Das Qwt Handbuch

Andreas Nicolai andreas.nicolai@gmx.de

Version 1.0.0, Mai 2025

Inhaltsverzeichnis

Über dieses Handbuch

Dieses Projekt ergänzt die Dokumentation zur Qwt-Bibliothek und bietet sowas wie ein Programmiererhandbuch mit vielen Details zu den Internas. Der Fokus liegt aber ganz klar auf der `QwtPlot` Diagrammkomponente.

Das Buch schreibe ich in Deutsch, aber Dank der modernen Webseiten-Übersetzer kann man das bestimmt auch locker in vielen anderen Sprachen lesen :-)

Die Texte, Bilder und Beispielquelltexte stehen unter der BDS-3 Lizenz (siehe Qwt Handbuch Repository). Wäre natürlich cool, wenn der eine oder andere hier etwas beisteuern würde. Das ist übrigends schon die 2. Auflage (komplett neu überarbeitet), weil es zum Zeitpunkt der ersten Ausgabe noch kein tolles AsciiDoc gab und ich irgendwie beim Textschreiben hängengeblieben bin.

Viel Spaß bei der Lektüre - und falls noch Inhalte fehlen, einfach Geduld haben und später wiederkommen (oder im Github-Repo ein Issue anlegen). Schaut Euch auch meine anderen Tutorials an, unter: Scheggenport-Blog und Tutorials!

— Andreas Nicolai

1. Überblick über die Qwt Bibliothek

Qwt - Qt Widgets for Technical Applications ist eine Open-Source Bibliothek für technische Anwendungen und stellt bestimmte Widgets für Anzeigen und Kontrollkomponenten bereit. Die wohl wichtigste Komponente der Qwt Bibliothek ist das `QwtPlot`, eine sehr flexible und mächtige Diagrammkomponente.

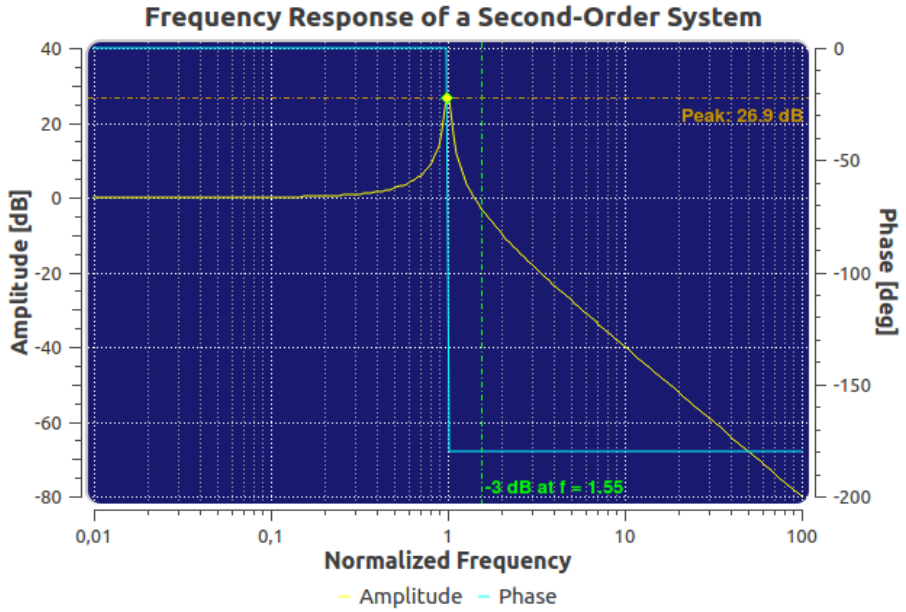


Abbildung 1. Beispiel für die QwtPlot-Komponente

Die Qwt Bibliothek steht unter einer Open-Source-Lizenz, wurde und wird aktiv vom Entwickler *Uwe Rathmann* gepflegt und wird auf SourceForge.net gehostet:

- Qwt Webseite (englisch)
- Qwt SourceForge Projektseite

1.1. Entwicklungsgeschichte

- die erste Version der Qwt-Bibliothek stammt noch aus dem Jahr 1997 von Josef Wilgen
- seit 2002 wird die Bibliothek von *Uwe Rathmann* entwickelt und gepflegt
- Version 5 ist wohl am weitesten verbreitet (erstes Release vom 26.02.2007)
- Version 6 (erstes Release vom 15.04.2011, kein Qt3 Support mehr) enthält wesentliche API-Änderungen
- aktuelle stabile Version 6.3.0 (Stand Mai 2025)
- im trunk gibt es zum Teil bereits wesentlich mehr und fortgeschrittene Funktionen

1.1.1. Download der Bibliothek

Die Qwt Bibliothek kann von der Qwt SourceForge Projektseite als Quelltextarchiv geladen werden. Unter Linux wird Qwt bei vielen Distributionen als Paket gehalten. Genau genommen gibt es mehrere Pakete für die unterschiedlichen Qwt-Bibliotheksversionen bzw. Qt Versionen. Details zur Installation und Verwendung der Bibliothek gibt es im Kapitel 6.

