Overview of Qt

A quick survey of features and functionality

By Jon Kroll, Ann Arbor Michigan

*“**Qt** is much more than just a cross-platform SDK - it's a technology strategy that lets you quickly and cost-effectively design, develop, deploy, and maintain software while delivering a seamless user experience across all devices.”*

Qt.io website

What I Consider Qt To Be

Agenda

- Where does Qt run?
- Qt developer resources
- Qt licensing
- Installing Qt
- Supported configurations
- Developer tools
- Widget-based user interfaces
- Core internals
- Internationalization
- Threading
- Data storage
- Multimedia
- Networking and connectivity
- Graphics
- Scripting
- Testing and debugging
- Accessibility
- Other cool stuff in Qt

Where Does Qt Run?

- **Embedded Devices** - embedded software with a responsive UX for nearly any hardware
- **Desktop Application Software** - native performance across a wide range of operating systems with a unified codebase for cross-platform development
- **Internet of Things** - from connected devices with high-performance graphics to non-GPU battery UIs, Qt is scalable to get the most performance out of your hardware
- **Mobile** - reuse your Qt code in a companion app; Qt runs flawlessly on multiple platforms including iOS and Android
- **Automotive** - HMIs for in-vehicle infotainment (IVI), digital instrument clusters, and HUD
- **Automation** - automation systems for smart factories, homes, and services that integrate with other systems, sensors and triggers
- **Set-top-box & DTV** - deploy your content across multiple screens with hybrid programming and HTML5
- **Medical** - Create safe and certified medical devices that comply with IEC 62304 & ISO 14971 and run RTOS on off-the-shelf hardware

Qt Development Resources

The Qt website has **developer resources** to learn how to use Qt and helps you create application, connected devices, and UIs.

Qt offers **exceptional support and guidance** on the use of Qt APIs, functions, methods, and programing techniques.

Qt provides **all levels of training** to get a introductory or comprehensive understanding of the framework.

Qt offers **professional services** for everything from UX design to hardware adaptation to help make your project a success.

Qt Licensing

Qt licensing for application development lets you create applications for desktop and mobile platforms. It contains all the APIs and the Qt Creator IDE for cross-platform development.

Qt is setup as dual-licensed for **commercial** and **open source** licenses.

- The **commercial** Qt license gives you the full rights to create and distribute software on your own terms without any open source licensing obligations. The commercial license also provides access to official Qt technical support.

- Qt is also available under GPL and LGPLv3 **open source** licenses. Qt tools and some libraries are only available under GPL. The Qt open source licensing is ideal for use cases such as open source projects with open source distribution, internal research, student/academic, hobby, or other projects where all (L)GPL obligations can be met.

Installing Qt

You can install the Qt Framework and tools with an installer that runs online or offline, by downloading and installing binary distributions, or by downloading and building the source packages yourself.

With the online installer, you can select commercial or open source versions of Qt, tools, and Add-Ons. Using the online installer requires a Qt account and sign in. The installer retrieves the license attached to the account from a server and provides a list of available components corresponding to the license.

The offline installer is a single package that contains all of Qt and Add-Ons for the required target platform.

Supported Configurations Qt 5.11

- Windows
- Linux/X11
- Android
- Darwin Platforms: macOS, iOS, tvOS, watchOS
- Embedded Platforms: Embedded Linux, QNX, INTEGRITY

Supported Configurations Qt 5.11: Windows

- Windows
  - Windows 10 (x86_64, x86) MSVC 2017, MSVC 2015, MinGW 5.3
  - Windows 8.1 (x86_64, x86) MSVC 2017, MSVC 2015, MinGW 5.3
  - Windows 7 (x86_64, x86) MSVC 2017, MSVC 2015, MinGW 5.3
  - Universal Windows Platform (UWP)
  - UWP 10 (x86, x86_64, armv7) MSVC 2017, MSVC 2015
- Linux/X11
- Android
- Darwin Platforms: macOS, iOS, tvOS, watchOS
- Embedded Platforms: Embedded Linux, QNX, INTEGRITY

Supported Configurations Qt 5.11: Linux/X11

- Windows
- Linux/X11
  - openSUSE 42.2 (x86_64), GCC 4.8, GCC 7, ICC
  - Red Hat Enterprise Linux 6.6 (x86_64), GCC 4.9.1, devtoolset-3
  - Red Hat Enterprise Linux 7.2 (x86_64), GCC 5.3.1, devtoolset-4
  - Ubuntu 16.04 (x86_64) GCC as provided by Canonical, GCC 6
  - Linux (x86 and x86_64) GCC 4.8, GCC 4.9, GCC 5, GCC 6, GCC 7
- Android
- Darwin Platforms: macOS, iOS, tvOS, watchOS
- Embedded Platforms: Embedded Linux, QNX, INTEGRITY

Supported Configurations Qt 5.11: Android

- Windows
- Linux/X11
- Android
  - Android 4.1, 5, 6, 7, 8 (armv7, x86)
  - API Level 16. GCC as provided by Google, MinGW 5.3. Hosts: RHEL 7.2 (x86_64), macOS 10.12 (x86_64), Windows 7 (x86_64)
- Darwin Platforms: macOS, iOS, tvOS, watchOS
- Embedded Platforms: Embedded Linux, QNX, INTEGRITY

Supported Configurations Qt 5.11: Darwin

- Windows
- Linux/X11
- Android
- Darwin Platforms
  - macOS 10.11, 10.12, 10.13 (x86_64), Clang from Apple, Xcode 8.2 (macOS 10.11), Xcode 8.3.3 (macOS 10.12), Xcode 9 (macOS 10.13)
  - iOS 10, iOS 11 (armv8), Clang from Apple, Xcode 9 (macOS 10.13)
  - tvOS 10, tvOS 11 (armv8) (tech preview), Clang from Apple, Xcode 9 (macOS 10.13)
  - watchOS 3, watchOS 4 (armv7k) (tech preview), Clang from Apple, Xcode 9 (macOS 10.13)
- Embedded Platforms: Embedded Linux, QNX, INTEGRITY

Supported Configurations Qt 5.11: Embedded

- Windows
- Linux/X11
- Android
- Darwin Platforms: macOS, iOS, tvOS, watchOS
- Embedded Platforms
  - Embedded Linux GCC ARM Cortex-A, Intel boards with GCC-based toolchains
  - Embedded Linux (Boot2Qt) (armv7, armv8, x86, x86_64) GCC 6.2, Yocto 2.3, Hosts: RHEL 7.2 (x86_64), Windows 7 (x86_64)
  - QNX 6.6.0, 7.0 (armv7 and x86), QCC as provided by QNX, Hosts: RHEL 7.2 (x86_64), Windows 7 (x86_64), Windows 10 (x86_64), Windows 7 (x86), macOS 10.12
  - INTEGRITY 11.4.4, As provided by Green Hills INTEGRITY, Host: RHEL 7.2 (x86_64)

Development Tools

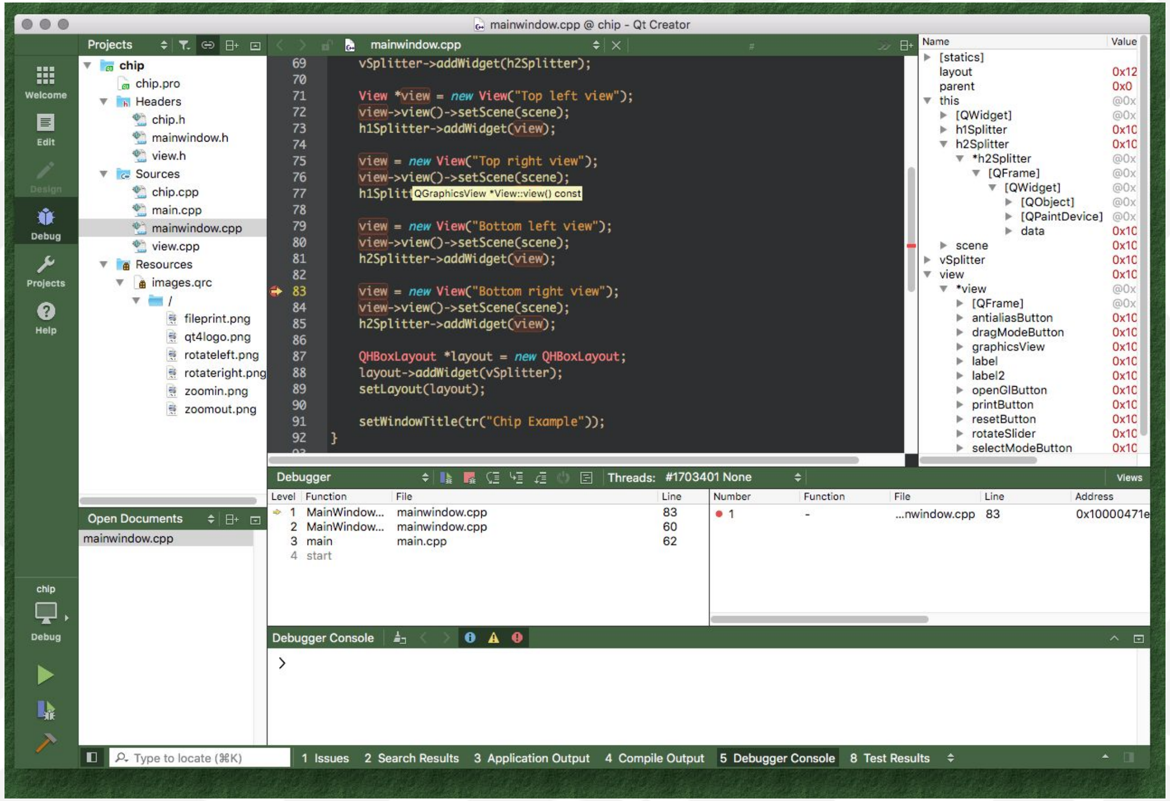
A Qt distribution comes with a complete suite of development tools to design, code, build, test, and deploy an application on any of the supported platforms. On Windows, starting with Qt 5.0.1, there are also binary installers that ship a Mingw-w64 based toolchain + pre-built Qt libraries.

Some of the development tools include: qmake, moc, rcc, Qt Creator, Qt Designer, Qt Linguist, application deployment scripts, and more.

Development Tools: Qt Creator

Qt Creator provides tools for accomplishing tasks throughout the whole application development life-cycle, from creating a project to deploying the application on the target platforms. Qt Creator automates some tasks, such as creating projects, by providing wizards that guide step-by-step through the project creation process, create the necessary files, and specify settings depending on the choices made. It also speeds up some tasks, such as writing code, by offering semantic highlighting, checking code syntax, code completion, refactoring actions, and other useful features.

