



Lycée La Fayette

Champagne-sur-Seine • Fontaineroux

Département SN-IR



Qt 5.3

© Alain Menu - édition 4.0, octobre 2014

Objectifs : Qt état de l'art. Présentation, procédures d'installation et programmes de tests...

Qt 5.x Intro - v4.0 - © octobre 2014 - Alain MENU



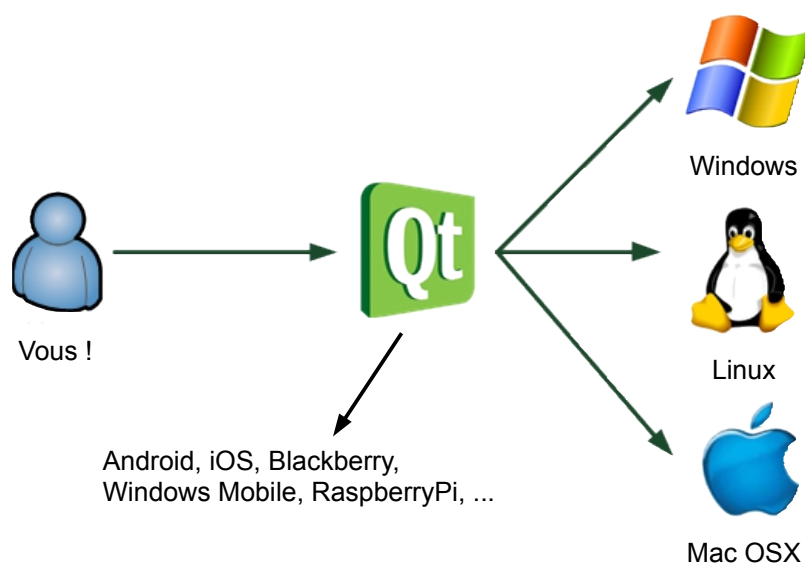
Cette création est mise à disposition selon le Contrat *Attribution-NonCommercial-ShareAlike 2.0 France* disponible en ligne <http://creativecommons.org/licenses/by-nc-sa/2.0/fr/> ou par courrier postal à Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.



Lycée La Fayette

Table des matières

Introduction.....	3
Rapide historique.....	3
Licences.....	5
Architecture.....	6
Structure générale.....	6
Outils de développement.....	6
Installation du SDK.....	10
Linux (Ubuntu).....	10
Mac OS X.....	10
Windows.....	10
Post-installation.....	11
Programmes de test de l'installation.....	12
Premier test : C++.....	12
Deuxième test : QML.....	13
Portabilité des applications.....	14
Au niveau du fichier projet.....	14
Au niveau du code.....	14
Déploiement d'une application.....	15
Linux.....	15
Mac OS X.....	15
Windows.....	15
Choix de l'interface graphique.....	16
L'application nécessite-t-elle réellement une interface graphique ?.....	16
Interfaces Web.....	16
Interfaces graphiques Desktop.....	16
Applications mobiles.....	16
Synthèse.....	17



Références

Site officiel :	http://qt-project.org
Wikipédia :	http://fr.wikipedia.org/wiki/Qt
Site dédié en français :	http://qt.developpez.com
Forum de la communauté francophone :	http://qtfr.org/
Complément de doc. sur qmake :	http://www.qtcentre.org/wiki/index.php?title=Undocumented_qmake
Livre (éditions D-BookeR) :	Créer des applications avec Qt 5 (écrit sous la direction de Jonathan Courtois)



Introduction

Qt (prononcez officiellement en anglais *cute* /kju:t/) est un framework orienté objet et développé en C++ par *Qt Development Frameworks*, filiale de Nokia. Il offre des composants d'interface graphique (widgets), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution, d'analyse XML, etc. Qt est par certains aspects un *framework* lorsqu'on l'utilise pour concevoir des interfaces graphiques ou que l'on architecture son application en utilisant les mécanismes des signaux et slots par exemple.

Qt permet la portabilité des applications qui n'utilisent que ses composants par simple recompilation du code source. Les environnements supportés sont les Unix (dont Linux et Mac OS X), Windows, ainsi que les systèmes nomades tels que Blackberry, Android ou iOS. Le fait d'être une bibliothèque logicielle multiplate-forme attire un grand nombre de personnes qui ont donc l'occasion de diffuser leurs programmes sur les principaux OS existants.

Qt est notamment connu pour être la bibliothèque sur laquelle repose l'environnement graphique KDE, l'un des environnements de bureau les plus utilisés dans le monde Linux.



Rapide historique



C'est au *Norwegian Institute of Technology* à Trondheim en Norvège, que Haavard Nord (CEO de Trolltech) et Eirik Chambe-Eng (président de Trolltech) se rencontrent.

En 1988, Haavard Nord est chargé par une entreprise suédoise de développer une bibliothèque logicielle en C++ pour gérer une interface graphique. Deux ans plus tard, avec Chambe-Eng, ils développent une application multiplate-forme (Unix, Macintosh et Windows) et commencent sérieusement à réfléchir à la conception d'une bibliothèque graphique multi-plateforme généraliste.



En 1991, ils entament le développement de cette bibliothèque. L'année suivante, Chambe-Eng propose le principe des « signaux et slots », qui devient la pierre angulaire de Qt. Et en 1993, le noyau de Qt est prêt et permet aux informaticiens de développer leurs propres composants graphiques. C'est à la fin de cette année que Haavard Nord propose de créer une entreprise pour commercialiser leur bibliothèque.

Quasar Technologies est créé le 4 mars 1994 et renommé six mois plus tard en *Troll Tech*, puis *Trolltech*, puis *Qt Software* et enfin *Qt Development Frameworks*.

TROLLTECH®

Les débuts sont particulièrement difficiles financièrement. Mais ils ont la chance d'être mariés : leurs femmes subviennent à leurs besoins...

Le projet a été nommé Qt parce que le caractère « Q » était joli dans l'écriture Emacs de Haavard, et le « t » provient de Xt Toolkit. Le tout se prononçant en anglais *cute*, ce qui se traduit par « mignon » (le « t » étant minuscule, ne pas prononcer *cutie*...).

C'est en avril 1995 que *Trolltech* a son premier client, l'entreprise norvégienne Metis. Et durant presque un an, elle n'en a pas d'autre, rendant l'entreprise très fragile financièrement. Son second client, l'Agence spatiale européenne (ESA), lui achète dix licences en mars 1996.

De plus en plus de développeurs utilisent Qt, y compris parmi de grandes entreprises. On peut notamment citer : Google, Adobe Systems ou encore la NASA. Le site officiel du produit recense les entreprises utilisant Qt et les applications basées sur Qt (KDE, VLC media player, Skype, Google Earth, Oracle VM virtual box, ...).



