Skriptum zum

Vortrag und den Übungen

über

Augmentend Reality mit Echtzeitdaten

von

Heinz PETERSCHOFSY

2020-1-DE02-KA202-007393

Krems und St. Pölten, 2023

Inhaltsverzeichnis

1 Arduino 1
1.1 Allgemeines 1
1.1.1 Was ist ein Arduino? 1
1.1.2 Die Arduino-Hardware 2
1.1.3 Einrichten der Software 3
1.2 Übungen 6
1.2.1 Upload-Test 6
1.2.2 Anschließen einer LED 9
1.2.3 Ansteuern eines digitalen Ausgangs 12
1.2.4 Verwenden eines Tasters 13
1.2.5 Einlesen eines digitalen Eingangs 14
1.2.6 Pulldown und Pullup 16
1.2.7 Interner Pullup-Widerstand 19
1.2.8 Speicherschaltung 19
1.2.9 Der Serielle Monitor 20
1.2.10 Zeitfunktion 1 (Ganglicht einfach) 23
1.2.11 Zeitfunktion 2 (Ganglicht komplex) 24
1.2.12 Personalisieren von Einstellungen 25
1.2.13 Speichern im nicht flüchtigen Speicher 28
1.2.14 Analoge Ausgaben 30
1.2.15 Analoge Eingänge 33
1.2.16 Der NTC als Temperatursensor 36
1.2.17 Helligkeitssensor 37
1.2.18 Abstandssensor 38
2 Virtual- und Augmented-Reality-Systeme 41
2.1 Allgemeines 41
2.1.1 Wo liegt der Unterschied zwischen VR und AR? 41
2.2 Virtual Reality 42
2.2.1 Anforderungen 42
2.2.2 Hardware 42
2.2.3 Software 43
2.2.4 Anwendungen im technischen Bereich 44
2.3 Augmented Reality 46
2.3.1 Anwendungen 46

This project has received funding from the European Union's Erasmus+ program under the registration number 2020-1-DE02-KA202-007393. This document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains

2.3.2 Herausforderungen 48
2.3.3 Zukunft 49
2.4 Vuforia Studio 50
2.4.1 Überblick 50
2.4.2 Funktionsprinzip 51
2.4.3 Erstes Projekt für ein Handheld-Device (Handy) 52
2.4.4 Mehrteiliges Modell für 2D-Device 61
2.4.5 Automatische Animation 70
2.4.6 Projekt für ein 3D-Device (Microsoft HoloLens) 76
2.4.7 User-Eingaben mit der Microsoft HoloLens 80
2.4.8 Weitere Eingabemöglichkeiten 84
3 Internet of Things 87
3.1 Allgemeines 87
3.2 Anforderungen 87
3.3 Anpassung unsers Arduino-Kits 87
3.4 Message Queuing Telemetry Transport (MQTT) 89
3.4.1 Kommunikation mit MQTT 89
3.4.2 Übertragen aus einem Arduino-Programm 97
3.5 Thingworx 101
3.5.1 Allgemeines 101
3.5.2 Anlegen unseres ersten Thingwork-Projekts 104
3.5.3 Schreiben/Lesen von Daten in das Thing 109
3.5.4 Verbindung zwischen Arduino und Thingworx 115
3.5.5 Versorgen des Things mit Live-Daten 119
3.5.6 Anzeigen von Daten in einem Mashup 121
3.5.7 Verwenden von Things in Vuforia-Experiences 129
4 Anhang 141
4.1 Arduino Referenz 141
4.1.1 Sprachstruktur 141
4.1.2 Befehle 144
4.1.3 Hinzufügen von Libraries 150
4.1.4 Wichtige Objekte 152
4.1.5 ASCII-Tabelle (Auszug) 154
4.2 Widerstandsfarbcode 155
4.3 Vuforia Studio Referenz 156

This project has received funding from the European Union's Erasmus+ program under the registration number 2020-1-DE02-KA202-007393. This document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains

4.3.1 Vorgehen bei der Projekterstellung in Vuforia Studio 156
4.3.2 Importieren eins Modells 156
4.3.3 Hinzufügen eines Modells 157
4.3.4 Ionicons -Codes für Symbole 158
4.4 MQTT-Broker Mosquitto auf Raspberry installieren 159
4.5 Die ESP-Programme für IoT 161
4.5.1 Befehle für den MQTT-Gateway 161
4.5.2 Befehle für den Thingworx-Gateway 164
4.5.3 Bibliotheken und deren Keywords 167
4.5.4 Ansprechend des ESP MQTT-Gateways 168
4.6 HTTP-Statuscodes 174

This project has received funding from the European Union's Erasmus+ program under the registration number 2020-1-DE02-KA202-007393. This document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains

1 Arduino

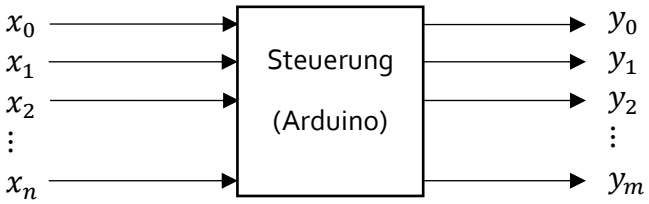
Playlist: https://www.youtube.com/playlist?list=PLVut1tKPvtvNNhOVwQmlc74szsikRsIDk

1.1 Allgemeines

1.1.1 Was ist ein Arduino?

Video-Link 1: https://youtu.be/o6HtUzjJP2c

„Arduino" bezeichnet einen Einplatinencontroller. Alle wichtigen Bauteile sind auf einem kleinen Stück Leiterplatte untergebracht. Von der Funktion her ist es eigentlich ganz einfach:



x_n ... Eingänge in die Steuerung (entweder digital oder analog)
y_m ... Ausgänge aus der Steuerung (entweder digital oder „PWM")

Ob und wann Eingänge eingelesen werden, bzw. ob und wann Ausgänge geschrieben werden und wie der Ablauf ist, wird mit einem „Programm" in der Steuerung hinterlegt. In diesem Programm wird also festgelegt, wie die Steuerung bei verschiedenen Zuständen reagieren soll (welche Ausgänge gesetzt werden). Mit den Ausgängen kann die Steuerung irgendetwas in der realen Welt beeinflussen.

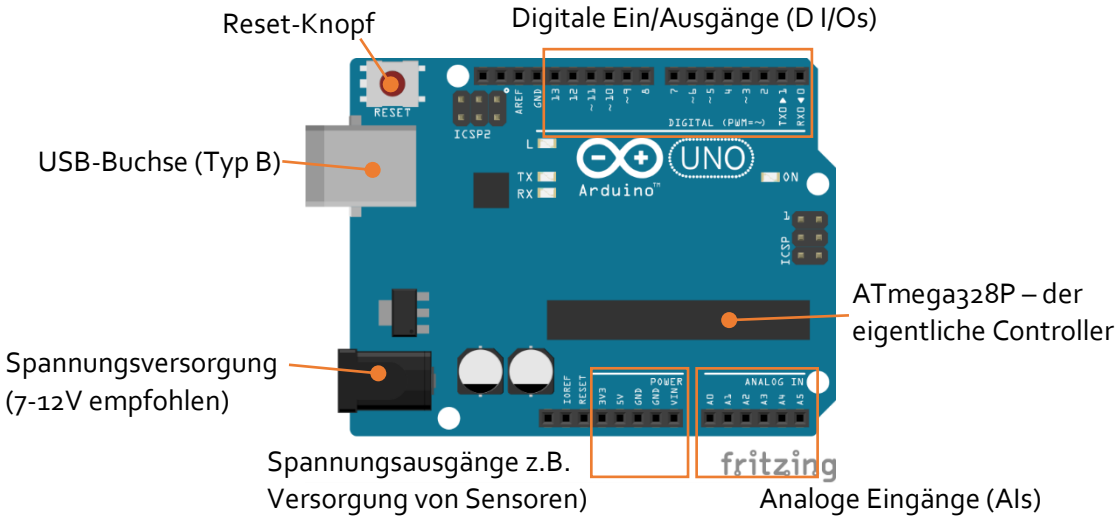
Unser Arduino soll also Dinge in der Umwelt einlesen und entsprechend eines Programms darauf reagieren können. In diesem Fall ist das „Programm" tatsächlich ein Stück Software, welches auf die Veränderung der Eingänge reagieren kann. Das Programm ist im Arduino in einem Speicher hinterlegt und austauschbar. Zum Austauschen benötigen wir nur einen Computer und schon können wir das Programm im Programmspeicher einfach überschreiben – nicht viel komplizierter als das Beschreiben eines USB-Flashdrives.

