطة البداية للمسلك بعد تنفيذه , في لغة الجافا (Java) يوجد أيضا الواجهة runnable عند تحقيقها ينتج كائن مسلك , لتنفيذ الصف المحقق للواجهة QRunnable في لغة Qt يجب أن نستدعي المنهج start(QRunnable*,int Priority=0) التابع للصف QThreadPool الخاص بإدارة مجموعة من المسالك :

```
void QThreadPool::start ( QRunnable * runnable, int priority = 0 )
```

مثال : لطباعة التاريخ الحالي على شاشة سطر الأوامر كل فترة زمنية محددة مع طباعة الوقت الحالي أيضا.

قم بفتح المثال السابق و أضف صف يدعى "runnable" مشتق من الصف QRunnable عرف المنهج ()run داخل القسم المحمي ضمن الملف "runnable.h" , اذهب إلى الملف "runnable.cpp" لتحقيق المنهج ()run اكتب الرمّاز التالي :

```
void runnable::run(){

   forever{

      std::cout << QDate::currentDate().day() << ":"

      << QDate::currentDate().month() << ":" <<
QDate::currentDate().year()

      << std::endl;

      for(int i=0;i < 900000000;i++){}

}

}
```

هنا حلقة التكرار for من اجل الانتظار لفترة زمنية وجيزة و من ثمّ متابعة عملية الطباعة .

ملاحظة : الفكرة سيئة أن تضع حلقة تكرار من أجل عملية الانتظار ولكن هنا وضعناها فقط من أجل تبسيط الفكرة .

داخل الكتلة الرئيسيّة main أضف الرمّاز التالي :

```
runnable* run = new runnable();

QThreadPool::globalInstance()->start(run);
```

المنهج الساكن ()globalInstance من الصف QThreadPool يعيد مؤشر لحدث الصف QThreadPool , استخدمنا المنهج start لتنفيذ الصف runnable المحقق للواجهة QRunnable .

سوف تلاحظ عند تنفيذ التطبيق أنه سوف يتم طباعة الوقت الحالي كل 5 ثوان و التاريخ الحالي كل فترة وجيزة قرابة 2 ثانية (حسب سرعة المعالج) .

لتحسين التطبيق : عندما تحاول أن تغلق شاشة سطر الأوامر فإنه سوف لن يسمح لك أن تغلقها , لذا يتوجب عليك أن تغلقها من خلال نظام التشغيل كعملية إنهاء التطبيق من خلال إدارة المهام في Win , ظهرت هذه المشكلة من عملية التكرار اللانهائي لطباعة التاريخ

داخل الصف runnable , لحل هذه المشكل يتوجب أن نخرج من الحلقة forever أولا و من ثمّ إغلاق التطبيق .

***الحل* : نصرح عن متحول من نمط عدد صحيح int داخل الصف runnable في القسم العام (public) ومن ثمّ إضافة رمّاز داخل الكتلة forever المضمنة في المنهج ()run و هو هذا :**

```
if(m_r == 1) break;
```

سوف يتم الخروج من الحلقة forever عندما تصبح قيمة المتحول m_r مساوية 1 .

الآن توجه إلى الكتلة main للتطبيق و أضف الرمّاز التالي قبل السطر البرمجي return a.exe() :

```
int i = 0;
forever{
std::cin >> i ;
if (i == 1){
   run->m_r = i;
   break;
}
}

thr1->terminate();
return 0;
```

عند إدخال القيمة 1 من خلال شاشة سطر الأوامر سوف تتوقف عملية طباعة التاريخ من خلال الخروج من الحلقة اللانهائية , الآن تستطيع إغلاق شاشة سطر الأوامر من دون مشاكل .

جرب أن تنفذ عملية طباعة الوقت و عملية طباعة التاريخ على شاشة سطر الأوامر من دون استخدام كائنات مسلك و اختبر النتيجة .

❖ مزامنة مسلك :

عملية المزامنة هي أن تمنع الوصول إلى مسلك أو منهج ضمنه في حال كان قيد الاستخدام و الانتظار حتى تنتهي عملية استخدامه من ثمّ الوصول إليه أو إلغاء الوصول إليه في حال كان قيد الاستخدام , للوصول نوعان وصول للقراءة و وصول للكتابة .

الصف QMutex الخاص بقفل كتلة من التعليمات ضمن مسلك , mutex تعني "mutual exclusion" أي "منع التشارك" , يوجد اثنين من المناهج يستطيعان تنفيذ عملية القفل ضمن الصف QMutex , الأول يدعى ()lock و الثاني ()tryLock :

void QMutex::lock ()

عند استدعاءه يتم قفل المسلك , في حال كان المسلك تمّ قفله من قبل طرف آخر , سوف يتم حظر القفل "أي يقف هذا الطرف ضمن الرتل في حالة انتظار" حتى يلغي الطرف الآخر فك القفل .

bool QMutex::tryLock ()

عند استدعاءه يتم قفل المسلك و يرجع القيمة true, في حال كان المسلك تمّ قفله من قبل طرف آخر سوف يحبط عملية محاولة قفله للمسلك و يعيد القيمة false .

bool QMutex::tryLock (int timeout)

مثيل المنهج السابق , لكون بوجود وسيط له من نمط int يأخذ المدة التي سوف ينتظرها ضمن رتل الإنتظار لمحاولة قفله للمسلك , المدة تكون بالملي – ثا .

و منهج خاص بفتح القفل يدعى ()unlock , عند تنفيذ عملية القفل سوف يتم قفل مثيل أو حدث الصف QMutex و بالتالي لن تستطيع أي كتلة تعليمات ضمن مسلك تستخدم نفس المثيل أو الحدث للصف QMutex أن تنفذ عملها حتى يتم فتح القفل .

لنرى مثال عن كيفية المزامنة بين المسالك :

سوف يقوم هذا التطبيق بترتيب تصاعديا من ثمّ تنازليا لمصفوفة مؤلفة من 10 خلايا من نمط عدد صحيح , سنقوم بفعل الترتيب التصاعدي و التنازلي بنفس الوقت , و المفعول به هو متحول للمصفوفة "عام" , إذا يوجد مصفوفة عامة و مسلكين و يوجد داخل كل منهما منهج خاص بالترتيب ,مسلك يتم استدعائه من أجل الترتيب التصاعدي و المسلك الآخر للترتيب التنازلي .

أولا سنقوم بناء التطبيق من دون استخدام المزامنة "القفل" ونختبر النتيجة ,من ثمّ نقوم بالتعديل عليه بحيث يستخدم المزامنة , أيضا نختبر النتيجة و الفرق بين النتيجتين .

أنشأ تطبيق سطر أوامر "Console" يدعى "mutex" , ثمّ أنشأ ملف رأسي يدعى "mutex.h" , اكتب الرمّاز التالي :

```
#ifndef MUTEX_H
#define MUTEX_H

#include <QThread>
#include <iostream>

class mutex1 : public QThread
{
   Q_OBJECT
public:
   explicit mutex1(QObject *parent = 0);
   ~mutex1();
protected :
   void run();
private:
   void order();
};

class mutex2 : public QThread
{
   Q_OBJECT
```

```
public:
   explicit mutex2(QObject *parent = 0);
   ~mutex2();
protected :
   void run();
private:
   void order();
};


#endif // MUTEX_H
```

عرفنا اثنان من الصفوف "mutex1" و "mutex2" تم اشتقاقهما من الصف "QThread" , صرحنا داخل كل واحد منهما المنهج "run" في القسم المحمي و المنهج "order" في القسم الخاص .

انشأ ملف يدعى "mutex.cpp" خاص بتحقيق هذان الصفان , اكتب الرمّاز التالي داخله :

```
#include "mutex.h"


int numbers[10] = {4,10,9,2,3,5,6,8,1,7};


mutex1::mutex1(QObject *parent) :
   QThread(parent)
{
}
mutex1::~mutex1(){wait();}
void mutex1::run(){
   order();
```

```
    exec();
}
void mutex1::order(){

    int val_num = 0;
    std::cout << "Numbers for order: " ;
    for(int i =0; i <= 9 ; i++)
        std::cout << numbers[i] << "  " ;
    std::cout << std::endl;

    for (int x=0; x<= 8; x++)
    for (int i=0; i<= 8; i++){

        if( numbers[i] > numbers[i+1] ){
            val_num = numbers[i] ;
            numbers[i] = numbers[i+1] ;
            numbers[i+1] = val_num ;
        }
    QThread::msleep(10);
    }
    std::cout << "Thread 1 :" << std::endl ;
    for (int i = 0; i <= 9;i++)
        std::cout << numbers[i] << "  " ;
    std::cout << std::endl << std::endl;
```

```
}
mutex2::mutex2(QObject *parent) :
    QThread(parent)
{
}

mutex2::~mutex2(){wait();}

void mutex2::run(){
    order();
    exec();
}

void mutex2::order(){
    int val_num = 0;

    std::cout << "Numbers for order: " ;
    for(int i =0; i <= 9 ; i++)
        std::cout << numbers[i] << "  " ;
    std::cout << std::endl;

    for (int x=0; x<= 8; x++)
    for (int i=0; i<= 8; i++){
```

```
        if( numbers[i] < numbers[i+1] ){
            val_num = numbers[i] ;
            numbers[i] = numbers[i+1] ;
            numbers[i+1] = val_num ;
        }
    QThread::msleep(10);
    }
    std::cout << "Thread 2 :" << std::endl ;
    for (int i = 0; i <= 9;i++)
        std::cout << numbers[i] << "  " ;
    std::cout << std::endl << std::endl;
}
```

صرحنا عن مصفوفة عامة ضمن الملف "mutex.cpp" تدعى "numbers" تحوي 10 خانات يوجد فيها أرقام مختلفة , حققنا الصفان المذكوران آنفا و حققنا المنهج "order" في كليهما و كتبنا داخلهما الرمّاز الخاص بترتيب الأرقام داخل المصفوفة و من ثمّ طباعتها على شاشة سطر الأوامر , طبعا يختلف المنهج "order" في الصف الأول عن المنهج "order" في الصف الثاني فقط بعملية الترتيب من أجل الترتيب التصاعدي و التنازلي, حقق المنهج "run" نقطة إقلاع المسلك و اكتب داخله رمّاز لاستدعاء المنهج "order" :

```
    order();
    exec();
```

ثمّ اذهب إلى الملف "main.cpp" و ضمّن ملف الترويسة "mutex.h" و اكتب ضمن الكتلة الرئيسية للتطبيق "main" الرمّاز الخاص بإنشاء حدث لكلى الصفين و أطلق تنفيذهما :

```
    mutex1* mThread1 = new mutex1();
    mutex2* mThread2 = new mutex2();
```

```
mThread1->start();


mThread2->start();
```

نفذ التطبيق و اختبر النتيجة ....

طبعا سوف تلاحظ وجود عشوائية في طباعة الأرقام و ترتيبها , و أيضا عند كل تنفيذ غالبا ما سوف تكون النتيجة مختلفة .

بما أننا استخدمنا نفس كتلة البيانات للمعالجة أي نفس المصفوفة لكلا المسلكين و أطلقنا تنفيذ كلا المسلكين بنفس الوقت , المعالج يقوم بتنفيذ عمل المسالك على التسلسل مثلما ذكرنا و التنقل بينها ليكمل عمل المسلك كل فترة قصيرة جدا و يقوم بتكرار هذه العملية حتى يتم تنفيذ كامل العمل الخاص بالمسلك من ثمّ تحرير موارد المسلك , فعلى هذا الأساس سوف تتم العملية كالآتي :

أولا اختار المعالج مسلك للتنفيذ ,ثمّ بدء تنفيذ المؤقت الخاص بهذا المسلك حيث بعد انتهاء المدة الزمنية المسموح بها أن يبقى بعمل المسلك ينتقل إلى مسلك آخر, ضمن المنهج الرئيسي "نقطة الإقلاع" للمسلك و الذي هو المنهج "run" تمّ استدعاء المنهج الخاص بالترتيب "order", المنهج "order" تم التصريح عن متحول من نمط عدد صحيح يدعى "val_num" , طبع أيضا على شاشة سطر الأوامر العبارة التالية " Numbers for order:" , بدء تنفيذ عملية التكرار لطباعة قيم خانات المصفوفة على الشاشة , طبع قيمة أول خانة ضمن المصفوفة و هي "4" , انتهت المدة المعطاة للمؤقت, تمّ انتقال التحكم إلى المسلك الآخر لبدء أو تتمة تنفيذ عمله, و هكذا أيضا في المسلك الحالي سوف يبدأ بتنفيذ عمل المؤقت ثم تنفيذ محتوى المسلك من أفعال حتى تنتهي مدة المؤقت و ينتقل إلى المسلك الآخر الخ...

على هذه الحال عندما تكون حلقة التكرار الخاصة بترتيب الأرقام تصاعديا ضمن المصفوفة قيد التنفيذ و انتهى الزمن المحدد للمؤقت الخاص بالمدة الزمنية لعمل المسلك قبل أن تتم عملية ترتيب الأرقام ضمن خانات المصفوفة بشكل كامل , ثمّ انتقل التحكم إلى المسلك الآخر و التي تكون نقطة التوقف فيه هي حلقة التكرار الخاصة بالترتيب التنازلي للأرقام الموجودة داخل نفس المصفوفة سوف تتابع هذه الحلقة عملها من عند عملية التكرار السادسة فرضا "ممكن أن تكون الخامسة أو غيرها" لذا سوف لن تأبه لأي تغيير حاصل على خانات المصفوفة الموجودة ضمن الخانات الأصغر للخانة السادسة "أي ما وراء الخانة السادسة الخانة الخامسة و ما تحتها .."

لذا نلاحظ وجود هذه العشوائيّة بخرج التطبيق , و عدم إعطائه للنتيجة المرجوة و التي هي ترتيب الأرقام الموجودة ضمن خانات المصفوفة تصاعديا و من ثمّ تنازليا .

الحل لهذه المشكلة أن نستخدم منع التشارك "المزامنة" لسلسة التعليمات التي لا نود أن يشارك تنفيذها مسلك آخر حتى تنتهي هذه السلسلة من عملها .

لنبدأ بحل المشكلة: أضف "ضمّن" ملف الترويسة "QMutex" ضمن الملف "mutex.h" ثمّ اذهب إلى الملف "mutex.cpp" و صرّح عن مثيل عام للصف "QMutex" يدعى "m_mutex" , اكتب داخل الصف "mutex1" ضمن المنهج "order" في بدايته الرمّاز التالي :

```
m_mutex.lock();
```

و في نهاية المنهج "order" :

```
m_mutex.unlock();
```

اكتب هذين السطرين في كلا المنهجين ضمن الصفين .

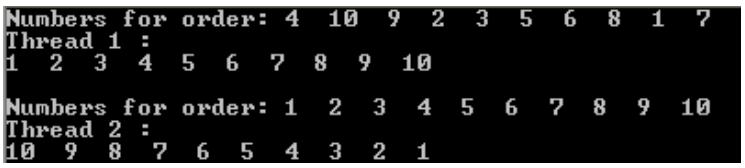
استخدمنا القفل لسلسة التعليمات الموجودة ضمن المنهج "order" باستخدام السطر البرمجي :

```
m_mutex.lock();
```

و الذي يعني أنه سوف لن يسمح بتنفيذ أي من التعليمات المستخدمة لمنع التشارك بوساطة المثل "m_mutex" حتى يتم فك قفل "unlock" المثيل "m_mutex" , و سوف يقوم المسلك الآخر الذي يستخدم المثيل "m_mutex" بعملية الإنتظار ضمن الرتل الخاص بالمسالك حتى يأتي دوره بالتنفيذ بعد فك قفل المثيل .

استخدمنا المثيل "m_mutex" في كلا الصفين لمنع تشارك سلسة تعليمات مع سلسلة أخرى تستخدم المثيل نفسه.

الآن نفذ التطبيق انظر الشكل (5.2) و اختبر النتيجة و الفرق بين هذه النتيجة و النتيجة السابقة .



الشكل 5.2

نستطيع أن نعوض بدلا من كتابة السطرين السابقين :

```
m_mutex.lock();
```

m_mutex.unlock();

بالرمّاز التالي :

m_locker(&m_mutex); QMutexLocker

سوف يتم قفل سلسلة "كتلة" التعليمات و فك قفلها بشكل تلقائي عند انتهاء عمل هذه السلسلة .

***ملاحظة* : يكتب هذا الرمّاز في بداية سلسلة التعليمات المراد منع تشاركها .**

❖ المزامنة الشرطية :

إذا كنا نريد أن يبدأ تنفيذ عمل المسلك الثاني (mutex2) "الخاص بترتيب الأعداد تنازليا" أولا من ثمّ الإنتقال لتنفيذ عمل المسلك الأول (mutex1) "الخاص بترتيب الأعداد تصاعديا" ما هو الحل :

يتوجب علينا استخدام الصف "QWaitCondition" الخاص بوضع المسلك في حالة انتظار حتى يتم إيقاظه "إعطاء التحكم له" , فهو يستخدم من أجل المزامنة بين المسالك بشكل شرطي .

لنبدأ بتنفيذ الحل :

افتح المشروع السابق و أضف ملف الترويسة "QWaitCondition" داخل الملف "mutex.h" , ثمّ صرح عن متحول عام ضمن الملف "mutex.cpp" يدعى "m_wait" من النمط "QWaitCondition" ,

المنهج "order" ضمن "mutex1" عدل السطر الأول من رمّازه كالآتي :

أولا احذف السطر :

QMutexLocker m_locker(&m_mutex);

ثانيا فعل منع التشارك و من ثمّ ضعه في حالة انتظار بوساطة الرمّاز الواضح أمامك :

```
m_mutex.lock();

m_wait.wait(&m_mutex);
```

لن يستيقظ "يعاد التحكم لهذا المسلك" حتى يتم استدعاء المنهج ()wakeOne أو ()wakeAll للمثيل الصف QWaitCondition , لا تنسى أن تلغي تفعيل "فك قفل" منع التشارك بإضافة السطر البرمجي التالي آخر كتلة المنهج "order" :

```
m_mutex.unlock();
```

الآن يجب أن نوقظ المسلك الأول بعد الانتهاء من عمل المسلك "mutex2" , اذهب إلى المنهج "order" ضمن الصف "mutex2" , احذف السطر البرمجي :

```
QMutexLocker m_locker(&m_mutex);
```

أضف التعليمات الخاصة بمنع التشارك في بداية كتلة المنهج:

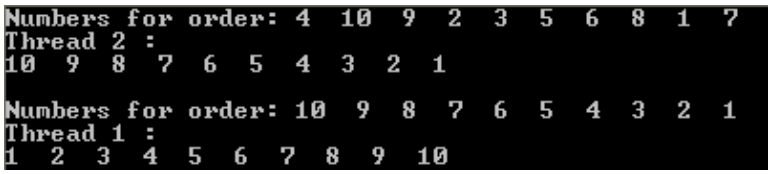
```
m_mutex.lock();
```

آخر سطرين ضمن المنهج أضف الرمّاز التالي :

```
m_wait.wakeAll();

m_mutex.unlock();
```

لإيقاظ "إعادة التحكم" للمسلك الأول "mutex1" استخدمنا السطر البرمجي الأول من هذين السطرين السابقين.

نفذ التطبيق و اختبر النتيجة , سوف ترى أنه تمّ تنفيذ عمل المسلك الثاني من ثمّ نفذ عمل المسلك الأول و هو المطلوب , انظر الشكل (5.3) :



الشكل 5.3

❖ حظر الوصول إلى مسلك من أجل القراءة \ الكتابة :

إذا كنا نريد أن نحظر كتلة من البيانات ضمن مسلك من الوصول للقراءة أو الكتابة حتى إتمام عملها, فماذا يتوجب علينا فعله ؟

ضمن مكتبات Qt الخاصة بالمسالك يوجد صف خاص بحل هذه المشكلة و يدعى "QReadWriteLock" , تفيد هذه التقنيّة من أجل منع الوصول للكتابة على كتلة بيانات كملف قرصي فرضا و نحن نكتب عليه حاليا أو نقرأ منه حتى يتم إكمال عملية الكتابة أو القراءة لهذا الملف من ثمّ الكتابة عليه .

لنرى البنية العامة لاستخدام الصف "QReadWriteLock" :

منع الوصول للكتابة حتى تتم إكمال مهمة القراءة "فك قفل منع الوصول" :

```
QReadWriteLock  m_ReadWriteLock;

type of data readFile(){

m_ ReadWriteLock.lockForRead();

//{read data from file…}

m_ ReadWriteLock.unlock();

return data;

}
```

منع الوصول للكتابة و القراءة حتى تتم إكمال مهمة الكتابة "فك قفل منع الوصول" :

```
void writeFile(){

m_ ReadWriteLock.lockForRead();

//{write data to file…}

m_ ReadWriteLock.unlock();

}
```

نستطيع بوساطة الصف "QWriteLocker" و الصف "QReadLocker" فك الحظر "القفل" بشكل تلقائي عند نهاية كتلة التعليمات الخاص بالقراءة أو الكتابة , سوف نعدل الرمّاز السابق لنرى كيف فك القفل بشكل تلقائي :

```
QReadWriteLock  m_ReadWriteLock;
```

```
type of data readFile(){
QReadLocker m_Rlocker(&m_ ReadWriteLock);
//{read data from file…}
return data;
}

void writeFile(){
QWriteLocker m_Wlocker(&m_ ReadWriteLock);
//{write data to file…}
}
```

❖ استخدام الإشارة للوصول المتعدد إلى مسلك " دعم التشارك ":

عندما تكلمنا عن منع التشارك ذكرنا أنه يمكن الوصول إلى مسلك فقط من قبل طرف واحد حتى يتم الانتهاء من عمله "فك قفل منع التشارك" , إذا كنا نريد الوصول من قبل أكثر من طرف إلى مسلك مع تحديد العدد الأقصى للأطراف المسموح بها للوصول إليه في نفس الوقت , ماذا يتوجب علينا فعله ؟

يوجد ضمن مكتبات Qt الصف "QSemaphore" الخاص بالوصول المتعدد مع التحديد للعدد الأقصى للأطراف المسموح بها من اجل الوصول إليه .

أهم مناهجه :

void QSemaphore::acquire (int n = 1)

المنهج الخاص بحجز "اكتساب" إشارة جديدة للمسلك الحالي .

void QSemaphore::release (int n = 1)

المنهج الخاص بفك حجز "إطلاق" إشارة من المسلك الحالي .

int QSemaphore::available () const

يعيد عدد الإشارات المسموح حجزها "المتبقية" ضمن هذا المسلك .

بالنسبة للقالب العام لاستخدامه :

```
QSemaphore  m_semaphore(3);

class thread1 {

run{

 m_ semaphore.acquire(1);

//{todo whatever u want…}

}

}

class thread2{

run{

 m_ semaphore.release(2);

//{todo whatever u want…}

}

}
```

صرحنا عن مثيل للصف "QSemaphore" يدعى "m_semaphore" لوسيط بنائه وضعنا القيمة 3 أي سوف يتم تهيئة ثلاث خانات من الإشارات قابلة للحجز , بمعنى آخر سوف يكون العدد الأقصى للأطراف التي تستطيع الوصول إلى هذا المسلك ثلاث أطراف , ضمن المسلك الأول داخل كتلة الإقلاع "run" سوف يتم حجز 1 إشارة (القيمة الإفتراضية) عند كل استدعاء لهذا المسلك إلى أن يتم وصول ثلاثة أطراف فلن يستطيع أن يصل طرف رابع للمسلك و سوف يبقى برتل الإنتظار إلى أن يتم إطلاق إشارة محجوزة من قبل طرف من الأطراف المستخدمة لهذا المسلك في الوقت الآني .

إذا مثلما تلاحظ يستطيع ثلاث أطراف بالوصول إلى هذا المسلك في نفس الوقت .

المسلك الثاني عند تنفيذه يطلق "يهيئ" خانتان من الإشارة من أجل الحجز , عند كل استعداء لهذا المسلك سوف يقوم بجمع عدد الخانات القابلة للحجز مع القيمة الممررة لوسيط المنهج "release(int)" , بالتالي في الاستدعاء الثاني له سوف تكون عدد الخانات المهيئة 4 و هكذا ...

❖ الدخل و الخرج إلى و من مسلك :

استخدام مسلك من دون وجود دخل إليه أو خرج منه لا يفيد المبرمج , لأننا أساسا نستخدم المسلك من أجل معالجة بيانات معطاة له بشكل مستقل عن مسلك التطبيق الرئيسي من ثمّ بعد انتهائه من معالجتها يعطي خرج البيانات التي عالجها لنعرضها حسب ماهيتها, مثال إذا كنا نريد أن نعالج صورة ضمن مسلك و من ثمّ بعد الانتهاء من معالجتها عرضها على كائن widget , هذه العملية تحتاج لأن ندخل الصورة إلى المسلك "دخل" من ثمّ بعد إنهاء هذا المسلك معالجة الصورة يتوجب عليه أن يعطينا الصورة "خرج" لعرضها على كائن widget , لنرى كيف تتم عملية الدخل و الخرج لمسالك .

لنرجع إلى المثال السابق حيث أننا سوف نعدله ليصبح كالآتي :

تكون مصفوفة الأرقام التي يتوجب على المسلك ترتيبها هي دخل إليه من الكتلة التي يوجد فيه تنفيذه أي الكتلة الرئيسية للتطبيق "main" , عند التنفيذ يتم إيقاف عمل المسلك الأول "mutex1" بشكل مؤقت باستخدام الكائن "QWaitCondition" , ثمّ يقوم المسلك الثاني بطباعة الأرقام المضمنة داخل المصفوفة المدخلة , و بعدها يرتب الأرقام الموجودة داخل المصفوفة بشكل تنازلي , بعد الانتهاء من عملية الترتيب يقدح حدث داخلي "Signal" يمرر لوسيطه خرج الترتيب , من ثمّ تنفيذ المقبس المرتبط بالحدث ليعرض الأرقام المرتبة على شاشة سطر الأوامر "المقبس هنا داخل صف المسلك الأول (mutex1) ", بعدها ينتقل التحكم إلى المسلك الأول لتنفيذ عملية الترتيب التصاعدي و طباعة النتيجة .

لبدأ بتعديل المثال السابق , افتح المثال السابق و اذهب إلى الملف الرأسي "mutex.h" , ضمن الصف "mutex2" أضف في القسم العام تصريح المنهج الخاص بعملية دخل المصفوفة :

```
Public :

  void setNumbers(int[],int);
```

وسيطه الأول من نمط مصفوفة أعداد صحيحة ,أما الثاني فهو عدد صحيح من أجل الإعطاء له حجم المصفوفة المدخلة.

عرّف حدث داخلي حيث يكون بوابة الخرج للمصفوفة التي قد تم ترتيب الأرقام داخلها تنازليا من خلال تمرير هذه المصفوفة لوسيطه بحيث يستطيع المقبس المرتبط به أن يأخذ قيمة هذه المصفوفة , يقدح هذا الحدث بعد الانتهاء من عملية الترتيب الأرقام داخل المصفوفة :

```
signals:

   void finishOrder(int*);
```

نصرح داخل صف المسلك الأول "mutex1" عن المقبس الخاص باستقبال المصفوفة من المسلك الثاني "mutex2" :

```
public slots:

   void printNumbers(int*);
```

داخل ملف التحقيق "mutex.cpp" , انتقل إلى الصف "mutex2" و احذف هذه السطور البرمجية الموجودة داخل المنهج "order" :

```
std::cout << "Thread 2 :" << std::endl ;

for (int i = 0; i <= m_size -1;i++)

     std::cout << numbers[i] << "  " ;

std::cout << std::endl << std::endl;
```

أضف مكانها السطر البرمجي التالي :

```
   emit finishOrder(numbers);
```

يعمل هذا الرمّاز على قدح الحدث "finishOrder(int*)" , مثلما تلاحظ أن موقعه بعد سلسة التعليمات الخاصة بالترتيب , و بالتالي سوف يتم قدح الحدث بعد الانتهاء من هذه العملية , عند قدحه سوف ينفذ المقبس "المستقبل" المتصل به .

احذف من المنهج "order" السطر الخاص بإيقاظ المسلك الأول :

```
m_wait.wakeAll();
```

في حال أبقينا عملية الإيقاظ للمسلك الأول هنا سوف يقوم بتنفيذ عمل المسلك الأول "ترتيب الأرقام تصاعديا و طباعتها" بنفس الوقت الذي يعمل فيه المستقبل للحدث "finishOrder(int*)" على طباعة الأرقام المرتبة تنازليا .

احذف متحول المصفوفة العام:

```
int numbers[10] = {4,10,9,2,3,5,6,8,1,7};
```

و صرح عوضا عنه المتحول التالي :

```
int* numbers;
```

صرح عن متحول عام ضمن الملف "mutex.cpp" يدعى "m_size" من نمط عدد صحيح ليخزن فيه حجم المصفوفة المدخلة .

الآن حقق المنهج "setNumbers(int[],int)" الخاص بعملية دخل المصفوفة :

```
 nums[], int size){ void mutex2::setNumbers(int

   numbers = nums ;

   m_size = size ;

   start();

}
```

المنهج "start()" لبدأ تنفيذ المسلك الحالي عند استدعاء المنهج "setNumbers" من طرف خارجي .

ضمن الصف "mutex1" حقق المقبس "printNumbers(int*)" كالآتي :

```
void mutex1::printNumbers(int* nums){

   std::cout << "Thread1 'SLOT'> print numbers :" ;

   for(int i=0;i <= m_size -1 ;i++)

      std::cout << nums[i] << " ";

   std::cout << std::endl << std::endl;

   m_wait.wakeAll();
```

```
}
```

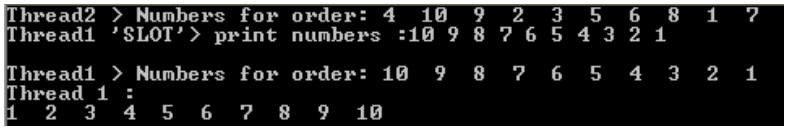
تمّ طباعة أرقام المصفوفة الممررة لوسيط المقبس من الحدث "finishOrder(int*)" , من ثمّ أيقظنا المسلك "mutex1" من ثباته "عملية انتظاره" .

داخل الكتلة الرئيسّة للتطبيق "main" اكتب الرمّاز التالي :

```
int numbers[10] = {4,10,9,2,3,5,6,8,1,7};

mutex1* mThread1 = new mutex1();

mutex2* mThread2 = new mutex2();

QObject::connect(mThread2,SIGNAL(finishOrder(int*)),mThread1,

SLOT(printNumbers(int*)));

mThread1->start();

mThread2->setNumbers(numbers,(sizeof(numbers)/4));
```

صرحنا مصفوفة من عشر 10 خانات تدعى "numbers" نمطها عدد صحيح , وضعنا ضمن خاناتها 10 أرقام عشوائيّة , صرحنا عن حدث للمسلك "mutex1" يدعى "mThread1" و حدث آخر للمسلك "mutex2" يدعى "mThread2" , قمنا بعملية اتصال بين الحدث الداخلي "finishOrder(int*)" التابع للمسلك "mThread2" الخاص بتنفيذ عملية إعطاء خرج الترتيب التي قام بها المسلك mThread2 إلى المسلك mThread1 بوساطة استقبال الخرج من خلال المقبس "printNumbers(int*)" التابع للمسلك "mThread1" , حيث ستنفذ هذه العملية عند قدح تنفيذ الحدث "finishOrder(int*)" , أطلق تنفيذ المسلك الأول من خلال المنهج "start()" , المسلك الثاني "mThread2" مررنا لوسطاء منهجه "setNumbers(int[],int)" المصفوفة التي هيئناها مسبقا ليتم ترتيبها و حجم هذه المصفوفة , هذا المنهج سوف يعمل على تنفيذ المسلك "mThread2" من خلال استدعاء المنهج "start()" الموجود داخله .

الآن نفذ التطبيق و اختبر النتيجة , انظر الشكل (5.4) :



الشكل 5.4

❖ تسجيل نمط جديد :

إذا كنا نريد استخدام صف خاص بنا كنمط لوسيط الحدث و المستقبل ماذا يجب علينا فعله ؟

يتوجب علينا تسجيل هذا النمط ليضاف إلى أنماط QVariant , يوجد المنهج الخاص بعملية التسجيل ضمن ملف الترويسة "QMetaType" و المنهج هو :

qRegisterMetaType<myType>("myType")

أيضا يتوجب علينا أن نكتب الماكرو التالي داخل ملف ترويسة النمط الخاص بنا :

Q_DECLARE_METATYPE(myType)

- **لنرى الشكل العام :**

نفترض أنه يوجد لدينا صف يدعى "myNewType" نريد استخدامه كوسيط لحدث و مقبس , يجب علينا أن نضيف في ملف تعريفه الماكرو المذكور آنفا :

Class myNewType : Q…{

Q_OBJECT

public :

myNewType(…);

~myNewType();

……

}

```
Q_DECLARE_METATYPE(myNewType)
```

ثمّ قبل استخدام هذا النمط يجب علينا أن نسجله كنمط جديد كالأتي :

```
qRegisterMetaType<myNewType>("myNewType");

QObject::connect(myObj1,SIGNAL(void(myNewType)),myObj2,

SLOT(void(myNewType)));
```

❖ مسالك عالية المستوى :

ضمن Qt يوجد فضاء تسمية يدعى "QtConcurrent" لكتابة رمّاز مسالك متعددة من دون استخدام الطريقة البدائيّة المنخفضة المستوى لإنشاء مسلك "أي اشتقاق صف من صف مسلك QThread" , تستطيع إنشاء منهج و تنفيذ عمل هذا المنهج ضمن مسلك منفصل و تمرير دخل له و أخذ الخرج منه من دون استخدام صف مشتق من صف مسلك .

نحتاج من أجل إعادة قيمة منهج في مسلك منفصل أن نستخدم الصف "QFuture" الخاص بإعادة النتيجة من هذا المنهج , نستخدم هذا الصف من أجل العمليات الغير متزامنة بين المسالك , أما الصف "QFutureSynchronizer" مثيل الصف السابق لكن الاختلاف هو أنه يستخدم من أجل العمليات المتزامنة بين المسالك .

لنرى كيفية تنفيذ منهج في مسلك منفصل بطريقة ممتعة .

سوف نبني تطبيق سطر أوامر مهمته طباعة أرقام تدريجية و بسرعة كبيرة جدا , عند تكبير شاشة سطر الأوامر تبدو كشاشة توقف رقمية , عندما ندخل الرقم و احد من ثمّ نضغط مفتاح "Enter" سوف تتوقف عملية الطباعة.

أنشأ تطبيق سطر أوامر يدعى "conCurrent" , ضمن ملف الترويسة "QFuture" و المكتبة "QtCore" التي تحوي فضاء التسمية "QtConcurrent" :

```
#include <QFuture>

#include <QtCore>

#include <iostream>
```

صرح عن متحولين من نمط عدد صحيح ,متحول من أجل الأرقام المتدرجة التي سوف يتم طباعتها, أما الثاني من أجل الرقم الذي سوف ندخل من شاشة سطر الأوامر :

```
int i = 0;

int x = 0;
```

عرّف منهج يدعى "printDigits" من ثمّ حققه, حيث سوف نضع هذه المنهج ضمن مسلك منفصل , مهمة هذا المنهج طباعة الأرقام على شاشة سطر الأوامر:

```
void printDigits();
```

```
void printDigits(){

forever {i++;

      std::cout << i;

         if(x==1)break;

     }

}
```

رمّاز هذا المنهج بسيط للغاية .

في الكتلة "main" أضف الرمّاز الخاص بتنفيذ المنهج "()printDigits" ضمن مسلك منفصل :

```
QFuture<void> future = QtConcurrent::run(printDigits);
```

صرحنا عن مثيل للصف "QFuture" يدعى "future" حيث سوف يتم تخزين القيمة المرجعة من المنهج "run" التابع للفضاء "QtConcurrent" الخاص بتنفيذ المنهج الممرر للمنهج "run" في مسلك منفصل , و بما أن المنهج "()printDigits" لا يعيد قيمة و ضعنا "void" لقالب المثيل "future" , في حال كان المنهج يعيد قيمة على فرض "int" يجب علينا أن نضع بدل "void" النمط "int" .

أضف السطر البرمجي الأخير في التطبيق الحالي و الذي يكون خاص بإدخال المحرف من اجل الخروج من الحلقة اللانهائيّة :

```
std::cin >> x;
```

نفذ التطبيق و اختبر النتيجة .

جرب أن تضع الرمّاز الموجود داخل المنهج "printDigits()" قبل السطر البرمجي الخاص بإدخال محرف و اختبر إذا كان بالإمكان إدخال محرف من لوحة المفاتيح , بالطبع لن تستطيع حتى يتم الخروج من حلقة التكرار اللانهائية لأنه يتم عمل هذه الحلقة ضمن المسلك الرئيسي للتطبيق , أما عندما استخدمنا المنهج

```
QtConcurrent::run(printDigits);
```

نستطيع إدخال محرف لأن حلقة التكرار اللانهائيّة تعمل في مسلك مستقل "ثانوي" عن المسلك الرئيسي للتطبيق.

❖ إدخال قيمة و إعادة نتيجة إلى و من منهج تم تنفيذه في مسلك عالي المستوى :

تنفيذ عملية إدخال بيانات إلى منهج مسلك عالي المستوى سهل جدا , لنرى ذلك :

```
int sum(int i, int j);

main{

QFuture<int> future = QtConcurrent::run(sum,5,10);

std::cout << future.result();

}
```

المنهج "result" يعيد القيمة المعادة من المنهج المحدد "sum".

• التحكم بكائن واجهة مستخدم باستخدام مسلك عالي المستوى :

نستطيع ذلك بوساطة تمرير الوسيط لمنهج المسلك كمرجع "مؤشر" لصنف الكائن المراد التحكم به , مثال بسيط لإضافة عنصر لكائن قائمة :

```
void addItem(QListWidget* listW);

void addItem(QListWidget* listW){

   listW->addItem("Hasan Al-Morhej");

}
```

ثمّ ننفذ المسلك كالآتي :

```
QFuture<void>  future = QtConcurrent::run(addItem, ui->listWidget);
```

❖ مراقبة التغيرات التي تطرأ على مسلك عالي المستوى :

إذا كنا نريد معرفة أن المسلك قد أكمل مهمته أو أنه قيد التوقف أو أعاد نتيجة الخ... ماذا نفعل ؟

تم إنشاء صف في Qt يدعى "QFutureWatcher" يحوي مجموعة من المقابس و الأحداث الداخلية المفيدة من أجل عملية مراقبة التغيرات التي تطرأ على المسلك المراقب , انظر الرمّاز :

```
   myObj* obj= new myObj();

   QFuture<void> future = QtConcurrent::run(void1);

   QFutureWatcher<void> watcher;

   watcher.setFuture(future);


QObject::connect(&watcher,SIGNAL(started()),obj,SLOT(printStarted));


QObject::connect(&watcher,SIGNAL(finished()),obj,SLOT(printFinished)
);
```

أهم الأحداث التي يحويها الصف "QFutureWatcher" :

```
void  canceled ()
```

void finished ()

void paused ()

void resumed ()

void started ()

❖ خلاصة الفصل :

رأينا في هذا الفصل كيفيّة عمل مسلك وفائدته , و التعامل مع مسالك منخفضة المستوى أي اشتقاق صف من صف مسلك و كيفية تحقيق الصف "QRunnable" من ثمّ تنفيذه بوساطة "QThreadPool" , و المزامنة بين المسالك و قفل الوصول للقراءة و الكتابة , أيضا منع التشارك و دعمه , و كيف تتم عملية الدخل و الخرج إلى و من مسلك , و استخدام مسالك عالية المستوى وتمرير دخل لها و إعادة الخرج , ستجد في هذا الفصل كل مل يلزمك لاستخدام المسالك ضمن Qt.

لننتقل إلى الفصل التالي و الذي يتكلم عن برمجة التطبيقات الشبكيّة .

الفصل السادس

برمجة التطبيقات الشبكيّة

Network Applications Programming

❖ المقدمة Intro:

كان ضربا من ضروب الخيال أن نكون لشخص ما أينما كان حضورا , و قد تمّ تحقيق ذلك في عصرنا الحالي من خلال استخدام وسائل الإتصال المتعددة السمعيّة منها و البصريّة, مثيل ذلك فينا فجسمنا يحوي على ما يدعى بالسّيّالة العصبيّة التي مهمتها نقل الإحساس من عضو ما إلى المخ, تدعى هذه العمليّة بعملية نقل "اتصال", في عالم البرمجيات كعادته بتجسيم المحسوس الطبيعي إلى عناصر رقميّة , فمثّل السّيّالة العصبيّة بالتطبيقات الشبكيّة , و التي أصبحت حاليا عمودا فقريا لتكنلوجيا المعلومات ,و المثال الأفضل هو الشبكة العنكبوتيّة التي لا نستطيع ممارسة كامل حياتنا اليومية من دون وجودها , فهي تمثّل وسيلة للحصول على آخر الأخبار و المعلومات و وسيلة إتصال بين الأفراد و الجماعات الخ..

لذا قررت أن أألف هذا الفصل الذي يتمحور حول كيفية برمجة التطبيقات الشبكية بوساطة Qt .

سوف تجد في هذا الفصل كيفيّة برمجة أربع أنواع من أهم البروتوكولات :

1- HTTP التي تتلخص فكرته بإرسال طلب إلى بريمج ويب و استقبال الإستجابة منه .

2- FTP يستخدم من أجل نقل الملفات بين العميل و المخدم .

3- TCP البروتوكول الأكثر استخداما لما فيه من وثوثقيّة, يعد من البروتوكولات الموجهة .

4- UDP البروتوكول الأكثر سرعة في نقل البيانات و لكنه غير موثوق و غير موجه , خاص بتطبيقات البث الإذاعي "Braodcast" .

البروتوكول هو عبارة عن نظام لمجموعة من القوانين لكيفية عمل إرسال و استقبال البيانات ضمن الشبكة و يتم ذلك بوجود إتفاق بين الطرفين "المرسل و المستقبل" على بروتوكول محدد .

لنبدأ بالتفصيل شرحا .

❖ بروتوكول HTTP :

وهو اختصار "Hyper Text Transfer Protocol" , يعمل على نقل طلب العميل "Request" إلى مخدم الويب "الشبكة" , و بعد معالجة المخدم للطلب يتم إرسال ناتج معالجة الطلب "Response" إلى العميل , تستطيع نقل جميع أنواع البيانات بوساطة بروتوكول "HTTP" .

أي مستعرض ويب يستخدم هذه الآليّة ليستطيع عرض الصفحة التي تمّ طلبها .

لنرى بوساطة Qt كيفيّة إرسال طلب إلى المخدم و استقبال الاستجابة منه .

الصفوف الخاصة بالتعامل مع بروتوكول "http" في "Qt" :

الصف "QNetworkAccessManager" خاص بإرسال طلب و استقبال إستجابته من المخدم .

الصف "QNetworkResquest" خاص باحتواء ترويسة الطلب الذي سوف يرسل , ترويسة الطلب تحوي العنوان و إصدار البروتوكول و نمط البيانات الخ...

الصف "QNetworkReply" خاص باحتواء ترويسة الاستجابة على الطلب و بيانات تلك الاستجابة .

مثال لإرسال طلب و استقبال استجابته :

أنشأ تطبيق سطر أوامر و سمه "http" , اذهب إلى ملف المشروع "http.pro" واكتب السطر البرمجي التالي :

```
QT    +=    network
```

أضفنا هنا المكتبة التي تحوي الصفوف الخاصة بالتطبيقات الشبكيّة و هي "network", حيث من دون تضمين هذه المكتبة داخل مشروعنا لا نستطيع التعامل مع أي صف من صفوف التي تدعم تطبيقات الشبكة ,سوف يعمل Qt على تضمين هذه المكتبة عند تجميع التطبيق لتعمل البرامج "سلسلة التعليمات" الداعمة للعمل الشبكيّ ضمن تطبيقنا الحالي .

ثمّ انشأ صف يدعى "networkAM" مشتق من الصف "QObject", داخل ملف ترويسة الصف "networkAM" :

ضمّن ملفات الترويسة التي سوف نستخدمها ضمن هذا الصف :

```
#include <QtNetwork>

#include <QUrl>

#include <iostream>
```

ثمّ عرّف التالي :

في القسم العام :

```
public :

void sendRequest(QString);
```

هذا المنهج خاص بإرسال الطلب , وسيطه خاص باحتواء عنوان مخدم الويب الذي سوف نرسل الطلب إليه .

في قسم المستقبلات الخاص :

```
private slots :

  void receive(QNetworkReply*);
```

يعمل هذا المقبس على استقبال البيانات المستجابة للطب المرسل .

في القسم الخاص :

```
private:

  QNetworkAccessManager*  netAM;
```

مؤشر من نمط الصف "QNetworkAccessManager" الذي سوف نستخدمه لإرسال الطلب .

في ملف التحقيق "networkAM.cpp" ضمن كتلة البنّاء اكتب الرمّاز التالي :

```
netAM = new QNetworkAccessManager();

connect(netAM,SIGNAL(finished(QNetworkReply*)),this,

SLOT(receive(QNetworkReply*)));
```

حيث أنشئنا حِدث للصف "QNetworkAccessManager" , ثمّ أنشئنا اتصال بين الحَدث "finished(QNetworkReply*)" التابع لـ "netAM" و المستقبل "receive(QNetworkReply*)" , بالتالي عند قدح الحَدث الداخلي "finished" يعلِم أنه قد انتهى من عملية استقبال استجابة الطلب الذي أرسِل.

تحقيق المنهج "sendRequest" :

```
void networkAM::sendRequest(QString url){

   netAM->get(QNetworkRequest(QUrl(url)));

}
```

المنهج "get(QNetworkRequest)" التابع للكائن "QNetworkAccessManager" خاص بإرسال طلب من نمط "GET" أي تمرير وسطاء الطلب بالعنوان , أما الصف "QUrl" خاص بتفسير النص و تنسيق تشفيره و تحويله إلى عنوان ويب .

تحقيق المستقبل "receive(QNetworkReply*)" :

```
void networkAM::receive(QNetworkReply* reply){

   QString str(reply->readAll());

   std::cout << str.toAscii().data() ;

}
```

الوسيط "reply" يحوي كامل بيانات الاستجابة على الطلب المرسَل , أما المنهج "realAll()" فهو يقرأ و يحول بيانات الاستجابة إلى مصفوفة بايت .

في كتلة التطبيق الرئيسيّة "main" أضف الرمّاز الخاص بإنشاء حِدث من الصف "networkAM" و من ثمّ استدعاء المنهج "sendRequest(QString url)" لإرسال طلب و طبع استجابته على شاشة سطر الأوامر :

```
   networkAM* http = new networkAM();

   http->sendRequest("http://www.google.com");
```

مررنا لوسيط منهج إرسال الطلب عنوان موقع "google" .

عند تنفيذ التطبيق سوف تلا حظ طباعة وسوم "html" على شاشة سطر الأوامر ,و هذه الوسوم هي وسوم الصفحة الرئيسيّة لموقع "google" .

❖ تمرير وسطاء تحوي بيانات إلى طلب :

لتمرير وسطاء إلى طلب يجب أن تمرر مصفوفة بايت تحوي البيانات التي تريد إلى المنهج "post" أو "put" التابعين للصف "QNetworkAccessManager",حيث المنهج "post" يعني إرسال طلب بنمط "POST" أي تمرير وسطاء الطلب ضمن جسم الترويسة وليس ضمن العنوان مثيل "GET".

لتمرير وسيط يدعى "uid" يحوي اسم المستخدم و آخر يدعى "pwd" يحوي كلمة المرور إلى مخدم ويب , يجب أن نفعل الآتي :

بعد التصريح عن حِدث للصف "QNetworkAccessManager" , نصرح عن مثيل لمصفوفة بايت , حيث سوف تحوي الوسطاء المذكورين آنفا مع بياناتهم :

```
QByteArray params;

params.append("username=hasan&");

params.append("password=morhej HB");

QNetworkAccessManager * netRRP = new
QNetworkAccessManager();

netRRP-
>post(QNetworkRequest(QUrl("http://webserver/pageSec")),params);
```

مثلما تلاحظ مررنا طلب بنمط "POST" إلى الصفحة "pageSec" المحتواة ضمن المخدم "webserver" , و مررنا وسيطين هما اسم المستخدم و كلمة المرور , الرمز "&" بعد اسم المستخدم "username=hasan&" هي للفصل بين الوسيط الأول "اسم المستخدم" و الوسيط الثاني "كلمة المرور" , عند وجود فراغ داخل قيم الوسيط سوف يتم تشفيره إلى التشفير المئوي – هذا التشفير خاص بعناوين الويب -, مثال : "20%" تعني فراغ " " بالتشفير المئوي .

يجب أن تكون جميع البيانات التي سوف تنقل إلى مخدم ويب بوساطة طلب أن تكون مشفرة مئوية , نستطيع تشفير البيانات بنمط مئوي في Qt من خلال الصف "QUrl" باستخدام المنهج الساكن :

```
QByteArray QUrl::toPercentEncoding ( const QString & input, const QByteArray & exclude = QByteArray(), const QByteArray & include = QByteArray() )
```

حيث نمرر للوسيط النصي "input" البيانات التي نود تشفيرها , يرجع هذا المنهج مصفوفة بايت تحوي البيانات التي تم تشفيرها مئويا .

أما بالنسبة لفك التشفير المئوي نستخدم المنهج الساكن :

```
QString QUrl::fromPercentEncoding ( const QByteArray & input )
```

الوسيط "input" هو البيانات التي نود فك تشفيرها , يرجع البيانات الذي تم فك تشفيرها مئويا .

❖ بروتوكول FTP :

وهو اختصار لـ "File Transfer Protocol" , يعمل على نقل الملفات من العميل إلى مخدم الـ "FTP" , حيث ينصت مخدم الـ "FTP" على المنفذ الإفتراضي له وهو "21" , يرسل العميل طلبا له على عنوانه لتحميل أو رفع ملف ما , يتحقق المخدم أولا من اسم المستخدم و كلم المرور المرسلة من العميل ثمّ بعد موافقته على المصادقة يفتح جلسة عمل بينه و بين العميل , *يرسل المخدم بياناته إلى العميل من خلال منفذ محدد غالبا "20"* , ثمّ يتم رفع و تحميل الملفات المرداة .

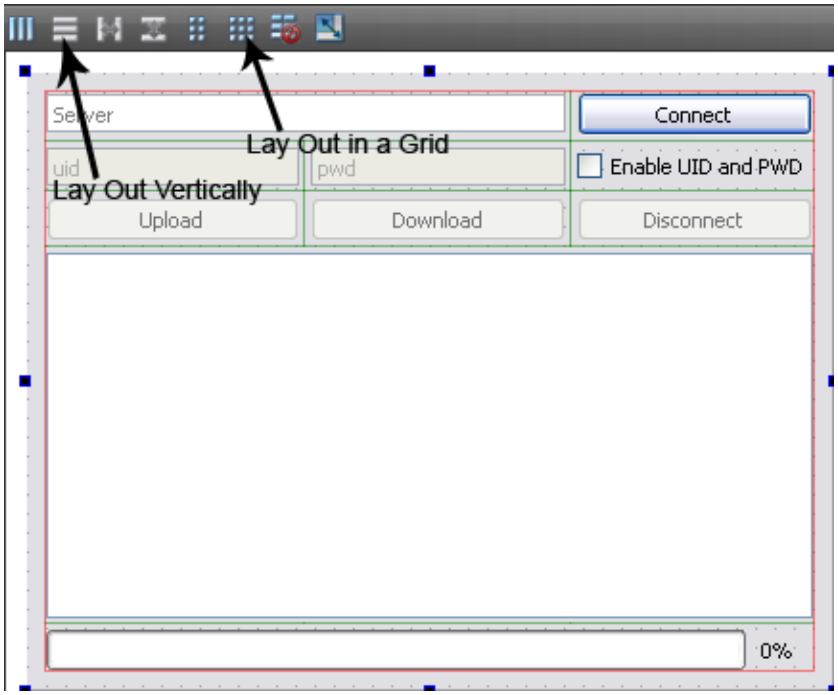
الصف "QFTP" ضمن "Qt" هو المستخدم من أجل التعامل مع تطبيقات "FTP" من قبل العميل , أما من قبل المخدم فهو يتعامل مع بروتوكول الـ "TCP" للإنصات على المنفذ "21" و المصادقة "تخويل دخول العميل" و تحميل الملفات منه و رفعها إليه الخ..

لنرى كيفيّة إنشاء تطبيق بسيط لعميل "FTP" في "Qt" .

في هذا المثال سوف نتعرف على كيفيّة إجراء اتصال بمخدم "FTP" , و عرض حالة العميل الراهنة بالنسبة للمخدم (متصل ,غير متصل , في حالة تسجل دخول الخ..), و كيفيّة عرض الملفات التي يحويها و طريقة تحميل ملف من المخدم و رفع ملف إليه , و أخيرا قطع الاتصال بالمخدم .

- **بناء تطبيق عميل FTP :**

أنشأ تطبيق "GUI" مشتق من الصف "QWidget" و سمه "ftpClient" , اذهب إلى واجهة النموذج "widget.ui" , ضع كائنات واجهة مستخدم على النموذج كما في الشكل (6.1) :



الشكل 6.1

ثمّ حدد جميع الكائنات التي على النموذج "CTRL+A" ثمّ اضغط على الزر " Lay Out in a Grid" اختصارا "CTRL+G" ليتم وضع جميع الكائنات في أداة "Grid Layout", بعدها اضغط على الزر "Lay Out Vertically" إختصارا "CTRL+L" لوضع أداة التخطيط الشبكيّة على النموذج بشكل أفقي , مثلما تلاحظ أكثر كائنات الأزرار غير مفعّلة , وهم " قطع الاتصال , تحميل , رفع " , خانات النص الغير مفعّلة "اسم المستخدم , كلمة المرور" , كائن القائمة الذي سوف يعرض فيه الملفات الموجودة داخل مخدم الـ "FTP" غير مفعّل , الآن في ملف المشروع "ftpClient.pro" ضف السطر البرمجيّ التالي :

```
QT += network
```

لتضمين المكتبة التي تحوي صفوف "Qt" المستخدمة في برمجة التطبيقات الشبكيّة ,بعدها اذهب إلى ملف الترويسة "widget.h" و ضمّن ملفات الترويسة التالية :

```
#include <QWidget>

#include <QFile>
```

```
#include <QFileDialog>

#include <QMessageBox>

#include <QtNetwork>
```

ملف الترويسة "QtNetwork" يضمّن جميع ملفات الترويسة الخاصة ببرمجة التطبيقات الشبكّة و من بينها "QFTP" و "QUrl" الذين سوف نستخدمها في هذا التطبيق .

أضف في القسم الخاص بالتصريح عن المستقبلات "slots" تعريف المناهج التاليّة :

```
void progress(qint64,qint64);

void state(int);

void listInfo(const QUrlInfo&);
```

حيث المستقبل "progress" خاص بعرض حالة التقدم لرفع و تحميل ملف ما , أما "state" فسنستخدمه لعرض حالة الاتصال الراهنة بالمخدم , و المستقبل "listInfo" خاص بعرض معلومات الملفات و المجلدات التي يحويها و يسمح بعرضها مخدم الـ "FTP" , *حيث يوجد ثلاث صلاحيات للعميل من المخدم الخاص بالـ "FTP" وهي : عرض "زيارة" و قراءة و كتابة* .

الآن في القسم الخاص صرّح عن المتحولات التالية :

```
QFtp* ftp;

QFile* file;

QUrl url;
```

"ftp" مؤشر على كائن "QFtp" لإنشاء حِدث منه يكون خاص بتحميل و رفع ملف, أما "file" فهو مؤشر على كائن "QFile" و "url" فهو مثيل لـ "QUrl" حيث سوف نستخدم من أجل تحويل عنوان المخدم من تنسيق نص بسيط إلى تنسيق نص عنوان موقع مخدم ويب .

لنكتب الرمّاز الخاص بتنفيذ العمليات المطلوبة من التطبيق و التي ذكرناها آنفا , ضمن حَدث النقر على زر الاختيار اكتب الرمّاز الخاص بتفعيل أو عدم تفعيل خانتي نص "اسم المستخدم و كلمة المرور" :

```
if(ui->checkUIDPWD->checkState() == Qt::Checked )
```

```
{
    ui->uid->setEnabled(true);
    ui->pwd->setEnabled(true);
} else{
    ui->uid->setEnabled(false);
    ui->pwd->setEnabled(false);
}
```

اذهب لحدث النقر على زر الاتصال و اكتب الرمّاز التالي :

```
url.setUrl(tr("ftp://")+ui->host->text());
if(ui->checkUIDPWD->checkState() == Qt::Checked)
{
    ftp = new QFtp(this);
    ftp->connectToHost(url.host());
    ftp->login(ui->uid->text(),ui->pwd->text());
}else{
    ftp = new QFtp(this);
    ftp->connectToHost(url.host());
    ftp->login();
}
connect(ftp,SIGNAL(stateChanged(int)),this,SLOT(state(int)));
```

أولا المنهج "setUrl(QString)" وضعنا داخله بادئة عنوان مخدم "FTP" و هي ftp:// , بالإضافة إلى محتوى خانة النص الخاصة بعنوان المخدم , العملية الشرطيّة خاص باختبار إن كان مفعّل إدخال اسم المستخدم و كلمة المرور لتسجيل الدخول أو لا , في حال كان قد تمّ تفعيل السماح بإدخال "pwd-uid" سيتم تنفيذ الخطوات التاليّة :

أولا إنشاء حِدث من الصف "QFtp" , بعدها إنشاء اتصال بمخدم الـ "FTP" بوضع عنوان المخدم المحتوى ضمن خانة النص الخاصة فيه لوسيط المنهج
"connectToHost(QString)" بعد تحويله إلى تنسيق عنوان مخدم ويب , من ثمّ تسجيل الدخول من خلال المنهج "login(QString uid,QString pwd)" , في حال كان غير مغفّل إدخال "uid-pwd" فسوف ينفذ نفس الخطوات السابقة ماعدا أنه سوف يتم تسجيل الدخول باستدعاء المنهج "login()" من دون التمرير لوسطائه اسم المستخدم و كلمة المرور .

بعدها سوف نقوم بإنشاء اتصال بين الحدث الداخلي "stateChanged(int code)" التابع لـ "ftp" و المستقبل "state(int)" , بالتالي عند قدح حَدث تغيير حالة الاتصال بالمخدم سوف ينفذ منهج المستقبل "state(int)" لعرض حالة الاتصال ضمن عنوان النافذة "widget" .

المستقبل "state(int code)" :

```
void Widget::state(int code){
  switch(code){
  case 0:
    setWindowTitle(tr("FTP Client is Unconnected"));
    ui->btn_connect->setEnabled(true);
    ui->btn_disConnect->setEnabled(false);
    ui->btn_upload->setEnabled(false);
    ui->btn_download->setEnabled(false);
    ui->fileslist->clear();
    ui->progressBar->setValue(0);
    break;
  case 1:
    setWindowTitle(tr("FTP Client is HostLookup"));break;
  case 2:
    setWindowTitle(tr("FTP Client is Connecting"));break;
```

```
    case 3:
        setWindowTitle(tr("FTP Client is Connected"));break;
    case 4:
        setWindowTitle(tr("FTP Client is LoggedIn"));

connect(ftp,SIGNAL(dataTransferProgress(qint64,qint64)),this,SLOT(pr
ogress(qint64,qint64)));
        connect(ftp,SIGNAL(listInfo(QUrlInfo)),this,SLOT(listInfo(const
QUrlInfo&)));
        ftp->cd(url.path());
        ftp->list();
        ui->btn_connect->setEnabled(false);
        ui->btn_disConnect->setEnabled(true);
        ui->btn_upload->setEnabled(true);
        ui->btn_download->setEnabled(true);
        break;
    case 5:
        setWindowTitle(tr("FTP Client is Closing"));
        ui->btn_connect->setEnabled(true);
        ui->btn_disConnect->setEnabled(false);
        ui->btn_upload->setEnabled(false);
        ui->btn_download->setEnabled(false);
        ui->fileslist->clear();
        ui->progressBar->setValue(0);
```

```
    break;
  }
}
```

يوجد 6 حالات للاتصال بمخدم "FTP" و هي :

الحالة "0" و الافتراضية : غير متصل .

الحالة "1" : استكشاف أن عنوان المخدم المعطى موجود .

الحالة "2" : محاولة الاتصال بالمخدم على العنوان المحدد .

الحالة "3" : عملية الاتصال بالمخدم قد تمّت بنجاح .

الحالة "4" : عملية تسجيل دخول المستخدم للمخدم قد تمّت بنجاح .

الحالة "5" : تمّ إغلاق الاتصال بالمخدم .

ضمن الرمّاز السابق في حال كان قد تمّ تسجيل الدخول بنجاح سوف ينفذ الخطوات التاليّة :

أولا الاتصال بين حدث مراقبة معالجة تقدم نقل البيانات لكائن "ftp" و بين المستقبل "progress(qint64 done,qint64 total)" , سوف يتم قدح حَدث التقدم في حالة رفع ملف إلى المخدم أو تحميل ملف من المخدم .

من ثمّ إنشاء اتصال بين الحَدث قائمة معلومات الملفات و المجلدات القابلة للعرض ضمن مخدم الـ "FTP" و المستقبل "listInfo(const QUrlInfo&)" , سوف يتم قدح هذا الحَدث عند استدعاء المنهج "list()" لكائن "ftp" , المنهج "cd(const QString& dir)" عند استدعاءه سيتم انتقال من عرض الدليل الحالي إلى الدليل الممرر لوسيطه و هنا وضعنا الدليل الرئيسي للمخدم "url.path()" ليتم عرض الملفات و المجلدات "الأدلة" التي يحويها من خلال استدعاء المنهج "list()" لتنفيذ المستقبل "listInfo" , حيث يحوي الرمّاز التالي :

```
void Widget::listInfo(const QUrlInfo &list){
ui->fileslist->addItem(list.name());
}
```

سوف يتم إضافة اسم الملف أو المجلد إلى كائن القائمة لعرضه , حيث يتم تنفيذ المستقبل "listInfo" لكل ملف أو دليل يجده الكائن "ftp" ضمن المخدم , في حال كائن دليل يجب علينا أن نختبر ذلك من خلال المنهج "()isDir" التابع للوسيط "QUrlInfo &list" بعدها نجري العمليات التي نرغب بها ...

الوسيط " const QUrlInfo &list " يحوي معلومات كاملة عن الملفات و الأدلة التي سمح مخدم الـ "FTP" عرضها .

ضمن كتلة حَدث النقر على زر "Download" نكتب الرمّاز التالي :

```
QString fileName(QFileDialog::getSaveFileName());

file = new QFile(fileName);

if (!file->open(QIODevice::WriteOnly)) {

   QMessageBox::information(this, tr("FTP Client"),tr("err in open"));

   delete file;

   return;

}


ftp->get(ui->fileslist->currentItem()->text(),file);
```

فتحنا نافذة حوار حفظ ملف لتحديد موقع و اسم الملف الذي سوف يتم تحميله من مخدم "FTP" , أنشئنا حِدث من صف الخاص بالوصول إلى الملفات و معالجتها "QFile" , فتحنا الملف بنمط الكتابة عليه , ثمّ استخدمنا المنهج "get(QString fileName,QIODevice*)" التابع للكائن "ftp" من اجل تجميل الملف, حيث الوسيط الأول خاص بتحديد اسم الملف من مخدم الويب الذي نود تحميله , أمّا الوسيط الثاني فهو لتحديد جهاز الدخل و الخرج لكتابة بيانات الملف الذي يتم تحميله من المخدم .

ضمن كتلة حدث التحميل "Upload" اكتب الرمّاز التالي :

```
QString fileName(QFileDialog::getOpenFileName());

QStringList strList = fileName.split("/");

file = new QFile(fileName);
```

```
  ui->progressBar->setMaximum(file->size());

  ftp->put(file,strList.at(strList.length()-1));
```

أولا حددنا مسار و اسم الملف الذي نود رفعه إلى المخدم و وضعناه في المتحول النصي "fileName" , صرحنا عن مثيل لصف القائمة النصية يدعى "strList" , استدعينا المنهج "split(QString)" الخاص بالمتحول النصي الذي يحوي مسار الملف الذي سوف يتم رفعه , ليقسم المسار إلى مصفوفة نصوص باستخدام العلّام "/"

حيث سيكون آخر نص من هذه المصفوفة هو اسم الملف مع امتداده ليتم وضعه كأسم للملف الذي سوف يرفع ,

أخذنا حِدث من الصف "QFile" والذي مشتق من الصف QIODevice , وضعنا لبنّائه مسار و اسم الملف الذي نود رفعه , القيمة العظمى لشريط التقدم الخاص بعرض حالة تقدم رفع الملف هي حجم الملف الذي حددناه , أما المنهج الخاص برفع الملف إلى المخدم هو "put(QIODevice*,QString fileName)" التابع للكائن "ftp".

رمّاز مستقبل حالة التقدم لنقل بيانات ملف "progress(qint64,qint46)" :

```
void Widget::progress(qint64 done, qint64 total){

  ui->progressBar->setMaximum(total);

  ui->progressBar->setValue(done);

  if(done == total){

    file->close();

  }

}
```

مهمته هو عرض تقدم شريط حالة التقدم نسبة لكمية البيانات التي تمّ نقلها من الكميّة الإجماليّة للملف .

