Forum Nokia

Qt UI Development

Andreas Jakl

Senior Technical Consultant
Forum Nokia

23 October, 2010
v3.2.0

Contents

- Ways of creating user interfaces with Qt
- Subclassing Widgets (Signals & Slots, continued)
- Dialogs
- Main Window
- Menus, Toolbars and Actions

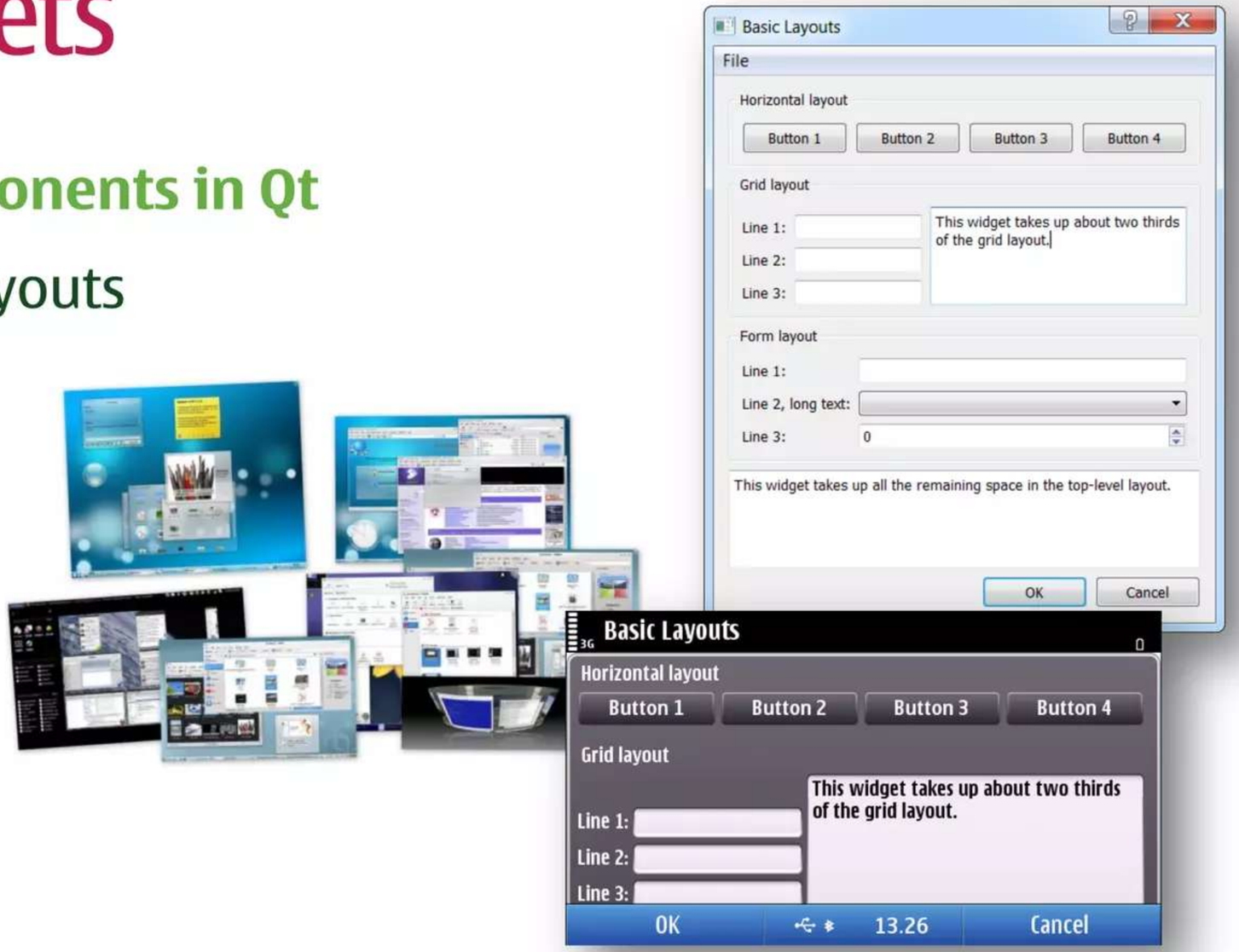
User Interfaces with Qt

Qt & Graphics

- **Available options**
  - Traditional widgets
  - Custom widgets and QPainter
  - QGraphicsView
  - QGLWidget
  - Qt Quick
  - QtWebKit

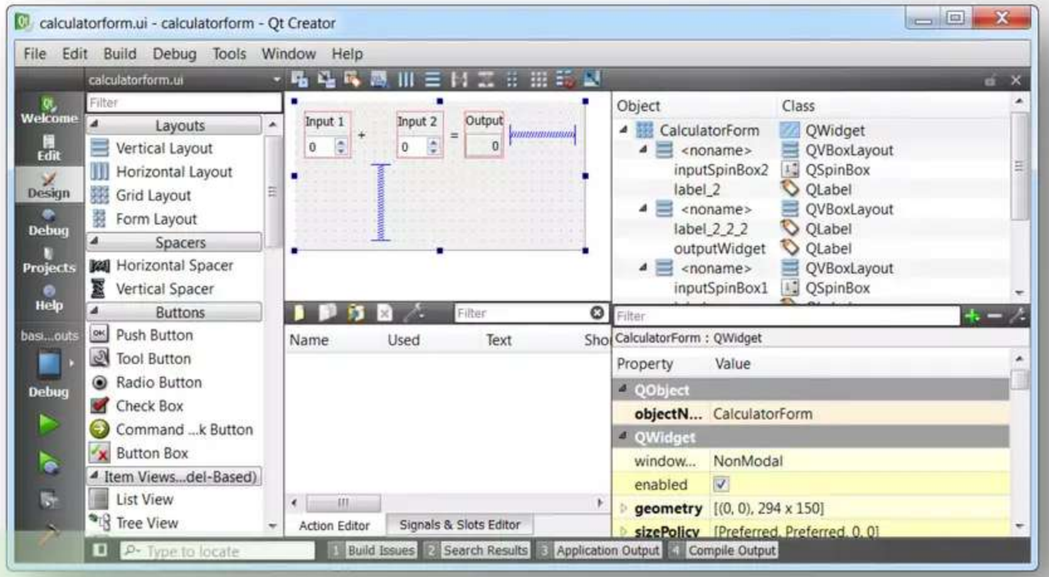
Traditional Qt Widgets

- **QWidget as base of UI components in Qt**
  - Window management, layouts
  - Use native UI design
  - Support style sheets
  - Ex.: QPushButton, QLabel, QLineEdit



Traditional Qt Widgets

- **UI Designer support**
  - Instant preview, drag & drop UI
  - Integrated even with translator tool

