

Qt Essentials - Dialogs 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: Dialogs and Designer

- Dialogs
- Common Dialogs
- Qt Designer



Module Objectives

- **Custom Dialogs**
 - Modality
 - Inheriting QDialog
 - Dialog buttons
- **Predefined Dialogs**
 - File, color, input and font dialogs
 - Message boxes
 - Progress dialogs
 - Wizard dialogs
- **Qt Designer**
 - Design UI Forms
 - Using forms in your code
 - Dynamic form loading



Module: Dialogs and Designer

- Dialogs
- Common Dialogs
- Qt Designer



Dialog Windows - QDialog

- Base class of dialog window widgets
- General Dialogs can have 2 modes
- Modal dialog
 - Remains in foreground, until closed
 - Blocks input to remaining application
 - Example: Configuration dialog
- Modeless dialog
 - Operates independently in application
 - Example: Find/Search dialog
- Modal dialog example

```
MyDialog dialog(this);  
dialog.setMyInput(text);  
if(dialog.exec() == Dialog::Accepted) {  
    // exec blocks until user closes dialog  
}
```



Modeless Dialog

- Use show()
 - Displays dialog
 - Returns control to caller

```
void EditorWindow::find() {  
    if (!m_findDialog) {  
        m_findDialog = new FindDialog(this);  
        connect(m_findDialog, SIGNAL(findNext()),  
                this, SLOT(onFindNext()));  
    }  
    m_findDialog->show(); // returns immediately  
    m_findDialog->raise(); // on top of other windows  
    m_findDialog->activateWindow(); // keyboard focus  
}
```



Custom Dialogs

- Inherit from QDialog
- Create and layout widgets
- Use QDialogButtonBox for dialog buttons
 - Connect buttons to accept()/reject()
- Override accept()/reject()

```
MyDialog::MyDialog(QWidget *parent) : QDialog(parent) {  
    m_label = new QLabel(tr("Input Text"), this);  
    m_edit = new QLineEdit(this);  
    m_box = new QDialogButtonBox( QDialogButtonBox::Ok|  
                                QDialogButtonBox::Cancel, this);  
    connect(m_box, SIGNAL(accepted()), this, SLOT(accept()));  
    connect(m_box, SIGNAL(rejected()), this, SLOT(reject()));  
    ... // layout widgets  
}  
  
void MyDialog::accept() { // customize close behaviour  
    if(isDataValid()) { QDialog::accept() }  
}
```



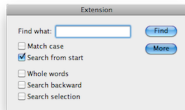
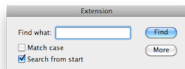
Other Dialog Topics

- Deletion of dialogs
 - No need to keep dialogs around forever
 - Call `QObject::deleteLater()`
 - Or `setAttribute(Qt::WA_DeleteOnClose)`
 - Or override `closeEvent()`
- Dialogs with extensions:
 - `QWidget::show()/hide()` used on extension

```
m_more = new QPushButton(tr("&More"));  
m_more->setCheckable(true);
```

```
m_extension = new QWidget(this);  
// add your widgets to extension  
m_extension->hide();  
connect(m_more, SIGNAL(toggled(bool)),  
        m_extension, SLOT(setVisible(bool)));
```

- Demo `$QTDIR/examples/dialogs/extension`



Module: Dialogs and Designer

- Dialogs
- **Common Dialogs**
- Qt Designer



Asking for Files - QFileDialog

- Allow users to select files or directories
- Asking for a file name

```
QString fileName =  
    QFileDialog::getOpenFileName(this, tr("Open File"));  
if(!fileName.isNull()) {  
    // do something useful  
}
```

- QFileDialog::getOpenFileNames()
 - Returns one or more selected existing files
- QFileDialog::getSaveFileName()
 - Returns a file name. File does not have to exist.
- QFileDialog::getExistingDirectory()
 - Returns an existing directory.
- setFilter("Image Files (*.png *.jpg *.bmp)")
 - Displays files matching the patterns



Showing Messages - QMessageBox

- Provides a modal dialog for ...
 - informing the user
 - asking a question and receiving an answer
- Typical usage, questioning a user

```
QMessageBox::StandardButton ret =  
    QMessageBox::question(parent, title, text);  
if(ret == QMessageBox::Ok) {  
    // do something useful  
}
```

