The Qt SDK

Matt Ripley
CSCI 5828
3/12/12

Executive Summary

- Qt is one of the leading GUI toolkits out there.
  - Great cross platform support (Linux, Windows, Mac)
- Allows for rapid development of tools and other native applications
  - Code less. Create More. Deploy Everywhere
- Extremely well tested and mature API
  - Full replacement for the STL
  - Pretty much anything you could ever want
- Extremely well optimized
- Built in concurrency framework
  - No more pthreads in C++!
- Cool Screencast on how to develop a basic application at the end of presentation!

Background

- What is Qt?
  - Qt is a cross platform application development framework
  - Evolved into a leading SDK for developing native applications
- Originally only a GUI toolkit.
- Has been extended to include support for nearly everything (GUI, STL replacement, OpenGL bindings, Sound support, DB support)

Background (2)

- The native app is dead, long live the native app.
- Who’s using Qt
  - Autodesk – Maya and other applications
  - Adobe – new versions of Photoshop and Creative Suite
  - VLC media player
  - Virtual Box
  - Skype
  - Google
  - Mathematica
  - KDE
  - Panasonic, Philips, Samsung, Volvo.

History

- Development started in 1991 at “Quasar Technologies”
- Company was renamed to Trolltech in 1994
- Named Qt because the founders liked the look of the letter Q in Emacs. T because original versions based off of Xt toolkit
- Started as a small GUI toolkit to compete with Xt and GTK.

History (2)

- Originally built as a Unix/X11 or Windows based SDK
  - Early Qt versions were closed source.
- In 1998 became the primary SDK for the KDE desktop environment
- Published under the GPL starting in 2000
- Mac OSX support was added in 2001 with Qt 3.0.

History (3)

- Support was added for embedded devices early 2010
  - MeeGo
  - Symbian OS
  - Windows CE
  - Wayland
- Extremely popular in the non apple non android smart phone market.

History (3)

- Open sourced mac version with Qt 3.2.
- Qt 4.0 released in 2005.
- Acquired by Nokia in 2008.
- Added LGPL support in 2009 to appeal to developers writing closed source applications.
- Source code now hosted on Gitorious for better community involvement
- Qt Labs provides cool cutting edge advancements

History (4)

- Recent advancements include language bindings for most popular languages
  - Java, Python, Scheme, Ruby, D.
  - http://en.wikipedia.org/wiki/Qt_(framework)#Bindings
- Qt has its own scripting language called QML
  - Based on java script
  - Designed for rapid tool development
  - Outside the scope of this talk

Qt Feature Set

- "… is big. Really big. You just won't believe how vastly, hugely, mindbogglingly big it is."
  - Douglas Adams, The Hitchhiker's Guide to the Galaxy.
- If you can think of it, Qt probably has support for it.
- Very "Java" like interfaces and conventions.
- You may be concerned about the size of Qt but …

Qt Feature Set (2)

Qt Feature Set (3)

- Qt is highly modularized.
- Designed with best software practices in mind
  - Design patterns
  - Cross platform
  - Optimized and well tested.
- Qt feature set as of 4.8:

Qt Feature Set (4)

- QtCore
  - STL replacement – fully STL compatible replacement including algorithms and container classes. More Java like then C++ like.
  - File System support – natively interfaces with systems file system
  - Concurrency frame work. Threads, thread pools, locks, barriers etc….
  - Basic signal / slot mechanism
  - Provides support for history and persistent user settings
- QtGui
  - All the standard widgets you'd expect from a GUI tool kit
  - Full signal / slot implementation
  - QtDesigner support
  - Interface for mouse and keyboard interaction
  - Support for printers and external display devices

Qt Feature Set (5)

- QtMultimedia
  - Support for video and audio
  - Full GPU support for video decoding
- QtNetwork
  - Support for network programming.
  - Cross platform socket layer
    - QSocket: is either winsock on windows or unix sockets
  - HTTP and FTP support
  - Full Web browser using webkit
  - SSL and encryption
- QtOpenGL
  - Full OpenGL bindings. Tuned for OpenGL > 3.x
  - Includes great support for shaders and FBO’s

Qt Feature Set (6)

- QtOpenVG
  - Support for vector graphics
- QtScript
  - Full support for the QML scripting language
- QtSQL
  - Data base tools for interacting with a SQL database
- QtSVG
  - Support for SVG file format
- QtWebKit
  - Web browser and HTML rendering engine
- QtXml
  - Handling XML content
  - Read and write XML files
  - DOM support
- QT Phonon

Qt Feature Set (8)

- Extra programs to aid developers
- QtCreator: A full IDE for developing Qt applications.
- QtCreator is made up of several programs
  - Qt Designer: A WYSIWYG GUI editor
  - Qt Assistant: Full documentation for the Qt SDK
  - GUI signal and slots editor
  - QML scripting
  - UIC - User interface compiler
  - MOC - meta object compiler
  - QMake – Qt make file generator.

