MeeGo Application and Platform Development

Course Setup

digia

digia ?

- That's where me and ~1499 other computerish professionals work
  - Global, based in Finland
- ~50 % work with mobile technologies
- A large, traditional Symbian house
- Since day 0, heavily involved with Qt on mobile devices

Qt Alliance Partner

- Qt Training Partner
- For figures, facts, open positions, contacts
  - Ask me, or see
  - **http://www.digia.com**

digia

2

Course Logistics

- Welcome
- Trainer
  - Senior Software Specialist Tino Pyssysalo
- Facilities
- Course timings 
- Refreshments & lunch

digia

3

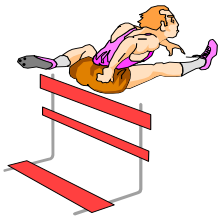
Course Objectives

- Learn MeeGo application and platform development
- Tools, QEMU
- MeeGo community
  - Projects, packages, community tools
- Qt Quick
- Platform development
  - GObject, D-Bus
- Qt Mobility APIs

digia

4

Course Prerequisites



- C/C++ programming experience
- Qt essentials

digia

5

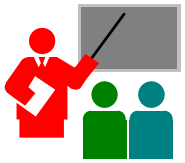
Course Contents

- **Introduction**
  - **What is MeeGo?**
  - **MeeGo Architecture**
- **MeeGo OSS Model**
  - **MeeGo OSS Community**
  - **GIT Repository**
  - **Contributions**
- **MeeGo Development**
  - **MeeGo SDK**
  - **QtCreator**
  - **Debugging**
  - **On-Device Debugging**
  - **Deployment**
  - **RPM Packaging**
  - **Image Creation**
  - **Quality Assurance**
- **MeeGo API**
  - **Handset and Netbook UX**
  - **MeeGo Touch**
- **Qt Quick**
  - **Qt Quick**
  - **QML Essentials**
  - **Layouts**
  - **User Interaction**
  - **States, Transitions, Animations**
  - **Data Models and Views**
  - **C++ Bindings**
- **Middleware Subsystems**
  - **Middleware Subsystems**
  - **Telephony, Messaging, SocialWeb, GStreamer etc**
- **Qt Mobility APIs**
- **Platform Development**
  - **Tools**
  - **Glib Library**
  - **Using D-Bus**
  - **Qt4 APIs**
  - **API Wrappers**

digia

6

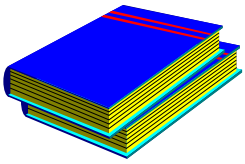
Course Delivery



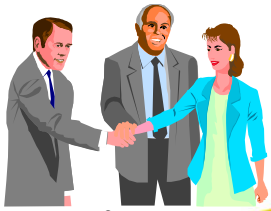
Presentations

Questions

Hands-on Sessions



Course Manuals



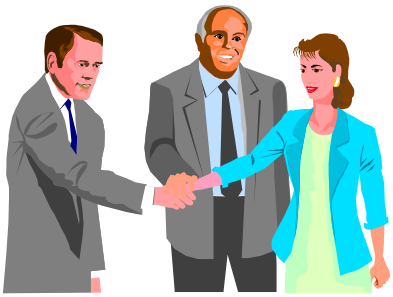
Learn from others

digia

7 © 2009 Digia Plc

Exercise

- Introduce yourselves
- MeeGo experience
- Describe your personal objectives for coming on the course



digia

8 © 2009 Digia Plc

MeeGo Application and Platform Development

Introduction

digia

Agenda

- What is MeeGo?
- MeeGo Architecture

digia

10

What is MeeGo?

Intel's Moblin

Maemo

MeeGo™

digia

11

Moblin – Mobile Linux

- Open source operating system and application stack for mobile Internet devices
- Built around the Intel Atom processor
- Plenty of devices
- Highly optimized boot time
- GTK and Clutter –based UI framework
- Does not use any base Linux distribution
- RPM-based package management

digia

12

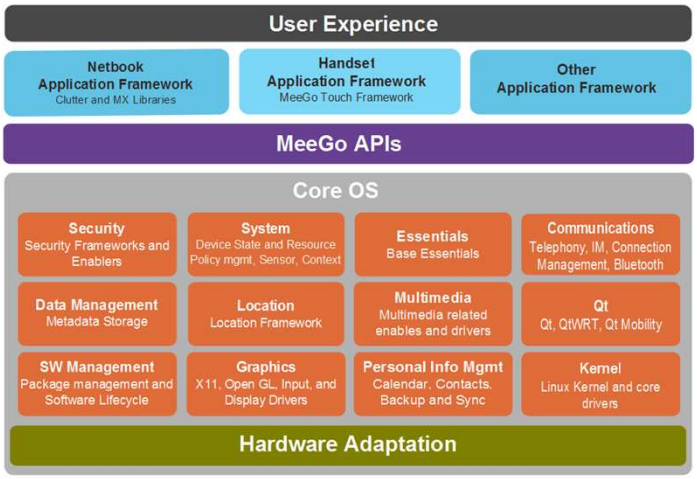
Maemo

- Open source development platform for Internet Tablets and mobile computers
  - 90% of source code is open-source
- Latest version 5.0 – Fremantle
  - Adds phone functionality to Internet tablets
  - GTK and Clutter –based UI framework
  - Qt libraries included in the latest firmware update (Qt libraries may be separately loaded)
  - One Fremantle-based device – N900
    - ARM Cortex A8 core
- Debian APT –based package management

digia

13

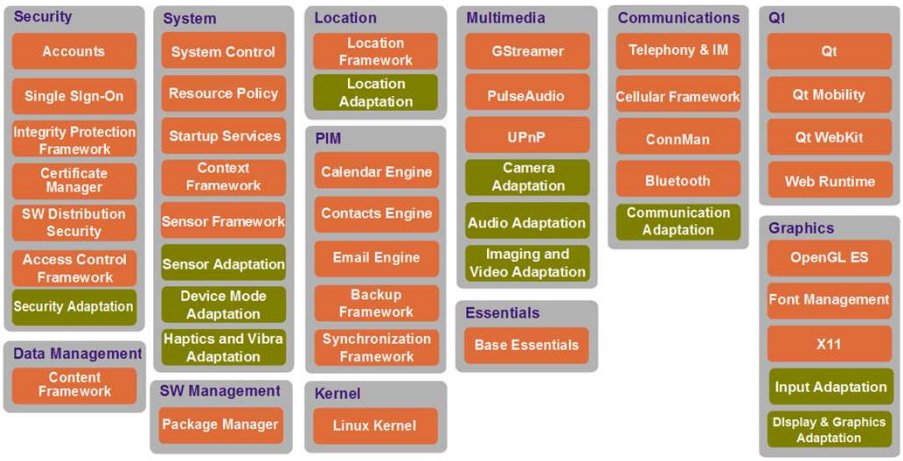
MeeGo Architecture Layers



digia

14

MeeGo Subsystem Components



15 © 2009 Digia Plc

MeeGo Application and Platform Development

MeeGo OSS Model

digia

Agenda

- MeeGo OSS Community
- GIT Repository
- Contributions

MeeGo OSS Community

- Led by MeeGo Technical Steering Group (TSG)
  - Under the auspices of Linux Foundation
- Actual work takes place in working groups
  - Handheld, netbook, in-vehicle, connected TV
- Bi-weekly TSG meetings open for anyone

GIT Repository

- MeeGo projects stored in meego.gitorius.org
  - QEMU MeeGo SDK, multimedia, trackers, bluetooth, cellular, handset and netbook UX, base, middleware
- New projects can be suggested
  - In a similar way as to submit a bug request
- Projects output one or more repositories organized in a directory hierarchy
  - New repositories may be suggested
  - All activities take place in repositories
    - Merge requests
- OpenSuse Build Service (OSB) is used to create and automatically build packages

digia

19

Contributions

- Team leaders and sub-system maintainers are selected based on meritocracy
- Each package has an owner
  - You may request to be an owner of a package
- Software development
  - Aligned closely with upstream projects
  - Patches accepted via the upstream project, if one exists
  - Follow the patch packaging guidelines and submit your package to bugzilla (bug report) or meego-dev@meego.com mailing list
- Marketing, MeeGo Greeter, documentation, testing

digia

20

MeeGo Application and Platform Development

MeeGo Development

digia

Agenda

- MeeGo SDK
- QtCreator
- Debugging
- On-Device Debugging
- Deployment
- RPM Packaging
- Image Creation
- Quality Assurance

digia

22

MeeGo SDK

- Supported CPU architectures
  - Ia32 – 32bit Intel Atom, Intel Core 2
  - Armv7l – ARM architecture version 7 compatible CPUs (Cortex-A8, Thumb-2, NEON, VPFv3)
- Current version 1.1
  - 1.2 RC 1 should be released on the 30th of March
  - 1.2 final version should be available on the 27th or April

digia

23

MeeGo Development

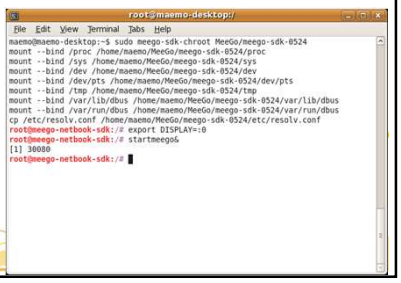
- Virtual environment
  - Using *chroot* or *scratchbox*
- Both virtualize the file system
- QtCreator the basic application and platform development tool
- QEMU
  - Emulates the real device code
  - Uses MADDE
  - Several run-times available
    - `mad list -v`
  - Packaging and deployment similar to QEMU and the real device
  - No need for the target device unless you want to test drivers

digia

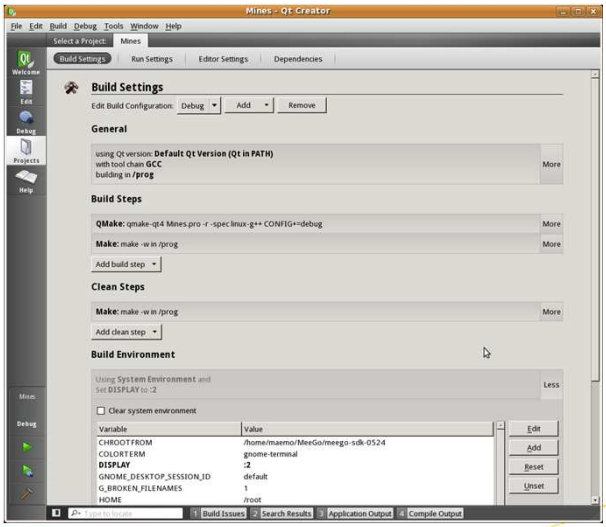
24

MeeGo SDK Setup - *chroot*

- Configure X to enable the MeeGo Simulator to access the display of the normal user
  - `xhost +SI:localuser:<user name>`
- Mount the MeeGo image
  - `sudo mount -o loop,offset=512 MeeGo/meego-handset-sdk-20101012-1.1.80.20101024.1603-sda.raw ImgMeeGo`
- Run the script to launch MeeGo virtual environment. The path depends on your actual installation path
  - `sudo meego-sdk-chroot ~/ImgMeeGo`
- Define a DISPLAY environment variable inside the virtual environment, because Xephyr does not see the variable in the host system
  - `export DISPLAY=:0`
- Run the Simulator (runs only with an Intel integrated graphics controller)
  - `startmeego &`
- Run any other programs
  - `DISPLAY=:2 glxgears`
- Exit from chroot
  - `exit`

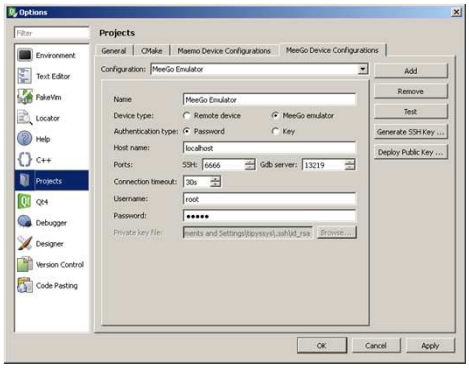


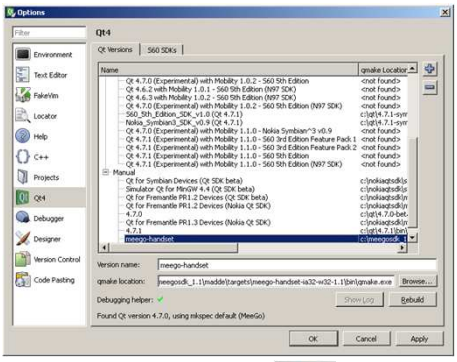
QtCreator with *chroot*



digia

QtCreator Setup for QEMU



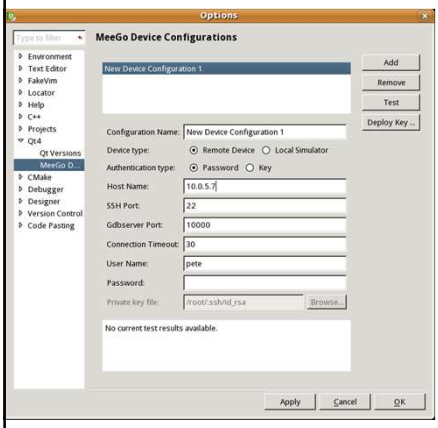


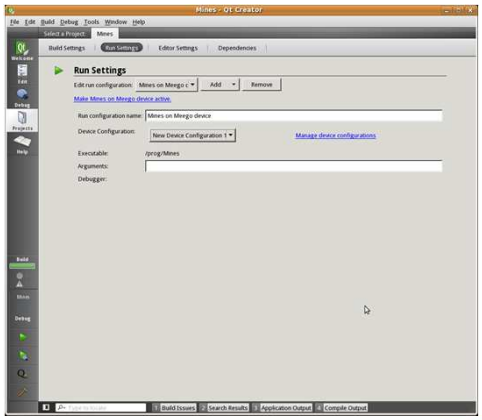
- Start QEMU (in Windows `net start kqemu`)
- Start the emulator
- Run/debug the application

digia

27 © 2009 Digia Plc

QtCreator Setup for On-Device Debugging





digia

28 © 2009 Digia Plc

On-Device Debugging

- For on-device debugging, you need SW to transfer your program to MeeGo device and debug it there
- For the deployment, Qt Creator uses SSH and for debugging GDB
- Install OpenSSH server on the device with:
  - `sudo zypper install openssh-server`
- To start it manually (you'll need to do this just after you've installed it, otherwise it won't be available until you reboot):
  - `sudo /etc/init.d/sshd start`
- To add it to the init sequence so it starts at boot time:
  - `sudo chkconfig --add sshd`
- Install GDB server on the device
  - `sudo zypper install gdb-gdbserver`

digia

29

RPM Packaging

- Copying the binaries directly to your device does not allow you to start your application from the MeeGo applications panel
  - You need to create an installation package, deploy that to the device and install it there
- MeeGo SW is deployed as RPM packages consisting of
  - A signature to verify the origin and integrity of the package
  - Metadata (name, version, architecture, authors etc.)
  - An archive of files to be installed on the destination file system (e.g. executables, images, documentation)

digia

30

Packaging an Application

- Copy your project (such as `mines`) into a `$workspace` folder and change to that folder
  - `cp –a mines/ $workspace/mines-0.0.1`
  - `cd $workspace/mines-0.0.1`
- Add an application icon (named as `mines.png`)
- Add a desktop file (`mines.desktop`)
  - Application will be shown in MeeGo application panel
- Edit the project file
  - Add icon and desktop file installations
- Enter the virtual environment or MADDE and create the source tarball, which is ready for packaging
  - `qmake PREFIX=/usr`
  - `make`
  - `make install`
  - `make distclean`
  - `cd ..`
  - `tar jcvf mines-0.0.1.tar.bz2 mines-0.0.1`

digia

31 © 2009 Digia Plc

Desktop File

- For GUI applications
  - http://standards.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html
  - Defined as key=value pairs
- Some keys and values
  - Type – Application, Link, Directory
  - Version – Desktop entry specification version
  - Name – Package name
  - GenericName – 2D game
  - NoDisplay – App not shown in the menu
  - Icon – Icon shown in the file manager
  - Categories – Audio, video, education, system, settings, graphics, development
  - StartupNotify – Notifies the system about startup
  - Terminal – Specifies whether the program runs in a terminal window
  - Exec – Program to execute with possible command line arguments

digia

32 © 2009 Digia Plc

Desktop File Example

```
[Desktop Entry]
Name=Mines
Comment=Minesweeper game
Exec=mines
Categories=Game;
Icon=mines
# Preferred to absolute path (.png, .svg, .xpm)
Type=Application
Terminal=false
StartupNotify=true
```

digia

33

Qt Project File Additions

```
# install
target.path = $$[install_prefix]/bin
icon.files = mines.png
icon.path = $$[install_prefix]/share/icons
desktop.files = mines.desktop
desktop.path =
  $$[install_prefix]/share/applications
INSTALLS += target icon desktop
```

digia

34

Create MeeGo Spec File

- Install Spectacle to your virtual environment
  - `yum install python-cheetah`
  - `yum install spectacle`
- Create YAML package meta-data file `$workspace/mines.yaml`
- Generate spec file from YAML meta-data file
  - `specify mines.yaml`

digia

35

YAML Package Meta-Data File

```
Name: mines
Summary: Minesweeper game
Version: 0.0.1~beta1
Release: 1 # handled by the build system
Group: Amusement/Games # Development/Tools
License: LGPLv2.1
URL: http://qt.nokia.com
Sources:
   - "%{name}-%{version}.tar.bz2"
Description: Minesweeper game
PkgConfigBR:
   - QtCore >= 4.6.0
   - QtGui
Configure: none
Builder: none
Files:
   - "%{_bindir}/mines"
   - "%{_datadir}/applications/*.desktop"
   - "%{_datadir}/icons/*.png"
```

digia

36

Build Pre and Install Post Sections

- Modify the mines.spec to add Qt build in "build pre" and "install post" sections.

```
# >> build pre
export PATH=/usr/lib/qt4/bin:$PATH
qmake PREFIX=%{_prefix}
# << build pre

# >> install post
make INSTALL_ROOT=%{buildroot} install
# << install post
```

digia

37

Create the RPM Package

- Install rpmbuild and MeeGo rpm build configuration (*chroot*)
  - `zypper install rpmbuild`
  - `zypper install meego-rpm-config`
- Copy source code and spec file to right place
  - `cp mines-0.0.1.tar.bz2 ~/rpmbuild/SOURCES/`
  - `cp mines.yaml ~/rpmbuild/SOURCES/`
  - `cp mines.spec ~/rpmbuild/SPECS/`
- Generate rpm package
  - `cd ~/rpmbuild/SPECS rpmbuild -ba mines.spec`
- Then the rpm packages will be generated at:
  - `~/rpmbuild/RPMS/i586/mines-0.0.1-1.i586.rpm`
  - `~/rpmbuild/SRPMS/mines-0.0.1-1.src.rpm`

digia

38

Any Simpler Ways to Create an RPM or Debian Package

- QtCreator
  - Not a complete but installable RPM package
  - E.g. no icon
  - Spec file skeleton created
- MADDE
  - A skeleton for a Debian package
  - `mad -t <target> pscreate -t qt-simple qthello`
  - `cd qthello`
  - `mad -t <target> qmake`
  - `mad -t <target> make`
- Eventually, in Nokia Qt SDK + remote compiler

digia

39

OBS – OpenSuse Build Service

- Contains projects
  - Projects contain resources to build one or more packages
  - The project outputs one or more repositories organized in a directory hierarchy
    - Official projects, factory projects, user-specific projects home:username
- Create a local project
  - Using a web client (http://build.opensuse.org/) or
  - Osc – OBS Client
    - Download the command line tools from http://software.opensuse.org/download/openSUSE:/Tools/
    - Create a project

```
cd <directory_to_contain_project_root>
osc checkout home:<username>
cd home:<username>
```

digia

40

OBS Usage

- Create and upload packages
  - Create package folder with template XML
    - `osc meta pkg -e home:<username> <packagename>`
  - Create the folder with package name
    - `osc up`
  - Add the files (.spec file) and submit
    - `osc add *`
    - `osc commit`
- Choose build targets
  - Check available repositories
    - `osc ls`
  - Edit the metadata
    - `osc meta prj -e home:<username>`
  - And add the reposiroty

```
<repository name="openSUSE_Factory">
    <path project="openSUSE:Factory" repository="standard"/>
    <arch>x86_64</arch>
</repository>
```

41 © 2009 Digia Plc

OBS – Package Build

- The package will be built automatically
- For manual build, use
  - `osc rebuildpac <project> <package> [<repo> [<arch>]]`
- To build locally, use
  - `osc build <platform> <arch> <specfile> [--clean|--noinit]`

42 © 2009 Digia Plc

QEMU Benefits

- Anytime MeeGo is running in QEMU, you can use SSH to connect to the image from another terminal on the host system
- Launch QEMU with MeeGo image
- From a terminal on the host, connect to the running image
  - `ssh meego@127.0.0.1 -p 6666`
- Two user accounts are available in the MeeGo images:
  User: meego, password: meego
  User: root, password: meego
- Install SW to the image without creating a new image

digia

43

Image Creation

- MeeGo images are crated with MIC2
  - MIC = Moblin Image Creator
- Possible to create
  - Live CD images
  - Live USB images
  - Raw images for KVM
  - VMDK images for Vmware
  - vdi images for VirtualBox,
  - loop images for IVI platforms
  - NAND images for Moorestown platforms
  - ubi images for N900
  - fs image for MeeGo developers

digia

44

MIC2 Tools

- mic-image-creator: create images
- mic-image-convertor: convert a raw/vmdk/vdi/live image into a live image
- mic-chroot: provide a MeeGo environment from a live/loop image for development, it also can translate that chroot file system into a live image
- mic-image-writer: write a MeeGo image to a USB disk

digia

45

Image Creation – 1(2)

- Download MIC2 either from http://meego.gitorious.org/meego-developer-tools/image-creator or in Debian-based package system
  - Add deb http://repo.meego.com/tools/repos/debian/5.0/ / to /etc/apt/sources.list and
  - Apt-get install mic2
- Get MeeGo kickstart file
  - Defines which repos to pull files from, which packages to include, which scripts to run, which image type to create http://repo.meego.com/MeeGo/builds/trunk/
  - The official MeeGo .ks files for ARM based Nokia N900 <version>/handset/images/meego-handset-armv7l-n900/
  - The official MeeGo .ks files for Intel Atom based netbook and handset (Moorestown)
    - <version>/{netbook,handset,ivi}/images
  - You can download and use them as a base for the MeeGo images you create. Modify these .ks files as you wish to create tailored images

digia

46

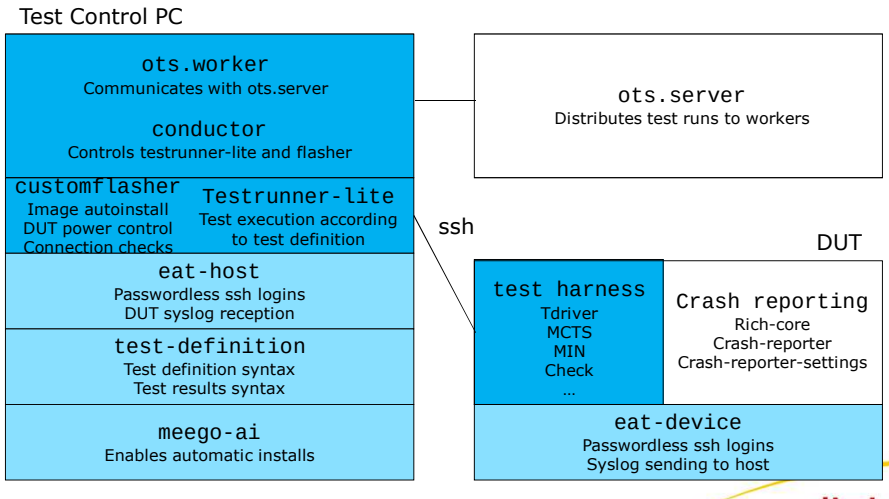
Image Creation – 2(2)

- Ceate a MeeGo liveusb image that you can transfer to a USB stick
  - `sudo mic-image-creator --config=default.ks --format=liveusb --cache=mycache`
- A file named meego-1.0-default-XX.usbimg will be created
- To burn it onto a USB stick, run the following command:
- `sudo mic-image-writer meego-1.0-default-XX.usbimg`

digia

47

Quality Assurance



digia

48

QA Process

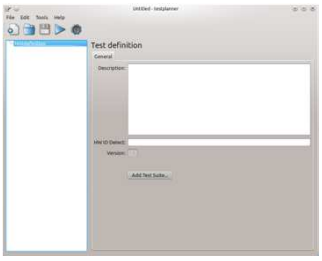
- Create test plan files with *Testplanner* or manually
- *Test plan* XML defines tests in test executable independent way
- Test plan information is stored in *Test definition*
  - Test definition is used to validate the test plans
  - Validation scheme included, if the testplanner downloaded from the repository
- Test plans are executes with *Testrunner* and *testrunner-lite*
  - Plans can be stored in git

digia

49

Test Planning

- A test plan consists of a test suite, test set, and test case
  - Test cases define commands executed on the target device
- Test suite with attributes
  - Inherited to test set and cases
- Test set
  - Pre and post steps
- Test case
  - Test steps with expected result "0"
  - Executed by *testrunner-lite*