- Very flexible in appearance
 - [See QMessageBox Class Reference Documentation](#)
- Other convenience methods
 - `QMessageBox::information(...)`
 - `QMessageBox::warning(...)`
 - `QMessageBox::critical(...)`
 - `QMessageBox::about(...)`



Feedback on progress - QProgressDialog

- Provides feedback on the progress of a slow operation

```
QProgressDialog dialog("Copy", "Abort", 0, count, this);  
dialog.setWindowModality(Qt::WindowModal);  
for (int i = 0; i < count; i++) {  
    dialog.setValue(i);  
    if (dialog.wasCanceled()) { break; }  
    //... copy one file  
}  
dialog.setValue(count); // ensure set to maximum
```

- Initialize with `setValue(0)`
 - Otherwise estimation of duration will not work
- When operation progresses, check for cancel
 - `QProgressDialog::wasCanceled()`
 - Or connect to `QProgressDialog::canceled()`
- To stay reactive call `QApplication::processEvents()`
- For modeless operation: [See QProgressDialog Documentation](#)

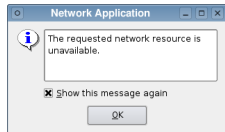


Providing error messages - QMessageBox

- Similar to QMessageBox with checkbox
- Asks if message shall be displayed again

```
m_error = new QMessageBox(this);  
m_error->showMessage(message, type);
```

- Messages will be queued
- QMessageBox::qtHandler()
 - installs an error handler for debugging
 - Shows qDebug(), qWarning() and qFatal() messages in QMessageBox box



Other Common Dialogs

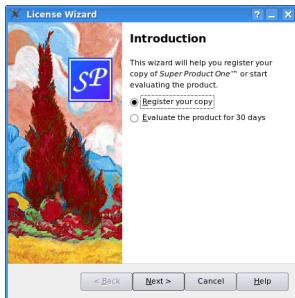
- Asking for Input - `QInputDialog`
 - `QInputDialog::getText(...)`
 - `QInputDialog::getInt(...)`
 - `QInputDialog::getDouble(...)`
 - `QInputDialog::getItem(...)`
- Selecting Color - `QColorDialog`
 - `QColorDialog::getColor(...)`
- Selecting Font - `QFontDialog`
 - `QFontDialog::getFont(...)`

Demo \$QTDIR/examples/dialogs/standarddialogs



Guiding the user - QWizard

- Input dialog
 - Consisting of sequence of pages
- Purpose: Guide user through process
 - Page by page
- Supports
 - Linear and non-linear wizards
 - Registering and using fields
 - Access to pages by ID
 - Page initialization and cleanup
 - Title, sub-title
 - Logo, banner, watermark, background
 - [See QWizard Documentation](#)
- Each page is a QWizardPage
- QWizard::addPage()
 - Adds page to wizard



Simple Wizard Example

```
QWizardPage *createIntroPage() {
    QWizardPage *page = new QWizardPage;
    page->setTitle("Introduction");
    // create widgets and layout them
    return page;
}

QWizardPage *createRegistrationPage() { ... }

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QWizard wizard;
    wizard.setWindowTitle("License Wizard");
    wizard.addPage(createIntroPage());
    wizard.addPage(createRegistrationPage());
    wizard.show();
    return app.exec();
}
```

Demo \$QTDIR/examples/dialogs/licensewizard



Questions And Answers

- What is the problem with this code:

```
QDialog *dialog = new QDialog(parent);  
QCheckBox *box = new QCheckBox(dialog);
```

- When would you use a modal dialog, and when would you use a non-modal dialog?
- When should you call `exec()` and when should you call `show()`?
- Can you bring up a modal dialog, when a modal dialog is already active?
- When do you need to keep widgets as instance variables?



Questions And Answers

- What is the problem with this code:

```
QDialog *dialog = new QDialog(parent);  
QCheckBox *box = new QCheckBox(dialog);
```

- When would you use a modal dialog, and when would you use a non-modal dialog?
- When should you call exec() and when should you call show()?
- Can you bring up a modal dialog, when a modal dialog is already active?
- When do you need to keep widgets as instance variables?



