ES3 Lecture 11

Qt + Maemo development

Maemo

- Nokia's Linux based platform
  - Almost entirely open source
  - Nokia N770, N800, N810, N900 only models
  - Only N900 has 3G/phone capability

- N900 has relatively fast ARM CPU, GPU acceleration
  - resistive touch screen -- so no multitouch
- Development is very flexible
  - C, C++, Java, Python, Ruby, Lua, Lisp, whatever you want

Maemo development

- Can develop on the device itself
  - e.g. using gcc (but not really practical for big projects -- too slow and memory intensive)
  - or just copy over python scripts and running them...
- Scratchbox provides a Linux-based cross-compilation toolkit
  - Makes it easy to develop on a Linux system and target for Maemo
  - Only available for Linux though, and a bit tricky to set up
- Maemo emulator available as part of the API
  - Runs in virtual machine
- Development can be very straightforward
  - e.g. **ssh** into device to execute and debug
  - files can be directly shared, so you can edit files on the device transparently

Maemo Development (II)

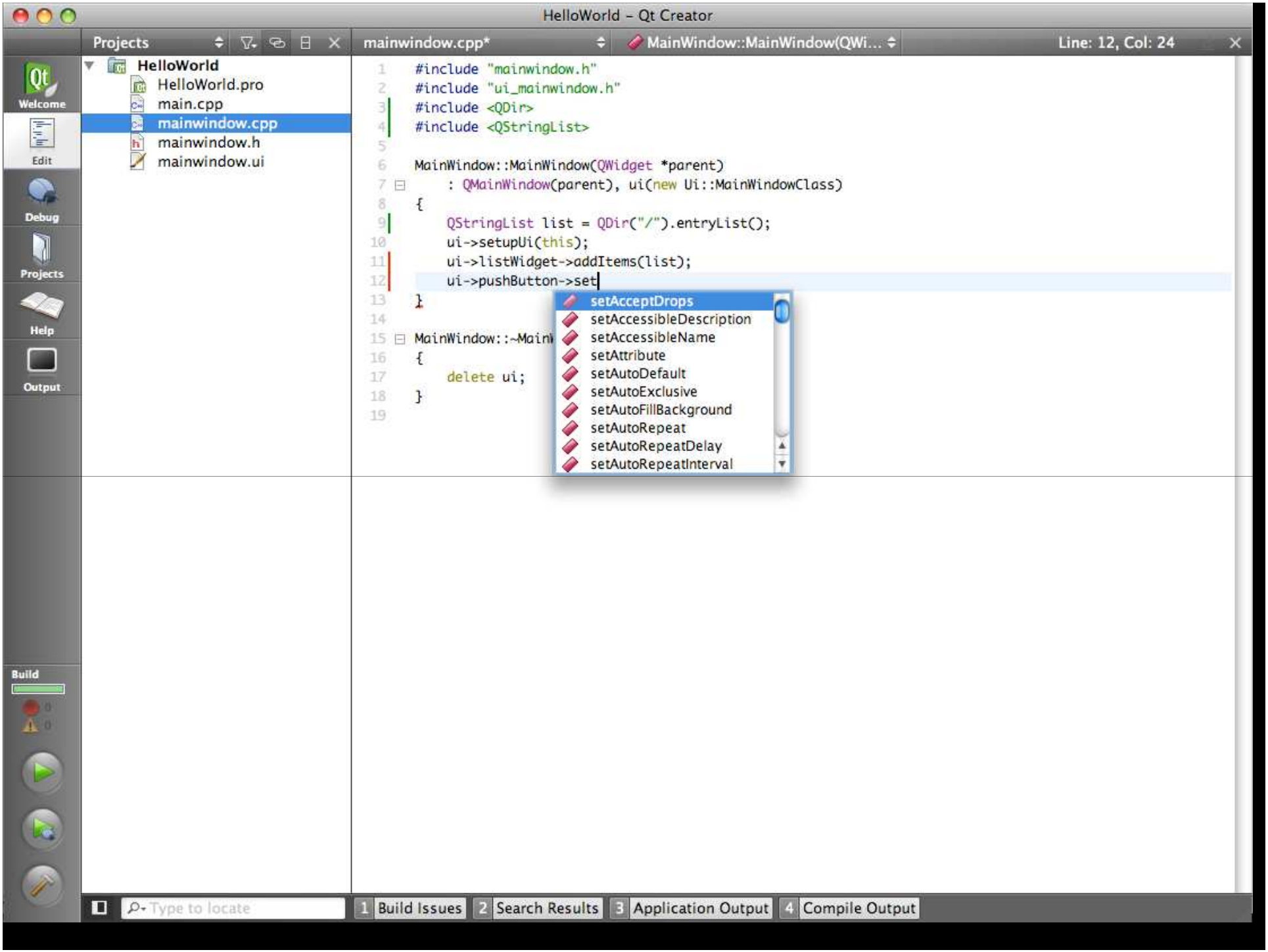
- Maemo uses a derivative of Debian
  - Many standard libraries and utilities are present
  - Porting new libraries is often feasible as well
- The Maemo UI is currently a custom UI built on GTK+ (Hildon)
  - adds "finger-friendly" extensions
  - supports a simple desktop environment
    - control panel, application manager
  - some common widgets for mobile systems implemented
- But Nokia will be moving to Qt across all their platforms shortly