Scope

- Qt is HUGE. Far beyond the scope of this talk.
- In this presentation we will cover
  - Basic Qt applications
  - Building a Qt Application
  - Designing a GUI in Qt
  - Signals and Slots
  - Qt concurrency framework.
    - Relevant to this class

Scope (2)

- Learning Qt is complicated and can't be easily linearized into a power point.
- But to understand best practices you have to understand a bit about the library.
- But to understand the library you need to know about the best practices.
- Understanding the Qt build tools requires understanding the best practices and the library

Basic Qt Application

- Most Basic “Hello World” Application
- QApplication provides needed services for Qt development
    - Signals and slots
    - Message loops
    - Other internal mechanisms
- Qlabel is a text widget
- All widgets have the ability to be considered a window.
- App.exec starts message loop.

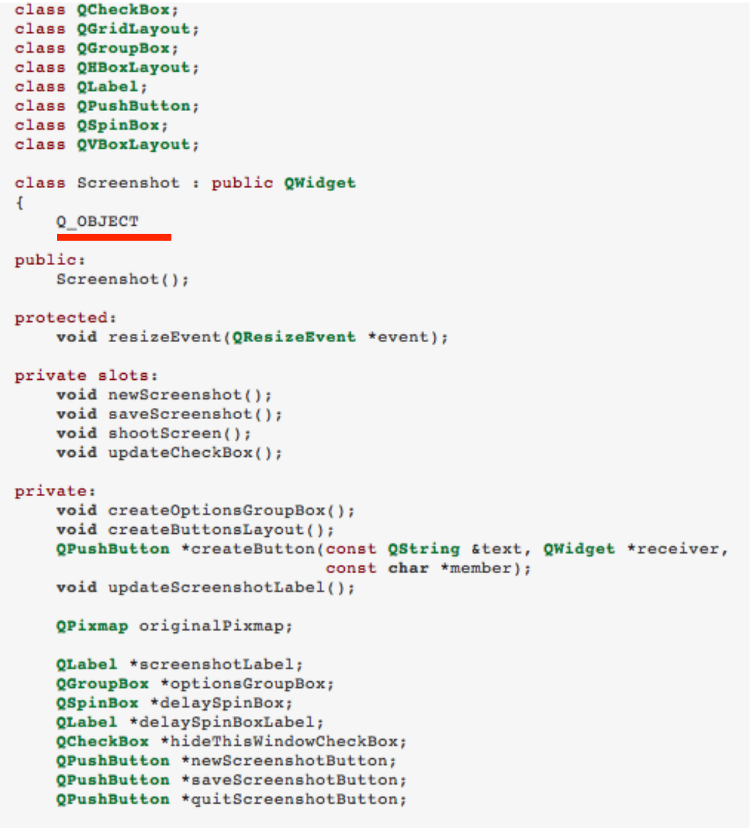
```
#include <QtGui>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel label("Hello, world!");
    label.show();
    return app.exec();
}
```

Signals and Slots

- Message passing handled by “Signals and Slots”
- Signals / slots implemented by extending the C++ language with new keywords
    - <public | private | protected> signals:
    - <public | private | protected> slots:
- Extensions handled by “MOC” the Meta Object Compiler.
- All Qt objects using signals and slots must declare the Q_OBJECT macro

Signals and Slots (2)

