Qt in Mobile Development

EFPL, Lausanne 14.-15.11.2009

Saturday

- Qt Essentials Review + Review Exercises
- Intermediate Qt App Development topics for the projects:
  - Networking
  - WebKit
  - File handling
  - Splash Screens
- Qt on Mobile Platforms
- Qt on Symbian
  - Intro, architecture
  - Tools, building Qt Apps for Symbian Platform
  - Symbian OS essentials for Qt developers
  - Mobile Extensions
  - Symbian Platform Security
- Project workshop
  - Enriching the applications with mobile features
  - Installing and testing the projects on a Symbian device

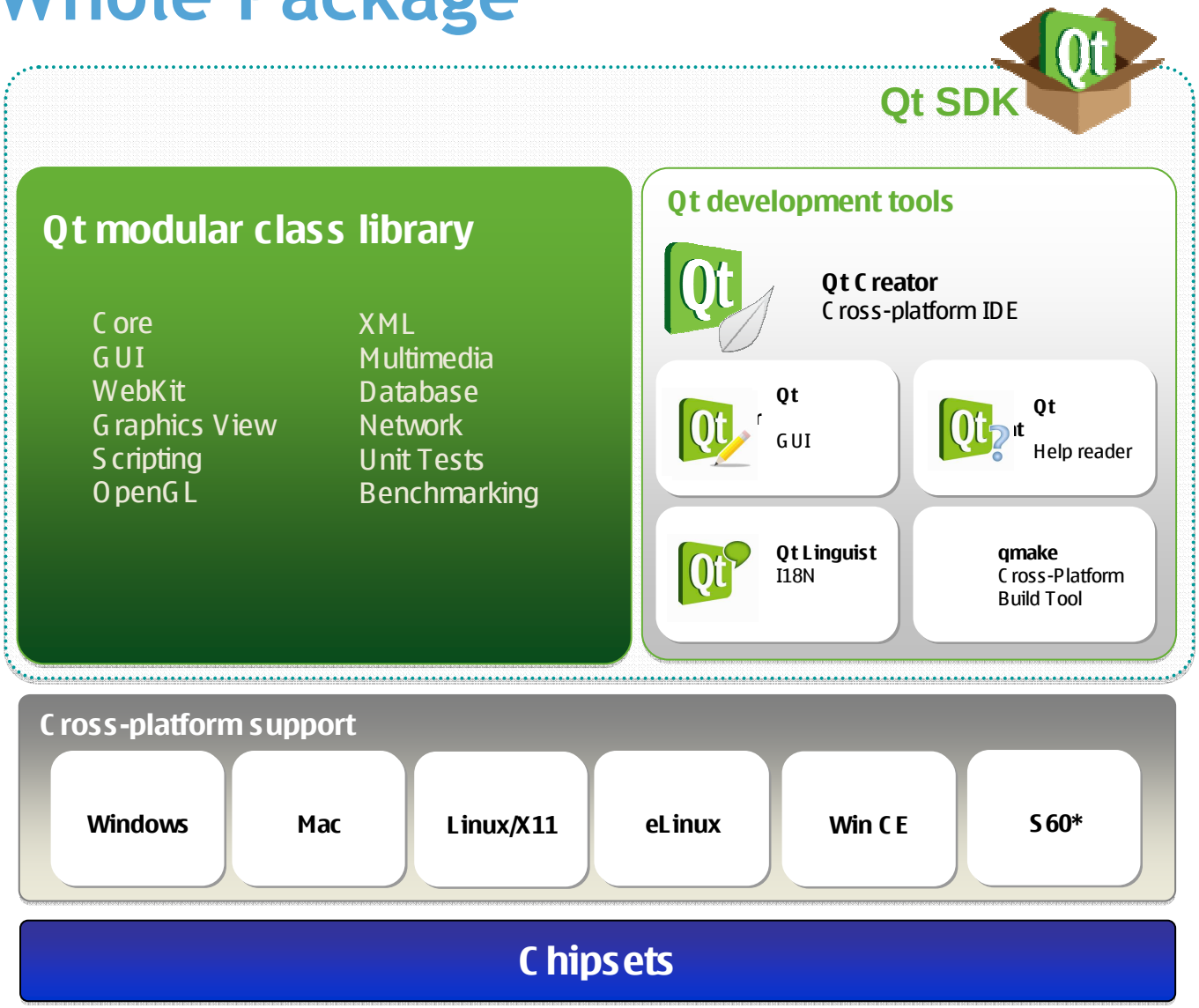
Sunday

- Intro to Maemo and Qt for Maemo
- Maemo 5 Development Environment
- Building and installing Qt projects on N900
- Project Workshop
- Q&A

Review of Qt Essentials

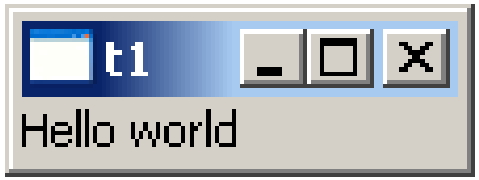
What is it and where you can use it?

As a Whole Package

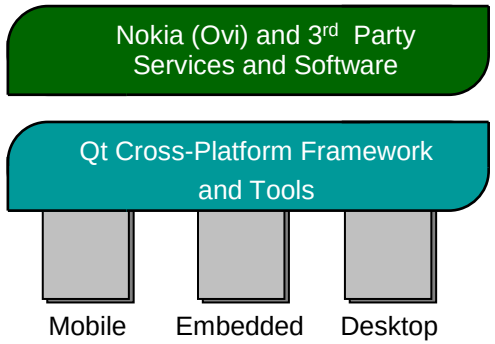


HelloWorld with Qt

```
#include <QApplication>
#include <QLabel>

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



Qt Everywhere



Relative Growth for Jobs matching "Qt C++"

Used with permission from Indeed.com
http://www.indeed.com/jobtrends?q=Qt+C%2B%2B&l=&relative=1

Qt as the de facto standard for UI and application development

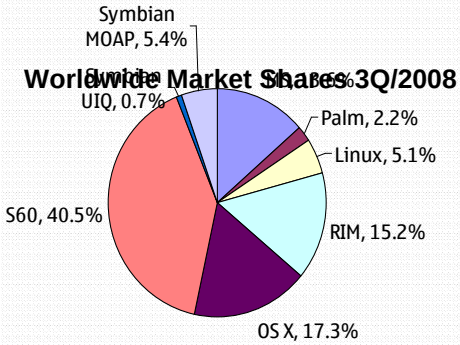
- Re-use code across devices and desktops
- Shorter time to market for devices and services

Vibrant ecosystem – 10x by 2011

- Targeting at a tenfold ecosystem with Nokia's increased investments, LGPL license option and new contribution model
- Broader use of Qt will result in feedback and increased contributions, ensuring that Qt remains at the leading edge

10

Qt on Nokia platforms



Qt for millions of S60 smart phones

S60, the world's most popular software for smart phones, will become one of Qt's supported platforms.

Qt 4.5 will be available for S60 as an optional add-on during 2009, preinstalled in devices in 2010.

Qt enables the renewal of the developer offering, with improved productivity and developer fun.

Qt for Maemo devices

The Maemo platform powers devices such as the Nokia N810 Internet Tablet.

Maemo is based on Linux/X11 – one of Qt's supported platforms. The Maemo community has enabled dedicated Qt support of Maemo in the maemo.org Garage.

Nokia services

Nokia's Internet services will run on a Qt and Webkit based common web runtime, across Nokia platforms and desktops.

11

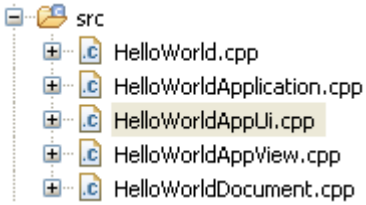
Qt for Maemo and Symbian

- Examples: N900 (Linux), N97 (Symbian)
- Cross-mobile platform Applications with Mobility APIs (coming soon)

Motivation for bringing in Qt to S60

- We already have huge mass of Symbian developers out there
- But… Symbian development productivity is not best possible
- Getting new developers to learn Symbian? The learning curve is too steep when compared to competitive offering.
- Runtimes like Java and Flash? Good, but runtime is a runtime, it always has it’s restrictions.
- We need a native solution that is productive and easy to learn.

Hello World with Symbian C++ (parts of it)




```
case ECommand2:
    {
    RFile rFile;

    //Open file where the stream text is
    User::LeaveIfError(rFile.Open(CCoeEnv::Static()->FsSession
            KFileName, EFileStreamText));//EFileShareReadersOn
    CleanupClosePushL(rFile);

    // copy stream from file to RFileStream object
    RFileReadStream inputFileStream(rFile);
    CleanupClosePushL(inputFileStream);

    // HBufC descriptor is created from the RFileStream object
    HBufC* fileData = HBufC::NewLC(inputFileStream, 32);

    CAknInformationNote* informationNote;

    informationNote = new (ELeave) CAknInformationNote;
    // Show the information Note
    informationNote->ExecuteLD(*fileData);

    // Pop loaded resources from the cleanup stack
    CleanupStack::PopAndDestroy(3); // filedata, inputFileStre
    }
    break;
```

```
void CHelloWorldAppUi::ConstructL()
    {
    // Initialise app UI with standard value.
    BaseConstructL(CAknAppUi::EAknEnableSkin);

    // Create view object
    iAppView = CHelloWorldAppView::NewL(ClientRect());

    // Create a file to write the text to
    TInt err = CCoeEnv::Static()->FsSession().MkDirAll(KFileName);
    if ((KErrNone != err) && (KErrAlreadyExists != err))
        {
        return;
        }

    RFile file;
    err = file.Replace(CCoeEnv::Static()->FsSession(), KFileName, EF
    CleanupClosePushL(file);
    if (KErrNone != err)
        {
        CleanupStack::PopAndDestroy(1); // file
        return;
        }

    RFileWriteStream outputFileStream(file);
    CleanupClosePushL(outputFileStream);
    outputFileStream << KText;

    CleanupStack::PopAndDestroy(2); // outputFileStream, file

    }
```

Hello World with Qt

```
#include <QtGUI\QApplication>
#include <QtGUI\QLabel>

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

How can I access Mobile features?

- Naturally, a mobile platform has features that are not cross-platform i.e. not applicable on desktop systems
  - Messaging
  - Telephony
  - Camera
  - Sensors…
- There's a Qt mobility project for creating APIs for mobile devices

Sending an SMS message with Symbian C++

```
void MessageSender::SendMessageL(const TDesC& recipient, const TDesC& message ) {
	RSendAs msgServer;  // The easier and restricted RSendAs API
	msgServer.Connect();
	CleanupClosePushL( msgServer );
	RSendAsMessage msg;
	msg.CreateL( msgServer, KSenduiMtmSmsUid );
	CleanupClosePushL( msg );	//Set values to recipient and body text
	msg.AddRecipientL( recipient, RSendAsMessage::ESendAsRecipientTo );
	msg.SetBodyTextL(message);
	msg.SendMessageAndCloseL();
	CleanupStack::PopAndDestroy( &msg );
	CleanupStack::PopAndDestroy( &msgServer );
}
```

Sending an SMS message with Qt

```
void MessageSender::SendMessage( QStringList recipients, QString message )
{
   XQMessaging *messaging = new XQMessaging( this );
   XQMessage message( recipient, message ));
   messaging->send(message);
}
```

Current Offering of Mobile Extensions

Mobile Extension	Purpose	Mobile Extension	Purpose
1.Access Point Manager	Listing available IAPs (Internet Access Points), setting the IAP to be used, scanning available WLANs etc.	**10.Profile**	Reading profile information and setting active profile.
2.Audio API	Providing Audio **recording** functionality	**11.Resource Access**	Accessing Symbian resource files.
3.Camera	Using device's onboard camera with viewfinder, focus and capturing images.	**12.Sensors**	Acceleration and orientation sensor data access
4.Contacts	You can access contacts database with this.	**13. Settings Manager API**	Accessing central repository and Publish & Subscribe.
5.Installer API	Silent install functionality	**14. System Information**	Accessing system information (language, battery, nw, ...) with this.
6.Landmarks	List available landmarks and add new landmarks	**15. Telephony**	Make a circuit switched call and getting call status notifications
7.Location	Accessing device location information.	**16. Utils**	Platform specific utils.
8.Media	Retrieving lists of music, image, video and sound files located in the gallery	**17. Vibra**	Using device's vibra
9.Messaging	Sending and receiving SMS and MMS messages.		

The UI is still a challenge…

- Naturally, desktop and mobile device UIs, logic and input methods are different
- Even though you implement a good Qt application on desktop, will it fit well on a mobile device?

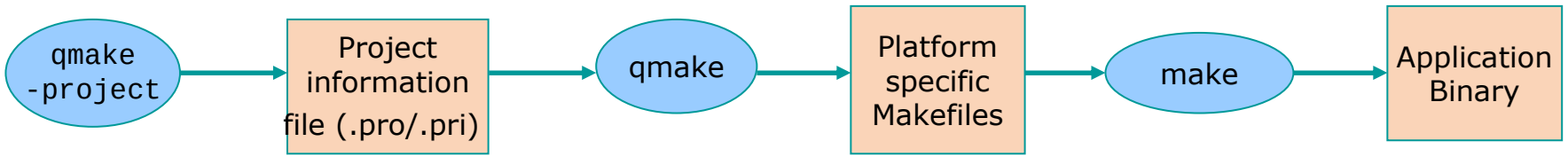
Review of Qt Essentials

Hello World

```
#include <QtGUI\QApplication>
#include <QtGUI\QLabel>

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

Building Qt Applications

1. qmake –project
   - Creates a **Qt project file (.pro)**. This can also be created manually.
2. qmake
   - Uses the **.pro** file as input and produces platform-specific Makefile(s)
   - *Generates make rules to invoke **moc** for project header files containing Q_OBJECT – as explained later*
3. make
   - Compiles the program for the current platform
   - Executes also **moc**, **uic** and **rcc**



Technical essentials to learn in Qt

- QObject
- Parent/child relationship (memory management)
- Building GUIs and layouts
- Signal-Slot mechanism
- Event handling
- Dynamic object properties
- The rest is just libraries and APIs ☺

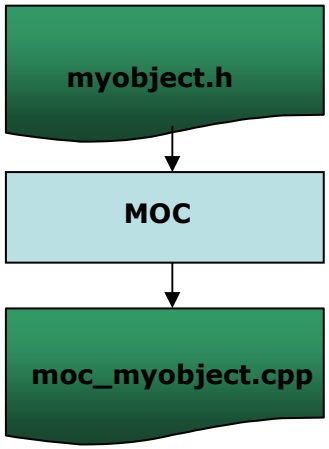
QObject

QObject class role

- Heart of Qt's object model
  - Base class for all object classes
  - Provides object trees and object ownership
  - QObject's responsibility is to provide a central location for the most important concepts in Qt
- Has four major responsibilities
  - Memory Management
  - Inter-object communications
  - Run-time type information and object properties
  - Timing services
  - Event handling
- Qt meta-object system (and compiler) needed for gaining all these

Qt's meta-Object System

- Meta-object system extends C++ with dynamic features – similar to those in Java, for example
- Dynamic features include
  - Mechanism to access any functions in the class
    - Also private ones
    - Used by signals and slots
  - Class information
    - Type without RTTI (Run-Time Type Information)
    - Information about base classes
  - Translate strings for internationalization
  - **Dynamic properties**



Qt's meta object system

```
class MyClass: public QObject
{
   Q_OBJECT
public:
    // declare public members here
private:
   // declace private members here
public slots:
    void mySlot();
  // slots can also be called as normal member functions
signals:
    void mySignal(int value);
};
```

Memory Management

It’s all about inter-object communications…

Memory management rules

- There are 2 types of classes: those that are inherited from QObject and those that are not.
- QObject derived classes are allocated on the heap using new
- The parent takes ownership of the object so no explicit delete needed

```
QPushButton* button = new QPushButton(”Ok”, parent);
```

- Objects not inheriting QObject are allocated on the stack

```
QColor color(100,12,100);
QString greeting(”Hello Qt”)
```

- Exceptions
  - QApplication and QFile are allocated on the stack

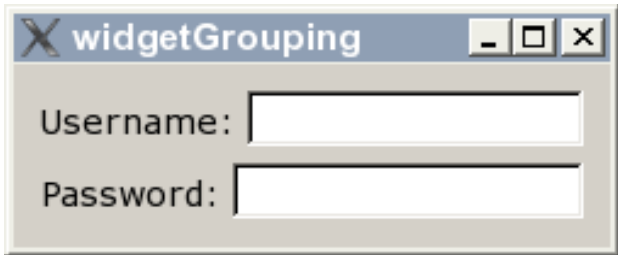
Parent-Child Relationship

- When a QObject is created, it is given its parent as an argument

  QObject (QObject * *parent* = 0)

- The child informs its parent about its existence, upon which the parent adds it to its own list of children
- For QWidgets: If *parent* is 0, the new widget becomes a window

```
int main(int argc, char *argv[])
{
   QApplication app( argc, argv );
   QWidget top;
   QLabel *nameLabel = new QLabel("Username:", &top);
   QLineEdit *nameEdit = new QLineEdit(&top);
   QLabel *passLabel = new QLabel("Password:", &top);
   QLineEdit *passEdit = new QLineEdit(&top);
   QVBoxLayout *vlay = new QVBoxLayout(&top);
   QHBoxLayout *nameLayout = new QHBoxLayout;
   QHBoxLayout *passLayout = new QHBoxLayout;
   vlay->addLayout(nameLayout);
   vlay->addLayout(passLayout);
   nameLayout->addWidget(nameLabel);
   nameLayout->addWidget(nameEdit);
   passLayout->addWidget(passLabel);
   passLayout->addWidget(passEdit);
   top.show();
   app.exec();
}
```

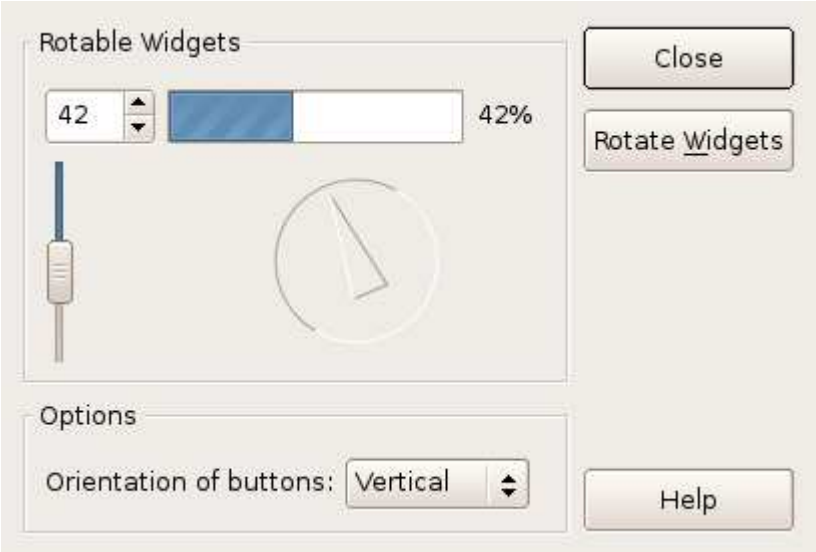


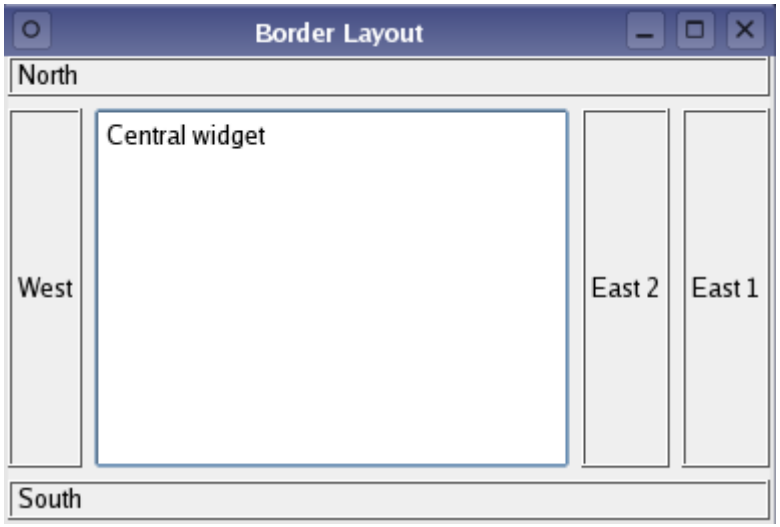
Creating GUIs

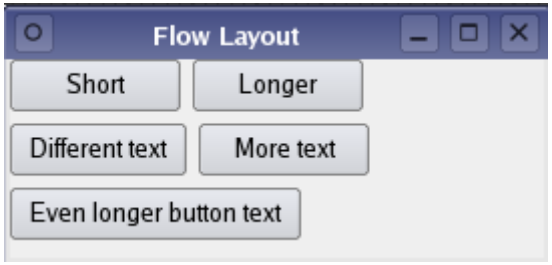
Building a GUI with Qt

- **Widgets** are UI components like buttons, message boxes, application windows etc.
- A GUI is built by using **widgets**, which can be arranged by **layout managers**
- GUI can be built by hand-coding or using Qt Designer tool
- We'll do it with the first option in this phase ☺

Building GUIs





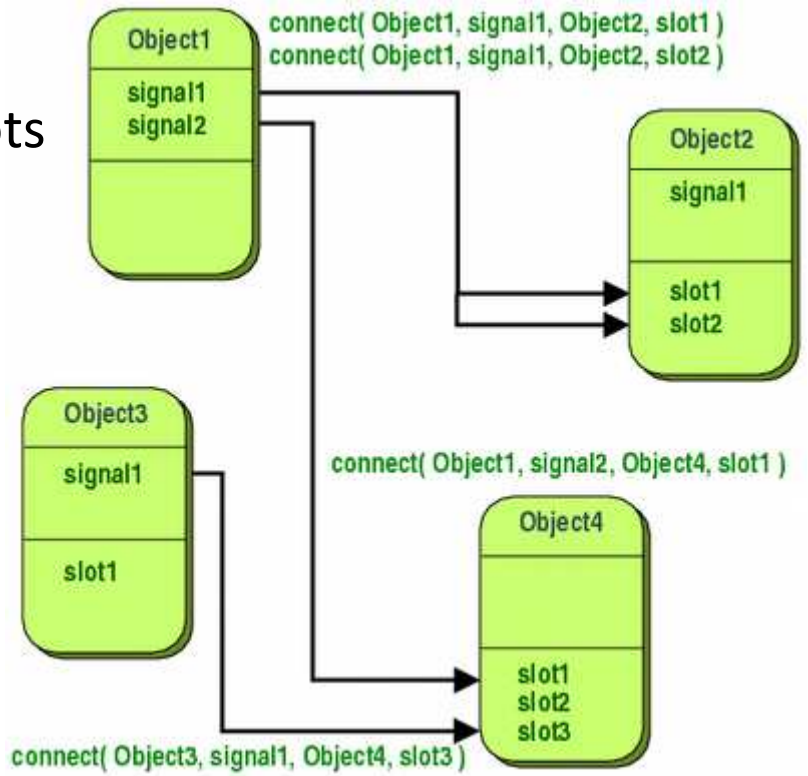


Signal-Slot mechanism

It’s all about inter-object communications…

Signals and Slots

- Inter-object communication mechanism between objects **inside your application**
- Type-safe callbacks
  - More secure than callbacks, more flexible than virtual methods
- Many-to-many relationship
- Loose coupling between signals and slots
- Implemented in QObject
- Observer pattern



Signals

- A *signal* is a way to inform a possible observer that something of interest has happened inside the observed class
  - A QPushButton is **clicked**
  - An asynchronous service handler is **finished**
  - Value of QSlider is **changed**
- Signals are member functions that are *automatically implemented in the meta-object*
  - Only the function declaration is provided by the developer
- Signal is sent, or *emitted*, using the keyword emit
  - `emit clicked();`
  - `emit someSignal(7, “Hello”);`

Slots

- A *slot* is a function that is to be executed when a signal has been *emitted*.
  - (When QPushButton is pressed), **close** QDialog
  - (When service is ready), **ask for the value and store it**
  - (When QSLider value is changed), **show a new value** in QLCDNumber

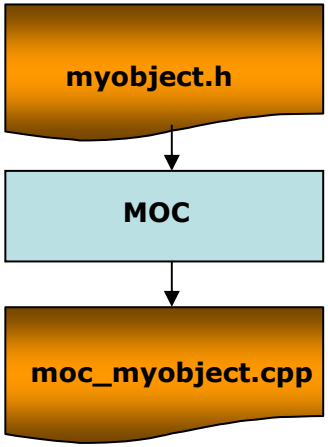
- A Slot function is a normal member function implemented by the developer

The connections in the code