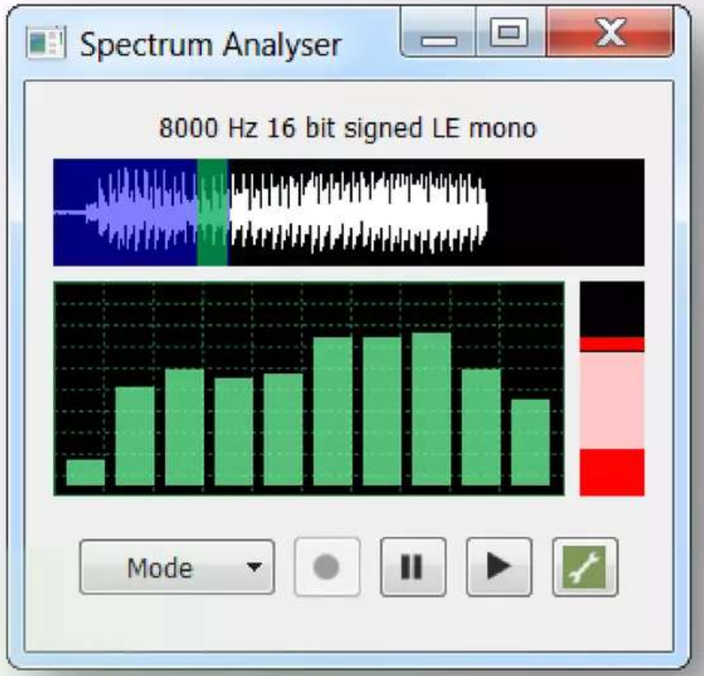


High level
Exactly the same code on all platforms

Less customization possible, UI effects difficult
Smaller UI, different paradigms

QWidget and QPainter

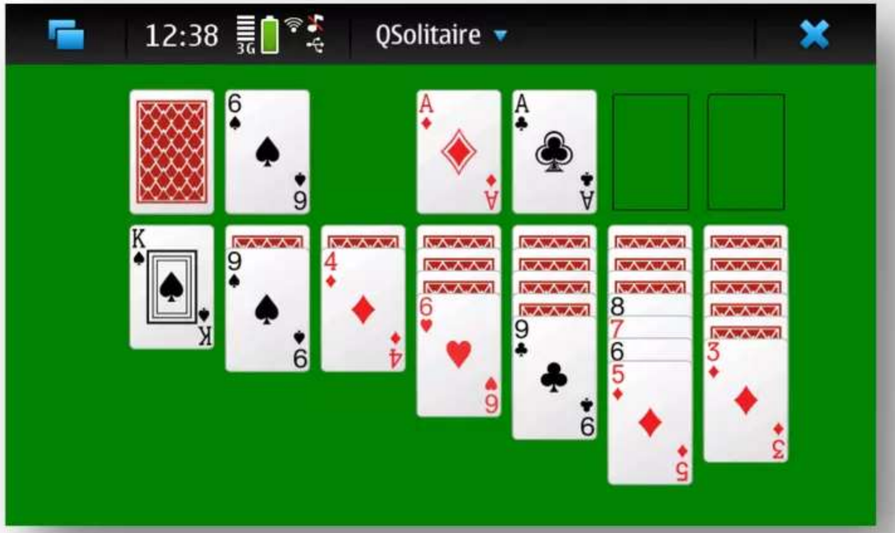
- **Draw manually using QPainter**
  - On QWidget surfaces
  - Flexible painting: text, gradients, polygons, bezier curves, transparencies, etc.



Lets you express your painting directly
Traditional

Makes you express your painting directly

QGraphicsView



- **Manage 2D scenes with a scene graph**
  - Can manage thousands of items with hierarchies
  - Collision detection, affine transformations, drag and drop support
  - Hardware acceleration and integration with QWidget possible

High level
Conceptually nice and flexible

No ready-made common UI components available yet

MeeGo Touch

- **UI Framework based on QGraphicsView**
  - Provides common UI components with theme support
  - Integrates transitions and effects

Great graphics effects, theme support
Easy to create smooth UIs
Already available from source repositories

Still in development

QGLWidget

- **Very low level**
  - Basic setup by Qt
  - You do the hard work

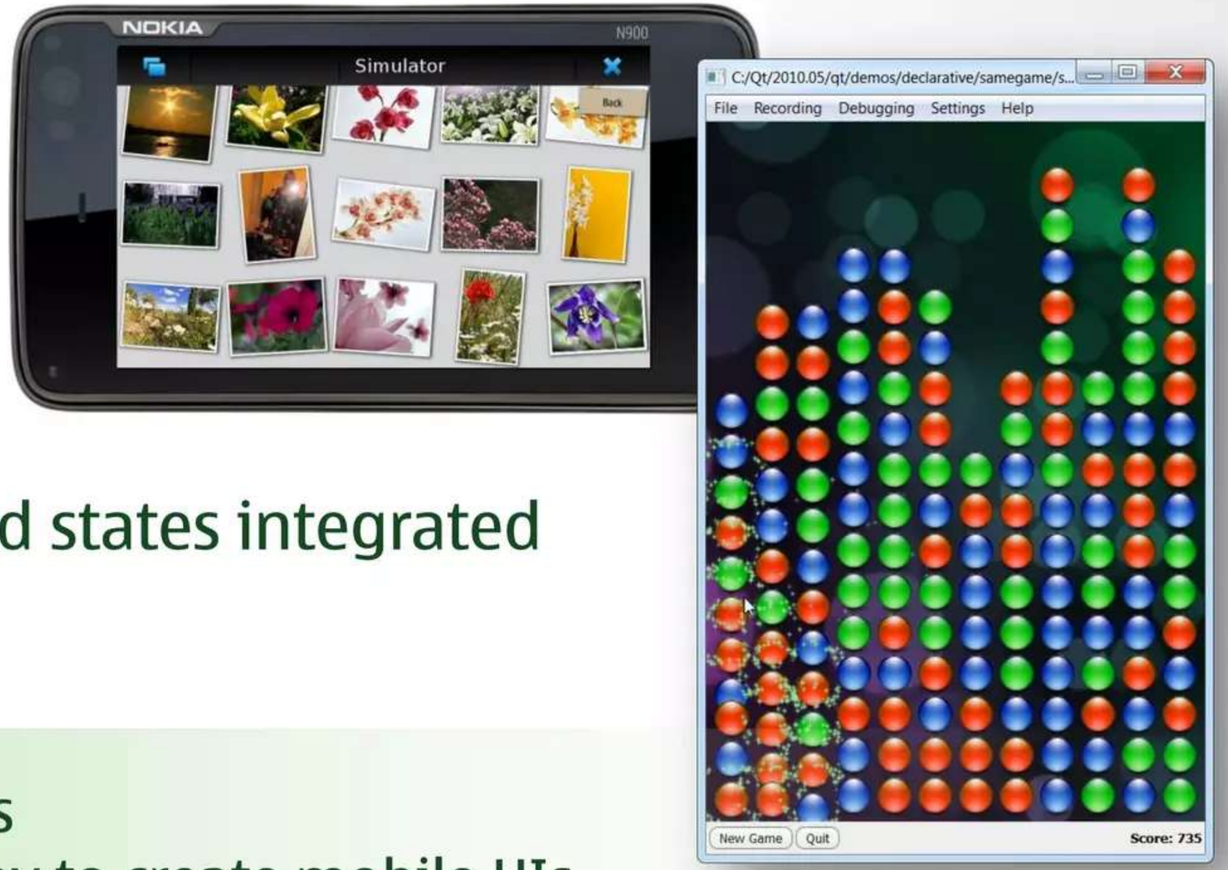
Angry Birds and Bounce by Rovio

Complete control over GL
Good if you have existing codebase & assets, cross-platform