Qt1

Le 26 mai 1995 est annoncée la première version publique de Qt sur le newsgroup comp.os.linux.announce. Et un an plus tard sort la version 0.97, puis le 24 septembre 1996 la version 1.0 est publiée et annoncée quelques jours plus tard.



C'est en 1997 que le projet KDE est lancé par Matthias Ettrich (qui est embauché par Trolltech l'année suivante). Ce dernier prend la décision d'utiliser Qt comme bibliothèque de base. Le fait qu'un projet de cette envergure utilise Qt sera une très bonne publicité pour Trolltech et sa bibliothèque. Depuis, les liens entre Trolltech et KDE n'ont fait que se renforcer.

Qt2

La seconde version majeure de Qt est publiée en juin 1999 et une version pour les systèmes embarqués, Qt/Embedded, connue depuis sous le nom de Qtopia, est publiée en 2000. Cette dernière version est conçue pour Linux et utilise directement son *framebuffer*, sans passer par le système de fenêtrage X11 (qui est inadapté pour les systèmes embarqués).

Qt3

Les deux premières versions majeures de Qt sont disponibles uniquement pour X11 et Windows, le support de Mac OS X arrive avec la version 3.0, publiée en 2001. Par rapport à la version 2.0, cette nouvelle version apporte un meilleur support de l'internationalisation, de l'Unicode ou encore des expressions rationnelles comme en Perl.

Qt4

Le 28 juin 2005, la version 4 est publiée. Elle améliore notamment le moteur de rendu (désormais appelé **Arthur**), la séparation entre données et présentation, et sépare la bibliothèque en modules :



- [QtCore](#) : pour les fonctionnalités non graphiques utilisées par les autres modules ;
- [QtGui](#) : pour les composants graphiques ;
- [QtNetwork](#) : pour la programmation réseau ;
-

À cela s'ajoute pour la version commerciale sous Windows deux autres modules liés à l'utilisation d'ActiveX : [QAxContainer](#) et [QAxServer](#).

Au cours de l'évolution de Qt 4, d'autres modules sont conçus, par exemple :

- [QtSvg](#) (4.1) : pour l'affichage d'images au format SVG ;
- [QtUiTools](#) (4.1) : pour charger dynamiquement les interfaces graphiques créées avec Qt Designer ;
- [QtScript](#) (4.3) : pour l'évaluation de scripts utilisant Qt Script ;
- [QtWebKit](#) (4.4) : portage du moteur de rendu web WebKit ;
- [QtXmlPatterns](#) (4.4) : pour manipuler des documents XML via XQuery et XPath ;
- [Phonon](#) (4.4) : intégration du framework multimédia de KDE 4, développé en collaboration avec la communauté KDE ;
- [QtMultimedia](#) (4.6) : diverses classes offrant des fonctionnalités multimédia de bas niveau comparées à [Phonon](#) ;
- [QtDeclarative](#) (4.7) : permet la conception d'interfaces graphiques dynamiques à l'aide d'un langage déclaratif (QML).



Le 28 janvier 2008, Nokia lance une OPA amicale pour racheter Qt et Trolltech. Trolltech, renommé en Qt Software, devient une division de Nokia. Dès lors, Nokia prend la décision en janvier 2009 d'abaisser le maximum de barrières pour faciliter l'adoption de Qt, qui depuis est utilisé par leurs développements en interne :

NOKIA
Connecting People



- ajout d'une licence plus permissive que la GPL, la LGPL ;
- ouverture du développement à des développeurs externes en rendant accessible le dépôt de Qt.

Nokia se recentrant sur Windows, elle cède en mars 2011 l'activité services et gestion des licences commerciales de Qt à la société Digia. Le 9 août 2012, elle cède intégralement la gestion du framework Qt à Digia pour une somme de 4,9 millions d'€ (à comparer au 150 millions de 2008 !).



Digia annonce vouloir étendre le support de Qt à Android, iOS et Windows 8.

Qt5

Qt 5.0 est sorti le 19 décembre 2012. Bien que marquant des changements majeurs sur bien des points (rôle important de QML et de JavaScript pour la création des interfaces graphiques avec Qt Quick, séparation en modules indépendants pour faciliter les livraisons, couche d'abstraction pour faciliter les portages, etc.), le passage à Qt5 casse au minimum la compatibilité au niveau des sources. De cette façon, le passage est bien plus facile que celui de Qt3 vers Qt4.

Qt Quick est un framework libre développé et maintenu par Digia faisant partie de la bibliothèque Qt. Il fournit la possibilité de créer des interfaces personnalisables et dynamiques avec des effets de transition fluides de manière déclarative. Ce type d'interface dynamique est de plus en plus commune, notamment sur les tablettes et smartphones.

Avec la version 5.3, le portage des applications sur les mobiles Android et iOS est complètement intégré à l'EDI (système de kits).

Licences

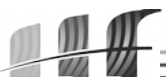
Qt Commercial License

Qt is available under a commercial license with various pricing models and packages that meet a variety of needs. A Qt commercial license keeps your code proprietary where only you can control and monetize on your end product's development, user experience and distribution. You also get great perks like additional functionality, world-class support and a close strategic relationship with The Qt Company to make sure your product and development goals are met.

Qt Open Source Licenses

Qt has been created under the belief of open development and providing freedom and choice to developers. To support that, The Qt Company also licenses Qt under various open source licenses, where some modules are available only under LGPLv2.1, LGPLv2.1 or LGPLv3, and some other modules under LGPLv3 only. In order to preserve the true meaning of open development and uphold the spirit of free software, it is imperative that the rules and regulations of open source licenses required by the GNU are followed.

Entrons dans le vif du sujet...



Architecture

Structure générale

L'API Qt est constituée de classes aux noms préfixés par Q et dont chaque mot commence par une majuscule (ex : `QLineEdit`), c'est la typographie *CamelCase*.

Ces classes ont souvent pour attributs des types énumérés déclarés dans l'espace de nommage Qt. Mis à part une architecture en pur objet, certaines fonctionnalités basiques sont implémentées par des macros (chaîne de caractères à traduire avec `tr`, affichage sur la sortie standard avec `qDebug...`).

Les conventions de nommage des méthodes sont assez semblables à celles de Java : le *lowerCamelCase* est utilisé, c'est-à-dire que tous les mots sauf le premier prennent une majuscule (ex : `indicatorFollowsStyle()`).

Les modificateurs sont précédés par `set`, en revanche les accesseurs prennent simplement le nom de l'attribut (ex : `text()`) ou commencent par `is` dans le cas des booléens (ex : `isChecked()`).

