Developing a Qt GUI on i.MX 6

Tuukka Ahoniemi
Senior Technical Trainer
Qt, Digia USA

tuukka.ahoniemi@digia.com
twitter: @tuukka_a
Digia: A Brief Introduction

- Digia Plc acquired Qt Commercial Licensing and Support services in March 2011 from Nokia
- September 2012, Digia acquired the full Qt asset from Nokia
- Worldwide leading Qt application and UI development, testing and UX design services and consulting company with 7 international locations
- Digia has 10+ years of Qt experience
- 11 offices, 1100+ employees with 300+ in-house Qt experts
- Trusted by over 5,000 customer worldwide, used by more than 400,000 developers in 70+ industries
Today’s Outline

- Qt Overview
- Qt Tools
  - Build tools
  - QtCreator
- First Qt Application
  - Basic Data Types
  - First Application
  - QObject
  - Parent/Child Relationship
  - Memory management
- QtGui - Widgets
  - Input & Selection Widgets
  - Layouts
- Signals and Slots
- Application UI Creation with Qt Designer
- Deploying to i.MX 6 with Qt Creator
- Event Handling
- Creating Own Widgets
  - Qt Painting Pipeline
- Qt UI Offering
  - QWidgets
  - Graphics View
  - Qt Quick
The Stack

- Application
- QtGui
- QtCore
- Qt/Embedded
- QWS
- QWS Drivers
- Display
- Keyboard
- Mouse
- OpenGL
- Embedded Linux
- i.MX 6
Today’s Focus Area

Today’s Focus Area

- Application
- QtGui
- QtCore
- Qt/Embedded
- QWS
- QWS Drivers
- Embedded Linux (or other OS)
- OpenGL
- Qt/Embedded
- i.MX 6
- Display
- Keyboard
- Mouse
Why Work with Desktop Builds?

- Application
- QtGui
- QtCore
- Qt/Windows
- Windows GDI
- Windows Kernel
- PC HW
What is Qt, Then?
Qt is Used Everywhere

FROM EMBEDDED DEVICES TO DESKTOP APPLICATIONS

5000+ COMPANIES FROM MANY INDUSTRIES USE Qt
Qt Powers Industry Leading Software on Desktop ...
... and embedded Devices

- Advanced Info Centers
- Network Analyzers
- Medical Devices
- Security Automation
- Digital Photo Frames
- Refrigerators & Coffee Machines
- Set Top Boxes
- Industrial/UMPCS

And many, many more...
What is Qt?

C++ CROSS-PLATFORM APPLICATION AND UI FRAMEWORK

- Cross-Platform Class Library
- Integrated Development Tools
- Cross-Platform IDE
Qt Cross-Platform Developer Offering:
Not just a GUI toolkit

**Presentation**
- QtGui
- QtSVG
- QtQuick 1.1
- QtOpenGL

**Engine**
- QtCore
- QtDeclarative
- QtMultimediaKit
- QtNetwork

**Data**
- QtSql
- QtXML

QtMobility APIs
Qt Stack

Framework
- Core
- GUI
- Database
- WebKit
- Graphic View
- Scripting
- Multimedia
- OpenGL
- Qt Quick
- XML
- Network
- Unit Tests
- Benchmarking
- Mobility
- ActiveQt
- dbus

Tools
- Cross-platform IDE
- GUI Designer
- I18N tools
- Help System
- Build tool
- Visual Studio Integration

Cross-platform support
- Windows
- Mac OS X
- Linux/X11
- Unix
- Windows Embedded
- Embedded Linux
- QNX
- INTEGRITY
- VxWorks

Qt Commercial Services
Support, Consulting & Training

© 2012 Digia Plc
Qt Apps are Native

- Wrappers around native services
  - Native compiler
  - Possibility to mix code
  - Native performance, no runtimes/sandboxes!
Contents of Qt

• Cross-platform Qt consists of ~15 modules
  • A little bit less than 700 API classes, all modules depend on QtCore
  • Additionally 3 platform specific modules: QAxContainer, QAxServer, QtDBus