Unser Arduino hat also Ein- und Ausgänge und ein veränderbares Programm, welches in einem Speicher liegt. Im Wesentlichen handelt es sich dabei also um eine sogenannte „Speicherprogrammierbare Steuerung" („SPS").

Ausgehend von einfachen Dingen werden wir versuchen, alle Funktionen des Arduino kennenzulernen.

1.1.2 Die Arduino-Hardware

Wir verwenden dazu einen „Arduino UNO". Dieses Board ist für den Einstig in die Arduino-Welt gut geeignet. Man kann praktisch alle Dinge damit ausprobieren. Für komplexe Steuerungsaufgaben ist es jedoch zu „schwach" bzw. für kleine Lösungen etwas zu groß.



Microcontroller	ATmega328P	Praktisch der ganze Arduino ist in diesem Chip inklusive Speicher
Logiklevel	5V	5V bedeutet logisch „1" und 0V bedeutet logisch „0" (TTL[1]-Pegel)
Eingangsspannung	7-12V empfohlen 6-20V Maximum	Pegel der Spannungsversorgung, wird auch über die USB-Buchse spannungsversorgt.
Maximaler Strom pro I/O-Pin	20 mA	
Maximaler Strom bei 3,3V	50 mA	
Programmspeicher (Flash)	32 kB	Eingebaut in ATmega328P
Arbeitsspeicher (SRAM)	2 kB	Eingebaut in ATmega328P
EEPROM	1 kB	Eingebaut in ATmega328P
Taktrate	16 MHz	
Abmessungen (lxb)	68,6 mm x 53,4 mm	
Gewicht	25 g	

Details zum UNO sowie Informationen zu anderen Produkten können auch unter der Homepage von Arduino gefunden werden: www.arduino.cc

[1] TTL: Transistor-Transistor-Logic

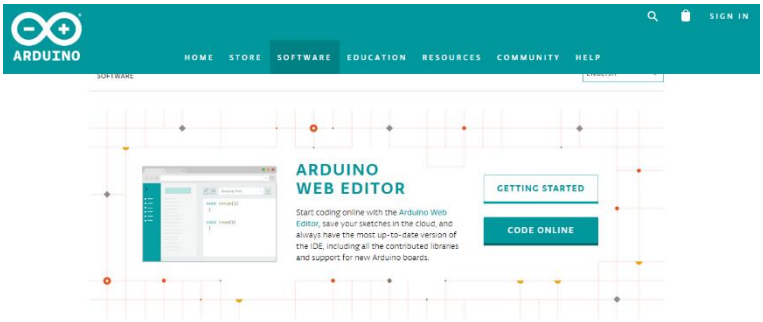
1.1.3 Einrichten der Software

Video-Link 2: https://youtu.be/o9lrs2I1CHc

1.1.3.1 Download

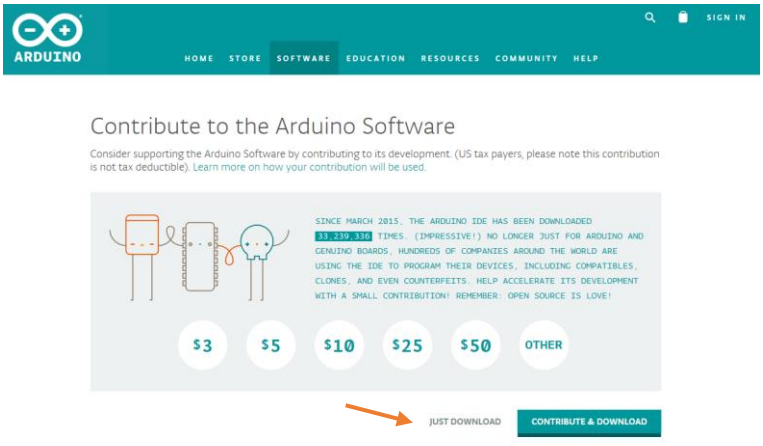
Die Programmiersoftware „Arduino IDE[2]" kann auf der Homepage von Arduino heruntergeladen werden:

Im Menü „Software" die Seite „Downloads" anwählen.



Auswahl des Windows „Installers"

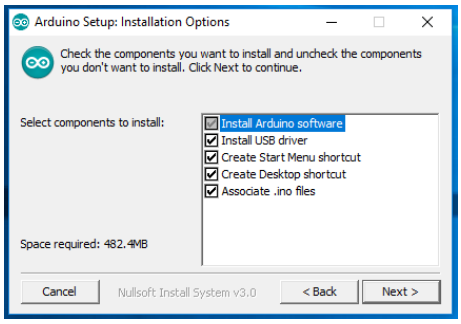
Wer will kann spenden, oder auf „Just download" klicken.



[2] IDE: Integrated Development Environment – Integrierte Entwicklungsumgebung: Kombiniert einen Text-Editor und die notwendigen Dinge, um das Programm zu übersetzen (Compiler) und auf den Arduino zu bringen (Loader).

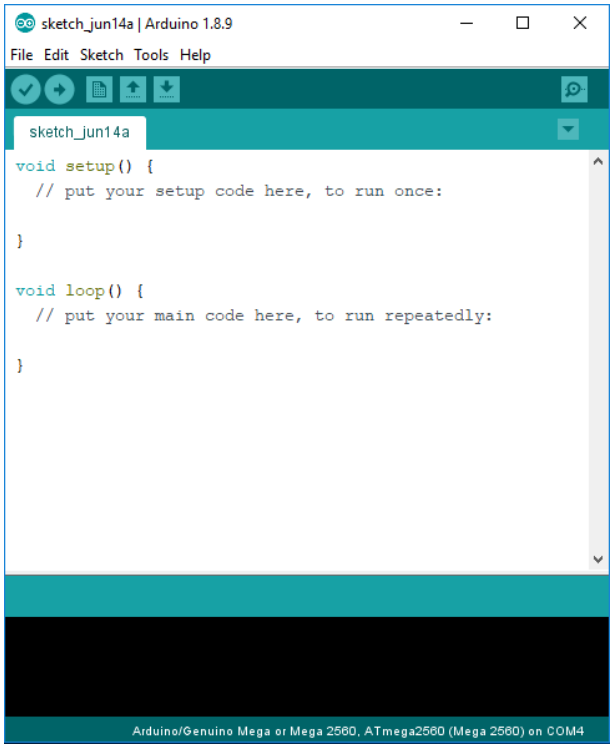
1.1.3.2 *Installieren*

Nach dem erfolgten Download kann die IDE installiert werden. Dazu aktiviere ich immer alle Optionen. Sollte jemand kein Icon am Desktop haben wollen oder *.INO-Files nicht automatisch mit der IDE öffnen wollen, dann bitte die entsprechenden Optionen abwählen.



