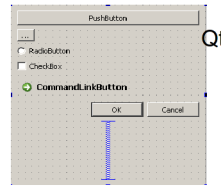
Qt Fundamentals: Widgets and Layouts

Click to add text

Common Widgets: Buttons

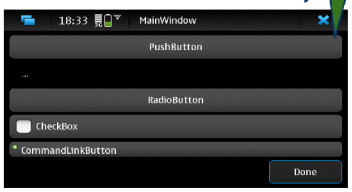
- All buttons inherit the QAbstractButton base class.
- Signals
  - clicked() - emitted when the button is clicked (button released)
  - toggled(bool) - emitted when the check state of the button is changed
- Properties
  - `checkable` - true if the button can be checked. Make a push button toggle
  - `checked` - true when the button is checked
  - text - the text of the button
  - icon - an icon on the button (can be displayed together with text)

Common Widgets: Buttons

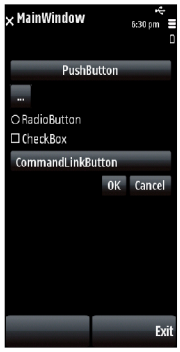


Qt Designer

Different look and feel

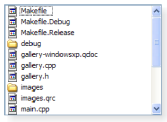


Nokia N900



Nokia 5800

Common Widgets: Item Widgets



- `QListWidget` is used to show a list of items
- Adding items
  - `addItem(QString)` - appends an item to the end of the list
  - insertItem(int row, QString) - inserts an item at the specified row
- Selection
  - `selectedItems` - returns a list of `QListWidgetItem`'s, use `QListWidgetItem::text()` to get the text
- Signals
  - `itemSelectionChanged` - emitted when the selection is changed

- `QComboBox` is another way (more compact) to show a list of strings.

Common Widgets: Containers

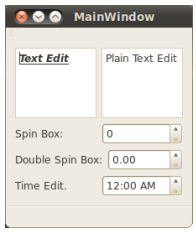
- Container widgets are used to give structure to the user interface
- They can be consider passive
- A plain QWidget can be used as a container
- **Designer:** Place widgets in the container and apply a layout to the container
- **Code:** Create a layout for the container and add the widgets to the layout

```
QGroupBox *box = new QGroupBox();
QVBoxLayout *layout = new QVBoxLayout(box);
layout->addWidget(...);
```

Common Widgets: Input Widgets

- There is a large choice of widgets for editing.
- Value input properties
  - value - current value
  - maximum - maximum value
  - minimum - minimum value

- Text widgets properties
  - readOnly
  - etc.



Common Widgets: Display Widgets

- The QLabel can be used to both display text or pictures
- Access through:
  - `setText()`
  - `setPixmap()`
- Input widgets made read-only also serve as display widgets

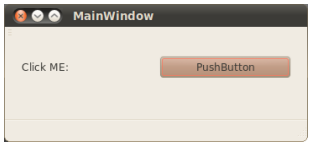

```
ui->forImage->setPixmap(QPixmap(":/qtlogo.jpg"));
ui->forImage->setScaledContents(true);
ui->forImage->setSizePolicy(QSizePolicy::Ignored, QSizePolicy::Ignored);
```


Size Policies

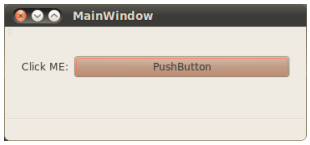
- Layouts manage size negotiation between the widget and the layout
- Layouts are used to bring structure
  - horizontal and vertical boxes (`QHBoxLayout`, `QVBoxLayout`)
  - grid (`QGridLayout`)
- Widgets supply
  - size policies for each direction
  - minimum and maximum sizes

Size Policies

Setting the horizontal size policy to expanding




```
ui->pushButton->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
```



Size Policies

- Each widget has a size hint that is combined with a policy for each direction

- `Fixed` - the size hint specifies the size of the widget
- `Minimum` - the size hint specifies the smallest size of the widget
- `Maximum` - the size hint specifies the largest size of the widget
- `Preferred` - the size hint specified is preferred, but not required
- `Expanding` - as Preferred, but wants to grow
- `MinimumExpanding` - as Minimum, but wants to grow
- `Ignored` - the size hint is ignored, widget gets as much space as possible