OpenGL is hard work (but there's more Qt coming for that *)
No Qt OpenGL for Symbian integration in Qt 4.7.0

* http://doc.qt.nokia.com/qt3d-snapshot/index.html

Qt Quick



- **Several components**
  - QML language and JavaScript
  - Declarative syntax, animations and states integrated
  - Available in Qt 4.7+

Very easy to make slick, fluid UIs
Will soon be most important way to create mobile UIs

More tooling is on its way
Ready-made UI components on their way *

* Project page: http://bugreports.qt.nokia.com/browse/QTCOMPONENTS

QtWebKit

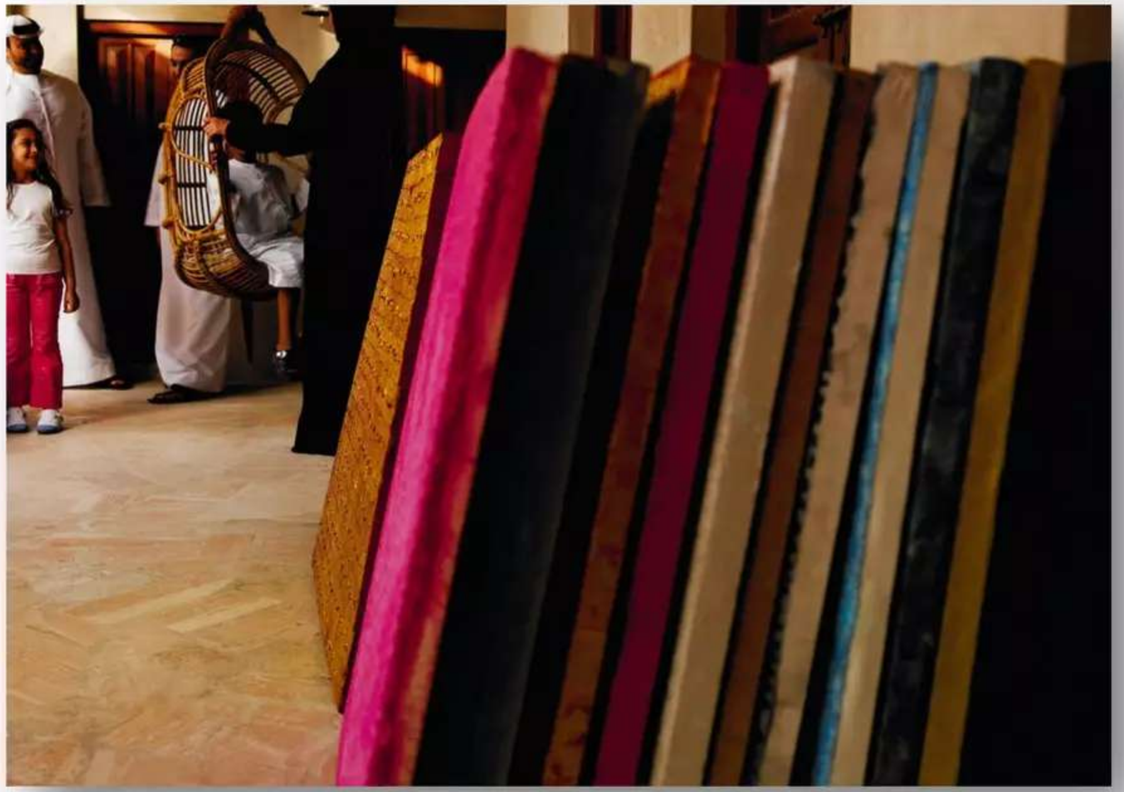
- **Open source browser engine**
  - Display HTML(5) and JavaScript content through a QWidget
  - Combine Qt C++ and web code if needed to get best of both worlds

Use your existing web skills to create the UI
Reuse web components (online help, etc.)

Complex UIs and interaction probably more difficult
Less performance than native code (but improving)

How to Choose?

- **Depends on**
  - Complexity of your UI
  - Existing code/assets
  - Experience with low level code
  - Which devices you're targeting
  - Your timeframe
- **Mix and match as appropriate!**



Subclassing Widgets

Forum Nokia

Signal and Slots, Continued

- **How to accomplish the following?**

```
QObject::connect(button, SIGNAL(clicked()),
                 label, SLOT(setText("new text")));
```

→ doesn't work that way, `clicked()`-signal doesn't give required number of arguments to `setText()`-slot.

Custom Widgets (Slots)

- **Commonly used: subclass widgets to extend functionality**
  - Class `MyWindow` is a widget (parent)
  - Also manages child widgets
- **Widget adds new slot**
  - Put text to display into this slot method instead of the connect statement

myWindow.h

```
#ifndef MYWINDOW_H
#define MYWINDOW_H

#include <QWidget>
#include <QVBoxLayout>
#include <QPushButton>
#include <QLabel>
#include <QObject>

class MyWindow : public QWidget
{
    Q_OBJECT
public:
    MyWindow(QWidget *parent = 0);

private:
    QLabel* label;
    QPushButton* button0;
    QPushButton* button1;
    QVBoxLayout* layout;

private slots:
    void setText();
};

#endif // MYWINDOW_H
```

myWindow.cpp

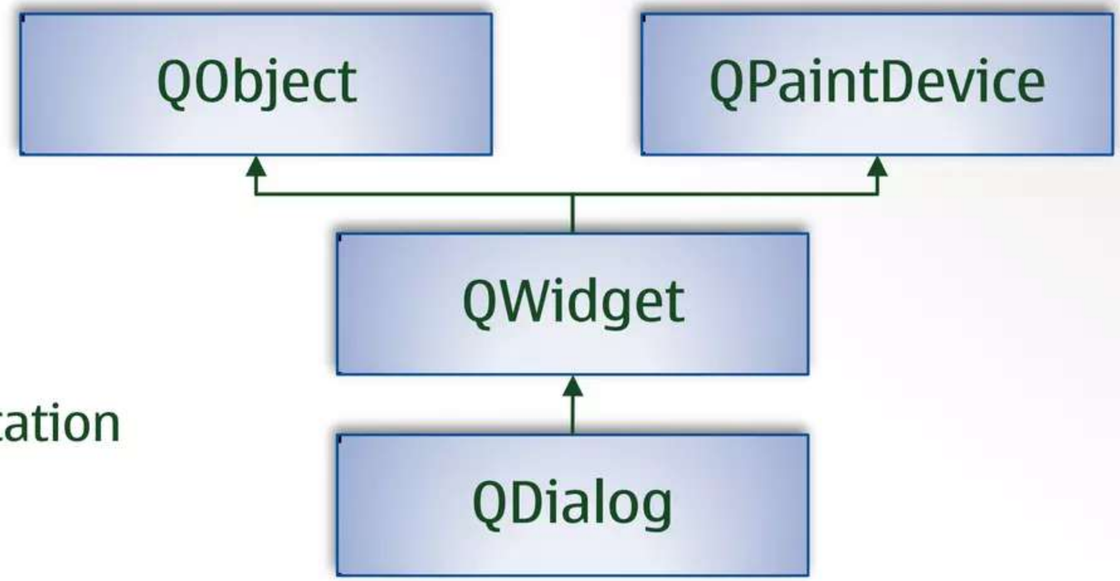
```
#include "MyWindow.h"

MyWindow::MyWindow(QWidget* parent)
    :
    QWidget(parent)
{
    label = new QLabel("old text");
    button0 = new QPushButton("Update labels");
    button1 = new QPushButton("Exit");

    layout = new QVBoxLayout(this);
    layout->addWidget(button0);
    layout->addWidget(button1);
    layout->addWidget(label);
    setLayout(layout);

    connect(button0, SIGNAL(clicked()),
            this, SLOT(setText()));
    connect(button1, SIGNAL(clicked()),
            this, SLOT(close()));
}

void MyWindow::setText()
{
    label->setText("new text");
}
```

```
#include <QtGui/QApplication>
#include "MyWindow.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MyWindow* window = new MyWindow();
    window->show();
    return a.exec();
}
```