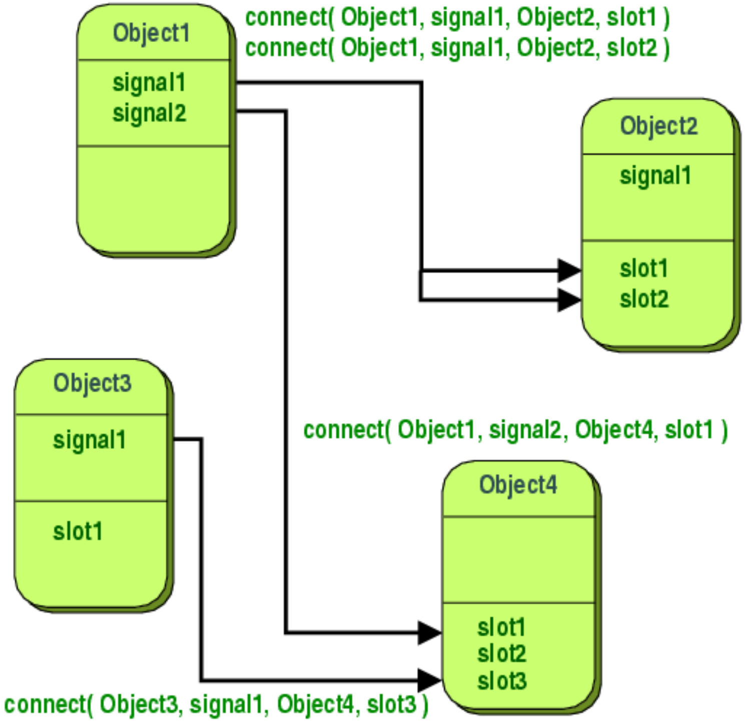
- All Qt objects that declare Q_OBJECT can declare signals and slots
  - Signals / slots are really just functions.
  - Under the hood signal slots connections are really just special call backs
- Special keywords only needed during declaration.



```
class QCheckBox;
class QGridLayout;
class QGroupBox;
class QHBoxLayout;
class QLabel;
class QPushButton;
class QSpinBox;
class QVBoxLayout;

class Screenshot : public QWidget
{
    Q_OBJECT

public:
    Screenshot();

protected:
    void resizeEvent(QResizeEvent *event);

private slots:
    void newScreenshot();
    void saveScreenshot();
    void shootScreen();
    void updateCheckBox();

private:
    void createOptionsGroupBox();
    void createButtonsLayout();
    QPushButton *createButton(const QString &text, QWidget *receiver,
                              const char *member);
    void updateScreenshotLabel();

    QPixmap originalPixmap;

    QLabel *screenshotLabel;
    QGroupBox *optionsGroupBox;
    QSpinBox *delaySpinBox;
    QLabel *delaySpinBoxLabel;
    QCheckBox *hideThisWindowCheckBox;
    QPushButton *newScreenshotButton;
    QPushButton *saveScreenshotButton;
    QPushButton *quitScreenshotButton;
```

Connecting Signals with Slots

- Any 2 Qt objects can be connected with the "connect()" macro
- Ex.
  - connect(ui->AddModelButton, SIGNAL(clicked()), this, SLOT(addNewModels()));
- Connect function breaks down as follows:
  - Connect(sender, signal, receiver, slot)


Object1
signal1
signal2
connect(Object1, signal1, Object2, slot1)
connect(Object1, signal1, Object2, slot2)
Object2
signal1
slot1
slot2
Object3
signal1
slot1
connect(Object1, signal2, Object4, slot1)
Object4
slot1
slot2
slot3
connect(Object3, signal1, Object4, slot3)

Connecting Signals with Slots (2)

- Just being able to call callback functions isn't super useful.

- Signals and slots can also pass objects between sender and receiver.

- Ex:
    - connect(ui->modelList, SIGNAL(itemClicked(QListWidgetItem*)), this, SLOT(newModelClicked(QListWidgetItem*)));

- In the above example a QListWidgetItem is passed to the slot

Connecting Signals with Slots (3)

- Signals are emitted with the "emit" keyword.
- The emit keyword is blocking
- Execution continues after the code in the connect slot completes
- Slots do not block the GUI. If 2 or more signals are emitted at the same time then the slots are queued and will execute in order of delivery.

Connecting Signals with Slots (4)

- The signal slots mechanism is slightly slower than traditional call backs but the simplicity is worth it
- Qt says that you can issue 2,000,000 signals to 1 receiver per second or around 1,200,000 signals to 2 receivers per second.

(less) Basic Qt Applications

- As the complexity of an app grows doing everything programmatically becomes tiresome.
- Leverage QtCreator to help with code completion and UI design.
- Compile static resources (icons, strings) into the application.