1.2. Widget-Konzept und Erscheinungsbild

Die Qwt Bibliothek liefert Komponenten, welche analog zu den normalen Qt-Widgets in Desktopanwendungen verwendet werden können. Die Komponenten verwenden die Qt Palette, sodass die Qwt-Widgets in die jeweilige Oberfläche passen. Dadurch integrieren sich die Widgets nahtlos in Programmoberflächen. Einzelne Komponenten des `QwtPlot` unterstützen auch Styles. So ermöglichen z.B. Abrundungseffekte beim Plot-Widget das Immitieren klassischer Anzeigen.

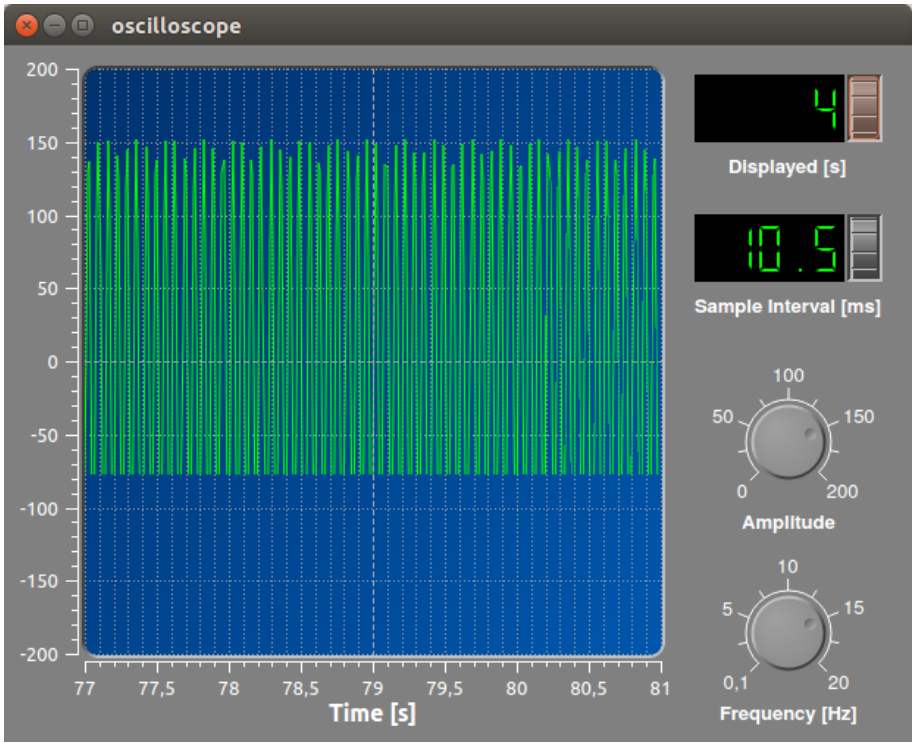


Abbildung 2. QwtPlot mit abgerundeten Ecken

Details zum Styling und zur Anpassung des Erscheinungsbildes sind im Kapitel 5 zu finden.

1.3. Besitzer/Eigentümer-Konzept des QwtPlot-Widgets

Eine grundlegende Eigenschaft der `QwtPlot`-Klasse ist die Besitzübername hinzugefügter Elemente. Dies gilt allgemein für alle Elemente des Plots (Linien, Marker, Legende, ...). D.h. nach Übertragung der Eigentümerschaft kümmert sich das `QwtPlot` um das Aufräumen des Speichers.

Einmal hinzugefügte Elemente werden nicht wieder losgelöst werden (bzw. nur über einen Trick, wie im [sec:XXX] beschrieben wird). Daher ist es sinnvoll, bei veränderlichen Diagrammelementen einen Mechanismus zur jeweiligen Neuerstellung eines Zeichenobjekts vorzusehen (Factory-Konzept).