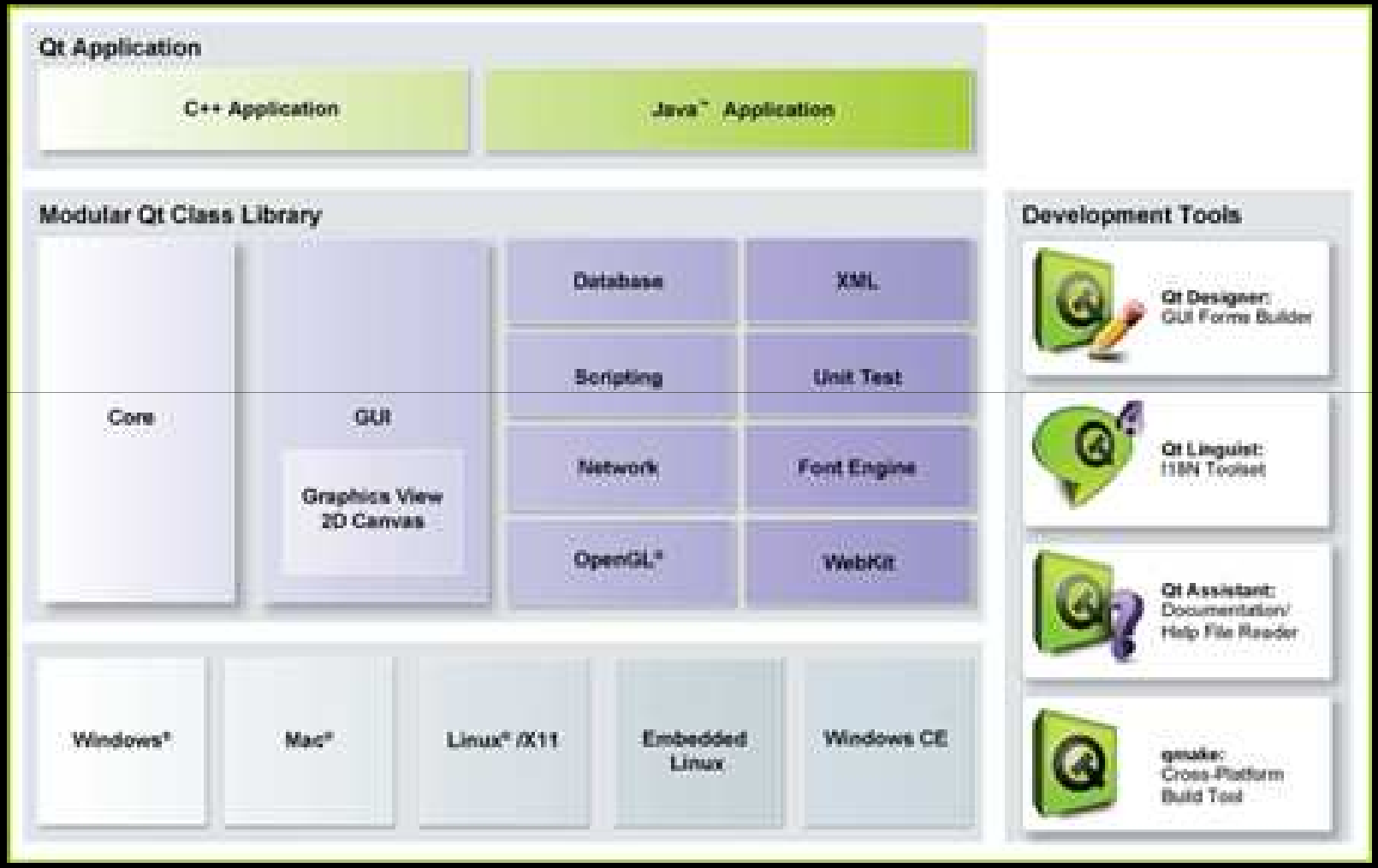
The Qt framework

- Qt is a full object-oriented framework with extensive GUI support
  - Written in C++
  - Large class library
- Provides basic container objects, file system access, multi-threading, networking, user interface components, scripting and database access
- Originally developed by TrollTech, recently bought by Nokia, who are pushing hard to standardize its use across their platforms
- Open-source, under the LGPL license
  - (Expensive!) commercial license available if you want to modify the *library* and redistribute without releasing the source