Development Tools: Qt Creator



```
    vSplitter->addWidget(h2Splitter);

    View *view = new View("Top left view");
    view->view()->setScene(scene);
    h1Splitter->addWidget(view);

    view = new View("Top right view");
    view->view()->setScene(scene);
    h1Split QGraphicsView *View::view() const

    view = new View("Bottom left view");
    view->view()->setScene(scene);
    h2Splitter->addWidget(view);

    view = new View("Bottom right view");
    view->view()->setScene(scene);
    h2Splitter->addWidget(view);

    QHBoxLayout *layout = new QHBoxLayout;
    layout->addWidget(vSplitter);
    setLayout(layout);

    setWindowTitle(tr("Chip Example"));
}
```

```
Name                          Value
▶ [statics]
  layout                      0x12
  parent                      0x0
▼ this                        @0x
  ▶ [QWidget]                 @0x
  ▶ h1Splitter                0x10
  ▼ h2Splitter                0x10
    ▼ *h2Splitter             @0x
      ▼ [QFrame]              @0x
        ▼ [QWidget]           @0x
          ▶ [QObject]         @0x
          ▶ [QPaintDevice]    @0x
          ▶ data              0x10
  ▶ scene                     0x10
▶ vSplitter                   0x10
▼ view                        0x10
  ▼ *view                     @0x
    ▶ [QFrame]                @0x
    ▶ antialiasButton         0x10
    ▶ dragModeButton          0x10
    ▶ graphicsView            0x10
    ▶ label                   0x10
    ▶ label2                  0x10
    ▶ openGlButton            0x10
    ▶ printButton             0x10
    ▶ resetButton             0x10
    ▶ rotateSlider            0x10
    ▶ selectModeButton        0x10
```

Level	Function	File	Line
1	MainWindow...	mainwindow.cpp	83
2	MainWindow...	mainwindow.cpp	60
3	main	main.cpp	62
4	start		

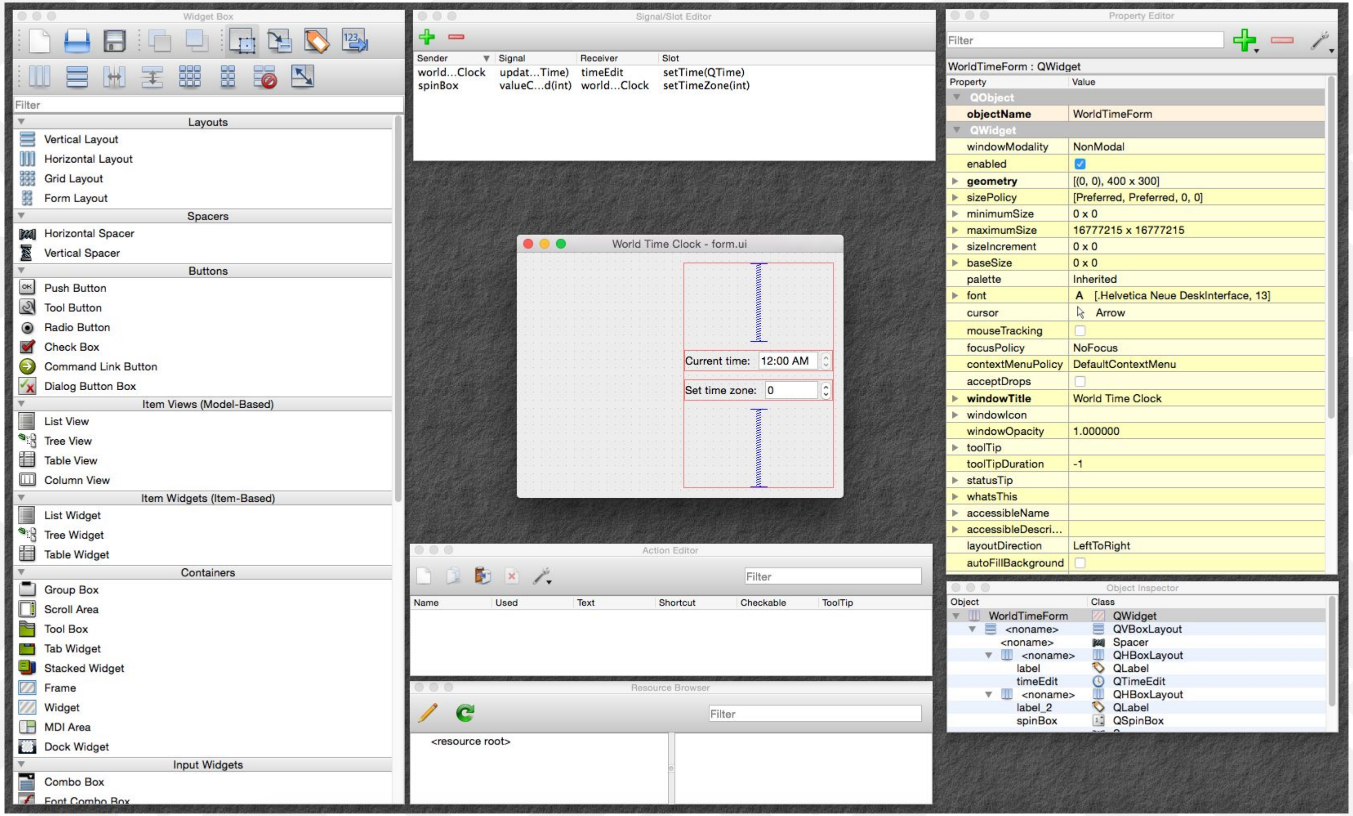
Number	Function	File	Line	Address
● 1	-	...nwindow.cpp	83	0x10000471e

```
>
```

Dev Tools: Tools integrated into Qt Creator

- **Qt Designer** is for designing and building graphical user interfaces from Qt widgets. You can compose and customize your widgets or dialogs in a visual editor, and test them using different styles and resolutions. You can access Qt Designer from Qt Creator in the Design mode.
- **qmake** is for building applications for different target platforms. There is also access to other build automation tools, such as CMake, Qbs, or Autotools. When using qmake or CMake, specify the build system in the Projects mode.
- **Qt Linguist** is for localizing applications. Qt Linguist contains tools for the roles typically involved in localizing applications: developers, translators, and release managers. lupdate and lrelease tools can be run from the Qt Creator Tools menu.
- **Qt Assistant** is for viewing Qt documentation. Documentation can also be viewed in Qt Creator. The documentation installed with Qt 5 is displayed automatically in the Help mode, and it is possible to add documents to the list.

Dev Tools: Qt Designer



Sender	Signal	Receiver	Slot
world...Clock	updat...Time)	timeEdit	setTime(QTime)
spinBox	valueC...d(int)	world...Clock	setTimeZone(int)

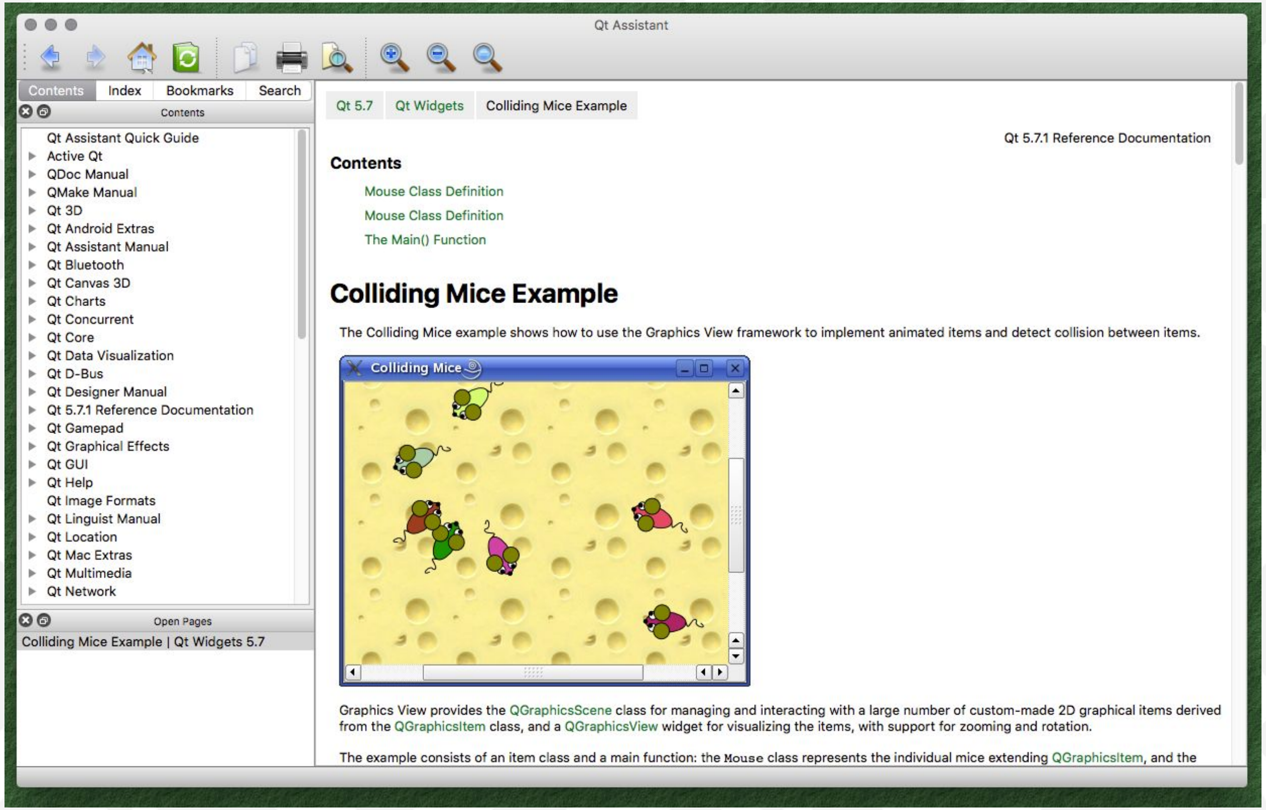
Name	Used	Text	Shortcut	Checkable	ToolTip

WorldTimeForm : QWidget

Property	Value
QObject	
objectName	WorldTimeForm
QWidget	
windowModality	NonModal
enabled	✓
geometry	[(0, 0), 400 x 300]
sizePolicy	[Preferred, Preferred, 0, 0]
minimumSize	0 x 0
maximumSize	16777215 x 16777215
sizeIncrement	0 x 0
baseSize	0 x 0
palette	Inherited
font	A [.Helvetica Neue DeskInterface, 13]
cursor	Arrow
mouseTracking	☐
focusPolicy	NoFocus
contextMenuPolicy	DefaultContextMenu
acceptDrops	☐
windowTitle	World Time Clock
windowIcon	
windowOpacity	1.000000
toolTip	
toolTipDuration	-1
statusTip	
whatsThis	
accessibleName	
accessibleDescri...	
layoutDirection	LeftToRight
autoFillBackground	☐

Object	Class
WorldTimeForm	QWidget
<noname>	QVBoxLayout
<noname>	Spacer
<noname>	QHBoxLayout
label	QLabel
timeEdit	QTimeEdit
<noname>	QHBoxLayout
label_2	QLabel
spinBox	QSpinBox

Dev Tools: Qt Assistant



Qt Assistant

Contents | Index | Bookmarks | Search

Contents

- Qt Assistant Quick Guide
- Active Qt
- QDoc Manual
- QMake Manual
- Qt 3D
- Qt Android Extras
- Qt Assistant Manual
- Qt Bluetooth
- Qt Canvas 3D
- Qt Charts
- Qt Concurrent
- Qt Core
- Qt Data Visualization
- Qt D-Bus
- Qt Designer Manual
- Qt 5.7.1 Reference Documentation
- Qt Gamepad
- Qt Graphical Effects
- Qt GUI
- Qt Help
- Qt Image Formats
- Qt Linguist Manual
- Qt Location
- Qt Mac Extras
- Qt Multimedia
- Qt Network