main.cpp


Dialogs

Dialogs

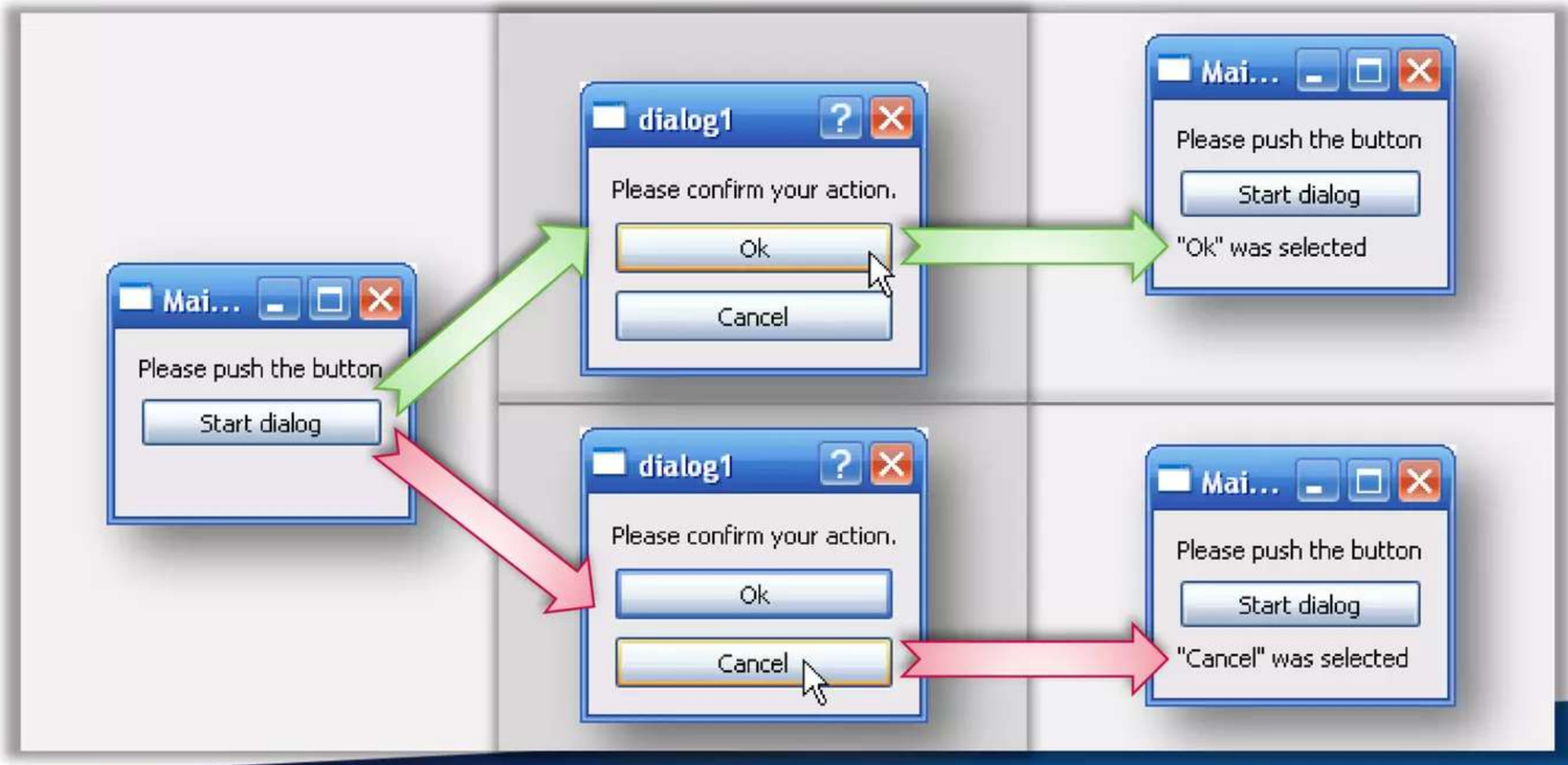


- **Dialog is:**
  - Top-level window
  - Used for short-term tasks, brief user communication
  - Can provide return value
- **Modal**
  - Dialog **blocks other application windows** (e.g., file open dialog)
  - Usually called with **exec()**, returns when dialog is closed
  - Call with show(): returns immediately, get results via signals
- **Modeless**
  - **Operates independently** from other windows (e.g., find & replace dialog)
  - Always called with show(): returns immediately

