

Qt Essentials - Application Creation Module

Training Course

Visit us at <http://qt.digia.com>

Produced by Digia Plc.

Material based on Qt 5.0, created on September 27, 2012

digia

Digia Plc.



digia

- Main Windows
- Settings
- Resources
- Translation for Developers
- Deploying Qt Applications

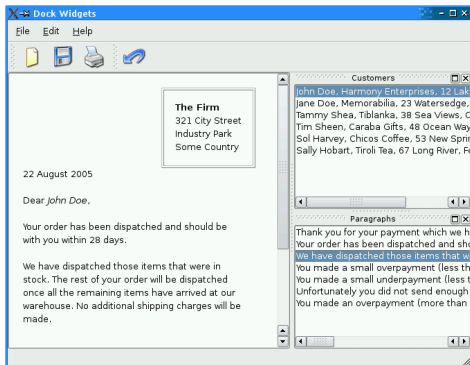
We will create an application to show fundamental concepts

- **Main Window:** How a typical main window is structured
- **Settings:** Store/Restore application settings
- **Resources:** Adding icons and other files to your application
- **Translation:** Short overview of internationalization
- **Deployment:** Distributing your application

- **Main Windows**
- Settings
- Resources
- Translation for Developers
- Deploying Qt Applications

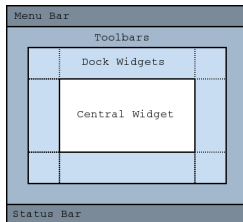


- Main window with
 - Menu bar
 - Tool bar, Status bar
 - Central widget
 - Often a dock window
- Settings (saving state)
- Resources (e.g icons)
- Translation
- Load/Save documents



Not a complete list

- QMainWindow: main application window
- Has own layout
 - Central Widget
 - QMenuBar
 - QToolBar
 - QDockWidget
 - QStatusBar



- Central Widget
 - `QMainWindow::setCentralWidget(widget)`
 - Just any widget object

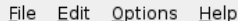
- *Action is an abstract user interface command*
- Emits signal triggered on execution
 - Connected slot performs action
- Added to menus, toolbar, key shortcuts
- Each performs same way
 - Regardless of user interface used

```
void MainWindow::setupActions() {
    QAction* action = new QAction(tr("Open ..."), this);
    action->setIcon(QIcon(":/images/open.png"));
    action->setShortcut(QKeySequence::Open);
    action->setStatusTip(tr("Open file"));
    connect(action, SIGNAL(triggered()), this, SLOT(onOpen()));

    menu->addAction(action);
    toolbar->addAction(action);
}
```

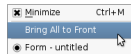
- `setEnabled(bool)`
 - Enables disables actions
 - In menu and toolbars, etc...
- `setCheckable(bool)`
 - Switches checkable state (on/off)
 - `setChecked(bool)` toggles checked state
- `setData(QVariant)`
 - Stores data with the action
- [See QAction Documentation](#)

- **QMenuBar**: a horizontal menu bar
- **QMenu**: represents a menu
 - indicates action state
- **QAction**: menu items added to **QMenu**



File Edit Options Help

```
void MainWindow::setupMenuBar() {  
    QMenuBar* bar = menuBar();  
    QMenu* menu = bar->addMenu(tr("&File"));  
    menu->addAction(action);  
    menu->addSeparator();  
  
    QMenu* subMenu = menu->addMenu(tr("Sub Menu"));  
    ...  
}
```



- Movable panel ...
 - Contains set of controls
 - Can be horizontal or vertical
- QMainWindow::addToolBar(toolbar)
 - Adds toolbar to main window
- QMainWindow::addToolBarBreak()
 - Adds section splitter
- QToolBar::addAction(action)
 - Adds action to toolbar
- QToolBar::addWidget(widget)
 - Adds widget to toolbar



```
void MainWindow::setupToolBar() {  
    QToolBar* bar = addToolBar(tr("File"));  
    bar->addAction(action);  
    bar->addSeparator();  
    bar->addWidget(new QLineEdit(tr("Find ...")));  
}
```