Open Pages

Colliding Mice Example | Qt Widgets 5.7

Qt 5.7 | Qt Widgets | Colliding Mice Example

Qt 5.7.1 Reference Documentation

Contents

- Mouse Class Definition
- Mouse Class Definition
- The Main() Function

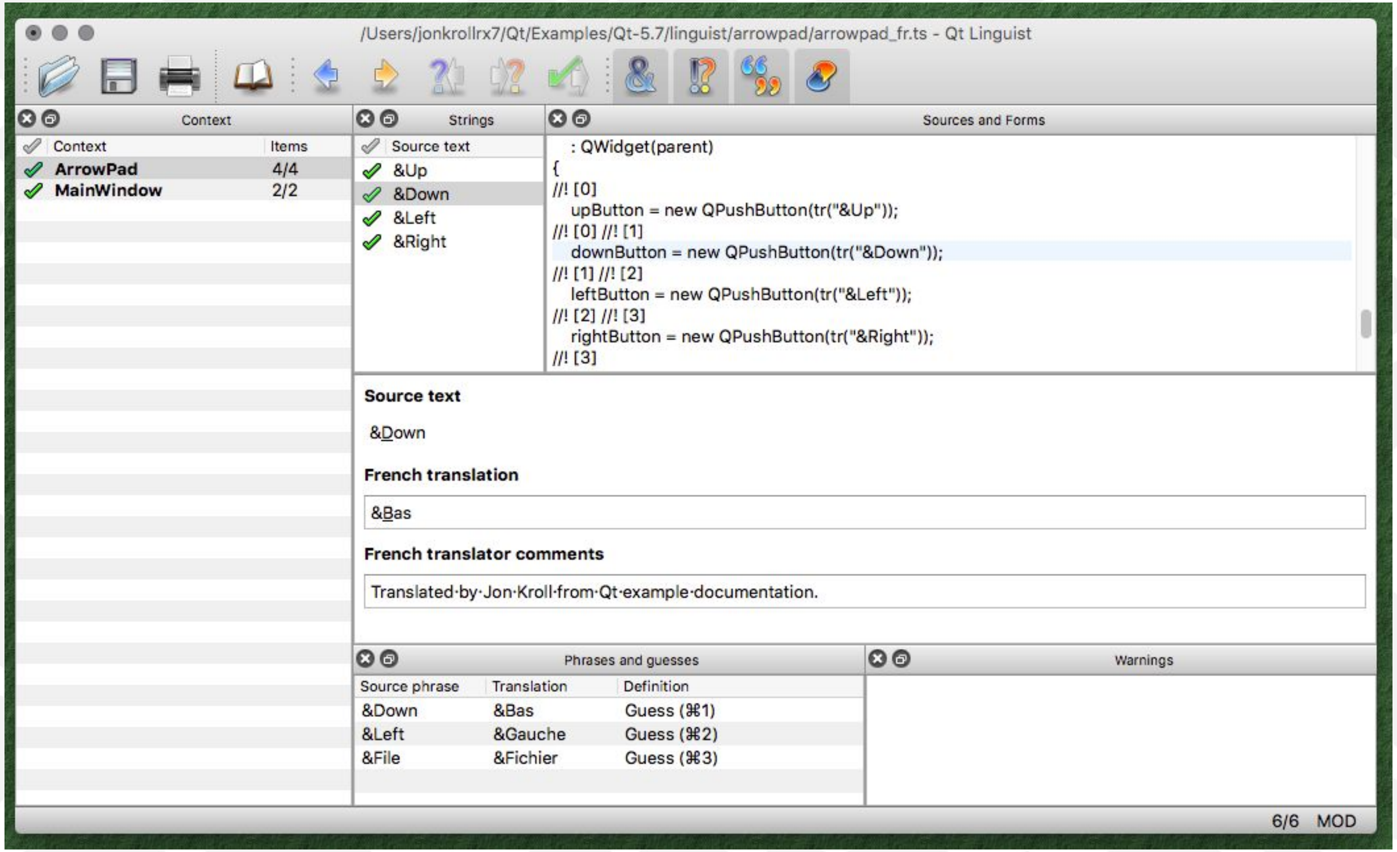
Colliding Mice Example

The Colliding Mice example shows how to use the Graphics View framework to implement animated items and detect collision between items.

Graphics View provides the QGraphicsScene class for managing and interacting with a large number of custom-made 2D graphical items derived from the QGraphicsItem class, and a QGraphicsView widget for visualizing the items, with support for zooming and rotation.

The example consists of an item class and a main function: the `Mouse` class represents the individual mice extending QGraphicsItem, and the

Dev Tools: Qt Linguist


/Users/jonkrollrx7/Qt/Examples/Qt-5.7/linguist/arrowpad/arrowpad_fr.ts - Qt Linguist
Context
Strings
Sources and Forms
Phrases and guesses
Warnings
6/6 MOD

Context	Items
✓ ArrowPad	4/4
✓ MainWindow	2/2

Source text
✓ &Up
✓ &Down
✓ &Left
✓ &Right

```
  : QWidget(parent)
{
//! [0]
  upButton = new QPushButton(tr("&Up"));
//! [0] //! [1]
  downButton = new QPushButton(tr("&Down"));
//! [1] //! [2]
  leftButton = new QPushButton(tr("&Left"));
//! [2] //! [3]
  rightButton = new QPushButton(tr("&Right"));
//! [3]
```

Source text

&Down

French translation

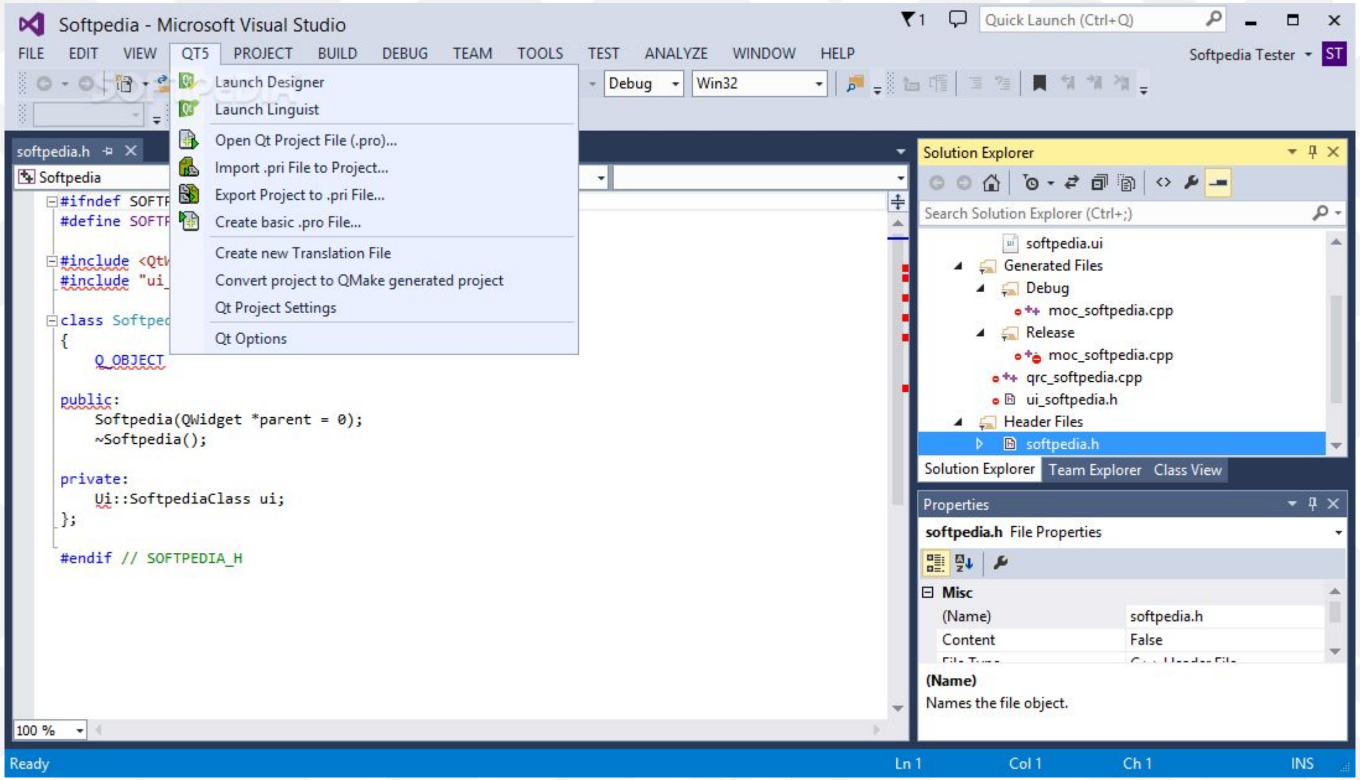
&Bas

French translator comments

Translated·by·Jon·Kroll·from·Qt·example·documentation.

Source phrase	Translation	Definition
&Down	&Bas	Guess (⌘1)
&Left	&Gauche	Guess (⌘2)
&File	&Fichier	Guess (⌘3)

Dev Tools: Qt Add-on for Visual Studio


Softpedia - Microsoft Visual Studio
FILE EDIT VIEW QT5 PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Launch Designer
Launch Linguist
Open Qt Project File (.pro)...
Import .pri File to Project...
Export Project to .pri File...
Create basic .pro File...
Create new Translation File
Convert project to QMake generated project
Qt Project Settings
Qt Options
Solution Explorer
softpedia.ui
Generated Files
Debug
moc_softpedia.cpp
Release
moc_softpedia.cpp
qrc_softpedia.cpp
ui_softpedia.h
Header Files
softpedia.h
Properties
Ready

```
#ifndef SOFTP
#define SOFTP

#include <QtW
#include "ui_

class Softped
{
    Q_OBJECT

public:
    Softpedia(QWidget *parent = 0);
    ~Softpedia();

private:
    Ui::SoftpediaClass ui;
};

#endif // SOFTPEDIA_H
```

softpedia.h File Properties

(Name)	softpedia.h
Content	False

(Name)
Names the file object.

Widget-based User Interfaces

Qt Widgets are traditional user interface elements that are typically found in desktop environments. The widgets integrate well to the underlying platform providing native look and feel on Windows, Linux, and Mac OS X. The widgets are mature and feature rich user interface elements suitable for mostly static user interfaces. In contrast to Qt Quick, the widgets do not scale that well for touch screens and fluid, highly animated modern user interfaces. The widgets are a good choice for applications with traditional desktop centric user interfaces, such as office type applications.

Widget-based UX: Important Concepts

- Application Main Window
- Desktop Integration
- Dialog Windows
- Layout Management
- Model/View Programming
- Rich Text Processing
- Drag and Drop

Widget-based UX: Important Concepts

- Application Main Window
  - Classes that provide everything you need for a typical modern main application window
  - QMainWindow is the central class around which applications can be built.
  - QMenuBar provides a horizontal menu bar
  - QDockWidget provides a widget that can be used to create detachable tool palettes or helper windows. Dock widgets keep track of their own properties, and they can be moved, closed, and floated as external windows.
  - QToolBar provides a generic toolbar widget that can hold a number of different action-related widgets, such as buttons, drop-down menus, comboboxes, and spin boxes. The emphasis on a unified action model in Qt means that toolbars cooperate well with menus and keyboard shortcuts.
  - QStatusBar provides a horizontal bar suitable for presenting status information

Widget-based UX: Important Concepts

- Desktop Integration
  - Qt applications behave well in the user's desktop environment, but certain integrations require additional, and sometimes platform specific, techniques. Various classes in Qt are designed to help developers integrate applications into users' desktop environments. These classes enable developers to take advantage of native services while still using a cross-platform API.
  - Desktop Services
  - System Tray Icons
  - Desktop Widgets

Widget-based UX: Important Concepts

- Desktop Integration
  - Qt applications behave well in the user's desktop environment, but certain integrations require additional, and sometimes platform specific, techniques. Various classes in Qt are designed to help developers integrate applications into users' desktop environments. These classes enable developers to take advantage of native services while still using a cross-platform API.
  - **Desktop Services**: QDesktopServices provides an interface to services offered by the user's desktop environment. In particular, the openUrl() function is used to open resources using the appropriate application, which may have been specifically configured by the user.
  - System Tray Icons
  - Desktop Widgets

Widget-based UX: Important Concepts

- Desktop Integration
  - Qt applications behave well in the user's desktop environment, but certain integrations require additional, and sometimes platform specific, techniques. Various classes in Qt are designed to help developers integrate applications into users' desktop environments. These classes enable developers to take advantage of native services while still using a cross-platform API.
  - Desktop Services
  - **System Tray Icons**: Many modern desktop environments feature docks or panels with system trays in which applications can install icons. Applications often use system tray icons to display status information, either by updating the icon itself or by showing information in balloon messages. Additionally, many applications provide pop-up menus that can be accessed via their system tray icons.
  - Desktop Widgets

Widget-based UX: Important Concepts

- Desktop Integration
  - Qt applications behave well in the user's desktop environment, but certain integrations require additional, and sometimes platform specific, techniques. Various classes in Qt are designed to help developers integrate applications into users' desktop environments. These classes enable developers to take advantage of native services while still using a cross-platform API.
  - Desktop Services
  - System Tray Icons
  - **Desktop Widgets**: On systems where the user's desktop is displayed using more than one screen, certain types of applications may need to obtain information about the configuration of the user's workspace to ensure that new windows and dialogs are opened in appropriate locations. The QDesktopWidget class can be used to monitor the positions of widgets and notify applications about changes to the way the desktop is split over the available screens.

Widget-based UX: Important Concepts

- Dialog Windows
  - Qt provides a simple API for modal and modeless dialogs.
  - Custom dialogs can be easily created by composing regular widgets into a QDialog.
  - Qt provides a set of ready-made dialogs
  - QColorDialog: a dialog for specifying colors
  - QFileDialog: a dialog that allow users to select files or directories
  - QFontDialog: a dialog for selecting a font
  - QInputDialog: a simple convenience dialog to get a single value from the user
  - QMessageBox: a modal dialog for informing the user or for prompting for confirmation
  - QProgressDialog: feedback on the progress of a slow operation

Widget-based UX: Important Concepts

- Layout Management
  - The Qt layout system provides a simple and powerful way of automatically arranging child widgets within a widget to ensure that they make good use of the available space.
  - All QWidget subclasses can use layouts to manage their children.
  - The easiest way to give your widgets a good layout is to use the built-in layout managers.

