Cross-platform C++ development using Qt®

Fall 2005
Overview

• About Trolltech®/QT
• Qt Features and Benefits
• Examples and Code
• Demos and more Code!
• Questions
About Trolltech/QT
Trolltech

• Founded in 1994
• Offices:
  – Oslo, Norway (HQ)
  – Santa Clara, California
  – Brisbane, Australia
• Ownership:
  – Majority by employees
  – 24% VC
• Main products: Qt and Qtopia®
Products

• Qt: a complete C++ application development framework
• Qtopia: a comprehensive application platform and user interface for Linux-based consumer electronics
• QSA: Qt Script for Applications
• Teambuilder: distributed compilation solution Linux®/Unix
Other Products First

- Any Linux based PDQ will be running QTopia, which is based on QT-embedded.
- If you know QT, you could easily write for this platform.
Qt in a nutshell

• Qt is a complete C++ application development framework, including
  – A comprehensive C++ class library
  – RAD GUI development tool (Qt Designer)
  – Internationalization tool (Qt Linguist)
  – Help browser (Qt Assistant)
  – Source code and comprehensive documentation
Qt is comprehensive

• Qt supplies 80-90% of commonly needed functionality for rich client developers
  – 400+ fully documented classes
  – Core libs: GUI, Utility, Events, File, Print, Network, Plugins, Threads, Date and Time, Image processing, Styles, Standard dialogs
  – Modules: Canvas, Iconview, Network, OpenGL®, SQL, Table, Workspace, XML
  – Tools: Designer, Assistant, Linguist
  – Extensions: ActiveQt, Motif migration, MFC migration

• Qt Solutions for platform-specific customer requests
• Easy to add/extend/customize
Qt is cross-platform

• The Qt API and tools are consistent across all supported platforms
  – Qt runs on mobile phones to Cray supercomputers

• Consequence for users and customers
  – Freedom of choice in terms of development and deployment platform
  – Protection against changing platform fashions
A few platforms it runs on:

- Windows® 95 through Server 2003
- Mac OS® X
- Linux and embedded Linux
- AIX, BSD/OS, FreeBSD, HP-UX, IRIX, NetBSD, OpenBSD, Solaris, Tru64 UNIX
- And more
Qt is native

- Qt builds on the native graphics layer
- Qt applications run at compiled speed
QT Presentation

Native look on Windows
Native look on Linux
Native look on Mac OS X
Qt is open source, but…

1. Trolltech develops products
2. Releases beta versions
3. Community (open source and commercial) feedback
4. Rapid product stabilization
5. Final release, sell to customers
6. Sales funds further development
QT is also Commercial

- QT requires a commercial license if you plan to sell your product written in QT.
- Per developer licensing
- License cost based on edition and number of platforms
- Windows does not have GPL version of QT
- NO runtime fees/roalties etc.
Qt is rock solid

- Qt is used as the basis of the Linux KDE (K Desktop Environment)
  - Millions of lines of code, strong reliability requirements, industry strength stability
- Widely used by Linux community
  - thousands of developers
  - millions of end-users
- Used commercially in a wide variety of demanding applications
  - Medical devices
  - Air traffic guidance
Qt customers

- Adobe, Agilent, ARM, Boeing, Bosch, Cadence, Canon, CEA Technologies, ChevronTexaco, DaimlerChrysler, Deutsche Telekom, Earth Decision Sciences, ESA, Fraunhofer, HP, IBM, Intel, i-penta, JD Edwards, Lockheed Martin, LogicaCMG, Mentor Graphics, NASA, NEC, NTT, PGS, Pioneer, Rohde & Schwarz, Scania, Schlumberger, Sharp, Shell, Siemens, Sony, STN-Atlas, Stryker Leibinger, Synopsys, Thales...

- Qt is used for a wide variety of applications: mass-market and custom software, internal apps, research, modeling, visualization, etc.
Sample application: Adobe Photoshop Album
Qt Features and Benefits
Qt Features

• Unique features in Qt
  – Cross-platform, consistent, compiled API
  – Signals and slots
• Class library overview
• Qt tools overview
• Documentation
• Third-party integration
Cross-platform, consistent API

- Use the standard native tools to build Qt apps (IDE, debugger etc.)
- Qt provides a *platform-independent encapsulation* of the local window system and operating system
- The Qt API is identical on every platform, applications are compiled to native executables
- *Result: Write once, compile everywhere*
Signals & Slots

• Unique inter-object communication mechanism, provides
  – Type-safe callback between objects
  – Facilitates loose coupling / encapsulation
    • Sender and receiver does not ”know about” each other
  – 1-to-many, many-to-1 communication between objects

• Fully Object-oriented
Class Library Overview

• Full set of GUI classes
  – Widget set and GUI toolkit

• Operating system encapsulation classes
  – OO, C++ class interface to C system calls

• SQL database access classes
  – Data storing and retrieval

• Utility classes
  – Commonly useful classes

• Integration & Migration classes
  – Using Qt with other libraries and legacy code
OS encapsulation classes

