Qt Essentials - Widgets Module

Qt Essentials - Training Course

Produced by Nokia, Qt Development Frameworks

Material based on Qt 4.7, created on December 15, 2010

http://qt.nokia.com

Module: Widgets

- Common Widgets
- Layout Management
- Guidelines for Custom Widgets

Module Objectives

- **Common Widgets**
  - Text widgets
  - Value based widgets
  - Organizer widgets
  - Item based widgtes
- **Layout Management**
  - Geometry management
  - Advantages of layout managers
  - Qt's layout managers
  - Size policies
- **Custom Widgets**
  - Rules for creating own widgets

Module: Widgets

- Common Widgets
- Layout Management
- Guidelines for Custom Widgets

Text Widgets

- **QLabel**

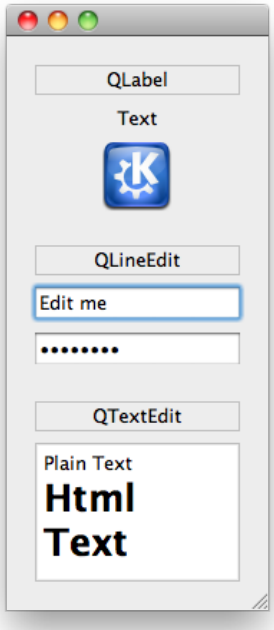
```
label = new QLabel("Text", parent);
```

  - `setPixmap( pixmap )` - as content
- **QLineEdit**

```
line = new QLineEdit(parent);
line->setText("Edit me");
line->setEchoMode(QLineEdit::Password);
connect(line, SIGNAL(textChanged(QString)) ...
connect(line, SIGNAL(editingFinished()) ...
```

  - `setInputMask( mask )` - See Input Mask Documentation
  - `setValidator( validator )` - See Validator Documentation
- **QTextEdit**

```
edit = new QTextEdit(parent);
edit->setPlainText("Plain Text");
edit->append("<h1>Html Text</h1>");
connect(edit, SIGNAL(textChanged(QString)) ...
```



Button Widgets

- **QAbstractButton**
  - Abstract base class of buttons
- **QPushButton**

```
button = new QPushButton("Push Me", parent);
button->setIcon(QIcon("images/icon.png"));
connect(button, SIGNAL(clicked()) ...
```

  - `setCheckable(bool)` - toggle button
- **QRadioButton**

```
radio = new QRadionButton("Option 1", parent);
```

- **QCheckBox**

```
check = new QCheckBox("Choice 1", parent);
```

- **QButtonGroup** - non-visual button manager

```
group = new QButtonGroup(parent);
group->addButton(button); // add more buttons
group->setExclusive(true);
connect(group, SIGNAL(buttonClicked(QAbstractButton*)) ...
```

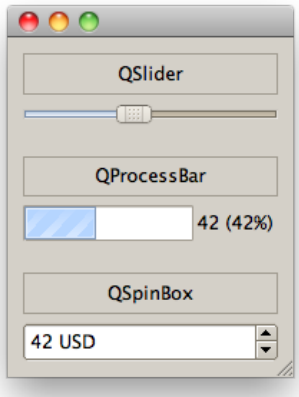
Value Widgets

- **QSlider**

```
slider = new QSlider(Qt::Horizontal, parent);
slider->setRange(0, 99);
slider->setValue(42);
connect(slider, SIGNAL(valueChanged(int)) ...
```

- **QProgressBar**

```
progress = new QProgressBar(parent);
progress->setRange(0, 99);
progress->setValue(42);
// format: %v for value; %p for percentage
progress->setFormat("%v (%p%)");
```



- **QSpinBox**

```
spin = new QSpinBox(parent);
spin->setRange(0, 99);
spin->setValue(42);
spin->setSuffix(" USD");
connect(spin, SIGNAL(valueChanged(int)) ...
```

Organizer Widgets

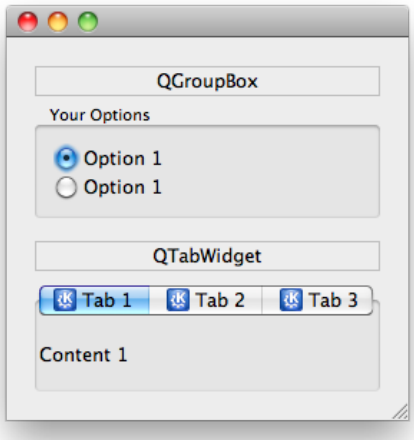
- **QGroupBox**

```
box = new QGroupBox("Your Options", parent);
// ... set layout and add widgets
```