Widget-based UX: Important Concepts

- Layout Management
  - The Qt layout system provides a simple and powerful way of automatically arranging child widgets within a widget to ensure that they make good use of the available space. Qt includes a set of layout management classes that are used to describe how widgets are laid out in an application's user interface. These layouts automatically position and resize widgets when the amount of space available for them changes, ensuring that they are consistently arranged and that the user interface as a whole remains usable.
  - All QWidget subclasses can use layouts to manage their children.
  - The easiest way to give your widgets a good layout is to use the built-in layout managers.

Widget-based UX: Important Concepts

- Layout Management
    - The Qt layout system provides a simple and powerful way of automatically arranging child widgets within a widget to ensure that they make good use of the available space.
    - All QWidget subclasses can use layouts to manage their children. The QWidget::setLayout() function applies a layout to a widget. When a layout is set on a widget in this way, it takes charge of the following tasks: Positioning of child widgets, Sensible default sizes for windows, Sensible minimum sizes for windows, Resize handling, Automatic updates when contents change: Font size, text or other contents of child widgets, Hiding or showing a child widget, and Removal of child widgets.
    - The easiest way to give your widgets a good layout is to use the built-in layout managers.

Widget-based UX: Important Concepts

- Layout Management
    - The Qt layout system provides a simple and powerful way of automatically arranging child widgets within a widget to ensure that they make good use of the available space.
    - All QWidget subclasses can use layouts to manage their children.
    - The easiest way to give your widgets a good layout is to use the built-in layout managers: QHBoxLayout, QVBoxLayout, QGridLayout, and QFormLayout. These classes inherit from QLayout, which in turn derives from QObject (not QWidget). They take care of geometry management for a set of widgets. To create more complex layouts layout managers can be nested.

Widget-based UX: Important Concepts

- Model/View Programming
  - Qt contains a set of item view classes that use a model/view architecture to manage the relationship between data and the way it is presented to the user. The separation of functionality introduced by this architecture gives developers greater flexibility to customize the presentation of items, and provides a standard model interface to allow a wide range of data sources to be used with existing item views.

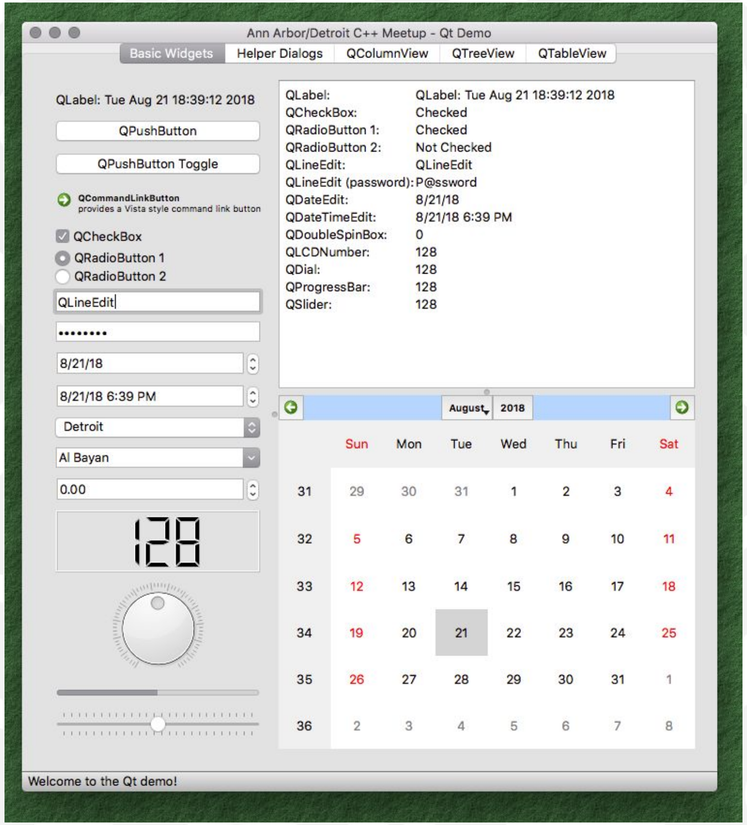
Widget-based UX: Important Concepts

- Rich Text Processing
  - The Scribe framework provides a set of classes for reading and manipulating structured rich text documents. Unlike previous rich text support in Qt, the new classes are centered around the QTextDocument class rather than raw textual information. This enables the developer to create and modify structured rich text documents without having to prepare content in an intermediate markup format.
  - The information within a document can be accessed via two complementary interfaces: A cursor-based interface is used for editing, and a read-only hierarchical interface provides a high level overview of the document structure. The main advantage of the cursor-based interface is that the text can be edited using operations that mimic a user's interaction with an editor, without losing the underlying structure of the document. The read-only hierarchical interface is most useful when performing operations such as searching and document export.

Widget-based UX: Important Concepts

- Drag and Drop
  - Drag and drop provides a simple visual mechanism which users can use to transfer information between and within applications.
  - Drag and drop operations are supported by many of Qt's controls, such as the item views and graphics view framework, as well as editing controls.
  - The following classes deal with drag and drop and the necessary mime type encoding and decoding: QDrag (*support for MIME-based drag and drop data transfer*), QDragEnterEvent, QDragLeaveEvent, QDragMoveEvent, and QDropEvent.

Widget-based UX: Demo



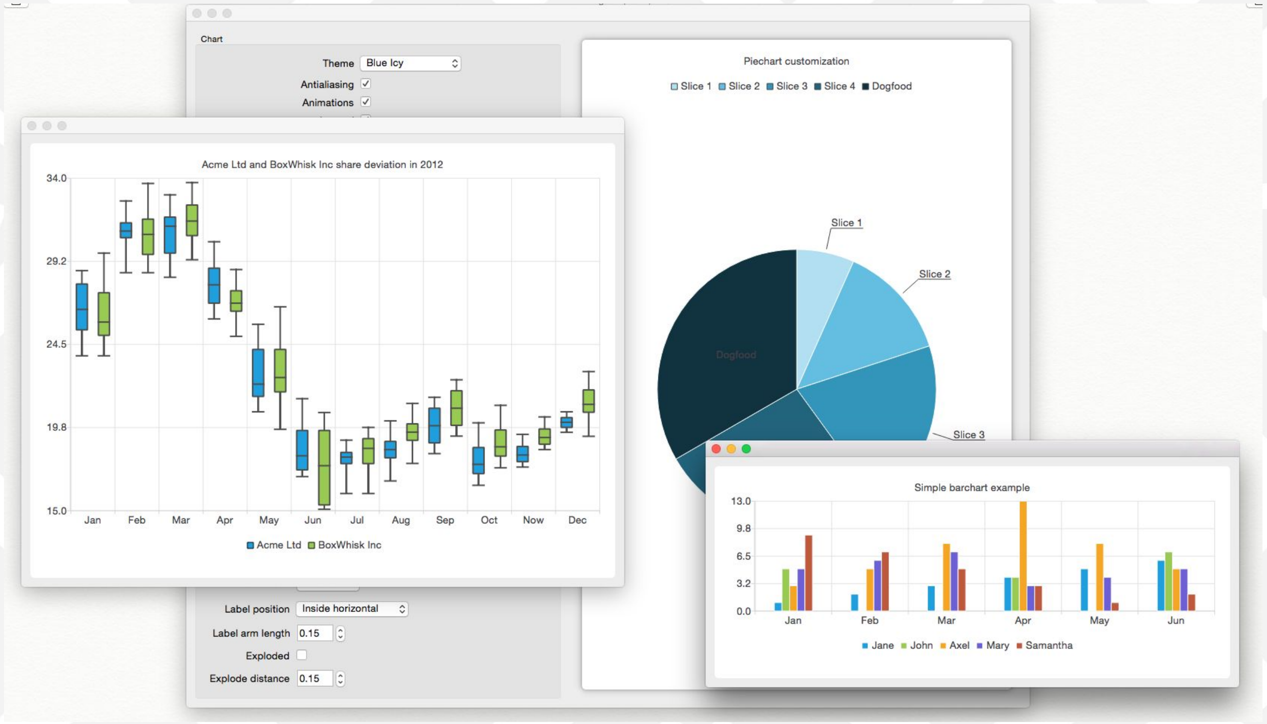
QLabel:	QLabel: Tue Aug 21 18:39:12 2018
QCheckBox:	Checked
QRadioButton 1:	Checked
QRadioButton 2:	Not Checked
QLineEdit:	QLineEdit
QLineEdit (password):	P@ssword
QDateEdit:	8/21/18
QDateTimeEdit:	8/21/18 6:39 PM
QDoubleSpinBox:	0
QLCDNumber:	128
QDial:	128
QProgressBar:	128
QSlider:	128

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	29	30	31	1	2	3	4
32	5	6	7	8	9	10	11
33	12	13	14	15	16	17	18
34	19	20	21	22	23	24	25
35	26	27	28	29	30	31	1
36	2	3	4	5	6	7	8

Widget-based UX: Visualizing Data

Qt provides ready-made C++ classes and QML types for visualizing data in the form of charts and graphs for analysis. Qt Charts and Qt Data Visualization are the two Qt add-ons that make data visualization using 2D and 3D models possible.

Widget-based UX: Visualizing Data Demo

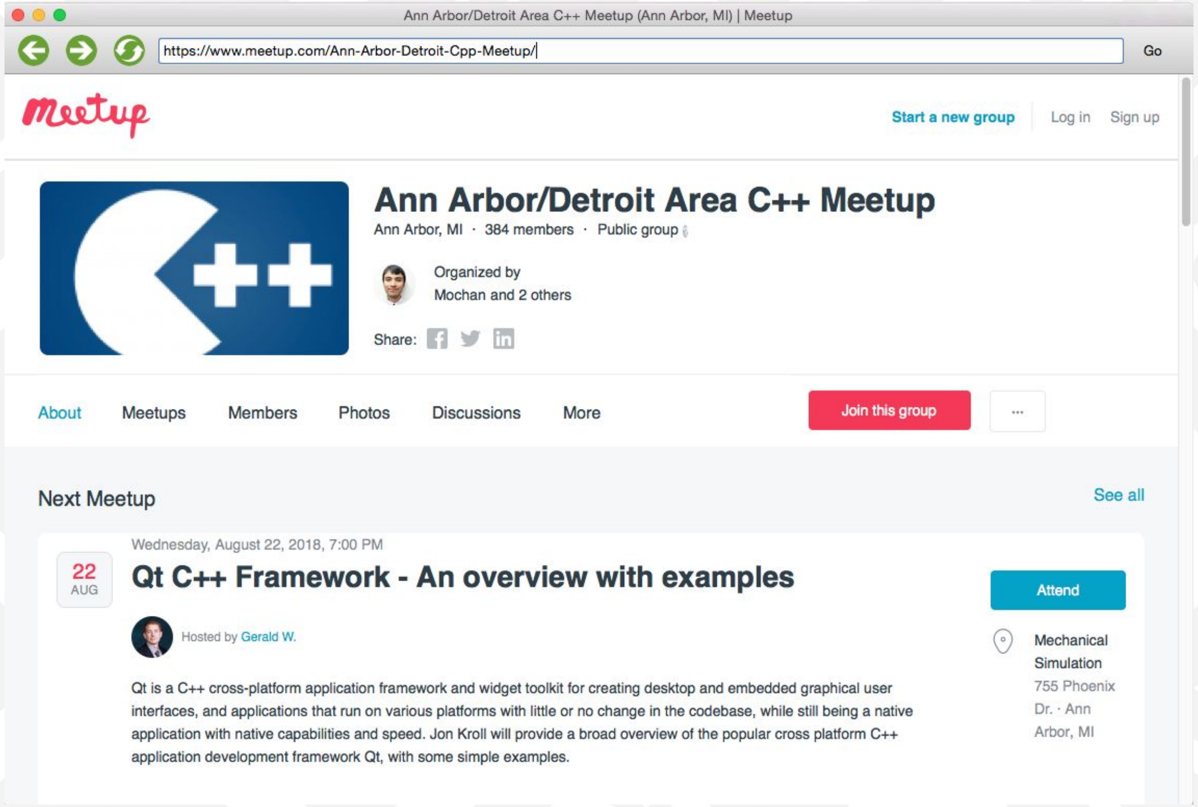


Widget-based UX: Displaying Web Content

Qt provides the Chromium-based WebEngine layout engine, which enables embedding web content into the Qt application. The engine can be integrated into both Qt Widget-based and Qt Quick-based applications.