Questions And Answers

- What is the problem with this code:

```
QDialog *dialog = new QDialog(parent);  
QCheckBox *box = new QCheckBox(dialog);
```

- When would you use a modal dialog, and when would you use a non-modal dialog?
- When should you call `exec()` and when should you call `show()`?
- Can you bring up a modal dialog, when a modal dialog is already active?
- When do you need to keep widgets as instance variables?



Questions And Answers

- What is the problem with this code:

```
QDialog *dialog = new QDialog(parent);  
QCheckBox *box = new QCheckBox(dialog);
```

- When would you use a modal dialog, and when would you use a non-modal dialog?
- When should you call exec() and when should you call show()?
- Can you bring up a modal dialog, when a modal dialog is already active?
- When do you need to keep widgets as instance variables?



Questions And Answers

- What is the problem with this code:

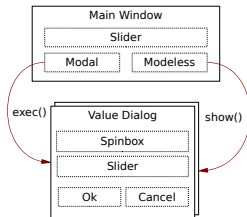
```
QDialog *dialog = new QDialog(parent);  
QCheckBox *box = new QCheckBox(dialog);
```

- When would you use a modal dialog, and when would you use a non-modal dialog?
- When should you call exec() and when should you call show()?
- Can you bring up a modal dialog, when a modal dialog is already active?
- When do you need to keep widgets as instance variables?



Lab: Custom Dialog

- We create a simple value dialog
 - Shows int value
 - As slider
 - As spin box
 - value must be < 50 to be accepted
- A main window will show result
 - Has a slider, connected to dialog
 - Two buttons to launch dialog in modal and modeless mode
- Lab dialogs/lab-dialog



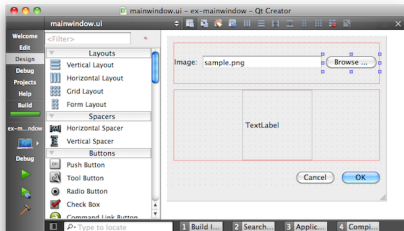
Module: Dialogs and Designer

- Dialogs
- Common Dialogs
- Qt Designer



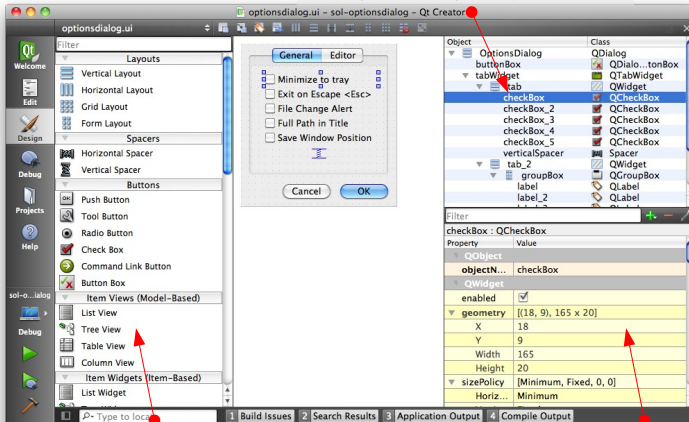
Qt Designer

- Design UI forms visually
- Visual Editor for
 - Signal/slot connections
 - Actions
 - Tab handling
 - Buddy widgets
 - Widget properties
 - Integration of custom widgets
 - Resource files



Designer Views

Object Inspector
Displays hierarchy of objects on form







Widget Box
Provides selection of widgets, layouts

Property Editor
Displays properties of selected object



Designer's Editing Modes

-  Widget Editing
 - Change appearance of form
 - Add layouts
 - Edit properties of widgets
-  Signal and Slots Editing
 - Connect widgets together with signals & slots
-  Buddy Editing
 - Assign buddy widgets to label
 - *Buddy widgets help keyboard focus handling correctly*
-  Tab Order Editing
 - Set order for widgets to receive the keyboard focus

Designer UI Form Files

- Form stored in .ui file
 - format is XML
- uic tool generates code
 - From myform.ui
 - to ui_myform.h