  - `setCheckable( bool )` - checkbox in title
- **QTabWidget**

```
tab = new QTabWidget(parent);
tab->addWidget(widget, icon, "Tab 1");
connect(tab, SIGNAL(currentChanged(int)) ...
```

  - `setCurrentWidget( widget )`
    - Displays page assoziated by widget
  - `setTabPosition( position )`
    - Defines where tabs are drawn
  - `setTabsClosable( bool )`
    - Adds close buttons

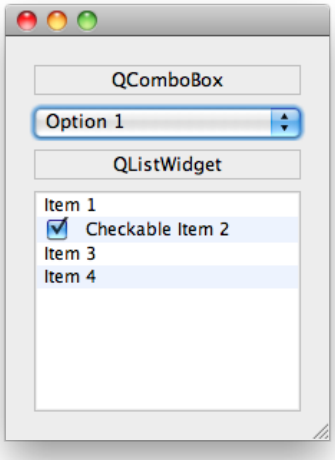


Item Widgets

- **QComboBox**

```
combo = new QComboBox(parent);
combo->addItem("Option 1", data);
connect(combo, SIGNAL(activated(int)) ...
QVariant data = combo->itemData(index);
```

  - `setCurrentIndex( index )`



- **QListWidget**

```
list = new QListWidget(parent);
list->addItem("Item 1");
// ownership of items with list
item = new QListWidgetItem("Item 2", list);
item->setCheckState(Qt::Checked);
connect(list, SIGNAL(itemActivated(QListWidgetItem*)) ...
```

  - `QListWidgetItem::setData(Qt::UserRole, data)`

- *Other Item Widgets*: `QTableWidget`, `QTreeWidget`

Other Widgets

- **`QToolBox`**
  - Column of tabbed widget items
- **`QDateEdit`**, **`QTimeEdit`**, **`QDateTimeEdit`**
  - Widget for editing date and times
- **`QCalendarWidget`**
  - Monthly calendar widget
- **`QToolButton`**
  - Quick-access button to commands
- **`QSplitter`**
  - Implements a splitter widget
- **`QStackedWidget`**
  - Stack of widgets
  - Only one widget visible at a time

See Widget Classes Documentation

Module: Widgets

- Common Widgets
- Layout Management
- Guidelines for Custom Widgets

Doing it Yourself

- Place and resize widgets
  - `move()`
  - `resize()`
  - `setGeometry()`
- Example:

```
QWidget *parent = new QWidget(...);
parent->resize(400,400);

QCheckBox *cb = new QCheckBox(parent);
cb->move(10, 10);
```

Making Qt do the Work

Definition

Layout: Specifying the relations of elements to each other instead of the absolute positions and sizes.

- Advantages:
  - Works with different languages.
  - Works with different dialog sizes.
  - Works with different font sizes.
  - Better to maintain.
- Disadvantage
  - Need to think about your layout first.

Thinking about layout is not really a disadvantage!

Managed Widgets and Sizes

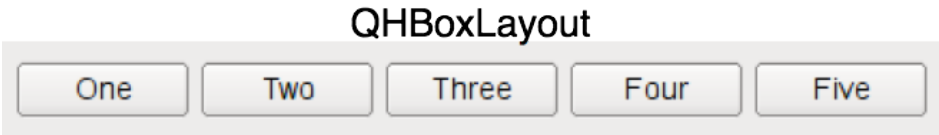
- On managed widgets never call
  - `setGeometry()`, `resize()`, or `move()`
- Preferred
  - Override
    - `sizeHint()`
    - `minimumSizeHint()`
  - Or call
    - `setFixedSize()`
    - `setMinimumSize()`
    - `setMaximumSize()`

Layout Managment Classes

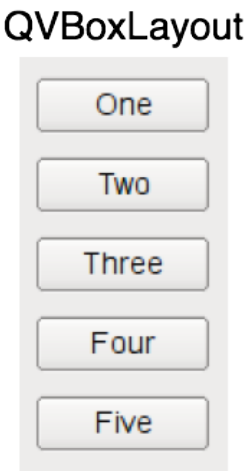
- **QHBoxLayout**
  - Lines up widgets horizontally
- **QVBoxLayout**
  - Lines up widgets vertically
- **QGridLayout**
  - Arranges the widgets in a grid
- **QFormLayout**
  - Lines up a (label, widget) pairs in two columns.
- **QStackedLayout**
  - Arranges widgets in a stack
    - only topmost is visible