- File & directory handling
- Date & time
- Registry / preferences
- Networking
  - URL, socket, TCP, DNS, FTP, HTTP
- Process handling
  - exec, terminate, kill, stdin/stdout/stderr I/O
- Threading
  - start, terminate, semaphore, mutex, wait
- Dynamic library loading
- Store, retrieve, query, traverse & modify DB data
- Database-independent API
  - Oracle, Sybase/MS SQL Server, MySQL, PostgreSQL, DB/2, Interbase, ODBC
- DB-aware widgets
  - Editable forms and tableviews
Utility classes

- String and regular expressions
  - Unicode
- Text and binary I/O
  - codecs
- Collections
  - Optional alternatives to STL collections
- XML I/O
  - Parser with SAX2 interface
  - DOM level 2 implementation
QT Presentation

Integration & Migration classes

• OpenGL
  – 3D graphics rendering in a Qt widget

• ActiveX
  – Host ActiveX controls in a Qt app
  – Use Qt to create ActiveX controls

• Motif
  – co-exist with legacy Motif code; stepwise migration

• Netscape Plugins
  – Create Netscape/Mozilla/Opera LiveConnect plugins with Qt
Development Tools Overview

- Qt Designer
  - Visual GUI builder
- Qt Linguist
  - Language translators’ tool
- Qt Assistant
  - Help browser
- qmake
  - Makefile generator, eases cross-platform builds
Qt Designer

- WYSIWYG, drag & drop GUI builder
- Supports the Qt auto-layout system
- Designs stored in open, documented XML format
- Does not interfere with user’s source code
Documentation

• Reference Manual
  – HTML
  – generated from the source code
  – Fully cross-referenced
  – Browsable from Qt Assistant
  – or normal web browser: doc.trolltech.com

• Tutorials
• Examples
• Qt programming books
Examples and Code

The fun stuff.
Hello World(ish)

```cpp
#include <qapplication.h>
#include <qlabel.h>

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

    QLabel* mylabel = new QLabel("Hello MSU", 0);
    mylabel->resize(80, 30);

    myapp.setMainWidget(mylabel);
    mylabel->show();
    return myapp.exec();
}
```
Signals and Slots (in-depth)

connect(Object1, signal1, Object2, slot1)
connect(Object1, signal1, Object2, slot2)
connect(Object1, signal2, Object4, slot1)
connect(Object3, signal1, Object4, slot3)
Signals and Slots – example

```cpp
// Recursive, i.e. slow, factorial function
int factorial( int n )
{
    if ( n <= 1 ) return 1;
    return n * factorial( n - 1 );
}
```

```cpp
connect( fontFamilyComboBox, activated(QString),
         textEdit, setFamily(QString) )
```

```cpp
connect( fontSizeSpinBox, valueChanged(int),
         textEdit, setPointSize(int) )
```

```cpp
connect( textEdit, modificationChanged(bool),
         customStatusBar, modificationStatus(bool) )
```
Defining Signals and Slots

• New C++ Syntax for defining Signals/Slots

```cpp
class myClass : public QObject {
    Q_OBJECT //required macro, no semicolon

    signals:
        void somethingHappened();

    ...

    public slots:
        void slotDoSomething();

    ...

    private slots:
        void slotDoSomethingInternal();

    ...}
```
Gory Details

• Signals: emit events
  – declare as signals, otherwise normal member functions
  – You don't implement them. Rather, you send them with the keyword `emit`
  – E.g. emit sliderChanged(5)

• Slots: receive and handle events
  – Normal member functions declared as slots

• Connect: must connect signals to slots
  – `QObject::connect( mymenu, SIGNAL(activated(int)), myobject, SLOT(slotDoMenuFunction(int)) );`

• moc: meta object compiler (preprocessor) converts these new keywords to real C++
int main(int argc, char* argv[]) {
    QApplication myApp(argc, argv);
    QWidget* myWidget = new QWidget();
    myWidget->setGeometry(400, 300, 120, 90);

    QLabel *myLabel = new QLabel("Hello MSU!", myWidget);
    myLabel->setGeometry(10, 10, 80, 30);

    QPushButton* myQuitButton = new QPushButton("Quit", myWidget);
    myQuitButton->setGeometry(10, 50, 100, 30);
    QObject::connect(myQuitButton, SIGNAL(clicked()), &myApp, SLOT(quit()));

    myapp.setMainWidget( myWidget );
    myWidget->show();
    return myApp.exec();
}
Hello with QT Designer

• Create a widget/dialog/wizard etc in Designer
• Use the code generated at compile time in the program.

#include "helloui.h" //Generated at compile time
int main(int argc, char* argv[]) {
    QApplication myApp(argc, argv);
    HelloUI* hello = new HelloUI(0);
    myApp.setMainWidget( hello );
    hello->show();
    return myApp.exec();
}
Look at Designer .ui and Demo

- Designer generates clean xml code
- Demo shows off QT well.
Thank you!

Additional information resources:

• http://www.trolltech.com/
• http://kde.org
• Gabe

Questions?