الوسيط "done" يحوي قيمة حجم البيانات التي تمّ نقلها .

الوسيط "total" يحوي القيمة الإجمالية لحجم البيانات التي سوف تنقل .

داخل حَدث زر قطع الإتصال اكتب الرمّاز التالي :

```
  ftp->abort();
```

```
ftp->deleteLater();

ui->btn_connect->setEnabled(true);

ui->btn_disConnect->setEnabled(false);

ui->btn_upload->setEnabled(false);

ui->btn_download->setEnabled(false);

ui->fileslist->clear();

ui->progressBar->setValue(0);
```

المنهج "()abort" لإحباط كافة العمليات التي يقوم بها "ftp" من رفع و تحميل و قراءة لقائمة الملفات الخ..

أما المنهج "()deleteLater" فهو من الصف "QObject" يستدعى لحذف الكائن, "هنا سوف يحذف كائن الـ ftp" .

سوف تجد رمّاز هذا التطبيق كاملا في القرص المرفق باسم ftpClient .

نفذ التطبيق و اختبر النتيجة .

في حال كان يوجد وكيل "proxy" للمخدم نستطيع أن نضع عنوان الوكيل و منفذه لكائن "QFtp" من خلال المنهج :

```
int QFtp::setProxy ( const QString & host, quint16 port )
```

لإنشاء مجلد داخل المساحة المتاحة ضمن مخدم "FTP" نستدعي المنهج التالي :

```
int QFtp::mkdir ( const QString & dir )
```

لحذف مجلد داخل المخدم نستدعي المنهج :

```
int QFtp::rmdir ( const QString & dir )
```

لحذف ملف موجود داخل المخدم نستدعي المنهج :

```
int QFtp::remove ( const QString & file )
```

أخيرا لإعادة تسمية مجلد موجود على مخدم "FTP" نستدعي المنهج :

int QFtp::rename (const QString & oldname, const QString & newname)

الآن لننتقل إلى شرح كيفيّة برمجة تطبيق شبكيّ يستخدم أهم بروتوكول في عالم الشبكات وهو "TCP" .

❖ بروتوكول TCP :

وهو اختصار لـ "Transmission Control Protocol" يعد هذا البروتوكول من البروتوكولات الموثوقة و الموجهة , حيث أنه يقوم بإعداد إتصال مباشر بين الأجهزة المتصلة بالشبكة بحفظ التشكيلات الجانبية للاتصال ضمن ترويسته حيث يبقي هذه التشكيلات محفوظة ضمن ترويسة الإتصال حتى يتم إغلاقه, كلمة موثوق هو أنه يتأكد من وصول البيانات كاملة إلى الطرف المرسل له , أما موجه أي يرسل ويستقبل إلىامن طرف محدد " عنوان محدد" .

الصفوف الخاصة بالتعامل مع بروتوكول "TCP" في "Qt" :

الصف الأب للمقابس هو "QAbstractSocket" و المشتق من "QIODevice" .

الصف الخاص ببرمجة تطبيق مخدم يستخدم بروتوكول "TCP" هو "QTcpServer" .

الصف الخاص ببرمجة تطبيق عميل يستخدم بروتوكول "TCP" هو "QTcpSocket" .

***ملاحظة* :هذه الصفوف مضمنة ضمن ملف الترويسة "QtNetwork" .**

• كيفيّة عمل تطبيق مخدم :

تتلخص الخطوات التي ينفذها تطبيق مخدم لشبكة تستخدم بروتوكول "TCP" بالمراحل التالية :

1 - الإنصات إلى منفذ و عنوان محددين لمراقبة أي إتصال جديد قادم من قبل عميل .
2 - عند قدوم إتصال جديد من عميل على العنوان و المنفذ المحدد يتم قدح "رفع" حَدث لإخبار تطبيق المخدم بقدوم الإتصال .
3 - يجلب تطبيق المخدم المعلومات عن هذا الإتصال و التي تكون موجودة في ترويسة الإتصال الجديد .
4 - إما يقبل المخدم الإتصال أو يرفضه .

5 - في حال قد تمت موافقة تطبيق المخدم على الإتصال الجديد يقوم بفتح مقبس شبكيّ "socket" خاص بهذا العميل يحوي ترويسة داخلها معلومات هذا الإتصال لاستقبال و لإرسال البيانات منه و إليه .

ملاحظة : يبقى المخدم محتفظ بمعلومات العميل من عنوان و منفذ و الخ.. حتى يتم إغلاق مقبس هذا العميل .

6 - يهيئ المخدم مقبس العميل لاستقبال البيانات منه .
7 - عند قدوم بيانات من العميل يهيئ المخدم تلك البيانات ثمّ يقدح حَدث لإخبار تطبيقه أنه مستعد للقراءة من مقبس العميل .
8 - في حال أرسل المخدم بيانات إلى تطبيق العميل فيرسلها بوساطة المقبس الخاص بالعميل المطلوب .
9 - أخيرا يغلق تطبيق المخدم الإتصال مع مقبس العميل ويزيل كائن المقبس الخاص بهذا العميل .

ملاحظة : إذا تمّ إغلاق مقبس العميل من طرفه فسيقدح حَدث لإخبار تطبيق المخدم أنه قد تمّ قطع الإتصال مع العميل .

• كيفيّة عمل تطبيق عميل :

تتلخص الخطوات التي ينفذها تطبيق عميل لشبكة تستخدم بروتوكول "TCP" بالمراحل التالية :

1 - يقوم بالتأكد من أن العنوان و المنفذ الذي سوف يجري عليهما الإتصال موجودين ضمن الشبكة .
2 - في حال تأكد من صحة عنوان و منفذ المخدم المعطى له سوف يقوم بإرسال طلب إتصال مع المخدم .
3 - إن تمت استجابة المخدم بالموافقة على طلبه سوف يقوم تطبيق العميل بقدح "رفع" حَدث أنه قد تم الإتصال بالمخدم .
4 - يهيئ تطبيق العميل نفسه لاستقبال البيانات من المخدم .
5 - عند قدوم بيانات من مخدم يهيئ العميل البيانات الواردة إليه من ثمّ يقدح حَدث يخبر بأن البيانات قد تمّ تجهيزها للقراءة .
6 - إرسال بيانات إلى مخدم يتم بوساطة مقبس العميل ذاته .
7 - أخيرا يغلق الإتصال مع تطبيق المخدم .

ملاحظة : إذا تمّ إغلاق مقبس العميل من طرف المخدم فسيقدح العميل حدث لإخبار تطبيق أنه قد تمّ قطع الإتصال مع المخدم .

الآن لنبني تطبيق مخدم و تطبيق عميل لبروتوكول "TCP" , لنسهل فهم كيفيّة برمجة تطبيقات الشبكة في Qt عبر الأمثلة .

• تطبيق معالجة الملفات من قبل المخدم :

تتلخص فكرة المشروع ببناء تطبيق مخدم و م تطبيق عميل , حيث مهمة المخدم الإنصات إلى عنوان و منفذ محددين لمراقبة الاتصالات القادمة من العملاء إليه , بعد إتصال عميل إليه ,يستقبل منه ملف نصي يقوم المخدم بقراءته و كتابة جملة في آخر الملف النصي تقول بأنه قد تمت معالجة الملف من قبل المخدم بعدها يرسل الملف ذاته بعد التعديل عليه إلى العميل الذي طلب معالجة الملف المحدد , بعدها يقوم تطبيق المخدم بإغلاق الإتصال مع هذا العميل .

أما تطبيق العميل يقوم بفتح نافذة تحوي زر لفتح ملف و معالجته من قبل المخدم , عند النقر على الزر تظهر نافذة حوار لإدخال عنوان "IP" المخدم المطلوب , بعدها تظهر نافذة حوار أخرى لإدخال منفذ المخدم ليتم الإتصال على عنوان و منفذ المخدم المدخلين , في حال نجح الإتصال مع المخدم يقوم تطبيق العميل بإظهار نافذة فتح ملف لاختيار الملف النصي المراد معالجته من قبل المخدم , بعد تحديد الملف يقوم العميل بإرساله إلى المخدم , ويهيئ نفسه لاستقبال الملف بعد معالجته , بعد أن تتم عملية المعالجة يرسل المخدم الملف المعالج إلى العميل , يستقبلها العميل من ثمّ يعرضها داخل كائن "QTextEdit" , أخيرا يغلق العميل اتصاله مع المخدم .

• تطبيق مخدم لمعالجة ملف :

أنشأ تطبيق سطر أوامر و سمّه "fileProcess-server" , داخل إلى ملف مشروعه أضف السطر التالي الخاص بتضمين المكتبة التي تحوي صفوف التطبيقات الشبكيّة في Qt عند تجميعه :

```
QT += network
```

أضف صف جديد يدعى "server" مشتق من الصف "QObject" , ضمّن ملفات الترويسة التالية داخل الملف "server.h" :

```
#include <iostream>

#include <QtNetwork>
```

في القسم العام أضف المنهج الخاص بالإنصات إلى عنوان و منفذ محددين :

int port); address, bool listen(std::string

حيث يعيد القيمة "true" في حال نجح في الإنصات على العنوان و المنفذ المحددين , و إلا يعيد "false" .

في قسم التصريح عن المستقبلات عرف المناهج "المستقبلات" التالية :

```
public slots:

   void newConn();

   void textProcessing();

   void displayError(QAbstractSocket::SocketError);
```

المستقبل الأول لمعالجة إتصال جديد لعميل , أما المستقبل الثاني لاستقبال ومعالجة الملف الوارد من العميل , المستقبل الأخير لعرض ماهية خطأ إذا حصل داخل المخدم .

في القسم الخاص صرح عن المؤشرات التالية :

```
private:

   QTcpServer* m_server;

   QTcpSocket* m_socket;
```

المؤشر "m_server" فهو خاص بكائن تطبيق مخدم "TCP" , أما المؤشر "m_socket" خاص باحتواء معلومات العميل الذي قد تم موافقة المخدم على اتصاله به .

اذهب إلى ملف تحقيق الصف "server.cpp" , ضمن كتلة بنّائه أضف الرمّاز التالي :

```
   m_server = new QTcpServer();

   connect(m_server,SIGNAL(newConnection()),this,SLOT(newConn()));
```

أنشئنا حِدث للصف "QTcpServer" , ثمّ أنشئنا إتصال بين الحَدث الداخلي "newConnection()" التابع للحِدث "m_server", و المستقبل "newConn()" , بهذه الحالة عندما يأتي إتصال جديد على المخدم يقدح هذا الحَدث و بالتالي تنفذ كتلة تعليمات المستقبل "newConn" لمعالجة الإتصال الوارد .

تحقيق المنهج العام "listen(std::string address , int port)" الخاص بإجراء عملية الإنصات على العنوان و المنفذ الممرر لوسطائه :

```
bool server::listen(std::string address,int port){
```

```
   m_server->listen(QHostAddress(address.data()),port);
      if( m_server->isListening() ){
         std::cout << "the server is listening on " << address << ":" << port
;
    return true;
   }
   else{
      std::cout << "can't listen on this address.\nplz try again..\n";
   return false;
   }
}
```

استخدمنا المنهج "listen(QString,int)" التابع للصف "QTcpServer" , المسؤول عن إجراء عملية الإنصات , في حال قد تمت عملية الإنصات على العنوان المحدد بنجاح سيطبع نص على شاشة سطر الأوامر تخبر المستخدم بأن عملية الإنصات نجت , و يعيد القيمة "true" , و في حال فشلها يخبر المستخدم بأن عملية الإنصات فشلت من خلال طباعة نص على شاشة سطر الأوامر و إعادة القيمة "false" .

تحقيق المستقبل "newConn()" :

```
void server::newConn(){
    m_socket =  m_server->nextPendingConnection();
std::cout << "\nReceive connection from " <<
 m_socket->peerAddress().toString().toAscii().data() << std::endl ;
connect(m_socket, SIGNAL(readyRead()),
      this, SLOT(textProcessing()));
connect(m_socket,
SIGNAL(disconnected()),m_socket,SLOT(deleteLater()));
```

```
connect(m_socket, SIGNAL(error(QAbstractSocket::SocketError)),
        this, SLOT(displayError(QAbstractSocket::SocketError)));
}
```

يتم تنفيذ هذا المستقبل عند قدوم إتصال جديد من قبل عميل , فيقوم بمعالجة هذا الإتصال كالأتي :

أولا يعطي قيمة حدث المقبس الشبكيّ الوارد الذي يحوي معلومات العميل المتصل للمؤشر "m_socket" من خلال المنهج "nextPendingConnection()" التابع للصف "QTcpServer" , بعدها يطبع على شاشة سطر الأوامر أنه قد تمّ استقبال إتصال من "العنوان و المنفذ المحدد" , يقوم بإنشاء إتصال بين الحَدث الداخلي "readyRead()" التابع للكائن "m_socket" و بين المستقبل "textProcessing()" الخاص باستقبال البيانات من العميل و معالجتها , و ينشأ اتصال أيضا بين الحَدث الداخلي "disconnected()" و المستقبل "deleteLater()" التابع للكائن "QObject" لإزالة كائن مقبس العميل عند إغلاق الإتصال معه ,

أخيرا ينشأ اتصال بين حَدث الخطأ التابع للصف "QTcpSocket" و المستقبل "displayError(QAbstractSocket::SocketError)" الخاص بإظهار رسالة فحواها مصدر خطأ مقبس العميل في حال حصل .

تحقيق المستقبل الخاص باستقبال البيانات "الملف هنا" المرسلة من قبل العميل و معالجتها :

```
void server::textProcessing(){
   QByteArray Textfile;
   QDataStream dataStream4R(m_socket);
   dataStream4R.setVersion(QDataStream::Qt_4_7);
   dataStream4R >> Textfile ;
   Textfile.append("\nthis file had been processed by Server...\n");

   QByteArray data4Send ;
   QDataStream dataStream4W(&data4Send,QIODevice::WriteOnly);
```

```
    dataStream4W.setVersion(QDataStream::Qt_4_7);

    dataStream4W << Textfile ;

    m_socket->write(data4Send);


    m_socket->waitForBytesWritten();

    std::cout << "the file had been processed and sent to " << m_socket-
>peerAddress().toString().toAscii().data() << "..\n" << std::endl ;

    m_socket->disconnectFromHost();

}
```

أولا أنشئنا مثيل مصفوفة بايت يدعى "Textfile" مهمته احتواء البيانات المرسلة من العميل , أنشئنا مثيل لصف مجرى بيانات يدعى "dataStream4R" مررنا لبنائه مقبس العميل المراد استقبال بيانات منه "m_socket" , بما أن الصف "QTcpSocket" مشتق من الصف "QAbstractSocket" و الصف "QAbstractSocket" مشتق من الصف "QIODevice" لهذا صف مجرى البيانات يقبل أن نضع قيمة وسيط بناءه الذي نمطه "QIODevice*" مقبس الشبكة كدخل للمجرى "QTcpSocket*" , إصدار مجرى البيانات هو "QDataStream::Qt_4_7" , وضعنا البيانات المقروءة من مقبس العميل داخل مصفوفة البايت "Textfile" بعد فك سلسلتها , ثمّ أضفنا النص " this file had been processed by Server..." في نهاية البيانات المقروءة من المقبس , الآن يجب علينا أن نرسل هذه البيانات إلى العميل بعد أن تمّت مهمة معالجتها بنجاح .

صرحنا عن مثيل لصف مصفوفة البايت يدعى "data4Send" ، ثم صرحنا عن مثيل لصف مجرى البيانات لسلسلة البيانات التي نود إرسالها , يدعى هذا المثيل "dataStream4W" مررنا لوسيط بنائه مرجع مصفوفة البايت الذي نود كتابة البيانات التي سوف نرسلها عليه طبعا بعد سلسلتها , وضعنا الإصدار الذي نريد و الذي سوف يكون متفق عليه من قبل الطرفين "تطبيق المخدم - تطبيق العميل" , بعدها كتبنا البيانات "النص المعالج" "Textfile" على مجرى البيانات , في هذه الحال سوف تكتب البيانات على المصفوفة "data4Send" بعد سلسلتها , أرسلنا البيانات إلى العميل المحدد باستخدام المنهج "write(QByteArray)" باستدعائه من الحدث "m_socket" , استخدمنا المنهج "waitForBytesWritten()" من أجل حظر الوصول إلى المقبس حتى يتم إرسال جميع البيانات المطلوب منه إرسالها , بعد أن يتم الإرسال بنجاح نطبع على شاشة سطر الأوامر أنه قد تمّت معالجة و إرسال البيانات بنجاح إلى العميل , أخيرا أغلقنا مقبس العميل مع المخدم ,

عند إغلاق الإتصال سوف يتم تلقائيا إزالة حدث مقبس العميل من الذاكرة لأننا أجرينا مسبقا إتصال بين الحدث الداخلي "disconnected()" و المستقبل "deleteLater()" .

تحقيق مستقبل عارض الخطأ :

```
void server::displayError(QAbstractSocket::SocketError err){

  switch(err){

  case QAbstractSocket::ConnectionRefusedError : std::cout <<
"Connection Refused Error" ;break;

  case QAbstractSocket::UnknownSocketError: std::cout << "Unknown
Socket Error" ; break;

  }

}
```

الوسيط "QAbstractSocket::SocketError" هو تعداد يحوي مجموعة العناصر لوصف الخطأ الحاصل ضمن المقابس ، هنا استخدمنا فقط عنصرين من التعداد , سوف نضع جدول في هذا الفصل يحوي أكثر العناصر من التعداد "QAbstractSocket::SocketError" استخداما لوصف الأخطاء .

آخر خطوة هي استدعاء صف المخدم الذي أنشئناه "server" من كتلة التطبيق الرئيسية "main" , لفعل ذلك اذهب لداخل الكتلة "main" , ضمّن ملف الترويسة "server.h" ليتاح لنا استخدام صف المخدم داخل الملف "main.cpp", ثمّ اكتب الرمّاز التالي :

```
  server* Srv = new server();

  forever{

  std::string address;

  int port;

  std::cout << "Enter IP 'address' for listen: " ;

  std::cin >> address ;

  std::cout << "then enter the port: " ;

  std::cin >> port ;
```

```
if(Srv->listen(address,port))break;

}
```

أنشئنا حدث لصف المخدم يدعى "Svr" , بعدها نفذنا حلقة تكرار لا نهائيّة من أجل إدخال المستخدم لعنوان المخدم و منفذه الذي سوف يتنصت عليه , في حال كان متاح التنصت على العنوان و المنفذ المحددين يخرج من حلقة التكرار و إلا يعيد للمستخدم التحكم بالإدخال .

• تطبيق العميل لمخدم معالجة ملف :

أنشأ تطبيق فارغ و سمّه "fileProcess-client" , اكتب داخل ملف مشروعه "fileProcess-client.pro" :

```
QT += network
```

ثمّ أنشأ صف مشتق من "QObject" و سمّه "client" , اذهب لداخل الملف "client.h" و نفذ التالي :

أولا ضمّن ملفات الترويسة للصفوف التي سوف نستخدمها بالتطبيق و هي :

```
#include <QtNetwork>

#include <QFileDialog>

#include <QLabel>

#include <QPushButton>

#include <QBuffer>

#include <QTextEdit>

#include <QInputDialog>

#include <QMessageBox>
```

ثمّ قي القسم العام للتصريح عن المستقبلات عرّف المناهج التالية :

```
public slots:

  void readData();
```

```
void click();
```

المستقبل "readData()" خاص بقراء بيانات الملف التي قد تم معالجته من قبل المخدم , أما المستقبل "click()"من أجل تنفيذ التعليمات الخاصة بفتح ملف و إرساله إلى المخدم لمعالجته , طبعا يتم تنفيذه عند النقر على الزر الذي سوف تكتب رمّاز خاص به بعد قليل .

في قسم التصريح الخاص اكتب التالي :

```
private:

   QTcpSocket* m_socket;

   QString host;

   int port;
```

صرّحنا عن مؤشر لمقبس شبكة يدعى "m_socket" , و عن مثيل للصف "QString" يدعى "host" سنستخدمه من أجل حفظ عنون المخدم , و آخر يدعى "port" لحفظ المنفذ الذي يستخدمه المخدم .

إنتقل إلى داخل ملف التحقيق للصف "client" و الذي يكون "client.cpp" و نفذ التالي :

ضمن كتلة بناء الصف اكتب الرمّاز :

```
   m_socket = new QTcpSocket();

   connect(m_socket,SIGNAL(readyRead()),this,SLOT(readData()));


   QPushButton* btn = new QPushButton(tr("Open File for process by
server.."));

   connect(btn,SIGNAL(clicked()),this,SLOT(click()));

   btn->show();

   btn->setWindowTitle(tr("Client.."));
```

أنشئنا حدث لصف مقبس شبكة لنستطيع الإتصال إلى مخدم , و إرسال و استقبال بيانات إلى \ من المخدم , ثمّ أنشئنا إتصال بين الحَدث الداخلي "readyRead()" التابع للحِدث "m_socket" لإخبارنا بوصول بيانات من مخدم , و بين المستقبل "readData()"

الخاص بمعالجة تلك البيانات و عرضها , بعدها صرحّنا عن كائن لزر يدعى "btn" و ربطنا حدث نقره بكائن المستقبل "()click" من ثمّ أظهرنا الزر .

تحقيق المستقبل "()click" :

```
void client::click(){
   host = QInputDialog::getText(0,tr("Client"),
   tr("Enter the server IP"),QLineEdit::Normal,"127.0.0.1");
   port = QInputDialog::getInt(0,tr("Client"),tr("Enter the server
Port"),6666);
   QString fileName(QFileDialog::getOpenFileName(0,0,0,tr("Text
File(*.txt)")));
   QFile file(fileName);
   file.open(QIODevice::ReadOnly);
   QByteArray byteArray;
   QDataStream dataStream4W(&byteArray,QIODevice::WriteOnly);
   dataStream4W.setVersion(QDataStream::Qt_4_7);
   dataStream4W << file.readAll() ;
   m_socket->connectToHost(QHostAddress(host),port);
   if(m_socket->waitForConnected()){
   if(m_socket->state() == QAbstractSocket::ConnectedState){;
   m_socket->write(byteArray);
   m_socket->waitForBytesWritten();
   }
   }else{
```

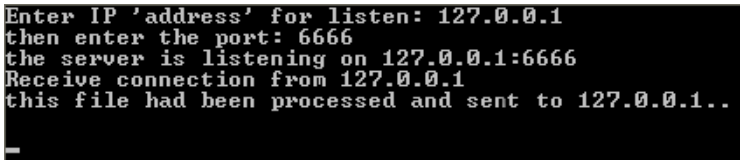
```
    QMessageBox::information(0,tr("error"),tr("can't connect to
server\nplz try again"));

  }


}
```

أولا أظهرنا إثنان من نوافذ الحوار الخاصة بالإدخال , الأولى خاصة بإدخال عنوان "IP" المخدم و الثانية بمنفذ المخدم , و حفظنا القيم المدخلة من عنوان و منفذ بالمتحولات "port - host" الذي قد تمّ التصريح عنهم مسبقا, بعدها أظهرنا نافذة حوار فتح ملف , ليختار المستخدم الملف النصي الذي يود معالجته من قبل المخدم , و و ضعنا مسار و اسم الملف المحدد داخل المتحول "fileName" , ثمّ انشئنا مثيل للصف "QFile" و مررنا لبنائه مسار الملف المحدد بعدها فتحنا الملف بنمط القراءة فقط "ReadOnly" , قرأنا الملف و وضعنا بياناته داخل مصفوفة البايت "byteArray" بعد سلسلته , اصدار مجر البيانات هو "Qt_4_7" , أجرينا إتصال على المخدم من خلال المنهج "connectToHost(QHostAddress,int port)" التابع للحِدث "m_socket" , عند التأكد من صحة الإتصال بالمخدم "أولا التأكد من عنوان و منفذ المخدم – ثانيا التأكد من قبول المخدم لإتصال العميل به" نكتب بيانات الملف المحدد على المقبس "m_socket" ليتم إرساله إلى المخدم المتصل به , بعدها استدعينا المنهج "waitForBytesWritten()" لحظر الوصول إلى المقبس حتى يتم إرسال جميع البيانات المراد إرسالها إلى المخدم , في حال فشل الإتصال بالمخدم تظهر رسالة خطأ تخبرنا بذلك .

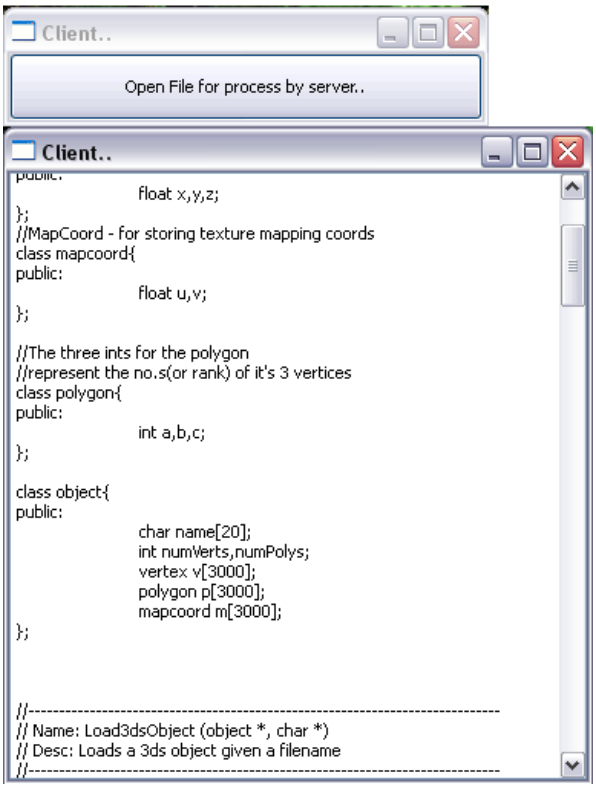
داخل ملف "main.cpp" ضمن كتلة التطبيق الرئيسية "main" اكتب السطر البرمجي الذي يستدعي صف العميل لتنفيذه :

```
client* Clnt = new client();
```

نفذ الآن تطبيق المخدم , بعدها نفذ تطبيق العميل و اختبر النتيجة , في الشكل (6.2) تطبيق المخدم و الشكل (6.3) تطبيق العميل :



الشكل 6.2



الشكل 6.3

❖ بروتوكول UDP :

وهو اختصار "User Datagram Protocol" يتميز هذا البروتوكول بسرعته في نقل البيانات لكنه بالمقابل منخفض الوثوقية و غير موجه , حيث يقوم تطبيق المرسل الذي يستخدم هذا البروتوكول بعملية الإرسال بنمط بث إذاعي أي لا يأبه بمن سوف يستقبل رسالته هذا معنى غير موجه , أما غير موثوق أي لا يختبر أن البيانات التي أرسلها قد تمة وصولها إلى المستقبل "العميل" بنجاح , في هذه الحالة ممكن أن تفقد بعض من البيانات المرسلة في عملية الإرسال ,لذ ممكن أن نستخدمه فقط في إرسال الوسائط المتعددة كالفيديو و الصوت حيث لا يأثر ضياع عدة رزم عليهم بشكل كبير .

ملاحظة : بروتوكول "UDP" لا يعمل بوجود موجه "Router" بين المرسل و بين المستقبل , لأن الموجه لا يدعم البث الإذاعي "Broadcast" , يوجد حل وحيد لتفادي هذه المشكلة لكنه خارج نطاق كتابنا هذا .

بما أننا قلنا لا يأبه تطبيق المرسل الذي يستخدم بروتوكول "UDP" بمن سوف يستقبل رسالته "بياناته المرسلة" , فهذا يعني أنه ليس على إتصال مباشر بين المخدم "المرسل" و العميل "المستقبل" مثل بروتوكول "TCP" , فبروتوكول "UDP" يضع "إن حدد" العنوان و المنفذ الذي سوف يرسل لهما ضمن رزمة البيانات التي سوف يرسلها "أيّ من دون تشكيل إتصال مباشر بين المرسل و المستقبل".

الصفوف الخاصة بالتعامل مع بروتوكول "UDP" في "Qt" :

الصف الأب للمقابس هو "QAbstractSocket" و المشتق من "QIODevice" .

الصف الخاص ببرمجة تطبيق شبكيّ يستخدم بروتوكول "UDP" هو "QUdpSocket" .

***ملاحظة* :هذه الصفوف مضمنة ضمن ملف الترويسة "QtNetwork" .**

• مشروع محادثة :

تتلخص فكرة المشروع ببناء تطبيق مخدم و تطبيق عميل يستخدمان بروتوكول "UDP", تطبيق المخدم مهمته استقبال الرسائل من عملاء على منفذ محدد , من ثمّ يرسلها إلى جميع العملاء , أمّا تطبيق العميل فيسمح للمستخدم بإدخال نص من ثمّ يرسل النص المدخل إلى المخدم المحدد , و يقوم العميل أيضا بمراقبة وصول رزمة "حزمة" بيانات التي تجسدها الرسالة المرسلة من المخدم ليعرضها .

• بناء مخدم المحادثة :

أنشأ تطبيق سطر أوامر و سمّه "ChatServer-udp" , و اكتب ضمن ملف مشروعه السطر التالي الخاص بتضمين مكتبة صفوف شبكة "Qt" عند التجميع :

```
QT += network
```

أنشأ صف يدعى "server" مشتق من "QObject", داخل ملف ترويسته ضمّن الصفوف التالية :

```
#include <QtNetwork>

#include <iostream>
```

في قسم التصريح عن المستقبلات صرّح عن المستقبل الخاص بقراءة رزمة البيانات المرسلة من العميل :

```
public slots:
   void readData();
```

في قسم التصريح الخاص , صرّح عن مؤشر للصف "QUdpSocket" :

```
private:
   QUdpSocket* m_Server;
```

اذهب إلى ملف التحقيق "server.cpp" و اكتب الرمّاز التالي ضمن كتلة بنائه :

```
   m_Server = new QUdpSocket(this);
   m_Server-
>bind(QHostAddress::Any,7777,QUdpSocket::DontShareAddress);
   connect(m_Server,SIGNAL(readyRead()),this,SLOT(readData()));
   if( m_Server->state() == QAbstractSocket::BoundState)
      std::cout << "the Server is running...\n" << std::endl ;
   else
      std::cout <<"can't bind on this address and port.\n";
```

أولا أنشئنا حدث للصف "QUdpSocket" , ثمّ ربطنا الحِدث "m_Server" على المنفذ "7777" , وسيط العنوان و ضعنا قيمته العنصر "QHostAddress::Any" , نقصد أنه سوف يقرأ رزم البيانات من أي عنوان و لكن على المنفذ "7777" , بالنسبة لنمط الربط و الذي و ضعنا قيمته على "QUdpSocket::DontShareAddress" أي أنه لن يسمح لأي تطبيق آخر استخدام نفس المنفذ و العنوان المحددين , بعدها أنشئنا إتصال بين الحدث الداخلي "readyRead()" التابع للحدث "m_Server" و المستقبل "readData()" الخاص بقراءة البيانات "الرزم" و من ثمّ بثّها "إرسالها" إلى جميع العملاء ,في حال تمت عملية الربط على العنوان و المنفذ المحددين سوف يطبع على شاشة سطر الأوامر أنه قد تمّ الربط بنجاح.

التعداد "QAbstarctSocket::SocketState" :

غير متصل	QAbstractSocket::UnconnectedState
البحث و التأكد من العنوان و المنفذ المحددين	QAbstractSocket::HostLookupState
محاولة الإتصال	QAbstractSocket::ConnectingState
نجاحه في عملية الإتصال	QAbstractSocket::ConnectedState
تمّ ربط المقبس على المنفذ و العنوان المحددين	QAbstractSocket::BoundState
إغلاق الإتصال	QAbstractSocket::ClosingState
الإنصات على العنوان و المنفذ المحددين	QAbstractSocket::ListeningState

التعداد "QUdpSocket::BindFlag" :

السماح لتطبيق آخر ربط المقبس على نفس العنوان و المنفذ المحددين في التطبيق الحالي, لا يعمل على نظام "Win"	QUdpSocket::ShareAddress
عدم السماح لتطبيق آخر ربط المقبس على نفس العنوان و المنفذ المحددين في التطبيق الحالي, لا يعمل على نظام " Mac OS X- Unix"	QUdpSocket::DontShareAddress
السماح لتطبيق آخر ربط المقبس على العنوان و المنفذ المحددين في التطبيق الحالي , لا يعمل على "Unix"	QUdpSocket::ReuseAddressHint
الخيار الإفتراضي للنظام الحالي , هو عدم السماح لتطبيق آخر ربط مقبسه على نفس العنوان و المنفذ المحددين في تطيقنا الحالي .	QUdpSocket::DefaultForPlatform

تحقيق المستقبل "readData()" الخاص بقراءة الرسائل من العملاء و من ثمّ إرسالها إلى جميع عملاءه:

```
void server::readData(){
   while (m_Server->hasPendingDatagrams()){
      QString array4Send;
   bool printData = true ;
   QByteArray datagram;
   datagram.resize(m_Server->pendingDatagramSize());
   QHostAddress hostIP;
   quint16 hostPort;

   m_Server->readDatagram(datagram.data(), datagram.size(),
                     &hostIP, &hostPort);

   QList<QHostAddress> localAdresses =
QNetworkInterface::allAddresses();
   for(int i=0;i <= localAdresses.size()-1 ;i++)
      if(localAdresses.at(i).toString() == hostIP.toString()) printData =
false ;

   QString strMsg(datagram);
   if(printData){
      array4Send.append( "Host IP: " + hostIP.toString() + "\n" );
      array4Send.append( "Host Port: " + QString::number(hostPort) +
"\n" );
```

```
    array4Send.append( "Messgae is " +  strMsg + "\n" );
    std::cout << array4Send.toAscii().data() << std::endl ;
   m_Server->writeDatagram(array4Send.toAscii().data()  ,
   QHostAddress::Broadcast,7777);
   strMsg.clear();
   array4Send.clear();
  }

  }
}
```

المنهج "()hasPendingDatagrams" التابع لـ "QUdpSocket" يعيد القيمة "true" طالما يوجد بيانات ضمن المقبس تنتظر قراءتها , و عندما يتم قراءتها يعيد هذا المنهج القيمة "false" , أعدنا تحجيم مصفوفة البايت "datagram" بحيث نهيئ مساحة فارغة من المصفوفة مساوي لحجم البيانات "الرزم" التي قد استقبلها المقبس من اجل أن نضع الرزم المستقبلة داخلها , قرئنا الرزم المستقبلة من خلال المنهج "readDatagram" , الذي قد مررنا لوسطائه مؤشر مصفوفة البايت "()datagram.data" و الحجم المتاح لتخزين البيانات داخلها , و مرجع لمتحول نصي من أجل تخزين عنوان المرسل و متحول آخر من نمط "int" لتخزين منفذ المرسل .

المنهج الساكن "()QNetworkInterface::allAddresses" يعيد قائمة من عناوين المضيفين الموجودين على النظام الحالي , هنا نختبر إذا كان عنوان المرسل هو نفسه المخدم "تطبيقنا الحالي" في هذه الحال عدم عرض الرسالة المرسلة منه .

هيّئنا متحول نصي يدعى "strMsg" يحوي الرزمة "الرسالة" التي أرسلها عميل ما من أجل أن يرسلها إلى جميع العملاء "Broadcast" , نرسل البيانات من خلال المنهج "writeDatagram" , من أجل الإرسال بنمط البث الإذاعي وضعنا "QHostAddress::Broadcast" ضمن الوسيط الخاص بالعنوان التابع للمنهج "writeDatagram" , المنفذ المحدد من أجل الإرسال هو "7777" .

التعداد "QHostAddress::SpecialAddress" :

كائن العنوان فارع , مساوي لـ QHostAddress()	QHostAddress::Null
العنوان المحلي لبروتوكول "IPv4" , مساوي لـ QHostAddress("127.0.0.1")	QHostAddress::LocalHost
العنوان المحلي لبروتوكول "IPv6" , مساوي لـ QHostAddress("::1")	QHostAddress::LocalHostIPv6
عنوان البث الإذاعي لبروتوكول "IPv4" , مسوي لـ QHostAddress("255.255.255.255")	QHostAddress::Broadcast
أي عنوان لبروتوكول "IPv4" , مساوي لـ QHostAddress("0.0.0.0")	QHostAddress::Any
أي عنوان لبروتوكول "IPv6" , مساوي لـ QHostAddress("::")	QHostAddress::AnyIPv6

الآن اذهب إلى الملف "main.cpp" و اكتب آخر سطر برمجي في التطبيق ضمن الكتلة "main" و الذي مهمته إنشاء حدث عن صف المخدم "server" و هو :

```
server* Server = new server();
```

لا تنسى تضمين ملف ترويسة صف المخدم "server.h" ضمن الملف "main.cpp" , الآن لننتقل إلى بناء تطبيق عميل المحادثة .

• بناء تطبيق العميل :

أولا أنشأ تطبيق سطر أوامر يدعى "ChatClient-udp" , اكتب ضمن ملف مشروعه :

```
QT += network
```

أنشأ صف يدعى "client" مشتق من "QObject", اذهب لداخل ملف ترويسته "client.h" ونفذ التالي :

ضمّن ملفات الترويسة التالي :

```
#include <QtNetwork>

#include <iostream>

#include <QtCore>

#include <QFuture>

#include <string>
```

عرّف المستقبل الخاص بقراءة رزم البيانات "الرسائل" القادمة من المخدم و عرضها على شاشة سطر الأوامر :

```
public slots:

   void readData();
```

ضمن قسم التصريح الخاص صرّح مؤشر لمقبس بروتوكول الـ "UDP" :

```
private:

   QUdpSocket* m_Client;
```

اذهب إلى داخل ملف التحقيق الخاص بالصف "client.cpp" , ضمن كتلة بنّائه اكتب الرمّاز التالي :

```
   m_Client = new QUdpSocket(this);

   m_Client->bind(7777,QUdpSocket::ReuseAddressHint);

   connect(m_Client,SIGNAL(readyRead()),this,SLOT(readData()));

   QFuture<void> future =
QtConcurrent::run(::sendData,m_Client,this);


   std::cout << "Client is ready to send and receive Messages...\n" <<
std::endl ;
```

أنشئنا حِدث للصف "QUdpSocket" , بعدها ربطنا المقبس على المنفذ "7777" , و بنمط السماح التطبيق آخر إعادة ربط مقبسه على نفس المنفذ الحالي للتطبيق , ثمّ أنشئنا إتصال بين الحدث "readyRead" و المستقبل "readData" , بعدها فتحنا مسلك عالي المستوى

لتنفيذ المنهج الذي يدعى "sendData(QUdpSocket*,Client*)" داخل هذا المسلك , الوسيط الأول له هو مؤشر لمقبس "UDP" , أمّا الوسيط الثاني هو مؤشر للصف الحالي "client" .

عرّف المنهج "sendData" الذي مهمته إدخال المحارف من سطر الأوامر "الرسالة الذي سوف نرسلها إلى المخدم" و من ثمّ إرسالها :

```
void sendData(QUdpSocket*,client*);
```

تحقيق المنهج "sendData" :

```
void sendData(QUdpSocket* udpClient,client* obj){
  forever{
    std::string str;
    std::getline(std::cin,str);

    if(str == "Exit")
    {
      udpClient->disconnect(obj,SLOT(readData()));
      QCoreApplication::instance()->quit();
      break ;
    }

    udpClient-
>writeDatagram(str.data(),QHostAddress("111.111.1.1"),7777);
  }
}
```

فتحنا حلقة لانهائيّة لمراقبة المفاتيح المدخلة و حفظها في متحول نصي من ثمّ بعد الضغط على مفتاح العودة "الإدخال - Enter" يرسل البيانات إلى المخدم , في حال أدخلنا الكلمة

"Exit" يتم الخروج من الحلقة اللانهائيّة و من ثمّ الخروج من التطبيق ككل , عنوان المخدم الذي سوف نرسل إليه رزم البيانات هو "111.111.1.1" .

تحقيق المستقبل الخاص بقراءة البيانات الواردة من المخدم :

```
void client::readData(){
   while (m_Client->hasPendingDatagrams()){
   QByteArray datagram;
   datagram.resize(m_Client->pendingDatagramSize());
   QHostAddress hostIP;
   quint16 hostPort;
   m_Client->readDatagram(datagram.data(), datagram.size(),
                          &hostIP, &hostPort);

   QString str(datagram);
   std::cout << str.toAscii().data() << std::endl ;
   }

}
```

ترجمة الرمّاز هي : طالما أنه يوجد بيانات للقراءة ضمن المقبس أعد تحجيم مصفوفة البايت "datagram" من ثمّ اقرأ رزم البيانات القادمة و ضعها ضمن المصفوفة "datagram" , اطبع على شاشة سطر الأوامر الرسالة الواردة من المخدم .

اكتب ضمن الكتلة الرئيسيّة للتطبيق "main" السطر البرمجي التالي :

```
client* Client = new client();
```

نفذ تطبيق المخدم و من ثمّ تطبيقين من تطبيق العميل و اختبر النتيجة .

❖ الحزمة Connectivity :

تحوي المكتبة "Connectivity" التابعة لـ الحزمة "QtMobility 1.2" على واجهة من المناهج الخاصة بالإتصال مع الأجهزة المحليّة المضمّنة داخل الجهاز الحامل للتطبيق , حيث تدعم مناهج هذه الحزمة التفاعل مع الـ "Bluetooth" و مع " Near Field Communication" إختصارا "NFC" .

تقنية الـ "Bluetooth" تستخدم لتراسل المعطيات لاسلكيّا بين جهازين , يجب أن يكون الجهازين ضمن مجال قطره "100 متر و أقل" , أقصى حد لمعدل نقل البيانات بين الجهازين هو "2.1 ميغا بت / ثانية" .

تقنية "NFC" تستخدم من أجل تراسل المعطيات لاسلكيّا بين جهازين , يجب أن تكون البعد بين الجهازين واحد سم "1 سم" تقريبا أي متلاصقين , معدل نقل البيانات بين الجهازين هو "424 كيلو بت / ثانية" .

المنصات التي تدعم تقنية "NFC" هي :

1- **Maemo6**
2- **Symbian^3**
3- **Qt Simulator**

عندما نود استخدام الحزمة "Connectivity" يتوجب علينا أن نضيف ضمن ملف المشروع الرمّاز التالي :

```
CONFIG += mobility

MOBILITY += connectivity
```

نستخدم هذا الرمّاز لـ تفعيل استخدام مكاتب الحزمة "QtMobility" الذي سنتكلم عنها ضمن الفصل التاسع و الفصل الخامس عشر .

و في حال أردنا أن يعمل التطبيق الذي يستخدم الحزمة "Connectivity" على منصة "Symbian" يتوجب ، نضيف الرمّاز التالي ضمن ملف المشروع :

```
symbian:TARGET.CAPABILITY += LocalSerxvices UserEnvironment
```

قابليّات وصول منصة Symbian **, من أجل أن تسمح منصة "Symbian" الوصول إلى خدمات و أحهزة الجهاز المحلي , أيضا سنتكلم عنها ضمن الفصل التاسع و الفصل الخامس عشر .**

إلى هنا نكون قد إنتهينا من الشرح لإساسيات برمجة التطبيقات الشبكيّة في "Qt" .

❖ خلاصة الفصل :

تكلمنا في هذا الفصل عن كيفيّة برمجة أربع انواع من البروتوكولات ضمن Qt , الأول HTTP والذي يكون البروتوكول القياسي للشبكة العنكبوتية لإرسال الطلبات إليها و استقبال الإستجابة منها , الثاني FTP المستخدم في نقل الملفات , الثالث TCP و حيث أنه الأكثر استخداما و يعد من البروتوكولات الموجهة, يوجد الكثير من البروتكولات المبني عليه لما فيه من وثوثقيّة , الرابع UDP البروتوكول الأكثر سرعة , و لكن بالمقابل هو عديم الإتصال و غير موثوق و غير موجه , أخيرا تكلمنا عموما عن تقنية "NFC" و عن "Bluetooth" التابعة للمكتبة "Connectivity" .

لننقل إلى فصل أساسيات برمجة "OpenGL" في Qt .

الفصل السابع

أساسيّات OpenGL في Qt

❖ مقدمة Intro :

OpenGL هي مكتبات برمجية تستخدم للتخاطب مع العتاد الرسومي للحاسوب , لتشكيل تصاميم ثلاثية الأبعاد , طبعا تتمتع باستقلالية عن العتاد الصلب للحاسوب , اي موجهة لعدة منصات مختلفة.

يمكنك من خلال هذه المكتبات رسم النقاط و الخطوط و المضلعات و المنحنيات و أسطح NURBS و باستخدام المفاهيم الرياضية و الفيزيائية من الهندسة الفراغية إلى علم الحركة الخ.. تستطيع رسم المجسمات الثلاثية الأبعاد و تحريكها .

بعد تطبيق العمليات الرياضية باستخدام OpenGL يوجد مرحلة تدعى التصيير "Render" و ذلك من أجل حساب هذه العمليات و من ثمّ رسمها باستخدام العتاد الرسومي و أخيرا إظهارها على شاشة العرض أو خرج رسومي آخر كـ الطابعة .

❖ OpenGL في Qt :

الملف "gl.h" و الملف "glu.h" في لغة C++ القياسية في Qt الملف "qgl.h" الذي يحوي الصف "QGLWidget" .. لا تنسى

❖ الصف QGLWidget :

كائن مرئي مهمته تصيير رسوميات "OpenGL" , مشتق من الصف "QWidget" , يتم العمل مع رسوميات "OpenGL" من خلال تهيئة إعدادات التصيير و إعداد منفذ العرض و أخيرا تصيير المشهد ككل ,

انظر المناهج المحميّة و الإفتراضيّة التالية :

1- void QGLWidget::initializeGL ()

مهمته تهيئة اعدادات التصيير , يستدعى قبل اتسدعاء المناهج "resizeGL" و "paintGL"

2- void QGLWidget::resizeGL (int width, int height)

مهمته إعداد منفذ العرض , يستدعى عند كل تغيير لحجم النافذة

3- void QGLWidget::paintGL ()

مهمته تصيير كامل مشهد رسوميات OpenGL

أما بالنسبة لقراءة دخل أجهزة الفأرة و لوحة المفاتيح ضمن الصف "QGLWidget" نستخدم مناهج الصف "QWidget" الخاصة باستقبال أحداث الفأرة و لوحة المفاتيح .

❖ المشروع Cube :

سوف نشرح أساسيّات OpenGL في Qt من خلال بناء مثال متكامل وهو الآتي :

مجسم مكعب على وجهه السفلي يظهر مكعب بشكل نافر و أيضا على الوجه العلوي منه , ألون أوجه المكعب مختلفة عن بعضها للتمييز بين الأوجه .

أنشأ مشروع "Qt" فارغ و سمّه "Cube" , ثمّ اذهب إلى ملف المشروع "" و أضف السطر البرمجي التالي :

```
QT += opengl
```

أضفنا حزمة "OpenGL" لإتاحة استخدام مكاتبها ضمن المشروع الحالي , الآن أضف ملف للمشروع "Cube" يدعى "main.cpp" و الذي سيحوي على الكتلة الرئيسيّة للتطبيق و التي تدعى "main" , اكتب الرمّاز التالي داخل الملف "main.cpp" :

```
#include <QApplication>

#include "glwidget.h"


int main(int argc, char** argv){

   QApplication app(argc, argv);

   glWidget widget;

   widget.show();

   return app.exec();
```

```
}
```

مثلما تلاحظ يوجد استدعاء للصف "glWidget" و الذي سيكون صف فرعي من الصف "QGLWidget" , أنشأ صف يدعى "glWidget" مشتق من الصف "QGLWidget" و نمط المعلومات هو "Inherits QWidget" , داخل الملف "glwidget.h" , ضمّن ملفات الترويسة التالية :

```
#include <QGLWidget>

#include <QTimer>

#include <QMouseEvent>

#include <QKeyEvent>
```

ثمّ عرّف الآتي :

ضمن القسم المحمي عرّف المناهج الإفتراضيّة الخاصة بتهيئة و تصيير رسوميات OpenGL :

```
protected:

  void initializeGL();

  void paintGL();

  void resizeGL(int w, int h);

  void mousePressEvent(QMouseEvent *);

  void keyPressEvent(QKeyEvent *);
```

المنهج "mousePressEvent" خاص باستقبال أحداث النقر بزر الفأرة , أما المنهج "keyPressEvent" خاص باستقبال أحداث الضغط على مفتاح من لوحة المفاتيح , عرّف ضمن القسم المحمي المنهج الخاص بـ إنشاء مجسم مكعب , ثمّ المستقبل الخاص بتحريك المجسم و الذي سوف يتصل بالحدث "timeout" لكائن المؤقت "QTimer" :

```
public slots:

  void move();
```

أخيرا التصريح عن الكائنات التي سوف نستخدمها :

```
private:

   QTimer* timer;

   GLfloat rotF ;
```

أولا تصريح عن كائن المؤقت الذي ستكون مهمته تغيير قيمة المتحول "rotF" من أجل عملية تدوير الكائن , المتحول "rotF" من نمط "GLfloat" فهو من نمط عدد ذي فاصلة عائمة , تمّ التصريح عنه ضمن التروسية "gl.h" كالآتي :

```
typedef  float        GLfloat
```

لننتقل إلى ملف التحقيق "glwidget.cpp" , أولا عملية تهيئة إعدادات تصيير رسوميات OpenGL , حقق المنهج الإفتراضي و المحمي "initializeGL" :

```
void glWidget::initializeGL(){


   glClearColor(0.0, 0.0, 0.0 , 0.0);

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glLoadIdentity();


   glEnable(GL_DEPTH_TEST);


}
```

المنهج "glClearColor" فهو من المناهج الأساسيّة ضمن "OpenGL" موجود داخل التروسية "gl.h" , مهمة هذا المنهج هي تحديد لون التهيئة للنافذة "بيئة الرسم" , وسطائه هم :

```
glClearColor( GLclampf red, GLclampf green, GLclampf blue, GLclampf
alpha )
```

الألوان الأساسّة "RGBA" مع ألفا ,يجب أن تكون قيم الوسطاء ضمن المجال 0 – 1 , النمط "GLclampf" معرّف ضمن "gl.h" كالآتي :

typedef float GLclampf

المنهج "glClear" يقوم بمسح النافذة "بيئة الرسم" باللون المحدد , بالنسة لوسيطه هو :

glClear(GLbitfield mask)

فهو الذواكر التي سوف يقوم بمسحها , تعرف النمط "GLbitfield" :

typedef unsigned int GLbitfield

مررنا للمنهج "glClear" , معرّف ذاكرة الألون "GL_COLOR_BUFFER_BIT" , و معرّف ذاكرة العمق "GL_DEPTH_BUFFERBIT" , بالتالي سوف يقوم بمسح جميع البيانات المحفوظة ضمن ذاكرة الألوان و ذاكرة العمق , من ثمّ يضع اللون الأسود لبيئة الرسم "النافذة" , استدعينا المنهج "glLoadIdentity()" و الذي تتلخص مهمته في تهيئة المصفوفة الحالية للتحويلات "الدوران - التقييس" إلى مصفوفة 4X4 واحديّة النمط أي جميع قيم خاناتها صفر ماعدا القطر الرئيسيّ لها قيم خاناته واحد ,لإنه عادة يتم ضرب قيم التحويلات بالمصفوفة الحالية بعدها يحفظ الناتج بالمصفوفة المحددة , سنرى في الفقرات الآتية كيف و أين يتوجب أن نستخدم هذا المنهج , أخيرا أمر التفعيل لواصف معين "glEnable" و المنهج العاكس له هو "glDisable" , وسيطه من نمط "GLenum" وهو :

typedef unsigned int GLenum

بما أننا نريد أن نرسم مجسم ثلاثي البعد فيجب أن نخبر البرنامج بذلك ليهيئ و يحدث ذاكرة العمق , وذلك يتم من خلال تمرير الثابت "GL_DEPTH_TEST" لـ المنهج "glEnable" , الآن لنحقق المنهج "resizeGL" من أجل تهيئة منفذ العرض عند تنفيذ التطبيق و عند تغيير حجم نافذته :

```
void glWidget::resizeGL(int width, int height)

{

  glViewport(0,0,width,height);

  gluLookAt(0.0, 0.0, 0.1, 0.0 , 0.0, 0.0, 0.0, 1.0, 0.0);

  glMatrixMode(GL_MODELVIEW);

}
```

المنهج "glViewport" يحدد موقع و حجم منفذ العرض , أما المنهج "glMatrixMode" فهو لـ تحديد نمط المصفوفة الحالية لتطبق الأوامر التي تليها على هذه المصفوفة , يوجد ثلاثة أنماط , هنا سنذكر اثنين فقط وهم :

1- GL_PROJECTION

مصفوفة الإسقاط : تمثّل نمط عدسة الكاميرا , تستخدم لـ التقريب و التبعيد و تحديد مجال الرؤيا

2- GL_MODELVIEW

مصفوفة النمذجة و العرض : النمذجة توضيع و تحويل النموذج "المجسم" , العرض توضيع و توجيه الكاميرا

المنهج الخاص بـ إعداد الكاميرا من توضع و تحديد مركز العدسة هو "gluLookAt" الموجود ضمن الترويسة "glu.h" , وسطاء المنهج :

gluLookAt (GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ, GLdouble centerX, GLdouble centerY, GLdouble centerZ, GLdouble upX, GLdouble upY, GLdouble upZ)

الوسطاء "eyeX – eyeY - eyeZ" تحديد موضع الكامير , أما الوسطاء " centerX – centerY - centerZ" تحدد النقطة التي تكرز عليها الكامير "عدسة الكاميرا" , الوسطاء "upX – upY - upZ" فهو لتحديد شعاع الإتجاه الرأسي , وضعنا قيمة الوسيط "upY" واحد , أي الإتجاه الرأسي للواقع الإفتراضي الذي نشكله هو محور الـ "Y" العمودي , لننتقل إلى تحقيق المنهج "paintGL" و التي مهمته رسم المكعب و تطبيق تحويل التدوير عليه :

```
void glWidget::paintGL(){


  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glLoadIdentity();


  glRotatef(rotF,0.1,0.5,0.1);


  glScalef(0.5,0.5,0.5);
```

```
   createCube(0,0,0);
   glScalef(0.5,0.5,0.5);

   createCube(0.5,1.5,0.5);
//   createCube(0.0,-2.0,0.0);

   glLoadIdentity();
   glScalef(0.5,0.5,0.5);
   glScalef(0.5,0.5,0.5);
   glRotatef(rotF,0.1,0.5,0.1);
   createCube(0.5,-0.5,0.5);

}
```

تطبيق تحويل التدوير بعكس عقارب الساعة على المجسم هنا ستطبق عملية التدوير على مجسم المكعب باستخدمنا لـ المنهج "glRotatef" , وسطاء المنهج :

```
glRotatef( GLfloat angle, GLfloat x, GLfloat y, GLfloat z )
```

الوسيط "angle" زاوية التدوير مقدر بالدرجة , حيث ستتم عملية الدروان حول " x – y - z" المعطاة, منهج التقييس "glScalef" استخدمناه من أجل إعادة تقييس المجسم , وسطاءه :

```
glScalef( GLfloat x, GLfloat y, GLfloat z )
```

حيث ستتم عملية التقييس بالنسبة لـ المحاور الثلاث " x – y - z " , في رمّازنا الحالي صغّرنا المجسم إلى منتصف حجمه , وذلك بسبب العملية التي قام بها منهج التقييس وهي ضرب مصفوفة جملة إحداثيّات المجسم بـ القيم المعطا " x – y - z " و التي تساوي "0.5" , يستخدم المنهج "glScalef" من اجل تكبير و تصغير و قلب مجسم , استدعينا المنهج "createCube" و الذي سوف نحققه , مهمة هذا المنهج إنشاء مجسم مكعب بموقع محدد , بعدها استدعينا منهج التقييس من أجل إعادة التقييس إلى نصف قيم المصفوفة السابقة و

ذلك لـ إنشاء مكعب في منتصف سطح المكعب السابق , بحيث يكون نصف حجم المكعب الأب "السابق" و بحيث يظهر منه نصف مجسمه :

```
glScalef(0.5,0.5,0.5);

createCube(0.5,1.5,0.5);
```

لاحظ الأحداثيّات الممرة لـ المنهج "createCube" نسبة لإحداثيّات مجسم المكعب الأب , الإحداثيّات هنا تمثل قيم إنسحاب المكعب على المحاور الثلاث , لإنشاء مكعب بمواصفات المكعب السابق ماعدا أنه يوجد على أرضيّة المكعب الأب , انظر الرمّاز :

```
createCube(0.0,-2.0,0.0);
```

لا حظ أنه مررنا قيم الإنسحاب نسبة لـ المكعب السابق و ليس المكعب الأب , لإنه أعدنا تقييس "تحويلات النمذجة" لـ المجسم و بالتالي تغير نقطة المركز لنظام الإحداثيّات لتصبح النقطة الأولى في مجسم المكعب السابق الموجود على سطح المكعب الأب , حصل ذلك لأننا لم نستعدي المنهج "glLoadIdentity" , بعد نهايتك من كتابة رمّاز المشروع جرّب أن تزيل الرمّاز الذي ينشأ المكعب الموجود على الوجه السفلي للمكعب الأب و هو :

```
createCube(0.0,-2.0,0.0);
```

و تكتب بدله الرمّاز التالي :

```
glLoadIdentity();

glScalef(0.5,0.5,0.5);

glScalef(0.5,0.5,0.5);

glRotatef(rotF,0.1,0.5,0.1);

createCube(0.5,-0.5,0.5);
```

لا حظ أننا أنشئنا المكعب الابن على الوجه السفلي للمكعب الأب نسبة لـ نقطة مرزو نظام إحداثيّات المكعب الأب و التي هي نفسها نقطة مركز منفذ العرض الذي قد هيئناه ببداية كتابتنا لرمّاز المشروع , حقق المنهج الخاص بإنشاء مجسم مكعب كالآتي :

```
void glWidget::createCube(const GLfloat &sx, const GLfloat &sy, const
GLfloat &sz){


glTranslatef(sx,sy,sz);
```

```
//floor
glBegin(GL_POLYGON);
glColor3f(1,0,0);
glVertex3f(0.0,0.0,0.0);
glVertex3f(0.0,0.0,1.0);
glVertex3f(1.0,0.0,1.0);
glVertex3f(1.0,0.0,0.0);
glEnd();

//face 1
glBegin(GL_POLYGON);
glColor3f(0,1,0);
glVertex3f(0.0,0.0,0.0);
glVertex3f(1.0,0.0,0.0);
glVertex3f(1.0,1.0,0.0);
glVertex3f(0.0,1.0,0.0);
glEnd();

//face 2
glBegin(GL_POLYGON);
glColor3f(0,0,1);
glVertex3f(0.0,0.0,0.0);
```

```
glVertex3f(0.0,1.0,0.0);
glVertex3f(0.0,1.0,1.0);
glVertex3f(0.0,0.0,1.0);
glEnd();

//face 3
glBegin(GL_POLYGON);
glColor3f(1,1,0);
glVertex3f(0.0,0.0,1.0);
glVertex3f(1.0,0.0,1.0);
glVertex3f(1.0,1.0,1.0);
glVertex3f(0.0,1.0,1.0);
glEnd();

//face 4
glBegin(GL_POLYGON);
glColor3f(1,0,1);
glVertex3f(1.0,0.0,0.0);
glVertex3f(1.0,1.0,0.0);
glVertex3f(1.0,1.0,1.0);
glVertex3f(1.0,0.0,1.0);
glEnd();
```

```
//roof
glBegin(GL_POLYGON);
glColor3f(0,1,1);
glVertex3f(0.0,1.0,0.0);
glVertex3f(0.0,1.0,1.0);
glVertex3f(1.0,1.0,1.0);
glVertex3f(1.0,1.0,0.0);
glEnd();

}
```

أولا استخدمنا المنهج الخاص بالإنسحاب على المحاور الثلاث وهو "glTranslatef" , وسطاءه :

```
glTranslatef( GLfloat x, GLfloat y, GLfloat z )
```

سوف يقوم بتحريك المجسم على المحاور الثلاث من موضعه الحالي إلى القيم المعطاء لوسطاءه ,عندما نود رسم شكل هندسي ما باستخدام توضع رؤوسه , يتوجب أن نستدعي المنهج "glBegin" قبل البدأ بعملية تحديد مواقع الرؤوس لهذا الشكل و عند نهايتنا من الرسم يتوجب إستدعاء المنهج "glEnd" , المنهج "glBegin":

```
glBegin( GLenum mode )
```

يجب أن نمرر لوسيطه نوع الشكل الهندسي الذي نريد رسمه مضلع , مثلث , خط الخ... , انظر الجدول يحوي أهم الأشكال الهندسية المعرّفة :

مجموعة من النقاط	GL_POINTS
كل اثنين من الرؤوس يشكلان خط مستقيم	GL_LINES
مجموعة من خطوط المستقيم المتصلة	GL_LINE_STRIP
مجموعة من خطوط المستقيم المتصلة مع وصل النقطة الأخيرة "الرأس الأخير"	GL_LINE_LOOP

بالنقطة الأولى

كل ثلاث من الرؤوس يشكلان مثلث GL_TRIANGLES

تشكيل مضلع حسب عدد الرؤوس GL_POLYGON

لإعطاء قيمة توضع الرأس نستخدم المنهج :

```
glVertex3f( GLfloat x, GLfloat y, GLfloat z )
```

وسطاءه هم النقطة "الرأس" على المحاور الثلاث " x – y - z " , بالنسبة لتغيير اللون نستخدم المنهج :

```
glColor3f( GLfloat red, GLfloat green, GLfloat blue )
```

لاحظ كيف تمّ رسم المكعب بوساطة رسم 6 أوجه اثنان منهما يشكلان الوجه العلوي و الوجه السفلي , و تحديد مواقع 8 من النقاط "الرؤوس" , الآن من أجل البدأ بعملية التحريك اكتب الرمّاز الخاص بـ إنشاء مؤقت من أجل تنفيذ المنهج "move" عند كل مدة زمنية صغيرة ,و الذي سـ يغير قيمة المتحول "rotF" و بالتالي دوران مجسم المكعب , اكتب الرمّاز داخل كتلة البناء :

```
timer = new QTimer(this);

connect(timer,SIGNAL(timeout()),this,SLOT(move()));

timer->setInterval(10);

timer->start();
```

حقق المستقبل "move" كالآتي :

```
void glWidget::move(){

  rotF == 360.0 ? rotF = 0.0 : rotF += 1.0 ;

  updateGL();

}
```

استدعينا المقبس الإفتراضي "updateGL" الذي سوف يستدعي المنهج الإفتراضي "glDraw" و الذي بدوره ينفذ المنهج "paintGL" , هذه المناهج الثلاث تابعة للصف

"QGLWidget" , بما أننا كتبنا ضمن كتلة المنهج "paintGL" السطر البرمجي الخاص بتدوير المجسم :

```
glRotatef(rotF,0.1,0.5,0.1);
```

و الذي مررنا لوسيط الزاوية المتحول "rotF" بالتالي سيتم تدوير المكعب عند كل استدعاء للمقبس "move" , ضمن الهدّام اكتب الرمّاز الخاص بحذف حدث كائن المؤقت :

```
glWidget::~glWidget(){delete timer;}
```

لننتقل إلى كتابة الرمّاز المتعلق بـ الأحداث الداخليّة , نريد أن تتوقف مؤقتا عملية دوران مجسم المكعب عند الضغط على مفتاح مسطرة "Space" , وإعادة تنفيذ عملية الدوران عند الضغط عليها مجددا , حقق المنهج الإفتراضي "KeyPressEvent" التابع للصف "QWidget" :

```
void glWidget::keyPressEvent(QKeyEvent *event){

  if(event->key() == Qt::Key_Space)

    timer->isActive() ? timer->stop() : timer->start() ;

}
```

فقط من خلال استدعاء المنهج "stop" لكائن المؤقت من أجل إيقاف عملية تغيير قيمة المتحول "rotF" , استدعاء المنهج "start" الخاص بتنفيذ عمل المؤقت , أخيرا نود عند الضغط بالزر الأيسر للفأرة على نافذة العرض ان يتم إلتقاط صورة لـ مجسم المكعب و من ثمّ حفظها , نستطيع أن نستخدم أحد المنهجين التاليين :

```
QImage grabFrameBuffer(bool withAlpha = false)
```

حيث سيعيد الإطار الحالي الموجود بالذاكرة الخاصة به , أي الإطار المهيئ للعرض , نمط القيمة المعادة هي "QImage"

```
QPixmap renderPixmap(int w = 0, int h = 0, bool useContext = false)
```

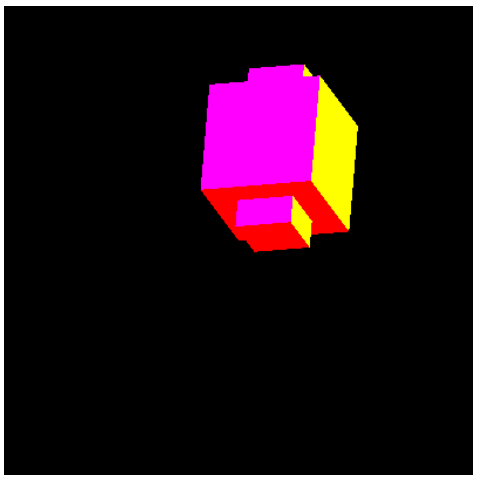
سيعيد صورة المشهد المعروض حاليا , نمط القيمة العائدة "QPixmap"

انظر الرمّاز التالي :

```
void glWidget::mousePressEvent(QMouseEvent *event){

  if(event->button() == Qt::LeftButton){
```

```
    QImage image = grabFrameBuffer(true);
    image.save("imgGrab.bmp");
    QPixmap pixmap = renderPixmap(this->width()-250,this->height()-
400);
    pixmap.save("imgRender.bmp");
  }
}
```

نفذ التطبيق و اختبر النتيجة , انظر الشكل (7.1) يعرض مشهد من مجسم المكعب :



الشكل 7.1

ستجد في القرص المرفق المشروع "Cube" كاملا .

