# Qt tutorial – Part 5

# GETTING STARTED WITH QT

**Welcome to the fifth and final part of our series on using the Qt toolkit for creating graphical applications. In this issue Jono Bacon covers the remaining features that Qt has to offer. For any of the features that we don't have space to cover refer to the documentation and the Qt-interest**

One of the many benefits of using Qt is that it comes with a rich array of readymade widgets, which you can pick and use. Let's now take a quick look at some of these widgets and how we can put them to work.

## Office land here we come

One of the main uses that Qt can be utilised for is building the typical office-style applications, which utilise many of the typical widgets you see in normal everyday applications. Here is a breakdown of these widgets and which classes you can use for them:

## Menus

The purpose of a menubar is to act as a placeholder for menus (which are called popup menus). The menubar is created using the QMenuBar class to create the bar, and then each item is created using a QPopupMenu for each entry. Once these have been created, we can then use insertItem() to add an item to each menu. You can also set slots to connect to when you add the item. The following code creates a couple of menu items on the menubar and then adds some entries:

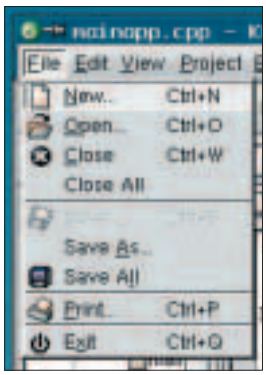


**Figure 1**

```
1  QMenuBar * menuBar = new QMenuBar(this);
2
3  QPopupMenu * fileMenu = new
QPopupMenu(menuBar);
4  QPopupMenu * itemMenu = new
QPopupMenu(menuBar);
5
6  fileMenu->insertItem( "&New Item", this,
SLOT( slotNewItem() ),  CTRL+KEY_N );
7  itemMenu->insertItem( "&Edit...", this,
SLOT( slotEdittem() ),  CTRL+KEY_E );
8
9  menuBar->insertItem( "&File", fileMenu );
10 menuBar->insertItem( "&Item", itemMenu );
```

When you have added items to a menu, you should get something looking similar to Figure 1:

## Toolbars

Toolbars are widgets, which can look similar to a menubar, but contain buttons (called toolbuttons) instead. The purpose of a toolbar is to present a button with an icon on, which connects to a frequently used function or action. Toolbars use the QToolBar class, and contain toolbuttons built from the QToolButton class.

Usage of a toolbar and toolbuttons is often coupled together, using a QmainWindow, which we will cover later.

## Status bar

The status bar widget is usually found at the bottom of the main window and is intended for showing concise information and detailing what is currently going on. In many ways it is intended as a metaphorical dashboard for an application. The status bar is implemented using the QStatusBar class, and it has basically three different modes:

- Temporary – occupies most of the status bar

## Action time

Although it is perfectly fine to use the QMenuBar, QPopupMenu, QToolBar and QtoolButton classes, it is often a good idea to also use the QAction class to build actions.

Actions are basically things a user will do while using the application; e.g. opening a file, printing a document etc. The QAction class lets us group the user interface elements for these actions (menus and toolbars) so when we add an action, we get the necessary user interface elements automatically.

Take a look at the QAction documentation for more details.

## Class Mania!

Here is a quick run down of the classes you would use for typical functions within your applications:

| | |
|---|---|
| **Create tab pages** | QTab |
| **Creating radio buttons** | QRadioButton |
| **Creating combo boxes** | QComboBox |
| **Creating a step by step wizard** | QWizard |
| **Let the user open a file** | QFileDialog |
| **Manipulate files** | QFile |
| **Manipulate regular expressions** | QRegExp |
| **Drawing graphics** | QCanvas, QPainter, QPixmap |
| **Manipulating mouse actions** | QMouseEvent, QEvent |
| **Editing multiple lines of text** | QMultiLineEdit |
| **Connecting multiple objects to slots and checking which was the caller** | QSignalMapper |
| **Dealing with data structures** | QArray, QList, QVector, QStack, QQueue, QDom[...] |
| **Storing coordinate points in a data structure** | QPointArray |
| **Creating tooltips** | QToolTip |
| **Playing sounds** | QSound |
| **Creating widget themes and styles** | QStyle |
| **Printing** | QPrinter |
| **Handling the mouse wheel event** | QWheelEvent |
| **Dealing with HTML code** | QDom[...] extension, QXml[...] extension |
| **Networking** | QSocket, QNetworkProtocol, QNetworkOperation, QFtp |

briefly. Used for explaining tool tip texts or menu entries, for example.