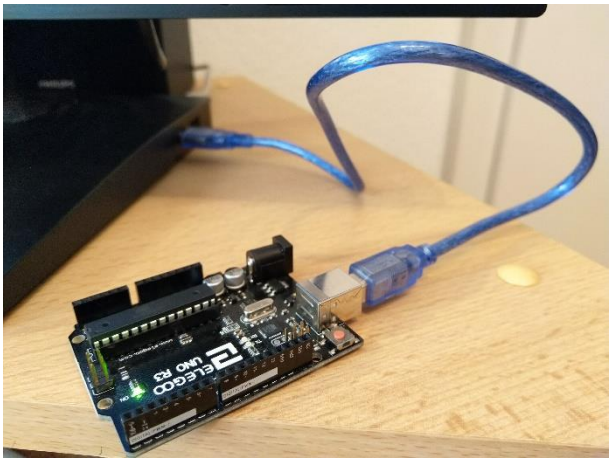
1.1.3.3 *Starten*

Nach erfolgter Installation kann die Arduino-IDE gestartet werden. Nach dem Start sollte es ungefähr so aussehen:



1.1.3.4 *Verbindungseinstellungen*

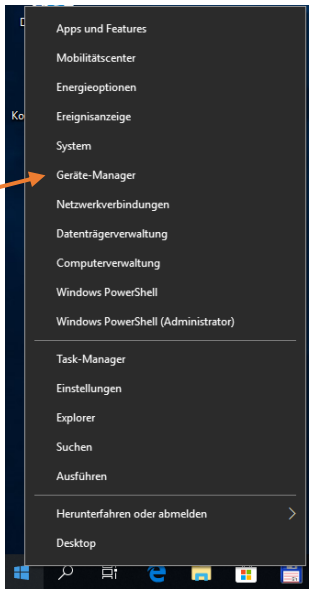
Video-Link 3: https://youtu.be/5M2wrfL5oPY

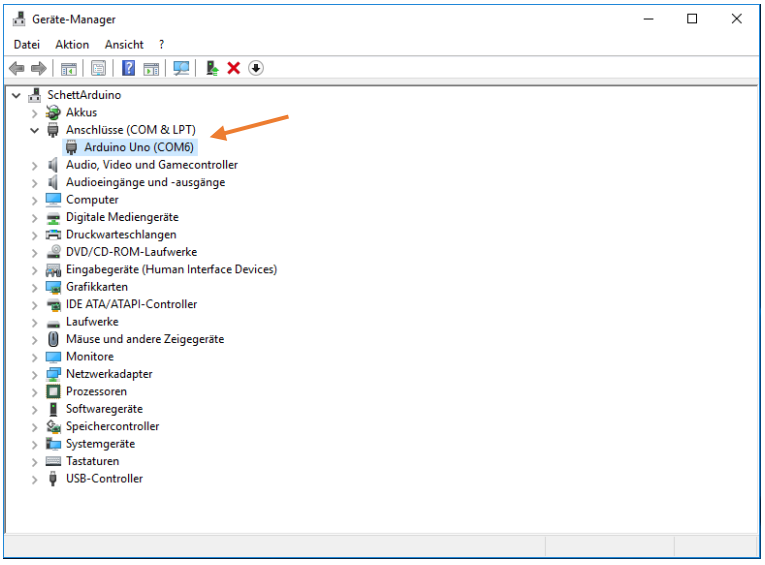


Wir haben nun also die IDE installiert. Jetzt müssen wir noch unseren Arduino und die IDE bekannt machen. Dazu müssen wir unser Board das erste Mal mit dem Computer verbinden. In dem Starter-Kit ist bereits ein passendes USB-Kabel enthalten. Deswegen sollte es kein Problem sein, den Arduino mit einem freien USB-Port am Computer zu verbinden.

Sofort leuchten die LEDs auf und der Arduino ist betriebsbereit. Im Besten aller Fälle wird auch gleich der passende Treiber am Computer geladen – zumindest sollte bei der Installation der IDEs der passende Treiber mit installiert worden sein. Am Computer finden wir, wenn alles stimmt, nun einen zusätzlichen COM-Port[3]. Wie kann jetzt herausgefunden, ob ein passender virtueller COM-Port angelegt wurde?

Bei Windows Computern kann der sogenannte „Geräte-Manager" aufgerufen werden. Dazu klickt man mit der RECHTEN Maustaste auf das Windows-Symbol (Startmenü). Es öffnet sich ein Auswahlmenü. Klickt man dabei auf „Geräte-Manger", öffnet sich dieser. Dort sollte man den COM-Port unter „COM&LPT" finden.





Praktischerweise steht dabei auch gleich die Nummer des COM-Ports dabei – im Beispiel ist es COM6. Genau diese Nummer muss der IDE mitgeteilt werden, denn woher soll das Programm sonst wissen, wo es „hinmuss"?

[3] COM: Communication Port: Ursprünglich eine serielle Schnittstelle, die auch als „Stecker" in realer Hardware da war (in Form einer meist 9-poligen Buchse). Mittlerweile wird für viele USB-Geräte ein „virtueller" COM-Port angelegt, der nur „da" ist, solange das Gerät angesteckt ist. Für Programme ist es dabei egal, ob sie auf einen tatsächlichen Hardware-COM-Port zugreifen, oder auf einen virtuellen COM-Port. Die Windows-Treiber erledigen die richtige „Übersetzung".

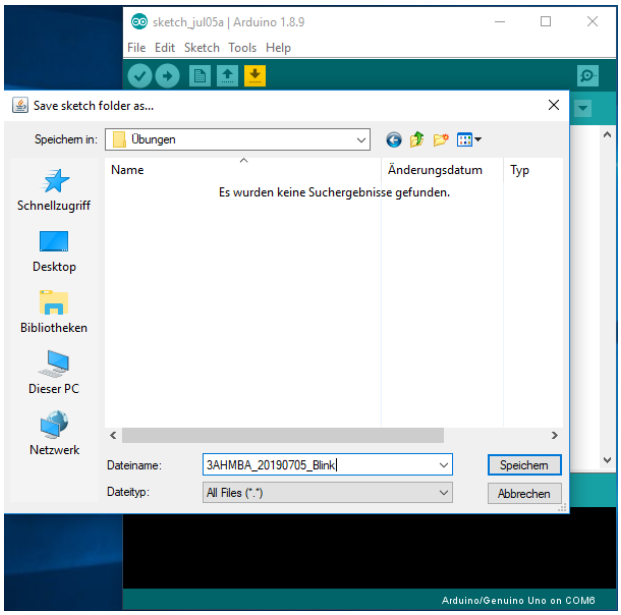
1.2 Übungen

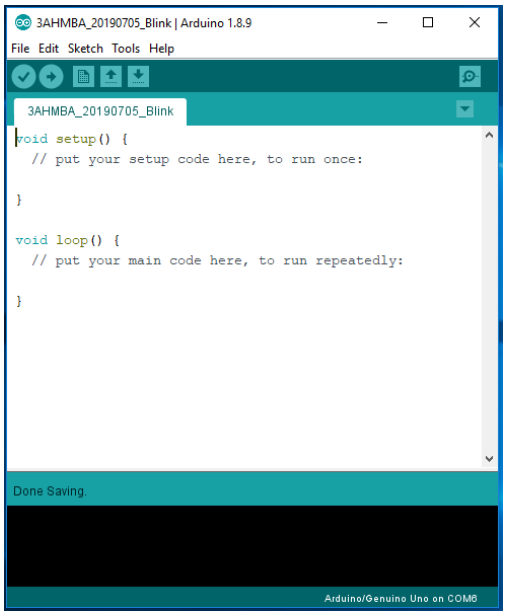
1.2.1 Upload-Test

Video-Link 4: https://youtu.be/VZI9ZnGPeN4

Nachdem wir nun unsere Software soweit eingerichtet haben, dass wir loslegen können, wollen wir auch nun gleich probieren ob wir denn ein Programm auf unseren Arduino bringen können („Upload").

Zunächst speichern wir unseren „Sketch" (das ist die Arduino-Bezeichnung für das Programm) unter einem anderen Namen. Wie üblich unter „File" und „Save as..." einen entsprechenden Platz aussuchen und der Datei einen Namen geben. Ich würde vorschlagen, ein gewisses Format anzuwenden. Im Beispiel sehen wir im Dateinamen die Klasse (3AHMBA), das Erstelldatum (20190705) und den Namen.





Die verwendete Programmiersprache ist praktisch C++. Eine kleine Referenz der Sprache findet sich im Anhang, oder auch auf www.arduino.cc.

Im Prinzip sehen wir jetzt zwei Standard-Funktionen:

`setup()`

Die Anweisungen in Setup werden nach einem Reset ausgeführt (also, wenn z.B. die Spannung angeschlossen wurde, oder auch wenn der Reset-Knopf gedrückt wurde, oder ein neues Programm geladen wurde). Dieser Programmteil läuft dann genau 1x und dann nicht mehr. Sinn dahinter ist es, irgendwelche Initialisierungen hier durchzuführen. Man kann hier z.B. festlegen, ob der Pin 5 ein Ein- oder Ausgang sein soll – aber davon später mehr.