1.4. Zeichenobjekte und deren Achsenabhängigkeit

Ein wesentliches Designmerkmal beim `QwtPlot` ist die Möglichkeit, beliebige Zeichenobjekte (Kurven, Marker, Legende, ...) dem Plot zu übergeben. Damit sich diese Zeichenobjekte (engl. *PlotItem*) am Koordinatengitter ausrichten können, wird ihnen eine Achsenabhängigkeit gegeben. Dadurch erhalten diese Zeichenobjekte eine Information, wann immer sich die Achsenskalierung ändert (durch Zoomen, oder Änderung der Wertebereiche etc.).

Diese Funktionalität definiert die zentrale Bedeutung der (bis zu) 4 Achsen im Diagramm. Deswegen sind diese auch fest im `QwtPlot` verankert und werden nicht wie andere Zeichenobjekte beliebig hinzugefügt.

1.5. Vererbungskonzept

Grundsätzlich ist das `QwtPlot` und die beteiligten Klassen auf maximale Anpassungsfähigkeit ausgelegt, d.h. es wird (fast) überall Polymorphie unterstützt. Wenn die eingebaute Funktionalität nicht zureichend ist, kann man einfach immer die entsprechende Klasse ableiten und die jeweils anzupassende Funktion re-implementieren und verändern. Dies wird anhand von Beispielen in den individuellen Kapiteln des Handbuchs beschrieben.

1.6. Verwendung der Designer Plugins

Die Qwt Bibliothek bringt Plugins für Qt Designer mit, welche das Einfügen von Qwt-Komponenten in ui-Dateien erleichtert. Es lassen sich jedoch keine QwtPlot-Eigenschaften festlegen oder Kurven hinzufügen. Die eigentliche Anpassung und Ausgestaltung des Plots erfolgt im Quelltext. Deswegen wird die Konfiguration und Anpassung des `QwtPlot` in diesem Handbuch ausschließlich durch normale API-Aufrufe demonstriert.

Soll das `QwtPlot` auch ohne Designer-Plugins im grafischen QtDesigner-Editor eingefügt werden, kann man einfach ein QWidget einfügen und dieses als Platzhalter für die `QwtPlot`-Klasse definieren (siehe auch [sec:insertWithWidget] im Einsteiger-Tutorial).

Eine Beschreibung, wie die Designer-plugins erstellt und in Qt Creator/Designer integriert werden ist im Kapitel 6.3 beschrieben.

2. Erste Schritte und ein interaktives Diagramm

Um mit der Qwt-Bibliothek warm zu werden, erstellen wir in einem einfachen Beispiel ein interaktives Diagramm mit der `QwtPlot`-Komponente.

2.1. Programmrohbau

2.1.1. QMake Projektdatei

Wir beginnen mit der qmake-Projektdatei, in der wir den Pfad für die Header-Dateien der Bibliothek und die zu linkende Bibliothek festlegen.

```
TARGET   = Tutorial1
QT       += core gui widgets
CONFIG   += c++11

win32 {
    # Pfad zu den Qwt Headerdateien hinzufügen
    INCLUDEPATH += C:/qwt-6.3.0/include
    CONFIG(debug, debug|release) {
        QWTLIB = qwtd
    }
    else {
        QWTLIB = qwt
    }
    # Linkerpfad
    LIBS += -LC://qwt-6.3.0/lib -l$$QWTLIB
}
else {
    # Pfad zu den Qwt Headerdateien hinzufügen
    INCLUDEPATH += /usr/local/qwt-6.3.0/include/
    # Linkerpfad, unter Linux wird standardmäßig nur die release-Version der Lib gebaut und installiert
    LIBS += -L/usr/local/qwt-6.3.0/lib -lqwt
}

SOURCES += main.cpp
```

Dies ist eine `.pro`-Datei für eine Qwt-6.3.0-Installation aus dem Quelltext mit Standardeinstellungen (siehe Kapitel 6.2).

Beachte, dass die im Debug-Modus kompilierte Qwt-Bibliothek ein angehängtes *d* hat. Unter Linux wird standardmäßig nur die release-Version gebaut und installiert, daher braucht man hier die Fallunterscheidung nicht.

2.1.2. Minimalistisches Hauptprogramm

Für die Verwendung des `QwtPlot` braucht man nur eine sehr minimalistische `main.cpp`.

