Inhalt

[Widget 2](#_Toc115731874)

[Beispiel für eine Aufteilung 2](#_Toc115731875)

[Neues Widget erstellen 3](#_Toc115731876)

[Formularklasse auswählen 3](#_Toc115731877)

[Namensvergabe 4](#_Toc115731878)

[Dateien richtig verschieben 4](#_Toc115731879)

[Dateiverknüpfung im Projekt bereinigen 5](#_Toc115731880)

[Programmieren im Team 6](#_Toc115731881)

[Aufgabenverteilung 6](#_Toc115731882)

[Koordination 6](#_Toc115731883)

[Schnittstellen 6](#_Toc115731884)

[Klassenname 6](#_Toc115731885)

[Globale Funktionen 6](#_Toc115731886)

[Protected members/functions 6](#_Toc115731887)

[Klassen erstellen im QT-Creator 8](#_Toc115731888)

[Hinzufügen in neue Ordnerstruktur 8](#_Toc115731889)

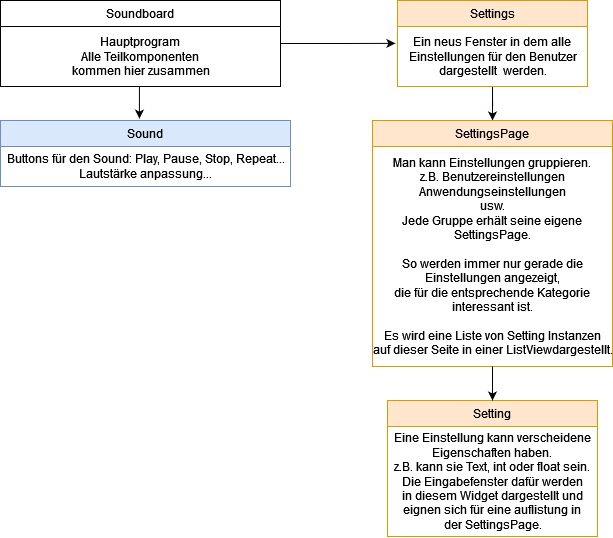
[Hinzufügen in bestehende Ordnerstruktur 9](#_Toc115731890)

[Erste Bearbeitung 10](#_Toc115731891)

# Widget

Jedes grafische Element auf der Benutzeroberfläche ist ein Widget. Widgets können auch selbst erstellt und programmiert werden. Dies macht dann sinn, wenn man von einem bestimmten Design mehrere Instanzen in der Anwendung haben möchte. Die Aufteilung ist abhängig von den Anforderungen der Anwendung, des Anwenders und des Entwicklers. Es gibt nicht die eine richtige Lösung, jedoch gibt es bessere und schlechtere Lösungen.  
Eine gute Aufteilung im Vorhinein wird einen riesigen umbauaufwand später verhindern, darum spielt die Aufteilung eine grosse Rolle.