• Build Tools
  • Configure, Qmake, Moc, Uic and Rcc

• Development Tools
  • Qt Creator, Qt Designer, Qt Assistant, Qt Linguist

=> Qt Creator IDE
Tools
Tools – configure

- **configure** is a tool for configuring Qt itself for the current platform
  - Compiles **qmake**
  - Executed only during Qt installation
Tools Integration

Qt Project

Qt/C++ code files

.pro file

qmake [Linux/X11]

qmake [Embedded Linux]

qmake [INTEGRITY]

Makefile

Makefile

Makefile

.gpj

MULTI

GCC [Linux compiler]

Cross compiler

Native tool chains

App in Linux

App in Target HW

App in Target HW

Qt Creator

Other IDE / Code Editor
Building Qt Applications in General

1. qmake –project
   - Creates a Qt project file (.pro). This can also be created manually.

2. qmake
   - Uses the .pro file as input and produces platform-specific Makefile(s)
   - Generates make rules to invoke moc for project header files containing Q_OBJECT – as explained later

3. make
   - Compiles the program for the current platform
   - Executes also moc, uic and rcc
First Qt App

Hello World!
Creating Your Own Qt Application

- **Widget**
  - A UI building block, base class QWidget
    - Label
    - One line editor
    - Empty window
    - Main window
    - Button
    - etc.

- Often your own application UI is a widget of your own which consists of multiple inner widgets
#include <QtGui/QApplication>
#include <QtGui/QLabel>

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

- Initializes application settings
  - Palette, font
- Defines application’s look and feel
- Provides localization of strings
- Knows application’s windows
  - `widgetAt()`
- Use global qApp pointer to access QApplication instance
- Derived from QCoreApplication [QtCore]
  - Used in console applications (or Qt processes without any UI, servers for instance).
More Widgets?

```cpp
int main( int argc, char** argv )
{
    QApplication app( argc, argv );
    QLabel label( "Hello world!" );
    label.show();
    QLabel label2( "...from a Qt app!" );
    label2.show();
    return app.exec();
}
```

- Not exactly what was intended...
- Each widget `without` a parent becomes a window of its own!
QObject Class Role

- Heart of Qt's object model
  - Base class for all object classes
  - So, all QWidgets are QObjects also
  - Provides object trees and object ownership
  - QObject's responsibility is to provide a central location for the most important concepts in Qt

- Has three major responsibilities
  - Memory Management
  - Introspection
  - Event handling
Parent/Child Relationship

- Each QObject instance may take a parent argument
- Child informs its parent about its existence, upon which the parent adds it to its own list of children
- *If a widget object does not have a parent, it is a window*
- The parent does the following for its children:
  - Hides and shows children, when hidden/shown itself
  - Enables and disables children when enabled or disabled itself
- Note that a child may be explicitly hidden, although the parent is shown
Creating Objects

- Objects inheriting from QObject are allocated on the heap using new
  - If a parent object is assigned, it takes ownership of the newly created object – and eventually calls delete
    - QLabel *label = new QLabel("Some Text", parent);
- Objects not inheritingQObject are allocated on the stack, not the heap
  - QStringList list;
  - QColor color;
- Exceptions
  - QFile and QApplication (inheriting QObject) are usually allocated on the stack
  - Modal dialogs are often allocated on the stack, too
Memory Management

- The ownership of all child QObjects is transferred to the parent
  - Automatic deletion by the parent
  - Allocated from the heap with new
  - Manual deletion won't however cause double deletion because the child informs its parent of the deletion
- All QObjects without a parent must be deleted manually
  - Stack allocation is a good option to avoid problems
- Occasionally it may seem like Qt would hold some sort of automatic garbage collection but this is not true!
  - Always pay attention to ownerships and responsibilities!
Another Try

```c++
int main( int argc, char** argv )
{
    QApplication app( argc, argv );
    QLabel label( "Hello world!" );
    QLabel* label2 = new QLabel( "...from a Qt app!

                           &label );

    label.show();
    return app.exec();
}
```