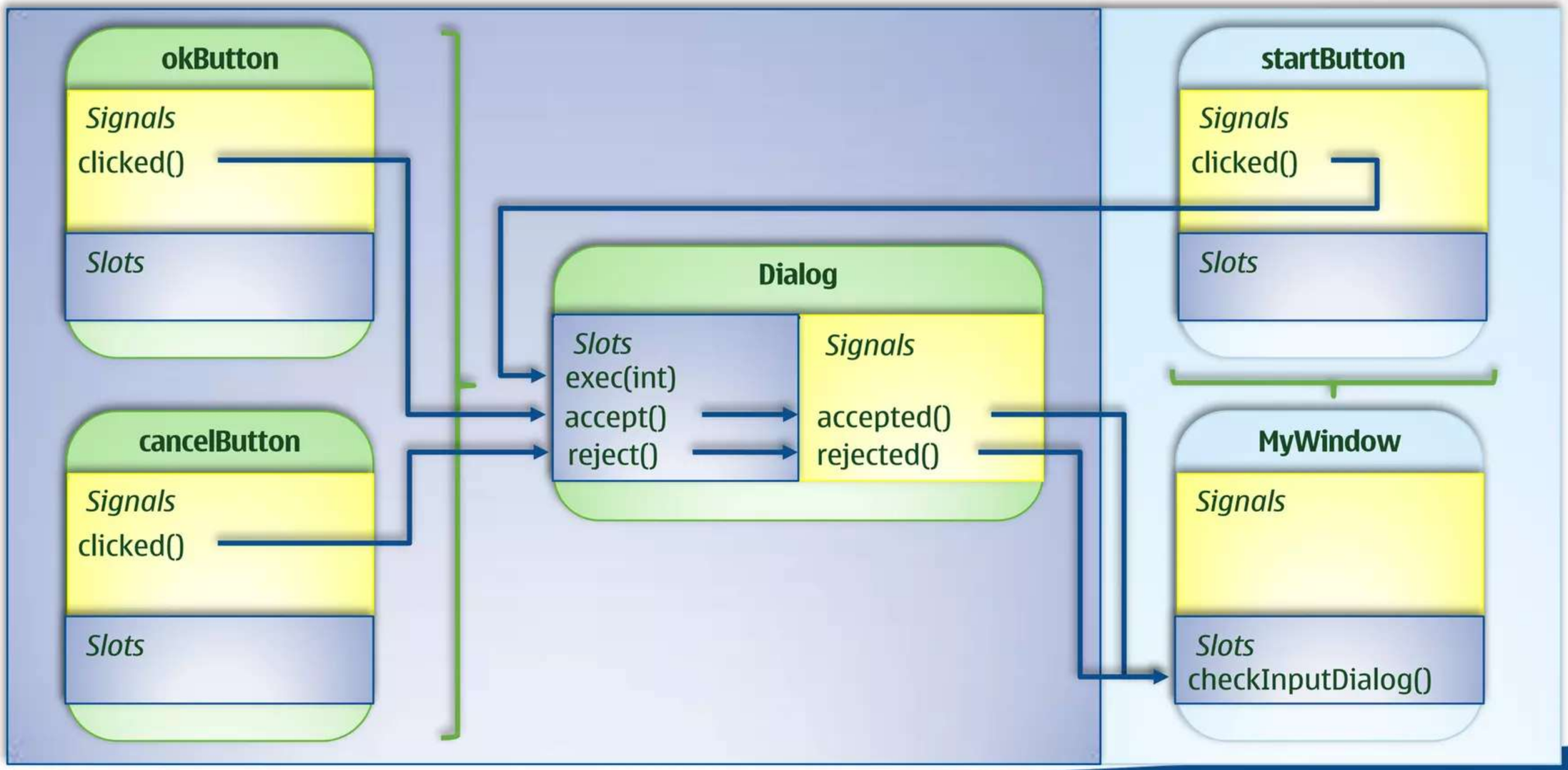
Custom Dialog

`clicked()` signal from button in main widget triggers dialog

Change label in main widget depending on user action selected in dialog



Signals & Slots Diagram



The Dialog

myDialog.h

```
#ifndef MYDIALOG_H
#define MYDIALOG_H

#include <QDialog>
#include <QPushButton>
#include <QVBoxLayout>
#include <QLabel>

class MyDialog : public QDialog
{
    Q_OBJECT
public:
    MyDialog();
};

#endif // MYDIALOG_H
```

myDialog.cpp

```
#include "mydialog.h"

MyDialog::MyDialog()
{
    setFixedSize(150, 100);
    QVBoxLayout* vbox = new QVBoxLayout();
    QLabel* label = new QLabel("Please confirm.");
    QPushButton* okButton = new QPushButton("Ok");
    QPushButton* cancelButton = new QPushButton("Cancel");

    // Set the ok button as default
    okButton->setDefault(true);
    vbox->addWidget(label);
    vbox->addWidget(okButton);
    vbox->addWidget(cancelButton);
    setLayout(vbox);

    // Connect the buttons to slots defined by QDialog
    connect(okButton, SIGNAL(clicked()),
            this, SLOT(accept()));
    connect(cancelButton, SIGNAL(clicked()),
            this, SLOT(reject()));
}
```

QDialog Base Class

Slot	Description
virtual void **accept**()	Hides the dialog and sets the result code to Accepted (1).
virtual void **reject**()	Hides the dialog and sets the result code to Rejected (0).
virtual void **done**(int r)	Closes the dialog and sets return value to r. If the dialog is started via exec(), done() causes the event loop to finish, and exec() to return r. Deletes the dialog if Qt::WA_DeleteOnClose is set.
int **exec**()	Shows the dialog as a modal dialog, blocking until the user closes it. Returns dialog return value (DialogCode) like Accepted or Rejected.

Signal	Description
void **accepted**()	Emitted when dialog has been accepted through calling accept() or done() with the argument QDialog::Accepted
void **rejected**()	Emitted when dialog has been rejected through calling reject() or done() with the argument QDialog::Rejected
void **finished**(int result)	Emitted when the dialog's result code has been set (setResult()) and either accept(), reject() or done() is called.

The Widget

mywidget.h

```
[...]

class MyWidget : public QWidget
{
    Q_OBJECT

public:
    MyWidget(QWidget *parent = 0);
    ~MyWidget();

private slots:
    void checkInputDialog();

private:
    QPushButton* startButton;
    QLabel* instructionsLabel;
    QLabel* resultLabel;
    QVBoxLayout* layout;
    MyDialog* dialog;

};

#endif // MYWIDGET_H
```

main.cpp is similar to the previous example

mywidget.cpp

```
#include "mywidget.h"

MyWidget::MyWidget(QWidget *parent) : QWidget(parent) {
    setWindowTitle("Main Window - Dialog Example");
    startButton = new QPushButton("Start dialog");
    instructionsLabel = new QLabel("Please push the button");
    resultLabel = new QLabel();

    layout = new QVBoxLayout(this);
    layout->addWidget(instructionsLabel);
    layout->addWidget(startButton);
    layout->addWidget(resultLabel);

    dialog = new MyDialog();

    connect(startButton, SIGNAL(clicked()),
            dialog, SLOT(exec()));
    connect(dialog, SIGNAL(accepted()),
            this, SLOT(checkInputDialog()));
    connect(dialog, SIGNAL(rejected()),
            this, SLOT(checkInputDialog()));
}

void MyWidget::checkInputDialog() {
    int res = dialog->result(); // Gets result (Accepted/Rejected)
    if (res == QDialog::Accepted) {
        resultLabel->setText("\"Ok\" was selected");
    } else if (res == QDialog::Rejected) {
        resultLabel->setText("\"Cancel\" was selected");
    }
}
```

Custom Return Values

mydialog.h

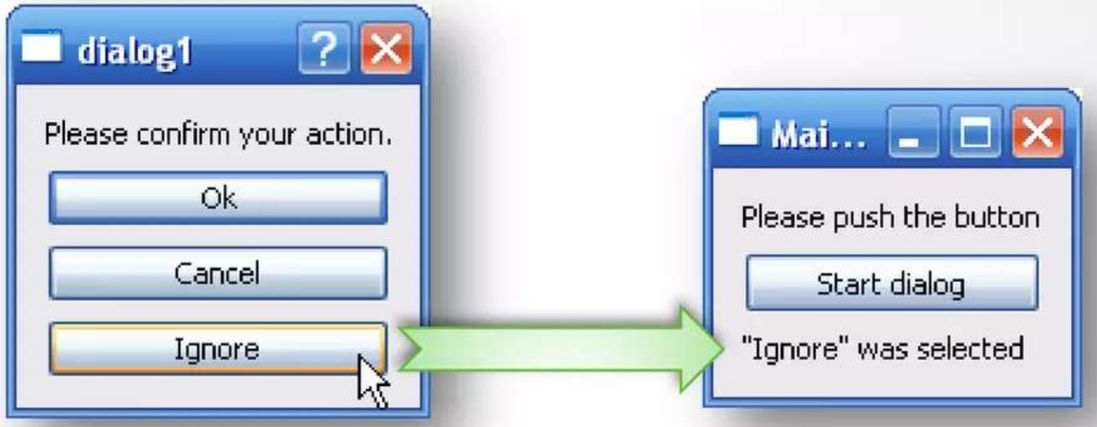
```
[...]
private slots:
    void setResult();
[...]
```

mydialog.cpp

```
[...]
    connect(ignoreButton, SIGNAL(clicked()),
            this, SLOT(setResult()));
[...]
void MyDialog::setResult()
{
    int result = 99;
    emit done(result);
}
```

mywidget.h

```
[...]
private slots:
    void checkInputDialog();
    void checkInputDialog(int);
[...]
```