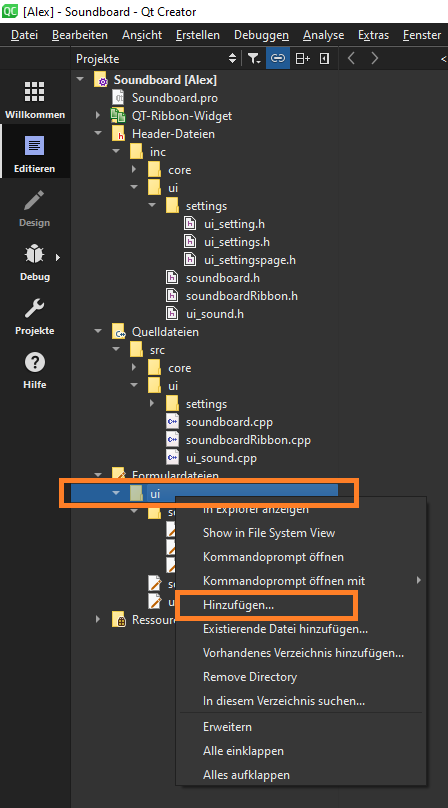
## Beispiel für eine Aufteilung



Jeder Kasten stellt eine Widget Klasse dar, so wie sie bereits im Projekt vorhanden ist.  
Orange sind alle Widgets, die mit Einstellungen zu tun haben und blau ist das Widget, welches die Buttons um den Sound abspielen zu können beinhaltet.  
Die Aufteilung ist so gewählt um möglichst modular sein zu können. Es wird nur eine «Settings» Instanz geben, jedoch kann es mehrere «SettingsPage» Instanzen geben welche wiederum mehrere «Setting» Instanzen haben können.

Bei der Aufteilung sollte man daher ein wenig überlegen welche Teile des Problems von einander getrennt behandelt werden können weil sie vielleicht an anderer Stelle genau so wieder verwendet werden können.

## Neues Widget erstellen

Für ein Widget werden drei Dateien benötigt:   
.h, .cpp und .ui  
Die .ui Datei beinhaltet die Layout Informationen und grafischen Elemente.  
Die .h und .cpp stellen die Klasse für dieses Widget dar. In dieser Klasse kann man auf alle Elemente auf dem Widget zugreifen.

Ein Widget wird erstellt, in dem man rechtsklick auf einen Projektodner->Hinzufügen klickt.

Es öffnet sich ein Auswahlfenster.

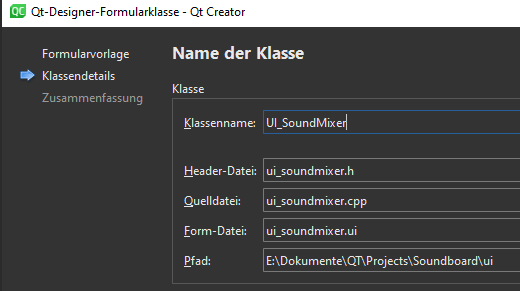
### Formularklasse auswählen

Wähle QT->QT-Designer-Formularklasse  
und klicke auf «Choose».

Es öffnet sich ein neues Auswahlfenster.  
Wähle «Widget» aus. Und dann auf «Weiter»

### Namensvergabe

Als Klassenname gebe einen sinnvollen Namen:  
Was macht die klasse? Was ist ihre Aufgabe?