`loop()`

Diese Anweisungen werden „immer wieder" ausgeführt. Immer wenn die Abarbeitung von `loop()` beendet wurde startet sie wieder von vorne – daher auch die Bezeichnung „loop" (Schleife). Sinn dahinter ist es, die Steuerungsanweisungen immer wieder durchlaufen zu lassen. Immer wieder alle notwendigen Eingänge abzufragen, und dann immer wieder die notwendigen Aktionen davon abzuleiten. Hier drinnen ist also die „Intelligenz" der Steuerung abgebildet.

Wir werden uns damit beschäftigen und es wird dann immer logischer für uns werden, was wir wohin schreiben müssen.

Was soll unser Programm jetzt machen? Wir wollen die eingebaute LED blinken lassen. In einem gewissen Rhythmus. Eigentlich geht es uns ja um den Upload und nicht um die Funktion.

Deswegen passen wir unser Programm jetzt entsprechend an:

```
#define LED_ONOFF_TIME 1000

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on
  delay(LED_ONOFF_TIME);              // wait for a second
  digitalWrite(LED_BUILTIN, LOW);     // turn the LED off
  delay(LED_ONOFF_TIME);              // wait for a second
}
```

Was passiert hier? Wir wollen eine LED auf und wieder abdrehen. Wir wollen also auf einen Ausgang schreiben. Nur auf welchen? Zum Glück bietet uns das IDE ein paar Kontanten an. Eine davon ist `LED_BUILTIN`. Da steht die Nummer des Ausgangs für die interne LED drinnen. Solche Konstanten kann man auch selbst definieren. Wir haben das in unserem Programm mit der Anweisung `#define` gemacht. Dort wird die Kontante `LED_ONOFF_TIME` definiert und auf einen Wert `1000` festgelegt – warum werden wir weiter unten sehen.

Sehen wir uns zunächst `setup()` an. Da gibt es nur eine einzelne Zeile:

`pinMode` ist dabei ein Befehl, der festlegt, was ein Ein- bzw. Ausgang sein soll. In diesem Fall legt er den PIN mit der Nummer `LED_BUILTIN` als Ausgang (`OUTPUT`) fest. Alle Optionen wären:

`OUTPUT`, `INPUT`, `INPUT_PULLUP`

Zu `INPUT` und `INPUT_PULLUP` später mehr.

Gut wir haben jetzt also einen Ausgang. Den müssen wir beschreiben, das machen wir periodisch in der Funktion `loop()`:

`digitalWrite` ist ein Befehl der einen Ausgang setzt (`+5V` darauf schaltet) oder ihn zurücksetzt (`0V` darauf schaltet). Man muss dem Befehl dabei die PIN-Nummer des Ausgangs sagen und ob er ihn setzen (`HIGH`) oder zurücksetzen (`LOW`) soll. Alternativ kann auch `1` (`HIGH`) oder `0` (`LOW`) geschrieben werden.

Die erste Zeile in `loop()` schaltet also die LED ein (setzen das Ausgangs auf `HIGH`). Die dritte Zeile schaltet die LED aus (setzen des Ausgangs auf `LOW`). Dazwischen gibt es noch die `delay`-Befehle:

Diese warten einfach. Als Argument muss man dem Befehl die Wartezeit in Millisekunden übergeben. In diesem Beispiel sind es `LED_ONOFF_TIME` ms (also 1000ms), also 1 Sekunde.

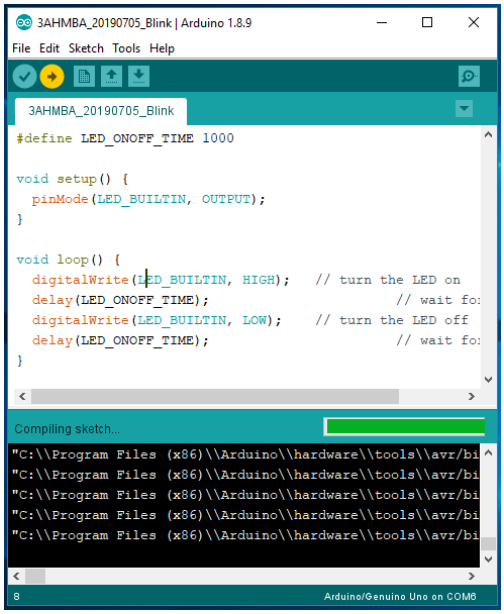
Die LED wird also aufgedreht, dann leuchtet sie eine Sekunde, um danach wieder abgedreht zu werden. Dann pausiert das Programm wieder 1 Sekunde und ist dann aus. Und weil `loop()` immer wiederholt wird, wird die LED auch wieder aufgedreht. Die LED blinkt also mit 0,5 Hz.

Jetzt geht es ans Uploaden. Es gilt nach wie vor eine uralte Regel:

„Save often, Save early!"

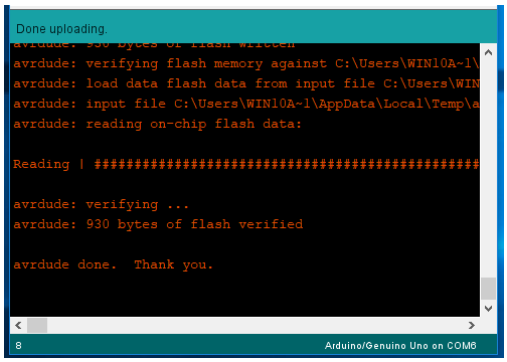
Deswegen vor dem Uploadversuch einen Klick auf das „Speichern"-Symbol ( ) in der Menü-Leiste. Danach kann man auf das „Upload" Symbol klicken. Dieses ist ein Pfeil, der aus irgendeinem Grund nach rechts zeigt. Wie wenn rechts oben heißen würde, aber sei's drum: Dort drücken wir drauf:

Was passiert jetzt? Der Compiler wird gestartet und der C++-Code wird in Maschinensprache übersetzt. Diese Übersetzung könnte man auch mit dem Haken-Symbol starten. Im Ausgabebereich unten im Fenster wird der momentane Status angezeigt.



Nach erfolgreichem Übersetzen startet der Upload automatisch. Es wird versucht, eine Verbindung zum Arduino aufzubauen, und den generierten Maschinencode auf den Flash-Speicher unseres Controllers zu kopieren.

Nach erfolgreicher Übertragung erhält man eine Fertigmeldung im Ausgabebereich:



Auch Fehlermeldungen werden im Ausgabebereich ausgegeben. Damit kann man erkennen, wo der Fehler liegt.

Im besten aller Fälle blinkt die eingebaute LED im 2-Sekunden-Takt – alles korrekt.

1.2.1.1 *Übungsaufgaben „Blink Intern"*

	Erstellen Sie so wie oben beschrieben ein Programm, welches die LED auf dem Arduino ansteuert, laden sie es auf ihren Arduino und zeigen Sie es vor.	

1.2.1.2 *Pflichtaufgaben „Blink Intern"*

	Modifizieren Sie das Programm so, dass die LED in einem anderen, asymmetrischen Rhythmus blinkt.	

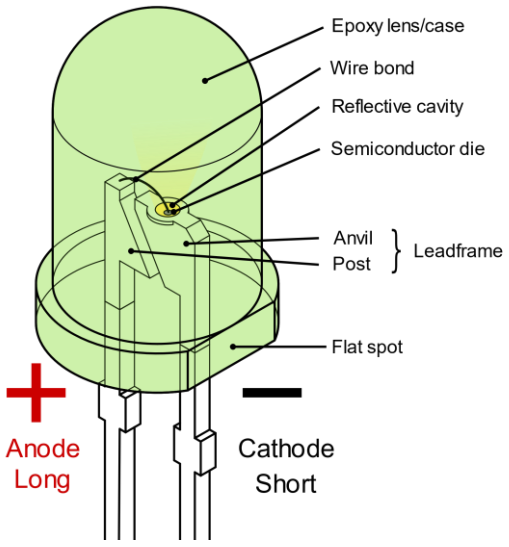
1.2.2 Anschließen einer LED

Video-Link 5 (Breadboard): https://youtu.be/EAAMMlay4yU Video-Link 6 (LED): https://youtu.be/G6ol3hucFbI

Wir wollen mit unserem Arduino echte Ausgänge ansteuern. Eine gute Methode zu erkennen, ob ein Ausgang angesteuert ist oder nicht, ist den Ausgang mit einer Leuchtdiode oder LED[4] zu bestücken. Dann leuchtet diese auf, wenn der Ausgang angesteuert wird.