- Nearly, but not quite enough…
- Let’s get back to this example later with layouts!
Display Widgets

- Labels (QLabel)
- Text Browsers (QTextBrowser)
- LCDNumbers (QLCDNumber)
- Progress Bar (QProgressBar)
- ...and more
Input & Selection Widgets

- Push buttons (QPushButton)
- Selection buttons (QRadioButton, QCheckBox)
- Text input
  - Line input (QLineEdit)
  - Plain text input (QPlainTextEdit)
  - Plain & rich text input (QTextEdit)
- Spin boxes (QSpinBox)
- Dial (QDial)
- Scroll bars (QScrollBar)
- Sliders (QSlider)
Customization of QWidgets

- Qt provides many ways of Styling QWidget based UI’s
  - QStyle
  - Style Sheets (CSS)
  - Custom Widgets
  - Custom gestures
  - Multitouch
Qt Assistant

- You can use Qt Assistant to see how to use all the widgets
- Provides not only profound class documentation but also good articles on Qt conventions
  - `<your desktop Qt folder>\qt\bin\assistant.exe`
Group & Layout Widgets

- Horizontal, vertical, grid layouts
  - QHBoxLayout, QVBoxLayout, QGridLayout
- Tab widgets, group boxes, etc.
  - QTabBar, QGroupBox
- Spacers (QSpacerItem, blank space in layout)
How To Use a Layout?

- Create layout(s)
- Create items
- Add items to layout with `addWidget()`
  - You can also create & add at the same time
- Layouts can be placed inside each other
  - Allows really flexible layouts
- Layouts should always be used instead of hard coded widget positioning
Back to Our Example: Final Try, Now With a Layout

```cpp
int main( int argc, char** argv )
{
    QApplication app( argc, argv );
    QWidget window; // Needed, 'cos layout can not become window
    QVBoxLayout* layout = new QVBoxLayout(&window);
    layout->addWidget( new QLabel( "Hello world!" ));
    layout->addWidget( new QLabel( "from a Qt app!" ));
    window.show();
    return app.exec();
}
```
Signals and Slots
Signals and Slots

- Observer pattern
- Type-safe callbacks
  - More secure than callbacks, more flexible than virtual methods
- Many-to-many relationship
- Implemented in QObject
  - Requires meta-object (macro Q_OBJECT)
class NewClass : public QObject
{
    Q_OBJECT // Meta-object file needed

    // Code convention recommends Q_OBJECT use always,
    // otherwise qobject_cast fails

signals:
    // Implementation in the meta-object
    void newSignal(int myInt, QString myString);
    void anotherSignal();

public slots:
    // Slots are implemented as normal member functions
    void newSlot(int i, QString s);
    void someSlot();
}
Signals

- A signal is a way to inform a possible observer that something of interest has happened inside the observed class
  - A QPushButton is **clicked**
  - An asynchronous service handler is **finished**
  - Value of QSlider is **changed** (and the new value is informed as a parameter)
- Signals are member functions that are *automatically implemented in the meta-object*
  - Only the function declaration is provided by the developer
- Signal is sent, or **emitted**, using the keyword emit
  - emit clicked();
  - emit someSignal(7, "Hello");
A slot is a function that is to be executed when a signal has been emitted.

- (When QPushButton is clicked), close QDialog
- (When service is ready), ask for the value and store it
- (When QSlider value is changed), display the value in QLCDNumber

A Slot function is a normal member function implemented by the developer.
Connections

- To receive a signal in a slot, signal and slot must be connected

```cpp
bool success = QObject::connect(
    senderObject, SIGNAL(valueChanged(int)),
    observerObject, SLOT(display(int)));
```

- Emitted signal results
  - In an immediate slot function call, if signal and slot implemented in objects in the same thread
  - In an delayed function call (next event loop), if signal and slot implemented in objects in separate threads