Arborescence des objets

Les objets Qt (ceux héritant de `QObject`) peuvent s'organiser d'eux-mêmes sous forme d'arbre. Ainsi, lorsqu'une classe est instanciée, on peut lui définir un objet parent.

Cette organisation des objets sous forme d'arbre facilite la gestion de la mémoire, car avant qu'un objet parent ne soit détruit, Qt appelle récursivement le destructeur de tous les enfants.


Cette notion d'arbre des objets permet également de déboguer plus facilement, via l'appel de méthodes comme `QObject::dumpObjectTree()` et `Object::dumpObjectInfo()`.

Signaux et slots

Les signaux et slots sont une implémentation du patron de conception observateur. L'idée est de connecter des objets entre eux via l'émission de signaux qui sont reçus par des slots.

Du point de vue du développeur, les signaux sont représentés comme de simples méthodes de la classe émettrice, dont il n'y a pas d'implémentation. Ces « méthodes » sont par la suite appelées, en faisant précéder « `emit` », qui désigne l'émission du signal.

Pour sa part, le slot connecté à un signal est une méthode de la classe réceptrice, qui doit avoir la même signature (autrement dit les mêmes paramètres que le signal auquel il est connecté), mais à la différence des signaux, il doit être implémenté par le développeur. Le code de cette implémentation représente les actions à réaliser à la réception du signal.

 C'est le MOC qui se charge de générer le code C++ nécessaire pour connecter les signaux et les slots.

Styles

La bibliothèque embarque divers thèmes de widgets qui lui donnent une bonne intégration visuelle sur toutes les plate-formes. Sur les environnements de bureau GNOME, Mac OS X et Windows les applications Qt ont ainsi l'apparence d'applications natives. Qt permet de personnaliser l'apparence des différents composants d'interface graphique en utilisant le principe des feuilles de style en cascade (CSS).

Outils de développement

Qt Development Frameworks fournit un ensemble de logiciels libres pour faciliter le développement d'applications Qt :

- Qt Designer est un concepteur d'interface graphique, il enregistre les fichiers `.ui` (XML) ;
- Qt Assistant permet de visualiser la documentation complète de Qt hors-ligne ;
- Qt Linguist est un outil dédié aux traducteurs, il leur permet d'ouvrir les fichiers `.ts` qui contiennent les chaînes de caractères à traduire, et d'entrer ensuite leur traduction ;



- **Qt Creator** est l'environnement de développement intégré dédié à Qt et facilite la gestion d'un projet Qt. Son éditeur de texte offre les principales fonctions que sont la coloration syntaxique, le complètement, l'indentation, etc. Qt Creator intègre en son sein les outils Qt Designer et Qt Assistant.

Même si Qt Creator est présenté comme l'environnement de développement de référence pour Qt, il existe des modules Qt pour les environnements de développement Eclipse et Visual Studio. Il existe aussi d'autres EDI dédiés à Qt et développés indépendamment de Nokia, comme QDevelop et Monkey Studio.

Des *bindings* existent afin de pouvoir utiliser Qt avec d'autres langages que le C++. Ainsi des langages tels que Java (Qt Jambi), Python (PyQt, PySide, PythonQt), Ruby (QtRuby), Ada (QtAda), C# (Qyoto), Pascal (FreePascal Qt4), Perl (Perl Qt4), Common Lisp (CommonQt) peuvent être utilisés...

Exemples avec le classique « Hello World » :

C++	Java (QtJambi)
<pre>#include <QApplication> #include <QPushButton> int main(int argc, char *argv[]) { QApplication app(argc, argv); QPushButton hello("Hello world!"); hello.show(); return app.exec(); }</pre>	<pre>import com.trolltech.qt.gui.QApplication; import com.trolltech.qt.gui.QPushButton; public class HelloWorld { public static void main(String args[]) { QApplication.initialize(args); QPushButton hello = new QPushButton("Hello World!"); hello.show(); QApplication.exec(); } }</pre>
Python (PyQt4)	C# (Qyoto)
<pre>from PyQt4 import QtGui, QtCore import sys app = QtGui.QApplication(sys.argv) hello = QtGui.QPushButton("Hello World!", None) hello.show() app.exec_()</pre>	<pre>using System; using Qyoto; public class HelloWorld { public static int Main(String[] args) { QApplication app = new QApplication(args); QPushButton hello = new QPushButton("Hello world!"); hello.Show(); return QApplication.Exec(); } }</pre>

Compilateur de méta-objets

Le **moc** (pour *Meta Object Compiler*) est un préprocesseur qui, appliqué avant compilation du code source d'un programme Qt, génère des méta-informations relatives aux classes utilisées dans le programme. Ces méta-informations sont ensuite utilisées par Qt pour fournir des fonctions non disponibles en C++, comme les signaux et slots et l'introspection.

L'utilisation d'un tel outil additionnel démarque les programmes Qt du langage C++ standard. Ce fonctionnement est vu par *Qt Development Frameworks* comme un compromis nécessaire pour fournir l'introspection et les mécanismes de signaux. À la sortie de Qt 1.x, les implémentations des *templates* par les compilateurs C++ n'étaient pas suffisamment homogènes.

qmake

Qt se voulant un environnement de développement portable et ayant le **moc** comme étape intermédiaire avant la phase de compilation/édition de liens, il a été nécessaire de concevoir un moteur de production spécifique. C'est ainsi qu'est conçu le programme **qmake**.

Ce dernier prend en entrée un fichier (avec l'extension **.pro**), facilement éditable par un développeur, décrivant le projet (liste des fichiers sources, dépendances, paramètres passés au compilateur...) et génère un fichier de projet spécifique à la plate-forme.