```
class MyWidget : public QWidget {
Q_OBJECT
…
public slots:
   void showNormal();
   void showMaximized();

private:
   MyAboutDialog* aboutDialog;
   QPushButton* but1;
   …
};

void MyWidget::MyWidget(…)
{
   …
   connect( but1, SIGNAL( clicked() ), this, SLOT( showNormal() ));
   connect( but2, SIGNAL( clicked() ), aboutDialog, SLOT( aboutQt() ));
   connect( but2, SIGNAL( clicked() ), this, SLOT( showMaximized() ));
   connect( but3, SIGNAL( clicked() ), qApp, SLOT( quit() ));
}
```

Signal and Slot Implementation

```
class MyComponent : public QObject
{
  Q_OBJECT // Meta-object file needed

signals:
   // Implementation in the meta-object
   void mySignal( int numberToPass );

public slots:
   // Slots are implemented as normal member functions
   void textEdited( QString s );
}
```



Event Handling

Event types

- Signals are sent **between the objects inside your application**. Events typically come from **outside your application**

1. Window system (GUI)
   - Mouse movements, clicks, key presses
2. System
   - Timer events
   - Socket notifier
3. Application itself
   - Application specific events (you can send events across your application)

- Some QEvent subclasses: QResizeEvent, QPaintEvent, QMouseEvent, QKeyEvent, QTimerEvent

Timer with events (QObject)

```
class MyObject : public QObject
{
   Q_OBJECT
public:
   MyObject(QObject *parent = 0);
protected:
   void timerEvent(QTimerEvent *event);
};

MyObject::MyObject(QObject *parent) : QObject(parent) {
   startTimer(50); // 50-millisecond timer
   startTimer(1000); // 1-second timer
   startTimer(60000); // 1-minute timer
}

void MyObject::timerEvent(QTimerEvent *event)
{
   qDebug() << "Timer ID:" << event->timerId();
}
```

Timer with signal-slot (QTimer)

```
QTimer* timer = new QTimer( this );
connect(timer, SIGNAL(timeout()), this, SLOT(handleTimout()));
timer->start( 50 );


void MyObject::handleTimeout()
{
   qDebug() << "Timer event…”;
}
```

Do not confuse events with signals/slots

- Signals are just emitted without knowing the receiver. Events go to a specic widget
- Signals can have any number of receivers. Events go to exactly one receiver
- Events pass through event filters, signals don’t

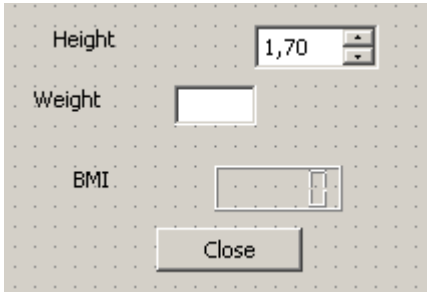
Do not confuse events with signals/slots

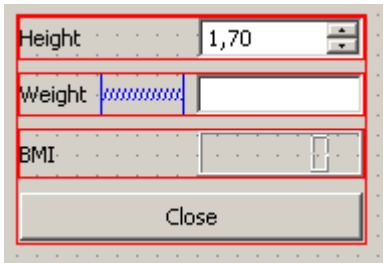
- Signals occur between two objects **inside the application**
- Events that come from **outside** of the application
  - User events like mouse or key press
  - Low level events from Qt modules like networking etc.
- Events that come from **inside** of the application
  - Child object added, removed etc.
  - User defined events
- Signals are used when **using** widgets, events are used when **implementing** widgets

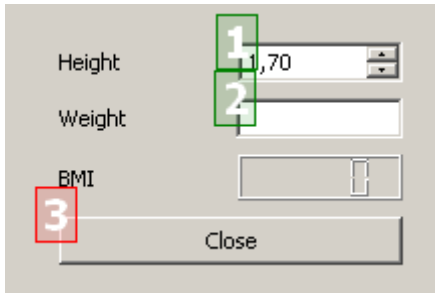
Creating GUIs with Qt Designer

Qt Designer

- Create widgets
- Specify layout
- Set the tab order



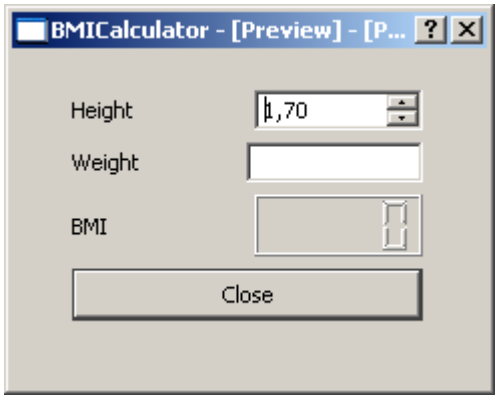




How to Use the Generated Code?

- Direct approach
  - Simple, but custom slots cannot be used
  - How to validate the input data, for example?
- Single inheritance
  - Custom slots can be used
  - Derive from the corresponding `QWidget` class
  - UI component is a member variable of your class
- Multiple inheritance
  - You derive a new class both from the `QWidget` (sub)class and your UI component

Direct Approach



- In the .pro file, add
  - **FORMS += BMICalculator.ui**

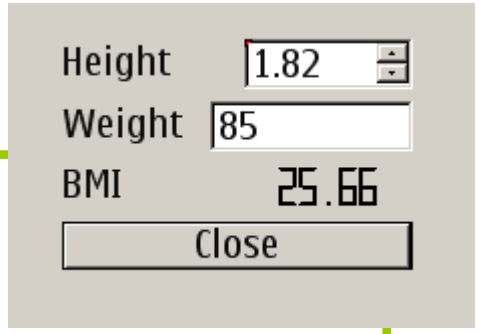
```
#include "ui_BMICalculator.h"
int main(int argc, char *argv[])
{
   QApplication app(argc, argv);
   Ui::BMICalculatorClass ui;
   QDialog dialog;
   ui.setupUi(&dialog); // In ui_BMICalculator.h
   dialog.show();
   return app.exec();
}
```

Single Inheritance

- Add ui_BMICalculator.h into HEADERS in the .pro file and include “ui_BMICalculator.h”
- Useful, if custom signals and slots needed

```
#include ”ui_BMICalculator.h”
class BMICalculator : public QDialog
{
    Q_OBJECT
public:
    BMICalculator(QWidget *parent = 0);
    ~BMICalculator();

private slots:
   // Custom slot function for calculating the BMI
    void calculateBMI();

private:
   Ui::BMICalculatorClass ui;
};
```

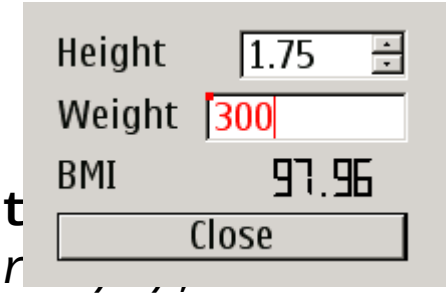
Single Inheritance




```
BMICalculator::BMICalculator(QWidget *parent)
    : QDialog(parent)
{
   ui.setupUi(this);
   connect( ui.weightLineEdit, SIGNAL(textEdited(QString)),
            this, SLOT(calculateBMI()) );
   connect( ui.heightSpinBox, SIGNAL(valueChanged(QString)),
            this, SLOT(calculateBMI()) );
}
...
void BMICalculator::calculateBMI()
{
   QString tmp = ui.weightLineEdit->text();
   int weight = tmp.toInt();
   if( ui.heightSpinBox->value() != 0 ) {
    // Calculate BMI and show it if height not zero
    double BMI = weight / (ui.heightSpinBox->value() *
   ui.heightSpinBox->value() );
    ui.lcdNumber->display(BMI);
   }
}
```

Auto-Connected Slot Functions

```
void BMICalculator::on_weightLineEdit_textEdited(const
   QString& s) {
    int weight = s.toInt();
    // "Accepted" values between 30 and 200
    if( weight < 30 || weight > 200 ) {
       QPalette pal = ui.weightLineEdit->palett
       pal.setBrush(QPalette::Text,QBrush(Qt::r
       ui.weightLineEdit->setPalette(pal);
    }
    else {
       // Use parent widgets palette (default palette)
       ui.weightLineEdit->setPalette(this->palette());
    }
}
```

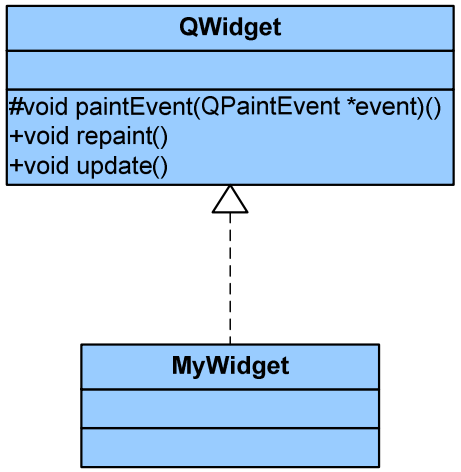


Basic Graphics

Painting and drawing

2D Graphics

- Overwrite paintEvent() if you want to paint on your widget
  - paintEvent() is a protected function which is called by the framework
  - repaint() causes paintEvent() to be called immediately. Could be used in animations and can cause recursion
  - update() causes paintEvent() to be called next time the event loop is entered. This is preferred as it permits Qt to optimize for speed and minimize flicker

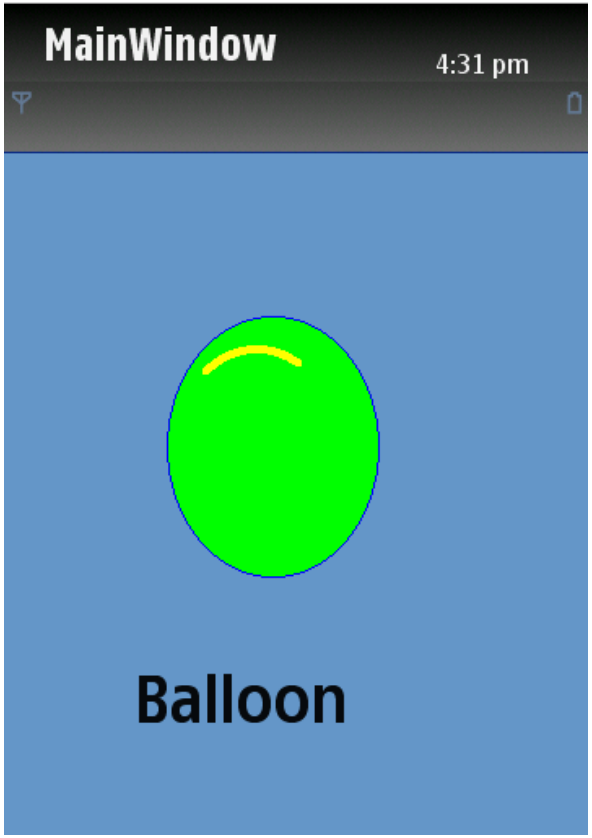


QPainter

- 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

2D Graphics

```
void MyPainterTest::paintEvent(QPaintEvent *event)
{
    setPalette(QPalette(QColor(100, 150, 200)));
    setAutoFillBackground(true);
    QPainter painter( this );
    painter.setPen(QPen(Qt::blue));
    painter.setBrush(QBrush(Qt::green, Qt::SolidPattern));
    painter.drawEllipse(100,100,130,160);
    painter.setPen(QPen(Qt::yellow, 5, Qt::SolidLine, Qt::RoundCap));
    painter.drawArc(100,120,110,150, 1000, 1000);
    painter.setPen(Qt::black);