- Signal and slot signatures must match (signal may have more parameters)
  - *No compile time error checking is made!*
```c
int main( int argc, char** argv ) {
    QApplication app( argc, argv );

    QWidget window;
    QVBoxLayout* layout = new QVBoxLayout( &window );
    QLCDNumber* lcd = new QLCDNumber( &window );
    QSlider* slider = new QSlider( Qt::Horizontal, &window );
    layout->addWidget( lcd );
    layout->addWidget( slider );

    QObject::connect( slider, SIGNAL(valueChanged(int)),
                      lcd, SLOT(display(int)));

    window.show();
    return app.exec();
}
```
int main( int argc, char** argv ) {
    QApplication app( argc, argv );
    QWidget window;
    QVBoxLayout* layout = new QVBoxLayout( &window );
    QLCDNumber* lcd = new QLCDNumber( &window );
    QSlider* slider = new QSlider( Qt::Horizontal, &window );
    layout->addWidget( lcd ); layout->addWidget( slider );
    QObject::connect( slider, SIGNAL(valueChanged(int)),
                      lcd, SLOT(display(int)));
    window.show();
    return app.exec();
}
Qt Designer

- Comes with Desktop Qt as a separate application
  - Integrated to QtCreator, as "Form Designer"
- Drag’n’drop UI designer
- Outputs XML-based .ui files
- .ui files are converted automatically to .h files by uic (ui compiler)
  - myproject.ui -> ui_myproject.h
Slider Example in Qt Designer

- Add widgets with drag’n’drop
- Set Layout
- Modify widget properties
- Edit Signals & Slots
What Does Qt Designer Output?

- Qt Designer directly manipulates a .ui file, which is XML
- uic will (automatically) compile the .ui file into a c++ header file
  - ui_classname.h
  - Holds a UI Configuration class Ui::ClassName with function setupUi that initializes an existing main window
Extending the Class

- Easy to add custom functionality with custom slot functions

```cpp
#include "ui_Slider.h"

class SliderApplication : public QMainWindow {
    Q_OBJECT
public:
    SliderApplication(QWidget *parent = 0);
    ~SliderApplication();

public slots:
    void changeLcdColor(int value);

private:
    Ui::Slider ui;
};
```
Single Inheritance - Implementation

```cpp
Slider::Slider(QWidget *parent) : QMainWindow(parent) {
    ui.setupUi(this);
    connect(ui.horizontalSlider, SIGNAL(valueChanged(int)),
            this, SLOT(changeLcdColor(int)));
    // Make sure the color value is correct in the beginning
    changeLcdColor(ui.lcdNumber->value());
}

void Slider::changeLcdColor(int value) {
    QPalette pal = ui.lcdNumber->palette();
    pal.setBrush(QPalette::WindowText,
                 QBrush(QColor(value*2.55, value*2.55, value*2.55)));
    ui.lcdNumber->setPalette(pal);
}
```

Dragging the slider now also changes the color of the LCD Number (from black to white)
Summary

- Own GUI often is an extended version of QMainWindow (or QDialog)
  - The main widget also acts as a parent for all other widgets/objects
- Custom behaviour is added with custom slot functions that are connected to suitable signals
- Parent/Child relationship also handles dynamic memory management
- Qt Designer can be used for quick UI layouting
- QWidgets provide a nice set of existing easy-to-start-with UI components
  - Customizable and extendable
Deploying to an i.MX 6
Installation Steps, Prepare Host and Board

1. Host computer with an Ubuntu and a C++ development environment
   - build-essentials (debian)

2. Install an Embedded Linux to your i.MX 6 board with Freescale’s instructions
   - SSH server
   - Connect to network, check IP address

3. Get a cross-compilation toolchain for your Embedded Linux to your host
   - From Freescale
Installation Steps, Qt Commercial SDK

4. Install Qt Commercial SDK, which contains
   - Qt Creator IDE
   - Qt Commercial libraries 4.8
   - Easy configurability to embedded cross-compilation tool chains
   - Updater