- Quick-access button to commands or options
- Used when adding action to QToolBar
- Can be used instead QPushButton
 - Different visual appearance!
- Advantage: allows to attach action

```
QToolButton* button = new QToolButton(this);
button->setDefaultAction(action);
// Can have a menu
button->setMenu(menu);
// Shows menu indicator on button
button->setPopupMode(QToolButton::MenuButtonPopup);
// Control over text + icon placements
button->setToolButtonStyle(Qt::ToolButtonTextUnderIcon);
```



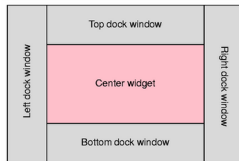
Ready

- Horizontal bar
 - Suitable for presenting status information
- showMessage(message, timeout)
 - Displays temporary message for specified milli-seconds
- clearMessage()
 - Removes any temporary message
- addWidget() or addPermanentWidget()
 - Normal, permanent messages displayed by widget

```
void MainWindow::createStatusBar() {  
    QStatusBar* bar = statusBar();  
    bar->showMessage(tr("Ready"));  
    bar->addWidget(new QLabel(tr("Label on StatusBar")));  
}
```

Creating Dock Windows - QDockWidget

- Window docked into main window
- `Qt::DockWidgetArea` enum
 - Left, Right, Top, Bottom dock areas
- `QMainWindow::setCorner(corner, area)`
 - Sets area to occupy specified corner
- `QMainWindow::setDockOptions(options)`
 - Specifies docking behavior (animated, nested, tabbed, ...)

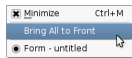


```
void MainWindow::createDockWidget() {  
    QDockWidget *dock = new QDockWidget(tr("Title"), this);  
    dock->setAllowedAreas(Qt::LeftDockWidgetArea);  
    QListWidget *widget = new QListWidget(dock);  
    dock->setWidget(widget);  
    addDockWidget(Qt::LeftDockWidgetArea, dock);  
}
```

- Launch via event handler

```
void MyWidget::contextMenuEvent(event) {  
    m_contextMenu->exec(event->globalPos());  
}
```

- or signal `customContextMenuRequested()`
 - Connect to signal to show context menu
- Or via `QWidget::actions()` list
 - `QWidget::addAction(action)`
 - `setContextMenuPolicy(Qt::ActionsContextMenu)`
 - Displays `QWidget::actions()` as context menu



- QWidget
 - setWindowModified(...)
 - setWindowTitle(...)
 - addAction(...)
 - contextMenuEvent(...)
- QMainWindow
 - setCentralWidget(...)
 - menuBar()
 - statusBar()
 - addToolBar(...)
 - addToolBarBreak()
 - addDockWidget(...)
 - setCorner(...)
 - setDockOptions(...)
- QAction
 - setShortcuts(...)
 - setStatusTip(...)
 - signal triggered()
- QMenuBar
 - addMenu(...)
- QToolBar
 - addAction(...)
- QStatusBar
 - showMessage(...)
 - clearMessage()
 - addWidget(...)

- Create a text editor with
 - *load, save, quit*
 - *about* and *About Qt*
- A `QPlainTextEdit` serves for editing the text.
- Optional:
 - Show whether the file is dirty
 - Ask the user whether to save if file is dirty when application quits
 - Make sure also to ask when window is closed via window manager
 - Show the cursor position in the status bar
 - Position is determined by cursors block and column count
 - Add printing support. [See Printing with Qt Documentation](#)

- Main Windows
- **Settings**
- Resources
- Translation for Developers
- Deploying Qt Applications

- Configure QSettings

```
QCoreApplication::setOrganizationName("MyCompany");  
QCoreApplication::setOrganizationDomain("mycompany.com");  
QCoreApplication::setApplicationName("My Application");
```