    QFont myFont("Helvetica");
    myFont.setPointSize(12);
    myFont.setBold(true);
    painter.setFont(myFont);
    painter.drawText(QPoint(80,350), "Balloon");
}
```



Event Processing and Painting

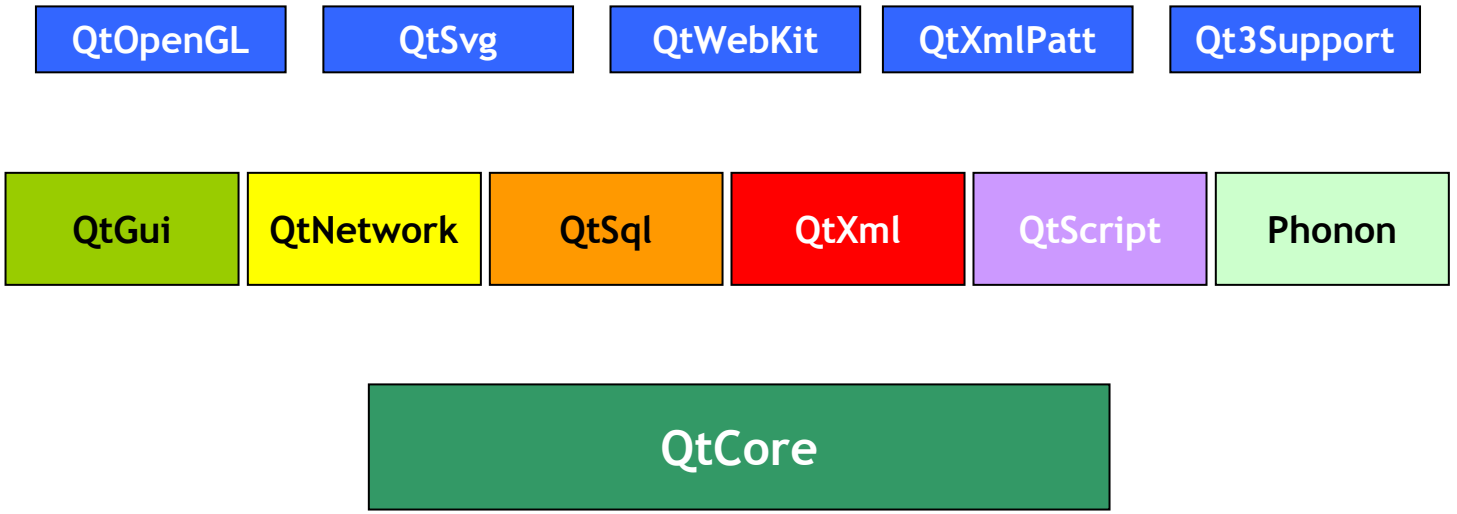
- Painting takes place when QWidget::paintEvent() function call is received by the application
- Two ways to request repaint (QWidget member functions)
  - repaint() – causes immediate call to paintEvent()
    - Avoid
    - Use only in animations, where immediate repaint() needed
  - update() – causes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

Time for a Review Exercise

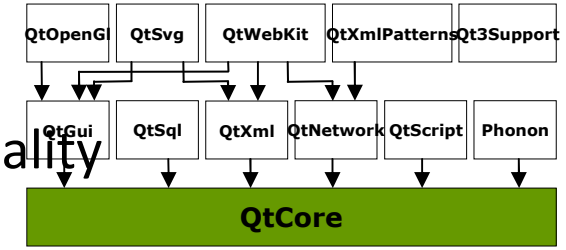
A quick Qt Modules Walkthrough

Qt as a whole package

- Cross-platform Qt consist of ~700 classes
- Build tools (qmake, moc, uic)
- Development tools (Qt Designer, Qt Assistant, QtLinguist)

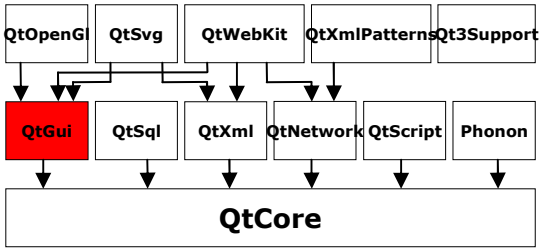


Qt - Modules (Core)



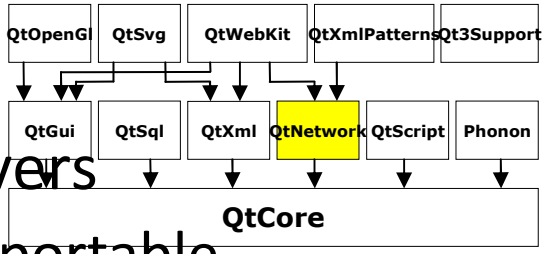
- Classes and methods for basic non-GUI functionality
  - Basic Type (Char, Date, Time, String)
  - File system access, Date and time handling
  - String handling, List and array handling
  - Threads and processes, Shared resources
  - Libraries and Plugins, Timers
- Classes and methods for basic graphical elements
  - Point, Rectangle
- Heart of the Qt object model, QObject
  - More information about Qt object model later on
- QtCore do not depend on underlying window system

Qt - Modules (GUI)



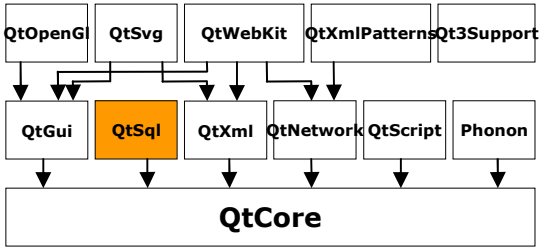
- Extends QtCore with GUI functionality
  - UI application base class
  - Set of widgets and dialogs
  - Widget layout management
  - Classes for UI Events
  - Input method integration and Paint Engine Abstraction
- Platform Abstraction
  - Platform specific clipboard integration
  - Platform specific look and feel (Style in Qt)
  - Integrates to platform specific window system for event handling and drawing

Qt - Modules (Network)



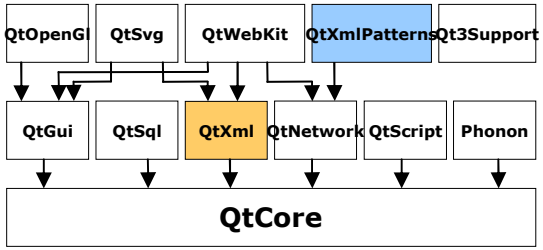
- Classes for writing TCP/IP clients and servers
  - Makes network programming easier and portable
- Support for
  - Lower level protocols (TCP and UDP)
  - Higher level application protocols (FTP and HTTP)
    - Only client support available
    - Seamless integration with progress widgets
    - Seamless integration with any IO device for up- and downloading
  - Host Name resolving services and URL parsing
  - SSL support added in 4.3

Qt - Modules (Sql)



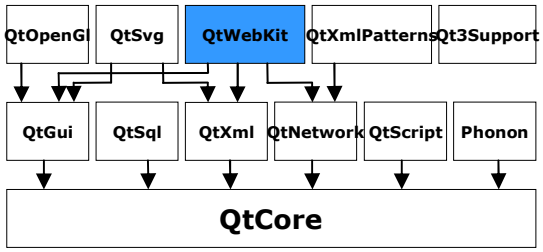
- Provides database integration to Qt applications
- Divided to three layers
  - UI Layer
    - Provides classes to link the data from database to data-aware widgets
    - Designed to work with Qt's model/view framework
  - API layer
    - Provides Qt's abstracted database-independent SQL API
  - Driver layer
    - Provides low-level bridge between the specific databases and the SQL API layer
    - Uses driver plugins to communicate with the different databases
    - All database-specific code is contained within these drivers
    - Several drivers are supplied with Qt

Qt XML Classes

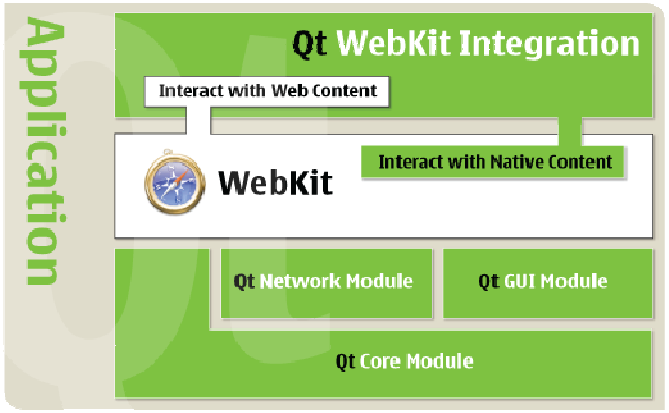


- Core Module
  - Simple XML stream reader and writer
- XML Module
  - A well-formed XML parser using the SAX2 (Simple API for XML) interface
  - Implementation of the DOM Level 2 (Document Object Model)
- XmlPatterns module
  - An implementation of the XQuery standard
  - Enable users to query XML files similar to SQL
  - Semantics for value assignment, filtering, and simple operations
  - Fully controllable output formatting

Qt WebKit Integration

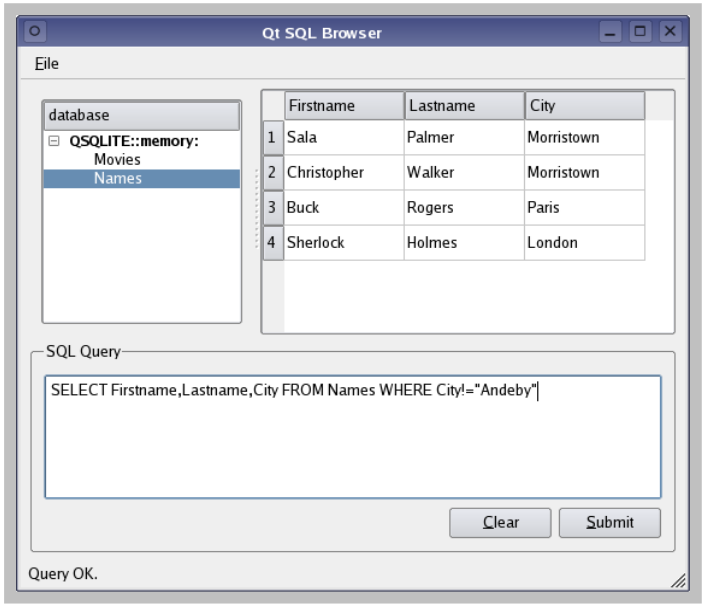


- An open source HTML rendering component integrated with Qt
- Web standards compliant
  - support for HTML, XHTML, XML, stylesheets, JavaScript, HTML editing, HTML canvas, AJAX, XSLT, XPath, some SVG.
- Deployable wherever Qt is: cross-platform/cross-version/cross-device
- Interact with Web environment, expose native objects

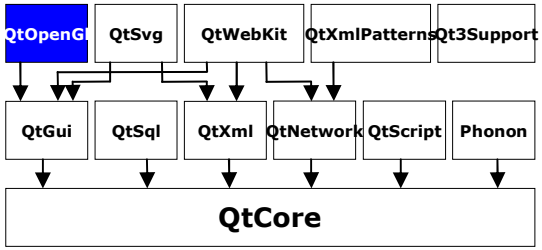


Qt Database Classes

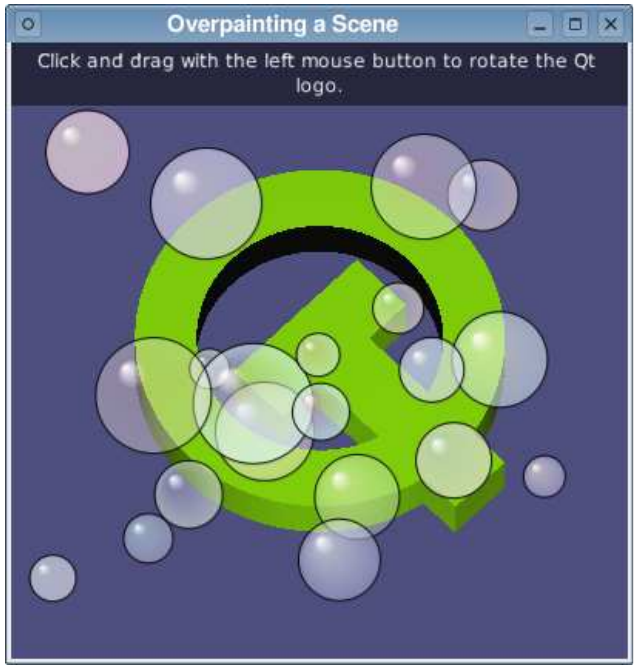
- Provide platform and database-independent access functionality
- Driver Layer
  - Low-level bridge between specific databases and the SQL API layer
- SQL API Layer
  - Provide access to databases
- User Interface Layer
  - Link data from a database to data-aware widgets
- Supports most major database drivers
  - DB2, IBASE, MySQL, OCI, ODBC, PSQL, SQLITE, TDS



Qt OpenGL Classes



- Allows you to build your user interface in Qt, display and manipulate 3D model in OpenGL®
- Integrates OpenGL canvas with Qt
- Provides frame buffer and pixel buffer abstraction
- Supports accelerating 2D painting with OpenGL
- Mix 2D painting and 3D scenes

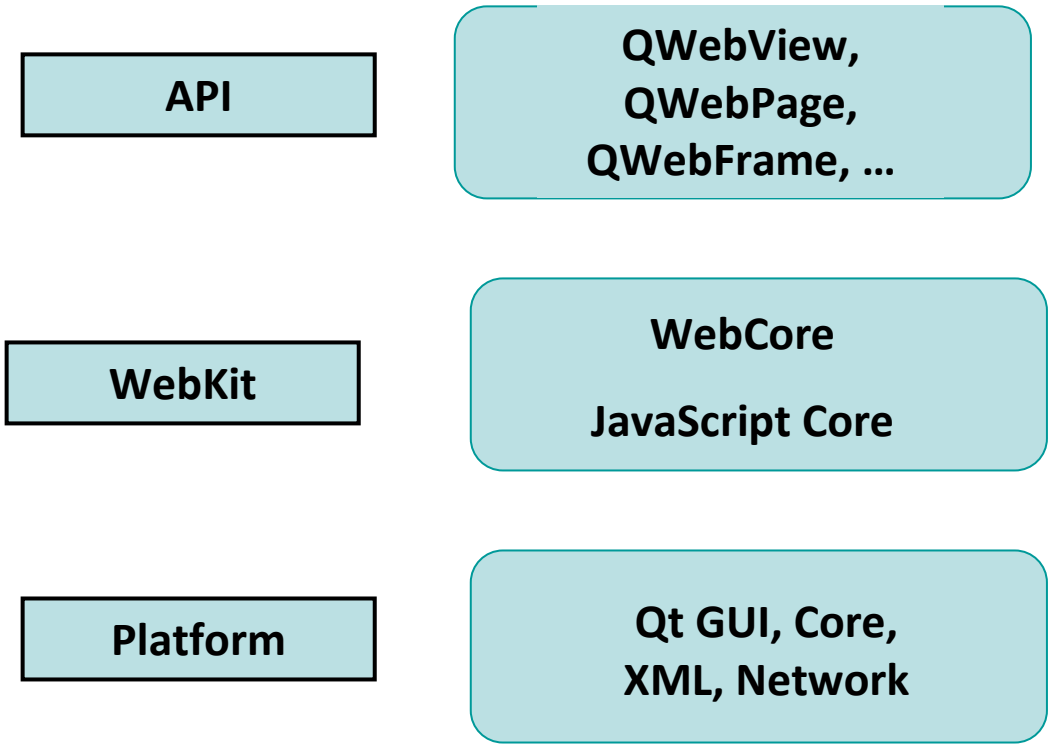


WebKit

WebKit

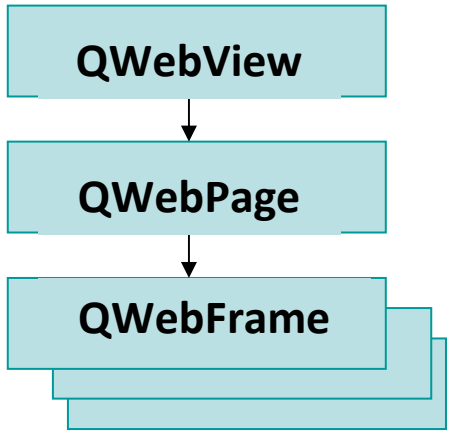
- Provides a Web browser engine
  - Easy to embed WWW content into an application
  - Content may be enhanced with native controls
- QtWebKit module provides facilities for rendering of
  - HyperText Markup Language (HTML)
  - Extensible HyperText Markup Language (XHTML)
  - Scalable Vector Graphics (SVG) documents
- WebKit is based on the Open Source WebKit engine
  - www.webkit.org

High-Level Architecture



QWebView and QWebPage Classes

- `QWebView` inherited from `QWidget`
  - Renders the contents of a `QWebPage`
  - *Easy to embed wherever a widget could be used*




```
QWebView *view = new QWebView( parent );
view->load(QUrl("http://www.nokia.com"));
view->show();
```

- `QWebPage` provides access to the document structure in a page
  - Frames
  - Navigation history
  - Undo/redo stack for editable content
- Each `QWebPage` has one `QWebFrame` object as its main frame
- HTML documents can be nested using frames

QWebView

- Allows viewing and editing of Web documents
  - `view.back(); view.forward(); view.reload();`
- Easy to embed Web engine anywhere
- Three essential signals
  - `loadStarted()`
  - `loadProgress()` – emitted whenever web page elements loaded
  - `loadFinished()` – view loaded completely
- Settings can be specified with `QWebSettings`
  - Font, Java enabled, plugins enabled, auto load images, etc.
  - Use `QWebSettings::globalSettings()` to modify global settings, and
  - `QWebPage::settings()` to modify page-specific settings and override global settings

File Handling in Brief

I/O

- Reading from or writing to files or other devices
- QIODevice as an abstraction
- Concrete classes
  - QFile
  - QTemporaryFile
  - QBuffer
  - QProcess
  - QTcpSocket
  - QUdpSocket
  - QSslSocket

QFile

- Supports reading and writing text files, binary files, and resource files
- Often QTextStream or QDataStream used to serialize and de-serialize the data

```
QFile file("in.txt");
if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
    return;
QTextStream in(&file);
while (!in.atEnd()) {
    QString line = in.readLine();
    process_line(line);
}
```

File Access Performance

- Read as little as possible
  - Read less data and read your data less frequently
  - The latter can be implemented by keeping data that you expect to use again in RAM (not always a good idea)
- One way to keep down the file sizes is to use binary file formats, as they are more compact by nature
- Qt's QDataStream provides means to do this in an easy and still platform independent way
- Design you files' internal structure with size and performance in mind
  - Make sure to store things in a way so that you avoid large unused areas in your files

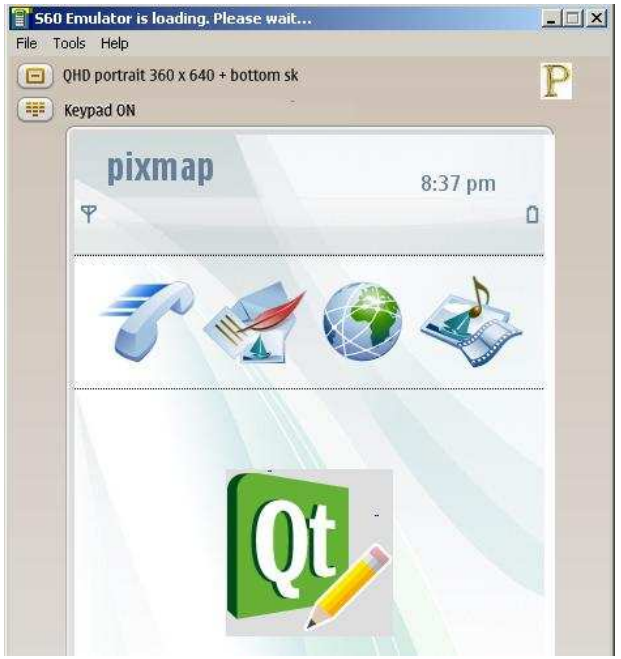
QFile snippet

```
QFile myDataFile(”myGame.dat”);
myDatafile.open(QIODevice::WriteOnly);
QDataStream outStream(&myDataFile);
out << quint32(10000) ;

QFile myTextFile(”myText.txt”);
myTextFile.open(QIODevice::WriteOnly);
QTextStream outStream(&myTextFile);
outStream << ”Hello” << endl;
outStream << ”I’m writing this to you because… ” << endl;
```

Splash Screens

Splash Screens

```
#include <QApplication>
#include <QPixmap>
#include <QSplashScreen>
#include <QWidget>
#include <QMainWindow>
int main(int argc, char *argv[])
{
QApplication app(argc, argv);
QPixmap pixmap("c://designer.png");
QSplashScreen splash(pixmap);
splash.show();
splash.showMessage("Wait...");
qApp->processEvents();
QMainWindow window;
window.setStyleSheet("* { background-color:rgb(199,147,88); padding: 7px}}");

window.show();
splash.finish(&window);
return app.exec();

}
```



Splash Screens

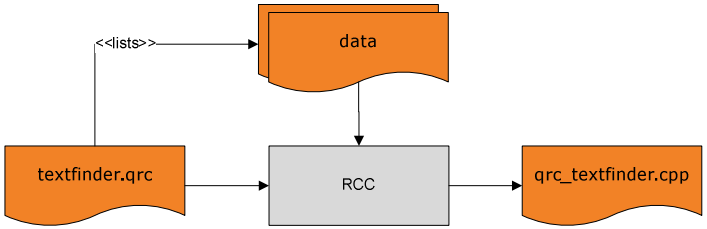
- Let's take a look at the examples:

  http://doc.trolltech.com/qq/qq04-splashscreen.html

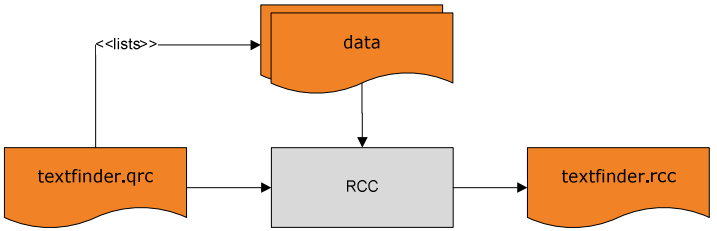
  http://wiki.forum.nokia.com/index.php/How_to_display_a_Splash_Screen_in_Qt_for_Symbian

Qt - Build Tools (RCC)

- RCC (Resource Compiler)
  - Resources can be compiled into binary or as a external binary resource files
  - Information about resources provided in XML format (.qrc)
- Compiled-in resources
  - RCC generates C++ source file containing data specified in a Qt resources specification (.qrc) file as a static C++ arrays
  - Provides platform-independent mechanism for storing binary files in the application's executable



- External binary resources
  - Compiled manually with rcc switch
  - Shall be registered in Qt application
    - by using QResource API

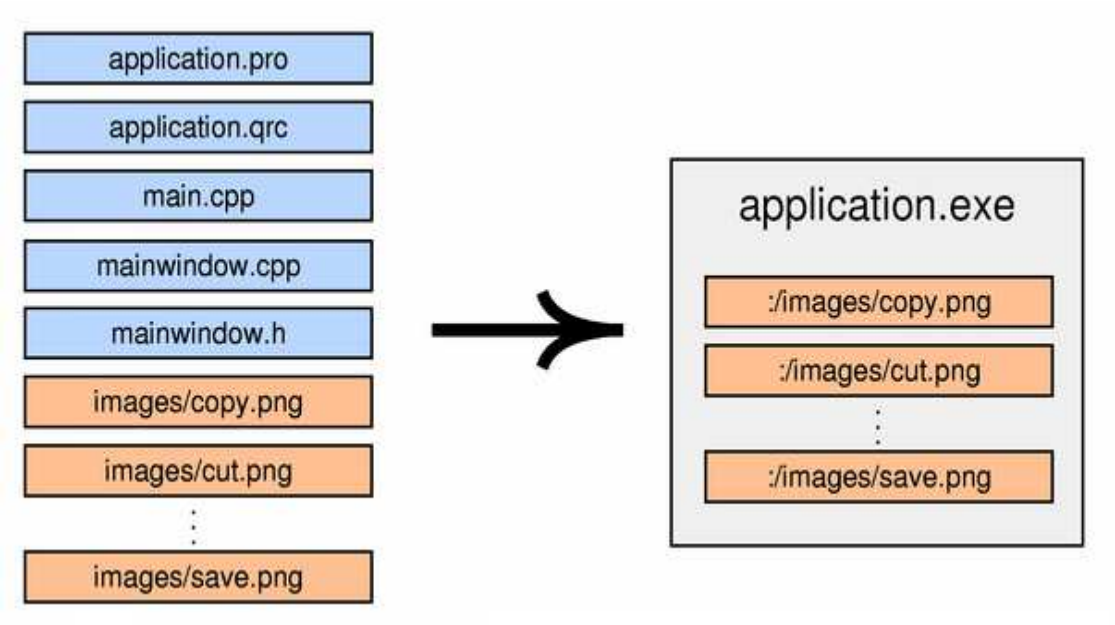


Resource Collection Files

- The resources associated with an application are specified in a **resource collection file,** a **.qrc** file
  - XML
  - Lists files on the disk
  - Optionally assigns them a resource name that the application must use to access the resource
  - A prefix can be used to logically group resources in the same folder

```
<!DOCTYPE RCC><RCC version="1.0">
<qresource>
   <file>images/star.png</file>
   <file alias="bg.jpg">images/swirl.jpg</file>
   <file>images/cherry.png</file>
</qresource>
</RCC>
```

Resource compiler



Using Resources in Application

- The resource is usable with preceeding ”:/” wherever a filename could be given:

```
ui.graphicsView->setBackgroundBrush( QPixmap(":/bg.jpg")
   );
m_pixmaps.append( QPixmap(":/images/cherry.png") );
```

Resource Paths and Localization

- Resources are accessible under the same name as they have in a source tree with a :/ prefix
- Path prefix can be changed using qresource tag's prefix attribute

```
<qresource prefix="/myresources">
     <file alias="cut-img.png">images/cut.png</file>
</qresource>
```

- Localization is as easy using qresource tag's lang attribute