   - qt.digia.com
Installation Steps, Create Target Conf

• Build a mkspecs for your target
  • qmake.conf under qt sources folder mkspecs/qws
  • Good starting point is mkspecs/qws/linux-arm-gnueabi-g++, modify it to right tool binaries:

  QMAKE_CFLAGS = -march=armv7-a -mfpu=neon -mfloat-abi=softfp
  QMAKE_CXXFLAGS = -march=armv7-a -mfpu=neon -mfloat-abi=softfp
  QMAKE_CC = arm-fsl-linux-gnueabi-gcc
  QMAKE_CXX = arm-fsl-linux-gnueabi-g++
  QMAKE_LINK = arm-fsl-linux-gnueabi-g++
  QMAKE_LINK_SHLIB = arm-fsl-linux-gnueabi-g++

  # modifications to linux.conf
  QMAKE_AR = arm-fsl-linux-gnueabi-ar cqs
  QMAKE_OBJCOPY = arm-fsl-linux-gnueabi-objcopy
  QMAKE_STRIP = arm-fsl-linux-gnueabi-strip

  load(qt_config)

• Set PATH so that your cross-compiler tools are in it
Installation Steps, Building Qt

- Configure

- Make
- Make Install

- Copy .so files to the device
  - scp
Configuring Qt Creator

- Create a new tool chain for Qt Creator:
  - Tools->options->Build&Run->Tool Chains->Add->GCC->Compiler Path->Browse->...g++ binary

- Add a new Qt version (locate the qmake binary you just compiled)
Configuring Qt Creator

- For direct deployment, configure a new Linux device:
  - You will need the device’s IP address
Setup For Individual Project

- Setup new build target (Embedded Linux) using the configured build tool chain, Qt version and the remote device deployment configuration
Run Settings for Individual Project

Deployment
Method: Deploy to Remote Linux Host
Device configuration: LMX6
Files to install for subproject: testi123.pro

Local File Path
/home/digia/testi123-build-embedded-Qt_4_8_3_FSL_Release/testi123
Remote Directory
/opt/testi123/bin

Upload files via SFTP
Add Deploy Step

Run
Run configuration: testi123 (on Remote Device)
Device configuration: LMX6
Executable on host: 
Executable on device: 
Alternate executable on device: qws
Arguments: 
Working directory: <default>
You’re All Set!

- Just change target and press play!
Event Handling
**Events vs. Signals**

1. User presses button in UI

2. Event is generated and delivered for the corresponding object’s event handler function

3. Pushbutton emits a signal `clicked()` to “shout out loud” of the situation

4. Previously connected slot functions get called
Events vs. Signals

- So, if you CREATE your own widgets

- You create event handlers:
  - What does this widget do when someone clicks it?
  - What does it do when keyboard is pressed?
  - What does it do when it is requested to paint itself (i.e. how does it paint itself)?

- You also tell, when does it emit signals…
Events vs. Signals

- If you USE existing widgets

• You are interested in existing signals

  • What do I want to do (somewhere else) when the widget is clicked()?

  • You connect slot functions to the signals

SomeObject

Hey!

valueChanged()

Hey! I'm clicked()

SomeObject

slot function

slot function
Event Handling in Brief

• In Qt, events are objects that are delivered to event handling functions
  • Base class QEvent, subclasses like QMouseEvent, QKeyEvent, QTimerEvent
• When an event occurs, Qt creates the event object and delivers the event to the particular QObject

=> QObject::event ( QEvent* e )
  • Does not handle the event, **redelivers to a corresponding virtual event handler**
  • Own virtual protected functions for different events
You Want to Have Your Widget to...

- Respond to key presses, implement:
  
  ```cpp
  void keyPressEvent(QKeyEvent*)
  ```

- Use timer, implement:

  ```cpp
  void timerEvent(QTimerEvent*)
  ```

- Respond to mouse presses, implement:

  ```cpp
  void mousePressEvent(QMouseEvent*)
  ```

- React to layout changes, implement:

  ```cpp
  void resizeEvent(QResizeEvent*)
  void moveEvent(QMoveEvent*)
  ```

- Etc.