Qt makes it easy to embed web content into a Qt application using features of the Qt WebEngine layout engine. The Qt WebEngine module equips Qt with support for a broad range of standard web technologies that make it possible to embed HTML content styled with CSS and scripted with JavaScript into a Qt application. Qt WebEngine facilitates both integration with traditional QWidget based desktop applications as well as embedding into Qt Quick QML applications.

Widget-based UX: Web Content Demo


Ann Arbor/Detroit Area C++ Meetup (Ann Arbor, MI) | Meetup
https://www.meetup.com/Ann-Arbor-Detroit-Cpp-Meetup/
Go
Meetup
Start a new group
Log in
Sign up

Ann Arbor/Detroit Area C++ Meetup

Ann Arbor, MI · 384 members · Public group

Organized by
Mochan and 2 others

Share:

About
Meetups
Members
Photos
Discussions
More
Join this group
...

Next Meetup

Wednesday, August 22, 2018, 7:00 PM

22 AUG

Qt C++ Framework - An overview with examples

Hosted by Gerald W.

Qt is a C++ cross-platform application framework and widget toolkit for creating desktop and embedded graphical user interfaces, and applications that run on various platforms with little or no change in the codebase, while still being a native application with native capabilities and speed. Jon Kroll will provide a broad overview of the popular cross platform C++ application development framework Qt, with some simple examples.

Mechanical Simulation
755 Phoenix Dr. · Ann Arbor, MI

Core Internals

Qt contains a rich set of fundamental enablers, mainly from the Qt Core module. Qt uses these enablers to provide higher-level UI and application development components. The following topics explain the most important enablers:

- Objects, Properties, Events
- Container Classes
- More Core Internals

Core Internals: Objects, Properties, Events

The QObject class forms the foundation of Qt's object model and is the parent class of many Qt classes. The object model introduces many mechanisms such as a meta-object system which allows run-time introspection, manipulation, and invocation of properties and methods in the object. It also serves as the basis for Qt's event system, which is a low-level way of communicating between QObject-based objects. Another high-level form of communication is provided in Qt's signals and slots mechanism.

Core Internals: Container Classes

Qt provides a set of general purpose, template-based container classes for structuring data in memory.

- Sequential containers
- Associative containers

Core Internals: Container Classes

Qt provides a set of general purpose, template-based container classes for structuring data in memory.

- Sequential containers
  - Qt provides the following sequential containers: QList, QLinkedList, QVector, QStack, and QQueue. For most applications, QList is the best type to use. Although it is implemented as an array-list, it provides very fast prepends and appends. If linked-list is required, use QLinkedList; for items to occupy consecutive memory locations, use QVector. QStack and QQueue are convenience classes that provide LIFO and FIFO semantics.
- Associative containers

Core Internals: Container Classes

Qt provides a set of general purpose, template-based container classes for structuring data in memory.

- Sequential containers
- Associative containers
    - Qt also provides these associative containers: QMap, QMultiMap, QHash, QMultiHash, and QSet. The "Multi" containers conveniently support multiple values associated with a single key. The "Hash" containers provide faster lookup by using a hash function instead of a binary search on a sorted set.
    - As special cases, the QCache and QContiguousCache classes provide efficient hash-lookup of objects in a limited cache storage.

Core Internals: More Core Internals

- Object Model
- The Meta-Object System
- The Property System
- The Event System
- Signals & Slots
- Differences between String-Based and Functor-Based Connections
- The State Machine Framework

More Core Internals: Object Model

The standard C++ object model provides very efficient runtime support for the object paradigm, but its static nature is inflexible in certain problem domains. Graphical user interface programming is a domain that requires both runtime efficiency and a high level of flexibility. Qt provides this, by combining the speed of C++ with the flexibility of the Qt Object Model.

More Core Internals: Object Model

The Qt object model adds these features to C++:

- A very powerful mechanism for seamless object communication called signals and slots
- Queryable and designable object properties
- Powerful events and event filters
- Contextual string translation for internationalization
- Sophisticated interval driven timers that make it possible to elegantly integrate many tasks in an event-driven GUI
- Hierarchical and queryable object trees that organize object ownership in a natural way
- Guarded pointers (QPointer) that are automatically set to 0 when the referenced object is destroyed, unlike normal C++ pointers which become dangling pointers when their objects are destroyed
- A dynamic cast that works across library boundaries.

More Core Internals: Meta-Object System

Qt's meta-object system provides the signals and slots mechanism for inter-object communication, run-time type information, and the dynamic property system.

The meta-object system is based on three things:

1. The QObject class provides a base class for objects that can take advantage of the meta-object system.
2. The Q_OBJECT macro inside the private section of the class declaration is used to enable meta-object features, such as dynamic properties, signals, and slots.
3. The Meta-Object Compiler (moc) supplies each QObject subclass with the necessary code to implement meta-object features.

More Core Internals: Meta-Object System

In addition to providing the signals and slots mechanism for communication between objects (the main reason for introducing the system), the meta-object code provides the following additional features:

- QObject::metaObject() returns the associated meta-object for the class.
- QMetaObject::className() returns the class name as a string at run-time, without requiring native run-time type information (RTTI) support through the C++ compiler.
- QObject::inherits() function returns whether an object is an instance of a class that inherits a specified class within the QObject inheritance tree.
- QObject::tr() and QObject::trUtf8() translate strings for internationalization.
- QObject::setProperty() and QObject::property() dynamically set and get properties by name.
- QMetaObject::newInstance() constructs a new instance of the class.

More Core Internals: Property System

Qt provides a sophisticated property system similar to the ones supplied by some compiler vendors. However, as a compiler- and platform-independent library, Qt does not rely on non-standard compiler features like __property or [property]. The Qt solution works with any standard C++ compiler on every platform Qt supports. It is based on the Meta-Object System that also provides inter-object communication via signals and slots.

More Core Internals: Property System

A property behaves like a class data member, but it has additional features accessible through the Meta-Object System.

- A READ accessor function
- A WRITE accessor function
- A MEMBER variable association
- A RESET function
- A NOTIFY signal
- A REVISION number
- The DESIGNABLE attribute

More Core Internals: Property System

A property behaves like a class data member, but it has additional features accessible through the Meta-Object System.

- A READ accessor function is required if no MEMBER variable was specified. It is for reading the property value. Ideally, a const function is used for this purpose, and it must return either the property's type or a const reference to that type.
- A WRITE accessor function
- A MEMBER variable association
- A RESET function
- A NOTIFY signal
- A REVISION number
- The DESIGNABLE attribute

More Core Internals: Property System

A property behaves like a class data member, but it has additional features accessible through the Meta-Object System.

- A READ accessor function
- A WRITE accessor function is optional. It is for setting the property value. It must return void and must take exactly one argument, either of the property's type or a pointer or reference to that type.
- A MEMBER variable association
- A RESET function
- A NOTIFY signal
- A REVISION number
- The DESIGNABLE attribute

More Core Internals: Property System

A property behaves like a class data member, but it has additional features accessible through the Meta-Object System.

- A READ accessor function
- A WRITE accessor function
- A MEMBER variable association is required if no READ accessor function is specified. This makes the given member variable readable and writable without the need of creating READ and WRITE accessor functions. It's still possible to use READ or WRITE accessor functions in addition to MEMBER variable association (but not both), if you need to control the variable access.
- A RESET function
- A NOTIFY signal
- A REVISION number
- The DESIGNABLE attribute

More Core Internals: Property System

A property behaves like a class data member, but it has additional features accessible through the Meta-Object System.

- A READ accessor function
- A WRITE accessor function
- A MEMBER variable association
- A RESET function is optional. It is for setting the property back to its context specific default value. The RESET function must return void and take no parameters.
- A NOTIFY signal
- A REVISION number
- The DESIGNABLE attribute

More Core Internals: Property System

A property behaves like a class data member, but it has additional features accessible through the Meta-Object System.

- A READ accessor function
- A WRITE accessor function
- A MEMBER variable association
- A RESET function
- A NOTIFY signal is optional. If defined, it should specify one existing signal in that class that is emitted whenever the value of the property changes. NOTIFY signals for MEMBER variables must take zero or one parameter, which must be of the same type as the property. The parameter will take the new value of the property.
- A REVISION number
- The DESIGNABLE attribute

More Core Internals: Property System

A property behaves like a class data member, but it has additional features accessible through the Meta-Object System.

- A READ accessor function
- A WRITE accessor function
- A MEMBER variable association
- A RESET function
- A NOTIFY signal
- A REVISION number is optional. If included, it defines the property and its notifier signal to be used in a particular revision of the API (usually for exposure to QML). If not included, it defaults to 0.
- The DESIGNABLE attribute

More Core Internals: Property System

A property behaves like a class data member, but it has additional features accessible through the Meta-Object System.

- A READ accessor function
- A WRITE accessor function
- A MEMBER variable association
- A RESET function
- A NOTIFY signal
- A REVISION number
- The DESIGNABLE attribute indicates whether the property should be visible in the property editor of GUI design tool (e.g., Qt Designer). Most properties are DESIGNABLE (default true). Instead of true or false, you can specify a boolean member function.

More Core Internals: Event System

In Qt, events are objects, derived from the abstract QEvent class, that represent things that have happened either within an application or as a result of outside activity that the application needs to know about. Events can be received and handled by any instance of a QObject subclass, but they are especially relevant to widgets.

More Core Internals: Signals & Slots

Signals and slots are used for communication between objects. The signals and slots mechanism is a central feature of Qt and probably the part that differs most from the features provided by other frameworks. Signals and slots are made possible by Qt's meta-object system.

In GUI programming, when we change one widget, we often want another widget to be notified. More generally, we want objects of any kind to be able to communicate with one another. For example, if a user clicks a Close button, we probably want the window's close() function to be called.

More Core Internals: Signals & Slots

- The signals and slots mechanism is type safe.
- Signals and slots are loosely coupled.
- All classes that inherit from QObject or one of its subclasses can contain signals and slots.
- Slots can be used for receiving signals, but they are also normal member functions.
- You can connect as many signals as you want to a single slot, and a signal can be connected to as many slots as you need.

More Core Internals: Signals & Slots

- **The signals and slots mechanism is type safe.** The signature of a signal must match the signature of the receiving slot. In fact, a slot may have a shorter signature than the signal it receives because it can ignore extra arguments.

- Signals and slots are loosely coupled.

- All classes that inherit from QObject or one of its subclasses can contain signals and slots.

- Slots can be used for receiving signals, but they are also normal member functions.

- You can connect as many signals as you want to a single slot, and a signal can be connected to as many slots as you need.

More Core Internals: Signals & Slots

- The signals and slots mechanism is type safe.

- **Signals and slots are loosely coupled.** A class which emits a signal neither knows nor cares which slots receive the signal. Qt's signals and slots mechanism ensures that if you connect a signal to a slot, the slot will be called with the signal's parameters at the right time. Signals and slots can take any number of arguments of any type. They are completely type safe.

- All classes that inherit from QObject or one of its subclasses can contain signals and slots.

- Slots can be used for receiving signals, but they are also normal member functions.

- You can connect as many signals as you want to a single slot, and a signal can be connected to as many slots as you need.

More Core Internals: Signals & Slots

- The signals and slots mechanism is type safe.
- Signals and slots are loosely coupled.
- **All classes that inherit from QObject or one of its subclasses (e.g., QWidget) can contain signals and slots**. Signals are emitted by objects when they change their state in a way that may be interesting to other objects. This is all the object does to communicate. It does not know or care whether anything is receiving the signals it emits. This is true information encapsulation, and ensures that the object can be used as a software component.
- Slots can be used for receiving signals, but they are also normal member functions.
- You can connect as many signals as you want to a single slot, and a signal can be connected to as many slots as you need.