Hauptprogramm, in dem nur das nackige Plot selbst erstellt und angezeigt wird

```
#include <QApplication>

#include <QwtPlot>

int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    QwtPlot plot;
    plot.resize(800,500);
    plot.show();
    return a.exec();
}
```

Wenn man das Programm compiliert hat und ausführen will, beklagt sich Windows über eine fehlende DLL. Dazu in den Projekteinstellungen, unter "Ausführen", im Abschnitt "Umgebung" die PATH-Variable bearbeiten und dort den Pfad `C:\qwt-6.3.0\lib` hinzufügen.

Das Programm zeigt ein ziemlich langweiliges (und hässliches) Diagrammfenster (später wird das noch ansehnlicher gestaltet).

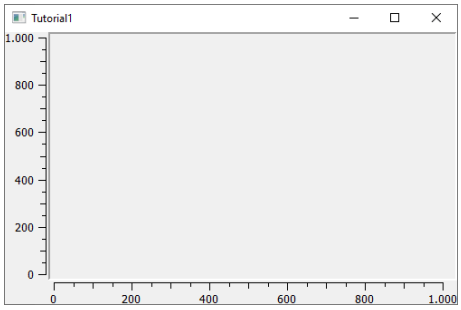


Abbildung 3. Das nackte Plotwidget

Ein Hinweis zu den Header-Dateien der Qwt-Bibliothek.

Analog zu Qt Klassen werden die Qwt-Klassen über den gleichnamigen Header eingebunden, also:

```
#include <QwtPlot>       // für Klasse QwtPlot
#include <QwtPlotCurve>  // für Klasse QwtPlotCurve
#include <QwtLegend>     // für Klasse QwtLegend
// ...
```

Diese Header-Dateien sind aber nur Wrapper um die eigentlichen Include-Dateien, mit dem Benennungsschema:

```
#include <qwt_plot.h>        // für Klasse QwtPlot
#include <qwt_plot_curve.h>  // für Klasse QwtPlotCurve
#include <qwt_legend.h>      // für Klasse QwtLegend
// ...
```

In früheren Versionen der Qwt-lib (auch der Debian-Paket-Version `libqwt-qt5-dev`) wurden die Wrapper-Headerdateien nach dem neuen Namensschema nicht installiert, sodass man die originalen `qwt_xxx.h` Includes verwenden muss. Wenn man also auch ältere Qwt-Versionen unterstützen möchte, bzw. unter Linux die Paketversion verwenden will, sollte die originalen Headerdateinamen verwenden.

2.2. Diagrammelemente hinzufügen

2.2.1. Linie hinzufügen

Als erstes fügen wir eine Linie bzw. Diagrammkurve hinzu (Header `QwtPlotCurve` bzw. `qwt_plot_curve.h`):

Hauptprogramm mit einer Linie

```
#include <QApplication>

#include <QwtPlot>
#include <QwtPlotCurve>

int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    QwtPlot plot;
    plot.resize(500,300);

    // etwas Abstand zwischen Rand und Achsentiteln
    plot.setContentsMargins(8,8,8,8);
    // Hintergrund der Zeichenfläche soll weiß sein
    plot.setCanvasBackground( Qt::white );

    // Daten zum Darstellen
    QVector<double> x = {0,4,5,10,12};
    QVector<double> y = {5.1,4,6.8,6.5,5.2};

    QwtPlotCurve *curve = new QwtPlotCurve();
    curve->setPen(QColor(180,40,20), 0);
    curve->setTitle("Line 1");
    curve->setRenderHint( QwtPlotItem::RenderAntialiased, true ); // Antialiasing verwenden
    curve->setSamples(x, y);
    curve->attach(&plot); // Plot takes ownership

    plot.show();
    return a.exec();
}
```

Im erweiterten Hauptprogramm wird zunächst der Header für die `QwtPlotCurve` eingebunden. Das Kurvenobjekt selbst wird mit `new` auf dem Heap erstellt.

Grundsätzlich gilt beim `QwtPlot`: Alle Plotelemente *müssen* via new auf dem Heap erstellt werden und dem Plot dann übergeben werden. Dieses wird dann Besitzer und gibt den Speicher frei. Deshalb dürfen Linien, Legende, Marker etc. *niemals* als Stack-Variablen erstellt werden, sonst gibt es (je nach Destruktoraufrufreihenfolge) einen Speicherzugriffsfehler.

Attribute wie Linienfarbe, Titel (wird später in der Legende angezeigt), und Antialising werden gesetzt (im Kapitel 3 werden alle Eigenschaften von Linien im Detail erläutert).

