# HANSER



## Leseprobe

zu

## Einführung in Qt

## von Achim Lingott

Print-ISBN: 978-3-446-46691-3 E-Book-ISBN: 978-3-446-46903-7 E-Pub-ISBN: 978-3-446-46995-2

Weitere Informationen und Bestellungen unter https://www.hanser-kundencenter.de/fachbuch/artikel/978-3-446-46691-3 sowie im Buchhandel

© Carl Hanser Verlag, München

## Inhalt

Vor	wort	IX
Dan	ksagung	Х
1	Einführung und erste Schritte	1
1.1	Download und Installation	3
1.2	Die Qt-Bibliothek1.2.1Die Klassen der Bibliothek1.2.2Die Module der Bibliothek	5 5 6
1.3	Die installierten Programme	8
1.4	Im Qt Creator erstellte Dateien und ihre Bedeutung1.4.1Dateien .pro und .pri1.4.2Datei CMakeLists.txt1.4.3Datei .ui1.4.4Datei ui_mainwindow.h1.4.5Datei main.cpp	10 10 12 13 14 16
1.5	Der Qt Designer	17
1.6	Das Signal-Slot-Prinzip	18
1.7	Ot in Microsoft Visual Studio	19
2	Das Erstellen von Qt-Widgets-Applikationen	25
2.1	Unterschiedliche Oberklassen	25
2.2	Eine Auswahl von Widgets	27
2.3	Qt-Widgets-Anwendung mit dem Qt Designer2.3.1Signal- und Slot-Funktionen miteinander verbinden	29 34
2.4	Erstellen ohne Qt Designer	39
2.5	Die Benutzung von Layouts	41

2.6	Das Erstellen und die Funktion von Menüs	43
2.7	Ein Beispiel mit QTabWidget	46
2.8	Ein zweites Formular hinzufügen	48
2.9	Maus- und Tastatur-Events	51
2.10	Shortcuts für die Bedienung des Qt Creators	53
2.11	Von den Programmen auszugebende Meldungen	55
3	Daten, Variablen und ihre Benutzung in Qt	57
3.1	Qt-Datentypen	57
3.2	Das Model-View-Prinzip und seine Realisierung	59
3.3	Das Qt-Event-System	62
3.4	Qt-Containerklassen	63
3.5	Die Speicherverwaltung in Qt	65
3.6	Multithreading in Qt	66
3.7	Die Klasse QVariant	73
4	Zeichnen in Widget-Applikationen	77
4.1	Grundlagen des Zeichnens	77
4.2	Zeichnen auf ein Fenster	78
4.3	Zeichnen auf ein Widget	81
4.4	Freie Zeichnungen mit der Maus auf ein Fenster	88
4.5	Transformationen	93
4.6	Farbverläufe bei Füllungen	96
4.7	Zeichnen mit QGraphicsView auf QGraphicsScene	98
5	Ressourcen in Qt	101
5.1	Das Erstellen einer Ressourcendatei	101
5.2	Die Benutzung von StyleSheets	106
5.3	Die Verwendung von RTF und HTML	113
5.4	Lokalisierung von Qt-Applikationen	117
5.5	Dynamischer Wechsel der Sprache	122
6	Datenbankanbindung an Qt-Applikationen	125
6.1	SQL und Datenbankdesign ganz kurz	125
6.2	Eine Datenbankanwendung	127

6.3	Verbindung zu einem MySQL-Datenbankserver	132
6.4	Verbindung zu einem Microsoft SQL Server	136
6.5	Verbindung zu einer MS Access-Datenbank	137
6.6	Lesen von Daten aus einer Tabelle	138
6.7	Einen neuen Datensatz eintragen	140
6.8	Die Anwendung des Model-View-Prinzips	142
7	Drucken und Dateibearbeitung	149
7.1	Grundlagen des Druckvorgangs	149
7.2	Ausdrucken von Text und Bildern	150
7.3	Dateien lesen und schreiben	158
7.4	XML-Dateien lesen und schreiben	161
7.5	JSON-Dateien lesen und schreiben	167
8	Qt Quick und QML	171
8.1	Das Erstellen einer Qt-Quick-Anwendung	172
8.2	Multimedia mit Qt Quick	176
8.3	Der Quick Designer	177
8.4	QML-Basistypen	178
8.5	JavaScript in QML	179
8.6	Canvas	184
8.7	Lokalisierung von Quick-Applikationen	188
9	Besonderes in Quick-Applikationen	191
9.1	Animationen mit QML	191
9.2	Zustände und ihre Übergänge	199
9.3	Das Zustandsdiagramm und der Zustandseditor	203
10	QML-Applikationen als GUI für C++-Klassen	209
10.1	QML-Datentypen	209
10.2	Signale von und zu QML-Oberflächen	210
10.3	Drucken über eine QML-Oberfläche	210
10.4	Datenaustausch QML-GUI und C++-Klasse	215
	<ul><li>10.4.1 Datenbankanbindung über QML-Oberfläche</li><li>10.4.2 Einlesen einer XML-Datei über eine QML-GUI</li></ul>	218 223

11	Verschiedenes	227
11.1	Einige ausgewählte Qt Widgets	227
11.2	Einige ausgewählte QML-Typen	235
11.3	Exceptions in Qt	239
11.4	Debugging von Qt-Anwendungen	240
11.5	Debugging von QML-Anwendungen	244
11.6	Dokumentation mit Doxygen	244
11.7	Dokumentation mit Qt	248
11.8	Deployment von Qt unter Windows	248
11.9	Ot auf anderen Betriebssystemen	250
11.10	) Qt für Android	252
Lite	ratur	253
Inde	xx	255

## Vorwort

Dieses Buch vermittelt Einsteigern mit C++-Vorkenntnissen die Grundlagen der Qt-Programmierung. Mit der hervorragenden Qt-Bibliothek lassen sich grafische User Interfaces für die unterschiedlichsten Anwendungsfälle programmieren.

Zum Zeitpunkt der Manuskripterstellung war die Version Otó gerade veröffentlicht und wird im Buch prinzipiell als Grundlage für die Beispiele benutzt. Es sind aber verschiedene Module aus Ot5 noch nicht in dieser neuen Version enthalten, weshalb einige Beispiele auf der LTS-(Long Time Support-)Version 5.15 aufbauen. Falls in den kommenden Monaten Ergänzungen zu Otó hinzukommen, werden diese immer sofort auf der Plus.Hanser-Webseite veröffentlicht.

#### ı گ

#### Ihr Plus – digitale Zusatzinhalte!

Auf unserem Download-Portal finden Sie zu diesem Titel kostenloses Zusatzmaterial.

Geben Sie auf plus.hanser-fachbuch.de einfach diesen Code ein:

Auf dieser Webseite sind alle hier im Buch verwendeten Quelltexte zu finden, teilweise auch etwas umfangreicher als im Buch gezeigt. Diese Quelltexte enthalten auch Zeilen zum Löschen der in einzelnen Funktionen evtl. erstellten Zeiger. Diese delete()-Angaben sind in den Ausschnitten der Quelltexte im Buch in der Regel nicht enthalten.

Außerdem finden Sie dort weitere Beispiele, deren Beschreibung den Umfang des Buches sprengen würde. Nachfolgend finden Sie eine Auswahl:

#### Texteditor

Er liest Dateien ein, kann sie verändern und Text ausdrucken. Außerdem kann der geschriebene oder gelesene Text als PDF-Datei abgespeichert werden.

#### Mediaplayer

Er bietet Ihnen alle wichtigen Funktionen an. Insbesondere geht es um eigene Aufzeichnungen und das Abspielen vorhandener Dateien. Verkehrsampel

Natürlich wird nicht nur das Aussehen der Ampel erstellt, sondern auch ihre Steuerung, sowohl automatisch als auch von Hand.

- Dashboard f
  ür ein Fahrzeug
   Es entsteht ein modernes Dashbord f
  ür ein Auto.
- Android-Applikationen

Hier wird die Installation von Qt für Android gezeigt sowie die Erstellung eigener Applikationen.

Touchscreen-Applikationen
 Sie sind insbesondere f
ür die Anwendung unter Android gedacht.

#### Netzwerk-Applikationen

Hier werden Beispiele für die Benutzung von HTML sowie von HTTP und anderen Netzwerkprotokollen gezeigt.