More Core Internals: Signals & Slots

- The signals and slots mechanism is type safe.
- Signals and slots are loosely coupled.
- All classes that inherit from QObject or one of its subclasses can contain signals and slots.
- **Slots can be used for receiving signals, but they are also normal member functions.** Just as an object does not know if anything receives its signals, a slot does not know if it has any signals connected to it. This ensures that truly independent components can be created with Qt.
- You can connect as many signals as you want to a single slot, and a signal can be connected to as many slots as you need.

More Core Internals: Signals & Slots

- The signals and slots mechanism is type safe.
- Signals and slots are loosely coupled.
- All classes that inherit from QObject or one of its subclasses can contain signals and slots.
- Slots can be used for receiving signals, but they are also normal member functions.
- **You can connect as many signals as you want to a single slot, and a signal can be connected to as many slots as you need.** It is even possible to connect a signal directly to another signal. (*This will emit the second signal immediately whenever the first is emitted.*)

More Core Internals: Functor Connections

Differences between String-Based and Functor-Based Connections

From Qt 5.0 onwards, Qt offers two different ways to write signal-slot connections in C++: The string-based connection syntax and the functor-based connection syntax. There are pros and cons to both syntaxes.

More Core Internals: Functor Connections

	String Based	Functor Based
Type checking is done at...	Run-time	Compile-time
Can perform implicit type conversions		YES
Can connect signals to lambda expressions		YES
Can connect signals to slots which have more arguments than the signal (using default parameters)	YES	
Can connect C++ functions to QML functions	YES	
Selecting an instance of an overloaded signal or slot is...	Simple	Complex

More Core Internals: State Machine

The State Machine framework provides classes for creating and executing state graphs. The concepts and notation are based on those from Harel's Statecharts: A visual formalism for complex systems, which is also the basis of UML state diagrams. The semantics of state machine execution are based on State Chart XML (SCXML).

Statecharts provide a graphical way of modeling how a system reacts to stimuli. This is done by defining the possible states that the system can be in, and how the system can move from one state to another (transitions between states). A key characteristic of event-driven systems (such as Qt applications) is that behavior often depends not only on the last or current event, but also the events that preceded it. With statecharts, this information is easy to express.

More Core Internals: State Machine

The State Machine framework provides an API and execution model that can be used to effectively embed the elements and semantics of statecharts in Qt applications. The framework integrates tightly with Qt's meta-object system; for example, transitions between states can be triggered by signals, and states can be configured to set properties and invoke methods on {QObject}s. Qt's event system is used to drive the state machines.

The state graph in the State Machine framework is hierarchical. States can be nested inside of other states, and the current configuration of the state machine consists of the set of states which are currently active. All the states in a valid configuration of the state machine will have a common ancestor.

Internationalization and Localization

Qt provides built-in support for internationalization and localization of an application. Internationalization means designing an application so that it can be adapted to various languages and regions without engineering changes. Localization means adapting software for a specific region by adding locale-specific components such as dates, time, and number formats.

Qt uses Unicode for the encoding of displayable text strings. Unicode provides support for all commonly used writing systems in the world and is ideal for cross-platform development. Applications can also be written to support any number of different languages and regions with one code base.

Threading

Qt provides thread support in the form of platform-independent threading classes, a thread-safe way of posting events, and signal-slot connections across threads. This makes it easy to develop portable multithreaded Qt applications and take advantage of multiprocessor machines. Multithreaded programming is also a useful paradigm for performing time-consuming operations without freezing the user interface of an application.

Threading: Technologies in Qt

- QThread: Low-Level API with Optional Event Loops
- QThreadPool and QRunnable: Reusing Threads
- Qt Concurrent: Using a High-level API
- WorkerScript: Threading in QML

Threading Technologies: QThread

QThread is the foundation of all thread control in Qt. Each QThread instance represents and controls one thread.

QThread can either be instantiated directly or subclassed. Instantiating a QThread provides a parallel event loop, allowing QObject slots to be invoked in a secondary thread. Subclassing a QThread allows the application to initialize the new thread before starting its event loop, or to run parallel code without an event loop.

Threading Technologies: QThreadPool

Creating and destroying threads frequently can be expensive. To reduce this overhead, existing threads can be reused for new tasks. QThreadPool is a collection of reusable QThreads.

To run code in one of a QThreadPool's threads, reimplement QRunnable::run() and instantiate the subclassed QRunnable. Use QThreadPool::start() to put the QRunnable in the QThreadPool's run queue. When a thread becomes available, the code within QRunnable::run() will execute in that thread.

Each Qt application has a global thread pool. This global thread pool automatically maintains an optimal number of threads based on the number of cores in the CPU. However, a separate QThreadPool can be created and managed explicitly.

Threading Technologies: Qt Concurrent

The Qt Concurrent module provides high-level functions that deal with some common parallel computation patterns: map, filter, and reduce. Unlike using QThread and QRunnable, these functions never require the use of low-level threading primitives such as mutexes or semaphores. Instead, they return a QFuture object which can be used to retrieve the functions' results when they are ready. QFuture can also be used to query computation progress and to pause/resume/cancel the computation, and enables interactions with QFutures via signals and slots.

Qt Concurrent's map, filter and reduce algorithms automatically distribute computation across all available processor cores, so applications written today will continue to scale when deployed later on a system with more cores.

Threading Technologies: WorkerScript

The WorkerScript QML type lets JavaScript code run in parallel with the GUI thread.

Each WorkerScript instance can have one .js script attached to it. When WorkerScript::sendMessage() is called, the script will run in a separate thread (and a separate QML context). When the script finishes running, it can send a reply back to the GUI thread which will invoke the WorkerScript::onMessage() signal handler.

Using a WorkerScript is similar to using a worker QObject that has been moved to another thread. Data is transferred between threads via signals.

Data Storage

- Saving and Loading Data
- SQL Support
- XML Support
- JSON
- QSettings Class
- Resources
- File Archiving

Data Storage: Saving and Loading Data

The QIODevice class is the base class for all file and data storage devices in Qt Core. All classes that are used for reading and writing data inherit from it.

Examples of devices

- **QFile** is used for reading and writing text, binary files, and resources.
- The **QBuffer** class provides a QIODevice interface for a QByteArray.
- **QTcpSocket** enables the developer to establish a TCP connection and transfer streams of data.
- **QProcess** is used to start external programs, and to read from and write to that process.

Data Storage: SQL Support

The Qt SQL module uses driver plugins to communicate with several database APIs. Qt has drivers for SQLite, MySQL, DB2, Borland InterBase, Oracle, ODBC, and PostgreSQL. It is also possible to develop your own driver if Qt does not provide the driver needed.

Qt's SQL classes can be divided in 3 layers:

- **Driver layer** - Low-level communication between database and the SQL API layer
- **SQL API layer** - Provide access to databases
- **User Interface layer** - Link data from a database to data-aware widgets

Data Storage: SQL Support: MySQL

With the MySQL driver, it is possible to connect to a MySQL server. In order to build the QMYSQL Plugin for Unix or macOS, you need the MySQL header files as well as the shared library, libmysqlclient.so. To compile the plugin for Windows, install MySQL.

If using the embedded MySQL Server, a MySQL server is not needed in order to use that database system. In order to do so, link the Qt plugin to libmysqld instead of libmysqlclient.

Data Storage: SQL Support: SQLite

The Qt SQLite plugin is very suitable for local storage. SQLite is a relational database management system contained in a small (~350 KiB) C library. In contrast to other database management systems, SQLite is not a separate process that is accessed from the client application, but an integral part of it. SQLite operates on a single file, which must be set as the database name when opening a connection. If the file does not exist, SQLite will try to create it.

SQLite has some restrictions regarding multiple users and multiple transactions. If you are reading or writing on a file from different transactions, your application might freeze until one transaction commits or rolls back.

Data Storage: XML Support

Qt provides APIs to read and parse XML streams, and also to write to these streams.

QXmlStreamReader class provides a parser to read XML. It is a well-formed XML 1.0 parser that does not include external parsed entities. It is not CPU-intensive, as it doesn't store the entire XML document tree in memory. It only stores the current token at the time it is reported.

The **QXmlStreamWriter** class provides an XML writer with a simple streaming API. It is the counterpart to QXmlStreamReader for writing XML, and it operates on a QIODevice specified with setDevice(). It is a simple API that provides a dedicated function for every XML token or event you want to write.

Data Storage: JSON

JSON is a text-based open standard for data interchange that is easy to read and parse. It is used for representing simple data structures and associative arrays, called objects. It is related to JavaScript, but is a language-independent notation form.

Qt provides support for dealing with JSON data. JSON is a format to encode object data derived from Javascript, but now widely used as a data exchange format on the internet.

The JSON support in Qt provides an easy to use C++ API to parse, modify and save JSON data. It also contains support for saving this data in a binary format that is directly "mmap"-able and very fast to access.

Data Storage: QSettings Class

The QSettings class provides persistent storage of application settings. An application usually remembers its settings from the previous session.

Settings are stored differently on different platforms. For example, on Windows they are stored in the registry, whereas on macOS they are stored in XML files.

QSettings enable you to save and restore application settings in a portable manner. Constructing and destroying a QSettings object is lightweight and fast. While creating an object of QSettings, it is a good practice to specify not only the name of the application, but also the name of your organization.

For example: QSettings settings("MyCompany", "Accountancy");

Data Storage: Resources

The Qt Resource System is a platform-independent mechanism for storing binary files in the application's executable. This is handy if your application frequently needs a certain file, or set of files. It also protects against loss of that particular file.

Resource data can either be compiled into the binary and accessed immediately in the application code, or a binary resource can be created dynamically and registered with the resource system by the application.

By default, resources are accessible from the application code by the same file name as they are stored in the source tree, with a :/ prefix, or by a URL with a qrc scheme.

Data Storage: File Archiving

An archive file is a collection of files or directories which are generally compressed in order to reduce the space they would otherwise consume on a drive. Examples of archive files are ZIP, TAR, RAR and 7z.

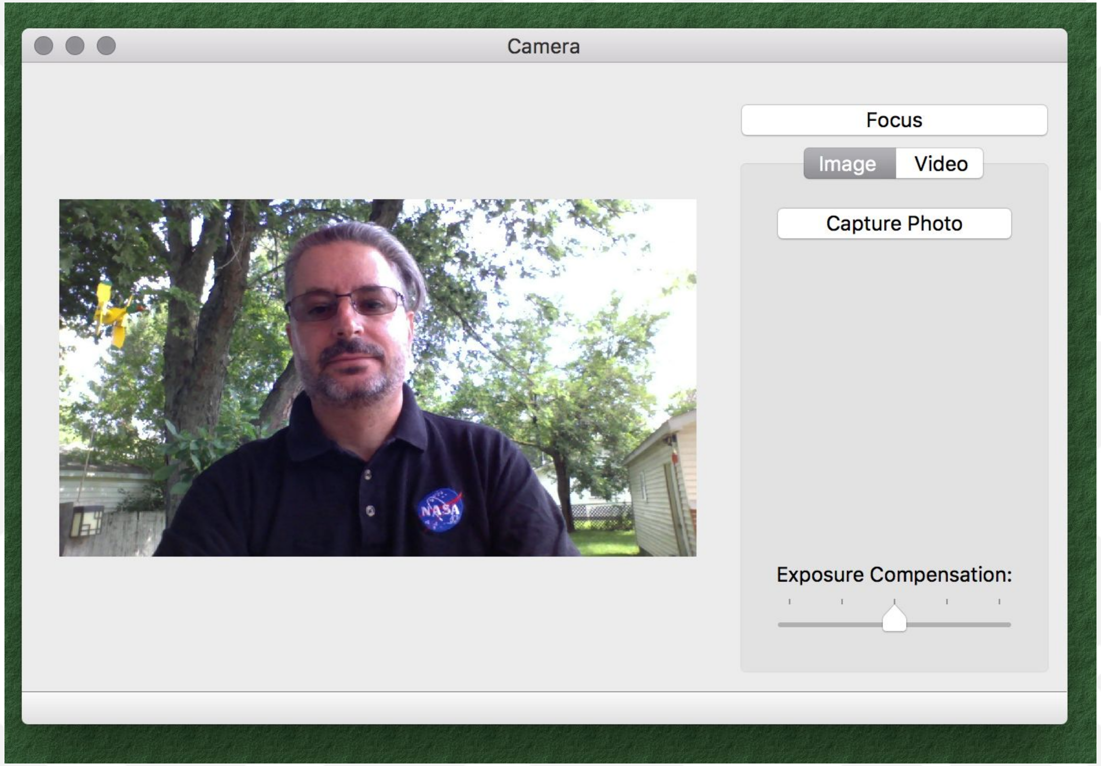
Qt has support for archives produced by zlib. See qCompress() and qUncompress().