❖ خلاصة الفصل :

تعد برمجة رسوميات الحاسوب من البرمجة الصعبة نوعا ما و ذلك لتنوع العلوم المستخدمة فيها أولا ثانيا لصعوبة تشكيل تصميم معقد من نقطة و مستقيم و مضلع و سطح .

بعد أن أنهينا أسس البرمجة الرسومية باستخدام OpenGL دعنا ننتقل إلى الفصل الأخير من قسم Qt C++ framework و الذي يتكلم حول كيفية برمجة التطبيقات المحمولة بوساطة Qt.

الفصل الثامن

تطبيقات Qt المحمولة

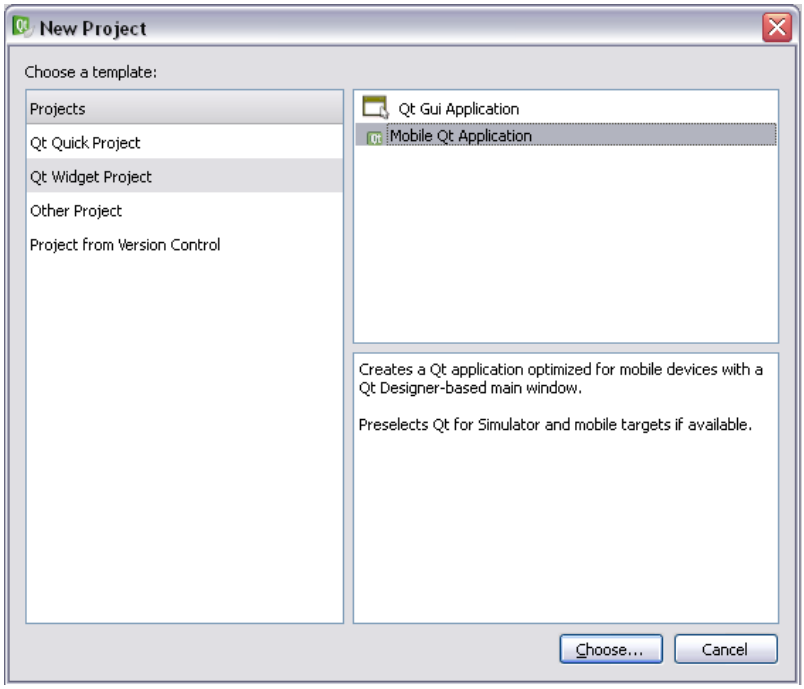
❖ مقدمة Intro :

أصبح العالم الحالي رقمي بامتياز أي النسبة العظمة من سكان العالم يستخدمون بشكل يومي الأحهزة المحمولة و الحاسوب , ونسبة لأهمية و كثرة استخدام الأجهزة الذكية المحمولة كـ iPhone و أنظمة Android التي تحضنها عدة أجهزة كـ Samsung Galaxy و Sony عدة فصول من هذا الكتاب حول كيفية برمجة التطبيقات المحمولة باستخدام Qt , طبعا و هذا ما يتميز به إطار عمل Qt , أي اكتب الرمّاز لمرة و احدة و انشر على عدة منصات .

❖ المشروع Mobile Qt Application :

من أجل بناء تطبيقات Qt ذي أمثليّة للأجهزة الذكيّة كـ أجهزة الجوال الحديثة , يتوجب أن ننشأ مشروع "Mobile Qt Application" , المعالج الذكي الخاص به يهيئ التطبيق لـ تتناسب نافذته مع حجم شاشة الحهاز المحمول عليه, ثانيا يعد التشكيلات الجانبية لمشروع التطبيق كـ وضع معرّف لمنصة "Symbian" و دعم الحزمة "QtMobility" و تفعيل قابليّات الوصول "CAPABILITY" لخدمات الجهاز المتاحة و تهيئة التطبيق للنشر الخ..

لإنشاء مشروع "Mobile Qt Application" من "New Project" نختار القالب " Qt Widget Project" ثمّ المشروع "Mobile Qt Application" , انظر الشكل (9.1) :



الشكل 9.1

- **جولة سريعة داخل ملفات القالب " Mobile Qt Application" :**

أولا ملف المشروع , نبدأ بالسطر البرمجي :

```
symbian:TARGET.UID3 = 0xEA3B4B00
```

هذا السطر خاص بوضع معرّف فريد للتطبيق عند تنفيذه على منصة "Symbian", بوساطة المعرّف يعطى التطبيق المحدد تسجيل "Symbian"

السطر البرمجي :

```
symbian:TARGET.CAPABILITY += NetworkServices
```

القابليّات , نقصد بها سماحية الوصول إلى خدمات محددة داخل منصة "Symbian" , كـ قراءة دخل جهاز و إعطاء خرج لجهاز , و كـ سماحية وصول التطبيق إلى الخدمات الشبكيّة الخ ...

السطر البرمجي :

```
CONFIG += mobility
```

MOBILITY +=

من اجل تضمين مكاتب الحزمة "QtMobility" داخل التطبيق , نضيف داخل المتحول "MOBILITY" المكاتب التي نود استخدامها داخل التطبيق كـ المكتبة الخاصة بتشغيل الوسائط المتعددة "multimedia" , تحوي الحزمة "QtMobility" على مجموعة من المكاتب الغنيّة بـ توابع "API" المتعددة الأفعال كـ التوابع الخاصة بالوصول إلى رسائل الجوال و التحكم بجهاز الـ "Bluetooth" الخ.. سنتكلم لاحقا عن الحزمة "QtMobility" بتفصيل أكثر

السطر البرمجي :

```
include(deployment.pri)

qtcAddDeployment()
```

إعداد المشروع من أجل النشر , و ذلك من خلال تضمين الملف "deployment.pri" الذي يحوي التشكيلات الجانبيّة الخاصة بتهيئة التطبيق للنشر

داخل الملف "mainwindow.cpp" :

المنهج "setOrientation" خاص بتحديد توجيه نافذة التطبيق , وذلك من خلال اختبار المنصة الحامل للتطبيق و وضع التوجيه المناسب المنصة .

الرمّاز :

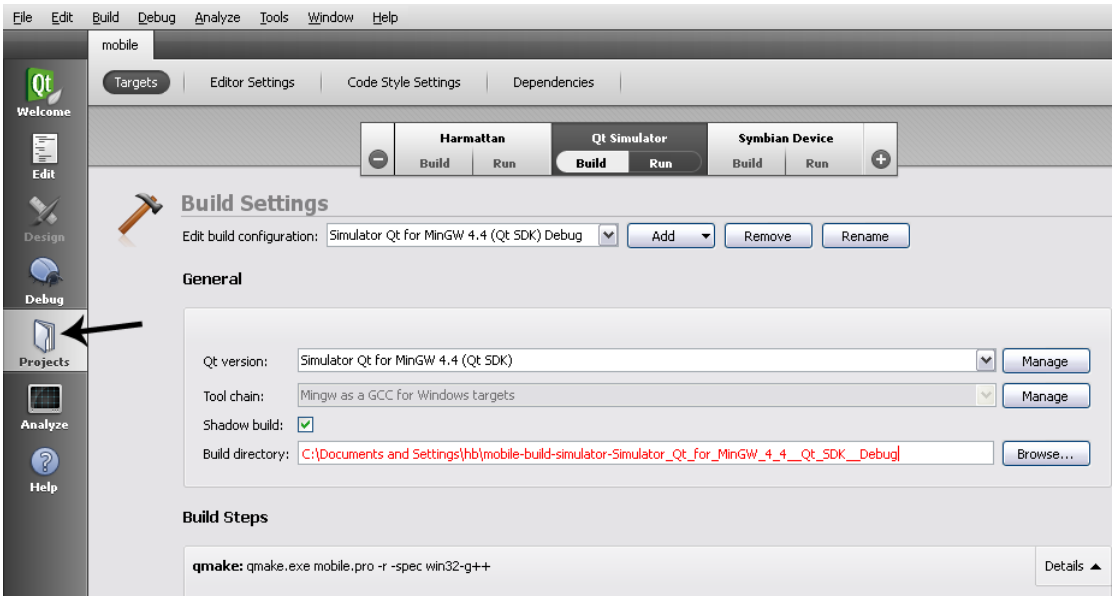
```
#if defined(Q_OS_SYMBIAN) || defined(Q_WS_SIMULATOR)
   showFullScreen();
#elif defined(Q_WS_MAEMO_5)
   showMaximized();
#else
   show();
#endif
```

الرمّاز السابق موجود داخل المنهج "showExpanded" , مهمة الرمّاز هي اختبار المنصة الحالية الحاملة للتطبيق , في حال كانت المنصة هي "Symbian" ستظهر نافذة التطبيق

على كامل مساحة الشاشة , في حال كانت المنصة هي "Maemo" ستظهر نافذة التطبيق بحالة التكبير , عدا الحالتين ستظهر النافذ بحالتها الإفتراضيّة

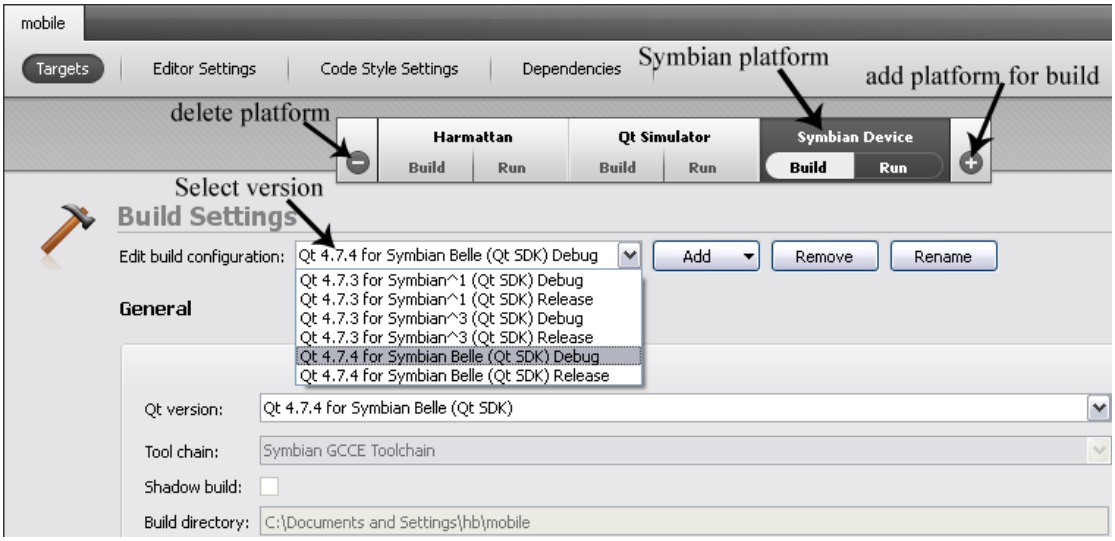
- النافذة "Projects" داخل "Qt Creator" :

هذه النافذة مختصة بـ تهيئة إعدادات البناء للتطبيق , حيث نستطيع من خلالها تحديد المنصة التي نود تشغيل التطبيق عليها و إضافة و تغيير اوامر تجميع المشروع , نظهر النافذة "Projects" من خلال النقر على التبويب "Projects" الموجود على يسار نافذة " Qt Creator" كما في الشكل (9.2) :



الشكل 9.2

من اجل إضافة منصة ليعمل عليها التطبيق يتوجب ان نضغط على زر "Add" الذي يأخذ شكل "+" , و في حال أردنا أن نزيل منصة نضغط على زر "Remove" الذي يأخذ شكل "- " , لتجميع و تنفيذ التطبيق على منصة موجودة ضمن مجموعة منصات يتوجب علينا فقط النقر على المنصة المرادة , كما في الشكل السابق يوجد ثلاث منصات و هي "Hamattan" و "Qt Simulator" و "Symbian Device" نحدد منصة من خلال النقر بزر الفأرة الأيسر ضمن لافتة المنصة , انظر الشكل (9.3) :



الشكل 9.3

- **نشر و تنقيح التطبيق على منصة Symbian :**

في حال أردنا نشر و تنفيذ التطبيق بشكل مباشر على جهاز جوال يحوي منصة "Symbian" , يتوجب فعل الآتي :

1 - **نصب التطبيق "Nokia Ovi Suite"**
2 - **صل جهاز الجوال المحدد بمنفذ الـ "USB" و انشأ اتصال به بوساطة البرنامج "Nokia Ovi Suite"**
3 - **نصب البرنامج "CODA.sis" على جهاز الجوال , يوجد عدة إصدارات للبرنامج "CODA" فـ لتصيب التطبيق "CODA" خاص بمنصة "S^3" , اذهب إلى " Qt SDK" ثمّ "Symbian^3 Qt 4.7.3" أخيرا افتح التطبيق " Install CODA (Debug Agent) on Symbian^3 device" , التطبيق "CODA" في منصة "Symbian" هو الجسر الواصل بين إطار عمل "Qt" و جهاز الجوال , حيث من دون تنصيبه على منصة "Symbian" لن تستطيع تنفيذ و تنقيح التطبيق بشكل مباشر على جهاز الجوال الحامل لمنصة "Symbian"**
4 - **ضمن "Qt Creator" احتر منصة التشغيل هي "Symbian" و ذلك من النافذة "Projects" , كما في الشكل (9.3)**
5 - **أخير حدد الخيار "Run" الموجود داخل لافتة المنصة "Symbian Device" و ذلك من أجل تنفيذ التطبيق بشكل مباشر على جهاز الجوال**

نفذ التطبيق و اختبر النتيجة

ملاحظة : عندما يتم اتصال جهاز الجوال بنجاح سوف تظهر أيقونة التشغيل مع رمز "صح"أخضر اللون كما في الشكل (9.4) :

الشكل 9.4

ملاحظة : في حال كان تطبيقنا يحوي استخداما للمكتبة "QtMobility" يتوجب أن ننصب التطبيق "QtMobility.sys" الموجود داخل الدليل "Qt SDK" ثمّ "Symbian^3 Qt 4.7.3" أو "Symbian^1 Qt 4.7.3" حسب الإصدار , و في حال كان التطبيق حاوي على استخدام للمكتبة "QtWebkit" نصب التطبيق "QtWebkit.sis"

- **تجميع تطبيق sis لمنصة Symbian :**

عندما نريد أن يكون خرج التجميع للمشروع هو ملف تنفيذي لمنصة "Symbian" ذي لاحقة "sis" نستطيع ذلك من خلال اتباع الخطوات التالية :

أولا افتح موجه الأوامر الخاص بالمنصة التي تود تجميع التطبيق ليعمل عليها , مثال منصة "S^3" نصل إلى موجه الأوامر الخاص بها من الدليل "Qt SDK" ثمّ "Symbian^3 Qt 4.7.3" بعدها نفتح الملف الدفعي "Qt 4.7.3 for Symbian^3 Command Prompt "

ثاينا ندخل إلى مسار المشروع المراد تجميعه و ذلك بوساطة الأمر "cd" ملحق بـ مسار دليل المشروع

ثالثا اكتب الأمر "qmake" لتجميع المشروع

رابعا اكتب الأمر "make sis" وذلك من اجل بناء التطبيق بصيغة "sis"

اذهب إلى دليل المشروع سوف تجد الملف التنفيذي باسم المشروع ذي لاحقة "sis"

- **تنفيذ التطبيق على برنامج المحاكي Simulator الخاص بـ تطبيقات Qt :**

في حال أردنا تنفيذ التطبيق و تنقيحه على جهازنا المحلي من دون وصل جهاز خارجي , فـ نستطيع ذلك من خلال تحديد منصة برنامج المحاكاة الخاص بتنفيذ و اختبار تطبيقات Qt المحمولة , برنامج المحاكاة هو تمثيل لجهاز الجوال على الآلة المحلية , فيسمح لك التحكم

بشكل كامل بـ خدمات جهاز الجوال اللإفتراضي كـ معدل شحن البطارية و الموقع الجغرافي الحالي و حساس البوصلة الخ..

من اجل تنفيذ التطبيق المحمول على برنامج المحاكاة فقط عليك تحديد المنصة "Qt Simulator"

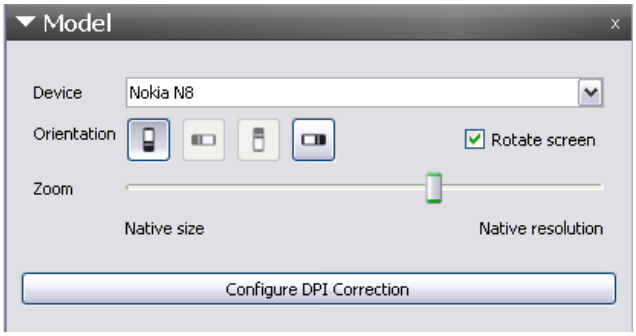
شكل برنامج المحاكة الممثل لجهاز "Nokia N8" , الشكل (9.5) :

الشكل 9.5

- **تبويبات برنامج المحاكاة Simulator :**

يوجد داخل برنامج المحاكاة ستة عشر تبويب كل منها يحوي على مجموعة من الأوامر و الخصائص الخاصة بـ التعديل على التشكيلات الجانبية للـ جوال الإفتراضي و مواصفاته , نستطيع إظهار صفحة التبويب إما من زر القائمة "Menu" أو من خلال الضغط على شعار التبويب , انظر الشكل (9.5) .

1 - التبويب Model : يحوي إعدادات التوجيه و قائمة الأجهزة المتاحة للتمثيل من قبل برنامج المحاكاة و أخيرا يحوي على إعدادات دقة شاشة العرض , انظر الشكل (9.6)



الشكل 9.6

2 - التبويب Application : يحوي على عنوان التطبيق المحدد ضمن برنامج المحاكاة و على زر للخروج من التطبيق الحالي , انظر الشكل (9.7)

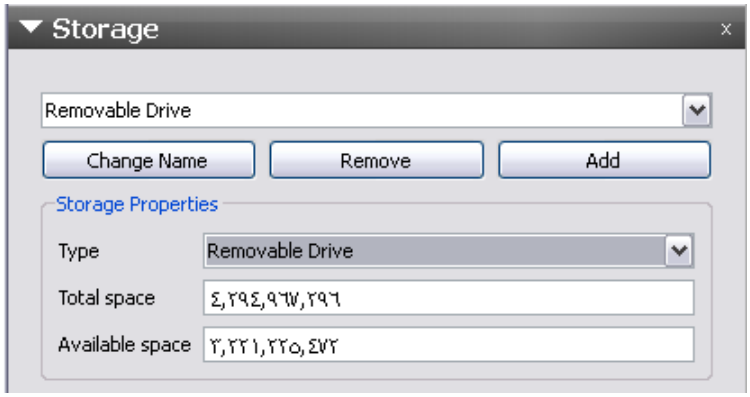
الشكل 9.7

3 - التبويب Generic : يحوي على خيارات خاصة بـ تعديل مستوى شحن للبطارية و حالة الطاقة كـ العمل على طاقة البطارية بشكل مفرد أو العمل على طاقة الشاحن , و يحوي أيضا على خيار خاص بإختيار حالة الجهاز الحالي كـ صامت أو خارج التغطية الخ.. انظر الشكل (9.8)

الشكل 9.8

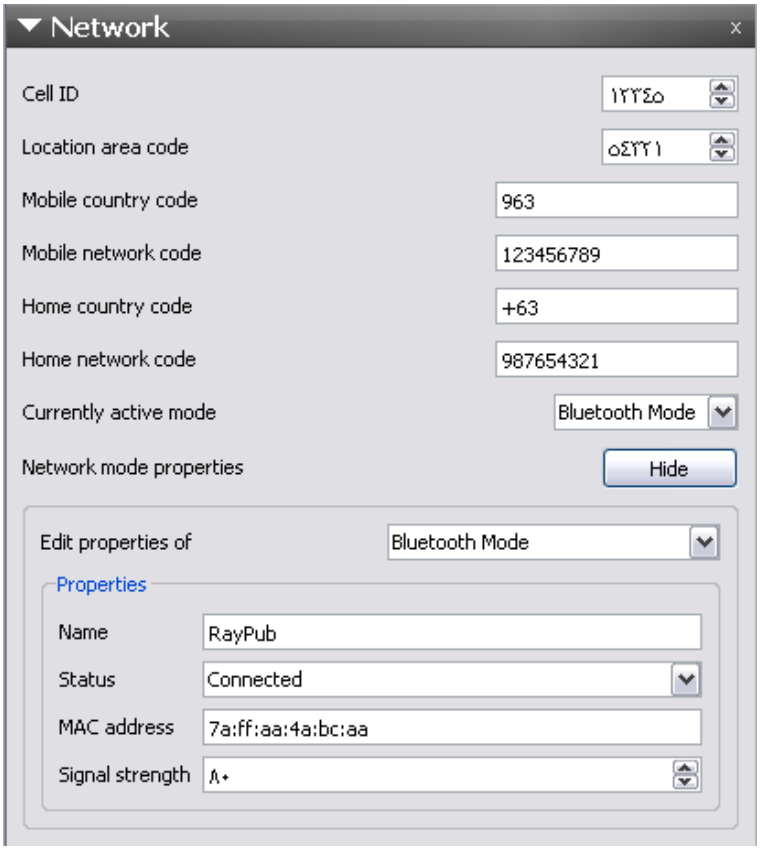
يوجد زر لإظهار خيارات متقدمة ضمن هذا التبويب , حيث يظهر لنا خيارات خاصة بـ إضافة و تحديد لغة و تفعيل و عدم تفعيل الأجهزة الملحقة بالجهاز كـ "Bluetooth" و تعديل الإضاءة و اسم التطبيق الخ ..

4 - التبويب Storage : يحوي على خيارات خاصة بـ وسائط التخزين , فنستطيع من خلاله إضافة و إزالة وسيط تخزين و تحديد نمط لهذا الوسيط أي وسيط تخزين داخلي , خارجي , أو وسيط تخزين قابل للإزالة الخ .. , ويسمح لنا أيضا أن نعديل بمساحة هذا الوسيط , انظر الشكل (9.9)



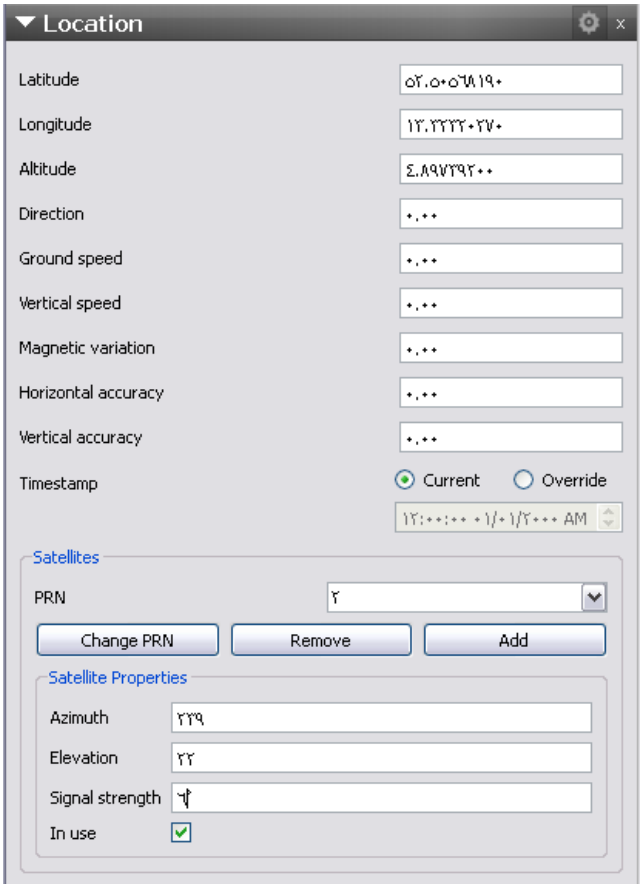
الشكل 9.9

5 - التبويب Network : خاص بالتعديل على جميع إعدادت شبكة العمل , كـ تحديد نمط الشبكة و التعديل على خصائص هذا النمط كـ تعديل عنوان الشبكة , و وضع رمز المنطقة الموجودين فيها الخ.. , انظر الشكل (9.10)



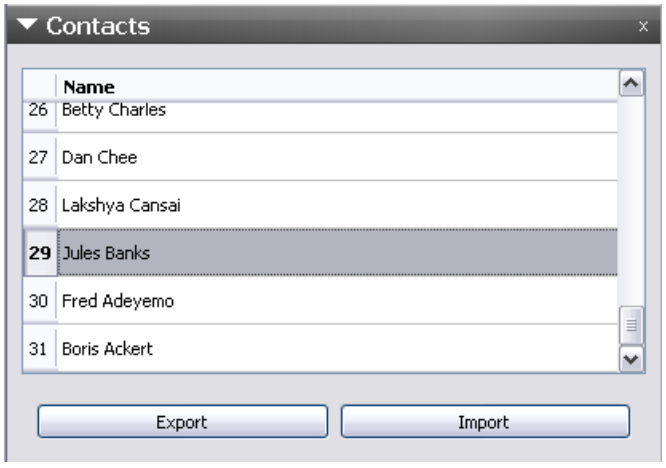
الشكل 9.10

6 - التبويب Location : يحوي جميع إعدادات الموقع , يتيح التعديل على درجتي الطول و العرض و زاوية السمت للموقع الحالي , و التعديل على وقت المنطقة المحلي , يفيد بـ اختبار التطبيقات التي تحوي على خدمة الخرائط , انظر الشكل (9.11)



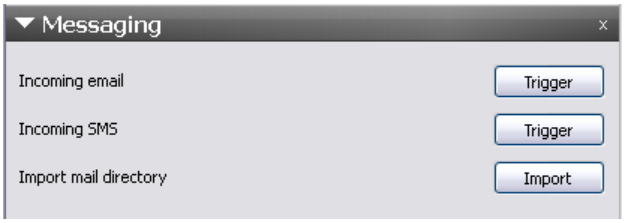
الشكل 9.11

7 - التبويب Contacts : يحوي على معلومات الإتصالات , أي تشكيلات الإتصالات الجانبيّة للمستخدمين من أسمائهم و أرقامهم الخ.. , انظر الشكل (9.12)



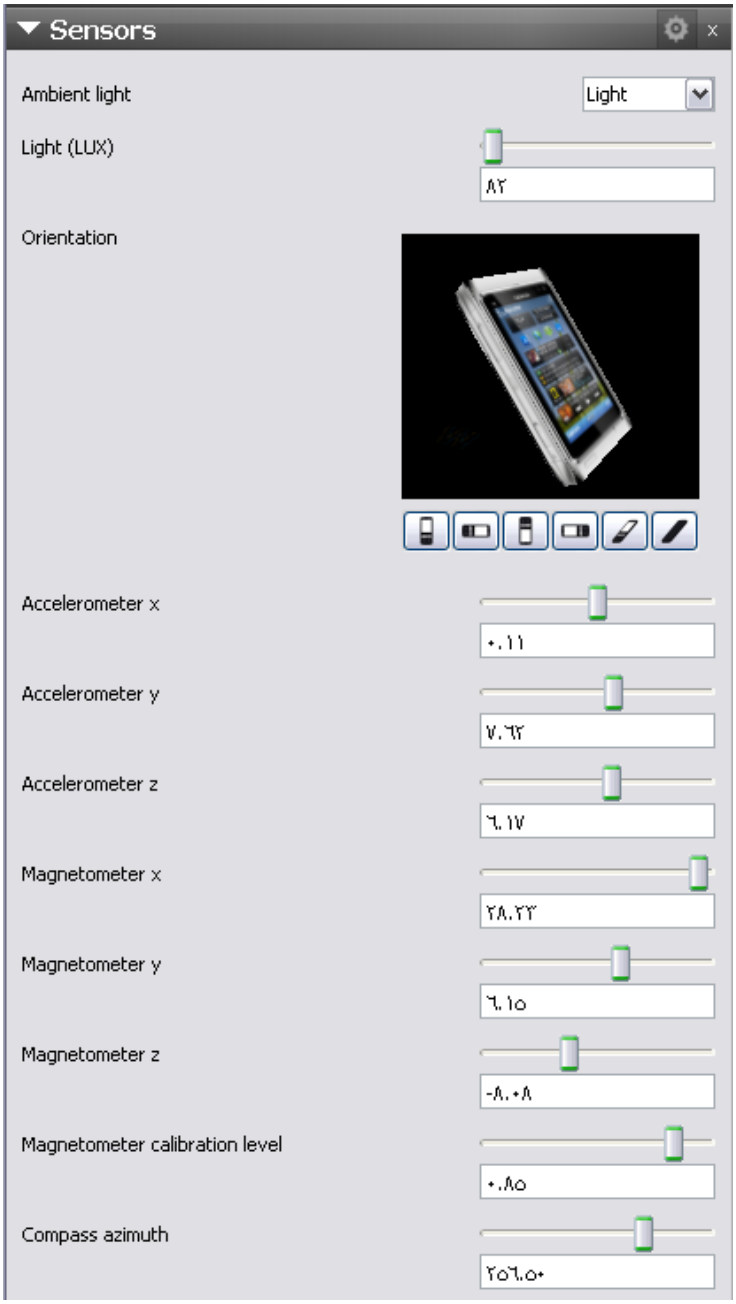
الشكل 9.12

8 - التبويب Messaging : يحوي على الأوامر الخاصة بإطلاق مراقبة الرسائل الواردة "email - sms" , انظر الشكل (9.13)



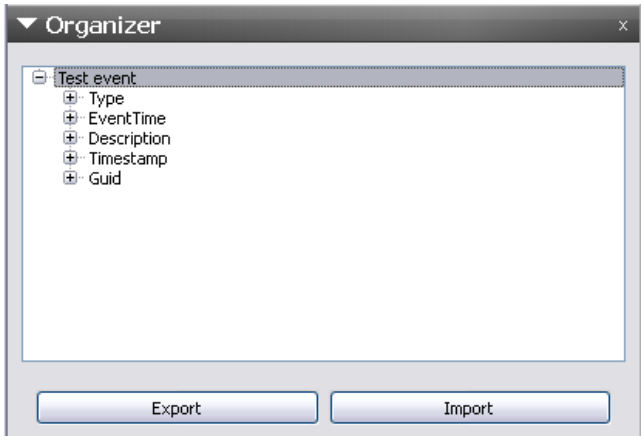
الشكل 9.13

9 - التبويب Sensors : يحوي على أوامر تعديل على جميع انواع الحسّاسات من حسّاس البوصلة "sensor of compass" و حسّاس التوجيه " sensor of orientation" , لكل جهة ثلاث محاور "x – y - z" , انظر الشكل (9.14)



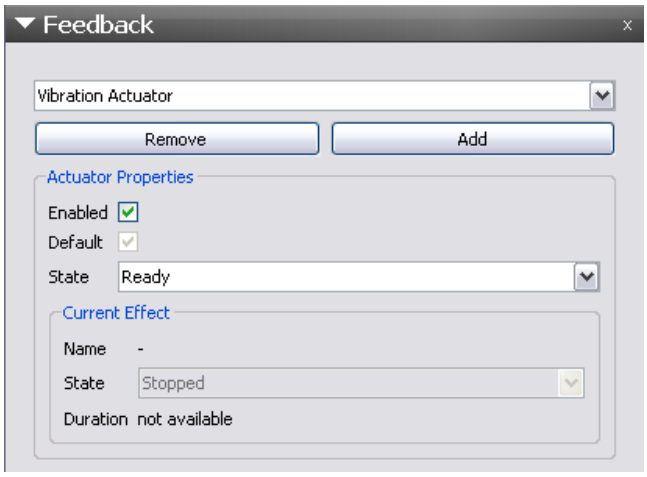
الشكل 9.14

10 - التبويب Organizer : يحوي على المهام و جداول الأعمال المحدثة من منصة التشغيل , انظر الشكل (9.15)



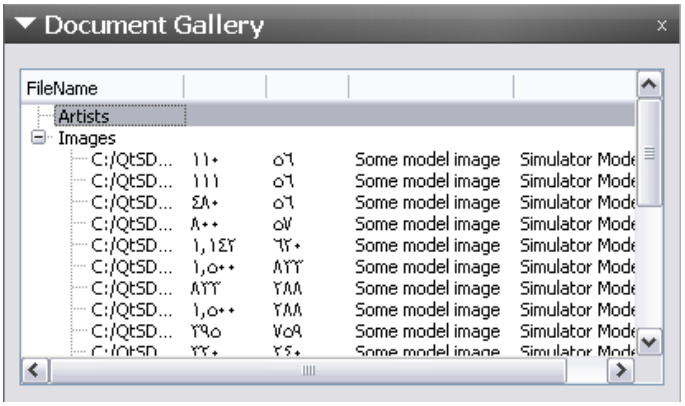
الشكل 9.15

11 - التبويب Feedback : خاص بـ تهيئة و مراقبة و عرض معلومات مشغل الإهتزاز و مشغل الصوت , انظر الشكل (9.16)



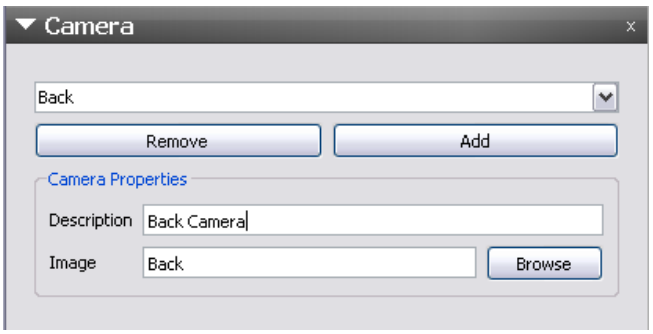
الشكل 9.16

12 - التبويب Document Gallery : يحوي على المستندات الخاصة بالعرض ضمن معرض منصة التشغيل , انظر الشكل (9.17)



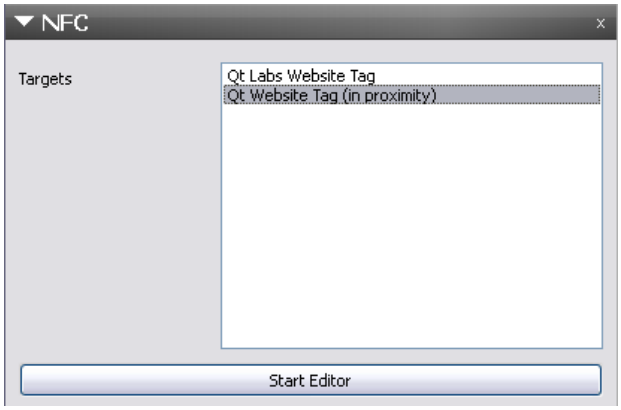
الشكل 9.17

13 - التبويب Camera : يحوي على خصائص جهاز الكام , نستطيع وضع صورة لتمثيل الكام الأمامي للجهاز , وصورة لتمثيل الكام الخلفي , انظر الشكل (9.18)



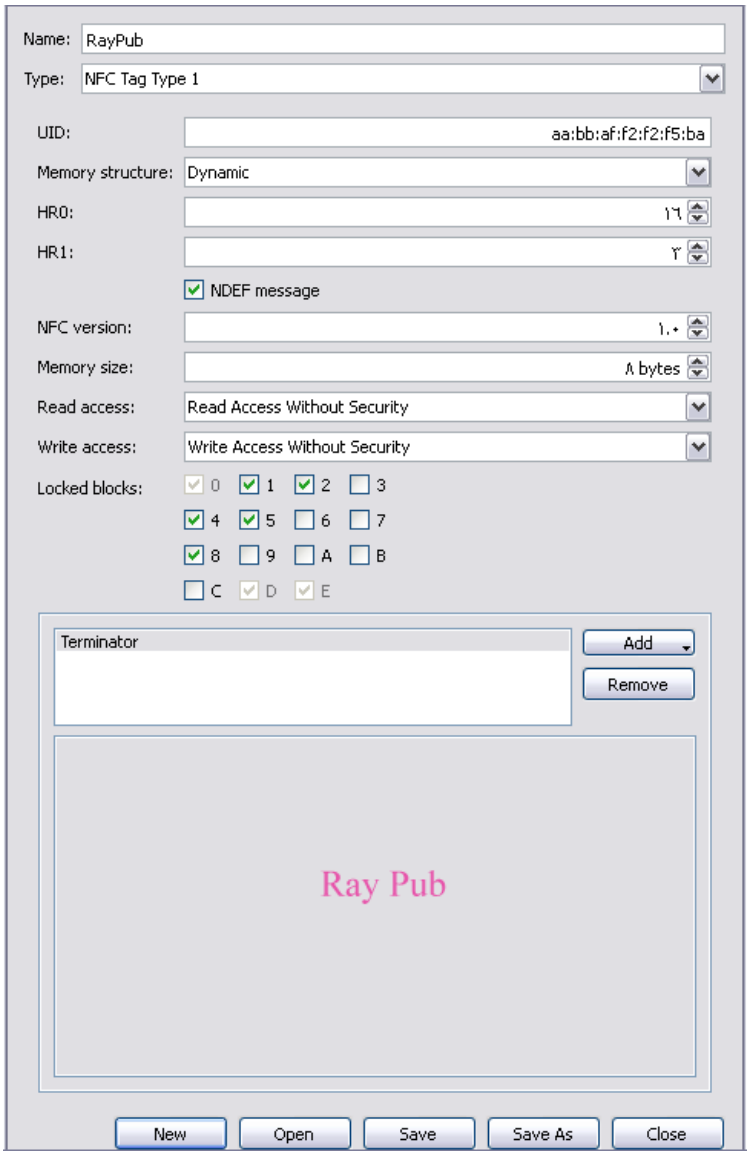
الشكل 9.18

14 - التبويب NFC : خاص يتعدل التشكيلات الجانبية لـ تقنية الـ " Near Field Communication" و استقبال البيانات المرسلة بنفس التقنية ثمّ عرضها و بث بيانات بوساطة التقنية "NFC" , انظر الشكل (9.19)



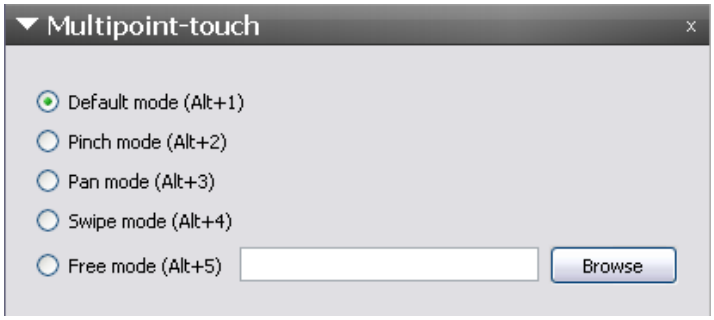
الشكل 9.19

عند الضغط على الزر "Start Editor" الخاص بتعديل التشكيلات الجانبية لاتصالات "NFC" , ستظهر نافذة كـ الشكل (9.20)



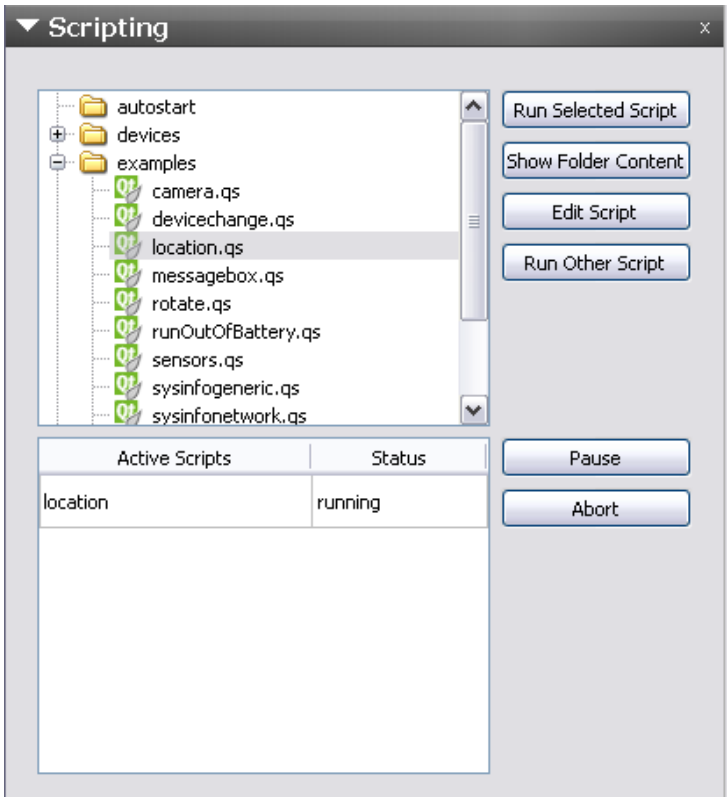
الشكل 9.20

15 - التبويب Multipoint-touch : يحوي على إعدادات حساس اللمس , انظر الشكل (9.21)



الشكل 9.21

16 - التبويب Scripting : يحوي على أوامر لتحرير و تنفيذ رمّازات خطّية لبرنامج المحاكاة, كـ رمّاز خطي خاص بـ تعديل إعدادات الكام , انظر الشكل (9.22)



الشكل 9.22

❖ **المنصة Harmattan :**

لتنفيذ و اختبار تطبيق "Qt" محمول على منصة "MeeGo" الإفتراضية , نستطيع استخدام محاكي "MeeGo" المرفق مع حزمة "Nokia Qt Creator" و الذي يدعى "QEMU" , لتنفيذ التطبيق على محاكي "MeeGo" يتوجب عليك أولا أن تضع المنصة "Hamattan" كـ منصة البناء للمشروع , ثمّ تشغيل المحاكي "QEMU" من خلال الضغط على الزر

"Start MeeGo Emulator" كما هو موضح في الشكل (9.23) و أخيرا الضغط على زر "Run" :

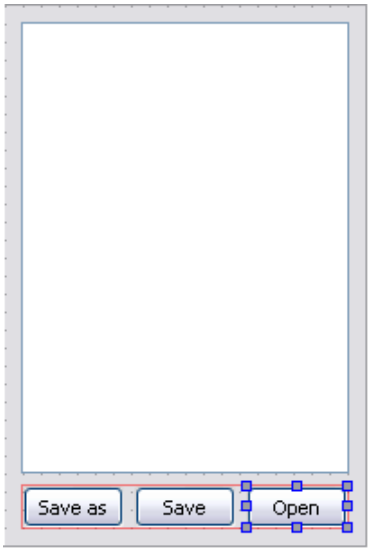
الشكل 9.23

يبدو مظهر المحاكي "QEMU" عند تشغيله كما في الشكل (9.24) :

الشكل 9.24

❖ **بناء تطبيق مذكرة قابل لـ الحمل :**

لنبني تطبيق مذكرة يعمل على منصات عدة , أنشأ مشروع "Mobile Qt Application" و سمّه "notebook" , أولا ابدأ صمم واجهة التطبيق لتبدو كالشكل (9.25) :



الشكل 9.25

ثانيا إعطاء سماحية الوصول إلى بيانات المستخدم من أجل القراءة و الكتابة على منصة "Symbian" كالآتي :

```
symbian:TARGET.CAPABILITY += NetworkServices ReadUserData
WriteUserData
```

ثالثا ضمّن ملفات التروسية لتالية داخل الملف "mainwindow.h" :

```
#include <QFileDialog>

#include <QFile>

#include <QTextStream>
```

ضمّنا ملف التروسية الأول من أجل استخدام نافذة حوار لإختيار ملف لـ القراءة و الكتابة, وضمّنا ملف التروسية الثاني من أجل الوصول و معالجة ملف , أما ملف التروسية الأخير لتغيير ترميز النص إلى "UFT8" , صرّح عن متحول نصي ضمن القسم الخاص :

```
   QString fileName4save;
```

حيث سيحفظ داخله مسار و اسم الملف النصي الذي قد تمّ حفظه, نستخدمه ضمن مقبس النقر على زر حفظ , ضمن كتلة مقبس زر الفتح اكتب التالي :

```
void MainWindow::on_open_clicked()

{

   QString str;
```

```
   QString fileName =
   QFileDialog::getOpenFileName(this,QString(),QString(),tr("Text Files
(*.txt)"));
   QFile file(fileName);
   if( file.open(QIODevice::ReadOnly | QIODevice::Text) )
      while (!file.atEnd())
         str += file.readLine() ;

   QTextStream textStream(&str);
   textStream.setCodec("UTF8");
   ui->textEdit->setText(textStream.readAll());
   fileName4save = fileName ;
   file.close();

}
```

أظهرنا نافذة الحوار الخاصة بتحديد ملف للقراءة , و خزّنا مسار و اسم الملف الذي قد تمّ تحديد ضمن المتحول النصي "fileName" فتحنا الملف من أجل القراءة , ثمّ قرءنا جميع بياناته و خزّناها ضمن المتحول النصي "str" , استخدمنا الصف "QTextStream" من أجل تحويل ترميز النص إلى الترميز "UFT8" , بعدها اسندنا لنص الكائن "textEdit" النص المقروء من الملف , أخيرا أغلقنا الملف النصي, لننقل إلى كتابة رمّاز مقبس زر "save as", انظر الرمّاز :

```
void MainWindow::on_save_as_clicked()
{
   QString str = ui->textEdit->toPlainText();
   QTextStream textStream(&str);
```

```
   textStream.setCodec("UTF8");
   QByteArray byteArray;
   byteArray.append(textStream.readAll());

   QString fileName =
   QFileDialog::getSaveFileName(this,QString(),QString(),tr("Text Files
(*.txt)")) ;
   QFile file(fileName);
   if( file.open(QIODevice::WriteOnly | QIODevice::Text) )
      file.write(byteArray) ;
   fileName4save = fileName ;

   file.close();

}
```

أولا اسندنا إلى المتحول النصي "str" النص الموجود داخل الكائن "textEdit" , ثمّ حوّلنا ترميز النص إلى الترميز "UFT8" , أسندنا إلى مصفوفة البايت النص المحول ترميزه , أظهرنا نافذة الحوار الخاصة بحفظ ملف , اسندنا إلى المتحول النصي "fileName" مسار و اسم الملف النصي , فتحنا الملف المحدد من اجل الكتابة , كتبنا داخل الملف مصفوفة البايت , أسندنا إلى المتحول "fileName4save" من اجل استخدامه داخل مقبس زر "save",اخيرا أفلقنا الملف , اكتي كتلة مقبس النقر على زر "save" :

```
void MainWindow::on_save_clicked()
{
   if(fileName4save != ""){
   QString str = ui->textEdit->toPlainText();
```

```
QTextStream textStream(&str);
textStream.setCodec("UTF8");
QByteArray byteArray;
byteArray.append(textStream.readAll());

QFile file(fileName4save);
if( file.open(QIODevice::WriteOnly | QIODevice::Text) )
   file.write(byteArray) ;

file.close();
}else
   on_save_as_clicked();
}
```

أولا اختبرنا ان المتحول النصي "fileName4save" غير فارغ و ذلك من اجل التأكد من فتح ملف مسبقا , في حال كان المتحول غير فارغ سينفذ لآتي :

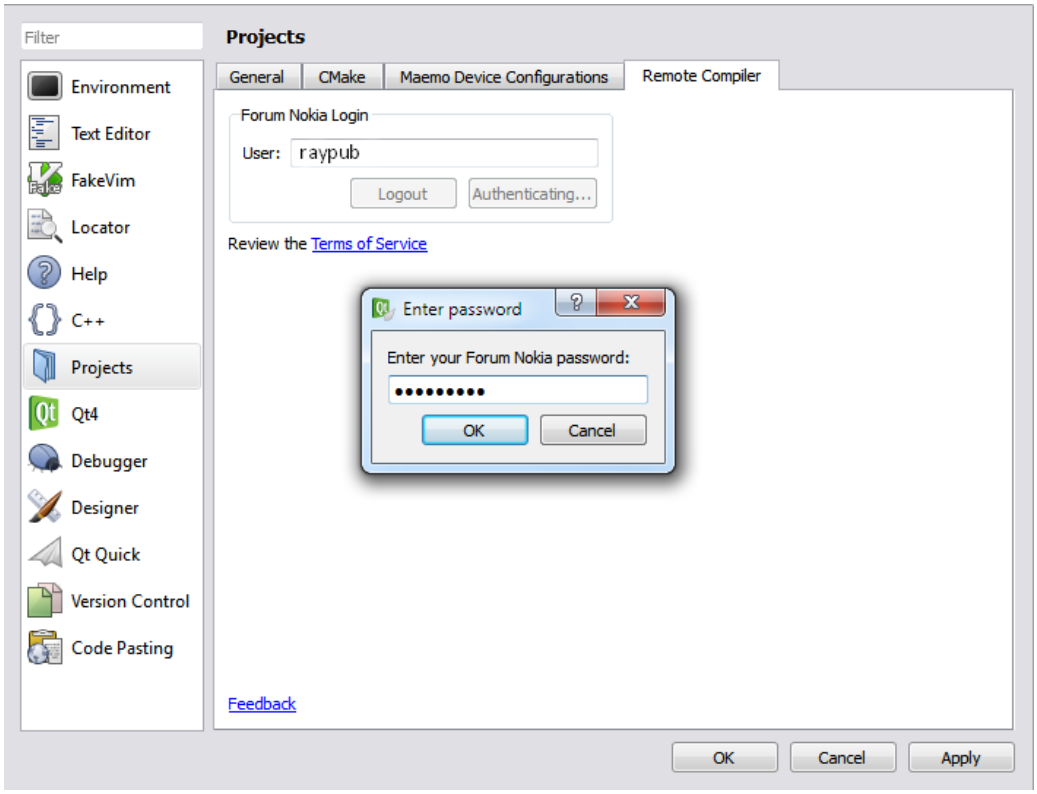
أولا اسناد نص الكائن "textEdit" إلى المتحول "str" و تحويل ترميز النص إلى "UFT8" , ثمّ إضافة النص المحول ترميزه إلى مصفوفة البايت "byteArray" , فتحنا ملف من أجل الكتابة , هذا الملف هو نفسه آخر ملف قد تمّ فتحه إن كان للقراءة أو للكتابة , كتبنا نص الكائن "textEdit" المحول ترميزه و المحفوظ داخل مصفوفة البايت على الملف المفتوح للكتابة , أخيرا أغلقنا الملف بعد الكتابة عليه , في حال كان المتحول "fileName4save" فارغ , فنستدعي المقبس "on_save_as_clicked()" الخاص بـ إظهار نافذة حوار خاصة باحفظ و من ثمّ حفظ الملف بعد تحديد مساره و اسمه , نفذ التطبيق باستخدام برنامج المحاكاة "Simulator" , و اختبر النتيجة , نستطيع تجميع التطبيق ليعمل على منصة "Symbian^3" , من خلال استخدام موجه الأوامر الخاصة بهذه المنصة و استخدام الأوامر "qmake" و "make sis" .

❖ المجمع البعيد :

في حال كنا نعمل على نظام "Win 7" و نريد تجميع التطبق ليعمل على منصة "Mac OS X" , فما يتوجب علينا فعله ؟

كل مل علينا هو استخدام المجم البيعيد .

- **المجمع البعيد "Remote Compiler" : هو خدمات "Qt" سحابية للتجميع, موجودة على مخدمات تابعة لشركة "Nokia" ,خاصة بتجميع الرمّاز المكتوب على الآلة المحلية "المشروع الذي تحدده" إلى عدة ملفات تنفيذيّة , كل ملف تنفيذي يعمل على منصة , فينشأ تطبيق لـ منصة "Symbian^3" و لمنصة "Meego"و لـ "Win CE" , "Mac OS X" الخ ..**
- **تجميع مشروع "Qt" باستخدام المجمع البعيد : اولا لاستخدام خدمة التجميع البعيد يتوحب أن تنشأ حساب على الموقع https://projects.developer.nokia.com/remotecompiler , ثمّ تحديد منصة العمل "Remote Compiler" الخاصة بـ تجميع المشروع بوساطة خدمات Nokia السحابية المختصة ببناء تطبيقات تنفيذيّة لجميع المنصات المتاحة , اضغط على زر التجميع و التنفيذ , لتظهر لك نافذة المصادقة الخاصة بتسجيل الدخول إلى خدمات Nokia السحابيّة ,انظر الشكل (9.26) :**



الشكل 9.26

❖ الحزمة QtMobility في Qt :

تحوي الحزمة "QtMobility" مجموعة متنوعة من التقانات , فيوجد بداخلها مجموعة من توابع "API" التي تعمل على معظم منصات العمل , مكتبات الحزمة "QtMobility" الإصدار "1.2" :

Bearer Management	**للتحكم بحالة اتصالات النظام**
Connectivity	**للإتصال مع الأجهزة المحلية , حيث يدعم تقنية الـ "Bluetooth" و تقنية "NFC" اختصارا لـ " Near Field Communication"**
Contacts	**يتيح للعملاء طلب بيانات إتصال من الجهاز المحلي أو البعيد**
Document Gallery	**لإستكشاف و استعلام عن مستندات الوسائط المتعددة**

Feedback	يتيح للعملاء استخدام تغذية عكسية لمسية و سمعية لأفعال المستخدم
Location API	لاستخدام الملاحة الإلكترونية , باستخدام تقنية "GIS"
Messaging	تفعيل وصول العميل إلى الخدمات الخاصة بالرسائل
Multimedia	الوصول إلى الوسائط المتعددة , تشغيل و تسجيل صوت و فيديو و إذاعة مجموعات من الوسائط المتعددة
Organizer	تتيح للعميل الوصول إلى التقويم و جداول الإعمال و البيانات الشخصية للمستخدم
Publish and Subscribe	للإتصال بين التطبيقات و قراءة و كتابة بيانات منها و إليها
Qt Service Framework	الوصول إلى خدمات المنصة الحاملة للتطبيق
QtMobility QML Plugins	وحدات نمطية من الحزمة "QtMobility" متوافقة مع "QML", نتكلم عنها في الفصل الخامس عشر ضمن قسم "Qt Quick"
Sensors	الوصول إلى أجهزة الحساس الموجودة بالجهاز
System Information	الوصول إلى معلومات الجهاز
Versit	استيراد و تصدير "vCard" و تنسيقات "iCalendar"

- **استخدام الحزمة QtMobility :**

عندما نود استيراد أحد مكاتب الحزمة "QtMobility" ضمن مشروعنا , يتوجب أن نصرح عن هذه المكتبة ضمن ملف المشروع , وذلك كالآتي :

نضيف ضمن ملف المشروع الرمّاز التالي :

CONFIG += mobility

MOBILITY += libraryMobilityName

في حال أردنا استخدام مكتبة الوسائط المتعددة "multimedia" يتوجب أن نضيف ضمن ملف المشروع الرمّاز الخاص بالتصريح عن الحزمة "QtMobility" و استيراد مناهج المكتبة "multimedia" منها :

CONFIG += mobility

MOBILITY += multimedia

ضمن الجدول التالي نعرض كل مكتبة و ما يقابلها من مفاتيح خاصة للتصريح :

Bearer Management	**Bearer**
Contacts	**Contacts**
Location	**Location**
Multimedia	**Multimedia**
Messaging	**Messaging**
Publish And Subscribe	**publishsubscribe**
Service Framework	**serviceframework**
Sensors	**Sensors**
System Information	**Systeminfo**
Versit	**Versit**
Document Gallery	**Gallery**
Organizer	**Organizer**
Tactile Feedback	**Feedback**

ملاحظة : عند إنشاء مشروع "Mobile Qt Application" سوف يضيف المعالج الذكي بشكل تلقائي ضمن ملف المشروع الرمّاز الخاص بـ تفعيل إتاحة استخدام الحزمة ضمن المشروع , لكن هذا الرمّاز يكون مكتوب بعد رمز التعليق "#" :

```
# CONFIG += mobility

# MOBILITY +=
```

- **تفعيل استخدام الحزمة QtMobility ضمن منصة Symbian :**

عند استخدام أحد مكاتب الحزمة "QtMobility" في تطبيق نود نشره على نظام Symbian , يتوجب علينا أن نفعّل قابية الوصول لـ تلك الخدمات التي اتصلنا بها من أحد مكتبات الحزمة "QtMobility" , مثال في حال أردنا أن نصل من تطبيقنا المحمول على منصة "Symbian" إلى جهاز الـ "Bluetooth" و إرسال ملف منه , يتوجب أن نضيف السطر البرمجي التالي ضمن ملف المشروع :

```
symbian:TARGET.CAPABILITY += WriteDeviceData UserEnvironment
```

- **انظر الجدول الذي يحوي الجمل الخاصة بتفعيل*قابلية الوصول* ضمن منصة "Symbian" :**

سماحية الوصول إلى الخدمات الشبكية	**NetworkServices**
سماحية الوصول لقراءة بيانات المستخدم الخاصة	**ReadUserData**
سماحية الوصول للكتابة في بيانات المستخدم الخاصة	**WriteUserData**
سماحية الوصول لقراءة دخل من أحد الأجهزة الداخلية كـ جهاز Bluetooth	**ReadDeviceData**
سماحية الوصول لإعطاء خرج لأحد الأجهزة الداخلية كـ جهاز Bluetooth	**WriteDeviceData**
سماحية استخدام الخدمات التي تتحكم بالبيئة الفيزيائيّة للجهاز كـ الوصول إلى الكاميرا أو المايك	**UserEnvironment**

سماحية الوصول إلى خدمات الموقع الجغرافي GPS	Location
سماحية استخدام الخدمات التطبيقيّة التي تعمل على الجهاز	LocalServices

ملاحظة : عند إنشاء مشروع "Mobile Qt Application" سوف يضيف المعالج الذكي بشكل تلقائي ضمن ملف المشروع الرمّاز الخاص بـ تفعيل قابليّة الوصول إلى الخدمات الشبكيّة لمنصة "Symbian" :

```
symbian:TARGET.CAPABILITY += NetworkServices
```

❖ الحصول على معلومات الجهاز :

في حال أردنا أن نعلم معلومات عن الجهاز الحامل للتطبيق , كـ مستوى شحن البطارية "المدخرة" , أو نوع الجهاز و اسمه الخ.. يتوجب علينا استخدام الصف "QSystemDeviceInfo" التابع للحزمة "QtMobility" , الصف "QSystemDeviceInfo" يحوي على مجموعة من المناهج الداخلية الخاصة بإرجاع معلومات عن الجهاز الحامل للتطبيق و مجموعة من الأحداث الداخلية التي ترفع عند تغيير قيمة أحد المعلومات القابلة للتغيير كـ قوة إشارة الشبكة لجهاز الجوال , لنبني مشروع نعرض فيه كيفيّة الوصول إلى معلومات الجهاز الحامل للتطبيق .

• المشروع SystemInfo :

سيعرض ضمن واجهة المستخدم للتطبيق المعلومات التالية :

1. **مستوى شحن البطارية الحالي مع عرض التغييرات التي تطرأ عليها**
2. **حالة شريحة الـ "SIM"**
3. **التشكيلات الجانبية لجهاز الجوال مع عرض التغيرات التي تطرأ عليها**
4. **اسم المنتج والمصنع و طراز الجهاز**

لنبدأ بناء المشروع ..

أولا أنشأ مشروع "Mobile Qt Application" و سمّه "SystemInfo" , ثمّ افتح ملف المشروع "SystemInfo.pro" و أضف للسطر البرمجي الخاص بقابيّات الصول لـ منصة "Symbian" :

```
symbian:TARGET.CAPABILITY += NetworkServices
```

القابليّات التالية :

1- ReadDeviceData
2- ReadUserData
3- UserEnvironment

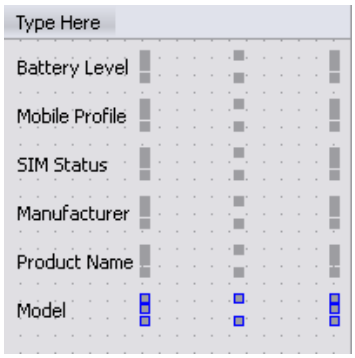
ليصبح السطر البرمجي كالآتي :

```
symbian:TARGET.CAPABILITY += NetworkServices ReadDeviceData
ReadUserData UserEnvironment
```

الآن أتح استخدام الحزمة "QtMobility" ضمن مشروعنا الحالي و ضمّن المكتبة "systeminfo" ليتاح استخدام صفوفها , انظر الرمّاز :

```
CONFIG += mobility

MOBILITY += systeminfo
```

اذهب إلى واجهة المستخدم و أضف اثنى عشر كائن لافتة "QLabel" , ستة منهم للتعرفة عن النص المجانب له و التسة الآخرين يعرض داخلهم قيمة المعلومات آنفة الذكر , انظر الشكل (9.27) :



الشكل 9.27

يجب أن يكون نموذج التخطيط هو "Form Layout" كما هو واضح في الشكل السابق , بالنسبة لأسماء كائنات اللافتة "QLabel" الخاصة بعرض القيمة يبدأ بـ "val_" , فـ اسم كائن اللافتة الخاص بـ عرض قيمة مستوى شحن البطارية هو "val_bl" الخ..

ضمن الملف "mainwindow.h" ضمّن ملف الترويسة التالي :

```
#include <qsystemdeviceinfo.h>
```

ثمّ أضف الماكرو الخاص بتعريف فضاء اسماء الحزمة "QtMobility" داخل المشوع الحالي

```
QTM_USE_NAMESPACE
```

ضمن قسم التعاريف الخاص صرّح عن مؤشر من نمط الصف "QSystemDeviceInfo" :

```
QSystemDeviceInfo* deviceInfo ;
```

و أضف المقابس التالية :

```
private slots:
   void batteryLevelChanged(int);
   void currentProfileChanged(QSystemDeviceInfo::Profile);
```

المقبس "batteryLevelChanged(int)" يستدعى عند تغيير مستوى شحن البطارية ,
أما المقبس "currentProfileChanged(QsystemDeviceInfo::Profile)" يستدعى عند تغيير التشكيلات الجانبية لجهاز الجوال كـ تغيير حالة الجهاز من "silent" إلى هزّاز "Vibrate"

الآن ضمن ملف التحقيق "mainwindow.cpp" أنشأ حدث للصف "QSystemDeviceInfo" داخل بناء الصف "MainWindow" :

```
deviceInfo = new QSystemDeviceInfo();
```

ثمّ أضف السطر البرمجي الخاص بإنشاء إتصال بين الحدث الداخلي "batteryLevelChanged(int)" التابع لـ "deviceInfo" و المقبس "batteryLevelChanged(int)" :

```
   connect(deviceInfo,SIGNAL(batteryLevelChanged(int)),this,
SLOT(batteryLevelChanged(int)));
```

ثمّ أنشأ اتصال بين الحدث الداخلي "currentProfileChanged(QSystemDeviceInfo::Profile)" و المقبس "currentProfileChanged(QSystemDeviceInfo::Profile)" :

```
connect(deviceInfo,SIGNAL(currentProfileChanged(QSystemDeviceInfo
::Profile)),

this,SLOT(currentProfileChanged(QSystemDeviceInfo::Profile)));
```

لعرض مستوى شحن البطارية و الحالة الراهنة للتشكيلات الجانبية أضف الرمّاز التالي الخاص باستدعاء المقبس "batteryLevelChanged(int)" و المقبس "currentProfileChanged(QsystemDeviceInfo::Profile)" :

```
batteryLevelChanged( deviceInfo->batteryLevel() ) ;

currentProfileChanged( deviceInfo->currentProfile() ) ;
```

سنقوم بعد الإنتهاء من كتابة الرمّاز الخاص بعرض معلومات الجهاز و الموجود داخل بناء الصف "MainWindow" بتحقيق المقبسين آنفي الذكر , لنكمل ...