- MeeGo Core Test Suite, Handset-UX tests, and MeeGo NetBook Test Suite exist

digia

50

Test Case Example

```
<case name="x-test-case" type="Functional">
      <description>Dummy test case</description>
      <step expected_result="-1">ls</step>
      <step>uname -r</step>
</case>
```

digia

51

Test Deployment

- Load or create a test plan
- Package your test
- Create the image with the test plan (or copy it to the device)
  - Add the location of your tests
    - *repo –name=home-timoph –baseurl=http://repo.pub.meego.com/home:/timoph/meego_current_extras_handset/ –save –debuginfo –source –gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-meego*
  - Add the package into the %packages section
    - Some-nice-tests
  - Finally, create the image
    - *sudo mic-image-creator –run-mode=0 –cache=mycachedir –format=raw –arch=armv7l –save-kernel –config=meego-handset-n900-host-based-testing.ks*

digia

52

Test Execution 1(2)

- Connect the USB cable to your device and configure that
  - *sudo ifconfig usb0 192.168.2.14 up*
- Enable passwordless logins to the device under test
  - Generate a key pair with ssh-keygen, unless you have a key
  - Copy your public key to the device's authorized keys list
    - *cat ~/.ssh/id_rsa.pub | ssh root@192.168.2.15 "cat >> ~/.ssh/authorized_keys"*
  - Test the passwordless login
    - *ssh root@192.168.2.15*

digia

53

Test Execution 2(2)

```
#!/bin/sh
# copy test.xmls from dut
scp -r root@192.168.2.15:/usr/share/*-tests .
# run tests
for i in `ls`
do
# remove the –automatic switch if you want also to run manual #
cases
testrunner-lite -f $i/tests.xml -v -o $i-results.xml -t
root@192.168.2.15 –automatic
done
return $?
```

digia

54

Test Reports

- Template under development
- Quality summary
  - Improvements compared to the previous build
  - Key issues compared to the previous build
- Test coverage
  - Sanity test, basic feature test, extended feature test, system performance test, system reliability test
- Testing Summary
  - Passed, failed, measured, executed, run rate, pass rate of total
- Testing details
  - Objective, location, environment, issue summary

digia

55

Summary

- There are several ways to develop SW for MeeGo
  - Nokia prefers currently Scratchbox
  - QEMU is very efficient, because almost the same image as in the device can be executed
- QtCreator is the tool for development and debugging
- MeeGo packaging system is based on RPM packages
  - Based on the .spec file
  - Easy to create with QtCreator
- Target images are created with MIC2 and kickstart files
- You do not need to put everything to the image
  - Use ssh to install test packages, SW or anything else to the target device or QEMU

digia

56

MeeGo Application and Platform Development

MeeGo API

digia

Agenda

- Handset and Netbook UX
- MeeGo Touch

digia

58

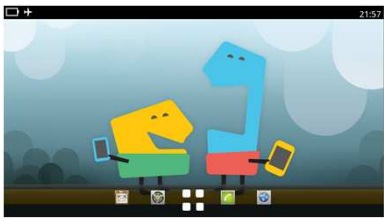
MeeGo API

- Currently, Qt 4.7
  - Qt modules: QtCore, QtGui, QtWebKit, QSvg etc.
- Qt Mobility 1.1
  - Extends Qt with platform functionality such as bearer management, service framework, messaging, navigation
- MeeGo Web Runtime
  - Allows applications development without C++ using Web technologies

digia

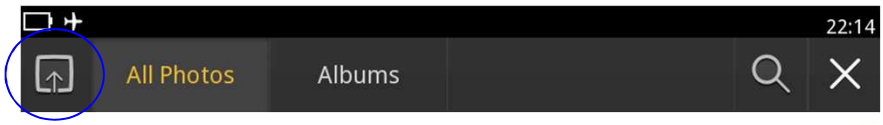
59

UI Model



Home screen with 4 app buttons and App launcher button

App launcher with app icons in two pages



Button to App switcher

digia

60

UI Guidelines

- Keep it simple
  - Keep the most important information at the top
  - Less important can be drilled down
  - Keep your application finger usable
- Do not hide information
  - Avoid scrolling except in lists or pannable windows
  - It may not be obvious that some functionality is available only if scrolled down
- MeeGo Touch common components help creating a consistent UI
- Design the application for multitasking
  - How to switch between two or more?
  - Is your application recognizable in the switcher thumbnail?

digia

61

Application Types

- Identify your application type
- Productive
  - Pragmatic tasks, such as sending a message
  - Simple, intuitive, typically drill-down and simple toolbar
  - Often using common components
- Immersion
  - Entertaining or visually rich UI
  - Video player, map player, games
  - Typically, full screen
  - Possibly, customized layout and controls

digia

62

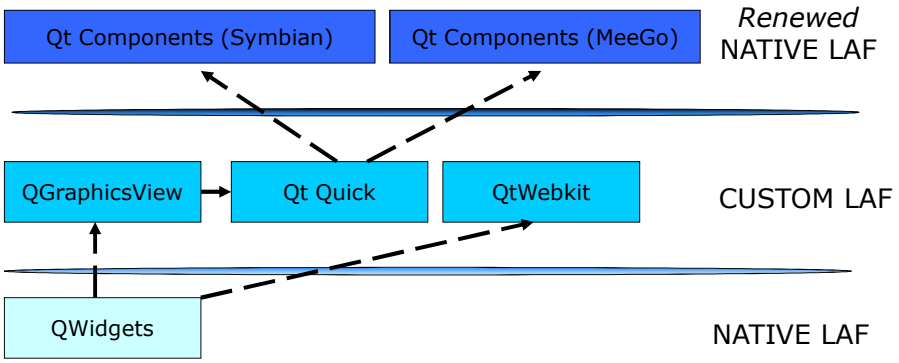
Layout Recommendations

- Productive apps often use lists
  - Good for pragmatic tasks
- Content or functionality
  - Maximize the use of content on the screen
  - Many images in the grid
  - Functionality may be flat (menu) or drilled down
- Landscape or portrait
  - Landscape provides better video watching experience
  - Lists are nicer to scroll in portrait orientation
  - In many cases, both orientation can be supported

digia

63

Mobile Qt UI Offering, Prospects



digia

64

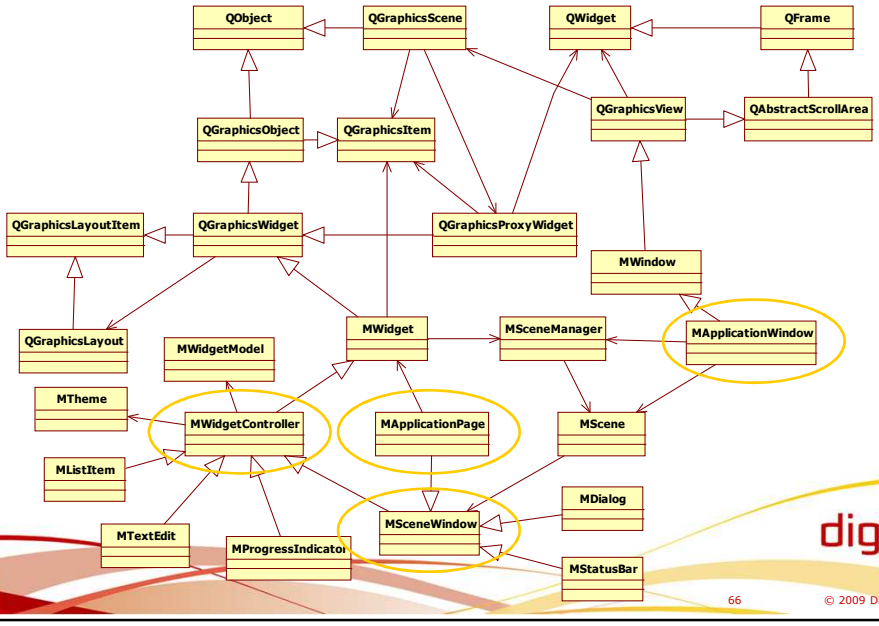
MeeGo Touch

- Cross-platform
- Style for finger optimized applications from handhelds to the desktop
  - Highly tactile user experience with panning, flicking, dragging, and pinching
- Direct manipulation of content
- Application structure based on views and transitions
  - Rather than multiple windows
- Graphics View-based
  - Allows creation of fluid user interfaces
- Typical architecture
  - MeeGo Touch UI
  - Qt application engine (with mobility APIs)

digia

65

MeeGo Touch Extends Graphics View

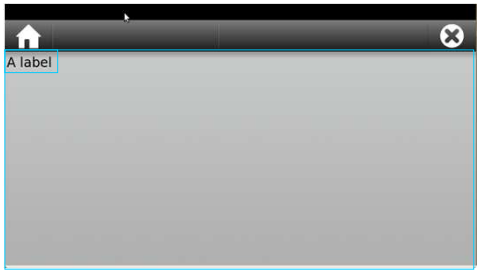


digia

66

Application Structure

```
#include <MApplication>
#include <MApplicationWindow>
#include <MApplicationPage>
#include <MLabel>

int main(int argc, char **argv)
{
    MApplication app(argc, argv);
    MApplicationWindow w;
    w.show();
    MApplicationPage p;
    p.appear(&w);
    MLabel b("A label", p.centralWidget());
    return app.exec();
}
```



digia

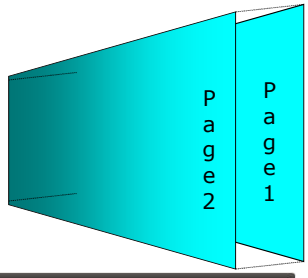
Application Window - MApplicationWindow

- Derived from QGraphicsView and MWindow
- MWindow completely empty
  - OpenGL viewport by default
  - MApplicationWindow provides MeeGo Touch UI style
- Window orientation
  - Signals and slots to set and notify orientation changes (0, 90, 180, 270 degrees – keyboard opened/closed)
- Scene management
  - MSceneManager – Manages in-scene UI elements (MSceneWindow objects) such as popups, dialogs, and menus and their transitions (appearance and orientation)

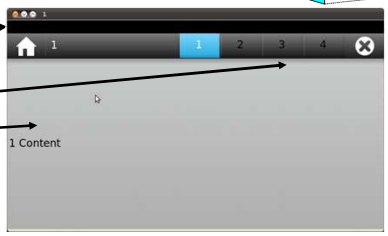
```
MWindow w;
MSceneWindow *sw = new MMessageBox("Hi");
w.show();
sw->appear(&w);
a.exec();
```

digia

Scene



- Application window creates a scene and a scene manager
- The scene stores scene windows and manages the Z order
  - Status bar
  - Navigator bar with home and escape buttons
  - Toolbar (tabbar)
  - Application page
  - Dialog, menu, banner, …
- The application page
  - Corresponds to a view
  - Has actual content in a central widget



digia

69 © 2009 Digia Plc

Pages

- MApplicationPage objects
- Pannable
  - Possible to set direction
- May have action (MAction) in the menubar
- May have a title
- Shows or hides the navigation bar

```
page->setComponentsDisplayMode( MApplicationPage::AllComponents,
   // EscapeButton, HomeButton
   MApplicationPageModel::AutoHide); // Show, Hide, Autohide
```

```
MApplicationWindow w;
MApplicationPage p;
new MButton("Press", p.centralWidget());
p.appear(&w);
w.show();
```

digia

70 © 2009 Digia Plc

Page Navigation

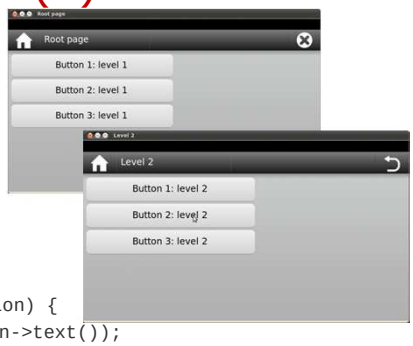
- Only one page displayed at any time
- Method appear()
  - Stores current page to the page history
  - Displays the new page
  - Possibly changes the escape button mode to back
    - If MApplicatiionPageModel::EscapeAuto set or
    - If MApplicationPageModel::EscapeManual set
      - Only emits backButtonClicked() signal if escape button pressed
    - And page history not empty

```
void showPage() {
  MApplicationPage *aboutPage = new AboutPage();
  aboutPage->appear(scene(),
   MSceneWindow::DestoryWhenDismissed); }
```

digia

71 © 2009 Digia Plc

Navigation Patterns – 1(2)

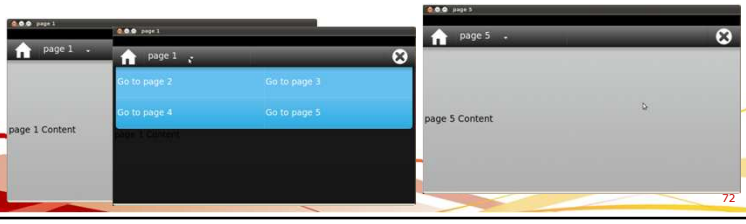


- Drill down
  - Very scalable
  - Structure visible at top level

```
nextPage->appear(scene(),
   MSceneWindow::DestroyWhenDismissed);
```

- Application menu
  - Flat experience
  - Content prioritized over navigation

```
void Window1::handleActions(QAction *action) {
   MApplicationPage *p = createPage(action->text());
   p->appear(this, MSceneWindow::DestroyWhenDone);
```



digia

72 © 2009 Digia Plc

Navigation Patterns – 2(2)

- Tabbar
  - Quick access between distinct areas or modes
  - At most four navigation options
  - Options should not change over time

```
QAction *Window1::createAction(const QString &name, bool checked) {
  MAction *a = new MAction(name, this);
  a->setLocation(MAction::ToolBarLocation);
    // ApplicationMenuLocation
  a->setCheckable(true);
  a->setChecked(checked);
  addAction(a);
  return a;
```

Central Widget - Layout

- Two options
  - QGraphicsLayout-based QGraphicsLinearLayout or QGraphicsGridLayout
    - Easy, if nested layouts are needed
  - MLayout-based on multiple policies (e.g. portrait and landscale)
    - Slightly heavier than QGraphicsLayout-based layouts
    - Different policies may have different items
- Layouts do not have widget children
  - Layouts may have children layouts
  - Widgets' parent is the widget on which the layout is installed

Layout Policies

- MLinearLayoutPolicy

```
MLayout *l = new MLayout;
MLinearLayoutPolicy *p = new MLinearLayoutPolicy(l,
   Qt::Vertical);
policy->addItem(uiItem);
```

- MGridLayoutPolicy

```
policy->addItem(uiItem, row, col);
```

- MFlowLayoutPolicy
  - Preferred size and size policy used in layout

```
policy->setRowLimit(nofRows);
```

- MFreestyleLayoutPolicy

```
policy->addItemAtGeometry(uiItem, QRect(itemCenterX,
   itemCenterY, itemWidth, itemHeight);
```

digia

75 © 2009 Digia Plc

Multiple Policies

- Landscape orientation with linear policy
  - 3 items shown
- Portrait orientation with grid policy
  - 5 items shown

```
MLayout *layout = new MLayout(centralWidget());
MLinearLayoutPolicy *linearPolicy = new
MLinearLayoutPolicy(layout, Qt::Vertical);
MGridLayoutPolicy *gridPolicy = new
   MGridLayoutPolicy(layout);

layout->setLandscapePolicy(linearPolicy);
layout->setPortraitPolicy(gridPolicy);
// …
layout->addItem();
```

digia

76 © 2009 Digia Plc

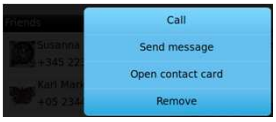
Controls – Common Components

- MWidget – QGraphicsWidget
  - Gesture event support
- MWidgetController – MWidget
  - Controller in the MVC widget model
  - Stores the widget's state to the model and delegates painting and event handling to the view
  - Initialized model provided in subclasses
  - MWidgetModel and MWidgetView are QObject type
- Common components
  - MeeGo Touch look and feel
  - Designed for finger use
  - Fully stylable through CSS
  - Theming support
  - Optimized performance and power consumption

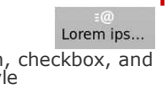
digia

77 © 2009 Digia Plc

Common Components



- MButton
  - Push, icon, checkbox, and switch style

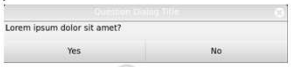


- MList
  - For large number of items

- MTextEdit
  - Supports on-screen virtual keyboard

- MLabel
  - Piece of text, no icon
- MImageWidget
- MVideoWidget
- MCombobox
  - Popup selection list
- MProgressIndicator
  - Bar and spinner type

- MObjectMenu
  - Tap and hold
  - Similar to context menu
- MApplicationPage
- MApplicationMenu
  - Prefer using actions in application page directly
- MDialog
  - Especially for modal functionality



- MSlider

- MSeekbar
  - Unlike slider, supports visualizing amount of loaded content
- MBanner
  - Local to the application
  - Notification framework allows global notifications

digia

78 © 2009 Digia Plc

Styling

```
MTheme::loadCSS("mgirdlayoutpolicy.css");
MLayout *l = new MLayout;
MGridLayoutPolicy *p = new MGridLayoutPolicy(l);
p->addItem(someUiItem, row, col);
someUiItem->setObjectName("uiItem");
```

- Style
  - Container storing styling attributes for an object
- Styling attribute
  - Type, name, and value
- Style can inherit from another style

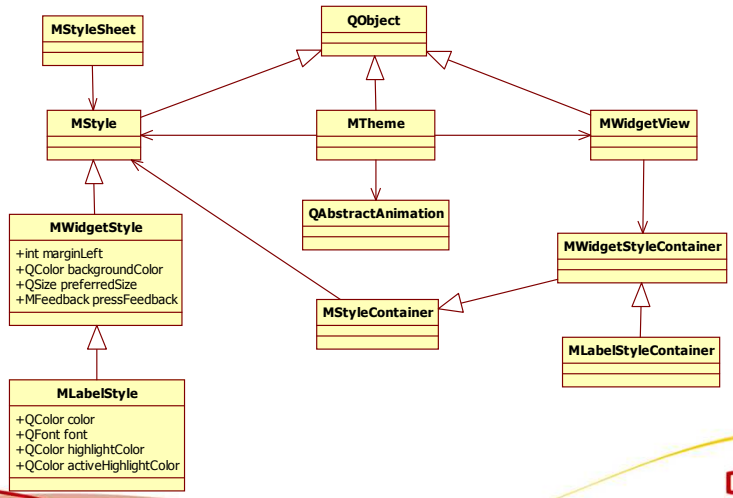
```
#uiItem {
  font: "Nokia Sans Wide" 15px;
  background-color: #777777;
  preferred-size: 100 100;
}
```

- Theme provides styles for the stylable object
- Style contains a set of style attributes
- Style gets initialized from the CSS style sheets

digia

79 © 2009 Digia Plc

Theme and Style Classes



digia

80 © 2009 Digia Plc

MeeGo Touch Debug Information

- Use mDebug() and mWarning()
  - Like qDebug() and qWarning() but take additional module name argument
- Use debug messages for statements for future debugging
- Do not repeat warning messages given by return value, for example

```
#include <MDebug>

mDebug("My_module::functionX()" << "X completed");
```

digia

81

Animations

- Similar to Qt Animation Framework
  - QPropertyAnimation animates an object property from a startValue to endValue possibly using an easing curve
- MParallelAnimationGroup extends from QParallelAnimationGroup
  - Adds styling support allowing CSS attributes to fill in animation values
  - The animations of the framework (pages, dialogs, menus, rotation, etc.) are private, but can be tweaked in the CSS
  - Derive a sub-class and implement the style properties of animations by deriving from MAnimationStyle

digia

82

Animation Example

```
class FadeOutAnimation : public MParallelAnimationGroup
{
   Q_OBJECT
   M_ANIMATION_GROUP(FadeOutAnimationStyle) // Animation style class
public:
   FadeOutAnimation(QGraphicsItem *i, QObject *p = NULL)
: MParallelAnimationGroup(parent) {
      QPropertyAnimation *opacity = new QPropertyAnimation;
      opacity->setPropertyName("opacity");
      opacity->setEasingCurve(style()->easingCurve());
      opacity->setDuration(style()->duration());
      opacity->setEndValue(0.0);
      opacity->setTargetObjectName(i);
      addAnimation(opacity);
   }
}
```

digia

83

Gestures

- Qt and MeeGo Touch support the same gestures
  - Pan, pinch, swipe, tap, and tap and hold
  - MWidget class has the event handler functions for these
  - Note that by default mutlitouch and gesture events are not delivered to widgets
    - Call setAcceptTouchEvents(true); to enable multitouch
    - Call QGraphicsItem::grapGesture(Qt::PinchGesture) to enable gesture event delivery
- MeeGo Touch default gestures
  - MApplicationPage (MPannableViewport) consumes QPanGesture, if the page is pannable
  - Any MWidget with one or more actions (either Qt or MeeGo touch) consumes QTapAndHoldGesture and opens the context menu (MObjectMenu) when the gesture completes

digia

84

Application Lifecycle

- Number of concurrent applications is not restricted
  - Application on the foreground is prioritized
  - MeeGo Touch does some optimization for background apps, but apps can still misbehave and drain the battery
  - Fine-grained application state information available
- Applications are singleton by default
  - Launching again raises the existing window or creates a new window in the same process
  - Applications can be pre-loaded, lazy-shutdown, and boosted

digia

85

Other Issues

- Internationalization
- Input Feedback
- Notifications
- Fast application startup
- Application extensions
- Service framework

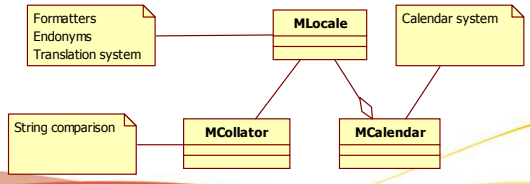
digia

86

Internationalization

- MeeGo Touch UI internationalization is a superset of Qt internationalization
  - Use of MeeGo Touch classes recommended
- MLocale – reads locale settings from GConf
  - Language (main language)
  - Time (date, time, calendar)
  - Collate (sorting)
  - Numeric (number formatting)
  - Monetary (formatting amounts of money)

```
MLocale l("fi_FI");
l.setCollation(MLocale::StrokeCollation);
```



Input Feedback

- Feedbacks are played back by a feedback daemon
- MeeGo Touch provides a simple API to use the feedbacks

```
MFeedback *fb = new MFeedback("press"); // release, cancel
// fb->play();
connect(src, SIGNAL(fired()), fb, SLOT(play()));
// MFeedback::play("press");
```

Notifications

- For global notifications
  - Battery low
  - Not for "File deleted" within an application (use the banner)
- MNotification derived from QObject

```
MNotification m(eventType, summary, body);
bool ret = m.publish();
// TransferCompleteEvent, NetworkEvent,
// PresenceOnlineEvent, ImErrorEvent,
// DeviceErrorEvent, MessageArrivedEvent
```

digia

89

Fast Application Startup

- Application launcher
  - Preloads (booster) large libraries and instantiates MApplication and MApplicationWindow (main() not called)
  - Invoker sets the application to the booster forked process and calls main()
- Application pre-starting
  - Only a small set of critical applications
- Lazy shutdown
  - Application is closed but not terminated

digia

90

Boosting Startup Time

- Put MApplication and MApplicationWindow instance into component cache

```
MApplication *a = MComponentCache::mApplication(argv, argc);
MApplicationWindow *w = MComponentCache::mApplicationWindow();
```

- Main function must be visible to invoker

```
#include <MExport>
M_EXPORT int main(int argc, char **argv) {
```

- Use the compilation and linking flags to have location independent executable

```
CONFIG += meegotouch-boostable
```

- Install applaunherd package and use it to run the program

```
invoker –type=m /usr/bin/app
```

digia

91

Application Extensions

- Plug-ins extending the application
- Written as Qt plug-ins
- The base class of all application extensions is MApplicationExtensionInterface (requires libmeegotouch-dev)
  - `bool MApplicationExtensionInterface::initialize(const QString &interface)`
- Create the extension in the application
  - `new MApplicationExtensionArea(interface, parent)`
- Write a desktop file to publish the extension

digia

92

Service Framework

- High-level IPC mechanism on the top of D-Bus
- Service user instantiates an interface
  - Interface asks the service mapper for a name of a service provider
- The service mapper maintains a map of services
  - Using /usr/share/dbus-1/services
  - Selects a service and returns the name of the interface
- The interface makes a regular D-Bus connection to the service name
- The service mapper uses signals to indicate new service providers for the interface or if there are no more service providers for a certain interface
  - maemo-meegotouch-interfaces-dev and libmeegotouch-dev Debian packages needed

digia

93

MeeGo Application and Platform Development

Qt Quick

digia

Agenda

- Qt Quick
- QML Essentials
- Layouts
- User Interaction
- States, Transitions, Animations
- Data Models and Views
- C++ Bindings

digia

95

What is Qt Quick?