- Quite straightforward from the developer’s point of view!
Creating Custom Widgets
Custom Widgets

- Derive from some QWidget sub-class and provide specialized behavior and/or data
  - For example hexadecimal number editor from QLineEdit
- …or derive directly from QWidget
  - Override needed event handlers
  - At least paintEvent()
  - Possibly mousePressEvent(), mouseMoveEvent(), focusInEvent(), focusOutEvent(), closeEvent(), keyPressEvent(), leaveEvent()
  - React to layout changes in S60 by overriding resizeEvent() (and possibly moveEvent())
- Custom widgets can be added to Qt Creator/Designer by writing a Qt Designer plug-in, which creates an instance of the custom widget
Event Processing and Painting

- Painting takes place when QWidget::paintEvent() function call is received by the application
- Two ways to request repaint (QWidget member functions)
  - repaint() – causes immediate call to paintEvent()
    - Avoid
    - Use only in animations, where immediate repaint() needed
  - update() – causes a repaint event to be queued
    - Several update() calls are combined by Qt
    - Does not cause flickering
Basic Drawing Pipeline

- The class `QPainter` performs drawing operations
  - Draws on a 2D drawing surface represented by a `QPaintDevice`
  - The class `QPaintEngine` provides a device- and platform-independent API for this

- Examples of classes inheriting `QPaintDevice`:
  - `QWidget`, `QPrinter`, `QPixmap`, `QImage`, `QPicture`, `QGLPixelBuffer`, `QGLFrameBufferObject`, ...
QPainter

- Draw different shapes
  - Polygon, rectangle, ellipse, pie, line, arc, text
- Specify pen width and style
  - Solid, dash, dot
- Enable/disable antialiasing
- Set brush
  - No brush, gradient color, texture
- Use transformations
  - Translate, rotate, scale
- Save and restore drawing context
  - QPainter settings
Qt UI Offering
Qt Combines Languages & Technologies

- QML
- HTML5, CSS3, JavaScript
- C++, OpenGL

Qt Application
Qt provides a set of UI tools and technologies for developers to choose from.

Different UI approaches are provided to best fit each different UI use case and development environment.
Rough Division

- Conventional UIs
  - QWidgets

- Custom UIs
  - Qt Quick
- Complete and customizable set of UI Controls
  - C++ classes
- Dynamic layout system
  - Different screen sizes and resolutions
QWidgets

- Complete and customizable set of UI Controls
  - C++ classes
- Dynamic layout system
  - Different screen sizes and resolutions
- Native look ‘n feel on desktop
• Complete and customizable set of UI Controls
  • C++ classes
• Dynamic layout system
  • Different screen sizes and resolutions
• Native look ‘n feel on desktop
• Easy prototyping on embedded
  • Multiple levels of customization
- Complete and customizable set of UI Controls
  - C++ classes
- Dynamic layout system
  - Different screen sizes and resolutions
- Native look ‘n feel on desktop
- Easy prototyping on embedded
  - Multiple levels of customization
- WYSIWYG drag’n drop UI editor available
QWidget Pros & Cons

- Very straightforward to use
- Extendable and customizable
- Cross-platform native look 'n feel
- Desktop oriented, traditional
- Missing some concepts, like views
- Customizing the whole user experience is tedious
The Other Side?

Conventional UIs

QWidgets

Custom UIs

Qt Quick
Custom UIs

Qt Quick

- Bridges the gap between UI Designers and developers
- QML
  - Qt Meta Language, Qt Markup Language
  - Declarative language for creating user interface
  - CSS-like
  - JavaScript
  - Property bindings
Custom UIs

Qt Quick

- Bridges the gap between UI Designers and developers
- QML
  - Qt Meta Language, Qt Markup Language
  - Declarative language for creating user interface
  - CSS-like
  - JavaScript
  - Property bindings
- Qt Declarative Module
  - C++
Qt Quick

- Bridges the gap between UI Designers and developers
- QML
  - Qt Meta Language, Qt Markup Language
  - Declarative language for creating user interface
  - CSS-like
  - JavaScript
  - Property bindings
- Qt Declarative Module
  - C++
- Qt Creator Tools
  - Visual drag 'n drop designer
  - States and Transitions
Anything Else?