أضف الرمّاز الخاص بعرض حالة شريحة الـ "SIM" :

```
switch(deviceInfo->simStatus()){

case QSystemDeviceInfo::SimNotAvailable:

ui->val_ss->setText("Sim Not Available");break;

case QSystemDeviceInfo::SingleSimAvailable:

 ui->val_ss->setText("Single Sim Available");break;

case QSystemDeviceInfo::DualSimAvailable:

 ui->val_ss->setText("Dual Sim Available");break;

case QSystemDeviceInfo::SimLocked:

 ui->val_ss->setText("Sim Locked");break;

default:ui->val_ss->setText("Unknown sim status");

}
```

التعداد "QSystemDeviceInfo::SimStatus" حالة شريحة الـ "SIM" :

SimNotAvailable	عدم وجود شريحة "SIM"

وجود شريحة "SIM" واحدة	SingleSimAvailable
وجود اثنان من شريحة "SIM"	DualSimAvailable
شريحة "SIM" مقفلة	SimLocked

بعدها أضف الرمّاز الخاص بعرض اسم المصنع و المنتج و الطراز للجهاز :

```
ui->val_manufacturer->setText(deviceInfo->manufacturer());
ui->val_pn->setText(deviceInfo->productName());
ui->val_model->setText(deviceInfo->model());
```

لنبدأ بتحقيق المقبسين , أولا رمّاز تحقيق المقبس "batteryLevelChanged(int)" :

```
void MainWindow::batteryLevelChanged(int level){
ui->val_bl->setText( QString::number(level) ) ;
}
```

ثانيا رمّاز تحقيق المقبس

"currentProfileChanged(QsystemDeviceInfo::Profile)" :

```
void MainWindow::currentProfileChanged
( QSystemDeviceInfo::Profile currentProfile ) {
ui->val_mp->setText("Unknown Profile") ; break ;
case QSystemDeviceInfo::SilentProfile :
ui->val_mp->setText("Silent Profile") ; break ;
case QSystemDeviceInfo::NormalProfile :
ui->val_mp->setText("Normal Profile") ; break ;
```

```
   case QSystemDeviceInfo::LoudProfile :
   ui->val_mp->setText("Loud Profile") ; break ;
   case QSystemDeviceInfo::VibProfile :
   ui->val_mp->setText("Vibrate Profile") ; break ;
   case QSystemDeviceInfo::OfflineProfile :
   ui->val_mp->setText("Offline Profile") ; break ;
   case QSystemDeviceInfo::PowersaveProfile :
   ui->val_mp->setText("Power save Profile") ; break ;
   case QSystemDeviceInfo::BeepProfile :
   ui->val_mp->setText("Beep Profile") ; break ;
   default: ui->val_mp->setText("Unknown Profile");
   }
}
```

التعداد "QSystemDeviceInfo::Profile" التشكيلات الجانبية للجهاز :

الوضع غير معروف	**UnknownProfile**
وضع صامت	**SilentProfile**
وضع الطبيعي	**NormalProfile**
وضع صوت مرتفع	**LoudProfile**
وضع رجاج	**VibProfile**
وضع غير متصل	**OfflineProfile**
وضع توفير الطاقة	**PowersaveProfile**
وضع اختياري	**CustomProfile**
الوضع نغمة "بيب" قصيرة	**BeepProfile**

نفذ التطبيق على برنامج المحاكاة "SIMULATOR" أو جهاز جوال يحوي منصة تعدم "Qt" كـ "Symbian" و اختبر النتيجة ..

❖ الوسائط المتعددة :

ضمن حزمة "QtMobility" يوجد المكتبة "QtMultimedia" الخاصة بتشغيل و معالجة و تسجيل الوسائط من صوت - فيديو الخ..

تم تطوير المكتبة "QtMultimedia" لتحوي عدة إضافات أهمها الوصول إلى الكاميرا الموصولة بالجاهز الحامل للتطبيق و التقاط صورة و فيديو منها و تصيير الصور و تسجيل الصوت و الفيديو بعدة تنسيقات , تدعى هذه المكتبة بعد تطويرها من قبل شركة "Nokia" المكتبة "QtMultimediaKit" .

• تشغيل ملف صوتي :

لتشغيل ملف صوتي نستخدم الصف "QMediaPlayer" التابع للمكتبة "QtMultimediaKit" , لنوضح كيفيّة تشغيل ملف صوتي من خلال بناء تطبيق "soundPlayer" .

• المشروع Sound Player :

أنشأ مشروع "Mobile Qt Application" و سمّه "soundPlayer" , ثمّ اذهب إلى ملف المشروع و فعّل الحزمة "QtMobility" ضمن مشروعنا الحالي , ثمّ أتح استخدام المكتبة "multimedia" , و ذلك بوساطة الرمّاز التالي :

```
CONFIG += mobility

MOBILITY += multimedia
```

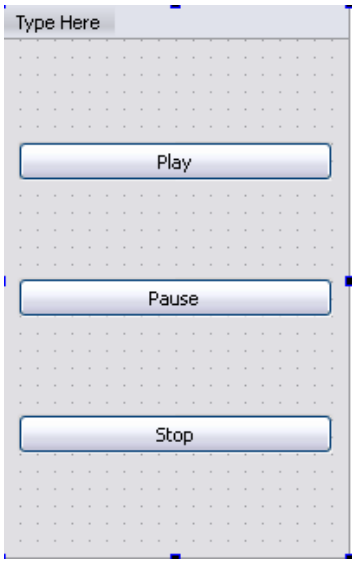
فعّل قابليات الوصول التالية الخاصة بمنصة التشغيل "Symbian" :

1- NetworkServices
2- ReadUserData

انظر السطر البرمجي :

symbian:TARGET.CAPABILITY += NetworkServices ReadUserData

الآن اذهب إلى واجهة المستخدم و أضف ثلاث أزرار , الأول زر التشغيل "Play" , الثاني زر الإيقاف المؤقت "Pause" , الثالث زر إيقاف "Stop" , انظر الشكل (9.28) :



الشكل 9.28

ضمن الملف "mainwindow.h" ضمّن ملف التروسية التالي :

```
#include <QtMultimediaKit/QMediaPlayer>
```

ثمّ صرح عن مؤشر للصف "QMediaPlayer" في القسم الخاص :

```
QMediaPlayer* mediaPlayer ;
```

اذهب إلى ملف التحقيق "mainwindow.cpp" و أضف الرمّاز الخاص بـ إنشاء حدث و تحديد مسار الملف الصوتي و ذلك داخل كتلة البناء للصف "MainWindow" :

```
mediaPlayer = new QMediaPlayer();

mediaPlayer->setMedia(QUrl::fromLocalFile("sound.mp3"));
```

في حال كان الملف الصوتي "sound.mp3" موجود على مخدم بعيد يتوجب علينا فقط كتابة الرمّاز كالآتي :

```
mediaPlayer->setMedia(QUrl("sound.mp3"));
```

مقبس الزر "play" :

```
void MainWindow::on_btn_play_clicked()
```

```
{
   mediaPlayer->play();
}
```

فقط استعدينا المنهج "play" التابع للصف "QMediaPlayer" , مقبس الزر "pause" :

```
void MainWindow::on_btn_pause_clicked()
{
   mediaPlayer->pause();
}
```

أخير مقبس الزر "stop" :

```
void MainWindow::on_btn_stop_clicked()
{
   mediaPlayer->stop();
}
```

الآن نفذ التطبيق و اختبر النتيجة ...

من أجل التحدم بشدة الصوت نستخدم المنهج :

```
QMediaPlayer::setVolume(int)
```

من أجل تغيير موقع التشغيل الحالي :

```
QMediaPlayer::setPosition(qint64)
```

من أجل كتم الصوت :

```
QMediaPlayer::setMuted(bool)
```

جلب مدة المقطع الصوتي :

```
qint64 QMediaPlayer::duration()
```

سيرجع المدة مقدرة بـ الميلي ثانية "ms" .

- **تشغيل قائمة من الملفات الصوتية :**

في حال أردنا أن نضع عدة ملفات صوتية في قائمة إنتظار و نشغيلها على التوالي يتوجب علينا أن نستخدم الصف "QMediaPlaylist" التابع للحزمة "QtMultimediaKit" , لتوضيح استخدام الصف "QMediaPlaylist" انظر الرمّاز :

```
QMediaPlaylist* mediaPlaylist = new QMediaPlaylist(mediaPlayer) ;

mediaPlaylist->addMedia(QUrl("http://myServer/snd.mp3")) ;

mediaPlaylist->addMedia(QUrl::fromLocalFile("video.mp4")) ;

mediaPlaylist->addMedia(QUrl("http://myServer/snd1.mp3")) ;

. .

mediaPlaylist->setCurrnetPosition(2) ;

mediaPlayer->play();
```

استخدمنا المنهج "addMedia" التابع للصف "QMediaPlaylist" من أجل إضافة مقطع صوتي في قائمة الإنتظار , و استخدمنا المنهج "setCurrentPosition" من أجل تحديد الملف الملف الذي سيعمل عند استدعاء المنهج "play" التابع للصف "QMediaPlayer" المرتبط بـ الحدث "mediaPlaylist" .

- **تسجيل الصوت :**

من أجل تسجيل الصوت يتوجب علينا ان نحدد جهاز التقاط الصوت "المايكروفون" , و تحديد تنسيق "ترميز" الصوت , يتوجب ان يكون ترميز الصوت مدعوم من المنصة الحاملة للتطبيق ..

نستخدم الصف "QAudioCaptureSource" من أجل جلب المعلومات عن الأجهزة الطرفية الخاصة بالتقاط الصوت الموصولة ضمن الجهاز الحامل للتطبيق

نستخدم الصف "QMediaRecorder" من أجل تسجيل مقطع صوت و ذلك بعد تحديد طرفية التقاط الصوت و وضع ترميز الصوت

للتوضيح :

رمّاز بسيط خاص بتسجيل مقطع صوتي و حفظه في ملف داخل ذاكرة الجهاز المحلية ..

```
QAudioCaptureSource* audioCaptureSource = new
QAudioCaptureSource ;
```

حددنا طرفية دخل الصوت و هي الطرفية الأولى و الإفتراضي , في حال أردنا أن نجلب جميع أسماء طرفيات دخل الصوت الموصولة بالجهاز نستدعي المنهج "audioInputs" التابع للصف "QAudioCaptureSource" انظر الرمّاز :

```
QStringList devices = audioCaptureSource->audioInputs() ;
```

للحصول على اسم طرفية دخل الصوت الإفتراضية نستدعي المنهج "defaultAudioInput" :

```
QString device = audioCaptureSource->defaultAudioInput();
```

و من أجل تحديد طرفية دخل صوت نستدعي المنهج "setAudioInput" :

```
audioCaptureSource ->setAudioInput(deviceName);
```

لنكمل ما بدأنا به ..

ننشأ حدث للصف "QMediaRecorder" الخاص بتسجيل الصوت :

```
QMediaRecorder* mediaRecorder = new
QMediaRecorder(audioCaptureSource) ;
```

مثلما تلاحظ مررنا لبناء الصف "QMediaRecorder" حدث الصف "audioCaptureSource" الخاص بتحديد طرفية دخل الصوت التي نود التقاط و تسجيل الصوت منها , الآن لتحديد موقع و اسم الملف الذي نود حفظ الصوت الملتقط داخله , نضيف الرمّاز التالي :

```
mediaRecorder->setOutputLocation(QUrl::fromLocalFile("snd.wav"));
```

من أجل وضع تنسيق للصوت يتوجب استدعاء المنهج "setEncodingSettings" حيث يأخذ وسيط من نمط "QAudioEncoderSettings" الخاص بـ تهيئة تنسيق لمقطع صوتي :

```
QAudioEncoderSettings  audioEncoderSettings ;

audioEncoderSettings.setCodec("audio/pcm");

mediaRecorder->setEncodingSettings( audioEncoderSettings ) ;
```

إذا أردنا معرفة أنواع الترميز المتاحة ضمن منصة التشغيل الحاملة للتطبيق انظر الرمّاز التالي :

```
QStringList codecs = mediaRecorder->supportedAudioCodecs();

 for(int i = 0; i < codecs.count(); i++)

    listBox->addItem(codecs.at(i));
```

لنكمل ..

لبدأ عملية التقاط الصوت نستدعي المنهج "record" :

```
mediaRecorder->record() ;
```

من اجل ايقاف تسجيل الصوت نستدعي المنهج "stop" :

```
mediaRecorder->stop();
```

ستجد داخل القرص المرفق مثال عن تسجيل الصوت يدعى "SoundRecorder"

- **تشغيل مقطع فيديو :**

نستخدم الصف "QMediaPlayer" من أجل تشغيل ملف الفيديو , و نستخدم الصف "QVideoWidget" من اجل تهيئة ملف الفيديو للعرض , للتوضيح :

```
   mediaPlayer = new QMediaPlayer();

   videoWidget = new QVideoWidget(ui->viewer);


   mediaPlayer->setVideoOutput(videoWidget);

   mediaPlayer->setMedia(QUrl::fromLocalFile("video.avi"));


   videoWidget->setGeometry(0,0,200,200);

   videoWidget->show();

  mediaPlayer->play() ;
```

وضعنا كائن الـ "QWidget" والذي يدعى "viewer" كـ شاشة لعرض مقطع الفيديو بعد تهيئته للعرض بوساطة الصف "QVideoWidget" و وضعنا جهاز خرج الفيديو لكائن الوسيط "mediaPlayer" حدص الصف "QVideoWidget" , ثمّ حددنا ملف الفيديو المراد عرضه , و حددنا مساحة العرض بوساطة المنهج "setGeometry" , و أظهرنا الفيديو , أخيرا شغلنا مقطع الفيديو بوساطة استدعاء المنهج "play" .

ملاحظة : في حال أردنا أن نعرض مشهد الفيديو على كامل الشاشة نستدعي المنهج "setFullScreen(bool)" بدلا من المنهج "setGeometry" .

ستجد داخل القرص المرفق مثال عن تشغيل ملف فيديو يدعى "videoPlayer"

❖ خلاصة الفصل :

حوى هذا الفصل عمومية استخدام Qt لبرمجة التطبيقات المحمولة , هنا كان التركيز على نظام Symbian طبعا في مؤلفي القادم إنشاء الله سيكون حول برمجة التطبيقات المحمولة لأنظمة Android و iPhone بوساطة Qt .

لننتقل إلى القسم الثاني من هذا الكتاب والذي يدعى QML.

قسم

QML

الفصل التاسع

مقدمة في QML

❖ مقدمة Intro :

لغة نمذجة Qt "Qt Modeling Language" و تستخدم لوصف و نمذجة واجهة المستخدم لتتطفي جمال و تنسيق عال على واجهة المستخدم بسهولة , بنية مكتبات هذه اللغة على لغة JavaScript .

تصمم هذه اللغة واجهة المستخدم لتطبيقات Qt الموجهة لأنظمة سطح المكتب و الأجهزة المحمولة , تدعم اللمس و الإدخال للأجهزة المحمولة أيضا .

تستطيع من خلالها تصميم أفضل الواجهات من خلال العناصر الرسومية الثابتة و المتحركة المتاحة ضمنها.

تحوي مكتباتها جميع عناصر المستخدم المعروفة من زر و كائن نص الإدخال و كائن عرض الصورة إلى كائنات الأشكال الهندسية كـ المستطيل و الخط و الدائرة , و حيث تستطيع إضافة تأثيرات حركية على هذه الكائنات , من تغير لون ما إلى لون آخر بشكل متدرج أو تغيير موقع كائن رسومي بشكل انسحابي أو دوراني و كل تلك الأفعال تستطيع تنفيذها من خلال إمّا رمّاز بسيط أو من خلال و اجهة QML الموجود في Qt Creator .

ملاحظة: بالنسبة للرسوميات المتحركة في QML سرعة عرضها 60 إطار بالثانية!!!

كل كائن رسومي في QML يدعى عنصر "Element" حبث تستطيع تحديد موضع و محاذاة و صفات هذا العنصر من خلال رمّاز QML البسيط .

دعنا نتكلم بتفصيل أكبر حول هذه اللغة من خلال الفصول الثلاث القادمة .

أساسيّات في QML

❖ مقدمة Intro :

بتعريف عناصر QML و وصفها باستخدام خصائصها و تغيير سلوكها و الإستجابة لأي فعل من المستخدم أو فعل معرف مسبقا كالمؤقت تكون وضعت حجر الأساس بلغة QML.

لكل عنصر خصائص معينة تصف شكله و لونه و سلوكه و حالته , حيث تستطيع تغيير هذه الخصائص من خلال كتابة كتل ضمنية "أي داخل جسم العنصر المعرف" .

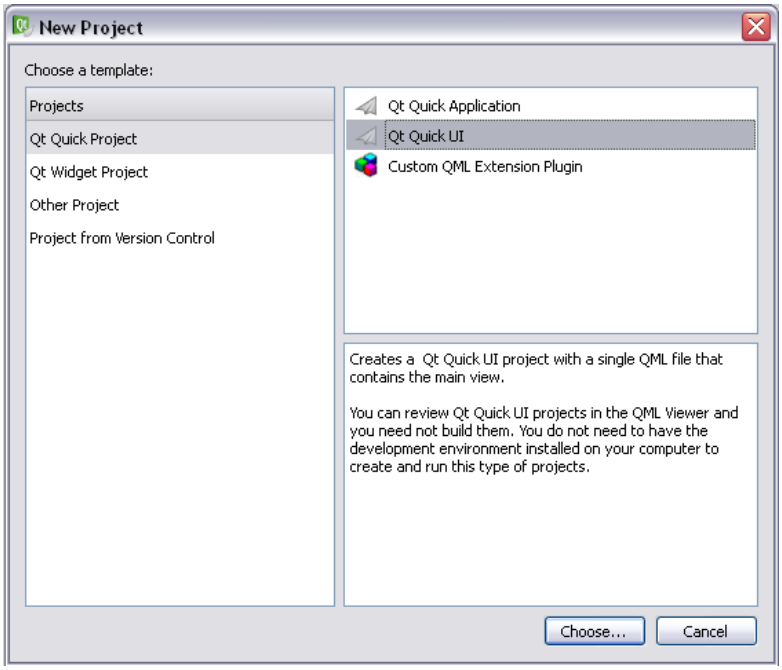
دعنا نبدأ بالتفصيل شرحا..

❖ إنشاء مشروع QML :

لإنشاء مشروع QML نأخذ من القائمة "File" الخيار "New Project" ثمّ نحدد التبويب "Qt Quick Project" بعدها نختار العنصر "Qt Quick UI" كما في الشكل (8.1), بعد إختياره نضع اسما للمشروع من ثمّ انهاء ,عند إنشاء مشروع QML سوف تجد ملفين داخل المشروع المنشأ , الأول هو ملف المشروع "prjectName.qmlproject" والذي يحوي الرمّاز الخاص باستيراد المكتبة الرئيسيّة لـ QML و الرمّاز الخاص بتحديد الملف الرئيسي للمشروع " نقطة إطلاق للمشروع" و أخيرا الرمّاز الخاص بتحديد مسارات الملفات الذي سوف تكون ضمن المشروع الحالي "المسارات الإفتراضيّة هي المسار الحالي للمشروع" , أنواع الملفات هي :

ملفات QML , ملفات JavaScript , ملفات صور

,الملف الثاني الموجود ضمن المشروع "projectName.qml" وهو الملف الرئيسي للمشروع ,و الذي يكتب داخله رمّاز لغة QML .



الشكل 8.1

داخل الملف الرئيسي لمشروع "QML" سوف تجد الرمّاز التالي :

```
import QtQuick 1.0


Rectangle {
   width: 360
   height: 360
   Text {
      anchors.centerIn: parent
      text: "Hello World"
   }
   MouseArea {
      anchors.fill: parent
```

```
    onClicked: {
        Qt.quit();
    }
  }
}
```

السطر البرمجي الأول لاستيراد الوحدة النمطية "QtQuick" الإصدار 1.0 , التي تحوي جميع عناصر QML القياسيّة من مستطيل إلى عنصر صورة و عنصر تدرج لوني الخ..

هذه الوحدة النمطية موجودة على جهازنا المحلي , في حال أردنا ان نستورد وحدة نمطية من جهاز بعيد "مخدم" نستطيع ذلك من خلال كتابة عنوان المخدم و اسم الوحدة النمطية ثمّ اصدارها كالآتي :

```
import "http://myServer/modules/module" 1.1
```

ملاحظة :

لإعطاء قيمة لخاصيّة في "QML" نضع أولا اسم الخاصيّة يتليها محرف النقطتين ثمّ القيمة التي نريد :

```
Property : value
```

استخدم العنصر "Rectangle" لملئ كامل شاشة تطبيق "QML" بعرض و ارتفاع "360*360" , يوجد داخل كتلة عنصر المستطيل عنصر ابن له وهو عنصر النص "Text" الخاص بطباعة نص ما ضمن المساحة المعطاة له , أما العنصر "MouseArea" فخاص باستقبال أحداث الفأرة ضمن منطقة معينة , هنا يعد العنصر "MouseArea" ابن لعنصر المستطيل لإنه موجود داخل كتلته , سوف يتم استقبال أحداث الفأرة على كامل مساحة المستطيل و يتمّ ذلك باستخدام الرمّاز :

```
anchors.fill: parent
```

عند الضغط على أي منطقة ضمن المستطيل سوف يتم الخروج من خلال تحقيق معالج حدث النقر"onClicked" , و استدعاء المنهج "Qt.quit()" ضمن كتلته للخروج :

```
onClicked:{
Qt.quit()
}
```

ملاحظة : البرنامج الخاص بتنفيذ رمّازات "Script" لغة "QML" يدعى "QML Viewer" .

لنبدأ الحديث حول أهم عناصر لغة "QML" .

❖ العنصر "Item" :

العنصر الأساسي "الأب" لمعظم العناصر المرئيّة في QML , يحوي الخصائص المشتركة لعناصر واجهة المستخدم .

أهم خصائص العنصر "Item" :

الشرح	الخاصيّة
إرساء الجهة السفلى للعنصر الحالي على الجهة المحددة للعنصر "AnchorLine"	anchors.bottom : AnchorLine
مسافة الهامش للجهة السفلى	anchors.bottomMargin : real
توضع العنصر الحالي في منتصف العنصر "Item"	anchors.centerIn : Item
إمتداد العنصر الحالي على كامل مساحة العنصر "Item"	anchors.fill : Item
إرساء الجهة اليسرى للعنصر الحالي على الجهة المحددة للعنصر "AnchorLine"	anchors.left : AnchorLine
مسافة الهامش للجهة اليسرى	anchors.leftMargin : real
إرساء الجهة اليمنى للعنصر الحالي على الجهة المحددة للعنصر "AnchorLine"	anchors.right : AnchorLine

مسافة الهامش للجهة اليمنى	anchors.rightMargin : real
إرساء الجهة العليا للعنصر الحالي على الجهة المحددة للعنصر "AnchorLine"	anchors.top : AnchorLine
مسافة الهامش للجهة العليا	anchors.topMargin : real
إرساء التوضع وسط المحور الأفقي للعنصر الحالي على الجهة المحددة للعنصر "AnchorLine"	anchors.horizontalCenter : AnchorLine
مسافة الهامش على المحور الأفقي	anchors.horizontalCenterOffset : real
إرساء التوضع وسط المحور العمودي للعنصر الحالي على الجهة المحددة للعنصر "AnchorLine"	anchors.verticalCenter : AnchorLine
مسافة الهامش على المحور العمودي	anchors.verticalCenterOffset : real
مسافة الهوامش على الجهات الأربع	anchors.margins : real
تعيد true في حال كان التركيز مفعّل على العنصر الحالي و إلَا تعيد false	activeFocus : bool
نقل التركيز للعنصر الحالي عند وضع القيمة true	focus : bool
إرتفاع العنصر الحالي	height : real
عرض العنصر الحالي	width : real
الإرتفاع الأساسي للعنصر الحالي	implicitHeight : real
العرض الأساسي للعنصر الحالي	implicitWidth : real
إظهار العنصر الحالي او إخفاءه	visible : bool
مقدار الشفافيّة للعنصر الحالي , تتروا قيمته بين 0.0 و 1.0	opacity : real
مقدار الدوران للعنصر الحالي مقاس بالدرجة	rotation : real

ضبط مقياس حجم العنصر الحالي	scale : real
تحديد نقطة المركز للتحولات التي تجري على العنصر من دوران "Rotation" و إعادة تقييس "Scale", نقاط المركز هي : 1 - Item.TopLeft 2 - Item.Top 3 - Item.TopRight 4 - Item.Left 5 - Item.Center 6 - Item.Right 7 - Item.BottomLeft 8 - Item.Bottom 9 - Item.BottomRight	transformOrigin : enumeration
تغيير العنصر الأب للعنصر الحالي	parent : Item
وضع قيمة نصية لوصف حالة العنصر الحالي	state : string
مبدل "switch" لحالات العنصر الحالي, *سيتم ذكره بالتفصيل ضمن هذا الفصل*	states : list<State>
نقطة التوضع للعنصر على المحور الأفقي	x : real
نقطة التوضع للعنصر على المحور العمودي	y : real
نقطة التوضع للعنصر على محور العمق "Z"	z : real

أنهينا شرح اهم خصائص العنصر الأساسي "Item" و التي سوف نرى الكثير من الأمثلة لإستخدام هذه الخصائص ضمن الصفحات القادمة من هذا القسم .

❖ العنصر Rectangle :

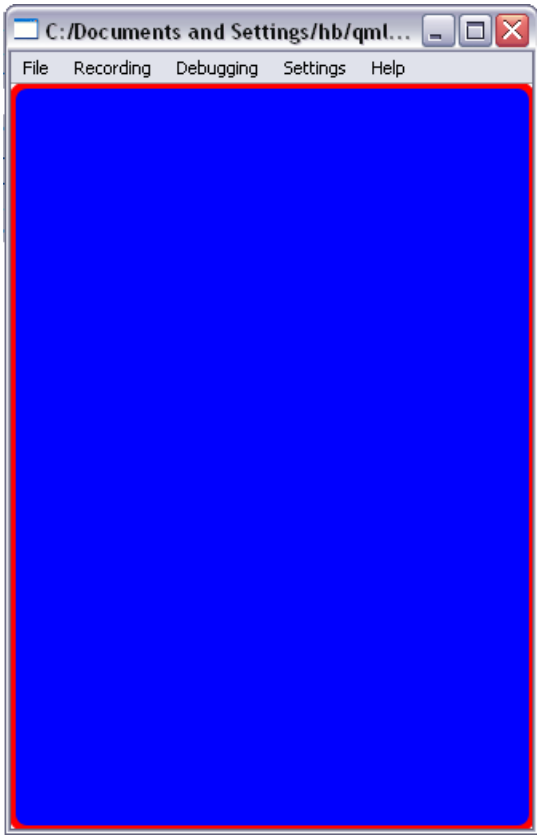
عنصر المستطيل خاص بملئ مناطق محددة من الواجهة أو الواجهة ككل, نستطيع ملئ هذا المستطيل بلون مصمت أو بألوان متدرجة , متاح إعادة تعيين قيم خصائص حوافه من لون إلى عرض و استدارة الخ.. , يرث "Item" و الذي يكون العنصر الأساسي "الأب" لجميع العناصر المرئيّة في QML .

أكثر إستخدامات عنصر المستطيل هي تعيين توضع عدة عناصر مرئيّة داخله .

مثال لإنشاء عنصر "Rectangle" :

```
import QtQuick 1.0

Rectangle {

   width: 320

   height: 460

   color: "blue"

   border.color : "red"

   border.width : 5

   radius : 10

   smooth : true

}
```

وضعنا اللون الأزرق لخلفيّة المستطيل و لون حده الأحمر أما عرض حدّه "الحرف" هو 5 بكسل , بالنسبة للخاصيّة "smooth" فهي خاصة بتنعيم نتوءات الحواف , أما الخاصية "radius" خاصة بوضع قيمة الإستدارة للمستطيل "نصف القطر", في حال كان عرض عنصر المستطيل مساوي للإرتفاع و كانت قيمة "radius" تساوي نصف العرض أو الإرتفاع يصبح شكل عنصر المستطيل دائري , نفذ التطبيق سوف يظهر كما في الشكل (8.2) :



الشكل 8.2

❖ العنصر Image :

خاص يتحميل و عرض صورة على واجهة المستخدم , يرث "Item" .

```
Rectangle {
    width: 320
    height: 460

    Image {
        id: img
        source: "pic.png"
```

```
    }
}
```

أضفا عنصر الصورة "img" كإبن لعنصر المستطيل , تمّ تحميل الصورة "pic.png" من خلال الخاصيّة "source" يتليها محرف نقطتين ":" بعدها مسار الصورة التي نريدها , في حال كانت الصورة موجودة على موقع ويب "مخدم ويب" :

```
Image {
source: "http://webhost/images/pic.png"
}
```

أهم خصائص عنصر الصورة :

asynchronous : bool

اذا كانت الصورة موجودة على نظام الملفات المحلي و وضعت القيمة true لهذه الخاصيّة سوف تتم عملية تحميل الصورة ضمن مسلك منفصل بشكل غير متزامن, في حال كانت الصورة موجدود على مخدم شبكيّ سوف تكون قيمة هذه الخاصيّة true افتراضيا .

cache : bool

لوضع الصورة ضمن الذاكرة المخبئة المخصصة للتطبيق , القيمة الإفتراضية true .

fillMode : enumeration

نمط التعبئة أيّ نسبيّة تغيير حجم الصورة من حجم العنصر "Image" الحواي لها , يوجد 6 عناصر ضمن التعداد الخاص بنمط التعبئة و هي :

- **Image.Stretch**

القيمة الإفتراضيّة للخاصيّة "fillMode" , حجم الصورة يساوي حجم العنصر الحواي .

- **Image.PreserveAspectFit**

ازدياد \ نقصان حجم الصورة بشكل يلائم حجم الصورة المثالي مع العنصر الحاوي لها داخل حدود هذا العنصر .

- **Image.PreserveAspectCrop**

ازدياد \ نقصان حجم الصورة بشكل يلائم حجم الصورة المثالي مع العنصر الحاوي لها داخل أو خارج حدود هذا العنصر .

- **Image.Tile**

تكرار الصورة عموديا و أفقيا ضمن مساحة العنصر الحاوي لها .

- **Image.TileVertically**

تكرار الصورة عموديا ضمن مساحة العنصر الحاوي لها .

- **Image.TileHorizontally**

تكرار الصورة أفقيا ضمن مساحة العنصر الحاوي لها .

نعود لإكمال خصائص عنصر الصورة ..

smooth : bool

تستخدم من اجل تنعيم الحواف "إزالة النتوءات الموجودة على حدود الصورة" .

progress : real

تعيد حاصل تقدم عملية تحميل الصورة .

status : enumeration

حالة الصورة الراهنة ضمن عملية التحميل , قيمة هذه الخاصيّة تعداد يحوي أربع عناصر لوصف الحالة :

- **Image.Null**

لم يتم وضع مسار الصورة المراد عرضها لعنصر الصورة .

- **Image.Ready**

الصورة قد تم تحميلها بنجاح .

- **Image.Loading**

يتم التحميل حاليا .

- **Image.Error**

حصول خطأ غير محدد عند محاولة تحميل الصورة , كأن يكون مسار الصورة المعطة للخاصيّة "source" غير موجود .

مثال لعرض الحالة الراهنة لتحميل الصورة :

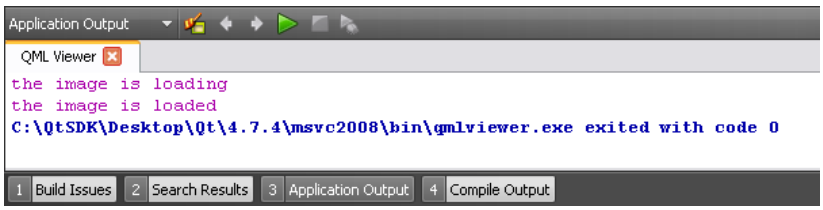
```
import QtQuick 1.0


   Image {
      id: img
      width: 200
      height: 200
      fillMode: Image.PreserveAspectFit
      source: "img.png"
      smooth: true
      onStatusChanged: {
         switch (img.status){
         case Image.Null : console.log("not set the image"); break;
         case Image.Ready : console.log("the image is loaded"); break;
         case Image.Loading : console.log("the image is loading"); break;
         case Image.Error : console.log("unknow err"); break;
         }
      }
}
```

راقبنا التغيرات التي تطرأ على عملية تحميل الصورة من خلال تحقيق معالج الحدث الداخلي "onStatusChanged" الذي سوف يتم قدحه عند أي تغير لحالة التحميل , كتلة التحقيق للحدث تكافئ المستقبل "slot" في Qt ,"*سوف نشرح بالتفصيل ضمن هذا الفصل الأحداث و معالجاتها في QML*", قد تم إختبار الحالات الأربع لعملية تحميل الصورة من خلال التعبير "switch" , بالنسبة للسطر :

console.log(arg)

فهو سوف يطبع النص الممرر له ضمن شاشة سطر الأوامر التي تم تنفيذ التطبيق من خلال تمرير أوامر لها, طبعا سوف يعمل فقط في حال التنقيح .

سوف تجد شاشة سطر الأوامر ضمن Qt Creator الخاصة بعرض أوامر التجميع و خرج التنقيح للتطبيق الخ.. بالجهة السفلى منه , حدد الخيار "Application Output" لعرض خرج التنقيح , انظر الشكل (8.3) :



الشكل 8.3

❖ العنصر BorderImage :

يمثل هذا العنصر عنصر الصورة لكن مع دعم التعامل مع الحواف , يرث "Item", انظر الرمّاز التالي :

```
import QtQuick 1.0

Rectangle{

   width: 600

   height: 600

BorderImage {

   anchors.centerIn: parent

   width: 128; height: 128

   border { left: 42; top: 42; right: 42; bottom: 42 }

   //border.left: 42
```

```
    //border.right: 42
    //border.top: 42
    //border.bottom: 42
    horizontalTileMode: BorderImage.Stretch
    verticalTileMode: BorderImage.Stretch
    source: ":/img.png"
}
}
```

الخاصية horizontalTileMode نمط العرض الأفقي , قيمته تأخذ أحد عناصر تعداد نمط العرض الثلاث :

- **BorderImage.Strech**

إمتداد الصورة *أفقيا* حتى تساوي عرض الحاوي لها .

- **BorderImage.Repeat**

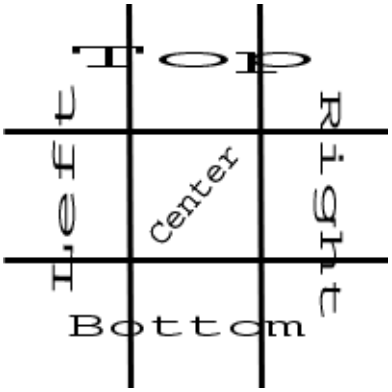
تكرار الصورة *أفقيا* ضمن مساحة العنصر الحاوي لها .

- **BorderImage.Round**

تكرار الصورة *أفقيا* ضمن مساحة العنصر الحاوي لها مع عدم قص أي جزء من الصورة المكررة , أي جميع النسخ المكررة للصورة تعرض كاملة من دون نقصان أي بكسل منها .

الخاصية verticalTileMode نفس عمل الخاصية horizontalTileMode لكن يتغير نمط عرض الصورة على المحور *العمودي* .

تقسم الصورةإلى خمس أجزاء : جزء علوي و سفلي و أيمن و أيسر و جزء أوسط , انظر الشكل (8.4) ,يأتي دور الخاصية "border" تغيير قيمة المسافة المعطاة عرضا و طولا لأي جزء من الأجزاء الأربع للصورة .



الشكل 8.4

ملاحظة : تتغير المساحة المعطاة للجزء الأوسط من الصورة بتغير مساحة "عرض أو طول" أحد الأجزاء الأربع .

في الرمّاز السابق وضعنا قيمة جميع الحدود هي 42 بكسل , بالتالي سوف يصبح عرض الجزء الأيسر 42 بكسل "يمتد من الحد الأيسر للصورة بإتجاه الحد الأيمن 42 بكسل" , و عرض الجزء الأيمن 42 بكسل "يمتد من الحد الأيمن للصورة بإتجاه الحد الأيسر 42 بكسل" ,و ارتفاع الجزء السفلي 42 بكسل "يمتد من الحد السفلي للصورة بإتجاه الحد العلوي 42 بكسل" ,و ارتفاع الجزء العلوي 42 بكسل "يمتد من الحد العلوي للصورة بإتجاه الحد السلفلي 42 بكسل" , نفذ التطبيق السابق و لاحظ التغريات التي طرأت على الحدود الأربع و الجزء الأوسط للصورة .

❖ العنصر AnimatedImage :

مثيل عنصر الصورة و مشتق منه , مع اضافة ميزة له و هي عرض صورة متحركة "Gif" داخله :

```
AnimatedImage{
   source: "anime.gif"
}
```

أهم خصائص هذا العنصر :

playing: bool

في حال كانت قيمته true يدل على أنه في حالة تنفيذ لتتالي الإطارات الذين يشكلوا الصورة لمتحركة , و في حال كانت القيمة false هذا يعني أنه في حالة ايقاف لتتالي الإطارات .

paused : bool

تستخدم من اجل الإيقاف المؤقت لتتالي الإطارات .

❖ العنصر Text :

خاص بعرض نص ذي تنسيق على واجهة المستخدم , متاح تغير تنسيق شكل النص من لون و نمط الحد للخط و نوعه الخ... , تستطيع تغيير تنسيق النص إما من خلال خصائص العنصر "Text" أو بإضافة وسوم HTML, يرث "Item" .

مثال لإستخدام العنصر "Text" :

```
import QtQuick 1.0


Rectangle{
   width: 600
   height: 600
   color: "black"

   Text {
      id:txt
      color: "white"
      font. pixelSize:  24
      text : "<a href=\"http://www.google.com\">goto Google</a>"
      onLinkActivated: Qt.openUrlExternally("http:// www.google.com
")
       }
```

}

وضعنا عنصر نص يدعى "txt" , حجم خطه 24 بكسل , و لونه أبيض , أما نص العنصر "txt" هو "goto Google" , بما أنه يقبل وسوم "HTML" كتبنا داخل نصه وسم الإرتبط التشعبي , يتم تنفيذ وسم الإرتباط التشعبي عند النقر عليه من خلال تحقيق معالج الحدث "onLinkActivated" , استخدمنا المنهج "Qt.OpenUrlExternally(url)" لفتح الموقع المراد ضمن مستعرض الويب الإفتراضي للنظام الحامل للتطبيق .

أهم خصائص الخط ضمن عنصر النص "Text" :

font.bold : bool

font.family : string

font.italic : bool

font.pixelSize : int

font.underline : bool

لإعطاء نمط رسومي للنص يجب علينا استخدام الخاصية "style" مع الخاصية "styleColor" ,اذهب إلى الرمّاز السابق , و أضف ضمن كتلة العنصر "txt" الرمّاز التالي :

```
style: Text.Sunken;

styleColor: "red"
```

عند تنفيذ التطبيق سوف يظهر شكل الخط كالشكل (8.5) :

الشكل 8.5

الخاصيّة "style" لإعطاء نمط المظهر الرسومي للخط , أما "styleColor" لإعطاء اللون للنمط المحدد لشكل الخط .

انماط مظهر الخط :

Text.Normal

Text.Outline

Text.Raised

Text.Sunken

أنماط تنسيق النص "textFormat" :

التنسق الإفتراضي , يأخذ التنسيق بشكل تلقائي , بعد تفسير النص يختار التنسيق الأفضل للنص	**Text.AutoText**
تنسيق بسيط , لا يدعم تفسير وسوم "HTML" , و لا يدعم نمط مظهر الخط	**Text.PlainText**
تنسيق غنيّ , يدعم تفسير وسوم "HTML" و نمط مظهر خط	**Text.RichText**
يدعم فقط نمط مظهر خط	**Text.StyledText**

❖ أحداث الفأرة و تفاعل التحول مع الحالة :

سوف نقدم في هذه الفقرة دراسة عن أحداث الفأرة بشكل تفصيلي , و الحالة و التحول , نقصد بالحالة التغير الذي حصل للعنصر , اما التحول هو كيفيّة الإنتقال من الحالة الأقدم إلى الحالة الأحدث , سوف نشرح بشكل تفصيلي أحداث الفأرة و الحالة و التحول من خلال بناء مثال يحوي هذه العناصر الثلاث آنفة الذكر .

لنبني تطبيق تحوي نافذته "التي هي عبارة عن عنصر مستطيل" على إثنان من عنصر "Rectangle" الأول يقع في الأسفل من الجهة اليمنى , و الثاني في الأعلى من الجهة اليسرى , المربع الثاني قابل للسحب, عند إفلات المربع الثاني داخل مساحة المربع الأول سوف يصبح المربع الأول هو الأب بالنسبة للمربع الثاني , خلال عملية سحب المربع الثاني يصبح شفافا , و عند إفلاته داخل المربع الأول سيتحول لونه من الأحمر "الإفتراضي له" إلى الأصفر بشكل تدريجي "تدعى هذه العملية بالتحول" ، و عند الضغط على المربع الأول سيتحول لونه تدريجيّا إلى فضي بشكل تدريجي أيضا.

لنبدأ : أنشأ مشروع "Qt Quick UI" يدعى "rect4drag" , أحذف الرمّاز الموجود داخل الملف "rect4drag.qml" , ثمّ نفذ التالي :

أولا ضمّن الوحدة النمطيّة "QtQuick" :

```
import QtQuick 1.0
```

بعدها أنشأ النافذة الرئيسيّة للتطبيق "عنصر مستطيل أب" بكتابة الرمّاز التالي :

```
Rectangle {
   id: win
   width: 600; height: 600
}
```

ثانيا أنشا عنصر المستطيل الذي سوف يحصل على الأبوّة بكتابة الرمّاز التالي داخل العنصر "win" :

```
   Rectangle{
      id:parentRect
      width: 200; height: 200
      anchors.bottom: parent.bottom
      anchors.right: parent.right
      color: "#4000ff"

      MouseArea{id:mouseArea
      anchors.fill: parent
      }

      states:State{
         name:"changeColor"
```

```
        when: mouseArea.pressedButtons & Qt.LeftButton
        PropertyChanges{ target: parentRect; color:"silver" }
    }
    transitions: Transition {
        ColorAnimation {  duration: 500 }
    }

}
```

تقتصر مهمة عنصر المستطيل "parentRect" على تغيير لونه بشكل تدريجي من الأزرق إلى الفضي عند الضغط عليه بزر الفأرة الأيسر , و قبول إسقاط المربع الثاني داخل ليصبح أب له.

غيرنا توضع عنصر "parentRect" ليصبح بالأسفل من الجهة اليمنى من خلال استخدام خاصيّة الإرساء "anchors", و جعلنا لونه الأزرق الخاصيّة "color", أضفنا عنصر "MouseArea" لإستقبال حدث النقر بزر الفأرة الأيسر داخل مساحة "parentRect" , و أنشئنا حالة "state" تدعى "changeColor" لتغيير لون المربع "parentRect" إلى لون فضي عند الضغط بزر الفأرة داخل مساحته , استخدمنا الخاصيّة "when" من أجل تفعيل حالة تعيير اللون عند الضغط بزر الفأرة الأيسر "Qt.LeftButton" , الرمّاز :

```
when: mouseArea.pressedButtons & Qt.LeftButton
```

استخدمنا الخاصيّة "pressedButtons" التي مهمتها إرجاع ثابت يدل على أزار الفأرة المضغوطة حاليا مع العملية المنطقية "&-AND" و الثابت "Qt.LeftButton" بالتالي عند الضغط على زر الفأرة الأيسر تصبح قيمة الجملة البرمجية السابقة "true" فيتم تفعيل الحالة "changeColor", و عند إفلات الضغط سوف تعود الحالة الأقدم للتفعيل, يتم تغيير اللون من خلال استخدام العنصر "PropertyChanges" و الذي مهمته تغيير قيمة خاصيّة ما للعنصر الهدف "target" .

سوف نرى ضمن الصفحات القادمة من الفصل الحالي كيفيّة تفعيل حالة ما من دون استخدام الخاصيّة " when" .

أضفنا نقطة إنتقال "تحول" للحالة السابقة و التي مهمتها تبطئة تغيير لون المربع من أزرق إلى فضي حيث مدة التحول "تغيير اللون" هي 500 ميلي / ثا , بالتالي سوف يتغير اللون بشكل تدريجي .

العنصر "ColorAnimation" خاص بتغيير اللون لعنصر ما لكن بشكل تدريجي حسب المدة الزمنية "duration" الموضوعة له , سوف نشرح بالتفصيل عناصر الحركة و من بينها العنصر "ColorAnimation" في فصل "عناصر الحركة" من هذا القسم .

التحول مرتبط بالحالة , سوف نشرح بعد قليل بالتفصيل عن الحالة و التحول .

بالنسبة لموضع كتابة جسم عنصر الحالة و التحول :

يجب أن يكتب عنصر الحالة داخل كتلة "خاصيّة" الحالات "states" و التي مهمتها إحتواء قائمة من عناصر الحالات للعنصر الحالي .

أما عنصر التحول "transition" يجب أن يكتب داخل كتلة "خاصيّة" التحولات "transitions" و التي مهمتها إحتواء قائمة من عناصر التحولات "نقاط الإنتقال" للعنصر الحالي .

ملاحظة : الخاصيّة "transitions" و "states" موجودين داخل العنصر الأب "Item" .

الآن لنتابع كتابة الرمّاز , اكتب داخل العنصر "win" بعد نهاية كتلة العنصر "parentRect" الرمّاز التالي :

```
Rectangle {
    id: rect
    width: 50; height: 50;
    color: "red"

    states: State{
    name: "changeParent";
    ParentChange{ target: rect; parent: parentRect; }
    PropertyChanges { target: rect; color:"yellow" }
```

```
    }
  transitions: Transition {
    ColorAnimation {duration: 1000; }
  }
  MouseArea {id:maRect
    anchors.fill: parent
    drag.target: rect
    drag.axis: Drag.XandYAxis
    drag.minimumX: 0
    drag.minimumY: 0
    drag.maximumX: parent.parent.width - rect.width
    drag.maximumY: parent.parent.height - rect.height
    onPressed: rect.opacity = 0.5
    onReleased: {
      rect.opacity = 1.0
      if(rect.x >= parentRect.x && rect.y >= parentRect.y ){
      rect.state = "changeParent"
      }
    }
  }
}
```

أنشئنا عنصر مستطيل يدعى "rect" , ارتفاعه و عرضه "50X50" بكسل , لونه أحمر , أنشئنا داخله حالة تدعى "changeParent" و التي مهمتها تغيير الأب لعنصر "rect" ليصبح الأب هو العنصر "parentRect" و ليس العنصر "win", و تغيير لون العنصر

"rect" إلى اللون الاصفر , ثمّ ربطنا بهذه الحالة نقطة إنتقال "تحول" خاص باللون مهمته تغيير لون العنصر "rect" تدريجيّا من اللون الأحمر إلى اللون الأصفر بمدة زمنية قدرها 1000 ميلي / ثا , أنشئنا عنصر "MouseArea" يدعى "maRect" , يستقبل أحداث الفأرة ضمن كامل مساحة العنصر "rect" , فعّلنا سماحية السحب على المحورين الأفقي و العمودي من خلال :

drag.axis: Drag.XandYAxis

المسافة المسموح للعنصر "rect" سحبه عليها هي :

على المحور الأفقي "X" من النقطة 0 "أقصى اليسار للعنصر الأب" إلى النقطة التي يكون موقعها عرض العنصر الأب "أقصى اليمين للعنصر الأب".

على المحور العمودي "Y" من النقطة 0 "أقصى الجهة العليا من العنصر الأب" إلى النقطة التي يكون موقعها إرتفاع العنصر الأب "أقصى الجهة السفلى من العنصر الأب".

مثلما تلاحظ وضعنا التعبير :

parent.parent.

هذا التعبير موجود داخل كتلة العنصر "maRect" الموجود داخل العنصر "rect" الذي يعد أبا لـ "maRect" , و بما اننا نريد عرض و ارتفاع أب العنصر "rect" (المحتوي للعنصر "rect") وجب علينا استدعاء الخاصيّة "parent" الثانية من الخاصيّة "parent" الأولى , *بتعبير آخر"أب الأب - الجد"* .

حققنا معالج حدث الضغط بزر الفأرة "onPressed" ليصبح العنصر "rect" شفاف طالما زر الفأرة في حالة ضغط عليه.

من ثمّ حققنا المعالج الخاص بحدث إفلات الضغط لزر الفأرة "onReleased" , حيث تحتوي كتلته على :

إرجاع الشفافيّة على قيمته الإفتراضيّة للعنصر "Rect" و هي "غير شفاف" , من ثمّ اختبار إذا كان العنصر "rect" داخل حدود "ضمن مساحة" العنصر "parentRect" في حال كان سوف يتم تفعيل الحالة "changeParent" من خلال الرمّاز :

rect.state = "changeParent"

استخدمنا الخاصيّة "state" الخاصة بالعنصر المراد تغيير حالته , عند تفعيل الحالة "changeParent" سوف يصبح أب العنصر "rect" هو "parentRect" .

هذه الطريقة الأخرى لتفعيل حالة ما , أما الطريقة الأولى هي "when" قد تمّ ذكرها مسبقا ضمن هذا الفصل .

نفذ التطبيق و اختبر ناتجه .

- **العنصر MouseArea :**

يستخدم من أجل معالجة مقابض الأحداث المدخلة من الفأرة ضمن منطقة محددة, كالنقر بزر الفأرة الأيسر على عنصر زر .

خصائص العنصر "MouseArea" :

acceptedButtons : Qt::MouseButtons	**تحديد أزرار الفأرة المسموح التفاعل معها ضمن منطقة العنصر "MouseArea" , التعداد "Qt::MouseButtons" :** ▪ **Qt.LeftButton** ▪ **Qt.RightButton** ▪ **Qt.MiddleButton** **مثال :** **MouseArea {** **acceptedButtons: Qt.LeftButton | Qt.RightButton** **}** **سوف يتم التفاعل مع أحداث زر الفأرة الأيمن و زر الفأرة الأيسر**
containsMouse : bool	**تعيد هذه الخاصيّة القيمة "true" في حال كان موقع مؤشر الفأرة داخل منطقة العنصر "MouseArea"**
drag.active : bool	**تعيد هذه الخاصيّة القيمة "true" في حال كان العنصر قيد عملية السحب**
drag.axis : enumeration	**تحديد المحور المسموح للعنصر سحبه عليه ,**

▪ **Drag.XAxis** **سماح السحب على المحور الأفقي فقط** ▪ **Drag.YAxis** **سماح السحب على المحور العمودي فقط** ▪ **Drag.XandYAxis** **سماح السحب على المحورين العمودي و الأفقي**	
تحديد أكبر مسافة على المحور الأفقي المسموح سحب العنصر عليها	**drag.maximumX : real**
تحديد أكبر مسافة على المحور العمودي المسموح سحب العنصر عليها	**drag.maximumY : real**
تحديد أصغر مسافة على المحور الأفقي المسموح سحب العنصر عليها	**drag.minimumX : real**
تحديد أصغر مسافة على المحور العمودي المسموح سحب العنصر عليها	**drag.minimumY : real**
تحديد العنصر المراد سحبه	**drag.target : Item**
في حال كانت القيمة "true" يعني تفعيل استقبال أحداث الفأرة "القيمة الإفتراضيّة" ,أما إذا كانت القيمة "false" عدم الإستجابة لأي حدث وارد من الفأرة كحدث النقر بزر الفأرة الأيسر .	**enabled : bool**
تفعيل أو عدم تفعيل أحداث حوم مؤشر الفأرة "تحريك مؤشر الفأرة فوق العنصر"	**hoverEnabled : bool**
تعيد موقع مؤشر الفأرة على المحور الأفقي	**mouseX : real**
تعيد موقع مؤشر الفأرة على المحور العمودي	**mouseY : real**
تعيد "true" في حال كان مضغوط حاليا	**pressed : bool**

بأحد أزرار الفأرة ضمن منطقة العنصر "MouseArea" المحددة , و إلا تعيد القيمة "false"	
ترجع ثابت تحوي أزرار الفأرة المضغوطة حاليا ضمن منطقة العنصر "MouseArea"	pressedButtons : MouseButtons

معالجات الأحداث لعنصر "MouseArea" :

يتم استدعائه عند إلغاء معالجة حدث ما تابع لأحداث الفأرة , كاستيلاء عنصر آخر على معالج حدث الفأرة قبل تتمة معالجة حدث ما ضمن العنصر الحالي	onCanceled
يتم استدعائه عند النقر بأحد أزرار الفأرة	onClicked (MouseEvent mouse)
يتم استدعائه عند النقر المضاعف بأحد أزرار الفأرة	onDoubleClicked (MouseEvent mouse)
يتم استدعائه عند دخول مؤشر الفأرة على منطقة العنصر "MouseArea" المحددة	onEntered
يتم استدعائه عند خروج مؤشر الفأرة من منطقة العنصر "MouseArea" المحددة	onExited
يتم استدعائه عند تغيير موقع مؤشر الفأرة الحالي	onPositionChanged (MouseEvent mouse)
يتم استدعائه عند الضغط بأحد أزرار الفارة ضمن منطقة العنصر "MouseArea" المحددة	onPressed
يتم استدعائه عند الضغط المتواصل "بعد مدة من الضغط قدرها 800 ميلي / ثا" بأحد أزرار الفارة ضمن منطقة العنصر "MouseArea" المحددة	onPressAndHold (MouseEvent mouse)
يتم استدعائه عند إفلات الضغط على زر	onReleased (MouseEvent mouse)

بالنسبة للوسيط " MouseEvent" له الخصائص التالية :

button : enumeration

buttons : int

الخاصيّة "button" و الخاصيّة "buttons" يرجعان احد العناصر الثلاث :

- Qt.LeftButton
- Qt.RightButton
- Qt.MiddleButton

يستخدمان لمعرفة أي زر من ازرار الفأرة قيد الضغط .

modifiers : int

المعدّل يستخدم لمعرفة إذا كان مفتاح من لوحة المفاتيح قيد الضغط عند الضغط على زر من أزرار الفأرة :

- Qt.NoModifier
- Qt.ShiftModifier
- Qt.ControlModifier
- Qt.AltModifier
- Qt.MetaModifier
- Qt.KeypadModifier

wasHeld : bool

يرجع "true" في حال كان زر الفأرة مضغوط من مدة قدرها "800" ميلي / ثا و ما فوق .

x : int

موقع المؤشر على المحور الأفقي .

y : int

موقع المؤشر على المحور العمودي .

مثال يوضح كيفيّة استخدام الوسيط "MouseEvent" :

```
MouseArea{

Anvhors.fill : parent

onClicked: {

if ( (mouse.button == Qt.LeftButton) && (mouse.modifiers & Qt.AltModifier) )

    txt.text = "X= " + mouse.x + " Y= " + mouse.y ;

}

}
```

الوسيط mouse من نمط "MouseEvent" .

عند الضغط على الفتاح "Alt" و الضغط بزر الفأرة الأيسر معا سوف يطبع نص يحوي موقع المؤشر الحالي للفأرة "لحظة الضغط" .

- **عنصر الحالة state :**

عنصر الحالة يستخدم من أجل تنفيذ عدة أوامر دفعة واحدة , يتم تفعيل عنصر الحالة إما باستخدام الخاصيّة "when" التابعة لعنصر الحالة , أو عند إعطاء قيمة الخاصيّة "state" اسم الحالة التي نود تفعيلها ضمن العنصر المحدد .

الخاصيّة "when" نقصد بها : طالما الحدث مازال ضمن حالة التنفيذ تبقى الحالة مفعّلة , و في حال الخروج من حلقة الحدث تصبح الحالة غير مفعّلة "أي ترجع إلى الحالة السابقة التي كانت عليها" .

الخاصيّة "state" : متى تمّ تفعيل الحالة بوساطة إعطاء اسم الحالة للخاصيّة "state" سوف تبقى الحالة مفعّلة .

الخاصيّة "name" هي اسم الحالة "معرّف الحالة" .

نستطيع كتابة عدة حالات لعنصر ما باستخدام الأقواس المربعة :

```
states : [
State { name: "S1"
} ,  State { name: "S2"
}, State { name: "S3"
}
]
```

انظر الرمّاز الموضح لكيفيّة كتابة مجموعة من الحالات و الموضح الفرق بين "when" و "state" :

```
import QtQuick 1.0


Rectangle{id:win
   width : 600; height : 600
   MouseArea{id:mouseArea
   anchors.fill: parent
   onClicked: {
      win.state = "s2"
   }}
   transitions: Transition {
      ColorAnimation {  duration: 1000 }
   }
   states :[
      State{name:"s1"; when : mouseArea.pressedButtons &
Qt.LeftButton
         PropertyChanges {
```

```
            target: win;color : "red"}
        },
        State{name:"s2";
           PropertyChanges {
            target: win;color : "blue"}
        }
    ]
}
```

- **عنصر التحول transition :**

عنصر التحول هو النقطة الفاصلة الواقعة بين القيمة القديمة و القيمة الحديثة لخاصيّة ما عند تغيير قيمتها , تنفذ هذه النقطة الفاصلة عند كل تغيير لقيمة الخاصيّة ضمن العنصر المحدد , بشرط ان تكون الخاصيّة التي تمّ تغيير قيمتها ضمن عنصر الحالة هي نفسها التي يتعامل معها عنصر التحول , كخاصيّة اللون في المثال السابق .

صيغته العامة :

```
transitions: [
Transition {
}, Transition{
}
]
```

الخاصيّة "from" : الحالة المراد بدء التحول منها .

الخاصيّة "to" : الحالة المراد إنهاء التحول فيها .

القيمة الإفتراضيّة لهما هي "*" تعني "من و إلى أي حالة" .

مثال للتوضيح :

```
import QtQuick 1.0
```

```
Rectangle{
   id: win
   width: 600; height: 600

Rectangle {
   id: rect
   width: 100; height: 100
   color: "red"

   MouseArea { id: mouseArea; anchors.fill: parent }

   states:[ State {
      name: "move"; when: mouseArea.pressed
      PropertyChanges { target: rect; x: 500 }
      PropertyChanges { target: rect; y: 500 }
   }
   ]

   transitions:[ Transition {
      from:"move" ;
      NumberAnimation  {properties: "x"; duration: 1000 }
   },Transition {
```

```
        to:"move"
        NumberAnimation {properties: "y"; duration: 1000 }
    }
  ]
 }
}
```

مثلما تلاحظ وجود حالة "state" تدعى "move" تنفذ عند الضغط بزر الفأرة الأيسر على العنصر المستطيل "rect" , مهمة هذه الحالة تغيير توضع العنصر "rect" على المحورين الأفقي و العمودي مسافة "500X500" بكسل بشكل آني , اما عنصري التحول "نقطة الإنتقال" مهمتهما وضع المدة الزمنية الخاصة بتغيير موضع العنصر "rect" على المحورين "X,Y" , مدة التحول هي ثانية ("1000" ميلي / ثا), الفرق بين عنصر التحول الأول و عنصر التحول الثاني هو الآتي :

عنصر التحول الأول ينتقل "من - from" عنصر الحال "move" من ثمّ يضع المدة الزمنيّة الخاصة بتغيير موضع العنصر "rect", بقول آخر عند الضغط على زر الفأرة سوف ينتقل العنصر "rect" مباشرة إلى الموقع 500 بكسل على المحور الأفقي , ثمّ عند إفلات زر الفأرة المضغوط سوف ينتقل العنصر "rect" إلى موضعه الأساسي بمدة زمنية قدرها 1 ثانية , أما عنصر التحول الثاني ينتقل "إلى - to" عنصر الحال "move" أي يضع المدة الزمنيّة الخاصة بتغيير موضع العنصر "rect" من ثمّ ينتقل العنصر "rect" إلى موضعه الأساس, بقول آخر عند الضغط على زر الفأرة بشكل متواصل سوف ينتقل العنصر "rect" بمدة زمنية قدرها "1 ثانية" إلى الموقع 500 بكسل على المحور العمودي , ثمّ عند إفلات زر الفأرة المضغوط سوف ينتقل العنصر "rect" إلى موضعه الأساسي بشكل مباشر .

نفذ التطبيق و اختبر النتيجة .

❖ كيفيّة إنشاء عنصر "Item" و مكون "Component" :

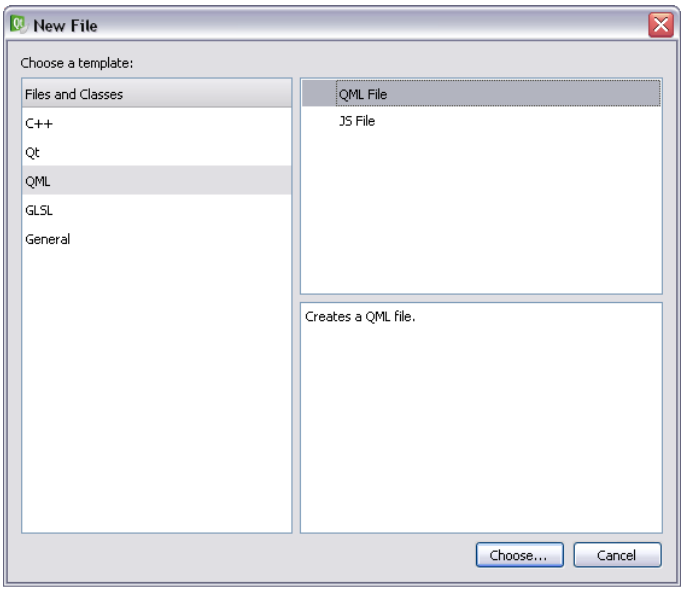
إذا كان لدينا رمّاز ما و نريد استخدامه في عدة مواقع ضمن مشروع "QML" بالتأكيد لا يتوجب علينا نسخه , نستطيع في "QML" أن نكتب الرمّاز لمرة واحدة ضمن كتلة عنصر "Item" أو "Component" و من ثمّ نستدعيه ليتم تنفيذه عند كل استدعاء له .

- **إنشاء عنصر "Item" :**

يجب أن يكون العنصر "Item" وحيد ضمن ملف "qml" , و يجب أن يبدأ اسم ملف "qml" بحرف كبير , يستخدم العنصر "Item" من أجل احتواء مكون أو أكثر , ليتم استدعائه من أجل تنفيذه فقط من خلال كتابة اسمه , فائدته هي عدم تكرار الرمّاز نفسه ضمن مشروع "QML" .

مثال لإنشاء عنصر دائري الشكل , واستدعاءه ضمن ملف مشروع "QML" الرئيسيّ , عند استدعاء العنصر سوف نقوم بتغيير لونه و موقعه .

أنشأ مشروع "Qt Quick UI" و سمه "test_item" , ثمّ انشأ ملف "qml" يدعى "Circle" , يتم إنشاء عنصر "qml" بالضغط بزر الفأرة الأيمن على ملف المشروع الموجود ضمن القائمة الشجرية الخاصة بعرض المشاريع المفتوحة حاليا , بعدها تظهر قائمة نختار منها "Add New" من ثمّ اختيار التبويب "QML" الموجود داخل النافذة التي ظهرت و اختيار العنصر "QML File" كما في الشكل (8.5).



الشكل 8.5

اكتب داخل الملف "Circle.qml" الرمّاز التالي :

```
import QtQuick 1.0

Item{

   id:circle
```

```
Rectangle {
   id:cir
   width: 100
   height: 100
   radius: 50
   color: "red"
}
}
```

من ثمّ اكتب داخل الملف "test_item.qml" السطر البرمجي الخاص باستدعاء العنصر "circle" :

```
Circle{ }
```

مثلما تلاحظ كتبنا اسم الملف الذي يحوي العنصر و ليس اسم العنصر , اسم العنصر أول حرف منه صغير "c", اما اسم الملف يبدا بحرف كبير "C" , نفذ التطبيق واختير النيجة , سوف يظهر ضمن النافذة دائرة حمراء اللون .

إذا أردنا تغيير موقع عنصر الدائرة "Circle" ضمن النافذة , فما علينا سوى تغيير قيمة الخاصية "X,Y" ضمن كتلة عنصر الدائرة :

```
Circle{x:100;y:100 }
```

لكن إذا كنّا نود تغيير لونها , لا نستطيع باستخدام نفس الطريقة السابقة , لأن خاصيّة اللون غير موجودة ضمن خصائص العنصر "Item" , الآن ما يتوجب علينا فعله ؟

نستطيع أن نعرّف عن خاصيّة ضمن العنصر "circle" مرتبطة بشكل مباشر بخاصيّة اللون , وذلك عن طريق استخدام الكلمة المفتاحيّة "property" , اذهب إلى الملف "Circle.qml" وعدّل رمّازه ليصبح كالأتي :

```
import QtQuick 1.0
Item{
   id:circle
```

```
    property alias color : cir.color
Rectangle {
    id:cir
    width: 100
    height: 100
    radius: 50
    color: "red"
}
}
```

أضفنا فقط سطر برمجي وحيد لتعريف خاصيّة تدعى "color" , و التي تعد من نمط وكيل "alias" , أي وكيل عن الخاصيّة الداخليّة التي نود تغيير قيمتها من خارج مجال رؤيتها , "alias" تكافئ وسيط متحول بالمرجع , الآن اذهب إلى الملف "test_item.qml" و عدل إستدعاء العنصر "Circle" ليصبح كالآتي :

```
Circle{x:100;y:100;color: "blue" }
```

نفذ التطبيق سوف ترى أن لون عنصر الدائرة أصبح أزرق .

- التصريح عن خاصيّة :

شكل الخاصيّة العام :

```
property type name : value
```

مثلما ذكرنا يمكن ان يكون نمط الخاصيّة "alias" , و أيضا الانماط الموجودة بالجدول :

```
Action
Bool
Color
Date
Double
```

Enumeration

Font

Int

List

Point

Real

Rect

Size

String

Time

url

Variant

vector3d

ملاحظة : جميع الأنماط الظاهرة ضمن الجدول يقابلها أمثالها في " Qt C++ Framework", كـ "url" يكافئ "QUrl" .