```
<qresource>
     <file>cut.jpg</file>
</qresource>
<qresource lang="fr">
     <file alias="cut.jpg">cut_fr.jpg</file>
</qresource>
```

External Binary Resources

- Create a resource data file (extension .rcc)
  - Must use a command line option
- Access the resource in your code using QResource

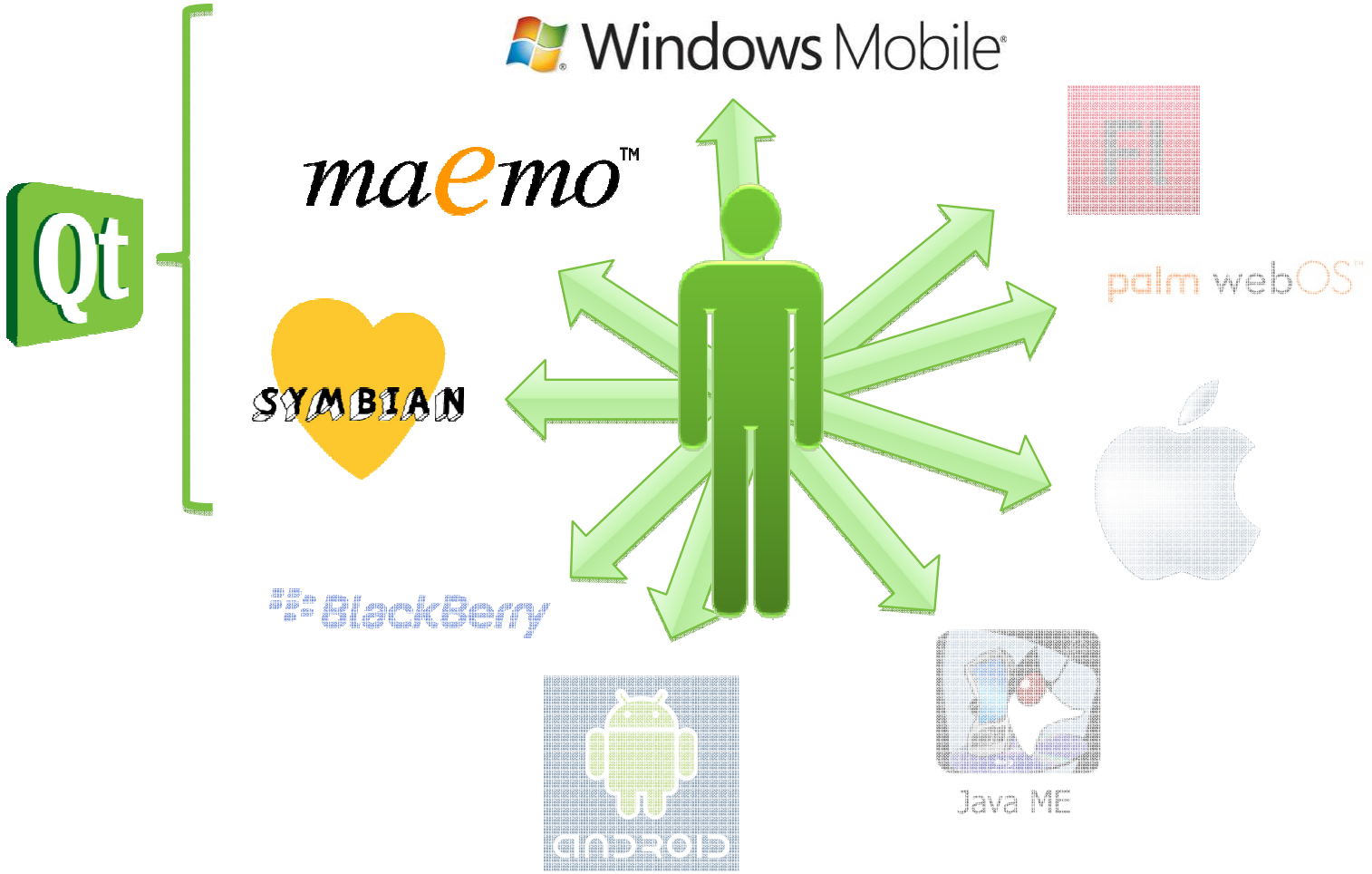
```
QResource::registerResource("/path/to/myresource.
  rcc");
```

Compiled-in Resources

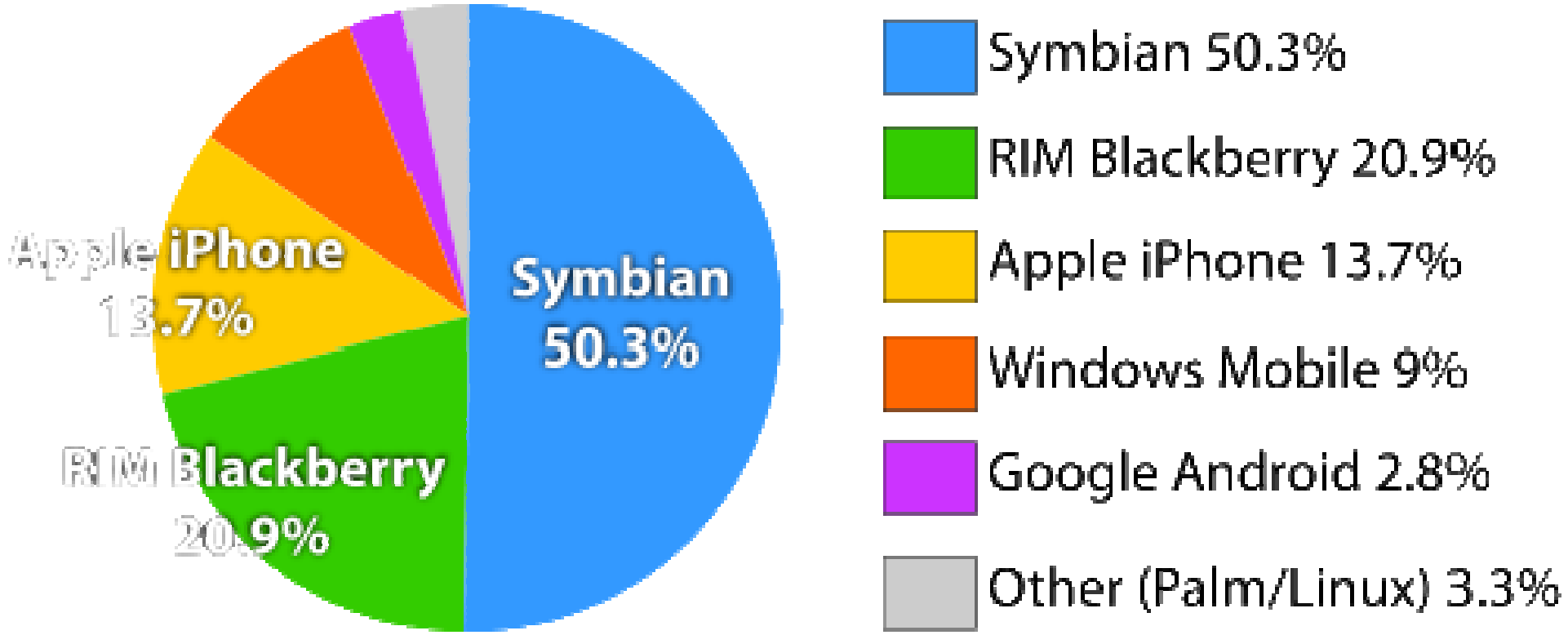
- Add resource collection file name into the project file
  - RESOURCES += myApp.qrc
- The qmake tool creates rules to generate qrc_application.cpp file
  - Contains all the data for the resources as static C++ arrays of compressed binary data
- Currently, Qt always stores the data directly into executable

Mobile Platforms overview and Qt for Nokia Mobile Platforms

Mobile Development 2010



Global Smartphone Sales, Q2 2009



Source: Gartner

Main players and SW development

- Apple: ObjectiveC on Os X
- Google: Java on Android
- Nokia: Qt on Symbian and Maemo
- Windows: C# and VB.NET
- Blackberry: Java

iPhone development

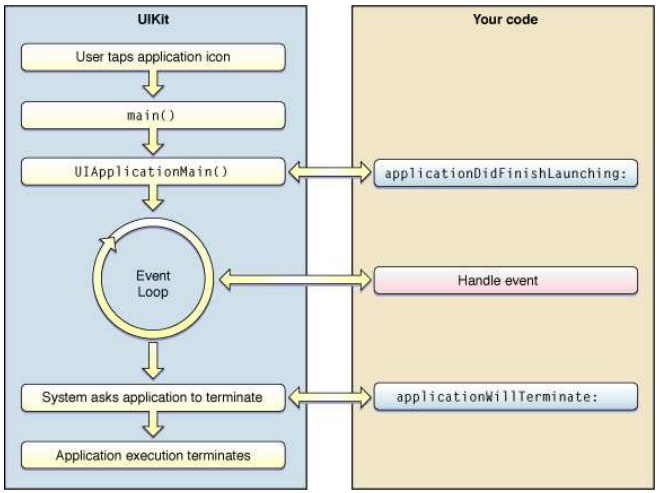
- MacOS needed
- Native development with **ObjectiveC**
- *“No. The iPhone will not support Java applications of any kind. Steve Jobs has been quoted as saying "Java's not worth building in. Nobody uses Java anymore. It's this big heavyweight ball and chain."*

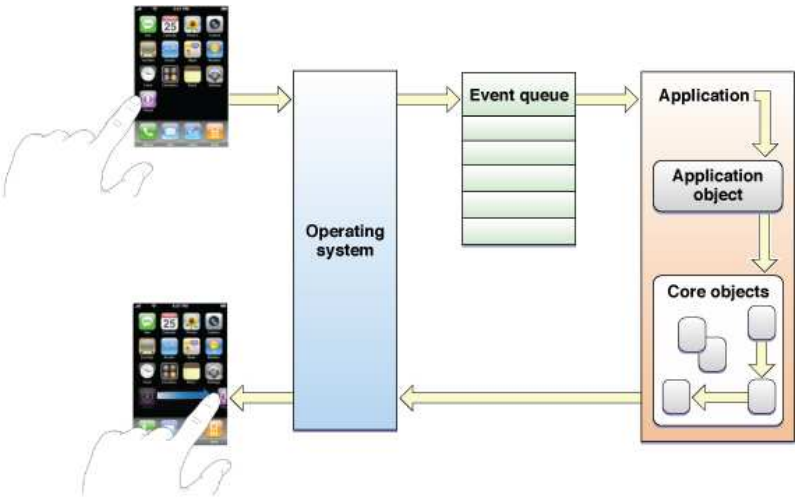
  - Steve Jobs, CEO 2007
- *“We'll put Java on the iPhone”*

  – Sun Microsystems 07/2008

iPhone development

```
#import <UIKit/UIKit.h>

int main(int argc, char *argv[])
{
NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
int retVal = UIApplicationMain(argc, argv, nil, nil);
[pool release];
return retVal;
}
```





Android

- Some Devices:
  - Motorola Droid
  - Samsung I7400
  - Nook
  - T-Mobile Fender

Google Android

- Android is a software stack for mobile devices that includes an operating system, middleware and key applications

Android Development

- By default, every application runs in its own Linux process. Android starts the process when any of the application's code needs to be executed, and shuts down the process when it's no longer needed and system resources are required by other applications.
- Each process has its own Java virtual machine (VM), so application code runs in isolation from the code of all other applications.

```
package com.example.helloandroid;
import android.app.Activity;
import android.os.Bundle;
public class HelloAndroid extends Activity {
/** Called when the activity is first created. */
@Override
     public void onCreate(Bundle savedInstanceState){
          super.onCreate(savedInstanceState);
          setContentView(R.layout.main);
     }
}
```

Qt on Nokia Platforms

- Examples: N900 (Linux), N97 (Symbian)
- Cross-mobile platform Applications with Mobility APIs

Symbian OS

- Open mobile operating system
- **Ownership:**
  - **Symbian Foundation**
  - **Previously:** Symbian Ltd. – company bought by Nokia and OS transferred to the Foundation
- **Used by major handset manufacturers:**
  - Nokia, Samsung, Sony Ericsson
  - Smartphone Sales: 51% Symbian (Q2 2009 – Gartner*)

* http://www.gartner.com/it/page.jsp?id=1126812

Symbian Future

- **Symbian^4: replaces S60 with Qt-based UI**
- **Components:**
  - **Orbit:** extension library for Qt,
    50+ UI elements tailored for mobile
  - **Direct UI:** new app framework based on Orbit
- **Availability:** end of 2010

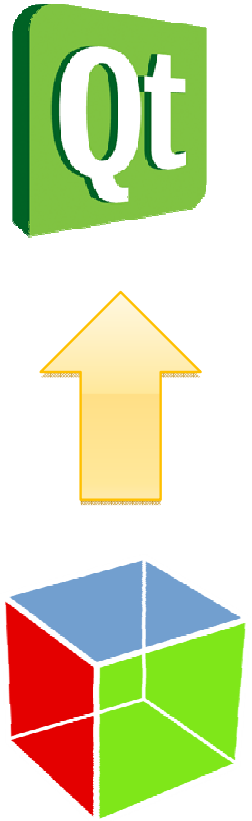
Maemo

maemo™

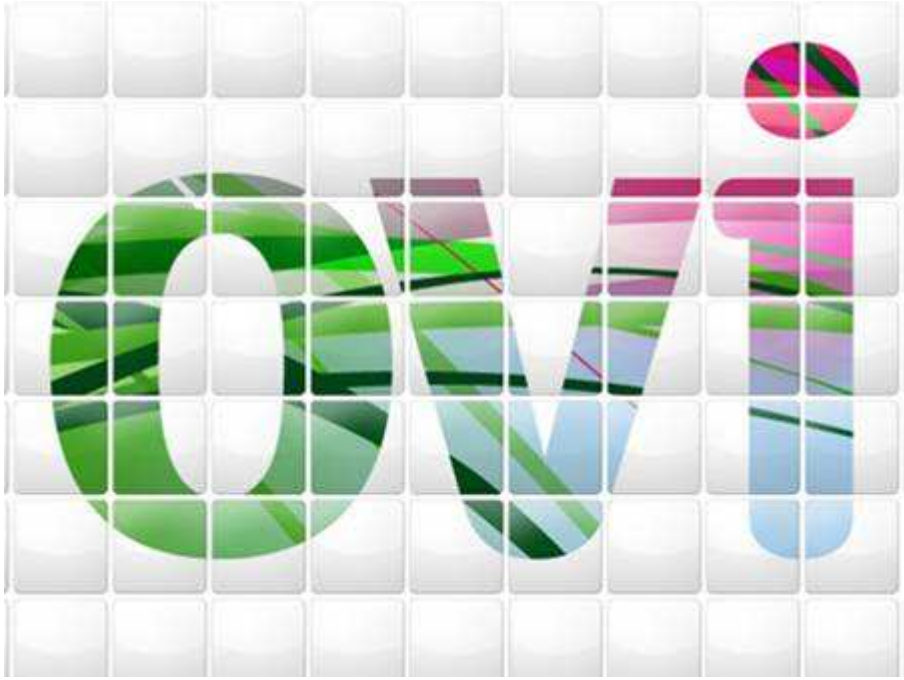
- Linux-based mobile operating system
  - Developed by Nokia, based on Debian Linux
- **N900:** First Maemo-device with telephony
  - Previously: N770, N800, N810 Internet Tablets

The future of Maemo

- **Maemo 5**
  - GTK+ based UI
  - Final Qt support: H1 2010
- **Maemo 6**
  - **GTK+ replaced with Qt**
  - Multi-touch, gestures support
  - Community powered GTK support



Distribution channels…



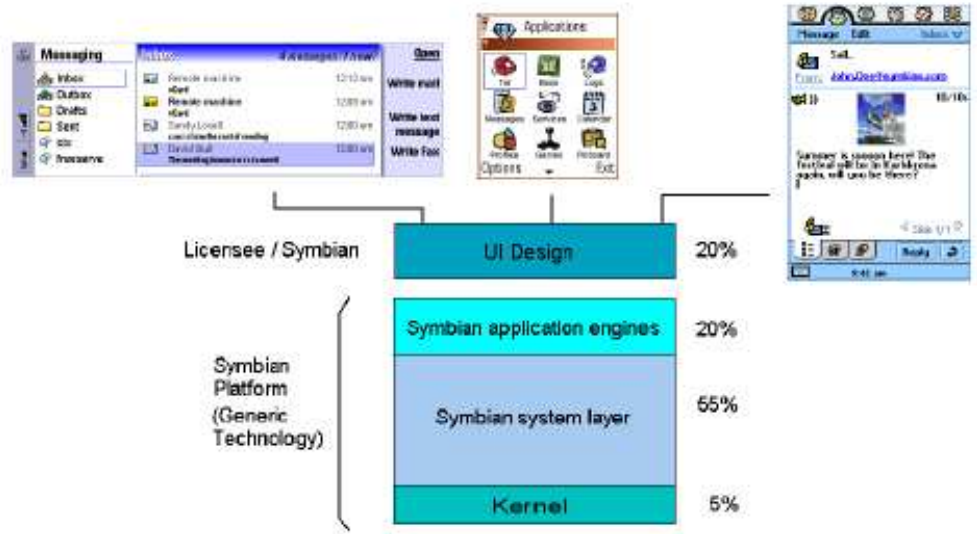
ANDROID

Qt for Symbian

Symbian OS Overview

- Runs on battery powered devices
  - has low power consumption
- Designed for devices with limited memory
- Open Operating System
  - 3rd party developers can write applications
- Reliable and stable
  - Applications can run for years without being closed or losing user data
- Object Orientated from the “ground up”
  - Provides a C++ API
- Component based
  - Can run on multiple platforms

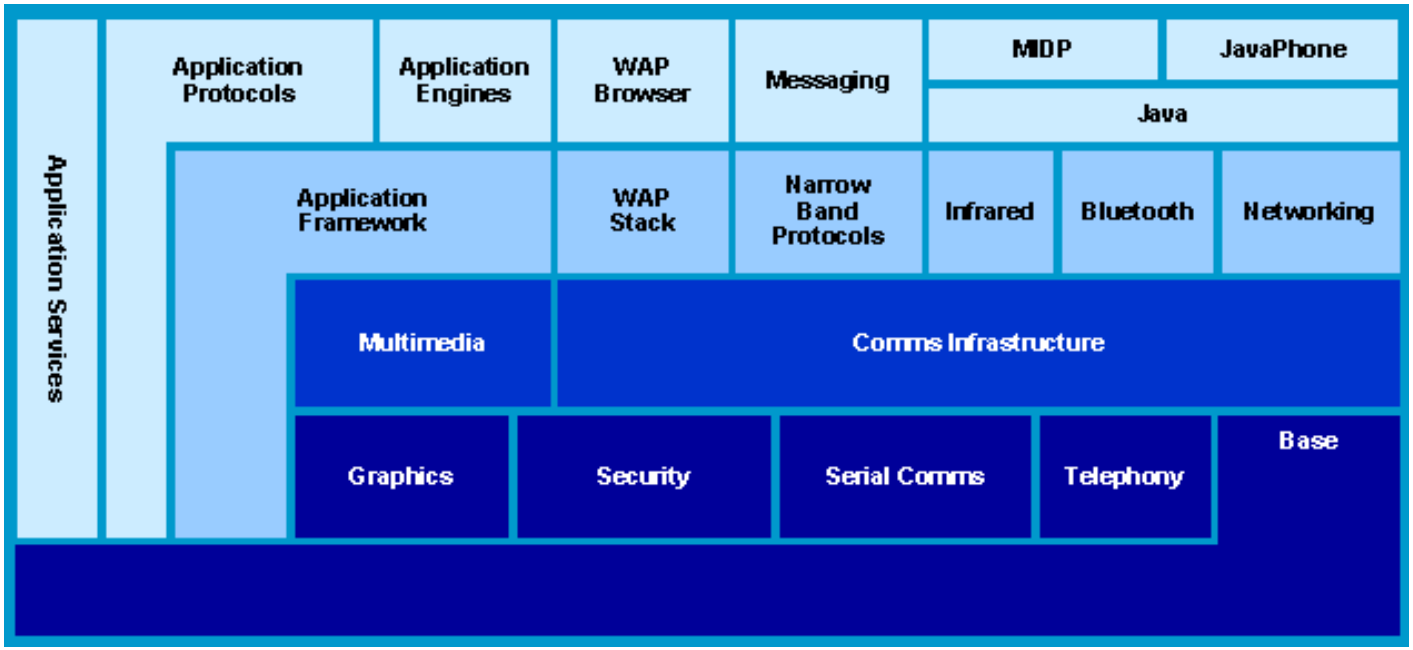
Main OS Layers



- Symbian application engines (20%) - these allow access to built in functionality such as contacts and calendar information. Engines concern themselves only with the data and not how it is presented.
- Symbian System layer (55%) - this contains the bulk of the OS APIs and provides functionality from string handling to event scheduling within an application.
- Kernel (5%) - not directly accessible from user programs.
- The remaining 20% of the Symbian OS is concerned with the UI design.

Source: symbian.com

Symbian OS v9.3 Subsystems



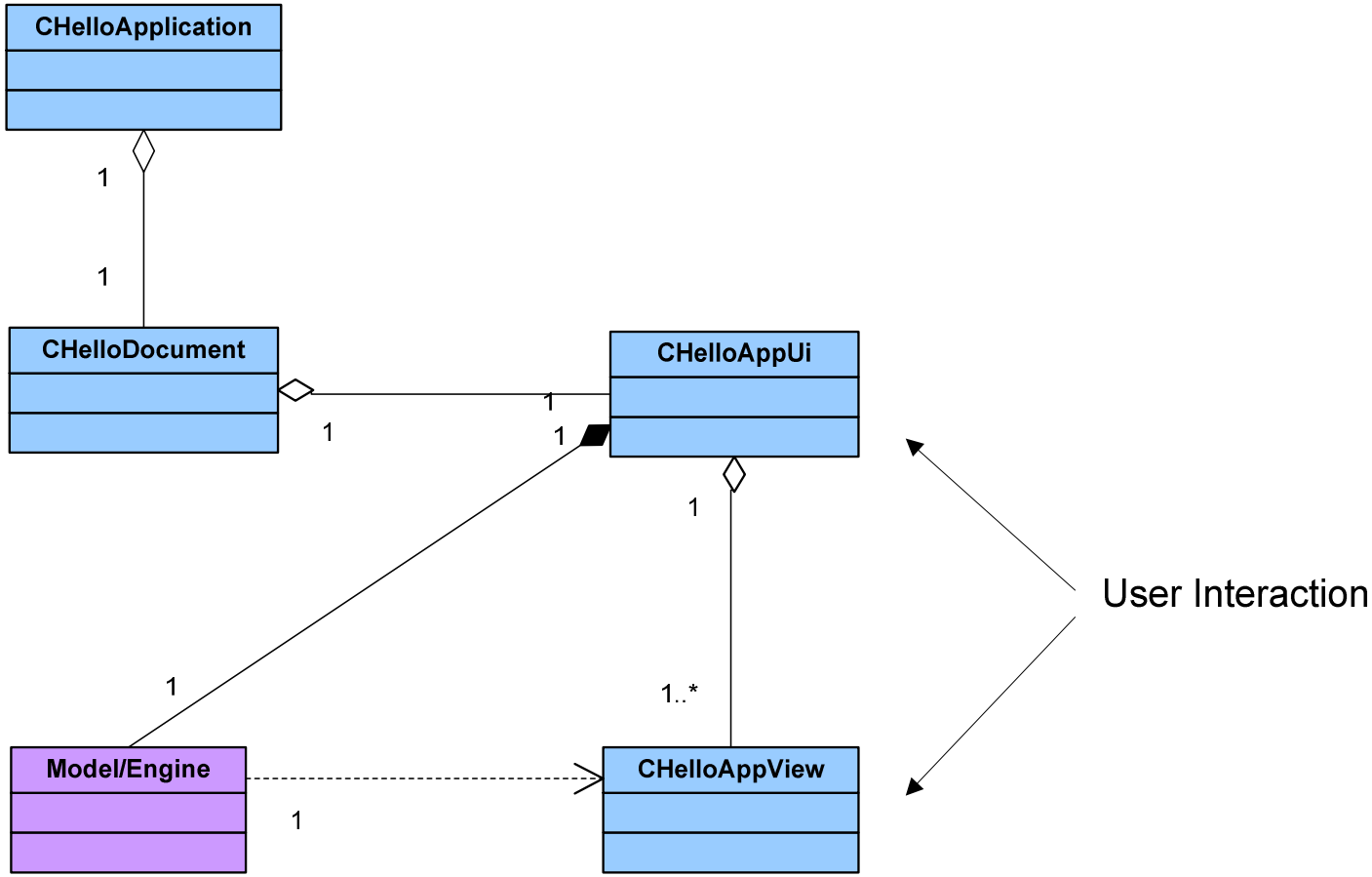
- Layering is not strictly hierarchical

Source: www.symbian.com

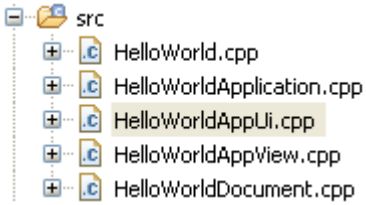
Source: Nokia

Source: Nokia

Symbian Concepts: Application Structure



Hello World with Symbian C++ (partly)




```
case ECommand2:
    {
    RFile rFile;

    //Open file where the stream text is
    User::LeaveIfError(rFile.Open(CCoeEnv::Static()->FsSession
            KFileName, EFileStreamText));//EFileShareReadersOn
    CleanupClosePushL(rFile);

    // copy stream from file to RFileStream object
    RFileReadStream inputFileStream(rFile);
    CleanupClosePushL(inputFileStream);

    // HBufC descriptor is created from the RFileStream object
    HBufC* fileData = HBufC::NewLC(inputFileStream, 32);

    CAknInformationNote* informationNote;

    informationNote = new (ELeave) CAknInformationNote;
    // Show the information Note
    informationNote->ExecuteLD(*fileData);