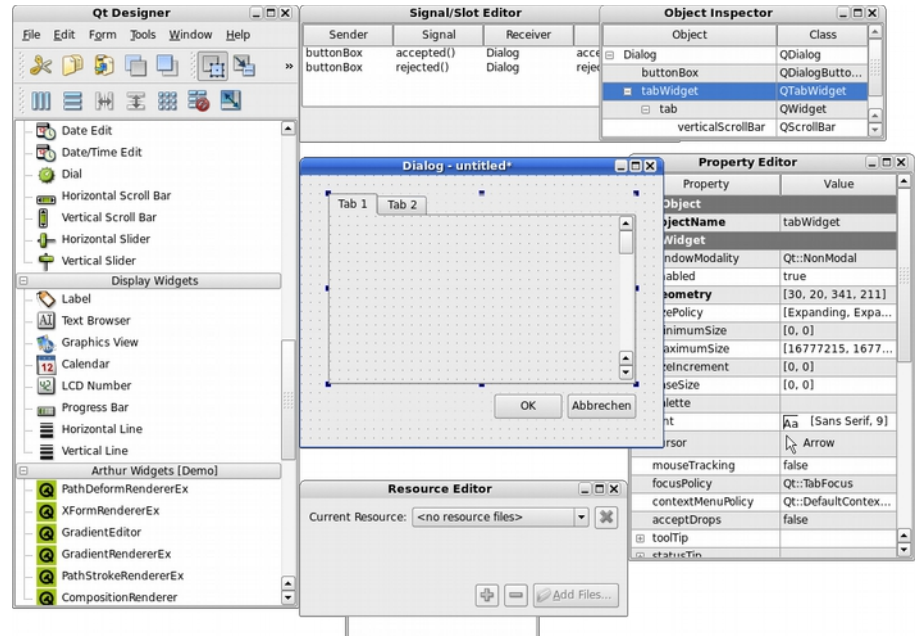
Ainsi, sous les systèmes UNIX, **qmake** produit un **Makefile** qui contient la liste des commandes à exécuter pour génération d'un exécutable, à l'exception des étapes spécifiques à Qt (génération des classes C++ lors de la conception d'interface graphique avec Qt Designer, génération du code C++ pour lier les signaux et les slots, ajout d'un fichier au projet...).

Concepteur d'interfaces

Qt Designer est un logiciel qui permet de créer des interfaces graphiques Qt dans un environnement convivial. L'utilisateur, par glisser-déposer, place les composants d'interface graphique et y règle leurs propriétés facilement. Les fichiers d'interface graphique sont formatés en XML et portent l'extension **.ui**.

Lors de la compilation, un fichier d'interface graphique est converti en classe C++ par l'utilitaire **uic** (*User Interface Compiler*). Il y a plusieurs manières pour le développeur d'employer cette classe :

- l'instancier directement et connecter les signaux et slots ;
- l'agréger au sein d'une autre classe ;
- en hériter pour en faire une classe mère avec ainsi accès à tous les éléments constitutifs de l'interface créée ;
- la générer à la volée avec la classe **QUiLoader** qui se charge d'interpréter le fichier XML **.ui** et retourner une instance de classe **QWidget**.



Internationalisation

Qt intègre son propre système de traduction, qui n'est pas foncièrement différent dans le principe de la bibliothèque **gettext**. Selon le manuel de Qt Linguist, l'internationalisation est assurée par la collaboration de trois types de personnes : les développeurs, le chef de projet et les traducteurs.

Dans leur code source, les développeurs entrent des chaînes de caractères dans leur propre langue. Ils doivent permettre la traduction de ces chaînes grâce à la macro **tr()**. En cas d'ambiguïté sur le sens d'une expression, ils peuvent également indiquer des commentaires destinés à aider les traducteurs.

Le chef de projet déclare les fichiers de traduction (un pour chaque langue) dans le fichier de projet. L'utilitaire **lupdate** parcourt les sources à la recherche de chaînes à traduire et synchronise les fichiers de traduction avec les sources.

Les fichiers de traductions sont des fichiers XML portant l'extension **.ts**.

Les traducteurs utilisent Qt Linguist pour renseigner les fichiers de traduction. Quand les traductions sont finies, le chef de projet peut compiler les fichiers **.ts** à l'aide de l'utilitaire **lrelease** qui génère des fichiers binaires portant l'extension **.qm**, exploitables par le programme.

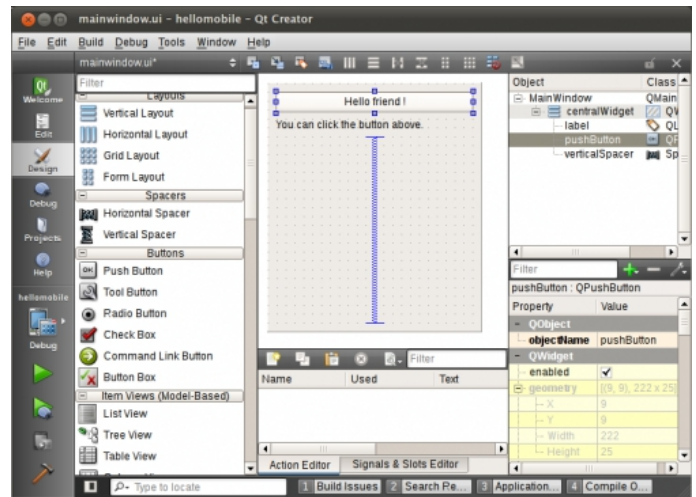
Ces fichiers sont lus à l'exécution et les chaînes de caractères qui y sont trouvées remplacent celles qui ont été écrites par les développeurs.

Qt Creator

Qt Creator est un EDI (environnement de développement intégré) multi plate-formes faisant partie du framework Qt. Il est donc orienté pour la programmation en C++.

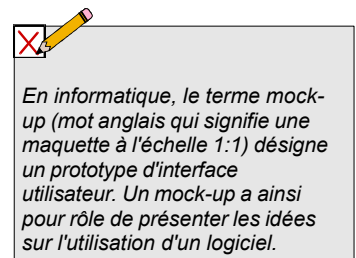
Il intègre directement dans l'interface un débogueur, un outil de création d'interfaces graphiques (Qt Designer), des outils pour la publication de code ainsi que la documentation Qt (Qt Assistant). L'éditeur de texte intégré permet l'auto-complétion ainsi que la coloration syntaxique.

Qt Creator utilise le compilateur **gcc/g++** sous Linux/Mac OS X, et **MinGW** (*Minimalist GNU for Windows*) par défaut sous Windows.



QML (Qt Markup Language)

Dans le milieu professionnel, les interfaces graphiques sont souvent créées par des *designers*, sans compétences en programmation. Ils utilisent des technologies comme Flash pour faire des *mock-up* qui sont ensuite fournis au programmeur qui doit alors refaire tout le travail pour implémenter le tout ; et souvent ce que le *designer* a produit n'est pas réellement possible en pratique ou est trop lent. Avec QML, le *designer* travaille directement sur le résultat final de l'interface et voit directement ce qui est possible ou pas...



En informatique, le terme *mock-up* (mot anglais qui signifie une maquette à l'échelle 1:1) désigne un prototype d'interface utilisateur. Un *mock-up* a ainsi pour rôle de présenter les idées sur l'utilisation d'un logiciel.

Qt Quick (Qt User Interface Creation Kit)

Qt Quick est un ensemble de trois produits qui permettant de créer des interfaces dynamiques facilement, rapidement et compatibles avec les technologies Qt déjà en place. Qt Quick comprend :

QML : un langage déclaratif type Javascript et qui simplifie la création d'interfaces animées et fluides ;

QtDeclarative : un ensemble de classes C++ qui traduisent le QML en objets `QGraphicScene`, et permettent la connexion entre l'interface QML et l'application en C++ ;

Qt Creator : l'IDE supporte pleinement le QML et offre ainsi une interface facile à utiliser via le *drag-n-drop*, l'auto-complétion et la détection d'erreurs de syntaxe.