Multimedia

Multimedia support in Qt is provided by the Qt Multimedia module. The Qt Multimedia module provides a rich feature set that enables you to easily take advantage of a platform's multimedia capabilities such as media playback and the use of camera and radio devices.

- Access raw audio devices for input and output
- Play low latency sound effects
- Play media files in playlists (such as compressed audio or video files)
- Record and compress audio
- Tune to radio stations
- Use a camera, viewfinder, image capture, and movie recording
- Play 3D positional audio with the Qt Audio Engine
- Decode audio media files into memory for processing
- Accessing video frames or audio buffers as played or recorded

Multimedia: Demo



Networking and Connectivity

Qt provides classes for both high-level and low-level network communication, classes for web integration, and classes for interprocess communication.

For high-level network traffic, Qt Network provides an abstraction layer over the operations used, showing only high-level classes and functions. Qt Network can also handle lower level protocols like TCP and UDP. Classes like QTcpSocket and QUdpSocket enable the developer to send and receive messages using the TCP or UDP protocol. Name resolution is done with QHostInfo. QHostInfo is called before creating a network connection with QTcpSocket or QUdpSocket. Filtering and redistributing network traffic via proxies can handled by the QNetWorkProxy class.

Networking: Network Communication

Qt supports a wide range of network communication, with a focus on HTTP, TCP, and UDP.

- HTTP level
- Socket level
- Secure via SSL
- For mobile

Networking: Network Communication: HTTP

Qt supports a wide range of network communication, with a focus on HTTP, TCP, and UDP.

- HTTP level
  - At the HTTP level, the Qt Network module offers the network access API, which consists mainly of QNetworkRequest, QNetworkAccessManager, and QNetworkReply. Put simply, the QNetworkRequest resembles an HTTP request, which gets passed to the QNetworkAccessManager to send the request on the wire; this class returns a QNetworkReply, which enables parsing the HTTP reply. The network access API uses the socket classes mentioned below (for TCP and SSL) internally.
- Socket level
- Secure via SSL
- For mobile

Networking: Network Communication: Socket

Qt supports a wide range of network communication, with a focus on HTTP, TCP, and UDP.

- HTTP level
- Socket level
  - For communication at the socket level, QTcpSocket, QUdpSocket and QSslSocket should be used. These classes offer a synchronous API through the waitFor* methods as well as an asynchronous API; if possible (i.e. if the event loop is running), the asynchronous API should always be preferred. Qt also offers QTcpServer to enable the server-side part of a TCP communication. Please note that there is no HTTP server class in Qt.
- Secure via SSL
- For mobile

Networking: Network Communication: Secure

Qt supports a wide range of network communication, with a focus on HTTP, TCP, and UDP.

- HTTP level
- Socket level
- Secure via SSL
    - For secure communication via SSL, Qt Network offers a wide range of classes alongside the central QSslSocket, e.g. QSslCertificate, QSslConfiguration and QSslError. The only supported backend for SSL in Qt is OpenSSL, which needs to be installed separately.
- For mobile

Networking: Network Communication: Mobile

Qt supports a wide range of network communication, with a focus on HTTP, TCP, and UDP.

- HTTP level
- Socket level
- Secure via SSL
- For mobile
  - For mobile devices, Qt Network offers the bearer management API to track the status of a connection (e.g. getting notified about online/offline status or whether Wifi or 3G is used).

Networking: WebSockets

An alternative to direct TCP or HTTP communication is to use the WebSocket protocol (RFC 6455). It is a two-way communication protocol on top of the TCP protocol to utilize existing web infrastructure without having to create additional client-server based communication. The Qt WebSockets module provides both a QML and C++ API, in addition to several examples to demonstrate its use.

Networking: Inter-Process Communication

Qt provides several ways to implement IPC in applications.

- Qt Network module
- Qt shared memory
- Qt D-Bus
- QProcess
- QLocalSocket

Networking: IPC: Qt Network module

Qt provides several ways to implement IPC in applications.

- Qt Network module
  - The cross-platform module provides classes that make network programming portable and easy. It offers high-level classes (e.g., QNetworkAccessManager, QFtp) that communicate using specific application-level protocols, and lower-level classes (e.g., QTcpSocket, QTcpServer, QSslSocket) for implementing protocols.
- Qt shared memory
- Qt D-Bus
- QProcess
- QLocalSocket

Networking: IPC: Qt Shared Memory

Qt provides several ways to implement IPC in applications.

- Qt Network module
- Qt shared memory
  - The cross-platform shared memory class, QSharedMemory, provides access to the operating system's shared memory implementation. It allows safe access to shared memory segments by multiple threads and processes. Additionally, QSystemSemaphore can be used to control access to resources shared by the system, as well as to communicate between processes.
- Qt D-Bus
- QProcess
- QLocalSocket

Networking: IPC: Qt D-Bus

Qt provides several ways to implement IPC in applications.

- Qt Network module
- Qt shared memory
- Qt D-Bus
    - The Qt D-Bus module is a cross-platform library you can use to implement IPC using the D-Bus protocol. It extends Qt's signals and slots mechanism to the IPC level, allowing a signal emitted by one process to be connected to a slot in another process.
- QProcess
- QLocalSocket

Networking: IPC: QProcess

Qt provides several ways to implement IPC in applications.

- Qt Network module
- Qt shared memory
- Qt D-Bus
- QProcess
  - The cross-platform class QProcess can be used to start external programs as child processes, and to communicate with them. It provides an API for monitoring and controlling the state of the child process. QProcess gives access to the input/output channels of child process by inheriting from QIODevice.
- QLocalSocket

Networking: IPC: QLocalSocket

Qt provides several ways to implement IPC in applications.

- Qt Network module
- Qt shared memory
- Qt D-Bus
- QProcess
- QLocalSocket
  - The QLocalSocket class provides a local socket. On Windows this is a named pipe and on Unix this is a local domain socket. The QLocalServer class provides a local socket based server. This class makes it possible to accept incoming local socket connections.

Networking: Serial Port Communication

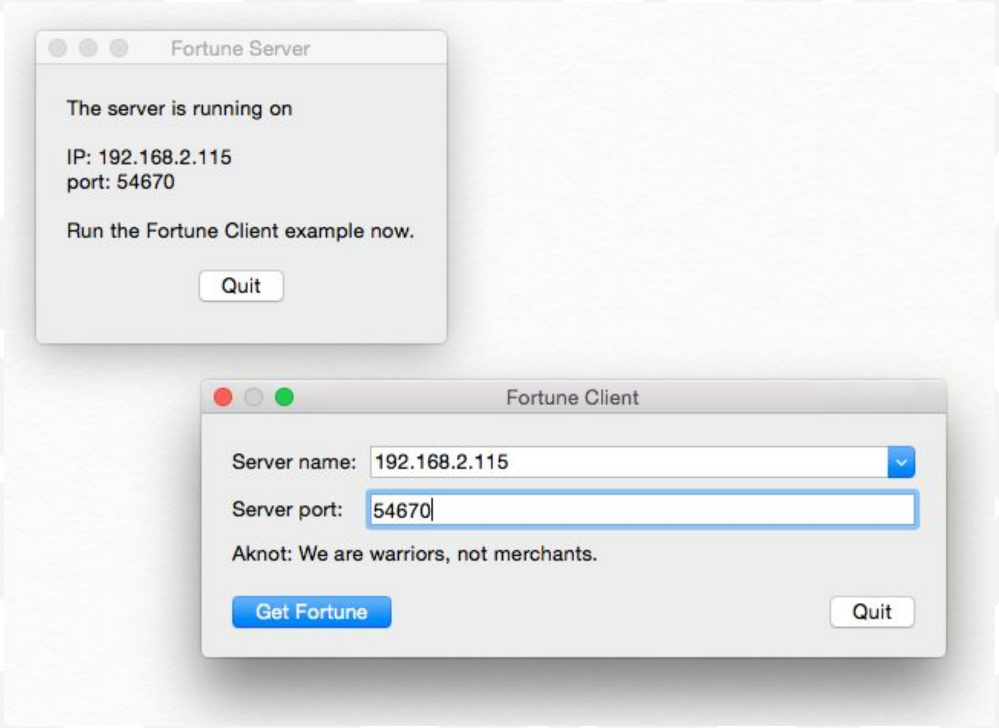
The Qt Serial Port module provides a C++ API for communicating through serial ports, using the RS-232 standard. It works with physical ports and also with drivers that emulate these ports. Examples of serial port emulators include virtual COM ports, com0com emulators, and the Bluetooth SPP.

This module is designed to make serial port programming easier and portable. As of Qt 5.2, it is supported on Windows, macOS, and Linux.

Networking: Bluetooth Communication

The Qt Bluetooth module provides both C++ and QML APIs for the short-range (less than 100 meters) wireless protocol developed by the Bluetooth Special Interest Group. It provides classic Bluetooth and Bluetooth Low Energy features.

Networking and Connectivity: Demo



The server is running on

IP: 192.168.2.115
port: 54670

Run the Fortune Client example now.

Server name: 192.168.2.115

Server port: 54670

Aknot: We are warriors, not merchants.

Graphics

Graphics in Qt 5 is primarily done either through the imperative QPainter API, or through Qt's declarative UI language, Qt Quick, and its scene graph back-end. Qt 5's graphics capabilities also includes support for printing, as well as the loading and saving of various image formats.

Graphics: 2D Graphics with QPainter

QPainter provides API for drawing vector graphics, text and images onto different surfaces, or QPaintDevice instances, such as QImage, QOpenGLPaintDevice, QWidget, and QPrinter. The actual drawing happens in the QPaintDevice's QPaintEngine. The software rasterizer and the OpenGL (ES) 2.0 back-ends are the two most important QPaintEngine implementations. The raster paint engine is Qt’s software rasterizer, and is used when drawing on a QImage or QWidget. Its strength over the OpenGL paint engine is its high quality when antialiasing is enabled, and a complete feature set.

Graphics: QPainter Rendering Targets

- **QImage** - A hardware-independent image representation with direct pixel access. QPainter will use the software rasterizer to draw to QImage instances.
- **QPixmap** - A image representation suited for display on screen. QPainter will primarily use the software rasterizer to draw to QPixmap instances.
- **QOpenGLPaintDevice** - A paint device to render to the current OpenGL (ES) 2.0 context. QPainter will use hardware accelerated OpenGL calls to draw to QOpenGLPaintDevice instances.
- **QBackingStore** - A backbuffer for top-level windows. QPainter will primarily use the software rasterizer to draw to QBackingStore instances.
- **QWidget** - A base class for pre-Qt Quick user interface classes. QPainter will render widgets using a QBackingStore.

Graphics: OpenGL

OpenGL is the most widely adopted graphics API for hardware accelerated and 3D graphics, implemented on all desktop platforms and almost every mobile and embedded platform. The Qt library contains a number of classes that help users integrate OpenGL into their applications.

- **QOpenGLWidget** is a widget that allows adding OpenGL scenes into QWidget-based user interfaces.
- Qt Canvas 3D - An add-on module that provides a way to make OpenGL-like 3D drawing calls from Qt Quick using JavaScript.

Graphics: Qt 3D

Qt 3D provides a fully configurable renderer that enables developers to quickly implement any rendering pipeline that they need. Further, Qt 3D provides a generic framework for near-realtime simulations beyond rendering.

Qt 3D is cleanly separated into a core and any number of aspects that can implement any functionality they wish. The aspects interact with components and entities to provide some slice of functionality. Examples of aspects include physics, audio, collision, artificial intelligence (AI), and path finding.