- "Qt User Interface Creation Kit"
  - An umbrella over concepts such as QML and QtDeclarative
  - We will take a look at these separately during the course
- A high-level UI technology for easily creating attractive UIs
  - No C++ skills needed, knowledge of JavaScript helps quite a bit
- Aimed at both UI designers and developers alike
  - Enables designers and developers to "speak the same language"!
  - Both parties can be involved in iterative development simultaneously
  - No need for separate Flash or PowerPoint UI prototypes
- Officially launched in Qt 4.7.0
  - Released 9/2010

digia

96

QML vs. QtDeclarative

- **QML:** ”Qt Meta-Object Language”
  - A declarative scripting language for defining the elements of a graphical UI
  - Actually an extension to ECMAScript (cf. JavaScript)
  - Interpreted at runtime – not compiled!
- **QtDeclarative:** a new C++ module in Qt (since Qt 4.7.0)
  - Contains the QML runtime environment
  - Also provides facilities for embedding QML content into Qt/C++ applications
  - Implements the necessary QML <-> Qt/C++ bindings

digia

97

Development Tools

- QtCreator is the de facto tool for Qt development
  - Also for Qt Quick, of course!
  - Just download Qt 4.7.x SDK and you are all set to go
  - http://qt.nokia.com/downloads
- A newer QtCreator 2.1 beta 2 also available separately
  - Enhanced development tool support (designer) for Qt Quick
- A tool called *qmlviewer.exe* is used for running standalone QML/JavaScript applications
  - I.e. allows viewing **`.qml`** files on their own
  - *Meant for development and testing purposes only!*
  - Comes with the SDK

digia

98

Qt Quick

QML Essentials

digia

Introduction

- As mentioned, QML is a declarative language for defining how:
  - An application looks like, and
  - How it behaves
- A QML UI is composed of a tree of elements with certain properties
  - There are both *graphical items* and *behavioral elements*
  - These are combined into *QML documents* to build more complex components and full QML applications
- Can be used to extend existing Qt applications or to build completely new ones
  - QML itself is also fully extensible with Qt/C++!

digia

100

QML Syntax, First Sneak Peek

```
/* woodenhead.qml starts here, with a
multiline comment */
import Qt 4.7

Rectangle {
   width: 350 // Single line comment
   height: 2 * 100
   color: "lightblue"
}
```


- Let's start off with a simple example: a light blue rectangular area on the screen
- Easy to read and understand?
  - Very "JavaScriptish", right?

digia

101 © 2010 Digia Plc

Example Concepts, `import`

```
/* woodenhead.qml starts here, with a
multiline comment */
import Qt 4.7

Rectangle {
   width: 350 // Single line comment
   height: 2 * 100
   color: "lightblue"
}
```


- To built-in QML elements, import the Qt module with `import Qt 4.7`
  - Syntax changedto `import QtQuick 1.0` in Qt 4.7.1 – the older still works as well
- Specify a version to get the features you want
  - Only imports the features from that version
  - Will not use features from later versions, even if available
  - Will not fall back to features from an earlier version
- Guarantees that the behavior of the code will not change
  - Modules include support for multiple versions
  - An upgraded module retains support for older versions of itself

digia

102 © 2010 Digia Plc

Example Concepts, Comments

```
/* woodenhead.qml starts here, with a
multiline comment */
import Qt 4.7

Rectangle {
   width: 350 // Single line comment
   height: 2 * 100
   color: "lightblue"
}
```


- Use // to add single line comments
- Put multi-line comments inside /* and */

digia

103 © 2010 Digia Plc

Example Concepts, Elements

```
/* woodenhead.qml starts here, with a
multiline comment */
import Qt 4.7

Rectangle {
   width: 350 // Single line comment
   height: 2 * 100
   color: "lightblue"
}
```


- Declare the elements you want to use
  - Exactly one main element in the file, the root element
- Each element has a body between { and }
- A set of default elements are included in the Qt module

digia

104 © 2010 Digia Plc

Standard QML Elements

- A number of ready-made graphical *QML items* are provided for convenience
  - `Item`, `Rectangle`, `Image`, `Text`, `MouseArea`, `WebView`, `ListView`, …
  - Some of them can be used as containers (parent) for other elements (children)
  - All graphical items in the UI inherit the `Item` element
- There are also non-graphical *QML elements*
  - Used for describing the *behavior* of the application
  - `State`, `PropertyAnimation`, `Transition`, `Timer`, `Connection`, ...

digia

105

Item Element

- Has no visual appearance, but defines all the properties that are common to all UI elements
  - As mentioned, all UI elements inherit the `Item` element
  - In Qt/C++ terms, is a `QDeclarativeItem`
- Provides, for example:
  - X, y, z position
  - Width and height
  - Anchors (explained later)
  - Opacity, rotation, scale
  - Visibility (true/false)
  - Parent and children
  - Key event handling
  - …

digia

106

Example Concepts, Properties

```
/* woodenhead.qml starts here, with a
multiline comment */
import Qt 4.7

Rectangle {
   width: 350 // Single line comment
   height: 2 * 100
   color: "lightblue"
}
```


- Elements have properties
- Each property is defined using its name and a value
  - `name: value`
  - Value can also be a piece of JavaScript
- Multiple properties can be declared on the same line
  - Separate with semi-colons

```
Rectangle {
   width: 350; height: 2 * 100
   color: "lightblue"
}
```

digia

Properties

- QML supports properties of many types
  - Int, bool, real, color, string, list, …
- Properties are type-safe
  - I.e. assigning a string where and integer is expected is not allowed

```
Item {
   x: 10.5 // a 'real' property
   ...
   state: "details" // a 'string' property
   focus: true // a 'bool' property
}

Item {
   x: "hello" // illegal!
}
```

Property Examples

- **Standard properties** can be given values:

```
Text {
    text: "Hello world"
    height: 50
}
```

- **Grouped properties** keep related properties together:

```
Text {
    font.family: "Helvetica"
    font.pixelSize: 24
}
```

- **Identity property** gives the element a reference:

```
Text {
    id: label
    text: "Hello world"
}
```

digia

Colors

- The colors of elements can be specified in many ways:
- As a named color in a string (using SVG names)
  - "red", "green", "blue", ...
- With RGB color components in a string (with alpha)
  - "#ff0000", "#008000", "#0000ff", ...
- Using a built-in function (red, green, blue, alpha)
  - Qt.rgba(0, 0.5, 0, 1)
- With an opacity using the opacity property
  - Values from 0.0 (transparent) to 1.0 (opaque)

digia

Example Summary

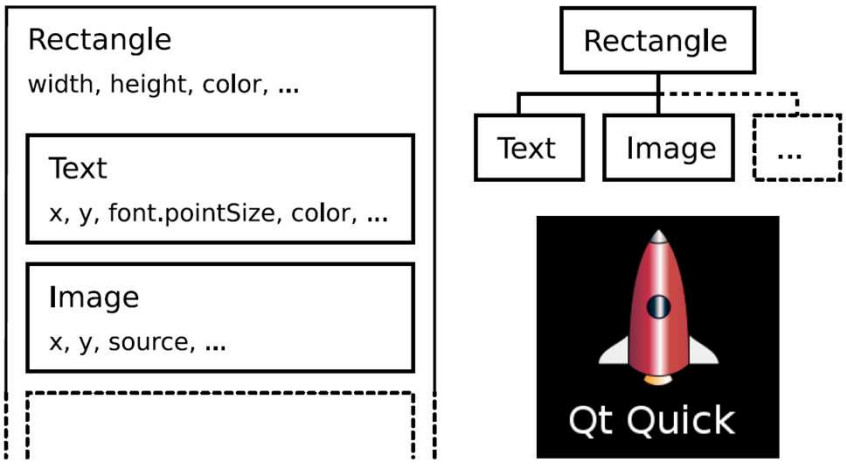
```
/* woodenhead.qml starts here, with a
multiline comment */
import Qt 4.7

Rectangle {
   width: 350 // Single line comment
   height: 200
   color: "lightblue"
}
```


- A *Rectangle item* with a body: { … }
- Three properties: width, height and color
- When executed in qmlviever, will produce a light blue window of size 350x200

digia

111 © 2010 Digia Plc

Tree of Elements

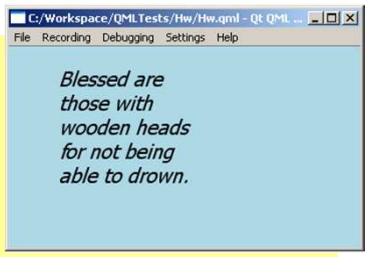


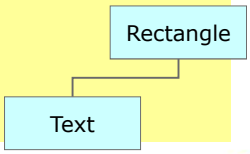
digia

112 © 2010 Digia Plc

First Actual Example

```
/* woodenhead.qml starts here, with a
multiline comment */
import Qt 4.7

Rectangle {
   width: 350; height: 2 * 100
   color: "lightblue"
   Text {
      x: 50; y: 20; width: 150
      wrapMode: Text.WordWrap
      font.pixelSize: 20; font.italic: true
      text: "Blessed are those with wooden heads
             for not being able to drown."
   }
}
```





digia

Text Element

```
Text {
     text: “Hello!”
}
```

- Simple text display
- Width and height determined by the font metrics and text
- Can also use rich text
  - Use HTML tags in the text: "Qt Quick"
- Editable Text with `TextInput`
  - No decorations, not a `QLineEdit`

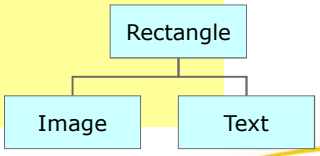
```
TextInput {
     width: 300
     text: “Editable hello!”
}
```

- Gets focus when clicked, but needs something to click on (either text or a width)
- `text` property changes when edited

digia

First Actual Example, Adding an Image

```
…
Rectangle {
   width: 350; height: 2 * 100
   color: "lightblue"

   Image {
         width: parent.width
         source: "justiina.jpg"
      }
   // Text element as earlier
   Text {
      …
   }
}
```



digia

Images

```
Image {
    width: parent.width
    source: "justiina.jpg"
}
```

- `source` contains a relative path
  - `"../"` refers to the parent directory
  - Adress can also be a URL, local file or a file in a resource
- `width` and `height` are obtained from the image file by default
  - If explicitly given, image gets scaled
  - Use property `fillMode` to affect aspect ratio
- Scale the image with `scale` property and rotate with `rotate` property (takes degrees)
  - Rotates around the image center
  - `transformOrigin` property affects the origin

digia

Binding Property Values

```
…
Rectangle {
   width: 350; height: 2 * 100
   color: "lightblue"

   Image {
         width: parent.width
         source: "justiina.jpg"
      }
```

- Property values can be bound to other values
  - Automatically updated
- Use `parent` for accessing parent or a given `id` to access another element (like sibling)
  - See next example

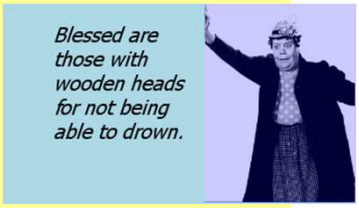
digia

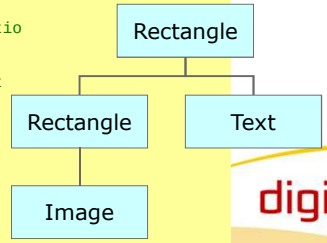
117 © 2010 Digia Plc

More Fine Tuning and Another Rectangle

```
Rectangle {
   width: 350; height: 2 * 100
   color: "lightblue"
   Rectangle {
      width: 150
      anchors.right: parent.right
      height: justiina.height
      color: "blue"
      Image {
         id: justiina
         smooth: true // Smoothens scaling
         width: parent.width
         fillMode: Image.PreserveAspectFit // Aspect ratio
         source: "justiina.jpg"
         opacity: 0.8 // 1.0 = opaque, 0.0 = transparent
      }
   }
 // Text element as previously
   Text {
      …
   }
}
```

Bound properties





digia

© 2010 Digia Plc

The Whole Example So Far

```
/* woodenhead.qml starts here, with a
multiline comment */
import Qt 4.7

Rectangle {
   width: 350; height: 2 * 100
   color: "lightblue"
   Text {
      x: 50; y: 20; width: 150
      wrapMode: Text.WordWrap
      font.pixelSize: 20; font.italic: true
      text: "Blessed are those with wooden heads
            for not being able to drown."
   }
```

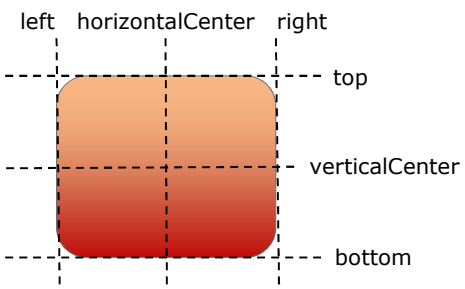
```
   Rectangle {
      width: 150
      anchors.right: parent.right
      height: justiina.height
      color: "blue"
      Image {
         id: justiina
         smooth: true
         width: parent.width
         fillMode: Image.PreserveAspectFit
         source: "justiina.jpg"
         opacity: 0.8
      }
   }
}
```

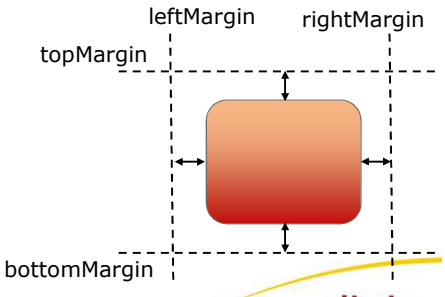
digia

Anchor Layout 1/4

```
Rectangle {
      anchors.right: parent.right
      …
}
```

- Each QML item can be thought of as having 6 invisible anchor and 4 margin lines:



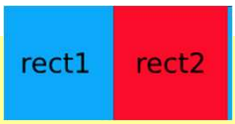


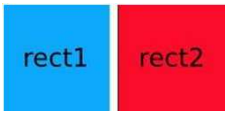
digia

Anchor Layout 2/4

- The anchors are used for specifying relative positions of items
  - As well as offsets and margins

```
Rectangle { id: rect1; ... }
Rectangle { id: rect2; anchors.left: rect1.right; ... }

Rectangle { id: rect1; ... }
Rectangle { id: rect2; anchors.left: rect1.right; anchors.leftMargin: 5; ... }
```

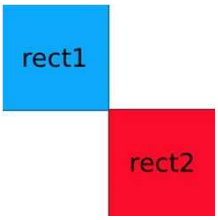




digia

121 © 2010 Digia Plc

Anchor Layout 3/4



- Multiple anchors can be specified
  - Can also be used to control the size of an item!

```
Rectangle { id: rect1; ... }
Rectangle { id: rect2; anchors.left: rect1.right; anchors.top: rect1.bottom;
   ... }

Rectangle { id: rect1; x: 0; ... }
Rectangle { id: rect2; anchors.left: rect1.right; anchors.right: Rect3.left;
   ... }
Rectangle { id: Rect3; x: 150; ... }
```

digia

122 © 2010 Digia Plc

Anchor Layout 4/4

- For performance reasons you can only anchor an item to its siblings and direct parent

```
Item {
   id: Group1
   Rectangle { id: rect1; ... }
}

Item {
   id: Group2
   Rectangle { id: rect2; anchors.left: rect1.right; ... } // Invalid anchor!
}
```

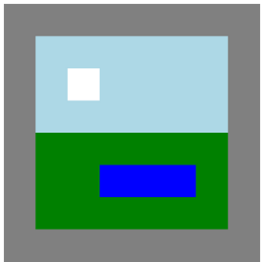
digia

123

Exercise - Items

- The image on the right shows two items and two child items inside a 400 x 400 rectangle.

1. Recreate the scene using Rectangle items.
2. Can items overlap?
   Experiment by moving the light blue or green rectangles.
3. Can child items be displayed outside their parents?
   Experiment by giving one of the child items negative coordinates.



digia

124

Qt Quick

More Layouts

digia

Introduction

- Hard-coding the positions of UI elements is never a good idea
  - Difficult to provide UI scalability
  - Difficult to maintain
- QML provides a number of different kinds of layouts that should be used instead
  - Basic positioners
    - Grid, Row, Column
  - Anchor layout

digia

126

Grid Layout

- Represented by the QML item `Grid`
  - Arranges child items in a grid formation so that they do not overlap each other
  - Provides for transition effects when items are added (shown), moved or removed (hidden)

```
Grid {
    columns: 3
    spacing: 2
    Rectangle { color: "red"; width: 50; height: 50 }
    Rectangle { color: "green"; width: 20; height: 50 }
    Rectangle { color: "blue"; width: 50; height: 20 }
    Rectangle { color: "cyan"; width: 50; height: 50 }
    Rectangle { color: "magenta"; width: 10; height: 10 }
}
```


Row Layout

- Represented by the QML item `Row`
  - Positions child items in a row so that they do not overlap each other
  - Provides for transition effects when items are added (shown), moved or removed (hidden)

```
Row {
    spacing: 2
    Rectangle { color: "red"; width: 50; height: 50 }
    Rectangle { color: "green"; width: 20; height: 50 }
    Rectangle { color: "blue"; width: 50; height: 20 }
}
```

digia

Column Layout

- Represented by the QML item `Column`
  - Positions child items vertically so that they do not overlap each other
  - Provides for transition effects when items are added (shown), moved or removed (hidden)

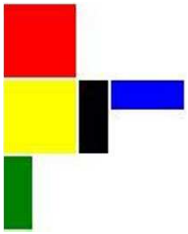
```
Column {
    spacing: 2
    Rectangle { color: "red"; width: 50; height: 50 }
    Rectangle { color: "green"; width: 20; height: 50 }
    Rectangle { color: "blue"; width: 50; height: 20 }
}
```

digia

129 © 2010 Digia Plc

Combining Layouts

- The basic positioners `Grid`, `Row` and `Column` can be combined, if needed
- For example, a `Row` inside a `Column`:



```
Column {
    spacing: 2
    Rectangle { color: "red"; width: 50; height: 50 }
    Row {
        spacing: 2
        Rectangle { color: "yellow"; width: 50; height: 50 }
        Rectangle { color: "black"; width: 20; height: 50 }
        Rectangle { color: "blue"; width:50; height: 20 }
    }
    Rectangle { color: "green"; width: 20; height: 50 }
}
```

130 © 2010 Digia Plc

Qt Quick

User Interaction

digia

Handling User Input

- Users interact with Qt Quick user interfaces:
  - Mouse movement, clicks and dragging
  - Keyboard input

digia

132

Event Handling

- For much (but not all) of its event handling Qt uses a mechanism called *signals and slots*
  - This is basically just an observer pattern
- Similarly in QML, when an event occurs an event-specific signal is emitted
- In order to react to the event, *a signal handler* is thus needed
  - The handler is nothing more than a special property!
  - The property depends on the event source (mouse click, timer, key press, ...)

digia

133

Mouse Areas

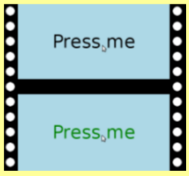
- Mouse areas define parts of the screen where cursor input occurs
- Placed and resized like ordinary items
  - Using anchors if necessary
- Two ways to monitor mouse input:
  - Handle signals
  - Dynamic property bindings

digia

134

Clickable Mouse Area

```
import Qt 4.7

Rectangle {
    width: 400; height: 200; color: "lightblue"
    Text {
        anchors.centerIn: parent
        text: "Press me"; font.pixelSize: 48
        MouseArea {
            anchors.fill: parent
            onPressed: parent.color = "green"
            onReleased: parent.color = "black"
        }
    }
}
```



digia

135 © 2010 Digia Plc

Positioning the Mouse Area

```
Text {
    // ……
    MouseArea {
        anchors.fill: parent
        onPressed: parent.color = "green"
        onReleased: parent.color = "black"
    }
}
```

- Define a Text element
- Define a MouseArea child element
- Use an anchor to apply the area to the entire parent
  - Note that the MouseArea can also be bigger or smaller than its parent!

digia

136 © 2010 Digia Plc

Mouse Area Signals

```
Text {
    // ......
    MouseArea {
        anchors.fill: parent
        onPressed: parent.color = "green"
        onReleased: parent.color = "black"
    }
}
```

- Define responses to signals with signal handlers
  - `onPressed`, `onReleased`, `onClicked`, …
  - By default, only left clicks are handled
  - Set the `acceptedButtons` property to change this
- In this case these simply change the color of the parent Text element

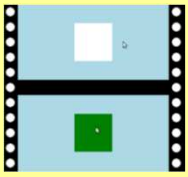
digia

137

Mouse Hover

- Mouse hover can be also detected to achieve similar results

```
import Qt 4.7
Rectangle {
    width: 400; height: 200; color: "lightblue"
    Rectangle {
        x: 150; y: 50; width: 100; height: 100
        color: mouse_area.containsMouse ? "green" : "white"
        MouseArea {
            id: mouse_area
            anchors.fill: parent
            hoverEnabled: true
        }
    }
}
```



138

Different Mouse Events

- Signal handlers are used for handling mouse events
  - `onClicked, onDoubleClicked, onPressAndHold, onReleased, …`
  - A `MouseEvent` called `mouse` is delivered with the signal

```
Rectangle {
    width: 100; height: 100; color: "green"
    MouseArea {
        anchors.fill: parent
        // See Qt::MouseButtons for a list of available buttons
        acceptedButtons: Qt.LeftButton | Qt.RightButton
        onClicked: {
            if (mouse.button == Qt.RightButton)
                parent.color = 'blue';
            else
                parent.color = 'red';
        }
    }
}
```


Dragging Elements

- `MouseArea` provides also a convenient way of making an item draggable with the `drag` property
- Quiz: what does the example below do?

```
Rectangle {
    id: opacitytest; width: 600; height: 200; color: "white"
    Image {
        id: pic; source: "qtlogo-64.png"
        anchors.verticalCenter: parent.verticalCenter
        opacity: (600.0-pic.x) / 600;
        MouseArea {
            anchors.fill: parent
            drag.target: pic
            drag.axis: "XAxis"
            drag.minimumX: 0
            drag.maximumX: opacitytest.width-pic.width
        }
    }
}
```

Mouse Area Hints and Tips

- A mouse area only responds to its `acceptedButtons`
  - By default only the left button
- Signal handlers are not called for other buttons, but
  - Any click involving an allowed button is reported
  - The `pressedButtons` property contains all buttons
  - Even non-allowed buttons, if an allowed button is also pressed
- With `hoverEnabled` set to false
  - `containsMouse` can be true if the mouse area is clicked

digia

141

Signals vs. Property Bindings

- Which to use?
- Signals can be easier to use in some cases
  - When a signal only affects one other item
- Property bindings rely on named elements
  - Many items can react to a change by referring to a property
- Use the most intuitive approach for the use case
  - Favor simple assignments over complex scripts

digia

142

Keyboard Input

- Use cases for keyboard input:

1. Accepting text input
   - TextInput (single-line) and TextEdit (multi-line)
2. Navigation between elements
   - Changing the focused element
   - Directional (arrow keys), tab and backtab
3. Raw keyboard input
   - Reacting to arbitrary key presses, a game for instance

digia

143

Assigning Focus

- UIs with just one TextInput
  - Focus assigned automatically
- More than one TextInput
  - Need to change focus by clicking
- What happens if a TextInput has no text?
  - No way to click on it
  - Unless it has a width or uses anchors
- Set the `focus` property to assign focus
  - See also `activeFocus` in the documentation

digia

144

Using TextInputs

```
import Qt 4.7
Rectangle {
    width: 200; height: 112; color: "lightblue"
    TextInput {
        anchors.left: parent.left; y: 16
        anchors.right: parent.right
        text: "Field 1"; font.pixelSize: 32
        color: focus ? "black" : "gray"
        focus: true
    }
    TextInput {
        anchors.left: parent.left; y: 64
        anchors.right: parent.right
        text: "Field 2"; font.pixelSize: 32
        color: focus ? "black" : "gray"
    }
}
```

Field 1
Field 2...

digia

145 © 2010 Digia Plc

Focus Navigation

```
TextInput {
    id: name_field
    ...
    focus: true
    KeyNavigation.tab: address_field
    }
TextInput {
    id: address_field
    ...
    KeyNavigation.backtab: name_field
}
```