Interface Qt Quick →

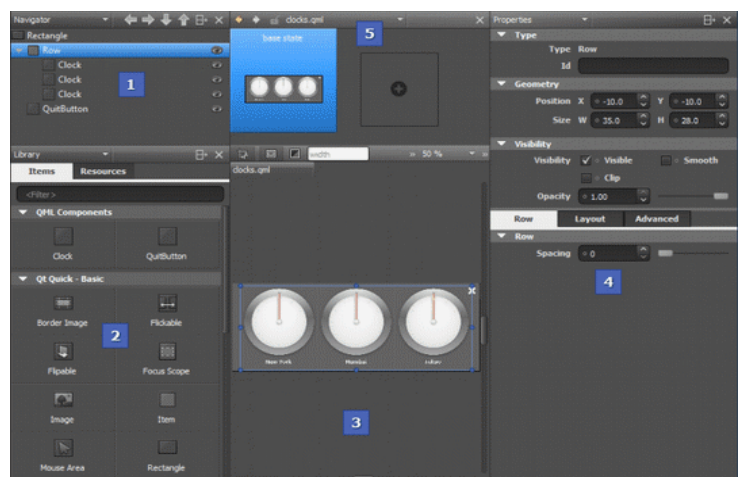
Navigator (1) : éléments du fichier QML courant sous forme d'arborescence.

Library (2) : bibliothèque des éléments disponibles ; éléments QML, composants Qt Quick, composants personnels, autres ressources...

Canvas (3) : espace de construction et de *design* des interfaces.

Properties (4) : panneau des propriétés du composant ou élément QML sélectionné ; les propriétés peuvent aussi être changées à partir de l'éditeur de code.

State (5) : montre les différents états d'un composant ; les états décrivent les configurations d'une interface utilisateur (les éléments, leurs propriétés, le comportement, les actions possibles...).



Installation du SDK

Le SDK (*Software Development Kit*) est une distribution intégrée basée sur l'EDI Qt Creator. Le développeur a ainsi accès directement à tous les outils (débugueur, assistant, designer, ...) au travers d'une unique IHM. Le SDK intègre bien évidemment le *Qt framework* de base qui peut aussi être utilisé en ligne de commande, sans passer par l'EDI.

Les paquetages nécessaires sont disponibles sur le site <http://www.qt.io/download-open-source/> dans la rubrique « *Qt Online Installers* ».

La version à ce jour est Qt SDK 1.6.0, qui comprend Qt Creator 3.2.2 basé sur Qt 5.3.2.

note : La version *online* est préférable pour bénéficier des dernières mises à jour en temps réel... mais l'installation nécessite bien sûr une connexion Internet active.

Lors de la procédure d'installation, dans la rubrique *Select Components*, des kits de développement pour mobiles sont sélectionnés par défaut, en plus de celui du système hôte (Desktop).



Le kit de développement pour Android (armv7) est disponible sous Linux, Mac OS X et Windows. Celui pour iOS n'est disponible que sous Mac OS X...

Linux (Ubuntu)

Archive utilisée : `qt-opensource-linux-x64-1.6.0-5-online.run` (64 bits)

ou : `qt-opensource-linux-x86-1.6.0-5-online.run` (32 bits)

Rendre l'archive exécutable par : `chmod u+x qt-opensource-linux-x64-1.6.0-5-online.run`

Lancer l'installation : `./qt-opensource-linux-x64-1.6.0-5-online.run`

En cas de problème (écran figé), ajouter l'option `-style cleanlooks` ...

Par défaut, le SDK est installé dans `$HOME/Qt`.

L'EDI est situé dans `$HOME/Qt/QtCreator/bin` ; un raccourci est normalement ajouté dans la rubrique « Applications | Développement » des Menus de commandes d'Ubuntu.



Mac OS X

Sous Mac OS X, il faut d'abord vérifier la disponibilité des outils de développement (gcc, make, ...).

Dans le cas contraire, installer si nécessaire XCode à partir de l'AppStore, puis dans la rubrique *Preferences | Downloads*, onglet *Components*, installer les *Command Line Tools*.

Archive utilisée : `qt-opensource-mac-x64-1.6.0-5-online.dmg`

Ouvrir l'archive et lancer l'application d'installation.

Par défaut, le SDK est installé dans `$HOME/Qt`.

L'EDI « Qt Creator.app » est situé directement dans `$HOME/Qt`.



Windows

Archive utilisée : `MinGW-gcc440_1.zip` (à rechercher sur le Web...)

Ce paquetage permet de disposer des outils de développement GNU (gcc, make, ...) sous Windows. Installer d'abord MinGW (*Minimalist GNU for Windows*), par extraction de l'archive directement à la racine du disque (le chemin doit être compatible DOS...) ; Dans ce qui suit, MinGW est supposé être dans `C:\MinGW`.

Cliquer sur « Démarrer » puis clic droit sur « Ordinateur » ; choisir « Paramètres système avancés » puis cliquer sur « Variables d'environnement ». Sélectionner la variable `Path` et lui ajouter en début de liste l'entrée `C:\MinGW\bin` (séparateur ;).



Ouvrir l'Invite de commande et tester l'accès aux outils par : `make -version` et `g++ --version`.

Archive utilisée : `qt-opensource-windows-x86-1.6.0-5-online.exe`

Lancer l'installation par double-clic sur le fichier `exe`.

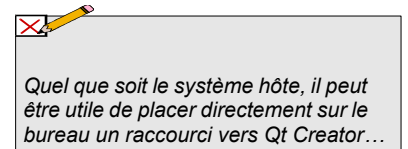
Par défaut, le SDK est installé dans `C:\Qt`.

L'EDI est situé dans `C:\Qt\QtCreator\bin` ; un raccourci est normalement ajouté dans les commandes accessibles via le bouton Démarrer.

Post-installation

Pour une utilisation minimale en ligne de commande, il faut :

- un terminal ;
- un éditeur ;
- l'accès aux outils de développement Qt.



Ces outils sont localisés dans `<install_dir>/<version>/<compiler>/bin`

Par exemple pour Mac OS X : `$HOME/Qt/5.3/clang_64/bin`

Et pour Linux 64 bits : `$HOME/Qt/5.3/gcc_64/bin`

Linux et Mac OS X

Éditer le fichier `.bashrc` (ou `.profile` sous Mac OS X) du répertoire `$HOME` (si ce fichier n'existe pas, le créer), et lui ajouter :

```
export QTDIR=<install_dir>/<version>/<compiler>/bin
export PATH=$QTDIR:$PATH
```

Le chemin d'accès au répertoire `bin` de Qt est ainsi ajouté à la liste des localisations des exécutables (il est placé en tête pour s'affranchir de la présence d'éventuelles versions précédentes de Qt).