مثلا لتعريف خاصيّة قيمتها من نمط "string" نصي اسمه "firstName" يتوجب علينا أن نصرح عن خاصيّة كالآتي :

property string firstName : ""

- **إنشاء مكون "Component" :**

إذا كنا نريد إنشاء عدة مكونات "Components" ضمن نفس العنصر "Item", يتوجب علينا استخدام العنصر "Component" , مثال لإنشاء إثنان من المكونات الأول شكله مربع و الثاني دائري , و بعدها نحمل "نستدعي" هذين المكونين بوساطة العنصر "Loader" ,

أنشأ مشروع "Qt Quick UI" يدعى "components" , أنشأ داخله ملف يدعى "Comps.qml" , و أكتب داخله الرمّاز الخاص بإنشاء المكونين "الدارة و المربع" :

```
import QtQuick 1.0
Item{
   Component{id:square
Rectangle {
   width: 100
   height: 100
   color: "red"
    }
   }

   Component{id:circle
Rectangle {
   width: 100
   height: 100
   radius: 50
   color: "blue"
    }
   }
   Loader{sourceComponent: square;}
   Loader{sourceComponent: circle;}
```

}

أنشأنا عنصر "Item" , داخله اثنان من المكونات , الأول يدعى "square" مربع الشكل و الثاني "circle" دائريّ الشكل , من ثمّ تم تحميلهم بوساطة العنصر "Loader" , والذي مهمته تحميل مكون و تهيئته من ثمّ تنفيذه .

إذا أردنا أن نغيير قيمة خاصيّة مكون نستطيع ذلك بوساطة كتابة اسم الخاصيّة ضمن كتلة عنصر "Loader" كالآتي :

```
Loader{sourceComponent: circle; x : 100; y : 100;}
```

يجب أن نكتب أسم المكون المراد تحميله ضمن الخاصيّة "sourceComponent" التابعة للعنصر "Loader" , في حال كان ملف العنصر الذي يحوي المكونات المطلوب تحميلها في ملف منفصل , أو خارج شجرة مشروعنا الحالي , يوجب علينا استخدام الخاصيّة ": source url" التابعة للعنصر "Loader" , ضمن العنصر "Loader" يوجد معالج حدث وحيد يدعى "onLoaded" و الذي يرفع عند الإنتهاء من تحميل المكون المطلوب تنفيذه .

الآن يجب علينا استدعاء العنصر "Comps" الذي يحوي المكونين "الدائرة - المربع" , اذهب إلى الملف "components.qml" و اكتب الرمّاز التالي :

```
Comps{Component.onCompleted: console.log("had been loaded") }
```

معالج الحدث "onCompleted" يرفع عند الإنتهاء من تحميل المكونات , ويوجد أيضا معالج حدث آخر تابع للعنصر "Component" يدعى "onDestruction" و الذي يرفع عند تهديم "إزالة" المكونات .

❖ الأحداث الداخليّة "signals" و معالجاتها "Handlers":

كما في "Qt" أحداث داخليّة "signals" كذلك في "QML" , و التي تتلخص مهمتها رفع حدث من كائن ما و من ثمّ استدعاء الأجراء المتصل بهذا الحدث و الذي يسمى في "QML" معالج حدث "Handler" لتنفيذ سلسلة الأوامر الموجودة داخل كتلته .

ملاحظة : معالج الحدث "handler" في "QML" يكافئ المستقبل "Slot" في "Qt" .

الشكل العام لتعريف حدث داخلي "signal" :

signal <name> (<type> <parameter name>, <type> <parameter name>,...)

- **مثال يوضح كيفيّة إنشاء و استخدام حدث داخلي "signal" :**

يحوي هذا المثال على اثنان من الأحداث الداخليّة الأول يرفع كل ثانية ليطبع داخل عنصر النص "Text" الرقم الحالي الموجود داخل الخاصيّة "num", أما الثاني يرفع عندما تصبح قيمة الخاصيّة "num" مساوية 10 , ليوقف عمل المؤقت الخالص برفع الحدث الأول كل ثانية , انظر الرمّاز :

```
import QtQuick 1.0


Rectangle{
   id: win
   width: 100; height: 100
   property int num: 0


   signal changePerSec(string txt)
   signal stop()
   onStop: timer.stop()


   onChangePerSec: { displayNum.text = txt; }
   Text {
      id: displayNum
      font.pixelSize: 22
```

```
    anchors.centerIn: parent
    color: "blue"
    text: qsTr("")
  }
  Timer{id:timer
  interval: 1000;running: true;repeat: true

  onTriggered: {
    increment()
    changePerSec (num)
  }
  }
  function increment(){
   num++;
    if(num == 10 ) stop()
  }

}
```

أنشئنا خاصيّة قيمتها من نمط "int" تدعى "num" و التي سوف تحوي الرقم التزايدي , ثمّ عرّفنا حدث داخلي يدعى "changePerSec(string txt)" , مثلما تلاحظ يحوي وسيط من نمط نصي , بعدها عرّفنا حدث آخر يدعى "stop()" , والذي مهمته ايقاف المؤقت "timer" الخاص برفع الحدث "changePesSec" كل ثانية , حققنا معالج الحدث "stop()" :

```
onStop :  timer.stop()
```

ملاحظة : عند إنشاء حدث داخلي سوف تقوم "QML" بشكل تلقائي إنشاء معالج لهذا الحدث ,حيث يبدأ اسمه بـ "on" ثمّ اسم الحدث , أول محرف منه يكون محرف كبير .

- ***الكلمة المفتاحيّة "var" :***

ملاحظة : نستطيع تبديل تعريف الخاصيّة "num" بـ تصريح عن متحول يدعى "num" كالآتي :

```
var num = 0;
```

استخدمنا الكلمة المفتاحيّة "var" للتصريح عن متحول , سوف يأخذ المتحول بشكل تلقائي نمط مناسب لقيمته .

لنعد إلى ما كنا فيه ..

معالج الحدث "onChangePerSec" مهمته كتابة الرقم الممرر للوسيط "txt" , أنشئنا عنصر نص لكتابة الرقم داخله , عنصر المؤقت "Timer" , يتم تنفيذ كتلة تعليماته كل "1000" ميلي / ثا , كتلة التعليمات تكتب ضمن جسم معالج الحدث "onTriggered" :

```
onTriggered: {
   increment()
   changePerSec (num)
}
}
```

أولا يستدعي الإجراء "increment" و الذي تتلخص مهمته بـ زيادة الخاصيّة "num" بمقدار واحد , و اختبار فيما إذا قد أصبحت قيمة هذه الخاصيّة 10 ليرفع الحدث "stop" ليتم إيقاف المؤقت .

الإجراء "increment" :

```
function increment(){
 num++;
   if(num == 10 ) stop()
}
```

نفذ التطبيق سوف يظهر كما في الشكل (8.6) :

الشكل 8.6

- **إنشاء إتصال بين حدث داخلي و إجراء :**

في حال أردنا إنشا معالج حدث خاص بنا , يجب علينا ان ننشأ إجراء ثمّ ننشأ إتصال بين الحدث و هذا الإجراء :

```
function increment(num) { }

signal addNum(string num)

addNum.connect(increment)
```

مثال سوف ننشأ نافذة عند النقر عليها يتم زيادر خاصيّة تدعى "num" بمقدار واحد :

```
import QtQuick 1.0


Rectangle{
   id: win
   width: 100; height: 100
   property int num: 0


   signal addNum(string num)
   Component.onCompleted:
     win.addNum.connect(increment)
```

```
function increment(num){
    txt.text = qsTr(num)
}
Text {
    id: txt
    font.pixelSize: 22
    anchors.centerIn: parent
    text: qsTr("")
}
MouseArea{
anchors.fill: parent
onClicked: {num++;
    addNum(num)}
}

}
```

- **إنشاء إتصال بين حدث داخلي و حدث داخلي آخر :**

منذ قليل رأينا كيفيّة إنشاء إتصال بين حدث داخلي و إجراء كـ "signal - slot" ضمن "Qt" , الآن إذا أردنا أن يتصل حدث داخلي بحدث داخلي آخر , بحيث عند رفع الحدث الداخلي المتصل به يرفع معه الحدث الآخر :

```
signal alarm()
```

```
onAlarm : console.log("beep")

focus: true

Keys.onPressed: {

    if (event.key == Qt.Key_C) {

        mouseArea.clicked.connect(alarm)

        event.accepted = true;

    }

}


MouseArea{id:mouseArea

    anchors.fill: parent

}
```

سوف يتم إتصال الحدث "alarm" بالحدث "clicked" عند الضغط على مفتاح "C" , بعدها كل نقرة بزر الفأرة الأيسرعلى النافذة سوف تستدعي الحدث "alarm" , بالتالي سيطبع النص "beep" , نفذ التطبيق و اختبر النتيجة .

استخدمنا لإستقبال أحداث لوحة المفاتيح العنصر "Keys" , تم التقاط المفاتيح بوساطة الحدث "onPreseed" التابع له , الوسيط "event" تابع للحدث "pressed" , اختبرنا المفتاح المضغوط من خلال استدعاء المنهج "key" الذي يعيد ثابت يدل على الفتاح الذي قد تمّ ضغطه , شكل الثابت :

Qt.Key_ character

إذا كنا نريد استقبال أحداث مفاتيح الأسهم نستطيع ذلك لوساكة العنصر "KeyNavigation" كالتالي :

```
KeyNavigation.right: doWhatever()

KeyNavigation.left: doWhatever()

KeyNavigation.up: doWhatever()
```

KeyNavigation.down: doWhatever()

KeyNavigation.tab: doWhatever()

السطر البرمجي الأخير يدل على ان المفتاح "tab" قد تمّ ضغطه .

لنعد إلى ما كنا فيه , الأحداث و معالجاتها و كيفيّة إجراء إتصالات بينها ..

- العنصر "Connections" :

في حال كنا نريد ان نجري إتصال بين معالج حدث خارج كتلة "مجال رؤية" مرسل الحدث ما يتوجب علينا فعله ؟

في "QML" يوجد عنصر إتصالات يدعى "Connections" , نستطيع بوساطته حل المشكلة السابقة كالآتي :

```
MouseArea{

Id:mouseArea

anchors.fill: parent

}

Connections {

target : mouseArea

onDoubleClicked: console.log('connection outside scope signal sender')

}
```

الخاصيّة "target" تستخدم من أجل تحديد العنصر الذي نود إجراء إتصال بأحداثه الداخليّة من خارج مجال رؤيته.

❖ معرفة التوجه الحالي للجهاز :

التوجيه : أكثر أجهزة الجوال الحديثة تحوي حساسات , كحساس البوصلة " Sensor Of Compass" و حساس اللمس " Sensor Of Touch" و أيضا حسّاس التوجيه " Sensor

Of Orientation" و الذي مشتق من حسّاس البوصلة "Sensor Of Compass" , بالنسبة لحساس التوجيه يوجد نوعين منه ,النوع الأول يدعم التدوير لـ الأربع جهات , و النوع الثاني يدعم التدوير لـ جهتين , في حال أردنا معرفة التوجيه الحالي من خلال كتابة رمَاز "QML" نستخدم المكون "runtime" و نستدعي المنهج "orientation" منه كالآتي :

runtime. orientation

حيث سيرجع قيمة التوجيه الحالي و هي قيمة أحد هذه الأربع ثوابت المبينة , انظر الجدول :

توجيه أفقي , الأعلى هي الجهة اليمنى	**Orientation.Landscape**
توجيه عمودي , الأعلى هي الجهة العليا	**Orientation.Portrait**
توجيه أفقي معكوس , الأعلى هي الجهة اليسرى	**Orientation.LandscapeInverted**
توجيه عمودي معكوس , الأعلى هي جهة الأسفل	**Orientation.PortraitInverted**

انظر الرمّاز التالي الخاص بتطبيق عند النقر على واجهته يكتب في منتصف شاشته التوجيه الحالي للجهاز :

```
import QtQuick 1.0


Rectangle {
   width: 360
   height: 360
   Text {id:txt
      anchors.centerIn: parent
   }
   MouseArea {
      anchors.fill: parent
```

```
    onClicked: {
        switch(runtime.orientation)
        {
        case Orientation.Landscape :txt.text = "Landscape" ; break;
        case Orientation.Portrait :txt.text = "Portrait" ; break;
        case Orientation.LandscapeInverted :txt.text =
"LandscapeInverted" ; break;
        case Orientation.PortraitInverted :txt.text = "PortraitInverted" ;
break;
        }
    }
  }
}
```

نفذ التطبيق من خلال برنامج "QML Viewer" و اختبر النتيجة من خلال الذهاب إلى القائمة "Settings" ثمّ "Properties" و اختر التوجيه الذي تريد ثمّ انقر بزر الفأرة على واجهة التطبيق و لاحظ كيف سيكتب ضمن التطبيف التوجيه الذي اخترته .

❖ تكامل Java Script مع QML :

QML تعتمد بشكل كبير على لغة Java Script , حيث أغلب مهام هذه اللغة تنفذ بوساطة دمج رمّاز Java Script معها , رأينا ضمن الصفحات السابقة كيفيّة تعريف إجراء و كيفيّة التصريح عن متحول و إنشاء حلقة التكرار و الإختبار الشرطيّ وذلك تمّ بوساطة كتابة رمّاز "Java Script"مدمج , ضمن "QML" نستطيع كتابة رمّاز "JS" مدمج خطّي أو ضمن ملف "JS" منفصل , و يسمح الوصول من داخل ملفات "JS" الموجودة ضمن ملفات المشروع إلى عناصر "QML" الموجودة ضمن ملفات المشروع أيضا , و بالتالي نستطيع تغيير جميع خصائص هذه العناصر بشكل مباشر .

- **دمج منهج Java Script خطّي ضمن QML :**

من اجل توسيع الأفعال التي تقوم بها QML تمّ بنائها على أساس التكامل مع لغة Java Script , هذا التكامل إما داخل رمّاز QML أو داخل رمّاز Java Script بدمج رمّاز QML ضمنه , لنرى كيفيّة دمج منهج Java Script خطّي ضمن QML و استدعائه :

```
Item {
   id: item
   function func(arg1, arg2){
      //doWhatever
   return ret;
   }

   Component.onCompleted: txt.text = func(2,200)

}
```

هذا الشكل العام لـ منهج "JS" خطّي مدمج مع "QML" , مثال لإنشاء منهج "JS" خطّي يدعى "sqrt" لحساب الجذر التكعيبي لرقم ممرر لوسيطه , من ثمّ استدعاء هذا المنهج و عرض ناتجه :

```
import QtQuick 1.0

Rectangle {id:win
   width: 360
   height: 360

   function sqrt(num){
```

```
    var result = 0 ;
    result = Math.sqrt( parseFloat(num) ) ;
  return result;
  }

  Text {id:txt
    anchors.centerIn: parent
  }
  MouseArea {
    anchors.fill: parent
    onClicked: {
      txt.text = sqrt(10)
    }
  }
}
```

استخدمنا منهج "JS" التي مهمته إعطاء نتيجة الجذر التربيعي للرقم المعطى لوسيطه حيث تكون النتيجة من نمط "float" , استخدمنا المنهج "parseFloat" التابع للغة "JS" من أجل تحويل الرقم المعطى للوسيط إلى رقم من نمط "float" , تم استدعاء المنهج "sqrt" من خلال كتابة اسمه فقط و تمرير لوسيطه الرقم المراد حساب جذره التربيعي .

- **استخدام ملف JS منفصل ضمن QML :**

كل ما علينا من أجل استخدام منهج JS ضمن QML موجود ضمن ملف منفصل هو استيراد ملف JS ضمن QML و إعطاء اسما مستعار له و ذلك كالآتي :

```
import "math.js" as JSMath
```

اسم ملف JS المراد هو "math.js" و اسمه المستعار "JSMath" حيث عندما نريد استدعاء منهج منه يتوجب أن نكتب أولا اسمه المستعار "JSMath" ثمّ عامل النقطة "." أخير اسمه المنهج الذي نود استدعائه منه .

ملاحظة : يجب أن يبدأ الإسم المستعار بمحرف كبير .

لننشأ نفس المثال السابق لكن سنضع المنهج "sqrt" ضمن ملف "JS" منفصل , أنشأ مشروه "Qt Quick UI" من ثمّ أضف ملف "JS" للمشروع و سمّه "math.js" و اكتب المنهج "sqrt" داخله :

math.js file

```
function sqrt(num){
   var result = 0 ;
   result = Math.sqrt( parseFloat(num) ) ;
return result;
}
```

ضمن ملف "QML" اكتب الآتي :

```
import QtQuick 1.0
import "math.js" as JSMath

Rectangle {
   width: 360
   height: 360
   Text {id:txt
      anchors.centerIn: parent
   }
```

```
    MouseArea {
        anchors.fill: parent
        onClicked: {
            txt.text = JSMath.sqrt(22.2)
        }
    }
}
```

ضمّنا الملف "math.js" ثمّ أعطيناه اسما مستعارا من خلال استخدام الكلمة المفتاحيّة "as" و اتباعها بالأسم المستعار , استدعينا المنهج "sqrt" من خلال كتابته بعد الأسم المستعار و عامل النقطة الملحقة به "JSMath.sqrt(num)" , نفذ التطبيق و اختبر النتيجة .

- **الوصول إلى مكون QML من ملف JS مضمّن :**

في حال أردنا أن نصل إلى عنصر QML من منهج موجود ضمن ملف JS منفصل فكل ما يتوجب علينا فعله هو كتابة اسم العنصر "المكون" المحدد و استخدام معامل النقطة من أجل الوصول إلى منهج أو خاصيّة له .

للتوضيح عدّل المثال السابق كالآتي :

أولا داخل ملف "math.js" عدّله ليصبح كالآتي :

```
function sqrt(num){
    var result = 0 ;
    result = Math.sqrt( parseFloat(num) ) ;
    txt.text = result ;
}
```

لاحظ كيف تعاملنا مع عنصر "Text" كأننا موجودين داخل ملف "QML" , داخل ملف "QML" عدّل فقط كتلة العنصر "MouseArea" :

```
MouseArea {
    anchors.fill: parent
    onClicked: {
        JSMath.sqrt(90)
    }
}
```

تمّ استدعاء المنهج "sqrt" مباشرة لـ تصبح قيمة العنصر "txt" مساوية للجذر التربيعي لقيمة الوسيط الممررة .

في أردنا منع الوصول إلى مكونات "QML" من داخل ملف "JS" مضمّن فتوجب علينا أن نكتب السطر البرمجي التالي في بداية رمّاز "JS" :

```
.pragma library
```

جرّب أن تضع هذا السطر البرمجي داخل الملف "math.js" السابق و نفذ التطبيق و اختبر النتيجة ؟

الرمّاز :

```
.pragma library


function sqrt(num){
    var result = 0 ;
    result = Math.sqrt( parseFloat(num) ) ;
    txt.text = result ;
}
```

سوف تلاحظ عدم تغيير قيمة العنصر "txt" لأنه قد تمّ منع الوصول إلى عناصر "QML" من داخل ملف "JS" المكتوب داخله هذا السطر البرمجي .

- **ربط حدث QML داخلي بمنهج JS :**

من أجل استقبال أحداث "QML" من داخل ملف "JS" مضمّن , يتوجب علينا أن نربط الحدث المحدد مع المنهج المراد استدعاءه عند رفع الحدث , و هذ يتم برمجيا كالآتي :

```
Component.onCompleted: {

  mouseArea.clicked.connect( JSMath.sqrt(9) )

}
```

استخدمنا المنهج "connect" و مررنا لوسيطه اسم منهج "JS" المحدد من أجل ربطه بـ حدث النقر .

- **تضمين ملف JS داخل ملف JS آخر :**

من أجل تضمين ملف "JS" داخل ملف "JS" مضمّن , يتوجب علينا أن نستدعي المنهج الآتي :

```
Qt.include("fn.js")
```

في هذه الحال ستصبح جميع مناهج الملف "fn.js" متاحة داخل ملف "JS" الحالي , و بالتالي سوف يكونو متاحين أيضا ضمن ملف "QML" المضمّن لـ ملف "JS" المحدد , انظر الشكل العام :

file1.js

```
func1(){}

func2(){}
```

file2

```
Qt.include("file1.js")

func3(){}

func4(){}
```

main.qml

```
import "file2.js" as JS

JS.func1()

JS.func2()

JS.func3()

JS.func4()
```

لاحظ كيف استدعينا المنهج "func1" و المنهج "func2" التابعين للملف "file1.js" , بذلك نكون انتهينا من الفصل الحالي و الذي يدعى "أساسيّات في QML" .

❖ المسالك في QML :

من أجل معالجة بيانات ضمن مسلك منفصل عن المسلك "الأب" الرئيسي لتطبيق "QML" , يتوجب علينا استخدام العنصر "WorkerScript" , الذي يسمح لنا معالجة البيانات داخل مسلك منفصل , حيث يجب أن يكون المنهج الرئيسي للمسلك ضمن ملف "JS" , منة ثمّ استدعائمه بوساطة العنصر "WorkerScript" .

يوجد لهذا العنصر خاصيّة وحيدة و هي "source" التي تسمح بتحديد ملف "JS" الموجود داخله منهج المسلك الرئيسي , و يملك هذا العنصر معالج حدث وحيد يدعى "onMessage(jsobject msg)" الذي يستدعى عند قدح الإستجابة من منهج المسلك الرئيسي حيث هذه الإستجابة مضمّنة داخل الوسيط "msg" , و أخيرا يملك منهج وحيد يدعى "sendMessage(jsobject msg)" خاص بـ تمرير رسالة بين المسلك الرئيسي و بين المسلك المنفصل , النمط "jsobject" يستطيع أخذ أحد قيم الأنماط التالية :

قيمة بوليانيّة "true - false"	**boolean**
عدد	**number**
نص	**string**
كائنات لغة جافا سكربت كـ "Date"	**JavaScript Object**

مصفوفة جافا سكربت	arrays
مكون "QML" نموذج قائمة	ListModel
جميع كائنات "*QObject" المتبقيّة غير متاحة	non QObject*

ملاحظة : غير مسموح بالوصول المباشر لـ مكونات "QML" من داخل المنهج الرئيسيّ للمسلك .

مثال توضيحيّ لتطبيق داخله حلقة تكرار القيمة العظمى له "9999999" داخل كتلة الحلقة سطر برمجي مهمته عرض قيمة الحلقة حاليا , أي عند كل زيادة لقيمة معرّف حلقة التكرار يعرض قيمة المعرف مباشرة ضمن كائن النص "Text" .

أنشأ مشروع "Qt Quick UI" و سمّه "workerScript" , ثمّ اكتب داخل الملف "workerScript.qml" الرمّاز التالي :

```
import QtQuick 1.0

Rectangle {
  id:win
  width: 300; height: 300

  Text { id: txt }

  MouseArea {
    anchors.fill: parent
    onClicked: {
      for(var i = 0 ; i <= 9999999 ; i ++)
```

```
            txt.text = i
        }
    }
}
```

عند تنفيذ التطبيق ستلاحظ عدم عرض الرقم الحالي لمعرف حلقة التكرار و في حال حاولة تحريك النافذة فلن تستطيع تحريكها حتى يتم الإنتهاء "الخروج" من حلقة التكرار , أي سيصبح التطبيق في حالة تجمد حتى الخروج من الحلقة , لحل هذه المشكلة يتوجب وضع حلقة التكرار ضمن مسلك منفصل عن المسلك الرئيسي , و ذلك باستخدام العنصر "WorkerScript" , عدّل الرمّاز السابق ليصبح كالآتي :

```
import QtQuick 1.0

Rectangle {
    id:win
    width: 300; height: 300

    Text { id: txt }

    WorkerScript {
        id: thread
        source: "thread.js"

        onMessage: txt.text = messageObject.result
    }

    MouseArea {
```

```
        anchors.fill: parent
        onClicked: {
          thread.sendMessage({'maxNum':9999999})
        }
    }
}
```

أضفنا عنصر "WorkerScript" يدعى "thread" , و اسندنا لخاصيّته "source" القيمة "thread.js" و الذي يمثّل ملف المسلك المنفصل , ضمن معالج الحدث "onMessage" أضفنا السطر البرمجي الخاص بإسناد قيمة حلقة التكرار الحالية ضمن عنصر النص , الكائن "messageObject" من نمط "QVariantMap" يحوي جميع القيم الممررة من المسلك المنفصل إلى المسلك الرئيسي , داخل كتلة معالج النقر لزر الفأرة أضفنا السطر البرمجي الخاص بإرسال قيمة أكبر رقم تصل إليه حلقة التكرار :

```
thread.sendMessage({'maxNum':9999999})
```

حيث اسم القيمة "معرّفها" هو "maxNum" و القيمة هي "9999999" نضع القيمة بعد معامل النقطتين الفوقيتين , الشكل العام لإسناد قيمة لمعرّف جديد :

```
{'id' : value}
```

الآن أضف ملف "JavaScript" و اكتب داخله الرمّاز التالي :

```
WorkerScript.onMessage = function(message) {
  for(var i = 0 ; i <= message.maxNum ; i ++)
    WorkerScript.sendMessage({ 'result': + i})
}
```

أضفنا معالج الحدث "onMessage" الذي يستدعى عند تنفيذ المنهج "sendMessage" التابع للعنصر "WorkerScript" , الوسيط "message" يحوي القيم المرسلة من المسلك الرئيسي , من أجل الوصول إلى قيمة ما يتوجب علينا كتابة اسمها بعد معامل النقطة للوسيط "message" :

```
message.maxNum
```

حيث "maxNum" أرسلنا قيمته للمسلك المنفصل بوساطة المنهج "sendMessage" , الآن نفذ التطبيق و اختبر النتيجة , ستلاحظ عدم إحالة التطبيق لحالة التجميد .

❖ تصدير ملف QML من برنامج Photoshop CS4 و ما فوقه من إصدارات :

نستطيع ان ننشأ ملف QML من خلال تصميم و تأطير صورة في برنامج Photoshop , حيث سيحوي هذا الملف على إطارات الصور مع خطّاط "QML" التفاعلي , بالنسبة للإطارات فهي كـ مثيل صفحة الويب عند تصميمها على برنامج Photoshop .

حمل البرنامج الفرعي "Export QML.jx" من الرابط :

http://qt.gitorious.org/qt-labs/photoshop-qmlexporter/trees/master

ثمّ افتح البرنامج الفرعي بعد تحميله ليضيف أمر التصدير بصيغة "QML" للقائمة الفرعية "Scripts" الموجودة داخل القائمة "File" لبرنامج Photoshop .

من أجل تصدير المشهد الحالي الموجودة ضمن برنامج Photoshop لـ ملف QML , نذهب إلى القائمة File ثمّ القائمة الفرعيّة Scripts و نختير الأمر "Export to QML" .

❖ خلاصة الفصل :

تكلمنا بالتفصيل حول تعريف عنصر في QML و وصفه و عن تفاعل عناصر واجهة المستخدم ,و عن كيفية استيراد مكتبات JS خارجية و استخدام مناهجها, و أيضا تكلمنا في هذا الفصل عن كيفية انشاء كائن "Component" و عنصر "Item" جديد و الفرق بينهما, و عن كيفية إنشاء الأحداث .

إلى هنا نكون قد انتهينا من فصل أساسيّات في "QML" , لننتقل إلى فصل عناصر واجهة المستخدم .

الفصل العاشر

عناصر واجهة المستخدم

❖ مقدمة Into :

تتألف الطبقة الأولى من واجهة المستخدم من عناصر الإدخال كـ صناديق النص و الأزرار و عناصر العرض كـ الجداول و القوائم و اللافتات , أما الطبقة الثانية فهي ترشيح المدخلات و التحقق من صحتها.

يحوي هذا الفصل كيفية كتابة رمّاز لتصميم واجهة مستخدم احترافية المؤلفة من طبقتين : طبقة طبقة عناصر الإدخال و العرض و طبقة ترشيح المدخلات و التحقق من صحتها .

❖ عناصر الإدخال :

يوجد عنصرين إدخال ضمن "QML" و هما عنصر الإدخال السطري "TextInput" , و عنصر الإدخال المتعدد "TextEdit" , كلا العنصرين يرثان العنصر "Item" .

- العنصر TextInput و التي مهمته عرض و تحرير نص غير متعدد السطور , أهم خصائصه :

لكتابة نص داخل العنصر "TextInput" , أو قراءة النص المكتوب داخله	text : string
عند إعطاء القيمة "true" لهذه الخاصيّة سوف تصبح مهمة العنصر "TextInput" هي عرض النص فقط , من دون السماح بتحريره	readOnly : bool
وضع نمط المحاكاة لعنصر الإدخال , انظر الفصل الثاني التابع لقسم "Qt" , التعداد QLineEdit::EchoMode	echoMode : enumeration
سوف تتفاعل هذه الخاصيّة مع نمط محاكاة كلمة مرور ,حيث سوف تظهر المحارف المكتوبة داخل عنصر الإدخال كلها على شكل	passwordCharacter : string

المحرف الموضوع لهذه الخاصيّة

يحتوي العنصر "TextInput" على عدة مناهج , أهمها :

نسخ النص المحدد إلى حافظة النظام	**copy**
قص النص المحدد إلى حافظة النظام	**cut**
لصق النص الموجود ضمن حافظة النظام إلى عنصر الإدخال	**paste**
تحديد كامل النص الموجود داخل عنصر الإدخال	**selectAll**

يوجد داخل العنصر "TextInput" معالج حدث وحيد وهو "onAccepted" و الذي يتم استدعائه عند الضغط على مفتاح "Enter" .

مثال توضيحيّ :

```
import QtQuick 1.0


TextInput{
   id:textLine
   width: 150;height: 50
   text: qsTr("Hasan Al- Morhej ")
   readOnly: true
   font.bold: true
   color: "green"
   onAccepted: {
      text = text + qsTr("* learning Qt")
```

```
    }
}
```

عند تنفيذ التطبيق سوف يظهر النص "Hasan Al-Morhej" داخل عنصر الإدخال بلون أخضر , بنمط خط عريض, تحرير النص غير مسموح و ذلك بسبب وضع القيمة "true" للخاصيّة "readOnly" ,عند الضغط على مفتاح "Enter" سوف يضاف النص " * learning Qt" , للنص السابق , انظر الشكل (9.1) :

الشكل 9.1

جرب أن تزيل السطر البرمجي "readOnly : true" , واختبر النتيجة .

- العنصر TextEdit و التي مهمته عرض و تحرير نص غني متعدد السطور , أهم خصائصه هي نفس خصائص العنصر السابق , و أيضا مناهجه كلذلك الأمر , اما بالنسبة لمعالج الحدث فيختلف , هما يملك عنصر الإدخال المتعدد معالج حدث وحيد وهو "onLinkActivated" و الذي يستدعى عند النقر على نص إرتبط تشعبي ,*هذا المعالج موجود في الوحدات النمطية "QtQuick" التي يكون إصدارها "1.1" و اصداراتها الأحدث*, انظر الرمّاز :

```
import QtQuick 1.1


TextEdit{
    id:textEdit
    width: 200;height: 100
    text: "<a href='http:// www.plutoit.co.uk ' >PlutoIT</a>"
    readOnly: true
    font.bold: true
    color: "green"

    onLinkActivated:Qt.openUrlExternally("http://www. plutoit.co.uk ")
```

}

عند الضغط على الإرتباط التشعبي الظاهر ضمن عنصر الإدخال سوف يفتح صفحة ويب على الموقع الإلكتروني الخاص بدار شعاع .

❖ عناصر التحقق من صحة الإدخال :

نستخدم عناصر التحقق من صحة الإدخال "Validators" من اجل اختبار و تقييد النص المدخل لعنصر إدخال ما , تملك "QML" , ثلاث عناصر خاصة بالتحقق من صحة الإدخال و هي :

1 - IntValidator : لإعطاء سماحية إدخال الأعداد الصحيحة المتدرجة ضمن مجال محدد .
2 - DoubleValidator : لإعطاء سماحية إدخال الأعداد الحقيقيّة المتدرجة ضمن مجال محدد .
3 - RegExpValidator : إعطاء سماحية الإدخال حسب نص التعبير القياسي , راجع الفقرة "الكائن QRegExpValidator" الموجود ضمن الفصل الثاني من قسم "Qt" .

مثال , نص إدخال مسموح داخله إدخال الأعداد الصحيحة الموجودة ضمن المجال " 10 - 100" :

```
import QtQuick 1.0


  TextInput{

    focus: true

  validator:
```

```
    IntValidator {
    bottom: 10;top: 100
    }
}
```

وضعنا العنصر "IntValidator" ضمن كتلة الخاصيّة "validator" الموجودة ضمن أي عنصر إدخال في "QML" .

بالنسبة للعنصر "RegExpvalidator" يملك خاصيّة تدعى "regExp" , يجب أن تكون قيمتها من نمط "RegExp" الذي بدوره يحوي نص التعبير القياسيّ .

❖ عناصر العرض التوسّعيّة :

ذكرنا سابقا كيفيّة عرض نص و صورة و عنصر مستطيل بشكل مفرد , في حال أردنا عرض عدة نصوص أو صور أو عناصر مستطيل الخ.. بجانب بعضها البعض بتنسيق معين , كعرضهم بشكل متتالي أفقيا , أو عرضهم كأنهم مرصوفون ضمن جدول الخ.. , يتوجب علينا استخدام أحد العناصر التالية :

1- Column : يستخدم من أجل ترتيب العناصر الحاوي لها بشكل عمودي .
2- Row : يستخدم من أجل ترتيب العناصر الحاوي لها بشكل أفقي .
3- Grid : يستخدم من أجل ترتيب العناصر الحاوي لها بشكل شبكي "تعرض كأنها خلايا ضمن جدول" .
4- Flow : يستخدم من أجل ترتيب العناصر الحاوي لها بشكل يناسب حجم الأب الحاوي له .

- أهم خصائص العنصر "Column" :

مقدار التباعد بين العناصر الأبناء مقدرا بالبكسل	spacing : int
تستخدم من أجل تطبيق تحول عند إضافة عنصر جديد	add : Transition

move : Transition **تستخدم من أجل تطبيق تحول عند تغيير موقع عنصر ابن**

مثال , نافذة تحوي ثلاث دوائر مرصوفين بشكل متتالي على المحور العمودي , عند تنفيذ التطبيف سوف يتم تحريك الدوائر الثلاث من النقطة العليا من النافذة إلى موقعهن بمدة زمنية قدره 2 ثا , انظر الرمّاز :

```
import QtQuick 1.0


Column {
   spacing: 5
   Rectangle { color: "silver"; width: 100; height: 100;radius: 50 }
   Rectangle { color: "gray"; width: 100; height: 100;radius: 50 }
   Rectangle { color: "black"; width: 100; height: 100;radius: 50 }
   add:Transition {
        NumberAnimation {
          properties: "y"
          easing.type: Easing.OutCubic
          duration: 2000
        }
     }
}
```

ضمن العنصر "Column" وضعنا قيمة الخاصيّة "spacing" هي "5" أي سوف يتم تباعد العناصر الأبناء عن بعضهم البعض 5 بكسلات , أضفنا ثلاث عناصر مستطيلات صيّرناها على شكل دائري, و أضفنا عنصر تحول ضمن الخاصيّة "add" مهمته هي تحريك أي عنصر يضاف بشكل "OutCubic" , ليصل إلى موقع الأساسي بمدة زمنية قدرها "2" ثانية .

ملاحظة : يضاف أول عنصر من الجهة العليا للنافذة والثاني أسفله ...

عند تنفيذ التطبيق سوف يظهر كما في الشكل (9.2) :

الشكل 9.2

- **خصائص العنصر "Row" عي نفسها خصائص العنصر "Column" باستثناء خاصيّة وحيدة غيرة موجودة داخل العنصر "Column" وهي :**

layoutDirection : enumeration

مهمتها تحديد إتجاه ترتيب العنصار من الجهة االيسرى إلى الجهة اليمنى "Qt.LeftToRight", أو من الجهة اليمنى إلى الجهة اليسرى "Qt.RightToLeft" ,*هذه الخاصيّة موجودة في الوحدات النمطية "QtQuick" التي يكون إصدارها "1.1" و اصداراتها الأحدث.*

مثال , نافذة تحوي ثلاث دوائر مرصوفين بشكل متتالي على المحور الأفقي , عند تنفيذ التطبيف سوف يتم تحريك الدوائر الثلاث من الجهة اليسرى من النافذة إلى موقعهن بمدة زمنية قدره 2 ثا , انظر الرمّاز :

```
import QtQuick 1.1


Row {

  spacing: 5

  Rectangle {id:r; color: "silver"; width: 100; height: 100;radius: 50 }
```

```
    Rectangle { color: "gray"; width: 100; height: 100;radius: 50 }
    Rectangle { color: "black"; width: 100; height: 100;radius: 50 }
    add:Transition {
        NumberAnimation {
            properties: "x"
            easing.type: Easing.OutCubic
            duration: 2000
        }
    }
}
```

مثلما تلاحظ نفس رمّاز المثال السابق لهذا المثال باستثناء :

استخدمنا هنا العنصر "Row" بدلا من العنصر "Column" من اجل ترتيب العناصر أفقيا

بما ان العناصر رتبت بشكل أفقي توجب علينا أن نغير الخاصيّة المتفاعل معها عنصر الحركة "NumberAnimation" من "y" (التحريك على المحور العمودي) إلى "x" (التحريك على المحور الأفقي) , عند تنفيذ التطبيق سوف ترى كالشكل (9.3) :

الشكل 9.3

- **أهم خصائص العنصر "Flow" :**

مقدار التباعد بين العناصر الأبناء مقدرا بالبكسل	**spacing : int**
تستخدم من أجل تطبيق تحول عند إضافة عنصر جديد	**add : Transition**
تستخدم من أجل تطبيق تحول عند تغيير موقع عنصر ابن	**move : Transition**

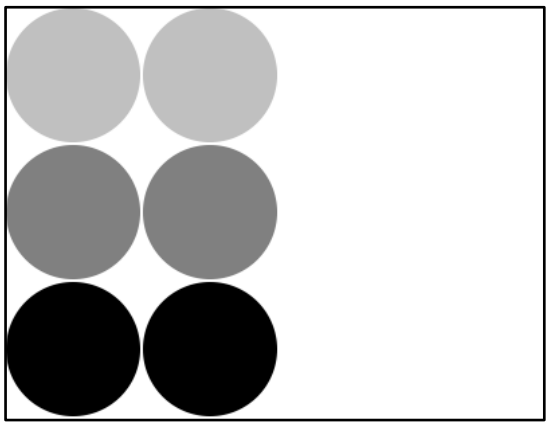
layoutDirection : enumeration	**مهمتها تحديد إتجاه ترتيب العنصار من الجهة االيسرى إلى الجهة اليمنى "Qt.LeftToRight", أو من الجهة اليمنى إلى الجهة اليسرى "Qt.RightToLeft"** ***,هذه الخاصيّة موجودة في الوحدات النمطية "QtQuick" التي يكون إصدارها "1.1" و ما فوق***
flow : enumeration	**إعطاء أولوية جهة الإنسياب للعناصر الأبناء , Flow.LeftToRight القيمة الإفتراضيّة , أولوية جهة الإنسياب للعناصر هي من اليمين إلى اليسار ,** **Flow.TopToBottom أولوية جهة الإنسياب للعناصر هي من الأعلى إلى الأسفل**

مثال , نافذة حجمها "406X306" , تحوي 6 عناصر دائرية الشكل , حجم كل عنصر منها هو "100X100", أولوية جهة إنسيابها هي من الأعلى إلى أسفل , انظر الرمّاز :

```
import QtQuick 1.0


Rectangle{
   width: 408;height: 306


Flow {
   anchors.fill: parent
   spacing: 2
   flow: Flow.TopToBottom


     Rectangle { color: "silver"; width: 100; height: 100;radius: 50 }
```

```
        Rectangle { color: "gray"; width: 100; height: 100;radius: 50 }
        Rectangle { color: "black"; width: 100; height: 100;radius: 50 }
        Rectangle { color: "silver"; width: 100; height: 100;radius: 50 }
        Rectangle { color: "gray"; width: 100; height: 100;radius: 50 }
        Rectangle { color: "black"; width: 100; height: 100;radius: 50 }
}
}
```

مثلما نلاحظ حجم كل عنصر دائري هو "100X100" و حجم النافذة هو "408X306" , و أولوية جهة الإنسياب من الأعلى إلى الأسفل "المحور العمودي", فالإرتفاع للشكل الدائري هو "100" و إرتفاع النافذة هو "306" أي يوجد مسافة على المحور الأفقي يستطيع استيعاب ثلاث أشكل دائرية مرتبة بشكل متتالي, بالتالي سوف يظهر عند تنفيذ التطبيق كالشكل (9.4) :



الشكل 9.4

جرب ان تزيد من إرتفاع النافذة مسافة قدرها "102" ليصبح إرتفاعها مساويا لـ "408" , فكيف سوف يصبح ترتيب الأشكال الدائريّة ؟ جرب و اختبر النتيجة بنفسك .

- **أهم خصائص العنصر "Grid" :**

مقدار التباعد بين العناصر الأبناء مقدرا	**spacing : int**

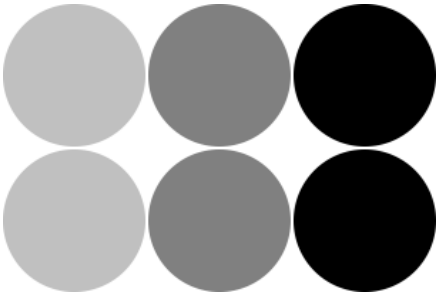
بالبكسل

عدد الأعمدة في الجدول	**columns : int**
عدد الصفوف في الجدول	**rows : int**
تستخدم من أجل تطبيق تحول عند إضافة عنصر جديد	**add : Transition**
تستخدم من أجل تطبيق تحول عند تغيير موقع عنصر ابن	**move : Transition**
مهمتها تحديد إتجاه ترتيب العنصار من الجهة االيسرى إلى الجهة اليمنى "Qt.LeftToRight", أو من الجهة اليمنى إلى الجهة اليسرى "Qt.RightToLeft" *,هذه الخاصيّة موجودة في الوحدات النمطية "QtQuick" التي يكون إصدارها "1.1" و ما فوق*	**layoutDirection : enumeration**
إعطاء أولوية جهة الإنسياب للعناصر الأبناء ضمن الأعمدة و الصفوفة المتاحة , Grid.LeftToRight القيمة الإفتراضيّة , أولوية جهة الإنسياب للعناصر هي من اليمين إلى اليسار , **Grid.TopToBottom أولوية جهة الإنسياب للعناصر هي من الأعلى إلى الأسفل**	**flow : enumeration**

مثال لإنشاء جدول يحوي 6 خلايا تقسيمها كالآتي ثلاث أعمدة و صفين , كل خليّة تحوي عنصر دائري الشكل , عند إضافة العناصر سوف يتم تحريكها بشكل "OutCubic" من النقطة "0X0" إلى موقعها الأساسي ضمن الخلايا حسب ترتيبها , زذلك يتم بمدة زمنية قدرها "2" ثانية , عند تنفيذ لاتطبيق سوف يظهر كالشكل (9.5) , رمّاز المثال الآني :

```
import QtQuick 1.0
```

```
Grid {
    columns: 3
    rows: 2
    spacing: 2

    add:Transition {
        NumberAnimation {
            properties: "x,y"
            easing.type: Easing.OutCubic
            duration: 2000
        }
    }

    Rectangle { color: "silver"; width: 100; height: 100;radius: 50 }
    Rectangle { color: "gray"; width: 100; height: 100;radius: 50 }
    Rectangle { color: "black"; width: 100; height: 100;radius: 50 }
    Rectangle { color: "silver"; width: 100; height: 100;radius: 50 }
    Rectangle { color: "gray"; width: 100; height: 100;radius: 50 }
    Rectangle { color: "black"; width: 100; height: 100;radius: 50 }
}
```



الشكل 9.5

❖ العنصر "Repeater" :

مهمة هذا العنصر هي إنشاء عدة مكونات من نفس المكون المحدد داخله, أي بدل تكرار نفس رمّاز المكون المراد إنشاء مجموعة منه , نستخدم العنصر "Repeater" .

يتفاعل العنصر "Repeater" مع عناصر العرض التوسعيّة كـ "Row" , بالتالي ينشأ عدة خلايا ضمن هذه العناصر تحوي نفس العنصر المحدد .

تتم عملية التكرار لعنصر ما موجود داخل العنصر "Repeater" من خلال إنشاء مندوب "delegate" لمثيل العنصر المراد تكراره , من ثمّ تتم عملية التكرار لهذا التفويض , و تعطى الخاصيّة "modelData" اسم التفويض الحالي و التي تدل على نسخة العنصر الذي ينشأ حاليا ضمن عملية التكرار, أما الخاصيّة "index" تعطى رقم الدليل الحالي للتفويض .

مثال لإنشاء نافذة تحوي عنصر "Row" بداخله عنصر المكرر "Repeater" ينشأ 4 نسخ من عنصر دائري الشكل , ويرتبها بشكل أفقي على التوالي ضمن خلايا العنصر "Row" , انظر الرمّاز :

```
import QtQuick 1.0


Rectangle {

   width: 400

   height: 100

   Row{
```

```
    add:Transition {
        NumberAnimation {
            properties: "x,y"
            easing.type: Easing.OutCubic
            duration: 2000
        }
    }
  Repeater{
  model: 4
    Rectangle { color: "silver"; width: 100; height: 100;radius: 50 }
  }
  }
}
```

استخدمنا الخاصيّة "model" من أجل تحديد عدد النسخ المراد إنشائها من العنصر ذي الشكل الدائري , عند تنفيذ التطبيق سوف ترى كالشكل (9.6) :

الشكل 9.6

نمط القيم التي تقبلها الخاصيّة "model" هي: رقم , نص , عنصر , أو أحد عناصر نموذج "model" كـ "ListModel" الذي سوف نتكلم عنه بالصفحات القادمة من هذا الفصل, و تقبل أيضا مصفوفة من هذه الأنماط آنفة الذكر .

في حال وضعنا مصفوفة من نمط نص للخاصيّة "model" , سوف ينشأ عنصر المكرر "Repeater" نسخ للعنصر المحدد عددها مساوي لعدد خلايا المصفوفة , و تصبح قيمة الوسيط "modelData" هي قيمة الخلية الحالية من المصفوفة و التي تمثّل نسخة العنصر الذي يتم أنشاءه "التمثيل هنا يدعى التفويض - delegate" .

ملاحظة : ***نسخ العنصر تنشأ على التوالي ضمن المكرر .***

مثال توضيحي , جدول يحوي 6 خلايا , ثلاثة أعمدة و صفان , يتم إنشاء الخلايا بوساطة عنصر المكرر "Repeater" , كل خلية تحوي على عنصر دائري الشكل , يتم إعطاء الألون للعناصر من خلال استخدام الوسيط "modelData" و الذي يحوي على اللون المراد , لإننا وضعنا قيمة الحاصيّة "model" هي مصفوفة نصيّة تحوي الألوان الثلاث "silver-gray-black" , انظر الرمّاز :

```
import QtQuick 1.0


Rectangle {
   width: 300
   height: 200
   Grid{
      columns : 3
      rows : 2
      add:Transition {
           NumberAnimation {
              properties: "x,y"
              easing.type: Easing.OutCubic
              duration: 2000
           }
        }
   Repeater{model : 2
       Repeater{
```

```
    model: ["silver","gray","black"]
    Rectangle { color: modelData ; width: 100; height: 100;radius: 50 }


     }
    }
   }
}
```

أنشئنا عنصر "Grid" يحوي ثلاث أعمدة و صفان , أضفنا نقطة إنتقال مهمتها تحريك العناصر الدائرية الشكل إلى مواقعها خلال مدة زمنية قدرها "2" ثانية , إنشئنا عنصر مكرر "Repeater" قيمة خاصيّته هي "2" بالتالي سوف يتم إنشاء نسختان من العنصر الذي يحويه والذي يمثّل الصف , العنصر الذي يحويه المكرر الحالي هو عنصر مكرر آخر , قيمة الخاصيّة "model" هي مصفوفة من ثلاث خلايا كل خليّة تحوي اسم لون, لاحظ كيف يتم تعريف المصفوفة باستخدام الأقواس المربعة و بين كل عنصر و آخر فاصلة ارضيّة , أخيرا أنشئنا عنصر ذي شكل دائري , لونه هو قيمة الخليّة الحالية من المصفوفة الموضوعة للخاصيّة "model" , الوسيط "modelData" يحوي قيمة الخلية الحالية من المصفوفة, قيمة الخليّة الحالية من المصفوفة تمثّل نسخة العنصر الذي يتم إنشاءها ضمن النموذج *"التفويض delegate"* , نفذ التطبيق و اختبر النتيجة .

معالجات الأحداث التي يملكها العنصر "Repeater" :

يستدعى عند لإضافة نسخة جديدة من العنصر المحدد , الوسيط "index" هو الدليل الحالي لنسخة العنصر المضافة , الوسيط "item" نسخة العنصر المضافة	**onItemAdded (int index, Item item)**
يستدعى عند إزالة نسخة عنصر من المكرر , الوسيط "index" هو الدليل الحالي لنسخة العنصر المزالة , الوسيط "item" نسخة العنصر المزالة	**onItemRemoved (int index, Item item)**

ملاحظة : معالجات الأحداث هذه موجودة فقط في الوحدات النمطية "QtQuick" ذي الإصدار "1.1" و الأحدث .

أخيرا نذكر الخاصيّة "count" والتي تعيد عدد النسخ المنشأة ضمن عنصر المكرر "Repeater" .

❖ عناصر نموذج القائمة :

لإنشاء قائمة تحوي بيانات و من ثمّ عرضها بتنسيق معين , يتوجب علينا استخدام ثلاث عناصر أساسيّة لهذا الغرض و هي :

1- ListElement : عنصر قائمة مهمته تضمين و حفظ بيانات تمثّل عنصر نموذج قائمة .
2- ListModel : نموذج قائمة مهمته احتواء مجموعة عناصر قائمة .
3- ListView : عرض قائمة مهمته عرض نموذج قائمة و تفويض عناصرها لقراءة قيم خصائصها.
4- GridView : عرض جدول مهمته عرض نموذج قائمة بشكل شبكي و تفويض عناصر نموذج القائمة لقراءة قيم خصائصها .

- الشكل العام لإنشاء عنصر قائمة "ListElement" هو :

```
ListElement {
property1 : value
property2 : value
.
.
}
```

نقصد بـ "property1 or 2 ..." اسم الخاصيّة التي نود حفظ قيمة داخلها , مثلما تلاحظ ضمن الشكل العام نستطيع تخزين أكثر من خاصيّة ضمن عنصر النموذج الواحد , مثال :

```
ListElement{
firstName : "Hasan"
lastName : "Al Morhej"
```

MobileNo : "+96399999999"

}

نلاحظ ضمن الرمّاز السابف تعريف ثلاث خصائص خاصة بتخزين البيانات الشخصيّة .

- **عنصر نموذج قائمة "ListModel" :**

مهمته احتواء "تضمين" مجموعة من عناصر القائمة , يحوي على خاصيّة وحيدة و هي "count" و التي ترجع عدد عناصر القائمة التي يحويها , اما بالنسبة لأهم المناهج الذي يحويها هي :

إضافة عنصر قائمة بموقع الدليل "index"	insert (int index,data)
إضافة عنصر قائمة بعد آخر عنصر	append (data)
نقل عدة عناصر التي يكون موقعها بين "from" و "from + n" إلى الموقع "to"	move (int from , int to , int n)
تغيير قيمة العنصر الذي يكون دليله هو "index" , في حال كان الدليل "index" مساوي لعدد العناصر الموجودة ضمن عنصر نموذج البيانات سوف يضاف عنصر جديد بعد آخر عنصر قائمة	set (int index, data)
إزالة العنصر ذي الدليل "index" من ثمّ إعادة ترتيب عناصر القائمة	remove (int index)
مسح جميع عناصر القائمة المضمّنة	clear ()
إرجاع قيمة العنصر ذي الدليل "index"	object get (int index)
تغيير قيمة الخاصيّة الذي يكون دليلها "index" و اسمها "property"	setProperty (int index, string property, variant val)

الشكل العام لقيمة الوسيط "data" الموضح ضمن الجدول السابق :

{"property1" : value, "property2" : value}

مثال توضيحيّ :

```
listModel.append( "firstName" : "Ahmed", "MobileNo" :
00963999222222 )
```

- **معالجات الأحداث الذي يملكها عنصر نموذج قائمة "ListModel" هي :**

onCountChanged	**يستدعى عندما يتغير العدد الكلي لعناصر القائمة الموجودة داخل عنصر نموذج القائمة**
onItemsInserted	**يستدعى عند إضافة عنصر قائمة جديد داخل عنصر نموذج القائمة**
onItemsChanged	**يستدعى عند تغيير قيمة عنصر قائمة موجود داخل عنصر نموذج القائمة**
onItemsMoved	**يستدعى عند تغيير موقع عنصر أو عدة عناصر قائمة موجودين داخل عنصر نموذج القائمة**
onItemsRemoved	**يستدعى عند إزالة عنصر قائمة من عنصر نموذج القائمة**

ملاحظة : معالجات الأحداث هذه موجودة فقط في الوحدات النمطية "QtQuick" ذي الإصدار "1.1" و الأحدث .

- **الشكل العام لنموذج قائمة "ListModel" :**

```
ListModel{

ListElement{ }

ListElement{ }

.
```

.

}

- **عنصر عرض قائمة "ListView"** :

مهمته عرض عنصر نموذج قائمة , يرث العنصر "Flickable" , لذا سوف تلاحظ عند مسك القائمة المرئيّة بزر الفأرة و سحبها من ثمّ إفلاتها أنها ستتحرك بطريقة شبيهة لحركة عنصر النابض لحظة إفلاته من فعل قوى الشد المطبق عليه ,*"سنلكم بالتفصيل ضمن الفصل القادم حول العنصر Flickable"* , أهم خصائص العنصر "ListView" هي :

delegate : Component	**مهمتها احتواء العنصر "المكون" المراد تمثيله "جعله مندوب "لعنصر القائمة من أجل عرضها داخل عنصر "ListView", أي تفويض المكون الموضوع ليمثل عنصر القائمة , في هذه الحالة نستطيع الوصول لقراءة جميع الخصائص المخزنة للبيانات الموجودة داخل عنصر القائمة , داخل كتلة العنصر الذي قد تمّ تفويضه**
header : Component	**وضع عنصر مرئي في رأس القائمة المرئيّة**
footer : Component	**وضع عنصر مرئي في أسفل القائمة المرئيّة**
highlight : Component	**وضع عنصر مرئي على كامل مساحة العنصر المحدد**
layoutDirection : enumeration	**تحديد جهة ترتيب العناصر فقط في حال كان التوجيه "orientation" أفقي , عناص التعداد :** ▪ **Qt.RightToLeft** ▪ **Qt.LeftToRight**
orientation : enumeration	**تحديد توجيه عناصر القائمة إما أفقيا أو عموديا عناصر التعداد :** ▪ **ListView. Horizontal** ▪ **ListView. Vertical**

تحديد مقدار التباعد بين عناصر القائمة , مقدر قيمته بالبكسل	spacing : real
وضع نموذج القائمة "ListModel" الذي سيتم عرضه	model : model
خاصيّة للقراءة فقط , ترجع العدد الكلي لعناصر القائمة الموجودين داخل نموذج القائمة	count : int

- مثال يوضّح كيفيّة استخدام عناصر نموذج قائمة :

لنبني تطبيق "دفتر هواتف" و الذي هو عبارة عن نافذة تحوي قائمة متعددة العناصر , محتوى عناصرها هو الأسم الأول و الأسم الثاني و رقم الجوال , انظر الرمّاز :

```
import QtQuick 1.0


Rectangle {
   width: 200
   height: 400
   color: "white"


   ListModel{
   id:listModel
   ListElement{
      fName : "Hasan"
      lName : "Al Morhej"
      mobileNo : "+9639999999"
   }
```

```
ListElement{
   fName : "Mouhammad"
   lName : "Ali"
   mobileNo : "+96399888888"
}
ListElement{
   fName : "Bouchra"
   lName : "Mansour"
   mobileNo : "+9639944444"
}
ListElement{
   fName : "Ihab"
   lName : "Mahmoud"
   mobileNo : "+9639221122"
}
}

Component{
   id:header
Rectangle{
id:rect
width: 200; height: 20;
radius: 20
```

```
color: "black"
smooth: true
Text {
   id: txtHeader
   color : "white"
   font.bold: true
   anchors.centerIn: parent
   text: qsTr("Phonebook")
}
}
}

ListView{
width: 200;height: 300
model: listModel

delegate: Rectangle{
   width: 200;height: 40
   color: "silver"
   Text {
      color: "gray"
      text: qsTr(fName + " " + lName + "\nMobile No. " + mobileNo)
   }
```

```
    }

      header: header

    }


}
```

إنشئنا عنصر مستطيل لونه أبيض ليمثّل النافذة , يحوي عنصر نموذج قائمة "ListModel" اسمه "listModel" بدوره يحوي العديد من عناصر قائمة "ListElement", بداخل كل عنصر قائمة ثلاث خصائص و هي "الأسم الأول – First Name" و "الأسم الثاني – Last Name" و "رقم الجوّال – Mobile No" , ثمّ انشئنا مكون يدعى "header" و الذي سوف نضعه كـ "header" لـ عنصر "ListView" , هذا المكون عبارة عن عنصر مستطيل لونه أسود يوجد في أوسطه نص وهو "Phonebook" , أضفنا عنصر عرض قائمة "ListView" و ضعنا قيمة خاصيّته "model" العنصر "listModel" , بالتالي سوف يتم عرض جميع عناصر نموذج القائمة المضمّنة داخله , اما بالنسبة لخاصيّة التفويض "delegate" أنشئنا داخلها عنصر مستطيل فضي اللون , يحوي عنصر نص "Text" ليتم كتابة قيم جميع خصائص عناصر القائمة داخل عنصر النص , حصلنا على قيم خصائص عناصر القائمة من خلال كتابة اسم الخاصيّة الموجودة داخل عنصر القائمة ,كالخاصيّة "mobileNo" , وضعنا المكون "header" و الذي يمثل عنوان دفتر الهواتف قيمة الخاصيّة "header" للعنصر "ListView" .

عند التنفيذ جرب النقر بزر الفأرة الأيسر على عنصر القائمة ومن ثمّ سحبه إلى الأسفل أو إلى الجهى العليا و و لاحظ كيف تكون حركة عنصر القائمة .

التطبيق سوف يبدو كالشكل (9.7) :

الشكل 9.7

- **عنصر عرض جدول "GridView" :**

مهمته عرض عنصر نموذج قائمة بشكل شبكي , يرث العنصر "Flickable" , أهم خصائصه و طريقة عمله و تفاعله مع العناصر هي نفس العنصر "ListView" , باستثناء أنه يمل اثنان من الخصائص غير موجودة داخل العنصر "ListView" و التي هي :

تحديد عرض الخليّة **cellWidth : int**

تحديد ارتفاع الخليّة **cellHeight : int**

لنوضح كيفيّة استخدام العنصر "GridView" , من خلال عرض نموذج القائمة الخاصة بـجهات الإتصال الموجودة ضمن المثال السابق داخل جدول "GridView", افتح المشروع السابق و حذف الرمّاز الخاص بالعنصر "ListView" و هو هذا :

```
ListView{

   width: 200;height: 300
```

```
model: listModel

delegate: Rectangle{
   width: 200;height: 40
   color: "silver"
   Text {
      color: "gray"
      text: qsTr(fName + " " + lName + "\nMobile No. " + mobileNo)
   }

}
   header: header
}
```

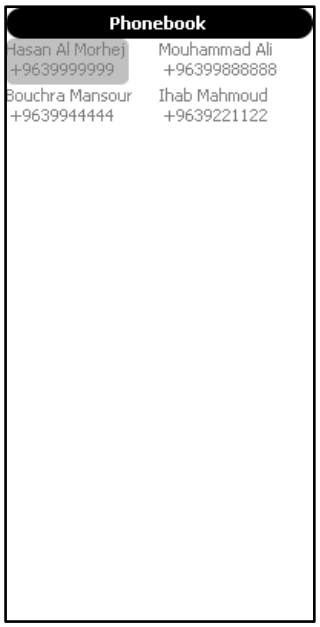
و من ثمّ اكتب بدل منه الرمّاز التالي :

```
GridView{
width: 200;height: 600
model: listModel
cellHeight: 30

delegate: Column{
   width: 80;height: 30
   Text {
```

```
        color: "gray"
        text: qsTr(fName + " " + lName + "\n " + mobileNo)
    }

}
    header: header
    highlight: Rectangle { color: "silver"; radius: 5 }
    focus: true
}
```

أنشئنا عنصر "GridView" لعرض جهات الإتصال على شكل جدول , وضعنا ارتفاع الخليّة ضمن الجدول هي "30" بكسل , من ثمّ أشئننا عنصر عمود "Column" داخل خاصيّة التفويض "delegate" ليتم انشاء عناصر عمود حسب عرض العنصر "GridView" و عرض عنصر العمود "Column" و عرض الخليّة "cell", أي في حال كان عرض العنصر "GridView" مساوي لـ "400" و عرض عنصر العمود "Column" مساوي لـ "400" و يوجد أربع خلايا عرض كل منها "100" , عنصر "GridView"سوف لن ينشأ أي صف , لأن الأربع خلايا عرضهم الكامل مساوي لـ "400" , لذا سوف يعرضهم بجانب بعضهم البعض , أي أربع أعمدة و صف واحد, أما في حال كان عدد الخلايا مساوي لـ "8" سوف يعرض أربع أعمدة وصفين و هكذا...

عند تنفيذ التطبيق و تحريك الأسهم سوف تلاحظ إنتقال علامة البقعة الأهم "highlight" لتمييز العنصر المحدد , انظر الشكل (9.8) :



الشكل 9.8

❖ عنصر نموذج العناصر المرئيّة "VisualItemModel" :

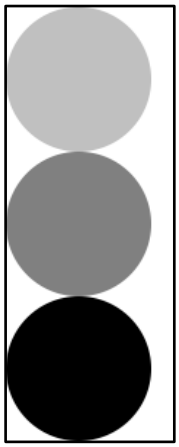
يستخدم من أجل تعريف مجموعة من العناصر المرئيّة كعنصر المستطيل ضمن نموذج ليتم عرضه داخل عنصر "ListView" أو "GridView" , لا يتطلب إنشاء أي عنصر للتفويض "delegate" لعرضه ضمن أي من عناصر عارض النموذج , يرث العنصر "Flickable" .

لإنشاء قائمة تحوي ثلاث أشكال دائرية , وعرضها ضمن عنصر "ListView" ,انظر الرمّاز التالي :

```
import QtQuick 1.0


Rectangle {
   width: 100;height: 300
   VisualItemModel {
      id: visualItemModel
      Rectangle { height: 100; width: 100; color: "silver";radius: 50 }
```

```
        Rectangle { height: 100; width: 100; color: "gray";radius: 50 }
        Rectangle { height: 100; width: 100; color: "black";radius: 50 }
    }

    ListView {
        anchors.fill: parent
        model: visualItemModel
    }
}
```

أنشئنا عنصر "VisualItemModel" يدعى "visualItemModel" و أضفنا ثلاث أشكال دائرية داخله , ثمّ أشئنا عنصر "ListView" وو ضعنا نموذجه هو "visualItemModel" ليتم عرض الأشكال الدائريّة الثلاث داخله , عند تنفيذ التطبيق سوف ترى كالشكل (9.9) :



الشكل 9.9

❖ عنصر نموذج بيانات العناصر المرئيّة "VisualDataModel" :

من أجل تغليف النموذج "model" الخاص باحتواء البيانات التي سوف تعرض , و المندوب "delegate" الخاص بتمثيل الكائن الذي سوف يعرض البيانات يتوجب علينا استخدام العنصر "VisualDataModel" , أهم خصائص هذا العنصر هي :

لوضع مكون مندوب عن البيانات التي ستعرض	**delegate : Component**
لإحتواء نموذج البينات التي ستعرض	**model : model**
من أجل الوصول لعنصر ما يحوي نموذج بيانات موجود داخل رزمة "Package" , ليتم وضعه كنموذج للعنصر لعنصر العرض	**parts : object**

الرزمة "Package" تستخدم من اجل احتواء مجموعة من العناصر "Items" ضمن نفس الملف أو الكتلة.

مثال لعرض نموذج يحوي خاصيّتين "الأسم الأول – الأسم الثاني" من ثمّ عرض بياناته من خلال إنشاء مندوب عن النموذج ليعرضها بوساطة العنصر "ListView" , بالنسبة للنموذج و المندوب موجودان داخل العنصر "VisualDataModel" الذي يهتم بعملية تغليفهم , انظر الرمّاز :

```
import QtQuick 1.0


Rectangle{

width: 100;height: 100

color: "silver"

VisualDataModel{

   id: visualDataModel

   model: ListModel{

   ListElement{
```

```
    fName : "Hasan"
    lName : "Al Morhej"
  }
  ListElement{
    fName : "Maxim"
    lName : "Naser"
  }
  }
  delegate:
      Rectangle{height: 20
        Text {
          id: txt
          text: qsTr(fName +" : " + lName)
        }
      }

  }

ListView{
width: 100;height: 100
model: visualDataModel
}
}
```

أنشئنا عنصر "VisualDataModel" يدعى "visualDataModel" يحوي على نموذج قائمة بداخلها اثنان من عناصر القائمة , و يحوي أيضا تعريف مندوب و الذي يمثل عنصر المستطيل الخاص بعرض قيم خصائص عناصر القائمة الموجودين داخل النموذج , ثمّ أنشئنا عنصر "ListView" و الذي أعطينا قيمة خاصيّتة "model" العنصر "visualDataModel" ليتم عرض بيانات النموذج , مثلما تلاحظ أنه لم يتوجب علينا إنشاء مندوب داخل عنصر عرض القائمة لإننا قد انشئنا مندوب داخل العنصر "visualDataModel" و الذي بالأساس تكون مهمته *"تغليف النموذج (model) و المندوب (delegate)"* , نفذ التطبيق و اختبر النتيجة .