Development

- **Cross-platform**
  - code using this framework should simply recompile on another platform
  - unlike other platforms we've covered ,this is just a very complete library
    - it runs on desktop as well as mobile platforms
- Supported platforms include: Linux, Windows, Mac OSX, Maemo, Windows CE, Symbian and Maemo
  - experimental support for Android and even the iPhone(!)
- New IDE recently released (**Qt Creator**)
  - provides code editor, GUI designer, debugger etc.
- Although written in C++, bindings exist for other languages
  - **Jambi** provides Java bindings
  - **PyQt** provides Python bindings but is proprietary
  - **PySide** is Nokia's PyQt reimplementation project (a bit ropey at the moment)


HelloWorld – Qt Creator
Projects
mainwindow.cpp*
MainWindow::MainWindow(QWi...
Line: 12, Col: 24
HelloWorld
HelloWorld.pro
main.cpp
mainwindow.cpp
mainwindow.h
mainwindow.ui
Welcome
Edit
Debug
Projects
Help
Output
Build
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDir>
#include <QStringList>
MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent), ui(new Ui::MainWindowClass)
{
QStringList list = QDir("/").entryList();
ui->setupUi(this);
ui->listWidget->addItems(list);
ui->pushButton->set
}
MainWindow::~Main
{
delete ui;
}
setAcceptDrops
setAccessibleDescription
setAccessibleName
setAttribute
setAutoDefault
setAutoExclusive
setAutoFillBackground
setAutoRepeat
setAutoRepeatDelay
setAutoRepeatInterval
Type to locate
1 Build Issues
2 Search Results
3 Application Output
4 Compile Output


Qt Application
C++ Application
Java™ Application
Modular Qt Class Library
Core
GUI
Graphics View
2D Canvas
Database
XML
Scripting
Unit Test
Network
Font Engine
OpenGL®
WebKit
Windows®
Mac®
Linux® /X11
Embedded Linux
Windows CE
Development Tools
Qt Designer:
GUI Forms Builder
Qt Linguist:
I18N Toolset
Qt Assistant:
Documentation/
Help File Reader
qmake:
Cross-Platform
Build Tool

Qt Structure

- Qt has:
  - Core module (data structures, OS services)
  - GUI module (widgets, canvas)
- Extension modules, including:
  - OpenGL(ES)
  - Database access
  - Networking (HTTP, FTP, sockets)
  - Database drivers
  - XML parsing
  - Media playback (video and audio)
  - HTML renderer

Qt and C++

- Qt is implemented in C++, and is thus strongly-typed
  - nothing like the message-based model of Objective-C...
  - All the benefits (performance, compatibility, flexibility) of C++, and all of the downsides too (awful syntax, complexity, manual memory management)
- All Qt objects inherit from **QObject**, the base object of the Qt hierarchy
- C++ has manual memory management
  - Qt provides some help by automatically destroying objects hierarchically (e.g. window is destroyed, all child widgets are destroyed)
  - Provides QPointer **smart pointers** which automatically null after the object is freed
- Adds features such as internationalised strings (**QString**), hashtables (dictionaries),
  - standard C++ datastructures (e.g. from STL) are little used

Signals and Slots

- Qt has to be able to work over many different platforms
- It has a custom communication interface called **Signals and Slots**
  - Allows typesafe communication between any Qt objects
- Each signal and slot has a method signature
  - These signatures must match when connecting objects!
  - e.g. void **f(int x) -> void g(int x)**, but not **void f(QObject *q) -> void g(int x)**
- A slot is just a object method which will be called when an event is sent
- A signal looks like a method, but is never called
  - instead it is **emitted**
  - this routes the call to the connected slot instead

Signals and Slots

- Signals and slots are connected using **QObject**::**connect**()
  - takes a sending object, a signal, a receiving object and its slot

```
QObject::connect(vehicle, SIGNAL(engineStarted(int)), vehicleRegistry,
SLOT(vehicleStartedEngine(int))
```

  - Signals and slots are declared in the class definition

```
class Vehicle : public QObject {
...


signals:
 void engineStarted(int started);
}

class VehicleRegistry : public QObject {

public slots:
  void vehicleStartedEngine(int started);
}
```