## Danksagung

An dieser Stelle möchte ich allen Beteiligten einen großen Dank aussprechen. Er gilt allen beteiligten Mitarbeitern des Carl Hanser Verlages, insbesondere Frau Sylvia Hasselbach, die sich stets allen meinen Fragen zu Inhalt und Gestaltung gestellt hat und hilfreich mit Ideen zur Seite stand.

Diesen Dank möchte ich aber auch an diejenigen richten, die, vermutlich ohne es zu ahnen, zur einen oder anderen Idee beigetragen haben: Das sind die Teilnehmer meiner Seminare, deren Fragen mich dazu anregten, das eine oder andere Problem etwas umfassender oder überhaupt anzugehen.

Und nun viel Spaß beim Lesen und bei der Arbeit mit Qt! Achim Lingott

## Das Erstellen von Qt-Widgets-Applikationen

Der Hauptgrund für Ihre Benutzung von Qt ist sicher, dass Sie GUIs für alle möglichen Programme erstellen wollen. Das ist sowohl mit dem klassischen Qt auf der Basis von C++-Programmierung als auch mit der später hinzugefügten Variante Qt Quick auf der Basis von QML (Qt Modeling Language oder auch Qt Meta Language genannt) möglich. Dazu mehr in Kapitel 8.

Beide Varianten haben sowohl Vor- als auch Nachteile. In diesem Kapitel soll erst einmal die Programmierung mit Qt gezeigt werden.

## 2.1 Unterschiedliche Oberklassen

Für die GUI-Erstellung stellt die Qt-Bibliothek drei Oberklassen zur Verfügung, QMainWindow, QWidget und QDialog.

Mit QMainWindow erzeugen Sie ein Formular, das als Hauptfenster (Formular) für eine Anwendung verwendet werden kann. Auf diesem Hauptfenster können Sie Menüs erzeugen und die Statuszeile nutzen.

Formulare mit QDialog werden oft als Zweitfenster benutzt, die sich z.B. nach einem Buttonklick oder nach dem Klick auf einen Menüpunkt öffnen (z.B. zur Anzeige eines Verzeichnisinhaltes oder für Hilfsprozesse).

QWidget ist die Basisklasse aller User-Interface-Objekte, wie z.B. QFrame, QProgressBar, QTabWidget usw.

Dieser Teil der Bibliothek sieht so aus:



Bild 2.1 Ausschnitt aus der Qt-Bibliothek

Wie Sie sehen, ist auch die Klasse QDialog von QWidget abgeleitet, somit stehen Ihnen auch alle Objekte zur Verfügung und Sie können z.B. mit dem Qt Designer auch Formulare mit der Oberklasse QDialog bearbeiten.

Die Erstellung eines Formulars mit der Oberklasse QMainWindow haben Sie bereits in Kapitel 1 gesehen. Falls Sie ein Formular mit der Oberklasse QDialog erzeugen wollen, gehen Sie ähnlich vor. Im entsprechenden Fenster des Qt Creators wählen Sie als Basisklasse jetzt aber QDialog aus. Die vorgegebenen Dateinamen ändern sich entsprechend.

÷	📧 Qt Widgets-Anwend	ung	
	Pfad	Klasseninf	ormationen
	Build-System Details		Geben Sie Informationen bezüglich der Klassen ein, für die Sie Quelltexte generieren wollen.
	Übersetzung Kits	Klassenname:	Dialog
	Zusammenfassung	Basisklasse:	QDialog v
		Header-Datei:	dialog.h
		Quelldatei:	dialog.cpp
		Form-Datei:	✓ Form-Datei erzeugen dialog.ui
			Weiter Abbrechen

Bild 2.2 Die Auswahl der Oberklasse und Namen für Dateien

Es entsteht ein Formular ohne Menü- und Statusbar, aber auch ohne die Möglichkeiten der Größenänderung in der rechten oberen Ecke. Diese Buttons sind nicht vorhanden.

III Dialog	×

Bild 2.3 Ein mit der Oberklasse QDialog erstelltes Formular

## ■ 2.2 Eine Auswahl von Widgets

Eine Liste der zu verwendenden Widgets oder Steuerelemente wird im Designmodus des Ot Creators links angezeigt. Darunter befinden sich die klassischen Elemente wie Push Button, Radio Button, Check Box, aber auch solche wie Line Edit, Spin Box oder Calendar Widget. Ebenso gehören die Layouts und die Spacers hinzu.

Hier die Beschreibung einer kleinen Auswahl.

#### Layouts

Das Layout-System bietet einen einfachen Weg, Widgets auf dem Formular optisch anzuordnen und zu positionieren. Für die einzelnen Layouts existieren in der Bibliothek Layout-Klassen, wie QVBoxLayout, QHBoxLayout, QGridLayout und andere. Über die Funktion addWidget() dieser Klassen können Sie Widgets diesen Layouts hinzufügen. Alle Klassen und Beispiele dazu finden Sie unter:

https://doc.qt.io/qt-6/layout.html

#### Spacers

Den Layouts können Sie auch über die Funktion addSpacerItem() Instanzen der Klasse QSpacerItem hinzufügen und damit Abstände in den Layouts bestimmen. Sowohl die Anbringung von Layouts als von Spacern lässt sich natürlich mit dem Qt Designer erledigen.

#### **Push Button**

Es steht eine Reihe von Funktionen zur Verfügung, die aufgrund der Ableitungshierarchie aus verschiedenen Oberklassen stammen können, z.B. die Funktion pressed() aus QAbstractButton, die Funktion addAction() aus QWidget und die Funktion killTimer() aus QObject.

#### **Check Box**

Zugrunde liegt die Klasse QCheckBox. Eine Instanz dieser Klasse erzeugt eine Checkbox mit einem Label zur Beschriftung. Mit einer Funktion isChecked() lässt sich der augenblickliche Status der Checkbox ermitteln (Rückgabedatentyp ist *bool*), oder mit der Funktion setChecked() können Sie den Status auch bestimmen. Ebenso gibt es eine Signal-Funktion, die immer dann ein Signal emittiert, wenn sich der Status ändert.

#### Line Edit

Eine Instanz der Klasse QLineEdit erstellt ein einzeiliges Textfeld. Mit setPlaceholder Text() können Sie einen Text hineinschreiben, der dem Benutzer mitteilt, was als Inhalt erwartet wird. Sollen bei diesem vom Benutzer einzutragenden Text nicht alle Zeichen zugelassen werden, z.B. weil nur Ziffern eingetragen werden sollen, dann ergänzen Sie den Konstruktor Ihrer Klasse MainWindow mit folgendem Quelltext:

```
QRegularExpression rx("[0-9]*");
QValidator *validator = new QRegularExpressionValidator(rx, this);
ui->lineEdit->setValidator(validator);
```

Natürlich müssen zur einwandfreien Funktion noch die betreffenden Klassen inkludiert werden. Jetzt lassen sich nur Ziffern in das Feld eintragen. Dem Konstruktor von QRegularExpression können Sie einen beliebigen gültigen regulären Ausdruck übergeben. Hier folgt ein ganz kleiner Ausschnitt der verfügbaren regulären Ausdrücke:

Zeichenfolge	Erklärung
*	Der davorstehende Ausdruck darf beliebig oft vorkommen (auch 0-mal).
+	Der davorstehende Ausdruck muss mindestens einmal vorkommen (aber auch beliebig oft).
?	Der davorstehende Ausdruck kann einmal vorkommen, braucht aber auch nicht (er ist optional).
{n}	Der davorstehende Ausdruck muss genau n-mal vorkommen.
{n,}	Der davorstehende Ausdruck muss mindestens n-mal vorkommen (bis beliebig oft).
{n,m}	Der davorstehende Ausdruck muss mindestens n-mal vorkommen, darf bis m-mal vorkommen.
[0-9]	Genau ein Zeichen, das eine Ziffer sein muss
[aBx@§]	Genau ein Zeichen der in der Klammer enthaltenen
[A-Za-z0-9]	Ein Zeichen, das Großbuchstabe, Kleinbuchstabe oder Ziffer sein kann
[^x]	Ein beliebiges Zeichen außer x

Tabelle 2.1	Eine Auswahl de	r Sonderzeichen	in regulären	Ausdrücken
-------------	-----------------	-----------------	--------------	------------

#### Slider

Zugrunde liegt die Klasse QSlider. Mit der Funktion setOrientation() lässt sich einstellen, ob es sich um einen horizontalen oder vertikalen Slider handeln soll.