QHBoxLayout and QVBoxLayout

- Lines up widgets horizontally or vertically
- Divides space into boxes
- Each managed widgets fills in one box




```
QWidget* window = new QWidget;
QPushButton* one = new QPushButton("One");
...
QHBoxLayout* layout = new QHBoxLayout;
layout->addWidget(one);
...
window->setLayout(layout);
```



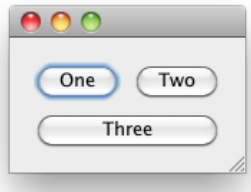
example *$QTDIR/examples/layouts/basiclayouts* (*See create[H,V]BoxLayout()*)

Widgets in a grid - QGridLayout

```
QWidget* window = new QWidget;
QPushButton* one = new QPushButton("One");
```

```
QGridLayout* layout = new QGridLayout;
layout->addWidget(one, 0, 0); // row:0, col:0
layout->addWidget(two, 0, 1); // row:0, col:1
// row:1, col:0, rowSpan:1, colSpan:2
layout->addWidget(three, 1, 0, 1, 2);
```

```
window->setLayout(layout)
```



- Additional
  - `setColumnMinimumWidth()` *(minimum width of column)*
  - `setRowMinimumHeight()` *(minimum height of row)*
- *No need to specify rows and columns before adding children.*

Demo widgets/ex-layouts (*See createGridLayout()*)

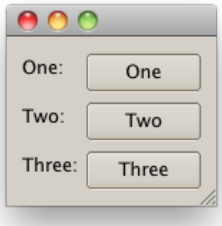
QFormLayout

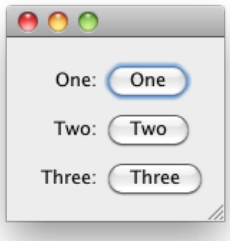
- A two-column layout
  - Column 1 a label (as annotation)
  - Column 2 a widget (as field)
- Respects style guide of individual platforms.

```
QWidget* window = new QWidget();
QPushButton* one = new QPushButton("One");
...
QFormLayout* layout = new QFormLayout();
layout->addRow("One", one);
...
window->setLayout(layout)
```

Demo widgets/ex-layouts (*See createFormLayout()*)

- Form layout with cleanlooks and mac style





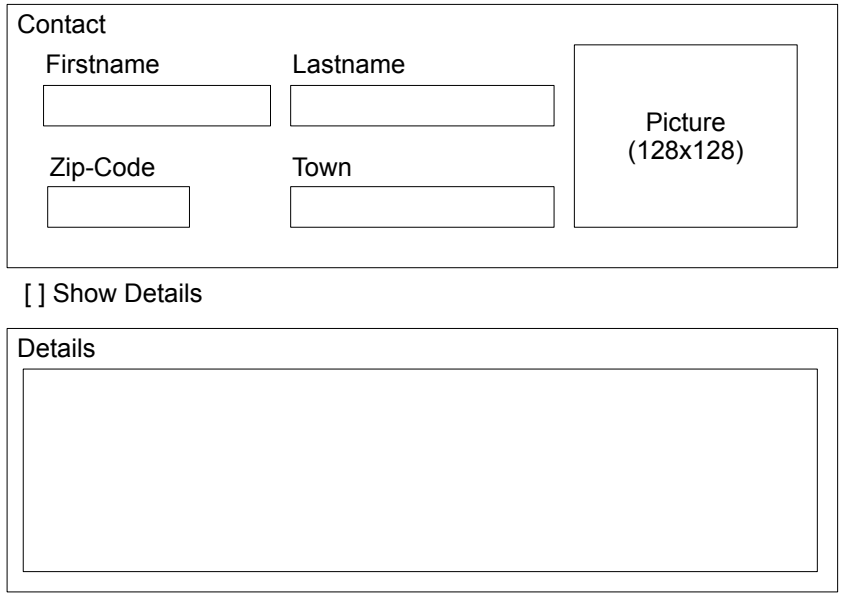
Lab: *Contact Form*

- Specified by graphic designer
  - Your task: implement it
  - Focus on correct layout
  - Details disabled by default
  - 'Show Details' enables details