Name
Address

- The `name_field` item assignes `KeyNavigation.tab`
  - Pressing **Tab** moves focus to the `address_field` item
- The `address_field` item assignes `KeyNavigation.backtab`
  - Pressing **Shift+Tab** moves focus to the `name_field` item

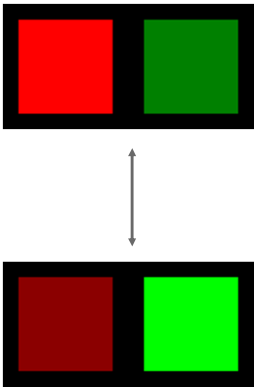
digia

146 © 2010 Digia Plc

Key Navigation

```
Rectangle {
    width: 400; height: 200; color: "black"
    Rectangle {
        id: leftRect
        x: 25; y: 25; width: 150; height: 150
        color: focus ? "red" : "darkred"
        KeyNavigation.right: rightRect
        focus: true
    }
    Rectangle {
        id: rightRect
        x: 225; y: 25; width: 150; height: 150
        color: focus ? "#00ff00" : "green"
        KeyNavigation.left: leftRect
    }
}
```

Left rectangle has the initial focus



- Using cursor keys with non-text items

digia

Raw Keyboard Input 1/2

- All visual elements automatically support key event handling via the `Keys` attached property
- There are multiple signals associated with this property
  - The "generic" ones: `onPressed, onReleased`
  - The "specialized" ones: `onReturnPressed, onSelectPressed, onVolumeUpPressed, …`
  - These contain a `KeyEvent` parameter called `event`
- When handling the *generic* signals
  - You should explicitly state if the event was handled
    `event.accepted = true;`
  - Otherwise the event is propagated to other objects
- *Specialized* handlers accept the event by default

digia

Raw Keyboard Input 2/2

```
Item {   // Handle a key event with a generic handler
   focus: true
   Keys.onPressed: {
        if (event.key == Qt.Key_Left) {    // See Qt::Key for codes
                console.log("move left");
                event.accepted = true;     // Generic handler, must
        }                                  // accept event explicitly
   }
}

Item {   // Handle a key event with a specialized handler
   focus: true
   Keys.onLeftPressed:            // Specialized handler, accepts
        console.log("move left")  // the event by default
}
```

digia

149 © 2010 Digia Plc

Another Example

```
import Qt 4.7
Rectangle {
    width: 400; height: 400; color: "black"
    Image {
            id: rocket
            x: 150; y: 150
            source: "../images/rocket.svg"
            transformOrigin: Item.Center
    }
    Keys.onLeftPressed:
            rocket.rotation = (rocket.rotation - 10) % 360
    Keys.onRightPressed:
            rocket.rotation = (rocket.rotation + 10) % 360
    focus: true
}
```

digia

150 © 2010 Digia Plc

Exercise

- Create a user interface (using layouts!) similar to the one shown above with these features:
  - Items change color when they have focus
  - Clicking on an item gives it focus
  - Focus can also be moved between items using the cursor keys

digia

151

Qt Quick

States, Transitions and Animations

digia

Animations 1/2

- It is possible to animate the properties of objects
  - Types: `real, int, color, rect, point, size`
- Three different forms of animations are available
  - Basic property animation, transitions, property behaviors
- All animations inherit the base element `Animation`
- Animations can be grouped, i.e. run in parallel or in sequence
  - `SequentialAnimation, ParallelAnimation, PauseAnimation`
- A set of pre-defined easing curves is available
  - `OutQuad, InElastic, OutBounce, …`
  - For more information, see `PropertyAnimation` documentation

digia

153

Animations 2/2

- For property animations, use any of the elements inheriting `PropertyAnimation`
  - `NumberAnimation, ColorAnimation, RotationAnimation, ...`
- For property behaviors, use the element `Behavior`
- For transitions between states, use the `Transition` element
  - Will be covered shortly

```
PropertyAnimation {                 // A basic property animation
   target: theObject                // The object (id) to be animated
   property: "size"                 // The property to be animated
   to: "20x20"; duration: 200       // End value & duration
}
```

digia

154

Animation Example 1/2

```
Rectangle {      // Example of a drop-and-bounce effect on an image
   id: rect
   width: 120; height: 200;
   Image {
        id: img
        source: "qt-logo.png"
        x: 60-img.width/2
        y: 0

        SequentialAnimation on y {
            running: true; loops: Animation.Infinite
            NumberAnimation {
                to: 200-img.height; easing.type: "OutBounce"; duration: 2000
            }
            PauseAnimation { duration: 1000 }
            NumberAnimation {
                to: 0; easing.type: "OutQuad"; duration: 1000
           }
        }
   }
}
```

Animation Example 2/2

```
PropertyAnimation {       // Animation as a separate element,
   id: animation          // referred to by its id
   target: image
   property: "scale"
   from: 1; to: .5
}

Image {
   id: image
   source: "image.png"
   MouseArea {   // The animation is started upon mouse press
        anchors.fill: parent
        onClicked: animation.start()  // or: animation.running = true;
   }
}
```

Property Behavior

- Specifies a default animation to run whenever the property's value changes
  - Regardless of what caused the change!
- The example below animates the x position of `redRect` whenever it changes

```
Rectangle {
    id: redRect
    color: "red"
    width: 100; height: 100
    Behavior on x {
        NumberAnimation { duration: 300; easing.type: "InOutQuad" }
    }
}
```

digia

157

State Machines – Purpose

- Can define user interface behavior using states and transitions:
  - Provides a way to formally specify a user interface
  - Useful way to organize application logic
  - Helps to determine if all functionality is covered
  - Can extend transitions with animations and visual effects

digia

158

States

- States manage named items
- Represented by the `State` element
- Each QML item can have a set of states
  - Using the `states` property
  - Current state is set with the `state` property
- Properties are set when a state is entered
- During state change you can also
  - Modify anchors
  - Change the parents of items
  - Run scripts

digia

159

States Example 1/3

```
import Qt 4.7
Rectangle {
    width: 150; height: 250
    Rectangle {
        id: stop_light
        x: 25; y: 15; width: 100; height: 100
    }
    Rectangle {
        id: go_light
        x: 25; y: 135; width: 100; height: 100
    }
    ...
```

- Prepare each item with an id
- Give initial values to properties
  - Can be changed when switching states later

digia

160

States Example 2/3

```
states: [
    State {
        name: "stop"
        PropertyChanges { target: stop_light; color: "red" }
        PropertyChanges { target: go_light; color: "black" }
    },
    State {
        name: "go"
        PropertyChanges { target: stop_light; color: "black" }
        PropertyChanges { target: go_light; color: "green" }
    }
]
...
```

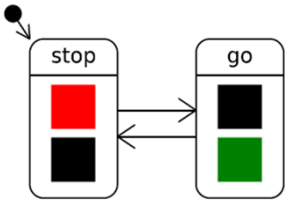
- Define states with names: "stop" and "go", for example
- Set up properties for each state with PropertyChanges
- These define differences from the default values for each item

digia

161 © 2010 Digia Plc

States Example 3/3

```
...
state: "stop" // Define initial state ("" by default)

MouseArea {
        anchors.fill: parent
        onClicked: parent.state == "stop" ?
                    parent.state = "go" : parent.state = "stop"
    }
}
```



- Use e.g. a `MouseArea` to switch between states
  - Toggles the parent's state property between "stop" and "go" states

digia

162 © 2010 Digia Plc

Changing Properties

- States change properties with the `PropertyChanges` element:

```
State {
    name: "stop"
    PropertyChanges { target: stop_light; color: "red" }
    PropertyChanges { target: go_light; color: "black" }
}
```

- Acts on the named target element
- Applies the other property definitions to the target element
  - One `PropertyChanges` element can re-define multiple properties
- Property definitions are evaluated when the state is entered

digia

163

Side-Step: Default Properties

- Each QML element can specify one default property
  - The property name tag can be omitted when the property is assigned a value
  - Consider the `changes` property, which is the default property of the `State` element

```
State {
   changes: [
        PropertyChanges {},
        PropertyChanges {}
   ]
}

// ... can be simplified to:
State {
   PropertyChanges {}
   PropertyChanges {}
}
```

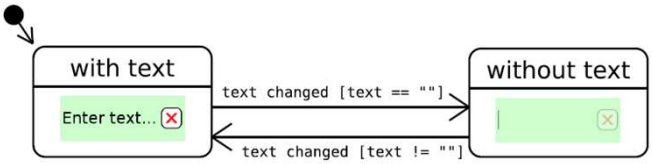
gia

164

State Conditions

- Another way to use states:
- Let the State itself decide when to be active!

- Define the when property
  - Using an expression that evaluates to true or false
- Only one state in a states list should be active
  - Ensure when is true for only one state at a time



digia

165 © 2010 Digia Plc

State Conditions Example

```
Rectangle {
    width: 250; height: 50; color: "#ccffcc"
    TextInput {
        id: text_field
        text: "Enter text..."
        ...
    }
    Image {
        id: clear_button
        source: "../images/clear.svg"
        ...
        MouseArea {
            anchors.fill: parent
            onClicked: text_field.text = ""
        }
    }
    states: [
        State {
            name: "with text"
            when: text_field.text != ""
            PropertyChanges { target: clear_button;
                              opacity: 1.0 }
            },
        State {
            name: "without text"
            when: text_field.text == ""
            PropertyChanges { target: clear_button;
                              opacity: 0.25 }
            PropertyChanges { target: text_field;
                              focus: true }
        }
    ]
}
```

digia

166 © 2010 Digia Plc

Transitions

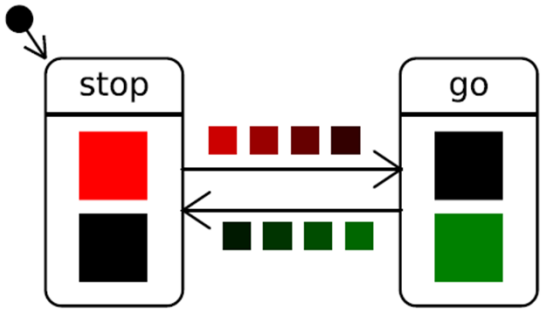
- The `Transition` element provides a way of adding animations to state changes
  - If fact, a transition can only be triggered by a state change
  - As usual, transition animations can be run in sequence and/or in parallel
- By specifying the `to` and `from` properties you can explicitly specify the states the transition is associated with
  - By default these have the value "*", i.e. any state
- A transition can be set to be `reversible` (default `false`)
  - When conditions triggering the transition are reversed, the transition is automatically run backwards
  - For example: state change 1 -> 2 and then 2 -> 1

digia

167

Transitions Example

- Let's add transitions to a previous example...



digia

168

Transitions Example

```
transitions: [
    Transition {
        from: "stop"; to: "go"
        PropertyAnimation {
          target: stop_light
          properties: "color"; duration: 1000
        }
    },
    Transition {
        from: "go"; to: "stop"
        PropertyAnimation {
          target: go_light
          properties: "color"; duration: 1000
        }
    }
]
```

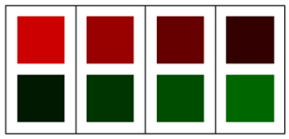
- The transitions property defines a list of transitions
- Transitions between "stop" and "go" states

digia

169

Wildcard Transitions

```
transitions: [
    Transition {
        from: "*"; to: "*"
        PropertyAnimation {
          target: stop_light
          properties: "color"; duration: 1000
        }
        PropertyAnimation {
          target: go_light
          properties: "color"; duration: 1000
        }
} ]
```



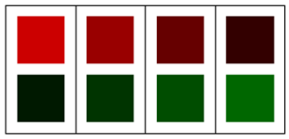
- Use "*" to represent any state (actually, default value is "*")
- Now the same transition is used whenever the state changes
- Both lights fade at the same time

digia

170

Choosing the Correct Animation Type

```
transitions: [
    Transition {
        from: "*"; to: "*"
        ColorAnimation {
            // target not explicitly needed here
            duration: 1000
        }
} ]
```



- Note that exactly the same effect as on the previous slide can be achieved by using ColorAnimation instead of PropertyAnimation
  - In fact, that is what we should have done here to start with
- Code becomes a lot simpler!
  - If the color property changes on any element during state transition, it is automatically animated
  - An explicit target element for the animation could still be set if needed

digia

171 © 2010 Digia Plc

Reversible Transitions

```
transitions: [
    Transition {
        from: "with text"; to: "without text"
        reversible: true
        PropertyAnimation {
            target: clear_button
            properties: "opacity"; duration: 1000
        }
    }
]
```

- Useful when two transitions operate on the same properties
- Transition applies from "with text" to "without text"
  - And back again from "without text" to "with text"
- No need to define two separate transitions

digia

172 © 2010 Digia Plc

Transition Example 2, 1/2

```
transitions: [ Transition {
    // Apply for state changes from any state to MyState and back (optional)
    from: "*"; to: "MyState"; reversible: true
    SequentialAnimation {
          ColorAnimation { duration: 1000 }
          PauseAnimation { duration: 1000 }
          ParallelAnimation {
              // Animate x and y of box1 and box2 simultaneously.
              // How do we know the start and end values of x and y?
              NumberAnimation {
                    duration: 1000; easing.type: "OutBounce"
                    target: box1
                    properties: "x,y"
              }
              NumberAnimation {
                    duration: 1000
                    target: box2
                    properties: "x,y"
              }
          }
    }
} ]  // End list of Transition elements
```

Transition Example 2, 2/2

```
// Example of an explicit transition animation
transitions: [ Transition {
    from: "*"; to: "MyState"; reversible: true
    SequentialAnimation {
          NumberAnimation {
              duration: 1000 easing.type: "OutBounce"
              // Animate myItem's x and y if they have changed in the state
              target: myItem
              properties: "x,y"
          }
          NumberAnimation {
              duration: 1000
              // Animate myItem2's y to 200, regardless of what happens in
              // the state – i.e. run an explicit animation on myItem2
              target: myItem2
              property: "y“
              to: 200                 // An end value given explicitly
          }
    }
} ]
```

174

Using States and Transitions

- Avoid defining complex state machines
  - Not just one state machine to manage the entire UI
  - Usually defined individually for each component
  - Link components together with internal states
- Setting state with script code
  - Easy to do, but might be difficult to manage
  - Cannot use reversible transitions
- Setting state with state conditions
  - More declarative style
  - Can be difficult to specify conditions

digia

175

Summary - States

- State items manage properties of other items:
- Items define states using the `states` property
  - Must define a unique `name` for each state
- Useful to assign `id` properties to items
  - Use `PropertyChanges` to modify items
- The `state` property contains the current state
  - Set this using JavaScript code, or
  - Define a `when` condition for each state

digia

176

Summary - Transitions

- Transition items describe how items change between states
- Items define transitions using the `transitions` property
- Transitions refer to the states they are between
  - Using the `from` and `to` properties
  - Using a wildcard value, `"*"`, to mean any state
- Transitions can be reversible
  - Used when the `from` and `to` properties are reversed

digia

177

Timer

- Timers are handled using the `Timer` item
  - Provides only one signal: `onTriggered`
  - Can be either a single-shot or a repetitive timer

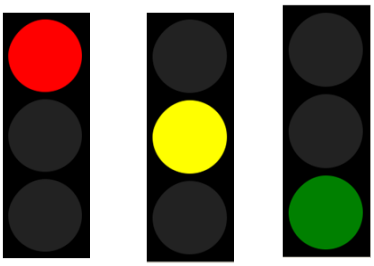
```
Timer {
   interval: 500;
   running: true;
   repeat: true
   onTriggered: time.text = Date().toString()
}

Text {
   id: time
}
```

digia

178

Exercise – Traffic Lights

- Using the following items, construct a simulation of a traffic light:
  - States
  - Transitions
  - Timer



  - Feel free to make your application as smart as possible (different times for different states, yellow behaves differently based on the direction, etc.)

digia

179

Qt Quick

Core QML Features

digia

QML Document 1/2

- Simply a block of QML code containing QML elements
  - A **`.qml`** file, or constructed from text data
  - Always encoded in UTF-8
  - Always begins with at least one `import` statement
    - Nothing is imported by default
    - Does not "include" code, rather just tells the interpreter where to find the definitions of elements at run-time
- Defines a single, top-level *QML component*
- Self-contained
  - No preprocessor or similar is run to modify the code before execution
  - Interpreted at run-time!

digia

181

QML Document 2/2

- Our HelloWorld is a QML document stored e.g. in the **`Helloworld.qml`** file

```
import Qt 4.7  // Import existing QML types to be used in this
               // application, such as Rectangle and Text
Rectangle {
   id: page
   width: 500; height: 200
   color: "lightgray"
   Text {
         id: helloText
         text: "Hello world!"
         font.pointSize: 24; font.bold: true
         y: 30; anchors.horizontalCenter: page.horizontalCenter
   }
}
```

gia

182

QML Component

- As mentioned, a QML document defines a single, top-level QML component
  - A template (cf. a class in C++/Java) out of which objects are created – i.e. the component is *instantiated* at run-time
  - For example, a "Button" component instantiated multiple times with different button text values
- Components are among the basic building blocks in QML
  - Easy to create your own re-usable components
  - Component file name starts with a capital letter (**`MyButton.qml`**)
- A component can contain *inline components*
  - Declared with the keyword `Component`
  - Share the characteristics and import list of the parent
  - Useful e.g. when re-using a component within a single QML file (component logically belongs only to that file)

digia

183

Top-Level QML Component

```
// Definition in MyButton.qml
// (Notice the capital "M" in the file name above)
import Qt 4.7
Rectangle {
    property alias text: textElement.text
    width: 100; height: 30
    source: "images/toolbutton.sci"
    Text {
          id: textElement
          anchors.centerIn: parent
          font.pointSize: 20
          style: Text.Raised; color: "white"
    }
}
// Usage e.g. in main.qml
// (Just an entry point file, thus lower-case "m")
import Qt 4.7
Rectangle {
     MyButton {
        anchors.horizontalCenter: parent.horizontalCenter
        text: "Orange"
     }
…}
```

digia

184

Inline QML Component

```
// In MyComponent.qml
import Qt 4.7

Item {
   Component {   // The inline component
        id: redSquare
        Rectangle {
                color: "red"
                width: 50
                height: 50
        }
   }
   Loader { sourceComponent: redSquare }
   Loader { sourceComponent: redSquare; x: 70 }
}
```

digia

185

QML Modules 1/2

- Multiple QML components can be grouped into *QML modules*
  - The easiest way is to just create a subdirectory containing all the components for the module
  - These modules can then be imported in QML documents: `import "path_to_mymodule"`
  - The path to the module is relative to the file importing it
- You can also use *named imports*
  - To allow identically named modules, or just for code readability

```
import Qt 4.7 as TheQtLibrary   // Into a namespace called TheQtLibrary
TheQtLibrary.Rectangle { ... }

// Multiple imports into the same namespace are also allowed:
import Qt 4.7 as Nokia
import Ovi 1.0 as Nokia
```

digia

186

QML Modules 2/2

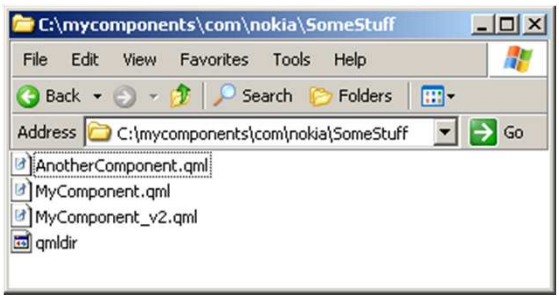
- QML component files can also be installed somewhere *outside* your project
  - In this case an unquoted URI is used:
    `import com.nokia.SomeStuff 1.0`
  - This would access files in folder **`com/nokia/SomeStuff/`** located somewhere in your system
    - E.g. under **`c:/mycomponents/`**
  - The path leading to this URI can be set either
    - In C++ by using `QDeclarativeEngine::addImportPath()`, or
    - By specifying the –L command line option to qmlviewer.exe
- Another possibility:
  `import "http://myserver.com/.../..." 1.0`
- Either way, a file called **`qmldir`** must be present in the directory describing the contents:

```
# <Comment>
<TypeName> <InitialVersion> <File>
```

digia

187 © 2010 Digia Plc

QML Modules – Example




```
// qmldir contents:
# This is a comment
MyComponent 2.0 MyComponent_v2.qml
MyComponent 1.0 MyComponent.qml
AnotherComponent 1.0 AnotherComponent.qml
```

digia

188 © 2010 Digia Plc

Network Transparency 1/2

- Simply means that all references from a QML document to other content are handled as URLs
  - Works for both local and remote content as well as relative and absolute URLs

```
// Test1.qml containing a reference to an absolute URL
Image { source: "http://www.example.com/images/logo.png" }

// Test2.qml with a relative URL
Image { source: "images/logo.png" }
```

digia

189

Network Transparency 2/2

- Relative URLs are resolved automatically into absolute ones
  - Absolute URLs always stay as they are
- Example 1: **`Test2.qml`** itself is loaded from **`http://www.example.com/mystuff/Test2.qml`**
  - URL to image is automatically resolved into **`http://www.example.com/mystuff/images/logo.png`**
- Example 2: **`Test2.qml`** itself is loaded from **`C:/temp/mystuff/Test2.qml`**
  - URL to image is automatically resolved into **`C:/temp/mystuff/images/logo.png`**

digia

190

Progressive Loading

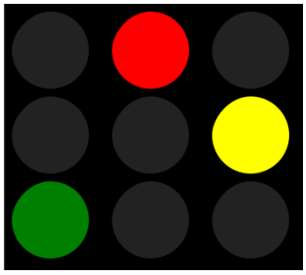
- QML objects that reference a network resource typically provide information on the loading status
  - Needed because networking is inherently asynchronous
- For example, the `Image` element has special properties related to this:
  - `status (Null, Ready, Loading, Error)`
  - `progress (0.0 – 1.0)`
  - `width` and `height` also change as the image is loaded
- Applications can bind to these properties to e.g. show a progress bar when applicable
- For local image files the `status` is `Ready` to start with
  - In future versions this might change
  - If you wish to remain network transparent, do not rely on this!

digia

191

Exercise 4 - Component

- Make your TrafficLight an individual QML component and do a ”main program” where you instantiate multiple traffic lights
- Try to have them running in different states at the same time



digia

192

Qt Quick

Data Models and Views

digia

Data Models and Views

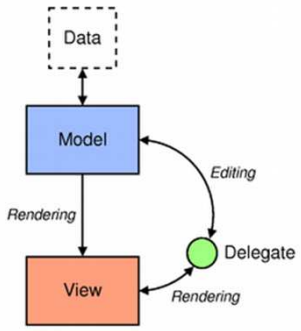
- QML uses a similar Model-View pattern as Qt
- *Model* classes provide data
  - Models can be either in QML (simple cases) or C++ (more complex cases)
  - **QML:** `ListModel, XmlListModel, VisualItemModel`
  - **C++:** `QAbstractItemModel, QStringList, QList<QObject*>`
- *View* classes are used for displaying the data in a model
  - `ListView, GridView, PathView, Repeater` (all QML)
  - All automatically support "scrolling by flicking"
- *Delegates* are used for creating instances of items in the model for the view
- *Highlight* components are used highlighting list items in the view

digia

194

For Reference: Model-View in Qt

- Model provides interface to data for other components
  - QAbstractItemModel
- View obtains model indices
  - Indices are references to the data
- Delegate usually handles custom rendering for the view
  - Delegate communicates directly with model when user edits



digia

195

What Do We Actually Need?