Doch wie schließt man so etwas an? Sehen wir uns die Funktionsweise einer LED einmal an:

Prinzipiell gilt: Je mehr Spannung an einer LED liegt, desto mehr Strom rinnt durch die LED. Je mehr Strom durch die LED liegt desto heller leuchtet die LED. So weit so logisch. Das Problem ist dabei die Kennlinie der LED: Wenn man dann die Spannung in die falsche Richtung anlegt, dann passiert nie etwas - kein Strom, kein Leuchten. Deswegen müssen wir auf die Polarität achten – wo schließen wir Plus und wo Minus an? Die Anschlüsse sind dabei gekennzeichnet: Der +-Anschluss (Anode) ist etwas länger. Der –-Anschluss (Kathode) ist kürzer und hat am Gehäuse eine Abflachung (Eselsbrücke: ein Minus ist „flacher" als ein Plus-Zeichen oder „k" wie kurz ist „K" wie Kathode):



[5]

Und wenn man die Spannung in der richtigen Richtung anlegt, dann passiert bei kleinen Spannungen fast nichts. Bei „passenden" Spannungen leuchtet die LED gut und bei etwas erhöhter Spannung rinnt gleich so viel Strom, dass die LED zerstört wird. Das geht dann relativ flott, weil eine

[4] LED: Light Emitting Diode (Licht emittierende Diode)

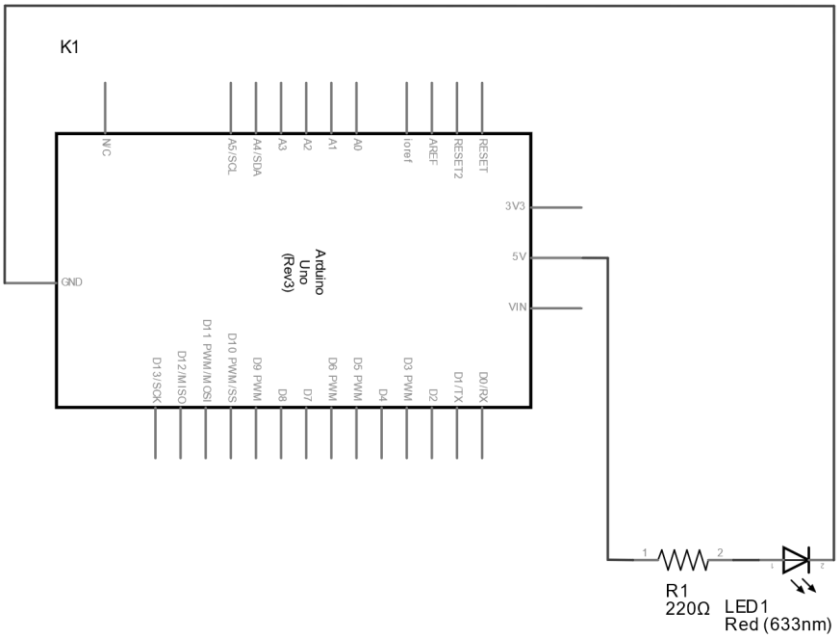
[5] Quelle: Von Inductiveload - Own work by uploader, drawn in Solid Edge and Inkscape., Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=6431789

LED (wie alle Dioden) bei einer bestimmten „Flussspannung" praktisch plötzlich anfangen den Strom zu leiten. Wenn wir also keine passende Spannung zur Verfügung haben, dann ist die LED gleich zerstört.

Zu allem Überfluss haben rote, grüne, blaue und weiße LEDs alle unterschiedliche Flussspannungen. Auch haben zwei LEDs derselben Farbe unterschiedliche Werte (Fertigungsstreuung). Das macht das alles nicht einfacher. Man benötigt also etwas, was den Strom passend begrenzt.

Im einfachsten Fall ist das praktisch eine „Engstelle" für den Strom, der dafür sorgt, dass der Stromfluss gedrosselt wird – wie in einer Wasserleitung. Solch eine Engstelle heißt „Widerstand". Man spricht davon, eine LED an einem „Vorwiderstand" zu betrieben. Der Vorwiderstand ist davon abhängig, wieviel Spannung insgesamt zur Verfügung steht. Je mehr Spannung, desto mehr muss gedrosselt werden. Bei der uns zur Verfügung stehenden Spannung von $U_{TTL} = 5V$ und den verwendeten Leuchtdioden ergibt sich ein Vorwiderstand von $R = 220\Omega$.

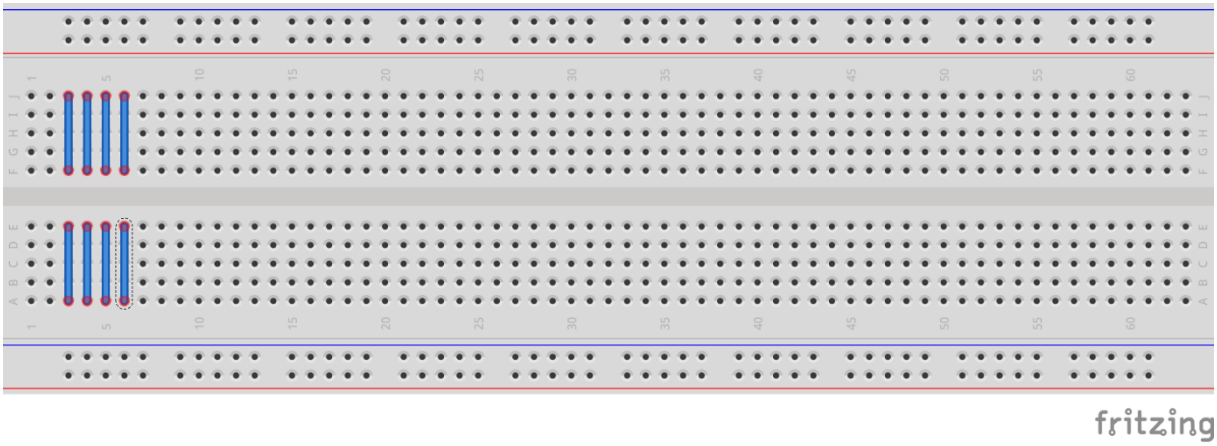
Wir sollen also so etwas aufbauen:



Wie erkennt man eigentlich den Widerstandswert? Das sind doch alles runde Dinger mit zwei Haxen ohne Zahl. Das Geheimnis ist der Farbcode. Jeder axial bedrahtete Widerstand trägt einen Farbcode in Form von bunten Ringen. Dadurch kann er von allen Seiten abgelesen werden, ohne dass es darauf ankommt, ob die Schrift gerade zufällig oben ist. Der Farbcode ist nach Tabellen aufgebaut (siehe Anhang).

Am besten verwenden wir dafür unser „Breadboard". Dieses Brotbrett ist für noch nicht ganz ausgebackene Schaltungen und zum Verifizieren von Schaltungsvarianten. Man kann dort Schaltungen ganz ohne Werkzeuge zusammenstellen und ausprobieren – ideal für uns. Wir werden das also verwenden, um unsere Schaltungen aufzubauen.

Breadboards, oder auch „Steckbretter" haben eine Reihe an Löchern. Manche dieser Löcher sind miteinander verbunden, andere gegeneinander isoliert. Auf dem nächsten Bild erkennt man, wie die Löcher miteinander verbunden sind.