- Typical usage

```
QSettings settings;  
settings.setValue("group/value", 68);  
int value = settings.value("group/value").toInt();
```

- Values are stored as QVariant
- Keys form hierarchies using '/'
 - or use beginGroup(prefix) / endGroup()
- value() excepts default value
 - settings.value("group/value", 68).toInt()
- If value not found and default not specified

Settings::value() returned

- Store geometry of application

```
void MainWindow::writeSettings() {  
    QSettings settings;  
    settings.setValue("MainWindow/size", size());  
    settings.setValue("MainWindow/pos", pos());  
}
```

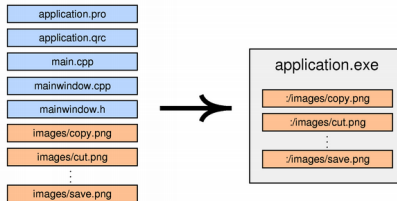
- Restore geometry of application

```
void MainWindow::readSettings() {  
    QSettings settings;  
    settings.beginGroup("MainWindow");  
    resize(settings.value("size", QSize(400, 400)).toSize());  
    move(settings.value("pos", QPoint(200, 200)).toPoint());  
    settings.endGroup();  
}
```

- Stored in platform specific format
 - Unix: INI files
 - Windows: System registry
 - MacOS: CFPreferences API
 - [See Platform-Specific Notes Documentation](#)
- *Value lookup will search several locations*
 - 1 User-specific location
 - 1 for application
 - 2 for applications by organization
 - 2 System-wide location
 - 1 for application
 - 2 for applications by organization
- [See Fallback Mechanism Documentation](#)
- QSettings creation is cheap! Use on stack

- Main Windows
- Settings
- **Resources**
- Translation for Developers
- Deploying Qt Applications

- Platform-independent mechanism for storing binary files
 - Not limited to images
- Resource files stored in application's executable
- Useful if application requires files
 - E.g. icons, translation files, sounds
 - Don't risk of losing files, easier deployment



[See The Qt Resource System Documentation](#)

- Resources specified in .qrc file

```
<!DOCTYPE RCC><RCC version="1.0">
<qresource>
  <file>images/copy.png</file>
  <file>images/cut.png</file>
  ...
</qresource>
</RCC>
```

- Can be created using QtCreator
- Resources are accessible with ':' prefix
 - Example: ":/images/cut.png"
 - Simply use resource path instead of file name
 - QIcon(":/images/cut.png")
- To compile resource, edit .pro file
 - RESOURCES += application.qrc
 - qmake produces make rules to generate binary file

- Path Prefix
 - `<qresource prefix="/myresources">`
 - File accessible via `"/myresources/..."`
- Aliases
 - `<file alias="cut.png">images/scissors.png</file>`
 - File accessible via `"/cut.png"`
- Static Libraries and Resources
 - Need to force initialization
 - `Q_INIT_RESOURCE(basename);`
- Loading resources at runtime
 - Use rcc to create binary and register resource
 - `rcc -binary data.qrc -o data.rcc`
 - `QResource::registerResource("data.rcc")`
- Traverse resource tree using `QDir(":/")`

Lab: Upgrade editor to use resources

- Use your previous editor, to use Qt resource system for icons
- Tip: You can use Qt Creator to create QRC files



- Main Windows
- Settings
- Resources
- **Translation for Developers**
- Deploying Qt Applications

- **This is by no means a complete guide!**
- Internationalization (i18n)
 - Designing applications to be adaptable to languages and regions without engineering changes.
- Localization (l10n)
 - Adapting applications for region by adding components and translations
- Qt supports the whole process:
 - QString supports unicode
 - On-screen texts (`QObject::tr()`)
 - Number and date formats (`QLocale`)
 - Icons loading (Resource System)
 - Translation tool (Qt Linguist)
 - LTR and RTL text, layout and widgets (e.g. arabic)
 - Plural handling (1 file vs 2 files)