Emitting signals

- Executing **emit engineStarted(1)** in this example would send the signal
  - **emit** will only work from within the class that defines the signal
  - Signals are always **private** and always return **void**
- **Any object connected to it would receive a call to the slot method**
  - Methods are executed synchronously
  - i.e. when an emit is encountered, each listening slot is executed, and then the code after the emit resumes
- Signals and slots are high-perfomance
  - slower than basic function calls, but not by much
- Sender of signal can be recovered in a slot using **QObject**::**sender**()
- Other more complex functionality available (e.g. asynchronous queued signals)

QPointer

- Qt provides guarded pointers (QPointers) which work with all Qt objects
  - Uses operator overloading to work like a normal pointer
  - But auto-nulls when the object it is pointing to is destroyed
    - avoids dangling pointers

- Use C++ generics so that **QPointer's** work just like ordinary pointers (except for no pointer arithmetic)

```
// standard pointer
QLabel *ql = new QLabel;

// guarded pointer
QPointer<QLabel> ql = new QLabel;
```

Javascript (ECMAScript) scripting

- Qt has built in ECMAScript support, called **QScript**
  - Basically Javascript
  - Integrated script debugger in the IDE
- QScript can access and manipulate Qt objects
  - properties, signals and slots are available
- Use is simple: create a QScript object, passing in any objects you want the script to access, and then execute it
  - The C++ code can set and get values that the script uses

```
QScriptEngine engine;
QScriptValue result = engine.evaluate(scriptString);
```

Statechart support

- Qt has powerful support for state machine models
  - especially the formalisms used in *statecharts*
- States and transitions can be defined, and actions can happen on transitions, and when states are entered and left
  - Guards and targetless transitions are supported
- States can be grouped to produce *hierarchical* state machines
  - History states (so that groups remember their previous state) are supported
  - Concurrent state machines are also possible
- State machines can receive messages from your code, and perform actions as a result (e.g. by sending messages back)
- State machines can be linked to the UI using the animation framework

GUI features

- Many standard widgets
- GUI editor
- Dialog creation tools (e.g. for wizards)

- High-quality anti-aliased drawing
  - Built in SVG support
- Animation support (similar to Android and iPhone, but more flexible)
- Multi-touch support
- Built in gesture-recognition (pinch etc.)

- Able to support *native* look and feel on all platforms
  - e.g. looks like Windows on Windows
  - This is quite unlike GTK...

GUI

- QtGui module provides standard widgets
  - text box, buttons, labels, combo boxes
  - advanced widgets like treeviews, toolboxes
  - printer support
  - undo support
  - drag and drop
  - accessibility functions
  - layout managers
- Item views support simple linkage of data sets to the GUI (e.g. standardized table views)

High-performance Canvas

- One of Qt's advantages is a powerful drawing module
  - Hardware accelerated, where supported
  - Highly scalable, and can draw huge numbers of graphical elements efficiently
  - Automatically supports printing
- Graphical effects like blurring, blending and shadowing are built in
- The **QGraphicsView** widget provides the canvas, and be used as any other widget
- Geometric primitives, Bezier curves, advanced type rendering are all supported

Qt Multithreading

- Qt has a cross-platform common interface for multithreaded applications
  - Thread starting and completion
  - Semaphores and mutexes
  - Inter-thread communication
- Threads can communicate using the standard signal and slot mechanism
- High-level concurrent programming interface allows parallel computation without using threads
  - e.g. mapping a function over a list, or doing map/reduce, or filtering sequences
  - makes it easy to scale applications across cores without rewriting any code
- **Inter-process** communication also has a standard interface
  - either using simple local sockets
  - or shared memory, for fast transfer of data between processes

Using PyQt

- PyQt provides standardized bindings for Qt
  - Open-source, but only licensable as GPL for free use
    - This means you must distribute soruce code
  - Commercial use requires a license
- PyQt is extremely complete, and covers virtually everything available in Qt
  - Even GUIs designed in the design tool can be converted to Python code with an automatic tool!
- Designed to make Qt appear as much like Python as possible
  - Most memory management issues just disappear
- There is a book
  - "Rapid GUI programming with Python and Qt" by **Mark Summerfield** published in 2007 (so reasonably up-to-date)

PySide

- Nokia's reimplementation of PyQt, using a free license (LGPL, like Qt)
- Incomplete as of now (e.g. no Windows port)
  - Supposed to be well supported on Maemo
  - It is not recommended for "production-level stability"
  - However, Users have reported than many applications written for PyQt work correctly with PySide as is
- API is currently intended to be one-for-one compatible with PyQt
  - i.e. you can just substitute the **import** line and everything will work the same