Conventional UIs

Qt Widgets

Custom UIs

Qt Quick
Anything Else?

Conventional UIs

QWidgets

Custom UIs

Qt Quick

Graphics View
Anything Else?

Conventional UIs

- QWidgets
- QtOpenGL

Custom UIs

- Qt Quick
- Graphics View
Hybrid UIs with QtOpenGL

- Direct OpenGL interface to embed 3D graphics into Qt app
  - Whole UI (games), parts of UI (hybrid)
  - Hardware accelerated graphic performance

Bounce Evolution game by Rovio in Nokia N900

Google Earth
Qt/C++ UI Panel

OpenGL (Car)
Even More?

Conventional UIs

- QWidgets

Custom UIs

- Qt Quick
  - QtOpenGL
  - QtWebkit

- Graphics View
Hybrid UIs w/ QtWebkit

- Write UI/parts of UI (+logic) with Web technologies, embed in a Qt Application
- QtWebkit is a powerful HTML5 engine with C++ backend

Qt Application

QWebView

MyUi.html

*.css

*.js

shows contents of
Deadline 6 – Render Management System

### Jobs

<table>
<thead>
<tr>
<th>Job Name</th>
<th>User</th>
<th>Errors</th>
<th>Comment</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>multi_cam_scene - occlusion</td>
<td>ryan</td>
<td>0</td>
<td>3ds Max</td>
<td></td>
</tr>
<tr>
<td>multi_cam_scene - spec - camera_LShape</td>
<td>ryan</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>multi_cam_scene - spec - camera_MISCShape</td>
<td>ryan</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>multi_cam_scene - occlusion - camera_MLShape</td>
<td>ryan</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>multi_cam_scene - occlusion - camera_RSShape</td>
<td>ryan</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>multi_cam_scene - beauty - camera_LShape</td>
<td>ryan</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>multi_cam_scene - beauty - camera_MISCShape</td>
<td>ryan</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>multi_cam_scene - beauty - camera_RSShape</td>
<td>ryan</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>multi_cam_scene - beauty - camera_MLShape</td>
<td>ryan</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>test</td>
<td>ryan</td>
<td>6</td>
<td>3ds Max</td>
<td></td>
</tr>
<tr>
<td>Lord of all stations v033</td>
<td>robert</td>
<td>6</td>
<td>3ds Max</td>
<td></td>
</tr>
<tr>
<td>Beneath suddenly interactive missile v039</td>
<td>justin</td>
<td>0</td>
<td>3ds Max</td>
<td></td>
</tr>
<tr>
<td>test_demo_scene_bhalaridas2</td>
<td>ryan</td>
<td>326</td>
<td>3ds Max</td>
<td></td>
</tr>
<tr>
<td>Beneath suddenly interactive missile v029</td>
<td>robert</td>
<td>0</td>
<td>3ds Max</td>
<td></td>
</tr>
<tr>
<td>Demonstrating coarse flux v038</td>
<td>edwin</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This is not a test v030</td>
<td>thomas</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Slaves