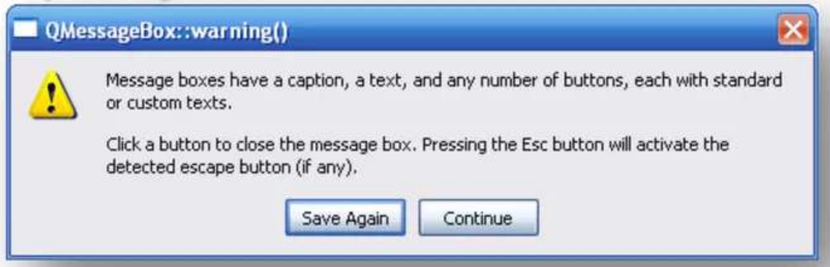


mywidget.cpp

```
[...]
    connect(dialog, SIGNAL(finished(int)),
            this, SLOT(checkInputDialog(int)));
[...]
void MyWidget::checkInputDialog(int res)
{
    if (res == 99)
    {
        resultLabel->setText("\"Ignore\" was selected");
    }
}
```

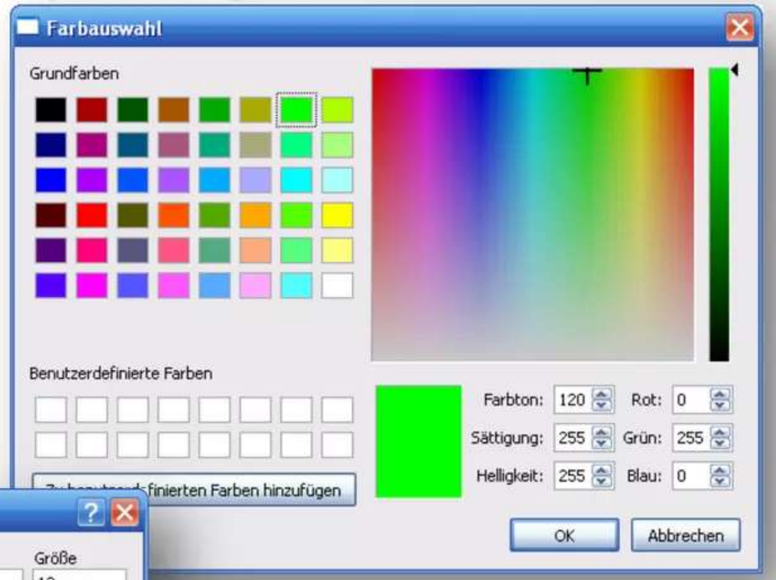
Predefined Dialogs

QMessageBox

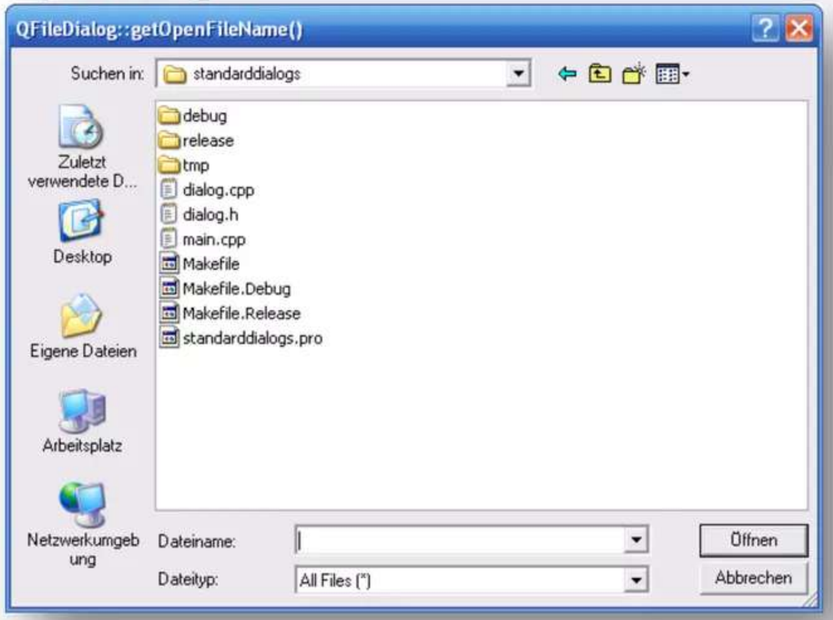


QInputDialog

QColorDialog



QFileDialog



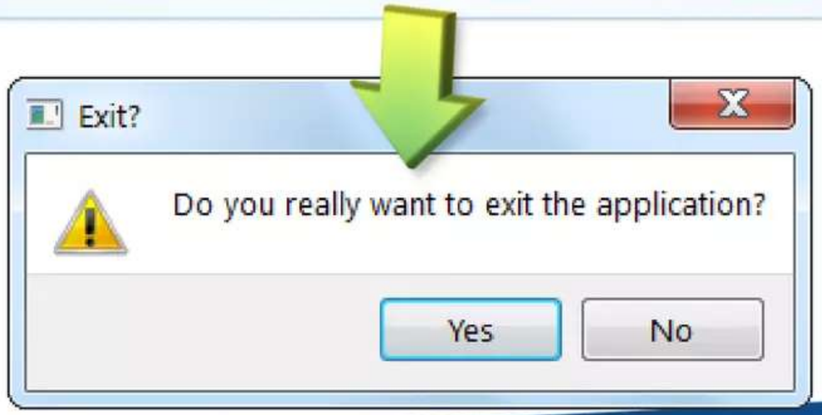
QFontDialog

QErrorMessage

Predefined Dialogs

- **Example: Message box**
  - Modal dialog
  - User selection → return value

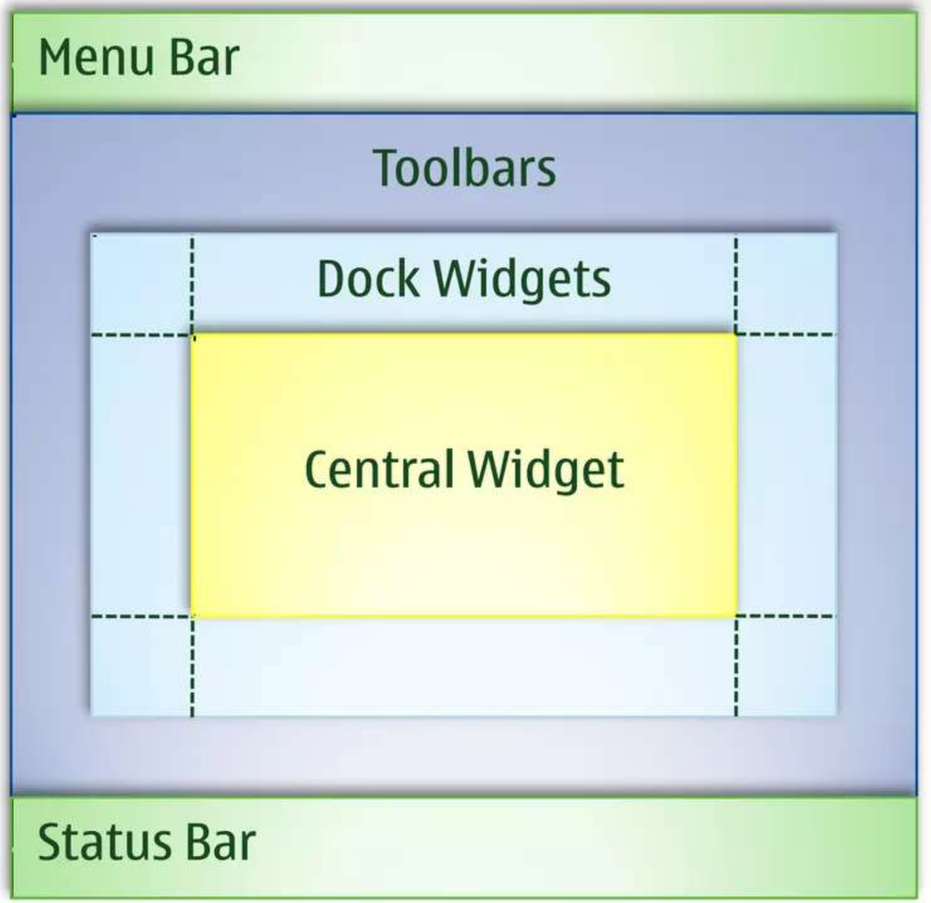
```
int ret = QMessageBox::warning( this, "Exit?",
          "Do you really want to exit the application?",
          QMessageBox::Yes | QMessageBox::No );
```



Main Window, Menu, Action

Main Window

- **Provides main application window**
  - **Pre-defined layout** for standard components