Optional:

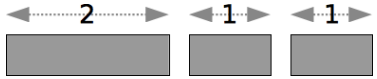
- Click on Picture
  - Lets user choose image
  - See lab description
- Validate Zip-Code as integers

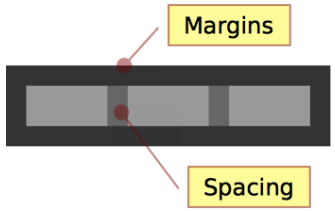
Lab widgets/lab-contactform



Some Layout Terms

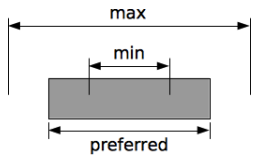
- **Stretch**
  - *Relative resize factor*
  - `QBoxLayout::addWidget(widget, stretch)`
  - `QBoxLayout::addStretch(stretch)`
  - `QGridLayout::setRowStretch(row, stretch)`
  - `QGridLayout::setColumnStretch(col, stretch)`
- **Contents Margins**
  - *Space reserved around the managed widgets.*
  - `QLayout::setContentsMargins(l,t,r,b)`
- **Spacing**
  - *Space reserved between widgets*
  - `QBoxLayout::addSpacing(size)`

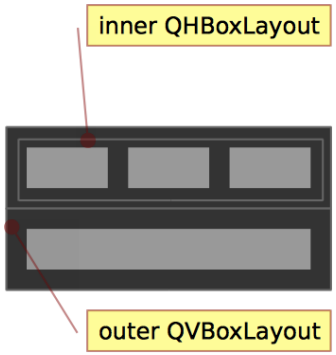




More Layout Terms

- **Strut**
  - *Limits perpendicular box dimension*
  - e.g. height for QHBoxLayout
  - *Only for box layouts*
- **Min, max and fixed sizes**
  - `QWidget::setMinimumSize( QSize )`
  - `QWidget::setMaximumSize( QSize )`
  - `QWidget::setFixedSize( QSize )`
  - *Individual width and height contraints also available*
- **Nested Layouts**
  - *Allows flexible layouts*
  - `QLayout::addLayout(...)`





Widgets Size Policies

- `QSizePolicy` describes interest of widget in resizing

```
QSizePolicy policy = widget->sizePolicy();
policy.setHorizontalPolicy(QSizePolicy::Fixed);
widget->setSizePolicy(policy);
```

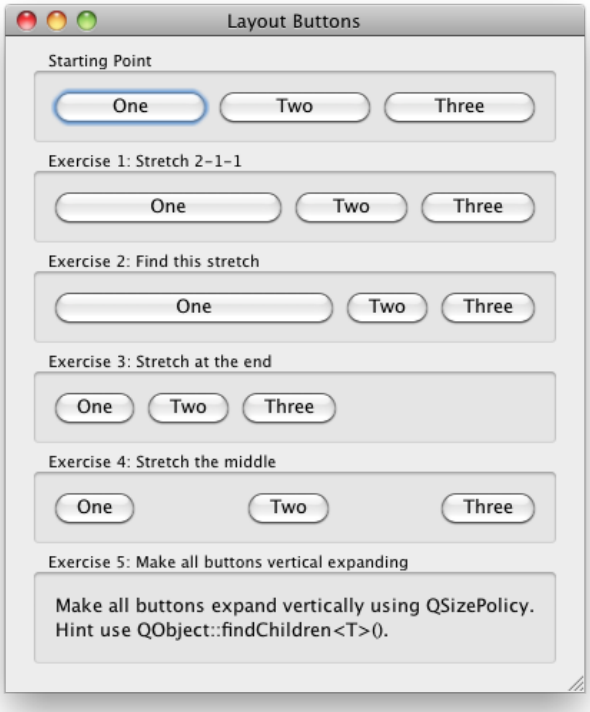
- One policy per direction (horizontal and vertical)
- Button-like widgets set size policy to the following:
  - may stretch horizontally
  - are fixed vertically
  - Similar to `QLineEdit`, `QProgressBar`, ...
- Widgets which provide scroll bars (e.g. QTextEdit)
  - Can use additional space
  - Work with less than sizeHint()
- `sizeHint()`: recommended size for widget

Available Size Policies

Policy	`sizeHint()`	Widget
Fixed	authoritative	can not grow or shrink
Minimum	minimal, sufficient	can expand, no advantage of being larger
Maximum	is maximum	can shrink
Preferred	is best	can shrink, no advantage of being larger
Minimum Expanding	is minimum	can use extra space
Expanding	sensible size	can grow and shrink

Lab: *Layout of buttons*

- Develop the following layouts
- Adjust the layouts as shown below.
- Optionally:
  - Make buttons resize vertically when making the window higher.