- Model
  - Your data
- Delegate
  - A Component that specifies how each item in the model is displayed in the view
- View
  - A graphical element that automatically shows the contents of a model using the delegate

digia

196

Example – A Simple List 1/3

```
// Define the data in MyModel.qml - static data in this simple case
import Qt 4.7

ListModel {
   id: contactModel
   ListElement {
        name: "Bill Smith"
        number: "555 3264"
   }
   ListElement {
        name: "John Brown"
        number: "555 8426"
   }
   ListElement {
        name: "Sam Wise"
        number: "555 0473“
   }
}
```


Example – A Simple List 2/3

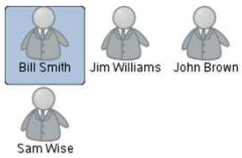
```
// Create a view to use the model e.g. in myList.qml
import Qt 4.7

Rectangle {
   width: 180; height: 200; color: "green"

   // Define a delegate component.  A delegate will be
   // instantiated for each visible item in the list.
   Component {
        id: delegate
        Item {
             id: wrapper
             width: 180; height: 40
             Column {
                 x: 5; y: 5
                 Text { text: '<b>Name:</b> ' + name }
                 Text { text: '<b>Number:</b> ' + number }
             }
        }
   }  // Rectangle continues on the next slide...
```

Example – A Simple List 3/3

```
// ...Rectangle continued...
   // Define a highlight component. Just one of these will be
   // instantiated by each ListView and placed behind the current item.
   Component {
        id: highlight
        Rectangle {
             color: "lightsteelblue"
             radius: 5
        }
   }

   // The actual list
   ListView {
        width: parent.width; height: parent.height
        model: MyModel{}          // Refers to MyModel.qml
        delegate: delegate        // Refers to the delegate component
        highlight: highlight      // Refers to the highlight component
        focus: true
   }
}  // End of Rectangle element started on previous slide
```


GridView

- GridView
  - Shows items in a grid formation
  - Usage is otherwise identical to ListView




```
GridView {
   width: parent.width; height: parent.height
   model: MyModel{}
   delegate: delegate
   highlight: highlight
   cellWidth: 80; cellHeight: 80
   focus: true
}
```

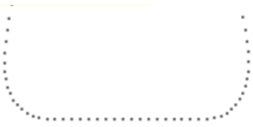
PathView 1/3

- PathView
  - Shows items in a formation specified by a separate Path object
  - Several pre-defined elements exist, of which the Path is constructed
    - PathLine, PathQuad, PathCubic
  - Distribution of items along different parts of the path are controlled with the element PathPercent
  - Appearence of items can be controlled with PathAttribute

digia

PathView 2/3

```
PathView {     // With equal distribution of dots
    anchors.fill: parent; model: MyModel{}; delegate: delegate
    path: Path {
          startX: 20; startY: 0
          PathQuad { x: 50; y: 80; controlX: 0; controlY: 80 }
          PathLine { x: 150; y: 80 }
          PathQuad { x: 180; y: 0; controlX: 200; controlY: 80 }
    }
}

PathView {     // With 50% of the dots in the bottom part
    anchors.fill: parent; model: MyModel{}; delegate: delegate
    path: Path {
          startX: 20; startY: 0
          PathQuad { x: 50; y: 80; controlX: 0; controlY: 80 }
          PathPercent { value: 0.25 }
          PathLine { x: 150; y: 80 }
          PathPercent { value: 0.75 }
          PathQuad { x: 180; y: 0; controlX: 200; controlY: 80 }
          PathPercent { value: 1 }
    }
}
```



digia

PathView 3/3

```
Component {
    id: delegate
    Item {
            id: wrapper; width: 80; height: 80
            scale: PathView.scale
            opacity: PathView.opacity
            Column {
                Image { ... }
                Text { ... }
            }
    }
}

PathView {
    anchors.fill: parent; model: MyModel{}; delegate: delegate
    path: Path {
            startX: 120; startY: 100
            PathAttribute { name: "scale"; value: 1.0 }
            PathAttribute { name: "opacity"; value: 1.0 }
            PathQuad { x: 120; y: 25; controlX: 260; controlY: 75 }
            PathAttribute { name: "scale"; value: 0.3 }
            PathAttribute { name: "opacity"; value: 0.5 }
            PathQuad { x: 120; y: 100; controlX: -20; controlY: 75 }
    }
}
```

Repeater 1/2

- An item for creating a large number of similar items
- Uses a model just like the view elements shown earlier
  - The model can be an object list, a string list, **a number**, or a Qt/C++ model
  - The current model index is exposed as an `index` property

```
Column {
   Repeater {
        model: 10 // The model is just a number here!
        Text { text: "I'm item " + index }
   }
}
```

I'm item 0
I'm item 1
I'm item 2
I'm item 3
I'm item 4
I'm item 5
I'm item 6
I'm item 7
I'm item 8
I'm item 9

Repeater 2/2

- Items created by the `Repeater` are inserted (in order) as children of the `Repeater`'s parent
  - Enables using the `Repeater` inside layouts
  - For example, a `Repeater` inside a `Row` layout:

```
Row {
   Rectangle { width: 10; height: 20; color: "red" }
   Repeater {
        model: 10
        Rectangle { width: 20; height: 20; radius: 10; color: "green" }
   }
   Rectangle { width: 10; height: 20; color: "blue" }
}
```

digia

205 © 2010 Digia Plc

Flickable

- An item that places its children on a surface that can be dragged and "flicked"
  - No need to create a `MouseArea` or manually handle mouse events in any other way
- The flickable surface is easily configurable via its properties
  - `flickDirection`, `flickDeceleration`, `horizontalVelocity`, `verticalVelocity`, `overShoot`, …
- Certain QML elements are flickable by default
  - The `ListView` element, for example

```
Flickable {
   width: 200; height: 200
   contentWidth: image.width; contentHeight: image.height
   Image { id: image; source: "bigimage.png" }
}
```

digia

206 © 2010 Digia Plc

Qt Quick

Advanced QML Features

digia

Extending Types with QML

- Many of the QML core types/elements are actually implemented in C++
- However, extending these types purely with QML is also possible
  - We will talk about extending QML types with C++ later
- With QML, a developer can
  - Add new properties,
  - Add new signals,
  - Add new methods, and
  - Define totally new QML Components
    - We covered this already

digia

208

Adding New Properties 1/4

- Each new property has to have a *type*
  - QML has a set of predefined types to use
  - Each QML type maps to a C++ type

```
// Syntax of adding a new property to an element
[default] property <type> <name>[: defaultValue]

// Example:
Rectangle {
    property color innerColor: "black"
    color: "red"; width: 100; height: 100
    Rectangle {
          anchors.centerIn: parent
          width: parent.width - 10
          height: parent.height - 10
          color: innerColor
    }
}
```

QML Type	C++ Type
int	int
bool	bool
double	double
real	double
string	QString
url	QUrl
color	QColor
date	QDate
var	QVariant
variant	QVariant

digia

Adding New Properties 2/4

- The new property can also be an alias of an existing property (a.k.a. *the aliased property*)
  - A new property (and the storage space for it) is not actually allocated
  - The type is determined by the aliased property

```
// Syntax of creating a property alias
[default] property alias <name>: <alias reference>

// The previous example using a property alias:
Rectangle {
    property alias innerColor: innerRect.color
    color: "red"; width: 100; height: 100
    Rectangle {
          id: innerRect; anchors.centerIn: parent
          width: parent.width – 10; height: parent.height - 10
          color: "black"
    }
}
```

Adding New Properties 3/4

- Property aliases are most useful when defining new *components*
- However, there are a few limitations with aliases
  - Can only be activated once the component specifying them is completed
    - I.e. you cannnot use the alias in the component itself!
    - An alias cannot refer to another alias in the same component

```
// Does NOT work:
property alias innerColor: innerRect.color
innerColor: "black"

// ...and neither does this:
id: root
property alias innerColor: innerRect.color
property alias innerColor2: root.innerColor
```

211

Adding New Properties 4/4

- Despite the limitations, the alias mechanism does provide quite a lot of flexibility as well
  - You can redefine the behaviour of existing property names, and
  - Still within the component use the property "as usual"
- In the example below:
  - The outer rectangle is always red and the user can only modify the color of the inner rectangle, by
  - Using the familiar property called `color` instead of `innerColor`!

```
Rectangle {
   property alias color: innerRect.color
   color: "red"; width: 100; height: 100
   Rectangle { id: innerRect; ...; color: "black" }
}
```

212

Adding New Signals (1/2)

- We saw earlier various signals used in existing QML elements
  - `MouseArea.onClicked, Timer.onTriggered, …`
- Custom signals can be defined as well
  - Can be used within QML
  - Also appear as regular Qt signals in the C++ side!
  - Signals can have arguments (of the QML types shown earlier)

```
Item {
   signal hovered() // A signal without arguments
   signal clicked   // The same as above, empty argument list can be omitted
   signal performAction(string action, var actionArgument)
}
```

digia

213 © 2010 Digia Plc

Adding New Signals (2/2)

```
// MyItem.qml
Rectangle {
   color: "red"; width: 100; height: 100
   signal superClicked
   MouseArea: {
        anchors.fill: parent
        onClicked(): { superClicked() }
    }
```

Defining a signal: *"Elements of this type can emit signal superClicked"*—part of the type interface

Emitting a signal, declaring the situation where this signal is emitted: *"When this rectangle is clicked, it will emit signal superClicked()"*—just internal implementation details…

```
// main.qml
Rectangle {
   …
   MyItem {
        onSuperClicked: {
                color: "yellow"
        }
   }
   …
}
```

Using the type and creating a signal handler ("slot") for it: *"What do we do when THIS object emits superClicked?"*

214 © 2010 Digia Plc

Adding New Methods

- New methods can be added to existing types
  - Normally implemented in JavaScript
  - Usable from QML directly and from C++ as slot functions
  - Can have un-typed parameters
    - Because JavaScript itself is un-typed
    - In C++ the parameter type is `QVariant`

```
// Define a method
Item {
   id: myItem
   function say(text) {
        console.log("You said " + text);
   }
}

// Use the method
myItem.say(”HelloWorld!”);
```

Qt Quick

QML and Scripting

Introduction

- We already saw earlier how to add new methods when extending QML elements
  - These are typically written in JavaScript and ”belong” to that element only
- However, logic for the application is written separately from the UI elements
- In order to utilize these ”external” functions, these need to be imported
  - Stored in a separate **`.js`** JavaScript file
- Applications can also use the services provided by the *QML Global Object*

digia

217

QML Global Object

- QML provides a global JavaScript object called `Qt`
  - Usable anywhere in QML code
  - You have seen this in action already with the `MouseArea` example:
    `acceptedButtons: Qt.LeftButton | Qt.RightButton`
  - All the other enums in the `Qt::` C++ namespace are accessible this way as well!
- Also contains a set of functions for
  - Creating QML types: `Qt.rect(…)`, `Qt.rgba(…)`, `Qt.point(…)`, and
  - Performing other common operations: `Qt.playSound(…)`, `Qt.openUrlExternally(…)`, `Qt.md5(…)`
- Functions for dynamic QML object creation, AJAX and local database access are provided as well

digia

218

Using JavaScript in QML

- Using JavaScript in QML has certain restrictions
  - Luckily not too many
  - We will take a look at these shortly
- There are two ways of using JavaScript
  - Inline JavaScript in QML files
  - Separate JavaScript files

digia

219

Inline JavaScript

```
Item {
    function factorial(a) {
        a = parseInt(a);
        if (a <= 0)
            return 1;
        else
            return a * factorial(a - 1);
    }

    MouseArea {
        anchors.fill: parent
        onClicked: console.log(factorial(10))
    }
}
```

220

Separate JavaScript Files

```
import "factorial.js" as MathFunctions
Item {
    MouseArea {
        anchors.fill: parent
        onClicked: console.log(MathFunctions.factorial(10))
    }
}
```

- Large blocks of JavaScript should be written in separate files
- Both relative and absolute JavaScript URLs can be imported
  - In case of a relative URL, the location is resolved relative to the location of the QML Document that contains the import

digia

221

QML Script Restrictions 1/2

- New members cannot be added to the QML global object from JavaScript
  - Unfortunately, this is fairly easy to do by mistake due to how JavaScript handles undeclared variables

```
// Assuming that “a” has not been declared anywhere before, this code
// is illegal – JavaScript would implicitly try to create “a” as
// a member of the global object, which is not allowed.
a = 1;
for (var ii = 1; ii < 10; ++ii) { a = a * ii; }
console.log("Result: " + a);

// To make it legal, simply declare ”a” properly first:
var a = 1;
for (var ii = 1; ii < 10; ++ii) { a = a * ii; }
console.log("Result: " + a);
```

gia

222

QML Script Restrictions 2/2

- If you import an external script file with some global code, this code is executed in a *reduced scope*
  - I.e. in a scope containing the external file itself and the QML global object
- This happens because it cannot be guaranteed that all relevant QML objects have been properly initialized yet
  - The global code cannot access QML objects and properties it normally would be able to access

```
// Global code outside a function – works, because there are no
// references to any QML objects or properties
var colors = [ "red", "blue", "green", "orange", "purple" ];

// Invalid global code – here the "rootObject" is a QML element id
var initialPosition = { rootObject.x, rootObject.y }
```

223

Startup Scripts

- Sometimes it is necessary to run a piece of code at application startup
  - Or more specifically, when a component is instantiated
- Having this code as a global piece of script in an external script file is not a good idea
  - All relevant pieces of the QML scope chain might not be fully initialized when the code is run
  - See the "Script Restrictions" slides
- The best solution is to use the attached property `onCompleted` of the `Component` element
  - Executed once the component has been fully initialized

```
Rectangle {
    function startupFunction() { // ... startup code }

    Component.onCompleted: startupFunction();
}
```

224

WorkerScript

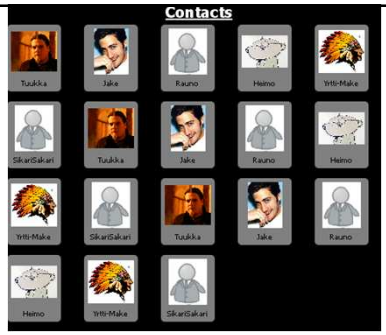
- The `WorkerScript` element enables running JavaScript in a separate thread
  - Background tasks that do not block the UI
- Can only modify ListModel element from a WorkerScript
  - Other QML elements or their properties cannot be altered currently
- Notice that `XMLHttpRequest` is asynchronous by default, no need to use `WorkerScript` with it

digia

225

Mega-Exercise

- Contacts
- List of Contacts (in a GridView, for example)
- Separate Model (as a separate Component)
- Click contact -> Open/Close details
- Start with small test app (text), extend bit-by-bit



Qt Quick

Using QML in Qt/C++ Applications

digia

Introduction

- There are four main classes in the `QtDeclarative` module for using QML from C++
  - `QDeclarativeView`
  - `QDeclarativeEngine`
  - `QDeclarativeComponent`
  - `QDeclarativeContext`
- Many QML elements also have a corresponding C++ class that gets instantiated when the element is used
  - `Item <-> QDeclarativeItem`
- In order to take `QtDeclarative` into use, add the following to your application .pro file:
  - `QT += declarative`

digia

228

QDeclarativeView

- `QDeclarativeView` is a simple view widget for displaying QML content
  - Derives from `QGraphicsView`

```
#include <QtGui/QApplication>
#include <QtCore/QUrl>
#include <QtDeclarative/QDeclarativeView>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QDeclarativeView canvas;
    canvas.setSource(QUrl("main.qml"));
    canvas.show();
    return app.exec();
}
```


QDeclarativeEngine

- Every application wishing to access QML from Qt/C++ needs at least one instance of `QDeclarativeEngine`
  - `QDeclarativeView` contains one of these
- Provides an environment for instantiating QML components from C++
  - Allows configuration of global settings applying to all QML component instances
    - E.g. the `QNetworkAccessManager` instance and path for persistent storage
  - Multiple instances of this class are only needed, if the settings need to differ between QML component instances

QDeclarativeComponent

- A simple class used for loading QML documents
  - Each `QDeclarativeComponent` instance represents a single QML document
  - Can be used without the `QDeclarativeView` class
- The content can be given as a document URL or raw text
  - The URL can point to local file system or any network URL supported by `QNetworkAccessManager`
- Contains status information about the document
  - Null, Ready, Loading, Error

digia

231

Example – Instantiating a QML Component

```
// Create the engine (root context created automatically as well).
// Notice that if you are using QDeclarativeView, it already has an
// engine – simply call QDeclarativeView::engine() to access it.
QDeclarativeEngine engine;

// Create a QML component associated with the engine
// (Alternatively you could create an empty component and then set
// its contents with setData().)
QDeclarativeComponent component(&engine, QUrl("main.qml"));

// Instantiate the component (as no context is given to create(),
// the root context is used by default)
QDeclarativeItem *item =
    qobject_cast<QDeclarativeItem *>(component.create());

// Add item onto a QGraphicsScene
// ...
```

digia

232

QDeclarativeContext 1/5

- Each QML component is instantiated in a `QDeclarativeContext`
  - The engine automatically creates a default root context
- Additional sub-contexts can be created as needed
  - Sub-contexts are arranged hierarchically
  - The root context is the parent of all sub-contexts
  - The hierarchy is managed by the `QDeclarativeEngine`
- Data meant to be available to all QML component instances should be put in the engine's root context
- Data meant for a subset of component instances should be put in a sub-context

QDeclarativeContext 2/5

- Using a context you can expose C++ data and objects to QML

```
// main.qml
import Qt 4.7
Rectangle {
    color: myBackgroundColor
    Text {
          anchors.centerIn: parent
          text: "Hello Light Steel Blue World!"
    }
}

// main.cpp
QDeclarativeEngine engine;  // Again, can be obtained from QDeclarativeView as well
// engine.rootContext() returns a QDeclarativeContext*
(engine.rootContext())->setContextProperty("myBackgroundColor",
                                           QColor(Qt::lightsteelblue));
QDeclarativeComponent component(&engine, "main.qml");
QObject *window = component.create();  // Create using the root context
```

QDeclarativeContext 3/5

- This mechanism would also be used when providing a C++ model for e.g. a QML `ListView`

```
QDeclarativeEngine engine;

// Expose modelData (e.g. of type QAbstractItemModel) by the name
// myModel to QML
(engine.rootContext())->setContextProperty("myModel", modelData);

// Create a QML component
QDeclarativeComponent component(&engine,
   "import Qt 4.7 \n ListView { model: myModel }");

// Instantiate component
component.create();
```

digia

235

QDeclarativeContext 4/5

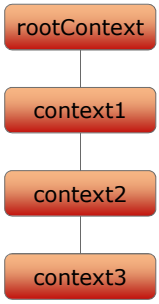
- As mentioned, contexts are hierarchal
  - A component instantiated in a context gets access to that context's data, as well as the data in the context's ancestors
- In case of multiply defined values, data defined in a sub-context overrides data defined in a parent context

digia

236

QDeclarativeContext 5/5

```
QDeclarativeEngine engine;
QDeclarativeContext context1(engine.rootContext());
QDeclarativeContext context2(&context1);
QDeclarativeContext context3(&context2);

context1.setContextProperty("a", 12);
context2.setContextProperty("b", 13);
context3.setContextProperty("a", 14);
context3.setContextProperty("c", 14);

// Instantiate QDeclarativeComponents using the sub-contexts
component1.create(&context1);  // a = 12
component2.create(&context2);  // a = 12, b = 13
component3.create(&context3);  // a = 14, b = 13, c = 14
```



digia

Structured Data

- In case you have larger data sets to expose, consider exposing *a context* `QObject` instead
- All `QProperties` defined in the context object become available in the QML context by name
  - The data exposed this way can be made *writable* from QML as well!
  - Slightly faster than manually exposing multiple property values using `setContextProperty()`

digia

Structured Data – A Simple Example

```
// MyDataSet.h
class MyDataSet : ... {
    ...
    // The NOTIFY signal informs about changes in the property's value
    Q_PROPERTY(QAbstractItemModel *myModel READ model NOTIFY modelChanged)
    Q_PROPERTY(QString text READ text NOTIFY textChanged)
    ...
};

// SomeOtherPieceOfCode.cpp exposes the QObject using e.g. a sub-context
QDeclarativeEngine engine;
QDeclarativeContext context(engine.rootContext());
context.setContextObject(new MyDataSet(...));
QDeclarativeComponent component(&engine, "ListView { model=myModel }");
component.create(&context);
```


Calling C++ Functions From QML

- Any public slot function of a `QObject` can be called from QML
- In case you do not want your function to be a slot, you can declare it as `Q_INVOKABLE`
  - `Q_INVOKABLE void myMethod();`
- These functions can have arguments and return types
- Currently the following types are supported:
  - `bool`
  - `unsigned int, int, float, double, real`
  - `QString, QUrl, QColor`
  - `QDate, QTime, QDateTime`
  - `QPoint, QPointF, QSize, QSizeF, QRect, QRectF`
  - `QVariant`

Example 1/2

```
// In C++:
class LEDBlinker : public QObject {
    Q_OBJECT
    // ...
public slots:
    bool isRunning();
    void start();
    void stop();
};

int main(int argc, char **argv) {
    // ...
    QDeclarativeContext *context =
          engine->rootContext();
    context->setContextProperty("ledBlinker",
          new LEDBlinker);
    // ...
}
```

```
// In QML:
import Qt 4.7

Rectangle {
  MouseArea {
    anchors.fill: parent
      onClicked: {
        if (ledBlinker.isRunning())
          ledBlinker.stop()
        else
          ledBlinker.start();
      }
  }
}
```

Example 2/2

- Notice that the same result could be achieved by declaring a "running" property
  - Leads to much nicer code
  - Implementation of `isRunning()` and `setRunning()` omitted here for simplicity

```
// In C++:
class LEDBlinker : public QObject {
    Q_OBJECT
    Q_PROPERTY(bool running READ isRunning WRITE setRunning)
    // ...
};

// In QML:
Rectangle {
    MouseArea {
          anchors.fill: parent
          onClicked: ledBlinker.running = !ledBlinker.running
    }
}
```

digia

Calling QML Functions From C++

- Obviously the reverse works as well – you can call functions declared in QML from your C++ code
- Any function you declare in QML appears as a slot function in C++
  - Simply connect a C++ signal to a QML function
  - …or use `QMetaObject::invokeMethod(…)` to call the QML function
- As mentioned before, any signals declared in QML can be connected to slots in C++

digia

243

Network Components 1/2

- As discussed earlier, a QML component can be loaded from a resource across a network
- In such a case, the component instantiation might take some time
  - Usually because of network latency
- To instantiate a network-based QML component in C++:
  - Observe the component's *loading status*, and
  - Only after the status is *Ready*,
  - Call `create()` on the component

digia

244

Network Components 2/2

```
MyObject::MyObject() {
    component = new QDeclarativeComponent(engine,
          QUrl("http://www.example.com/main.qml"));
    // Check for status before creating the object – notice that this kind of
    // code could (should?) be used regardless of where the component is located!
    if (component->isLoading())
        connect(component, SIGNAL(statusChanged(QDeclarativeComponent::Status)),
                this, SLOT(continueLoading()));
    else
        continueLoading(); // Not a network-based resource, load straight away
}

// A slot that omits the Status parameter of the signal and uses the isXxxx()
// functions instead to check the status – both approaches work the same way
void MyObject::continueLoading() {
    if (component->isError()) {
          qWarning() << component->errors();
    } else if (component->isReady()) {
          QObject *myObject = component->create();
    } // The other status checks here ...
```

QML Components in Resource File 1/2

- Probably the most convenient way of including QML components in your Qt project is to put them into a resource file
  - Also JavaScript files can be included, of course
- Easier access to the files
  - No need to know the exact path to the file
  - Simply pass a URL pointing to the resource file
- *Resource files get compiled into the application binary*
  - These files are thus automatically distributed along with the binary, just like any other resource (e.g. images)

digia

246

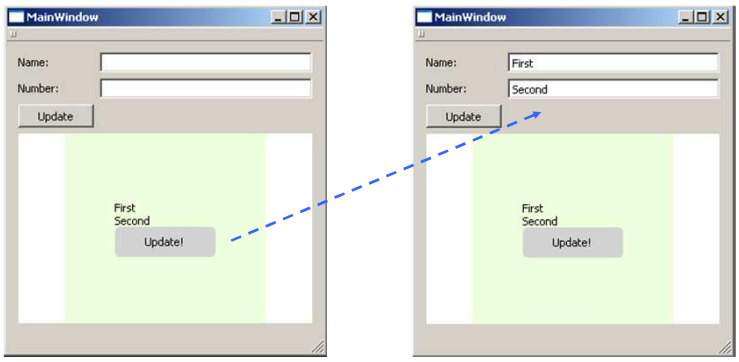
QML Components in Resource File 2/2

```
// MyApp.qrc
<!DOCTYPE RCC>
<RCC version="1.0">
   <qresource> <file>qml/main.qml</file> </qresource>
</RCC>

// MyObject.cpp
MyObject::MyObject() {
   component = new QDeclarativeComponent(engine,
        QUrl("qrc:/qml/main.qml"));
   if (!component->isError()) {
        QObject *myObject = component->create();
   }
}

// main.qml
import Qt 4.7
Image { source: "images/background.png" }
```


Qt Quick

Concrete Example on QML/C++ Connections

From QML to C++

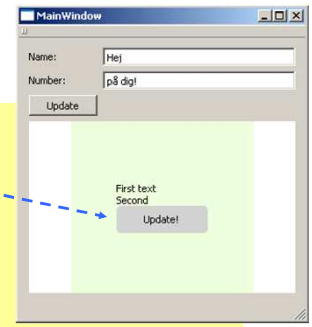


Qt Side, Exposing QObject to QML

```
// mainwindow.h
class MainWindow : public QMainWindow
{
   …
public slots:
   void updateValues(const QString& s1, const QString& s2);
   …
}

// mainwindow.cpp
MainWindow::MainWindow(QWidget *parent) : …
{
   // m_view is a QDeclarativeView*
   m_view->rootContext()->setContextProperty("mainWindow",this);
   …
}

void MainWindow::updateValues(const QString& s1, const QString& s2) {
   ui->lineEdit->setText(s1);
   ui->lineEdit_2->setText(s2);
}
```

QML Side

```
Rectangle {
        id: button
        color: "lightgray"
        width: 100
        height: 30
        radius: 5
        Text {
            anchors.centerIn: parent
            text: "Update!"
        }
        MouseArea {
            anchors.fill: parent
            onClicked: mainWindow.updateValues(one.text,
                       two.text)
        }
    }
```

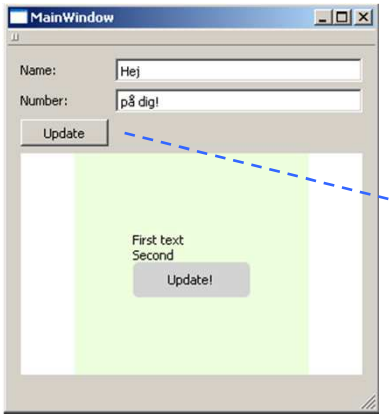


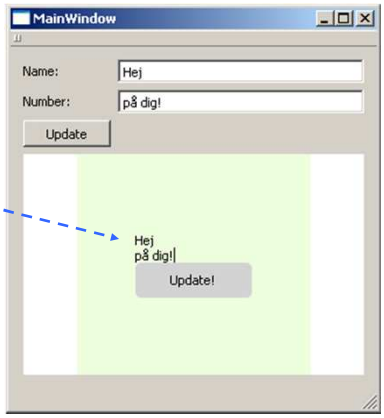
Calls slot function updateValues with these arguments

digia

251 © 2010 Digia Plc

From C++ to QML?

