

Helmut Herold: *Das Qt-Buch*

---





**Helmut Herold**

# **Das Qt-Buch**

**Portable GUI-Programmierung unter  
Linux / Unix / Windows  
2., überarbeitete Auflage**



Alle in diesem Buch enthaltenen Programme, Darstellungen und Informationen wurden nach bestem Wissen erstellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund ist das in dem vorliegenden Buch enthaltene Programm-Material mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autoren und die SUSE LINUX AG übernehmen infolgedessen keine Verantwortung und werden keine daraus folgende Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programm-Materials, oder Teilen davon, oder durch Rechtsverletzungen Dritter entsteht. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann verwendet werden dürften.

Alle Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt und sind möglicherweise eingetragene Warenzeichen. Die SUSE LINUX AG richtet sich im Wesentlichen nach den Schreibweisen der Hersteller. Andere hier genannte Produkte können Warenzeichen des jeweiligen Herstellers sein.

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Druck, Fotokopie, Microfilm oder einem anderen Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

#### **Bibliografische Information Der Deutschen Bibliothek**

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.  
ISBN 3-89990-122-3

© 2004 SuSE Linux AG, Nürnberg (<http://www.suse.de>)

Umschlaggestaltung: Fritz Design GmbH, Erlangen

Gesamtlektorat: Nicolaus Millin

Fachlektorat: Matthias Eckermann, Michael Eicks, Stefan Fent, Thomas Fricke, Bruno Gerz, Ralf Haferkamp, Michael Hager, Stefan Probst, Harri Porten, Wolfgang Rosenauer, Chris Schläger, Lukas Tinkl

Satz: L<sup>A</sup>T<sub>E</sub>X

Druck: Kösel, Kempten

Printed in Germany on acid free paper.

## Vorwort zur ersten Auflage

Wenn wir bei Trolltech an Qt denken, so denken wir an ein Werkzeug für „real programmers“. Dies bedeutet keinesfalls, dass wir noch mit Fortran arbeiten oder unser API nicht dokumentieren. „Real programmers“ sind für uns Profis, die ihre Arbeit gemacht bekommen müssen – schnell, effizient und mit wartbarem Code in bestmöglicher Qualität.

„Real programmers“ springen nicht auf den letzten Hype auf – wie auch immer die Programmiersprache des Monats oder das Dogma der Woche lauten sollten. Für sie zählt Ausführungsgeschwindigkeit, Eleganz, Wartbarkeit, Flexibilität und nicht zuletzt die eigene Effizienz bei der Entwicklung der Programme: für immer mehr Projekte bedeutet das fast schon automatisch Qt und C++.

Verglichen mit alternativen Toolkits benötigt eine mit Qt geschriebene Anwendung in der Regel nur einen Bruchteil des Codes, ohne dabei langsamer zu sein. Wohlgemerkt, das gilt bereits für eine einzige Plattform, für cross-platform-Anwendungen fällt das Ergebnis für Qt noch viel günstiger aus. Unser Geheimrezept hierfür sind – neben der konsequenten Anwendung Objekt-orientierter Techniken – sorgfältig gestaltete APIs und Abstraktionen, mit dem Ziel die „normale“ Anwendung einer Klasse so einfach wie nur irgend möglich zu machen, ohne dabei die avanciertere Verwendung zu erschweren. Das setzt voraus, dass sich ein Toolkit nicht als Black-box präsentiert, sondern als offene Lösung auch die dahinterliegenden Konzepte offenbart.

Ein angenehmer Nebeneffekt soll hier nicht verschwiegen werden: Programmieren mit Qt macht einfach Spaß! Zumindest mir ging das vor fünf Jahren so, als ich das Toolkit erstmals als Anwender entdeckte – und bis heute hat sich daran nichts geändert. Und solange ich unsere Entwicklungsabteilung in ihrer Freizeit an privaten Qt-basierten Projekten basteln sehe, wird sich daran auch nichts ändern.

Viel Spaß also auch Ihnen mit Qt und dem bis dato umfangreichsten gedruckten Werk zur Qt-Programmierung!

Eine Bitte noch: Wenn sie einen Fehler in Qt gefunden oder einen Verbesserungsvorschlag zu machen haben, bitte zögern Sie nicht, uns eine Mail auf:

`qt-bugs@trolltech.com`

zu schicken.

Oslo, im Juni 2001

*Matthias Ettrich*

---

## Vorwort zur zweiten Auflage

Die Erfolgsstory von Trolltech AS sucht in den heutigen Tagen ihresgleichen. Ge-gründet vor zehn Jahren mit einem kleinen, aber enorm schlagkräftigen Team von Spezialisten schaffte es die norwegische Softwareschmiede mit einer beispiellosen Performance Jahr für Jahr ihre Umsätze nahezu stetig zu verdoppeln und etablierte mit Qt in der Industrie genauso wie im Open Source-Umfeld das plattformüber-greifende GUI Toolkit schlechthin.

Mittlerweile zählen mehr als 3 800 Unternehmen in über 55 Ländern zu den Kun-den von Trolltech, davon allein die Hälfte ISVs (Independent Software Vendors). Kein anderes Application Development Framework erlaubt es, mit einem einheit-lichen Quellcode derart viele unterschiedliche Plattformen und Umgebungen zu unterstützen. Auf Qt basierte Anwendungen lassen sich mit minimalem Aufwand praktisch unverändert unter Windows, Mac OS X, Unix, Linux oder Embedded Linux übersetzen und nativ verwenden. Qt ist heute bei mehr als 30 der weltwei-ten TOP-500 Unternehmen ein zentraler Baustein zur plattform-unabhängigen Ent-wicklung von Anwendungen und Grundlage für tausende von professionellen Ap-plikationen.

Qt wird derzeit in Bereichen wie Wissenschaft und Forschung, Luft- und Raum-fahrt, CAD/CAM/CAE, Unternehmenslösungen, Automobilindustrie, Animati-onsstudios, Öl- und Erdgassuche sowie der elektronischen Design Automation (EDA) sehr erfolgreich eingesetzt. Es findet Anwendung bei unzähligen kommerzi-ellen Anwendungen, wie z. B. Adobes Photoshop Album, MainConcepts MainAc-tor oder Systemen der europäischen Raumfahrtbehörde zur Erforschung des roten Planeten.

Trolltech hat es dabei geschafft, ein erfolgreiches „dual-licensing“-Geschäftsmodell zu etablieren, das neben einem herkömmlichen, kommerziellen Lizenzmodell auch eine freie Verwendung von Qt für Open Source-Software erlaubt. Dabei hat sich nicht zuletzt Linux als strategischer Inkubator für Qt erwiesen und mit dem Sie-geszug von Linux und Open Source konnte auch Qt die Welt erorbern. Neben den ganz großen Qt-basierten Open Source-Projekten, wie dem KDE Desktop oder dem DTP-System Scribus, gibt es eine kaum mehr zu überschauende Anzahl von klei-neren und mittleren Projekten, die auf Qt aufsetzen.

Interessant ist auch der Blick auf die Zufriedenheit der Kunden und Entwickler, die Qt einsetzen. 81 % sagen aus, daß Qt ihre Erwartungen überstiegen hat, in vielen Fällen sogar bei weitem. 97 % empfehlen Qt uneingeschränkt weiter. Daraus lässt sich vielleicht ableiten, warum sogar umfangreichste Software-Projekte, wie Open-Office.org an einer Qt-Portierung arbeiten oder Volvo seine Telematiksysteme für intelligente Fahrzeugkontrolle komplett auf Qt/embedded umstellt. Offensichtlich ist es an der Zeit, daß auch Sie jetzt in die Welt von Qt eintauchen und Ihr erstes erfolgreiches Qt-Projekt beginnen.

Nürnberg, im Juli 2004

*Stefan Probst, SUSE LINUX AG, R&D*

## Danksagung

Zunächst möchte ich einmal *Nico Millin* und *Dr. Markus Wirtz* von SuSE PRESS meinen Dank dafür aussprechen, dass sie beim Setzen dieses umfangreichen Buches nie den Mut verloren und immer nur positiv nach vorne – oder sagen wir in diesem Falle besser: nach hinten – geblickt haben. Trotz aller Widrigkeiten, die das Setzen eines Buches nun einmal mit sich bringt, war die Zusammenarbeit mit ihnen immer angenehm. Dafür möchte ich mich an dieser Stelle nochmals recht herzlich bedanken.

Des Weiteren möchte ich mich noch bei meinen ehemaligen Kollegen von SuSE Labs bedanken, die selbst in den größten Stresszeiten stets Zeit fanden, mir bei nervenden Fragen zu helfen, und mir mit Rat und Tat zur Seite standen. Ein dickes Dankeschön an *Klaas Freitag*, *Michael Hager*, *Stefan Hundhammer*, *Chris Schläger* und *Adrian Schroeter*.

Auch möchte ich mich bei *Michael Eicks*, *Stefan Fent*, *Thomas Fricke*, *Bruno Gerz*, *Ralf Haferkamp*, *Lukas Tinkl*, *Matthias Eckermann* und *Stefan Probst* für ihre konstruktiven Beiträge und Anregungen während des Korrekturlesens herzlich bedanken, die ich sehr gerne in dieses Buch eingearbeitet habe.

Schließlich möchte ich mich noch bei der norwegischen Firma Trolltech bedanken, ohne die es keine Qt-Bibliothek und somit auch nicht dieses Buch gäbe. Ein dickes Dankeschön an *Matthias Ettrich* und *Harri Porten* von Trolltech für ihre Unterstützung, auf die ich gerade in der Endphase der Entstehung dieses Buches angewiesen war.

Und natürlich danke ich meiner Frau Micha, die meiner Sucht zum Schreiben dieses Buches mit mehr Verständnis und Entbehrung entgegenkam als ich verdient habe. Meinen beiden Söhnen Niklas und Sascha hatte ich bei der ersten Auflage versprochen, dass wir wieder öfter Angeln gehen würden. Das Versprechen habe ich auch gehalten, wobei es trotzdem noch öfter hätte sein können.

Weisendorf-Neuenbürg, im Juli 2004

*Dr. Helmut Herold*





# Inhaltsverzeichnis

Einleitung	1
<b>1 Allgemeines zu Qt</b>	<b>5</b>
1.1 Was ist Qt und warum Qt?	5
1.2 Der Begriff „Widget“	6
1.3 Die Qt-Online-Dokumentation im HTML-Format	7
1.4 Der Qt-Assistent	9
1.5 Kompilieren von Qt-Programmen	10
1.5.1 Direkte Eingabe der Kommandozeile	11
1.5.2 Arbeiten mit dem Tool <code>qmake</code> und Makefiles	12
<b>2 Grundlegende Konzepte und Konstrukte von Qt</b>	<b>17</b>
2.1 Grundsätzlicher Aufbau eines Qt-Programms	17
2.2 Das Signal-Slot-Konzept von Qt	21
2.2.1 Schiebepalken und Buttons zum Erhöhen/Erniedrigen von LCD-Nummern	21
2.2.2 Schiebepalken und Buttons zum Ändern der Schriftgröße mit Textanzeige	24
2.2.3 Regeln für die Deklaration eigener Slots und Signale	28
2.3 Die Klasse <code>QString</code> für Zeichenketten	29
2.3.1 Wichtige Konstruktoren und Methoden	29
2.3.2 Ver-/Entschlüsseln eines Gedichts von Ringelnatz <sup>B</sup>	34
2.3.3 Palindrome durch Addition von Kehrzahlen <sup>B</sup>	35
2.4 Farbmodelle, Farbgruppen und Paletten in Qt	37
2.4.1 Farbmodelle ( <code>QColor</code> )	37
2.4.2 Farbgruppen ( <code>QColorGroup</code> )	39
2.4.3 Palette eines Widgets ( <code>QPalette</code> )	40
2.4.4 Farbenwahl über RGB- bzw. HSV-Schiebepalken <sup>B</sup>	41
2.4.5 Anzeigen von <code>Active</code> , <code>Disabled</code> , <code>Inactive</code> <sup>Z</sup>	44
2.4.6 Beispielprogramm zu <code>light()</code> und <code>dark()</code> <sup>Z</sup>	44
2.5 Ein Malprogramm mit Menüs, Events und mehr	45
2.5.1 Eine erste einfache Version eines Malprogramms	45
2.5.2 Eine zweite erweiterte Version des Malprogramms	49
2.5.3 Eine dritte Version des Malprogramms mit Menüs	51

## Inhaltsverzeichnis

---

2.5.4	Eine vierte Version des Malprogramms mit Laufbalken	55
2.5.5	Eine fünfte Version des Malprogramms mit einigen netten Erweiterungen	61
2.5.6	Eine letzte Version des Malprogramms mit Speichern und Laden von Dateien	70
<b>3</b>	<b>Die wesentlichen Qt-Widgets</b>	<b>77</b>
3.1	Allgemeine Widget-Methoden und -Parameter	79
3.1.1	Allgemeine Widget-Methoden	79
3.1.2	Parameter-Konventionen für die meisten Konstruktoren	80
3.2	Der Widget-Stil	80
3.3	Properties von Widgets	81
3.4	Buttons	83
3.4.1	Die Klasse QPushButton	83
3.4.2	Die Klassen QRadioButton und QCheckBox	83
3.4.3	Pushbuttons mit der Wirkungsweise von Radiobuttons	84
3.4.4	Gruppieren von Buttons mit QButtonGroup	84
3.4.5	Beispiel zu Buttons <sup>B</sup>	85
3.4.6	Beispiel zu Gruppen von Buttons <sup>B</sup>	88
3.4.7	Popupmenüs bei Pushbuttons <sup>Z</sup>	89
3.4.8	Synchronisieren von Radio- mit Togglebuttons <sup>Z</sup>	90
3.5	Auswahl-Widgets (List- und Komboboxen)	90
3.5.1	Die Klasse QListBox	90
3.5.2	Die Klasse QComboBox	92
3.5.3	Beispiel zu List- und Komboboxen <sup>B</sup>	93
3.5.4	Eingabe von Daten über List- und Komboboxen <sup>Z</sup>	96
3.6	Schieberegler, Drehknöpfe und Spinboxen	97
3.6.1	Schieberegler (QSlider)	97
3.6.2	Drehknopfeinstellungen (QDial)	97
3.6.3	Spinboxen (QSpinBox)	98
3.6.4	Eingabe von Datum und Zeit (QDateEdit, QTimeEdit, QDateTimeEdit)	99
3.6.5	Beispiel zu Schieberegler und Spinboxen <sup>B</sup>	101
3.6.6	Beispiel zu QDateEdit, QTimeEdit, QDateTimeEdit <sup>B</sup>	103
3.6.7	Größeneinstellung eines Rechtecks über Schieberegler und Spinboxen <sup>Z</sup>	105
3.6.8	Farbeinstellung mit Potentiometern und Spinboxen <sup>Z</sup>	106
3.7	Widgets zum Anzeigen von Informationen	106
3.7.1	Einfache Labels (QLabel)	106
3.7.2	Komfortable Textanzeige (QTextBrowser)	107
3.7.3	7-Segment-LCD-Anzeige (QLCDNumber)	107
3.7.4	Beispiel zu Labels: Anzeige von Graphikbildern <sup>B</sup>	108
3.7.5	Beispiel zur Anzeige von RichText <sup>B</sup>	109
3.7.6	Beispiel zu LCD-Zahlen: Synchronisierte Darstellung in verschiedenen Zahlensystemen <sup>Z</sup>	112
3.7.7	Mausposition als LCD-Nummer und als Fadenkreuz <sup>Z</sup>	112

3.8	Texteingabe	113
3.8.1	Einzeilige Texteingabefelder (QLineEdit)	113
3.8.2	Mehrzeilige Texteingabefelder (QTextEdit)	113
3.8.3	Beispiel zu QLineEdit: Potenzieren von Zahlen <sup>B</sup>	115
3.8.4	Verschiedene Eingabemodi und Ausrichtungen bei Texteingabefeldern <sup>Z</sup>	117
3.8.5	Synchronisierte Darstellung in verschiedenen Zahlensystemen <sup>Z</sup>	118
3.9	Menüs	118
3.9.1	Die Basisklasse für Menüs (QMenuData)	119
3.9.2	Auswahl einer Farbe über ein Menü <sup>B</sup>	120
3.9.3	Festlegen von Beschleunigern (Accelerators)	122
3.9.4	Popupmenüs, deren Inhalt sich dynamisch ändert	122
3.9.5	Die Menüleiste (QMenuBar)	122
3.9.6	Kontextmenüs	122
3.9.7	Beispiel zu Menüs <sup>B</sup>	123
3.9.8	Die Klasse QCustomMenuItem und die virtuelle Methode contextMenuEvent()	126
3.9.9	Einstellen des Fonts über Menüs, Beschleuniger oder Kontextmenüs <sup>Z</sup>	127
3.10	Hauptfenster mit Menüleiste, Werkzeugleisten, Statuszeile und Hilfstexten	128
3.10.1	Das Hauptfenster (QMainWindow)	129
3.10.2	Dock-Windows (QDockWindow, QDockArea)	131
3.10.3	Verschiebbare Menüleiste	133
3.10.4	Werkzeugleisten (QToolBar und QToolButton)	133
3.10.5	QMainWindow mit Menüs und Werkzeugleisten <sup>B</sup>	134
3.10.6	Multiple Document Interface (QWorkspace)	137
3.10.7	Kurze Hilfstexte mit QToolTip	139
3.10.8	Dynamische Tooltips (Möglichkeit 1) <sup>B</sup>	140
3.10.9	Dynamische Tooltips (Möglichkeit 2) <sup>Z</sup>	142
3.10.10	Längere Hilfstexte mit QToolTipGroup und QWhatsThis	142
3.10.11	Statuszeilen (QStatusBar)	144
3.10.12	Beispiel zu QMainWindow mit Werkzeugleisten, Statuszeile und Hilfstexten <sup>B</sup>	145
3.10.13	Einfacher Texteditor <sup>Z</sup>	149
3.11	Füllbalken	150
3.11.1	Horizontaler Füllbalken (QProgressBar)	150
3.11.2	Dialog mit Text, Füllbalken und Cancel-Button (QProgressDialog)	153
3.11.3	Demoprogramm zu QProgressDialog <sup>Z</sup>	155
3.11.4	Steuern eines Füllbalkens über Schiebepfeile <sup>Z</sup>	156
3.11.5	Würfeln der Gaußschen Glockenkurve <sup>Z</sup>	156
3.12	Listenansichten	157
3.12.1	Die Klasse QListView	157
3.12.2	Die Klasse QListViewItem	159

## Inhaltsverzeichnis

---

3.12.3	Die Klasse <code>QListViewItemIterator</code>	160
3.12.4	Drag-and-Drop bei <code>QListView</code> -Objekten	161
3.12.5	Einfaches Beispiel zu einer Listenansicht <sup>B</sup>	161
3.12.6	Directorybrowser in Form einer Listenansicht <sup>Z</sup>	162
3.12.7	Qt-Klassenhierarchie in einer Listenansicht <sup>Z</sup>	163
3.13	Fenster mit Laufbalken (Scrollviews)	164
3.13.1	Die Klasse <code>QScrollView</code>	164
3.13.2	Vorgehensweisen abhängig von der Fensterfläche	166
3.13.3	Beispiel zu den unterschiedlichen Vorgehensweisen <sup>Z</sup>	168
3.13.4	Die Klasse <code>QScrollBar</code>	168
3.13.5	Scrollen eines Bildes mit <code>QScrollBar</code> <sup>Z</sup>	169
3.13.6	Geschachtelte Fenster mit Laufbalken <sup>Z</sup>	170
3.14	Tabellen	170
3.14.1	Einfache Tabellen mit <code>QGridView</code>	170
3.14.2	Beispiel zu <code>QGridView</code> : Multiplikationsaufgaben <sup>B</sup>	171
3.14.3	Die Klasse <code>QTable</code> für Tabellen im Spreadsheet-Stil	175
3.14.4	Die Klasse <code>QHeader</code>	177
3.14.5	Die Klassen <code>QTableWidgetItem</code> und <code>QTableSelection</code>	178
3.14.6	Spreadsheet für eine Personalverwaltung <sup>Z</sup>	178
3.14.7	Tabelle mit Multiplikationsaufgaben <sup>Z</sup>	179
3.15	Widgets mit verschiebbaren Icons	180
3.15.1	Die Klasse <code>QIconView</code>	181
3.15.2	Die Klasse <code>QIconViewItem</code>	183
3.15.3	Einfaches Drag-and-Drop bei <code>QIconView</code> -Widgets <sup>Z</sup>	183
<b>4</b>	<b>Zuordnung und Layout von Widgets</b>	<b>185</b>
4.1	Zuordnung von Widgets untereinander	186
4.1.1	Die Klasse <code>QFrame</code>	186
4.1.2	Die Klassen <code>QGroupBox</code> , <code>QHGroupBox</code> und <code>QVGroupBox</code>	188
4.1.3	Die Klasse <code>QButtonGroup</code>	189
4.1.4	Die Klasse <code>QSplitter</code>	190
4.1.5	Die Klasse <code>QWidgetStack</code>	194
4.2	Layout von Widgets	197
4.2.1	Einfaches Layout mit <code>QHBoxLayout</code> , <code>QVBoxLayout</code> und <code>QGridLayout</code>	197
4.2.2	Fortgeschrittenes Layout mit <code>QLayout</code> -Klassen	200
4.2.3	Benutzerdefinierte Layouts	214
<b>5</b>	<b>Vordefinierte Dialogfenster</b>	<b>223</b>
5.1	Die Basisklasse <code>QDialog</code>	224
5.2	<code>QColorDialog</code> zur Auswahl einer Farbe	227
5.3	<code>QFileDialog</code> zur Auswahl einer Datei	230
5.4	<code>QFontDialog</code> zur Auswahl eines Fonts	236
5.5	<code>QMessageBox</code> für Mitteilungen	239
5.6	<code>QErrorMessage</code> zur ausschaltbaren Anzeige von Meldungen	244
5.7	<code>QTabDialog</code> zur Stapelung von Widgets im Karteikartenformat	245
5.8	<code>QWizard</code> zum Blättern in einer vorgegebenen Menge von Widgets	250

5.9	QInputDialog für einfache Benutzereingaben . . . . .	252
5.10	QProgressDialog zur Fortschrittsanzeige in einem Prozentbalken	258
5.11	QPrinter zum Drucken mit möglichen Druckereinstellungen	258
<b>6</b>	<b>Portable Datei- und Directory-Operationen</b>	<b>267</b>
6.1	Die Basisklasse QIODevice für Zugriffe auf E/A-Geräte . . . . .	267
6.2	Klasse QFile für Datei-Operationen . . . . .	268
6.3	Klasse QBuffer für Speicherzugriffe wie bei einem E/A-Gerät	272
6.4	Die Klassen QTextStream und QDataStream . . . . .	274
6.4.1	Klasse QTextStream zum Lesen und Schreiben von Text in Dateien . . . . .	274
6.4.2	Klasse QDataStream zum plattformunabhängigen Lesen und Schreiben in Dateien . . . . .	278
6.5	Klasse QFileInfo zum Erfragen von Informationen zu Dateien	285
6.6	Klasse QDir für Directory-Operationen . . . . .	290
6.7	Klasse QSettings für Konfigurationsdaten . . . . .	297
<b>7</b>	<b>Vordefinierte Datentypen und Datenstrukturen (Containerklassen)</b>	<b>303</b>
7.1	Vordefinierte Datentypen (QPoint, QSize und QRect) . . . . .	305
7.1.1	Die Klasse QPoint . . . . .	305
7.1.2	Die Klasse QSize . . . . .	306
7.1.3	Die Klasse QRect . . . . .	308
7.1.4	Durchschnitt und umschließendes Rechteck von zwei Rechtecken <sup>Z</sup> . . . . .	311
7.2	Data-Sharing . . . . .	312
7.3	Arrays . . . . .	314
7.3.1	Die Klasse QMemArray . . . . .	314
7.3.2	Die Klasse QPtrVector . . . . .	319
7.3.3	Die Klasse QValueVector . . . . .	319
7.3.4	Die Klasse QByteArray . . . . .	319
7.3.5	Die Klasse QBitArray . . . . .	319
7.3.6	Die Klasse QPointArray . . . . .	322
7.4	Hashtabellen und zugehörige Iterator-Klassen . . . . .	326
7.4.1	Die Klasse QDict . . . . .	326
7.4.2	QDictIterator - Eine Iterator-Klasse für QDict . . . . .	329
7.4.3	Die Klassen QAsciiDict, QPtrDict, QAsciiDictIterator und QPtrDictIterator . . . . .	333
7.4.4	Die Klassen QIntDict und QIntDictIterator . . . . .	333
7.4.5	Die Klasse QCache . . . . .	335
7.4.6	QCacheIterator - Eine Iterator-Klasse für QCache	337
7.4.7	Die Klassen QAsciiCache und QAsciiCacheIterator	338
7.4.8	Die Klassen QIntCache und QIntCacheIterator	338
7.4.9	Die Klasse QMap für wertebasierende Hashtabellen . . . . .	339
7.4.10	Cross-Reference-Liste für C-Programme <sup>Z</sup> . . . . .	342
7.5	Listen und zugehörige Iterator-Klassen . . . . .	343
7.5.1	Die Klasse QPtrList . . . . .	343

## Inhaltsverzeichnis

---

7.5.2	QPtrListIterator – Iterator-Klasse für QPtrList	346
7.5.3	Klasse QValueList – Eine wertebasierende Liste	349
7.5.4	Die Iterator-Klassen QValueListIterator und QValueListConstIterator	351
7.5.5	Die Klasse QStringList	353
7.5.6	Die Klassen QStrList und QStrIList	354
7.5.7	Das Josephus-Spiel <sup>Z</sup>	356
7.5.8	Die Klassen QList und der Iterator QListIterator	357
7.6	Stacks (LIFO-Strategie)	357
7.6.1	Die referenzbasierende Klasse QPtrStack	357
7.6.2	Die wertebasierende Klasse QValueStack	359
7.6.3	Umwandlung einer Dezimalzahl in Dualzahl <sup>Z</sup>	361
7.7	Queues (FIFO-Strategie)	361
7.7.1	Die Klasse QPtrQueue	361
7.7.2	Simulation einer Warteschlange <sup>Z</sup>	362
<b>8</b>	<b>Datenaustausch zwischen verschiedenen Applikationen</b>	<b>363</b>
8.1	Verwendung des Clipboard	363
8.1.1	Die Klasse QClipboard	364
8.1.2	Austausch von Bildern über das Clipboard <sup>B</sup>	365
8.1.3	Zufälliges Austauschen von Bildern über das Clipboard <sup>Z</sup>	367
8.2	Drag-and-Drop	367
8.2.1	Drag-and-Drop von Text oder Bildern in Qt	367
8.2.2	Drag-and-Drop in Verbindung mit dem Clipboard	375
8.2.3	Definieren von eigenen Typen für Drag-and-Drop	376
8.2.4	Die Klasse QUriDrag zum Austausch von Dateinamen	381
8.2.5	Zusammenstellen eines Kartenblattes mit Drag-and-Drop <sup>Z</sup>	381
<b>9</b>	<b>Datum, Zeit und Zeitschaltuhren</b>	<b>383</b>
9.1	Die Klasse QDate	383
9.1.1	Demonstrationsprogramm zur Klasse QDate <sup>B</sup>	385
9.1.2	Tagesdifferenz zwischen zwei Daten <sup>B</sup>	387
9.1.3	Ermitteln des Ostertermins für ein Jahr <sup>Z</sup>	388
9.2	Die Klasse QTime	388
9.2.1	Addition von zwei Zeiten <sup>B</sup>	390
9.2.2	Ein Reaktionstest <sup>B</sup>	390
9.2.3	Anzeige einer Uhr mit Fortschrittsbalken <sup>Z</sup>	393
9.3	Die Klasse QDateTime	393
9.3.1	Sekunden und Tage bis zu fixen Daten <sup>B</sup>	394
9.3.2	Automatische Arbeitszeiterfassung <sup>Z</sup>	395
9.4	Zeitschaltuhren (Timer)	396
9.4.1	Die Klasse QTimer	396
9.4.2	Timer-Events	398

<b>10</b>	<b>Graphik</b>	<b>401</b>
10.1	Allozieren von Farben . . . . .	401
10.1.1	Farballozierung mit Methoden der Klasse QColor	402
10.1.2	Farballozierung mit Methoden der Klasse QApplication . . . . .	403
10.2	Grundlegendes zur Klasse QPainter . . . . .	403
10.2.1	Konstruktoren und Methoden der Klasse QPainter	404
10.2.2	Beispiele . . . . .	405
10.3	Einstellungen für QPainter-Objekte . . . . .	410
10.3.1	Einstellen des Zeichenstifts . . . . .	410
10.3.2	Festlegen eines Füllmusters . . . . .	411
10.3.3	Festlegen eines neuen Zeichenfonts . . . . .	411
10.3.4	Zufällige Zeichenstifte mit allen Füllmustern <sup>Z</sup> . . . . .	412
10.3.5	Interaktives Einstellen von Stift und Füllmuster <sup>Z</sup> . . . . .	412
10.4	Ausgeben von Figuren und Text . . . . .	414
10.4.1	Zeichnen geometrischer Figuren . . . . .	414
10.4.2	Ausgeben von Text . . . . .	417
10.4.3	Beispiele . . . . .	418
10.5	Alternatives Ausgeben von Figuren . . . . .	420
10.5.1	Funktionen aus <qdrawutil.h> . . . . .	420
10.5.2	Beispielprogramme zu <qdrawutil.h> . . . . .	421
10.6	Transformationen . . . . .	423
10.6.1	Einfache World-Transformationen . . . . .	423
10.6.2	World-Transformationen mit der Klasse QWMatrix	429
10.6.3	View-Transformationen . . . . .	432
10.7	Clipping . . . . .	440
10.7.1	Festlegen von Clipping-Regionen . . . . .	440
10.7.2	Die Klasse QRegion . . . . .	441
10.7.3	Beispielprogramm zu Clipping <sup>B</sup> . . . . .	442
10.7.4	Vorbeifliegende Gegenstände an Fensterwand <sup>Z</sup> . . . . .	443
10.8	QPainter-Zustand sichern und wiederherstellen . . . . .	444
10.8.1	Die QPainter-Methoden save() und restore()	444
10.8.2	Drehen einer sich ständig ändernden Figur <sup>B</sup> . . . . .	444
10.8.3	QPainter-Zustände im Stack <sup>Z</sup> . . . . .	447
10.9	Flimmerfreie Darstellung . . . . .	447
10.9.1	Ausschalten der Hintergrundfarbe . . . . .	447
10.9.2	Vermeiden überflüssigen Zeichnens mittels Clipping	448
10.9.3	Doppel-Pufferung . . . . .	450
10.10	Die QCanvas-Klassen . . . . .	454
10.10.1	Die Klasse QCanvas . . . . .	454
10.10.2	Die Klasse QCanvasItem . . . . .	456
10.10.3	Die Klasse QCanvasLine . . . . .	457
10.10.4	Die Klasse QCanvasRectangle . . . . .	458
10.10.5	Die Klasse QCanvasEllipse . . . . .	458
10.10.6	Die Klasse QCanvasPolygon . . . . .	459
10.10.7	Die Klasse QCanvasPolygonalItem . . . . .	459