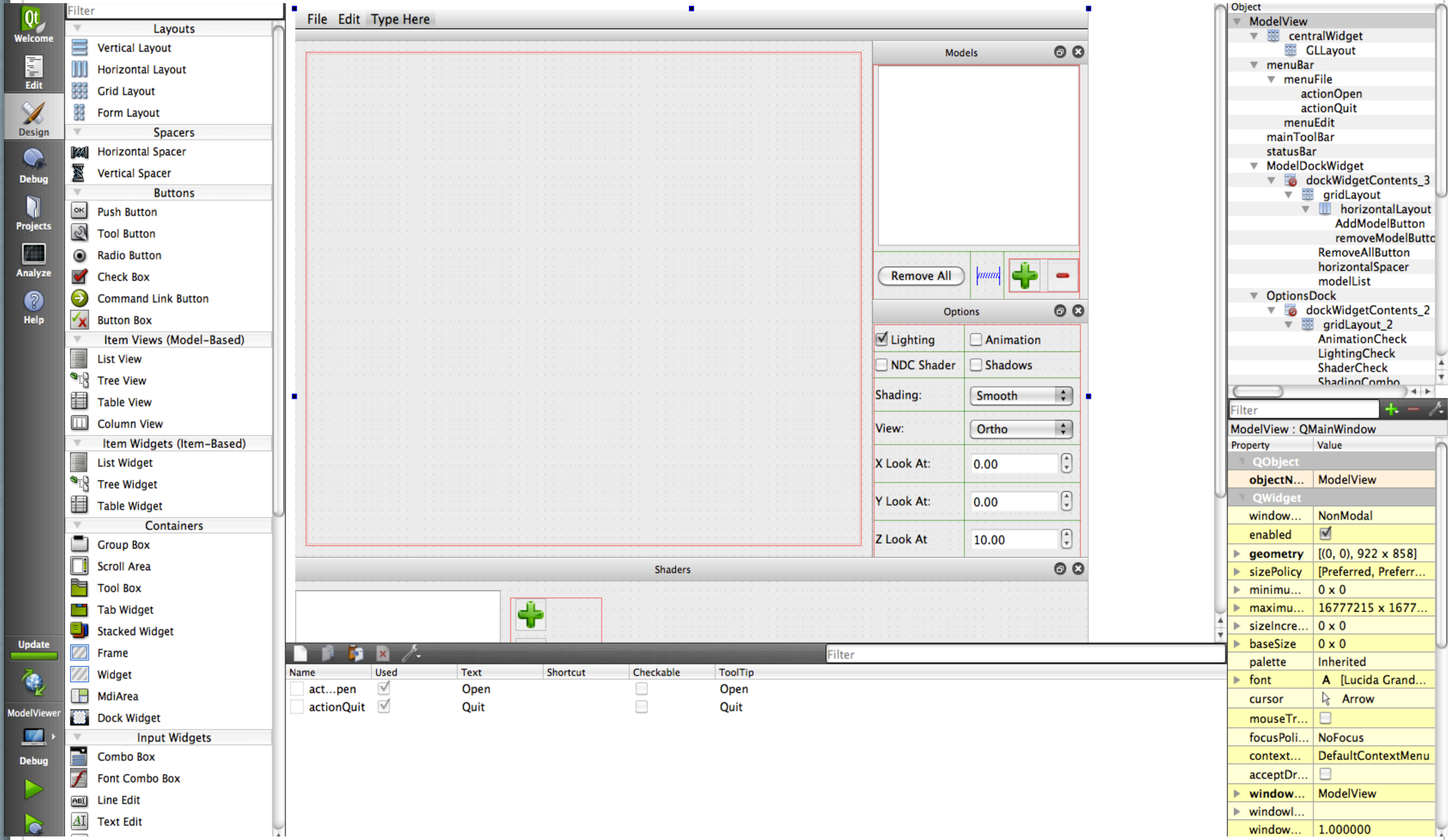
QtCreator

- More recent versions of Qt ship with a Qt specific IDE
- Extremely powerful editing capabilities
  - Very eclipse like but not as many refactoring tools
  - Very dynamic Qt based UI.
  - Autocomplete and bug detection support
- Handles all of the more complicated Qt build steps
- Can be used with non Qt projects.
- Built in GUI debugger (either GDB or MSVC debugger)

Qt Creator (2)

- Bundles all the Qt tools together
  - UIC
  - MOC
  - QtDesigner – GUI builder
  - QtAssistent – Qt documentation
- Built in support for version control systems
  - SVN
  - Git