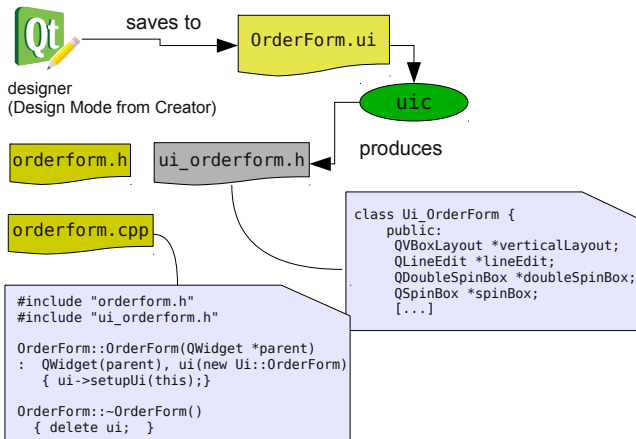
```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>MainWindow</class>
  <widget class="QMainWindow" name="MainWindow">
    <widget class="QLineEdit" name="fileName">
      <property name="text">
        <string>sample.png</string>
      </property>
    </widget>
  </ui>
```

```
// ui_mainwindow.h
class Ui_MainWindow {
public:
    QLineEdit *fileName;
    ... // simplified code
    void setupUi(QWidget *) { /* setup widgets */ }
};
```

- Form ui file in project (.pro)
 - FORMS += mainwindow.ui

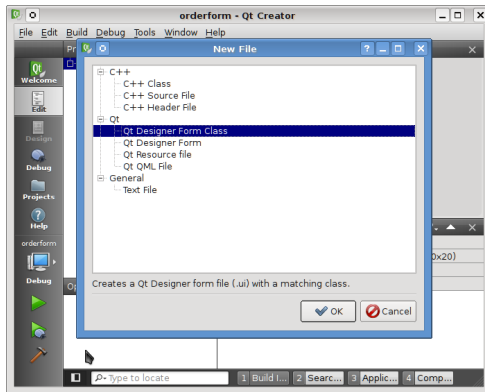


From .ui to C++



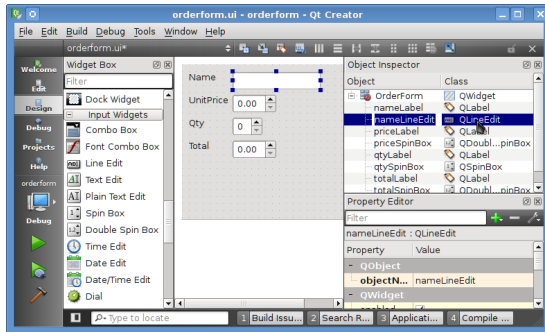
Qt Creator - Form Wizards

- **Add New...** "Designer Form"
 - or "Designer Form Class" (for C++ integration)



Naming Widgets

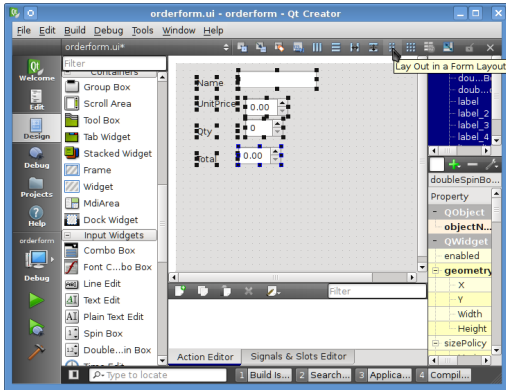
- 1 Place widgets on form
- 2 Edit `objectName` property