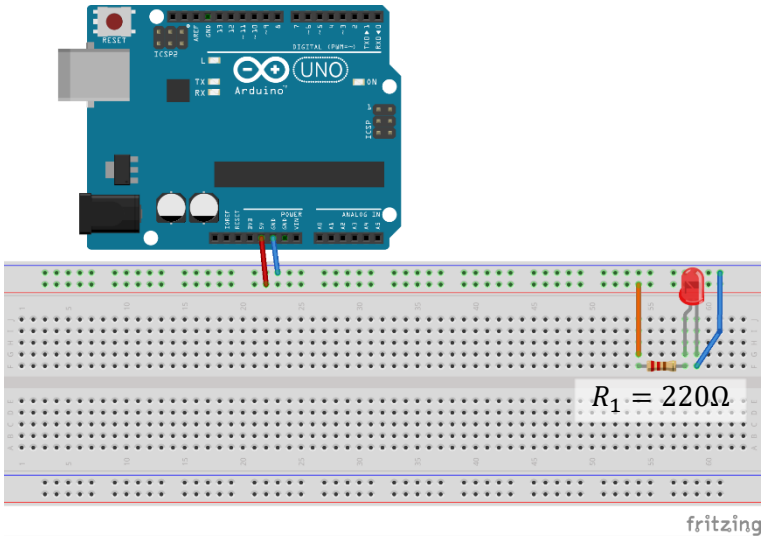


Die äußeren Reihen sind mit Rot und Blau gekennzeichnet. Diese sind durchkontaktiert. Jede Reihe für sich. Diese sollen dazu dienen, um eine Spannungsversorgung überall auf das Brett zu bringen.

Im mittleren Bereich gibt es viele Löcher, wobei alle, die untereinander liegen, miteinander verbunden sind. Im Bild ist das mit ein paar blauen Linien herausgehoben – es sind jedoch alle Reihen gleich aufgebaut.

In diese Löcher passen die „Füßchen" unserer elektronischen Bauelemente genau hinein. Dem genormten Rastermaß von $1/10\ inch$ $(2{,}54mm)$ sei Dank.

Damit lassen wir jetzt unsere LED leuchten, indem wir folgende Schaltung aufbauen:



Sobald der Arduino stromversorgt ist (z.B. über die USB-Buchse) sollte die LED leuchten.

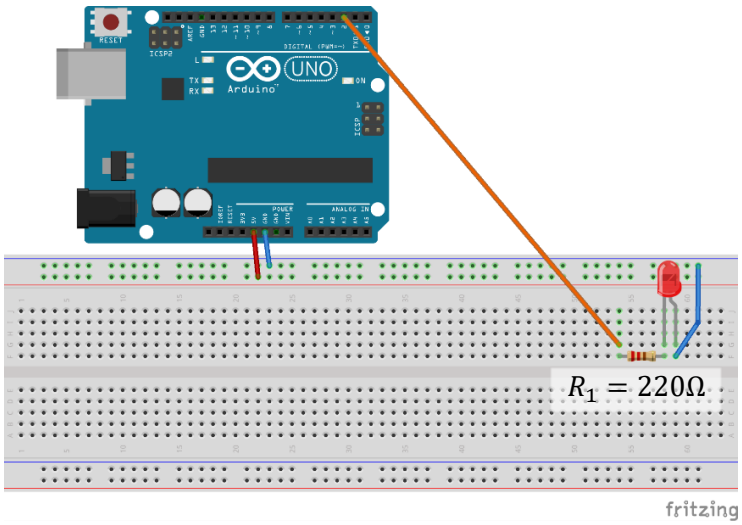
1.2.2.1 *Übungsaufgabe „LED connect"*

	Bauen Sie die oben beschriebene Schaltung nach, so, dass die LED leuchtet und nichts „abbrennt".	

1.2.3 Ansteuern eines digitalen Ausgangs

Video-Link 7: https://youtu.be/7Kllis44IN8

Anstatt die LED immer leuchten zu lassen, wollen wir sie mit der Steuerung ein- und wieder ausschalten. Dazu „verheiraten" wir die vorigen zwei Übungen. Zunächst bauen wir die Schaltung um. Es sollte so aussehen:



Und aus der ersten Lektion nehmen wir das Programm, speichern es unter einem anderen Namen und ändern es ab, dass nicht die interne LED angesteuert wird, sondern die externe LED:

```
#define LED_ONOFF_TIME 1000
#define LED_PIN 2

void setup() {
  pinMode(LED_PIN, OUTPUT);       // define LED_PIN as output
}

void loop() {
  digitalWrite(LED_PIN, HIGH);    // turn the LED on
  delay(LED_ONOFF_TIME);          // wait for a second
  digitalWrite(LED_PIN, LOW);     // turn the LED off
  delay(LED_ONOFF_TIME);          // wait for a second
}
```

Man erkennt die Änderungen halten sich in sehr engen Grenzen: Der Pin für die LED hat sich von `LED_BUILTIN` auf die (neu definierte) Konstante `LED_PIN` geändert – fertig. Schon blinkt die externe LED – in einer Farbe, die uns gefällt.

1.2.3.1 Übungsaufgabe „Digital Output"

	Erstellen Sie so wie oben beschrieben eine Schaltung mitsamt zugehörigem Programm, welches die externe LED ansteuert, laden sie es auf ihren Arduino und zeigen Sie es vor.	

1.2.3.2 Pflichtaufgabe „Digital Output"

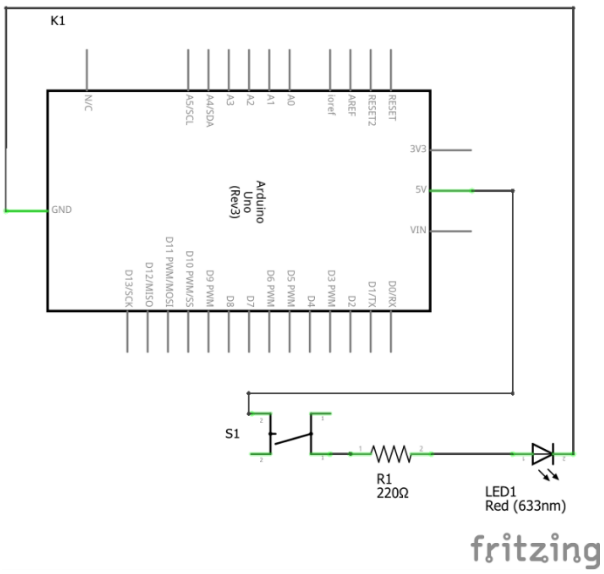
	Bauen Sie eine zweite LED ein. Diese soll immer dann leuchten, wenn die Erste gerade nicht leuchtet und umgekehrt. Modifizieren Sie das Programm entsprechend.	

1.2.4 Verwenden eines Tasters

Video-Link 8: https://youtu.be/L3gwYoOS7PQ

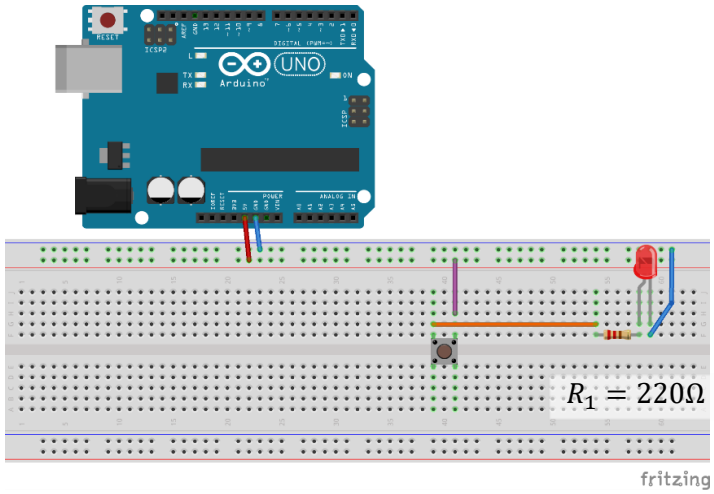
Eine Lampe, die immer leuchtet, ist natürlich nicht gerade das was man sich unter Nachhaltigkeit vorstellt. Die Beleuchtung abzustellen ist ein ganz selbstverständlicher Wunsch. Was man dafür tun muss, ist den Stromkreis zu unterbrechen. Das geschieht am besten mit einem Kontakt. Der Kontakt ist entweder geschlossen (dann kann Strom fließen) oder er ist geöffnet (dann fließt kein Strom). Wir haben einen solchen Kontakt in Form eines „Tasters". Ein Taster rastet nicht – solange man ihn betätigt, ist der Kontakt geschlossen – lässt man ihn los, so öffnet auch der Kontakt wieder[6].