## Inhaltsverzeichnis

---

10.10.8	Die Klasse <code>QCanvasSpline</code>	460
10.10.9	Die Klasse <code>QCanvasSprite</code>	460
10.10.10	Die Klasse <code>QCanvasText</code>	460
10.10.11	Die Klasse <code>QCanvasPixmap</code>	461
10.10.12	Die Klasse <code>QCanvasPixmapArray</code>	461
10.10.13	Die Klasse <code>QCanvasView</code>	462
10.10.14	Demoprogramm zu den <code>QCanvas</code> -Klassen	462
10.10.15	Weiteres Beispiel zu den <code>QCanvas</code> -Klassen <sup>Z</sup>	470
<b>11</b>	<b>Bildformate und Cursorformen</b>	<b>471</b>
11.1	Pixmap-Klassen	471
11.1.1	Format einer Pixmap	471
11.1.2	Die Klasse <code>QPixmap</code>	472
11.1.3	Die Klasse <code>QPixmapCache</code>	474
11.1.4	Die Klasse <code>QBitmap</code>	475
11.1.5	Erstellen eines Screenshot durch Ziehen der Maus <sup>B</sup>	475
11.1.6	Screenshot vom Bildschirm oder eines Teilbereichs <sup>Z</sup>	478
11.2	Die Klasse <code>QCursor</code>	478
11.2.1	Konstruktoren und Methoden der Klasse <code>QCursor</code>	478
11.2.2	Beispiel zu vor- und benutzerdefinierten Cursorformen <sup>B</sup>	480
11.2.3	Ändern des Mauscursor bei einem Mausklick <sup>Z</sup>	483
11.3	Die Klassen <code>QImage</code> und <code>QImageIO</code>	483
11.3.1	Konstruktoren und Methoden der Klasse <code>QImage</code>	484
11.3.2	Routinen zum Setzen bzw. Erfragen einzelner Pixel	487
11.3.3	Die Klasse <code>QImageIO</code> für benutzerdefinierte Bildformate	488
11.3.4	Skalierungsarten bei <code>smoothScale()</code> <sup>B</sup>	488
11.3.5	Einfache Bildbearbeitung <sup>B</sup>	490
11.3.6	Erweiterte Bildbearbeitung <sup>Z</sup>	494
11.4	Die Klasse <code>QPicture</code>	494
11.4.1	Konstruktoren und Methoden der Klasse <code>QPicture</code>	495
11.4.2	Demoprogramm zur Klasse <code>QPicture</code> <sup>B</sup>	495
<b>12</b>	<b>Animationen und Sounds</b>	<b>499</b>
12.1	Animationen mit der Klasse <code>QMovie</code>	499
12.1.1	Konstruktoren und Methoden der Klasse <code>QMovie</code>	499
12.1.2	Abspielen von Filmen <sup>B</sup>	502
12.1.3	Einstellen der Abspielgeschwindigkeit <sup>Z</sup>	507
12.1.4	Anzeigen einzelner Frames (Bilder) eines Films <sup>Z</sup>	507
12.2	Sound mit der Klasse <code>QSound</code>	508
12.2.1	Konstruktor und Methoden der Klasse <code>QSound</code>	508
12.2.2	Demoprogramm zur Klasse <code>QSound</code> <sup>B</sup>	509
12.2.3	Auswahl einer Sound-Datei über <code>QFileDialog</code> <sup>Z</sup>	510
<b>13</b>	<b>Konsistente Eingaben und reguläre Ausdrücke</b>	<b>511</b>
13.1	Konsistente Eingaben	512



13.1.1	Die Basisklasse <code>QValidator</code>	. . . . .	512
13.1.2	Die Klasse <code>QIntValidator</code>	. . . . .	515
13.1.3	Die Klasse <code>QDoubleValidator</code>	. . . . .	518
13.2	Reguläre Ausdrücke ( <code>QRegExp</code> und <code>QRegExpValidator</code> )		521
13.2.1	Unterschiedliche Arten von regulären Ausdrücken		521
13.2.2	Escape-Sequenzen	. . . . .	522
13.2.3	Konstruktoren und Methoden der Klasse <code>QRegExp</code>		523
13.2.4	Die Klasse <code>QRegExpValidator</code>	. . . . .	529
<b>14</b>	<b>Tastaturfokus</b>		<b>533</b>
14.1	Einstellen und Erfragen von Fokusregeln	. . . . .	533
14.2	Reihenfolge beim Wechseln des Tastaturfokus mit Tabulatortaste		537
<b>15</b>	<b>Ereignisbehandlung (Event-Handling)</b>		<b>541</b>
15.1	Allgemeines zu Events	. . . . .	541
15.2	Die Basisklasse <code>QEvent</code> und die davon abgeleiteten Event-Klassen		542
15.2.1	Die Basisklasse <code>QEvent</code>	. . . . .	542
15.2.2	Mal-Events ( <code>QPaintEvent</code> )	. . . . .	543
15.2.3	Maus-Events ( <code>QMouseEvent</code> und <code>QWheelEvent</code> )		544
15.2.4	Tastatur-Events ( <code>QKeyEvent</code> )	. . . . .	550
15.2.5	Timer-Events ( <code>QTimerEvent</code> )	. . . . .	553
15.2.6	Fokus-Events ( <code>QFocusEvent</code> )	. . . . .	553
15.2.7	Größenänderungs-Events ( <code>QResizeEvent</code> )	. . . . .	556
15.2.8	Positionsänderungs-Events ( <code>QMoveEvent</code> )	. . . . .	557
15.2.9	Eintritts- und Austritts-Events	. . . . .	559
15.2.10	Versteck-, Sichtbarkeits- und Schließ-Events ( <code>QHideEvent</code> , <code>QShowEvent</code> , <code>QCloseEvent</code> )	. . . . .	559
15.2.11	Subwidget-Events ( <code>QChildEvent</code> )	. . . . .	562
15.2.12	Drag-and-Drop-Events ( <code>QDragEnterEvent</code> , <code>QDrag- MoveEvent</code> , <code>QDragLeaveEvent</code> , <code>QDropEvent</code> )		562
15.2.13	Anzeigen bestimmter Event-Informationen <sup>Z</sup>	. . . . .	563
15.3	Event-Filter	. . . . .	563
15.4	Senden eigener (synthetischer) Events	. . . . .	566
15.5	Benutzerdefinierte Events ( <code>QCustomEvent</code> )	. . . . .	568
<b>16</b>	<b>Signale und Slots</b>		<b>573</b>
16.1	Verbindung zwischen einem Signal und einer Slotroutine	. . . . .	573
16.2	Verbindung zwischen einem Signal und einem anderen Signal		574
16.3	Auflösen bzw. temporäres Ausschalten von bestehenden Signal- Slot-Verbindungen	. . . . .	575
16.4	Verbinden mehrerer Buttons mit einem Slot	. . . . .	576
16.4.1	Bilden einer Buttongruppe mit Klasse <code>QButtonGroup</code>		577
16.4.2	Bilden einer Signalgruppe mit Klasse <code>QSignalMapper</code>		579
16.5	Senden eigener Signale mit <code>QSignal</code>	. . . . .	581
<b>17</b>	<b>Erstellen eigener Widgets</b>		<b>583</b>
17.1	Grundlegende Vorgehensweise beim Entwurf eigener Widgets		583

## Inhaltsverzeichnis

---

17.2	Beispiel: Ein Funktionsplotter-Widget <sup>B</sup>	584
17.2.1	Beschreibung des Funktionsplotters	584
17.2.2	Headerdatei für den Funktionsplotter <sup>B</sup>	585
17.2.3	Implementierung des Funktionsplotters <sup>B</sup>	586
17.2.4	Plotten einer Sinusfunktion mit Funktionsplotter-Widget <sup>B</sup>	589
17.2.5	Plotten beliebiger Funktionen mit Funktionsplotter-Widget <sup>Z</sup>	591
17.2.6	Plotten der Funktion $1 - \exp(0.5x)$ und deren Ableitung <sup>Z</sup>	592
17.3	Ein Bildertabellen-Widget <sup>B</sup>	592
17.3.1	Beschreibung des Bildertabellen-Widget	592
17.3.2	Headerdatei für das Bildertabellen-Widget	593
17.3.3	Implementierung des Bildertabellen-Widget	594
17.3.4	Ein Memory-Spiel mit dem Bildertabellen-Widget <sup>B</sup>	596
17.3.5	Ein Puzzle-Spiel mit dem Bildertabellen-Widget <sup>Z</sup>	599
17.3.6	Ein Poker-Spiel mit dem Bildertabellen-Widget <sup>Z</sup>	599
17.3.7	Ein Bilderbrowser mit Dia-Vorschau mit dem Bildertabellen-Widget <sup>Z</sup>	600
<b>18</b>	<b>Zugriff auf Datenbanken</b>	<b>601</b>
18.1	Installation der SQL-Module von Qt	601
18.2	Herstellen von Verbindungen zu Datenbanken	602
18.3	Lesen von Daten in einer Datenbank	602
18.3.1	Einfaches Lesen von Daten in einer Datenbank	602
18.3.2	Lesen von Daten in einer Datenbank mit <code>QSqlCursor</code>	606
18.3.3	Sortieren und Filtern der gelesenen Daten mit <code>QSqlIndex</code>	608
18.4	Anzeigen und Ändern von Datenbank-Daten mit <code>QDataTable</code>	609
18.4.1	Anzeigen von Daten mit <code>QDataTable</code>	609
18.4.2	Einfaches Ändern mit <code>QDataTable</code>	612
18.5	Erstellen von Formularen mit <code>QSqlForm</code> und Navigieren mit <code>QDataBrowser</code>	612
18.6	Manipulieren von Daten einer Datenbank	616
18.6.1	Manipulieren von Datenbank-Daten mit <code>QDataBrowser</code>	616
18.6.2	Manipulieren von Datenbank-Daten mit <code>QSqlCursor</code>	617
18.6.3	Manipulieren von Datenbank-Daten mit <code>QSqlQuery</code>	620
<b>19</b>	<b>Thread-Programmierung und -Synchronisation</b>	<b>623</b>
19.1	Die Klasse <code>QThread</code>	623
19.2	Synchronisation von Threads mit <code>QMutex</code>	628
19.3	Synchronisation von Threads mit <code>QSemaphore</code>	632
19.4	Synchronisation von Threads mit <code>QWaitCondition</code>	638
19.5	Zuteilung von Plätzen in einem Speiseraum <sup>Z</sup>	642

<b>20</b>	<b>Mehrsprachige Applikationen und Internationalisierung</b>	<b>643</b>
20.1	Die Klasse <code>QString</code> für Unicode . . . . .	643
20.2	Automatische Übersetzung in andere Sprachen . . . . .	644
20.3	Automatische Anpassung an lokale Besonderheiten . . . . .	649
20.4	Konvertierung von Text-Kodierungen . . . . .	649
20.5	Landabhängiges Einblenden eines Datums <sup>Z</sup> . . . . .	650
<b>21</b>	<b>Test- und Debugging-Möglichkeiten</b>	<b>651</b>
21.1	Funktionen <code>qDebug()</code> , <code>qWarning()</code> , <code>qFatal()</code> . . . . .	651
21.2	Die Makros <code>ASSERT()</code> und <code>CHECK_PTR()</code> . . . . .	652
21.3	Debuggen mittels Objektnamen . . . . .	654
21.4	Testausgaben in eine Log-Datei <sup>Z</sup> . . . . .	656
<b>22</b>	<b>Netzwerkprogrammierung mit Qt</b>	<b>657</b>
22.1	Protokollunabhängige Netzwerkprogrammierung . . . . .	657
22.1.1	Die Klasse <code>QUrlOperator</code> . . . . .	657
22.1.2	Die Klasse <code>QUrl</code> . . . . .	664
22.1.3	Die Klasse <code>QUrlInfo</code> . . . . .	666
22.1.4	Die Klasse <code>QNetworkOperation</code> . . . . .	667
22.1.5	Herunterladen einer Datei von einem FTP-Server mit Fortschrittsanzeige <sup>B</sup> . . . . .	670
22.1.6	Gleichzeitiges Herunterladen mehrerer Dateien mit Fortschrittsanzeigen <sup>Z</sup> . . . . .	672
22.2	Implementieren eigener Netzwerkprotokolle mit <code>QNetworkProtocol</code> . . . . .	672
22.2.1	Vorgehensweise beim Implementieren eines eigenen Netzwerkprotokolls . . . . .	673
22.2.2	Die Klasse <code>QNetworkProtocol</code> . . . . .	676
22.2.3	Die Klasse <code>QFtp</code> . . . . .	678
22.2.4	Die Klasse <code>QHttp</code> . . . . .	679
22.2.5	Die Klasse <code>QLocalFs</code> . . . . .	679
22.3	High-Level Socket-Programmierung . . . . .	679
22.3.1	Die Klasse <code>QSocket</code> . . . . .	679
22.3.2	Die Klasse <code>QSocketDevice</code> . . . . .	682
22.3.3	Die Klasse <code>QServerSocket</code> . . . . .	682
22.3.4	Ein einfacher http-Dämon <sup>B</sup> . . . . .	683
22.3.5	Einfache Implementierung des nntp-Protokolls <sup>Z</sup> . . . . .	686
22.4	Low-level Socket-Programmierung . . . . .	687
22.4.1	Die Klasse <code>QSocketNotifier</code> . . . . .	687
22.4.2	Die Klasse <code>QHostAddress</code> . . . . .	690
22.4.3	DNS-Lookups mit der Klasse <code>QDns</code> . . . . .	691
22.4.4	Alternative Implementierung eines einfachen http-Servers <sup>B</sup> . . . . .	691
22.4.5	Ein kleines Mail-Programm <sup>Z</sup> . . . . .	694
22.4.6	Schicken einer Datei an Webbrowser <sup>Z</sup> . . . . .	695

<b>23 Parsen von XML-Dokumenten</b>	<b>697</b>
23.1 SAX2 und die zugehörigen Qt-Klassen . . . . .	697
23.1.1 Die SAX2-Schnittstelle von Qt . . . . .	697
23.1.2 Die Klassen der SAX2-Schnittstelle von Qt . . . . .	701
23.1.3 Plotten einer Funktion über ein XML-Dokument <sup>B</sup>	708
23.1.4 Anzeigen von XML-Dokumenten im Richtext-Format <sup>Z</sup>	712
23.2 DOM Level 2 und die zugehörigen Qt-Klassen . . . . .	713
23.2.1 Die DOM-Schnittstelle von Qt . . . . .	713
23.2.2 Die Klasse QDomNode und von ihr abgeleitete Klassen	716
23.2.3 Die Klasse QDomImplementation . . . . .	721
23.2.4 Die Containerklassen QDomNamedNodeMap und QDom- NodeList . . . . .	722
23.2.5 Plotten einer Funktion über ein XML-Dokument <sup>B</sup>	723
23.2.6 Anzeigen von XML-Dokumenten im Richtext-Format <sup>Z</sup>	725
23.3 Plotten einer Funktion als Linie oder in Balkenform über ein XML- Dokument <sup>Z</sup> . . . . .	726
<b>24 Qt-Programmierung mit anderen Bibliotheken und Sprachen</b>	<b>727</b>
24.1 OpenGL-Programmierung mit Qt . . . . .	727
24.1.1 Die Klasse QGLWidget . . . . .	727
24.1.2 Die Klassen QGLContext, QGLFormat, QGLColormap	728
24.1.3 Drehen eines Quaders in x-, y- und z-Richtung <sup>B</sup>	729
24.1.4 Drehen einer Pyramide in x-, y- und z-Richtung <sup>Z</sup>	732
24.2 Qt-Programmierung mit Perl . . . . .	732
24.2.1 Ein erstes PerlQt-Beispiel ("Hello World") . . . . .	733
24.2.2 Eigene Subwidgets und Vererbung in PerlQt . . . . .	734
24.2.3 Membervariablen (Attribute) in PerlQt . . . . .	734
24.2.4 Signale, Slots und Destruktoren . . . . .	736
24.2.5 Einige weiteren Regeln zu PerlQt . . . . .	738
24.2.6 Beispiele zur PerlQt-Programmierung . . . . .	739
<b>25 Schnittstellen zum Betriebssystem</b>	<b>745</b>
25.1 Kommunikation mit externen Programmen (QProcess) . . . . .	745
25.2 Laden dynamischer Bibliotheken (QLibrary) . . . . .	750
<b>26 Der Qt GUI-Designer</b>	<b>753</b>
26.1 Erstellen einer ersten einfachen Applikation mit dem Qt-Designer	753
26.1.1 Starten des Qt-Designers und Auswahl eines Form-Typs	753
26.1.2 Der Property-Editor . . . . .	755
26.1.3 Der Form-Editor . . . . .	755
26.1.4 Sichern des Dialogs und Erzeugen von C++-Quellcode	759
26.1.5 Testen des entworfenen Dialogs . . . . .	759
26.1.6 Öffnen eines schon existierenden Dialogs . . . . .	760
26.1.7 Hinzufügen vordefinierter Signal-Slot-Verbindungen	761
26.1.8 Festlegen eigener Slots und deren Verknüpfung mit Si- gnalen . . . . .	762

26.1.9	Implementieren selbstdefinierter Slots	.	.	763
26.1.10	Initialisierungen für Widgets	.	.	767
26.2	Layout-Management im Qt-Designer	.	.	768
26.3	Ändern der Tab Order	.	.	775
26.4	Projektmanagement	.	.	778
26.5	Erstellen von Hauptfenstern (Mainwidgets)	.	.	779
26.6	Erstellen von Datenbank-Zugriffen	.	.	780
26.7	Verwenden eigener Widgets (Custom Widgets)	.	.	780
<b>Index</b>				<b>783</b>



# Einleitung

*Ursachen erkennen, das eben ist Denken, und dadurch  
allein werden Empfindungen zu Erkenntnissen und  
gehen nicht verloren, sondern werden wesentlich und  
beginnen auszustrahlen.*

– Herrmann Hesse

Dieses Buch stellt die von der norwegischen Firma *Trolltech* entwickelte C++-Klassenbibliothek Qt (Version 3.2) vor, die eine einfache und portable GUI-Programmierung ermöglicht. Mit Qt entwickelte Programme sind sofort ohne zusätzlichen Portierungsaufwand sowohl unter allen Unix- wie auch unter allen Windows-Systemen lauffähig. Sie müssen lediglich mit den entsprechenden Compilern (Visual C++ oder Borland C++ unter Windows-Systemen oder dem auf dem jeweiligen Unix-System angebotenen cc-Compiler) kompiliert und gelinkt werden.

In Zukunft wird die Portabilität von entwickelter Software wohl immer wichtiger werden, denn warum sollte man ein Programm entwickeln, das nur unter Windows-Systemen lauffähig ist, und sich damit freiwillig aus dem Markt der Millionen von Unix- und Linux-Systemen ausschließen. Der umgekehrte Fall gilt selbstverständlich auch.

Es stellt sich hier natürlich die Frage, warum man bei Neuentwicklungen nicht die Programmiersprache Java verwenden sollte, die auch das Erstellen portabler Programme ermöglicht. Nun, Java hat sicherlich wie Qt den Vorteil der Portabilität, aber die Ablaufgeschwindigkeit von Java-Programmen kann nicht mit der von Qt-Programmen mithalten. Außerdem sollte man berücksichtigen, dass Qt eine C++-Bibliothek ist, und schon von Programmierern, die die Programmiersprache C beherrschen und nur grundlegende Kenntnisse in C++ besitzen, verwendet werden kann. Und die Anzahl von Softwareentwicklern mit diesem Profil ist sehr groß.

## Voraussetzungen des Lesers

Ein gutes *Beherrschen der Programmiersprache C* wird in diesem Buch ebenso vorausgesetzt wie die Kenntnis der *wesentlichen C-Standardfunktionen*<sup>1</sup>. Leser, die neu in der Welt der Objektorientierung und nicht der Programmiersprache C++ mächtig

---

<sup>1</sup>Leser, die sich zuerst die Programmiersprache C aneignen wollen, seien auf das bei SuSE PRESS erschienene Buch *C-Programmierung* verwiesen

sind, seien auf das auf Seite 3 erwähnte Buch verwiesen, das unter anderem einen C++-Schnellkursus enthält.

### Ein paar Worte zu diesem Buch

Bei einer so mächtigen Bibliothek wie Qt mit Hunderten von Klassen und Tausenden von Methoden, die zum Teil an Subklassen weitervererbt werden, stellt sich natürlich die Frage, wie man so etwas vermitteln kann. Ein bloßes Vorstellen der Klassen mit ihren Methoden und Datentypen wäre wenig sinnvoll, da der Neuling zum einen von der Vielzahl der Informationen erschlagen würde und zum anderen nicht das Zusammenspiel der einzelnen Klassen, was wohl für die Praxis mit am wichtigsten ist, kennenlernen würde. Hier wurde daher folgende Vorgehensweise gewählt:

- *Vorstellen der wesentlichen Konstrukte, Klassen und Methoden von Qt*  
Es werden nicht alle Klassen, sondern nur die wesentlichen Klassen sowie dort nur die wichtigsten Methoden vorgestellt. So soll vermieden werden, dass der Leser sich durch die Vielzahl der Informationen überfordert fühlt. Eine vollständige Referenz zu Qt befindet sich in der bei der Qt-Software mitgelieferten Online-Dokumentation, ohne die auch erfahrene Qt-Programmierer kaum auskommen.
- *Zusammenfassen der Klassen zu funktionalen Themengebieten*  
Eine Beschreibung der Klassen in alphabetischer Reihenfolge würde dazu führen, dass der Leser die Klassen – ähnlich dem Lernen nach einem Lexikon – nur unstrukturiert kennenlernen würde. Um dies zu vermeiden, ist dieses Buch nach Themengebieten strukturiert, in denen jeweils die Klassen zusammengefasst sind, die artverwandt sind und Ähnliches leisten. Dieses didaktische Prinzip „Vom Problem (Aufgabenstellung) zur Lösung (Klasse)“ bringt zwei Vorteile mit sich: Zum einen wird der Leser in einer strukturierten Form an die mächtige Qt-Bibliothek herangeführt, und zum anderen ermöglicht dieser Aufbau ein schnelles Nachschlagen beim späteren praktischen Programmieren, wenn zu einer gewissen Aufgabenstellung entsprechende Informationen benötigt werden.
- *Beispiele, Beispiele und noch mehr Beispiele*  
Um das Zusammenspiel der einzelnen Klassen und Methoden aufzuzeigen, werden immer wieder Beispiele<sup>2</sup> (über 350 Beispielprogramme mit etwa 35.000 Codezeilen) eingestreut, um typische Konstrukte oder Programmier-techniken aufzuzeigen, die es ermöglichen, später beim selbstständigen Programmieren in diesen Beispielprogrammen nachzuschlagen, wenn ähnliche Problemstellungen vorliegen. Es sei an dieser Stelle darauf hingewiesen, dass sich so bisweilen auch umfangreichere Programmlistings ergeben, die vielleicht zunächst etwas abschreckend wirken; aber praxisnahes Programmieren, das durch dieses Buch vermittelt werden soll, hat eben selten etwas mit „Zwanzig-Zeilern“ zu tun. Um diese Listings nicht allzu groß werden zu lassen, werden die Methoden oft inline definiert, was hoffentlich verziehen wird.

---

<sup>2</sup>Beispiele sind mit einem hochgestellten <sup>B</sup> gekennzeichnet.



## Download der Beispielprogramme und weitere Unterlagen

### Webseite zum Download aller Beispielprogramme

Alle Programmbeispiele dieses Buches sowie die des nachfolgend vorgestellten Ergänzungsbuches können von der Webseite <http://www.susepress.de/de/download/index.html> heruntergeladen werden.

### Ergänzungsbuch mit einer Einführung in die Objektorientierung und C++

Zu diesem Qt-Buch existiert noch ein Ergänzungsbuch, das weitere Beispiele und Techniken enthält. Diese 180 Beispielprogramme wurden ausgelagert, um den Umfang dieses Buches in einem erträglichen Maß zu halten. Diese ausgelagerten und teilweise sehr anspruchsvollen Beispielprogramme, die mit einem hochgestellten <sup>z</sup> gekennzeichnet sind, kann der Leser auch als Übungen heranziehen, um seine in einem Kapitel erworbenen Kenntnisse selbst zu testen. Dabei muss er zwangsweise immer wieder in der Qt-Online-Dokumentation nachschlagen, um die Aufgabenstellungen zu lösen. Dies ist beabsichtigt, da man in der späteren Programmierpraxis auch selbstständig arbeiten muss und kaum ohne ein Nachschlagen in der Qt-Online-Dokumentation auskommen wird.

Leser, die nur an den Quellen der ausgelagerten Beispiele interessiert sind, können sich die zugehörigen Programmbeispiele von der zuvor angegebenen Webseite herunterladen.

Leser, die an folgendes interessiert sind:

- ❑ eine Einführung in die Welt der Objektorientierung und insbesondere in die wichtigsten Konzepte der objektorientierten Sprache C++, um sich diesbezüglich grundlegende Kenntnisse anzueignen, die für eine Programmierung mit der objektorientierten C++-Klassenbibliothek Qt erforderlich sind, und/oder
- ❑ den gedruckten Lösungen zu den ausgelagerten Beispielen mit entsprechenden Erläuterungen

seien auf das folgende Ergänzungsbuch hingewiesen:

**Beispiele und Ergänzungen zum Qt-Buch** **24,90 €**  
*mit einer Einführung in die Objektorientierung und C++*  
*(über 400 Seiten, DIN A4)*

Dieses Ergänzungsbuch kann gegen Vorkasse zum obigen Preis, der bereits die Versandkosten beinhaltet, bei folgender Adresse bzw. bei untenstehender Telefonnummer oder Email-Adresse bestellt werden:

**less & more**

**Vertrieb von Lern- und Lehrunterlagen**

**Postfach 53**

**91084 Weisendorf**

Email: [lessamore@web.de](mailto:lessamore@web.de)

Tel: 09135/799483



# Kapitel 2

## Grundlegende Konzepte und Konstrukte von Qt

*Gib einem Menschen einen Fisch, und er hat einen Tag zu Essen. Lehre ihn Fischen, und er hat immer zu Essen.*

– Sprichwort

Dieses Kapitel gibt einen Überblick über die wesentlichen Konzepte und Konstrukte von Qt, wobei es sich in folgende Abschnitte unterteilt:

- ❑ Grundsätzlicher Aufbau eines Qt-Programms
- ❑ Das Signal/Slot-Konzept von Qt
- ❑ Die Qt-Klasse `QString`
- ❑ Die Qt-Farbmodelle
- ❑ Ein Malprogramm, das schrittweise erweitert wird

### 2.1 Grundsätzlicher Aufbau eines Qt-Programms

Unser erstes Qt-Programm 2.1 erzeugt ein kleines Window mit einem Text in der Mitte, über und unter dem sich ein Button befindet (siehe Abbildung 2.1). Beide Buttons sind zur Beendigung des Programms gedacht, wobei der obere Button bereits aktiviert wird, sobald man auf ihm nur die Maustaste drückt, und der untere Button erst nach einem vollständigen Klick (Drücken und Loslassen).

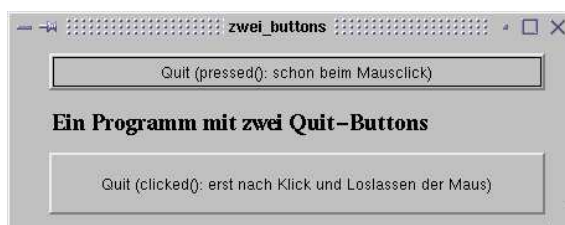


Abbildung 2.1: Window zum Programm 2.1 (`zwei_buttons.cpp`)

## 2 Grundlegende Konzepte und Konstrukte von Qt

---

### Programm 2.1 – zwei\_buttons.cpp:

Erstes Qt-Programm mit einem Window, das zwei Buttons und Text enthält

```
1 #include <qapplication.h>    //... in jedem Qt-Programm notwendig
2 #include <qlabel.h>         //... für Klasse QLabel
3 #include <qpushbutton.h>    //... für Klasse QPushButton
4 #include <qfont.h>          //... für Klasse QFont
5
6 int main( int argc, char* argv[] )
7 {
8     // Instantiierung eines QApplication-Objekts; immer notwendig
9     QApplication myapp( argc, argv );
10
11     // Hauptwidget, in dem Buttons und Text untergebracht werden.
12     QWidget* mywidget = new QWidget();
13     //... horizontale, vertikale Position, Breite, Höhe in Pixel
14     mywidget->setGeometry( 200, 100, 450, 150 );
15
16     // Instantiierung eines ersten Buttons
17     QPushButton* erster_button = new QPushButton(
18         "Quit (pressed(): schon beim Mausklick)", mywidget );
19     // Rel. Position (30,10) in mywidget (400 breit, 30 hoch)
20     erster_button->setGeometry( 30, 10, 400, 30 );
21     //... Tritt Signal 'pressed' bei erster_button auf, ist der
22     //... SLOTcode 'quit' (Verlassen des Programms) aufzurufen
23     QObject::connect( erster_button, SIGNAL( pressed() ), &myapp, SLOT( quit() ) );
24
25     // Instantiierung eines Labels (nur Text)
26     QLabel* mylabel = new QLabel( "Ein Programm mit zwei Quit-Buttons", mywidget );
27     // Rel. Position (30,40) in mywidget (400 breit, 50 hoch)
28     mylabel->setGeometry( 30, 40, 400, 50 );
29     mylabel->setFont(QFont("Times", 18, QFont::Bold) );
30
31     // Instantiierung eines zweiten Buttons
32     QPushButton* zweiter_button = new QPushButton(
33         "Quit (clicked(): erst nach Klick und Loslassen der Maus)", mywidget );
34     // Rel. Position (30,90) in mywidget (400 breit, 50 hoch)
35     zweiter_button->setGeometry( 30, 90, 400, 50 );
36     // Tritt Signal 'clicked' bei zweiter_button auf, ist der
37     // SLOTcode 'quit' (Verlassen des Programms) aufzurufen
38     QObject::connect( zweiter_button, SIGNAL( clicked() ), &myapp, SLOT( quit() ) );
39
40     myapp.setMainWidget( mywidget ); // 'mywidget' ist das Hauptwidget
41     mywidget->show(); // Zeige Hauptwidget mit seinen Subwidgets an
42     return myapp.exec(); // Übergabe der Kontrolle an Methode 'exec'
43                             // von QApplication
44 }
```

Nachfolgend einige Erläuterungen zu diesem Programm:

*Zeilen 1 – 4: Inkludieren der Qt-Headerdateien*

In den meisten Fällen haben die Qt-Headerdateien den gleichen Namen (ohne .h) wie die entsprechenden Klassennamen, die dort deklariert sind.

*Zeile 9: QApplication myapp( argc, argv )*

Dieses Objekt ist für das ganze *Event-Handling* verantwortlich und wird immer in Qt-Programmen benötigt. Die Übergabe der Kommandozeilenargumente an den Konstruktor des QApplication-Objekts ist notwendig, da QApplication einige spezielle Kommandozeilenoptionen kennt, die es – wenn solche angegeben sind – selbst auswertet und dann aus argv mit Dekrementierung von argc entfernt. Eine solche spezielle Qt-Option ist z. B. -style, die es bei Aufruf ermöglicht, einen speziellen Widget-Stil als Voreinstellung für die Applikation festzulegen. Bietet das Anwendungsprogramm eigene Kommandozeilenargumente an, sollte es diese grundsätzlich erst nach der Instantiierung des QApplication-Objekts auswerten.

*Zeile 12: QWidget\* mywidget = new QWidget()*

Diese Anweisung erzeugt das Hauptwidget mywidget, das als Container für die später erzeugten Subwidgets (Text und zwei Buttons) dient.

*Zeile 14: mywidget->setGeometry( 200, 100, 450, 150 )*

- ❑ Die horizontalen und vertikalen Positionen (die ersten beiden Argumente) sind immer relativ zum Elternwidget. Da mywidget als Hauptwidget kein Elternwidget hat, beziehen sich die hier angegebenen Koordinaten auf den ganzen Bildschirm, während sie sich bei den folgenden Anweisungen relativ auf die linke obere Ecke ihres Hauptwidgets (Objekt mywidget) beziehen:

```
19   erster_button->setGeometry( 30, 10, 400, 30 );
27   mylabel->setGeometry( 30, 40, 400, 50 );
33   zweiter_button->setGeometry( 30, 90, 400, 50 );
```

- ❑ Der dritte Parameter legt dabei die Breite und
- ❑ der vierte die Höhe des betreffenden Widgets in Pixel fest.

*Zeile 17: QPushButton\* erster\_button = new QPushButton(*

*"Quit (pressed(): schon beim Mausklick)", mywidget )*

Mit dieser Anweisung wird ein erster Button erzeugt. Der erste Parameter des Konstruktors QPushButton legt den Button-Text und der zweite das Elternwidget fest, in dem dieser Button (hier mywidget) erscheinen soll.

*Zeile 22: QObject::connect( erster\_button, SIGNAL(pressed()), &myapp, SLOT(quit()) )*

Mit der Methode connect() (von Klasse QObject) wird als Reaktion auf Eintreffen des Signals pressed() (von erster\_button) die vordefinierte Slotroutine quit() eingerichtet, die zur sofortigen Beendigung des Programms führt.