- *objectName* defines member name in generated code

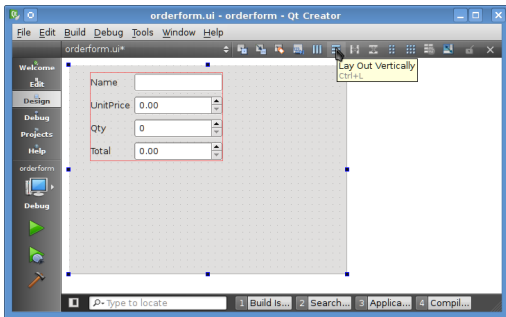
Form layout in Designer

- QFormLayout: Suitable for most input forms



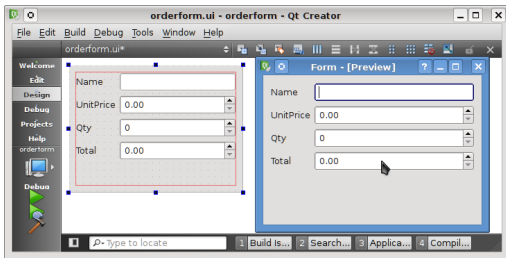
Top-Level Layout

- 1 First layout child widgets
- 2 Finally select empty space and set top-level layout



Preview Widget in Preview Mode

- Check that widget is nicely resizable



Code Integration - Header File

```
// orderform.h
class Ui_OrderForm;

class OrderForm : public QDialog {
private:
    Ui_OrderForm *ui;    // pointer to UI object
};
```

- "Your Widget" derives from appropriate base class
- ***ui** member encapsulate UI class
 - Makes header independent of designer generated code



Code Integration - Implementation File

```
// orderform.cpp
#include "ui_orderform.h"

OrderForm::OrderForm(QWidget *parent)
: QDialog(parent), ui(new Ui_OrderForm) {
    ui->setupUi(this);
}

OrderForm::~OrderForm() {
    delete ui; ui=0;
}
```

- *Default behavior in Qt Creator*



Signals and Slots in Designer

- Widgets are available as public members
 - `ui->fileName->setText("image.png")`
 - *Name based on widgets object name*
- You can set up signals & slots traditionally...
 - `connect(ui->okButton, SIGNAL(clicked()), ...`
- Auto-connection facility for custom slots
 - Automatically connect signals to slots in your code
 - Based on object name and signal
 - `void on_objectName_signal(parameters);`
 - Example: `on_okButton_clicked()` slot
 - [See Automatic Connections Documentation](#)
- Qt Creator: right-click on widget and "Go To Slot"
 - Generates a slot using auto-connected name



Using Custom Widgets in Designer

Choices for Custom Widgets

1 *Promote to Custom Widget*

- Choose the widget closest
- From context menu choose *Promote to Custom Widget*
- Code generated will now refer to given class name



2 *Implement a Designer plug-in*

- Demo \$QTDIR/examples/designer/customwidgetplugin
- See Creating Custom Widgets for Qt Designer Documentation



Dynamically loading .ui files

- Forms can be processed at runtime
 - Produces dynamically generated user interfaces
- Disadvantages
 - Slower, harder to maintain
 - Risk: .ui file not available at runtime

See Run Time Form Processing Documentation

- Loading .ui file

```
QUiLoader loader;  
QFile file("forms/textfinder.ui");  
file.open(QFile::ReadOnly);  
QWidget *formWidget = loader.load(&file, this);
```

- Locate objects in form

```
ui_okButton = qFindChild<QPushButton*>(this, "okButton");
```

Demo \$QTDIR/examples/designer/calculatorbuilder

- *Handle with care!*



Lab: Designer Order Form

- Create an order form dialog
 - With fields for price, quantity and total.
 - Total field updates itself to reflect quantity and price entered

The screenshot shows a standard Qt Designer dialog box. The title bar says 'Dialog'. Inside, there are four text input fields arranged vertically. The first field is labeled 'Name' and contains the text 'Some product I want to buy'. The second field is labeled 'Price' and contains '6.00'. The third field is labeled 'Qty' and contains '4'; this field has a blue selection border and a mouse cursor is visible on its right side. The fourth field is labeled 'Total' and contains '24.00'. At the bottom right of the dialog are two buttons: 'OK' with a checkmark icon and 'Cancel' with a red circle and slash icon.

Lab dialogs/lab-orderform



© 2010 Nokia Corporation and its Subsidiary(-ies).

The enclosed Qt Training Materials are provided under the Creative Commons Attribution ShareAlike 2.5 License Agreement.



The full license text is available here:

<http://creativecommons.org/licenses/by-sa/2.5/legalcode>

Nokia, Qt and the Nokia and Qt logos are the registered trademarks of Nokia Corporation in Finland and other countries worldwide.