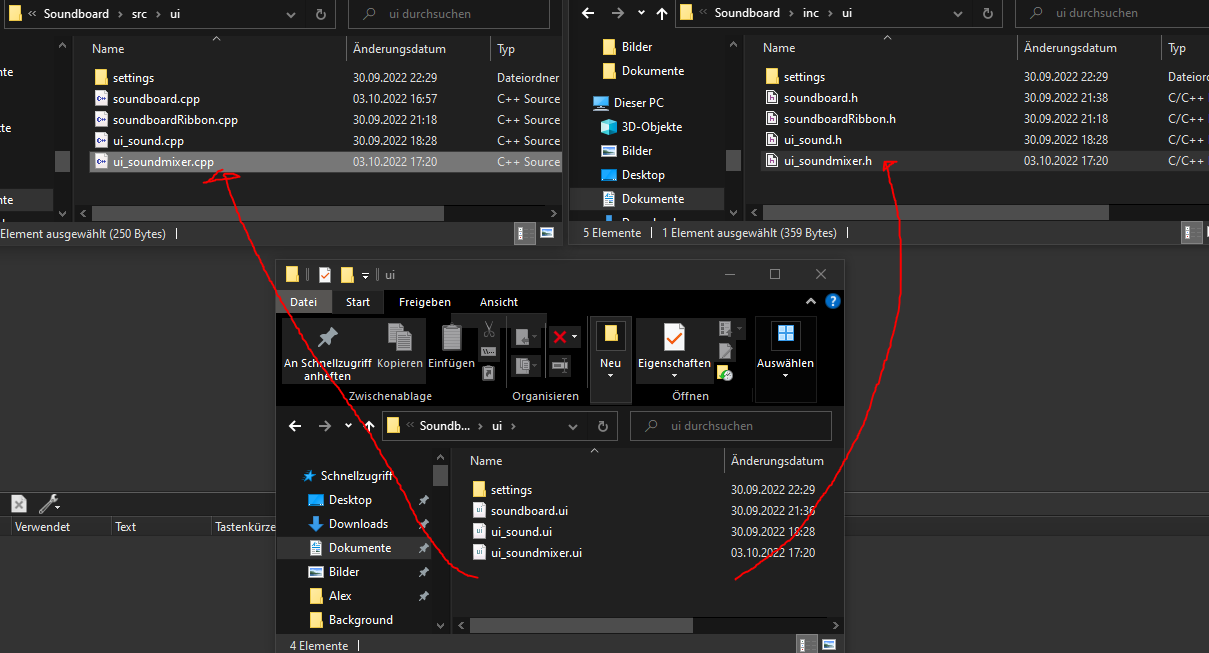
  
Der Name soll mit «UI\_» beginnen.   
z.B. «UI\_SoundMixer»  
Das UI\_ macht es einfacher zu erkennen, dass es sich um UserInterface Angelegenheiten handelt.  
Zudem soll der Anfangsbuchstabe der eigentlichen Klassenbezeichnung auch gross sein.

Die Namen der Dateien sollen so belassen werden.

Der Speicherort soll sein: «Soundboard\ui», alternativ kann man auch ein Unterordner machen falls dies sinn macht. z.B. sind alle UI-Elemente, die mit «Settings» zu tun haben im Unterordner «settings».  
Danach klicke auf «Weiter» und dann auf «Abschliessen»

Da unser Projekt eine andere Dateistruktur hat, müssen die .h und .cpp Dateien noch verschoben werden.

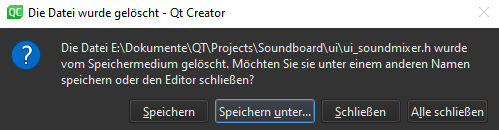
### Dateien richtig verschieben



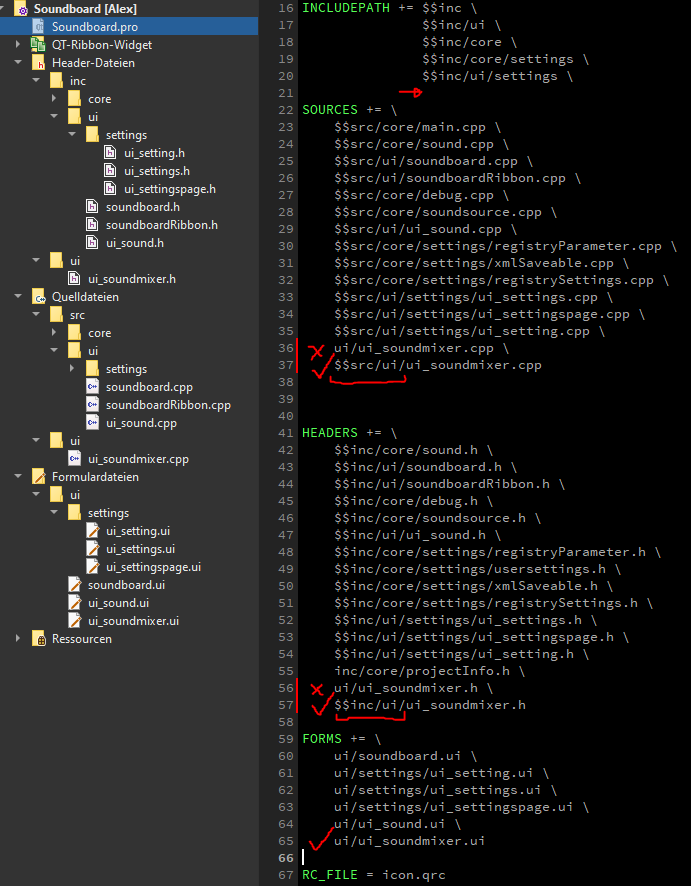
Ziehe die .h Datei in «Soundboard\inc\ui». Falls das .ui File noch in einem Unterordner ist, wird auch ein gleicher Unterordner im «inc\ui» erstellt und die .h Datei dort eingefügt. Siehe settings.