Sous Mac OS X, ajouter des alias vers les outils graphiques :

```
alias designer='open $QTDIR/Designer.app'
alias assistant='open $QTDIR/Assistant.app'
alias qmlviewer='open $QTDIR/QMLViewer.app'
```

Lancer un nouveau terminal (pour prendre en compte le nouveau `.bashrc`). Le contenu de la variable d'environnement `PATH` peut être vérifié par : `echo $PATH`.

Windows

Cliquer sur « Démarrer » puis clic droit sur « Ordinateur » ; choisir « Paramètres système avancés » puis cliquer sur « Variables d'environnement ». Sélectionner la variable `Path` et lui ajouter en début de liste l'entrée `<install_dir>/<version>/<compiler>/bin` (séparateur ;).

Vérification

À partir d'un répertoire quelconque de travail, vérifier l'accès à `qmake` et sa version :

```
prompt> qmake --version
QMake version 3.0
Using Qt version 5.3.2 in<install_dir>/<version>/<compiler>/lib
```

Vérifier aussi le démarrage correct du `designer` :

```
prompt> designer &
```



Programmes de test de l'installation

Premier test : C++

Créer un répertoire de travail nommé `HelloWorld`. Ajouter un fichier source `hello.cpp` contenant les quelques lignes C++ proposées page 7 du présent document.

Créer un fichier projet `HelloWorld.pro` contenant les lignes ci-contre →

```
QT      += widgets
TEMPLATE = app
TARGET   = HelloWorld
SOURCES += hello.cpp
```

Linux

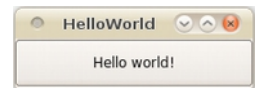
Fabriquer le Makefile par `qmake` puis générer le programme par `make`

Tester : `./HelloWorld`

Les bibliothèques dynamiques (extension `.so`) utilisées par l'application peuvent être listées par la commande : `ldd HelloWorld`.

(c'est l'option `-rpath` du compilateur qui permet l'utilisation de bibliothèques dynamiques situées ailleurs que dans les répertoires spécifiés habituellement par `ldconfig`... Un petit coup d'œil au fichier Makefile peut être très instructif ; voir plus loin le chapitre « déploiement d'une application »).

La version *debug* s'obtient en remplaçant la commande `qmake` par `qmake -config debug` (l'exécutable obtenu passe alors de 8 ko à 120 ko).



Mac OS X

Fabriquer le Makefile par `qmake` puis générer le programme par `make`

Le résultat obtenu en tant qu'application est une arborescence (*bundle*) de point d'entrée `HelloWorld.app`.

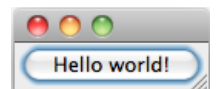
Le lancement via la ligne de commande s'effectue par : `open HelloWorld.app`.

Dans le *bundle*, le chemin d'accès relatif à l'exécutable est `HelloWorld.app/Contents/MacOS/HelloWorld`.

Pour obtenir directement un exécutable en ligne de commande à la place du *bundle*, il suffit d'ajouter au fichier projet la ligne : `CONFIG -= app_bundle`. Le programme peut alors être lancé par : `./HelloWorld`.

Les bibliothèques dynamiques (extension `.dylib`) utilisées par l'application peuvent être listées par la commande : `otool -L HelloWorld.app/Contents/MacOS/HelloWorld`.

La version *debug* s'obtient en utilisant la commande `qmake -config debug` (l'exécutable obtenu passe alors de 16 ko à 20 ko).



Windows

Fabriquer le Makefile par `qmake` puis générer le programme par `make release`

L'utilitaire `qmake` créé automatiquement deux sous-répertoires *debug* et *release*. Le résultat `HelloWorld.exe` est placé dans le sous-répertoire nommé *release*.

Tester : `release\HelloWorld.exe`

Les bibliothèques dynamiques (extension `.dll`) utilisées par l'application peuvent être listées par la commande : `tasklist /m /fi "imagename eq HelloWorld.exe"`, mais ceci seulement lorsque le programme est en cours d'exécution...

La version *debug* s'obtient en remplaçant la commande `make release` par `make debug` (l'exécutable est alors placé dans le sous-répertoire *debug*, il passe alors de 58,5 ko à 446 ko !).



Note : Avant de changer de cible (passage de *debug* à *release* ou le contraire), il est indispensable de forcer la recompilation des sources en tapant au préalable : `make clean`.



Deuxième test : QML



Créer un répertoire de travail nommé par exemple `HelloQML`. Ajouter le fichier `hello.qml` contenant les lignes suivantes :

```
import QtQuick 2.2

Rectangle {
    id: page
    width: 500; height: 200
    color: "lightgray"

    Text {
        id: helloText
        text: "Hello world!"
        y: 30
        anchors.horizontalCenter: page.horizontalCenter
        font.pointSize: 24; font.bold: true

        MouseArea { id: mouseArea; anchors.fill: parent }

        states: State {
            name: "down"; when: mouseArea.pressed == true
            PropertyChanges { target: helloText; y: 160; rotation: 180; color: "red" }
        }

        transitions: Transition {
            from: ""; to: "down"; reversible: true
            ParallelAnimation {
                NumberAnimation { properties: "y,rotation"; duration: 500; easing.type: Easing.InOutQuad }
                ColorAnimation { duration: 500 }
            }
        }
    }
}
```

`qmlscene` est un outil qui permet de tester le rendu des fichiers Qt Quick 2.0. Tester l'IHM par la commande : `qmlscene hello.qml` (et cliquer sur le texte...).



Pour Qt Quick 1.0, il faut utiliser la commande en ligne `qmlviewer`.

Ajouter les fichiers ci-dessous afin de transformer le QML en application exécutable :

```
// main.cpp

#include <QApplication>
#include <QQuickView>

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

    QQuickView* view = new QQuickView() ;
    view->setSource(QUrl::fromLocalFile("hello.qml") ) ;
    view->show() ;

    return app.exec() ;
}
```

```
# HelloQML.pro

QT      += widgets qml quick

TEMPLATE = app
TARGET  = HelloQML

SOURCES += main.cpp

macx: CONFIG -= app_bundle
```

Générer le programme en suivant les procédures décrites lors du premier programme de test.

Tester l'application (Attention : l'exécution doit toujours être lancée depuis le répertoire contenant le fichier QML, sous peine d'obtenir une fenêtre vide...).