Den Taster bauen wir jetzt so in unsere Testschaltung ein:



Mit dem Taster sollte es uns gelingen, die LED ein- und auszuschalten. Ist der Taster gedrückt, leuchtet die LED. Ist der Kontakt geöffnet, so ist die LED erloschen. Unbequem dabei ist, dass wir den Taster gedrückt halten müssen, wenn wir wollen, dass unsere LED leuchtet.

Auf dem Breadboard sieht das so aus:



[6] Im Unterschied dazu gibt es auch noch „Schalter". Diese haben zwei stabile Lagen und man schaltet den Kontakt um. Die Betätigungskraft kann nachher wieder verschwinden.

1.2.4.1 *Übungsaufgabe „LED Taster"*

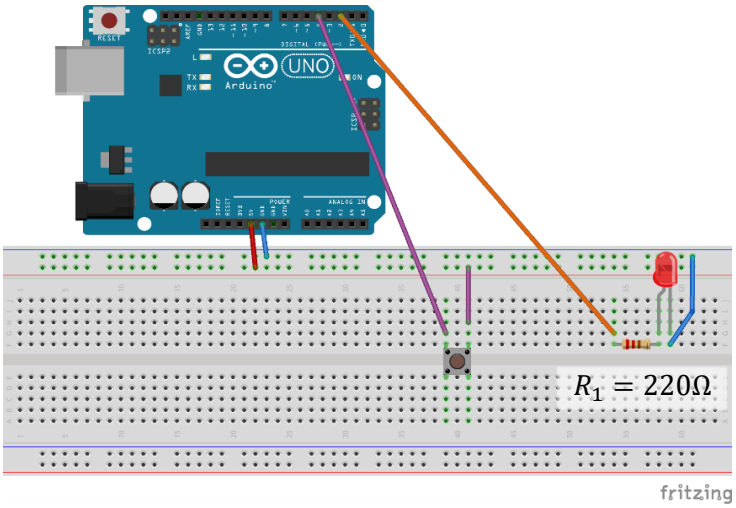
	Bauen Sie die oben beschriebene Schaltung nach, so, dass die LED leuchtet und geschaltet werden kann.	

1.2.5 Einlesen eines digitalen Eingangs

Video-Link 9: https://youtu.be/6CGzEhBUuig

Jetzt wollen wir dieses Verhalten nachprogrammieren. Dazu müssen wir das erste Mal den Zustand eines Schalters einlesen und eine entsprechende Reaktion setzen. Wir wollen also, dass uns der Taster entweder 5V auf einen Eingang bringt oder eben nicht. Wenn wir logisch 1 einlesen, dann setzen wir einen Ausgang und die LED leuchtet. Wenn wir logisch 0 einlesen, dann drehen wir die LED wieder ab.

Dazu bauen wir die Schaltung ein wenig um:



Der Schalter bringt uns also entweder 5V zum Pin 4 oder Pin 4 hängt in der Luft. Die LED wird wieder über Pin 2 angesteuert. Unser Eingang ist also Pin 4 und unser Ausgang wieder Pin 2. Entsprechend gestalten wir unser Programm:

```
#define LED_OUT 2
#define SWITCH_IN 4

void setup() {
  pinMode(LED_OUT, OUTPUT);       // define LED_OUT as output
  pinMode(SWITCH_IN, INPUT);      // define SWITCH_IN as input
}

void loop() {
  int readVal = 0;

  readVal = digitalRead(SWITCH_IN);  // Read state of digital input

  if (readVal) {                    // a value of 0 is logical FASLE
    digitalWrite(LED_OUT, HIGH);    // turn the LED on
  } else {
    digitalWrite(LED_OUT, LOW);     // turn the LED off
  }
}
```

Dieses Mal haben wir im `setup()` zwei Pins definiert. Einen wie gehabt als Ausgang (über die Konstante `LED_OUT` ist das Pin Nummer 2) und einen als Eingang (über die Konstante `SWITCH_IN` Pin Nummer 4). Also genau wie wir sie angeschlossen haben.

Im `loop()` wird zunächst der Eingang gelesen. Dies geschieht mit dem Befehl `digitalRead`. Man braucht nur die entsprechende Pin Nummer angeben und man bekommt den Status dieses Pins. Liegt dort eine Spannung von 5V so wird man `HIGH` (1) einlesen. Liegt dort eine Spannung von 0V wird man `LOW` (0) einlesen.

Das `if`-Statement unterscheidet in der Bedienung eigentlich `TRUE` oder `FALSE`. Es wird jedoch jeder Wert ungleich `0` als `TRUE` interpretiert. Und der Wert `0` als `FALSE`. Damit wird so bald `HIGH` (1) eingelesen wird der Ausgang gesetzt, und sobald `LOW` (0) eingelesen wird der Ausgang gelöscht. Die LED schaltet entsprechend mit.

Damit sollte es eigentlich funktionieren, oder?

1.2.5.1 Pflichtaufgaben „LED Taster Controller"

	Bauen Sie die oben beschriebene Schaltung nach, programmieren Sie das Programm und testen Sie die einwandfreie Funktion. Funktioniert das Programm wie erwartet oder zeigt sich manchmal „seltsames" Verhalten?	

1.2.6 Pulldown und Pullup

Video-Link 10: https://youtu.be/bzdUVutLuNo

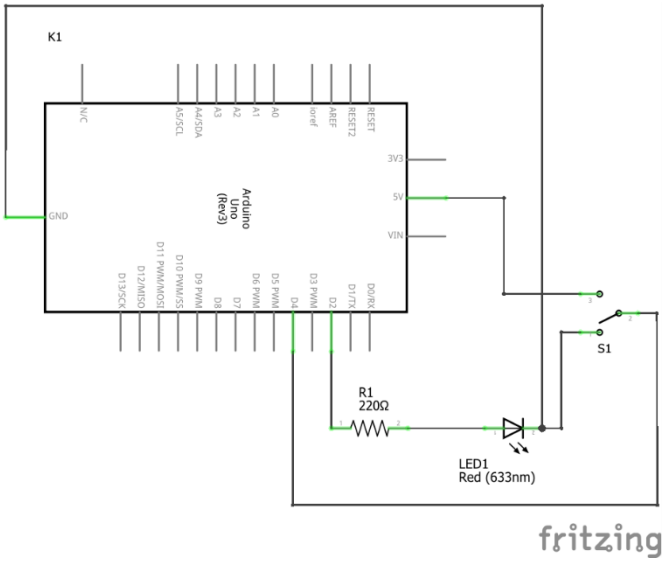
Warum geht die Schaltung in der vorigen Übung nicht optimal und zeigt immer wieder seltsames LED-Flackern? Das Problem ist die offene Stellung; wenn also der Eingang nicht mit `+5V` verbunden ist, sondern einfach „in der Luft" hängt.

Dadurch ist das Potential des Eingangs nicht definiert. Und wenn das Potential nicht definiert ist, ist auch die Potentialdifferenz und damit die Spannung zwischen dem Eingang und dem Bezugspotential (Ground, oder `GND`) nicht definiert.

Manchmal funktioniert es. Manchmal ist der Eingangs-Draht zufällig etwas „aufgeladen" und die LED geht nicht aus. Wenn man auf den Draht klopft, blitzt die LED – das alles sind Zeichen für ein solches Potentialproblem.