Das gleiche mit der .cpp Datei im Pfad «Soundboard\src\ui».

### Dateiverknüpfung im Projekt bereinigen



Zurück im QT-Creator wird dieses Popup erscheinen, weil die Dateien verschoben wurden. Klicke auf «Alle schliessen» Gespeichert waren sie bereits zuvor.



Öffne die Datei «Soundboard.pro»

Falls du neuen Ordner für die .h Dateien erstellt hast musst du diesen Ordner unter:

«INCLUDEPATH +=» eintragen.

Ändere den Pfad für das .cpp File wie im Bild und lösche den alten Eintrag heraus (Dieser mit dem   
Kreutz davor).

Das Gleiche mit der .h Datei.

Das **\** hinter den Zeilen wird benötigt falls in der Folgezeile noch ein Eintrag erfolgt.

Wenn diese Anpassungen vorgenommen sind speichere die Datei und nach einer kurzen Zeit werden die Verschobenen Dateien in der Projektstruktur auf der linken Seite sichtbar werden.

Du kannst nun beginnen die .ui Datei zu bearbeiten und den dazugehörigen Code in den .h und .cpp Dateien.

# Programmieren im Team

Wenn man alleine ein Projekt bewältigen muss, kann man es so machen wie man will, niemand spricht dazwischen. Man kann sich die Struktur des Projekts und selbst ausdenken und es genügt oft, diese nur im Kopf zu haben. In einer Gruppe ist das aber unmöglich, daher ist es sehr wichtig, dass alle von der gleichen Struktur reden um das gemeinsame Ziel erreichen zu können.

## Aufgabenverteilung

Das Projekt kann in viele Einzelteile aufgeteilt werden, welche für sich meist individuell gelöst werden können. Jeder bekommt deshalb seinen Teilbereich zugeteilt.   
Es ist nicht gestattet, im Code von jemand anderem herum zu pfuschen. Falls etwas nicht funktioniert oder sich nicht so verhält wie gewünscht muss das Problem der verantwortlichen Person gemeldet werden. Mit GitHub kann man den Fehler auch beheben und als mögliche Korrektur vorschlagen.

## Koordination

Es ist sehr wichtig, dass allen klar ist, was sie zu tun haben, welche Anforderungen erfüllt werden sollen und welche Voraussetzungen dafür erfüllt sein müssen. Natürlich kann man den eigenen Bereich so umsetzen wie man es möchte, solange die Schnittstellen nicht verletzt werden und solange es im Gesamtkonzept noch funktioniert.

Wenn du an einem Modul arbeitest, dass von einem anderen Modul abhängt ist es wichtig, dass die beiden Entwickler die Schnittstelle genau definieren.

## Schnittstellen

Die Schnittstelle ist das Bindeglied von allen Teilen des Projektes. Es ist wichtig, dass die Schnittstelle eines Modules früh definiert wird. Denn dadurch können nach der Definition beide Parteien beginnen ihre Module zu programmieren ohne, dass sie den code des anderen Entwicklers sehen zu müssen.

Eine Schnittstelle ist z.B. der Header einer Klasse (.h -Datei).  
Welche Informationen der Schnittstelle sind wichtig?:

### Klassenname

Die Benennung der Klasse ist wichtig, damit man weiss mit welchen Objekttypen man zu tun hat.