*Zeilen 25, 27, 28:*

```
// Zeile 25: erzeugt ein Label (1. Parameter legt den Label-Text und der
//           2. Parameter das Elternwidget (mywidget) für das Label fest.
QLabel* mylabel = new QLabel( "Ein Programm mit zwei Quit-Buttons", mywidget );
// Zeile 27: Mit der von Klasse QWidget geerbten Methode setGeometry() wird nun
//           relative Position dieses Label im Hauptwidget (mywidget) festgelegt.
mylabel->setGeometry( 30, 40, 400, 50 );
// Zeile 28: legt mit Methode setFont() den Zeichensatz unter Verwendung der
//           Klasse QFont für den auszugebenden Text
mylabel->setFont( QFont( "Times", 18, QFont::Bold ) );
```

## 2 Grundlegende Konzepte und Konstrukte von Qt

---

Zeilen 31, 33, 36:

```
// Zeile 31: Einrichten eines 2. Pushbuttons
QPushButton* zweiter_button = new QPushButton(
    "Quit (clicked(): erst nach Klick und Loslassen der Maus)", mywidget );
// Zeile 33: Festlegen der relativen Position dieses Buttons
zweiter_button->setGeometry( 30, 90, 400, 50 );
// Zeile 36: Wenn zweiter_button das vordefinierte Signal clicked() schickt,
//           wird die vordefinierte Slotroutine quit() ausgeführt, was zur
//           sofortigen Beendigung des Programms führt. Anders als das Signal
//           pressed() wird Signal clicked() erst dann ausgelöst, wenn nach
//           Drücken der Maustaste auf Button diese wieder losgelassen wird.
QObject::connect( zweiter_button, SIGNAL(clicked()), &myapp, SLOT(quit()) );
```

Zeile 38: `myapp.setMainWidget( mywidget )`

Mit dieser Anweisung wird dem `QApplication`-Objekt mitgeteilt, dass `mywidget` die Rolle des Hauptwidgets übernimmt. Die Besonderheit eines Hauptwidgets ist, dass das jeweilige Programm vollständig beendet wird, wenn man dieses Hauptwidget schließt. Legt man nicht mit `setMainWidget()` ein Hauptwidget fest und der Benutzer schließt das entsprechende Widget mit einem Mausklick auf den *Schließen*-Button (meist rechts oben in der Titelleiste der Fenster), so wird zwar das Widget vom Bildschirm entfernt, das Programm läuft jedoch im Hintergrund weiter und belastet unnötigerweise die CPU. Wenn man in einem Programm mehr als ein Hauptwidget benötigt, so kann man die Beendigung des Programms erzwingen, wenn das letzte Fenster geschlossen wird. Dazu muss man die folgende Zeile im Programm angeben:

```
QObject::connect( qApp, SIGNAL( lastWindowClosed() ), qApp, SLOT( quit() ) );
```

`qApp` ist immer ein globaler Zeiger auf das `QApplication`-Objekt.

Zeile 39: `mywidget->show()`

legt fest, dass das Hauptwidget mit allen seinen Subwidgets auf dem Bildschirm anzuzeigen ist. Hier ist zu erwähnen, dass jedes Widget entweder sichtbar oder aber auch versteckt (nicht sichtbar) sein kann. Die Voreinstellung ist, dass Widgets, die keine Subwidgets von einem anderen sichtbaren Widget sind, unsichtbar bleiben.

Zeile 40: `return myapp.exec()`

Mit dieser letzten Anweisung wird die vollständige Kontrolle des Programmablaufs an das zu Beginn erzeugte Objekt `myapp` (der Klasse `QApplication`) übergeben.

Hier können wir also festhalten, dass unsere Qt-Programme von nun an die folgende Grundstruktur haben:

```
#include <q...h>
...
int main( int argc, char* argv[] ) {
    QApplication myapp( argc, argv );
    QWidget* mywidget = new QWidget();
    mywidget->setGeometry( x_pos, y_pos, breite, hoehe );
    .....
    myapp.setMainWidget( mywidget );
    mywidget->show();
    return myapp.exec();
}
```

## 2.2 Das Signal-Slot-Konzept von Qt

Hier wird zunächst ein zweites Qt-Programm erstellt, um an diesem das wichtige Signal-Slot-Konzept von Qt zu verdeutlichen, bevor ein weiteres Programmbeispiel aufzeigt, wie man sich eigene Slots in Qt definieren kann.

### 2.2.1 Schiebepalken und Buttons zum Erhöhen/Erniedrigen von LCD-Nummern

Das hier vorgestellte Programm 2.2 erzeugt ein kleines Window, in dem in der Mitte eine 7-Segment-LCD-Nummer angezeigt wird, die sich sowohl über den darüber angezeigten Schiebepalken (*Slider*) als auch über die beiden darunter angezeigten Buttons erhöhen bzw. erniedrigen lässt (siehe Abbildung 2.2).

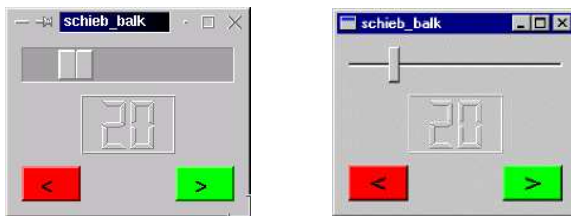


Abbildung 2.2: In- bzw. Dekrementieren einer LCD-Nummer über Schiebepalken bzw. Buttons (im Motif- und Windows-Stil)

Im Programm 2.2 sind neu eingeführte Konstrukte fett hervorgehoben.

Programm 2.2 – `schieb_balk.cpp`:

LCD-Nummer, die mit Schiebepalken bzw. Buttons verändert werden kann

```

1  #include <qapplication.h>
2  #include <qpushbutton.h>
3  #include <qslider.h>
4  #include <qlcdnumber.h>
5
6  int main( int argc, char* argv[] )
7  {
8      QApplication myapp( argc, argv );
9
10     QWidget* mywidget = new QWidget();
11     mywidget->setGeometry( 400, 300, 200, 150 );
12
13     //.... Erzeugen eines Schiebepalkens
14     QSlider* myslider = new QSlider( 0, // kleinstmögl. Wert
15                                     99, // größtmögl. Wert
16                                     1, // Schrittweite
17                                     20, // Startwert
18                                     QSlider::Horizontal, // Richtung
19                                     mywidget ); // Elternwidget
20     myslider->setGeometry( 10, 10, 180, 30 );
21

```

## 2 Grundlegende Konzepte und Konstrukte von Qt

```
22 //.... Erzeugen eines Objekts zur Anzeige einer LCD-Nummer
23 QLCDNumber* mylcdnum = new QLCDNumber( 2,          // Ziffernzahl
24                                     mywidget ); // Elternwidget
25 mylcdnum->setGeometry( 60, 50, 80, 50 );
26 mylcdnum->display( 20 ); // zeige Startwert an
27
28 // Verbinde Schiebepalken und Nummernanzeige
29 QObject::connect( myslider, SIGNAL( valueChanged( int ) ),
30                 mylcdnum, SLOT( display( int ) ) );
31
32 // Zwei Buttons zum schrittweisen Erhöhen und
33 // Erniedrigen der Schiebepalken-Werte
34 QPushButton* decrement = new QPushButton( "<", mywidget );
35 decrement->setGeometry( 10, 110, 50, 30 );
36 decrement->setFont(QFont("Times", 18, QFont::Bold) );
37 decrement->setPaletteBackgroundColor( Qt::red );
38
39 QPushButton* increment = new QPushButton( ">", mywidget );
40 increment->setGeometry( 140, 110, 50, 30 );
41 increment->setFont(QFont("Times", 18, QFont::Bold) );
42 increment->setPaletteBackgroundColor( Qt::green );
43
44 // Verbinde das clicked()-Signal der Buttons mit den Slots, die
45 // den Schiebepalken-Wert erhöhen bzw. erniedrigen
46 QObject::connect( decrement, SIGNAL( clicked() ),
47                 myslider, SLOT( subtractStep() ) );
48 QObject::connect( increment, SIGNAL( clicked() ),
49                 myslider, SLOT( addStep() ) );
50
51 myapp.setMainWidget( mywidget );
52 mywidget->show();
53 return myapp.exec();
54 }
```

Zur Erläuterung:

```
26 mylcdnum->display( 20 ); // zeige Startwert an
```

Hiermit wird festgelegt, dass beim ersten Einblenden des mylcdnum-Widgets als Startwert die Nummer 20 zu verwenden ist. `display()` ist eigentlich keine Methode der Klasse `QLCDNumber`, sondern ein von dieser Klasse zur Verfügung gestellter Slot. Wie diese Anweisung aber zeigt, können Slots genauso wie Methoden verwendet werden, was umgekehrt nicht gilt: Methoden können nämlich nicht wie Slots in einem `connect()`-Aufruf mit einem Signal verbunden werden.

```
29 QObject::connect( myslider, SIGNAL( valueChanged( int ) ),
30                 mylcdnum, SLOT( display( int ) ) );
```

Mit dieser Anweisung wird festgelegt, dass bei jeder Änderung des Schiebepalken-Werts die Slotroutine `display(int)` mit dem aktuellen Schiebepalken-Wert aufgerufen ist. Bei jeder Änderung des Schiebepalken-Werts wird vom `myslider`-Widget das Signal `valueChanged(int)` mit dem aktuellem Schiebepalken-Wert als Argument gesendet, und dieses Argument wird als Argument an `display(int)` weitergereicht.



```

37   decrement->setPaletteBackgroundColor( Qt::red );
42   increment->setPaletteBackgroundColor( Qt::green );

```

Mit diesen beiden Anweisungen wird für den Button `decrement` als Hintergrundfarbe rot und für den Button `increment` grün festgelegt.

```

46   QObject::connect( decrement, SIGNAL( clicked() ),
47                   myslider, SLOT( subtractStep() ) );
48   QObject::connect( increment, SIGNAL( clicked() ),
49                   myslider, SLOT( addStep() ) );

```

Dieser Codeausschnitt legt fest, dass beim Schicken des Signals `clicked()` von einem der beiden Buttons `decrement` bzw. `increment` die vordefinierte Slotroutine `subtractStep()` bzw. `addStep()` des `myslider`-Widgets auszuführen ist.

Die einzelnen Qt-Klassen bieten unterschiedliche Signale und Slotroutinen an, die sich mit `QObject::connect` verbinden lassen. Im Programm 2.2 wurden die in Abbildung 2.3 gezeigten Signal-/Slotverbindungen zwischen den einzelnen Objekten eingerichtet.

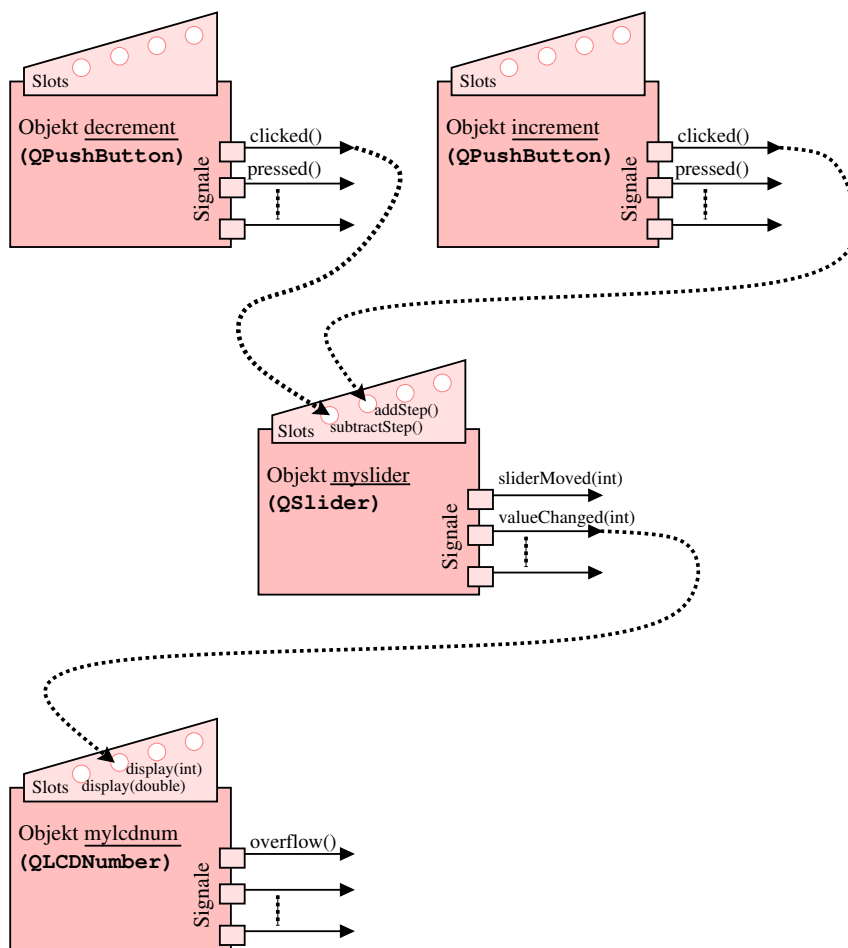


Abbildung 2.3: Signal-/Slotverbindungen im Programm 2.2

## 2.2.2 Schieberegler und Buttons zum Ändern der Schriftgröße mit Textanzeige

Hier werden wir kennenlernen, wie man eigene Slots definieren kann und was bei der Generierung des Programms zu beachten ist. Das hier vorgestellte Programm 2.3 ist eine Erweiterung zum vorherigen Programm `schieb_balk.cpp`, indem es die angezeigte LCD-Nummer als Schriftgröße interpretiert und das Aussehen eines Textes mit dieser Schriftgröße exemplarisch rechts anzeigt (siehe Abbildung 2.4).

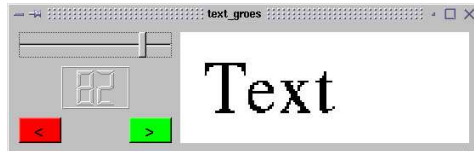


Abbildung 2.4: Schieberegler und Buttons zum Ändern der Schriftgröße mit Textanzeige

Für dieses Programm 2.3 soll ein eigener Slot definiert werden, der immer aufzurufen ist, wenn der Schieberegler-Wert sich ändert. Dieser Slot ist für die Darstellung des Textes mit der neuen Schriftgröße, die dem Schieberegler-Wert entspricht, zuständig. Für das Signal-Slot-Konzept hat Qt einige neue Schlüsselwörter eingeführt, die vom Präprozessor in die entsprechende C++-Syntax übersetzt werden. Um sich Klassen zu definieren, die eigene Slots und/oder Signale deklarieren, muss folgende Qt-Syntax eingehalten werden:

```
Class MyClass : public QObject {
    Q_OBJECT
    ....
signals:
    //.... hier werden die entsprechenden Signale deklariert, wie z.B.:
    void buchstabe_a_gedrueckt();
public slots:
    //.... hier werden die public Slots deklariert, wie z.B.:
    void lasse_text_blinken();
private slots:
    //.... hier werden die privaten Slots deklariert, wie z.B.:
    void ich_bin_interner_slot();
    //.... Weitere Deklarationen sind hier möglich
}
```

Bei der Deklaration von Slots und Signalen sind die folgenden Punkte zu beachten:

1. **Deklaration von Slots und Signalen ist nur in Klassen erlaubt**  
Die Deklaration einer Funktion, die als Slotroutine dienen soll, oder eines Signals außerhalb einer Klasse ist also nicht möglich, was im Übrigen ja auch der C++-Philosophie widersprechen würde.
2. **Klassen mit eigenen Slots bzw. Signalen müssen von QObject abgeleitet sein.**  
Da man wohl in den meisten Fällen beim Programmieren mit Qt ohnehin Klassen verwendet, die direkt oder indirekt von der Klasse `QWidget` abgeleitet sind, ist dies keine große Einschränkung, da `QWidget` seinerseits von `QObject` abgeleitet ist.
3. **Klassen mit eigenen Slots bzw. Signalen müssen Q\_OBJECT aufrufen**  
Hinter dem Makro `Q_OBJECT` darf kein Semikolon angegeben werden.

Mit diesen Kenntnissen können wir uns nun eine eigene Klasse `Schrift` definieren, die von der Klasse `QLabel` abgeleitet ist und einen eigenen Slot `stelle_neu_dar(int)` zur Neuanzeige des Textes (mit Schiebepfeil-Wert als Schriftgröße) anbietet. Dazu erstellen wir eine eigene Headerdatei `text_groes.h`:

```
#include <qlabel.h>
class Schrift : public QLabel {
    Q_OBJECT    // notwendig, da Schrift Slots enthält
public:
    Schrift( char const* text, QWidget *parent ) : QLabel( text, parent ) { }
public slots:
    void stelle_neu_dar( int groesse ) {
        setFont( QFont( "Times", groesse ) );
        repaint();
    }
};
```

Im Slot `stelle_neu_dar(int groesse)` wird als neue Schriftgröße für den auszugebenden Text der über den Parameter `groesse` gelieferte Wert eingestellt. Mit der Methode `repaint()` wird dann veranlasst, dass der Text auch wirklich mit dem neuen Font im Label angezeigt wird. Mit dieser neuen Klassendefinition können wir nun unser Programm `text_groes.cpp` erstellen. Die sind gegenüber dem vorherigen Programm 2.2 neu hinzugekommenen Konstrukte fett hervorgehoben.

Programm 2.3 – `text_groes.cpp`:

Schiebepfeil und zwei Buttons zum Ändern der Größe eines Textes

```
1  #include <qapplication.h>
2  #include <qpushbutton.h>
3  #include <qslider.h>
4  #include <qlcdnumber.h>
5  #include <qlabel.h>
6  #include "text_groes.h" // enthaelt neue Klasse 'Schrift'
7                          // mit eigenem Slot 'stelle_neu_dar'
8  int main( int argc, char* argv[] )
9  {
10     QApplication myapp( argc, argv );
11
12     QWidget* mywidget = new QWidget();
13     mywidget->setGeometry( 400, 300, 460, 150 );
14
15     //.... Erzeugen eines Schiebepfeils
16     QSlider* myslider = new QSlider( 0, // kleinstmögl. Wert
17                                     99, // größtmögl. Wert
18                                     1, // Schrittweite
19                                     20, // Startwert
20                                     QSlider::Horizontal, // Richtung
21                                     mywidget ); // Elternwidget
22     myslider->setGeometry( 10, 10, 180, 30 );
23
24     //.... Erzeugen eines Widgets zur Anzeige von LCD-Nummern
25     QLCDNumber* mylcdnum = new QLCDNumber( 2, // Ziffernzahl
26                                             mywidget ); // Elternwidget
```

## 2 Grundlegende Konzepte und Konstrukte von Qt

---

```
27 mylcdnum->setGeometry( 60, 50, 80, 50 );
28 mylcdnum->display( 20 ); // zeige Startwert an
29
30 // Verbinde Schiebepalken und Nummernanzeige
31 QObject::connect( myslider, SIGNAL( valueChanged( int ) ),
32                  mylcdnum, SLOT( display( int ) ) );
33
34 // Zwei Buttons zum schrittweisen Erhöhen und
35 // Erniedrigen der Schiebepalken-Werte
36 QPushButton* decrement = new QPushButton( "<", mywidget );
37 decrement->setGeometry( 10, 110, 50, 30 );
38 decrement->setFont( QFont( "Times", 18, QFont::Bold ) );
39 decrement->setPaletteBackgroundColor( Qt::red );
40
41 QPushButton* increment = new QPushButton( ">", mywidget );
42 increment->setGeometry( 140, 110, 50, 30 );
43 increment->setFont( QFont( "Times", 18, QFont::Bold ) );
44 increment->setPaletteBackgroundColor( Qt::green );
45
46 // Verbinde das clicked()-Signal der Buttons mit den Slots, die
47 // den Schiebepalken-Wert erhöhen bzw. erniedrigen
48 QObject::connect( decrement, SIGNAL( clicked() ),
49                  myslider, SLOT( subtractStep() ) );
50 QObject::connect( increment, SIGNAL( clicked() ),
51                  myslider, SLOT( addStep() ) );
52
53 // Label zur Anzeige der Schrift(-größe)
54 Schrift* anzeige = new Schrift( "Text", mywidget );
55 anzeige->setGeometry( 200, 10, 250, 130 );
56 anzeige->setFont( QFont( "Times", 20 ) );
57 anzeige->setPaletteBackgroundColor( Qt::white );
58
59 // Verbinde Schiebepalken und Label (für Textanzeige)
60 QObject::connect( myslider, SIGNAL( valueChanged( int ) ),
61 anzeige, SLOT( stelle_neu_dar( int ) ) );
62
63 myapp.setMainWidget( mywidget );
64 mywidget->show();
65 return myapp.exec();
66 }
```

Nachfolgend werden nun die neuen Anweisungen, die zum Verständnis von Programm 2.3 benötigt werden, näher erläutert:

*Zeile 54:* Hier wird zunächst ein Objekt `anzeige` der Klasse `Schrift` angelegt, was ein Label-Widget ist, da die Klasse `Schrift` von der Klasse `QLabel` abgeleitet ist. In diesem Label wird der Text „Text“ angezeigt.

*Zeile 55:* legt die Position und Größe des Widgets `anzeige` fest, und

*Zeile 56:* legt den zu verwendenden Font des auszugebenden Textes fest.

*Zeile 57:* Hier wird als Hintergrund für das Label-Widget `anzeige` die Farbe „Weiß“ (`Qt::white`) festgelegt.

Zeilen 60 und 61: legen fest, dass bei Änderung des Schiebepalken-Werts, was durch Schicken des Signals `valueChanged(int)` angezeigt wird, die von `anzeige` definierte Slotroutine `stelle_neu_dar(int)` auszuführen ist, was zu einer Anzeige des Textes mit der neuen Schriftgröße führt, die dem Schiebepalken-Wert entspricht.

Im Programm 2.3 wurden die in Abbildung 2.5 gezeigten Signal-/Slotverbindungen zwischen den einzelnen Objekten eingerichtet.

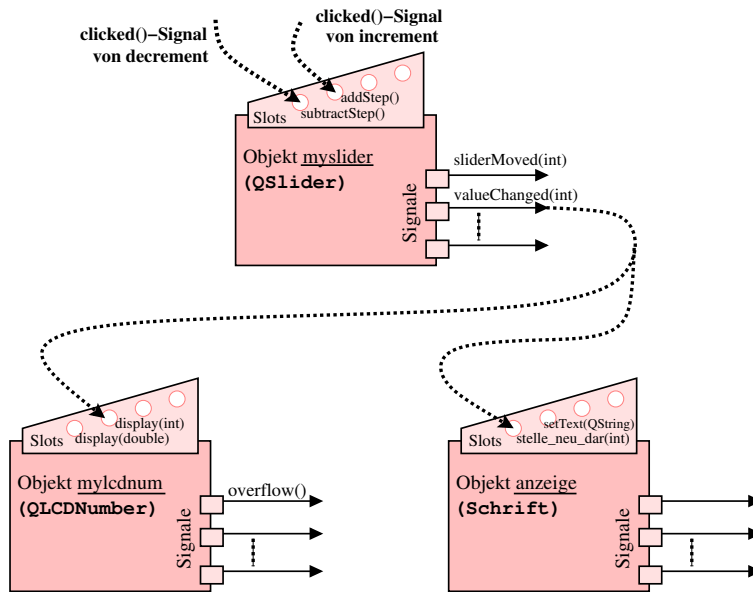


Abbildung 2.5: Signal-/Slotverbindungen im Programm 2.3

Immer wenn man Klassen definiert, die eigene Slots und/oder Signale definieren, muss man diese zunächst mit dem bei der Qt-Distribution mitgelieferten *Meta-Object-Compiler* (`moc`) kompilieren. Verwendet man `qmake` zum Erstellen des `Makefiles`, so erkennt dieses Tool automatisch, dass ein entsprechender `moc`-Aufruf im `Makefile` generiert werden muss. Nehmen wir z. B. für unser Programm hier die folgende Datei `text_groes.pro`:

```
CONFIG += qt warn_on release
SOURCES = text_groes.cpp
HEADERS = text_groes.h
TARGET = text_groes
```

und rufen dann

```
qmake text_groes.pro -o Makefile
```

auf, so wird ein `Makefile` generiert.

Ein anschließender `make`-Aufruf führt dann zur Generierung eines ausführbaren Programms.

Neben dem expliziten Dazulinken einer von einem `moc`-Aufruf erzeugten `moc`-Datei besteht auch die Möglichkeit, eine solche `moc`-Datei mithilfe von

```
#include "mocdatei.cpp"
```

in der entsprechenden Quelldatei zu inkludieren.

### 2.2.3 Regeln für die Deklaration eigener Slots und Signale

Hier werden nochmals wichtige Regeln, die beim Signal-Slot-Konzept gelten, zusammengefasst. Sie ergänzen die auf Seite 24 aufgeführte Liste.

1. Slots können wie jede andere C++-Methode deklariert und implementiert werden. Slots sind eigentlich Methoden, die auch wie diese außerhalb eines `connect()`-Aufrufs direkt aufgerufen werden können. Umgekehrt können Methoden nicht als Argument eines `connect()`-Aufrufs angegeben werden.
2. Bei der Definition von Slots muss nur zuvor das Schlüsselwort `slots` zum entsprechenden Schutztyp-Schlüsselwort `public` bzw. `private` hinzugefügt werden. Natürlich ist es auch möglich, `protected slots:` zu definieren und diese als `break virtual` zu deklarieren. Während Methoden auch `static` deklariert sein können, ist dies bei Slots nicht erlaubt.
3. Slots können wie Methoden Parameter besitzen. Es ist dabei nur zu beachten, dass das bei einem `connect()`-Aufruf angegebene Signal die gleichen Parametertypen besitzt wie der entsprechende Slot. Ein Slot kann dabei jedoch auch weniger Parameter haben als das mit ihm verbundene Signal, wenn er diese vom Signal gelieferten Parameter nicht alle benötigt.
4. Die Syntax für Slot-Namen entspricht der für Methoden. Einige Programmierer betten jedoch die Zeichenkette `slot` in den Namen von Slots ein, um diese sofort als Slots identifizieren zu können. Dieser Konvention folgt Qt bei den Namen seiner vordefinierten Slots jedoch nicht.
5. Um Signale in einer Klasse zu definieren, muss zuvor das Schlüsselwort `signals:` angegeben werden. Andernfalls entspricht die Deklaration von Signalen der von anderen Memberfunktionen, mit einem wichtigen Unterschied, dass Signale nur deklariert und niemals direkt implementiert werden dürfen.
6. Zum Senden eines Signals in einer Komponente steht das Qt-Schlüsselwort `emit` zur Verfügung. Wenn z. B. das Signal `void farbe_geaendert(int)` in der Klassendefinition deklariert wurde, wäre z. B. folgender Aufruf möglich:

```
emit farbe_geaendert(173);
```

7. Die Verbindung von Signalen und Slots erfolgt mit `QObject::connect()`. Diese Methode wird in überladenen Varianten angeboten, hier aber werden wir nur die statische Variante mit vier Parametern verwenden:

```
QObject::connect( signal_object,          // Objekt, das das Signal schickt
                  SIGNAL(signal_name(...)), // Signal, das mit Slot zu verbinden
                  slot_object,            // Objekt, das Signal empfängt
                  SLOT(slot_name(...)) ); // Slot, der mit Signal zu verbinden
```

Es können sowohl eine beliebige Anzahl von Slots mit einem Signal als auch umgekehrt eine beliebige Anzahl von Signalen mit einem Slot verbunden werden. Da die Reihenfolge, in der Slots aufgerufen werden, bisher noch nicht von Qt festgelegt ist, kann man sich nicht darauf verlassen, dass Slots auch in der Reihenfolge aufgerufen werden, in der sie mittels `connect()` mit Signalen verbunden wurden.

8. Bei den SIGNAL- und SLOT-Routinen sind immer nur Typen als Parameter erlaubt. Ein häufiger Fehler ist, dass hier versucht wird, einen Wert statt eines Typs anzugeben, wie z. B. im Folgenden, wo versucht wird, bei Auswahl des vierten Menüpunkts die Slotroutine `quit()` aufzurufen:

```
QObject::connect( menu, SIGNAL( activated( 4 ) ), // Falsch
                 qApp, SLOT( quit( int ) ) );
```

Dieser Code ist zwar naheliegend, aber nicht erlaubt, stattdessen muss Folgendes angegeben werden:

```
QObject::connect( menu, SIGNAL( activated( int ) ), // Richtig
                 qApp, SLOT( menuBehandle( int ) ) );
```

In der Slotroutine `menuBehandle()` muss dann der übergebene Parameter ausgewertet werden, wie z. B.

```
void menuBehandle(int meupkt) {
    switch (menuPkt) {
        ...
        case 3: qApp->quit();
        ...
    }
}
```

## 2.3 Die Klasse `QString` für Zeichenketten

Die Klasse `QString` ist eine Abstraktion zu Unicode-Text und zum String-Konzept im klassischen C (ein `char`-Array (`char *`), in dem das Stringende durch ein 0-Byte gekennzeichnet ist).

In allen `QString`-Methoden, die `const char*`-Parameter akzeptieren, werden diese Parameter als klassische C-Strings interpretiert, die mit einem 0-Byte abgeschlossen sind. Es ist zu beachten, dass Funktionen, die C-Strings in ein `QString`-Objekt kopieren, das abschließende 0-Byte nicht mitkopieren, da ein String in einem `QString`-Objekt nicht wie in klassischen C-Strings mit einem 0-Byte abgeschlossen ist, sondern die Länge des jeweiligen String intern mitgeführt wird. Ein nicht-initialisiertes `QString`-Objekt hat den Wert `null`, was bedeutet, dass seine interne Länge und sein Daten-Zeiger auf 0 gesetzt sind.

Hier ist noch zu erwähnen, dass man in der Objektorientierung zwei Möglichkeiten unterscheidet, um ein Objekt zu kopieren:

- ❑ „tiefe Kopie“ (*deep copy*): Hierbei wird das entsprechende Objekt vollständig dupliziert, was abhängig von der Objektgröße sehr speicher- und zeitintensiv ist.
- ❑ „flache Kopie“ (*shallow copy*): Hierbei wird nur der Zeiger auf den gemeinsam nutzbaren Datenblock kopiert und der Referenzzähler um 1 erhöht, was wesentlich schneller als ein tiefes Kopieren ist. Diese Möglichkeit wird im Kapitel „Data Sharing“ ab Seite 312 näher erläutert.

### 2.3.1 Wichtige Konstruktoren und Methoden

#### Wichtige Konstruktoren der Klasse `QString`

<code>QString()</code>	legt einen <code>null</code> -String an
<code>QString(QChar ch)</code>	legt String an, der das Zeichen <code>ch</code> enthält
<code>QString(QByteArray&amp; s)</code>	legt eine tiefe Kopie zum String <code>s</code> an
<code>QString(QString&amp; s)</code>	legt Kopie ( <i>implicit sharing</i> ) von String <code>s</code> an
<code>QString(QChar *s, uint n)</code>	legt tiefe Kopie zu den ersten <code>n</code> Zeichen von <code>s</code> an

# Kapitel 3

## Die wesentlichen Qt-Widgets

*Willst du dich am Ganzen erquicken,  
so musst du das Ganze im Kleinsten erblicken.*

– J. W. Goethe

Dieses Kapitel gibt einen Überblick über wichtige von Qt angebotenen Widgets:

### Buttons

<code>QPushButton</code>	Pushbutton mit einem Text oder einer QPixmap als Label
<code>QRadioButton</code>	Radiobutton mit Text oder QPixmap als Label. Radiobuttons erlauben dem Benutzer, aus mehreren Alternativen eine auszuwählen.
<code>QCheckBox</code>	Checkbox mit einem Text oder einer QPixmap als Label. Checkboxes erlauben dem Benutzer, aus mehreren Alternativen keine, eine oder auch mehrere auszuwählen.
<code>QButtonGroup</code>	Mehrere zu einer Gruppe zusammengefasste Buttons.

### Auswahl-Widgets

<code>QListBox</code>	Liste von Alternativen, die durchblättert werden kann.
<code>QComboBox</code>	Kombination aus einem Button und einer Listbox.

### Schieberegler-, Drehknopf- und Spinbox-Widgets

<code>QSlider</code>	Horizontaler oder vertikaler Schieberegler.
<code>QDial</code>	Drehknopf zum Einstellen eines Wertes.
<code>QSpinBox</code>	Texteingabefeld mit zwei Pfeil-Buttons zum Erhöhen bzw. Erniedrigen der Zahl im Texteingabefeld; direkte Eingabe der gewünschten Zahl im Texteingabefeld ist dabei auch möglich.
<code>QDateEdit</code>	Spinbox zur Eingabe eines Datums
<code>QTimeEdit</code>	Spinbox zur Eingabe einer Zeit
<code>QDateTimeEdit</code>	Spinbox zur Eingabe eines Datums und einer Zeit

### Texteingabe

<code>QLineEdit</code>	Einzeiliges Texteingabefeld.
<code>QTextEdit</code>	Editor mit RichText-Unterstützung.