Die Funktion `setSamples()` setzt die Daten der Linie. Wichtig ist hier, dass die übergebenen Vectoren die gleiche Länge haben. Es handelt sich um eine parametrische Kurve, d.h. weder x noch y Werte müssen monoton sein oder sonstwelchen Regeln folgen. Jedes x,y Wertepaar definiert einen Punkt und diese Punkte werden mit der Linie verbunden.

Die Funktion `attach()` fügt das `QwtPlotCurve`-Objekt zum Diagramm hinzu.

Beim Hinzufügen der Linie mittels `attach()` zum Diagramm wird das Plot neuer Eigentümer und kümmert sich um das Aufräumen des Speichers. Man muss also nicht mehr manuell `delete` für das `QwtPlotCurve`-Objekt aufrufen.

Zusätzlich zu dem Code, welcher die Linie hinzufügt, wurden noch 2 kleine Anpassungen am Erscheinungsbild vorgenommen:

- Ränder wurden mittels `setContentsMargins()` hinzugefügt (siehe auch `QWidgdet::setContentsMargins()`)
- der Hintergrund der Zeichenfläche (*canvas*) wurde weiß gefärbt.

Das Ergebnis sieht schon eher nach Diagramm aus.

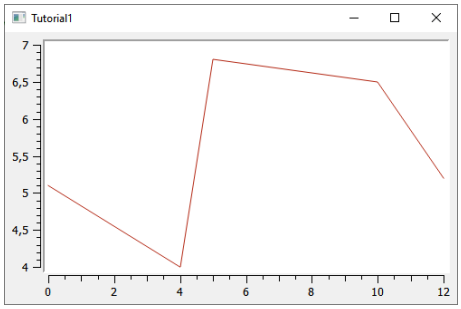


Abbildung 4. Diagramm mit Linie

2.2.2. Legende hinzufügen

Als nächstes wird eine Legende eingefügt (Header `QwtLegend` bzw. `qwt_legend.h`):

```
// Legende anzeigen
QwtLegend * legend = new QwtLegend();
plot.insertLegend( legend , QwtPlot::BottomLegend); // plot takes ownership
```

Auch hier wird oben wieder der Header für die Klasse `QwtLegend` eingebunden.

Die Legende kann links, rechts, oberhalb oder unterhalb der Zeichenfläche liegen, oder in der Zeichenfläche selbst. Das Anpassen der Legende wird in Kapitel 4 beschrieben.

Das Plot nimmt beim Aufruf von `insertLegend()` wiederum Besitz vom Legendenobjekt und kümmert sich um das Aufräumen des Speichers.

2.2.3. Diagrammtitel hinzufügen

```
// Titel hinzufügen
QwtText text("Ein Beispieldiagramm");
QFont titleFont;
titleFont.setBold(true);
titleFont.setPointSize(10);
text.setFont(titleFont);
plot.setTitle(text);
```

Die Klasse `QwtText` (Header `QwtText` bzw. `qwt_text.h`) kapselt einen QString und ergänzt Funktionalität zum Rendern von mathematischen Symbolen mittels MathML (siehe [sec:mathML]).

2.2.4. Diagrammraster hinzufügen

Gitterlinien werden durch das Zeichenobjekt `QwtPlotGrid` gezeichnet (Header `QwtPlotGrid` bzw. `qwt_plot_grid.h`):

```
// Hauptgitter anzeigen
QwtPlotGrid *grid = new QwtPlotGrid();
```

```
QPen gridPen(Qt::gray);
gridPen.setStyle(Qt::DotLine);
gridPen.setWidth(0);
grid->setPen(gridPen);
grid->attach( plot ); // plot takes ownership
```

Man kann auch mehrere Raster hinzufügen, z.B. eins für Hauptgitterlinien und eines für Nebengitterlinien.

Inzwischen sieht das Diagramm schon ganz ansehnlich aus.

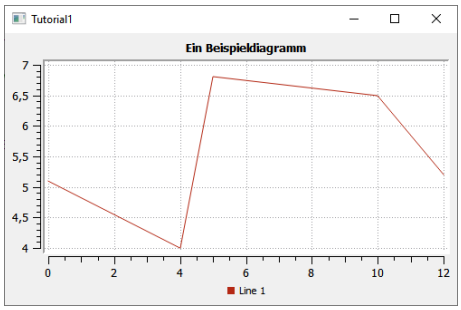


Abbildung 5. Diagramm mit Linie, Legende, Titel und Gitterlinien

2.2.5. Achsenkonfiguration

Das `QwtPlot` hat 4 Achsen eingebaut, genannt:

- `QwtPlot::yLeft` und `QwtPlot::yRight`
- `QwtPlot::xBottom` und `QwtPlot::xTop`

Standardmäßig sind die Achsen `xBottom` und `yLeft` sichtbar, wie im bisher verwendeten Plot.