### Globale Funktionen

#### Konstruktor

Wie muss ich das Objekt instanziieren, welche Parameter werden dafür benötigt?

#### Sonstige

Welche Funktionen stehen mir zur Verfügung?  
Was kann ich am Objekt einstellen?  
Welche Werte kann ich aus dem Objekt auslesen?

### Protected members/functions

Welche Funktionen stehen mir zur Verfügung, wenn ich vom Objekt erbe?  
Auf welche Variablen habe ich bei einer Vererbung Zugriff?

Brauchst du von deinem Partner noch zusätzlich Informationen über sein Modul, so kann man das untereinander bei der Definition der Schnittstelle berücksichtigen. Grundsätzlich versucht man sein Modul so flexibel wie möglich zu gestalten.

Eine Klasse für ein Mathematischen Vektor sollte z.B. nicht nur die Addition beinhalten, nur weil man zurzeit nicht mehr benötigt.   
Man sollte möglichst alle Funktionen, die mit dem Vektor zu tun haben im Vektor implementieren.

* Skalarprodukt
* Vektorprodukt
* Skalierung
* Addition & Subtraktion
* Rotation um den Ursprung oder um einen Punkt
* Winkelberechnung des Vektors
* Definieren des Vektors anhand Polar- und Kartesische-koordinaten.
* Längenberechnung des Vektors
* …

Alle Aufgaben gehören zum Vektor und durch das einbinden dieser Funktionen in der Schnittstelle kann das Skalarprodukt später im Code bereits verwendet werden, auch wenn der Entwickler des Vektors die Implementierung für diese Funktion noch nicht gemacht hat. Sobald er diese aber hat, kann der code auch getestet werden.

So können beide Entwickler unabhängig arbeiten, da der Vektorentwickler die Schnittstelle festgelegt hat und klar definiert ist was der Vektor kann und wie man ihn benutzt.

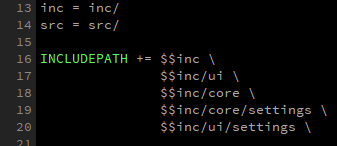
# Klassen erstellen im QT-Creator

## Hinzufügen in neue Ordnerstruktur

Die Quelldateien sind in einer Ordnerstruktur abgelegt, dabei sind .h und .cpp Dateien getrennt, jedoch sind beide Ordner Strukturen gleich aufgebaut.

**.h** Dateien gehören in de Header Ordner «**inc/**».  
**.cpp** Dateien gehören in den Quellenordner «**src/**»

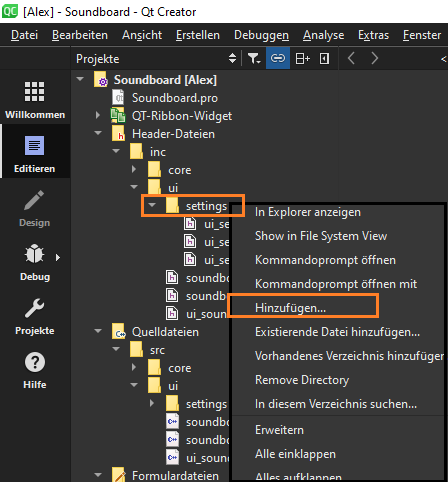
Willst du einen neuen Ordner erstellen, kannst du den leeren Ordner in beiden Verzeichnissen einfach von Hand erstellen.