#### **List View**

Die Klasse QListView kann zum Erstellen einer linearen Liste oder einer Liste von Icons benutzt werden.

#### **Table View**

Es wird eine Tabelle dargestellt, die Daten von einem Modell übernimmt. Diese Klasse gehört zum sog. Model-View-Framework von Qt. Weitere Informationen dazu finden Sie unter:

https://doc.qt.io/qt-6/model-view-programming.html

#### **Group Box**

Mithilfe der Klasse QGroupBox können Widgets optisch zusammenhängend mit einem gemeinsamen Titel angeordnet werden.

#### Tab Widget

Mithilfe der Klasse QTabWidget können einzelne Widgets in Form von einzelnen Tabs angeordnet werden. Jedem dieser Tabs kann eine eigene Klasse mit eigenen Funktionen zugeordnet werden.

Ein Beispiel zum Tab Widget finden Sie in Kapitel 2.7.

## 2.3 Qt-Widgets-Anwendung mit dem Qt Designer

Lassen Sie uns hier beispielhaft eine Anwendung erstellen, die Test heißen soll. Der erste Schritt ist die Erstellung eines neuen Projekts. Das können Sie auf der Startseite unter *Projekte* oder im Hauptmenü unter dem Punkt *Datei* tun. Sie erhalten ein Fenster, in dem Sie *Qt-Widgets-Anwendung* auswählen. Auf der Folgeseite geben Sie dem Projekt einen Namen und wählen einen Projektordner aus. Als Build-System ist *qmake* zu wählen.

Reue Datei oder neues Projekt - Qt Creator X				
Vorlage: Alle Vorlagen V				
Projekte Anwendung (Qt)	Qt Widgets-Anwendung	Erstellt eine Qt-Anwendung für den Desktop mit einem Qt Designer-basierten Hauptfenster.		
Anwendung (Qt Quick) Anwendung (Qt for Python) Bibliothek Anderes Projekt	Qt Konsolenanwendung	Wählt eine für Desktop-Entwicklung geeignete Qt- Version aus, sofern sie verfügbar ist. Unterstützte Plattformen: •Desktop		

Bild 2.4 Projektauswahl im Qt Creator

Am Anfang ist es völlig ausreichend, die vom Qt Creator erfolgten Vorgaben für Klassenund Dateinamen zu benutzen. Als Basisklasse wird auch die vorgegebene Klasse QMainWindow benutzt. Die Checkbox zum Erzeugen der Form-Datei bleibt aktiviert.

				Х
÷	📧 Qt Widgets-Anwend	ung		
	Pfad Build-System	Klasseninf	Ormationen Geben Sie Informationen bezüglich der Klassen ein, für die Sie Quelltexte generieren wollen.	
	Details Übersetzung Kits Zusammenfassung	Klassenname: Basisklasse:	MainWindow QMainWindow	~
		Header-Datei: Quelldatei:	mainwindow.h mainwindow.cpp	
		Form-Datei:	mainwindow.ui	
			Weiter Abbrech	en

Bild 2.5 Namensvergabe im Qt Creator

Im Fenster *Kit-Auswahl* wählen Sie den gewünschten Compiler und damit auch den dazugehörigen Teil der Bibliothek. Ich empfehle Ihnen MinGw. Danach werden noch einmal alle zu erstellenden Dateien angezeigt.

				×
÷	🗈 Qt Widgets-Anwen	dung		
	Pfad Build-System Details Übersetzung Mts Zusammenfassung	<ul> <li>Kit-Auswahl</li> <li>Für das Projekt Test können die folgenden Kits verwendet werden:</li> <li>Kits nach Namen filtern</li> <li>Alle Kits auswählen</li> <li>Desktop (x86-windows-msys-pe-32bit)</li> <li>Desktop (x86-windows-msys-pe-64bit)</li> <li>Desktop Qt 6.0.0 MSVC2019 64bit</li> <li>Desktop Qt 6.0.0 MinGW 64-bit</li> <li>Ersatz für "Desktop Qt 5.15.0 MSVC2015 64bit"</li> <li>Verwalten</li> </ul>	Details   Detail	<b>^</b>
		Weiter	Abbrech	en

Bild 2.6 Auswahl eines Compilers und einer Qt-Bibliothek

Der Qt Creator erstellt die für dieses Projekt grundlegenden Dateien.



**Bild 2.7** Projektexplorer im Qt Creator

Die Inhalte der Dateien *Test.pro, main.cpp* und *mainwindow.ui* wurden bereits in Kapitel 1 erklärt.

Hier sehen Sie die beiden übrigen Dateien:

```
Listing 2.1 Datei mainwindow.h des Projekts "Test"
```

```
#ifndef MAINWINDOW H
    #define MAINWINDOW H
04
    #include <QMainWindow>
05
    #include <QDebug>
06
    QT BEGIN NAMESPACE
80
    namespace Ui { class MainWindow; }
09
    QT END NAMESPACE
    class MainWindow : public QMainWindow
    {
        Q_OBJECT
14
15
    public:
16
        MainWindow(QWidget *parent = nullptr);
        ~MainWindow();
18
19
   private slots:
        void buttonClick();
   private:
        Ui::MainWindow *ui;
24
   };
25 #endif // MAINWINDOW_H
```

In dieser Datei wird in Zeile 5 eine Klasse QDebug inkludiert. Das wird zur Zeit der Programmierung gern gemacht, da dann vom Programmierer für Testausgaben eine Funktion qDebug() aufgerufen werden kann. Dieses #include <QDebug> und der Funktionsaufruf sollten aber nach Fertigstellung des Programms wieder entfernt werden.

In der Zeile 20 steht der Prototyp einer privaten Slot-Funktion buttonClick(). Sie ist in der Datei *mainwindow.cpp* in den Zeilen 11 bis 14 implementiert.

```
Listing 2.2 Datei mainwindow.cpp des Projekts Test
    #include "mainwindow.h"
   #include "ui mainwindow.h"
04 MainWindow::MainWindow(QWidget *parent)
         : QMainWindow(parent)
06
         , ui(new Ui::MainWindow)
    {
80
        ui->setupUi(this);
09
    }
11 void MainWindow::buttonClick()
12
    {
        qDebug() << "buttonClick() ...";</pre>
14
    }
15
16
    MainWindow::~MainWindow()
    {
18
        delete ui;
19
   }
```

Zeile 2 zeigt das Inkludieren einer Datei *ui\_mainwindow.h.* Diese Datei entsteht beim ersten Kompilieren des Programms. Ihr Inhalt wurde bereits in Kapitel 1 gezeigt.

Die im Konstruktor aufgerufene Funktion setupUi() wurde in dieser Datei angelegt, und ihr Inhalt wird vom Qt Designer bestimmt.

Die Slot-Funktion buttonClick() soll später dazu dienen, durch Klick auf einen Button eine Funktionalität auszulösen. Mit ihrer Hilfe kann das Signal-Slot-Prinzip (wie in Kapitel 1 beschrieben) realisiert werden. Die Implementierung dieser Funktion ist in *mainwindow. cpp* in den Zeilen 11 bis 14 zu finden. Hier wird qDebug() verwendet, um testweise eine Ausgabe im Qt Creator zu erzeugen.

Um Ihr Projekt mit einem Button zu versehen und diesen einfach auf dem Formular zu platzieren, verwenden Sie den Qt Designer. Er ist in den Qt Creator integriert, und Sie erreichen ihn durch Doppelklick auf die Datei *mainwindow.ui* im Projektexplorer des Qt Creators.



Bild 2.8 Benutzung des Qt Designers - Zeigername

Aus dem linken Werkzeugkasten wählen Sie unter der Überschrift *Buttons* das Element *Push Button* aus und ziehen es mit gedrückter linker Maustaste auf das Formular an die gewünschte Stelle. Die Größe des Formulars können Sie ebenfalls über den Designer verändern.

Bei markiertem *PushButton* können Sie rechts im Eigenschaftsfenster den Namen des vom Qt Creator verwendeten Zeigers auf die Klasse QPushButton sehen. Ändern Sie den Namen so, dass er später im Quelltext jederzeit eindeutig identifiziert werden kann. (Das ist z.B. wichtig, wenn Sie mehrere Buttons verwenden.)

Mit diesem Eigenschaftsfenster können Sie die Eigenschaften der entsprechenden Widgets verändern.