Qt Designer



☑ Lighting
□ Animation
□ NDC Shader
□ Shadows

Shading: Smooth

View: Ortho

X Look At: 0.00

Y Look At: 0.00

Z Look At 10.00

Name	Used	Text	Shortcut	Checkable	ToolTip
act…pen	✓	Open		□	Open
actionQuit	✓	Quit		□	Quit

```
Object
ModelView
    centralWidget
        GLLayout
    menuBar
        menuFile
            actionOpen
            actionQuit
        menuEdit
    mainToolBar
    statusBar
    ModelDockWidget
        dockWidgetContents_3
            gridLayout
                horizontalLayout
                    AddModelButton
                    removeModelButto
                RemoveAllButton
                horizontalSpacer
                modelList
    OptionsDock
        dockWidgetContents_2
            gridLayout_2
                AnimationCheck
                LightingCheck
                ShaderCheck
                ShadingCombo
```

Filter

ModelView : QMainWindow

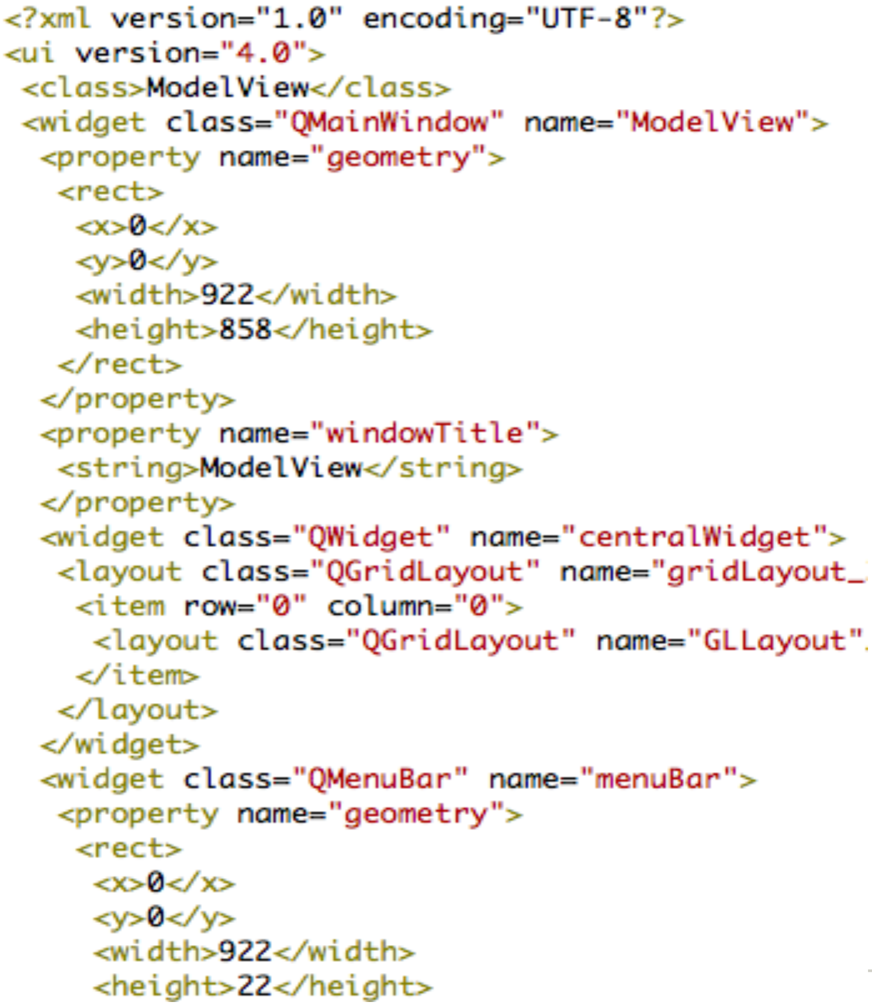
Property	Value
QObject	
objectN…	ModelView
QWidget	
window…	NonModal
enabled	☑
geometry	[(0, 0), 922 x 858]
sizePolicy	[Preferred, Prefer…
minimu…	0 x 0
maximu…	16777215 x 1677…
sizeIncre…	0 x 0
baseSize	0 x 0
palette	Inherited
font	A [Lucida Grand…
cursor	Arrow
mouseTr…	□
focusPoli…	NoFocus
context…	DefaultContextMenu
acceptDr…	□
window…	ModelView
windowI…	
window…	1.000000

Advanced Qt UI

- Qt Designer stores all UI information in *.uic files.
  - XML description of the UI.
  - XML matches nearly 1:1 with C++ classes.
- Convention has UI stored in a ui_<class name>.h file.
  - Declares all the UI elements needed
  - All signals and slots created in Qt Designer
- By convention the best practice is to subclass this ui file.
  - GUI changes don’t effect logic.