Portabilité des applications

Dans certaines situations, le code source ne peut être intégralement portable d'un OS à l'autre. Le développeur doit alors prévoir différentes versions tenant compte du système hôte.

Exemple : En ligne de commande, ouverture d'un fichier avec l'application qui est associée à son extension :

Sous Linux Gnome : `gnome-open <filename>`
Sous Linux KDE : `kfmclient exec file:<filename>`
Sous Mac OS X : `open <filename>`
Sous Windows : `cmd /C <filename>`

Il existe plusieurs possibilités pour favoriser une auto-adaptation du code...

Au niveau du fichier projet

Le projet logiciel peut contenir des clauses exclusives en fonction de la station d'accueil :

<pre># Linux only unix: !macx { }</pre>	<pre># MacOSX only macx { }</pre>	<pre># Windows only win32 { }</pre>
---	-------------------------------------	---------------------------------------

Le développeur peut ainsi mettre en place les éléments spécifiques requis en fonction de l'OS destinataire.

Au niveau du code

Qt propose aussi en global des constantes en fonction du système hôte :

<pre>// Linux only #ifdef Q_OS_LINUX #endif</pre>	<pre>// MacOSX only #ifdef Q_OS_MAC #endif</pre>	<pre>// Windows only #ifdef Q_OS_WIN32 #endif</pre>
---	--	---

Ces constantes peuvent être exploitées directement dans le code. Elles sont définies dans `<QtGlobal>`.

L'exemple ci-dessus peut donc être traité de la manière suivante, avec lancement d'un *thread* secondaire (en supposant que l'instance `path` de classe `QString` contienne l'URI du fichier à ouvrir) :

```
#ifdef Q_OS_LINUX
    QString cmd = QString("gnome-open \"%1\"").arg( path );
#endif

#ifdef Q_OS_MAC
    QString cmd = QString("open \"%1\"").arg( path );
#endif

#ifdef Q_OS_WIN32
    QString cmd = QString("cmd /C \"%1\"").arg( path );
#endif
ExtCmd* extcmd = new ExtCmd( cmd );
extcmd->run();
```

Classe utilitaire de création d'un *thread* (version simplifiée) →

```
// ExtCmd.cpp

#include "ExtCmd.h"
#include <QProcess>

ExtCmd::ExtCmd(const QString& cmd )
{
    this->cmd = cmd ;
}

void ExtCmd::run()
{
    QProcess* proc = new QProcess ;
    proc->start( cmd ) ;
}
```

```
// ExtCmd.h

#ifndef EXTCMD_H
#define EXTCMD_H

#include <QThread>
#include <QString>

class ExtCmd : public QThread
{
public :
    ExtCmd(const QString& cmd ) ;
    void run() ;

private :
    QString cmd ;
};

#endif
```



Déploiement d'une application

Pour distribuer une application développée en Qt, il ne suffit pas de fournir son exécutable ; il faut aussi que le système hôte dispose des bibliothèques liées dynamiquement lors de l'exécution.

Exemple : le premier test *HelloWorld* vu précédemment nécessite les modules partageables `QtCore`, `QtGui` et `QtWidgets` (voir les commandes `ldd`, `otool` et `tasklist` présentées page 12).

Pour déployer l'application sur un autre poste, il faut donc disposer de ces bibliothèques ou les embarquer...

Linux



Les bibliothèques Qt dynamiques sont de la forme `libQt*.so` (*shared object*) :

- elles peuvent être copiées dans le répertoire `/usr/lib` de la cible ; mais cette méthode nécessite les droits *root* et risque d'écraser une éventuelle version déjà présente (et de briser des liens au niveau du gestionnaire de paquet) ;
- elles peuvent être livrées avec l'exécutable, dans le même répertoire, sous réserve d'ajouter un script de démarrage de l'application spécifiant l'emplacement au *dynamic linker*.

Cette deuxième solution est celle généralement préconisée.

Le script ci-contre peut jouer ce rôle →

il suffit de le nommer `HelloWorld.sh` et de le rendre exécutable par `chmod +x`.

L'application est ensuite lancée par `./HelloWorld.sh`.

```
#!/bin/sh

appname=`basename $0 | sed s,\.sh$,`
dirname=`dirname $0`
tmp="${dirname#?}"

if [ "${dirname$tmp}" != "/" ]; then
    dirname=$PWD/$dirname
fi

LD_LIBRARY_PATH=$dirname
export LD_LIBRARY_PATH

$dirname/$appname "$@"
```

Il existe cependant une troisième solution plus élégante et qui ne nécessite pas de script. Il suffit de modifier le fichier `.pro` de l'application en ajoutant :

```
QMAKE_LFLAGS += '-Wl,-rpath,\"$${ORIGIN}/libs\"'
```

Cette configuration priorise la recherche des bibliothèques dynamiques dans un sous-répertoire `libs` de l'exécutable... Sous-répertoire dans lequel il suffit de fournir les `libQt*.so` requis !

Mac OS X



Les bibliothèques Qt dynamiques sont disponibles sous forme de dossiers *frameworks*.

Le répertoire `$QTDIR` contient un programme utilitaire nommé `macdeployqt` automatisant la fabrication d'un *bundle* conforme à la philosophie OS X (sans dépendance extérieure). Il suffit d'appliquer cet outil à `HelloWorld.app` par la commande :

```
macdeployqt HelloWorld.app.
```

Windows



Les bibliothèques Qt dynamiques sont des DLL (*Dynamic Link Library*).

Le moyen le plus simple de déployer une application Qt Windows est de livrer les DLL nécessaires dans le même répertoire que le programme.



Choix de l'interface graphique

Qt propose une panoplie d'outils de création d'interfaces graphiques. Cette panoplie peut être intimidante au premier abord, d'autant que certains composants sont combinables ou interdépendants, voire paraissent parfois concurrents. Dans les grandes lignes, une application graphique Qt peut être basée sur Qt Widgets, Graphics View ou Qt Quick.

Le principal critère de choix entre ces différents composants est le type d'application visé.

L'application nécessite-t-elle réellement une interface graphique ?

Une application de type service ou démon peut ne pas avoir d'interface utilisateur du tout, ou être pilotée à distance par une interface IPC (*Inter-Process Communication*) ; l'interface peut aussi être textuelle, par exemple pour un usage en ligne de commande : dans ces cas, l'application Qt sera généralement basée sur `QCoreApplication`.

Interfaces Web

Les composants graphiques Qt ne permettent pas de créer une interface purement Web (c'est-à-dire utilisable par un navigateur du même nom).

Par contre, il est possible de réaliser des applications dites « hybrides » en intégrant un composant `WebView` dans une interface Qt Quick ou Qt Widgets. Ce composant, basé sur `WebKit`, permet de réutiliser du code HTML5 ou d'intégrer des pages Web en ligne dans une application native.