Über dem Eigenschaftsfenster wird angezeigt, dass ein Zeiger mit dem Namen *pushButton\_ok* ein Zeiger auf die Klasse QPushButton ist und vermutlich die Adresse einer Instanz von QPushButton enthält.

Wenn Sie ein Widget auf dem Formular markieren und dann die **F1**-Taste drücken, erscheint eine Information zur zuständigen Klasse aus der Klassenbibliothek.



Bild 2.9 Benutzung des Qt Designers - Textname

#### 2.3.1 Signal- und Slot-Funktionen miteinander verbinden

Eine wichtige Aufgabe ist nun, die Verbindung zwischen einer Signal-Funktion der Klasse QPushButton und der von Ihnen selbst geschriebenen Slot-Funktion buttonClick() herzustellen. (Man könnte auch davon sprechen, beide zu verdrahten.)

Wie lässt sich herausfinden, welche Signal- oder Slot-Funktionen bereits in der jeweiligen Klasse zur Verfügung stehen, also nicht mehr selbst programmiert werden müssen?

- Suchen Sie in der Klassenbibliothek die zum Widget gehörende Klasse (z. B. QPushButton).
- Wenn Sie auf der Seite nach unten scrollen, finden Sie eine Überschrift *Public Slots*. Dort steht aber keine für uns brauchbare Funktion. Diese Slot-Funktion könnte aber auch in einer Klasse stehen, von der QPushButton abgeleitet ist. (Hier ist die Klasse QAbstractButton genannt.)
- Sehr weit oben gibt es einen Link List of all members, including inherited members.
- In dieser Liste befindet sich eine Funktion clicked(). Der Name ist als Link ausgeführt. Ein Klick darauf zeigt, dass es sich um eine Signal-Funktion handelt. Damit haben wir also eine passende Funktion gefunden, die uns auch in der Klasse QPushButton zur Verfügung steht.

Diese Verbindung zwischen Signal- und Slot-Funktionen können Sie mit dem Qt Designer herstellen. Wählen Sie aus dem Menü über der Fensterdarstellung den gezeigten Button (Signals und Slots bearbeiten) aus oder betätigen Sie **F4** (Bild 2.10). Der linke Werkzeugkasten mit Buttons, Items etc. ist jetzt nicht mehr zu verwenden. Dafür werden die auf dem Fenster verwendeten Widgets (wie z.B. unser Button) rot gefärbt, wenn Sie mit dem Mauszeiger darüber kommen.



Bild 2.10 Qt Designer - Einschalten der Bearbeitung von Signalen und Slots

Klicken Sie jetzt auf den Button, halten Sie die Maustaste gedrückt und ziehen den Mauszeiger in das freie Fenster. Lassen Sie die Taste los, und es ist Bild 2.11 zu sehen. Außerdem entsteht ein Fenster nach Bild 2.12. Links in diesem Fenster sehen Sie die über QPushButton zu erreichenden Signal-Funktionen. Wenn Sie links clicked() markieren (die aus QAbstractButton stammende Signal-Funktion), ist das rechte Fenster leer. Es müsste die parametergleichen Slot-Funktionen aus QMainWindow enthalten. Es gibt keine. Eine passende Slot-Funktion müssen wir in unserer Klasse MainWindow (die ja von QMainWindow abgeleitet ist) selbst schreiben (siehe Zeilen 19 und 20 der Datei mainwindow.h).



Bild 2.11 Ot Designer - ein Slot wird in der Fenster-Klasse gesucht.

Klicken Sie im Fenster (siehe Bild 2.12) auf *Ändern*. Es erscheint ein weiteres Fenster (siehe Bild 2.13). Wir wollen die selbst programmierte Slot-Funktion eintragen. Klicken Sie also in der oberen Fensterhälfte auf + und tragen Sie in die neu erscheinende Zeile des oberen Fensters den Namen der Slot-Funktion ein (hier buttonClick()). Nach Betätigen des Buttons *OK* gelangen Sie zum vorigen Fenster (siehe Bild 2.12) zurück, allerdings ist jetzt im rechten Teil bei MainWindow nun buttonClick() eingetragen. Nach Klick auf den *OK*-Button

erhalten Sie wieder die Ansicht Ihres Fensters im Qt Designer, allerdings jetzt mit eingetragener Verbindung zwischen Signal- und Slot-Funktion (Bild 2.14).

📧 Verbindung bearbeiten - Qt Creator		×
pushButton_ok (QPushButton) clicked() clicked(bool) pressed() released() toggled(bool)	MainWindow (QMainWindow)	
Ändern	Ändern OK Abbreche	n

Bild 2.12 Qt Designer - keine passende Slot-Funktion vorhanden

I Signale/Slots von MainWindow - Qt Creator	
Slots	
showMinimized() showMaximized() showFullScreen()	^
showNormal() close() raise0	
lower() setAnimated(bool)	
setDockNestingEnabled(bool) seconneamaenmanolBarOnMac(bool) buttonClickN	
	¥
Signale	
destroyed(QObject*) destroyed()	
objectNameChanged(QString) windowTitleChanged(QString)	
windowlconChanged(Qlcon)	
customContextMenuRequested(QPoint)	
iconSizeChanged(QSize)	
tabifiedDockWidgetActivated(QDockWidget*)	
	OK Abbrechen

Bild 2.13 Ot Designer – Eintragen der selbst programmierten Slot-Funktion



Bild 2.14 Anzeigen der Verbindung zwischen Signal- und Slot-Funktionen

Wählen Sie, wie hier beschrieben, die Verbindungsherstellung zwischen Signal und Slot mithilfe des Ot Designers, wird die dafür nötige connect()-Funktion in der Datei *ui\_mainwindow.h* erzeugt.

Sie können aber auch ohne Designer arbeiten und die Funktion direkt in den Konstruktor der Klasse MainWindow in die Datei *mainwindow.cpp* schreiben. Das würde dann folgendermaßen aussehen:

```
01 MainWindow::MainWindow(QWidget *parent)
02 : QMainWindow(parent)
03 , ui(new Ui::MainWindow)
04 {
05 ui->setupUi(this);
06 connect(ui->pushButton_ok, SIGNAL(clicked()), this, SLOT(buttonClick()));
07 }
```

Die Funktion connect() erwartet fünf Parameter.

- Zuerst einen Zeiger auf die Klasse mit der Signalfunktion. Dieser Zeiger ui wird in Zeile 3 mit einer Instanz der Klasse Ui::MainWindow initialisiert, hier mit dem Teil der Klasse MainWindow, der vom Qt Designer in die Datei ui\_mainwindow.h geschrieben wurde. Über diesen Zeiger ist die Instanz von QPushButton erreichbar (wir haben sie pushButton\_ok genannt).
- Der zweite Parameter nennt die Signal-Funktion, hier clicked().
- Der dritte Parameter ist ein Zeiger auf die Klasse mit der Slot-Funktion. Das ist hier die Klasse MainWindow, in der sich der Konstruktor mit der connect()-Funktion befindet, also this.
- Der vierte Parameter nennt die Slot-Funktion der Klasse.
- Ein fünfter Parameter könnte den Connection-Typ nennen. Er besitzt einen Defaultwert. Die einzelnen Typen wurden bereits in Kapitel 1 genannt.

Die Syntax der connect-Funktion besitzt noch eine neuere Variante. Mit dieser Syntax könnten Sie z.B. ein Signal auch mit einer C++-Lambda-Funktion als Slot verbinden. Sie sieht so aus:

connect(ui->pushButton ok, &QPushButton::clicked, this, MainWindow::buttonClick);

Wenn Sie die connect-Funktion nicht selbst im Konstruktor geschrieben haben und Sie dieses Projekt weiterverwenden möchten, löschen Sie wieder die Verbindung zwischen Signal und Slot aus dem Qt Designer. Markieren Sie dazu die rote Verbindungslinie und entfernen Sie sie.

Falls Sie einen Datenaustausch zwischen einer *SpinBox* und einem *Slider* herstellen müssen, ist dies mit dem Qt Designer relativ einfach zu realisieren (Bild 2.15 und Bild 2.16). Es werden jeweils die Funktionen valueChanged(int) und setValue(int) miteinander verbunden. Es existieren also sowohl die Signal- als auch die Slot-Funktionen bereits in den entsprechenden Klassen.