- **استخدام عنصر الرزمة "Package" مع العنصر "VisualDataModel" :**

في حال أردنا أن نفوض عن عدة مندوبين داخل الخاصيّة "delegate" التابعة للعنصر "VisualDataModel" من أجل التبديل بينها فما يتوجب علينا فعله ؟

نستخدم عنصر الرزمة "Package" من اجل عدة عناصر داخلها لتفويض أحداها كمندوب لبيانات النموذج التي ستعرض , سنستخدم نفس الرمّاز السابق لتوضيح عملية إضافة عدة مندوبين من خلال استخدام عنصر الرزمة "Package" , انظر الرمّاز :

```
import QtQuick 1.0


Rectangle{
width: 100;height: 100
color: "silver"
VisualDataModel{
  id: visualDataModel
  model: ListModel{
  ListElement{
    fName : "Hasan"
    lName : "Al Morhej"
  }
```

```
ListElement{
   fName : "Maxim"
   lName : "Naser"
}
}
delegate: Package{
   Item {
      id: rect1
      height: 20
      Package.name: 'bg1'
      Rectangle{
         Text {
            id: txt1
            text: qsTr(fName +" : " + lName)
         }
      }
   }
   Item {
      id: rect2
      height: 20
      Package.name: 'bg2'
      Rectangle{
         Text {
```

```
            id: txt2
            color: "white"
            text: qsTr(fName +" : " + lName)
        }
    }
  }
 }
}
ListView{
width: 100;height: 100
model: visualDataModel.parts.bg2
}
}
```

مثلما تلاحظ *الإختلاف الأول* عن الرمّاز السابق لهذا الرمّاز هو ضمن كتلة الخاصيّة "delegate" التابعة للعنصر "VisualDataModel", انشئنا داخلها عنصر رزمة "Package" يحوي على عنصرين , العنصر الأول من جهة الحزمة يدعى "bg1" و اعنصر الثاني يدعى "bg2" , الإختلاف بين العنصر الأول و العنصر الثاني هو لون الخط للعنصر الأول "أسود" أما لون خط العنصر الثاني "أبيض" .

الإختلاف الثاني الخاصيّة "model" الموجودة ضمن كتلة العنصر "ListView" , حيث قيمتها أصبحت :

```
model: visualDataModel.parts.bg2
```

أي مندوب نموذج البيانات هو العنصر الذي يدعى "bg2" ذي لون الخط الأبيض , جرب أن تضع بدل من "bg2" العنصر "bg1" ستلاحظ أنه أصبح لون الخط أسود , استخدمنا الخاصيّة "parts" من أجل الوصول للعنصر المراد تفويضه كمندوب لنموذج البيانات , نفذ التطبيق و اختبر النتيجة .

- شكل عنصر الرزمة "Package" العام هو :

```
Package{
Item{ Package.name : "item1" }
Item{ Package.name : "item2" }
. . .
}
```

❖ مكوّنات إضافيّة في QML :

يوجد العديد من عناصر واجهة المستخدم الإضافيّة ضمن "QML" , تمّ بنائها من أجل سهولة تصميم واجهة مستخدم غنيّة و سهلة , أهم الوحدات النمطيّة الحاوية لهذه العناصر :

حاوية لعناصر واجهة المستخدم التي تعمل على منصة Symbian	com.nokia.symbian 1.0
حاوية لعناصر واجهة المستخدم التي تعمل على منصة Meego/Harmattan	com.nokia.meego 1.0
حاوية لعناصر واجهة المستخدم التي تعمل على جميع المنصات	Qt.labs.components.native 1.0

سنتكلم ضمن هذه الفقرة عن الوحدة النمطيّة "Qt.labs.components.native 1.0" .

- أهم عناصر الوحدة النطيّة "Qt.labs.components.native 1.0" :
  - عنصر النافذة "Window" : عنصر حاضن لعناصر واجهة المستخدم كـ عنصر الزر "Button" .
  - عنصر الزر "Button" :

```
import Qt.labs.components.native 1.0

Window{
   width: 300;height: 400
```

```
Button{
   text : "Exit"
   onClicked: Qt.quit()
}
}
```

- عنصر عمود أزرار **ButtonColumn** خاص باحتواء مجموعة من الأزرار و ترتيبها بشكل أفقي :

```
ButtonRow{

   Button{
      text : "Btn1"
      iconSource: "ico"
   }
   Button{
      text : "Btn2"
      iconSource: "ico"
   }
   Button{
      text : "Btn3"
      iconSource: "ico"
   }
}
```

- عنصر صف أزرار **ButtonRow** خاص باحتواء مجموعة من الأزرار و ترتيبها بشكل عمودي :

```
ButtonRow{ Button{} Button{} ... }
```

- عنصر زر تحديد ChekBox :

```
CheckBox{
   text: "Check"
   checked: true
   onCheckedChanged: doWhatever
}
```

- عنصر زر إختيار RadioButton :

```
   ButtonRow{
      RadioButton{
         text: "radio1"
         checked: true
      onCheckedChanged: doWhatever
       }
    RadioButton{
         text: "radio2"
         checked: false
      onCheckedChanged: doWhatever
     }
   }
```

- عنصر شريط تقدم ProgressBar :

```
ProgressBar{
maximumValue: 100
```

```
minimumValue: 0

value: 50

onValueChanged: doWhatever

}
```

- عنصر شريط إنزلاق Slider :

```
Slider{

maximumValue: 100

minimumValue: 0

value: 50

onValueChanged: doWhatever

}
```

- عنصر رمز مشغول BusyIndicator يظهر هذا العنصر على شكل دائرة ذي حد متحرك :

```
BusyIndicator{

width: 100;height: 100

running: true

}
```

- عنصر زر تبديل Switch :

```
Switch{

id:switchButton

}

Text {

   text: switchButton.checked ? "On" : "Off"
```

```
}
```

- عنصر مربع نص TextArea :

```
TextArea{
   placeholderText: "Write Here.."
}
```

- عنصر شريط تبويب TabBar
- عنصر زر تبويب TabButton
- عنصر حاوية لمجموعة صفحات TabGroup
- عنصر صفحة لتبويب Page

```
import Qt.labs.components.native 1.0

Window{
   width: 300;height: 400
TabBar{
   TabButton{tab: page1 ;text: "page1"}
   TabButton{tab: page2 ;text: "page2"}
   TabButton{tab: page3 ;text: "page3"}
}
TabGroup{
   anchors.centerIn: parent
Page{
id: page1
Text {text: qsTr("Page1"); color: "blue" }
}
```

```
Page{
id: page2
Text {text: qsTr("Page2"); color: "blue" }
}
Page{
id: page3
Text {text: qsTr("Page3"); color: "blue" }
}
}
}
```

- عنصر شريط الأدوات ToolBar
- عنصر زر خاص بشريط الأدوات ToolButton

```
Window {
   height: 350
   width: 35
   ToolBar {
      anchors.bottom: parent.bottom
      tools: ToolBarLayout {

         ToolButton {
            iconSource: "toolbar-back"
            onClicked: doWhatever
         }
```

```
        ToolButton {
            iconSource: "toolbar-menu"
        }
        ToolButton {
            iconSource: "toolbar-search"
        }
    }
  }
}
```

اسندنا لقيمة الخاصيّة "iconSource" الثابت "toolbar-...." الذي سيأخذ رمز معرف ضمن العنصر "ToolButton" , الرموز المعرّفة ضمن العنصر "ToolButton" :

	toolbar-settings		toolbar-share
	toolbar-add		toolbar-back
	toolbar-delete		toolbar-search
	toolbar-mediacontrol-pause		toolbar-dialer
	toolbar-mediacontrol-stop		toolbar-mediacontrol-play
	toolbar-mediacontrol-backwards		toolbar-mediacontrol-forward
	toolbar-next		toolbar-previous
	toolbar-home		toolbar-list
	toolbar-refresh		toolbar-menu

إلى هنا نكون أنتهينا من فصل عناصر واجهة المستخدم في "QML" .

❖ خلاصة الفصل :

بإنتهاء هذا الفصل تستطيع إنشاء واجهة مستخدم باحترافية و سهولة , لننتقل إلى الفصل الأخير من قسم QML والذي يدعى عناصر الحركة .

الفصل الحادي عشر

عناصر الحركة

Animation Elements

❖ مقدمة Intro :

متعة QML في عناصرها الحركية , حيث يمكن إضافة تأثيرات حركية ساحرة و بسهولى إلى عناصر QML الرسومية , فلذا خصصنا فصل كامل يتكلم حول هذه العناصر الحركية في QML و عن كيفية تفاعلها مع االمستخدم و العناصر الرسومية .

❖ العنصر "Animation" :

هو العنصر الأب "الأساس" لجميع عناصر الحركة في "QML" , شكله العام :

Animation on property { }

الجملة البرمجية "on property" تعني :

الخاصيّة التي سوف يتم تغيير قيمتها بشكل تدريجي خلال مدة زمنية محددة .

- **أهم خصائصه :**
  - **loops : مهمتها تحديد عدد مرات تكرار الحركة المحددة , إذا كانت قيمتها "Animation.Infinte" يعني انه عدد لانهائي من التكرار**
  - **paused : تستخدم من أجل الإيقاف المؤقت للحركة**
  - **running : تستخدم من اجل البدء بعملية الحركة**

- **أهم مناهجه :**
  - **start : عند استدعائه سوف يبدأ تنفيذ عملية الحركة للعنصر**

- **stop : عند استدعائه سوف يتم إيقاف تنفيذ عملية الحركة للعنصر**
- **restart : عند استدعائه سوف يعيد تنفيذ عملية الحركة للعنصر**
- **pause : عند استدعائه سوف يتم إيقاف تنفيذ عملية الحركة للعنصر بشكل مؤقت**
- **resume : عند استدعائه سوف يخرج من حالة الإيقاف المؤقت للعنصر**

❖ العنصر "PropertyAnimation" :

مهمته تغيير قيم الخصائص المرئيّة كـ "عرض عنصر" بمدة زمنيّة محددة , بالتالي سوف تتم عملية الحركة للعنصر المحدد , يحوي خاصيّة تدعى "easing" تستخدم من أجل تحديد منحني سير الحركة المراد "تم ذكره بالتفصيل ضمن الفصل الثالث الموجود في قسم Qt" , العنصر "PropertyAnimation" يرث العنصر "Animation" و الذي يكون الصف الأب "الأساس" لجميع عناصر الحركة في "QML" .

- **مثال لإنشاء نافذة تحوي دائرة تتحرك من الجهة اليسرى إلى الجهة اليمنى بمدة زمنيّة قدرها "5" ثوان :**

```
import QtQuick 1.0


Rectangle{
   id:win
   height: 500;width: 500
   color: "red"
   Rectangle{id:rect
      width:200;
      height:200;
      radius: 100
      color: "green"
   PropertyAnimation{target:rect;running: true;
```

```
   property: "x";from:0;to:300;duration: 5000;easing.type:
Easing.OutSine}

   }


}
```

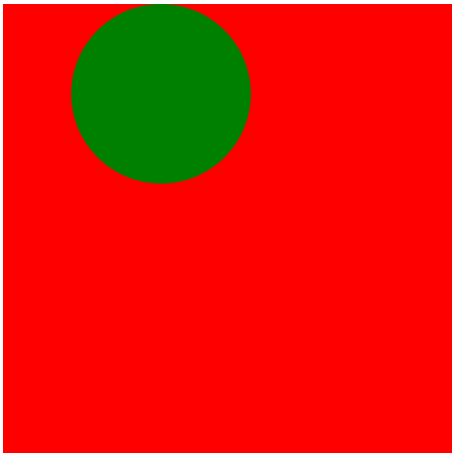
الخاصيّة "target" لتحديد العنصر المراد تحريكه , يوجد خاصيّة مثيلها تدعى "targets" لتحديد مجموعة من العناصر, يتم ذلك بوضع أسماء جميع العناصر بين أقواس مربعة و بين كل اسم عنصر و آخر فاصلة أرضيّة :

```
   PropertyAnimation{targets:[rect,win];running: true;

   property: "x";from:0;to:300;duration: 5000;easing.type:
Easing.OutSine}
```

أما الخاصيّة "running" لبدأ عملية الحركة في حال كانت قيمتها "true" , الخاصيّة "property" لتحديد الخاصيّة المراد تغيير قيمتها , يوجد خاصيّة مثيلها تدعى "properties" لتحديد مجموعة من الخصائص , بشرط أن يكون بين كل خاصيّة و أخرى فاصلة أرضيّة :

```
PropertyAnimation{target:rect;running: true;
properties:"width,height";

from:0;to:300;duration: 5000;easing.type: Easing.OutSine}
```

الخاصيّة "from" تعني "من القيمة" , أما "to" تعني "إلى القيمة" و بالتالي سوف تتغير قيمة الخاصيّة المحددة (من – إلى) بشكل تدريجي خلال المدة الموضوعة , لتحديد المدة المراد تغيير قيم الخصائص ضمنها بشكل تدريجي نستخدم الخاصيّة "duration" , بالنسبة لتحديد نوع منحني سير الحركة "easing" نأخذ منه الخاصيّة "type" من ثمّ نضع المنحني المراد "جميع انواع منحنيات سير الحركة موجود في الفصل الثالث ضمن الجدول الخاص بالتعداد QEasingCurve::Type" , نفذ التطبيق و اختبر النتيجة , انظر الشكل (10.1) :



الشكل 10.1

❖ العنصر "NumberAnimation" :

مهمته تغيير قيم الخصائص التي يكون نمطها "real" عدد حقيقي كـ "X" , خلال مدة زمنية محددة , يرث العنصر "PropertyAnumation" .

مثال لإنشاء مربع ضمن نافذة يتحرك هذا المربع 300 بكسل على المحور العمودي خلال مدة قدرها ثانية :

```
import QtQuick 1.0


Rectangle{

id:win

width: 500; height: 500

color: "blue"

Rectangle{

id:rect

width: 100; height: 100

x:100;y:100

radius: 10
```

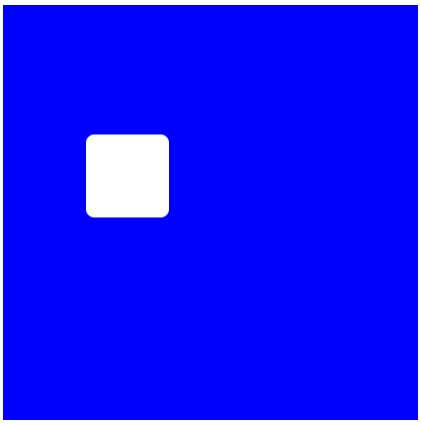
```
NumberAnimation on y{to:300;duration: 1000}

}

}
```

التعبير "on y" يعني انه سوف نقوم بتغيير قيمة الخاصيّة "y" , لتغيير مقدار الدوران نستبدل "y" بـ "rotation" :

```
NumberAnimation on rotation {to:360;duration:1000}
```

بما أننا لم نضع الخاصيّة "from" فهذا يعني قيمة "from" تساوي القيمة الحاليّة للخاصيّة المحددة .

نفذ التطبيق ,سوف يظهر كما في الشكل (10.2) :



الشكل 10.2

ملاحظة : لا نحتاج أن نضع الخاصيّة "running" لتنفيذ عملية الحركة .

❖ العنصر "RotationAnimation" :

مهمته تغيير قيمة الخاصيّة "rotation" الخاصة بتحديد مقدار الدوران مقدر بالدرجة, حيث يتم تغير قيمة الدوران بشكل تدريجي خلال مدة زمنية محددة , يرث العنصر "PropertyAnimation" .

مثال نافذة تحوي مربع يقوم بالدروان حول نفسه , يكون اتجاه الدوران بعكس حركة عقارب الساعة :

```
import QtQuick 1.0
```

```
Rectangle{
id:win
height: 500;width: 500
color:"#00ffff"
Rectangle{id:rect
   height: 100;width: 100
   anchors.centerIn: parent
   color: "red"
   radius: 10
   smooth: true

   RotationAnimation on rotation{from:360;to:0;duration: 5000;
   direction: RotationAnimation.Counterclockwise;easing.type
:Easing.InQuad}


  }
}
```

الخاصيّة "direction" لتحديد اتجاه الدوران , انظر الجدول :

RotationAnimation.Numerical	القيمة الإفتراضية ,الدوران باتجاه عقارب الساعة, الدوران تدريجيا بقيمة تساوي الفرق بين "from" و "to"
RotationAnimation.Clockwise	الدوران باتجاه قارب الساعة

RotationAnimation.Counterclockwise	الدوران باتجاه عكس قارب الساعة
RotationAnimation.Shortest	الدوران باتجاه القيمة الأقصر , في حال كان "from" مساوية لـ "100" و "to" مساوي لـ "200" , فسوف يتم الدوران باتجاه عقارب الساعة لأن الفرق بين القيمتين "100" (اذا كان الحساب باتجاه عقارب الساعة), و الفرق بين القيميتين باتجاه عكس عقارب الساعة هي "260"

❖ العنصر "ColorAnimation" :

يستخدم من أجل تغيير قيمة اللون بشكل تدريجي خلال فترة زمنية محددة , يرث العنصر "PropertyAnimation" .

مثال لتغيير لون مستطيل إلى اللون الأزرق بشكل تدريجي خلال مدة زمنية و قدرها 2 ثانية :

```
import QtQuick 1.0


Rectangle{
id:win
height: 500;width: 500
color:"blue"
Rectangle{id:rect
   height: 100;width: 100
   anchors.centerIn: parent
   color: "#ff00ff"
   radius: 10
   smooth: true
```

```
    ColorAnimation on color{to:"blue";duration: 2000}
}
}
```

نفذ التطبيق و اختبر النتيجة .

❖ تفاعل عناصر الحركة مع التحولات (نقاط الإنتقال) :

مثلما ذكرنا مسبقا أن التحول هو المرحلة المحصورة بين القيمة القديمة لخاصيّة و قيمتها الجديدة , إذا التحول خلص بتغيير اسلوب الإنتقال بين قيم الخصائص , اسلوب الإنتقال يعني استخدام احد عناصر الحركة "Animation Elements" , فلنلقي نظرة على هذا الرمّاز :

```
import QtQuick 1.0


Rectangle{
id:win
width: 500;height: 500
color: "silver"
states: State {
    name: "changeColor"
    when:mouseArea.pressed & Qt.LeftButton
    PropertyChanges {
        target: win
        color:"black"
    }
}
transitions: Transition {
```

```
    ColorAnimation { duration: 1000 }
}

MouseArea{id:mouseArea
anchors.fill: parent
}
}
```

استخدمنا التحول من أجل عدم تغيير لون خلفية النافذة من اللون الفضي إلى اللون الأسود بشكل مباشر , بل تتم عملية تغيير اللون بشكل تدريجي من الفضي إلى الأسود خلال مدة زمنية قدرها ثانية , هذا ما يدعى بتفاعل عنصر حركي مع نقطة إنتقال .

مثال آخر , تطبيق يحوي في وسط نافذته عنصر دائري الشكل حجمه "10X10" , عند النقر بزر الفأرة الأيسر عليها و الإستمرار في الضغط سوف بزداد حجمها حتى تملئ كامل النافذة , هنا سوف نوضح كيفيّة تفاعل عدة خصائص ضمن عنصر حركي موجود داخل كتلة عنصر تحول , انظر الرمّاز :

```
import QtQuick 1.0

Rectangle{
id:win
width: 500;height: 500
color: "silver"
Rectangle{id:circle
    anchors.centerIn: parent
    width: 10;height: 10
    radius: 5
    color: "black"
```

```
states: State {
   name: "changeSize"
   when:mouseArea.pressed & Qt.LeftButton
   PropertyChanges {
      target: circle
      width:500
      height:500
      radius:250
   }
}
transitions: Transition {
   PropertyAnimation {properties: "width,height,radius"; duration:
1000 }
}

MouseArea{id:mouseArea
anchors.fill: parent
}
}
}
```

عملية التحول قد اجريت على الخصائص الثلاث التالية :

1- width 2- height 3- radius

لننتقل للتكلم عن عنصر السلوك "Behavior" .

❖ العنصر "Behavior" :

عنصر السلوك يستخدم من أجل تغيير السلوك الإفتراضي الذي ينحا له عنصر ما عند تغيير قيمة خاصيّة له, كخاصيّة الموقع الأفقي للعنصر "x" , لنرى مثال توضيحيّ :

```
import QtQuick 1.0


Rectangle{id:win
   width: 500;height: 100


Rectangle {
   id: rect
   width: 100; height: 100
   color: "silver"


   Behavior on x {
      NumberAnimation { duration: 1000 }
   }


   MouseArea {
      anchors.fill: parent
```

```
        onClicked: rect.x = 400

    }

}


}
```

عند النقر بزر الفأرة الأيسر على المربع الفضي سوف يتغيير موقعه على المحور الأفقي خلال مدة زمنية قدرها ثانية , وذلك بسبب تعيين سلوك للمحور اللافقي "x" , يتضمن عنصر الحركة "NumberAnimation" و الذي وضع مدة تغيير الخاصيّة "x" هي "1000" ميلي / ثا , التعبير العام لعنصر السلوك :

```
Behavior on property { Animation {  } }
```

السلوك هو المنحى الذي يأخذه عنصر عند تغيير قيمة خاصيّة مرتبطة بعنصر السلوك "Behavior" .

❖ العنصر "SpringAnimation" :

مهمته تحريك عنصر مرئي بشكل شبيه بحركة النابض , أهم خصائص يملكها العنصر "SpringAnimation" هي :

الخاصيّة "spring" تأخذ القيم الموجود ضمن المجال 0.0 و 5.0 , مهمتها ضبط قوة الدفع باتجاه الموقع المراد للعنصر الرئي المحدد .

الخاصيّة "damping" تأخذ القيم الموجود ضمن المجال 0.0 و 1.0 , مهمتها ضبط مقدار ذبذبات لحركة النابض .

مثال نافذة تحوي عنصر مستطيل عند النقر بزر الفأرة الأيسر على أي موقع من النافذة ينتقل موقع عنصر المستطيل إليه بحركة شبيهة بحركة النابض :

```
import QtQuick 1.0


Rectangle{
```

```
id:win
width: 500;height: 500
color:"black"
Rectangle{
id:rect
width: 100;height: 100
states: State {
   name: "s1"
   when: mouseArea.pressed & Qt.LeftButton
   PropertyChanges {target:
rect;x:mouseArea.mouseX;y:mouseArea.mouseY}
}

transitions: Transition {
   SpringAnimation {target:rect;properties: "x,y";spring: 1;damping:
0.1}
}
}

MouseArea {id:mouseArea
   anchors.fill: parent
}
}
```

ضمن الرمّاز استخدمنا عنصر الحالة و عنصر التحول , عند تفعيل عنصر الحالة باستخدام النقر على زر الفأرة ضمن النافذة سوف ينتقل موقع عنصر المستطيل إلى الموقع الذي قد تم نقر زر الفأرة فيه , بالتالي سيقوم عنصر التحول بتحريك عنصر المستطيل بحركة شبيهة بحركة نابض , عند إفلات الضغط عن زر الفأرة الأيسر , سوف ينتقل عنصر المستطيل إلى موقعه الأساسي .

مثال آخر لنافذة تحوي عنصر مستطيل يتحرك بتحريك مؤشر الفأرة ليتبع موقعه ضمن النافذة , حركة عنصر المستطيل شبيه بحركة النابض , انظر الرمّاز :

```
import QtQuick 1.0


Rectangle{

id:win

width: 500;height: 500

color:"black"

Rectangle{

id:rect

width: 100;height: 100

Behavior on x {SpringAnimation {spring: 4;damping: 0.2} }

Behavior on y {SpringAnimation {spring: 4;damping: 0.2} }

}


MouseArea {

   anchors.fill: parent

   hoverEnabled: true

onPositionChanged: {

   rect.x = mouseX;
```

```
    rect.y = mouse.y;
}
}
}
```

استخدمنا عنصر السلوك "Behavior" من أجل إعطاء شكل الحركة لعنصر المستطيل عند تغيير موقعه ,و الذي يتغير موقعه عند تحريك مؤشر الفأرة ضمن النافذة الرئيسيّة , ضمن كتلة معالج الحدث "onPositionChanged" سلسلة التعليمات الخاصة بتغيير موقع عنصر المستطيل إلى موقع مؤشر الفأرة الحالي , مثلما تلاحظ استخدمنا الخاصيّة "mouseX" من اجل جلب قيمة موقع مؤشر الفأر على المحور الأفقي , و استخدمنا الوسيط "mouse" من أجل جلب موقع مؤشر الفأرة على المحور العمودي من خلال استدعاء الخاصيّة "y" منه .

❖ العنصر "SequentialAnimation" :

مهمته احتواء مجموعة من عناصر الحركة و تنفيذها بشكل متتالي , مثال لإنشاء نافذة تحوي مستطيل , يتحرك بشكل لا نهائي إلى أقصى اليمين من النافذة, يتليها توقف عن الحركة لمدة "500" ميلي / ثا , ثمّ إلى يعود للتحرك إلى أقصى اليسار, يتليها توقف عن الحركة لمدة "500" ميلي / ثا , انظر الرمّاز :

```
import QtQuick 1.0

Rectangle {
    id:win
    width: 500
    height: 500
    Rectangle {
        id:rect
        width: 100
        height: 100
```

```
    color: "black"
    SequentialAnimation{
       running: true
    loops: Animation.Infinite
    NumberAnimation {target: rect; property: "x"; to: 400; duration:
1000 }
    PauseAnimation { duration: 500 }
    NumberAnimation {target: rect; property: "x"; to: 0; duration:
1000 }
    PauseAnimation { duration: 500 }
    }
  }
}
```

استخدمنا العنصر "SequentialAnimation" من اجل احتواء إثنان من عناصر الحركة و تنفيذهما بشكل متتالي, العنصر الأول مهمته تحريك عنصر المستطيل إلى أقصى اليمين من النافذة بمدة زمنية قدرها 1 ثانية , أما العنصر الثاني مهمته تحريك عنصر المستطيل إلى أقصى اليسار من النافذة بمدة زمنية قدرها 1 ثانية , استخدمنا الخاصيّة "loops" من اجل تحديد عدد مرات تكرار تنفيذ عناصر الحركة بشكل متتالي , هنا عدد مرات التكرار لا نهائي , بالنسبة للعنصر "PauseAnimation" خاص بإيقاف الحركة مؤقتا مدة زمنية قدرها "500" ميلي / ثا .

في حال كانت قيمة الخاصيّة "running" مساوية "true" سوف يتم بدأ تنفيذ العنصر "SequentialAnimation" .

نفذ التطبيق و اختبر النتيجة .

❖ العنصر "ParallelAnimation" :

مهمته احتواء مجموعة من عناصر الحركة و تنفيذها بشكل متوازي , مثال لإنشاء نافذة تحوي مستطيل , يتحرك بشكل من الجة العليا اليساريّة إلى أقصى اليمين من الأسفل ضمن النافذة , انظر الرمّاز :

```
import QtQuick 1.0


Rectangle {
   id:win
   width: 500
   height: 500
   Rectangle {
      id:rect
      width: 100
      height: 100
      color: "black"
      ParallelAnimation{
         running: true
      NumberAnimation {target: rect; property: "x"; to: 400;
     duration: 1000 ;easing.type: Easing.OutCirc}
      NumberAnimation {target: rect; property: "y"; to: 400;
     duration: 1000 ;easing.type: Easing.OutCirc}
      }
   }
}
```

استخدمنا العنصر "ParallelAnimation" من اجل احتواء إثنان من عناصر الحركة و تنفيذهما على التوازي, العنصر الأول مهمته تحريك عنصر المستطيل إلى أقصى اليمين من النافذة خلال مدة زمنية قدرها 1 ثانية , أما العنصر الثاني مهمته تحريك عنصر المستطيل إلى أسفل النافذة خلال مدة زمنية قدرها 1 ثانية .

بالتالي عند تنفيذ التطبيق سوف يقوم المستطيل بالتحرك إلى الزاوية اليمنى السفلى من النافذة خلال مدة زمنية قدرها ثانية .

نفذ التطبيق و اختبر النتيجة .

❖ عنصر التدرج اللوني "Gradient" :

نتكلم هنا عن إنشاء تدرج لوني "Gradiant" ضمن عنصر ما , و تغيير ألوان هذا التدرج بشكل متدرج ضمن مدة زمنية محددة .

- **إنشاء تدرج لوني :**

- **الخاصيّة "gradient" موجودة ضمن واجهة العنصر "Rectangle", مهمتها احتواء مجموعة من عناصر التدرج اللوني "Gradient"**
- **العنصر "Gradient" موجودة ضمن واجهة العنصر "Rectangle", مهمتها احتواء مجموعة من عناصر نقاط التدرج اللوني "GradientStop"**
- **الخاصيّة "stops" تستخدم من اجل حزم مجموعة من نقاط التدرج اللوني "GradientStop" داخلها .**
- **العنصر "GradientStop" نقطة علّام التدرج اللوني للعنصر "Gradient" .**

مثال لإنشاء نافذة ألوانها "أبيض – فضي - أسود" على التوالي بشكل متدرج ضمن كامل مساحة النافذة :

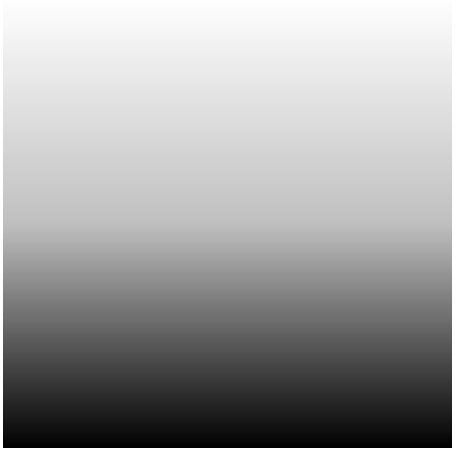
```
import QtQuick 1.0


Rectangle {

   width: 360

   height: 360
```

```
    gradient: Gradient{
        stops: [
            GradientStop{position: 0.0;color: "white"},
            GradientStop{position: 0.5;color: "silver"},
            GradientStop{position: 1.0;color: "black"}
        ]
    }

    MouseArea {
        anchors.fill: parent
        onClicked: {
            Qt.quit();
        }
    }
}
```

أنشئنا داخل كتلة الخاصيّة "gradient" عنصر تدرج لوني "Gradient" و الذي يحوي على ثلاث نقاط , النقطة الأولى موقعها القسم الأعلى بلون أبيض , النقطة الثانية موقعها منتصف النافذة بلون فضي , النقطة الثالثة موقعها أسفل النافذة بلون أسود , سوف يقوم عنصر التدرج "Gradient" بدمج هذه الألون الثلاثة بشكل متدرج عند حد كل نقطة , بالتالي سوف تكون الجهة العليا من النافذة لونها أبيض , بشكل متدرج ينتقل اللون من الأبيض إلى الفضي كلما اقتربت من منتصف النافذة , ثم بشكل متدرج ينتقل اللون من الفضي إلى الأسود كلما اقتربت من أسفل النافذة .

الخاصيّة "position" التابعة للعنصر "GradientStop" تستخدم من أجل إعطاء موقع النقطة التدرج اللوني , القيم التي تقبلها هي المجال المحصور بين "0.0" و "1.0" .

عند تنفيذ التطبيق سوف يظهر كالشكل (10.3) :



الشكل 10.3

- **تغيير اللون بشكل متدرج ضمن العنصر "Gradient" :**

لتغيير لون ما بشكل متدرج ضمن العنصر "Gradient" يتوجب علينا استخدام أحد العناصر التالية :

1 - "SequentialAnimation"
2 - "ParallelAnimation"

مثال توضيحيّ :

لنعدل الرمّاز السابق بحيث يصبح كالأتي :

يتغير اللون الأبيض الموجود ضمن النقطة الأولى إلى الأسود بشكل متدرج خلال مدة زمنية قدرها 2 ثانية , من ثمّ العكس , يتغير اللون من الأسود إلى الأبيض خلال مدة زمنية قدرها 2 ثانية, و يتغير اللون الأسود الموجود ضمن النقطة الثالثة إلى الأبيض بشكل متدرج خلال مدة زمنية قدرها 2 ثانية , من ثمّ العكس .

افتح المشرع السابق و عدّل رمّازه كالأتي :

```
import QtQuick 1.0
```

```
Rectangle {
    width: 360
    height: 360

    gradient: Gradient{
        GradientStop{position: 0.0;
            SequentialAnimation on color{
                loops: Animation.Infinite;
                ColorAnimation{from:"white";to:"black";duration: 2000}
                ColorAnimation{from:"black";to:"white";duration: 2000}
            }}
        GradientStop{position: 0.5;color: "silver"}
        GradientStop{position: 1.0;
            SequentialAnimation on color{
                loops: Animation.Infinite;
                ColorAnimation{from:"black";to:"white";duration: 2000}
                ColorAnimation{from:"white";to:"black";duration: 2000}
            }}
    }

    MouseArea {
        anchors.fill: parent
        onClicked: {
```

```
        Qt.quit();
    }
  }
}
```

ضمن نقطة التدرج الأولى الموجودة داخل العنصر "Gradient" , بدّلنا الخاصيّة "color" بـ العنصر "SequentialAnimation" المتفاعل مع العنصر "color" , يحوي العنصر "SequentialAnimation" على اثنان من العنصر "ColorAnimation" و الذي مهمتهما الإنتقال من اللون الأبيض إلى اللون الأسود بشكل متدرج خلال مدة زمنيّة قدرها 2 ثانية و العكس أي الإنتقال من اللون الأسود إلى اللون الأبيض خلال مدة زمنيّة قدرها 2 ثانية, أما نقطة التدرج الثالثة تحوي على نفس الرمّاز المذكور آنفا , ماعدا فرق وحيد و هو أنّ اللون الأولي للعنصر يكون أسود بدلا من اللون الأبيض , ستبقى عملية تغيير اللون تدرجيّا متواصلة التنفيذ , وذلك من خلال السطر البرمجي التالي :

```
loops: Animation.Infinite
```

و الموجود داخل كتلة العنصر "SequentialAnimation" المضمن داخل نقطة التدرج الأولى و النقطة الثالثة, نفذ التطبيق و اختبر النتيجة .

❖ العنصر "Flickable " :

مهمته تحريك عنصر مرئي على شكل حركة نابض لحظة إفلاته من قوة الشد المطبقة عليه , حيث يتم تنفيذ هذه الحركة لحظة إفلات العنصر المراد من فعل السحب المطبق عليه , يرث العنصر "Item".

مثال , نافذة تحوي عنصر مستطيل حيث عند سحب هذا العنصر و من ثمّ إفلاته يرتد إلى موقعه الأصلي بحركة شبيهة بحركة النابض لحظة إفلاته من شدّه , و ذلك من خلال استخدام العنصر "Flickable" ,

انظر الرمّاز :

```
import QtQuick 1.0
```

```
Flickable {
   width: 500; height: 500
   contentWidth: rect.width; contentHeight: rect.height

   Rectangle{id:rect
   width: 100;height: 100
   color: "black"
   }
}
```

أنشئنا عنصر "Flickable" , مساحته "500X500" , وضعنا قيمة الخاصيّة "contentWidth" مساوية لعرض عنصر المستطيل , و الخاصيّة "contentHeight" مساوية لإرتفاع عنصر المستطيل , مهمة الخاصيّة "contentWidth" و "contentHeight" هي تحديد حجم العنصر المراد تفاعل حركته مع العنصر "Flickable" .

نفذ التطبيق و اختبر النتيجة .

ملاحظة : جميع العناصر المرئيّة التي تحويها كتلة العنصر "Flickable" سوف تتفاعل معه .

يوجد أيضا عنصر يدعى العنصر "Flipable" و التي تكون مهمته إحتواء وجهين من عنصرين مرئيين , وجه أمامي و وجه خلفي , كالقطعة النقدية المعدنية تماما , يتفاعل مع العناصر التالية : "Rotation" و "State" و "Transition" من أجل تكييف حركة التبديل بين وجهينه .

أهم خصائصه :

front : لوضع عنصر مرئي على الوجه الأمامي للعنصر "Flipable"

back : لوضع عنصر مرئي على الوجه الخلفي للعنصر "Flipable"

❖ **تخصيص عرض بوساطة مسار تخطيطي "Path" لنموذج بيانات :**

من أجل إنشاء عرض مخصص لبيانات نموذج يتوجب علينا استخدام العناصر التالية :

1 - **العنصر "PathView" : عنصر عرض مسار تخطيطي مهمته عرض نموذج بيانات مخصص بوساطة العنصر "Path" .**
2 - **العنصر "Path" : عنصر مسار تخطيطي مهمته إحتواء مجموعة من عناصر "Path Elements" .**
3 - **عناصر المسار التخطيطي "Path Elements" : تستخدم من أجل تشكيل اسلوب عرض عناصر نموذج البيانات , و هي :**

عرض عناصر نموذج البيانات بشكل خطي	**PathLine**
عرض عناصر نموذج البيانات بشكل منحني بيزييه التربيعي "نقطة تحكم فقط"	**PathQuad**
عرض عناصر نموذج البيانات بشكل منحني بيزييه التكعيبي "نقطتين تحكم"	**PathCubic**
يستخدم من أجل وضع قيمة خاصيّة تابعة للعنصر المندوب لنموذج البيانات , تتعدل قيمة الخاصيّة المحددة بشكل تدريجي "من الصفر إلى القيمة الموضوعة" على مد المسار	**PathAttribute**
التحكم بمسافة تباعد عنصر البيانات عن عنصر بيانات آخر يقعان بنفس مسار التخطيط , ملاحظة : يتحكم العنصر "PathPercent" فقط بعنصر مسار التخطيط الذي يقع قبله و ذلك من خلال وضع نسبة مئوية لمسافة التباعد بين العناصر , ايضا يتم تعديل مسافة التباعد بشكل تدريجي على مد المسار	**PathPercent**

- **أهم خصائص العنصر "PathView" :**

- **model : model**
- **delegate : Componenet**
- **path : Path**
- **highlight : Component**

- **أهم خصائص العنصر "Path" :**

تحديد نقطة بداية عنصر المسار التخطيطي على المحور الأفقي	**startX : real**
تحديد نقطة بداية عنصر المسار التخطيطي على المحور العمودي	**startY : real**

- **أهم خصائص العنصر "PathLine" :**

الخاصيّة "x" و الخاصيّة "y" من نمط "real" يحددان نقطة النهاية لخط المسار

- **أهم خصائص العنصر "PathQuad" :**

يحدد نقطة نهاية منحني المسار على المحور الأفقي	**x : real**
يحدد نقطة نهاية منحني المسار على المحور العمودي	**y : real**
يحدد نقطة تحكم منحني المسار على المحور الأفقي	**controlX : real**
يحدد نقطة تحكم منحني المسار على المحور العمودي	**controlY : real**

انظر الشكل (10.4) :

الشكل 10.4

- **أهم خصائص العنصر "PathCubic" :**

يحدد نقطة نهاية منحني المسار على المحور الأفقي	**x : real**
يحدد نقطة نهاية منحني المسار على المحور العمودي	**y : real**
يحدد نقطة التحكم الأولى لمنحني المسار على المحور الأفقي	**control1X : real**
يحدد نقطة التحكم الأولى لمنحني المسار على المحور العمودي	**control1Y : real**
يحدد نقطة التحكم الثانية لمنحني المسار على المحور الأفقي	**control2X : real**
يحدد نقطة التحكم الثانية لمنحني المسار على المحور العمودي	**control2Y : real**

انظر الشكل (10.5) :

الشكل 10.5

- **أهم خصائص العنصر "PathAttribute" :**

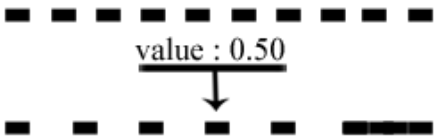
اسم الخاصيّة التي نود تغيير قيمتها	**name : string**

قيمة الخاصيّة	value : real

- اهم خصائص العنصر "PathPercent" :

الخاصيّة "value : real" من اجل تعديل مسافة التباعد بين عناصر النموذج لتساوي النسبة المئوية المعطاة "value" بالنسبة لمسافة التباعد الحاليّة .

انظر الشكل التوضيحيّ (10.6) :



الشكل 10.6

- مثال يوضح كيفيّة استخدام عنصر مسار تخطيطي من أجل تخصبص عرض نموذج بيانات :

تطبيق يحوي عنصر نموذج بيانات بداخله 10 عناصر , كل عنصر يمثل رقم بشكل تدريجي من الواحد إلى العشرة , حيث سوف يتم عرض عناصر القائمة ضمن مسار تخطيطي خطي الشكل , بتدرج لخاصيّة الشفافيّة التابعة لهؤولاء العناصر , انظر الرمّاز :

```
import QtQuick 1.0


Rectangle {
   width: 400
   height: 200
   ListModel{
      id:numbersModel
      Component.onCompleted: {
   for(var i=1;i <= 10;i++)
```

```
    contactModel.append({num : i});
     }
}

 Component {
    id: delegate
    Column {
    opacity: PathView.itemopacity
       Text {
          id: nameText
          text: num
          font.pointSize: 16
       }

    }
 }

 PathView{
    anchors.fill: parent
    model: numbersModel
    delegate: delegate
    path: Path {
       startX: 50; startY: 50
```

```
        PathLine{x:200;y:50}


        PathAttribute{name:"itemopacity";value: 0.8}
    }
  }



}
```

أنشئنا عنصر نموذج قائمة يدعى "numbersModel" يحوي على 10 عناصر قائمة ,محتوياتها أرقام متدرجة من رقم الواحد إلى الرقم عشرة , أننشنا العناصر العشر من خلال استخدام حلقة التكرار "for" و التي نحويتها في "QML" مثيلة لنحوية حلقة "for" في "Java Script" , داخل كتلة حلقة التكرار استخدمنا المنهج "append" التابع للعنصر "numbersModel" من اجل إضافة العناصر العشر , ثمّ انشئنا مكون يدعى "delegate" مهمته تمثيل عنصر مندوب لنموذج البيانات , يحوي هذا المندوب على عنصر عمود داخله عنصر تص , مهمتة عنصر النص عرض خاصيّة "num" الرقم التابعة لعنصر القائمة الموجود داخل عنصر نموذج القائمة "numbersModel" , ضمن العنصر "Column" وضعنا قيمة الخاصيّة "opacity" الخاصة بتحديد مقدار الشفافية هي :

```
opacity: PathView.itemopacity
```

قيمة الخاصيّة "itemopacity" التابعة لعنصر عرض مسار التخطيط الذي سوف يضمّن مكون المندوب "delegate" , أضفنا عنصر "PathView" لعرض مسار التخطيط , وضعنا قيمة نموذجه هي النموذج "numbersModel" , و قيمة خاصيّة المندوب المكون "delegate" , أنشئنا عنصر مسار ضمن الخاصيّة "path" التابعة للعنصر "PathView" , من أجل تشكيل أسلوب عرض نموذج البيانات , حيث نقطة بداية العرض لأول عنصر قائمة "أي الرقم 1" هي الموقع "50X50" , أضفنا داخل العنصر "Path" عنصر مسار تخطيطي خطي الشكل "PathLine" , وضعنا قيمة نقطة نهاية خط المسار هي "200X50" , أس سوف يمتد خط المسار الخاص بعرض عناصر القائمة "الأرقام من الواحد إلى عشرة" من الموقع "50X50" إلى "200X50" , أما بالنسبة للسطر البرمجي :

```
PathAttribute{name:"itemopacity";value: 0.8}
```

وضعنا من خلاله قيمة الخاصيّة "itemopacity" مساوية لـ "0.8" , هذه الخاصيّة وضعناها كقيمة لشفافية عنصر العمود الخاص باحتواء عنصر قائمة , عند تنفيذ التطبيق سوف يبدو كالشكل (10.7) :

الشكل 10.7

جرب أن تسحب هذه العناصر بزر الفأرة و راقب ما سيحصل .

ضمن الرمّاز السابق لنجرب أن ننشأ مسار منحني الشكل بدل من مسا خطي ,عدل الرمّاز الخاص بإضافة مسار خطي "PathLine" :

```
PathLine{x:200;y:50}
```

بدّله بالرمّاز التالي :

```
PathQuad{
        x:200;y:50;
        controlX: 150;controlY: 150
    }
```

عند تنفيذ التطبيق بعد هذا التعديل سوف يبدو كالشكل (10.8) :

الشكل 10.8

بنهاية هذا الفصل نكون قد انهينا قسم "QML" .

❖ خلاصة الفصل :

تكلمنا في هذا الفصل حول عناصر الحركة الساحرة في QML من تسلسل الحركات و تفاعل العناصر الرسومية مع عناصر الحركة , و رأينا كيف و بسهولة إضافة تأثيرات حركية جذابة إلى العناصر الرسومية.

للنتقل إلى القسم الأخير من هذا الكتاب و الذي يتكلم عن Qt Quick و المؤلف من ثلاث فصول.

قسم

Qt Quick

الفصل الثاني عشر

مقدمة في Qt Quick

❖ مقدمة Into :

و هل تستطيع كتابة تطبيقات غنية في QML !!

بما ان لغة QML موجهة لتصميم واجهات مستخدم احترافية فقط , فما هو الحل لكتابة تطبيقات احترافية من الوصول إلى قواعد معطيات و برمجة التطبيقات الشبكية الخ..

الحل في استخدام وحدات نمطية مجموعة في تقنية تدعى Qt Quick ..

ما هذه الوحدات النمطية ؟؟

هي مكتبات تحوي محركات لربط واجهة المستخدم الرسومية المصممة بوساطة QML بصفوف Qt C++ قياسية..

ما هذا الربط ؟ و ما هذه المحركات ؟؟

أولا المحركات هي واجهة برمجية "صفوف" تستطيع من خلالها حضن عناصر QML داخل رمّاز Qt C++.

أما الربط أو الحضن مثلما ذكرنا فهو تفسير رمّاز QML و استخراج عناصر و خصائص هذه العناصر من قبل المحرك و بدوره يقدمها إلى لغة Qt C++ ككائنات معرفة فيها , فبذلك تسطيع من خلال رمّاز Qt C++ تغير و قراءة خصائص عنصار QML .

و كل ما ذكرناه تقدمه لنا تقنية Qt Quick .

أساسيّات في Qt Quick

❖ مقدمة Into :

كيف الوصول إلى عنصر QML من رمّاز Qt C++ ؟ و كيف نستطيع تغير خصائصه ؟

و هل نستطيع إنشاء إنشاء صف Qt C++ و استيراده إلى QML و تنفيذ مناهجه؟ و هل نستطيع عرض عنصر QML منخلال Qt C++ و قراءة حدث منه و تنفيذ رمّاز Qt C++ بعد قدحه ؟

سوف نتلكم بالتفصيل عن كل ما ذكرنا بالإضافة إلى التفويض في Qt Quick ضمن هذا الفصل.

❖ التفاعل بين"QML" و "Qt C++" :

مثلما قلنا مسبقا أن "QML" خاصة فقط بإنشاء واجهات مستخدم تخاطبية , فلذا لغة "QML" بمفردها عديمة الفائدة في صناعة التطبيقات البرمجية , لتصبح ذي فائدة يتوجب تفاعلها مع لغة "Qt" , وذلك يتم بوساطة استيراد ملفات "QML" داخل تطبيق "Qt" , وعرضها إما بنافذة منفصلة , أو ضمن كائن "widget" , تتيح "Qt" تفاعل كامل مع ملفات "QML" , وذلك من خلال العديد من صفوف "Qt" الخاصة بتفاعل "Qt" مع "QML" , أهم هذه الصفوف هي :

خاص باستيراد و عرض ملف "QML" ضمن كائن "widget"	**QDeclarativeView**
يمثل بيئة مكونات "QML" , أي هو جسر المحاكاة بين "Qt" و "QML"	**QDeclarativeEngine**
خاص بتغليف مكونات "QML" ضمن "Qt"	**QDeclarativeComponent**
يستخدم من أجل تعريف بيئة مكونات "QML" داخل محرك "QML"	**QDeclarativeContext**

لتمثيل عناصر "QML" المرئيّة ضمن "Qt"	QDeclarativeItem
وصول إلى خصائص كائنات "QML" و التحكم بها	QDeclarativeProperty

- **استيراد ملف "QML" و عرضه داخل تطبيق "Qt" :**

لاستيراد ملف "QML" و عرضه داخل تطبيق "Qt" , يتوجب علينا أن نستخدم الصف "QDeclarativeView" , مثال لاستيراد ملف "QML" يحوي عنصر مستطيل فضي اللون , و عنصر نص داخله موضوع في أوسطه يحوي النص "Hasan Al-Morhej" , حيث سوف يتماستيراد الملف ضمن كتلة "main" الخاصة بالمشروع ككل :

أنشأ مشروع فارغ و سمه "qmlViewer" , أنشأ داخله ملف "QML" يدعى "rectItem" , اكتب داخله الرمّاز التالي :

```
import QtQuick 1.0


Rectangle {
   width: 400
   height: 400
   color: "silver"
   Text {
      id: txt
      anchors.centerIn :  parent
      text: qsTr("Hasan Al-Morhej")
   }
}
```

ثمّ أضف السطر الأوامر التالي ضمن ملف المشروع "qmlViewer.pro" :

```
QT += declarative
```

مهمته استيراد المكتبات الخاصة بتفاعل "Qt" مع "QML" .

ثمّ انشأ ملف "main.cpp" , و اكتب داخله الرمّاز التالي :

```
#include <QApplication>

#include <QtDeclarative>

#include <QUrl>


int main(int argc, char** argv)

{

   QApplication app(argc, argv);


   QDeclarativeView view;

   view.setSource(QUrl::fromLocalFile("rectItem.qml"));

   view.show();


   return app.exec();

}
```

ضمّنا الملف "QtDeclarative" الذي يحوي جميع المكتبات الخاصة بتفاعل "Qt" مع "QML", ضمن كتلة "main" صرّحنا عن مثيل للصف "QDeclarativeView" يدعى "view" الخاص باستيراد و عرض ملفات "QML", ثمّ استدعينا المنهج "setSource(QUrl)" من أجل استيراد الملف "rectItem.qml" , بعدها أظهرنا النافذة التي تحوي الواجهة "rectItem" , نفذ التطبيق و اختبر النتيجة .

في حال أردنا أن نستورد ملف "QML" موجود داخل ملف مورد "Resource File" , يتوجب علينا ان نمرر للوسيط "QUrl" النص التالي :

```
view.setSource(QUrl("qrc:/qmlFiles/main.qml"));
```

ملاحظة : عند تنفيذ التطبيق إذا أعطى منقح "Qt" خطأ فحواه أن ملف "rectItem.qml" غير موجود , يتوجب عليك أن تضع الملف "rectItem.qml" داخل المجلد الخاص بتجميع المشروع , غالبا مسار المجلد هو نفس مسار المجلد الذي يحوي على المشروع أما اسمه يكون :

" qmlViewer-build-desktop-Qt_4_7_4_for_Desktop_-_MinGW_4_4__Qt_SDK__Debug " .

البنية الهرميّة للصف "QDeclarativeView" هي :

QObject → QWidget →QGraphicsView → QDeclarativeView

• الوصول إلى خصائص عنصر "QML" :

للوصول إلى خصائص عنصر "QML" كـ خاصيّة اللون "color" لعنصر المستطيل , يتوجب علينا استخدام الصف "QDeclarativeProperty" , وذلك من خلال استخدام المنهج "()read" من أجل القراءة و المنهج "()write" من أجل تعيين قيمة للخاصيّة المحددة , أضف الرمّاز التالي ضمن الكتلة "main" الموجودة في المشروع السابق :

```
QDeclarativeProperty pro(view.rootObject(),"color");

pro.write("blue");
```

صرّحنا عن مثيل للصف "QDeclarativeProperty" يدعى "pro" , مررنا لوسطاءه الكائن المراد تغير خاصيّة اللون له "()rootObject" والذي يمثل عنصر المستطيل , و اسم الخاصيّة "color" , بالنسة للمنهج "()rootObject" فهو يعيد قيمة من نمط "QGraphicsObject*" , والذي يمثل العنصر الرئيسي لملف "QML" الذي قد تمّ استيراده ,العنصر الرئيسي هو عنصر المستطيل "rect" , و بما أننا نريد تغيير لونه , يتوجب علينا تغيير قيمة الخاصيّة "color" , لذا مررنا لوسيطه الثاني النص "color" والذي يمثل اسم الخاصيّة المراد الوصول إليها إن كان من أجل القراءة أو من اجل الكتابة , بالنسبة للمنهج "()read" يرجع قيمة من نمط "QVariant" .

• إنشاء خاصيّة "QML" ديناميكية من خلال "Qt C++" :

لإنشاء خاصيّة "QML" ديناميكيّة بوساطة "Qt" يتوجب علينا استخدام الصف "QDeclarativeContext" و استدعاء المنهج

" setContextProperty(const QString &name,QObject* value) "

للتوضيح : لننشأ خاصيّة تدعى "winColor" من أجل تحديد لون الخلفيّة لكامل النافذة , وخاصيّة أخرى اسمها "fntColor" لتحديد لون الخط , انظر رمّاز "QML" :

```
Rectangle {id:win
   objectName: "win"
   width: 400
   height: 400
   color: winColor
   TextInput{
   id:txtInput
   objectName: "txtInput"
   color: fntColor
   text: "Hasan Al-Morhej"
}
}
```

مثلما تلاحظ وضعنا الخاصيّة "winColor" لقيمة لون النافذة "عنصر المستطيل" و الخاصيّة "fntColor" للون الخط , ولكن نحن لم نعرف بعد هذه الخصائص !! , لنرى كيفيّة تعريفهم بوساطة "Qt" :

```
   QDeclarativeView view;
   view.rootContext()->setContextProperty("winColor","black");
   view.rootContext()->setContextProperty("fntColor","white");
   view.setSource(QUrl::fromLocalFile("main.qml"));
```

```
view.show();
```

استخدمنا المنهج "rootContext()" و الذي يعيد مؤشر للكائن "QDeclarativeContext" التابع لـ محرك كائن العرض "QDeclarativeView" , من خلال المنهج "setContextProperty" أنشئنا خاصيّة ديناميكية و وضعنا قيمة لها , مثال لهذا الموضوع المشروع "dynamicProperty" تجده داخل القرص المرفق .

• تفاعل مقابس "Qt" مع أحداث "QML" :

في حال أردنا أن نضيف العنصر "MouseArea" داخل ملف "rectItem.qml" , بحيث عند الضغط عليه يتم الخروج من النافذة , فهل نستطيع فعل ذلك فقط من خلال إضافة الرمّاز الإعتيادي الخاص بنقر زر الفأرة على المساحة المعطاة بعدها استدعاء المنهج "Qt.quit()" الخاص بالخروج , جرب أن تضيف هذا الرمّاز داخل الملف "rectItem.qml" ضمن كلتة عنصر المستطيل, و اختبر النتيجة :

```
MouseArea{

anchors.fill: parent

onClicked: Qt.quit()

}
```

عندما يتم الضغط بزر الفأرة الأيسر ضمن مساحة النافذة , سوف يظهر ضمن نافذة المنقح رسالة التنبيه التالية :

```
Signal QDeclarativeEngine::quit() emitted, but no receivers connected
to handle it.
```

التي تدل على أن الحدث الداخلي "quit" قد تم رفعه لكن لا يوجد أي استجابة من المقبس المتصل به , وذلك لإن المنهج "quit" ليست مهمته تعديل أي من خصائص عناصر "QML" بل مهمته على مستوى تطبيق "Qt Quick" ككل أي على مستوى الصف "QDeclarativeEngine", في هذه الحالة يتوجب علينا أن ننشاء اتصال بين حدث "QML" و مقبس "Qt" .

مثلما تلاحظ ضمن الرسالة السابقة أنه تمّ رفع الحدث من الصف "QDeclarativeEngine" و الذي يعد جسر التواصل بين "QML" و "Qt" لإنه يمثل بيئة مكونات "QML" في "Qt", من هذا الكلام نستنتج أنه يجب علينا ان ننشأ إتصال بين

الحدث الداخلي "quit" التابع للحدث "QDeclarativeView::Engine" في الرمّاز السابق يكافئ الحدث "view.engine()", و بين المقبس "close()" التابع للمثيل "QDeclarativeView" في الرمّاز السابق يكافئ المثيل "view" , من أجل الخروج من النافذة عند الضغط بزر الفأرة الأيسر على النافذة , افتح الرمّاز السابق و أضف ضمن الكتلة "main" فوق السطر البرمجي "view.show()" , السطر البرمجي التالي :

```
QObject::connect(view.engine(),SIGNAL(quit()),&view,SLOT(close()));
```

نفذ التطبيق و اختبر النتيجة .

- **بناء اتصال بين حدث "QML signal" و بين مقبس "Qt slot" :**

إذا أردنا أن ننفذ رمّاز "Qt C++" عند رفع حدث ما من واجهة "QML" التخاطبية فما يتوجب علينا فعله ؟

لنرى الخطوات التي يجب ان نتبعها لإنجاز هذه المهمة :

نصرح عن حدث داخلي ضمن "QML" :

```
signal sendMsg(string msg)
```

من ثمّ نستدعيه :

```
Keys.onReturnPressed: win.sendMsg(txtInput.text)
```

داخل رمّاز "Qt C++" يجب أن نصرح عن صف يحوي مقبس له نفس وسطاء الحدث المراد إنشاء اتصال به :

```
class testSlot : public QObject {

Q_OBJECT
```

```
public :
   explicit testSlot(QObject* parent = 0) ;
   ~testSlot();

public slots:
   void showMsg(const QString &msg);
};
```

حقق رمّاز المستقبل بحيث عند استدعائه تظهر رسالة تحوي نص الوسيط "msg" :

```
void testSlot::showMsg(const Qstring &msg){
QMessage::information(0,"",msg);
}
```

لاحظ وسيط المقبس "showMsg(const QString &msg)" يكافئ "string" في "QML", بعدها ننشأ الإتصال بين الحدث "sendMsg(string)" و المقبس "showMsg(QString)" كالآتي :

```
   QDeclarativeView view;
   view.setSource(QUrl::fromLocalFile("main.qml"));
   QObject* objRect = view.rootObject() ;
   testSlot test;
   QObject::connect(objRect,SIGNAL(sendMsg(QString)),&test,
   SLOT(showMsg(QString)));
   view.show();
```

المنهج "rootObject" يعيد مؤشر للعنصر الأساسي الموجود داخل رمّاز "QML" المحدد , بما ان العنصر الأساسي يحوي الحدث "sendMsg(string)" أي سوف يتم رفع الحدث منه, فيجب أن يمرر للوسيط المرسل التابع للمنهج "connect" ,أما المستقبل "showMsg(Qstring)" محتوى ضمن الصف "testSlot" , فلذا مررناه للوسيط

المستقبل , عند تنفيذ التطبيق سوف تلاحظ أمه عند الضغط على مفتاح "Enter" داخل العنصر "TextInput" سوف تظهر رسالة تحوي نص العنصر "TextInput" , ***سوف تجد رمّاز المشروع "qmlSignal-qtSlot" كامل داخل القرص المرفق .***

• تنفيذ منهج "QML" بوساطة "Qt C++" :

نستطيع تنفيذ منهج "QML" بوساطة "Qt" من خلال استخدام "invoke" أي تنفيذ منهج زمن تنفيذ التطبيق , و هذا يعد الجزء الأساسي من البرمجة الديناميكية "+ delegate invoke" , الشكل العام لاستدعاء منهج ديناميكيا :

لنفرض وجود المنهج "func(msg)" ضمن رمّاز "QML" :

```
function func(msg){
   console.log(msg)
   var date = new Date();
   return msg + ":" + date.toTimeString();
}
```

نستدعيه من "Qt" بوساطة الرمّاز التالي :

```
QObject* objRect = view.rootObject() ;
QVariant ret;
QMetaObject::invokeMethod(objRect,"func",
Q_RETURN_ARG(QVariant, ret),Q_ARG(QVariant, "send msg to qml func"));
```

أولا أخذنا قيمة مؤشر العنصر الرئيسي لرمّاز "QML" الذي يحوي المنهج المراد استدعائه , صرّحنا عن مثيل للنمط "QVariant" يدعى "ret" والذي مهمته تخزين القيمة المعادة من المنهج المحدد , استخدمنا النمط "QVariant" لأننا لا نعلم ما القيمة التي سوف يرجعها منهج "QML" ((func(msg) , QVariant سوف يقوم بتحديد النمط المطابق للقيمة

المخزنة داخله, وذلك يتم بشكل تلقائي زمن التنفيذ , سوف يتم استدعاء المنهج "invokeMethod" التابع لـ الصف "QMetaObject", الصف

"QMetaObject" مختص بتوصيف محتوى الكائن , أما المنهج "invokeMethode" فهو لإستدعاء منهج بشكل ديناميكي زمن التنفيذ , بالنسبة لوسطاءه فالوسيط الأول هو الكائن الذي يحوي المنهج المراد تنفيذه , أما الوسيط الثاني اسم المنهج هنا "func" , الوسيط الثالث لتحديد متحول من أجل تخزين القيمة المرجعة من المنهج , استخدمنا الماكرو "Q_RETURN_ARG(type, type value)" من أجل تحديد نمط القيمة العائدة من المنهج , وسيطه الأول هو اسم النمط المراد تحديده لنمط القيمة المعادة , أما وسيطه الثاني هو المتحول المراد تخزين القيمة داخله , لنعد إلى تكملة تعداد وسطاء المنهج "invokeMethode" , الوسيط الرابع يستخدم لتمرير قيمة الوسيط الأول للمنهجة المراد تنفيذه "func(msg)" , بالنسبة للماكرو "Q_ARG(type,const type &value)" فهو من أجل تحديد نمط قيمة الوسيط للمنهج المراد ااستدعائه , وقيمة الوسيط , في حال كان يوجد أكثر من وسيط ضمن المنهج المراد تنفيذه , يتيح المنهج "invokeMethode" تمرير تسعة "9" وسطاء , سوف تجد داخل القرص المرفق مثال توضيحيّ كامل لهذا الموضوع يدعى "invoke" مطور عن المثال السابق .

• الوصول إلى عنصر "QML" ابن بوساطة "Qt" :

في حال أردنا أن نصل إلى عنصر "QML" موجود داخل عنصر آخر من خلال "Qt" فما الذي يتوجب علينا فعله لتنفيذ هذه المهمة ؟

من أجل الوصول إلى عنصر محدد نستطيع استخدام المنهج

```
T QObject::findChild<T>( const QString &name = QString())
```

اما إذا أردنا جلب جميع العناصر الموجودة ضمن عنصر "QML" أب فنستطيع ذلك من خلال استخدام المنهج :

```
QList<T> QObject::findChildren<T>( const QString &name = QString())
```

للتوضيح :

لنفرض وجود عنصر مستطيل أب لعنصرين مستطيل :

```
Rectangle{
objectName:"win"
width : 400;height: 200
color : "black"
Rectangle{
objectName:"child1"
width : 200;height: 200
color : "silver"

}
Rectangle{
objectName:"child2"
width : 200;height: 200
color : "gray"

}
}
```

الخاصيّة "objectName" من اجل وضع اسم حدث العنصر , إذا أنشئنا داخل النافذة "المستطيل الأب" اثنان من عناصر المستطيل , الأول يدعى "child1" لونه "فضي" , الثاني يدعى "child2" لونه "رمادي" , لننتقل إلى رمّاز "Qt" الخاص باستيراد هذا الرمّاز و الوصل إلى العنصرين,

"child1-child2" , انظر الرمّاز :

```
QDeclarativeView view;
```

```
view.setSource(QUrl::fromLocalFile("main.qml"));


QObject* rootObj = view.rootObject() ;

QDeclarativeItem* item = rootObj-
>findChild<QDeclarativeItem*>("child1");

if(item)

   item->setProperty("color","red");


view.show();
```

صرحنا عن مؤشر لكائن العنصر الأساسي ضمن "QML" وهو "win" , ثمّ وصلنا إلى عنصر المستطيل الأول "child1" من خلال استدعاء المنهج " findChild<T>(QString name)" , والذي سوف يرجع مؤشر لعنصر المستطيل , و بما أن عناصر "QML" المرئية مشتقة من الصف "QDeclarativeItem" حددناه لنمط العنصر , غيرنا قيمة خاصيّة اللون للعنصر "child1" إلى اللون الأحمر , نستطيع تغيير أي خاصيّة لهذا العنصر من خلال استدعاء المنهج :

"setProperty(const char* name,const QVariant &value)"

إذا أردنا أن نستعرض أسماء العناصر الأبناء جميعها , يتوجب علينا استخدام المنهج "findChildren<T>(QString name)" , انظر الرمّاز :

```
QStringList objNnames ;

QList<QDeclarativeItem*> items = rootObj-
>findChildren<QDeclarativeItem*>();

if(items)

   for(int i=0; i <= items.length() -1 ; i++)

      objNnames.append( items.at(i)->objectName() );
```

سوف يحوي المتحول "objNames" جميع أسماء العناصر الأبناء للعنصر الأب "win" .