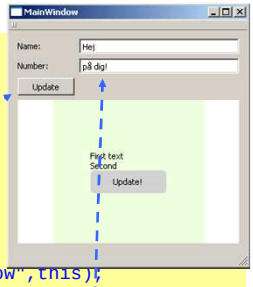




digia

252 © 2010 Digia Plc

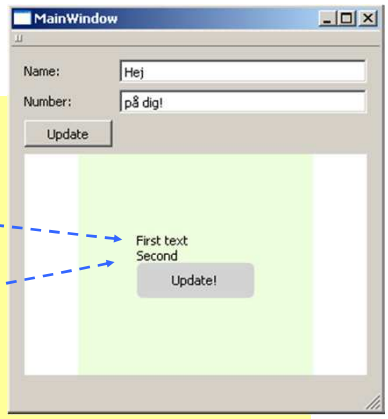
Qt Side, Emitting Signal when Clicked




```
class MainWindow : public QMainWindow
{
signals:
   void updateRequested(QString v1, QString v2);
…
}


MainWindow::MainWindow(QWidget *parent) : …
{
    m_view->rootContext()->setContextProperty("mainWindow",this);
    connect( ui->pushButton, SIGNAL(clicked()),
             this, SLOT(updateClicked()));
…


}


void MainWindow::updateClicked() {
    emit updateRequested(ui->lineEdit->text(),ui->lineEdit_2->text());
}
```

253 © 2010 Digia Plc

QML Side




```
Column {
        id: columnLayout
        anchors.centerIn: parent
        TextInput {
            id: one
            text: "First text"

        }
        TextInput {
            id: two
            text: "Second"
        }
         …


 Connections {
        target: mainWindow
        onUpdateRequested: {
            one.text = v1
            two.text = v2
        }
    }
```

"When mainWindow emits updateRequested()"

v1 and v2 are names of signal arguments

2010 Digia Plc

Exposing C++ to QML

- Public slots
- Q_INVOKABLE
- `contextObject->setContextProperty("objName",this);`

- qmlRegisterType
  - `qmlRegisterType<CoolClass>("com.digia.qmlcomponents", 1, 0, "Cool");`

  - `import com.digia.qmlcomponents 1.0`
  - `Cool { ... }`

digia

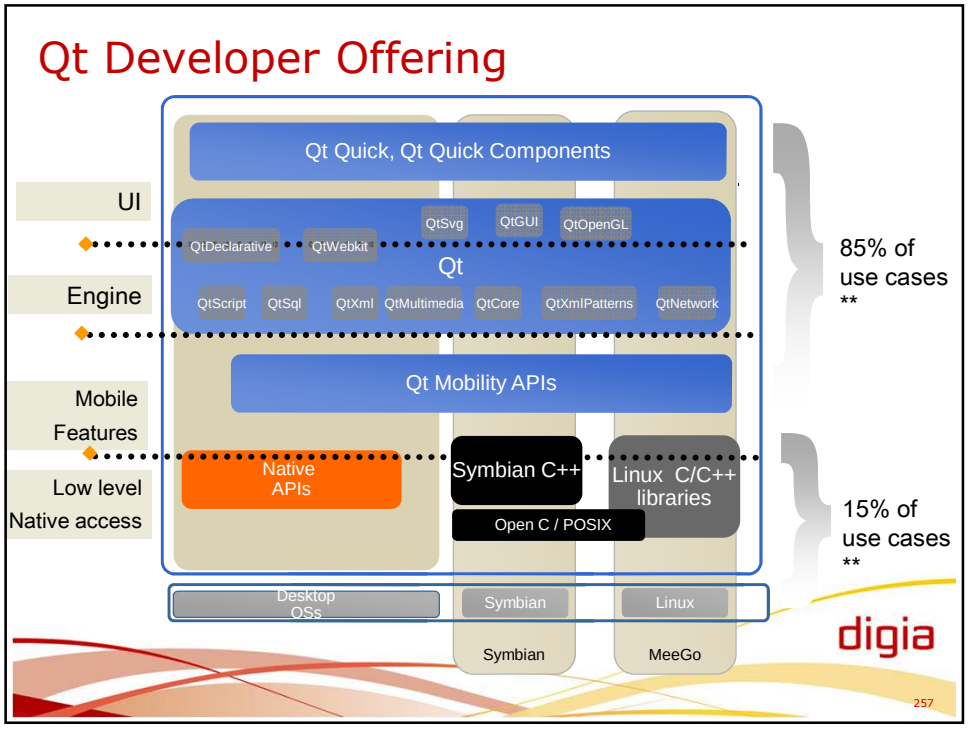
255

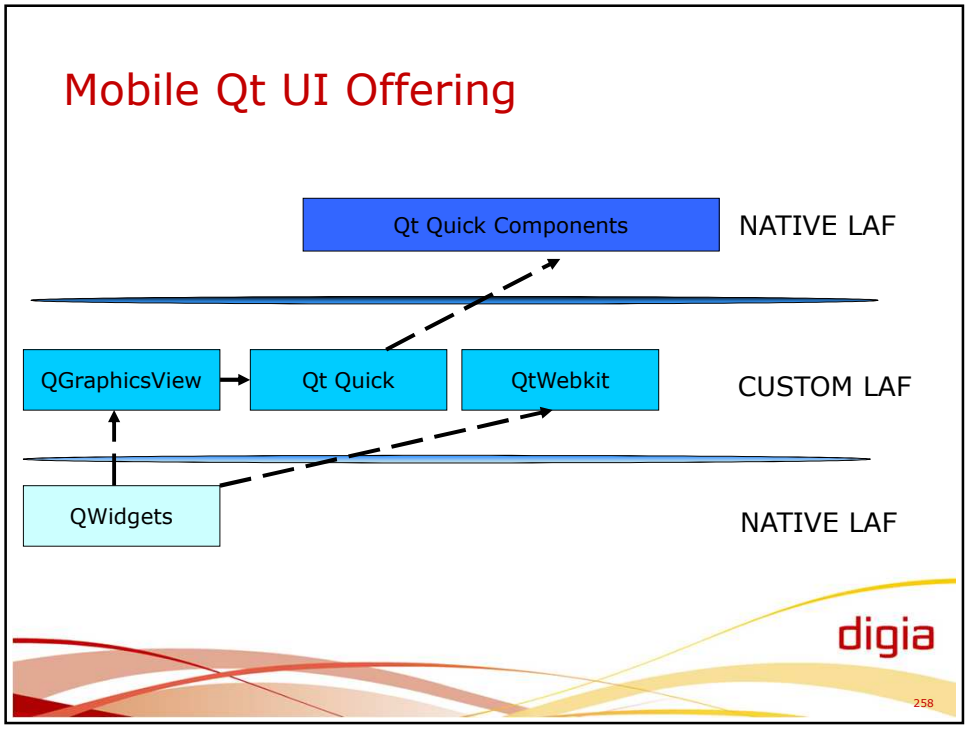
Calling QML functions from C++

```
Item {
      function say(text) {
            console.log("You said " + text);
      }
}

QDeclarativeEngine engine;
QDeclarativeContext *context = new
      QDeclarativeContext(engine.rootContext());
QDeclarativeComponent component(&engine,
      QUrl::fromLocalFile("main.qml"));
QObject *object = component.create(context);
QVariant str("Hello");
QMetaObject::invokeMethod(object, "say",
      Q_ARG(QVariant, str));
```

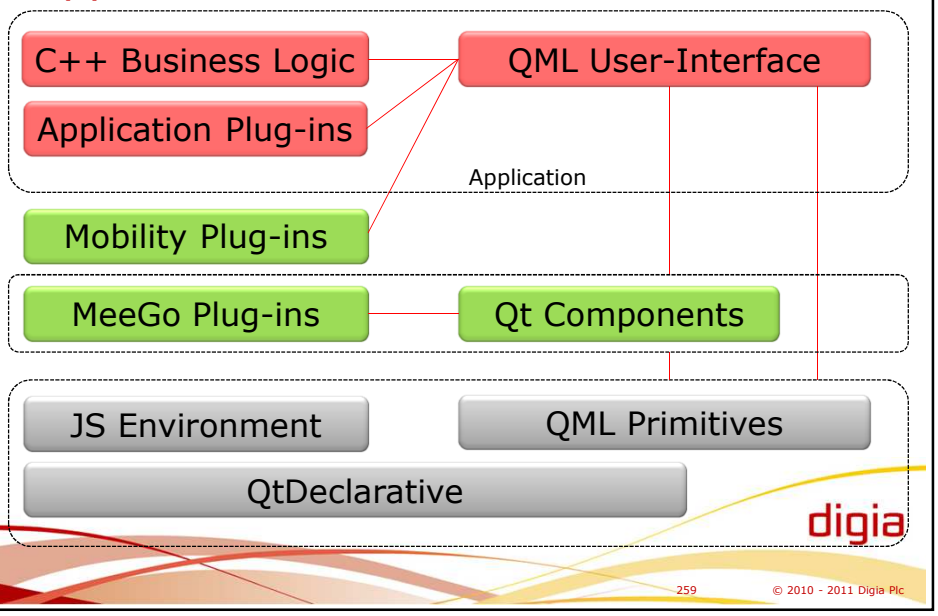
digia

256

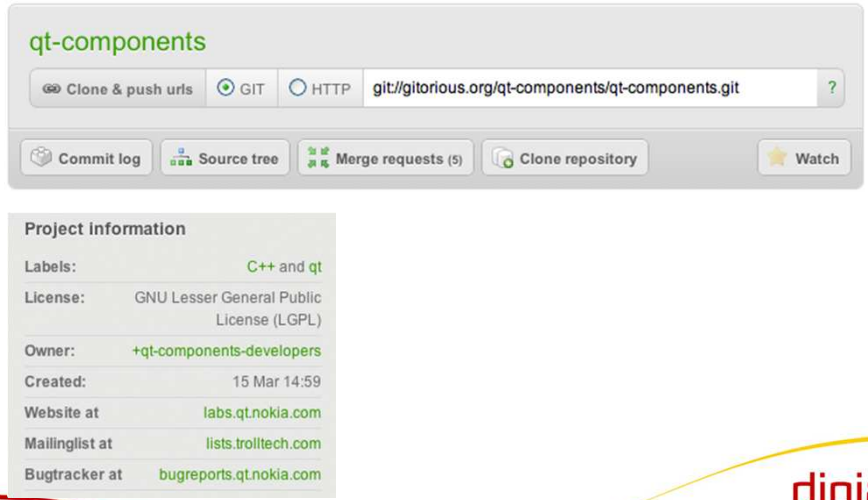

Qt Developer Offering
Qt Quick, Qt Quick Components
UI
Engine
Mobile Features
Low level Native access
QtSvg
QtGUI
QtOpenGL
QtDeclarative
QtWebkit
Qt
QtScript
QtSql
QtXml
QtMultimedia
QtCore
QtXmlPatterns
QtNetwork
Qt Mobility APIs
Native APIs
Symbian C++
Linux C/C++ libraries
Open C / POSIX
85% of use cases **
15% of use cases **
Desktop OSs
Symbian
Linux
Symbian
MeeGo
digia
257


Mobile Qt UI Offering
Qt Quick Components
NATIVE LAF
QGraphicsView
Qt Quick
QtWebkit
CUSTOM LAF
QWidgets
NATIVE LAF
digia
258

Application Architecture



Repo



Dependencies

- Qt 4.7
- MeeGo Touch (optional)
  - libmeegotouch
  - Meegotouch-theme

Installation

1. git clone git://gitorious.org/qt-components/qt-components.git
2. ./configure –config meego
3. make

Content

- Window
- Page
- Label
- PushButton
- CheckBox
- SwitchButton
- IconButton
- Slider
- ProgressBar
- ListItem
- ComboBox
- Spinner
- ScrollArea
- ButtonGroup
- LineEdit

digia

263

Virtual Keyboard

- Custom Virtual Keyboard
- Modifiable appearance
- Positioning
- Effects and content
- Configuration API

digia

264

Window

```
import Qt 4.7
import com.meego 1.0

Window {
      id: theWindow
      Component {
            id: theStuff
            Page { … }}
      Component.onCompleted: {
            theWindow.nextPage(theStuff) } }
```

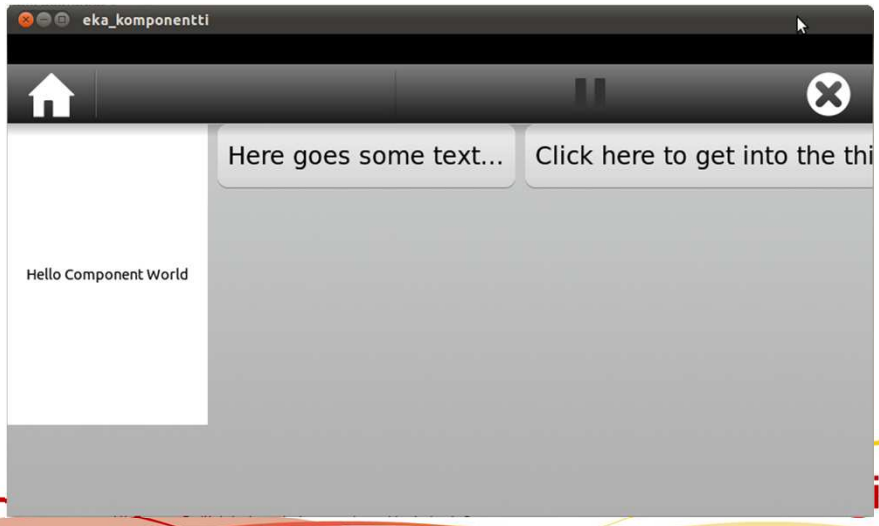

Features of Window

- A way to switch between pages
- View, or Page, transition animation
- Navigation history
  - Possiblity to navigate back
  - Automatically enabled
- Orientation support
  - Signals: orientationChangeAboutToStart, Started, Finished
  - State: portrait, landscape
  - Orientation change animation

Page – The Content

```
Page {
  Row {
    spacing: 10
      Rectangle {…}
      Button {
        text: "Here goes some text..."
        onClicked:theWindow.nextPage(otherStuff)
      }
      Button {…}
  }
}
```


Result: Page in the Window



MeeGo Application and Platform Development

Middleware Subsystems

digia

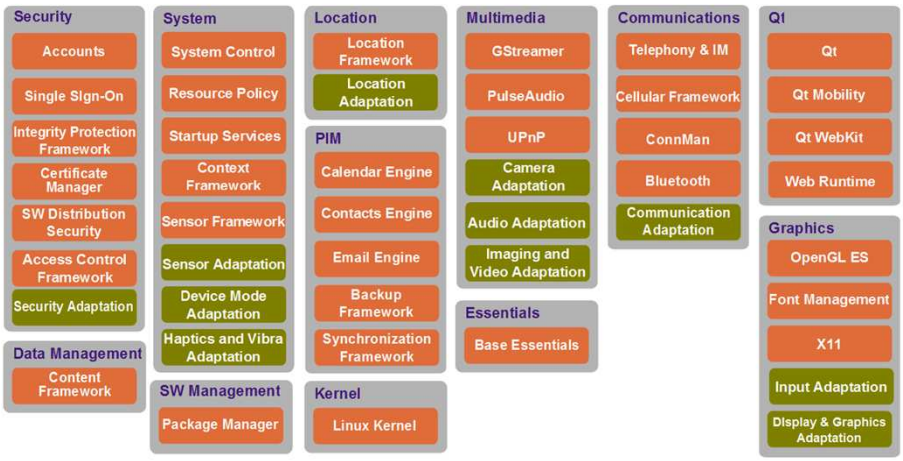
Agenda

- Middleware Subsystems
- Telephony, Messaging, SocialWeb, GStreamer etc

digia

270

MeeGo Subsystem Components



271 © 2009 Digia Plc

Subsystem Services

- A subsystem may be a library, a collection of libraries (plug-ins or application extensions) or a server daemon
- A MeeGo subsystem service is typically accessed using
  - A GObject like interface
  - D-Bus interface or
  - Higher-level interface –based on this (MeeGo touch service framework)
- So at the low level, a service could use itself
  - Libglib
  - Libdbus
  - Libdbus-glib
  - Libdbus-qt
- However, a higher-level API is often used

digia

272 © 2009 Digia Plc

Communication Services – 1(2)

- Voice (cellular, VoIP) and data connectivity (WiFi, WiMax, BT) management
  - Connection management – a daemon for Internet connection management
    - D-Bus like interface
  - Telephony APIs provided by oFono components
    - Daemon – Plug-in and driver loading, modem abstraction, SMS codec
    - Atoms – Access to calls, SMSs, SIM management
    - Drivers – Interfaces to a modem (3GPP TS 27.0.0.7)
    - Plug-ins – Provide drivers

digia

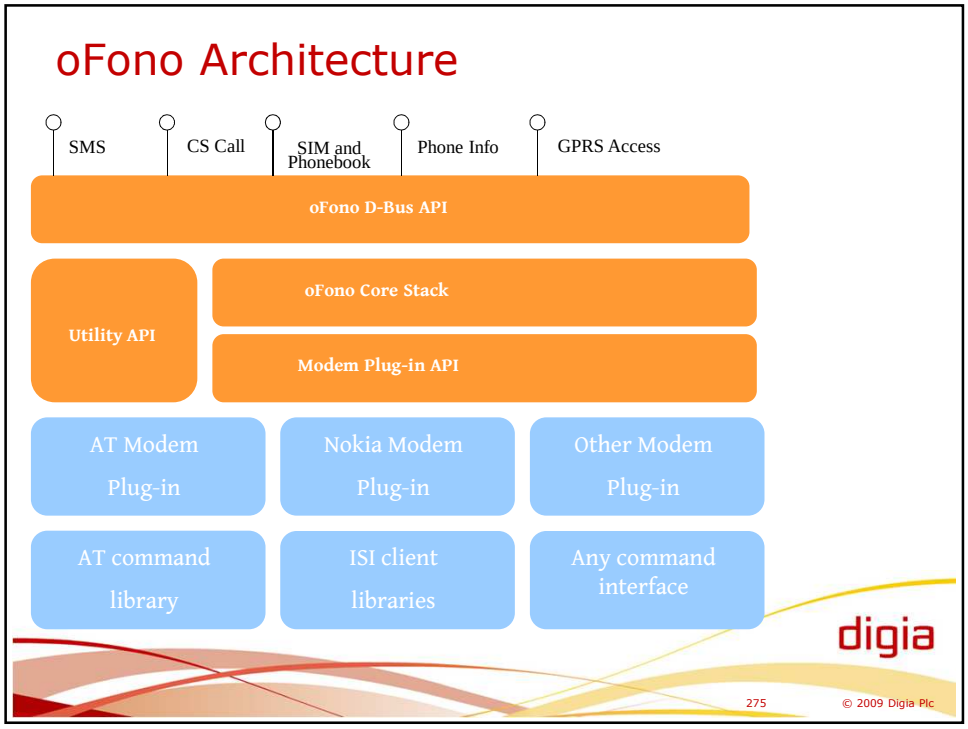
273

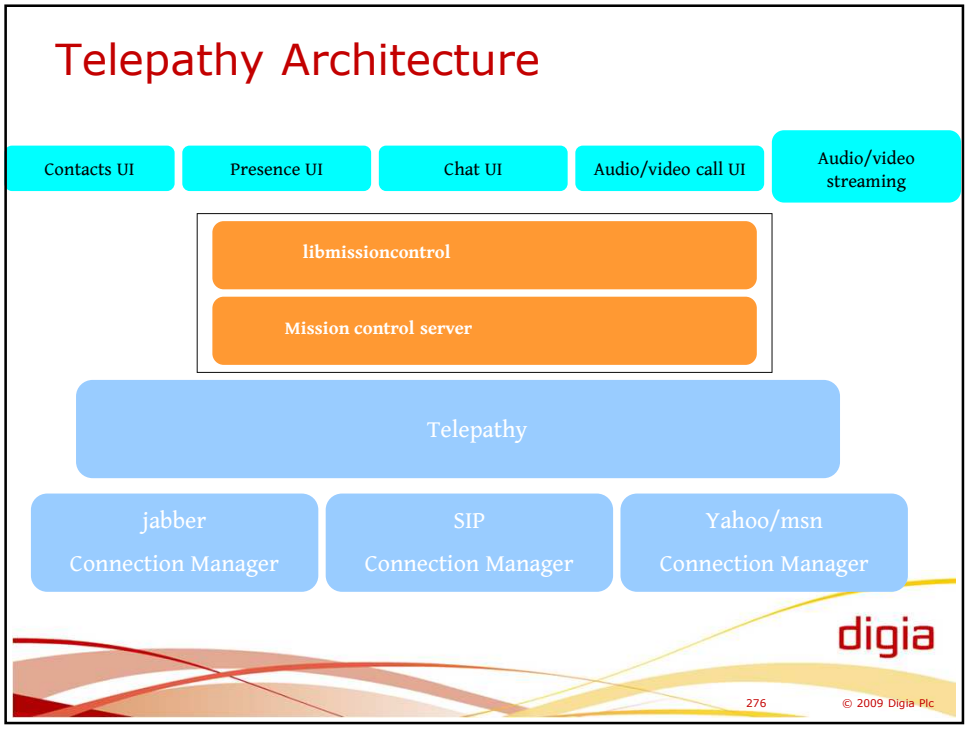
Communication Services – 2(2)

- Telepathy – VoIP, IM, and presence functionality
  - Provides a D-Bus interface for clients to make use of real-time communication over any protocol
  - Connection managers implement telepathy connection manager interface
  - Mission control provides account and presence management
    - New protocols may be added by deploying a new connection manager
- Bluetooth
  - Based on official Linux BT stack BlueZ
  - User side profiles
  - Kernel side HCI interface to RFComm (serial like) and L2CAP (Logical Link Control and Adaptation)

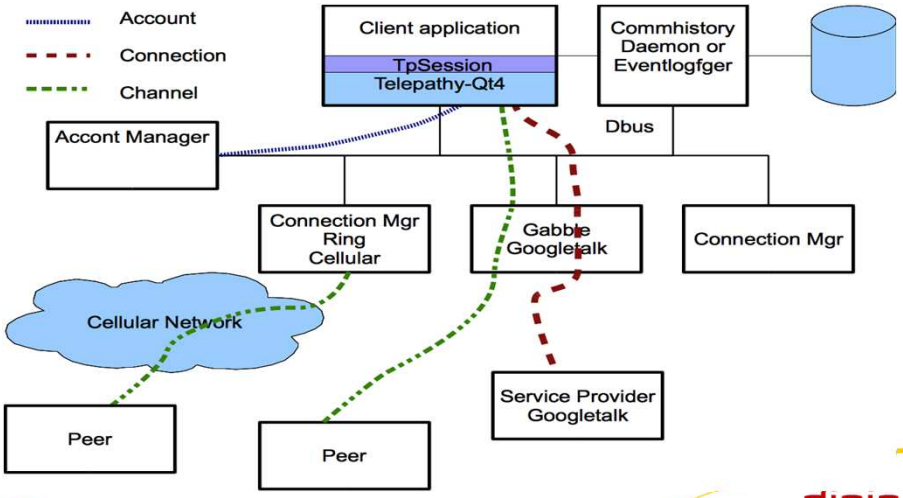
digia

274


oFono Architecture
SMS
CS Call
SIM and Phonebook
Phone Info
GPRS Access
oFono D-Bus API
Utility API
oFono Core Stack
Modem Plug-in API
AT Modem Plug-in
Nokia Modem Plug-in
Other Modem Plug-in
AT command library
ISI client libraries
Any command interface
digia
275
© 2009 Digia Plc