Jedes Zeichenelement im Plot (Kurven, Marker, ...) wird einer oder mehrerer Achsen zugeordnet. In unserem Einführungsbeispiel verwendet die `QwtPlotCurve` standardmäßig die Achsen `xBottom` und `yLeft`.

Die Achsen können wie folgt konfiguriert werden.

```
// Achsen formatieren
QFont axisFont;
axisFont.setPointSize(8);
axisFont.setBold(true);
QFont axisLabelFont;
axisLabelFont.setPointSize(8);
// X-Achse
QwtText axisTitle("X-Werte");
axisTitle.setFont(axisFont);
// Titel Text und Font setzen
plot.setAxisTitle(QwtPlot::xBottom, axisTitle);
// Font für Achsenzahlen setzen
plot.setAxisFont(QwtPlot::xBottom, axisLabelFont);
// Y-Achse
axisTitle.setText("Y-Werte");
plot.setAxisTitle(QwtPlot::yLeft, axisTitle);
plot.setAxisFont(QwtPlot::yLeft, axisLabelFont);
```

Der Titel jeder Achse wird wiederum über ein `QwtText`-Objekt (enthält Text und Font) gesetzt. Der Font für die Zahlen an den Achsen selbst wird über `setAxisFont()` geändert.

Die Achsen selbst lassen sich vielfältig anpassen, siehe [sec:axes].

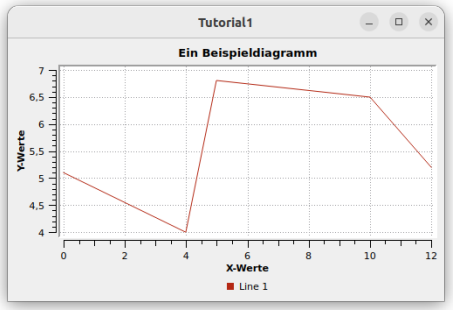


Abbildung 6. Vollständig formatiertes Diagramm (Linux Screenshot)

Die Achsen passen sich standardmäßig automatisch an den Wertebereich der angezeigten Kurven an. Das kann man natürlich auch ändern, siehe [sec:axes].

2.3. Interaktion mit dem Diagramm

Das `QwtPlot` bietet die üblichen Interaktionsmöglichkeiten für den Anwender, wie z.B. Herein- und Herauszoonmen, oder Verschieben des Plotausschnitts.

2.3.1. Zoomfunktionalität mit QwtPlotZoomer

Die Zoom-Funktionalität wird über die Klasse `QwtPlotZoomer` hinzugefügt (Header `QwtPlotZoomer` bzw. `qwt_plot_zoomer.h`):

```
// Zoomer hinzufügen
// Achtung: NICHT QwtPlot selbst als 3 Argument übergeben, sonder das canvas()
QwtPlotZoomer * zoomer = new QwtPlotZoomer(QwtPlot::xBottom, QwtPlot::yLeft, plot.canvas());  // plot takes ownership
zoomer->setTrackerMode( QwtPlotPicker::AlwaysOn ); // Kurvenvwerte unterm Cursor anzeigen
```

Wenn man mit der Maus über das Diagramm fährt, sieht man bereits einen veränderten Cursor und dank des Aufrufs `setTrackerMode(QwtPlotPicker::AlwaysOn)` sieht man nun auch die x- und y-Werte (des Achsen `xBottom` und `yLeft`) unter dem Cursor.

Hineinzoomen kann man, indem man die Linke Maustaste gedrückt hält, und ein Zoom-Rechteck aufzieht. Das kann man auch mehrmals hintereinander machen. Das `QwtPlot` merkt sich intern diese Zoomstufen. Herauszoomen kann durch Klick auf die rechte Maustaste, wobei immer eine Zoomstufe hinausgezoomt wird.

Die äußerste Zoomstufe wird im Konstruktor der `QwtPlotZoomer`-Klasse basierend auf den aktuellen Wertebereichen der *bereits hinzugefügten Kurven* bestimmt. Sollte man die Werte der Kurven nachträglich ändern, oder den Zoomer hinzufügen, *bevor* man dem Plot Kurven gegeben hat, so kann man die Funktion `QwtPlotZoomer::setZoomBase()` aufrufen. Details dazu gibt es im [sec:zoomer].