14 14 14 18 18 18 18 18 18 18 18 18 18 18 18 18		Filter	
Geben Sie Text ein	T. T	T HOLI	
	QU Verbindung bearbeiten - Qt Creator		×
	horizontalSlider (QSlider)	spinBox (QSpinBox)	
	range(hanged(int)	clear()	
	sliderMoved(int)	setValue(int)	
-	sliderPressed()	stepDown()	
	sliderReleased()	stepUp()	
	valueChanged(int)		
	Ändern	Ändern	
	Signale und Slots von QWidget anzeigen		
		ОК	Abbrechen

Bild 2.15 In den Klassen vorhandene Signal- und Slot-Funktionen

<b>喝喝喝喝三日又:</b>	9			
Geben Sie Text ein	Ŧ		Filter	
			Objekt	Klasse
	Verbindung bearbeiten - Qt Creator			×
	spinBox (QSpinBox) editingFinished() textChanged(QString) valueChanged(QString) valueChanged(int) Andern Signale und Slots von QWidget anzeigen	korizonta setValue Ändern.	ISIder (QSider) e(int)	Abbrechen

Bild 2.16 Herstellen der Verbindung zwischen Signal- und Slot-Funktionen

Sehen Sie sich Bild 2.16 einmal genau an, stellen Sie fest, dass zwei Funktionen value Changed() zur Verfügung stehen, valueChanged(int) und valueChanged(QString). Falls Sie diese Verbindungen nicht mit dem Designer erstellen lassen, sondern selbst programmieren möchten, stoßen Sie eventuell auf ein Problem.

Nach der älteren connect()-Syntax können Sie im Konstruktor folgende Zeile schreiben, die auch funktioniert:

Ändern Sie den Wert in der SpinBox, bewegt sich der Slider.

Versuchen Sie jetzt, die connect()-Zeile mit der neuen Syntax zu schreiben. Sie würden es wahrscheinlich so probieren:

Einige kennen das Problem eventuell schon. Erstellen Sie ein Projekt mit der Version Qt5.15, würde der Qt Creator jetzt beim Startversuch folgenden Fehler ausgeben:

```
Fehler: no matching function for call to 'MainWindow::connect(QSpinBox*&, <unresolved overloaded function type>, QSlider*&, void (QAbstractSlider::*)(int))'
```

In der Version Qt6 gibt es diesen Fehler nicht mehr, und das Programm funktioniert einwandfrei.

Bei einer älteren Qt-Version behelfen Sie sich dann damit, dass Sie eine Typumwandlung durchführen (eigentlich suchen Sie in der Klasse QSpinBox die richtige Funktion valueChanged()).

Diese Zeile sieht dann so aus:

### 2.4 Erstellen ohne Qt Designer

Sie können ein Formular der Oberklasse QMainWindow ohne Designer folgendermaßen erstellen. Zuerst erzeugen Sie eine neue Qt-Widgets-Anwendung. Achten Sie darauf, dass diesmal keine Form-Datei erzeugt wird (die entsprechende Checkbox bleibt leer).

Wenn Sie dieses Programm starten, entsteht schon ein Formular von einer Mindestgröße, ohne jeglichen zusätzlichen Quelltext. Sie haben jetzt keine Möglichkeit, das Layout des Formulars über den Designer zu ändern (z.B. einen Button oder ein Label hinzuzufügen). Den Designer können Sie nicht benutzen. Sie haben seine Verwendung ja beim Anlegen des Projekts ausgeschlossen. Ergänzen Sie den vorhandenen Quelltext folgendermaßen: Listing 2.3 Datei mainwindow.h in Projekten ohne Designer

```
01 #ifndef MAINWINDOW H
02 #define MAINWINDOW H
03 #include <QMainWindow>
04 #include <QPushButton>
05 #include <QLabel>
06
07 class MainWindow : public QMainWindow
08 {
09
        Q OBJECT
        QPushButton* button;
        QLabel* label;
  public:
        MainWindow(QWidget *parent = nullptr);
14
        void fensterDesign();
15
        ~MainWindow();
16 };
17 #endif // MAINWINDOW H
```

Zusätzlich sind hier die beiden Klassen QPushButton und QLabel zu inkludieren. Wir wollen sie dann benutzen. Ebenso wird eine Funktion fensterDesign() hinzugefügt. Zu beachten sind auch die beiden Zeiger *button* und *label*.

```
Listing 2.4 Datei mainwindow.cpp in Projekten ohne Designer
```

```
01 #include "mainwindow.h"
03 MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent)
04 {
        fensterDesign();
06 }
07 void MainWindow::fensterDesign()
08 {
09
        resize(400,300);
10
        button = new QPushButton(this);
        button->setGeometry(70,200,75,23);
        button->setText("OK");
        label = new QLabel(this);
14
        label->setGeometry(70,100,250,30);
15
        QFont font;
16
        font.setPointSize(14);
17
        label->setFont(font);
18 }
```

In der Funktion fensterDesign() (Zeile 7–18) erfolgt die Gestaltung des Formulars. Es werden Instanzen der Klassen QPushButton und QLabel erzeugt und Größe, Standort und Beschriftung übergeben.

Da der Button auch funktionieren soll, erhält die Klasse eine zusätzliche Slot-Funktion, und es wird die connect()-Funktion im Konstruktor hinzugefügt:

Neu ist die Slot-Funktion buttonClick() in der mainwindow.h

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
    QPushButton* button;
    QLabel* label;
private slots:
    void buttonClick();
```

und ihre Implementierung in der mainwindow.cpp.

```
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent)
{
    fensterDesign();
    connect(ui->pushButton_ok, SIGNAL(clicked()), this, SLOT(buttonClick()));
}
void MainWindow::buttonClick()
{
    label->setText("Button wurde geklickt...");
}
```

Wenn Sie jetzt den Button klicken, erscheint der Text auf dem Label. Sie haben also mit insgesamt wesentlich weniger Quelltext den gleichen Effekt erreicht wie mit dem Designer.

## **2.5** Die Benutzung von Layouts



Bild 2.17 Ausschnitt aus der Qt-Bibliothek

Wollen Sie die Anordnung von Widgets auf dem Formular nicht Ihrem Gefühl überlassen, bietet es sich an, im Ot Designer für das Aussehen Layouts zu verwenden. Markieren Sie dazu die anzuordnenden Elemente auf dem Formular gleichzeitig und wählen Sie aus dem oberen Menü z. B. *Objekte waagerecht anordnen* (Bild 2.18). Im neuen Eigenschaftsfenster können Sie wieder die gewünschten Eigenschaften auswählen und verändern (hier erhält z. B. *layoutSpacing* den Wert 20). Wenn Sie das für jedes Paar gemacht haben, fügen Sie noch ein Layout für alle Elemente hinzu. Es entsteht Bild 2.20.



Bild 2.18 Verwendung von Layouts mit dem Qt Designer



Bild 2.19 Layouts im Qt Designer angewandt

Im Formular sind die gewünschten Widgets jetzt entsprechend angeordnet. Eventuell benötigen Sie ein wenig Übung dabei, insbesondere, wenn Sie bei größeren Formularen verschachtelte Layouts verwenden. Zum Layout-Management finden Sie umfangreiche Informationen und Beispiele in der Qt-Dokumentation unter:

https://doc.qt.io/qt-6/layout.html

## 2.6 Das Erstellen und die Funktion von Menüs

Erstellen Sie ein neues Projekt mit der Oberklasse QMainWindow und benutzen den Qt Designer, erhalten Sie bereits einen Formularvorschlag nach Bild 2.20. Dort ist bereits eine Vorbereitung für ein Menü enthalten.

-	💷 mainwindow.ui @ Test - Qt Creator							
Datei Bearbeiten Ansicht Erstellen Debuggen Analyse Extras Fenster Hilfe								
n 🍞 mainwindow.ui 💦 👻 🖼 🖳 🖼 🖳 🥅 💷 🗮 📰 🔛								
Filter	Ţ							
Layouts								
Willkommen 🗧 Vertical Layout								
E Horizontal Layout								
Editieren 🗱 Grid Layout								
Form Layout								

Bild 2.20 Bereits vorhandenes Menü im Qt Designer

Tragen Sie dort den Namen für ein gewünschtes Menü ein und drücken Sie die **Enter**-Taste. Es wird der erste Menüpunkt aktiviert, und Sie können auch dort wieder einen Text eingeben und die **Enter**-Taste drücken (Bild 2.21).



Bild 2.21 Erstellen eines Menüs im Qt Designer

Das machen Sie so oft, wie Sie Menüpunkte hinzufügen möchten. Vergessen Sie nicht, auch nach dem letzten Eintrag die **Enter**-Taste zu drücken. Ansonsten erscheint der Text nicht.

Beachten Sie bitte, dass ein Menü einen Namen hat (hier z.B. File) und die anklickbaren Menüpunkte senkrecht darunter angeordnet sind. Es können also noch weitere Menüs daneben angeordnet werden.

QC ma	📧 mainwindow.ui @ Test - Qt Creator																		
Datei	Bearbeiten	Ansicht	Erstellen	Debuggen	An	alyse	Extras	Fens	ster	Hilfe									
	🖬 📝	mainwindo	w.ui*			Б Ц	, R. I	s, III	≡	МZ	: ::		10						
	Filter					Fil	e Gel	oen Sie	Text	ein		•						1	
Willkom		rtical Lavo	Layouts		_^		Ope	n		-	1			 		 	: :	: :	
	Шн	prizontal La	iyout				Clos	e		•									
Editier	en gan Gr	id Layout	,				Geb	en Sie Te	ext e	in									
1	Ba Fo	rm Layout					iren	ier ninz	urug	gen	]::								

Bild 2.22 Im Qt Designer erstelltes Menü

Starten Sie dieses Programm, erhalten Sie folgendes Formular:

MainWindow	-	×
Open		
Close		
ОК		

Um den Menüpunkten auch eine Funktion zu geben, benötigen Sie wieder Signal- und Slot-Funktionen.

In der Datei *ui\_mainwindow.h* ist inzwischen durch die Arbeit mit dem Ot Designer eine Ergänzung geschehen.

Listing 2.5 Ausschnitt dieser Datei mit erzeugtem Menü