Lab widgets/lab-layoutbuttons

Questions And Answers

- How do you change the minimum size of a widget?
- Name the available layout managers.
- How do you specify stretch?
- When are you allowed to call `resize` and `move` on a widget?

Questions And Answers

- How do you change the minimum size of a widget?
- Name the available layout managers.
- How do you specify stretch?
- When are you allowed to call `resize` and `move` on a widget?

Questions And Answers

- How do you change the minimum size of a widget?
- Name the available layout managers.
- How do you specify stretch?
- When are you allowed to call `resize` and `move` on a widget?

Questions And Answers

- How do you change the minimum size of a widget?
- Name the available layout managers.
- How do you specify stretch?
- When are you allowed to call `resize` and `move` on a widget?

Module: Widgets

- Common Widgets
- Layout Management
- Guidelines for Custom Widgets

Guidelines: Creating a Custom Widget

- It's as easy as deriving from QWidget

```
class CustomWidget : public QWidget
{
public:
  explicit CustomWidget(QWidget* parent=0);
}
```

- If you need custom Signal Slots
  - add Q_OBJECT
- Use layouts to arrange widgets inside, or paint the widget yourself.

Guidelines: Base class and Event Handlers

- **Do not reinvent the wheel**
  - See Widget Gallery Documentation
- **Decide on a base class**
  - Often `QWidget` or `QFrame`
- **Overload needed event handlers**
  - Often:
    - `QWidget::mousePressEvent()`, `QWidget::mouseReleaseEvent()`
  - If widget accepts keyboard input
    - `QWidget::keyPressEvent()`
  - If widget changes appearance on focus
    - `QWidget::focusInEvent()`, `QWidget::focusOutEvent()`

Guidelines: Drawing a Widget

- **Decide on composite or draw approach?**
  - *If composite*: Use layouts to arrange other widgets
  - *If draw*: implement paint event
- **Reimplement `QWidget::paintEvent()` for drawing**
  - To draw widget's visual appearance
  - Drawing often depends on internal states
- **Decide which signals to emit**
  - Usually from within event handlers
  - Especially `mousePressEvent()` or `mouseDoubleClickEvent()`
- **Decide carefully on types of signal parameters**
  - General types increase reusability
  - Candidates are `bool`, `int` and `const QString&`

Guidelines: Internal States and Subclassing

- **Decide on publishing internal states**
  - Which internal states should be made publically accessible?
  - Implement accessor methods
- **Decide which setter methods should be slots**
  - Candidates are methods with integral or common parameters
- **Decide on allowing subclassing**
  - If yes
    - Decide which methods to make protected instead of private
    - Which methods to make virtual

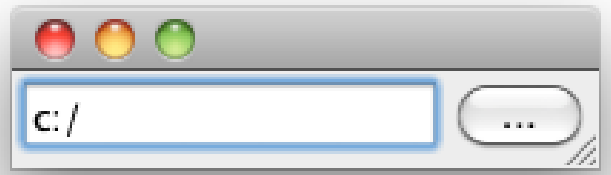
Guidelines: Widget Constructor

- **Decide on parameters at construction time**
  - Enrich the constructor as necessary
  - Or implement more than one constructor
  - If a parameter is needed for widget to work correctly
    - User should be forced to pass it in the constructor
- **Keep the Qt convention with:**

```
explicit Constructor(..., QWidget *parent = 0)
```

Lab: File Chooser

- Create a reusable file chooser component
- 2 Modes
  - Choose File
  - Choose Directory
- *Think about the Custom Widget Guidelines!*
- *Create a reusable API for a `FileChooser`?*



Lab widgets/lab-filechooser

- After lab discuss your API

Lab: Compass Widget

- Implement a “compass widget” and let user ...
  - Select a direction
    - north, west, south, east
    - and optionally none
- Provide API to ...
  - change direction programmatically
  - get informed when direction changes
- Optional
  - Add direction None
  - Select direction with the keyboard

Lab widgets/lab-compasswidget