Im Quelltext gibt es eine Besonderheit. Während die bisherigen Plotelemente immer mit Memberfunktionen der `QwtPlot`-Klasse hinzugefügt wurde, bzw. mittels `attach()`, wird das Zoomerobjekt analog zu Qt Klassen als Kindobjekt der Zeichenfläche gegeben und registriert sich darüber als interaktives Element bei Plot.

Es ist wichtig darauf zu achten, dass man beim Konstruktor der Klasse `QwtPlotZoomer` als 3. Argument das Canvas-Objekt des Plots übergibt. Dieses erhält man mit der Funktion `QwtPlot::canvas()`. Wenn man hier stattdessen das Plot selbst übergibt, führt dies zu einem Speicherzugriffsfehler.

Im Konstruktor der `QwtPlotZoomer` Klasse registriert sich das Objekt als Kind des Canvas-Widgets, wodurch das QObject-System sich um die Speicherverwaltung kümmert. Man muss also das `QwtPlotZoomer` Objekt nicht freigeben.

Damit das Zoomer weiß, welche Achsen beim Zoom manipuliert werden sollen, muss man die x- und y-Achse im Konstruktor angeben. Möchte man z.B. beide y-Achsen gleichzeitig zoomen, braucht man zwei `QwtPlotZoomer` -Objekte.

2.3.2. Plotausschnitt verschieben mit QwtPlotPanner

Wenn man Ausschnitt eines hineingezoomten Plots interaktiv verschieben möchte, kann man den `QwtPlotPanner` hinzufügen (Header `QwtPlotZoomer` bzw. `qwt_plot_zoomer.h`):

```
// Panner hinzufügen, wie auch beim PlotZoomer muss das Canvas-Objekt als Argument übergeben werden
QwtPlotPanner * panner = new QwtPlotPanner(plot.canvas());  // plot takes ownership
panner->setMouseButton(Qt::MidButton); // Mittlere Maustaste verschiebt
```

Wie beim `QwtPlotZoomer` wird das Objekt als Kindobjekt des Canvas-Widgets hinzugefügt. Üblich ist das Verschieben von Bildschirminhalten mit gedrückter mittlerer Maustaste, also legt man das mit `setMouseButton()` fest.

2.4. Das QwtPlot in eine Designer-Oberfläche/ui-Datei integrieren

Wenn man mittels Qt Designer eine Programmoberfläche baut, möchte man da vielleicht auch ein `QwtPlot` einbetten. Das kann man auf zwei verschiedene Arten machen:

1. ein QWidget als Platzhalter einfügen und zu einem Platzhalterwidget für das `QwtPlot` machen, oder
2. die Qwt-Designer-Plugins verwenden.

2.4.1. Definition eines Platzhalterwidgets

Zur Erklärung wird im Qt Designer ein einfaches Widget entworfen:

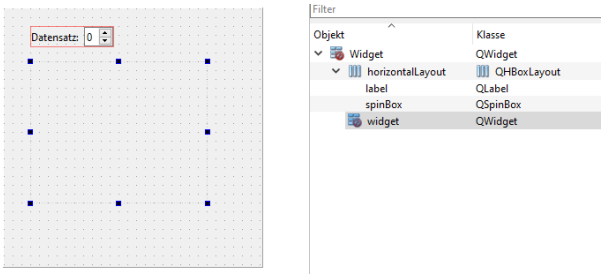
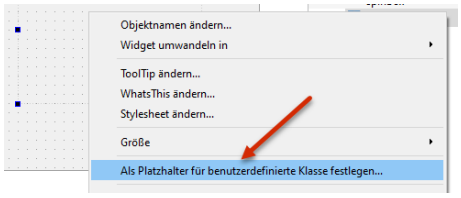
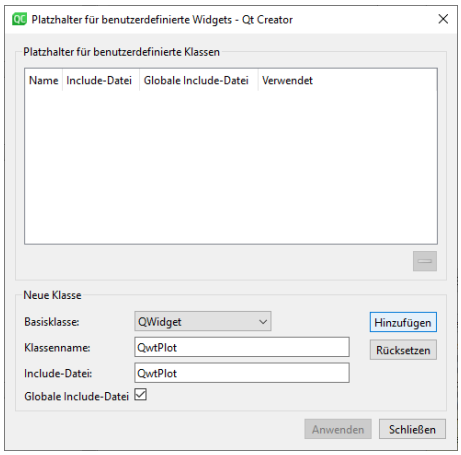