Telepathy Architecture
Contacts UI
Presence UI
Chat UI
Audio/video call UI
Audio/video streaming
libmissioncontrol
Mission control server
Telepathy
jabber Connection Manager
SIP Connection Manager
Yahoo/msn Connection Manager
digia
276
© 2009 Digia Plc

Messaging Architecture



Messaging APIs

- Telepathy-Qt4
  - Low level very powerful API
  - You can do everything if you know how to use it
  - Very complicated, sending single SMS, few hundred lines of code
  - Only sending/receiving, no message store access
- Libtpsession
  - Easy to use wrapper over telepathy-qt4
  - Sending SMS/IM couple of lines of code
  - Extra package, not part of standard SDK
  - Only sending/receiving, no message store access
- Qt Mobility Messaging
  - Relatively easy to use, sending SMS/IM dozen lines of code
  - Not yet released
  - Sending and message store access, no direct receive
  - Message store access performance issues

How to use libtpsession

- Create TpSession instance with preferred connection manager and synchronous mode

```
TpSession* tps = new TpSession("ring", true);
tps->sendMessageToAddress("ring", "+3584011111", "Where
  are you?");
```

- To e able to receive messages, connect to the signal messageReceived

```
connect(tps, SIGNAL(messageReceived(const
  Tp::ReceivedMessage &, TpSessionAccount *)),
  SLOT(onMessageReceived(const Tp::ReceivedMessage &,
  TpSessionAccount *)));
// in the handler
qDebug() << "Msg received" << msg.text() << "from " <<
  msg.sender()->id();
```

digia

Incoming Call Notification

```
TpSessionAccount* tpsa = tps->getAccount("ring");

connect(tpsa,SIGNAL(newChannel(TpSessionAccount *, Qstring,
   QString, const Tp::ChannelDetails)),
            SLOT(onNewChannels(TpSessionAccount *, Qstring,
   QString, const Tp::ChannelDetails)));


Testprog::onNewChannels(TpSessionAccount *tpsa, Qstring
   channelType, QString peerId, const Tp::ChannelDetails)
{
   qDebug() << "Incoming call type " << channelType << "from " <<
   peerId;
}
```

digia

Internet Services

- Web content rendering, web run-time support, data exchange with web services, and location determination
- Layout engine for web rendering Qt WebKit
- Web run-time support in not in MeeGo 1.0 release
- Web services
  - Provided by libSocialWeb
  - Daemon which fetches data, such as blog posts, photos, upcoming events, recently played tracks
- Location services (GeoClue)
  - Location determination using GPS, cellular, and WLAN networks
  - Location and street address transformations

digia

281

Location Example Using GLib Wrapper

```
#include <geoclue/geoclue-position.h>
int main() {
    GeocluePosition *gPos;
    GeocluePositionFields fields; double lat, lon;
    GError *error = NULL;
    g_type_init ();
    gPos = geoclue_position_new ("org.freedesktop.Geoclue.Providers.Hostip",
    "/org/freedesktop/Geoclue/Providers/Hostip");
    fields = geoclue_position_get_position (gPos, NULL, &lat, &lon, NULL, NULL,
    &error);
    if (error) ; // Handle error
    if (fields & GEOCLUE_POSITION_FIELDS_LATITUDE && fields &
    GEOCLUE_POSITION_FIELDS_LONGITUDE) {
          g_print ("Hostip.info provides current location at %.3f, %.3f.\n",
    lat, lon); }
    else ; // No location data available
    g_object_unref (pos);
    return 0;
}
```

digia

282

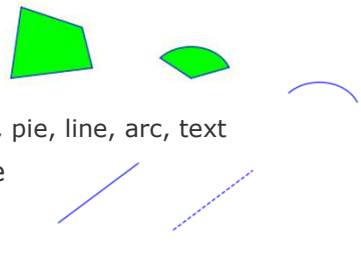
Visual Services

- 2D graphics - Cairo, QPainter
  - Cairo provides operations similar to the drawing operators of PostScript and PDF
  - Cairo and QPainter operations include stroking (pen) and filling (brush) cubic Bézier splines, transforming and compositing translucent images, and antialiased text rendering
  - All drawing operations can be transformed by any affine transformation (scale, rotation, shear, etc.)
- 3D graphics
  - OpenGL ES
- Internationalization rendering
  - Pango, QText
- X window system
  - Window server services

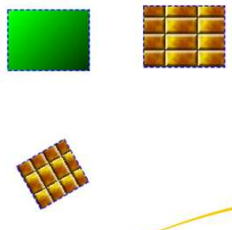
digia

283

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



digia

284

Media Services

- Provide audio/video playback, streaming and imaging functionality to the system
- GStreamer provides basic functionality for playback, streaming, and imaging functionality
  - GStreamer plug-in provides the camera functionality
  - Another kind of GStreamer plug-ins are codecs
- Audio input, pre and postprocessing, and audio output is managed by PulseAudio
- GUPnP provides universal plug and play functionality
  - For example, managing media playback in a remote media server device

digia

285

Data Management

- Provides services for extracting and managing file meta-data, retrieving data about the device context (position, cable status), package management
- Tracker content framework
  - Indexing, meta-data extraction, and search capabilities for media files
- ContextKit context framework
  - Provides access to context properties of the device
- Package manager by PackageKit
  - Package installation and update

digia

286

MeeGo Application and Platform Development

Qt Mobility APIs

digia

Agenda

- Overview
- API Contents
- Usage Examples
- Limitations

digia

288

Mobility API Roadmap

1.0.2 | 1.1 | 1.2 or later?

- Service Framework
- Publish&Subscribe
- Messaging
- Contacts
- System Information
- Bearer Management *(to QtNetwork in 4.7)*
- Versit
- Sensors
- Multimedia
  - Camera
- Location
  - Landmarks, Maps, Navigation
- Organizer
- Document Gallery
- Feedback
- Local Connectivity (BT, NFC)
- *Augmented Reality*
- *Face Recognition*

digia

289

Status at 1.0.2

	API Maturity Level	Tier 1 Platforms					Tier 2 Platforms			
		S60 3rd Edition, Feature Pack 1	S60 3rd Edition, Feature Pack 2	S60 5th Edition	Symbian^3	Maemo 5	Windows CE/Mobile	Windows XP/Vista	Linux	Mac OS X
Service Framework (in-process)	FINAL									
Messaging	FINAL									
Bearer Management	FINAL									
Publish and Subscribe	FINAL									
Contacts	FINAL									
Location	FINAL									
Multimedia	FINAL									
System Information	FINAL									
Sensors	FINAL									
Versit	FINAL									

290

Status at 1.1.0

	Tier 1 Platforms						Tier 2 Platforms		
	S60 3rd Edition, Feature Pack 1	S60 3rd Edition, Feature Pack 2	S60 5th Edition	Symbian	Maemo 5	Harmattan	Windows XP/Vista	Linux	Mac OS X
Service Framework (in-process)									
Messaging									
Bearer Management									
Publish and Subscribe									
Contacts									
Location									
Multimedia									
System Information									
Sensors									
Versit(vCard)									
Versit(Organizer)									
Camera									
Service Framework(OOP)									
Organizer									
Landmarks									
Document Gallery	*)	*)	*)						
Maps/Navigation									
Feedback									

*) Backend not enabled in pre-built packages

291

Installation

- Comes with Nokia Qt SDK
- Simply install the provided .sis file to the device

digia

292

First Intro to Mobility APIs

We're using class QSystemInfo from the System Information API

In Symbian, we'll need to use `#include <qsysteminfo.h>` elsewhere
`#include <QSystemInfo>` is ok

```
#include <QtGui/QApplication>
#include <QtGui/QLabel>
#include <qsysteminfo.h>

QTM_USE_NAMESPACE;

int main( int argc, char *argv[] )
{
        QApplication app( argc, argv );
        QSystemInfo s;
        QLabel *label = new QLabel( "Current language is "+ s.currentLanguage() +
          " and you're using Qt " + s.version(QSystemInfo::QtCore) );
        label->show();

        return app.exec();
}
```

digia

293

First Intro to Mobility APIs

Equivivalent to
`using namespace QtMobility;`

All classes within Mobility APIs are placed inside namespace QtMobility.
You can raise the whole namespace or either use syntax
`QtMobility::<ClassName>`, like
`QtMobility::QSystemInfo`

```
#include <QtGui/QApplication>
#include <QtGui/QLabel>
#include <qsysteminfo.h>

QTM_USE_NAMESPACE;

int main( int argc, char *argv[] )
{
        QApplication app( argc, argv );
        QSystemInfo s;
        QLabel *label = new QLabel( "Current language is "+ s.currentLanguage() +
          " and you're using Qt " + s.version(QSystemInfo::QtCore) );
        label->show();

        return app.exec();
}
```

digia

294

Mobilizing the Project File

```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

CONFIG += mobility
MOBILITY += systeminfo
# Input
SOURCES += main.cpp
```

digia

295

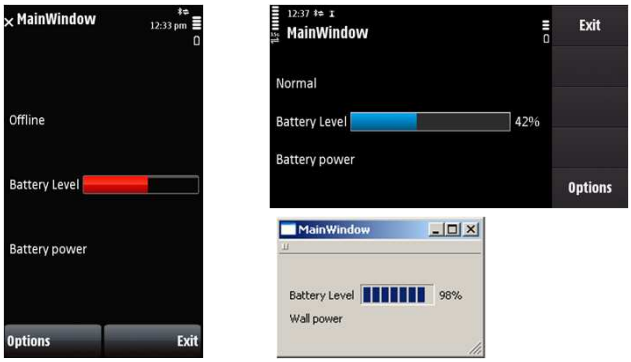
More System Information API Classes

QSystemDeviceInfo	Device information (battery, power state, input method type, IMEI, manufacturer, profile status etc.)
QSystemDisplayInfo	Display information (color depth, brightness)
QSystemInfo	General system information (like in the previous example)
QSystemNetworkInfo	Network information (network name, signal strength, network mode, etc.)
QSystemScreenSaver	Access to screen saver (inhibiting it)
QSystemStorageInfo	Memory and disk information (drive types, free space)

digia

296

QSystemDeviceInfo Example



- Small application to show the current profile, battery level and power status
  - Profile status naturally not available on Windows

digia

297

Using QSystemDeviceInfo – Example (1/4) Window Class Header File

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

public slots:
    void changePowerState( QSystemDeviceInfo::PowerState state );
    void changeProfileInfo( QSystemDeviceInfo::Profile profile );

private:
    Ui::MainWindow *ui;
    QtMobility::QSystemDeviceInfo* m_device;
};
```

digia

298

Using QSystemDeviceInfo – Example (2/4) Constructor


```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow), m_device( new QSystemDeviceInfo(this) )
{
    ui->setupUi(this);
    // batteryBar is a QProgressBar
    ui->batteryBar->setValue(m_device->batteryLevel());
    changePowerState(m_device->currentPowerState()); // Self-created slot function
    changeProfileInfo(m_device->currentProfile()); // Self-created slot function

    connect( m_device, SIGNAL(batteryLevelChanged(int)),
             ui->batteryBar, SLOT(setValue(int)) );
    connect( m_device, SIGNAL(powerStateChanged(QSystemDeviceInfo::PowerState)),
             this, SLOT(changePowerState(QSystemDeviceInfo::PowerState)));
    connect( m_device, SIGNAL(currentProfileChanged(QSystemDeviceInfo::Profile)),
             this, SLOT(changeProfileInfo(QSystemDeviceInfo::Profile)));
}
```

digia

299

Using QSystemDeviceInfo (3/4) Examining the Power State

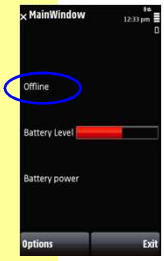
```
void MainWindow::changePowerState( QSystemDeviceInfo::PowerState state ) {
    switch( state ) {
    case QSystemDeviceInfo::BatteryPower:
        ui->stateLabel->setText( "Battery power");
        break;
    case QSystemDeviceInfo::WallPower:
        ui->stateLabel->setText( "Wall power" );
        break;
    case QSystemDeviceInfo::WallPowerChargingBattery:
        ui->stateLabel->setText( "Wall power and charging" );
        break;
    default:
        ui->stateLabel->clear();
    }
}
```

digia

300

Using QSystemDeviceInfo (4/4) – Examining the Profile Change

```
void MainWindow::changeProfileInfo( QSystemDeviceInfo::Profile profile ) {
    switch( profile ) {
    case QSystemDeviceInfo::SilentProfile:
        ui->profileLabel->setText( "Silent" );
        break;
    case QSystemDeviceInfo::NormalProfile:
        ui->profileLabel->setText( "Normal" );
        break;
    case QSystemDeviceInfo::LoudProfile:
        ui->profileLabel->setText( "Loud!" );
        break;
    case QSystemDeviceInfo::OfflineProfile:
        ui->profileLabel->setText( "Offline" );
        break;
    case QSystemDeviceInfo::CustomProfile:
        ui->profileLabel->setText( "Custom" );
        break;
    default:
        // VibProfile or PowersaveProfile are not handled separately
        ui->profileLabel->clear();
    }
}
```



digia

301

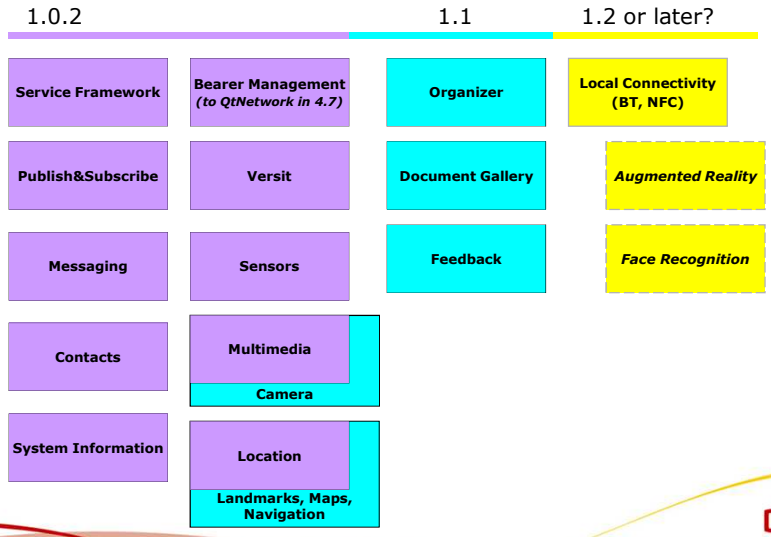
Qt Mobility APIs

API Overviews

digia

302

Mobility API Roadmap

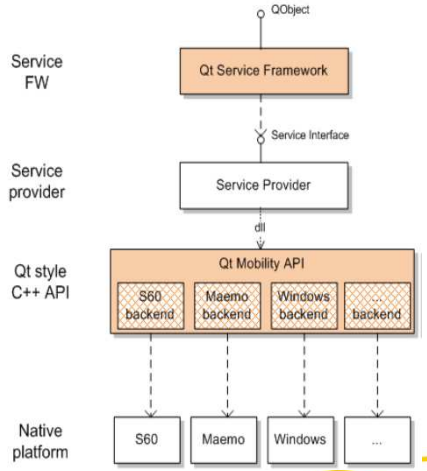


digia

303

Service Framework API

- Defines a unified way of finding, implementing and accessing services across multiple platforms.
  - Client-server (service provider) abstraction
- Functionality can be shared between applications
- Developer has device independent methods for finding/using/implementing services



Picture: Qt Mobility Whitepaper, http://qt.nokia.com/files/pdf/qt-mobility-whitepaper-1.0.0

digia

304

Service Client

- A client application loads services with QServiceManager
  - Refers to the name of the interface
  - For example: com.digia.qt.example.clock_service
- The client uses the service as a QObject
  - Methods are used through the meta-object system
  - Signals & slots
  - There is no dependency (source code or binary) between the client and the service

digia

305

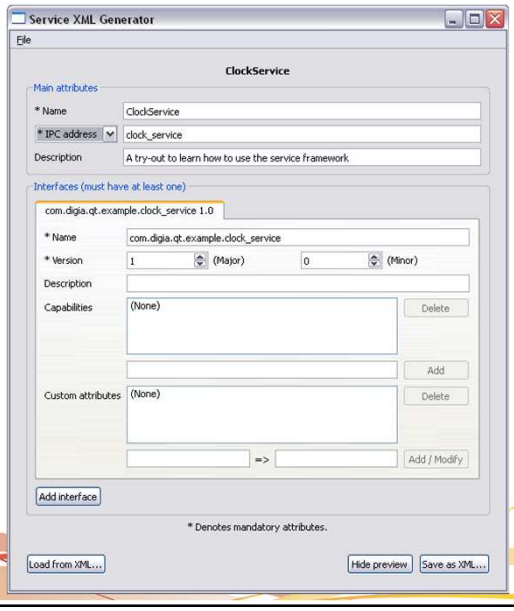
Service Provider: Server

- Service description needs to be specified
  - Programmatically by calling QServiceManager::addService()
  - (in future, QtMobility 1.2), Installing the service description XML file
  - bin/servicefw add myservice.xml
- Server Application
  - QCoreApplication
  - Running the standard Qt event loop
  - Implements services as QObjects registered to the service framework
  - Services/QObjects can be shared (nor not shared) with multiple clients.
- Service framework launches the server automatically
  - The first client
  - Server can be configured to shutdown when the last client exits

digia

306

Service Specification



digia

307

Snippet from an Example Client

```
const QString interfaceName(
      "com.digia.qt.example.clock_service");
QtMobility::QServiceManager manager;
QObject* service = manager.loadInterface(interfaceName);

QObject::connect(
      button, SIGNAL(clicked()),
      service, SLOT(requestTime()));
QObject::connect(
      service, SIGNAL(hereItComes(QString)),
      &label, SLOT(setText(QString)));
const int result = QApplication::exec();
```

digia

308

Example Server, Adding the Service

```
QCoreApplication app(argc, argv);

const QString serviceName("ClockService");
const QString interfaceName("com.digia.qt.example.clo…");
const QString serviceVersion("1.0");

QtMobility::QServiceManager manager;
bool addServiceOk(manager.addService("clockserv.xml"));
Q_ASSERT(addServiceOk);

// … continues …
```

digia

309

Example Server

```
QtMobility::QRemoteServiceRegister::Entry
  entry = serviceRegister.createEntry<ClockProvider>(
      serviceName, interfaceName, serviceVersion);

QtMobility::QRemoteServiceRegister serviceRegister;
serviceRegister.publishEntries("clock_service");
serviceRegister.setQuitOnLastInstanceClosed(true);

return QCoreApplication::exec();
```

digia

310

The Actual Service

- The actual service is implemented as just a QObject
  - Nothing special
- The service is used, for example, through signals and slots specified in the interface of the class.

digia

311

Publish & Subscribe

- The Publish and Subscribe API enables applications to read item values, navigate through and subscribe to change notifications
- Values are represented by a `QValueSpace`
  - Hierarchical tree of which each node or leaf can optionally contain a `QVariant` value
    - `QVariant` is a union data type
    - Nodes act as "Paths" which can be subscribed to

```
Serialized QValueSpace example:

/Device/Buttons = 3
/Device/Buttons/1/Name = Menu
/Device/Buttons/1/Usable = true
/Device/Buttons/2/Name = Select
/Device/Buttons/2/Usable = false
/Device/Buttons/3/Name = Back
/Device/Buttons/3/Usable = true
```

- Access with `QValueSpaceSubscriber`
  - Read values, receive change notifications, navigate through QValueSpace
- New values are added with `QValueSpacePublisher`

digia

312

Messaging API

- Access to messaging services
  - Search and sort
  - Create and modify
  - Send and retrieve
  - Launch preferred message client
- A unified interface for manipulation and storage of SMS, MMS, Email and XMPP messages is provided

digia

313

Key Classes for Messaging

- Composition and manipulation of messages:
  - `QMessage`
  - `QMessageAddress`
- Accessing message accounts
  - `QMessageAccount`
  - `QMessageFolder`
- Sorting and filtering
  - `QMessageStore`
  - `QMessageFilter`
- Accessing message services
  - `QMessageService`

digia

314

Creating a Message

```
// The developer creates a QMessage object and then sets the
   necessary message details. First set the message type, the
   default account for messages of the specified type will be used
   for sending :
QMessage message;
message.setType(QMessageAddress::Email);
// Now a recipient is set :
QString recipient("user@example.com");
message.setTo(QMessageAddress(QMessageAddress::Email, recipient));
// For email a subject and a body are set, and any relevant
   attachments added :
message.setSubject("Example subject");
message.setBody("Example body text");
QStringList attachmentPaths;
attachmentPaths << "images/landscape.png";
message.appendAttachments(attachmentPaths);
```

digia

315

Sending the Message

```
// The message is ready to be sent. Next, create a service
   object and call the send() function
QMessageService *m_service = new QMessageService();
if (!m_service->send(message)) {
   QMessageBox::warning(0, tr("Failed"), tr("Unable to send
                                            message"));
}
```

Opening the Default Composer

```
// Will open the default composer for messages of this type and
  with the existing message as the initial situation
QMessageService *m_service = new QMessageService();
m_service->compose(message)
```

digia

316

Location API

- This API provides an easy to use interface that encapsulates basic geographical information obtained from satellite or other sources about the user
  - QGeoPositionInfo class contains information gathered on a global position, direction and velocity at a particular point in time
- Multiple methods for receiving location data
  - GPS
  - Cell ID
  - Anything derived from QGeoPositionInfoSource
- Also support for direct access to any NMEA data
  - Common text-based protocol for navigational data

digia

317

Monitoring Position Data

- Default position source may be available on some platforms
  `QGeoPositionInfoSource::createDefaultSource()`
- QGeoPositionInfoSource is simple to use
  - `requestUpdate()`
  - `startUpdates()` *(and `setUpdateInterval()`)*
  - `positionUpdated( QGeoPositionInfo )` [signal] is emitted after each interval
  - `stopUpdates()`

digia

318

Simple Client Receiving Location Data