Size Policies

- Each widget has a size hint that is combined with a policy for each direction

- `Fixed` – fixed to size hint
- `Minimum` – can grow
- `Maximum` – can shrink
- `Preferred` – can grow, can shrink
- `Expanding` – can grow, can shrink, wants to grow
- `MinimumExpanding` – can grow, wants to grow
- `Ignored` – the size hint is ignored, can grow, can shrink

More on sizes

- Widgets sizes can be further controlled using the minimum and maximum sizes

```
ui->pushButton->setMinimumSize(100,150);
ui->pushButton->setMaximumSize(150, 200);
```

Top-level Windows

- Widgets without a parent widget automatically become a window

- `QWidget` - a plain window, usually non-modal
- `QDialog` - a dialog, usually expecting a result such as Ok, Cancel, etc.
- `QMainWindow` - an application window with menus, toolbars, statusbar, etc.

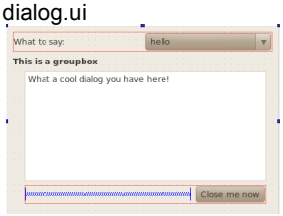
- `QDialog` and `QMainWindow` inherit `QWidget`

Using QDialog

- Dialogs may be used for settings, query, information etc.
- Dialogs inherit from `QDialog`
- As always we can either use Qt Designer to create our user interface or writing it manually in code.

Using QDialog - the code

dialog.ui



dialog.h

```
#ifndef DIALOG_H
#define DIALOG_H

#include <QtGui>
#include "ui_dialog.h"

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

private:
    Ui::Dialog *ui;
};

#endif // DIALOG_H
```

Using it

```
TestDialog *d = new TestDialog(this);
d->show();
```

```
#include "dialog.h"

TestDialog::TestDialog(QWidget *parent)
    : QDialog(parent), ui(new Ui::Dialog)
{
    ui->setupUi(this);

}
```

dialog.cpp

Ready made dialogs

- `QFileDialog` - the `QFileDialog` class provides a dialog that allow users to select files or directories.
- `QFontDialog` - the QFontDialog class provides a dialog widget for selecting a font
- `QColorDialog` - the QColorDialog class provides a dialog widget for specifying colors.
- `QInputDialog` - the QInputDialog class provides a simple convenience dialog to get a single value from the user.
- `QProgressDialog` - the QProgressDialog class provides feedback on the progress of a slow operation
- and more

Message boxes

The `QMessageBox` class provides a modal dialog for informing the user or for asking the user a question and receiving an answer.

- Easy to use through the static functions:

Static functions are available for creating information(), question(), warning(), and critical() message boxes.

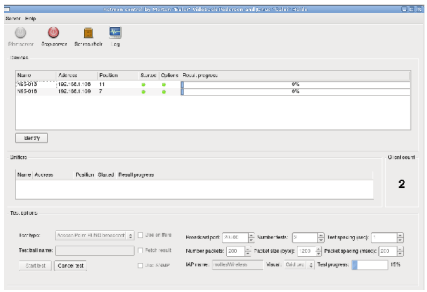
```
int ret = QMessageBox::warning(this, tr("My Application"),
                               tr("The document has been modified.\n"
                                  "Do you want to save your changes?"),
                               QMessageBox::Save | QMessageBox::Discard
                               | QMessageBox::Cancel,
                               QMessageBox::Save);
```

Note, one some platforms they all look the same :)

Using QMainWindow

A `QMainWindow` is the document windows of typical desktop applications

- Menus
- Toolbar
- Statusbar
- Docks
- Central widget



Introducing QAction

A QAction can be inserted into user interface components like toolbars and menus (but also used for keyboard shortcuts).

Lets see one in action:

```
openAct = new QAction(QIcon(":/images/open.png"), tr("&Open..."), this);
openAct->setShortcuts(QKeySequence::Open);
openAct->setStatusTip(tr("Open an existing file"));
connect(openAct, SIGNAL(triggered()), this, SLOT(open()));

fileMenu->addAction(openAct);
fileToolBar->addAction(openAct);
```