  - **Central widget** must be defined, others optional
  - **Subclass** to create your own implementation
- **Differences?**
  - Possible with dialog / widgets as well
  - **But:** more comfortable, consistent and efficient



Example - QMainWindow

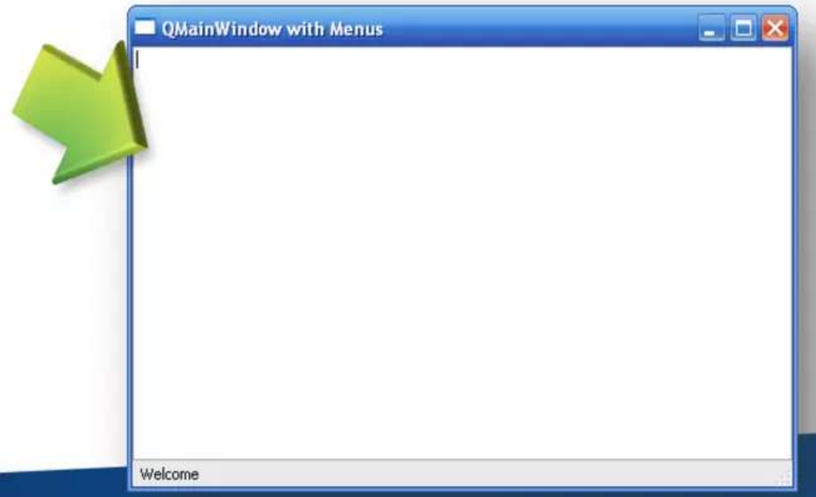
mainwindow.h

```
[...]

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = 0,
               Qt::WFlags flags = 0);
    ~MainWindow();

private:
    QTextEdit* editor;

[...]
};

#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent, Qt::WFlags flags)
    : QMainWindow(parent, flags)
{
    editor = new QTextEdit();
    setMinimumSize(160, 160);
    resize(480, 320);
    setCentralWidget(editor);
    setWindowTitle("QMainWindow with Menus");

    QString message = "Welcome";
    statusBar()->showMessage(message);
}
```

main.cpp is similar to the previous example



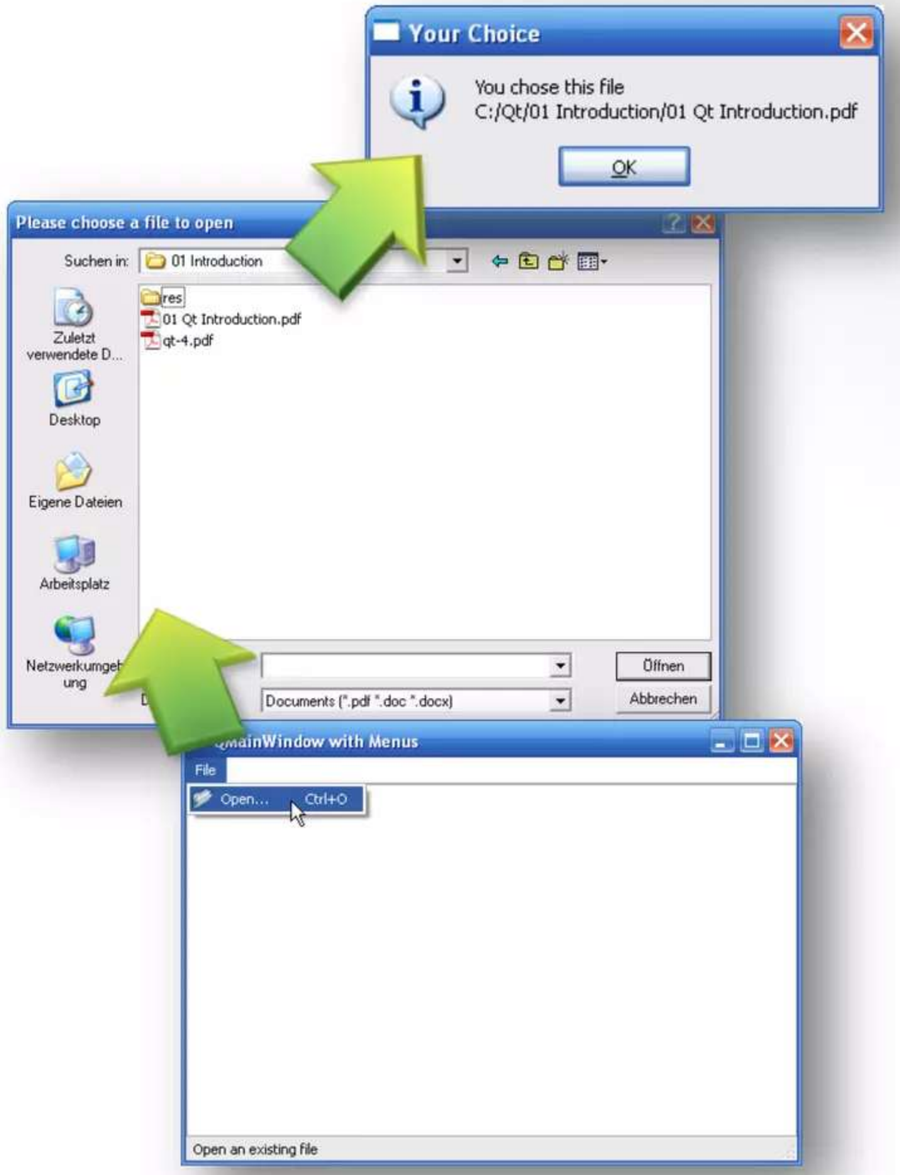
Action

Image by Anna Cervova (Public Domain)