    // Pop loaded resources from the cleanup stack
    CleanupStack::PopAndDestroy(3); // filedata, inputFileStre
    }
    break;
```

```
void CHelloWorldAppUi::ConstructL()
    {
    // Initialise app UI with standard value.
    BaseConstructL(CAknAppUi::EAknEnableSkin);

    // Create view object
    iAppView = CHelloWorldAppView::NewL(ClientRect());

    // Create a file to write the text to
    TInt err = CCoeEnv::Static()->FsSession().MkDirAll(KFileName);
    if ((KErrNone != err) && (KErrAlreadyExists != err))
        {
        return;
        }

    RFile file;
    err = file.Replace(CCoeEnv::Static()->FsSession(), KFileName, EF
    CleanupClosePushL(file);
    if (KErrNone != err)
        {
        CleanupStack::PopAndDestroy(1); // file
        return;
        }

    RFileWriteStream outputFileStream(file);
    CleanupClosePushL(outputFileStream);
    outputFileStream << KText;

    CleanupStack::PopAndDestroy(2); // outputFileStream, file

    }
```

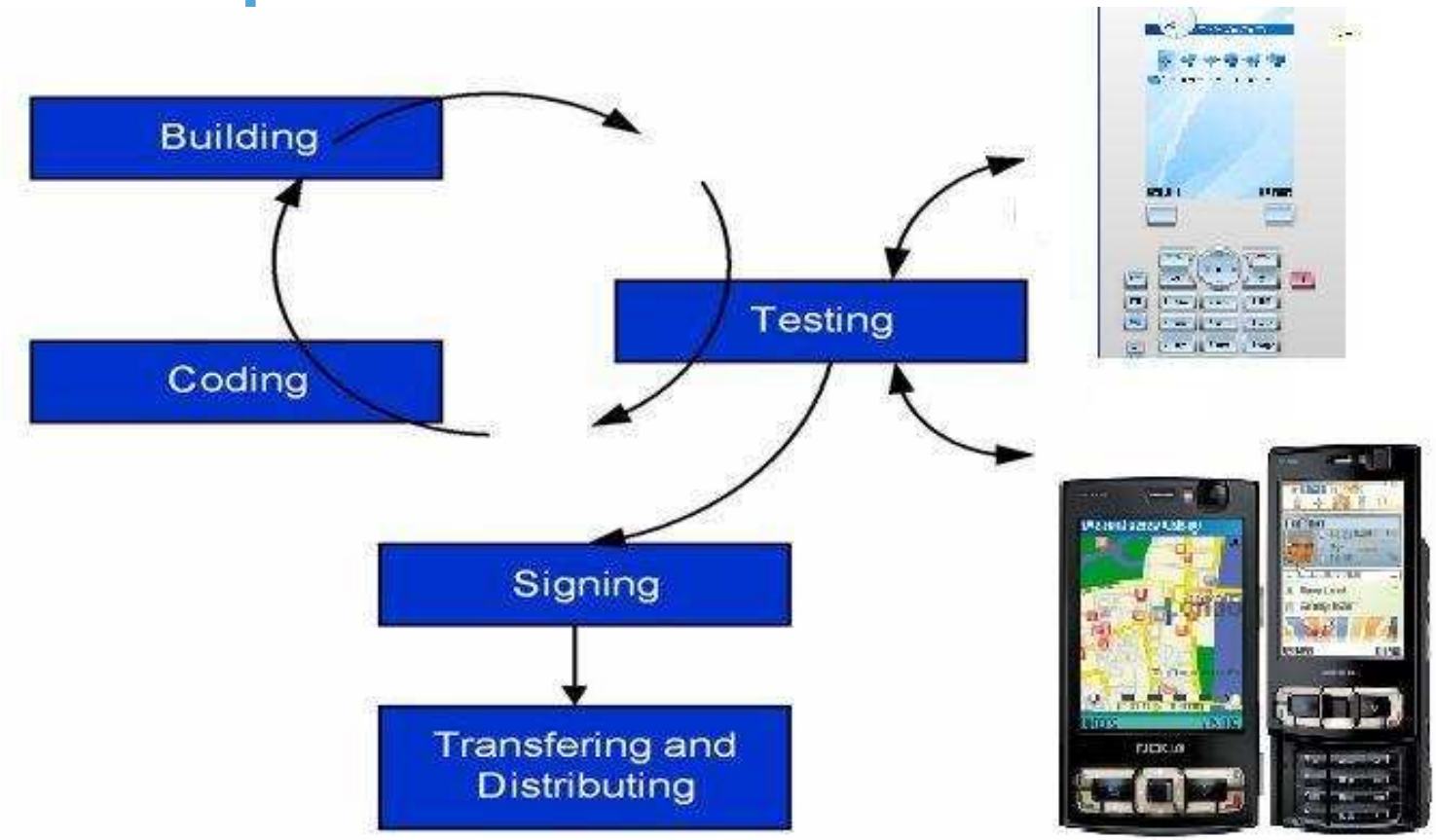
Symbian Concepts:
Build Info - mmp project file (1/2)

```
TARGET          HelloWorld_0xE1245789.exe
TARGETTYPE      exe
UID             0x100039CE 0xE1245789

SOURCEPATH      ..\src
SOURCE          HelloWorld.cpp
SOURCE          HelloWorldApplication.cpp
SOURCE          HelloWorldAppView.cpp
SOURCE          HelloWorldAppUi.cpp
SOURCE          HelloWorldDocument.cpp

SOURCEPATH      ..\data

START RESOURCE  HelloWorld.rss
HEADER
TARGET          HelloWorld_0xE1245789
.rsc
TARGETPATH      resource\apps
END // RESOURCE
```

Symbian Concepts:
Build Info - mmp project file (2/2)

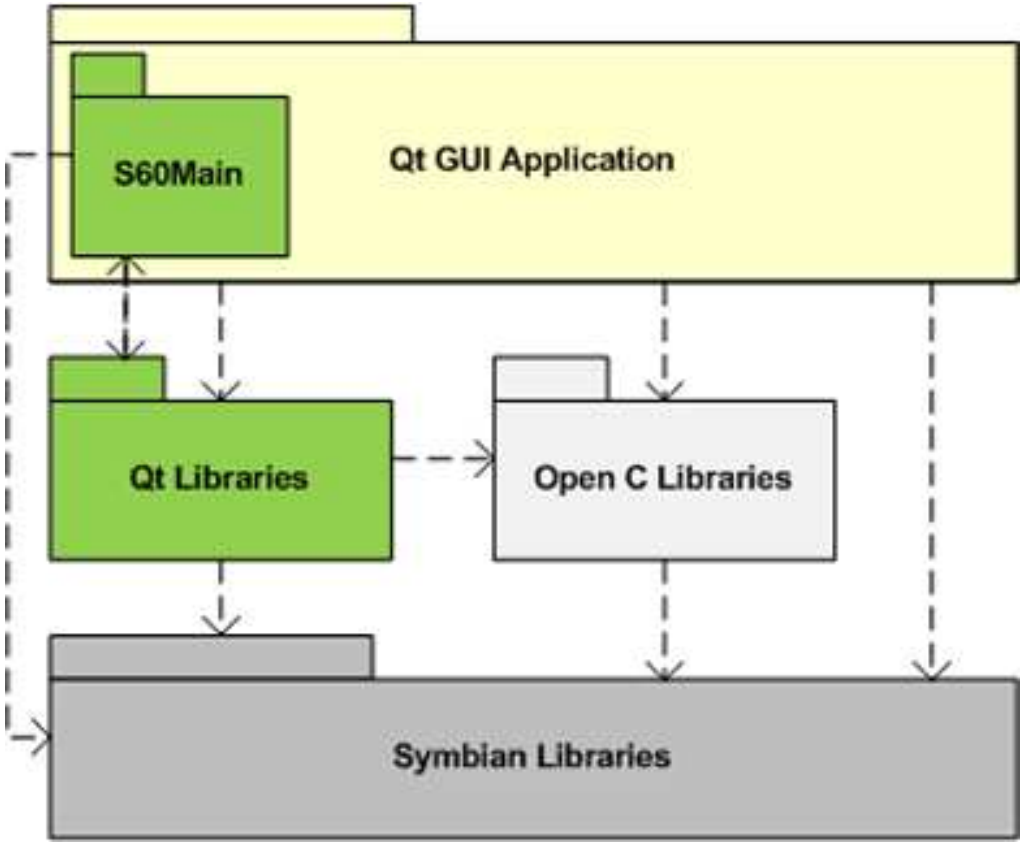
```
START RESOURCE    HelloWorld_reg.rss
TARGET            HelloWorld_0xE1245789
_reg.rsc
TARGETPATH         \private\10003a3f\apps
END

USERINCLUDE       ..\inc
USERINCLUDE        ..\help

SYSTEMINCLUDE     \epoc32\include

LIBRARY           euser.lib
LIBRARY           apparc.lib

LANG SC

VENDORID          0
SECUREID          0xE1245789

CAPABILITY        Location WriteDeviceData
// End of File
```

Development

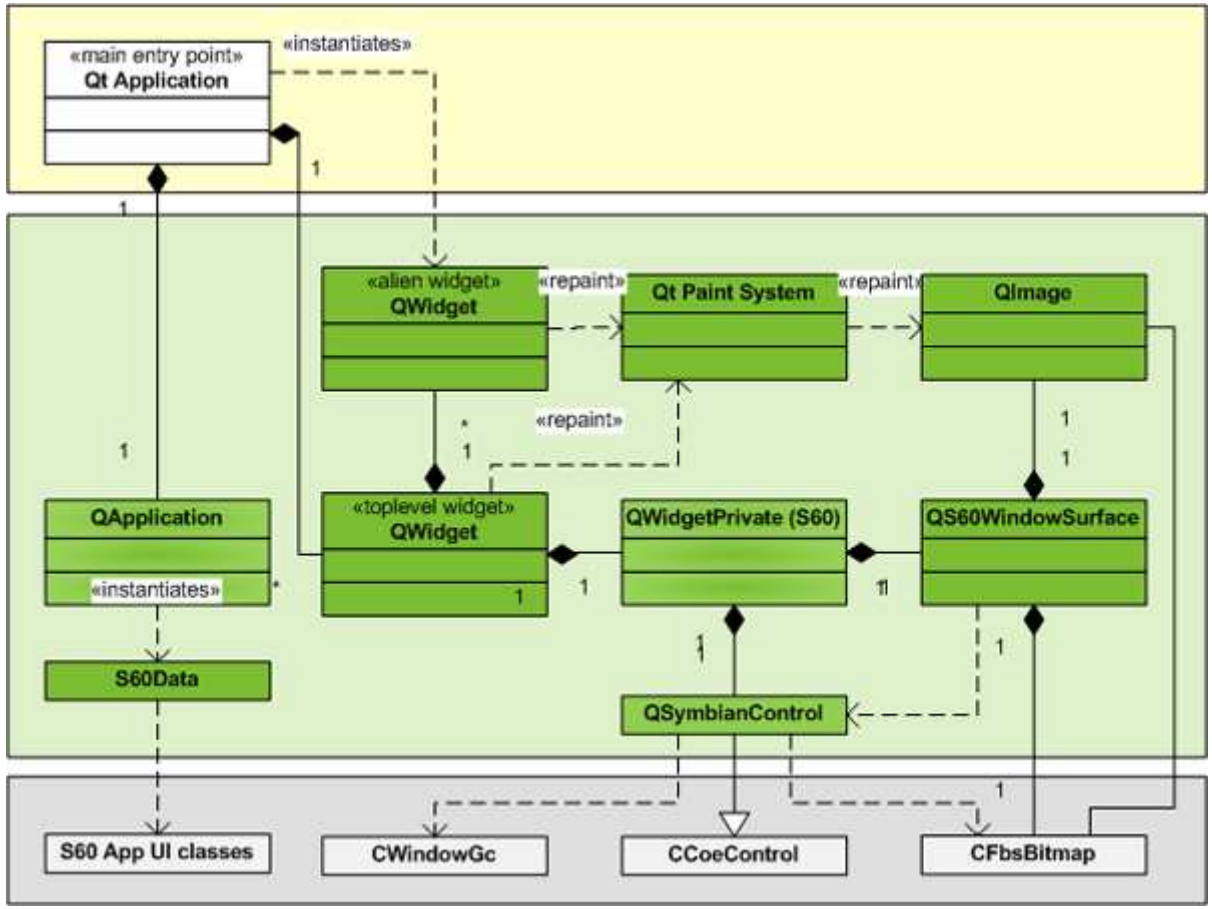


What is Qt for Symbian?

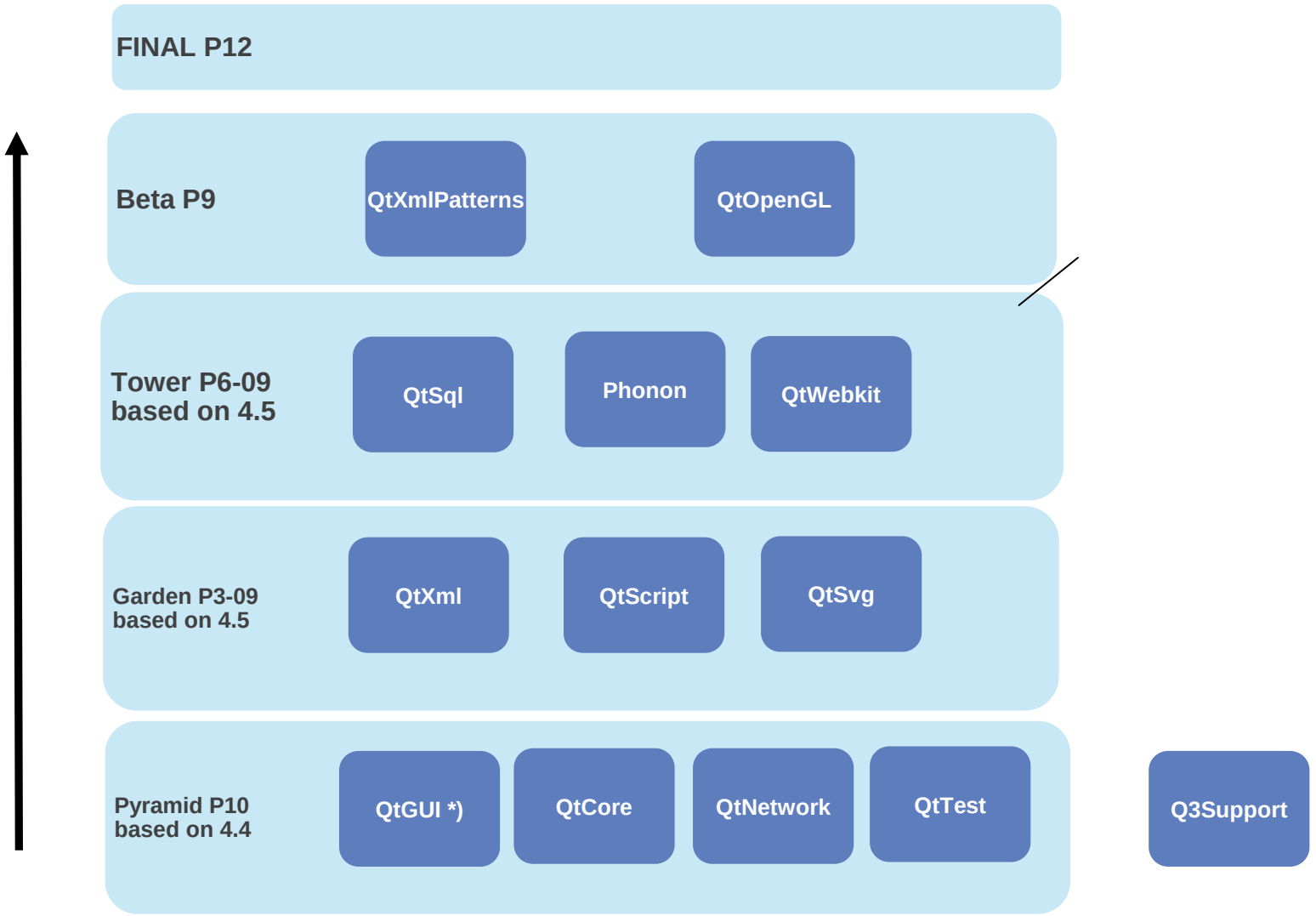
- Qt APIs have been implemented on top of OpenC/Symbian APIs
- The purpose is to have Qt programs running on the S60 platform



Qt/S60 GUI Integration



Qt for Symbian Platform - releases



http://pepper.troll.no/s60prereleases/

Qt for Symbian - Total Offering

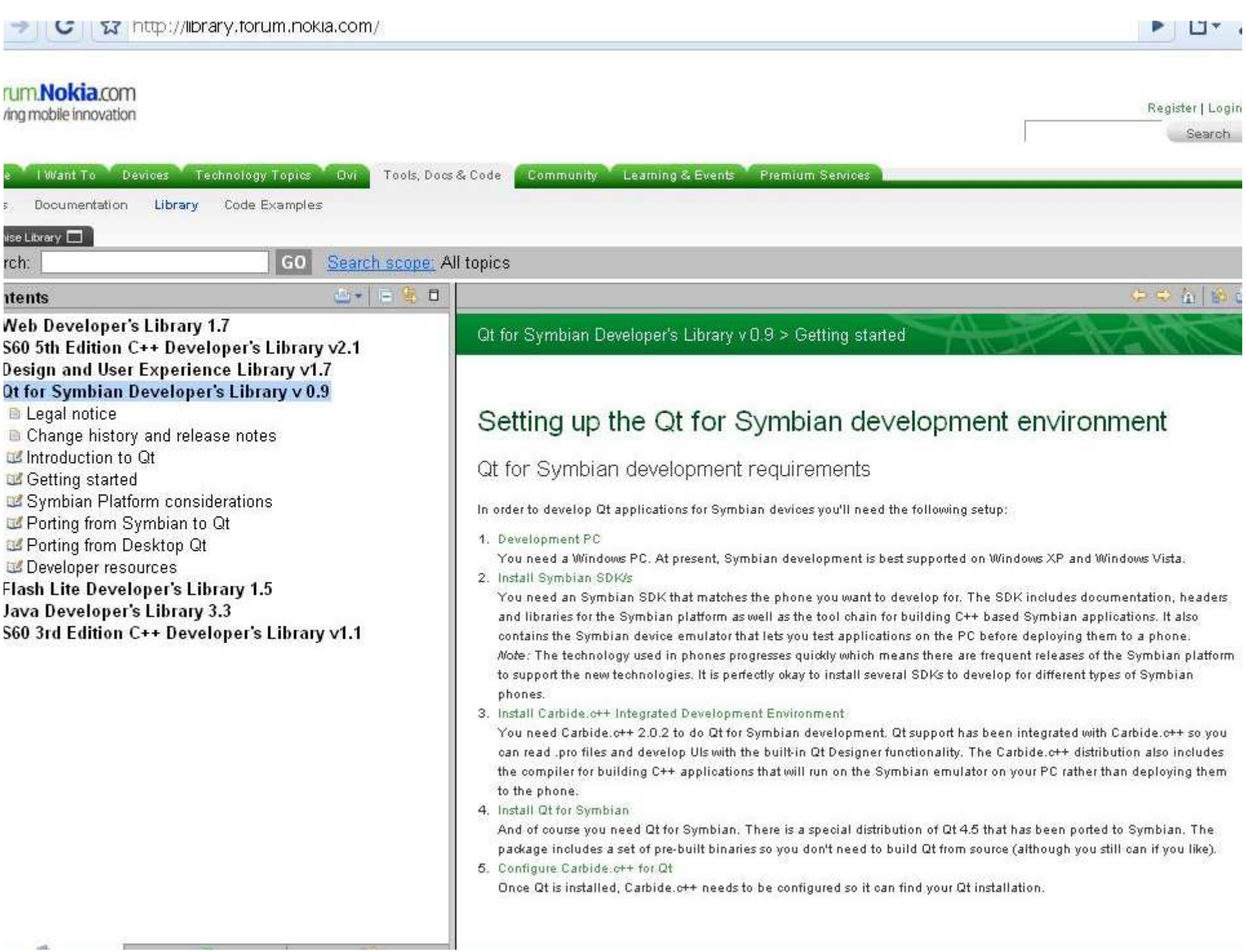
- **Qt for Symbian port** – Qt 4.6 ported to S60 3rd Edition FP1 and FP2, 5th edition
- **Carbide.c++ 2.0.2** onwards supporting Qt development on Symbian.
  - **Note : Qt creator** will also support Symbian development
  - Fully integrated form editor (*Qt Designer*)
  - Wizards for creating new Qt projects and classes
  - Integrated .pro file editor
  - Automated build setup for moc, uic, and rcc
  - http://www.forum.nokia.com/Resources_and_Information/Tools/IDEs/Carbide.c++/
- Mobility APIs
  - Cross-mobile project going on: http://labs.qt.nokia.com/page/Projects/QtMobility
  - Currently: Qt for S60 Mobile Extension
    - High abstraction level APIs making it possible to use most important services without using native Symbian C++
- Documents, examples, discussion board, wiki, Technical support, consultancy etc

Qt for Symbian - Forum Nokia resources

- Qt for Symbian **Developer's Library** (online version & Carbide plugin)
  - http://www.forum.nokia.com/info/sw.nokia.com/id/c41e7898-2dd8-4f23-a629-d27727519ffa/Qt_for_S60_Developers_Library.html
- Qt for Symbian **Discussion board** (monitored by FN experts)
  - http://discussion.forum.nokia.com/forum/forumdisplay.php?f=196
- Qt for Symbian **Wiki**
  - http://wiki.forum.nokia.com/index.php/Category:Qt_for_S60
- All the FN documents will be linked to from the following page
  - http://www.forum.nokia.com/Resources_and_Information/Documentation/Qt_for_S60.xhtml
- **Carbide.c++** 2.0 as an IDE for development
  - http://www.forum.nokia.com/Resources_and_Information/Tools/IDEs/Carbide.c++/
- **Symbian S60 SDKs** that can be used for Qt development on S60
  - 3rd Edition FP1 http://www.forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html
  - 3rd Edition FP2 and 5th Edition http://www.forum.nokia.com/info/sw.nokia.com/id/ec866fab-4b76-49f6-b5a5-af0631419e9c/S60_All_in_One_SDKs.html
- **Open C/C++**
  - http://www.forum.nokia.com/Resources_and_Information/Explore/Runtime_Platforms/Open_C_and_C++/
- Qt for Symbian **overview**
  - http://www.forum.nokia.com/Resources_and_Information/Tools/Runtimes/Qt_for_S60/
- Qt for Symbian **Quickstart** guide
  - http://www.forum.nokia.com/Resources_and_Information/Tools/Runtimes/Qt_for_S60/QuickStart.xhtml
- Qt for Symbian **Examples** (more in the SDK)
  - http://www.forum.nokia.com/info/sw.nokia.com/id/63313e06-8fd7-4a68-8610-80dd7ee22745/Qt_for_S60_Examples.html

Qt for Symbian Development Environment

- Let’s check that we have everything properly installed
- library.forum.nokia.com is a good source for that (see the next slide)


http://library.forum.nokia.com/
rum.Nokia.com
ring mobile innovation
Register | Login
Search
I Want To
Devices
Technology Topics
Ovi
Tools, Docs & Code
Community
Learning & Events
Premium Services
Documentation
Library
Code Examples
rch:
GO
Search scope: All topics
ntents
Web Developer's Library 1.7
S60 5th Edition C++ Developer's Library v2.1
Design and User Experience Library v1.7
Qt for Symbian Developer's Library v 0.9
Legal notice
Change history and release notes
Introduction to Qt
Getting started
Symbian Platform considerations
Porting from Symbian to Qt
Porting from Desktop Qt
Developer resources
Flash Lite Developer's Library 1.5
Java Developer's Library 3.3
S60 3rd Edition C++ Developer's Library v1.1
Qt for Symbian Developer's Library v 0.9 > Getting started
Setting up the Qt for Symbian development environment
Qt for Symbian development requirements
In order to develop Qt applications for Symbian devices you'll need the following setup:
1. Development PC
You need a Windows PC. At present, Symbian development is best supported on Windows XP and Windows Vista.
2. Install Symbian SDKs
You need an Symbian SDK that matches the phone you want to develop for. The SDK includes documentation, headers and libraries for the Symbian platform as well as the tool chain for building C++ based Symbian applications. It also contains the Symbian device emulator that lets you test applications on the PC before deploying them to a phone.
Note: The technology used in phones progresses quickly which means there are frequent releases of the Symbian platform to support the new technologies. It is perfectly okay to install several SDKs to develop for different types of Symbian phones.
3. Install Carbide.c++ Integrated Development Environment
You need Carbide.c++ 2.0.2 to do Qt for Symbian development. Qt support has been integrated with Carbide.c++ so you can read .pro files and develop UIs with the built-in Qt Designer functionality. The Carbide.c++ distribution also includes the compiler for building C++ applications that will run on the Symbian emulator on your PC rather than deploying them to the phone.
4. Install Qt for Symbian
And of course you need Qt for Symbian. There is a special distribution of Qt 4.5 that has been ported to Symbian. The package includes a set of pre-built binaries so you don't need to build Qt from source (although you still can if you like).
5. Configure Carbide.c++ for Qt
Once Qt is installed, Carbide.c++ needs to be configured so it can find your Qt installation.

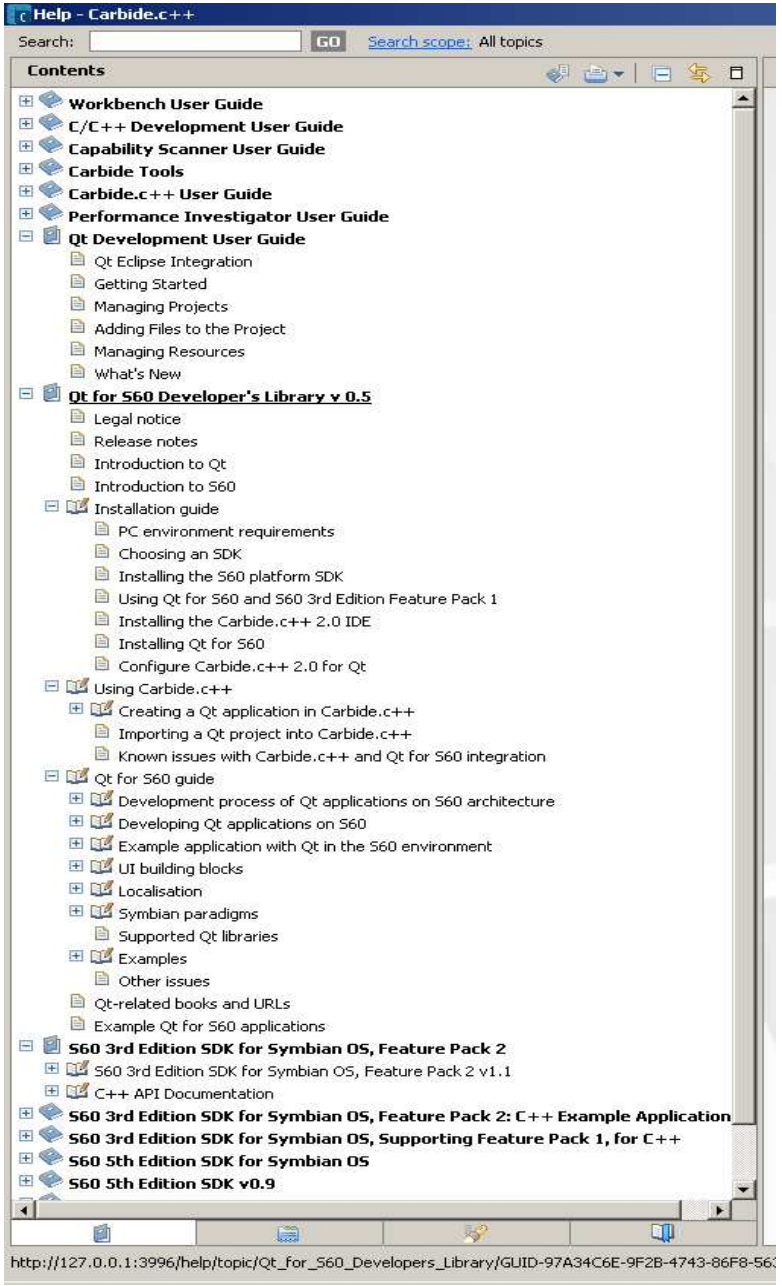
Qt for Symbian Developer's Library

- http://www.forum.nokia.com/info/sw.nokia.com/id/c41e7898-2dd8-4f23-a629-d27727519ffa/Qt_for_S60_Developers_Library.html
- Available as an online version and as a Carbide plugin
- Installing the Carbide Eclipse plugin:
  1. Download the Eclipse plugin zip file to any folder on your computer
  2. Unzip the zip file.
  3. Copy the JAR file included in the zip file.
  4. Paste the JAR file in the eclipse\plugins folder of your Eclipse installation, f.ex C:\Program Files\Nokia\Carbide.c++ v2.0\plugins

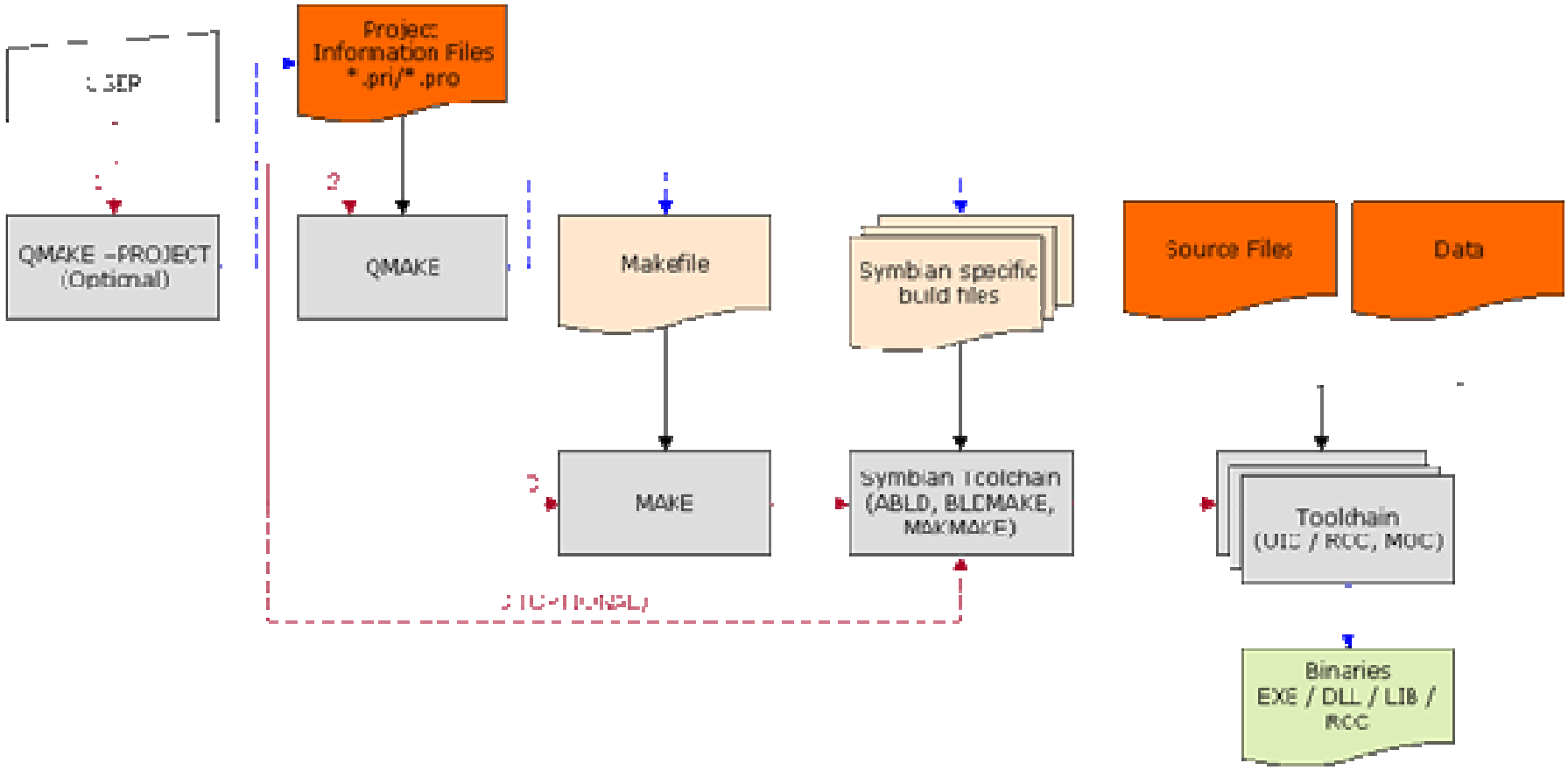
- The Qt for Symbian Developer's Library appears in the Eclipse Help Contents menu the next time you open your Eclipse installation.

Qt for Symbian Developer's Library

- After installing the 'Qt for Symbian Developer's Library' Carbide plugin you will see it in Carbide help (select Help->Help Contents from Carbide menu bar)
  - Note that the Qt Development User Guide that is 'built-in' with Carbide 2.0 is a separate library
- The developers library contains the following sections
  - Legal notice & Release notes
  - Introduction to Qt
  - Introduction to S60
  - Installation guide
  - Using Carbide.c++
  - **Qt for S60 guide**
  - Qt-related books and URLs
  - Example 'Qt for S60' applications



Qt and Symbian OS Toolchain Integration



Hand-building Qt Application on S60

- In a normal Qt way for the Qt part:
- `qmake -project`
- `qmake`
  - Component description file created (`bld.inf`)
  - Project description file created (`.mmp file`)
  - Extension Makefiles created
    - Take care of executing moc, uic, rcc tools
    - Integrated like mifconv to the toolchain
- Then, the actual compilation either with
  - Generated platform specific Makefile: `make debug-winscw`
  - Or: `bldmake bldfiles + abld build winscw udeb`
  - Both do the same

make command (for cmd line building)

Command	Description
make	Creates abld and the project makefiles and builds debug build of the application for the emulator (winscw udeb).
make debug	Creates debug builds (winscw/gcce/armv5 udeb).
make debug-winscw	Creates winscw debug build.
make debug-gcce	Creates gcce debug build.
make debug-armv5	Creates armv5 debug build.
make release	Creates release builds (gcce/armv5 urel).
make release-gcce	Creates gcce release build.
make release-armv5	Creates armv5 release build.
make export	Copies the exported files to their destination.
make cleanexport	Removes files created with `make export`.
make mocclean	Removes the header and source files created by the moc tool.
make mocables	Runs moc tool on necessary files.
make clean	Removes everything built with abld target, exported files and makefiles.
make distclean	As make clean, but also removes all Symbian specific files created with qmake.
make confclean	As make distclean, but also cleans everything generated by configure call. Note that this command is only available in the Qt root directory.
make run	Launches the application in the emulator.

MMP File Definitions in a .pro Project File

- Everything you need in a Symbian OS MMP file, can be defined in the Qt .pro file. Symbian-specific extensions should be defined inside a special block as shown below