Abbildung 7. Widget mit Platzhalter-Widget für das Diagramm

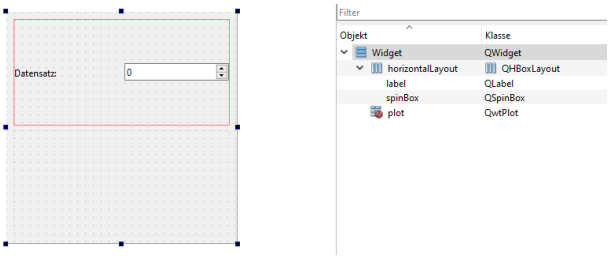
Unter der Spinbox wurde ein `QWidget` eingefügt. Dieses soll nun als Platzhalter für das `QwtPlot` dienen. Dazu im Kontextmenü des Widgets die Option "Als Platzhalter für benutzerdefinierte Klasse festlegen..." auswählen:



Und im Dialog eine neue Platzhalterklasse wie folgt definieren:



Die Eingabe mit "Hinzufügen" bestätigen und dann auf "Anwenden" klicken, um das Platzhalter-Widget in das `QwtPlot` zu wandeln. Wir benennen das noch in *plot* um, und füge das horizontale Layout und das Plotwidget in ein vertikales Layout ein:



Damit sich das Plotwidget den ganzen vertikalen Platz schnappt, wählt man das Top-Level Widget aus und scrollt in der Eigenschaftsleiste bis nach unten zu den Einstellungen für das vertikale Layout. Dort gibt man bei den Stretch-Faktoren "0,1" ein, wodurch sich das 2. Widget im Layout (das Plot) komplett ausdehnt.

2.4.2. Verwendung der Designer-Plugins

Wenn man die erstmal installiert hat (siehe [sec::designerPlugins]), kann man ein `QwtPlot` direkt aus der Komponentenpalette in den Entwurf zeihen und ist fertig.

3. Liniendiagramme

4. Legende

5. Anpassung/Styling der Qwt Komponenten

6. Download/Installation/Erstellung der Qwt Bibliothek

6.1. Download fertiger Pakete

6.1.1. Linux

Unter Linux kann man auf die Pakete des Paketmanagers zurückgreifen.

Debian/Ubuntu

```
# Paket mit Headern für die Entwicklung
sudo apt install libqwt-qt5-dev
```

Headerdatei-Pfad: `/usr/include/qwt`

6.2. Erstellung aus dem Quelltext

6.2.1. Windows

- Release `qwt-6.3.0.zip` herunterladen und entpacken.
- Datei `qwtconfig.pri` bearbeiten und Optionen ein-/ausschalten
- Kommandozeile mit Qt Umgebungsvariablen öffnen: Startmenu → Qt 5.15.2 (MinGW 8.1.0 64-bit)
- Ins Verzeichnis mit der `qwt.pro` wechseln

6.2.2. Windows - MinGW32

Es wird eine MinGW32 Installation mit `mingw32-make` im PATH erwartet.

```
:: Makefile erstellen
qmake qwt.pro
:: Bibliothek und Plugin/Beispiele bauen
mingw32-make -j8
:: Biblithek installieren
mingw32-make install
```

Die `-j8` sind für das parallele Bauen auf 8 CPUs.

Sofern nicht in der Datei `qwtconfig.pri` ein anderer Installationspräfix in der Variable `QWT_INSTALL_PREFIX` eingestellt wurde, ist die Bibliothek nun unter `c:\Qwt-6.3.0` installiert:

```
c:\Qwt-6.3.0\include  - Header-Dateien
c:\Qwt-6.3.0\lib      - Bibliothek/DLLs
c:\Qwt-6.3.0\doc\html - API Dokumentation (`index.html` in diesem Verzeichnis öffnen)
```

6.2.3. Linux/Mac

6.3. Qt Designer Plugins

i. wie erstellt man die Designerplugins und bekommt die in die Komponentenpalette…

6.4. Verwendung des Plots in eigenen Programmen

6.4.1. Windows

6.4.2. Linux/Mac

7. Exportieren und Drucken

7.1. Exportieren des Plots als Pixelgrafik

7.2. Exportieren des Plots als Vektorgrafik

7.3. Drucken