<table>
<thead>
<tr>
<th>Slave Name</th>
<th>Machine Name</th>
<th>Machine User</th>
<th>Descriptor</th>
<th>Status</th>
<th>Last Status</th>
<th>Status Mes</th>
<th>Connects</th>
</tr>
</thead>
<tbody>
<tr>
<td>TB-RN-03</td>
<td>TB-RN-03</td>
<td>Administrator</td>
<td>Offline</td>
<td>2.67 m</td>
<td>Jun 18/1...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>TB-RN-02</td>
<td>TB-RN-02</td>
<td>Administrator</td>
<td>Offline</td>
<td>2.67 m</td>
<td>Jun 18/1...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>AGENT-001-AGE</td>
<td>Agent-001</td>
<td>chrissbourn</td>
<td>Offline</td>
<td>2.46 m</td>
<td>Jul 12/12...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>T-B-RN-01</td>
<td>TB-RN-01</td>
<td>Administrator</td>
<td>Offline</td>
<td>2.46 m</td>
<td>Jun 18/1...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>T-B-RN-04</td>
<td>TB-RN-04</td>
<td>Administrator</td>
<td>Offline</td>
<td>2.46 m</td>
<td>Jun 18/1...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>T-B-RN-05</td>
<td>TB-RN-05</td>
<td>Administrator</td>
<td>Offline</td>
<td>2.46 m</td>
<td>Jun 18/1...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>MOBILE-005</td>
<td>mobile-005</td>
<td>johnquaddet</td>
<td>Stalled</td>
<td>1.39 m</td>
<td>Aug 1/...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>RNDVM-WIN7-32</td>
<td>RNDVM-WIN7-32</td>
<td>deadline</td>
<td>Offline</td>
<td>1.1 m</td>
<td>Aug 23/...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>AGENT-002</td>
<td>Agent-002</td>
<td>cb</td>
<td>Offline</td>
<td>4.05 w</td>
<td>Aug 28/...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Quirk</td>
<td>Quirk: local</td>
<td>administrator</td>
<td>Offline</td>
<td>4.04 w</td>
<td>Aug 28/...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>MOBILE-010</td>
<td>Mobile-010</td>
<td>Edwin</td>
<td>Offline</td>
<td>2.73 w</td>
<td>Sep 06/...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>SMILES</td>
<td>Smiles</td>
<td>diana.carrier</td>
<td>Offline</td>
<td>4.98 d</td>
<td>2012/09/...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>NATSUMI</td>
<td>natsumi</td>
<td>matter</td>
<td>Offline</td>
<td>2.12 d</td>
<td>2012/09/...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Ubuntu12</td>
<td>Ubuntu12</td>
<td>thinkbox</td>
<td>Offline</td>
<td>4.62 hrs</td>
<td>2012/09/...</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>MOBILE-004</td>
<td>mobile-004</td>
<td>Ryan</td>
<td>Offline</td>
<td>4.62 hrs</td>
<td>2012/09/...</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

**User:** justin  **Last Update:** 0
QNX Medical Demo

- By QNX Software Systems
- Written in Qt Commercial 4.7
- Running on the QNX Neutrino real time operating system
Accessing Hardware

- Qt Quick UI
- Qt/C++ Backend
- Other C++ Code
- HW Peripherals
Eykona Wound Measurement System

- Estimated time save of 6 to 9 months by using Qt as it is cross platform

- Sleek and modern looking UI
  - Based on widgets and style sheets

- Certified medical camera

- Both an embedded Linux and a desktop application
  - Minimize investments, training, maintenance etc.
Wrap-Up
Highlights of Qt Commercial Releases

4.6
- Expanded support for embedded development
- Enablers for cutting edge UI development
- OpenVG graphics
- Stability and reliability improvements

4.7
- Performance optimization
- New Qt Mobility APIs
- Qt Creator
- Qt Quick
- Significant WebKit improvements
- HTML5 support

4.8
- Performance and quality improvements
- Desktop and embedded specific improvements
- New WebKit
- Platform abstraction
- New supported platforms

5.0
- Binary break, mostly source compatible
- Qt Quick 2 with V8 JavaScript Engine and OpenGL based graphics architecture
- C++ Widgets still supported and do not require OpenGL
- Modularization allows easier configuration
- WebKit 2 engine architecture available, WebKit 1 API supported
- Improved contribution tools and processes
Thank You!

tuukka.ahoniemi@digia.com

qt.digia.com
Blog: qt.digia.com/qtcommercialblog
Twitter: @Qtcommercial

© 2012 Digia Plc 11/7/2012