```
TEMPLATE = app
TARGET = Juuba
QT += core \
   gui \
   network
HEADERS += Juuba.loc \
   Juuba.h
SOURCES += Juuba.rss \
   Juuba_reg.rss \
   main.cpp \
   Juuba.cpp
FORMS += Juuba.ui
RESOURCES +=
symbian:TARGET.UID3 = 0xEA61E576
```

- This way the same .pro file would potentially work in other environments as well

Symbian Extensions to Project Files

- Some other examples of Symbian specific keywords:
  - `TARGET.SID = 0xA000017F`
  - `TARGET.VID = 0x70000001`
  - `TARGET.EPOCSTACKSIZE = 0x5000 // 20kb`
  - `TARGET.EPOCHEAPSIZE = 0x20000 0x1000000 // Min 128kb, Max 16Mb`
  - `TARGET.CAPABILITY = NetworkServices`

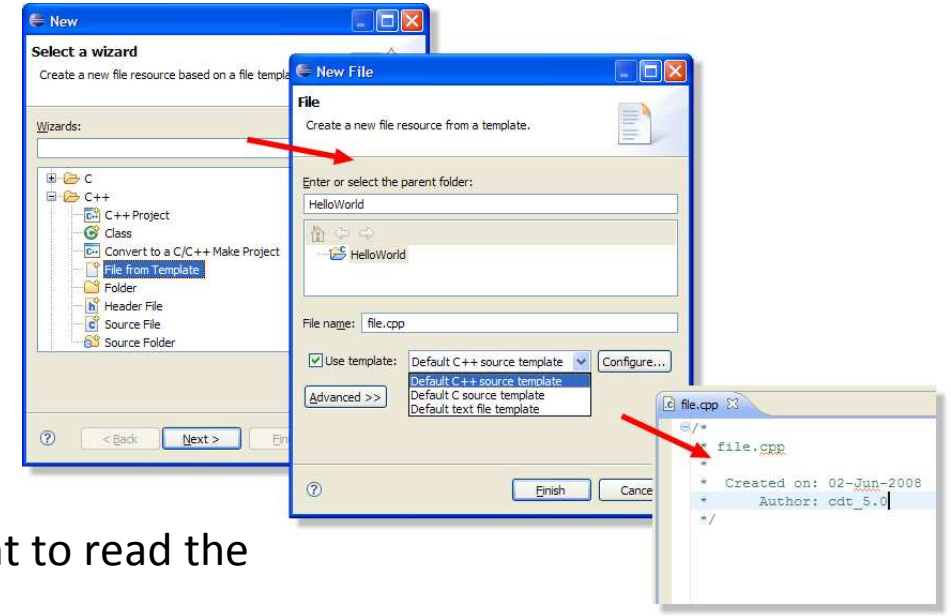
Qt for S60 Example applications?

- The Qt for Symbian release contains 50 examples and 10 demo applications
- Additional examples in Forum Nokia web site
- In addition to these you can **take any Qt open source project** and compile it for Symbian
  - For examples targeted for desktop naturally **UI optimization** needs to be done
  - If the example you've found doesn't work on Symbian it indicates that some of the used features is not ready yet!
- http://www.qt-apps.org/ contains hundreds of applications (with source codes)

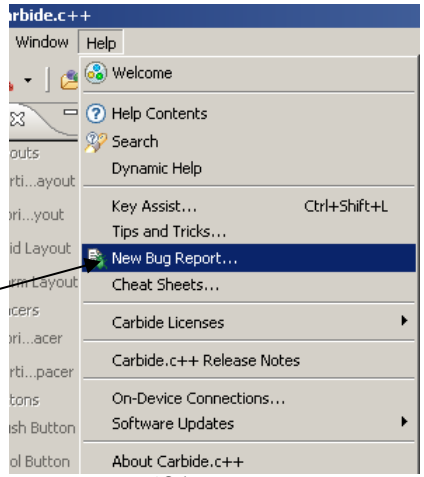
Example : Build, run, install to device

- C:\Qt\4.5.0-tower\examples\graphicsview\collidingmice
  - First change to `view.showMaximized();`
- `qmake collidingmice.pro`
  - Have a look at the generated files
- `make debug-winscw`
- `epoc`
- Build for device : `make debug-gcce`
- Create self signed package and install to device (connected via PC Suite)
  - `createpackage -i collidingmice_gcce_udeb.pkg`

Carbide.c++ 2.0.2

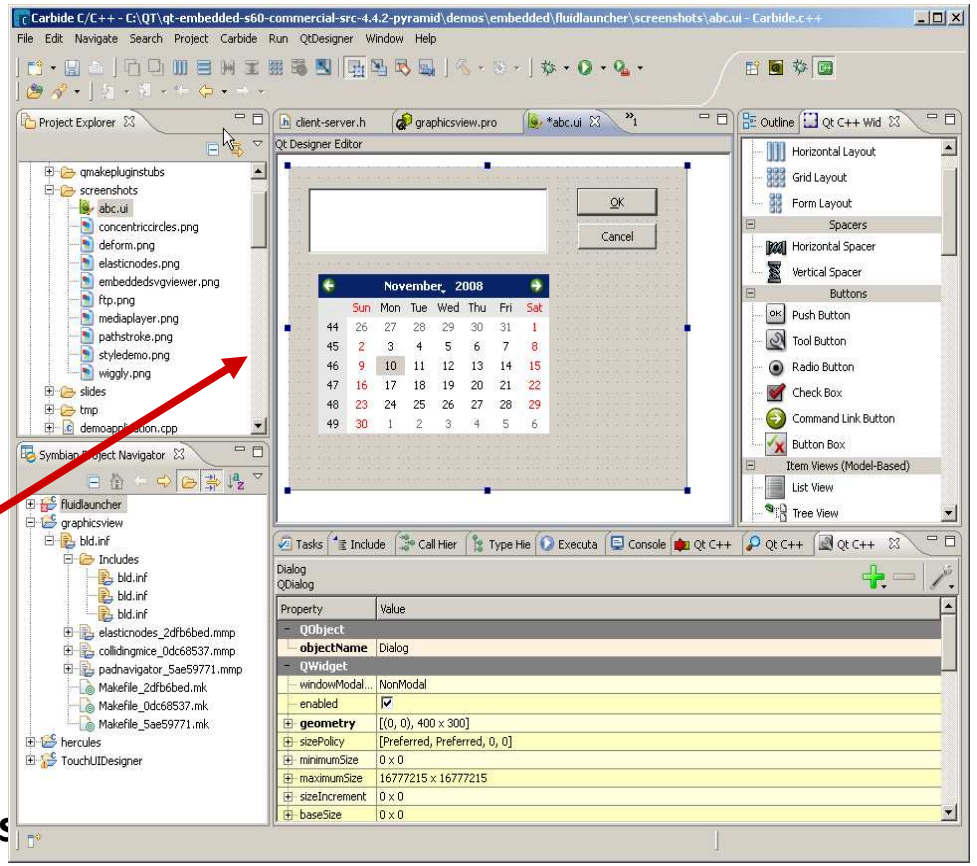


- Now Qt for Symbian support!
- With Carbide.c++ installed, you might want to read the Carbide.c++: Introductory White Paper
  - http://www.forum.nokia.com/info/sw.nokia.com/id/cae9ea59-eee0-4b98-aaa2-1b6ecd879222/Carbide_cpp_Introductory_White_Paper_V1_1_en.pdf.html
- And view the Getting Started with Carbide.c++ Express Screencast, which will take you through the basics of creating, building, and deploying your first application.
  - http://www.forum.nokia.com/info/sw.nokia.com/id/af80987a-a72d-44cf-bf00-1926be01058c/Carbide_cpp_Getting_Started_v1_1_en.exe.html
- Bug reports : Carbide Help menu, ”New Bug Report”

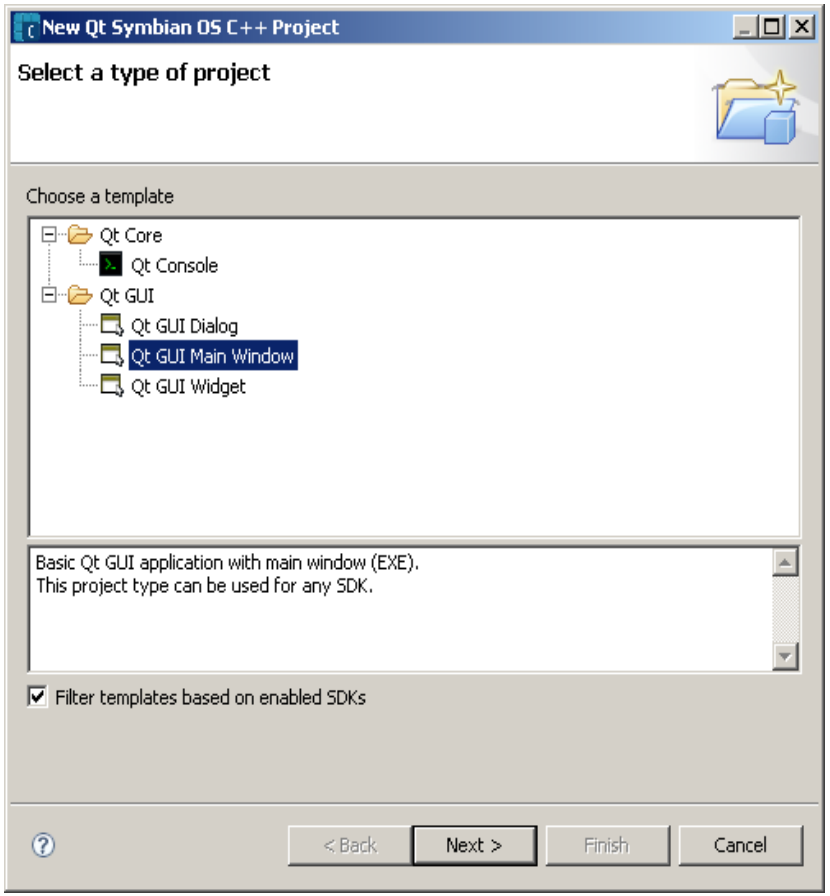


Qt Support in Carbide.c++ 2.0.2

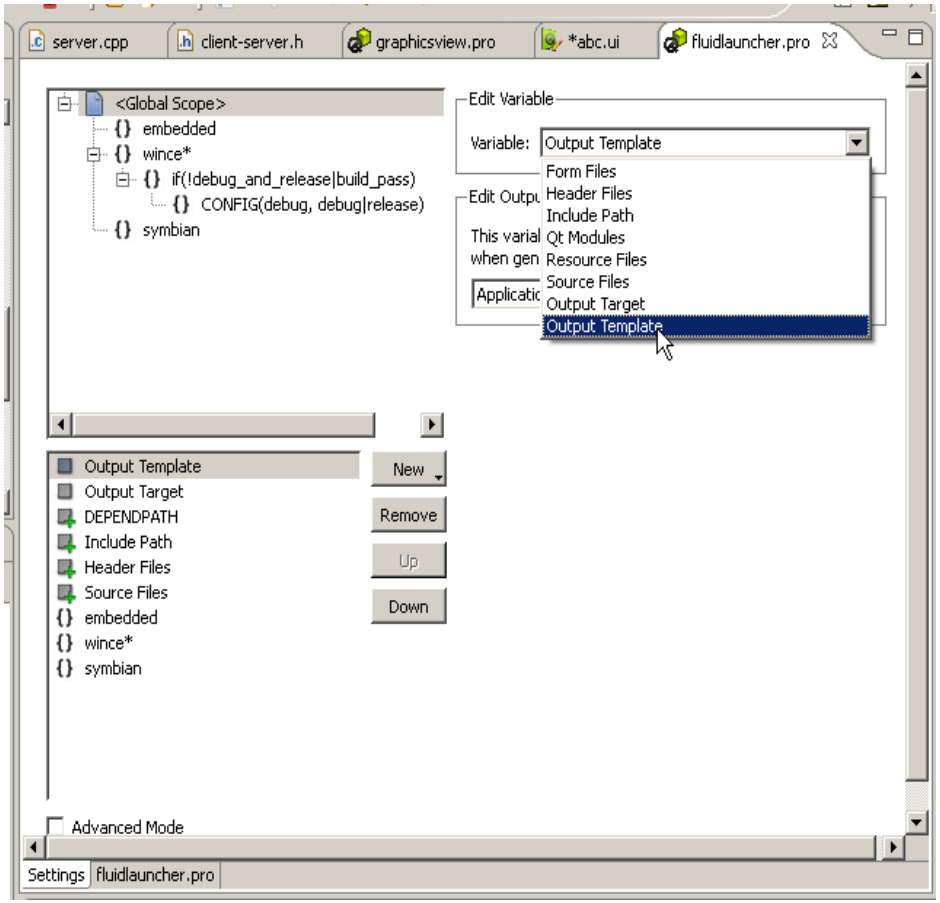
- Support for Qt development
- Carbide works with Qt projects
  - .PRO editor for easy project configuration
  - Package /deploy projects to phone
  - Qt plug-ins are included in Carbide
  - Qt project Wizard
  - Fully integrated form editor (*Qt Designer*)
  - Wizards for creating new Qt projects and classes
  - Integrated .pro file editor
  - Automated build setup for moc, uic, and rcc
- Since **Qt application on Symbian device is a Symbian application**, you can use **same tools for debugging**, profiling etc.



Additional Qt support in Carbide



Qt Template Wizard



.PRO Project Editor

Symbian Mobility APIs

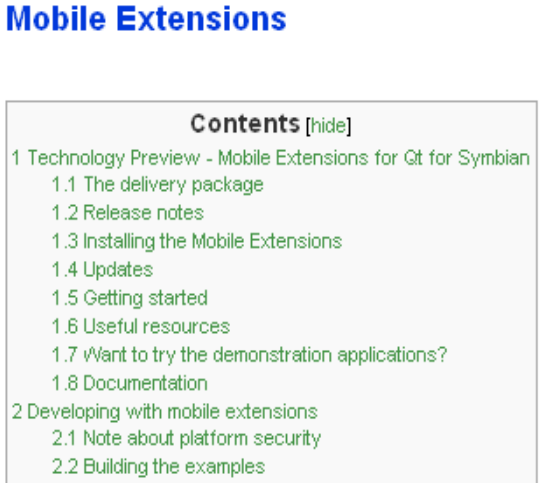
Using mobile features from Qt applications

Background

- Qt currently provides no APIs for mobile devices related features
- QtMobility will fill this gap in future
  - http://labs.qt.nokia.com/page/Projects/QtMobility
- Common mobile-specific API for e.g. Windows CE, Maemo, and S60 devices
- Technology preview - Qt mobile extensions
  - Sensors, location, messaging, telephony, contacts, notifications, profile, landmarks, alert, vibra, and sensor

Qt/Symbian Mobile Extensions

- Set of special APIs covering mobile devices related features currently not supported by official Qt
- Qt for S60 only
- Available in Forum Nokia
  - http://wiki.forum.nokia.com/index.php/Mobile_Extensions
  - wide feature set
  - mature enough to be used in projects
- APIs will be replaced by Mobility APIs in the future



Sending an SMS message with Symbian C++