#### Widgets zum Anzeigen von Informationen

QLabel	Anzeige von Text, Pixmaps, Vektorgraphiken oder Animationen.
QTextBrowser	Komfortable Anzeige von Text im RichText-Format mit Hypertext-Navigation und automatische Laufbalken bei größeren Texten.
QLCDNumber	Anzeige für Zahlen oder Texte in 7-Segment-LCD-Darstellung.
QIconView	Widget mit verschiebbaren Icons

#### Menüs

QMenuData	Basisklasse für für QMenuBar und QPopupMenu. Diese Klasse enthält Methoden zum Einfügen von Menüpunkten.
QMenuBar	Horizontale Menüleiste zum Einfügen von Menüeinträgen.
QPopupMenu	Popupmenüs, die bei rechten Mausklick bzw. bei Klick auf einen Menüpunkt in der Menüleiste eingeblendet werden.

#### Hauptfenster mit Menüleiste, Werkzeugleisten, Statuszeile und Hilfstexten

QMainWindow	Typisches Hauptfenster für eine Applikation mit einer Menüleiste, einer oder mehreren Werkzeugleisten und einer Statuszeile.
QToolBar	Verschiebbare Werkzeugleiste, in der QToolButton-Objekte in Form von Pixmaps oder als Text eingefügt werden können.
QToolButton	Pushbuttons für eine Werkzeugleiste (der Klasse QToolBar).
QToolTip	Einblenden einer kurzen Information zu einem Widget, wie z. B. zu einem Pushbutton in einer Werkzeugleiste.
QToolTipGroup	Einblenden einer kurzen Information zu einem Widget mit gleichzeitiger Anzeige einer längeren Hilfsinformation in der Statuszeile.
QWhatsThis	Einblenden umfangreicherer Information zu einem Widget, wie z. B. zu einem Menüpunkt oder zu einem Werkzeugleisten-Button.
QStatusBar	Horizontale Statuszeile zur Anzeige von Statusmeldungen.

#### Füllbalken

QProgressBar	Horizontaler Füllbalken (zur Fortschrittsanzeige).
QProgressDialog	Dialogfenster mit Text, Füllbalken und <i>Cancel</i> -Button.

#### Listenansichten

QListView	Widget zum Anzeigen von Informationen in Baumform.
QListViewItem	Einträge in einem QListView-Widget.

#### Fenster mit Laufbalken (Scrollviews)

QScrollView	Widget mit einer Fensterfläche, zu der zwei Laufbalken hinzugefügt sind, die automatisch verwaltet werden.
QScrollBar	Vertikale oder horizontale Laufbalken als eigene Widgets.

#### Tabellen

QGridView	Anzeige von Information in Tabellenform.
QTable	Komfortable Tabellen im Spreadsheet-Stil.

## 3.1 Allgemeine Widget-Methoden und -Parameter

### 3.1.1 Allgemeine Widget-Methoden

Alle in diesem Kapitel vorgestellten Widgets sind von der Klasse `QWidget` abgeleitet und bieten somit auch die Methoden, Slots und Signale dieser Klasse an. Einige wichtige solcher allgemeinen Methoden sind:

```
setEnabled(bool enable)
```

legt fest, ob Widget Benutzerinteraktionen zulässt

```
setFont(QFont& font)
```

legt den Zeichensatz für Textausgaben fest.

Die Klasse `QFont` bietet die folgenden Konstruktoren an:

```
QFont(QString& family, int size=12,  
       int weight=Normal, bool italic=false)
```

```
QFont(QFont&)
```

```
setPalette(QPalette & p)
```

legt die im Widget zu verwendende Farbpalette fest.

```
setGeometry(int x, int y, int w, int h)
```

```
setGeometry(QRect& r)
```

bestimmen die Position des Widgets im Elternwidget bzw. auf dem Bildschirm. Die Klasse `QRect` folgende Konstruktoren bereit:

```
QRect(QPoint& topleft, QPoint& bottomright)
```

```
QRect(QPoint& topleft, QSize& size)
```

```
QRect(int left, int top, int width, int height)
```

Die Klasse `QPoint` bietet folgenden Konstruktor an:

```
QPoint(int xpos, int ypos)
```

und die Klasse `QSize` bietet folgenden Konstruktor an:

```
QSize(int w, int h)
```

```
setMinimumSize(int w, int h)
```

```
setMaximumSize(int w, int h)
```

```
setMinimumSize(QSize& s)
```

```
setMaximumSize(QSize& s)
```

```
setMinimumWidth(int minw)
```

```
setMaximumWidth(int maxw)
```

```
setMinimumHeight(int minh)
```

```
setMaximumHeight(int maxh)
```

legen minimale/maximale Größe, Breite und Höhe fest, auf die Widget verkleinert/vergrößert werden kann

```
QSize minimumSizeHint()
```

```
QSize sizeHint()
```

liefern eine Empfehlung für die (minimale) Größe des Widgets, oder aber eine ungültige Größe, wenn dies nicht möglich ist.

```
bool QSize::isValid()
```

liefert `true`, wenn sowohl Breite als auch Höhe des Widgets größer oder gleich 0 ist, ansonsten liefert diese Methode `false`.

### 3.1.2 Parameter-Konventionen für die meisten Konstruktoren

Die meisten Qt-Konstruktoren für Widgets verfügen über eine gleiche Teil-Schnittstelle, nämlich die folgenden Parameter, die auch immer in dieser Reihenfolge vorliegen:

<code>QWidget *parent=0</code>	Elternwidget
<code>const char *name=0</code>	interner Widgetname (für Debugging-Zwecke)
<code>WFlags f=0</code>	Widgetflags; nur bei Toplevel-Widgets

Da die Default-Werte dieser Parameter 0 sind, kann man Toplevel-Widgets ohne Angabe jeglicher Argumente beim Aufruf des Konstruktors erzeugen. Bei Widgets, die keine Toplevel-Widgets sind, muss immer zumindest das Elternwidget (*parent*) angegeben werden. Einige Klassen bieten weitere überladene Konstruktoren an, die zusätzliche Parameter besitzen, wie z. B. Text, der in einem Widget anzuzeigen ist. Grundsätzlich hält sich Qt jedoch an die Konvention, dass sich zusätzliche Parameter in einem Konstruktor immer vor den drei obigen Standard-Parametern befinden. Um dies für Konstrukturen weiter zu verdeutlichen, sind nachfolgend einige Beispiele für Widget-Konstrukturen aufgelistet:

```
QWidget( QWidget *parent=0, const char *name=0, WFlags f=0 )

QPushButton( QWidget *parent, const char *name=0 )
QPushButton( const QString& text, QWidget *parent, const char *name=0 )

QCheckBox( QWidget *parent, const char *name=0 )
QCheckBox( const QString& text, QWidget *parent, const char *name=0 )

QSlider( QWidget *parent, const char *name=0 )
QSlider( Orientation orientation, QWidget* parent, const char *name=0 )
QSlider( int minValue, int maxValue, int pageStep, int value,
         Orientation orientation, QWidget *parent, const char *name=0 )

QListBox( QWidget *parent=0, const char *name=0, WFlags f=0 )

QButtonGroup( QWidget *parent=0, const char *name=0 )
QButtonGroup( const QString& title, QWidget *parent=0, const char *name=0 )
QButtonGroup( int strips, Orientation o, QWidget* parent=0, const char* name=0 )
```

## 3.2 Der Widget-Stil

Qt ist in der Lage, Widgets im Windows-Stil unter Linux/Unix oder aber auch Widgets im Motif-Stil unter Windows anzuzeigen. Es ist sogar möglich, Widgets mit unterschiedlichen Stilen in einer Anwendung zu mischen, was allerdings nicht empfehlenswert ist. Zum Festlegen des zu verwendenden Widget-Stils gibt es mehrere Möglichkeiten:

1. durch Aufruf von  
`setStyle(new QWindowsStyle())` oder  
`setStyle(new QMotifStyle())` für das entsprechende Widget.
2. durch Aufruf der statischen Methode  
`QApplication::setStyle(new QWindowsStyle())` bzw.  
`QApplication::setStyle(new QMotifStyle())`. Dadurch wird als Voreinstellung für alle Widgets der entsprechende Stil festgelegt. Dies sollte aber erfolgen, bevor bereits ein Widget erzeugt wurde, andernfalls ist eine Neudarstellung aller existierenden Widgets notwendig, was doch einige Zeilen Code erfordert.

3. durch Angabe einer der beiden Optionen `-style=windows` oder `-style=motif` auf der Kommandozeile beim Programmaufruf. Diese Möglichkeit funktioniert jedoch nur, wenn keine der beiden vorherigen Möglichkeiten verwendet wird.

Neben den beiden hier erwähnten Stilarten `QWindowsStyle` (`-style=windows`) und `QMotifStyle` (`-style=motif`) bietet Qt noch weitere Stilarten (*look and feel*) an, die in Tabelle 3.1 gezeigt sind.

Tabelle 3.1: Stilarten

Klasse (Option)	Stilart
<code>QMotifPlusStyle</code> ( <code>-style=motifplus</code> )	Verbessertes Motif-look-and-feel
<code>QCDEStyle</code> ( <code>-style=cde</code> )	CDE (Common Desktop Environment)
<code>QSGIStyle</code> ( <code>-style=sgi</code> )	SGI-look-and-feel
<code>QPlatinumStyle</code> ( <code>-style=platinum</code> )	Platinum (ähnlich Macintosh-Stil)
<code>QMacStyle</code> ( <code>-style=mac</code> )	Aqua-Stil von MacOS X
<code>QXPStyle</code> ( <code>-style=xp</code> )	Windows XP-Stil auf Windows XP
<code>QCompactStyle</code> ( <code>-style=compact</code> )	für Qt/Embedded (ähnlich <code>QWindowsStyle</code> )

### 3.3 Properties von Widgets

Objekte der Klassen, die von `QObject` abgeleitet sind, können so genannte *Properties* besitzen, die in den entsprechenden Seiten zu den jeweiligen Klassen in der Online-Dokumentation als eigener Unterpunkt angegeben sind. *Properties* sind lediglich Namen (Strings), denen entsprechende Methoden zum Setzen bzw. zum Erfragen bestimmter Widget-Eigenschaften oder auch andere Methoden zugeordnet sind. So hat die Klasse `QSlider` z. B. unter anderem eine Property `int maxValue`, der die Methode `setMaxValue()` zum Setzen des maximalen Werts eines Schiebebalkens zugeordnet ist. Somit ist es dann möglich, statt

```
sliderObj->setMaxValue(50);
```

auch Folgendes aufzurufen:

```
sliderObj->setProperty("maxValue", 50);
```

Da solcher Code aber nicht unbedingt lesbarer ist, ist von der Verwendung der Methode `setProperty()` abzuraten. Properties sind nur für Entwicklungswerkzeuge wie dem Qt-Designer vorgesehen, die explizit die angebotenen Eigenschaften (Properties) als Strings anbieten, um sie vom Benutzer auswählen zu lassen.

Um eigene Properties zu definieren, muss man das Makro `Q_PROPERTY()` verwenden. Interessierte Leser seien hier auf die Online-Dokumentation verwiesen.

Programm 3.1 erzeugt zwei Schiebebalken mit LCD-Anzeige, wobei die Eigenschaften des oberen Schiebebalkens und der zugehörigen LCD-Anzeige mit den entsprechenden Methoden gesetzt werden, während diese Eigenschaften für den unteren Schiebebalken und die zugehörige LCD-Anzeige mit `setProperty()` festgelegt werden (siehe auch Abbildung 3.1).

## 3.6 Schiebebalken, Drehknöpfe und Spinboxen

Mit diesen Widgets kann man den Benutzer einen numerischen Wert aus einem bestimmten Wertebereich auswählen lassen. Während man bei Schiebebalken (*sliders*) und Drehknöpfen eine feste Unter- und Obergrenze für den Wertebereich festlegen kann, die der Benutzer nicht unter- oder überschreiten kann, ist dies bei Spinboxen, die auch eine direkte Eingabe eines Wertes durch den Benutzer erlauben, nicht garantiert.

### 3.6.1 Schiebebalken (QSlider)

Um einen Schiebebalken zu erzeugen, muss der Konstruktor der Klasse `QSlider` aufgerufen werden, wie z. B.:

```
QSlider* schiebbalk = new QSlider( 1, 150, // Unter- und Obergrenze
                                   5, 30,  // Schrittweite und Startwert
                                   QSlider::Horizontal, // Ausrichtung; auch Vertical moegl.
                                   elternwidget, "name");
```

Dieser Aufruf erzeugt einen horizontalen Schiebebalken für den Zahlenbereich von 1 bis 150. Klickt man mit der Maus in den Balken links bzw. rechts vom Knopf, so wird der Schiebebalken-Knopf um den Wert 5 nach links bzw. nach rechts geschoben. Beim Anlegen des Schiebebalkens wird der Knopf auf die Position gestellt, die dem Startwert 30 entspricht. Daneben bietet die Klasse `QSlider` noch zwei weitere Konstruktoren an:

```
QSlider(Orientation o, QWidget *parent=0, const char *name=0)
QSlider(QWidget *parent=0, const char *name=0)
```

Die hier fehlenden Spezifikationen können dann nachträglich mit folgenden Methoden festgelegt werden:

```
setMinValue(int) , setMaxValue(int) , setOrientation(Orientation) ,
setPageStep(int) , setValue(int)
```

Neben diesen Methoden ist noch die Folgende erwähnenswert:

```
setTickmarks(int)
```

legt fest, ob der Schiebebalken zusätzlich mit einer Skala zu versehen ist oder nicht. Als Argument kann dabei `NoMarks` (Voreinstellung), `Above`, `Left`, `Below`, `Right` oder `Both` angegeben werden.

Folgende Signale schickt ein `QSlider`-Widget, wenn:

```
sliderMoved(int)
```

Schiebebalken-Knopf bewegt wird. Über den Parameter wird dabei der neue eingestellte Positionswert des Schiebebalkens zur Verfügung gestellt.

```
valueChanged(int)
```

Schiebebalken-Knopf neu positioniert wurde, also nachdem der Benutzer die Maustaste wieder losgelassen hat. Über den Parameter wird dabei der neu eingestellte Positionswert des Schiebebalkens zur Verfügung gestellt.

```
sliderPressed() , sliderReleased()
```










Benutzer den Schiebebalken-Knopf mit der Maus anklickt bzw. wieder loslässt.

### 3.6.2 Drehknopfeinstellungen (QDial)

Die Klasse `QDial` ist der Klasse `QSlider` sehr ähnlich, nur dass Sie den Benutzer einen Wert nicht über einen Schiebebalken, sondern über einen Drehknopf (ähnlich einem Potentiometer) einstellen lässt. Um ein `QDial`-Objekt zu erzeugen, muss der Konstruktor der Klasse `QDial` aufgerufen werden, wie z. B.:

```
QDial *drehKnopf = new QDial( 0, 359, // Unter- und Obergrenze
                             1, // Schrittweite beim Drücken "Bild auf/ab"
                             45, // Startwert
                             elternwidget, "name" );
```

Dieser Aufruf erzeugt einen Drehknopf für den Zahlenbereich 0 bis 359.

Der Drehknopf-Zeiger kann nicht nur mit der Maus, sondern auch mit den folgenden Tasten gedreht werden:  und  erniedrigen den Wert bzw.  und  erhöhen den Wert um die mit `setLineStep(int)` eingestellte `lineStep`-Schrittweite.  erniedrigt den Wert um die eingestellte `pageStep`-Schrittweite, und  erhöht den Wert um die eingestellte `pageStep`-Schrittweite.  (bzw. ) setzt den Drehknopf-Zeiger auf den kleinstmöglichen,  auf den größtmöglichen Wert. Daneben bietet die Klasse `QDial` noch folgenden Konstruktor an:

```
QDial(QWidget *parent=0, const char *name=0)
```

Die hier fehlenden Spezifikationen können dann nachträglich mit folgenden Methoden festgelegt werden:

```
setMinValue(int) ,    setMaxValue(int) ,
setPageStep(int) ,    setValue(int)
```

Einige wichtige Methoden, die die Klasse `QDial` anbietet, sind:

```
setWrapping(bool enable) virtual slot
    legt fest, ob der Drehknopf beim Erreichen des höchsten bzw. niedrigsten Werts
    weitergedreht werden kann (enable=true) oder nicht (enable=false). Bei
    enable=true wird der Drehknopf als vollständiger Kreis, andernfalls mit einem
    Zwischenraum zwischen dem niedrigsten und dem höchsten Wert angezeigt.

setNotchTarget(double target) virtual
    legt die Anzahl der Pixel zwischen den Skalenstrichen fest.

setNotchesVisible(bool b) virtual slot
    legt fest, ob eine Skala anzuzeigen ist (b=true) oder nicht (b=false).
```

Folgende Signale schickt ein `QDial`-Widget, wenn ...:

```
dialMoved(int value) , valueChanged(int value)
    Drehknopf-Zeiger bewegt wird bzw. neu positioniert wurde, also nachdem der Be-
    nutzer den Zeiger mit der Maus wieder losgelassen hat. Über den Parameter wird
    dabei der neu eingestellte Positionswert des Drehknopfs zur Verfügung gestellt.

dialPressed() , dialReleased()
    Benutzer den Drehknopf-Zeiger mit der Maus anklickt bzw. wieder loslässt.
```

#### 3.6.3 Spinboxen (`QSpinBox`)

Spinboxen setzen sich aus einem Texteingabefeld und zwei Pfeil-Buttons zum Erhöhen und zum Erniedrigen der Zahl im Texteingabefeld zusammen. Um dem Benutzer auch die direkte Eingabe seines gewünschten Werts zu ermöglichen, steht ihm das Texteingabefeld der Spinbox zur Verfügung. Die Klasse `QSpinBox` bietet die folgenden Konstruktoren an:

```
QSpinBox(int min, int max, int s=1, QWidget *p=0, char *nam=0)
    legt ein QSpinBox-Objekt mit einem Zahlenbereich von min bis max und der Schritt-
    weite s an. Der aktuelle Wert wird dabei auf min gesetzt.

QSpinBox(QWidget *par=0, char *nam=0)
    legt ein QSpinBox-Objekt mit einem Zahlenbereich von 0 bis 99 und der Schrittwei-
    te 1 an. Der aktuelle Wert wird hier auf 0 gesetzt.
```

Einige wichtige von einem `QSpinBox`-Objekt angebotene Methoden sind:

```
setRange(int minValue, int maxValue)
```

zum Festlegen des Zahlenbereichs für eine Spinbox. Mittels der Pfeil-Buttons kann sich der Benutzer nicht außerhalb dieses Zahlenbereichs klicken. Normalerweise wird bei Erreichen der festgelegten Unter- bzw. Obergrenze nicht weitergezählt. Soll der Wert in diesem Fall automatisch von der Untergrenze auf die Obergrenze umspringen bzw. umgekehrt, muss dies mit `setWrapping(true)` eingestellt werden.

```
setValue(int) , value()
```

setzt bzw. liefert den aktuellen Wert in einer Spinbox

```
setPrefix(const QString& text) , setSuffix(const QString& text)
```

legen Text fest, der vor bzw. nach dem Zahlenwert im Texteingabefeld einer Spinbox anzuzeigen ist; z. B. `setPrefix("Euro")` oder `setSuffix("km")`.

```
setSpecialValueText(const QString& text)
```

legt einen Text fest, der anstelle des kleinsten Werts anzuzeigen ist.

```
setButtonSymbols(ButtonSymbols newSymbols)
```

legt Buttonanzeige fest: `UpDownArrows` ( $\Delta$  und  $\nabla$ ) oder `PlusMinus` (+ und -).

Ein `QSpinBox`-Objekt schickt die beiden folgenden Signale

```
valueChanged(int) und valueChanged(const QString&),
```

wenn der Wert der Spinbox geändert wird. Die erste Variante liefert über den Parameter den neu eingestellten Wert, während die zweite den ganzen String aus dem Texteingabefeld liefert, welcher auch das Präfix und Suffix enthält, wenn solches festgelegt wurde.

### 3.6.4 Eingabe von Datum und Zeit (`QDateEdit`, `QTimeEdit`, `QDateTimeEdit`)

Die hier vorgestellten Klassen sind Spinboxen, die speziell zur Eingabe eines Datums bzw. einer Zeit oder beides dienen.

#### Die Klasse `QDateEdit` zur Eingabe eines Datums

Beim Anlegen eines `QDateEdit`-Objekts sollte man dies bereits initialisieren, wie z. B.

```
//.... Initialisieren mit dem heutigen Datum
QDateEdit *dateEdit = new QDateEdit( QDate::currentDate(), this );

//.... Änderungen bis 365 Tage vor und nach heutigen Datum erlaubt
dateEdit->setRange( QDate::currentDate().addDays( -365 ),
                  QDate::currentDate().addDays( 365 ) );

//... Datum-Format ist: Tag.Monat.Jahr
dateEdit->setOrder( QDateEdit::DMY );
dateEdit->setSeparator( "." );
```

Einige wichtige Methoden und Signale der Klasse `QDateEdit` sind:

```
setRange(const QDate& min, const QDate& max)
```

```
setMinValue(const QDate& d) , setMaxValue(const QDate& d)
```

legen wie bei Klasse `QSpinBox` den erlaubten Eingabebereich fest

```
setOrder(order)
```

legt Datumsformat fest, wobei für Parameter `order` Folgendes anzugeben ist:

<code>QDateEdit::MDY</code>	Monat-Tag-Jahr	<code>QDateEdit::DMY</code>	Tag-Monat-Jahr
<code>QDateEdit::YMD</code>	Jahr-Monat-Tag	<code>QDateEdit::YDM</code>	Jahr-Tag-Monat

### 3 Die wesentlichen Qt-Widgets

---

```
setSeparator(const QString& s)
```

legt das Trennzeichen fest

```
setAutoAdvance(true)
```

legt fest, dass der Fokus automatisch auf die nächste Komponente weiterbewegt wird, wenn eine Komponente vollständig eingegeben wurde. So wird automatisch auf das Jahr positioniert, wenn man z. B. für den Monat 10 eingegeben hat.

Das folgende Signal wird immer dann geschickt, wenn sich das Datum ändert:

```
valueChanged(const QDate& date)
```

aktuell eingestelltes Datum wird als `QDate`-Argument mitgeschickt.

#### Die Klasse `QTimeEdit` zur Eingabe einer Zeit

Beim Anlegen eines `QTimeEdit`-Objekts sollte dies bereits initialisiert werden, wie z. B.

```
//.... Initialisieren mit aktueller Zeit
QTime timeNow = QTime::currentTime();
QTimeEdit *timeEdit = new QTimeEdit( timeNow, this );
//.... Änderungen nur zwischen aktueller Zeit bis eine Stunde später erlaubt
timeEdit->setRange( timeNow, timeNow.addSecs( 60 * 60 ) )
```

Die Klasse `QTimeEdit` kennt bis auf `setOrder()` die gleichen Methoden, die zuvor bei der Klasse `QDateEdit` vorgestellt wurden, nur dass diese sich hierbei eben auf die Zeit beziehen. Zusätzlich ist folgende Methode vorhanden:

```
setDisplay(uint disp)
```

legt fest, welche Zeitkomponenten anzuzeigen sind, wobei für *disp* folgende Konstanten, die mit bitweisem OR (`|`) verknüpft werden können, erlaubt sind:

<code>QTimeEdit::Hours</code>	Stunden	<code>QTimeEdit::Minutes</code>	Minuten
<code>QTimeEdit::Seconds</code>	Sekunden	<code>QTimeEdit::AMPM</code>	AM/PM

Folgendes Signal wird immer dann geschickt, wenn sich die Zeit ändert:

```
valueChanged(const QTime& time)
```

gerade eingestellte Zeit wird als `QTime`-Argument mitgeschickt.

#### Die Klasse `QDateTimeEdit` zur Eingabe von Datum und Zeit

Ein `QDateTimeEdit`-Widget beinhaltet zugleich ein `QDateEdit`- und `QTimeEdit`-Widget. Ein `QDateTimeEdit`-Objekt sollte man bereits beim Anlegen initialisieren, wie z. B.

```
//.... Initialisieren mit akt. Datum und Zeit
QDateTimeEdit *dateTimeEdit = new QDateTimeEdit( QDateTime::currentDateTime(),
                                                    this );
//.... Änderungen von Datum nur ab heute bis in einer Woche
dateTimeEdit->dateEdit()->setRange( QDateTime::currentDate(),
                                   QDateTime::currentDate().addDays( 7 ) );
```

Die folgenden Methoden liefern Zeiger auf die beiden internen Widgets:

```
QDateEdit *dateEdit(), QTimeEdit *timeEdit()
```

Mit diesen Zeigern kann man dann alle zuvor vorgestellten Methoden dieser beiden Klassen nutzen. Daneben bietet `QDateTimeEdit` noch die Methode `setAutoAdvance()` an, sodass man für das ganze `QDateTimeEdit`-Widget das automatische Weiterschalten auf einmal einstellen kann, und dies nicht einzeln für jedes der beiden Subwidgets durchführen muss. Folgendes Signal wird geschickt, wenn Datum oder Zeit geändert wird, wobei aktuell eingestelltes Datum und Zeit als `QDateTime`-Argument mitgeschickt wird:

```
valueChanged(const QDateTime& datetime)
```



### 3.6.5 Beispiel zu Schieberegler und Spinboxen<sup>B</sup>

Programm 3.5 ermöglicht es dem Benutzer, sich eine zu einem bestimmten RGB-Wert gehörige Farbe direkt anzeigen zu lassen. Um die einzelnen RGB-Werte zu ändern, stehen ihm sowohl ein Schieberegler als auch eine Spinbox zur Verfügung. In der Spinbox wird vor dem 0-Wert (also an Indexstelle -1) der Text *Default* angezeigt. Wählt der Benutzer diesen *Default*-Eintrag, wird automatisch die Voreinstellung für diesen Farbwert (hier 120) angenommen (siehe Abbildung 3.8).

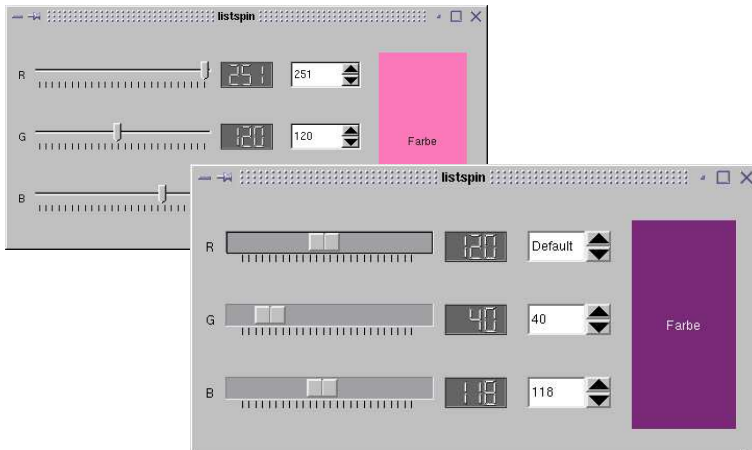


Abbildung 3.8: Schieberegler und Spinboxen zum Ändern von RGB-Werten einschließlich Farbanzeige im Windows- und Motif-Stil

Programm 3.5 – listspin.cpp:

Schieberegler und Spinboxen zum Ändern von RGB-Werten

```
#include ....
//..... SpinListLCD
class SpinListLCD : public QWidget {
    Q_OBJECT
public:
    SpinListLCD( QWidget *parent, const char* text ) : QWidget( parent, text ) {
        //... Text-Label
        name = new QLabel( this );
        name->setText( text );
        name->setGeometry( 0, 10, 15, 30 );
        //...Schieberegler
        schiebbalk = new QSlider( 0, 255, 5, 120, QSlider::Horizontal, this );
        schiebbalk->setTickmarks( QSlider::Below );
        schiebbalk->setTickInterval( 10 );
        schiebbalk->setGeometry( 20, 10, 200, 30 );
        //... LCD-Anzeige
        lcdzahl = new QLCDNumber( 3, this );
        lcdzahl->setSegmentStyle( QLCDNumber::Filled );
        lcdzahl->setPalette( QPalette( Qt::darkGray, Qt::darkGray ) );
```

### 3 Die wesentlichen Qt-Widgets

---

```
lcdzahl->setGeometry( 230, 10, 60, 30 );
lcdzahl->display( 120 );
    //... Drei Spinboxen
spinbox = new QSpinBox( -1, 255, 1, this );
spinbox->setValue( 120 ); // Startwert
spinbox->setSpecialValueText( "Default" ); // ist: 120
spinbox->setWrapping( true );
spinbox->setGeometry( 310, 10, 80, 30 );
    //... Verbinde Schiebepalken und Spinbox mit LCD-Anzeige
QObject::connect( schiebepalk, SIGNAL( valueChanged( int ) ),
                  this,        SLOT( aktualisiereSpinListLCD( int ) ) );
QObject::connect( spinbox, SIGNAL( valueChanged( int ) ),
                  this,        SLOT( aktualisiereSpinListLCD( int ) ) );
}
signals:
    void farbeGeeandert( int farbwert );
private slots:
    void aktualisiereSpinListLCD( int wert ) {
        schiebepalk->setValue( wert== -1 ? 120 : wert );
        spinbox->setValue( wert );
        lcdzahl->display( wert== -1 ? 120 : wert );
        emit farbeGeeandert( wert== -1 ? 120 : wert );
    }
private:
    QLabel*      name;
    QSlider*     schiebepalk;
    QLCDNumber* lcdzahl;
    QSpinBox*    spinbox;
};
//..... Farblabel
class Farblabel : public QLabel {
    Q_OBJECT
public:
    Farblabel( QWidget *parent ) : QLabel( parent ) {
        setPaletteBackgroundColor( QColor( rot=120, gruen=120, blau=120 ) );
    }
public slots:
    void rotNeueFarbe( int neuefarbe ) {
        setPaletteBackgroundColor( QColor( rot=neuefarbe, gruen, blau ) );
    }
    void gruenNeueFarbe( int neuefarbe ) {
        setPaletteBackgroundColor( QColor( rot, gruen=neuefarbe, blau ) );
    }
    void blauNeueFarbe( int neuefarbe ) {
        setPaletteBackgroundColor( QColor( rot, gruen, blau=neuefarbe ) );
    }
private:
    int rot, gruen, blau;
};
#include "listspin.moc"
```

```
//..... main
int main( int argc, char* argv[] ) {
    QApplication myapp( argc, argv );
    QWidget* mywidget = new QWidget();
    mywidget->setGeometry( 100, 100, 540, 250 );

    //..... Erzeugen von drei SpinListLCD-Balken
    SpinListLCD* rotBalken = new SpinListLCD( mywidget, "R" );
    rotBalken->setGeometry( 10, 30, 390, 60 );
    SpinListLCD* gruenBalken = new SpinListLCD( mywidget, "G" );
    gruenBalken->setGeometry( 10, 100, 390, 60 );
    SpinListLCD* blauBalken = new SpinListLCD( mywidget, "B" );
    blauBalken->setGeometry( 10, 170, 390, 60 );

    //..... Erzeugen einer Farbanzeige
    FarbLabel* farbanzeige = new FarbLabel( mywidget );
    farbanzeige->setGeometry( 420, 30, 100, 200 );

    //..... Verbinde RGB-Balken mit Farbanzeige
    QObject::connect( rotBalken, SIGNAL( farbeGeaendert( int ) ),
                     farbanzeige, SLOT( rotNeueFarbe( int ) ) );
    QObject::connect( gruenBalken, SIGNAL( farbeGeaendert( int ) ),
                     farbanzeige, SLOT( gruenNeueFarbe( int ) ) );
    QObject::connect( blauBalken, SIGNAL( farbeGeaendert( int ) ),
                     farbanzeige, SLOT( blauNeueFarbe( int ) ) );

    myapp.setMainWidget( mywidget );
    mywidget->show();
    return myapp.exec();
}
```

### 3.6.6 Beispiel zu QDateTimeEdit, QTimeEdit, QDateTimeEdit<sup>B</sup>

Programm 3.6 ist ein Demoprogramm, bei dem man das Datum bzw. die Zeit sowohl über jeweils eine QDateTime- bzw. eine QTimeEdit-Spinbox als auch über eine QDateTimeEdit-Spinbox ändern kann. Die Änderungen sind dabei synchronisiert und werden in zwei QLCDNumber-Widgets angezeigt (siehe Abbildung 3.9).