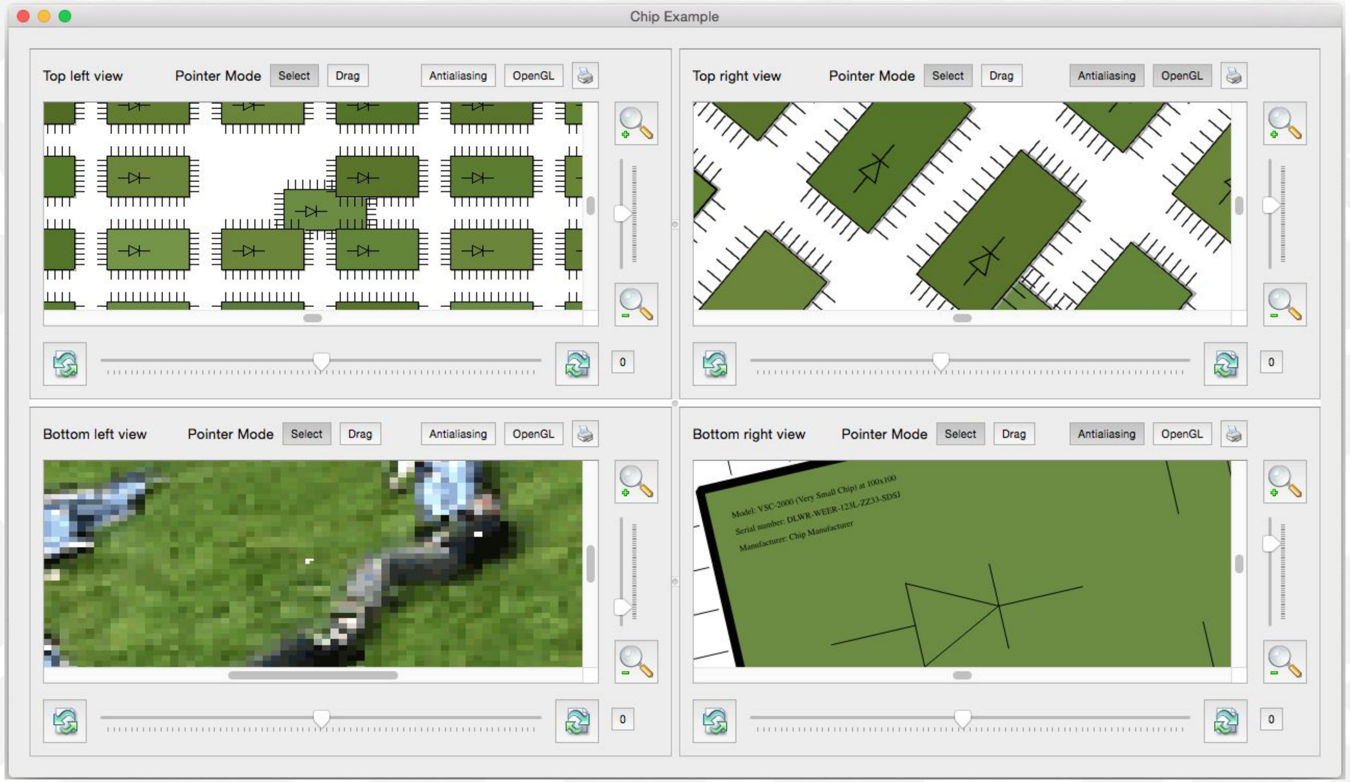
Graphics: Qt 3D

- 2D and 3D rendering for C++ and Qt Quick applications
- Meshes and Geometry
- Materials
- Shaders
- Shadow mapping
- Ambient occlusion
- High dynamic range
- Deferred rendering
- Multitexturing
- Instanced rendering
- Uniform Buffer Objects

Graphics: Images

Qt supports convenient reading, writing, and manipulating of images through the QImage class. In addition, for more fine grained control of how images are loaded or saved, you can use the QImageReader and QImageWriter classes respectively. To add support for additional image formats, outside of the ones provided by Qt, you can create image format plugins by using QImageIOHandler and QImageIOPlugin.

Graphics: Demo



Scripting

Qt has three main ways to help make an application scriptable. All of them allow easy integration of the ECMAScript (more widely known as JavaScript) language into the application. Depending on how deep the integration should be, one of these APIs can be used:

- Qt Script Module
- JS API
- QML

All of these three scripting solutions seamlessly inter-operate with the Meta-Object System, which means that all signals and slots and properties of a QObject instance are accessible in an ECMAScript environment.

Scripting: Qt Script Module

Qt Script is a separate module, designed with scripting applications in mind. It has a mature and rich API that permits a really deep integration of scripting functionality. It allows evaluating and debugging of scripts, and advanced use of objects and functions. It also gives access to a really low level ECMAScript engine API.

Scripting: JS API

This is a simple API, very similar to Qt Script, but limited to basic functionality. The main classes are QJSEngine and QJSValue, which can be used to embed pure ECMAScript functionality like evaluating scripts and calling functions.

Scripting: QML

QML exposes a modified ECMAScript environment. It is designed to mix well with the JavaScript API mentioned earlier. QML may be used without Qt Quick components, which may be useful for server side scripting. With QML it is possible to mix pure ECMAScript and declarative components.

Testing and Debugging

Qt provides various functionality to help develop high quality code. There are features that assist in debugging to track down bugs, and testing facilities that help to keep the bugs out.

Qt Test - a framework for unit tests of C++ code. Qt Test provides all the functionality commonly found in unit testing frameworks as well as extensions for testing graphical user interfaces.

Qt Quick Test - a framework for unit tests for QML applications

Testing and Debugging: Autotests

Qt itself has a significant number of unit tests written with the Qt Test and Qt Quick Test frameworks (autotests). These autotests are available for study in the tests/auto directories of the source code of each Qt module. The autotests are an important part of the continuous quality assurance in the development of Qt.

Testing and Debugging: with Qt Creator

Qt Creator's full debugging workflow supports debugging both C++ and QML code as well as the ability to profile code.

Qt Creator provides a debugger plugin that acts as an interface between the Qt Creator core and external native debuggers such as the GNU Symbolic Debugger (GDB), the Microsoft Console Debugger (CDB), a QML/JavaScript debugger, and the debugger of the low level virtual machine (LLVM) project, LLDB.

Accessibility

Accessibility in computer software is making applications usable for people with different abilities. It is important to take different people's needs into account, for example, in case of low vision, hearing, dexterity, or cognitive problems.

When we communicate with the assistive technologies, we need to describe Qt's user interface in a way that they can understand. Qt applications use QAccessibleInterface to expose information about the individual UI elements. Currently, Qt provides support for its widgets and widget parts, e.g., slider handles, but the interface could also be implemented for any QObject if necessary. QAccessible contains enums that describe the UI. The description is mainly based on MSAA and is independent of Qt.

Accessibility: Instances supported by Qt

- **Usability** - Usability and user centric design generally lead to more usable applications, including improvements for people with various abilities.
- **Fonts** - Font settings should follow the system/platform. This allows users to select fonts for readability and increasing the font size.
- **Colors** - Provide enough contrast and consider the most common cases of low vision and color blindness. Make sure that the application is usable, for example, for people with red/green blindness, and don't depend on colors only.
- **Scalable UI** - A user interface that works in various sizes and properly supports different fonts and accommodates size changes.
- **Sounds** - Do not exclusively rely on sound notifications, provide a visual alternative when a sound signal is imperative to using the application.
- **Assistive Technology** - Support the use of assistive tools (AT). Either use standard widgets/controls which support ATs out of the box, or make sure that your custom widgets and controls support accessibility properly. In order to learn more about this read on below.

Other cool stuff in Qt

- Regular Expressions
- QString
- Qt Positioning
- Qt Purchasing
- Qt Print Support

Other cool stuff in Qt: Regular Expressions

- Regular Expressions
  - The QRegularExpression and QRegExp classes provide pattern matching using regular expressions.
  - Uses: Validation, Searching, Search and Replace, String Splitting
  - QRegularExpression implements Perl-compatible regular expressions
  - Unicode is fully supported
- QString
- Qt Positioning
- Qt Purchasing
- Qt Print Support

Other cool stuff in Qt: QString

- Regular Expressions
- QString
  - The QString class provides a Unicode character string.
  - QString uses implicit sharing (*copy-on-write*) to reduce memory usage and avoid the needless copying of data.
  - String modification: append(), prepend(), insert(), replace(), remove(), trimmed(), etc.
  - Querying string data: startsWith(), endsWith(), contains(), count(), etc.
  - Conversion: toUtf8(), toLatin1(), toLocal8Bit(), toStdString(), etc.
  - Distinction Between Null and Empty Strings
  - Argument Formats
- Qt Positioning
- Qt Purchasing
- Qt Print Support

Other cool stuff in Qt: Qt Positioning

- Regular Expressions
- QString
- Qt Positioning
  - The Qt Positioning API gives developers the ability to determine a position by using a variety of possible sources, including satellite, WiFi, text file, etc. That information can then be used to, for example, determine a position on a map.
  - In addition satellite information can be retrieved and area based monitoring can be performed.
- Qt Purchasing
- Qt Print Support

Other cool stuff in Qt: Purchasing

- Regular Expressions
- QString
- Qt Positioning
- Qt Purchasing
  - Qt Purchasing is an add-on library that enables Qt apps to support in-app purchases. It currently supports purchases made to the Mac App Store on OS X, App Store on iOS, and Google Play on Android.
  - In-app purchases are a way to monetize an application. These purchases are made from inside the application and can include anything from unlocking content to virtual items. The Qt Purchasing API is built on the system APIs for in-app purchases, which means the purchase process is more familiar to the user, and the information already stored by the platform can be used to simplify the purchase process.
- Qt Print Support

Other cool stuff in Qt: Print Support

- Regular Expressions
- QString
- Qt Positioning
- Qt Purchasing
- Qt Print Support
  - Qt provides extensive cross-platform support for printing. Using the printing systems on each platform, Qt applications can print to attached printers and across networks to remote printers. Qt's printing system also supports PDF file generation, providing the foundation for basic report generation facilities.

Some Organizations using Qt *(from Wikipedia.org)*

- AMD
- Blizzard Entertainment
- Electronic Arts
- European Space Agency
- DreamWorks
- Danaher Corporation
- John Deere
- Lucasfilm
- Luxoft
- Microsoft
- Panasonic
- Philips
- Robert Bosch GmbH
- Samsung
- Siemens
- Volvo
- German Air Traffic Control
- HP
- Walt Disney Animation Studios
- Valve Corporation

Some cross-platform apps using Qt *(from wikipedia.org)*

- Adobe Photoshop Album
- Adobe Photoshop Elements
- AMD's Radeon Software Crimson Edition
- Autodesk Maya
- Bitcoin Core
- Bitcoin ABC
- CryEngine V editor
- Dragonframe
- EAGLE by CadSoft Computer / Autodesk
- FreeMat
- Google Earth
- Krita
- Mathematica
- Orange
- QGIS
- Scribus
- Sibelius
- Microsoft Skype
- Source 2
- Stellarium
- Subsurface
- Teamviewer
- Telegram
- VirtualBox
- VLC media player
- WPS Office
- XnView MP

Resources on the Web

The Qt homepage:	https://qt.io
Developer portal:	https://www.qt.io/developers/
Qt blog:	https://blog.qt.io/
Download Qt:	https://www.qt.io/download
About Qt Company:	https://www.qt.io/company
About Qt:	https://www.qt.io/what-is-qt/
Qt professional services:	https://www.qt.io/find-a-qt-advisor/
Licensing info:	https://www.qt.io/licensing
Upcoming webinars:	https://resources.qt.io/upcoming-webinars
Qt whitepapers:	https://resources.qt.io/whitepaper

Books

- Mastering Qt 5: Create stunning cross-platform applications
  (2016-December) by Guillaume Lazar, Robin Penea
- Learn Qt 5: Build modern, responsive cross-platform desktop applications with Qt, C++, and QML
  (2018-February) by Nicholas Sherriff
- Game Programming Using Qt
  (2016-January) by Witold Wysota, Lorenz Haas
- Qt 5 Projects: Develop cross-platform applications with modern UIs using the powerful Qt framework
  (2018-February) by Marco Piccolino
- Qt 5 C++ GUI Programming Cookbook
  (2016-July) by Lee Zhi Eng

The End

Code less.
Create more.
Deploy everywhere.