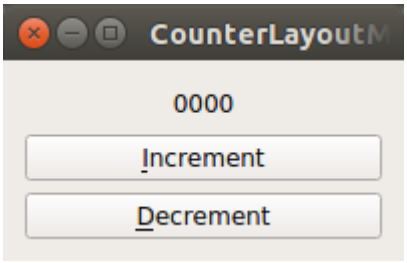
Thema, Ziele: Signal & Slots, eigene Widgets

Aufgabe 1: Qt Counter: Layout Manager



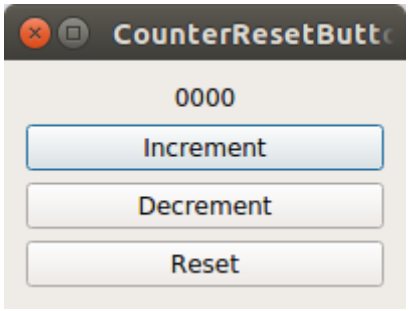
Auf dem Skriptserver, im Unterverzeichnis "Vorlage" finden Sie die Klasse Counter (counter.cpp und counter.h). Erstellen Sie ein Qt-Programm mit dem ein Zähler realisiert wird. Sie können dazu die Counter Klasse verwenden.

Verbinden Sie die Signale `clicked` der beiden Buttons mit den Counter Slots (Memberfunktionen) `incValue` resp. `decValue`.

Die beiden Buttons sollen nun so definiert werden, dass sie auch mit den Tasten <Alt>+I und <Alt>+D (sogenannte "Shortcut's") bedient werden können. – Anleitung: Schreiben Sie ein &-Zeichen beim Buttontext vor den Shortcut-Buchstaben. Also "&Increment" und "&Decrement". Probieren Sie andere Buchstaben.

Hinweis: Ausserhalb eines QObjects muss die `connect`-Methode über den QObject Klassenscope aufgerufen werden → `QObject::connect(…)`

Aufgabe 2: Qt Counter: Reset-Button

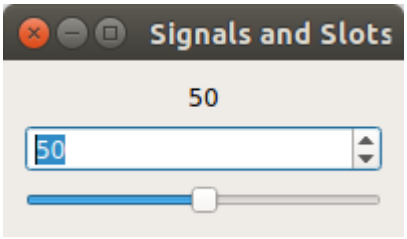


Entwerfen Sie eine Counter Applikation mit GUI mittels QtCreator. Zeichnen Sie das GUI mit dem eingebauten GUI-Designer. Verwenden Sie wiederum die Counter Klasse aus dem Vorlageordner auf dem Skripteserver.

Verbinden Sie auch wieder die entsprechenden Signale mit den dazugehörigen Slots (inklusive Reset-Button).

Das Programm soll in etwa so wie rechts dargestellt aussehen.

Aufgabe 3: Signals and Slots



Erstellen Sie mit dem Qt-Creator und GUI-Designer das rechts dargestellte GUI. Falls der Wert beim QSpinBox-Objekt mit Hilfe der kleinen "Up-Down-Buttons" geändert wird oder den Schieberegler betätigt wird, sollen alle angezeigten Werte (inklusive Schieberegler) dieser Änderung folgen. Verbinden Sie die benötigten Signale mit den entsprechenden Slots (Memberfunktionen).

Hinweise:

Wird beim QSlider (Schieberegler) mit der Maus der Regler verschoben, sendet das QSlider-Objekt das Signal "`void valueChanged(int x)`" aus. Um andererseits die Schiebereglerposition "von aussen" zu setzen, gibt die Memberfunktion (Slot) "`void setValue(int x)`". Der Wertebereich des Sliders ist 0..99.

Beim Anklicken der (kleinen) "Up-Down-Buttons" beim QSpinBox-Objekt in der Mitte, wird von diesem Objekt das Signal "`void valueChanged(int x)`" ausgesendet. Zum Setzen des angezeigten Wertes gibt es bei dieser Klasse den Slot (Memberfunktion) " `void setValue(int x)`". Diese Signals und Slots heissen also gleich wie beim QSlider. Der Wertebereich der QSpinBox ist 0..99.

Erweiterung "QMessageBox"

Das QLabel- QSpinBox- und QSlider-Objekt zeigen bekanntlich immer den gleichen Wert im Bereich 0..99 an. Beim Start des Programms soll dieser Wert nun 40 sein.

Wenn beim Verstellen dieser Wert nun >= 80 oder <= 20 wird, soll ein Dialogfenster mit der Fehlermeldung "Wert ausserhalb Bereich!" erscheinen. Nach dem "Wegklicken" dieser Fehlermeldung soll dann der Wert auf 50 gesetzt werden.

Erweitern Sie ihr Programm entsprechend.

Hinweis: Verwenden Sie `QMessageBox::warning( this, " ", "Wert ausserhalb Bereich!");`

Aufgabe 4: Callback in C

Die in der Vorlesung gezeigte Callback Version für C ist nur für das Registrieren eines Clients gedacht. Der entsprechende Code finden Sie im Vorlagen Ordner unter *CCallback*. Als Erweiterung sollen sich mehrere Clients registrieren und die Registrierung löschen können (*unregister*). Erweitern Sie den Code entsprechend, damit sich neben dem Modul a auch das Modul a2 registrieren können und die Registrierung wieder löschen kann.
Der folgende Code soll zum Test verwendet werden:

```
int main (void)
{
  aInit();
  a2Init();

  bFooX(); /* fire event */

  printf("aFoo should be called now\n");
  printf("a2Foo should be called now\n");

  aClear();

  bFooX(); /* fire event */
  printf("a2Foo should be called now\n");

  a2Clear();
  aInit();

  bFooX(); /* fire event */
  printf("aFoo should be called now\n");

  return 0;
}
```

Aufgabe 4: Observer mit C++

Ähnlich wie in der vorherigen Aufgabe soll die Observer C++ Variante 3 der Vorlesung für mehrere Observer erweitert werden. Im Vorlage Ordner finden Sie den Code unter *CppObserver*. Erweitern Sie den Code, damit sich mehrere Objekte der Klasse A bei der Klasse B registrieren können.

```
int main()
{
  B b;
  {
    A a1(b);
    A a2(b);

    // fire event
    b.fooX();

    std::cout << "A::foo should be called twice" << std::endl;
  }
  {
    A a2(b);

    // fire event
    b.fooX();

    std::cout << "A::foo should be called once" << std::endl;
  }

  return 0;
}
```