- Normal – occupies part of the status bar and may be hidden by temporary messages. Used for displaying the page and line number in a word processor, for example.
- Permanent – is never hidden. Used for important mode indications. Some applications put a Caps Lock indicator in the status bar.

The statusbar is often used with the QMainWindow class, which we will cover later.

## The magical main window

Qt has special support for another type of window (we have already covered Qdialog-based windows and we have discussed widgets). The intention of this type of window is that it forms the main body of your application and contains the menubar, menus, toolbar, toolbuttons, status bar, documents etc. that your application provides. The class used to create this special type of window is QMainWindow.

A QMainWindow is basically a normal window, but it has a number of convenience methods, which can be used to make life a little easier. The usual usage for a QMainWindow is to inherit from it, and then use these convenience methods where needed.

The usual behaviour for the window building process is to create some methods, which build the various parts of the window and execute these methods in the constructor. I usually create the following methods to build the various parts:

```
initActions() - Creates the menus (menubar and
menu items)
initToolbar() - Creates the toolbar and
toolbuttons
initView() - Creates the main view of the
application (usually the document for example)
initStatusbar() - Creates the status bar
```

Many coders often include some other methods for building other parts of the application:

```
initConfig() - Load the config file for the app
initDefault() - Setup any default settings
initDoc() - Create any document data related
objects and constants
```

Once these things have been executed the window is pretty much built.

The QMainWindow has a number of different convenience methods, but the most common ones you are likely to use are for setting up the menu, toolbar and status bar and for setting the main widget for the application. Facilities available in QMainWindow include addToolbar() for adding toolbar items, menuBar() returns the menu bar and creates a new one if needed. menuBar() and statusBar() are also available as convenience methods for building those widgets and they manage the relevant space needed by those widgets.

One of the main uses of a QMainWindow is for managing the screen space of the main area in the window (the place where the document traditionally is). This space is called the main or central widget, and you can set any widget as the central widget using setCentralWidget(). QmainWindow won't actually affect the widget itself – it will just manage the geometry of it.

## Non graphical aspects of Qt

Although most people who have seen Qt will think of it as a GUI toolkit, the functionality of Qt certainly doesn't end with on-screen widgets. Qt has substantial support and classes for non-graphical processing.

The first thing we can look at is the data structures that Qt has. The first and possibly the simplest is QString. The QString class offers a number of methods and facilities for common string usage, and due to the fact that QString uses implicit sharing, it is fast. Another useful class is the QStack class. There is support for pushing and popping data onto the stack with push() and pop() and much more. Another

useful class is the QList class. QList gives a lot of support for typical lists, and is often used in conjunction with a QListView or QTable widget. QList is a full template class with support for double linked lists. QList also uses the internal QLNode class to hold pointers to the usual next and previous items. Using this class can make handling lists a breeze so I suggest a good read of the documentation for QList. Other classes such as QVector, QQueue and QArray are worth looking into regarding data structures.

## KDE support

Qt is a fantastic widget set, and if you use KDE you'll be pleased to know that KDE is written using Qt. The KDE project has developed a number of extension classes and technologies, which extend Qt applications for desktop integration, inter-application communication and more. These extensions are called the KDE Libraries, and if you are planning on writing an application for use on a desktop UNIX-based system, looking into providing KDE support is a wise idea.

KDE supplies the following services built from Qt to extend Qt:

- Integration with the desktop – integrating files, directories, icons and more.
- KParts component model – KParts enables support for applications that can be embedded, applications within applications
- DCOP – Powerful interprocess communication and scripting support Addressbook, kded, shared resources – KDE has shared address books, daemons and other resources
- aRts – KDE natively uses the aRts digital synthesis server for powerful music capabilities

All of these are natively supported in Qt as most of them were coded using Qt. This desktop support extends your application if you wish.

## Wrapping things up

Well it has been an interesting journey into the world of Qt development, and I hope I have helped you get started with Qt development. Qt is a truly powerful API and it has a learning curve, but once you are started, progress can be made smoothly. I am always eager to hear how you get on, so drop me an email, via the magazine, and let me know. Good luck!