```
01 class Ui MainWindow
02 {
03 public:
04
        QAction *actionOpen;
05
        QAction *actionClose;
06
        QWidget *centralwidget;
        QPushButton *pushButton ok;
        QMenuBar *menubar;
08
09
        QMenu *menuFile;
        QStatusBar *statusbar;
        void setupUi(QMainWindow *MainWindow)
13
        {
14
            if (MainWindow->objectName().isEmpty())
15
                MainWindow->setObjectName(QString::fromUtf8("MainWindow"));
16
            MainWindow->resize(392, 199);
```

```
actionOpen = new QAction(MainWindow);
18
            actionOpen->setObjectName(QString::fromUtf8("actionOpen"));
            actionClose = new QAction(MainWindow);
            actionClose->setObjectName(QString::fromUtf8("actionClose"));
            centralwidget = new QWidget(MainWindow);
            centralwidget->setObjectName(QString::fromUtf8("centralwidget"));
            pushButton ok = new QPushButton(centralwidget);
24
            pushButton ok->setObjectName(QString::fromUtf8("pushButton ok"));
            pushButton ok->setGeometry(QRect(40, 120, 75, 23));
26
            MainWindow->setCentralWidget(centralwidget);
            menubar = new QMenuBar(MainWindow);
28
            menubar->setObjectName(QString::fromUtf8("menubar"));
29
            menubar->setGeometry(QRect(0, 0, 392, 21));
            menuFile = new OMenu(menubar);
            menuFile->setObjectName(QString::fromUtf8("menuFile"));
            MainWindow->setMenuBar(menubar);
            statusbar = new QStatusBar(MainWindow);
34
            statusbar->setObjectName(QString::fromUtf8("statusbar"));
            MainWindow->setStatusBar(statusbar);
            menubar->addAction(menuFile->menuAction());
38
            menuFile->addAction(actionOpen);
            menuFile->addAction(actionClose);
40
41
            retranslateUi(MainWindow);
42
43
            QMetaObject::connectSlotsByName(MainWindow);
44
        } // setupUi
45
46 };
```

Es ist zu sehen, dass zwei Zeiger auf die Klasse QAction (Zeile 4 und 5) sowie Instanzen dieser Klasse erzeugt wurden (Zeile 17–20). Ebenso wurde ein Zeiger auf QMenu erzeugt und in Zeile 30 eine Instanz gebildet und dem Menü hinzugefügt. Die Menüpunkte *Open* und *Close* werden dann zu diesem Menü ebenfalls hinzugefügt (Zeile 38 und 39).

Schauen Sie nun einmal in die Klasse QAction in der Bibliothek. Dort finden Sie unter der Überschrift *Signals* eine Funktion triggered(), die ein Signal aussendet, wenn Sie den entsprechenden Menüpunkt aktivieren. (Falls ein Fokus auf dem Menüpunkt liegt, kann das auch mit der Enter-Taste geschehen.) Sie benötigen für die Funktion des Menüs also nur noch Slot- und die connect()-Funktionen. Beide schreiben Sie wieder in die Dateien *mainwindow.h* bzw. *mainwindow.cpp*.

mainwindow.h (Ausschnitt):

```
private slots:
    void buttonClick();
    void menuOpen();
    vod menuClose();
```

Listing 2.6 Ausschnitt der Datei mainwindow.cpp

```
01 MainWindow::MainWindow(QWidget *parent) :
02  QMainWindow(parent),
03  ui(new Ui::MainWindow)
04 {
```

```
05
        ui->setupUi(this);
06
        connect(ui->pushButton ok, SIGNAL(clicked()), this, SLOT(buttonClick()));
        connect(ui->actionOpen,SIGNAL(triggered()),this,SLOT(menuOpen()));
80
        connect(ui->actionClose,SIGNAL(triggered()),this,SLOT(menuClose()));
09
    }
   void MainWindow::menuOpen()
    {
        gDebug() << "Open...";</pre>
14
   }
15
16
   void MainWindow::menuClose()
18
        aDebua() << "Close...";</pre>
19
    }
```

Neu hinzugekommen sind die Zeilen 7 und 8 sowie 11–19. Die beiden Slot-Funktionen menuOpen() und menuClose() werden nun nach einem Klick auf den entsprechenden Menüpunkt aufgerufen.

Weitere Menüs können Sie mit dem Designer leicht hinzufügen, indem Sie neben dem vorhandenen ein weiteres eintragen. Alle weiteren Schritte entsprechen den bisherigen.

### 2.7 Ein Beispiel mit QTabWidget

Wir bauen dafür eine neue Qt-Widgets-Anwendung und benutzen den Qt Designer. Die Anwendung nennen Sie *TabWidget*. In der Datei *mainwindow.h* wird die Klasse QTabWidget inkludiert und ein Zeiger auf diese Klasse hinzugefügt: QTabWidget\* tabWidget;

Weiterhin fügen Sie dem Projekt zwei Klassen hinzu, *Datei* und *Sprache*. Diese beiden Klassen werden in der *MainWindow* bekannt gemacht, indem Sie beide in die Datei *mainwindow.h* eintragen:

```
#include "datei.h"
#include "sprache.h"
```

Die Klassen *Sprache* und *Datei* erhalten einen Inhalt Ihrer Wahl. In diesem Beispiel sollen sie einfach zwei Buttons erhalten. Natürlich können Sie auch jegliches andere Widget verwenden. Auch können Sie zu diesen Klassen weitere Formulare erstellen und mit dem Qt Designer arbeiten. Im nächsten Abschnitt sehen Sie, wie das funktioniert.

Die Dateien .h erhalten jeweils einen Zeiger auf QPushButton und sind abgeleitet von QWidget.

Listing 2.7 Datei sprache.h des Projekts "TabWidget"

```
01 #ifndef SPRACHE_H
02 #define SPRACHE_H
03
04 #include <QWidget>
05 #include <QPushButton>
```

```
06
07 class Sprache : public QWidget
08 {
09 Q_OBJECT
10
11 QPushButton* pb;
12
13 public:
14 explicit Sprache(QWidget *parent = nullptr);
15
16 };
17
18 #endif // SPRACHE H
```

Listing 2.8 Datei datei.h des Projekts TabWidget

```
#ifndef DATEI H
    #define DATEI H
04
    #include <QWidget>
   #include <QPushButton>
06
   class Datei : public QWidget
80
    {
09
        Q OBJECT
        QPushButton* pb;
11
12
    public:
13
        Datei(QWidget* = nullptr);
14
        ~Datei();
   };
16
    #endif // DATEI H
```

Nun wird im Konstruktor der Klasse MainWindow das TabWidget aufgebaut.

Listing 2.9 Datei mainwindow.cpp des Projekts "TabWidget"