Advanced Qt UI (2)

- Qt .ui example



```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>ModelView</class>
 <widget class="QMainWindow" name="ModelView">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>922</width>
    <height>858</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>ModelView</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <layout class="QGridLayout" name="gridLayout_
    <item row="0" column="0">
     <layout class="QGridLayout" name="GLLayout".
    </item>
   </layout>
  </widget>
  <widget class="QMenuBar" name="menuBar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>922</width>
     <height>22</height>
```

Advanced Qt UI (3)

- Qt has a lot of functionality to create very dynamic UI
  - Widgets can be windows and windows can be widgets
- Allows for very user configurable interfaces.
  - Qframes allow for detachable windows and widgets
- Most UI elements can be “skinned” using regular CSS
- QML can be mixed with C++ to create custom widget animations

Advanced Qt UI (4)

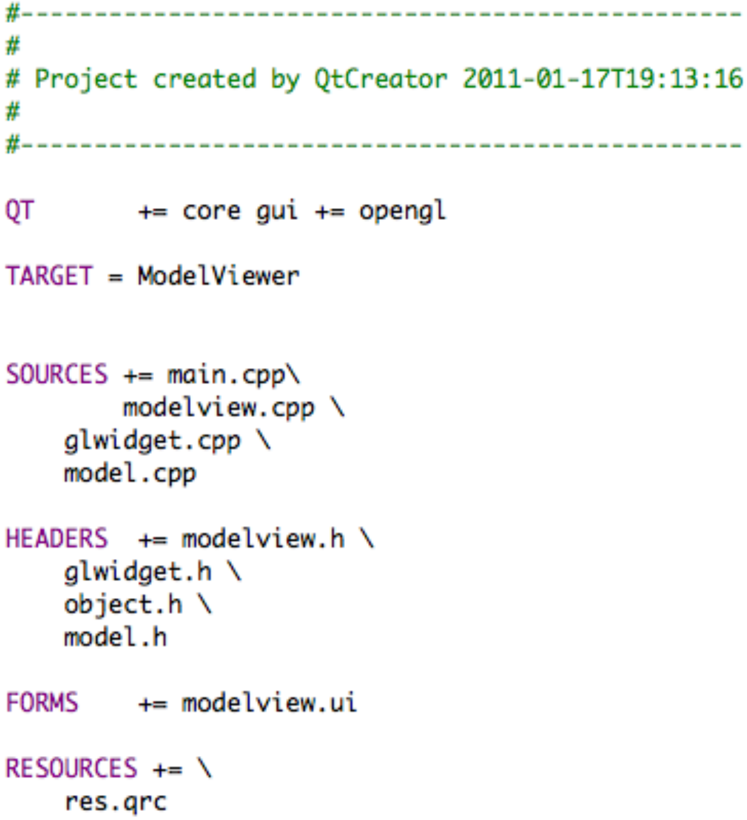
- Qt provides translation support.
  - If you app is distributed in multiple countries then you can encode all strings in a resource file and tag them with a locale string.
  - Qt automatically determines what the default language is and attempts to load strings in that languages if possible.
- If you don’t like the default UI widgets you can subclass and extend any widget
- Qt designer allows you to integrate your own widgets by inserting basic QObjects and then promoting them to your new subclass.

Building a Qt Application

- With the extensions to the C++ language and other special features Qt apps can’t be compiled normally.
- This is where QMake comes in.
  - QMake is a makefile / project file generator.
- Qt can be built against
  - GCC
  - Clang – either using gcc-clang or XCode
  - MinGW
  - MSVC

Building a Qt Application

- Qt projects are defined by a .pro file.
- .Pro files are a meta makefile
  - QT: sets Qt options like which modules are included
  - Sources / Headers: The source code
  - Forms: All .ui files
  - Resources: any resource files to be be compiled
- Libraries can be added with LIBS option



```
#-------------------------------------------------
#
# Project created by QtCreator 2011-01-17T19:13:16
#
#-------------------------------------------------

QT       += core gui += opengl

TARGET = ModelViewer


SOURCES += main.cpp\
        modelview.cpp \
    glwidget.cpp \
    model.cpp

HEADERS  += modelview.h \
    glwidget.h \
    object.h \
    model.h

FORMS    += modelview.ui

RESOURCES += \
    res.qrc
```

Building a Qt Application (2)