Dans le même esprit, bien qu'il ne s'agisse pas d'un composant graphique, le moteur JavaScript de Qt permet de combiner JavaScript avec QML et C++ dans toutes les parties d'une application.

Interfaces graphiques Desktop

Toutes les interfaces graphiques de Qt ont accès à des surfaces de dessin OpenGL, et donc peuvent bénéficier d'une accélération matérielle 2D ou 3D (Qt Quick 2 est même exclusivement basé sur OpenGL). Pour des effets graphiques personnalisés, il est possible d'écrire directement du code GLSL (*shaders*), et le module Qt 3D fournit une API de plus haut niveau spécifiquement pour les interfaces 3D.

Dans le cas d'applications de type bureautique, Qt Widgets convient pour des interfaces basées sur des composants standards de la plateforme (listes, arbres, boutons, champs d'édition, etc.). Depuis Qt version 5.1, le module Qt Quick Controls peut être envisagé en remplacement de Qt Widgets dans les cas les plus courants.

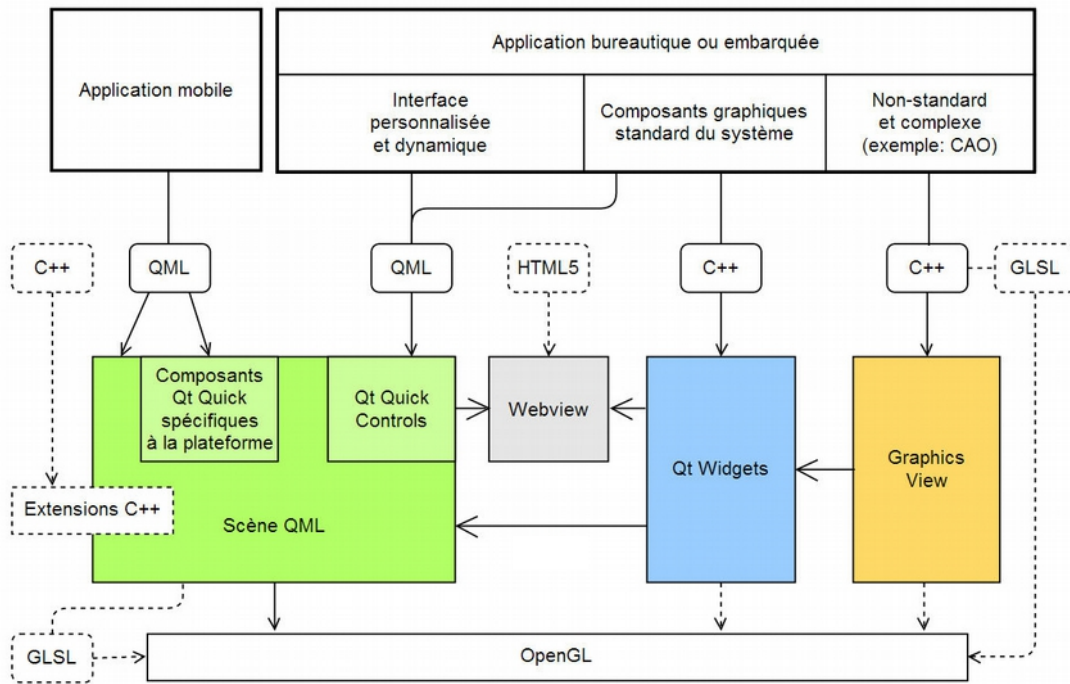
Graphics View permet une utilisation libre d'une surface d'affichage 2D tout en intégrant de nombreux services de gestion des objets de l'interface (création, sélection, déplacement, transformation, zoom, etc.). La vue graphique peut gérer sans ralentissement de très nombreux éléments, elle peut également intégrer les widgets du module Qt Widgets. Cette API est donc bien adaptée à la création d'applications de type CAO, d'affichage de graphes interactifs ou de contrôle/commande industriel ou domotique.

Applications mobiles

Qt Quick est particulièrement recommandé pour des applications mobiles, auxquelles il apporte sa facilité de création d'interfaces tactiles animées et personnalisées. Chaque plateforme ajoute ses composants spécifiques, le tout formant un SDK permettant une parfaite intégration de l'application dans l'environnement.

Qt Quick est également bien adapté au domaine de l'embarqué (véhicules, appareils industriels et médicaux, etc), domaine dans lequel, comme pour les applications mobiles, les interfaces tactiles et personnalisées sont prépondérantes.





Qt Essentials (*foundation of Qt on all platforms*)

Module	Description	Usage C++ / .pro / QML
Qt Core	Core non-graphical classes used by other modules.	#include <QtCore> included by default
Qt GUI	Base classes for graphical user interface (GUI) components. Includes OpenGL.	#include <QtGui> included by default, unless QT -= gui
Qt Multimedia	Classes for audio, video, radio and camera functionality.	#include <QtMultimedia> QT += multimedia import QtMultimedia 5.0
Qt Multimedia Widgets	Widget-based classes for implementing multimedia functionality.	#include <QtMultimediaWidgets> QT += multimediawidgets
Qt Network	Classes to make network programming easier and more portable.	#include <QtNetwork> QT += network
Qt QML	Classes for QML and JavaScript languages.	#include <QtQml> QT += qml import QtQml 2.0
Qt Quick	A declarative framework for building highly dynamic applications with custom user interfaces.	#include <QtQuick> QT += quick import QtQuick 2.x
Qt Quick Controls	Reusable Qt Quick based UI controls to create classic desktop-style user interfaces (window parts like bar and menu, views, controls).	#include <QtQuick.Controls> QT += qml quick import QtQuick.Controls 1.2
Qt Quick Dialogs	Types for creating and interacting with system dialogs from a Qt Quick application (dialog, color, file, font, message).	import QtQuick.Dialogs 1.2
Qt Quick Layouts	Layouts are items that are used to arrange Qt Quick 2 based items in the user interface (row, column, grid).	import QtQuick.Layouts 1.1
Qt SQL	Classes for database integration using SQL.	#include <QtSql> QT += sql
Qt Test	Classes for unit testing Qt applications and libraries.	#include <QtTest> QT += testlib
Qt WebKit	Provides the WebView API, which allows QML applications to render regions of dynamic web content.	import QtWebKit 3.0
Qt WebKit Widgets	Provides a Web browser engine that makes it easy to embed content from the World Wide Web into your Qt application.	#include <QtWebKitWidgets> QT += webkitwidgets
Qt Widgets	Classes to extend Qt GUI with C++ widgets.	#include <QtWidgets> QT += widgets