```
01 #include "mainwindow.h"
02 #include "ui_mainwindow.h"
04
   MainWindow::MainWindow(QWidget *parent) :
        QMainWindow(parent),
06
        ui(new Ui::MainWindow)
   {
08
        ui->setupUi(this);
09
        tabWidget = new QTabWidget(this);
        tabWidget->setFixedSize(390,290);
        tabWidget->addTab(new Datei(),"Datei");
        tabWidget->addTab(new Sprache(),"Sprache");
13 }
14
```

In den Zeilen 9–12 ist zu sehen, dass eine Instanz der Klasse QTabWidget erzeugt und die Größe dieses *TabWidget* bestimmt wird. Diesem Element werden dann Instanzen der beiden Klassen *Datei* und *Sprache* hinzugefügt und die Beschriftung der Tabs benannt. Das Ergeb-

nis ist ein Formular mit zwei Tabs, über die Sie auswählen können, welche Klasse aktiviert und welcher Inhalt angezeigt werden soll (Bild 2.24). Sie können damit jetzt verschiedene Funktionalitäten in einem Programm vereinen. Selbstverständlich lassen sich auch mehr als zwei Tabs programmieren.

MainWindow	_	$\times$	I MainWindow -	×
Datei Sprache			Datei Sprache	
Datei			Sprache	

Bild 2.24 Die beiden Möglichkeiten der Benutzung von Tabs

### 2.8 Ein zweites Formular hinzufügen

Lassen Sie uns dafür ein neues Projekt erstellen. Nennen Sie es *ZweiFenster*. Erzeugen Sie mit dem Qt Creator zuerst eine Qt-Widgets-Anwendung mit diesem Namen. Nennen Sie die Klasse *ErstesFenster*. Die Oberklasse bleibt QMainWindow. Die Form-Datei soll erzeugt werden. Als Compiler wählen Sie wieder *MinGW*. Das Formular erhält einen Button, der mit *Fenster 2* beschriftet wird. In die Klasse *ErstesFenster* werden eine private Slot-Funktion fensterZwei() sowie die connect()-Funktion eingetragen.

Jetzt fügen Sie das zweite Formular hinzu. Markieren Sie dazu den Namen des Projekts (*ZweiFenster*). Nach Klick mit der rechten Maustaste erscheint ein Kontextmenü. Wählen Sie dort *Hinzufügen* aus (Bild 2.25).



Bild 2.25 Möglichkeit des Hinzufügens im Projektexplorer

Im darauf erscheinenden Fenster wählen Sie *Qt-Designer-Formularklasse* aus (Bild 2.26) und danach *Dialog without Buttons* (Bild 2.27). Die Klasse nennen Sie *ZweitesFenster*. Auch dazu wird wieder eine Form-Datei erstellt.

💶 Neue Datei - Qt Creator		×
Vorlage:		Alle Vorlagen $\checkmark$
Dateien und Klassen C/C++	Qt Item Model	Erstellt ein Qt-Designer-Formular mit zugehörigem Klassenrumpf (bestehend aus C+++Header- und - Ouelldatei) für Implementierungszwecke. Sie können
Modellierung Qt	uith Qt-Designer-Formularklasse	Formular und Klasse zu einem existierenden Qt- Widget-Projekt hinzufügen. Unterstützte Plattformen:
GLSL Allgemein	Qt-Designer-Formular	Desktop     WebAssembly Runtime     Windows Runtime (lokal)
Java Python	Qt Ressourcen-Datei	
	qml QML-Datei (Qt Quick 2)	
	ui.cml QtQuick UI-Datei	
	js JS-Datei	
		Auswählen Abbrechen

Bild 2.26 Auswahl einer Formularklasse

		×
Ot Designer Formularklasse		
Qt-Designer-Formularkiasse		
Formularvorlage	Formularvorlage auswählen	
Klassendetails	✓ templates\forms	
Zusammenfassung	Dialog with Buttons Bottom Dialog with Buttons Right Dialog without Buttons	
	Main Window	
	> Widgets	-
	· mugeo	
		Embedded-Entwurf
		Geräteprofil: Kein 🗸
		Bildschirmgröße: Vorgabe
		Weiter Abbrechen

Bild 2.27 Auswahl der Art des Formulars

Der Projektexplorer sieht jetzt so aus:





Sie können nun beide Formulare im Qt Designer bearbeiten. Klicken Sie dazu auf *erstesfenster.ui* oder *zweitesfenster.ui*.

Fügen Sie eine Zeile #include "zweitesfenster.h" in die Datei *erstesfenster.h* ein. Nun steht die Klasse *ZweitesFenster* im Quelltext zur Verfügung. Erstellen Sie weiterhin einen privaten Zeiger auf die Klasse *ZweitesFenster* (im Beispiel wird er *zf* genannt).

In der Slot-Funktion der Klasse ErstesFenster können Sie nun das zweite Formular erzeugen.

```
void ErstesFenster::fensterZwei()
{
    zf = new ZweitesFenster();
    zf->show();
}
```

Sollen aus dem ersten Formular Werte in das zweite Formular übergeben werden, ist das z.B. über einen weiteren Konstruktor der Klasse *ZweitesFenster* möglich, der dann hier anstelle des Standardkonstruktors aufgerufen wird.

Sie können so viele Zweitfenster erzeugen, wie Sie möchten. Bei jedem Klick auf den Button entsteht ein neues Fenster.

Fügen Sie als letzte Zeile in diese Funktion fensterZwei() den Aufruf der Funktion close() aus der Klasse QWidget ein, wird nach dem Öffnen des zweiten Fensters das erste Fenster geschlossen.

Falls nach dem Öffnen des zweiten Fensters das erste (und damit auch der Button) nicht mehr benutzt werden soll, tauschen Sie show() gegen exec() aus. Während nach dem Aufruf von show() die Funktion fensterZwei() beendet wird, wartet exec() auf ein Schließen des Fensters. Erst danach wird die Funktion fensterZwei() ebenfalls beendet. Im Destruktor der Klasse löschen Sie mit delete zf den Zeiger zf.

## 2.9 Maus- und Tastatur-Events

Im Modul QtGUI gibt es die Klassen QMouseEvent und QKeyEvent. Mit ihnen können Mausoder Tastatur-Events ausgewertet werden, die das Betriebssystem feststellt. Die Ableitungshierarchie der Klasse QMouseEvent wurde in Qt 6 ein wenig geändert. Sie sieht jetzt so aus:



Bild 2.29 Ausschnitt aus der Qt-Bibliothek

Erstellen Sie ein Beispielprojekt, um die beiden Events auszuprobieren.

Sie erstellen eine neue Qt-Widgets-Anwendung. Nennen wir sie *MausTastatur*. Die Form-Datei lassen Sie erstellen. Es entsteht dann beim Starten des Programms ein Formular. Der Qt Designer wird in diesem Projekt aber gar nicht benutzt. Die Datei *mainwindow.h* sieht so aus:

```
Listing 2.10 Datei mainwindow.h des Projekts "MausTastatur"
```

```
#ifndef MAINWINDOW H
02 #define MAINWINDOW H
04 #include <QMainWindow>
05 #include <QKeyEvent>
06 #include <QMouseEvent>
07 #include <ODebug>
08
09 namespace Ui {
10 class MainWindow;
   }
13 class MainWindow : public QMainWindow
14
   {
15
        Q OBJECT
16
    public:
18
        explicit MainWindow(QWidget *parent = nullptr);
19
        void keyPressEvent(QKeyEvent*);
        void mousePressEvent(QMouseEvent*);
        ~MainWindow();
23 private:
24
        Ui::MainWindow *ui;
25
   };
26
    #endif // MAINWINDOW H
28
```

Über #include werden die beiden schon genannten Klassen QKeyEvent und QMouseEvent eingebunden.

Zusätzlich überschreiben Sie die beiden Funktionen keyPressEvent() und mousePress Event(). Sie sind virtuelle Funktionen der Klasse QWidget.

In der Datei mainwindow.cpp werden beide Funktionen wie folgt implementiert.

Listing 2.11 Datei mainwindow.cpp des Projekts MausTastatur

```
#include "mainwindow.h"
   #include "ui mainwindow.h"
04 MainWindow::MainWindow(QWidget *parent) :
05
        OMainWindow(parent).
06
        ui(new Ui::MainWindow)
   {
80
        ui->setupUi(this);
09
   }
11 void MainWindow::keyPressEvent(QKeyEvent* e)
12 {
13
        if(e->key() == Qt::Key F1) qDebug() << "F1...";
```