- Building a Qt application goes through multiple steps

1. Qmake *.pro -> builds a system specific makefile. Make is invoked

2. Uic (User Interface compiler) -> converts .ui files into .h and .cpp files

3. Moc (meta object compiler) -> expands all the signal and slots macros and adds extra code to glue together a project.

4. Compilation

5. Linking

6. Final executable

Qt Concurrency Framework

- Introduced in Qt 4.4
- Developed as a extension to Qt’s existing threading model
  - Threads
  - Thread specific storage
  - Thread Pools
  - Locks
  - Semaphores

Qt Concurrency Framework (2)

- QThread similar to Java threads
- To make a new thread inherit from Qthread and add implementation to virtual run method
- Very similar to java
  - Start method
  - Can set thread priority

Qt Concurrency Framework (3)

- Qt also provides QThreadStorage class which provides storage for individual threads in a thread safe way
- Template class to store pointers to any object
- Synchronization is done at a high level
  - Similar to tagging all getter and setter functions with Synchronized key word in java

Qt Concurrency Framework (4)

- Qt provides basic thread pool class.
- Functions as a collection of threads
- Not as evolved as java concurrent thread pools
- Submit a Qrunnable to the start method of the thread pool
  - If the number of running threads is < maxThreads then a new thread starts.

Qt Concurrency Framework (5)

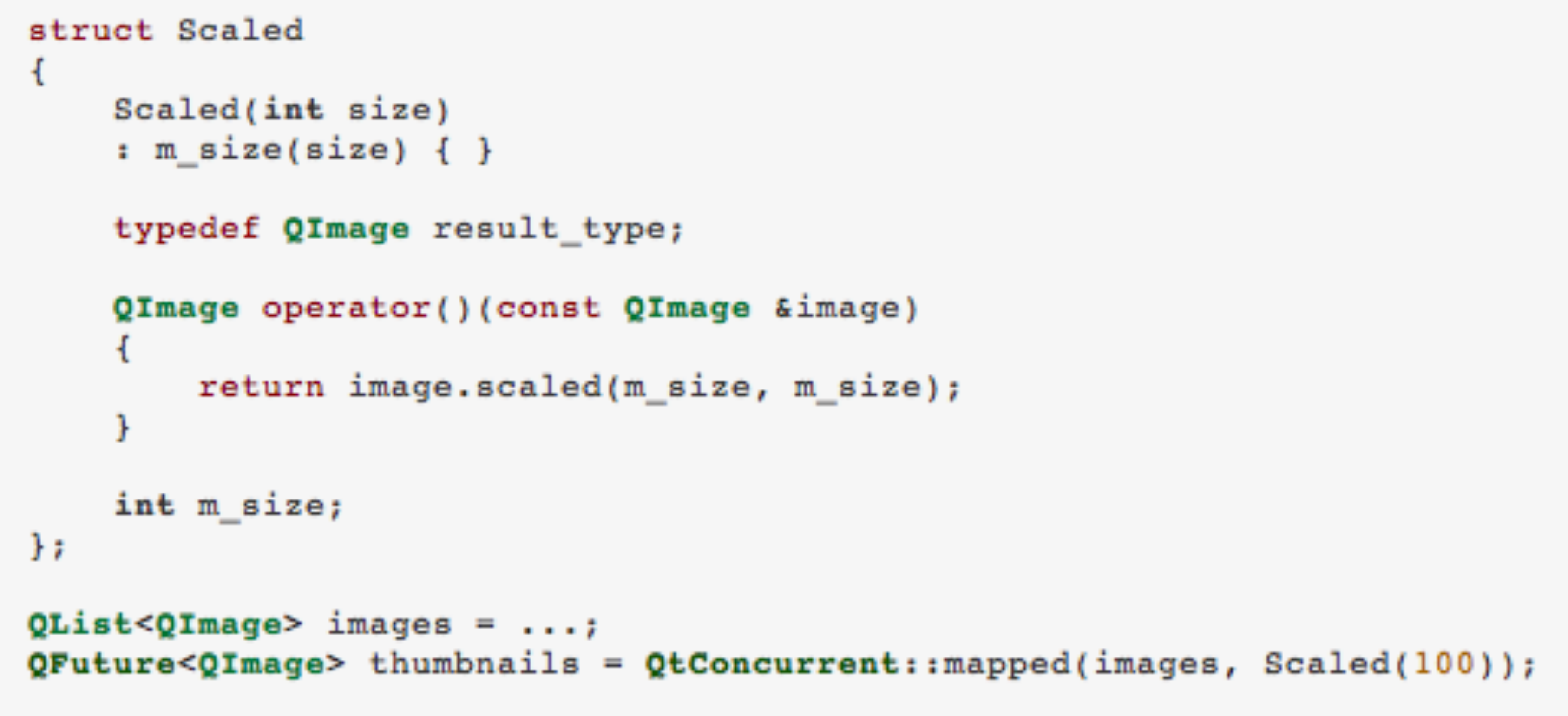
- Qt concurrency frame work still young
- Exports basic functions for concurrent operations.
  - Map
  - mapReduce
  - blockingMap
  - BlockingMapReduce