Abbildung 3.9: Synchronisiertes Eingeben von Datum und Zeit mit Anzeige

Programm 3.6 – datetimededit.cpp:

Synchronisiertes Editieren von Datum und Zeit mit Anzeige

```
#include <qapplication.h>
#include <qlabel.h>
#include <qdatetimeedit.h>
#include <qlcdnumber.h>
```

### 3 Die wesentlichen Qt-Widgets

```
//..... zeitDatumEingabe
class zeitDatumEingabe : public QWidget {
    Q_OBJECT
public:
    zeitDatumEingabe( QWidget* parent = 0, const char* name = 0 );
    ~zeitDatumEingabe() { }
    QDateEdit      *datumEdit;
    QTimeEdit      *zeitEdit;
    QDateTimeEdit  *datumZeitEdit;
    QLCDNumber      *anzeige[2]; // [0] = Datum, [1] = Zeit
private slots:
    void datumGeeandert(const QDate &datum) {
        anzeige[0]->display( datum.toString( "dd.MM.yyyy" ) );
        if ( datumZeitEdit->dateEdit()->date() != datum )
            datumZeitEdit->dateEdit()->setDate( datum );
    }
    void zeitGeeandert(const QTime &zeit) {
        anzeige[1]->display( zeit.toString( QString( "hh:mm:ss" ) ) );
        if ( datumZeitEdit->timeEdit()->time() != zeit )
            datumZeitEdit->timeEdit()->setTime( zeit );
    }
    void datumZeitGeeandert(const QDateTime &datumZeit) {
        anzeige[0]->display( datumZeit.date().toString( "dd.MM.yyyy" ) );
        anzeige[1]->display( datumZeit.time().toString( "hh:mm:ss" ) );
        if ( zeitEdit->time() != datumZeit.time() )
            zeitEdit->setTime( datumZeit.time() );
        if ( datumEdit->date() != datumZeit.date() )
            datumEdit->setDate( datumZeit.date() );
    }
};
//..... zeitDatumEingabe-Konstruktor
zeitDatumEingabe::zeitDatumEingabe( QWidget* parent, const char* name )
    : QWidget( parent, name ) {

    datumEdit = new QDateEdit( this );
    datumEdit->setDate( QDate::currentDate() );
    datumEdit->setAutoAdvance( true );
    datumEdit->setOrder( QDateEdit::DMY );
    datumEdit->setSeparator( "/" );
    datumEdit->setGeometry( 10, 10, 150, 40 );

    zeitEdit = new QTimeEdit( this );
    zeitEdit->setTime( QTime::currentTime() );
    zeitEdit->setAutoAdvance( true );
    zeitEdit->setDisplay( int( QTimeEdit::Seconds |
                             QTimeEdit::Minutes | QTimeEdit::Hours ) );
    zeitEdit->setSeparator( "-" );
    zeitEdit->setGeometry( 10, 60, 150, 40 );

    datumZeitEdit = new QDateTimeEdit( this );
    datumZeitEdit->setDateTime( QDateTime::currentDateTime() );
    datumZeitEdit->setGeometry( 10, 140, 300, 40 );
    for ( int i=0; i<2; i++ ) {
```

```

    anzeige[i] = new QLCDNumber( this );
    anzeige[i]->setGeometry( 240, 10+i*50, 190, 40 );
    anzeige[i]->setPaletteBackgroundColor( Qt::yellow );
    anzeige[i]->setFont( QFont( "Times", 20 ) );
    anzeige[i]->setSegmentStyle( QLCDNumber::Filled );
    anzeige[i]->setNumDigits( 10 );
}
connect( zeitEdit, SIGNAL( valueChanged(const QTime&) ),
        this,      SLOT( zeitGeeandert(const QTime&) ) );
connect( datumEdit, SIGNAL( valueChanged(const QDate&) ),
        this,      SLOT( datumGeeandert(const QDate&) ) );
connect( datumZeitEdit, SIGNAL( valueChanged(const QDateTime&) ),
        this,          SLOT( datumZeitGeeandert(const QDateTime&) ) );
datumZeitGeeandert( QDateTime::currentDateTime() );
}
#include "datetimeedit.moc"
//..... main
int main( int argc, char *argv[] ) {
    QApplication myapp( argc, argv );
    zeitDatumEingabe* mywidget = new zeitDatumEingabe();
    mywidget->setGeometry( 20, 20, 480, 200 );
    myapp.setMainWidget( mywidget );
    mywidget->show();
    return myapp.exec();
}

```

### 3.6.7 Größeneinstellung eines Rechtecks über Schieberegler und Spinboxen<sup>Z</sup>

Das Programm `lspin.cpp` ermöglicht es dem Benutzer die Höhe und Breite eines Rechtecks mit Schieberegler und Spinboxen zu verändern (siehe auch Abbildung 3.10).

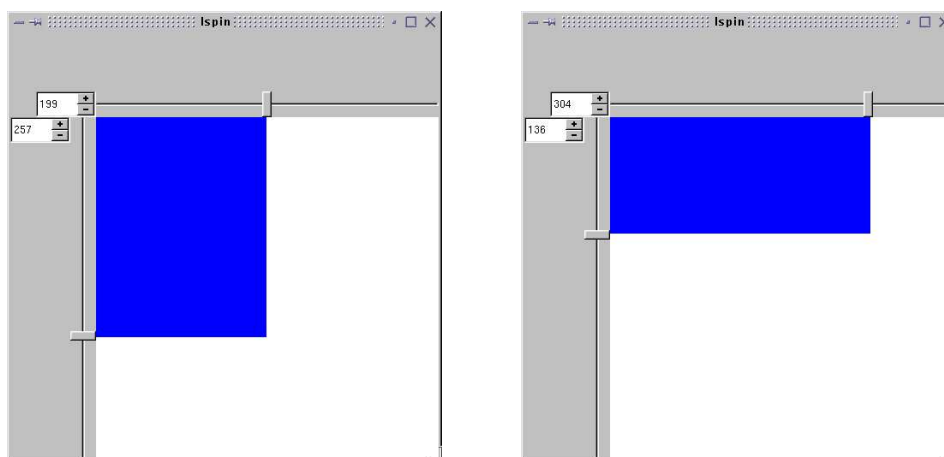


Abbildung 3.10: Größeneinstellung eines Rechtecks über Schieberegler und Spinboxen

### 3.6.8 Farbeinstellung mit Potentiometern und Spinboxen<sup>Z</sup>

Das Programm `dial.cpp` leistet Ähnliches wie Programm 3.5, nur dass es statt Schieberegler Drehknöpfe für die Einstellung der einzelnen RGB-Komponenten anbietet. Zusätzlich zeigt dieses Programm die einzelnen Drehknöpfe in der Farbe an, die gerade für diese RGB-Komponente eingestellt ist (siehe auch Abbildung 3.11).

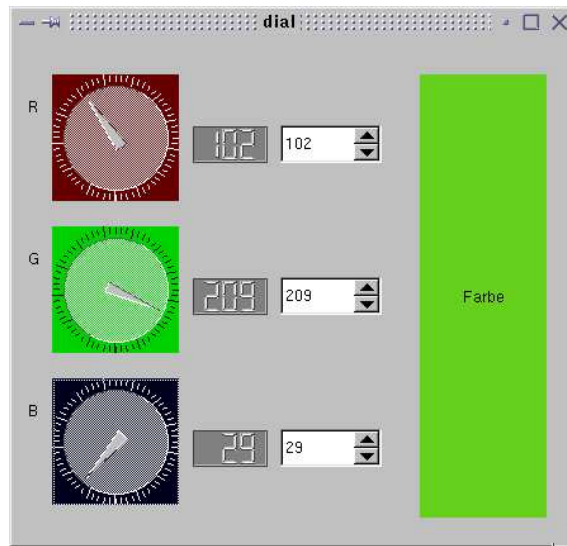


Abbildung 3.11: Farbanzeige mit Potentiometer und Spinboxen

## 3.7 Widgets zum Anzeigen von Informationen

Die hier vorgestellten Widgets lassen keinerlei Interaktionen mit dem Benutzer zu, sondern dienen lediglich der Anzeige von Information. Qt bietet hierfür drei Klassen an:

- `QLabel` einfache Anzeige von Text, Pixmaps oder einer Animation.
- `QTextBrowser` komfortable Anzeige von Text im RichText-Format mit Hypertext-Navigation und Laufbalken bei größeren Texten.
- `QLCDNumber` Anzeige einer Zahl bzw. eines Textes in 7-Segment-LCD-Darstellung.

### 3.7.1 Einfache Labels (`QLabel`)

Soll Text in einem Label-Widget angezeigt werden, kann dieser bereits beim Erzeugen des `QLabel`-Objekts als erstes Argument beim Aufruf des Konstruktors angegeben werden. Die Klasse `QLabel` bietet unter anderem die folgenden Methoden an:

- `setText(const QString& text) virtual slot`
- `setPixmap(const QPixmap& pixmap) virtual slot`  
zum Festlegen des im Label anzuzeigenden Textes bzw. der anzuzeigenden Pixmap.
- `setNum(int num) virtual slot`
- `setNum(double num) virtual slot`  
zum Festlegen der im Label anzuzeigenden `int`- bzw. `double`-Zahl.

### 3.13.6 Geschachtelte Fenster mit Laufbalken<sup>Z</sup>

Das Programm `scrollview2.cpp` blendet ein Hauptfenster mit Laufbalken ein. In diesem Hauptfenster werden nun unterschiedliche Bilder in Form einer Tabelle eingeblendet, die ihrerseits wieder Laufbalken enthalten, wenn dies erforderlich ist. Zusätzlich wird bei allen diesen Laufbalken-Fenstern in der Ecke zwischen dem jeweiligen horizontalen und vertikalen Laufbalken die Position des aktuellen Sichtfensters in Form eines kleinen blauen Rechtecks angezeigt, wie dies in Abbildung 3.40 gezeigt ist.



Abbildung 3.40: Bilder mit Laufbalken in einem gemeinsamen Laufbalken-Fenster

## 3.14 Tabellen

Um Information in Form einer Tabelle bereitzustellen, bietet Qt zwei Klassen an:

- `QGridView` Abstrakte Klasse zur Anzeige von Information in Tabellenform
- `QTable` sehr flexibles Tabellen-Widget, das ein Editieren der einzelnen Zellen im Spreadsheet-Stil ermöglicht.

### 3.14.1 Einfache Tabellen mit `QGridView`

Da es sich bei `QGridView` um eine abstrakte Klasse handelt, kann diese nicht direkt benutzt werden. Stattdessen muss man eine eigene Klasse von `QGridView` ableiten, um dann die von `QGridView` angebotene Funktionalität benutzen zu können.

Die Klasse `QGridView` bietet einen Konstruktor an:

```
QGridView(QWidget *parent=0, const char *name=0, WFlags f=0)
```

Nachfolgend werden einige wichtige Methoden der Klassen `QGridView` kurz vorgestellt:

```
setNumRows(int rows) virtual
setNumCols(int cols) virtual
```

legen Zeilen- bzw. Spaltenzahl der Tabelle fest. Die Indizes beginnen dabei immer bei 0, und nicht bei 1.

```
setCellWidth(int cellWidth) virtual
```

```
setCellHeight(int cellHeight) virtual
```

legen die Breite bzw. Höhe für die Zellen in der Tabelle fest.

```
int cellWidth() const, int cellHeight() const
```

liefern die eingestellte Breite bzw. Höhe der Zellen.

```
int rowAt(int y) const, int columnAt(int x) const
```

liefern den Index der Zeile bzw. der Spalte an der  $y$ -Koordinate bzw. an der  $x$ -Koordinate, wenn diese Koordinaten sich innerhalb der Tabelle befinden.

```
ensureCellVisible(int zeile, int spalte)
```

versetzen das Sichtfenster so, dass die Zelle (zeile, spalte) sichtbar ist.

```
updateCell(int row, int col)
```

aktualisiert die Zelle in der Zeile row und der Spalte col.

```
paintCell(QPainter *p, int row, int col) virtual protected
```

Diese rein virtuelle Funktion wird immer aufgerufen, um eine einzelne Zelle an der Position (row, col) unter Benutzung des QPainter-Objekts p zu zeichnen. Das QPainter-Objekt p ist bereits offen, wenn diese Funktion aufgerufen wird, und es muss auch offen bleiben.

### 3.14.2 Beispiel zu QGridView: Multiplikationsaufgaben<sup>B</sup>

Hier wird als Beispiel eine Tabelle erstellt, deren Zellen alle Kombinationen für die Multiplikation zweier Zahlen zwischen 0 und 30 enthalten. Der Benutzer kann sowohl über die Steuertasten  $\uparrow$ ,  $\downarrow$ ,  $\leftarrow$  und  $\rightarrow$  als auch mit Mausklick eine Zelle auswählen. In der aktuell ausgewählten Zelle kann er nun das entsprechende Ergebnis eintragen. Ist es richtig, wird die Ergebniszahl in dieser Zelle hinterlegt. Ist das Ergebnis falsch, wird wieder der alte Multiplikationstext eingeblendet. Zellen, in denen der Benutzer Eingaben vorgenommen hat, werden farblich hinterlegt: grün, wenn die Rechenaufgabe richtig gelöst wurde, und rot, wenn sie falsch gelöst wurde. Die aktuell angewählte Zelle wird immer mit gelbem Hintergrund und durch einen gestrichelten Rahmen gekennzeichnet. Abbildung 3.41 zeigt ein mögliches Aussehen der Tabelle, nachdem der Benutzer einige Rechenaufgaben gelöst bzw. zu lösen versucht hat.

8 x 2	8 x 3	8 x 4	8 x 5	8 x 6	8 x 7
9 x 2	9 x 3	9 x 4	9 x 5	9 x 6	9 x 7
10 x 2	10 x 3	10 x 4	10 x 5	10 x 6	10 x 7
11 x 2	33	11 x 4	11 x 5	11 x 6	11 x 7
12 x 2	12 x 3	48	12 x 5	12 x 6	12 x 7
13 x 2	13 x 3	13 x 4	13 x 5	13 x 6	13 x 7
14 x 2	14 x 3	14 x 4	14 x 5	14 x 6	14 x 7
15 x 2	45	15 x 4	15 x 5	15 x 6	15 x 7
16 x 2	16 x 3	16 x 4	16 x 5	16 x 6	16 x 7
17 x 2	17 x 3	17 x 4	17 x 5	17 x 6	17 x 7
18 x 2	18 x 3	18 x 4	90	18 x 6	18 x 7
19 x 2	19 x 3	19 x 4	19 x 5	19 x 6	19 x 7
20 x 2	20 x 3	20 x 4	20 x 5	20 x 6	20 x 7

Abbildung 3.41: Eine Multiplikationstabelle



### 3 Die wesentlichen Qt-Widgets

---

Programm 3.18 zeigt die Headerdatei und Programm 3.19 zeigt die Implementierung zu dieser Aufgabenstellung.

Programm 3.18 – tableview.h:  
Headerdatei zur Multiplikationstabelle

```
#ifndef TABLEVIEW_H
#define TABLEVIEW_H
#include <qgridview.h>

class Tableview : public QGridView {
public:
    Tableview( int zeilZahl, int spaltZahl, QWidget* p=0, const char* nam=0 );
    ~Tableview() { delete[] inhalt; }
    const char* zellenInhalt(int z, int s) const { return inhalt[indexOf(z, s)]; }
    void setzeZellenInhalt(int z, int s, const char* c) {
        inhalt[indexOf(z, s)] = c; updateCell(z, s);
    }
    void setzeZellenFarbe(int z, int s, QColor f) {
        zellenFarbe[indexOf(z, s)] = f; updateCell(z, s);
    }
protected:
    void paintCell( QPainter*, int zeile, int spalte );
    void mousePressEvent( QMouseEvent* );
    void keyPressEvent( QKeyEvent* );
private:
    int indexOf(int z, int s) const { return (z*numCols()) + s; }
    void pruefEingabe( int zeile, int spalte ) {
        if ( eingabeGemacht ) {
            if ( spalte * zeile == eingabewert )
                setzeZellenFarbe( zeile, spalte, Qt::green );
            else {
                setzeZellenFarbe( zeile, spalte, Qt::red );
                setzeZellenInhalt( zeile, spalte, alt );
            }
            eingabewert = 0;
            eingabeGemacht = false;
        } else
            updateCell( zeile, spalte );
    }
    QString *inhalt, alt;
    QColor *zellenFarbe;
    int aktZeile, aktSpalte, eingabewert;
    bool eingabeGemacht;
};
#endif
```

Programm 3.19 – tableview.cpp:  
Implementierung der Multiplikationstabelle

```
#include <qapplication.h>
#include <qwidget.h>
#include <qpainter.h>
```

```

#include <qkeycode.h>
#include <qgridview.h>
#include "tableview.h"
const int zahlZeilen = 31;
const int zahlSpalten = 31;
//..... Tableview (Konstruktor)
Tableview::Tableview(int zeilZahl, int spaltZahl, QWidget *p, const char *nam)
    : QGridView(p,nam) {
    aktZeile = aktSpalte = 0;          // Aktuell ausgewaehlte Zelle
    setNumCols( spaltZahl ); setNumRows( zeilZahl ); // Spalten- und Zeilenzahl
    setCellWidth( 100 );      setCellHeight( 30 ); // Zellenbreite/-höhe in Pixel
    resize( 600, 400 );
    inhalt      = new QString[zeilZahl * spaltZahl]; // Speicher fuer Inhalt
    zellenFarbe = new QColor[zeilZahl * spaltZahl]; // Speicher fuer Farben
    eingabewert = 0;
    eingabeGemacht = false;
}
//..... paintCell
void Tableview::paintCell( QPainter* p, int zeile, int spalte ) {
    int w = cellWidth(), h = cellHeight();
    if ( zeile == aktZeile && spalte == aktSpalte ) {
        p->setBrush( Qt::yellow ); p->setPen( DotLine );
        p->drawRect( 0, 0, w-1, h-1 );
        p->setPen( SolidLine );
    } else
        p->fillRect( 0, 0, w, h, zellenFarbe[ indexOf( zeile, spalte ) ] );
    p->drawLine(w-1, 0, w-1, h-1); p->drawLine(0, h-1, w-1, h-1); // Zellenrahmen
    //... Inhalt (Text) der Zelle ausgeben
    p->drawText( 0, 0, w, h, AlignCenter, inhalt[indexOf(zeile,spalte)] );
}
//..... mousePressEvent
void Tableview::mousePressEvent( QMouseEvent* e ) {
    int altZeile = aktZeile, altSpalte = aktSpalte;
    //... Position ermitteln, an der Mauszeiger geklickt wurde
    QPoint clickedPos = viewportToContents( e->pos() );
    aktZeile = rowAt( clickedPos.y() );
    aktSpalte = columnAt( clickedPos.x() );
    //... Wenn aktuelle Zelle sich geaendert hat
    if ( aktZeile != altZeile || aktSpalte != altSpalte ) {
        pruefEingabe( altZeile, altSpalte ); // alten Rahmen loeschen
        updateCell( aktZeile, aktSpalte ); // Rahmen fuer neue Zelle
    }
    eingabewert = 0;
}
//..... keyPressEvent
void Tableview::keyPressEvent( QKeyEvent* e ) {
    int altZeile = aktZeile, altSpalte = aktSpalte;
    QString s;
    if ( eingabewert == 0 )
        alt = inhalt[ indexOf( aktZeile, aktSpalte ) ];

```

### 3 Die wesentlichen Qt-Widgets

---

```
switch( e->key() ) {
    case Key_Left: pruefEingabe( aktZeile, aktSpalte );
                    if ( aktSpalte > 0 )
                        ensureCellVisible(aktZeile, --aktSpalte);
                    break;
    case Key_Right: pruefEingabe( aktZeile, aktSpalte );
                    if ( aktSpalte < numCols()-1 )
                        ensureCellVisible(aktZeile, ++aktSpalte);
                    break;
    case Key_Up:    pruefEingabe( aktZeile, aktSpalte );
                    if ( aktZeile > 0 )
                        ensureCellVisible(--aktZeile, aktSpalte);
                    break;
    case Key_Down:  pruefEingabe( aktZeile, aktSpalte );
                    if ( aktZeile < numRows()-1 )
                        ensureCellVisible(++aktZeile, aktSpalte);
                    break;
    default:        if ( e->key() >= Key_0 && e->key() <= Key_9 ) {
                        eingabeGemacht = true;
                        eingabewert = eingabewert * 10 + e->key()-Key_0;
                        s.sprintf("%d", eingabewert);
                        setzeZellenInhalt( aktZeile, aktSpalte, s );
                    } else if ( e->key() == Key_Return ) {
                        pruefEingabe( aktZeile, aktSpalte );
                    } else {
                        e->ignore(); //.... Alle anderen Tasten ignorieren
                        return;
                    }
}

//... Wenn aktuelle Zelle sich geaendert hat
if ( aktZeile != altZeile || aktSpalte != altSpalte ) {
    updateCell( altZeile, altSpalte ); // alten Rahmen loeschen
    updateCell( aktZeile, aktSpalte ); // Rahmen fuer neue Zelle
}
}

int main( int argc, char *argv[] ) {
    QApplication a(argc,argv);
    Tableview t( zahlZeilen, zahlSpalten );
    QString s;
    for( int i = 0; i < zahlZeilen; i++ ) {
        for( int j = 0; j < zahlSpalten; j++ ) {
            s.sprintf("%d x %d", i, j);
            t.setzeZellenInhalt( i, j, s ); //... Zelleninhalt (Malaufgabe)
            t.setzeZellenFarbe( i, j, Qt::white ); //... Zellenfarbe
        }
    }
    a.setMainWidget( &t );
    t.show();
    return a.exec();
}
```

### 3.14.3 Die Klasse `QTable` für Tabellen im Spreadsheet-Stil

Hier wird die Klasse `QTable` vorgestellt, die ein sehr flexibles Tabellen-Widget zur Verfügung stellt, das ein leichtes Editieren der einzelnen Zellen ermöglicht.

Die Klasse `QTable` bietet die folgenden Konstruktoren an:

```
QTable(QWidget *parent=0, char *nam=0)
```

legt eine leere Tabelle an. Die Anzahl der Zeilen und Spalten können dann mit `setNumRows()` und `setNumCols()` festgelegt werden.

```
QTable(int numRows, int numCols, QWidget *par=0, char *nam=0)
```

legt eine Tabelle mit `numRows` Zeilen und `numCols` Spalten an.

Nachfolgend sind einige wichtige Methoden aufgezählt, die die Klasse `QTable` anbietet:

```
setColumnWidth(int col, int w) virtual slot
```

```
setRowHeight(int row, int h) virtual slot
```

legen Breite der Spalte `col` auf `w` Pixel bzw. Höhe der Zeile `row` auf `h` Pixel fest.

```
setText(int row, int col, const QString& text) virtual
```

```
setPixmap(int row, int col, const QPixmap& pix) virtual
```

schreibt den Text `text` bzw. stellt das Bild `pix` in die Zelle, die sich in Zeile `row` und Spalte `col` befindet.

```
QString text(int row, int col) const virtual
```

```
QPixmap pixmap(int row, int col) const virtual
```

liefern den Text bzw. das Bild aus der Zelle `(row, col)`.

```
setNumRows(int r) virtual slot, setNumCols(int c) virtual slot
```

legen Anzahl der Zeilen (`r`) bzw. Spalten (`c`) für die Tabelle fest.

```
int numRows() const, int numCols() const
```

liefern die Zeilen- bzw. Spaltenzahl der Tabelle.

```
setShowGrid(bool b) virtual slot
```

legt fest, ob Trennlinien zwischen den Zellen anzuzeigen sind (`b=true`) oder nicht (`b=false`). Die Voreinstellung ist, dass diese Trennlinien sichtbar sind.

```
bool showGrid() const
```

liefert zurück, ob Trennlinien zwischen den Zellen gerade sichtbar sind.

```
setCurrentCell(int row, int col) virtual slot
```

setzt Fokus auf Zelle `(row, col)` und macht diese so zur aktuellen Zelle.

```
int currentRow() const, int currentColumn() const
```

liefern aktuelle Zeile bzw. Spalte der Tabelle.

```
hideRow(int row) virtual slot, hideColumn(int col) virtual slot
```

```
showRow(int row) virtual slot, showColumn(int col) virtual slot
```

verstecken die Zeile `row` bzw. die Spalte `col` oder machen diese sichtbar.

```
sortColumn(int col, bool ascend=true, bool whole=false) virtual
```

sortiert die Spalte `col` in aufsteigender (`ascend=true`) bzw. absteigender (`ascend=false`) Ordnung. Bei `whole=true` werden die Zeilen vollständig mittels der virtuellen Slotroutine `swapRows(int row1, int row2)` und bei `whole=false` werden nur die Daten der Spalte mittels der virtuellen Slotroutine `swapCells(int row1, int col1, int row2, int col2)` vertauscht.

```
setSorting(bool b) virtual slot
```

legt für die Tabelle fest, ob bei einem Klick auf die Kopfzeile eine Sortierung der entsprechenden Spalte stattfinden soll (`b=true`) oder nicht (`b=false`).

### 3 Die wesentlichen Qt-Widgets

---

`swapRows(int row1, int row2, bool swapHead=false) virtual slot`  
`swapColumns(int col1, int col2, bool swapHead=false) virtual slot`  
vertauschen die beiden Zeilen `row1` und `row2` bzw. die beiden Spalten `col1` und `col2`. Diese Methode wird auch beim Sortieren aufgerufen oder wenn der Benutzer selbst zwei Spalten bzw. zwei Zeilen vertauscht. Bei `swapHead=true` werden auch die Zeilen- und Spaltenbeschriftungen mit vertauscht.

`swapCells(int row1, int col1, int row2, int col2) virtual slot`  
vertauscht die beiden Zellen (`row1, col1`) und (`row2, col2`). Diese Methode wird auch beim Sortieren aufgerufen.

`setTopMargin(int m) virtual slot`

`setLeftMargin(int m) virtual slot`

legen den oben bzw. links einzuhaltenden Rand für Tabelle fest. Um Spalten- bzw. Zeilenbeschriftung unsichtbar zu machen, kann folgender Code verwendet werden:

```
setTopMargin( 0 ); //... obere Beschriftungszeile unsichtbar machen
horizontalHeader()->hide();
setLeftMargin( 0 ); //... linke Beschriftungszeile unsichtbar machen
verticalHeader()->hide();
```

`setColumnMovingEnabled(bool b) virtual slot`

`setRowMovingEnabled(bool b) virtual slot`

legen fest, ob Spalten bzw. Zeilen durch den Benutzer verschoben werden können (`b=true`) oder nicht (`b=false`).

`adjustColumn(int col) virtual slot`

`adjustRow(int row) virtual slot`

verändern die Breite der Spalte `col` bzw. die Höhe der Zeile `row` so, dass ihr ganzer Inhalt sichtbar ist. Die benötigte Breite bzw. Höhe bestimmt dabei die Zelle mit dem breitesten bzw. höchsten Inhalt.

`setColumnStretchable(int col, bool stretch) virtual slot`

`setRowStretchable(int row, bool stretch) virtual slot`

legen fest, dass Spalte `col` bzw. Zeile `row` dehnbar (`stretch=true`) bzw. nicht dehnbar (`stretch=false`) sein soll. Dehbare Spalten bzw. Zeilen werden automatisch verkleinert bzw. vergrößert, wenn Tabelle verkleinert bzw. vergrößert wird, und können nicht manuell durch den Benutzer verkleinert bzw. vergrößert werden.

`int columnWidth(int col) const virtual`

`int rowHeight(int row) const virtual`

liefern die Breite der Spalte `col` bzw. die Höhe der Zeile `row` als Pixelzahl.

`QRect cellGeometry(int row, int col) const virtual`

liefert die Position und Ausmaße der Zelle (`row, col`) in `QRect`-Form.

`int columnPos(int col) const virtual`

`int rowPos(int row) const virtual`

liefern die x- bzw. y-Position der Spalte `col` bzw. der Zeile `row` als Pixelzahl.

`ensureCellVisible(int row, int col)`

verschiebt das Sichtfenster in der Tabelle so, dass die Zelle (`row, col`) sichtbar wird.

`int columnAt(int pos) const virtual`

`int rowAt(int pos) const virtual`

liefern die Spalte bzw. Zeile, die sich an der Pixel-Position `pos` befindet.

Folgende Signale stellt die Klasse `QTable` zur Verfügung:

`currentChanged(int z, int s)`

wird geschickt, wenn die aktuelle Zelle auf  $(z, s)$  geändert wurde.

`clicked(int z, int s, int button, const QPoint& mPos)`

`doubleClicked(int z, int s, int button, const QPoint& mPos)`

`pressed(int z, int s, int button, const QPoint& mPos)`

werden geschickt, wenn Benutzer mit der Maustaste `button` auf Zelle  $(z, s)$  einen einfachen bzw. doppelten Mausklick durchgeführt hat oder eben bei `pressed()` schon, wenn er die Maustaste dort gedrückt hat. `mPos` liefert dabei die Mausposition.

`valueChanged(int z, int s)`

wird geschickt, wenn Benutzer den Inhalt der Zelle  $(z, s)$  ändert.

`selectionChanged()`

wird geschickt, wenn sich eine Auswahl in der Tabelle ändert.

`contextMenuRequested(int z, int s, const QPoint& pos)`

wird geschickt, wenn Benutzer in der Zelle  $(z, s)$  mit der rechten Maustaste (oder einer anderen speziellen Tastenkombination) das Einblenden eines Kontextmenüs wünscht. `pos` liefert dabei die Mausposition als globale Koordinaten.

`dropped(QDropEvent *e)`

wird geschickt, wenn mittels einer *Drag-and-Drop*-Aktion das Ablegen (*drop*) eines Elements in Tabelle versucht wird. `e` enthält die Information über die *Drop*-Aktion.

### 3.14.4 Die Klasse `QHeader`

Die Klasse `QHeader` ermöglicht das Bearbeiten der Tabellenbeschriftungen, die sich oben und links von einer Tabelle (`QTable`-Objekt) befinden. Die Objekte zu diesen Tabellenbeschriftungen kann man sich, wie zuvor erwähnt, mit den beiden folgenden `QTable`-Methoden liefern lassen:

`QHeader* horizontalHeader() const`

`QHeader* verticalHeader() const`

Nachfolgend sind einige wichtige Methoden angegeben, die die Klasse `QHeader` anbietet:

`int mapToIndex(int section) const`

wird benötigt, wenn Spalten bzw. Zeilen verschoben wurden. `section` ist dabei die Position, an der die Spalte bzw. Zeile zu Beginn eingeordnet wurde. Diese Methode liefert den aktuellen Index, an dem sich diese Spalte bzw. Zeile nun befindet.

`int mapToSection(int index) const`

liefert die `section`, die sich an der Position `index` befindet.

`int addLabel(QString& s, int g=-1)`

`int addLabel(QIconSet& is, QString& s, int g=-1)`

fügt einen neuen Eintrag mit Beschriftung `s` und eventuell einer QPixmap `is` zur Beschriftung hinzu und liefert dessen Index als Rückgabewert. Über den Parameter `g` kann dabei die Breite dieses neuen Eintrags festgelegt werden. Ist `g` negativ, wird diese Breite automatisch berechnet.

`setLabel(int section, QString& s, int g=-1) virtual`

`setLabel(int section, QIconSet& is, QString& s, int g=-1) virtual`

Diese beiden Methoden entsprechen weitgehend den beiden vorherigen Methoden, nur dass man hier noch explizit die Position (`section`) angeben kann, an der der neue Eintrag einzuordnen ist.

### 3.14.5 Die Klassen `QTableWidgetItem` und `QTableSelection`

Der Vollständigkeit halber seien noch die beiden Klassen `QTableWidgetItem` und `QTableSelection` hier erwähnt:

- ❑ `QTableWidgetItem`: ermöglicht die eigene Verwaltung von Zelleninhalten. Eine wichtige Methode ist dabei die virtuelle Methode `createEditor()`, die man reimplementieren muss, um in einer Zelle z. B. einen eigenen (mehrzeiligen) Editor zu hinterlegen. In diesem Fall muss man zusätzlich noch die virtuelle Methode `setContentFromEditor()` reimplementieren, um den Editor-Inhalt in die Zelle übernehmen zu können. Um eine Kombo- oder Checkbox in einer Zelle unterzubringen, stehen zwei eigene Klassen: `QComboBoxItem` und `QCheckBoxItem` zur Verfügung.
- ❑ `QTableSelection`: ermöglicht Zugriff auf ausgewählte Bereiche in Tabelle.

Interessierte Leser seien hier auf die Qt-Online-Dokumentation verwiesen.