- **عرض عنصر "QML" داخل كائن مشهد :**

بما أن الصف "QDeclarativeView" مشتق من الصف "QWidget" فهذا يعني أننا نستطيع عرض عنصره الأساسي "عنصر (QML) الأب" ضمن أي كائن "widget" كالكائن "QVBoxLayout" :

```
QDeclarativeView view;

. . .

QVBoxLayout layout;

layout.addWidget(view);
```

لكن في حال أردنا أن نعرض فقط عنصر "QML" ابن "كالعنصر (child1) في المثال السابق" فما يتوجب علينا فعله ؟

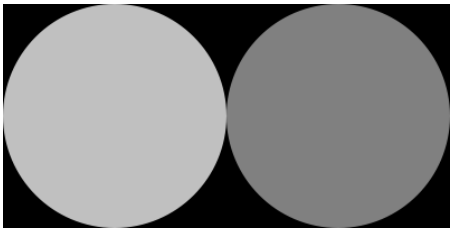
الصف "QDeclarativeComponent" نستورد من خلاله رمّاز "QML" المراد , و نحدد محرك هذا الرمّاز "QDeclarativeEngine" الخاص بتنفذه , نستدعي المنهجيّة "create()" التابعة للصف "QDeclarativeComponent" و التي تعيد مؤشر لحدث مكون "QML" الذي قد تمّ استيراده وهو من نوع "*QObject", نحول "نكسر"هذا الحدث إلى الصف "*QGraphicsObject" , بالتالي سوف نستطيع التعديل على شكل هذا العنصر و أيضا إضافته إلى عنصر مشهد رسومي "QGraphicsScene" , بعدها نضع هذا المشهد ضمن كائن "عارض المشهد الرسومي - QGraphicsView" , مثال توضيحيّ :

نافذة "QML" تحوي على اثنان من العنصر "Rectangle" ذي شكل دائري :

```
import QtQuick 1.0


Rectangle {id:win

  objectName: "win"

  width: 400
```

```
    height: 200

    color: "black"

    Rectangle {id:child1
        objectName: "child1"
        width: 200
        height: 200
        color: "silver"
        radius: 100

    }
    Rectangle {id:child2
        objectName: "child2"
        width: 200
        height: 200
        color: "gray"
       anchors.right: parent.right
       radius: 100
    }
}
```

الشكل (11.1) هو رمّاز "QML" السابق عند تنفيذه :



الشكل 11.1

من أجل الوصول إلى العنصرين الدائريين الشكل "child1-child2" بوساطة "Qt", من ثمّ عرضهم ضمن مشهد رسومي , اتبع الرمّاز التالي :

```
QGraphicsView view;

QGraphicsScene* scene = new QGraphicsScene();

QDeclarativeEngine* engine = new QDeclarativeEngine;

QDeclarativeComponent rootComponent(engine,

QUrl::fromLocalFile("main.qml"));


QGraphicsObject* rootObj =

qobject_cast<QGraphicsObject*>(rootComponent.create());


if(rootObj){

QGraphicsObject* obj1 = rootObj-
>findChild<QGraphicsObject*>("child1");

QGraphicsObject* obj2 = rootObj-
>findChild<QGraphicsObject*>("child2");


if(obj1 != NULL && obj2 != NULL){

scene->addItem(obj1);

scene->addLine(200,0,200,200,QPen(QColor(qRgb(0,255,0))));
```

```
scene->addLine(0,200,200,200,QPen(QColor(qRgb(0,255,0))));
obj2->setOpacity(0.2);
obj2->setProperty("color","red");
obj2->setY(200);
scene->addItem(obj2);
view.setScene(scene);
}
}

view.show();
```

صرحنا عن مثيل للصف "QDeclarativeComponent" يدعى "rootComponent" , مررنا لوسيطه الأول محرك بيئة "QML" والذي يدعى "engine" , أما وسيطه الثاني مررنا له مسار ملف "QML" الذي نود تمثيل عنصره الرئيسي كمكون "QDeclarativeComponent" , اما مثيل الكائن الرسومي "rootObject" صرحنا عنه من أجل احتواء حدث مكون "QML" الرئيسي :

```
qobject_cast<QGraphicsObject*>(rootComponent.create())
```

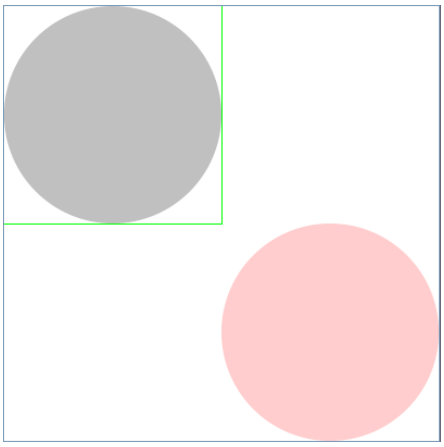
حوّلنا حدث مكون "QML" الرئيسي "win" من نمط "QObject" إلى نمط "QGraphicsObject" , لإننا نود أن نضع عذا المكون ضمن كائن مشهد "Scene" , صرّحنا عن اثنان من مؤشرات لكائن رسومي "obj1 – obj2" , يشير الأول إلى العنصر "child1" أي يشير إلى الشكل الدائري الأول , أما الثاني يشير إلى "child2" الشكل الدائري الثاني :

```
QGraphicsObject* obj1 = rootObj-
>findChild<QGraphicsObject*>("child1");
QGraphicsObject* obj2 = rootObj-
>findChild<QGraphicsObject*>("child2");
```

استخدمنا المنهج "findChild" من أجل إرجاع مؤشر لكائن العنصر المراد , أضفنا هذه العناصر إلى كائن المشهد الرسومي "scene" , من ثمّ عدلنا على العنصر الرسومي الثاني , الخصائص الذي قد تمّ تعديلها هي اللون أصبح لونه أجمر , و الشفافيّة أصبحت مساوية لـ "0.2" , و موقع العنصر على محور "Y" العمودي .

سوف تجد هذا المثال باسم "elementViewer" .

عند تنفيذ التطبيق سوف يظهر كالشكل (11.2) , قارن بين الشكل "11.1" و الشكل "11.2" :



الشكل 11.2

- ## استخدام صف / كائن "Qt" ضمن "QML" :

في حال أردنا أن نستخدم صف "Qt C++" ضمن "QML" ما يتوجب علينا فعله ؟

تكلمنا في فقرات سابقة عن كيفيّة إنشاء خاصيّة "QML" و تغيير قيمتها بوساطة "Qt" من ثمّ إضافتها لتطبيق "QML" المحدد , من أجل إنشاء خاصيّة "QML" بوساطة "Qt" استخدمنا المنهج :

"setContextProperty(const QString &name, QObject *value)" التابع لـ الصف "QDeclarativeContext" , من أجل استخدام صف مبني بوساطة "Qt C++" ضمن "QML" , يتوجب علينا أن نستخدم نفس المنهج السابق
"setContextProperty(..)" من أجل إنشاء حدث للكائن ضمن "QML", و ضمن رمّاز "QML" يتوجب علينا أن نستخدم اسم الصف "الكائن" الذي قد تمّ تعريفه من ثمّ نستدعي منه المنهج المراد تنفيذه "بعد معامل النقطة", بالنسبة للمنهج الذي نود استدعائه الموجود ضمن صف "Qt" , يجب أن تكون بادئة تعريفه هي الماكرو "Q_INVOKABLE" و الذي

تتلخص مهمته في تسجيل هذه المنهج ضمن نظام توصيف*الكائن* , و تفعيل تنفيذه بوساطة المنهج :

QMetaObject::invokeMethod()

تعليق : نقصد بـ*الكائن* هو *الصف* المحتوي للمنهج الذي قد تمّ تسجيله , و الذي دعّم نظام توصيف محتوى الكائن من خلال استخدام الماكرو "Q_OBJECT" .

ملاحظة : سوف يقوم الماكرو "Q_OBJECT" بتسجيل جميع المقابس "slots" ضمن نظام توصيف محتوى الكائن "Meta-Object System" .

للتوضيح : على فرض أنه يوج لدينا صف "Qt C++" يدعى "msgs" , يحوي هذا الصف على منهجين و مقبس واحد :

```
public:

   Q_INVOKABLE void error(const QString&);

   Q_INVOKABLE void info(const QString&);


public slots:

   void about();
```

مهمة المنهج "error" عند استدعائه هي إظهار صندوق رسالة خطأ تحوي محتوى الوسيط , أما المنهج "info" مهمته هي إظهار صندوق رسالة معلومات تحوي محتوى الوسيط , بالنسبة للمقبس "about()" من أجل إظهار صندوق رسالة حول البرنامج , لاحظ أنه عندما عرّفنا المقبس "about()" لم نكتب في بادئته الماكرو "Q_INVOKABLE" , أما ضمن المناهج "info(..) – error(..)" كتبنا ضمن بادئتهم الماكرو "Q_INVOKABLE" , تحقيق المناهج السابقة :

```
void msgs::error(const QString &msg){

   QMessageBox::critical(0,"error",msg);

}


void msgs::info(const QString &msg){
```

```
    QMessageBox::information(0,"info",msg);
}

void msgs::about(){
    QMessageBox::about(0,"Hasan Al-Morhej","Qt C++ Framework");
}
```

ضمن تحقيق المناهج يجب أن لا نكتب في بادئتهم الماكرو "Q_INVOKABLE" , ضمن كتلة التطبيق الرئيسية "main" يجب أن نضيف كائن الصف "mags" لـ ملف "QML" المحدد, وذلك بوساطة المنهج

```
QDeclarativeContext::setContextProperty(const QString &name,
QObject* value)
```

انظر الرمّاز التالي :

```
    QDeclarativeView view;
    msgs msg;
    view.rootContext()->setContextProperty("msgs", &msg);
    view.setSource(QUrl::fromLocalFile("main.qml"));
    view.show();
```

ضمن ملف "main.qml" يتم استدعاء منهج ما من الكائن "msgs" كالآتي :

```
msgs.about()
Component.onComplete: msgs.about()
```

أو :

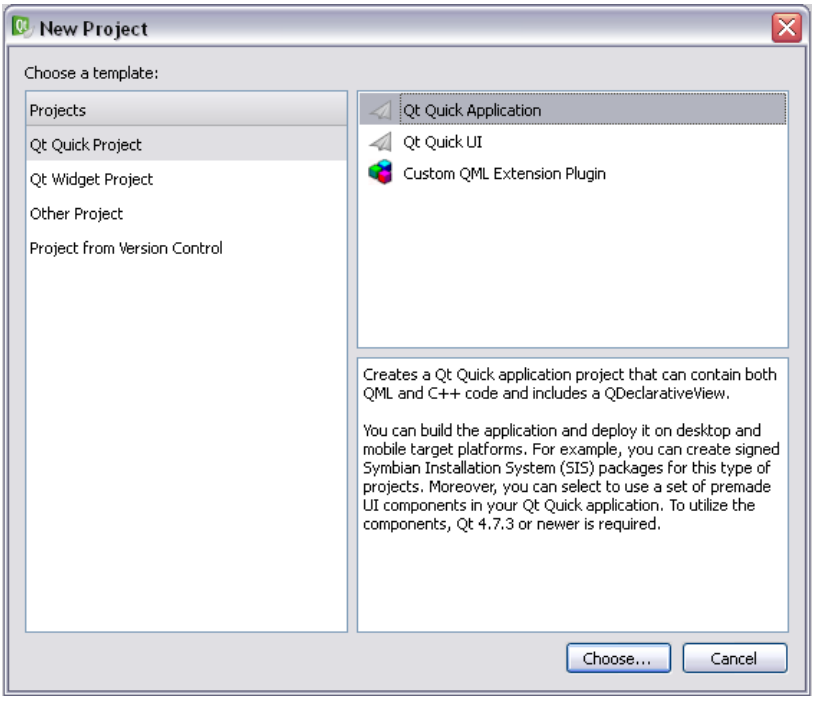
```
msgs.info(qsTr("info..))
onClicked: msgs.info(qsTr("info.."))
```

سوف تجد المشروع "qmlClass" ضمن القرص المرفق .

❖ إنشاء مشروع "Qt Quick Application" :

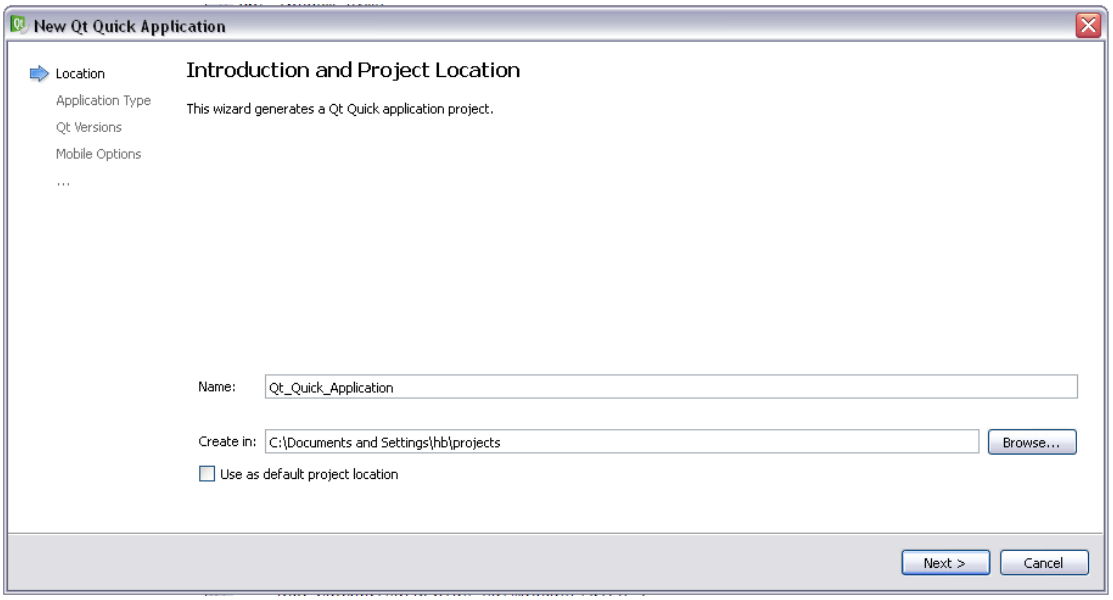
يوجد خيار ضمن بيئة "Qt Creator" لإنشاء تطبيق "Qt Quick" , بحيث عند إنشاءه يضيف الرمّاز المطلوب لتحميل و تهيئة ملفات "QML", بدلا من كتابة رمّاز "Qt C++" الخاص بتحميل و تهيئة و عرض ملف "QML", ننشأ مشروع " Qt Quick Application" .

لإنشاء مشروع "Qt Quick Application" , نختار من قائمة ملف " New File Or Project" من ثمّ نحدد القالب "Qt Quick Project" بعدها نحدد من القائمة نوع المشروع "Qt Quick Application" , كما في الشكل (11.3) :



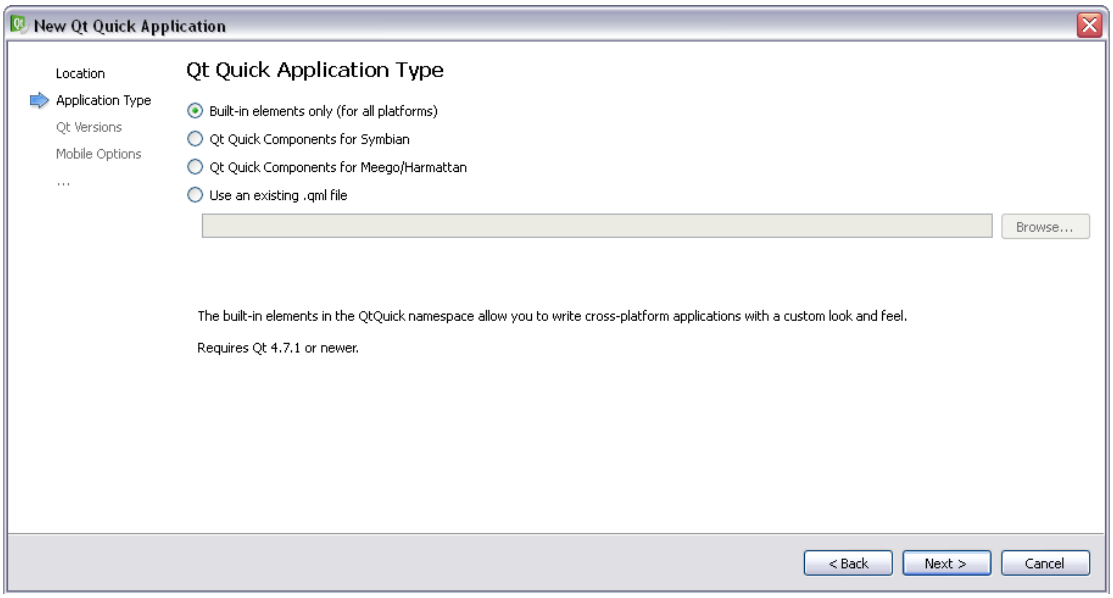
الشكل 11.3

عند الضغط على زر "Choose..." , سوف تظهر نافذة الموقع الخاص بالتطبيق "تحديد اسم و مسار التطبيق" كما في الشكل (11.4) :



الشكل 11.4

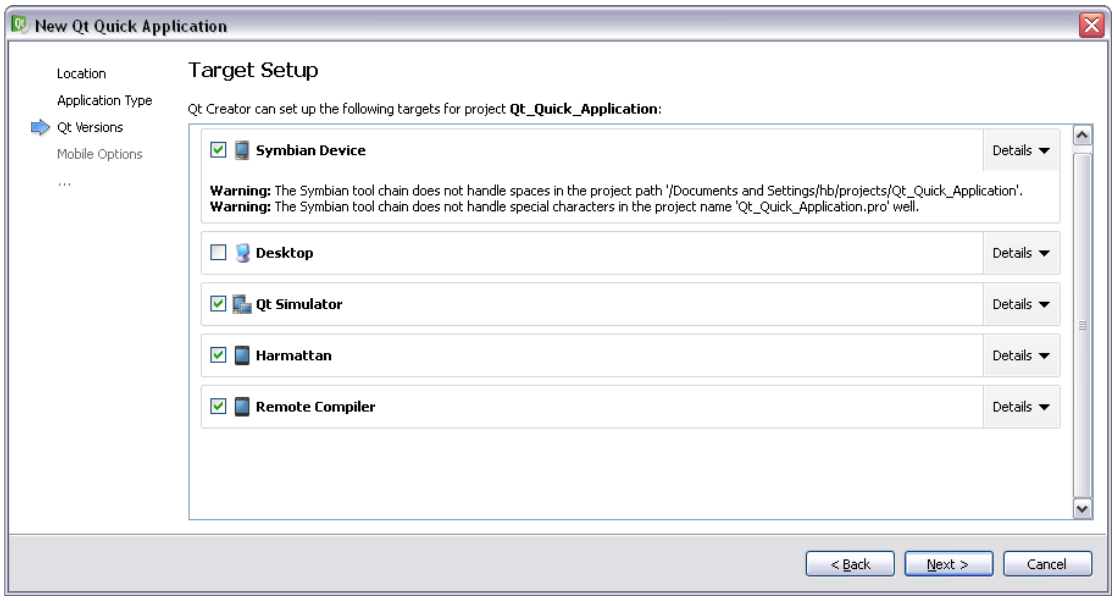
عند الضغط على زر التالي "Next >" سوف تظهر النافذة بتحديد نمط التطبيق انظر الشكل (11.5) :



الشكل 11.5

تسمح هذه النافذة بتحديد المكونات "الوحدة النمطيّة التي تحوي عناصر QML" التي سوف يتم استيرادها داخل ملف "QML" الرئيسي للتطبيق , مثلما تلاحظ بالشكل (11.5) الخيار الأول هو استيراد العناصر التي تعمل على جميع المنصات ', أم الخيار الثاني هو استيراد العناصر التي تعمل على منصة "Symbian" , الخيار الثالث هو استيراد العناصر التي تعمل على منصة "Meego" , أما الخيار الأخير هو استيراد وحدة نمطيّة من ملف خاص , عند

تحديد نمط التطبيق من ثمّ الضغط على زر التالي "Next >" , سوف تظهر النافذة الخاصة بتحديد منصة التجميع و التنفيذ للتطبيق , انظر الشكل (11.6) :



الشكل 11.6

ضمن النافذة السابقة نختار المنصة التي سوف نختبر التطبيق عليها , المنصة الأولى هي :

"Symbian Device" أي تنقيح التطبيق مباشرة على جهاز خارجي حامل لمنصة "Symbian" كجهاز "Nokia N8" , يظهر عند هذا الخيار تنبيه فحواه أنه لا يستطيع الوصول إلى الأدوات الخاصة بتجميع التطبيق لمنصة "Symbian" , وذلك بسبب عدم وجود مسار التطبيق ضمن فهرس "Qt" الرئيسي و الذي هو :

QtSDK/QtCreator/bin

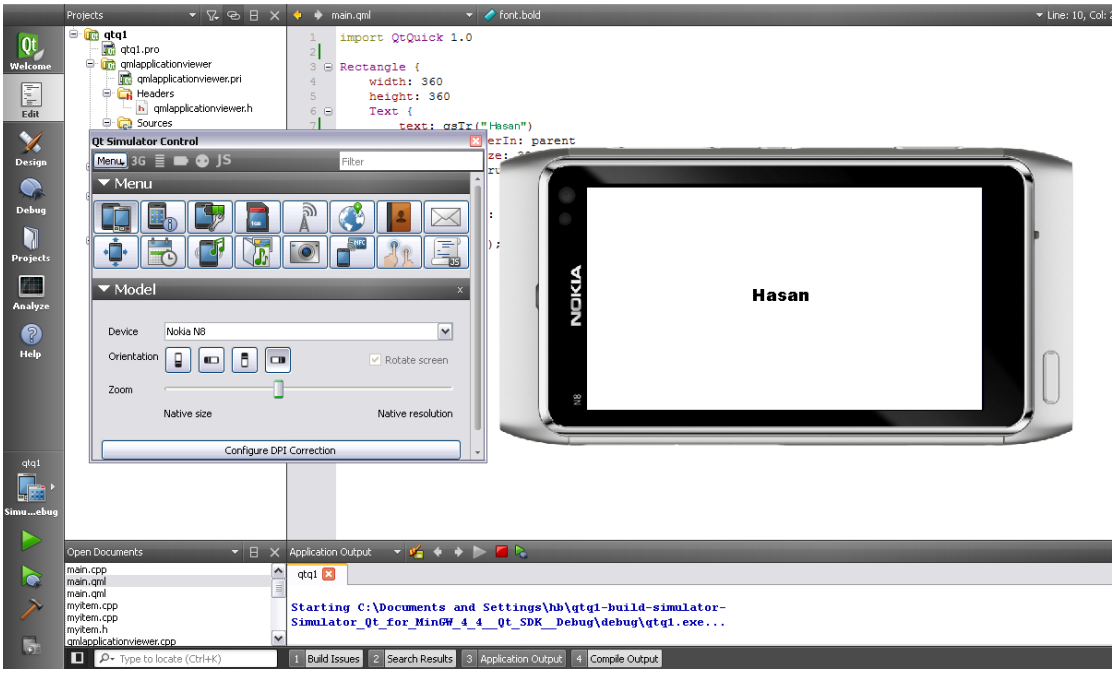
في نظام "Win" إن كان قد نصّب إطار عمل "Qt" داخل قرص "C" هو :

C:\ QtSDK\QtCreator\bin

سوف نتكلم ضمن ملحق الكتاب بالتفصيل عن كيفيّة تجميع و تنقيح التطبيق على جهاز "Symbian" بشكل مباشر .

المنصة الثانية هي "Desktop" أي تنقيح التطبيق على المنصة نظام التشغيل الحالي .

المنصة الثالثة هي "Qt Simulator" أي تنقيح التطبيق على محاكي لنظام "Symbian – S60" , انظر الشكل (11.7) :



الشكل 11.7

يظهر الشكل السابق البرنامج المحاكي لجهاز "Nokia N8" , إطار عمل " Nokia Qt Creator" بشكل إفتراضي يدعم "6" برامج محاكاة و هي :

1- (Nokia N900) 2- (Nokia E71) 3- (Nokia E6) 4- (Nokia N8) 5- (Nokia N95)

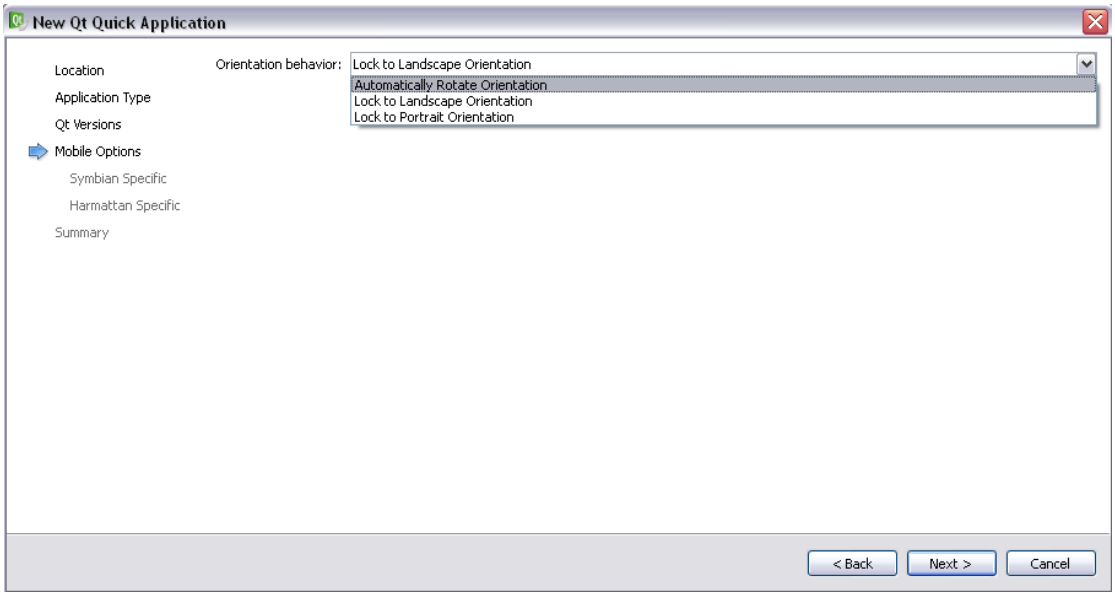
6- (Nokia N97)

سوف نشرح عن برنامج المجاكاة ضمن ملحق من هذا الكتاب .

المنصة الرابعة هي "Harmattan" من أجل التنقيح على منصة "Meego" بوساطة المحاكي المرفق مع حزمة Qt و الذي يدعى "QEMU" .

المنصة الأخير هي "Remote Compiler" و التي نستطيع من خلالها تجميع التطبيق لجميع المنصات التي نود أن يعمل التطبيق عليها كنظام لينكس و ماك و سمبيان الخ ..,و ذلك من خلال خدمة من شركة نوكيا موجود على مخدم نت خاص بهم , حيث يجب أن تملك حساب "qt.nokia" من أجل أن تتاح لك هذه الخدمة , "*سوف تكلم عنها ضمن ملحق الكتاب*" .

النافذة التي تلي هذه النافذة هي "تحديد سلوك التوجيه" , انظر الشكل (11.8) :

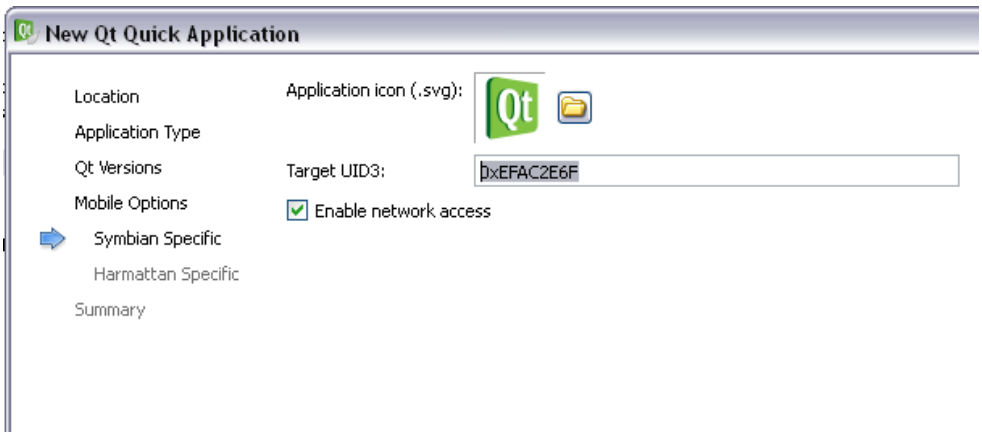


الشكل 11.8

يوجد ثلاث أنواع لسلوك التوجيه وهي :

Automatically Rotate Orientation	**ترك مهمة التوجيه لمنصة العمل الحاملة للتطبيق "حسب توجيه الحساس ضمن أجهزة الجوال"**
Look to Landscape Orientation	**تركيز التوجيه على المحور الأفقي**
Look to Portrat Orientation	**تركيز التوجيه على المحور العمودي**

النافة التي تلي النافذة السابقة هي "تخصيص منصة Symbian" , انظر الشكل (11.9) :



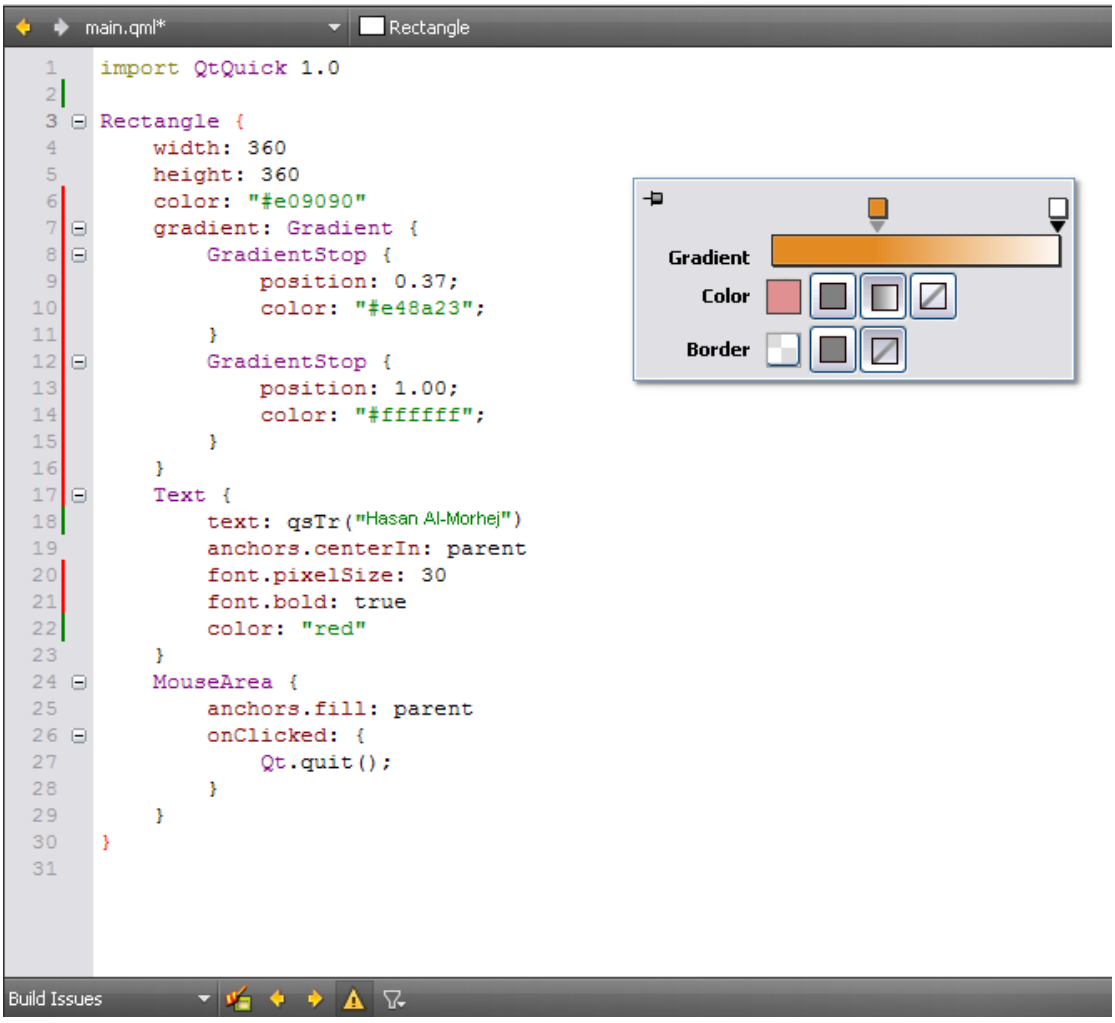
الشكل 11.9

تحوي هذه النافذة على خيار من أجل تغير شعار التطبيق ,سوف يظهر الشعار الحالي في حال قد تم تنفيذ التطبيق على منصة "Symbian", و خيار آخر من اجل وضع معرف المستخدم لتطبيق "Symbian" , أما الخيار الأخير فهو لتفعيل وصول التطبيق لخدمة الشكبة الموجود داخل جهاز الجوال الحواي لمنصة "Symbian" , النافذة التي تلي هذه لنافذة خاصة بتحديد شعار التطبيق الذي سيظهر في حال قد نفذ التطبيق على منصة "Meego" .

عندما ننتهي من إنشاء المشروع باستخدام المعالج الذكي "Wizard" لـ "Qt Creator" سوف نجد أنه قد أنشأ مشروع فرعي "Sub Project" ضمن المشروع الرئيسي , باسم "qmlapplicationviewer", المشروع "qmlapplicationviewer" هو مشروع مكتبة تحوي واجهة خاصة باستيراد و تهيئة ملفات "QML" , المشروع الرئيسي فهو يستخدم تلك المكتبة من أجل استيراد ملفات "QML" , يحوي المشروع الرئيسي أيضا على ملف "main.qml" و الذي يعد ملف "qml" الرئيسي الذي سوف يعرض عند تنفيذ التطبيق .

❖ أشرطة أدوات محرر "QML" :

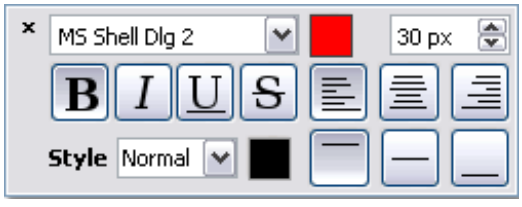
يوجد أربع أشرطة أدوات ضمن محرر "QML" , تظهر أشرطة الأدوات في حال وضعنا المشيرة على اسم عنصر كـ عنصر "Rectangle",حيث يظهر رمّز على شكل مصبح مضاء 💡 ,عند الضغط عليه سوف يظهر شريط الأدوات المخصص لهذا العنصر , نستطيع أيضاً أن نجعل أشرطة الأدوات دائمة الظهور ضمن محرر "QML" لـ "Qt Creator" , من خلال الذهاب إلى قائمة "Tools" ثمّ "Options" تظهر لنا نافذة الخيارات, من النافذة نحدد العنصر "Qt Quick" ثمّ نحدد التبويب "Qt Quick ToolBar" , نفعّل الخيار " Always show Qt Quick Toolbar", بالتالي بمجرد وضع المشيرة ضمن كتلة عنصر سوف يظهر شريط الأدوات الخاص به , أولا شريط أدوات تحرير اللون انظر الشكل (11.10) :



الشكل 11.10

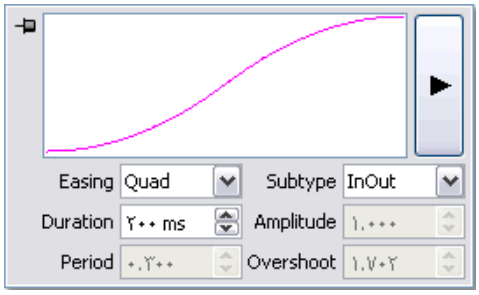
يحوي هذا الشكل على شريط الأدوات الخاص بتغيير اللون لعنصر ما ضمن محرر "QML", تستطيع بوسطته تغيير اللون إما بشكل مصمت أو بشكل تدريجي"Gradient" , حيث سيضاف الرمّاز الخاص بتغيير اللون تلقائيا و بشكل مباشر إلى رمّاز "QML" المحدد عند تغيير اللون .

شريط الأدوات الثاني خاص بتغيير إعدادات الخط من لون إلى نوع الخط الخ .. انظر الشكل (11.11) :



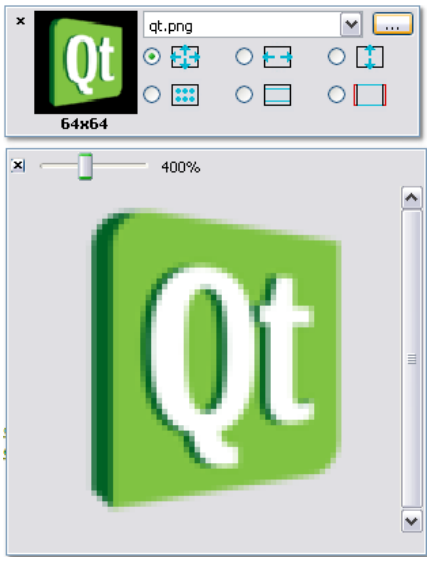
الشكل 11.11

الشريط الثالث خاص بتحديد إعدادت عنصر حركة , انظر الشكل (11.12) :



الشكل 11.12

الشريط الرابع و الأخير خاص بتغيير إعدادات عنصر الصورة , انظر الشكل (11.13) :



الشكل 11.13

❖ مصمم واجهات QML :

ضمن بيئة "Qt Creator" يوجد برنامج فرعي خاص بتصميم واجهات "QML" , تستطيع بوساطته إنشاء واجهات "QML" احترافية بغاية السهولة , انظر الشكل (11.14) :


Navigator
Rectangle
Text
NumberAnimation
MouseArea
عارض العناصر
المضافة ضمن ملف QML
الحالي
main.qml
base state
State1
الصندوق الخاص
بإضافة و إزالة حالة
صندوق
خصائص عنصر
Properties
Type
Type Text
Id
Geometry
Position X Y
Size W H
Visibility
Visibility Visible Smooth
Clip
Opacity
width
height
whole document
100 %
صندوق عارض مشهد
QML
Hasan
Text
Layout
Advanced
Color
Text
Style
Text
Text Hasan
Wrap mode NoWrap
Alignment
Font
Font MS Shell Dlg 2
Size pixels
Font style
Style Normal
Library
Items
Resources
<Filter>
Qt Quick - Ba...
صندوق عناصر
QML
Border Image
Flickable
Flipable
Focus Scope
Image
Item
Mouse Area
Rectangle
Welcome
Edit
Design
Debug
Projects
Analyze
Help
qtq1
MeeG...ease
Type to locate (Ctrl+K)
1 Build Issues
2 Search Results
3 Application Output
4 Compile Output

الشكل 11.14

1 - صندوق الخصائص "Properties" : يحوي جميع خصائص العنصر المحدد ضمن مشهد QML .
2 - صندوق عارض المشهد "Scene Viewer" : يعرض المشهد الحالي لواجة QML , و يعرض أي تغير طرأ على أي عنصر موجود ضمن المشهد .
3 - صندوق المستكشف "Navigator" : يحوي جميع العناصر المضاوجودة ضمن مشهد "QML" الحالي .
4 - صندوق العناصر "Items" : يحوي جميع عناصر "QML" التي نستطيع إضافتها إلى المشهد بوساطة السحب و الإفلات .
5 - صندوق عناصر الحالة "states" : يحوي جميع عناصر الحالة الموجودة ضمن عنصر "QML" الحالي , نستطيع إضافة عنصر حالة من خلال الضغط على زر "+" الموجود ضمن الصندوق , من أجل إزالة حالة موجود نضغط على الزر "-" الموجود في أعلى اليمين لمشهد الحالة .

❖ خلاصة الفصل :

كان الفصل غنيا بالمعلومات التي تتحدث عن التفاعل بشكل كامل بين عناصر QML و رمّاز Qt C++ , من استيراد عنصر QML و تصييره و تغير خصائصه بوساطة Qt C++ و التفويض ..

لننتقل إلى الفصل التالي الذي يتكلم عن كيفية إنشاء مكون QML بوساطة Qt.

الفصل الثالث عشر

إنشاء مكون QML بوساطة Qt

❖ مقدمة Intro :

إذا أردنا إنشاء مكون QML احترافي و بمواصفات تلبي التطبيق الذي نقوم ببنائه و بخصائص معينة و بمناهج غير مقيدة الفعل مثل الوصول إلى الشكبة أو معالجة بيانات معينة بطريقة إحترافية , فنستطيع ذلك من خلال بناء مكون بوساطة Qt C++ موجه إلى QML .

❖ مكون QML :

رأينا في الفصول السابقة كيفيّة استخدام عناصر و مكونات "QML" , منها المرئي و منها الغير مرئي , مثل عنصر المستطيل "Rectangle" عنصر مرئي , أما مكون المؤقت "Timer" غير مرئي , ولكن هل هذه العناصر بنيت بوساطة "QML" ؟

نستطيع إنشاء عناصر و مكونات "QML" ذي استخدام محدود بوساطة "QML" نفسها , و لكن إذا أردنا ان ننشأ عنصر أو مكون "QML" واسع الإستخدام , فيتوجب علينا بنائه بوساطة "Qt C++" , بالنسبة لعناصر "QML" المرئيّة مشتقة من الصف "QDeclarativeItem" , أما مكونات "QML" الغير مرئيّة فـ مشتقة من الصف "QObject" .

❖ تسجيل مكون "QML" :

عند بناء عنصر / مكون "QML" بوساطة "Qt C++" , فيتوجب تسجيله كمنط جديد ضمن مكونات "QML" ليتاح استخدامه ضمنها , و ذلك باستخدام المنهج :

```
int qmlRegisterType<T>(const char *uri, int versionMajor, int
versionMinor, const char *qmlName)
```

الموجود داخل المكتبة "QtDeclarative" , بالنسبة للقالب "<T>" هو الصف الذي يمثل عنصر / مكون "QML" الذي قد تمّ إنشاءه .

وسطاء المنهج :

- **الوسيط "const char* uri" : اسم الفضاء "الوحدة النمطية" الحاوي لـ المكون .**
- **الوسيط "int versionMajor" : الإصدار الأساسي للمكون .**
- **الوسيط "int versionMinor" : الإصدار الثانوي للمكون .**
- **الوسيط "const char* qmlName" : اسم المكون ضمن "QML" .**

يعيد هذا المنهج قيمة بنمط عدد صحيح و هي معرف "ID" النمط الذي قد تم تسجيله ضمن مكونات "QML" .

الشكل العام للمنهج :

***Qt C++* :**

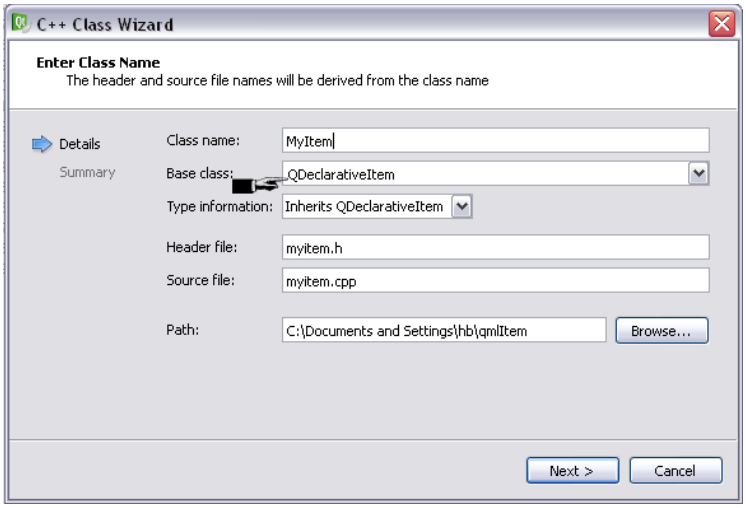
```
Int typeId = qmlRegisterType<MyItem>("com.comp.qmlItem", 1, 0,
"MyComp")

class MyItem : QDeclarativeItem or QObject
```

***QML* :**

```
import com.comp.qmlItem 1.0

MyComp{ id: ... }
```

❖ بناء عنصر QML :

أنشأ مشرع "Qt Quick Application" باسم "qmlItem" , ثمّ أنشأ صف يدعى "MyItem" مشتق من الصف "QDeclarativeItem" انظر الشكل (12.1) :



الشكل 12.1

يمثل الصف "MyItem" عنصر "QML" , مهمة هذا الصف هي فقط رسم عنصر مربع الشكل بلون أزرق على سطح العنصر , اذهب إلى ملف التروسية "myitem.h" , و ضمّن ملف التروسية التالي :

```
#include <QPainter>
```

ثمّ صرح عن المنهج الإفتراضي "paint" و التي تتلخص مهمته في الرسم المباشر على العنصر :

```
public :

  void paint(QPainter *, const QStyleOptionGraphicsItem *, QWidget
*);
```

ضمن ملف التحقيق "myitem.cpp" , اكتب السطر البرمجي التالي داخل كتلة بناء الصف :

```
  setFlag(QGraphicsItem::ItemHasNoContents, false);
```

مهمة هذا السطر هي تفعيل الرسم داخل العنصر الحالي , الآن حقق المنهج "paint" :

```
void MyItem::paint(QPainter *painter, const
QStyleOptionGraphicsItem *

, QWidget *){

  painter->setPen(QPen(QColor(0,0,255),2));
```

painter->drawRect(10,10,380,380);

}

رسمنا مربع غير مصمت لون حدّه أزرق , الآن يتوجب علينا تسجيل العنصر "MyItem" ضمن مكون "QML" , اذهب الملف "main.cpp" و ضمّن التالي :

#include <QtDeclarative>

#include "myitem.h"

المكتبة "QtDeclarative" تحوي المنهج "qmlRegisterType" , أما "myitem" فهو ليتاح استخدام الصف "MyItem" ضمن الملف "main.cpp" , أضف السطر البرمجي التالي :

qmlRegisterType<MyItem>("com.MyNewItem",1,0,"MyItem");

الآت سجلنا الصف "MyItem" كنمط جديد ضمن "QML" يدعى "MyItem" , الوحدة النمطية هي :

com.MyNewItem 1.0

الآن لنجرب أن نضيف هذا العنصر ضمن ملف "main.qml" :

أولا ضمن الوحدة :

import com.MyNewItem 1.0

ثم أضف العنصر "MyItem" ضمن كتلة عنصر المستطيل :

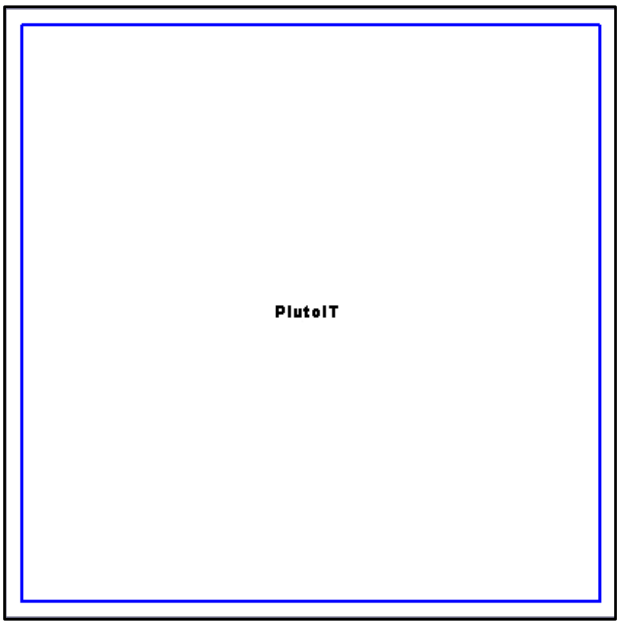
Rectangle{ ....

.. .

MyItem { id: myitem }

}

عند تنفيذ التطبيق سوف ترى كالشكل (12.2) :



الشكل 12.2

إذا قد ظهر المربع الذي رسمناه ضمن العنصر "MyItem" .

❖ الخصائص الديناميكيّة :

عند التصريح عن خاصيّة ديناميكيّة سوف يقوم "Qt" بتسجيل هذه الخاصيّة ضمن نظام توصيف كائن "Qt" بشكل تلقائي, بالتالي يتاح استدعائها من خلال الإستحداث "invoke" , ضمن "QML" لا نستطيع أن أن نستدعي منهج أو خاصيّة غير مسجل ضمن "meta-object system" نظام توصيف الكائنات , أما بالنسبة للمناهج التي نود أن نستدعيها ضمن "QML" فيتوجب أن نضيف الماكرو "Q_INVOKABLE" في بادئة تعريف المنهج المراد .

التصريح عن خاصيّة ديناميكيّة :

```
Q_PROPERTY(type propertyName READ propertyName WRITE set
propertyName NOTIFY signalName)
```

بالنسبة للصفة "NOTIFY" تستخدم من أجل تعريف حدث داخلي "signal" , يجب أن تلحق بـ اسم الحدث الداخلي الذي سوف يقدح "يرفع" عند تغيير قيمة الخاصيّة .

❖ **إنشاء خاصيّة ديناميكيّة لمجمع عناصر :**

من أجل إنشاء خاصيّة تستطيع احتواء قائمة من عناصر "QML" كالخاصيّة "states" ضمن عنصر "Rectangle" , يتجوب علينا أن نستخدم الصف "QDeclarativeListProperty<T>" كنمط لقيمة الخاصيّة :

```
QDeclarativeListProperty<T> ( QObject * object, void * data,
AppendFunction append, CountFunction count = 0, AtFunction at = 0,
ClearFunction clear = 0 )
```

الوسيط "object" : يستخدم من أجل تمرير الكائن الأب للخاصيّة .

الوسيط "data" : يستخدم من أجل تمرير أي بيانات إضافيّة للمناهج المرتبطة بهذه الخاصيّة كالمنهج "AppeandFunction" .

الوسيط "AppendFunction" يستخدم من أجل تمرير المنهج الذي يمثل إضافة عنصر .

الوسيط "CountFunction" يستخدم من أجل تمرير المنهج الذي يرجع عدد العناصر المضافة .

الوسيط "ClearFunction" يستخدم من أجل تمرير المنهج الخاص بإزالة عنصر .

الشكل العام لتعريف خاصيّة مجمع عناصر :

```
Q_PROPERTY(QDeclarativeListProperty<MyItem> items READ items)
```

التصريح عن منهج خاصيّة القراءة :

```
QDeclarativeListProperty< MyItem > items
```

تحقيق منهج خاصيّة القراءة :

```
QDeclarativeListProperty<MyItem> MyClass::items()
{
   return QDeclarativeListProperty<PieSlice>(this, 0,
&MyClass::append_ func)
}
```

ضمن كتلة منهج خاصيّة القراءة أتحنا إضافة عناصر , ارجع للشكل العام للتصريح عن الصف :

QDeclarativeListProperty<T>(...)

التصريح عن منهج الإضافة :

```
static void append_func(QDeclarativeListProperty<MyItem> *list,
MyItem *myItem)
```

التصريح عن متحول خاص باحتواء مجموعة عناصر :

```
QList<MyItem *> m_myItem
```

تحقيق منهج الإضافة :

```
void MyClass::append_func(QDeclarativeListProperty<MyItem> *list,
MyItem * myItem)
{
   MyClass *cls = qobject_cast<MyClass *>(list->object);
   if (cls) {
      myItem ->setParentItem(cls);
      cls->m_ myItem.append(myItem);
   }
}
```

النمط "MyItem" هو صف العنصر المراد احتواء مجموعة منه ضمن هذه الخاصيّة .

الصف "MyClass" هو العنصر الذي يمثل الصف الأب للعنصر "MyItem" , و الذي يستطيع احتواء مجموعة منه بوساطة الخاصيّة "items" .

استخدام الخاصيّة "items" في "QML" :

```
MyClass{
items:[
```

```
MyItem{ id:mi1 },
MyItem{ id:mi2 },
MyItem{ id:mi3 }
]
}
```

العنصر الأب هو "MyClass" الذي سيحوي مجموعة من نمط العنصر "MyItem", داخل الخاصيّة "items" أضفنا العناصر "MyItem" .

❖ مكتبة "Qt" تحوي مكونات QML :

في حال أردنا أن ننشا حزمة تحوي مجموعة من مكونات "QML" , و تكون هذه الحزمة موجودة ضمن ملف تجمع , بحيث نستطيع نقلها و استخدامها بمجرد استيرادها ضمن "QML" ,

فمالذي يتوجب علينا فعله ؟

تتيح "Qt" إنشاء مكتبة تحوي عناصر / مكونات "QML" , وذلك من خلال إنشاء مشروع "Custom QML Extension Plugin" , حيث يحوي على صف مشتق من "QDeclarativeItem" و صف آخر خاص بتسجيل العناصر الموجودة ضمن المكتبة .

إن بناء عنصر "QML" موجود ضمن مكتبة فهو كبناء عنصر "QML" الذي رأينا شرح كيفيّة بناءه ضمن الفقرات السابقة من هذا الفصل .

الفرق الوحيد بين عنصر "QML" موجود ضمن مكتبة و عنصر "QML" غير مضمّن في مكتبة هو في طريقة استيرادهم لمشروع "QML" , حيث لتضمين عناصر "QML" موجودين ضمن مكتبة يتوجب أن ننشأ ملف ضمن مسار مشرع "QML" يدعى "qmldir" , يحتوي على السطر البرمجي الخاص باستيراد المكتبة إلى مشروع "QML" الموجود ضمن نفس مسار الملف "qmldir" .

شكل السطر البرمجي العام هو :

```
plugin libraryName directoryName
```

أما بالنسبة لاستيراد عنصر "QML" موجود داخل مشروع "QML" يتوجب علينا فقط أن نضيف اسم فضائه ضمن ملف "QML" الذي سوف نستخدم هذا العنصر داخله :

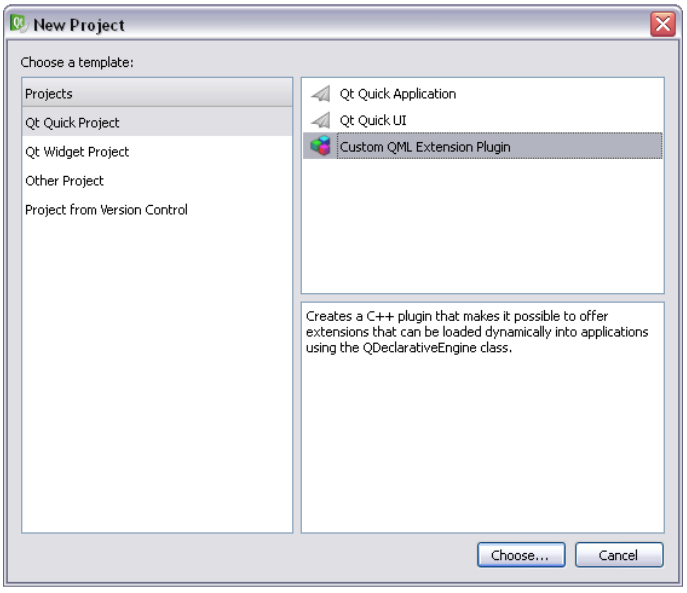
```
import com.itemName  version (1.0)
```

رأينا كيفيّة استيراده ضمن الفقرات السابقة من الفصل الحالي .

❖ إنشاء مكتبة "Qt" تحوي مكونات QML :

لنوضح عن طريق مثال عملي كيفيّة بناء الإضافات "Plugin" و التي نقصد بها مكتبة "Qt" تحوي على مكونات "QML" , المثال هو عبارة عن مكتبة "Plugin" تحوي عنصر صف مهمته تمثيل عنصر "QML" , يعمل هذا العنصر على رسم مضلع ضمن مساحته , حيث يتم رسم المضلع عن طريق إعطاء المستخدم حرية إدخال عدد نقاط المضلع و مواقعها , بعدها يتم رسم المضلع من خلال وصل النقاط مع بعضها البعض , تتيح المكتبة مناهج خاصة بـ اختيار لون حد المضلع و لون مساحته , سوف نترجم هذه المهام برمجيا داخل المكتبة "plugin" , لنبدأ ..

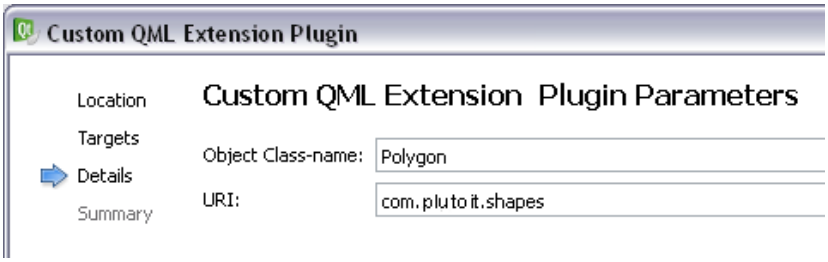
لننشأ مشروع "Custom QML Extension Plugin" , من ملف نختار " New File Or Project" , بعدها نحدد القالب "Qt Quick Project" ثمّ نختار العنصر " Custom QML Extension Plugin" , انظر الشكل (12.3) :



الشكل 12.3

بعد الضغط على زر "Choose" سوف تظهر نافذة حوار خاصة بوضع اسم المشروع , نضع اسم المشروع هو "polygonPlugin" , ثمّ التالي نحدد المنصات التي نود أن تنقيح و تنفيذ

هذه المكتبة عليها , من ثمّ نضغط على زر التالي , تظهر نافذة التفاصيل "Details" , نضع اسم العنصر هو "Polygon" و اسم الفضاء "com.plutoit.shapes" , انظر الشكل (12.4) :



الشكل 12.4

نضغط على زر التالي , بعدها نضغط على زر إنهاء "Finish" , سوف نجد أن المعالج الذكي "Wizard" الخاص بإنشاء "QML Plugin" , قد أنشأ ملفان رأسيان و هما :

polygon.h

خاص بتعريف الواجهة البرمجية للعنصر (الصف) "Polygon" , الصف "Polygon"مشتق من الصف "QDeclarativeItem" .

polygonplugin_plugin.h

خاص بتعريف الواجهة البرمجية للصف الذي سيسجل العناصر الموجودي ضمن المكتبة "PolygonPlugin" , الصف "PolygonPluginPlugin" مشتق من الصف "QDeclarativeExtesionPlugin" الخاص بتهيئة مكتبة "Plugin" و تسجيل عناصرها بنمط عنصر "QML" .

و أيضا أضاف "Wizard" ملفات تحقيق الملفين الرأسيين آنفين الذكر و هما :

polygon.cpp

خاص باحتواء تحقيق صف العنصر "Polygon" .

polygonplugin_plugin.cpp

خاص باحتواء تحقيق صف تسجيل عناصر الـ "Plugin" .

و أخيرا أضاف الـ "Wizard" ضمن الدليل "Other files" الملف "qmldir" و الذي يحوي سطر برمجي خاص باستيراد الـ "Plugin" الحالي لـ مشروع "QML" , من اجل

استخدام عناصر هذه المكتبة ضمن مشروع "QML" , يجب أن نضع الملف "qmldir" ضمن نفس مسار مشروع "QML".

السطر البرمجي هو :

```
plugin polygonPlugin
```

في حال كان ملف المكتبة موجود داخل مجلد يجب أن نكتب اسم / مسار ذلك المجلد بعد اسم المكتبة :

```
plugin polygonPlugin directory
```

أولا بما أننا نريد ان نرسم داخل مساحة العنصر يتوجب أن نزيل محارف التعليق "//" عن السطر البرمجي التالي :

```
setFlag(ItemHasNoContents, false);
```

الموجود ضمن كتلة بناء الصف "Polygon" , اذهب إلى الملف "polygon.h" , ضمّن ملفات التروسية التالية :

```
#include <QGraphicsScene>

#include <QInputDialog>
```

ثمّ ابدأ بتعريف الوجهة البرمجيّة للصف "Polygon" كالتالي :

ضمن القسم العام للصف "Polygon" صرّح عن التالي :

```
Q_INVOKABLE void beginInsert();
```

المنهج الخاص بإظهار نافذة إدخال من أجل تحديد عدد نقاط المضلع , أضفنا الماكرو "Q_INVOKABLE" في بادئته لنستطيع استدعاءه ضمن رمّاز "QML" , أضف المنهج :

```
void insertPoints(int);
```

المنهج الخاص بإظهار نوافذ حوار إدخال على التوالي , من أجل تحديد مواقع نقاط المضلع , عدد نوافذ الإدخال ضعف عدد النقاط , وذلك لإن كل نقطة تحوي على موقع ضمن المحور الأفقي "X" و موقع على المحور العمودي "Y" , أما الوسيط "int" لتمرير عدد نقاط المضلع , أضف المنهج :

```
void draw(QPoint*);
```

المنهج الخاص برسم المضلع بعد تمرير مصفوفة النقاط لوسيطه , في القسم الخاص صرح عن المتحولات التالية :

int m_pc;

يستخدم لتخزين عدد النقاط , أضف المتحول :

QPoint* m_points;

يستخدم لتخزين مصوفة نقاط المضلع , أضف المتحولين :

QColor m_pColor;

QColor m_bColor;

المتحول الأول يخزن لون حد المضلع "لون القلم" , أما المتحول الثاني يخزن لون مساحة المضلع "لون الفرشاة" , صرح عن الخصائص الديناميكيّة التالية تحت الماكرو "Q_OBJECT" :

Q_PROPERTY(int count READ count)

خاصيّة للقراءة فقط , تدعى "count" ترجع عدد نقاط المضلع

Q_PROPERTY(QString points READ points)

خاصيّة للقراءة فقط , تدعى "points" ترجع مواقع جميع نقط المضلع على المحورين "X-Y"

Q_PROPERTY(QColor pColor READ pColor WRITE setPColor

NOTIFY pColorChanged)

خاصيّة للقراءة و الكتابة , خاصة بلون الحد "القلم" لشكل المضلع , الحدث الداخلي "pColorChannged" يرفع عند تغيير قيمتها

Q_PROPERTY(QColor bColor READ bColor WRITE setBColor

NOTIFY bColorChanged)

خاصيّة للقراءة و الكتابة , خاصة بلون مساحة "الفرشاة" للمضلع , الحدث الداخلي "bColorChannged" يرفع عند تغيير قيمتها

التصريح عن مناهج الخصائص , ضمن القسم العام داخل الصف "Polygon" , أضف التعريفات التالية :

```
public:
   int count() const ;
   QString points() const ;
   QColor pColor() const ;
   QColor bColor() const ;
   void setPColor(QColor) ;
   void setBColor(QColor) ;
```

ضمن قسم الأحداث الداخليّة "signals" لـ الصف "Polygon" , صرح عن التالي :

```
signals:
   void pColorChanged() ;
   void bColorChanged() ;
    void end() ;
```

يرفع الحدث "end" عند الانتهاء من رسم المضلع .

انتهينا من التصريح عن الواجهة البرمجيّة للصف "Polygon" , الآن لنبدأ بتحقيق هذه الواجهة ...