Qt Concurrency Framework(6)

- All functions in the concurrency framework follow similar conventions
- Pass in a list of futures and a function to apply
- Blocking variants will block until all functions complete
- Threads are allocated from the global thread pool
  - When including the concurrency module is global thread pool is created automatically.
- Non blocking ones depend on the blocking functionality of the Futures.

Qt Concurrency Examples

- Example using mapped to rescale images:



```
struct Scaled
{
    Scaled(int size)
    : m_size(size) { }

    typedef QImage result_type;

    QImage operator()(const QImage &image)
    {
        return image.scaled(m_size, m_size);
    }

    int m_size;
};

QList<QImage> images = ...;
QFuture<QImage> thumbnails = QtConcurrent::mapped(images, Scaled(100));
```

Qt Concurrency Examples

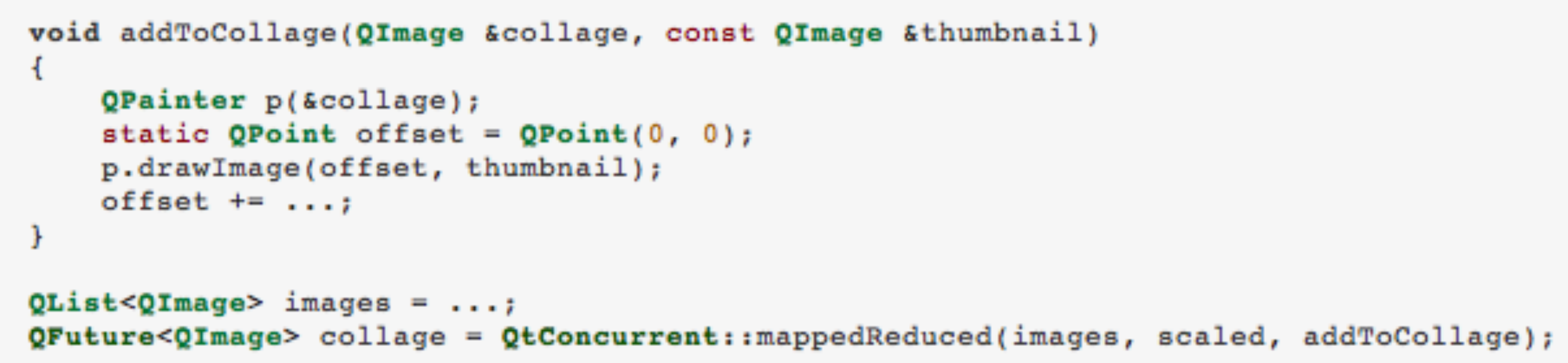
- Previous example makes use of a “function object”
  - Allows you to quickly develop parallel code without the need to subclass runnables or threads
  - Overloaded () operator means that when the object is called by the mapped function the operator is invoked.

Qt Concurrency Examples

- Using map reduce
  - Takes function pointers similar to map.
  - Must follow certain interface
- Map functions must have the form
  - U function(T &t)
- Reduce function must have the form
  - U function(T &result, const V intermediate)

Qt Concurrency Examples

- Extend the previous example by creating a collage of images



```
void addToCollage(QImage &collage, const QImage &thumbnail)
{
    QPainter p(&collage);
    static QPoint offset = QPoint(0, 0);
    p.drawImage(offset, thumbnail);
    offset += ...;
}

QList<QImage> images = ...;
QFuture<QImage> collage = QtConcurrent::mappedReduced(images, scaled, addToCollage);
```

Summary

- Qt is a full cross platform application development framework
- Handy for internal tool development
- Good alternative for many problem domains that don’t need a web app
  - Often times simpler and easier to write.
- Tons of support and large community and user base.

Summary

- Qt empowers developer to quickly create rich applications with a min of effort
- Create more code less.
- Highly tested and stable
- Check out the short screencast on creating a quick Qt based tool!

Resources

- Qt main webpage
- Qt language bindings
- Documentation for Qt 4.8
- Qt’s tutorial site