- **Represent abstract user interface action**
  - Define once, use in multiple components
  - Inserted into widgets
    - Menus (can create actions implicitly)
    - Toolbar buttons
    - Keyboard shortcuts
- **Stores information about**
  - Icons
  - Text
  - Keyboard shortcut
  - Status text
  - “What’s This?” text
  - Tooltip

Menu Bar

- **QMenu provides:**
  - a menu widget for menu bars, context menus and other popup menus
- **Supports:**
  - Triggered Items
  - Separators
  - Submenus
  - Tear-off menus
- **QMenuBar automatically created by QMainWindow**
- **QMenu(s) contains individual menu items (→ Actions)**



Example – QAction

mainwindow.h

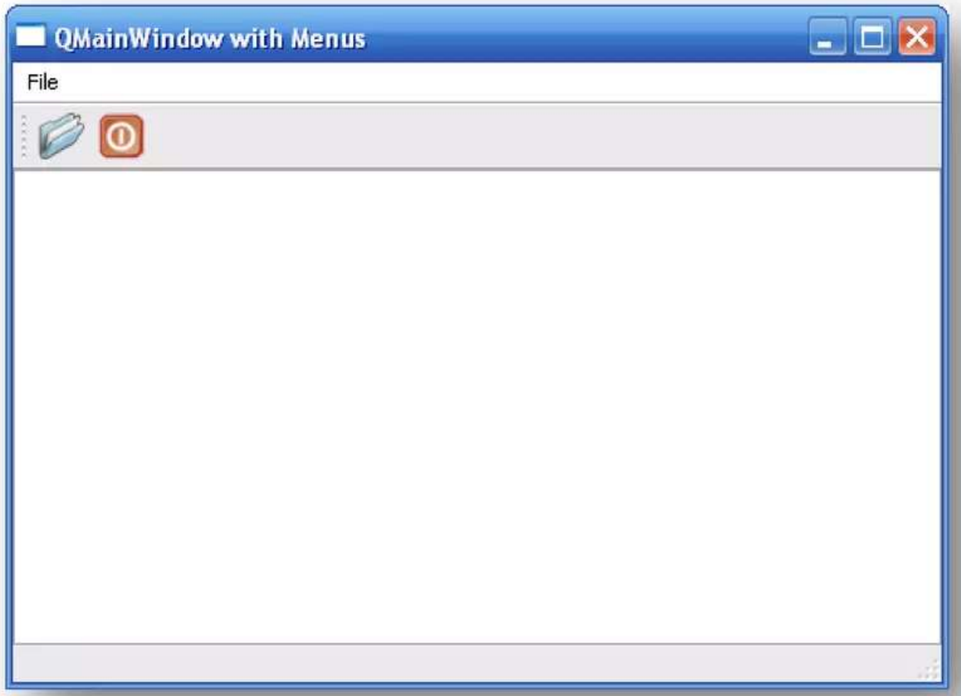
```
[...]

class MainWindow
    : public QMainWindow
{
    Q_OBJECT
[...]

private slots:
    void openFile();

private:
    QMenu *fileMenu;
    QAction *openAct;
};
```

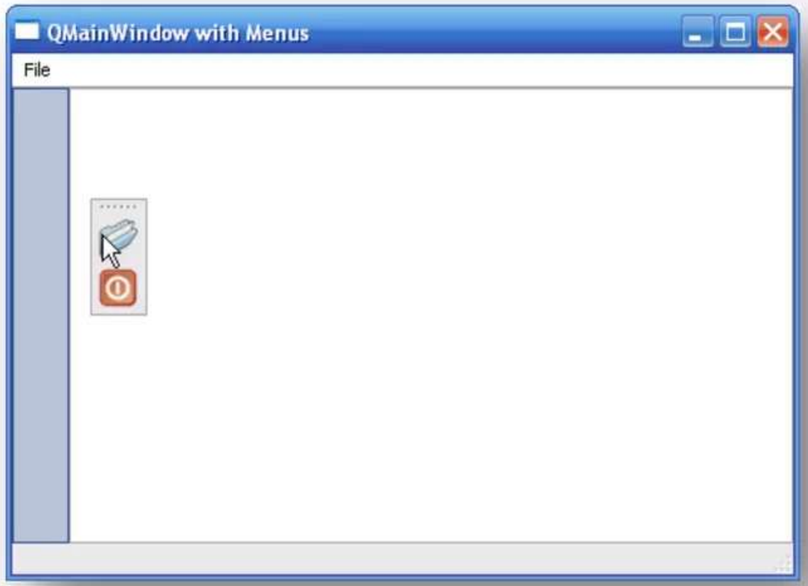
mainwindow.cpp

```
[...]
    // Create a new "Open" action with an icon, keyboard shortcut and
    // info-text for the status bar.
    openAct = new QAction("&Open...", this);
    openAct->setIcon(QIcon("images/open.png"));
    openAct->setShortcut(tr("Ctrl+O"));
    openAct->setStatusTip(tr("Open an existing file"));
    connect(openAct, SIGNAL(triggered()), this, SLOT(openFile()));

    // Add the action to the menu
    fileMenu = menuBar()->addMenu(tr("&File"));
    fileMenu->addAction(openAct);

[...]

void MainWindow::openFile()
{
    // Define two filter options - one for documents, one for all files
    // The filter mask is automatically parsed, ";;" separates lines
    QString file = QFileDialog::getOpenFileName(this,
        "Please choose a file to open", QDir::homePath(),
        "Documents (*.pdf *.doc *.docx);;All files (*.*)");

    if (!file.isNull())
    {
        QString info("You chose this file\n");
        info.append(file);
        QMessageBox::information(this, "Your Choice", info, QMessageBox::Ok);
    }
}
```

Toolbar

- Same actions as for menu can be used for toolbar
- Default automatically enables drag & drop





Example – QToolBar

mainwindow.h

```
[...]

class MainWindow
    : public QMainWindow
{
    Q_OBJECT
[...]

private:
    QToolBar *toolFile;
};
```

mainwindow.cpp

```
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent, Qt::WFlags flags)
    : QMainWindow(parent, flags) {
[...]
    // Open action
    openAct = new QAction("&Open...", this);
    openAct->setIcon(QIcon("images/open.png"));
    openAct->setShortcut(tr("Ctrl+O"));
    openAct->setStatusTip(tr("Open an existing file"));
    connect(openAct, SIGNAL(triggered()), this, SLOT(openFile()));

    // Exit action
    exitAct = new QAction("E&xit", this);
    exitAct->setIcon(QIcon("images/exit.png"));
    exitAct->setShortcut(tr("Ctrl+Q"));
    exitAct->setStatusTip(tr("Exit the application"));
    connect(exitAct, SIGNAL(triggered()), this, SLOT(close()));

    // Create the file menu
    fileMenu = menuBar()->addMenu(tr("&File"));
    fileMenu->addAction(openAct);
    fileMenu->addSeparator();
    fileMenu->addAction(exitAct);

    // Add the actions to the toolbar
    toolFile = addToolBar("File");
    toolFile->addAction(openAct);
    toolFile->addAction(exitAct);
[...]
}
```

Forum Nokia

Thank You.