```
void MessageSender::SendMessageL(const TDesC& recipient, const TDesC& message ) {
	RSendAs msgServer;  // The easier and restricted RSendAs API
	msgServer.Connect();
	CleanupClosePushL( msgServer );
	RSendAsMessage msg;
	msg.CreateL( msgServer, KSenduiMtmSmsUid );
	CleanupClosePushL( msg );	//Set values to recipient and body text
	msg.AddRecipientL( recipient, RSendAsMessage::ESendAsRecipientTo );
	msg.SetBodyTextL(message);
	msg.SendMessageAndCloseL();
	CleanupStack::PopAndDestroy( &msg );
	CleanupStack::PopAndDestroy( &msgServer );
}
```

Sending an SMS message with Qt (Wrapper)

```
void MessageSender::SendMessage( QStringList recipients, QString message )
{
   XQMessaging *messaging = new XQMessaging( this );
   XQMessage message( recipient, message ));
   messaging->send(message);
}
```

Current Offering of Mobile Extensions

Mobile Extension	Purpose	Mobile Extension	Purpose
1.Access Point Manager	Listing available IAPs (Internet Access Points), setting the IAP to be used, scanning available WLANs etc.	**10.Profile**	Reading profile information and setting active profile.
2.Audio API	Providing Audio **recording** functionality	**11.Resource Access**	Accessing Symbian resource files.
3.Camera	Using device's onboard camera with viewfinder, focus and capturing images.	**12.Sensors**	Acceleration and orientation sensor data access
4.Contacts	You can access contacts database with this.	**13. Settings Manager API**	Accessing central repository and Publish & Subscribe.
5.Installer API	Silent install functionality	**14. System Information**	Accessing system information (language, battery, nw, ...) with this.
6.Landmarks	List available landmarks and add new landmarks	**15. Telephony**	Make a circuit switched call and getting call status notifications
7.Location	Accessing device location information.	**16. Utils**	Platform specific utils.
8.Media	Retrieving lists of music, image, video and sound files located in the gallery	**17. Vibra**	Using device's vibra
9.Messaging	Sending and receiving SMS and MMS messages.	**18. Alarms**	Alarm APIs
		19. Calendar	Access to Calendar

Content of the Mobile Extensions package

- Binaries
  - DLLs and LIBs (and headers)
  - SIS package to be installed in device
- Examples
  - one example application per API
- Demos
  - utilizes several APIs at the same time
- Documentation
- Source code

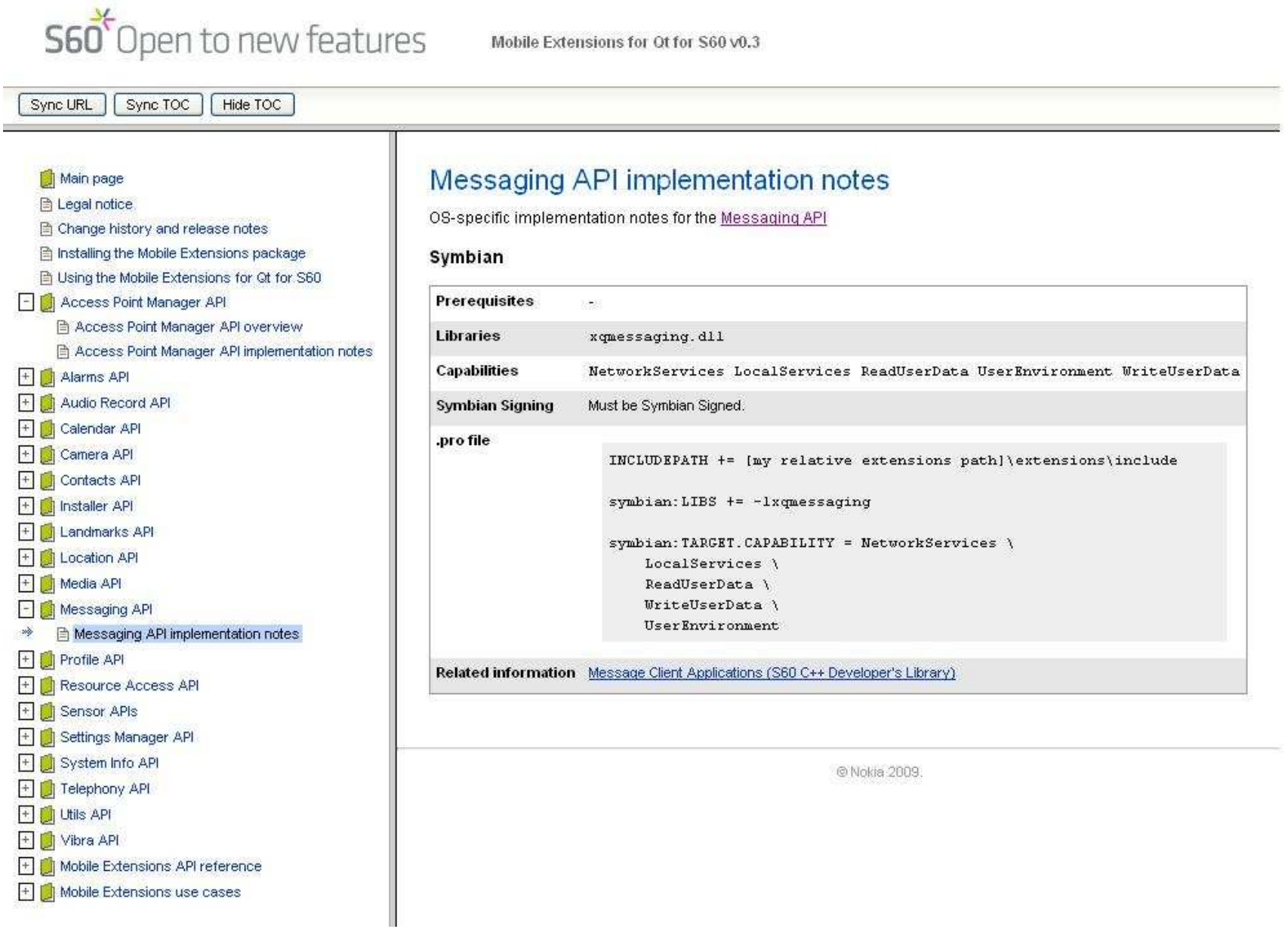
Naming conventions

- Libs start with xq
  - xqsensor.lib, xqsensor.dll
  - xqvibra.lib, xqvibra.dll
  - etc.
- Classes start with XQ
  - XQAccelerationSensor
  - XQVibra
- Headers are named according to their roles
  - no strict convention, e.g.
    - xqaccsensor.h
    - xqvibra.h
  - system include directory contains also standard C++ style headers without .h extension
    - XQAccelerationSensor
    - XQVibra

Mobile Extensions - Installation

- Download the package
  - http://www.forum.nokia.com/piazza/wiki/images/b/b4/Mobile_extensions_preview_3.zip
- From ./install_to_sdk unzip the zip package in EPOCROOT
  - e.g. D:\S60\devices\Nokia_N97_SDK_v1.0
- From ./install_to_device install the SISx package(s) in the device
- From ./doc unzip the documentation

Mobile Extension Documentation



Use

- In .pro file
  - define INCLUDEPATH += ./extensions/include
  - define needed libraries
    - e.g. symbian:LIBS += -lxqsensor
- In source code
  - include needed headers

Good to know

- Mobile Extensions use Symbian or S60 native APIs
- Some APIs work only in HW
  - Installer
  - Camera
  - Sensor (emulator's simulation is poor)
- Some APIs need restricted capabilities
  - needs Symbian signing or IMEI based certificates
- More on PlatSec in the following slides!!

Symbian Platform Security

Introduction

- Platform Security (PlatSec) is a security model implemented for securing the data and integrity of the phone.
- PlatSec is implemented at the software level
- Main task is to prevent applications from making unauthorized access to hardware, software and system or user data

Goal

- ”To protect system integrity and to shield data and functionality from malicious applications”
- Malicious applications
  - Trojans
  - Viruses
  - Accidentally badly behaving programs

Capabilities

- A capability is a “token” to access to a certain sensitive service
  - E.g. an application opening a network socket must have capability to network services
  - E.g. a server using the user location must have capability to user location services
- Capability checks are performed by corresponding servers (TCE)

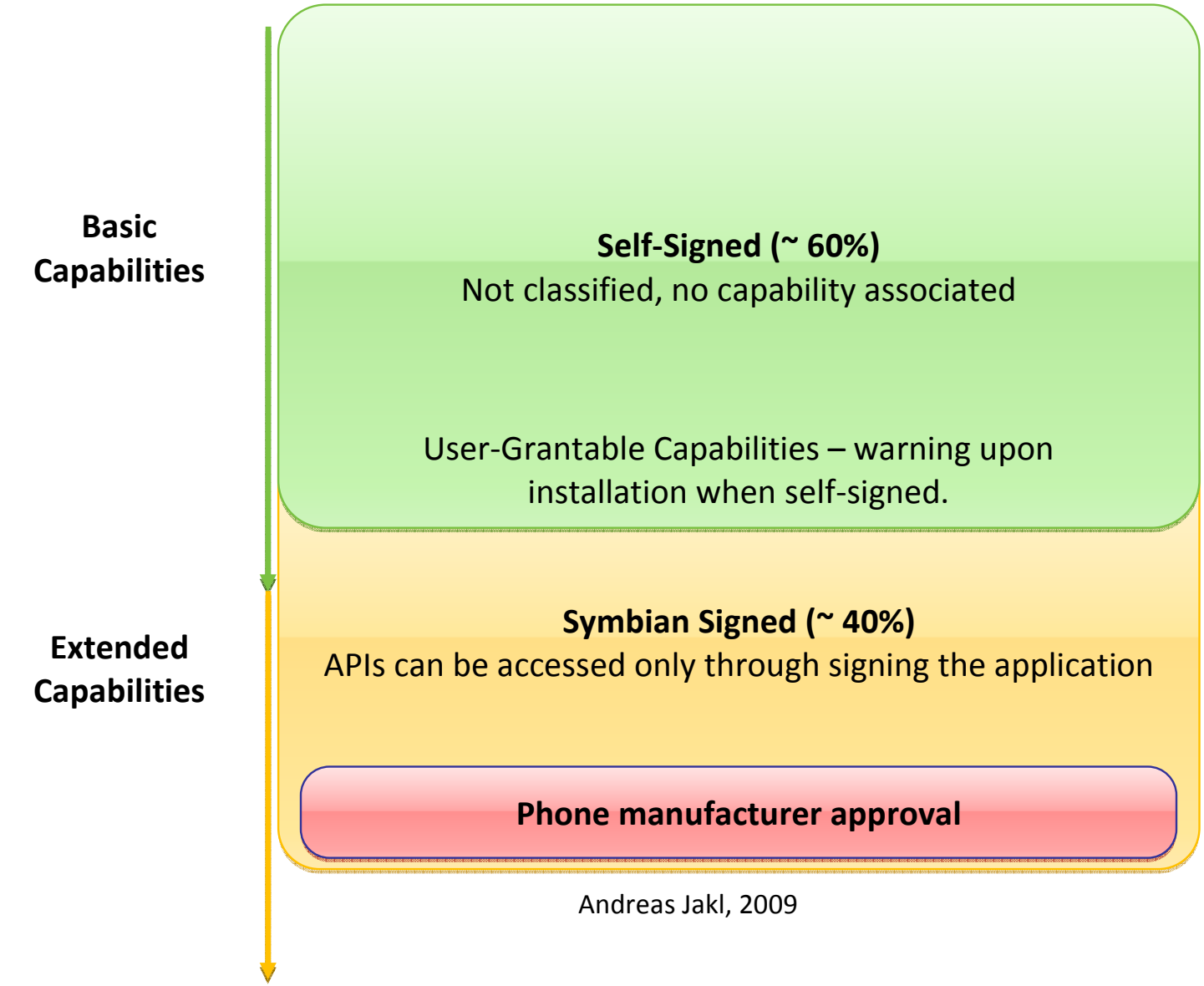
Symbian Capabilities - Model

- *„How trustworthy is the application?“*
- **Capabilities check level of trust**
  - Have to be defined for an application when compiling
  - Managed by the kernel
  - Can not be modified after installation
  - Defined for each process
  - Choose capabilities depending on required functionality

Capabilities

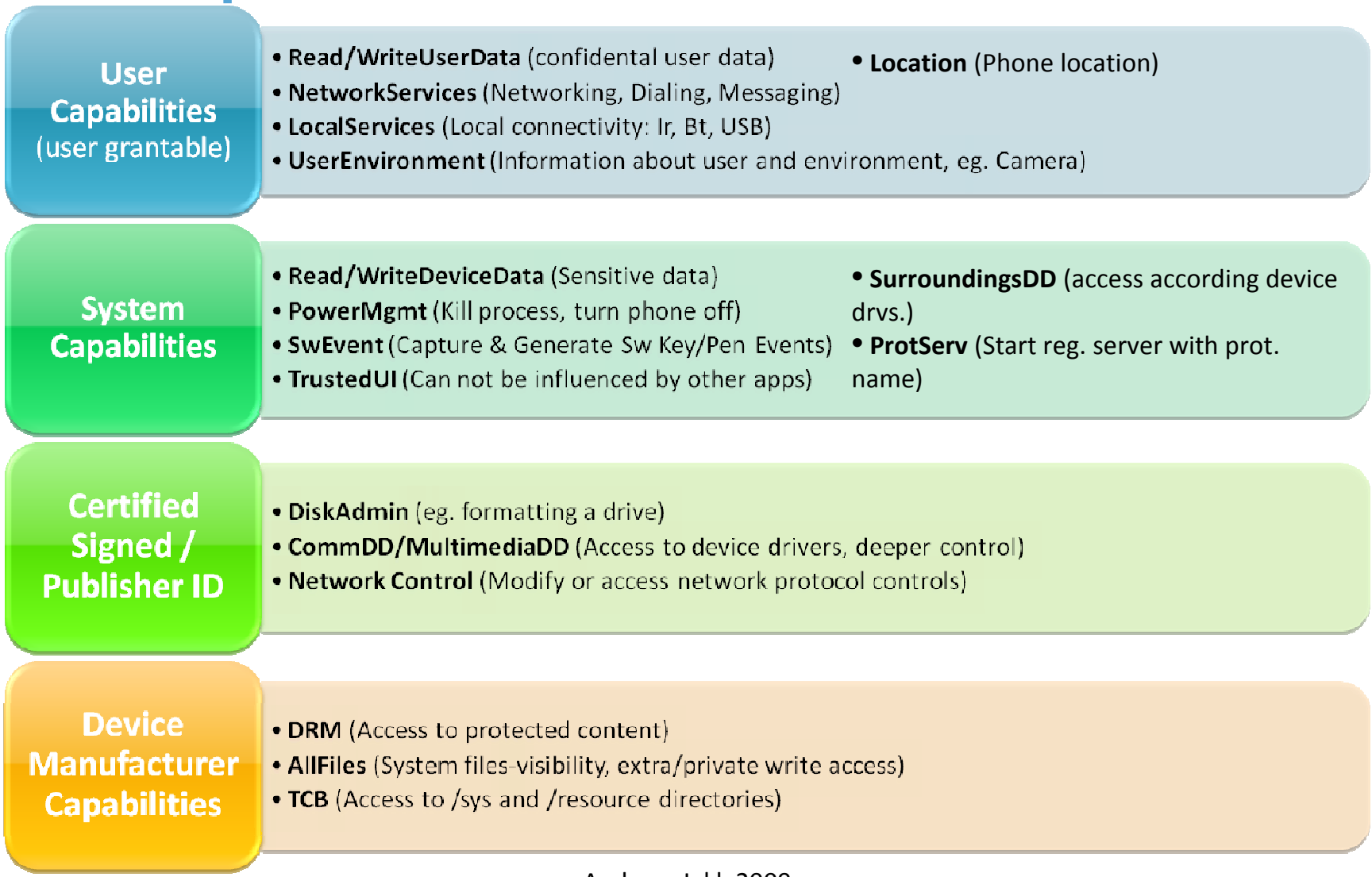
- **User / Basic Capabilities**
  - Can be allowed by the user
  - Easy to understand
  - App. has to be at least self-signed (Automatically done by IDE)
- **System / Extended Capabilities**
  - App. has to be Symbian Signed http://www.symbiansigned.com/
  - Apps can go deeper into the system

API Access



Andreas Jakl, 2009

Capabilities - Overview



Andreas Jakl, 2009

User Capability API Examples

- Recording audio with CMdaAudioRecorderUtility::RecordL() requires UserEnvironment capability
- Deleting an SMS entry with RMsvServerSession::DeleteEntriesL(const CMsvEntrySelection &,TMsvOp) requires ReadUserData and WriteDeviceData capabilities

System Capability API Examples

- Turning the device off with UserHal::SwitchOff() or killing a process RProcess::Terminate() requires PowerMgmt capability
- Starting a server with CServer2::Start(const TDesC &) requires ProtServ capability

Restricted Capability API Examples

- Opening an RFormat session (subsession to RFs used for formatting a disk) requires DiskAdmin capability
- Using CVideoRecorderUtility for recording a video clip requires MultimediaDD

Platform Security

- All **platform security rules apply for Qt applications** in the Symbian environment.
- Because Qt is mainly ported on top of Open C, the required capabilities are also derived from those APIs.
- **Platform security requires that needed capabilities be defined in the project file**.
- The Qt application may require, for example, the following capabilities:
  - AllFiles, when using file operations and accessing protected folders [6];
  - NetworkServices should be enough in most cases when using the QtNetwork module, but there might be certain API calls that also require NetworkControl.
- When using Symbian APIs the capabilities needed are, of course, the ones that the APIs define.

Capabilities in Symbian Project MMP file

- Capabilities of EXE, DLL or LIB are set in project mmp file by keyword CAPABILITY

```
TARGET          MyApp.exe
TARGETTYPE      exe
...
CAPABILITY      ReadUserData  Location TrustedUI
```

Capabilities and Qt .pro file

- Qt Extensions are **wrappers** implemented on top of Symbian APIs
- We have to ”know” which APIs an extension is using and which capabilities are needed when we are using the extensions. in our code
- Capabilities are listed in .pro –file, e.g.