### 3.14.6 Spreadsheet für eine Personalverwaltung<sup>Z</sup>

Das Programm `table.cpp` realisiert einen einfachen Spreadsheet für eine Personalverwaltung, in dem man Daten eingeben kann, wie in Abbildung 3.42 zu sehen. Klickt man auf die obere Beschriftung einer Spalte, werden die Zeilen nach den Daten in dieser Spalte aufsteigend sortiert (siehe links in Abbildung 3.43). Klickt man erneut auf die gleiche Beschriftung, werden die Zeilen nach den Daten in dieser Spalte absteigend sortiert (siehe auch rechts in Abbildung 3.43). Klickt man mit der rechten Maustaste in eine Zelle der beiden Spalten „Monatsgehalt“ oder „Alter“, wird in einem eigenen Widget ein Balkendiagramm zu den Daten in dieser Spalte angezeigt (siehe links in Abbildung 3.44). Klickt man mit der rechten Maustaste in eine Zelle der ersten fünf Spalten, wird in einem eigenen Widget der gesamte Inhalt dieser Spalte gezeigt (siehe rechts in Abbildung 3.44).

	Name	Vorname	Wohnort	Strasse	PLZ	Monatsgeh	Alter	
1	Meier	Hans	Müldendorf	Waldstrasse	87123	6389	27	
2	Aller	Marianne	Buxenhausen	Baumstrasse	79191	5987	35	
3	Zander	Fritz	Kullerstadt	Bergstrasse	83474	6873	42	
4	Kullmann	Anja	Badenstadt	Hohlgasse	17272	4873	39	
5	Wallner	Egon	Hasenheim	Schwalbenweg	37828	5899	28	
6	Malker	Rudi	Nordhausen	Eichengasse	45666	7899	48	
7	Kerner	Michaela	Westhausen	Rindermarkt	87234	6783	39	
8								
9								
10								
11								
12								
13								
14								
15								
16								

Abbildung 3.42: Tabelle nach Eingabe der Daten

The top window 'table' shows the following data (sorted by Alter ascending):

	Name	Vorname	Wohnort	Strasse	PLZ	Monatsgeh	Alter
1	Meier	Hans	Mudendorf	Waldstrasse	87123	6389	27
2	Wallner	Egon	Hasenheim	Schwalbenweg	37828	5899	28
3	Aller	Marianne	Buxenhausen	Baumstrasse	79191	5987	35
4	Kullmann	Anja	Badenstadt	Hohlgasse	17272	4873	39
5	Kerner	Michaela	Westhausen	Rindermarkt	87234	8783	39
6	Zander	Fritz	Kullerstadt	Bergstrasse	83474	6873	42
7	Malker	Rudi	Nordhausen	Eichengasse	45666	7889	46

The bottom window 'table' shows the same data (sorted by Alter descending):

	Name	Vorname	Wohnort	Strasse	PLZ	Monatsgeh	Alter
1	Malker	Rudi	Nordhausen	Eichengasse	45666	7889	46
2	Zander	Fritz	Kullerstadt	Bergstrasse	83474	6873	42
3	Kullmann	Anja	Badenstadt	Hohlgasse	17272	4873	39
4	Kerner	Michaela	Westhausen	Rindermarkt	87234	8783	39
5	Aller	Marianne	Buxenhausen	Baumstrasse	79191	5987	35
6	Wallner	Egon	Hasenheim	Schwalbenweg	37828	5899	28
7	Meier	Hans	Mudendorf	Waldstrasse	87123	6389	27

Abbildung 3.43: Tabelle nach Mausklicks auf „Alter“-Header (nach Alter aufwärts bzw. abwärts sortiert)

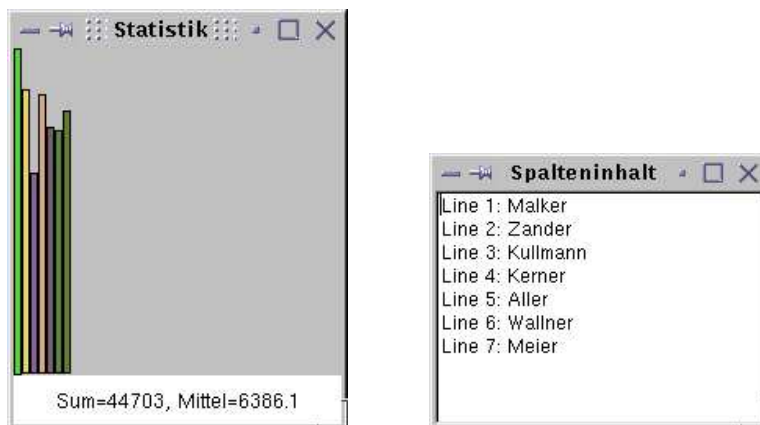


Abbildung 3.44: Anzeige einer Statistik nach rechtem Mausklick in „Monatsgehalt“-Spalte bzw. aller Inhalte der ersten Spalte nach rechtem Mausklick in „Namen“-Spalte

### 3.14.7 Tabelle mit Multiplikationsaufgaben<sup>Z</sup>

Das Programm `multtable.cpp` blendet mit Hilfe der Klasse `QTable` eine Tabelle ein, deren Zellen alle Kombinationen für die Multiplikation zweier Zahlen zwischen 1 und 30 enthalten. Der Benutzer kann sowohl über die Cursortasten als auch durch Mausklick



eine Zelle auswählen. In der aktuell ausgewählten Zelle kann er nun das entsprechende Ergebnis eintragen. Ist es korrekt, wird in dieser Zelle das richtige Ergebnis und eine andere QPixmap eingeblendet. Ist seine Eingabe falsch, wird wieder der alte Multiplikationstext eingeblendet, allerdings mit einer anderen QPixmap, die anzeigen soll, dass in dieser Zelle schon ein Lösungsversuch unternommen wurde. Abbildung 3.45 zeigt ein mögliches Aussehen der Tabelle, nachdem der Benutzer einige Rechenaufgaben gelöst bzw. zu lösen versucht hat.

	1	2	3	4	5	6	7	8
1	1 x 1	1 x 2	1 x 3	1 x 4	1 x 5	1 x 6	1 x 7	1 x 8
2	2 x 1	2 x 2	2 x 3	2 x 4	2 x 5	2 x 6	2 x 7	2 x 8
3	3 x 1	3 x 2	3 x 3	3 x 4	3 x 5	3 x 6	3 x 7	3 x 8
4	4 x 1	4 x 2	4 x 3	4 x 4	4 x 5	4 x 6	4 x 7	4 x 8
5	5 x 1	5 x 2	5 x 3	5 x 4	5 x 5	5 x 6	5 x 7	5 x 8
6	6 x 1	6 x 2	6 x 3	6 x 4	6 x 5	6 x 6	6 x 7	6 x 8
7	7 x 1	7 x 2	7 x 3	7 x 4	7 x 5	7 x 6	7 x 7	7 x 8
8	8 x 1	8 x 2	8 x 3	8 x 4	8 x 5	8 x 6	8 x 7	8 x 8
9	9 x 1	9 x 2	9 x 3	9 x 4	9 x 5	9 x 6	9 x 7	9 x 8
10	10 x 1	10 x 2	10 x 3	10 x 4	10 x 5	10 x 6	10 x 7	10 x 8
11	11 x 1	11 x 2	11 x 3	11 x 4	11 x 5	11 x 6	11 x 7	11 x 8
12	12 x 1	12 x 2	12 x 3	12 x 4	12 x 5	12 x 6	12 x 7	12 x 8

Abbildung 3.45: Eine Multiplikationstabelle mit der Klasse `QTable`

## 3.15 Widgets mit verschiebbaren Icons

Die Klasse `QIconView` stellt eine Fensterfläche zur Verfügung, in der Icons platziert werden, die sich dann verschieben lassen, wie man es z.B. von Dateimanagern her kennt, wenn diese im Symbolansicht-Modus betrieben werden, oder aber auch von Desktops. Jedes Icon ist dabei ein Objekt der Klasse `QIconViewItem`, das durch einen Text und/oder ein Bild dargestellt wird.

Programm 3.20 zeigt eine einfache Möglichkeit, wie man ein `QIconView`-Widget mit Icons anlegen kann, die man dann mit der Maus innerhalb des `QIconView`-Widgets verschieben kann (siehe auch Abbildung 3.46).

Programm 3.20 – `iconview.cpp`:

Fenster mit allen XPM-Icons aus dem Working-Directory

```
#include <qapplication.h>
#include <qiconview.h>
#include <qpixmap.h>
#include <qdir.h>

int main( int argc, char *argv[] ) {
    QApplication a( argc, argv );
```

# Kapitel 17

## Erstellen eigener Widgets

*Man muss etwas Neues machen,  
um etwas Neues zu sehen.*

– G. C. Lichtenberg

In diesem Kapitel wird gezeigt, wie man sich eigene Widget-Klassen erstellen kann. Dazu stellt dieses Kapitel zunächst kurz einige wichtige Punkte vor, die dabei zu beachten sind, bevor es anhand von zwei umfangreicheren Beispielen das Erstellen eigener Widgets verdeutlicht.

### 17.1 Grundlegende Vorgehensweise beim Entwurf eigener Widgets

Wenn man sich eigene Widgets entwerfen möchte, sollte man unter anderem die folgenden Punkte beachten:

❑ **Ableiten des Widget von einer geeigneten Basisklasse**

Bietet Qt kein vordefiniertes Widget an, das sich als Basisklasse für das zu entwerfende Widget eignet, so sollte man `QWidget` als Basisklasse verwenden. Stellt jedoch bereits Qt ein Widget zur Verfügung, das dem zu entwerfenden Widget ähnlich ist, so spart man sich viel Arbeit, wenn man das spezielle Qt-Widget als Basisklasse für sein neues Widget wählt, da so das neue Widget die gesamte Grundfunktionalität der entsprechenden Basisklasse erbt. Möchte man z. B. ein eigenes Dialog-Widget entwerfen, so empfiehlt sich `QDialog` als Basisklasse. Die Verwendung der Basisklasse `QGridView` bzw. `QTable` wäre dagegen zu empfehlen, wenn man ein Widget entwerfen möchte, das seine Informationen graphisch oder als Text in Tabellenform anzeigt.

❑ **Reimplementieren der entsprechenden Event-Handler**

Alle Event-Handler, die für die Funktionalität des eigenen Widget benötigt werden, sollten reimplementiert werden. Nahezu in allen Fällen sind dies die virtuellen Methoden `mousePressEvent()` und `mouseReleaseEvent()`. Soll das eigene Widget auch Tastatureingaben zulassen, so muss man z. B. auch die virtuelle Methode `keyPressEvent()` und eventuell auch die beiden virtuellen Methoden `focusInEvent()` und `focusOutEvent()` reimplementieren.

**❑ Festlegen eigener Signale**

Möchte man, dass das Auftreten gewisser Ereignisse auch dem Benutzer dieser Klasse mitgeteilt wird, so muss man ihm dies über eigene Signale mitteilen. Üblicherweise wird man diese selbst definierten Signale in den reimplementierten virtuellen Methoden wie z. B. `mousePressEvent()`, `mouseReleaseEvent()` oder `mouseDoubleClickEvent()` mit `emit` schicken.

**❑ Reimplementieren der virtuellen Methode `paintEvent()`**

Verwendet das eigene Widget `QPainter`-Objekte, um Zeichnungen durchzuführen, so muss man die virtuelle Methode `paintEvent()` reimplementieren, damit sich dieses eigene Widget selbst neu zeichnen kann, wenn dies erforderlich ist.

**❑ Festlegen geeigneter Slotroutinen**

Methoden, die `public` sind und den Zustand des eigenen Widget ändern, sind potentielle Kandidaten für `public`-Slotroutinen. Typischerweise haben Slotroutinen keine Parameter (wie z. B. `clear()` oder `reset()`) oder eben nur einen Parameter (wie z. B. `setInitialValue(int)` oder `setExpandable(bool)`).

**❑ Entscheiden, ob vom eigenen Widget weitere Subklassen ableitbar sind**

Wenn es möglich sein soll, dass vom eigenen Widget weitere Subklassen abgeleitet werden können, so sollte man festlegen, welche Methoden in Subklassen reimplementierbar sein sollen. Diese Methoden sollte man dann nicht als `private`, sondern als `protected` deklarieren.

**❑ Eventuelles Reimplementieren der Methoden `sizeHint()` und `sizePolicy()`**





Möchte man, dass das eigene Widget problemlos in ein Layout eingefügt werden kann, kann man die beiden Methoden `QWidget::sizeHint()` und `QWidget::sizePolicy()` entsprechend reimplementieren.

Dies waren einige wichtige Punkte, die es beim Entwurf eigener Widgets zu beachten gilt. In den nächsten beiden Kapitel werden nun Beispiele für den Entwurf eigener Widgets gezeigt.

## 17.2 Beispiel: Ein Funktionsplotter-Widget<sup>B</sup>

Als erstes Beispiel für ein selbstdefiniertes Widget soll hier ein Funktionsplotter entworfen werden.

### 17.2.1 Beschreibung des Funktionsplotters

Dieser Funktionsplotter soll zunächst mit  $(x, y)$ -Wertepaaren versorgt werden und dann die entsprechende Funktion graphisch anzeigen. Mittels Maus- und Tastaturtasten soll diese angezeigte Funktion dann entsprechend gezoomt werden können. Ein Vergrößern des anzuzeigenden Funktionsbereichs soll dabei mit den Cursortasten , ,  und  möglich sein. Ein „Hineinzoomen“ (also Vergrößern einer Teilansicht der angezeigten Funktion) soll mittels Ziehen der Maus möglich sein. Dabei wird immer das aktuell ausgewählte Rechteck der Ansicht mit roter bzw. grüner Hintergrundfarbe angezeigt. Rot bedeutet dabei, dass das Zoomen noch nicht aktiviert ist, während grün bedeutet, dass nun das Zoomen aktiviert ist. Ein Wechseln von der Hintergrundfarbe Rot nach Grün erfolgt immer erst dann, wenn der Benutzer die Maus bei gedrückter Maustaste eine bestimmte Strecke entfernt hat. So soll verhindert werden, dass der Benutzer durch ein versehentliches Drücken der Maustaste einen zu kleinen Bereich auswählt. Lässt der Benutzer bei

einer grünen Markierung die Maustaste los, wird die alte Funktionsansicht gelöscht und ihm nur der von ihm markierte Bereich der Funktion vergrößert im Widget angezeigt. Wenn der Benutzer bei einer roten Markierung die Maustaste loslässt, geschieht nichts und der alte Funktionsbereich bleibt eingeblendet.

## 17.2.2 Headerdatei für den Funktionsplotter<sup>B</sup>

Da für diesen zu entwerfenden Funktionsplotter keine geeignete Klasse in Qt existiert, wird `QWidget` als Basisklasse herangezogen.

Programm 17.1 zeigt die Headerdatei für den Funktionsplotter.

Programm 17.1 – `functionplot.h`:

Headerdatei für den Funktionsplotter

```
#ifndef FUNCTIONPLOT_H
#define FUNCTIONPLOT_H

#include <qapplication.h>
#include <qwidget.h>

typedef struct {
    double x, y;
} valuePair;

//..... Klasse FunctionPlot
class FunctionPlot : public QWidget {
public:
    //... Konstruktor, dem die Anzahl der Wertepaare fuer
    //... die zu plottende Funktion uebergeben wird
    FunctionPlot( int n, QWidget *p=0, const char *name=0 );
    //... Destruktor; gibt den fuer die Wertepaare
    //... reservierten Speicherplatz wieder frei
    ~FunctionPlot() { delete [] values; }
    //... setzt das i.te Wertepaar auf die Werte v.x und v.y
    void setValue( int i, valuePair v );
    //... setzt die ersten n Wertepaare auf die Werte aus dem Array v
    void setValues( int n, valuePair v[] );
    //... legt die minimalen und maximalen x- und y-Werte
    //... des Bereichs fest, der von der Funktion anzuzeigen ist
    void setPlotView( double minx, double miny, double maxx, double maxy );
    //... bewirkt, dass die Funktion gezeichnet wird
    void plotIt( void );
protected:
    //... bewirkt ein Vergroessern des Funktionsbereichs
    //... "Cursor rechts": negative x-Achse wird verlaengert
    //... "Cursor links" : positive x-Achse wird verlaengert
    //... "Cursor hoch" : negative y-Achse wird verlaengert
    //... "Cursor tief" : positive y-Achse wird verlaengert
    virtual void keyPressEvent( QKeyEvent *ev );
    //... leitet ein Zoomen (Verkleinern des Funktionsbereichs) ein
    virtual void mousePressEvent( QMouseEvent *ev );
    //... Zoomen (Verkleinern des Funktionsbereichs) findet gerade statt
```

```

virtual void mouseMoveEvent( QMouseEvent *ev );
    //... Zoomen (Verkleinern des Funktionsbereichs) wird abgeschlossen
virtual void mouseReleaseEvent( QMouseEvent *ev );
    //... bewirkt ein Neumalen der Funktion in dem aktuell
    //... festgelegten Funktionsbereich
virtual void paintEvent( QPaintEvent *ev );
private:
    //... zeichnen die x- bzw. y-Achse
    void drawXScale( QPainter *p, double i, int yp );
    void drawYScale( QPainter *p, double i, int xp );
    int      valueNo; // Anzahl der Wertepaare der Funktion
    valuePair *values; // enthaelt die einzelnen Wertepaare
    bool      plotViewSet; // zeigt an, ob explizit ein eigener
                        // anzuzeigender Funktionsbereich festgelegt
                        // wurde, also setPlotView() aufgerufen wurde.
                        // plotViewSet=false, werden die minimalen
                        // und maximalen x- und y-Werte des
                        // anzuzeigenden Bereichs implizit bestimmt
    double    minX, minY, maxX, maxY, // minimale und maximale x- und y-Werte
                        // des anzuzeigenden Bereichs
    double    xFactor, yFactor; // interne Projektionsfaktoren
    QPoint     startPos, letztePos, neuePos; // Mauspositionen, die fuer
                        // das Zoomen mit der Maus
                        // benoetigt werden.
    bool      dragging, ersteMal; // fuer Ziehen der Maus benoetigt
    QColor     farbe; // zeigt an, ob Zoomen aktiviert ist (gruen)
                        // oder nicht (rot)
};
#endif

```

### 17.2.3 Implementierung des Funktionsplotters<sup>B</sup>

Programm 17.2 zeigt die Implementierung des Funktionsplotters.

Programm 17.2 – functionplot.cpp:

Implementierung des Funktionsplotters

```

#include <qpainter.h>
#include <stdio.h>
#include "functionplot.h"
FunctionPlot::FunctionPlot( int n, QWidget *p, const char *name )
    : QWidget( p, name ) {

    valueNo = n;
    values = new valuePair [n];
    for ( int i=0; i<n; i++ )
        values[i].x = values[i].y = 0.0;
    plotViewSet = false;
}

void FunctionPlot::setValue( int i, valuePair v ) {
    if ( i >= valueNo || i < 0 ) return;
    values[i] = v;
}

```

```

}

void FunctionPlot::setValues( int n, valuePair v[] ) {
    for ( int i=0; i<n; i++ )
        setValue( i, v[i] );
}

void FunctionPlot::setPlotView( double minx, double miny,
                                double maxx, double maxy ) {
    plotViewSet = true;
    minX = minx;  minY = miny;  maxX = maxx;  maxY = maxy;
}

void FunctionPlot::plotIt( void ) {
    if ( !plotViewSet ) {
        minX = maxX = values[0].x;
        minY = maxY = values[0].y;
        for ( int i=1; i<valueNo; i++ ) {
            if ( values[i].x < minX ) minX = values[i].x;
            if ( values[i].x > maxX ) maxX = values[i].x;
            if ( values[i].y < minY ) minY = values[i].y;
            if ( values[i].y > maxY ) maxY = values[i].y;
        }
        //.... Sicherstellen, dass Achsen sichtbar
        if ( minX > 0 ) minX = 0;
        if ( maxX < 0 ) maxX = 0;
        if ( minY > 0 ) minY = 0;
        if ( maxY < 0 ) maxY = 0;
    }
    repaint();
}

void FunctionPlot::keyPressEvent( QKeyEvent *ev ) {
    double diffX = (maxX-minX) / 10,  diffY = (maxY-minY) / 10;
    switch (ev->key() ) {
        case Key_Right:  minX -= diffX; repaint(); break;
        case Key_Left:   maxX += diffX; repaint(); break;
        case Key_Up:     minY -= diffY; repaint(); break;
        case Key_Down:   maxY += diffY; repaint(); break;
    }
}

void FunctionPlot::mousePressEvent( QMouseEvent *ev ) {
    ersteMal = true;  dragging = false;  startPos = ev->pos();
    repaint( false );
}

void FunctionPlot::mouseMoveEvent( QMouseEvent *ev ) {
    if ( !dragging )
        //.... Dragging (Ziehen) einschalten, wenn
        //.... neue Position mind. 20 Pixel entfernt ist
        if ( QABS( startPos.x() - ev->pos().x() ) >= 20 ||
            QABS( startPos.y() - ev->pos().y() ) >= 20 )
            dragging = true;
    if ( dragging ) {
        neuePos = ev->pos();
    }
}

```

## 17 Erstellen eigener Widgets

```
//.... Farbe zeigt an, ob Zoom aktiviert ist
if ( QABS( startPos.x() - ev->pos().x() ) >= 50 &&
      QABS( startPos.y() - ev->pos().y() ) >= 50 )
    farbe = Qt::green.light( 60 );
else
    farbe = Qt::red.light( 150 );
repaint( false );
}
}

void FunctionPlot::mouseReleaseEvent( QMouseEvent *ev ) {
    //.... Bedingung soll versehentliches Zoomen verhindern
    if ( QABS( startPos.x() - ev->pos().x() ) >= 50 &&
          QABS( startPos.y() - ev->pos().y() ) >= 50 ) {
        if ( ev->pos().x() > startPos.x() ) {
            maxX = ev->pos().x()/xFactor+minX; minX = startPos.x()/xFactor+minX;
        } else {
            maxX = startPos.x()/xFactor+minX; minX = ev->pos().x()/xFactor+minX;
        }
        if ( ev->pos().y() > startPos.y() ) {
            minY = maxY-ev->pos().y()/yFactor; maxY = maxY-startPos.y()/yFactor;
        } else {
            minY = maxY-startPos.y()/yFactor; maxY = maxY-ev->pos().y()/yFactor;
        }
    }
    dragging = false;
    repaint();
}

void FunctionPlot::paintEvent( QPaintEvent * ) {
    QPainter p( this );
    double xp, yp, xpAlt, ypAlt, diffX, diffY, i;
    xFactor = width() / (maxX -minX),
    yFactor = height() / (maxY -minY);
    if ( dragging ) {
        if ( !ersteMal )
            p.eraseRect( QRect( startPos, letztePos ) );
        p.fillRect( QRect( startPos, neuePos ), farbe );
        letztePos = neuePos; ersteMal = false;
    }
    p.setPen( Qt::yellow ); //.... x- und y-Achsen zeichnen
    xp = -minX * xFactor; yp = maxY * yFactor;
    p.drawLine( 0, int(yp), width(), int(yp) ); // x-Achse
    p.drawLine( int(xp), 0, int(xp), height() ); // y-Achse
    diffX = (maxX-minX) / 10; //.... x-Skalen zeichnen/beschriften
    for ( i = -diffX; i>=minX; i+=diffX )
        drawXScale( &p, i, int(yp) );
    for ( i = diffX; i<=maxX; i+=diffX )
        drawXScale( &p, i, int(yp) );
    diffY = (maxY-minY) / 10; //.... y-Skalen zeichnen/beschriften
    for ( i = -diffY; i>=minY; i+=diffY )
        drawYScale( &p, i, int(xp) );
}
```

```

for ( i = diffY; i<=maxY; i+=diffY )
    drawYScale( &p, i, int(xp) );

                                //.... Funktion zeichnen
xpAlt = xp = ( values[0].x - minX ) * xFactor;
ypAlt = yp = ( maxY - values[0].y ) * yFactor;
p.setPen( Qt::black );    p.setBrush( Qt::black );
p.drawEllipse( int(xp-1), int(yp-1), 2, 2 );
for ( i=1; i<valueNo; i++ ) {
    xp = ( values[(int)i].x - minX ) * xFactor;
    yp = ( maxY - values[(int)i].y ) * yFactor;
    p.setPen( Qt::blue.light( 140 ) );
    p.drawLine( int(xpAlt), int(ypAlt), int(xp), int(yp) );
    p.setPen( Qt::blue.light( 120 ) );
    p.drawEllipse( int(xp), int(yp), 2, 2 );
    xpAlt = xp;    ypAlt = yp;
}
}

void FunctionPlot::drawXScale( QPainter *p, double i, int yp ) {
    QString text;    double xs = (i-minX) * xFactor;
    text.sprintf( "%.3g", i );
    p->drawLine( int(xs), int(yp-2), int(xs), int(yp+2) );
    p->drawText( int(xs+1), int(yp-2), text );
    p->setPen( QPen( Qt::yellow, 0, Qt::DotLine ) ); // Raster
    p->drawLine( int(xs), 0, int(xs), height() );
    p->setPen( QPen( Qt::yellow, 0, Qt::SolidLine ) );
}

void FunctionPlot::drawYScale( QPainter *p, double i, int xp ) {
    QString text;    double ys = (maxY-i) * yFactor;
    text.sprintf( "%.3g", i );
    p->drawLine( int(xp-2), int(ys), int(xp+2), int(ys) );
    p->drawText( int(xp+4), int(ys), text );
    p->setPen( QPen( Qt::yellow, 0, Qt::DotLine ) ); // Raster
    p->drawLine( 0, int(ys), width(), int(ys) );
    p->setPen( QPen( Qt::yellow, 0, Qt::SolidLine ) );
}

```

### 17.2.4 Plotten einer Sinusfunktion mit Funktionsplotter-Widget<sup>B</sup>

Nachdem das Funktionsplotter-Widget entworfen wurde, müssen wir es auch testen. Dazu soll eine einfache Sinus-Funktion aus dem Bereich von  $-2\pi$  bis  $+3\pi$  geplottet werden, wie in Programm 17.3 gezeigt.

Programm 17.3 – sinusfunc.cpp:

Plotten einer Sinusfunktion mit dem eigenen Funktionsplotter

```

#include <qapplication.h>
#include <qwidget.h>
#include <math.h>
#include "functionplot.h"

```



## 17 Erstellen eigener Widgets

```
int main( int argc, char *argv[] ) {
    QApplication myapp( argc, argv );
    const double pi = 4*atan(1);
    valuePair      v;
    int            i=0, z=0;
    for ( double x=-2*pi; x<3*pi; x+=0.01 ) // Zaehlen der Werte
        z++;
    FunctionPlot* plotWindow = new FunctionPlot( z );
    plotWindow->resize( 500, 500 );
    for ( v.x=-2*pi; v.x<=3*pi; v.x+=0.01 ) {
        v.y = sin(v.x);
        plotWindow->setValue( i, v );
        i++;
    }
    plotWindow->plotIt();
    myapp.setMainWidget( plotWindow );
    plotWindow->show();
    return myapp.exec();
}
```

Startet man nun Programm 17.3, blendet es das links in Abbildung 17.1 gezeigte Widget ein. Markiert man nun durch Ziehen der Maus einen bestimmten Funktionsbereich, den man „herauszoomen“ möchte (siehe rechts in Abbildung 17.1), und lässt anschließend die Maustaste los, wird die alte Funktionsanzeige gelöscht und nur der zuvor markierte Teil der Funktion entsprechend vergrößert im Widget angezeigt (siehe links in Abbildung 17.2). Möchte der Benutzer die angezeigte Sinusfunktion verkleinern, also den Funktionsbereich vergrößern, so kann er dies mit den Cursorstasten erreichen (siehe rechts in Abbildung 17.2).

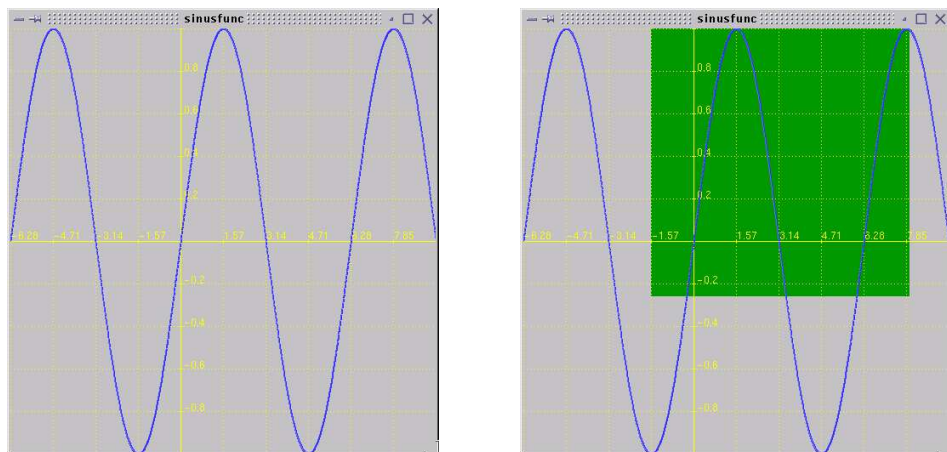


Abbildung 17.1: Sinusfunktion von  $-2\pi$  bis  $+3\pi$  und Markieren für Zoom

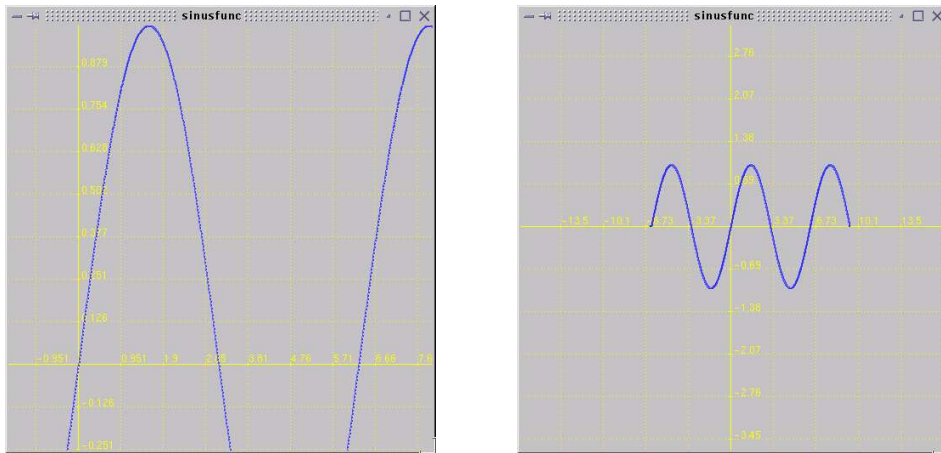


Abbildung 17.2: Gezoomte und verkleinerte Sinusfunktion

### 17.2.5 Plotten beliebiger Funktionen mit Funktionsplotter-Widget<sup>Z</sup>

Das Plotten einer Sinusfunktion ist zwar ganz nett, aber wir wollen nun noch den Test unseres Funktionsplotters weiterführen und dem Benutzer den Term der zu plottenden Funktion interaktiv eingeben lassen. Dazu muss zunächst ein Funktionsparser erstellt werden. Programm `parser.h` realisiert die Headerdatei und Programm `parser.cpp` die Implementierung des Funktionsparsers. Abbildung 17.3 zeigt den Funktionsparser im Einsatz. Natürlich kann die angezeigte Funktion wieder mit Ziehen der Maustaste und den Cursortasten entsprechend gezoomt werden, da diese Funktionalität über den selbst definierten Funktionsplotter zur Verfügung gestellt wird.

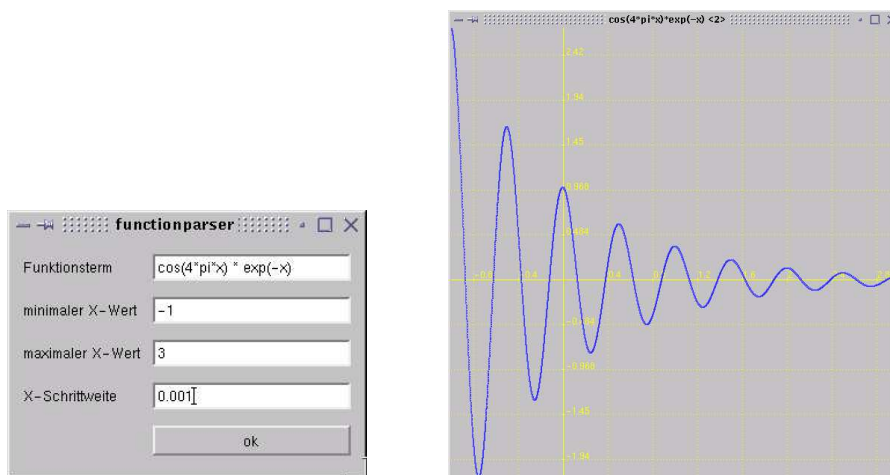


Abbildung 17.3: Eingabe eines Funktionsterms und Anzeige der entsprechenden Funktion

### 17.2.6 Plotten der Funktion $1 - \exp(-0.5x)$ und deren Ableitung<sup>Z</sup>

Das Programm `functionplot2.cpp` stellt unter Zuhilfenahme des in diesem Kapitel eingeführten Funktionsplotters aus den Programmen 17.1 und 17.2 in einem Widget links die Funktion  $1 - \exp(-0.5x)$  und rechts die Ableitung dieser Funktion  $0.5 \exp(-0.5x)$  dar (siehe dazu auch Abbildung 17.4).