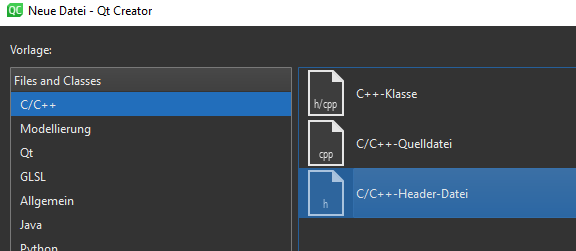


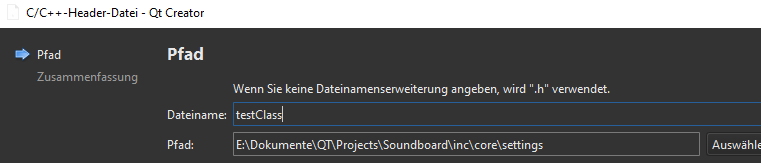
Wenn du beide Ordner erstellt hast, kannst du im Editor die Datei «**Soundboard.pro**» öffnen.  
Wir müssen dem Projekt sage, dass es nun einen neuen Ordner geben kann in dem man die Headerdateien finden kann.  
beim grünen Wort werden alle Ordner und Unterordner aufgelistet, die das Projekt beinhaltet.  
**$$inc** ist eine Variable und ist im Bild auf Zeile 13 definiert als «inc/»  
Alle Variablen haben in dieser Datei zwei $ davor um auf den Wert zu zu greifen.  
Das **\** am Ende der Zeile signalisiert, dass in der Auflistung noch ein weiteres Element folgt.  
Füge nun den neuen Pfad wie folgt ein:  
**$$inc/meinOrdner/evNochEinAndererOrdner \**

Im nächsten Abschnitt wird gezeigt, wie man nun eine Klasse in diesen Ordner erstellen kann. Achte darauf, dass der neue Ordner noch nicht in der Projektansicht auftaucht. Dies liegt daran, da sich noch keine Datei in diesem Projekt im Ordner befindet. Es ist daher am Anfang egal auf welchen Ordner du rechtsklickst, solange du am Schluss beim Fenster, welches nach dem Namen der Datei fragt, den richtigen Pfad angibst.

## Hinzufügen in bestehende Ordnerstruktur

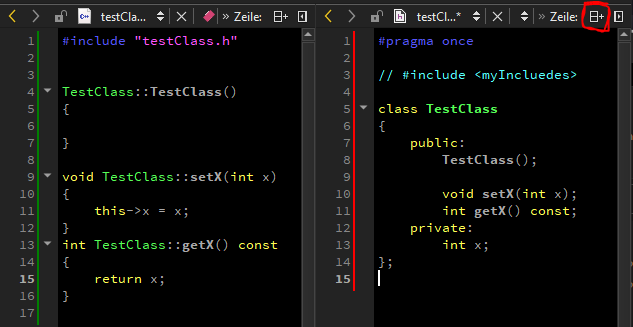
Eine Klasse beinhaltet immer zwei Dateien, .h und .cpp. Diese sind jedoch nicht im selben Ordner.  
Ich erstelle daher beide Dateien separat.  
Im Editor kann man einfach rechtsklick auf den Ordner, indem man die Datei erstellen möchte machen und dann auf «Hinzufügen» klicken.

Wähle C/C++ -> Header Datei, wenn du im «inc» Ordner etwas hinzufügen willst und Quelldatei, wenn du im «src» Ordner eine Datei erstellen möchtest.



Die Dateien müssen den gleichen Namen haben und sollten den Namen der Klasse haben, jedoch mit kleinem Anfangsbuchstaben. Stelle sicher, dass der Pfad auch stimmt, besonders dann, wenn du die Datei in einen neu erstellten Ordner einfügen möchtest, den du zuvor manuell erstellt hast und der darum nicht in der Auswahlliste war. Sobald sich eine eingebundene Datei im Ordner befindet, erscheint dieser auch in der Projektansicht.

## Erste Bearbeitung

Ich empfehle den Editor zu splitten, dies kannst du oben Rechts   
(rot eingerahmt machen).

Rechts habe ich immer die .h Datei und links das .cpp aber machs so wies dir besser passt.

Dann würde ich das

**#ifndef…  
#define …   
…  
#endif**   
löschen und durch ein   
**#pragma once**  
ersetzen. Es macht das gleiche, ist aber weniger Text.