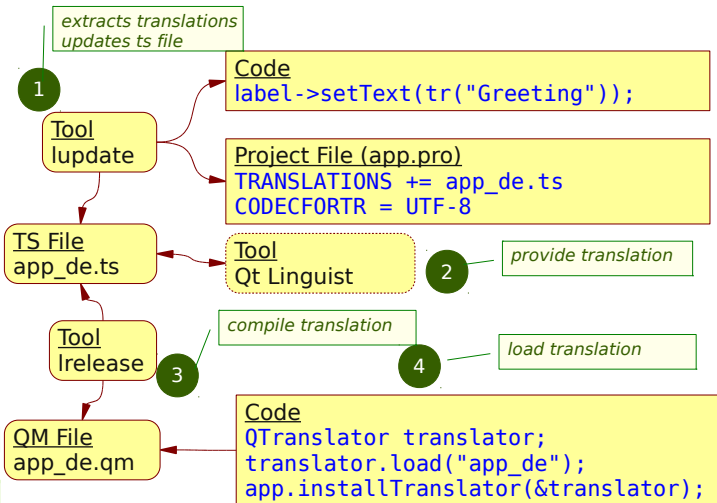
- **lupdate** - scan C++ and .ui files for strings. Create/update .ts file
- **linguist** - edit .ts file for adding translations
- **lrelease** - read .ts and creates .qm file for release.
- **QObject::tr()** - mark translatable strings in C++ code.
 - combine with `QString::arg()` for dynamic text

```
setWindowTitle(tr("File: %1 Line: %2").arg(f).arg(l));
```

```
//: This comment is seen by translation staff
```

```
label->setText(tr("Name: %1 Date: %2")  
              .arg(name, d.toString()));
```

- [See Writing Source Code for Translation Documentation](#)



- Qt classes are locale aware
- Numbers

```
QLocale::setDefault(QLocale::German); // de_DE
QLocale german; bool ok;
german.toDouble("1234,56", &ok); // ok == true
QLocale::setDefault(QLocale::C); // en_US
value = QString("1234.56").toDouble(&ok) // ok == true
```

- QDate, QTime and QDateTime

```
qDebug() << QDate().toString(); // prints localized date
```

- [See QDate Documentation](#)

- Translating Media (Resource System)

- [See Qt Resource System Documentation](#)

- Use QKeySequence for shortcut values

```
action->setShortcut(QKeySequence::New);
action->setShortcut(QKeySequence(tr("Ctrl+N")));
```

- We use our existing editor
- In the handout you will find a list of translation words
- Germany country code is de
- Tip: You can use Qt Linguist to edit translations

- Main Windows
- Settings
- Resources
- Translation for Developers
- **Deploying Qt Applications**



- **Static Linking**
 - Results in stand-alone executable
 - + Only few files to deploy
 - – Executables are large
 - – No flexibility
 - – You cannot deploy plugins
- **Shared Libraries**
 - + Can deploy plugins
 - + Qt libs shared between applications
 - + Smaller, more flexible executables
 - – More files to deploy
- Qt is by default compiled as shared library
- If Qt is pre-installed on system
 - Use shared libraries approach
- [See Deploying Qt Applications Documentation](#)



- **Shared Library Version**

- If Qt is not a system library
 - Need to redistribute Qt libs with application
- Minimal deployment
 - Libraries used by application
 - Plugins used by Qt
- Ensure Qt libraries use correct path to find Qt plugins

See [Using qt.conf Documentation](#)

- **Static Linkage Version**

- Build Qt statically
 - `$QTDIR/configure -static <your other options>`
 - Specify required options (e.g. sql drivers)
- Link application against Qt
- Check that application runs stand-alone
 - Copy application to machine without Qt and run it

See [Platform-Specific Notes Documentation](#)



© Digia Plc.

Digia, Qt and the Digia and Qt logos are the registered trademarks of Digia Plc. in Finland and other countries worldwide.