```
INCLUDEPATH += [my relative extensions path]\extensions\include
   symbian:LIBS += -lxqlocation
   symbian:TARGET.CAPABILITY = Location
```

- Extension API documentation tells us, which capabilities to add when using a certain API.

Capability example: Sending an SMS message with Symbian C++

```
void MessageSender::SendMessageL(const TDesC& recipient, const TDesC& message ) {
	RSendAs msgServer;  // The easier and restricted RSendAs API
	msgServer.Connect();
	CleanupClosePushL( msgServer );
	RSendAsMessage msg;
	msg.CreateL( msgServer, KSenduiMtmSmsUid );
	CleanupClosePushL( msg );	//Set values to recipient and body text
	msg.AddRecipientL( recipient, RSendAsMessage::ESendAsRecipientTo );
	msg.SetBodyTextL(message);
	msg.SendMessageAndCloseL();
	CleanupStack::PopAndDestroy( &msg );
	CleanupStack::PopAndDestroy( &msgServer );
}
```

Capability example: Sending an SMS message with Qt (Wrapper)

```
void MessageSender::SendMessage( QStringList recipients, QString message )
{
   XQMessaging *messaging = new XQMessaging( this );
   XQMessage message( recipient, message ));
   messaging->send(message);
}
```

- Calling XQMessage::send() causes SendMessageAndCloseL() to be called.
- Calling SendMessageAndCloseL() requires NetworkServices capability
  - Using QWMessage::send() requires NetworkServices capability for the Application project (and corresponding signing)

Messaging API implementation notes

OS-specific implementation notes for the Messaging API

Symbian

Prerequisites	-
Libraries	`xqmessaging.dll`
Capabilities	`NetworkServices LocalServices ReadUserData UserEnvironment WriteUserData`
Symbian Signing	Must be Symbian Signed.

.pro file	`INCLUDEPATH += [my relative extensions path]\extensions\include`
	
	`symbian:LIBS += -lxqmessaging`
	
	`symbian:TARGET.CAPABILITY = NetworkServices \`
	`  LocalServices \`
	`  ReadUserData \`
	`  WriteUserData \`
	`  UserEnvironment`

Related information	Message Client Applications (S60 C++ Developer's Library)

Capability Checks

- Capability checks are done for the calling process at run-time
- The corresponding sensitive API service provider (server) is responsible for checking the capabilities of the calling process.
- From the system's point of view all capability types checks are handled in similar way.

Qt for Maemo

What is Maemo?

- Maemo is an open-source development platform for Internet Tablets
  - 90% of source code is open-source
  - Latest platform contains also phone functionality (mobile computer)

- Most of the tools, libraries and development processes that are used in Maemo are equally used and applied in the desktop arena
  - Easy SW portability
  - UI design should consider mobility issues

Maemo Devices and OS Versions

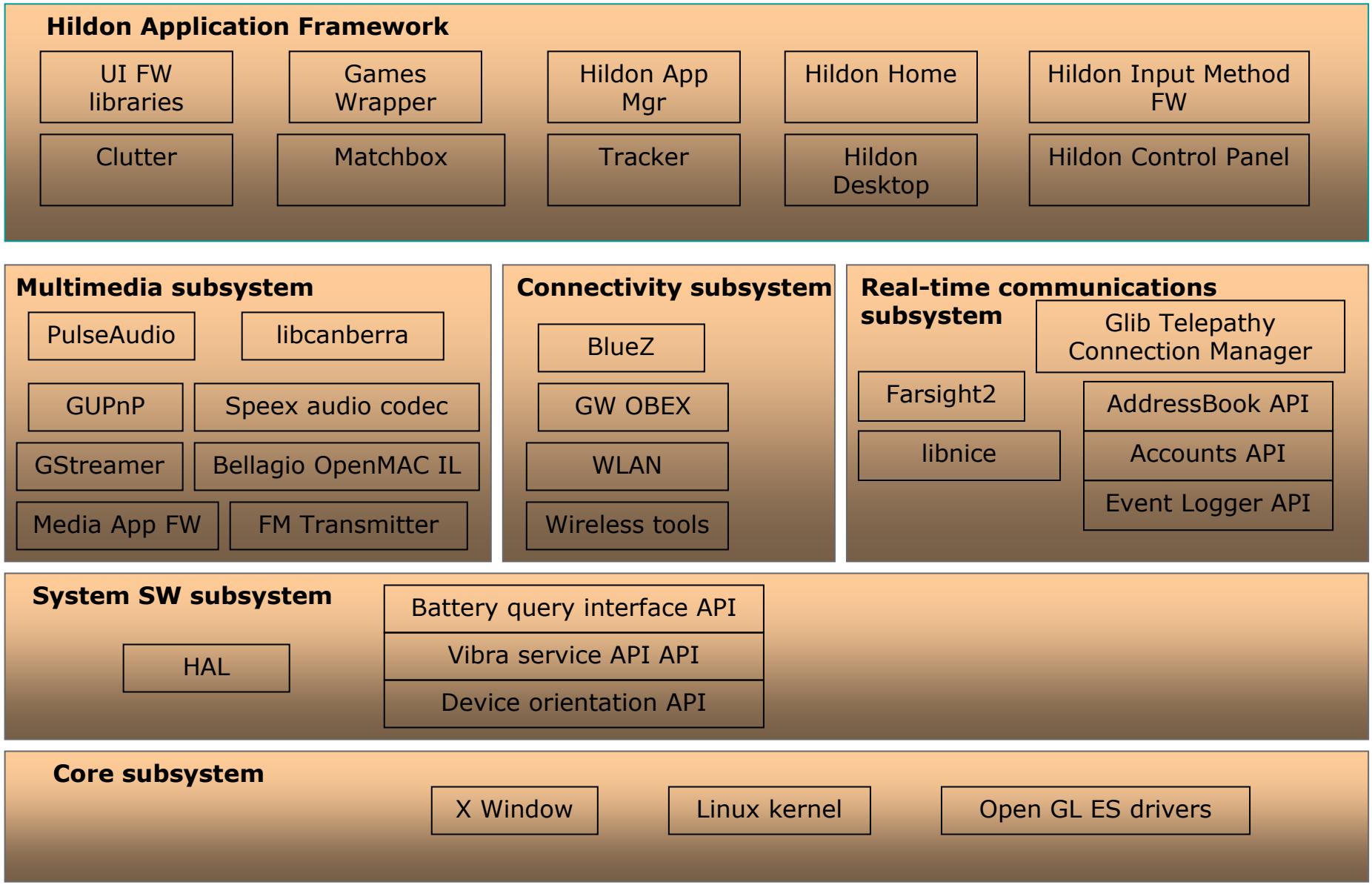
- Nokia 770 Internet Tablet
  - Based on Internet Tablet OS 2006 edition and maemo platform v2.2 – Gregale
- Nokia N800
  - Based on Internet Tablet OS 2007/2008 edition and maemo platform v3.2/v4.0 – Bora
  - GTK+ based UI framework
- Nokia N810
  - Based on Internet Tablet OS 2008 edition and maemo platform v4.0/v4.1 – Chinook, Diablo
  - Any UI framework running on the top of X11 (Qt libraries, Java)
- Nokia N900
  - Based on OS 2009 edition and maemo platform v5.0 – Fremantle
  - Qt 4.6 port under development

Mobile Computer Hardware

Nokia N900

TI OMAP 3 processor, Cortex A8 CPU
HSPA/3G support – online anywhere 3G connectivity
High definition camera support (integrated Image Signal Processor)
HW acceleration for OpenGL ES 2.0
XGA (1024 x 768) display with 16 M colors
High-speed USB 2.0

Maemo Software Architecture

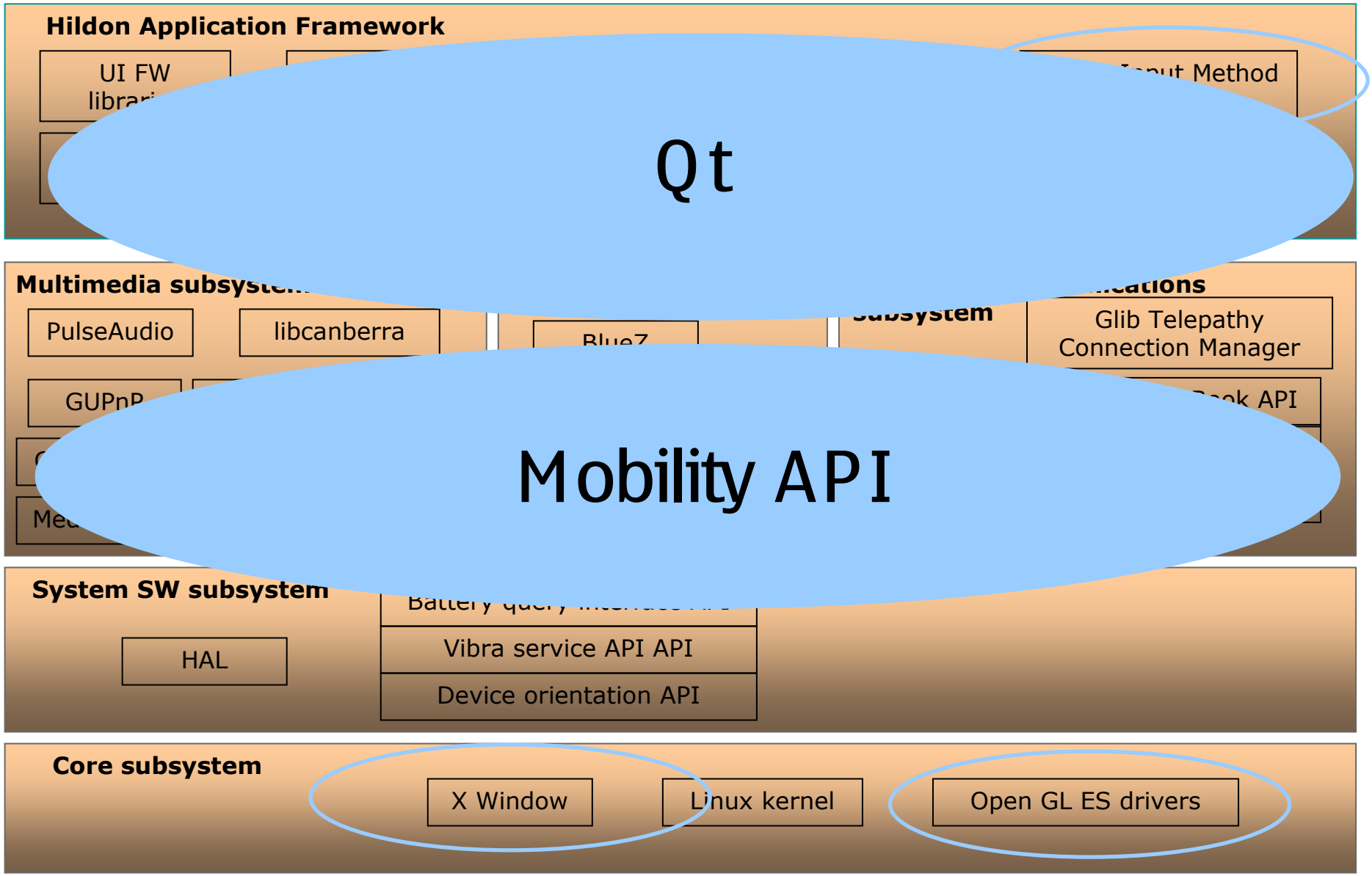


Development on Maemo

maemo™

- GTK
- Qt

Qt Integration (Qt Case)



Maemo Apps: Typical Source File Organization

- Common application file structure
  - src/ - source code
  - debian/ - files related to debian packaging
  - data/ - icons, .desktop file, D-BUS service file
  - po/ - localized files

Using Scratchbox

- Start it by typing the command
  - `scratchbox/login/`
- Configure targets
  - `sb-menu`
- Switch between targets
  - `sb-conf select FREMANTLE_X86`
  - `sb-conf select FREMANTLE_ARMEL`

- Otherwise work in a similar way like outside SB
- Single source concept
  - Single source – multiple targets

XServer

- With this you can emulate and test your apps on linux PC
- X server allows applications to create windows to which they can draw on the screen
- Applications are X server clients
- X server also provides key press and pointer events to applications
- Maemo applications require a pseudo X server to be installed
  - It acts as an X client to a real X server

- Xephyr X server is typically used with Maemo SDK
  - Kdrive-based X server capable of emulating 16-bit color depth for clients

XServer Initialization

- Outside Scratchbox, say
  - `Xephyr :2 -host-cursor -screen 800x480x16 -dpi 96 -ac -kb &`
  - To install Xephyr, say
    - `apt-get install xserver-xephyr`

- In Scratchbox, direct the display to window 2 by typing
  - `export DISPLAY=:2`

- Your maemo window does not look very nice yet…
  - Fonts, decorations, UI framework
  - Your application needs to be integrated with Maemo platform (managed by Maemo window manager)
  - Application must be killed with CTRL-c – clicking X does not kill application window yet

Tools: Scratchbox

- Cross-compilation in Linux is difficult – well is it actually?
  - Partly because of autotools – which should be helpful
- Some Linux distributors solve the problem by not supporting cross-compilation
- Scratchbox solves the problem by totally isolating host and target environments
  - Supports two separate targets: X86 and ARMEL
  - www.scratchbox.org
- Autotools-based build scripts can be run on Scratchbox without modification when building for the target

Hildon Application Framework Start

- Launch the application framework
  - `[sbox-FREMANTLE_X86: ~/Hw]: af-sb-init.sh start`
  - Hint! If you cannot launch application framework, the process may be running already. Give a command `af-sb-init.sh stop` in that case
- Run standalone shell (themes, fonts will work)
  - `[sbox-FREMANTLE_X86: ~/Hw]: run-standalone.sh ./helloworld`

Maemo Framework

- A few files are required to make your maemo application visible in the maemo framework
- Maemo .desktop file
  - Makes the application visible to Task Navigator
- D-BUS service file
  - Allows application launching and connects to D-BUS services (e.g. some events are received from D-BUS)
- Maemo initialization
  - Without this, application is killed soon after launch

Maemo .desktop File

- Should be named as [app name].desktop
- Copied to /usr/share/applications/hildon

```
[Desktop Entry]
Encoding=UTF-8
Version=1.0
Type=Application
Name=Hello World
Exec=/usr/bin/helloworld
Icon=HelloWorldPic
```

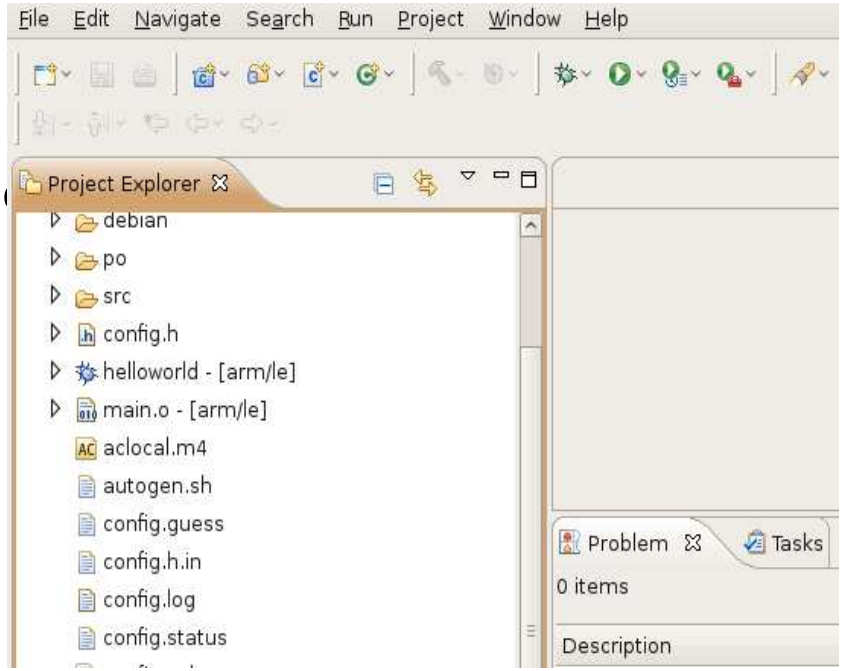
Control File

- Edit the maintainer field
- Define build dependencies
- Add depends info
- Add some package description
- Add Qt library dependency, if Qt used (*libqt4-gui*)

```
Source: HelloWorld
Priority: extra
Maintainer: Your Name <your.name@gmail.com>
Standards-Version: 3.7.2
Architecture: all
Depends: libqt4-gui
Description: This is a simple helloworld
   application
```

Lost Interest Already?

- How do I remember all this?
- Esbox IDE helps
  - Eclipse using Scratchbox
- Provides
  - Documentation
  - Buidling (and generation of all required files)
  - XServer initialization
  - Debugging
  - Debian package creati
  - Device installation



Building Qt Applications in Maemo

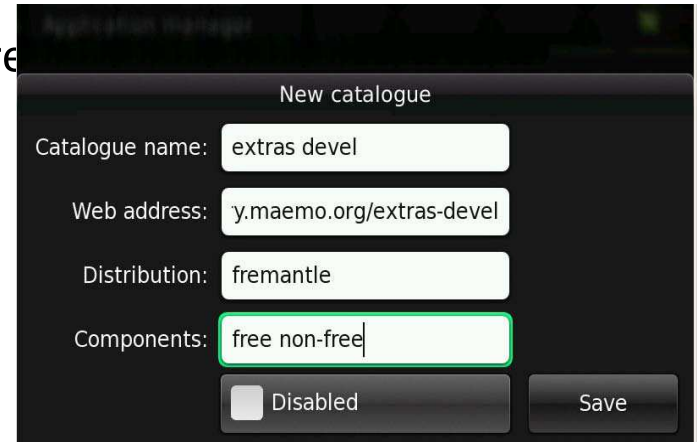
- Building in Qt way
  - `qmake -project`
    - Crate the .pro project file
  - `qmake`
    - Create Makefile with Qt add-ons (.moc files, .rsc filesetc.)
  - `make`
    - Build
    - Create Maemo application framework files (desktop file and D-Bus service file)

Installing Qt Libraries to a Device

- First, SSH and PC connectivity package are needed
- Look at the installation details in
  - http://pc-connectivity.garage.maemo.org/installation.html
  - This installation contains also the openssh-server

Installing Qt Libraries to a Device

- Open the **Application Manager** in the device
- Make sure Extras-devel is enabled
  - Go to **menu > *Application Catalogue***
  - If there is no Extras-devel catalogue in the list you can create one
  - Inside ***Applications Catalogue*** window press the ***New*** button and fill out the dialog box that appear with the following information:
    - Catalogue name: Extras-devel
    - Web address: http://repository.maemo.org/extras-devel
    - Distribution: fremantle
    - Components: free non-fre



Installing Qt Libraries to a Device

- Open an ssh session with the device
  - `PC_$> ssh root@DEVICE-IP`
- Install Qt binaries
  - `#apt-get install libqt4-gui`
- To install other binaries, run
  - `#apt-get install libqt4-webkit libqt4-opengl libqt4-dbus`
- Note! The Qt 4.5.3 packages are expected to be moved under Extras repository in near future.

Qt Application Deployment

- Move the application binary or Debian package to the device
- You may use SCP to transfer the binary file
  - `PC_$> scp filename root@DEVICE-IP:/home/user`
  - If you are a Windows user, read http://www.winscp.net/ to learn, how to get SCP
  - And run it
    - `PC_$> ssh root@DEVICE-IP`
    - `tablet_#> su - user`
    - `tablet_$> run-standalone.sh ./hello`
  - Obviously the application is not shown in the Task Switcher
- Or create a Debian package and install that
  - `$ssh root@DEVICE-IP`
  - `#dpkg -i filename.deb`

The task today

- Setup the Maemo SDK environment containing Qt
- Test your Qt application in the SDK (PC)
- Make a debian installation package
- (Install Qt to your Maemo device)
- Install and test your application on Maemo

- If you do your project for N810, then you use Maemo 4 SDK (Diablo)
- If you do your project for N900, then you use Maemo 5 SDK (Fremantle)