```
else if(e->key() == Qt::Key_H) qDebug() << "H...";</pre>
14
    }
16
    void MainWindow::mousePressEvent(OMouseEvent* e)
18
        if(e->button() == Qt::LeftButton)
             qDebug() << "links...";</pre>
             qDebug() << "x" << e->position().rx();
             qDebug() << "y" << e->position().ry();
24
        else if(e->button() == Qt::RightButton)
26
             aDebug() << "rechts...":</pre>
             qDebug() << "x" << e->position().rx();
             qDebug() << "y" << e->position().ry();
29
         }
    }
    MainWindow::~MainWindow()
    {
        delete ui;
36
    }
```

In der Funktion keyPressEvent() (Zeile 11-15) steht die als Parameter übergebene Instanz von QKeyEvent (die Adresse steht im Zeiger *e*) zur Verfügung. Diese Instanz wird erzeugt, wenn Sie eine Taste der Tastatur drücken. Mithilfe von *e* lässt sich jetzt über die Funktion key() der Klasse QKeyEvent und das Enum Qt::Key die aktivierte Taste ermitteln.

Genauso stehen in der Klasse QMouseEvent eine Funktion button() aus der Klasse QSingle PointEvent und das Enum Qt::MouseButtons zur Verfügung. Die beiden Enums stammen aus dem Qt Namespace, der über QtCore mit inkludiert wird. Sie erhalten Informationen zu diesem Namespace über:

#### https://doc.qt.io/qt-6/qt.html

Während Sie in der Funktion keyPressEvent() die entsprechende Taste direkt ermitteln können, geht es bei einem MouseEvent um die Koordinaten des Punktes des Mauszeigers auf dem Formular, an denen der Klick erfolgt ist. Die x- und y-Werte dieser Koordinaten ermitteln Sie mit den Zeilen 22 und 23 oder 28 und 29, je nach verwendeter Maustaste. Aus dem Enum können Sie auch noch weitere Maustasten entnehmen.

### 2.10 Shortcuts für die Bedienung des Qt Creators

Außer über die Menüs können Sie den Ot Creator auch über die Tasten der Tastatur bedienen. Einige davon werden bei einzelnen Menüpunkten auch angezeigt. Hier ist noch einmal eine Übersicht wichtiger Standard-Tastenkombinationen.

- Ctrl + B Erstellen des Projekts (Build)
- Ctrl + R Starten des Projekts (Run)

## Index

#### Α

Animationen 191 ff. Animationstypen von QML 191 ApplicationWindow 189 Assoziative Qt-Container 63

#### В

Bild zeichnen 79 Breakpoint 241 Build-Ordner 14 Button 236

#### С

Canvas 184 CMake 12 CMakeLists.txt 12 Compiler 30 connect()-Funktion 37, 40 connect()-Syntax 39 Connections 214 Connection String 136 Containerklassen 58, 63

#### D

Datei 149 – .pri 11 – .pro 10 – .ui 13 – ui\_[Klassenname].h 14 Datenbank 125 Datenbankklassen 129 Datenbankserver 125 Datenbanktreiber 135 Datentypen 57 Debugger 240 Delegates 59 ff. Dokumentation 244 Dokumentationskommentar 245 DOM 162 Domainnamen 136 Doxygen 244 Drucken per QML 210 Druckvorgang 149 Dynamisches Linken 248

#### Ε

easing.type 191 emit 19 Ereignistypen 62 Event Handling 62 event loop 173 Eventklassen in Qt 62 Exceptions 239 exec() 173

#### F

Farbverläufe 96 FIFO 63

#### G

getContext() 185 GUI-Erstellung 25

#### Н

Haltepunkte 240 High-level API 66 HTML 113

#### I

Input-Feld für Text 238 Installation 4 isChecked() 155 Item 198 Iterator 63

#### J

JavaScript-Funktionen 179 JSON 167

#### Κ

keyPressEvent() 52 Kommentarzeichen 17

#### L

Layouts 27, 41 LIFO 63 Low-Level-Funktionen 66 lupdate 189 lupdate.exe 117

#### М

main()-Funktion 16 main.gml 174 MariaDb 132 Maus-/Tastatur-Events 51 Meldungen 55 Menü 43 MenuBar 189 Menultem 189 Meta-Object Compiler 2, 13 Microsoft Access 137 Microsoft SQL Server 136 MinGw 30 Model 142 Model-View-Framework 28 Model-View-Prinzip 142 Module 6

MouseArea 179 Mouse Events 88, 238 mousePressEvent() 52 MVC-Prinzip 59 MySQL 132

#### Ν

NumberAnimation 196

#### 0

open() 159

#### Ρ

paintEvent() 77, 81 ParallelAnimation 196 PDF-Datei erzeugen 149 phpMyAdmin 132 printsupport 150 Projektdatei 10 Property 216 Property-System 73 PropertyAnimation 193 PropertyChanges 202

#### α

Q\_PROPERTY() 6, 215 ff. qDebug() (Funktion) 32 QDebug (Klasse) 32, 55 QDialog 25 QDoc 248 QFile 158f. QFileDevice 158 QFrame 87 QGradient 96 QGraphicsScene 90, 98 QGraphicsView 98 Qlmage 79 QIODevice::Append 158 QIODevice::ReadOnly 158 QIODevice::WriteOnly 158 QLabel 154 QMainWindow 25 QMessageBox 55 QML (Qt Modeling Language) 25, 171, 209 - Animationstypen 191 - Drucken 210

OMI-Animationen 235 QML-Basistypen 178, 209 QML-Debuggerkonsole 244 Q OBJECT 6, 225 QObject 5, 227 QPaintDevice 77 QPaintEngine 77 QPainter 152 QPrintDialog 149, 212 QPrinter 149, 152, 212 QPushButton 33 QSqlDatabase 129, 135 QSalQuery 129 QStatusBar 56, 154 Ωt - Eventklassen 62 - Zustandseditor 202 Ot Assistant 8 Qt-Bibliothek 3 **Qt-Container** - assoziative 63 - sequenzielle 63 Qt-Containerklassen 63 Qt Creator 4.8 - Tastenkombinationen 53 Qt-Datentypen 58 Ot Designer 9, 18 Qt Linguist 9, 117ff., 189 Qt Meta Language 25 Ot Modeling Language 25 Qt-Plug-ins 2 QtPrintSupport 149, 212 **Qt Property System 215** QtQML 171 Qt Quick 171 Qt Ressourcen-Datei 102 QtSCXML 202 **Qt StyleSheets** 107 QTabWidget 46 QTextBrowser 82 QTextStream 160 **OtWidgets** 227 QtXML 163 quick 172 Quick Designer 177 Quick-Oberflächen 209 QVariant 73 QWidget 25 QXmlStreamReader 164 QXmlStreamWriter 164

#### R

read-only-Property 225 Registerinhalte 240 Regulärer Ausdruck 28 Ressourcen 79, 103 Ressource Compiler 13 - (rcc) 113 Ressourcensystem 101 Rich Text 113 Rich-Text-Dokumente 114 rotate() 95 RotationAnimation 199

#### S

scale() 95 Scribe 114 Sequenzielle Qt-Container 63 setupUi() 32 shear() 95 signals 6,18 Signal-Funktion 212 Signal- und Slot-Funktionen 18 - verbinden 34 slots 6.18 Slot-Funktion 142, 213 Sprachumstellung 122 SQL (Structured Query Language) 125 State 200 Statisches Linken 249 Statusmeldung 56 Statuszeile 56, 142 StyleSheets 106 ff., 146, 229 - externe Datei 111 Stylesheet-Editor 109

#### Т

Table View 146 TabWidget 233 Threads 66, 69 Top Level Window 235 tr() 120 Transformation 93 translate() 95 triggered() 45

#### U

Übersetzen - von Texten im Programm 117 Übersetzungsdatei 117 ui\_[Klassenname].h 14 User Interface Compiler 13ff.

#### V

View 142 Visual Studio 19

#### W

Widgets 27, 228, 232 Window 173

#### Х

XML 161,223

#### Ζ

Zeichnen – freihändig 89 – auf Widgets 77 Zustandseditor 199, 203 – von Qt 202 Zweites Fenster 51