اذهب إلى الملف "polygon.cpp" , و نفذ التالي :

أولا تحقيق المنهج "beginInsert()" :

```
void Polygon::beginInsert(){
   m_pc = QInputDialog::getInteger(0,"Polygon",
            "enter count of polygon points",1,1,30);
   insertPoints(m_pc);
}
```

عند استدعاء هذا المنهج سوف تظهر نافذة إدخال عدد صحيح من أجل تحديد عدد نقاط المضلع , بعدها يستدعي المنهج الخاص بإدخال مواقع هذه النقاط , وهو المنهج "insertPoints(int)" :

```
   m_points = new QPoint[count];

   for (int i=0 ; i <= count -1 ; i++){

      m_points[i].setX(QInputDialog::getInteger(0,"Polygon",

                              "point " + QString::number(i) + " set
'X'",0,0,400));

      m_points[i].setY(QInputDialog::getInteger(0,"Polygon",

                              "point " + QString::number(i) + " set
'Y'",0,0,400));

   }

   draw(m_points);
```

بعد وضع جميع توضعات النقاط , نستدعي المنهج "draw(QPoint*)" الخاص برسم المضلع بعد تمرير النقاط لوسيطه :

```
void Polygon::draw(QPoint *ar_point){

   QPolygon pol;

   for(int i=0; i <= m_pc -1 ; i++)

      pol.append(ar_point[i]);


   scene()->addPolygon(pol,QPen(QColor(m_pColor),2),

                              QBrush(QColor(m_bColor) ) );

   emit end();

}
```

صرّحنا عن مثيل لصف مضلع يدعى "pol" , ثمّ أضفنا النقاط لـ مثيل المضلع , بعدها أضفنا مثيل شكل المضلع "pol" إلى كائن المشهد "scene()" و الذي يعيد مؤشر لمشهد العنصر الحالي , المتحول "m_pColor" و المتحول "m_bColor" سوف نضع قيمهما داخل كتلة تحقيق خصائص لون القلم "pColor" و لون الفرشاة "bColor", رفعنا الحدث "end()" و الذي يشير إلى أن المضلع قد تمّ رسمه على مشهد العنصر , الآن لنحقق الخصائص :

خاصيّة عدد نقاط المضلع :

```
int Polygon::count() const {
   return m_pc ;
}
```

خاصيّة تعيد مواقع النقاط :

```
QString Polygon::points() const {
   QString pointsInfo;

   for(int i=0; i <= m_pc -1 ; i++){
      pointsInfo.append("Point "+QString::number(i+1)+
                        " 'X' = "+QString::number(m_points[i].x()) + "\n") ;
      pointsInfo.append("Point "+QString::number(i+1)+
                        " 'Y' = "+QString::number(m_points[i].y()) + "\n") ;;
   }

return pointsInfo;
}
```

تحقيق خصائص لون القلم و لون الفرشاة للمضلع :

```
QColor Polygon::pColor() const {
    return m_pColor ;
```

```
}

QColor Polygon::bColor() const {
    return m_bColor ;
}

void Polygon::setPColor(QColor color){
   m_pColor = color ;
   emit pColorChanged() ;
}

void Polygon::setBColor(QColor color){
   m_bColor = color ;
   emit bColorChanged() ;
}
```

الأن لنجمع مشروع المكتبة من خلال الضغط على الإختصار "Ctrl+B" , اذهب إلى مجلد "الدليل" الذي قد تمّ تجميع المكتبة داخله , و انسخ الملفين "polygonPlugind.dll" – "libpolygonPlugind.a" إلى مجلد أنشأه انت وسمّه "lib" ثمّ افتح الملف "qmldir" و أضف إليه في نهاية سطره البرمجي النص "lib" :

```
plugin polygonPlugin lib
```

انتهينا من بناء مكتبة تحوي عنصر "Polygon" , لنرى كيفيّة استخدام هذا العنصر ضمن مشروع "QML" , أنشأ ملف "QML" يدعى "mqin.qml" و آخر يدعى "Button.qml" , اذهب إلى ملف "Button" و الذي يمثل عنصر زر, ثمّ اكتب التالي :

```
import QtQuick 1.0
```

```
Rectangle {
   id:btn
   height: 40;width: 200
   radius: 10
   signal clicked
   property alias text: label.text

   color: "silver"
   MouseArea{
   anchors.fill: btn
   hoverEnabled: true
   onHoveredChanged: btn.color = "gray"
   onExited: btn.color = "silver"
   onPressed: btn.color = "#666464"
   onClicked: {
      btn.clicked()
   }
   }

   Behavior on color{
   ColorAnimation { duration: 300 }
   }
```

```
    Text {
        id: label
        anchors.centerIn: btn
    }
}
```

بعدها اذهب إلى الملف "main.qml" و اكتب التالي :

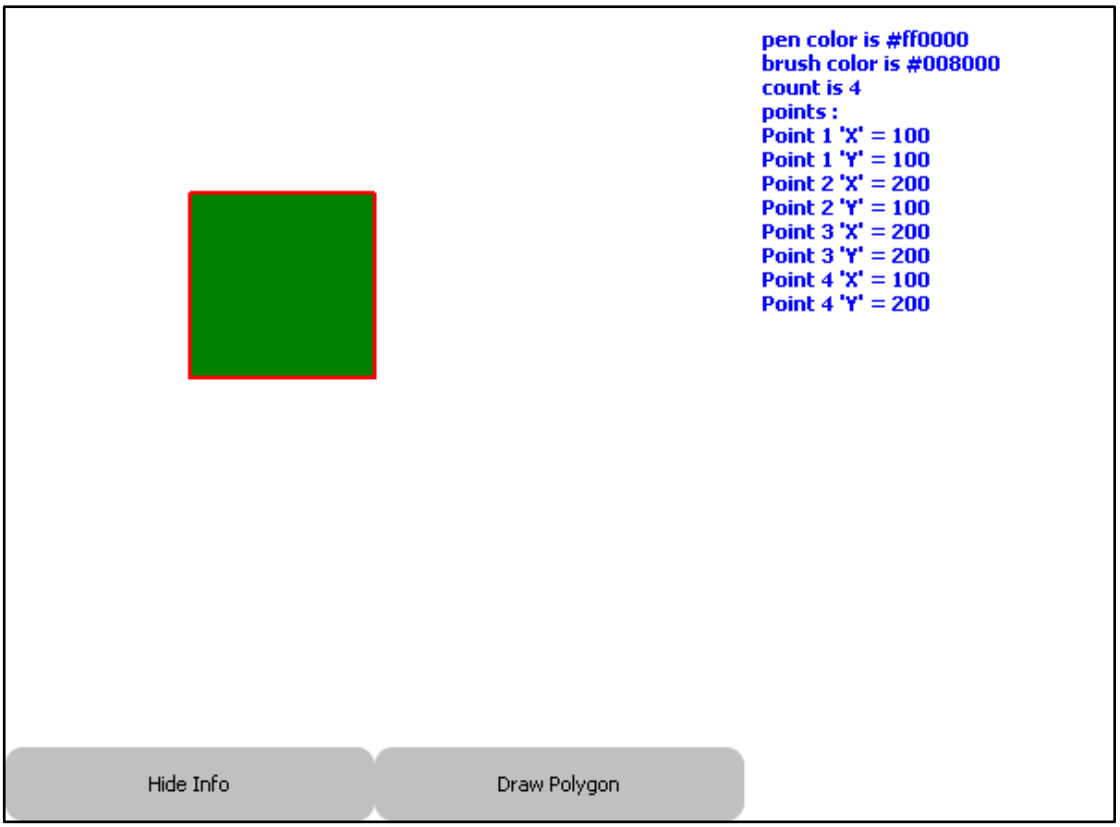
```
import QtQuick 1.0

Rectangle {
    id:win
    width: 600
    height: 440
    property string pnts: ""

    Button{id:btnpol
        text: "Draw Polygon"
        anchors.left: btnInfo.right
        anchors.top: pol.bottom
    onClicked: {
        pol.pColor = "red"
        pol.beginInsert()
    }
```

```
}
Button{id:btnInfo
   text: "Show Info"
   anchors.left: parent.left
   anchors.top: pol.bottom
onClicked: {
   (win.state == "info") ?
   (win.state = "Scene") & (btnInfo.text = "Show Info") :
   (win.state = "info") & (btnInfo.text = "Hide Info")  ;
}
}

Text {
   id: infoLbl
   anchors.top: win.top
   anchors.topMargin: 10
   anchors.left: pol.right
   anchors.leftMargin: 10
   font.bold: true
   color: "blue"
   opacity: 0.0
}
```

```
states: [
    State {
        name: "Scene"
        PropertyChanges {
            target: infoLbl
            opacity: 0.0
        }
    },
    State {
        name: "info"
        PropertyChanges {
            target: infoLbl
            opacity: 1.0
        }
    }
]
transitions: [
    Transition {
        NumberAnimation { properties: "opacity"; duration: 2000;
                                    easing.type: Easing.InOutQuad }
    }
]
```

```
    Polygon{
        id:pol
        width: 400;height: 400
        x:0;y:0
        bColor : "green"
        onEnd:{
            pnts = points ;
            infoLbl.text = qsTr("pen color is " + pColor + "\n" +
                                "brush color is " + bColor + "\n" +
                                "count is " + count + "\n" +
                                "points :\n"  + pnts) ;

        }
    }
}
```

الآن افتح البرنامج "QML Viewer" المرفق مع حزمة "Qt" , ثم خذ من قائمة ملف العنصر فتح و حدد الملف "main.qml" , سوف ترى الشكل (12.5) :



الشكل 12.5

جرب أن تضغط على الزر "Draw Polygon"و لاحظ كيف سيظهر لك نافذة إدخال لتحديد عدد نقاط المضلع , بعدها سوف يتليها نوافذ لتحديد مواقع النقاط , ثمّ سوف يرسم المضلع ضمن المشهد , أما عند الضغط على زر "Show Info" فسوف يظهر من الجهة اليمنى للنافذة معلومات حول المضلع لون حدّه , عدد نقاطه مواضعها الخ ...

سوف تجد في القرص المرفق مشروع المكتبة "polygonplugin" كاملا مع مشروع "QML Polygon"

برنامج "QML Viewer" سوف تجده في المسار التالي :

QtSDK/Desktop/Qt/4.7.4/mingw/bin

انتهينا من انشاء مكتبة عناصر "QML" و اختبارها ...

❖ خلاصة الفصل :

رأينا كيفية إنشاء مكون QML بوساطة Qt C++ و إضافة مناهج و خصائص غير مقيدة لهذا العنصر , و رأينا أيضا كيفيّة تسجيل هذا العنصر ليكون نمط جديد ضمن QML.

لننتقل إلى الفصل الأخير من هذا القسم و هذا الكتاب و الذي يدعى Qt Quick في التطبيقات المحمولة.

الفصل الرابع عشر

Qt Quick في التطبيقات المحمولة

❖ مقدمة Intro :

بعد أمن رأينا قوة و سحر تقنية Qt Quick يتوجب أن نرى الجانب الأجمل منها ألا و هو كيفية بنا تطبيقات محمولة احترافية من خلال هذه التقنية , حيث نجمع بين جمال واجهة المستخدم الرسومية و قوة Qt C++.

تحوي هذه التقنية الكثير من الوحدات النمطية الغنية الاستخدام من قراءة وتسجيل الوسائط المتعددة و عرض الويب و قراءة GPS الخ...

دعنا نرى على ما تحوي ...

❖ عارض صفحات الويب :

من أجل بناء متصفح ويب في "Qt Quick" بتوجب استخدام الوحدة النمطية " QtWebKit 1.0" , يوجد داخلها عنصر يدعى "WebView" مهمته عرض صفحات الويب من خلال تمرير عنوان الصفحة المرادة للخاصيّة "url" : انظر الرمّاز :

```
import QtQuick 1.1

import QtWebKit 1.0


Rectangle {

   id: window

   width: 640; height: 480


   WebView{

   id:webView
```

```
    width: 640; height: 480
    url: "http://www.google.com"
    }
}
```

❖ الحزمة QtMobility في Qt Quick :

عملت شركة "Nokia" بعد أخذها لـ "Qt" من شركة "trollTech" على تطويرها بشكل كلي , فبنت لغة مرفقة لـ "Qt" تدعى "QML" لتضاهي لغة الـ "Carbon" من شركة "Apple" و أنشات تقنية "Qt Quick" من أجل تفاعل لغة توصيف المستخدم "QML" مع لغة "Qt C++" , وطورت العديد من مكتبات "Qt" القابلة*للحمل* و أهم هذه المكتبات هي مكتبات الحزمة "QtMobility" , حيث تحوي على مجموعة من المكتبات التي تعمل على معظم أنظمة التشغيل و التي ماتزال شركة "Nokia" تعمل على تطويرها , نستطيع استخدام هذه الحزمة ضمن بيئة "Qt C++" و أيضا ضمن لغة "QML" باستخدام تقنية " Qt Quick", هذه الحزمة تحوي واجهات لـ العمل مع الوسائط المتعددة "multimedia" و الإتصالات "contacts" و حساس اللمس "sensor of touch" و البوصلة " sensor of compass" , و تحديد الموقع بالاستعانة بـ درجتي الطول و العرض "location" , و قراءة معلومات الجهاز "System Device Info" الخ ..

- **إضافات "Plugin" حزمة "QtMobility" لـ "QML" :**

Contacts	تحوي على مجموعة من توابع API خاصة بالتعامل مع الإتصالات بجميع تفاصيلها ,مثال بحث عن اسم موجود ضمن قائمة اتصالات الجوال و إنشاء إتصال بهذا الأسم الخ ..
Feedback	تحوي على مجموعة من توابع API خاصة بالتعامل مع التغذية العكسية اللمسية و الصوتية لجهاز الجوال , كـ استقبال إشارة تدل على أن محرك الإهتزاز قد بدأ اهتزازه
Gallery	تحوي على مجموعة من توابع API خاصة

بالتعامل معرض مستندات , كـ إضافة مستندات داخل المعرض و بحث عن مستند داخله الخ ..	
تحوي على مجموعة من توابع API خاصة بالتعامل موقع الجغرافي لمنطقة , كـ تحديد موقعنا الحالي من خلال قرءاة درجات الطوال و العرض لموقعنا الحالي و إعطائها لـ العنصر "location" من أجل تحديد الشارع الموجودين فيه حاليا , هذه الميزة مستخدم في أغلب المركبات الحديثة	Location
تحوي على مجموعة من توابع API خاصة بالتعامل الوسائط المتعدد , كـ تسجيل صوت و عرض فيديو و التقاط صورة من كاميرا الخ ..	Multimedia
تحوي على مجموعة من توابع API خاصة بالتعامل المهام و جداول المواعيد	Organizer
تحوي على مجموعة من توابع API خاصة بالتعامل مع النشر و المشاركة , كـ نشر قيمة ضمن التطبيق باسم مستعار , و الوصول إليها من تطبيق آخر باستخدام اسمها المستعار	Publish and Subscribe
تحوي على مجموعة من توابع API خاصة بالتعامل مع الخدمات , الوصول إلى الخدمات نظام التشغيل و عرض مناهجها و تحقيق المتاح منها	Service Framework
تحوي على مجموعة من توابع API خاصة بالتعامل مع جميع أنواع الرسائل (SMS – MMS - EMAIL) الخ .. , كقراءة رسالة و إرسال رسالة ..	Messaging
تحوي على مجموعة من توابع API خاصة بالوصول لـ جميع مواصفات الجهاز من النظام المحمول عليه , كـ جلب قيمة مستوى	System Information

شحن بطارية الجهاز	
تحوي على مجموعة من توابع API خاصة بالوصول لـ حسّاسات الجهاز , كـ قراءة مقدار زاوية سمت الحالي من حساس البوصلة	Sensors
تحوي على مجموعة من توابع API خاصة باالتعامل مع الشبكات اللاسلكيّة (WiFi – Bluetooth)	Connectivity

- تتكلم الفقرات القادمة حول :

1 - الوحدة النمطيّة QtMobility.systeminfo 1.2 : الحصول على معلومات الجهاز
2 - الوحدة النمطيّة QtMobility.publishsubscribe 1.2 : نشر قيمة في تطبيق و الوصول إليها من تطبيق آخر
3 - الوحدة النمطيّة QtMobility.location 1.2 : قراءة الموقع الجغرافي الحالي و وضع علامة عنده ضمن خريطة العالم
4 - الوحدة النمطيّة QtMultimediaKit 1.1 : الإستماع إلى ملف صوتي و مشاهدة ملف فيديو , و التحكم بموقع التشغيل الحالي لمقطع الفيديو و بشدة الصوت

❖ استخدام الحزمة QtMobility :

ليتاح استخدام مكتبات الحزمة "QtMobility" ضمن تطبيقنا الحالي يتوجب فعل الآتي :

بعد إنشاء المشروع الخاص ببناء تطبيق محمول نذهب إلى ملف المشروع , ثمّ نزيل محارف التعليق عن السطرين التاليين :

```
CONFIG += mobility

MOBILITY +=
```

نضيف لـ المتحول "MOBILITY" مكتبات الحزمة "QtMobility" التي نود استخدامها ضمن تطبيقنا الحالي , كـ مكتبة الوسائط المتعددة "multimedia" , انظر الرمّاز :

CONFIG += mobility

MOBILITY += multimedia

❖ استخدام الحزمة QtMobility في Symbian :

عند استخدام أحد مكاتب الحزمة "QtMobility" في تطبيق نود نشره على نظام Symbian , يتوجب علينا أن نفعّل قابية الوصول لـ تلك الخدمات التي اتصلنا بها من أحد مكتبات الحزمة "QtMobility" , للتوضيح إذا أردنا أن نصل من تطبيقنا الحالي إلى شبكة الويب , و نريد أن يعمل التطبيق على منصة "Symbian" ,

يجب أن نكتب ضمن ملف المشروع السطر البرمجي التالي :

symbian:TARGET.CAPABILITY += NetworkServices

يستخدم من أجل السماح بالوصول إلى خدمة الشكبة ضمن منصة "Symbian" , و في حال أردنا الوصول لقراءة معلومات اتصال موجودة على الجوال , نضيف الجملة "ReadUserData" :

symbian:TARGET.CAPABILITY += NetworkServices ReadUserData

- انظر الجدول الذي يحوي الجمل الخاصة بتفعيل قابلية الوصول ضمن منصة "Symbian" :

سماحية الوصول إلى الخدمات الشبكية	NetworkServices
سماحية الوصول لقراءة بيانات المستخدم الخاصة	ReadUserData
سماحية الوصول للكتابة في بيانات المستخدم الخاصة	WriteUserData
سماحية الوصول لقراءة دخل من أحد الأجهزة الداخلية كـ جهاز Bluetooth	ReadDeviceData
سماحية الوصول لإعطاء خرج لأحد الأجهزة الداخلية كـ جهاز Bluetooth	WriteDeviceData

UserEnvironment	سماحية استخدام الخدمات التي تتحكم بالبيئة الفيزيائيّة للجهاز كـ الوصول إلى الكاميرا أو المايك
Location	سماحية الوصول إلى خدمات الموقع الجغرافي GPS
LocalServices	سماحية استخدام الخدمات التطبيقيّة التي تعمل على الجهاز

❖ الحصول على معلومات الجهاز الحامل للتطبيق :

في حال كنّا نريد الحصول على مواصفات الجهاز الحامل للتطبيق كـ اللغة الحالية أو مستوى شحن البطارية , مفتاح الجهاز و اسمه الخ .. فما يتوجب علينا سوى استخدام الوحدة النمطيّة :

QtMobility.systeminfo 1.2

تحوي هذه الوحدة على مجموعة من العناصر الخاصة بقراءة معلومات عن الجهاز, أهم هذه العناصر :

DeviceInfo	قراءة معلومات الجهاز العامة كـ موديل الجهاز و معرفة إذا كان محرك الهزاز قيد التفعيل , و أيضا استقبال إشعارات الجهاز كـ الإشعار الذي يقدح عندما تفعيل خدمة الـ Bluetooth
BatteryInfo	قراءة معلومات البطاريّة كاملة , و استقبال جميع الإشعارات التي تصدر منها
DisplayInfo	قراءة معلومات جهاز العرض
NetworkInfo	قراءة معلومات شبكة الإتصال
GeneralInfo	قراءة المعلومات الخاصة باللغات
StorageInfo	قراءة معلومات وسائط التخزين

- العنصر **DeviceInfo** :

معرفة حالة جهاز الـ "Bluetooth" , أولا أنشأ مشروع "Qt Quick Application" وسمّه "bluetoothState" , و اضف كمنصة للتطبيق برنامج المحاكاة "Simulator" ,ثمّ بعد انشاء المشروع اذهب إلى ملف المشروع "bluetoothState.pro" و أزل التعليق عن السطرين البرمجيين الخاصي بتفعيل سماحية استخدام مكتبات الحزمة "QtMobility" :

```
CONFIG += mobility

MOBILITY += systeminfo
```

و أضف الجملة البرمجية "systeminfo" لـ المتحول "MOBILITY" من أجل السماح باستخدام المكتبة الخاصة بالحصول على معلومات الجهاز , لننتقل إلى تفعيل قابلية الوصول لـ منصة "Symbian" :

```
symbian:TARGET.CAPABILITY += NetworkServices LocalServices
UserEnvironment  ReadDeviceData
```

بما أننا نريد الوصول إلى جهاز الـ "Bluetooth" من أجل القراءة أضفنا "ReadDeviceData" , الآن اذهب إلى الملف "main.qml" و ضمّن الوحدة النمطية التالية :

```
import QtMobility.systeminfo 1.2
```

و أضف الرمّاز التالي داخل كتلة العنصر "Rectangle"الرئيسية :

```
 DeviceInfo{

id:deviceInfo

monitorBluetoothStateChanges : true

onBluetoothStateChanged: txtBluetoothState.text =

deviceInfo.currentBluetoothPowerState ? "Bluetooth State is on" :

 "Bluetooth State is off"

}
```

عند إطفاء أو تشغيل جهاز الـ "Bluetooth" سوف يرسل إشعار بذلك , نستقبله بوساطة الحدث "currentBluetoothStateChanged" , بدأت عملية المراقبة لحالة جهاز الـ "Bluetooth" من خلال إعطاء القيمة "true" لـ الخاصيّة "monitorBluetoothChanged", نفذ التطبيق على برنامج المحاكاة "Simulator" و ختبر النتيجة من خلال تغيير نمط التفعيل الحالي لـ "Bluetooth" الموجودة ضمن التبويب "Network" .

مراقبة مستوى الشحن للبطارية , أنشأ مشروع "Qt Quick Application" يدعى "batteryState" , ثمّ اذهب إلى ملف المشروع "batteryState.pro" و أزل التعليق الخاص بـ إتاحة استخدام الحزمة "QtMobility" , و أضف "systeminfo" لـ المتحول "MOBILITY" , ثمّ فعّل القابليّات لمنصة "Symbian" الخاصة بقراءة دخل جهاز :

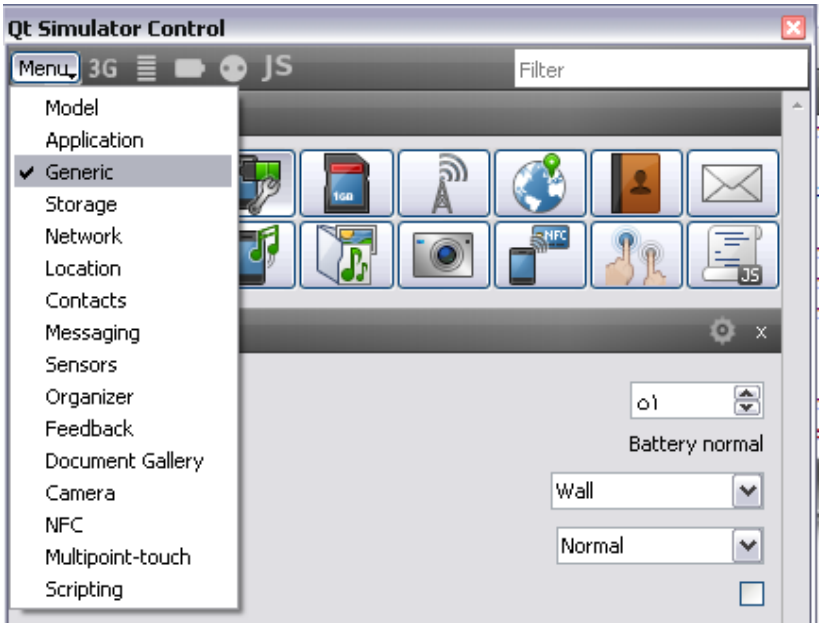
```
symbian:TARGET.CAPABILITY += ReadDeviceData
```

و ضمّن الوحدة النمطيّة "systeminfo 1.2", ثمّ ضمن ملف "main.qml" اكتب الرمّاز التالي داخل كتلة المسطيل الرئيسية :

```
DeviceInfo{
id:batteryInfo
monitorBatteryLevelChanges: true
monitorBatteryStatusChanges: true

onBatteryLevelChanged: txtBatteryLevel.text =
"Battery Level is " + batteryInfo.batteryLevel.toString()

onBatteryStatusChanged: {
  switch (batteryInfo.batteryStatus){
  case DeviceInfo.BatteryCritical : txtBatteryStatus.text =
  "Battery Status is Critical";break;
```

```
    case DeviceInfo.BatteryLow : txtBatteryStatus.text =
    "Battery Status is Low";break;
    case DeviceInfo.BatteryVeryLow : txtBatteryStatus.text =
    "Battery Status is Very Low";break;
    case DeviceInfo.BatteryNormal : txtBatteryStatus.text =
    "Battery Status is Normal";break;
    case DeviceInfo.BatteryPower : txtBatteryStatus.text =
    "Battery Status Power";break;
    }
  }
 }
```

أولا بدأ عملية مراقبة تغيير مستوى شحن البطارية و تغيير مستوى حالة البطارية و ذلك من خلال إعطاء القيمة "true" لـ كل من الخاصيّة "monitorBatteryLevelChanges" و الخاصيّة "monitorBatteryStatusChanges" , ضمن معالج الحدث "onBatteryLevelChanged" الذي يستدعى عند تغيير قيمة مستوى الشحن للبطارية , سيقرأ قيمة مستوى الشحن و يضعها ضمن لوحة النص , و ضمن معالج الحدث "onBatteryStatusChanged" الذي يستدعى عند تغيير حالة شحن البطارية , سيقرأ القيمة الحالية للبطارية و يكتبها ضمن لوحة النص , نفذ التطبيق على برناج المحاكي , ثمّ اذهب إلى التبويب "Generic" , في حال كان غير ظاهر تستطيع إظهاره من خلال تحديده من زر القائمة "menu" انظر الشكل (13.1) , بعدها غير مستوى الشحن للبطارية "Battery Level" و حالة البطارية "Battery State" , و اختبر النتيجة .



الشكل 13.1

عرض معلومات حول جهاز عرض الجهاز , وذلك بوساطة استخدام العنصر "DisplayInfo" , أولا أنشأ مشروع "Qt Quick Application" يدعى "DisplayInfo" , أضف الرمّاز الخاص بتفعيل استخدام المكتبة "systeminfo" من الحزمة "QtMobility" ضمن ملف المشروع "displayInfo.pro" و فعّل "ReadDeviceData" لـ منصة الـ "Symbian" , ثمّ اذهب إلى الملف "main.qml" , و ضمّن الوحدة النمطيّة "systeminfo 1.2", ثمّ اكتب الرمّاز التالي ضمن عنصر المستطيل الرئيسي :

```
DisplayInfo{ id:displayInfo }


Component.onCompleted: {

    txtDisplayInfo.text = "color depth is " + displayInfo.colorDepth +
"\n" +

          "physical height is " + displayInfo.physicalHeight + "\n" +

          "physical width is " + displayInfo.physicalWidth + "\n" +

          "contrast of the screen " + displayInfo.contrast + "\n" +
```

```
        "dots per inch for height " + displayInfo.dpiHeight + "\n" +
        "dots per inch for width " + displayInfo.dpiWidth + "\n" +
        "brightness of the screen " + displayInfo.displayBrightness +
"\n"
  }
```

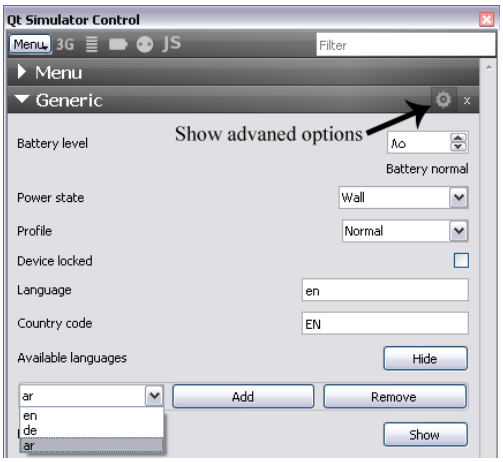
معلومات جهاز العرض , الإرتفاع و العرض , و التباين "contrast" , عدد النقاط بالبوصة , و السطوع .

لعرض معلومات عن الغة الحالية للجهاز و اللغات التي يدعمها , نستخدم العنصر "GeneralInfo" , أولا أنشأ مشروع "Qt Quick Application" يدعى "LanguageInfo" , أضف الرمّاز الخاص بتفعيل استخدام المكتبة "systeminfo" من الحزمة "QtMobility" ضمن ملف المشروع "LanguageInfo.pro" و فعّل "ReadDeviceData" لـ منصة الـ "Symbian" , ثمّ اذهب إلى الملف "main.qml" , و ضمّن الوحدة النمطيّة "systeminfo 1.2", ثمّ اكتب الرمّاز التالي ضمن عنصر المستطيل الرئيسي :

```
  GeneralInfo{ id: langInfo
    onCurrentLanguageChanged: txtlangInfo.text =
    "current language is " + currentLanguage.toString()
  }

  Component.onCompleted: {
    langInfo.startCurrentLanguageChanged()
    txtlangInfo.text = "current language is " +
langInfo.currentLanguage.toString()
    for(var i=0; i <= langInfo.availableLanguages.length -1 ; i ++)
      txtlanglist.text = txtlanglist.text + langInfo.availableLanguages[i]
+ "\n"
```

}

أولا أضفنا عنصر "GeneralInfo" باسم "langInfo" , ثمّ أضفنا داخله معالج الحدث "currentLanguageChanged" , و الذي مهمته كتابة اللغة الحالية ضمن لوحة النص في حال تغييرها , راقبنا تغيير اللغة الحالية من خلال استدعاء المنهج "startCurrentLanguageChanged()" , أضفنا اللغات الموجودة ضمن النظام الحامل للتطبيق بوساطة استداعء المنهج "availabelLanguages" والذي يرجع قائمة من النمط "Variable" تحوي جميع اللفات المتاحة ضمن النظام , عند تنفيذ التطبيق بوساطة برنامج المحاكي , اذهب إلى التبويب "Generic" و أظهر الخيارات المتقدمة لهذا التبويب من خلال الضغط على الزر "show advaned options" انظر الشكل (13.2) , ثمّ غير اللغة الحالية و انظر ضمن التطبيق كيف ستتغير اسم اللغة الموجودة ضمن النص الخاص بعرض اللغة الحالية .



الشكل 13.2

لعرض معلومات عن أجهزة التخزين الموجودة على الجهاز, نستخدم العنصر "StorageInfo" , أولا أنشأ مشروع "Qt Quick Application" يدعى "StorageInfo" , أضف الرمّاز الخاص بتفعيل استخدام المكتبة "systeminfo" من الحزمة "QtMobility" ضمن ملف المشروع "storageInfo.pro" و فعّل "ReadDeviceData" لـ منصة الـ "Symbian" , ثمّ اذهب إلى الملف "main.qml" , و ضمّن الوحدة النمطيّة "systeminfo 1.2", ثمّ اكتب الرمّاز التالي ضمن عنصر المستطيل الرئيسي :

```
StorageInfo{ id:storageInfo

    Component.onCompleted: {
```

```
        for(var i=0; i <= storageInfo.logicalDrives.length -1; i++ ){
            txtInfo.text = txtInfo.text + "drive name " +
            storageInfo.logicalDrives[i] +"\n" +
                "total disk space " +
            storageInfo.totalDiskSpace(storageInfo.logicalDrives[i]) + "\n"
+
                 "available disk space " +
            storageInfo.availableDiskSpace(storageInfo.logicalDrives[i]) +
"\n\n"
        }
        }
    }
```

المنهج "logicalDrivers" يعيد قائمة أسماء أجهزة التخزين الموجودة على الجهاز الحامل للتطبيق , أما المنهج "totlaDiskSpace(driveName)" يعيد الحجم الإجمالي لجهاز التخزين , الوسيط "driveName" هو اسم جهاز التخزين الذي نود معرفة حجمه الإجمالي , المنهج "availableDiskSpace(driveName)" يعيد حجم التخزين المتاح , عند تنفيذ التطبيق بوساطة برنامج المحاكي , اذهب إلى التبويب "Storage" لتجد قائمة التحكم الكامل بأجهزة التخزين .

لعرض معلومات عن أجهزة شبكات الإتصال الموجودة على الجهاز, نستخدم العنصر "NetworkInfo" , أولا أنشأ مشروع "Qt Quick Application" يدعى "networkInfo" , أضف الرمّاز الخاص بتفعيل استخدام المكتبة "systeminfo" من الحزمة "QtMobility" ضمن ملف المشروع "networkInfo.pro" و فعّل "ReadDeviceData" لـ منصة الـ "Symbian" , ثمّ اذهب إلى الملف "main.qml" , و ضمّن الوحدة النمطيّة "systeminfo 1.2", ثمّ اكتب الرمّاز التالي ضمن عنصر المستطيل الرئيسي :

```
NetworkInfo{id: network
    monitorNameChanges: true
```

```
    monitorSignalStrengthChanges: true
    monitorModeChanges: true
    onNetworkModeChanged: {
      switch(network.mode){
      case NetworkInfo.UnknownMode :txtNM.text = "Unknow
Mode";break;
      case NetworkInfo.GsmMode :txtNM.text = "Gsm Mode";break;
      case NetworkInfo.CdmaMode :txtNM.text = "Cdma
Mode";break;
      case NetworkInfo.WlanMode :txtNM.text = "Wlan Mode";break;
      case NetworkInfo.EthernetMode :txtNM.text = "Ethernet
Mode";break;
      case NetworkInfo.BluetoothMode :txtNM.text = "Bluetooth
Mode";break;
      case NetworkInfo.WimaxMode :txtNM.text = "Wimax
Mode";break;
      case NetworkInfo.LteMode :txtNM.text = "Lte Mode";break;
      }
    }
  }

  Text {id:txtNetwirkInfo
    text: "Network Name "+ network.networkName + "\n" +
      "Signal Strength "+ network.networkSignalStrength
    anchors.centerIn: parent;
```

```
}
Text {id: txtNM
anchors.top: txtNetwirkInfo.bottom
text: network.currentMode
}
```

لمعرفة اسم الشكبة استدعينا المنهج "networkName" , و لمعرفة قوة الإشارة استخدمنا المنهج "networkSignalStrength" , أما المنهج "mode" هو نمط الشبكة الحالي "bluetooth" أو "Wireless" شبكة لا سلكية الخ .. , ضمن برنامج المحاكي ستجد التبويب "Network" خاص بـ التحكم الكامل بـ شكبة الإتصال .

لعرض حالة وضع الجوال كـ صامت أو مطفأ "offline" , نستخدم العنصر "DeviceInfo" , أولا أنشأ مشروع "Qt Quick Application" يدعى "profileStatus" , أضف الرمّاز الخاص بتفعيل استخدام المكتبة "systeminfo" من الحزمة "QtMobility" ضمن ملف المشروع "profileStatus.pro", ثمّ اذهب إلى الملف "main.qml" , و ضمّن الوحدة النمطيّة "systeminfo 1.2", ثمّ اكتب الرمّاز التالي ضمن عنصر المستطيل الرئيسي :

```
DeviceInfo{id:deviceInfo
monitorCurrentProfileChanges: true
onCurrentProfileChanged: txtProfile.text = profile(currentProfile)
}

Component.onCompleted: txtProfile.text =
profile(deviceInfo.currentProfile)

function profile(cProfile){
switch(cProfile){
case DeviceInfo.UnknownProfile:return "Unknown Profile";break;
case DeviceInfo.SilentProfile:return "Silent Profile";break;
```

```
        case DeviceInfo.NormalProfile:return "Normal Profile";break;
        case DeviceInfo.LoudProfile:return "Loud Profile";break;
        case DeviceInfo.VibProfile:return "Vibrate Profile";break;
        case DeviceInfo.OfflineProfile:return "Offline Profile";break;
        case DeviceInfo.PowersaveProfile:return "Powersave
Profile";break;
        case DeviceInfo.CustomProfile:return "Custom Profile";break;
        }
    }
```

أولا أضفنا العنصر "DeviceInfo" و الذي يحوي على المناهج و الخصائص الخاصة بـ مراقبة التغيرات التي تطرأ على وضع الجهاز "تشكيلاته الجانبيّة" , أعطينا للخاصيّة "monitorCurrentProlfileChanged" القيمة "true" وذلك لمراقبة حالة الوضع , المنهج "currentProfile" يعيد الوضع الحالي للجهاز , ستجد التحكم بوضع الجهاز ضمن برنامج المحاكي داخل التبويب "Generic" , نفذ التطبيق و لاحظ عند تغيير وضع الجهاز "التشكيلات الجانبية" ما قيمة النص الظاهر ضمن لوحة النص الخاصة بعرض الحالة .

❖ النشر و المشاركة :

في حال أردنا أن نضع بعض القيم المتغيرة ضمن تطبيقنا زمن التنفيذ كـ قيم عامة يستطيع أي تطبيق الوصول لها من دون أن نكون قد خزناها ضمن أي وسيط تخزين "ملف – قاعدة بيانات" , فنستطيع ذلك من خلال استخدام مكتبة "publishsubscribe" الخاصة بتوزيع "نشر" قيمة زمن التنفيذ للتطبيق و الوصول إليها من تطبيق آخر , النشر و المشاركة يعمل في كل منصة بطريقة مختلفة ضمنيا , فـ على نظام "Win" يستخدم مسجل النظام لنشر المعطيات, أما على نظام "Unix" يستخدم تشاركية الذاكرة , و في نظام "Symbina" يستخدم أوضاع "Symbian".

• اهم عناصر الوحدة النمطية "publishsubscribe" :

العنصر "ValueSpacePublisher" يستخدم من أجل نشر قيمة , لأي قيمة يوجد مسار "path" لنستطيع الوصول إليها من خلاله .

العنصر "ValueSpaceSubscriber " يستخدم من أجل مشاركة القيمة "الوصول إلى القيمة" من خلال مسارها .

لنبني مشروع ينشر قيمة و هي مستوى شحن البطارية , أولا أنشأ مشروع "Qt Quick Application" يدعى "publish_and_subscribe" , أضف الرمّاز الخاص بتفعيل استخدام المكتبة "systeminfo" و المكتبة "publishsubscribe" من الحزمة "QtMobility" ضمن ملف المشروع "publish_and_subscribe.pro", ثمّ اذهب إلى الملف "main.qml", ضمّن الوحدات النمطية التالية :

```
import QtMobility.systeminfo 1.2

import QtMobility.publishsubscribe 1.2
```

و اكتب الرمّاز التالي ضمن عنصر المستطيل الرئيسي :

```
  ValueSpacePublisher{

    path: "Device/battery/level"

    value: deviceInfo.batteryLevel

  }


  DeviceInfo{id:deviceInfo

  monitorBatteryLevelChanges: true }


  ValueSpaceSubscriber{id:valueSubscriber

    path: "Device/battery/level"

  }

  Text {
```

```
    text: "Battery Level is " + valueSubscriber.value

    anchors.centerIn: parent

}
```

أولا استخدمنا العنصر "ValueSpcePublisher" من أجل بث قيمة مستوى شحن البطارية , مسار فضاء القيمة هو "Device/battery/level" وضعناه لـ الخاصيّة "path" أما الخاصيّة "value" فهي لنشر القيمة داخل فضاء القيمة المخصص , حصلنا على مستوى شحن البطارية من العنصر "DeviceInfo", استخدمنا العنصر "ValueSpaceSubscriber" من أجل الوصول إلى فضاء القيمة المشاركة من خلال وضع مسارها , الخاصيّة "value" لـ العنصر "ValueSpaceSubscriber" يرجع القيمة التتي قد تمّ نشرها ضمن المسار آنف الذكر , نستطيع الوصول إلى قيمة التي قد تمّ نشرها من أي تطبيق آخر فقط من خلال استخدام المسار المحدد لها "Device/battery/level" .

❖ خدمة تحديد الموقع Location :

تحوي حزمة "QtMobility" مكتبة خاصة بـ عرض خريطة الموقع الجغرافي الذي نود من خلال تحديد احداثيا الطول و العرض لهذا الموقع , هذه المكتبة هي "location" , نستطيع بوساطة هذه المكتبة أن نحصل على احداثيا الطول و العرض للموقع و الحالي و معلومات عنه كالوقت , و ايضا عرضه ضمن خريطة تضريسية أو خريطة يوضح الطرقات "streets" فيها , او خريطة من مشهد القمر الصناعي , أهم عناصر هذه المكتبة هي :

1 - العنصر "Map" : مهمته عرض مشهد خريطة
2 - العنصر "MapCircle" : مهمته عرض دائرة ضمن الخريطة من خلال إعطاءها إحداثيا الطول و العرض و نصف قطرها مقاسا بالمتر
3 - العنصر "PositionSource" : مهمته الحصول على معلومات الموقع الحالي من درجتي الطول و العرض و الوقت الحالي للمنطقة الموجودين فيها الخ ..

لنوضح كيفيّة استخدام أهم عناصر الوحدة النمطية "location" من خلال بناء مثال يدعى أين أكون "where am i" , سنستخدم خدمة الخرائط المقدمة من شركة "nokia" , لنبدأ :

أولا أنشأ مشروع "Qt Quick Application" يدعى "where_am_i" ثمّ اذهب إلى ملف المشروع "where_am_i.pro" و أزل التعليق عن السطرين البرمجين الخاصين بـ تفعيل استخدام الحزمة "Mobility" و أضف لـ المتحول "MOBILITY" المفتاح "location" من أجل تفعيل استخدام المكتبة "location" :

```
CONFIG += mobility

 MOBILITY += location
```

ثمّ ضمن قابليات منصة "Symbian" فعّل استخدام الخدمة "Location" كالآتي :

```
symbian:TARGET.CAPABILITY += NetworkServices Location
```

الآن اذهب إلى ملف "main.qml" و ضمّن الوحدة النمطية "location 1.2" :

```
import QtMobility.location 1.2
```

احذف كل ما أضافه المعالج الذكي داخل عنصر المستطيل الرئيسي ليصبح الرمّاز :

```
Rectangle {
   width: 360
   height: 360
   focus: true
}
```

أضف ضمن كتلة عنصر المستطيل العنصر "PositionSource" الخاص بـ جلب معلومات حول الموقع الحالي :

```
   PositionSource{
   id:currenPosition
   updateInterval: 1000
   active: true
   onPositionChanged: txtInfo.text = position.timestamp
   }
```

الخاصيّة "updateInterval" نحدد من خلالها زمن تحديث معلومات الموقع الحالي , هنا حددنا كل 1 ثانية يتم تحديث معلومات الموقع الحالي كـ احداثيا الطول و العرض و الوقت الحالي للمنطقة , أما معالج الحدث "positionChanged" فـ يستدعى بشكل تلقائي عند تغيير احداثيات الموقع الحالي , هنا سيعرض ضمن عنصر النص "txtInfo" الوقت الحالي

للمنطقة الموجودين فيها , وذلك من خلال استدعاء المنهج "timestamp" , الآن أضف العنصر "Map" الخاص بـ عرض الخريطة الجغرافية :

```
Map{id:map

size.width: parent.width;size.height: parent.height

zoomLevel: 10
```

حددنا طول و عرض الخريطة هو نفسه حجم عنصر الأب أي عنصر المستطيل الرئيسي , ثمّ حددنا مستوى التقريب هو 10 , القيمة الإفتراضية له 8 , الآن أضف الرمّاز الخاص بـ تحديد مزود خدمة الخرائط :

```
plugin: Plugin{name:"nokia"}
```

هنا حددنا مزود شركة "nokia" الخاصة بالخرائط الجغرافية , يوجد مخزن لشركة "nokia" يدعى "ovi" خاص بحفظ خرائط العالم ككل , فـ أهم مهام خدمة الخرائط "nokia" هي عرض فقط الجزء المراد من خريطة العالم و ذلك من خلال إعطائها درجتي الطول و العرض , أضف الرمّاز الخاص بـ تحديد الموقع الجغرافي الذي نود عرضه :

```
center: Coordinate{

latitude: currenPosition.position.coordinate.latitude

longitude: currenPosition.position.coordinate.longitude

}
```

ضمن الخاصيّة "center" حددنا إحداثيات درجتي الطول و العرض و ذلك بعد وضعهم ضمن العنصر "Coordinate" , حصلنا على درجتي الطول و العرض للموقع الموجودين فيه حاليا من العنصر "PositionSource" , *نستطيع أن نضع أي درجتي طول و عرض نريد لأي نقطة من العالم لعرضها ضمن مشهد الخريطة* , من أجل عرض دائة صغيرة تدل على موقعنا الحالي ضمن مشهد الخريطة نستخدم العنصر "MapCircle" :

```
MapCircle{

center: Coordinate{

   latitude: currenPosition.position.coordinate.latitude

   longitude: currenPosition.position.coordinate.longitude
```

```
}
SequentialAnimation on color{
ColorAnimation { from: "red"; to: "blue"; duration: 2000 }
ColorAnimation { from: "blue"; to: "red"; duration: 2000 }
loops: Animation.Infinite
}

border.color:"red"
border.width:1
radius: 200.0
}
```

الخاصيّة "radius" نمرر لها نصف قطر الدائرة مقاس بالمتر , بالتالي ستكون الدائرة الظاهرة بمشهد الخريطة تمثل مسافة 200 متر من الخريطة , من أجل تحديد نمط الخريطة أي خريطة تضريسية أو مشهد خريطة الموقع الجغرافي المحدد من قمر صناعي نضيف الرمّاز التالي :

```
mapType: Map.StreetMap
```

أنماط الخرائط المتاحة هي :

1- Map.StreetMap
2- Map.SatelliteMapDay
3- Map.SatelliteMapNight
4- Map.TerrainMap

بذلك نكون أنهينا عنصر الخريطة "Map" , لا تنسى إغلاق كتلته بوضع القوص الكبير "}" , من أجل تقريب مشهد الخريطة و تبعيدها نستخدم الخاصيّة "zoomLevel" التابعة بعنصر مشهد الخريطة "Map":

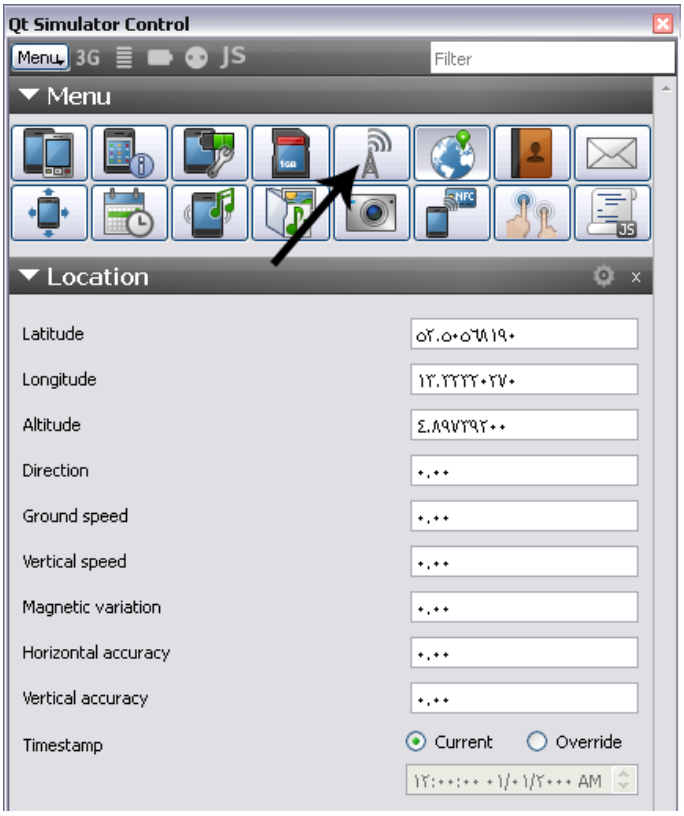
```
Keys.onLeftPressed: map.zoomLevel--
Keys.onRightPressed: map.zoomLevel++
```

أخير أضف عنصري النص التاليين :

```
Text {
   id: txtInfo
   anchors.top: parent.top
   anchors.horizontalCenter: parent.horizontalCenter
}
Text {
   anchors.bottom: parent.bottom
   anchors.horizontalCenter: parent.horizontalCenter
   text: qsTr("press left key to zoom out\npress right key to zoom in")
}
```

عنصر النص الأول خاص بعرض الوقت الحالي للمنطقة , أما الثاني فقط من اجل إعطاء تلميح عن كيفية تقريب و تبعيد مشهد الخريطة .

نفذ التطبيق و اختبر النتيجة , يجب أن تكون في حالة اتصال بالإنترنت من أجل الوصول إلى خدمة شركة نوكيا الخاصة بالخرائط , نستطيع من خلال برنامج المحاكاة أن نتحكم بدرجتي الطول و العرض و إرتفاع المنطقة عن سطح البحر الخ.. من خلال فتح التبويب "Location" , انظر الشكل (13.3) :



الشكل 13.3

❖ الوسائط المتعدد في Qt Quick :

المكتبة "QtMultmediaKit" تحوي على مجموعة من توابع "API" لتشغيل و تسجيل وسائط متعددة صوت – فيديو و التحكم بـ كيفية عملها , أيضا نستطيع كم خلالها أن نتفاعل مع جهاز الكاميرا , هذه المكتبة جزء من الحزمة "QtMobility" .

أهم عنصر الموحدة النمطيّة "QtMultimediakit 1.1" :

1- **Audio**
2- **Video**
3- **Camera**

لنستطيع التعامل مع هذه المكتبة يتوجب تفعيل الحزمة "QtMobility" من ملف المشروع للتطبيق , و أن نضيف المفتاح "multimedia" للمتحول "MOBILITY" .

- **عنصر مشغل الصوت "Audio" :**

تشغيل ملف صوتي و تغيير شدة الصوت و موقع زمن التشغيل الحالي , سندرجهم ضمن المشروع التالي :

أولا أنشأ مشروع "Qt Quick Application" و سمّه "soundPlayer" , ثمّ اذهب إلى ملف مشروعه و فعّل استخدام الحزمة "QtMobility" ثمّ أدرج المفتاح "multimedia" من اجل اتاحة استخدام المكتبة "QtMultimediaKit" ضمن المشروع :

```
CONFIG += mobility

MOBILITY += multimedia
```

بعدها اذهب إلى الملف "main.qml" و ضمّن الوحدة النمطيّة " QtMultimediaKit 1.1" :

```
import QtMultimediaKit 1.1
```

ضمن عنصر المستطيل الرئيسي أضف العنصر "Audio" من اجل تشغيل الملف الصوتي :

```
Audio{id:audio

   source: "melody.mp3"

   playing: true

   volume: 0.5

}
```

سيتم تشغيل الملف الصوتي "melody.mp3" بعد تحميله مباشرة , الخاصيّة "volume" تستخدم من أجل تحديد مستوى الصوت , من أجل تغيير موقع زمن التشغيل الحالي إلى الأمام و إلى الخلف باستخدام المفتاحين السهم الأيمن و السهم الأيسر :

```
Keys.onLeftPressed: audio.position -=2000

Keys.onRightPressed: audio.position +=2000
```

الخاصيّة "position" تستخدم من أجل تحديد و قراءة زمن التشغيل الحالي , سيزداد زمن التشغيل الحالي 2 ثانية عند كل ضغطة على مفتاح السهم الأيمن أما مفتاح السهم الأيسر ينقص زمن التشغيل اثنان من الثانية , لتغيير شدة مستوى الصوت نسنخدم الخاصيّة "volume" كالأتي :

```
Keys.onUpPressed: audio.volume += 0.1

Keys.onDownPressed: audio.volume -= 0.1
```

تتراوح قيمة الخاصيّة "volume" بين المجال 0.0 و 1.0 , من أجل عرض المدة الكلية للمقطع الصوتي و عرض موقع زمن التشغيل الحالي , أضف الرمّاز التالي :

```
Row{

Text {

   id: txtDuration

   text: qsTr("totlat time "+audio.duration.toString())

}

Text {

   id: txtCurrentTime

   text: qsTr(" / current time " +audio.position.toString())

}

}
```

الخاصيّة "duration" تعيد المدة الكليّة للمقط الصوتي , أما الخاصيّة "position" تعيد موقع زمن التشغيل الحالي , نفذ التطبيق و اختبر النتيجة . يوجد منهج للإيقاف المؤقت للمقط الصوتي و هو "pause" أما المنهج "stop" فهو للإيقاف الكلي , لتشغيل المقطع الصوتي نستخدم المنهج "play" .

رمّاز خاص بـ كتم الصوت و إعادته عند كل لمسة "نقرة" على الشاشة :

```
MouseArea{anchors.fill: parent; onClicked: audio.muted =
!audio.muted}
```

يوجد معالج حدث للعنصر "Audio" يدعى "posistionChanged" يرفع عند تغيير زمن التشغيل الحالي لمقطع الوسيط .

- **عنصر مشغل الفيديو Video :**

لتشغيل مقطع فيديو نستخدم العنصر "Video" و الذي مهمته عرض مقطع الفيديو و التحكم به , جميع الخصائص و المناهج الذي قد ذكرناها لعنصر "Audio" موجودة ضمن العنصر "Video" , يضاف عليها الخاصيّة "fillMode" نمط امتداد مشهد الفيديو , و الذي يأخذ وسيط تعداد :

1- Video.Stretch
2- Video.PreserveAspectFit
3- Video.PreserveAspectCrop

انظر الرمّاز التالي الخاص بـ تشغيل مقطع فيديو و عرض شريط تقدم لـ موقع المقطع الحالي :

```
import QtQuick 1.0

import QtMultimediaKit 1.1

import Qt.labs.components.native 1.0


Rectangle{

   width: 360;height: 360

Video{

id:videoP

width: 260;height: 260

source: "vidio.avi"

playing: true

fillMode: Video.Stretch

}


ProgressBar{id:progressBar
```

```
width: parent.width - txtInfoCT.width - txtInfoTT.width;height:
txtInfoCT.height

anchors.left: parent.left

anchors.top: videoP.bottom

maximumValue: videoP.duration

value: videoP.position

Text {

   id: txtInfoCT

   anchors.top: parent.bottom

   text: qsTr(" current time "+videoP.position.toString()+" ms ")

}

Text {

   id: txtInfoTT

   anchors.top: parent.bottom

   anchors.left: txtInfoCT.right

   text: qsTr("/ total time "+videoP.duration.toString()+" ms")

}

}

}
```

الرمّاز الغريب ضمن هذا المثال هو استخدامنا للوحدة النمطيّة "Qt.labs.components.native 1.0" و التي تحوي على مجموعة من العناصر المرئيّة كـ عنصر شريط الأدوات و عنصر قائمة و عنصر شريط تقدم الخ.. هنا استخدمناها من أجل عرض عنصر شريط تقدم "ProgressBar" أهم خصائصه :

1 - الخاصيّة "maximumValue" : لتحديد أكبر قيمة لشريط التقدم
2 - الخاصيّة "value" : لتحديد القمية الحالية التي يظهرها شريط التقدم

سنتكلم ضمن الملحق عن اغلب عناصر الوحدة النمطيّة " Qt.labs.components.native 1.0" .

- **عنصر استخدام جهاز الـ Camera :**

سوف لن نتوسع في شرح التحكم بجهاز الكاميرا , لكن سنغطي كيفيّة استخدام جهاز الكام و عرض مشهده ضمن تطبيقنا و كيفيّة التقاط صورة و عرضها ضمن عنصر صورة .

عنصر "Camera" موجود ضمن مكتبة "QtMultimediakit 1.1" التابعة للحزمة "QtMobility" , لذا عندما نود استخدام العنصر "Camera" يتوجب علينا تفعيل استخدام الحزمة "QtMobility" و إضافة المكتبة "multimedia" للمشروع :

```
CONFIG += mobility

MOBILITY += multimedia
```

أما بالنسبة لـ قابليات منصة "Symbian" يتوجب علينا ان نضيف المفتاح "UserEnvironment" من أجل أن يسمح لتطبيقنا بالوصول إلى جهاز الكام :

```
symbian:TARGET.CAPABILITY += UserEnvironment
```

رمّاز استخدام جهاز الكام و التقاط صورة منه بسيط للغاية , انظر الرمّاز التالي :

```
Camera {id:cam


    flashMode: Camera.FlashOn


    onImageCaptured : {
        photo.source = preview
        txtInfo.text = preview
    }


}
```

```
MouseArea{anchors.fill: parent;
   onClicked: cam.captureImage()}

Image { id: photo }

Text { id: txtInfo }
```

أضفنا عنصر كاميرا يدعى "cam" , و شغلنا ضوء "Flash" لجهاز الكام , ضمن معالج الحدث "imageCaptured" , وضعنا الصورة التي التقطها جهاز الكام ضمن عنصر الصورة "photo" ثمّ وضعنا مسار الصورة ضمن عنصر النص "txtInfo" , من اجل التقاط صورة ثابتة من جهاز الكام يتوجب علينا استدعاء المنهج "captureImage()" من العنصر "cam" , نفذ التطبيق و اختبر النتيجة .

❖ خلاصة الفصل :

رأينا الكثير من الوحدات الكثيرة الإستخدام في أغلب التطبيقات الموجهة إلى الأجهزة الذكية من حالة البطارية إلى الطلبات الشبكية و عرض الويب و الوسائط المتعددة و اخير قراءة الموضع الجغرافي الحالي لحامل التطبيق باستخدام تقنية GPS.

بفضل الله أنهينا تأليف هذا الكتاب والمسمى برمجة التطبيقات المحمولة بوساطة Qt, و نرجو أن يكون من المؤلفات المفيدة لجميع المبرمجين العرب.

نرجو من القراء العرب الإستفادة منه و استثماره بما حلل الله و لكم جزيل الثواب.

قد توكلت على الله و باشرت بتألف كتاب يتكلم حول برمجة التطبيقات الموجهة إلى نظام Android و أجهزة iPhone بوساطة Qt C++ framework.

للتواصل مع المؤلف المبرمج حسن نورالدين المرهج من أجل أي استفسار على الإيميل التالي:

hasanmorhej@gmail.com

السيرة الذاتية

حسن نورالدين مرهج

البريد الإلكتروني: hasanmorhej@gmail.com

❖ **البيانات الشخصية:**

- **تاريخ الميلاد:17 تموز - 1989**
- **الجنس: ذكر**
- **الحالة الاجتماعية: عازب**
- **الجنسية: سورية**
- **المهنة:مطور برمجيات**

❖ **الخبرة العملية:**

❖ **2009-2008**

مبرمج تطبيقات

- **مناهج تعليمية للجوال و أجهزة الجيب (Pocket PC) –Net-Java. شركة إماراتية (Smartik)**

مبرمج تطبيقات

- **نظام تعليم الإلكتروني عن بعد**
- **إمتحانات عبر النت**
- **برنامج الجمعية الوطنية للمتقاعدين (السعودية)**
- **برنامج مراقبة الدوام (برنامج سيرفر – برنامج عميل للحاسب الشخصي – برنامج عميل للجوال)**
- **برنامج لإرسال رسائل MMS**
- **شركة (first point)-(الإمارات العربية- دبي) برمجة تطبيقات للجوال و الحاسب- 2011**

مبرمج تطبيقات في شركة بلوتو اي تي بريطانيا 2014

المهارات:

- **اللغات التي أجيدها :**
  - **اللغة العربية (اللغة الأم)**
  - **اللغة الإنكليزية (جيد)**

- **الحاسب:**
  - **إحادة نظام ويندوز و برامج مايكروسوفت أوفيس**
  - **نظام لينكس (BT5 r3 - Ubuntu)**

- **برمجة التطبيقات**
- **لغات البرمجة التي أجيد:**
  - **C++(MFC-C++ Builder-C++.Net Qt , QML , Qt Quick) (متقدم)**
  - **Java(J2EE-J2SE-J2ME) for PC , Mobile and web Applications (متقدم)**
  - **Delphi(متقدم)**
  - **MS VC#.Net(متقدم)**
  - **MS VB.Net(متقدم)**
  - **Qt من أجل برمجة تطبيقات لأنظمة سطح المكتب (Win , Mac OS, Linux) و الجوال (Android , iPhone , iPad, Symbian^3,^2,^1, Meego Linux ,Win CE)**
  - **QML لبرمجة واجهات رسومية**

- Qt Quick
- جيد جدا في لغة التجميع (Assemble)
- Perl & Python & BASH جيد
- النمذجة و المحاكاة بوساطة Matlab

❖برمجة مواقع الوب

- لغات البرمجة التي أجيدها:
  - PHP (متقدم)
  - JSP (متقدم)
  - ASP.Net (متقدم)

❖برمجة قواعد البيانات

- لغات البرمجة التي أجيدها:
  - MS SQL Server 2005 (متقدم)
  - My SQL Server (متقدم)
  - Borland InterBase (متوسط)
  - PL SQL Oracle / (متوسط)

❖ فن القرصنة و الأمن:

- الهندسة العكسية و التصدي لها
- تشريح الفيروسات و الوقاية منها
- الإستغلال
- التعمية و فك شيفرتها
- اختراق مخدم الشبكة و حمايته
- بروتوكولات أمن الشبكة (SSL-TCL-HTTPS...)
- جدار النار بانواعه الثلاث الخ..

❖ الكترون و تحكم :

- معلومات جيدة بـ الإلكترون التماثلي
- جيد بـ الإلكترون الرقمي
- معالجات
- برمجة المتحكمات الصغرية (PIC-ATMEL AVR- Arduino)

❖ علوم تطبيقية:

- رياضيات تطبيقية (تحليل - احصاء -فراغية)
- ستاتيك و ديناميك (فيزياء)
- كهرباء(فيزياء)

❖ الذكاء الصنعي:

- التنقيب عن البيانات (Data Mining)
- الشبكات العصبونية (Neural Net)
- المنطق العائم (Fuzzy Logic) أو (Float Logic)

❖ التقنيات في لغات لبرمجة التي أتقنها:

- خدمات الوب في (j2ee-C#.net-VB.Net-Delphi)

- الشبكات و برمجة الإتصال عبر البروتوكولات المعرفة مثل (TCP/IP-HTTP-HTTPS-FTP-UDP…) في لغات البرمجة (Java-C#.Net-VB.Net-Delphi(Indy)-C++-Qt)
- النظام الموزعة (Snap-Broker-corba-IntraWeb-Jini)
- قواعد البيانات في جميع لغات البرمجة (Java-C#-VB6-VB.Net-Delphi-C++) عبر (ADO-ADO.Net-DATA-JDBC-JDO…..)
- البرمجة غرضية التوجه (OOP)
- XML
- البرمجة التفرعية
- برمجة الوسائط و معالجتها (الصوت و القيديو و الصورة)
- EJB in J2EE
- الحوسبة السحابية (Java-Windows Azure-.Net)- Google App engine
- C++ Directx9-11(Direct3D-DirectSound-DirectPlay-HLSL)
- C++,Qt OpenGL - OpenGL 2 - OpenGLShading Language
- OpenCL
- Java3D-Java2D
- Crystal Report – MVC-WPF
- Ajax-html-java script-css
- البرامج:
- Photoshop
- DreamWaver
- MS Expression
- Autodesk Maya
- Adobe Flash(Middle)-and Action Script 3
- Proteus 7 Professional لتصميم الدارات الإلكترونية
- والمزيد منها...

❖ محركات الألعاب:

- UDK - UnrealScript جيد
- CryEngine - C++ -CryScript جيد
- Unity 3D -C# جيد

معلومات أخرى:

- قمت بتطوير برنامج بنك الدم (سورية) منظمة (وورلد لينكس) عام 2007 و كان من الحاضرين على تسليم البرنامج السيدة الأولى (كنت في الثالث الثانوي - تقنيات حاسوب) و منحت عليه رسالة توصية و امتياز . 2007-2006

هدفي:

- تطوير محرك ألعاب

❖ الشهادات الحاصل عليها:

- شهادة خبرة(توصية) ل برنامج بنك الدم (سورية) منظمة (وورلد لينكس) 2007
- شهادة ICDL) 2007
- شهادة Photoshop 2007
- شهادة Visual Basic 2007
- شهادة System Development 2007
- شهادة تصميم الفيروسات و الوقاية منها بتقدير ممتاز 2007
- شهادة أمن المعلومات- Anti Hacking 2007

بالإضافة لحضور العديد من الندوات في دبي :

- شهادة حضور Qt Nokia في هيلتون دبي 2011
- شهادة حضور Qt , Qt Quick , QML من شركة Nokia في كراون بلازا دبي 2011

❖ الهوايات :

- العزف على آلة العود.

❖ بعض التطبيقات التي قمت ببرمجتها:

- برنامج بنك الدم للجمهورية العربية السورية(C#.net & MS SQL Server 2005)
- وظيفة البرنامج –مزامنة بيانات بنوك الدم خسب الحاجة و بشكل تلقائي-
- برنامج مراقب الدوام عبر الأجهزة الذكية –J2ME-
- برنامج محادثة (صوت و صورة) (غرف)-(Delphi 7)-
- برنامج الآذان للجوال (C++ Carbide)
- برمجة نظام للعيادات الطبية (الإمارات - دبي) (Win Azure-C#.Net)
- برمجة نظام للعيادات الطبية (الإمارات - دبي) (J2EE-google app engine)
- التعليم الإلكتروني : http://www.saudiedunet.com التعليم الإلكتروني عبر النت
- جدول مواعيد للمرضة(عيادة طبية)تقنية الحوسبة السحابية جوجل أب انجين
- برمجة لعبة حرب الفضاء (2D) (C#.Net)
- برمجة لعبة ساحة القتال للجوال (Qt Quick)
- لعبة حربية 3D عبر الشبكة (UDK)

والمزيد من البرامج و الألعاب....

❖ **من مؤلفاتي :**

- **كتاب برمجة التطبيقات المحمولة بوساطة Qt (666 صفحة - أربع أقسام (Qt C++ framework - QML - Qt Quick - في اللغة العربية و يتم الآن ترجمته إلى الإنكليزية .**

أخيرا أقوم بتصميم روبوت آلي ذكي يقوم باكتشاف الكائنات و نقلها إلى موضع محدد حسب الأوامر الصوتية بالإضافة ميزة التعلم الذاتي و تحديد الموقع بشكل تلقائي.

شكرا

السلام عليكم.

جدول المحتويات