```
class MyClass : public QObject {
  Q_OBJECT
  public:
   MyClass(QObject *parent = 0) : QObject(parent) {
      QGeoPositionInfoSource *source =
           QGeoPositionInfoSource::createDefaultSource(this);

      if (source) {
         connect(source, SIGNAL(positionUpdated(QGeoPositionInfo)),
                 this, SLOT(positionUpdated(QGeoPositionInfo)));
         source->startUpdates();
      }
   }

 private slots:
   void positionUpdated(const QGeoPositionInfo &info) {
      qDebug() << "Position updated:" << info;
   }
};
```

digia

319

Contacts API

- The Contacts API allows developers to manage contact data in a platform independent way.
  - A contact is the digital representation of a person, group or entity
  - A contact consists of a set of contact details with own semantics of usage and storage with different context info (like separate phone number for work and home)
- `QContactManager` unifies one or more platform-specific contact backends

digia

320

Main Contact Classes

`QContact`	Addressbook contact
`QContactDetail`	Single detail of a `QContact`
`QContactManager`	Access to contacts stored in particular backend
`QContactFilter`	Used to select contacts through `QContactManager`
`QContactAction`	Interface for performing actions to contacts (like ”Send email” or ”Dial”)

digia

321

Versit API

- Functionality for reading and writing Versit documents such as vCards
  - `QVersitDocument`
  - `QVersitReader`
  - `QVersitWriter`
- Utilities to import/export QContacts from/to Versit documents
  - `QVersitContactImporter`
  - `QVersitContactExporter`

digia

322

Bearer Management API

- Now part of Qt 4.7 QtNetwork!
- Manages the connectivity state to the network
  - Allows user to start or stop network interfaces
  - Is device online and how many available interfaces there are
- Allows comparison and prioritization of the access and use of grouped access points
- When using Bearer Management the developer does not need to worry about locating the best connection
  - User selects best
  - Transparent selection
- Automatic roaming between cellular and WLAN networks

digia

323

Classes for Bearer Management

`QNetworkConfiguration`	Abstraction of one or more access point configurations.
`QNetworkConfigurationManager`	Manages the network configurations provided by the system
`QNetworkSession`	Control over the system's access points and enables session management for cases when multiple clients access the same access point

digia

324

Small Example on Bearer Management

- Application lists available network configurations and shows information on the selected one
  - UI consists only of a `QListView` and a `QTextBrowser`
  - `QListView` uses `QStringListModel` (which has the names of the configurations)
- For a more complex example, see Bearer Monitor example of the Mobility APIs

digia

325

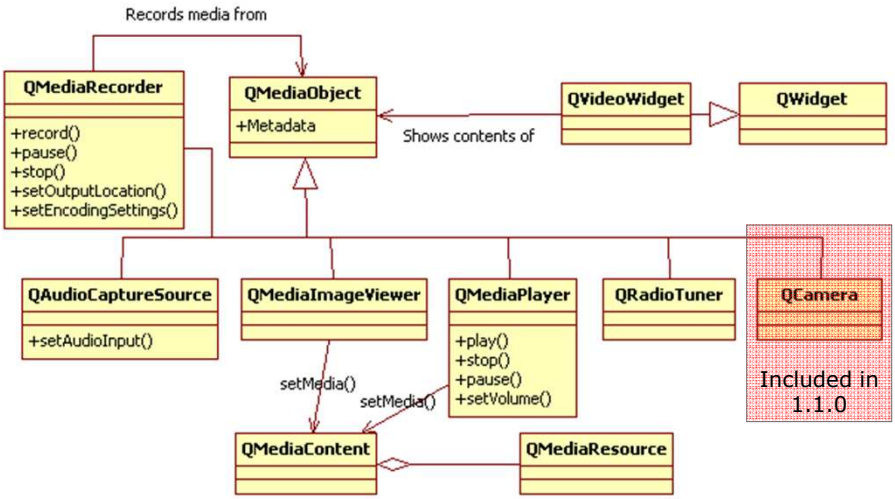
Multimedia APIs

- Play audio & video of various formats
- Record audio
- Playing and managing of an FM radio
- With QtMultimedia, will eventually replace Phonon API
- Access to multimedia services with minimal code and maximal flexibility

digia

326

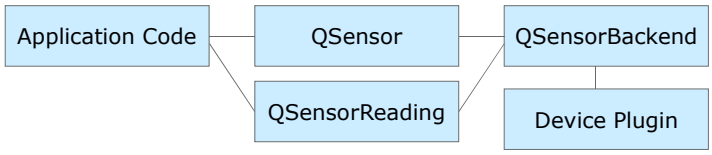
Overview on Multimedia API (Main Classes Only)



327

Sensors API

- The API supports sensors that poll for their data and sensors that push data to the app as it arrives



- `QSensor` (and its subclasses) provide application with access to data input from a sensor
  - Direct subclass instantiation
  - QSensorReading subclasses represent single readings from a single sensor
- `QSensorBackend` can be used to make sensors available through the same API, by creating plugins

digia

328

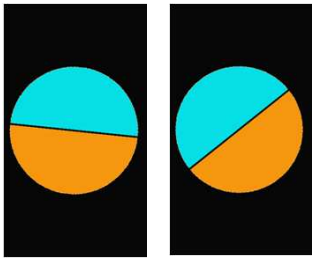
Existing QSensor Subclasses

`QAccelerometer`	Linear acceleration along the X, Y and Z axes
`QAmbientLightSensor`	Ambient light sensor
`QCompass`	Compass
`QMagnetometer`	Magnetometer
`QOrientationSensor`	Orientation
`QPromiximitySensor`	Proximity (”if something is close”)
`QRotationSensor`	Rotation
`QTapSensor`	Tap sensor (registers tap and double tap events in 6 directions)

digia

329

Demo: Using Sensors



Horizonize with Orientation sensor

Proximity Sensor Test

digia

330

Now you, Adaptive Flashlight Hands-On!

- Start with the given code template
  - Add an ambient light sensor to the example

digia

331

Mobility API Checklist

- Mobility API libraries in the device
- Mobilize .pro file
- Include classes (with classname.h)
- QTM_USE_NAMESPACE;

digia

332

MeeGo Application and Platform Development

Platform Development

digia

Agenda

- Tools
- Glib Library
- Using D-Bus
- Qt4 APIs
- API Wrappers

digia

334

Platform Debugging Tools

- Useful tools:
  - strip: removes symbols and sections which are not useful for running/loading the ELF binary (not exhaustive as it could be, but safe)
  - sstrip: does strip job, a bit better though.
  - objdump: displays information for a binary file
  - ht (hteditor): hexadecimal editor, disassembler. Many others exists …
- Binaries are loaded via libbfd for gdb or objdump tools
- Libbfd is a really powerful library which can manage many different files format and system architecture

digia

335

Tracing

- The most common tool in Linux is: strace
- Basic usage: strace <your_command>
- It will print out all system calls and partially their parameters, it is useful when an error occurs on a system call as a first look (easier than directly going on with a debugger)
- Might be used as well to get some statistics on which and how many times a system call has been launched. (-c option)
- You can filter which system calls you want to survey, to reduce the amount of dump displayed. (-e trace=<type of syscall> for instance: -e trace=network to see all network related sys calls like socket, recv, send ...)

digia

336

Profiling and Instrumenting: Valgrind

- Valgrind it a tool suite, mainly to watch out all memory related part of a program in runtime.
- Very powerful to locate any kind of leaks that could happen.
- Also useful to instrument your program on memory usage, in different levels: heap, stack, cache...
- The tools are so complete and full of features that we will just introduce the basic stuff about them.
- Has KDE/Qt based GUI: kcachegrind

digia

337

Profiling and Instrumenting: valgrind

- Basic usage:
  - Valgrind tool=<tool name> <parameter> <your command>
    - -v parameter adds verbosity
    - --help parameter for more information on tool
- Checking for memory leaks:
  - Memcheck
    - --leak-check=full parameter shows you where, in C source, an non-freed allocation or a leak occurred
- Checking for heap utilization:
  - massif
- Instrumenting cache usage:
  - cachegrind
- Checking for thread safety (from resource point of view):
  - helgrind
- Usually: memcheck is the most widely used. It permits to find all possible memory leaks and memory intensive functions

digia

338

Profiling and Instrumenting: gprof

- When a software is tested and leak free, it is good to determine where optimization could be done
- Gprof is dedicated to that (valgrind with tools like callgrind can do about the same, it is even easier through kcachegrind)
- Just add « -pg » CFLAGS to gcc when compiling your software.
- Then run it, at the end you should get a file in same location named « gmon.out ».
- Now you can use gprof <options> <your software> to get the profile informations:
  - -p -q: flot profile + call graph
  - -b: same thing without legend
- Now take the function which takes most of time, and determine if it can be optimized or not, rebuilt/relaunch and continue with each call, until it becomes unnecessary.

digia

339

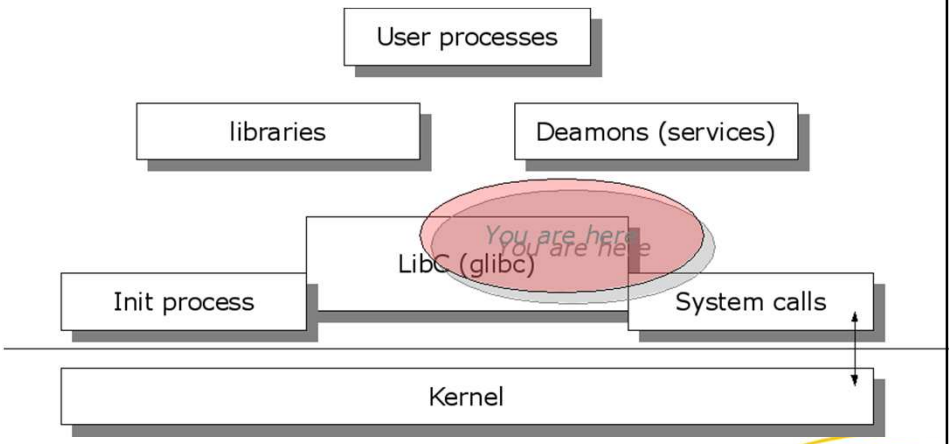
Profiling and Instrumenting: gcov

- Gcov is a basic code coverage tool it is useful to help for finding dead code
- As gprof, it first needs CFLAGS to gcc:
  - -fprofile-arcs -ftest-coverage
- Then run your program, test program, whatever
- It should create 2 files with extension:
  - .gcda
  - .gcno
- Run gcov <your program main source file>
- You should get the percentage of executed lines, and it creates a .gcov file where you will find the non executed lines targeted by a « ##### » instead of a number.

digia

340

Platform Architecture



digia

341 © 2009 Digia Plc

Libglib

- Provides GType runtime type system and GObject frameworks
  - Supports runtime binding of GTK+ widgets to interpreted languages
  - Uses private structures like Qt
  - For type identification and management
- GObject provides a generic type system with inheritance and a powerful signal system based on callbacks
  - Compare to QObject callback system

digia

342 © 2009 Digia Plc

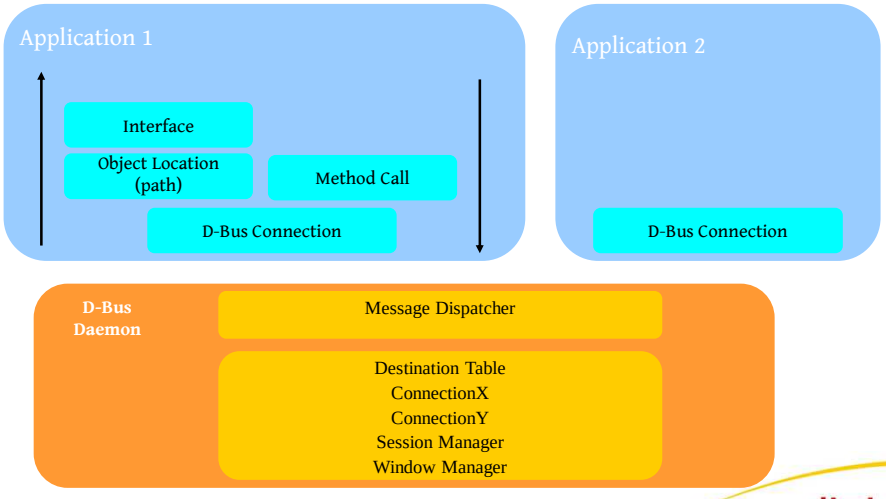
Libdbus

- A library to access D-Bus daemon at the low level
  - Libdbus-glib (qt, java etc) provides GLib-like interface (GObject, GType system) to access D-Bus
  - Use of these wrapper libraries preferred to direct libdbus use
- D-Bus itself is a server
  - Message (remote procedure call) –based communication between processes
- Applications (service providers) send messages to D-Bus, which routes the messages to one or more receiver
- Abstract like CORBA
  - D-Bus does not define, which mechanism (e.g. sockets) actually transfers the messages
- Two kinds of daemons
  - System-wide singleton (for system messages, such as signal strength, battery level)
  - User session –specific (between applications)

digia

343

D-Bus Basics



digia

344

D-Bus Concepts – 1(2)

- Object **paths**
  - A mechanism to locate, which native object (GObject, Java Object, Qt QObject) provides a service
  - E.g. `company/services/serviceX`
- Each object may have method and signal **members**
  - Methods are (remote procedures) operations which can be invoked on an object with optional input and (possibly several) output values
  - Signals are broadcast from an object to all its observers (may contain data)
  - E.g. `doSomething, notify`
- Member group is mapped to an **interface**
  - Mapped to Java interface or C++ pure virtual class
  - Identified as `com.company.InterfaceName`

digia

345

D-Bus Concepts – 2(2)

- **Bus names**
  - D-Bus daemon assigns a unique connection name for each connection from applications
  - After a name is mapped to an application, the application owns that name
  - Applications may ask to own well-known names, e.g. `com.digia.MessageEditor`
- **Addresses** specify where a server will listen and where a client will connect
  - Possibly, your service is a server daemon to which applications send messages

digia

346

D-Bus Example

```
#include <dbus/dbus.h>
#define SYSNOTE_NAME "com.digia.Services"
#define SYSNOTE_OPATH "/com/digia/Services"
#define SYSNOTE_IFACE "com.digia.Services"
#define SYSNOTE_NOTE "ServiceX"
DBusConnection* bus = NULL;
DBusMessage* msg = NULL;
DBusError error;
int arg=1;
dbus_error_init(&error); // Clear error
bus = dbus_bus_get(DBUS_BUS_SESSION, &error); // Init connection and handle error
msg = dbus_message_new_method_call(SYSNOTE_NAME, SYSNOTE_OPATH, SYSNOTE_IFACE,
    SYSNOTE_NOTE);
if (msg == NULL) ; // Handle error
if (!dbus_message_append_args(msg, DBUS_TYPE_UINT32, &arg, DBUS_TYPE_INVALID)) ;
    // Handle error
dbus_message_set_no_reply(msg, TRUE);
if (!dbus_connection_send(bus, msg, NULL)) ; // Queue msg and handle error
dbus_connection_flush(bus);
```

digia

347 © 2009 Digia Plc

Bindings

- There are D-BUS bindings/wrappers to several other framework, such as GLib, Qt, Java
- Maps D-Bus types (primitive types, arrays, hash tables) to GType, Qt or Java types
  - <dbus/dbus-glib.h>
- You may create interface bindings from Introspection XML files
  - `dbus-binding-tool --mode=glib-client some-object.xml > some-object-bindings.h`

digia

348 © 2009 Digia Plc

D-Bus Example with GLib Bindings

```
DBusGConnection *connection;
GError *error;
DBusGProxy *proxy; // Native object representing a remote object
char **name_list;
g_type_init (); // Required to initialize GType system
error = NULL;
connection = dbus_g_bus_get (DBUS_BUS_SESSION, &error);
if (connection == NULL) ; // Handle error
/* Create a proxy object for the "bus driver" (name "org.freedesktop.DBus") */
proxy = dbus_g_proxy_new_for_name (connection, DBUS_SERVICE_DBUS,
    DBUS_PATH_DBUS, DBUS_INTERFACE_DBUS);
/* Call ListNames method, wait for reply */
error = NULL;
if (!dbus_g_proxy_call (proxy, "ListNames", &error, G_TYPE_INVALID, G_TYPE_STRV,
    &name_list, G_TYPE_INVALID)) ; // Handle error
/* Handle result and free resources */
g_strfreev (name_list);
g_object_unref (proxy);
```

digia

349 © 2009 Digia Plc

Method Invocation Using Glib Bindings

- `dbus_g_proxy_call()`
  - Send a message and wait for the reply
  - Outgoing arguments specified as an array of type and var names terminated with `G_TYPE_INVALID`
    - `G_TYPE_INT`, 7
  - Pointers to return values specified in the same way
- `dbus_g_proxy_begin_call()`
  - Asynchronous call
  - Set a callback notification function using `dbus_g_pending_call_set_notify()`
- Errors are handled using `GError`
  - It may represent an internal D-Bus error (domain `DBUS_ERROR`) – often ignored by the application or
  - An exception thrown by the peer application

digia

350 © 2009 Digia Plc

Implementing Service Objects – 1(2)

- Specify a service object and its methods

```
<?xml version="1.0" encoding="UTF-8" ?>
<node name="/com/digia/ExampleObject">
   <interface name="com.digia.ExampleObject">
     <annotation name="org.freedesktop.DBus.GLib.CSymbol"
   value="“example_object"/>
     <method name="DoIt">
       <!-- This is optional, and in this case is redununudant -->
       <annotation name="org.freedesktop.DBus.GLib.CSymbol"
   value="“example_object_do_it"/>
       <arg type="u" name="“age" direction="in" />
       <arg type="s" name="“name" direction="in" />
       <arg type="d" name="d_ret" direction="out" />
       <arg type="s" name="str_ret" direction="out" />
     </method>
   </interface>
</node>
```

digia

351 © 2009 Digia Plc

Implementing Service Objects – 2(2)

- Create a server header file
  - `dbus-binding-tool --mode=glib-server example-object.xml > example-object-glue.h`
- Initialize the service object in the class initializer, passing the object class and "object info" included in the header
  - `dbus_g_object_type_install_info (COM_DIGIA_EXAMPLE_OBJECT, &com_digia_example_object_info);`
- Implement the object methods
  - `gboolen example_object_do_it(ExampleObject* obj, /* Args */ ) { return true } // Must return TRUE on success, FALSE otherwise`
  - The first parameter is a pointer to an instance of the object, followed by the method input values, followed by the pointers to return values
  - The final parameter must be a GError **
    - If the function returns FALSE for an error, the error parameter must be initalized with g_set_error
- Finally, export the object using dbus_g_connection_register_g_object
  - `dbus_g_connection_register_g_object (connection, "/com/digia/ExampleObject", obj);`

digia

352 © 2009 Digia Plc

D-Bus, The Qt Way

- QtDBus allows to call methods of D-Bus objects
- QtDBus allows to connect signals and slots between D-Bus objects
- Since it uses the meta object information, it is not necessary to know the interface of the remote object
- QtDBus takes care of mapping Qt datatypes to the defined D-Bus datatypes
- QtDBus resolves object names to interfaces with the correct signals and slots

digia

353

Calling Methods on D-Bus Objects

- In QtDBus, the slots on the remote object can be called as if the object was local
- To call a method, an QDBusInterface has to be retrieved for it first
- The method can then be called using QDBusInterface::call
- Example:

```
QDBusReply<QString> reply =
                    iface.call( "echo", "hi" );
```

calls the slot named echo on the remote object with argument “hi”.

digia

354

Signals and Slots between D-Bus Objects

- Signals of one D-Bus object can be connected to slots of another one seamlessly
- First, retrieve a QDBusInterface for a known object name
- The QDBusInterface object has the same signals and slots as the remote object it represents
- Note: QDBusInterface uses D-Bus introspection to discover the signals and slots of the remote object
- Connections are created with the regular QObject::connect methods
- The exposed signals and slots can be restricted by the remote object (`QDBusConnection::RegisterOptions()`)

digia

355

Mapping between QtDBus and D-Bus Datatypes

- QtDBus needs to map Qt datatypes to types known by D-Bus
- All arguments marshaling is taken care of by Qt
- Supported Datatypes: uchar, bool, short, ushort, int, uint, qlongling, qulonglong, double, QString, QStringList, QByteArray, and special D-Bus types
- Compound types can be formed as arrays, structs, and maps
- To use custom datatypes,
  - declare the type using Q_DECLARE_METATYPE(),
  - and register it using qDBusRegisterMetaType().

digia

356

Important Methods

- **QDBusConnection::sessionBus()/systemBus():** access to the bus objects
- **QDBusConnection::registerService():** register a service ("host part")
- **QDBusConnection::registerObject():** register an object ("file part")
- **QDBusInterface::call():** synchroneously call a method on a remote object
- **QDBusInterface constructor** constructs a QObject that represents the signals and slots of the remote object

digia

357

Qt Servers

- Initialize QCoreApplication instead of QApplication
- Use D-Bus Qt bindings

```
#include <QtDBus/QtDBus>
//
class MyServer : public QObject
{
    Q_OBJECT
public slots:
    QString method1(const QString &arg);
};
// Main starts
if (!QDBusConnection::sessionBus().isConnected()) { }

if (!QDBusConnection::sessionBus().registerService(SERVICE_NAME))
   { }

MyServer server;
QDBusConnection::sessionBus().registerObject("/", &server,
   QDBusConnection::ExportAllSlots);
```

digia

358

Inter-Process Communication

- Sockets
  - Other processes send event through socket notifiers
  - Socket notifiers are added to QAbstractEventDispatcher
  - Create events from socket notifications
- QSharedMemory
  - Reference count object
  - Can be opened by any process
  - Memcpy() used to read/write to the process
- Servers (QCoreApplication instances)
  - No UI
  - QLocalSocket used (local loop TCP socket)
- QCop (Qt Communication protocol)
  - Available only in Qt for embedded Linux
- DBus
  - Extends signal/slot mechanism between processes
  - DBus protocol must be supported by the platform

digia

359

Shared Memory

- Use QSharedMemory (or QtSharedMemory)
- Works between processes and threads
  - Processes recognize the piece of shared memory using a key (QString)
  - Process attach and detach to shared memory using the key
- Do not deallocate shared memory buffer
  - Reference count – will be freed, when all QSharedMemory objects referencing it have been deleted
- Mutual exclusion is taken care by the developer
  - Use lock() and unlock()

digia

360

Shared Memory Example

```
QSharedMemory mem(QString("thekey"), 0);
// Create a new/old shared memory
mem.create(SIZE);
mem.lock(); // uses QSystemSemaphore internally
char *to = (char*)mem.data();
const char *from = buffer.data();
memcpy(to, from, qMin(mem.size(), SIZE));
mem.unlock();
```

digia

361

MeeGo Platform

Wrapping Native MeeGo APIs

digia

Contents

- Writing API Wrappers
- Using D-Bus Services

digia

363 © 2009 Digia Plc

Native MeeGo APIs

- MeeGo platform services may be accessed in several ways
  - Using Qt libraries (prefer this, if a suitable library exists)
  - Linking to native libraries providing typically a GObject-like interface
  - Accessing services over sockets (e.g. Bluetooth)
  - Using Desktop Bus (D-Bus) to access services in other processes
- If any other approach except Qt libraries is used, the native API access must be hidden from Qt

digia

364 © 2009 Digia Plc

GObject

- GObject is an object-oriented framework based on C
- It is based on GType runtime type system
- GObject/GType is part of Glib

- Compare GObject to QObject
  - Observer pattern based on type-cast callbacks
  - Much simpler than QObject

- A typical library API provides a set of synchronous and asynchronous functions
  - Asynchronous functions' completions are handled with callbacks
  - GType types must be converted to Qt types

digia

365

D-Bus

- Inter-process communication mechanism to access services provided in the platform and devices
- Service is provided by libdbus or higher level library
- To be able to use the service, the client needs to know
  - Service well-known name: digia.services
  - Service object(s) path(s): /digia/services
  - Service interface: digia.services
  - Service methods with parameters: serviceX
- D-Bus has its own reply and error messages, which are naturally different from Qt messages

digia

366

Hiding Platform Dependency 1 – Simple Wrapper

- By simplest, MeeGo platform API can be wrapped with a hybrid class
  - Qt class with Qt interface and mechanisms (like Signals&Slots)
  - Yet, dependencies on MeeGo platform API

⇒ UI code stays "clean" and "plain Qt" but is actually dependent on MeeGo through EngineWrapper's interface

+ Quick'n'easy

- Not very flexible, possible problems with BC if going cross-platform

digia

367

"Example" Interface

```
#include <location/location-gps-device.h>
#include <location/location-gpsd-control.h> // Middleware dependency
…
class EngineWrapper : public QObject {
   Q_OBJECT
   public:
     // Qt-style interface for engine services
     …
   signals:
     …
   private:
   LocationGPSDControl *m_control;
   LocationGPSDDevice *m_device; // Middleware dependency
};
```

digia

368

Hiding Platform Dependency 2 – Private Implementation

- More flexibility can be achieved by keeping even the EngineWrapper's interface as pure platform independent Qt
- All platform dependent functionality and data are placed inside a private class
  - A simple design pattern called *private implementation*
- Qt itself has been implemented for different platforms using the same mechanism!

digia

369 © 2009 Digia Plc

Private Implementation in Its Simplest

```
// a.h
class A {
   ...
private:
   int x;
   int y;
};
```

```
// a.h
class AP;

class A {
   ...
private:
   AP*
   d_ptr;
};
```

```
// a.cpp
class AP {
 …
 private:
   int x;
   int y;
};
...
```

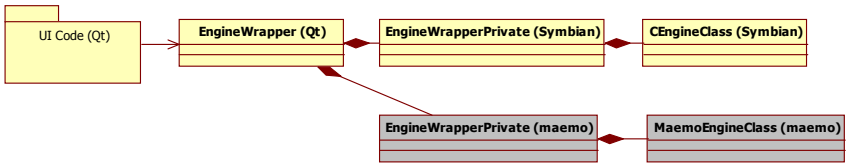
- Header file stays the same regardless of the contents of class AP
  - Binary Compatibility for A remains despite AP being changed

digia

370 © 2009 Digia Plc

More on Private Implementation

- If engine is ported for different platforms, only the private class needs to be replaced.



- This mechanism is also used for building cross-platform APIs (like Qt itself!)
  - One unified API
  - Platform-specific private implementation classes

More on Private Implementation

- In MeeGo, using a native service quite often requires the use of D-Bus or at least GObject API
  - Asynchronous RPC like communication
- The private implementation class can act as a GObject / D-Bus wrapper
  - Connects to GObject signals (different from Qt signals and slots)
    - Implemented as callack functions
  - D-Bus or GObject callback notifiy complition of an asynchronous service
  - The callback is handled in MeeGo way
    - Parameters may need to be converted to Qt types
  - The callback function notifies the public object and further the client about the completion of a request

Private Implementation Summary

+Really flexible
- Can port engine to multiple platforms
- Can integrate/transform native mechanisms (like asynchronous DBus calls) to Qt mechanisms

+All platform-specific implementation details are hidden from the interface
- Maintains Binary Compatibility

-A bit more labourous than just plain hybrid engine wrapper
- But not much

digia

373

Summary

- Avoid using native APIs if possible
- If you need to access a native API, such as Bluetooth, hide all MeeGo platform-dependent issues
- One way to achieve this is to use a private implementation design pattern, which is heavily used in Qt libraries
- QtMobility is one effort to reduce the need to access native APIs

digia

374

Thank You!

digia

www.digia.com

digia

375