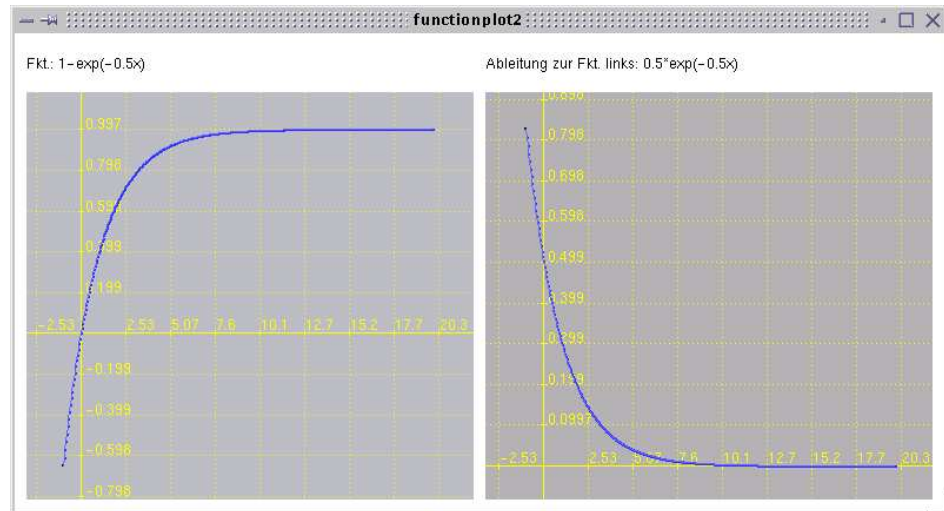


Abbildung 17.4: Anzeige der Funktion  $1 - \exp(-0.5x)$  (links) und deren Ableitung (rechts)

## 17.3 Ein Bildertabellen-Widget<sup>B</sup>

Als zweites Beispiel für ein selbstdefiniertes Widget soll ein Widget entworfen werden, das Bilder in Tabellenform anzeigen kann.

### 17.3.1 Beschreibung des Bildertabellen-Widget

Die Anzahl der Zeilen und Spalten des Bildertabellen-Widget soll der Benutzer über den Konstruktor bestimmen.

Folgende `public`-Methoden soll dieses Bildertabellen-Widget zur Verfügung stellen:

```
setCell(int i, int j, QString bildName)
```

```
setCell(int i, int j, QPixmap bild)
```

legen für Zelle  $(i, j)$  im Bildertabellen-Widget den Dateinamen des anzuzeigenden Bildes bzw. das Bild selbst fest.

```
setCellSize(int w, int h, int border=0, int frameWidth=0)
```

definiert für alle Zellen der Bildertabelle deren Breite ( $w$ ) und Höhe ( $h$ ). Über die beiden Parameter `border` kann dabei zusätzlich noch für die Zellen ein frei zu bleibender Rand, und über den Parameter `frameWidth` eine Rahmenbreite festgelegt werden.

```
setCellMark(int i, int j, bool mark)
```

setzt für die Zelle (i, j) im Bildertabellen-Widget eine Markierung (mark=true) bzw. löscht eine Markierung für die Zelle (i, j) bei mark=false.

```
bool isCellMarkSet(int i, int j)
```

liefert zurück, ob für Zelle (i, j) eine Markierung gesetzt ist oder nicht.

```
resetAllMarks(void)
```

Diese Slotroutine löscht alle gesetzten Markierungen in der Bildertabelle.

Folgende Signale soll dieses Bildertabellen-Widget schicken, wenn entsprechende Ereignisse auftreten:

```
pressPos(int z, int s)
```

wird geschickt, wenn eine Maustaste über der Zelle (z, s) im Bildertabellen-Widget gedrückt wird.

```
doubleClickPos(int z, int s)
```

wird gesendet, wenn ein Maus-Doppelklick über der Zelle (z, s) im Bildertabellen-Widget auftritt.

```
movePos(int z, int s)
```

wird geschickt, wenn die Maus bei gedrückter Maustaste über die Zelle (z, s) im Bildertabellen-Widget bewegt wird.

```
releasePos(int z, int s)
```

wird zurückgegeben, wenn eine gedrückte Maustaste über der Zelle (z, s) im Bildertabellen-Widget wieder losgelassen wird.

```
keyPressed(int key)
```

wird geschickt, wenn eine Taste von der Tastatur im Bildertabellen-Widget gedrückt wird. Der Parameter key liefert dabei den entsprechenden Tastencode.

### 17.3.2 Headerdatei für das Bildertabellen-Widget

Anders als beim vorherigen Beispiel (Funktionsplotter) existiert zu dieser Aufgabenstellung bereits ein geeignetes Qt-Widget, nämlich `QGridView`, welches als Basisklasse für das Bildertabellen-Widget herangezogen werden kann. Programm 17.4 zeigt die Headerdatei für das Bildertabellen-Widget.

Programm 17.4 – `bildtabelle.h`:

Headerdatei für das Bildertabellen-Widget

```
#ifndef BILDAUSWAHL_H
#define BILDAUSWAHL_H
#include <qgridview.h>
#include <qpixmap.h>
#define POSITION rowAt(e->y()), columnAt(e->x())
//..... Klasse Bildtabelle
class Bildtabelle : public QGridView {
    Q_OBJECT
public:
    //... Konstruktor, dem die Anzahl der Zeilen und Spalten
    //... der Bildertabelle uebergeben werden
    Bildtabelle( int zeilZahl, int spaltZahl, QWidget* p=0, const char* nam=0 );
    //... Destruktor; gibt den für die Bilder reservierten
    //... Speicherplatz wieder frei
    ~Bildtabelle() { delete[] bilder; }
```

```
//... legt fuer Zelle (i,j) den Dateinamen des
//... anzuzeigenden Bildes bzw. das Bild selbst fest
void setCell( int i, int j, QString bildName );
void setCell( int i, int j, QPixmap bild );
//... legt fuer alle Zellen deren Breite (w) und Hoehe (h) fest; auch
//... Festlegung einer Rand- (border) und Rahmenbreite (frameWidth) möglich
void setCellSize( int w, int h, int border = 0, int frameWidth = 0 );
//... setzt fuer Zelle (i,j) eine Markierung (mark=true) bzw.
//... loescht eine Markierung fuer Zelle (i,j) bei mark=false
void setCellMark( int i, int j, bool mark );
//... liefert zurueck, ob fuer Zelle (i,j) eine Markierung
//... Markierung gesetzt ist oder nicht
bool isCellMarkSet( int i, int j ) { return marks[ indexOf( i, j ) ]; }

public slots:
    //... loescht die Markierungen aller Zellen
    void resetAllMarks( void );

signals:
    void pressPos( int z, int s );
    void doubleClickPos( int z, int s );
    void movePos( int z, int s );
    void releasePos( int z, int s );
    void keyPressed( int key );

protected:
    void paintCell( QPainter*, int zeile, int spalte );
    void mousePressEvent( QMouseEvent *e)          { emit pressPos( POSITION ); }
    void mouseDoubleClickEvent( QMouseEvent *e) { emit doubleClickPos( POSITION ); }
    void mouseMoveEvent( QMouseEvent *e)          { emit movePos( POSITION ); }
    void mouseReleaseEvent( QMouseEvent *e)        { emit releasePos( POSITION ); }
    void keyPressEvent( QKeyEvent *e)              { emit keyPressed( e->key() ); }

private:
    int indexOf( int zeil, int spalt ) const { return ( zeil * numCols() ) + spalt; }
    QPixmap *bilder;
    bool *marks;
    int border, frameWidth;
};
#endif
```

### 17.3.3 Implementierung des Bildertabellen-Widget

Programm 17.5 zeigt die Implementierung des Bildertabellen-Widget:

Programm 17.5 – bildtabelle.cpp:

Implementierung des Bildertabellen-Widget

```
#include <qwidget.h>
#include <qpainter.h>
#include <qkeycode.h>
#include <qpainter.h>
#include <qdrawutil.h>
#include "bildtabelle.h"
```

```

Bildtabelle::Bildtabelle( int zeilZahl, int spaltZahl,
                          QWidget *p, const char *nam ) : QGridView(p,nam) {
    setFocusPolicy( StrongFocus ); // Tastaturfokus ist erlaubt
    setNumCols( spaltZahl ); // Festlegen der Spalten- und
    setNumRows( zeilZahl ); // Zeilenzahl in der Tabelle
    setCellWidth( 100 ); // Voreingest. Festlegen der Breite und
    setCellHeight( 100 ); // Hoehe der Zellen (in Pixel)
    frameWidth = border = 0; resize( 600, 600 );
    bilder = new QPixmap [zeilZahl * spaltZahl]; // Bilder
    marks = new bool [zeilZahl * spaltZahl]; // Markierungen
    resetAllMarks();
}

void Bildtabelle::setCell( int i, int j, QString bildName ) {
    if ( i < 0 || i >= numRows() || j < 0 || j >= numCols() ) return;
    bilder[ indexOf( i, j ) ].load( bildName );
    repaintCell( i, j, false );
}

void Bildtabelle::setCell( int i, int j, QPixmap bild ) {
    if ( i < 0 || i >= numRows() || j < 0 || j >= numCols() ) return;
    bilder[ indexOf( i, j ) ] = bild;
    repaintCell( i, j, false );
}

void Bildtabelle::setCellSize( int w, int h, int border, int frameWidth ) {
    setCellWidth( w ); setCellHeight( h );
    this->border = border; this->frameWidth = frameWidth;
}

void Bildtabelle::setCellMark( int i, int j, bool mark ) {
    if ( i < 0 || i >= numRows() || j < 0 || j >= numCols() ||
        marks[ indexOf( i, j ) ] == mark )
        return;
    marks[ indexOf( i, j ) ] = mark;
    repaintCell( i, j, false );
}

void Bildtabelle::resetAllMarks( void ) {
    for ( int i=0; i<numRows(); i++ )
        for ( int j=0; j<numCols(); j++ )
            setCellMark( i, j, false );
}

void Bildtabelle::paintCell( QPainter* p, int zeile, int spalte ) {
    int width = cellWidth(), height = cellHeight();
    double w = width-border*2-frameWidth*2, h = height-border*2-frameWidth*2;
    QBrush b( Qt::lightGray );
    if ( marks[ indexOf( zeile, spalte ) ] ) {
        b.setColor( Qt::blue );
        qDrawShadeRect (p, 0, 0, width, height, colorGroup(), true,
                       frameWidth, 0, &b );
    } else
        qDrawShadeRect (p, 0, 0, width, height, colorGroup(), false,
                       frameWidth, 0, &b );
    //... Bild ausgeben
}

```

```

QPixmap bild = bilder[ indexOf( zeile, spalte ) ];
double  xFactor = w / bild.width(),  yFactor = h / bild.height();
p->scale( xFactor, yFactor );
p->drawPixmap( int((frameWidth+border)/xFactor),
               int((frameWidth+border)/yFactor), bild );
p->scale( 1/xFactor, 1/yFactor );
}

```

### 17.3.4 Ein Memory-Spiel mit dem Bildertabellen-Widget<sup>B</sup>

Nachdem das Bildtabellen-Widget entworfen wurde, müssen wir es auch testen. Dazu soll zunächst ein kleines Memory-Spiel entwickelt werden, wie in Programm 17.6 gezeigt. Es blendet verdeckte Bilder ein, die man durch Mausklick aufdecken kann. Deckt man zwei gleiche Bilder auf, bleiben diese aufgedeckt und erhalten einen blauen Hintergrund, andernfalls werden sie wieder verdeckt (siehe Abbildung 17.5). Durch einen Maus-Doppelklick kann ein neues Memory-Spiel gestartet werden.

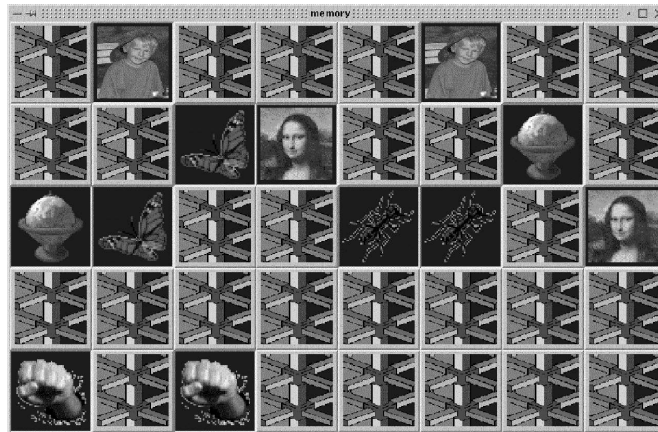


Abbildung 17.5: Memory-Spiel mit dem Bildertabellen-Widget

Programm 17.6 – memory.cpp:

Memory-Spiel unter Verwendung des Bildertabellen-Widget

```

#include <qapplication.h>
#include <qwidget.h>
#include <qmessagebox.h>
#include <stdlib.h>
#include <time.h>
#include "bildtabelle.h"
const char *picNames[] = {
    "Ameise.xpm",    "Butterfly.xpm", "monalisa.xpm", "Deutschl.xpm",
    "Grossbri.xpm", "Krebs.xpm",    "Pinguin.xpm",  "Suse.xpm",
    "lausbub.xpm",  "marble.xpm",   "ruins.xpm",    "calvin2.xpm",
    "Wizard.xpm",   "HandReach.xpm", "HomeOpen.xpm", "HandOpen.xpm",
    "HandPunch.xpm", "Eggs.xpm",     "3dpaint.xpm",  "DeskGlobe.xpm" };
const int picNo = sizeof( picNames ) / sizeof( *picNames );

```

```

const int cardNo = picNo * 2;
const int bord    = 5;
const int frameWid = 2;
const int cellWid  = 100 + 2*bord + 2*frameWid;
const int cellHigh = 100 + 2*bord + 2*frameWid;
class Memory : public Bildtabelle {
    Q_OBJECT
public:
    Memory( int zeilen, int spalten, QWidget* p=0, const char* name=0 )
        : Bildtabelle( zeilen, spalten, p, name ) {

        ::srand( time(NULL) );
        zeilZahl = zeilen;
        spaltZahl = spalten;
        setCellSize( cellWid, cellHigh, bord, frameWid );
        if ( !back.load( "back.bmp" ) ) {
            QMessageBox::information( 0, "Ladefehler",
                "Kann Bild-Datei 'back.bmp' nicht laden" , QMessageBox::Ok );
            qApp->quit();
        }
        for ( int i=0; i<picNo; i++ )
            if ( !bild[i].load( picNames[i] ) ) {
                QString text;
                text.sprintf( "Kann Bild-Datei '%s' nicht laden", picNames[i] );
                bild[i].resize( cellWid, cellHigh );
                bild[i].fill( QColor( ::rand()&255,::rand()&255,::rand()&255 ) );
                QMessageBox::information( 0, "Ladefehler", text, QMessageBox::Ok );
                qApp->quit();
            }
        QObject::connect( this, SIGNAL( pressPos( int, int ) ),
            this, SLOT( pressHandle( int, int ) ) );
        QObject::connect( this, SIGNAL( doubleClickPos( int, int ) ),
            this, SLOT( doubleClickHandle( int, int ) ) );
        newGame();
    }
    void newGame( void ) {
        int i, c;
        oldIndex1 = oldIndex2 = -1;
        clickNo = 0;
        resetAllMarks();
        for ( i=0; i<picNo; i++ )
            picDrawn[ i ] = 0;
        for ( i=0; i<cardNo; i++ ) {
            do { } while ( picDrawn[ c = ::rand()%picNo ] >= 2 );
            picDrawn[ c ]++;
            boardCard[i] = c;
            setCell( i/spaltZahl, i%spaltZahl, back );
            boardCardOpen[ i ] = false;
        }
    }
private slots:

```



```

void pressHandle( int z, int s ) {
    int index = z*spaltZahl + s;
    if ( isCellMarkSet( z, s ) || boardCardOpen[ index ] )
        return;
    if ( ++clickNo >= 3 ) {
        clickNo = 1;
        if ( boardCard[ oldIndex1 ] != boardCard[ oldIndex2 ] ) {
            setCell( oldIndex1/spaltZahl, oldIndex1%spaltZahl, back );
            setCell( oldIndex2/spaltZahl, oldIndex2%spaltZahl, back );
            boardCardOpen[ oldIndex1 ] = boardCardOpen[ oldIndex2 ] = false;
        }
    }
    if ( clickNo == 1 )
        oldIndex1 = index;
    else if ( clickNo == 2 ) {
        oldIndex2 = index;
        if ( boardCard[ oldIndex1 ] == boardCard[ oldIndex2 ] ) {
            setCellMark( oldIndex1/spaltZahl, oldIndex1%spaltZahl, true );
            setCellMark( oldIndex2/spaltZahl, oldIndex2%spaltZahl, true );
        }
    }
    setCell( index/spaltZahl, index%spaltZahl, bild[ boardCard[index] ] );
    boardCardOpen[ index ] = true;
}

void doubleClickHandle( int, int ) { newGame(); }

private:
    int        zeilZahl, spaltZahl;
    QPixmap    back;
    QPixmap    bild[ picNo ];
    int        picDrawn[ picNo ];
    int        boardCard[ cardNo ];
    bool        boardCardOpen[ cardNo ];
    int        clickNo;
    int        oldIndex1, oldIndex2;
};

#include "memory.moc"

int main( int argc, char *argv[] ) {
    QApplication a(argc,argv);
    int lines = cardNo/8,
        cols = 8;
    if ( cardNo%8 != 0 )
        lines++;
    Memory *m = new Memory( lines, cols );
    a.setMainWidget( m );
    m->resize( 8*cellWid+5, 5*cellHigh+5 );
    m->show();
    return a.exec();
}

```

### 17.3.5 Ein Puzzle-Spiel mit dem Bildertabellen-Widget<sup>Z</sup>

Als weiteres Beispiel zum Testen des Bildtabellen-Widget soll ein kleines Puzzle-Spiel entwickelt werden, bei dem der Benutzer durcheinander gewürfelte Puzzle-Teile eines Bildes richtig ordnen muss. Bei diesem Puzzle-Spiel kann der Benutzer zwei Puzzle-Teile vertauschen, indem er auf ein Puzzle-Teil klickt und dann die Maus bei gedrückter Maustaste auf den Zielort bewegt. Lässt er die Maustaste los, werden das angeklickte Puzzle-Teil und das Puzzle-Teil, über dem er die Maustaste losließ, vertauscht. Befindet sich ein Puzzle-Teil am richtigen Ort, wird dies durch einen blauen Rand um dieses Puzzle-Teil, das nun auch nicht mehr wegbewegt werden kann, angezeigt (siehe auch Abbildung 17.6). Durch einen Maus-Doppelklick kann ein neues Puzzle-Spiel gestartet werden. Das Programm `puzzle.cpp` zeigt die Implementierung dieses Puzzle-Spiels unter Zuhilfenahme unseres Bildtabellen-Widget.

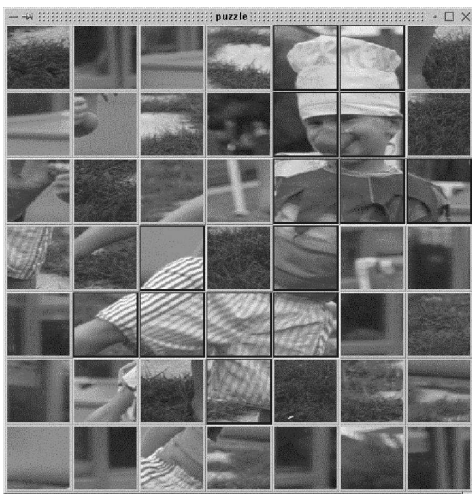


Abbildung 17.6: Puzzle-Spiel mit dem Bildtabellen-Widget

### 17.3.6 Ein Poker-Spiel mit dem Bildertabellen-Widget<sup>Z</sup>

Als weiteres Beispiel zum Testen des Bildtabellen-Widget soll ein interaktives Poker-Spiel entwickelt werden, bei dem dem Benutzer immer zunächst fünf zufällig ausgewählte Poker-Karten angezeigt werden. Er kann nun die Karten mit einem Mausklick markieren, die er behalten möchte, bevor er sich dann für die übrigen unmarkierten Karten neue geben lässt. Markierte Karten werden durch einen blauen Rand angezeigt. Nachdem sich der Benutzer durch einen Klick auf den Button *Draw* neue Karten hat geben lassen, ist eine Runde beendet und ihm wird angezeigt, was er gewonnen hat. Die Karten, die zu dem Gewinn führten, werden dabei wieder mit einem blauen Rand angezeigt. Mit einem Klick auf den Button *Draw* kann dann der Benutzer die nächste Runde starten. Abbildung 17.7 zeigt eine Runde bei diesem Pokerspiel: links die vom Benutzer markierten und behaltenen Karten; rechts das Aussehen der Karten nach Klicken auf den Button *Draw*. Die Karten, die zu seinem Gewinn „Zwei Paare“ führten, sind dabei blau markiert. Programm `poker.cpp` zeigt die Implementierung dieses Poker-Spiels unter Zuhilfenahme unseres Bildtabellen-Widget.

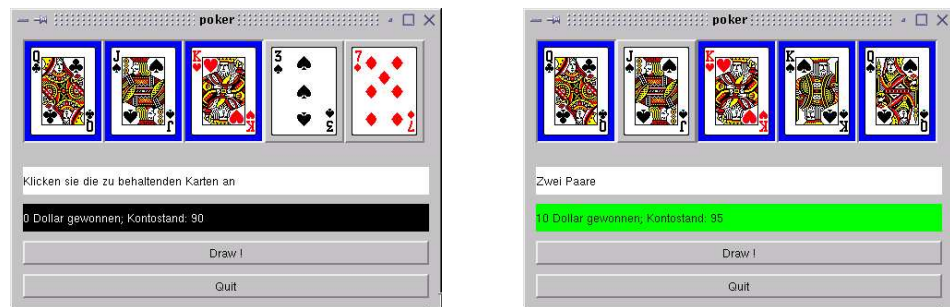


Abbildung 17.7: Eine Runde im Poker-Spiel

### 17.3.7 Ein Bilderbrowser mit Dia-Vorschau mit dem Bildertabellen-Widget<sup>Z</sup>

Das Programm `picoverview.cpp` blendet unter Zuhilfenahme des in diesem Kapitel eingeführten Bildertabellen-Widget aus den Programmen 17.4 und 17.5 alle auf der Kommandozeile angegebenen Bilddateien als Dias ein. Mittels der Cursortasten oder aber mit einem einfachen Mausklick auf das entsprechende Bild kann der Benutzer sich ein Bild auswählen. Das gerade aktuelle Bild wird immer mit einem blauen Rahmen gekennzeichnet (siehe auch Abbildung 17.8). Das Anzeigen eines Bildes in seiner Originalgröße kann der Benutzer entweder mit Drücken der (↵)-Taste erreichen, wobei ihm dann das aktuell markierte Bild angezeigt wird, oder aber auch mit einem Maus-Doppelklick auf ein beliebiges Dia. Zusätzlich wird dem Benutzer noch beim Bewegen der Maus über die Dias immer der Dateiname als *Bubble help* angezeigt, wenn er den Mauszeiger auf ein anderes Dia bewegt.

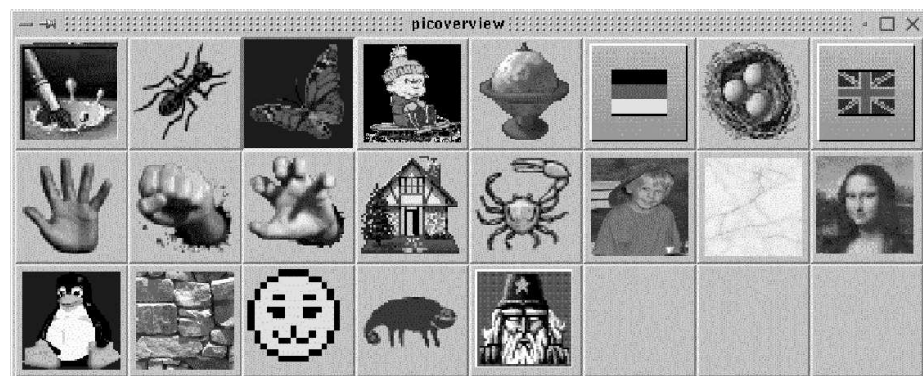


Abbildung 17.8: Auswahl von Bildern, die als Dias angezeigt werden

# Kapitel 18

## Zugriff auf Datenbanken

*Das Vernünftigste ist immer, dass jeder sein Metier treibe, wozu er geboren ist und was er gelernt hat, und dass er den anderen nicht hindere, das Seinige zu tun.*

– J. W. Goethe

Qt3 unterstützt auch Zugriffe auf Datenbanken, wobei zur Zeit die folgenden Datenbanken unterstützt werden:

QMYSQL3	MySQL 3.x und MySQL 4.x
QOCI8	Oracle Call Interface (OCI)
QODBC3	Open Database Connectivity (schliesst Microsoft SQL-Server mit ein)
QPSQL7	PostgreSQL Version 6 und 7
QTDS7	Sybase Adaptive Server
QDB2	IBM DB2 Driver (v7.1 oder höher)

Jedoch werden nicht alle diese Datenbanken in der *Qt Free Edition* unterstützt.

### 18.1 Installation der SQL-Module von Qt

Standardmäßig sind die SQL-Module wegen ihrer Größe nicht zu Qt hinzugebunden. Um die SQL-Module verwenden zu können, gibt es zwei Möglichkeiten:

#### ❑ Neue Generierung von Qt

Um Qt unter Linux/Unix bzw. Mac OS X neu zu generieren, muss man im QTDIR-Directory `./configure` mit entsprechenden Optionen aufrufen, wie z. B.:

```
./configure .. -qt-sql-mysql ..          # Dazubinden des MYSQL-Treibers oder
./configure .. -qt-sql-oci -qt-sql-odbc .. # der Oracle- und ODBC-Treiber
```

Weitere Optionen zu `configure` lassen sich mit folgenden Aufruf anzeigen:

```
./configure -help
```

Unter Windows muss man im Installationsprogramm zunächst SQL auf der entsprechenden Tabseite und dann in der Advanced-Tabseite die gewünschten Module auswählen. In beiden Fällen ist jedoch zu beachten, dass man `configure` bzw. dem Installationsprogramm die Pfade der Headerdateien und Bibliotheken für die ausgewählten Datenbanken mitteilt. Diese kann man `configure` über die Optionen `-I` und `-L` mitteilen.

# Index

## Symbole

Überlappendes Layout .. 215

## A

Animation ..... 460, 499  
 ASSERT Makro ..... 652  
 assistant Tool ..... 9  
 Austritts-Events ..... 559  
 Auswahlwidgets ..... 90

## B

Balloon help ..... 139, 142  
 Bedingungs-Variablen .. 638  
 Benutzerdefinierte Events ...  
     568  
 Benutzerdefiniertes Layout .  
     214  
 Beschleuniger ..... 122  
 Bibliothek  
     dynamisch ..... 750  
 Bildformate ..... 471  
 Browser ..... 107  
     Directory ..... 162  
 Bubble help ..... 139, 142  
 Buttongruppe ..... 189  
 Buttons ..... 83

## C

CHECK\_PTR Makro ..... 653  
 Clipboard ..... 363  
     mit Drag-and-Drop ... 375  
 Clipping (Graphik) ..... 440  
 Close-Events ..... 560  
 ColorRole Typ ..... 39  
 connect () ..... 23, 573  
 Containerklasse  
     QPtrList ..... 343  
     Hashtabelle ..... 326  
     Listen ..... 343  
     QAsciiCache ..... 338  
     QAsciiDict ..... 333

QBitArray ..... 319  
 QByteArray ..... 319  
 QCache ..... 335  
 QDict ..... 326  
 QIntCache ..... 338  
 QIntDict ..... 333  
 QMap ..... 339  
 QMemArray ..... 314  
 QObjectList ..... 357  
 QPointArray ..... 322  
 QPtrDict ..... 333  
 QPtrQueue ..... 361  
 QPtrStack ..... 357  
 QPtrVector ..... 319  
 QStrIList ..... 354  
 QStringList ..... 353  
 QStrList ..... 354  
 QValueList ..... 349  
 QValueStack ..... 359  
 QValueVector .. 319, 362  
 Containerklassen ..... 303  
 critical () ..... 55  
 Cursorformen ..... 471, 478

## D

Danksagung ..... VII  
 Data-Sharing ..... 312  
 Dateiauswahl-Dialogbox 230  
 Dateioperationen ..... 267  
 Datenbanken ..... 601  
 Datenstruktur  
     QPtrList ..... 343  
     FIFO ..... 361  
     Hashtabelle ..... 326  
     LIFO ..... 357  
     Listen ..... 343  
     QAsciiCache ..... 338  
     QAsciiDict ..... 333  
     QBitArray ..... 319  
     QByteArray ..... 319  
     QCache ..... 335

QDict ..... 326  
 QIntCache ..... 338  
 QIntDict ..... 333  
 QMap ..... 339  
 QMemArray ..... 314  
 QPointArray ..... 322  
 QPtrDict ..... 333  
 QPtrQueue ..... 361  
 QPtrStack ..... 357  
 QPtrVector ..... 319  
 Queue ..... 361  
 QValueList ..... 349  
 QValueStack ..... 359  
 QValueVector .. 319, 362  
 Stack ..... 357  
     Warteschlange ..... 361  
 Datenstrukturen ..... 303  
 Datentypen ..... 303  
 Datum ..... 383  
 Datum und Zeit ..... 393  
 Debugging ..... 651  
 deep copy-Sharing ..... 312  
 Dialogbox ..... 223  
     Dateiauswahl ..... 230  
     Drucker ..... 258  
     Einfache Eingabe .... 252  
     Farbe ..... 227  
     Fehlermeldung ..... 244  
     Font ..... 236  
     Fortschrittsanzeige ... 153,  
         258  
     Karteikarten ..... 245  
     Kombobox-Eingabe ... 252  
     Message ..... 239  
     Mitteilungen ..... 239  
     Nachrichten ..... 239  
     Tabdialog ..... 245  
     Texteingabe ..... 252  
     Wizards ..... 250  
     Zahleneingabe ..... 252  
 Dialogfenster ..... 223

## Index

- Directorybrowser ..... 162
- Directoryoperationen .. 267, 290
- Dock-Windows ..... 131
- Dokumentation ..... 7
- DOM-Schnittstelle ..... 713
- Doppel-Pufferung ..... 450
- Drag-and-Drop ..... 367
  - MIME-Typen ..... 376
  - mit Clipboard ..... 375
- Drag-Events ..... 368, 562
- Drehknopf ..... 97
- Drop-Events ..... 368, 562
- Druckerdialog ..... 258
- Druckerinformationen .. 262
- dumpObjectInfo Funktion ..... 654
- dumpObjectTree Funktion ..... 654
- Dynamische Bibliothek . 750
- Dynamisches Layout ... 217
- E**
- E/A-Geräte ..... 267
- Editor
  - einzeilig ..... 113
  - mehrzeilig ..... 113
- Einfaches Layout ..... 197
- Eintritts-Events ..... 559
- Ereignisse ..... 541
- Event
  - Austritts- ..... 559
  - Benutzerdefiniert .... 568
  - Close- ..... 560
  - Custom- ..... 568
  - Drag- ..... 368, 562
  - Drop- ..... 368, 562
  - Eintritts- ..... 559
  - Fokus- ..... 553
  - Größenänderungs- ... 556
  - Hide- ..... 559
  - Mal- ..... 543
  - Maus- ..... 544
  - Positionsänderungs- .. 557
  - Resize- ..... 556
  - Schließ- ..... 560
  - Senden ..... 566
  - Show- ..... 559
  - Sichtbarkeits- ..... 559
  - Subwidget- ..... 562
  - Synthetisch ..... 566
  - Tastatur- ..... 550
  - Timer- ..... 398, 553
  - Versteck- ..... 559
- Event-Filter ..... 541, 563
- Events ..... 541
- Explizites Data-Sharing . 312
- F**
- Füllbalken ..... 150
  - einfach ..... 150
  - Fortschrittsanzeige ... 153
- Farb-Dialogbox ..... 227
- Farballozierung ..... 401
- Farbgruppen in Qt ..... 39
- Farbmodelle in Qt ..... 37
- Farbname
  - QColor-Objekte ..... 38
  - vordefinierte Strings ... 38
- Fehlermeldung ..... 244
- Filter (Event) ..... 541, 563
- Flache Kopie ..... 312
- Flimmerfreie Graphik ... 447
- Fokus ..... 533
- Fokus-Events ..... 553
- Font-Dialogbox ..... 236
- Form
  - Cursor ..... 471
- Format
  - Bilder ..... 471
  - Pixmap ..... 471
- Fortgeschrittenes Layout 200
- Fortschrittsanzeige ..... 258
- Fortschrittsanzeige Klasse .. 153
- Frames ..... 186
- FTP Protokoll ..... 678
- Funktionsplotter ..... 584
- G**
- Geschachteltes Layout .. 207
- getOpenFileName() ... 75
- getSaveFileName() ... 75
- Größenänderungs-Events ... 556
- Graphik ..... 401
  - Clipping ..... 440
  - Doppel-Pufferung .... 450
  - Füllmuster ..... 411
  - Flimmerfrei ..... 447
  - Geometrische Figuren 414, 420
  - Rasterung ..... 450
  - Text ..... 417
  - Transformation ..... 423
  - View-Transformation . 433
  - Viewport-Transformation . 436
  - Window-Transformation .. 433
  - World-Transformation 423
  - Zeichenstift ..... 411
- Gruppenboxen ..... 188
- GUI-Builder ..... 753
- Qt-Designer ..... 753
- H**
- Hashtabellen ..... 326
- Hauptfenster ..... 129
- Hide-Events ..... 559
- High-Level Events ..... 45
- Hilfstexte ..... 139, 142
- HSV-Farbmodell ..... 37
- HTML ..... 107
- HTTP Protokoll .... 679, 683
- I**
- Implizites Data-Sharing . 312
- information() ..... 55
- Informationswidgets .... 106
- Internationalisierung ... 643
- Iterator
  - QAsciiCacheIterator . 338
  - QAsciiDictIterator .. 333
  - QCacheIterator ... 337
  - QDictIterator ..... 329
  - QIntCacheIterator ... 338
  - QIntDictIterator . 333
  - QMapConstIterator ... 341
  - QMapIterator ..... 341
  - QObjectListIterator . 357
  - QPtrDictIterator . 333
  - QPtrListIterator . 346
  - QStrListIterator . 354
  - QValueListConstIterator 351
  - QValueListIterator .. 351
- K**
- Karteikarten ..... 245
- Komboboxen ..... 92
- Kompilieren
  - Qt-Programme ..... 10
- Konfigurationsdaten .... 297
- Konsistente Eingaben ... 512
- Kopie
  - flache ..... 312
  - tiefe ..... 312
- L**
- Labels ..... 106
- Laufbalken ..... 164
- Layout ..... 197

- Überlappend ..... 215
- Benutzerdefiniert ..... 214
- Dynamisch ..... 217
- Einfach ..... 197
- Fortgeschritten ..... 200
- Geschachtelt ..... 207
- Tabellenform ..... 209
- LCD-Anzeige ..... 107
- Library
  - shared ..... 750
- linguist Tool ..... 646
- Listboxen ..... 90
- Listenansicht ..... 157
- Low-Level Events ..... 45
- lrelease Tool ..... 647
- lupdate Tool ..... 645
- M**
- Mainwindow ..... 128
- make Tool ..... 12
- Makefile ..... 12
- Mal-Events ..... 543
- Maus-Events ..... 544
- Mauscursor ..... 478
- MDI ..... 137
- Mehrsprachige
  - Applikationen ..... 643
- Menüleiste ..... 122
- Menüs ..... 118
  - Kontext ..... 122
  - Popup ..... 122
- Messagebox ..... 239
- Meta Object Compiler . *siehe* moc
- MIME-Typen ..... 376
- moc ..... 27
- mouseDoubleClickEvent()
  - 50
- Mutual Exclusion ..... 628
- N**
- Namen von Objekten ... 654
- Netzwerkprogrammierung .
  - 657
- nnntp Protokoll ..... 686
- O**
- Objektnamen ..... 654
- Online-Dokumentation ... 7
- Online-Hilfe ..... 139, 142
- OpenGL ..... 727
- P**
- Paletten in Qt ..... 40
- PerlQt ..... 733
- Pixmap-Format ..... 471
- Plotter ..... 584
- Popupmenü ..... 89
- Positionsänderungs-Events .
  - 557
- Programm
  - ampel.cpp ..... 641
  - ampeltimer.cpp ... 400
  - analoguhr.cpp ..... 430
  - auswahlw.cpp ..... 94
  - bildflow.cpp ..... 221
  - bildformat.cpp ... 373
  - bildtabelle.cpp .. 594
  - bildtabelle.h .... 593
  - biszeit.cpp ..... 395
  - bitarray.cpp ..... 322
  - bitblt.cpp ..... 451
  - button\_gruppe.cpp 88
  - button\_popup.cpp .. 90
  - button\_slot.cpp .. 577
  - buttons.cpp ..... 85
  - canvas.cpp ..... 468
  - canvas.h ..... 463
  - canvas2.cpp ..... 470
  - childevent.cpp ... 562
  - clipboardimage.cpp ..
    - 365
  - clipboardsend.cpp ...
    - 367
  - clipping.cpp ..... 442
  - clipping2.cpp ..... 443
  - colordialog.cpp .. 229
  - colordialog2.cpp . 229
  - countrydate.cpp .. 650
  - cursor.cpp ..... 480
  - cursor2.cpp ..... 483
  - customcolor.cpp .. 229
  - customevent.cpp .. 569
  - customevent2.cpp . 571
  - customevent3.cpp . 571
  - custommenu.cpp ... 127
  - datastream.cpp ... 282
  - datastream2.cpp .. 284
  - datediff.cpp ..... 387
  - datenbank1.cpp ... 604
  - datenbank2.cpp ... 606
  - datenbank3.cpp ... 608
  - datenbank4.cpp ... 609
  - datenbank5.cpp ... 613
  - datenbank6.cpp ... 616
  - datenbank7.cpp ... 618
  - datenbank8.cpp ... 621
  - datetimeedit.cpp . 103
  - dial.cpp ..... 106
  - dialog.cpp ..... 225
  - dialog2.cpp ..... 226
  - dialogext.cpp ..... 226
  - dict.cpp ..... 328
  - dict2.cpp ..... 329
  - dictiterator.cpp . 330
  - DigitalClock.pm .. 743
  - digitaluhr.cpp ... 397
  - digitaluhr2.cpp .. 399
  - dirbrows.cpp ..... 162
  - displaystd.in.cpp . 688
  - dom1.cpp ..... 714
  - domplot.cpp ..... 723
  - domplot2.cpp ..... 726
  - domrichtext.cpp .. 725
  - doublevalidator.cpp .
    - 519
  - download.cpp ..... 670
  - dragdropcards.cpp ...
    - 381
  - dragdropimage.cpp ...
    - 370
  - dragdropmime.cpp . 377
  - dragdroptext.cpp . 369
  - drawutil.cpp ..... 421
  - dualstack.cpp ..... 361
  - dumpobject.cpp ... 654
  - dynfokus.cpp ..... 540
  - dynlayout.cpp ..... 213
  - dynvalidator.cpp . 520
  - editor.cpp ..... 149
  - enterleave.cpp ... 559
  - entferbi.cpp ..... 35
  - errormessage.cpp . 244
  - eventfilter.cpp .. 564
  - eventfilter2.cpp . 566
  - events.cpp ..... 563
  - farbe.cpp ..... 41
  - farbmenu.cpp ..... 121
  - figrotate.cpp ..... 444
  - filebytes.cpp ..... 274
  - filedialog2.cpp .. 235
  - fileinfo.cpp ..... 287
  - filepreview.cpp .. 234
  - filetransfer.cpp . 663
  - flowlayout.cpp ... 217
  - flywindow.cpp ..... 453
  - focusevent.cpp ... 555
  - fokus1.cpp ..... 535
  - fokus2.cpp ..... 539
  - fontdialog.cpp ... 238
  - fontdialog2.cpp .. 239
  - frame.cpp ..... 187
  - frame2.cpp ..... 187
  - functionparser.cpp ..
    - 591
  - functionplot.cpp . 586
  - functionplot.h ... 585
  - functionplot2.cpp ...
    - 592

- Generierung ..... 12  
glbox.cpp ..... 729  
glpyramide.cpp ... 732  
groupbox.cpp ..... 189  
groupbox2.cpp ..... 189  
hauptfenster.cpp . 135  
hexd.cpp ..... 280  
http.cpp ..... 683  
httpd2.cpp ..... 691  
httpd3.cpp ..... 695  
huhn.cpp ..... 408  
icondrag.cpp ..... 183  
iconview.cpp ..... 180  
image.cpp ..... 490  
image2.cpp ..... 494  
inputdialog.cpp .. 255  
inputdialog2.cpp . 258  
intdict.cpp ..... 334  
intdict2.cpp ..... 335  
international.cpp ...  
    648  
intvalidator.cpp . 516  
josephus.cpp ..... 356  
katzmaus.cpp ..... 409  
kehrzahl.cpp ..... 36  
keyevent.cpp ..... 551  
keyevent2.cpp ..... 552  
kompilieren ..... 10  
layout1.cpp ..... 198  
layout2.cpp ..... 203  
layout3.cpp ..... 204  
layout4.cpp ..... 204  
layout5.cpp ..... 206  
layout6.cpp ..... 208  
layout7.cpp ..... 211  
layout8.cpp ..... 212  
lcdwandel.cpp .... 112  
lcdwandel2.cpp ... 118  
life.cpp ..... 423  
lineedit.cpp ..... 117  
linkreis.cpp ..... 432  
listbox.cpp ..... 96  
listspin.cpp ..... 101  
listview1.cpp ..... 161  
listview2.cpp ..... 163  
logging.cpp ..... 656  
lspin.cpp ..... 105  
machebi.cpp ..... 34  
maintextsize.cpp . 759  
maintextsize2.cpp ...  
    764  
maintextsize3.cpp ...  
    766  
mainwindow.cpp ... 146  
makexmlfunction.c ...  
    712  
malprog1.cpp ..... 46  
malprog2.cpp ..... 50  
malprog3.cpp ..... 52  
malprog4.cpp ..... 56  
malprog5.cpp ..... 61  
malprog6.cpp ..... 70  
manntisch.cpp .... 419  
mapdemo.cpp ..... 341  
mastermind.cpp ... 200  
meinls.cpp ..... 290  
memarray.cpp ..... 317  
memarray2.cpp ..... 318  
memory.cpp ..... 596  
menues.cpp ..... 124  
menues2.cpp ..... 127  
messagebox.cpp ... 242  
messagebox2.cpp .. 242  
mouseevent.cpp ... 548  
moveevent.cpp .... 558  
movieframes.cpp .. 507  
movies.cpp ..... 502  
movies2.cpp ..... 507  
multidownload.cpp ...  
    672  
multtable.cpp .... 179  
mutex.cpp ..... 628  
mutex2.cpp ..... 629  
mymail.cpp ..... 694  
networkprotocol.cpp .  
    686  
noflimmer1.cpp ... 448  
noflimmer2.cpp ... 449  
noflimmer3.cpp ... 452  
numausg.cpp ..... 271  
numausg2.cpp ..... 277  
oel.cpp ..... 405  
ostern.cpp ..... 388  
overlayout.cpp ... 215  
painter2.cpp ..... 418  
paintstack.cpp ... 447  
parser.cpp ..... 591  
penfill.cpp ..... 412  
perl\_attributes .. 735,  
    736  
perl\_dclock ..... 742  
perl\_helloworld .. 733  
perl\_helloworld2 . 734  
perl\_lcdwandel ... 740  
perl\_listspin ..... 744  
perl\_malprog ..... 744  
perl\_textgroes ... 739  
philosophen.cpp .. 638  
picoverview.cpp .. 600  
picture.cpp ..... 496  
pointarray.cpp ... 324  
pointarray2.cpp .. 326  
poker.cpp ..... 599  
postistda.cpp ..... 290  
potenz.cpp ..... 116  
printbild2.cpp ... 266  
printer1.cpp ..... 261  
printerinfo.cpp .. 263  
printviel.cpp ..... 266  
process.cpp ..... 747  
process2.cpp ..... 749  
progressbar.cpp .. 151  
progressbar2.cpp . 156  
progressdialog.cpp ..  
    155  
progressdia-  
    log2.cpp ..... 156  
property.cpp ..... 82  
ptrlist.cpp ..... 346  
ptrlistiter.cpp .. 348  
ptrstack.cpp ..... 358  
punktrotate.cpp .. 427  
puzzle.cpp ..... 599  
pythagoras.cpp ... 439  
radioadd.cpp ..... 580  
rahmenlayout.cpp . 219  
readerwriter.cpp . 633  
reaktest.cpp ..... 391  
regexpl.cpp ..... 526  
regexp2.cpp ..... 528  
regexvalidator.cpp .  
    530  
regexvalidator2.cpp  
    532  
rennen.cpp ..... 626  
resizeevent.cpp .. 556  
restaurant.cpp ... 642  
richtext.cpp ..... 109  
rotate.cpp ..... 424  
scale.cpp ..... 425  
schachbrett.cpp .. 433  
schachbrett2.cpp . 437  
schieb\_balk.cpp ... 21  
scrollbar.cpp ..... 169  
scrollview.cpp ... 168  
scrollview2.cpp .. 170  
sendevent.cpp ..... 567  
setting.cpp ..... 300  
setviewport.cpp .. 438  
shear.cpp ..... 425  
showhideclose.cpp ...  
    561  
signal.cpp ..... 581  
signalmapper.cpp . 579  
sinus.cpp ..... 434  
sinusfunc.cpp ..... 589  
smoothscale.cpp .. 488  
snapshot.cpp ..... 476



- snapshot2.cpp ..... 478  
 sound.cpp ..... 509  
 sound2.cpp ..... 510  
 spiro.cpp ..... 435  
 splitter.cpp ..... 192  
 splitter2.cpp ..... 194  
 stringlist.cpp ... 354  
 strlist.cpp ..... 356  
 tabdialog.cpp ..... 247  
 tabdialog2.cpp .. 249  
 table.cpp ..... 178  
 tableview ..... 172  
 tableview.h ..... 172  
 text\_groes.cpp .... 25  
 textsizeimpl.cpp . 764  
 textsizeimpl.h ... 764  
 textsizeimpl4.cpp ...  
     767  
 thread.cpp ..... 623  
 timeprogress.cpp . 393  
 togglebutton.cpp .. 90  
 tooltip.cpp ..... 140  
 tooltip2.cpp ..... 142  
 translate.cpp .... 424  
 treesize.cpp ..... 296  
 validator.cpp ..... 513  
 valuelist.cpp ..... 352  
 valuestack.cpp ... 360  
 waldbbrand.cpp .... 419  
 welchtag.cpp ..... 386  
 wellen.cpp ..... 418  
 widgetstack.cpp .. 195  
 widgetstack2.cpp . 196  
 wizard.cpp ..... 251  
 wizard2.cpp ..... 251  
 woist.cpp ..... 297  
 woistmaus.cpp .... 112  
 workspace.cpp .... 138  
 worktime.cpp ..... 395  
 wortstat.cpp ..... 331  
 xml1.cpp ..... 699  
 xmlplot.cpp ..... 708  
 xmlplot2.cpp ..... 726  
 xmlrichtext.cpp .. 712  
 xref.cpp ..... 342  
 zeigbild.cpp ..... 108  
 zeilzeich.cpp .... 272  
 zeilzeich2.cpp ... 278  
 zeitadd.cpp ..... 390  
 zwei\_buttons.cpp .. 18  
 zweirect.cpp ..... 311  
 Protokoll  
     FTP ..... 678  
     HTTP ..... 679, 683  
     nntp ..... 686
- Q**  
 QAction Klasse ..... 130  
 QApplication Klasse .. 18  
 QAsciiCache Klasse ... 338  
 QAsciiCacheIterator  
     Klasse ..... 338  
 QAsciiDict Klasse .... 333  
 QAsciiDictIterator  
     Klasse ..... 333  
 QBitArray Klasse .... 319  
 QBitmap Klasse ..... 475  
 QBitVal Klasse ..... 320  
 QBoxLayout Klasse .... 200  
 QBrush Klasse ..... 411  
 QBuffer Klasse ..... 272  
 QButtonGroup Klasse . 577  
 QButtonGroup Klasse .. 84,  
     189  
 QByteArray Klasse .... 319  
 QCache Klasse ..... 335  
 QCacheIterator Klasse ...  
     337  
 QCanvas Klasse ..... 454  
 QCanvasPolygonalItem  
     Klasse ..... 459  
 QCanvasEllipse Klasse ...  
     458  
 QCanvasItem Klasse ... 456  
 QCanvasItemList Klasse .  
     455  
 QCanvasLine Klasse ... 457  
 QCanvasPixmap Klasse 461  
 QCanvasPixmapArray  
     Klasse ..... 461  
 QCanvasPolygon Klasse ...  
     459  
 QCanvasRectangle Klasse  
     458  
 QCanvasSpline Klasse 460  
 QCanvasSprite Klasse 460  
 QCanvasText Klasse ... 460  
 QCanvasView Klasse ... 462  
 QChar Klasse ..... 29  
 QCheckBox Klasse ..... 83  
 QCheckTableItem Klasse .  
     178  
 QChildEvent Klasse ... 562  
 QClipboard Klasse .... 364  
 QCloseEvent Klasse ... 560  
 QColor Klasse ..... 37, 401  
 QColor-Objekte  
     (vordefiniert) ..... 38  
 QColorDialog Klasse . 227  
 QColorDrag Klasse .... 374  
 QColorGroup Klasse .... 39  
 QComboBox Klasse ..... 92  
 QComboTableItem Klasse .  
     178  
 QCString Klasse ..... 373  
 QCursor Klasse ..... 478  
 QCustomEvent Klasse . 568  
 QCustomMenuItem Klasse .  
     126  
 QDataBrowser Klasse . 612  
 QDataStream Klasse ... 278  
 QDataTable Klasse .... 609  
 QDateView Klasse ..... 615  
 QDate Klasse ..... 383  
 QDateEdit Klasse ..... 99  
 QDateTime Klasse ..... 393  
 QDateTimeEdit Klasse 100  
 qDebug Funktion ..... 651  
 QDial Klasse ..... 97  
 QDialog Klasse ..... 224  
 QDict Klasse ..... 326  
 QDictIterator Klasse 329  
 QDir Klasse ..... 290  
 QDns Klasse ..... 691  
 QDockArea Klasse .... 131  
 QDockWindow Klasse ... 131  
 QDomAttr Klasse ..... 719  
 QDomCDATASection Klasse  
     720  
 QDomCharacterData  
     Klasse ..... 719  
 QDomComment Klasse ... 720  
 QDomDocument Klasse . 717  
 QDomDocumentFragment  
     Klasse ..... 720  
 QDomDocumentType Klasse  
     720  
 QDomElement Klasse ... 718  
 QDomEntity Klasse .... 721  
 QDomEntityReference  
     Klasse ..... 721  
 QDomImplementation  
     Klasse ..... 721  
 QDomNamedNodeMap Klasse  
     722  
 QDomNode Klasse ..... 716  
 QDomNodeList Klasse . 723  
 QDomNotation Klasse . 721  
 QDomProcessingInstruction  
     Klasse ..... 721  
 QDomText Klasse ..... 720  
 QDoubleValidator Klasse  
     518  
 QDragEnterEvent Klasse .  
     562  
 QDragEnterEvent Klasse .  
     375

## Index

---

- QDragLeaveEvent Klasse . 563
- QDragMoveEvent Klasse ... 375, 562
- QDragObject Klasse ... 371
- QDropEvent Klasse 375, 563
- QErrorMessage Klasse 244
- QEuclJpCodec Klasse ... 649
- QEuclKrCodec Klasse ... 649
- QEvent Klasse ..... 542
- qFatal Funktion ..... 651
- QFile Klasse ..... 268
- QFileDialog Klasse ... 230
- QFileIconProvider Klasse ..... 233
- QFileInfo Klasse ..... 285
- QFileInfoList Klasse 293
- QFileInfoListIterator Klasse ..... 293
- QFilePreview Klasse . 233
- QFocusData Klasse ... 538
- QFocusEvent Klasse ... 554
- QFont Klasse ..... 79, 236
- QFontDialog Klasse ... 236
- QFontInfo Klasse ..... 314
- QFontMetrics Klasse . 314
- QFrame Klasse ..... 186
- QFtp Klasse ..... 678
- QGb18030Codec Klasse 649
- QGLColormap Klasse ... 728
- QGLContext Klasse .... 728
- QGLFormat Klasse .... 728
- QGLWidget Klasse .... 727
- QGrid Klasse ..... 197
- QGridLayout Klasse ... 209
- QGridView Klasse .... 170
- QGroupBox Klasse .... 188
- QHBoxLayout Klasse ..... 197
- QHBoxLayout Klasse ... 200
- QButtonGroup Klasse 189
- QHeader Klasse ..... 177
- QGroupBox Klasse .... 188
- QHideEvent Klasse .... 559
- QHostAddress Klasse . 690
- QHttp Klasse ..... 679
- QIconDrag Klasse . 371, 375
- QIconDragItem Klasse 375
- QIconSet Klasse ..... 314
- QIconView Klasse . 180, 375
- QIconViewItem Klasse 180
- QImage Klasse ..... 483
- QImageDrag Klasse .... 373
- QImageFormatType Klasse 487
- QImageIO Klasse ..... 488
- QInputDialog Klasse . 252
- qInstallMsgHandler Funktion ..... 652
- QIntCache Klasse ..... 338
- QIntCacheIterator Klasse ..... 338
- QIntDict Klasse ..... 333
- QIntDictIterator Klasse 333
- QIntValidator Klasse 515
- QIODevice Klasse .... 267
- QJisCodec Klasse .... 649
- QKeyEvent Klasse .... 550
- QKeySequence Klasse . 644
- QLabel Klasse ..... 106
- QLayoutItem Klasse .. 214, 217
- QLayoutIterator Klasse . 214
- QLCDNumber Klasse .... 107
- QLibrary Klasse ..... 750
- QLineEdit Klasse .... 113
- QListBox Klasse ..... 90
- QListView Klasse .... 157
- QListViewItem Klasse 159
- QListViewItemIterator Klasse ..... 160
- QLocalFs Klasse ..... 679
- qm2ts Tool ..... 647
- QMainWindow Klasse ... 129
- qmake Tool ..... 12
- qmake Tool ..... 5, 12
- QMap Klasse ..... 339
- QMapConstIterator Klasse ..... 341
- QMapIterator Klasse . 341
- QMemArray Klasse .... 314
- QMenuBar Klasse ..... 54
- QMenuData Klasse .. 54, 119
- QMessageBox Klasse ... 55, 239
- QMotifStyle Klasse .... 80
- QMouseEvent Klasse ... 545
- QMoveEvent Klasse .... 557
- QMovie Klasse ..... 499
- QMutex Klasse ..... 628
- QNetworkOperation Klasse ..... 667
- QNetworkProtocol Klasse 672
- QNetworkProtocol Klasse 676
- QObject Klasse ..... 19
- QObject Klasse ..... 24
- QObjectList Klasse ... 357
- QObjectListIterator Klasse ..... 357
- QPaintDeviceMetrics Klasse ..... 262
- QPainter Klasse ..... 401
- QPainter-Zustand Sichern ..... 444 Wiederherstellen .... 444
- QPaintEvent Klasse ... 544
- QPalette Klasse ..... 40
- QPen Klasse ..... 411
- QPicture Klasse ..... 494
- QPixmap Klasse ..... 472
- QPixmapCache Klasse . 474
- QPoint Klasse ..... 305
- QPointArray Klasse 322
- QPopupMenu Klasse .... 53
- QPrinter Klasse ..... 258
- QProcess Klasse ..... 745
- QProgressBar Klasse . 150
- QProgressDialog Klasse . 153, 258
- QPtrDict Klasse ..... 333
- QPtrDictIterator Klasse 333
- QPtrList Klasse ..... 343
- QPtrListIterator Klasse 346
- QPtrQueue Klasse .... 361
- QPtrStack Klasse .... 357
- QPtrVector Klasse . 319
- QPushButton Klasse ... 83
- QSqlForm Klasse ..... 612
- QRadioButton Klasse .. 83
- QRect Klasse ..... 308
- QRegExp Klasse ..... 521
- QRegExpValidator Klasse 529
- QRegion Klasse ..... 441
- QResizeEvent Klasse . 556
- QRgb Klasse ..... 37, 228
- QScrollBar Klasse .... 168
- QScrollView Klasse ... 55, 164
- QSemaphore Klasse .... 632
- QServerSocket Klasse 682
- QSettings Klasse ... 297
- QShowEvent Klasse .... 560
- QSignal Klasse ..... 581
- QSignalMapper Klasse 579
- QSize Klasse ..... 306
- QSjisCode Klasse .... 649
- QSlider Klasse ..... 21, 97
- QSocket Klasse ..... 679
- QSocketDevice Klasse 682
- QSocketNotifier Klasse . 687
- QSound Klasse ..... 508

- QSpinBox Klasse ..... 98  
 QSplitter Klasse ..... 190  
 QSqlCursor Klasse 606, 617  
 QSqlDatabase Klasse . 602  
 QSqlError Klasse ..... 602  
 QSqlIndex Klasse ..... 608  
 QSqlQuery Klasse . 605, 620  
 QSqlRecord Klasse .... 618  
 QStoredDrag Klasse ... 381  
 QStoredDrag Klasse ... 371  
 QStringList Klasse ..... 354  
 QString Klasse ..... 29, 643  
 QStringList Klasse ... 353  
 QStringList Klasse ..... 354  
 QStringListIterator Klasse  
     354  
 QStyle Klasse ..... 80  
 QSyntaxHighlighter  
     Klasse ..... 114  
 Qt-Assistant ..... 9  
 Qt-Designer ..... 753  
 Qt-Referenz ..... 7  
 QTabBar Klasse ..... 247  
 QTabDialog Klasse .... 245  
 QTable Klasse ..... 175  
 QTableWidgetItem Klasse .... 178  
 QTableSelection Klasse .  
     178  
 QTabWidget Klasse .... 247  
 QTextBrowser Klasse . 107  
 QTextCodec Klasse .... 649  
 QTextDrag Klasse ..... 372  
 QTextEdit Klasse ..... 113  
 QTextStream Klasse ... 274  
 QThread Klasse ..... 623  
 QTime Klasse ..... 388  
 QTimeEdit Klasse .... 100  
 QTimer Klasse ..... 396  
 QTimerEvent Klasse ... 553  
 QTimerEvent Klasse ... 398  
 QToolButton Klasse ... 133  
 QToolTip Klasse ..... 139  
 QToolTipGroup Klasse ....  
     130, 142  
 QTranslator Klasse ... 648  
 QTsciiCodec Klasse ... 649  
 QUriDrag Klasse ..... 381  
 QUrl Klasse ..... 664  
 QUrlInfo Klasse ..... 666  
 QUrlOperator Klasse . 657  
 QValidator Klasse .... 512  
 QValueList Klasse .... 349  
 QValueListConstIterator  
     Klasse ..... 351  
 QValueListIterator  
     Klasse ..... 351  
 QValueStack Klasse ... 359  
 QValueVector Klasse ...  
     319  
 QValueVector Klasse . 362  
 QVariant Klasse ..... 605  
 QVBox Klasse ..... 197  
 QVBoxLayout Klasse ... 201  
 QVButtonGroup Klasse 189  
 QVGroupBox Klasse .... 188  
 QWaitCondition Klasse ...  
     638  
 qWarning Funktion .... 651  
 QWhatsThis Klasse .... 143  
 QWheelEvent Klasse ... 547  
 QWidget Klasse ..... 19  
 QWidgetStack Klasse . 194  
 QWindowsStyle Klasse . 80  
 QWizard Klasse ..... 250  
 QWMMatrix Klasse ..... 429  
 QWorkspace Klasse .... 137  
 QDomAttributes Klasse ...  
     704  
 QDomContentHandler  
     Klasse ..... 701  
 QDomDeclHandler Klasse .  
     703  
 QDomDefaultHandler  
     Klasse ..... 704  
 QDomDTDHandler Klasse ...  
     703  
 QDomEntityResolver  
     Klasse ..... 702  
 QDomErrorHandler Klasse  
     701  
 QDomInputSource Klasse .  
     705  
 QDomLexicalHandler  
     Klasse ..... 704  
 QDomLocator Klasse ... 705  
 QDomNamespaceSupport  
     Klasse ..... 707  
 QDomParseException  
     Klasse ..... 702  
 QDomReader Klasse .... 706  
 QDomSimpleReader Klasse  
     707  
**R**  
 Rasterung (Graphik) .... 450  
 Referenz ..... 7  
 Reguläre Ausdrücke .... 521  
 Resize-Events ..... 556  
 RGB-Farbmodell ..... 37  
 Richtext ..... 107  
**S**  
 SAX2-Schnittstelle ..... 697  
 Schiebebalken ..... 21, 97  
 Schließ-Events ..... 560  
 Screenshots ..... 475  
 Scrollviews ..... 164  
 Semaphore ..... 632  
 setGeometry() ..... 79  
 setMaximumHeight() . 79  
 setMaximumSize() .... 79  
 setMinimumHeight() . 79  
 setMinimumSize() .... 79  
 setMinimumWidth() ... 79  
 setPalette() ..... 79  
 setProperty() ..... 81  
 setStyle() ..... 80  
 shallow copy-Sharing ... 312  
 Shared Library ..... 750  
 Show-Events ..... 559  
 Sichtbarkeits-Events .... 559  
 Signal-Slot-Konzept .. 21, 28,  
     573  
 sizeHint() ..... 79  
 Sound ..... 508  
 Spinboxen ..... 98  
 Splitter ..... 190  
 Statuszeile ..... 144  
 Strings in Qt ..... 29  
 Subwidget-Events ..... 562  
 Syntaxhighlighting ..... 114  
 Synthetische Events ..... 566  
**T**  
 Tabdialog ..... 245  
 Tabelle  
     Einfach ..... 170  
     Kalkulation ..... 175  
     Komfortabel ..... 175  
     Kopf ..... 177  
     Spreadsheet ..... 175  
 Tabelle von Bildern .... 592  
 Tastatur-Events ..... 550  
 Tastaturfokus ..... 533  
 Testausgaben ..... 651  
 Textanzeige  
     HTML ..... 107  
     Komfortabel ..... 107  
     Richtext ..... 107  
 Textbrowser ..... 107  
 Texteingabe ..... 113  
 Threads ..... 623  
 Tiefe Kopie ..... 312  
 Timer ..... 396  
 Timer-Events ..... 398, 553  
 Tool  
     assistant ..... 9  
     linguist ..... 646  
     lrelease ..... 647  
     lupdate ..... 645

## Index

---

- make ..... 12
- qm2ts ..... 647
- qmake ..... 5, 12
- Qt-Designer ..... 753
- uic ..... 759
- Tooltips ..... 139, 142
- Transformation (Graphik) ...  
423
- U**
- uic Compiler ..... 759
- Unicode ..... 643
- URI-Referenzen ..... 381
- V**
- Validierung von Eingaben ..  
512
- Versteck-Events ..... 559
- View-Transformation  
(Graphik) ..... 433
- Viewport-Transformation ...  
436
- Vordefinierte Datentypen ...  
303
- Vordefinierte Dialogfenster .  
223
- Vordefinierte  
Datenstrukturen .... 303
- W**
- warning() ..... 55
- Werkzeugleiste ..... 133
- Widget  
Begriff ..... 6
- Widgetstack ..... 194
- Wizards ..... 250
- World-Transformation .. 423
- X**
- XML  
DOM ..... 713  
SAX2 ..... 697
- XML-Parser ..... 697
- Z**
- Zeichenketten in Qt ..... 29
- Zeit ..... 388
- Zeit und Datum ..... 393
- Zeitschaltuhren ..... 396
- Zuordnung von Widgets 186