Sehen wir uns mögliche Lösungen an:

- Wir schalten nicht einfach „aus" sondern zwischen `+5V` und `GND` hin- und her:

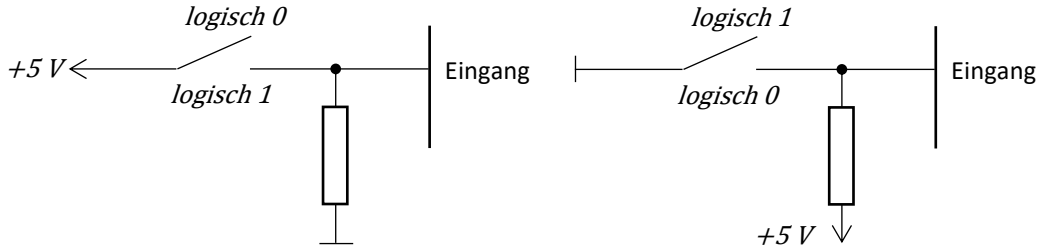


Das wäre eine relativ gute Lösung. Solch ein Kontakt heißt „Wechselkontakt", weil er zwischen zwei Anschlüssen wechseln kann[7]. Das Potential ist immer definiert und es wird keine unnötige Energie verbraucht. Einzig zum kurzen Umschaltzeitpunkt ist das Potential nicht definiert. Der große Nachteil dieser Methode ist jedoch der zusätzlich benötigte Draht von `GND` zum Schalter. Das braucht Platz und erhöhten Aufwand in der Leiterbahnführung und-planung.
Deswegen werden solche Lösungen, insbesondere bei integrierten Schaltungen, nicht verwendet. In anderen Branchen, wie z.B. im KFZ-Bereich ist dies eine Standard-Lösung[8].

[7] Der oben verwendete Taster hat einen sogenannten „Schließerkontakt" – das bedeutet, wenn man ihn betätigt, so wird er geschlossen, also leitend verbunden. Neben „Schließer" und „Wechsler" gibt es auch noch „Öffnerkontakte". Diese sind normal verbunden und öffnen bei Betätigung. Im Englischen heißt dies NO (normal open) oder NC (normal closed).

[8] Manchmal kann man bei KFZ bei den Hecklichtern seltsame Lichtspiele beobachten. Auf einer Seite leuchten statt des Bremslichtes andere Lichter, aber nur schwach. Oder statt dem Blinker blinken Schluss- und Bremslicht herum. Dann ist es ein Masse-Problem. An dieser Heckleuchte ist die `GND`-Leitung kaputt und der Strom rinnt über eine andere Lampe retour (die ja vom Wechsler auf `GND` geschalten wird). Somit leuchten beide, aber nur schwächer, denn die Spannung von im KFZ üblicherweise 12V bleibt gleich.

- Pull-Widerstände:



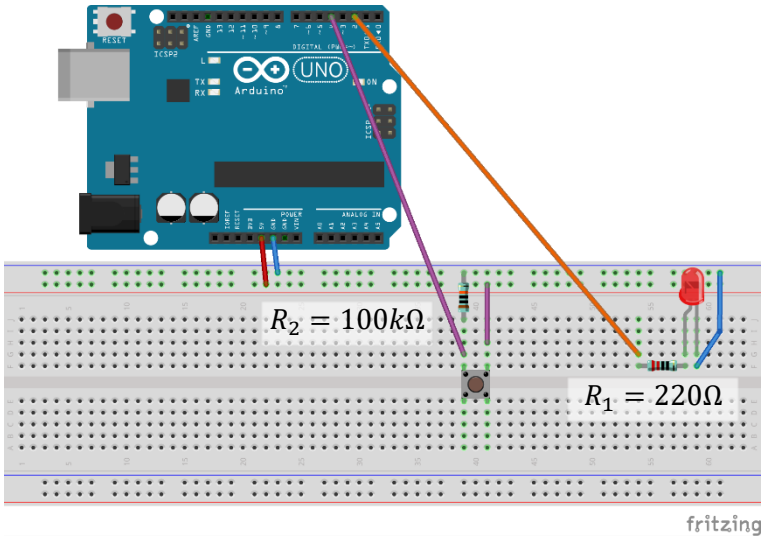
Pull-Down-Schaltung mit Pull-Down Widerstand Pull-Up-Schaltung mit Pull-Up Widerstand

Hier werden zusätzliche Widerstände verwendet. Sehen wir uns den Pulldown-Widerstand an: Der ist zwischen den Eingang und `GND` geschalten – beinhart. Das bedeutet aber auch, wenn man mit dem Schalter `+5V` auf den Eingang bringt rinnt ein Strom durch den Widerstand. Der Widerstand muss groß genug sein, dass nur ein kleiner Strom rinnt, der uns nicht weiter stört (irgendwo im höheren $k\Omega$-Bereich, wir werden $100k\Omega$ verwenden.
Gut, also wenn der Schalter geschlossen ist liegen die 5V am Eingang an und ein kleiner Strom rinnt über den Pulldown-Widerstand weg. Wird der Schalter jetzt geöffnet, so wird eine eventuelle Aufladung des Leiters über den Widerstand gegen Masse abgeleitet. Der Eingang wird definiert auf Ground gezogen.
Das ist genau das Verhalten, welches wir wollen: Ist der Schalter geschlossen, hängen wir definitiv auf `+5V`. Ist der Schalter geöffnet, hängen wir definiert auf oV – passt! Ein Pulldown-Widerstand zieht das Potential also auf des Bezugspotential hinunter – daher der Name.
Im Falle von Pullup zieht der Widerstand das Potential auf den `HIGH`-Pegel hinauf. Der hängt also zwischen dem Eingang und `+5V`. Der Kontakt verbindet den Eingang jetzt nicht mehr mit `+5V` sondern mit `GND`. Dadurch wird der Eingang bei Betätigung des Schalters definiert auf oV gezogen. Wir haben die Reaktion nun also umgekehrt: Beim Drücken des Tasters wird der Eingang gelöscht. Das ist aber unwichtig, Hauptsache, wir bekommen sicher mit, dass der Schalter betätigt wurde. Was jetzt `LOW` und `HIGH` wirklich bedeuten legen wir ja mit unserem Programm fest.

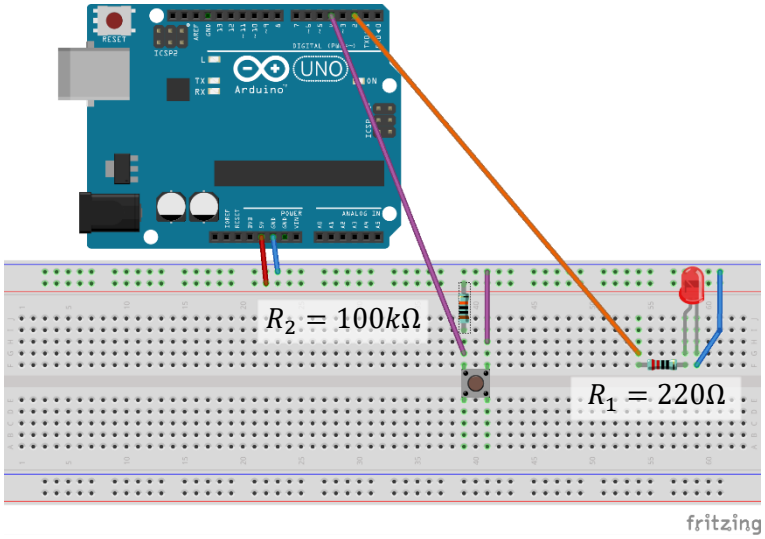
Normalerweise werden Pull-Widerstände verwendet? Warum? Da braucht man ja einen ganzen Bauteil mehr und nicht nur eine Leitung. Ja, das stimmt, nur ist ein Widerstand sehr klein herzustellen (und damit billig) und `HIGH`- und `LOW`-Pegel sind beim Eingang ohnehin verfügbar, der Aufwand hält sich in Grenzen.

Außerdem kann solch ein Widerstand gleich im Chip integriert sein. Das bedeutet, von außen sieht man gar nichts. Aus dem Datenblatt kann man dann erfahren, ob ein Pull-Widerstand verbaut ist oder nicht. Damit funktionieren die Eingänge mit den Pull-Widerständen „transparent" für uns. Was wir schon wissen müssen ist, ob es sich um einen Pullup- oder einen Pulldown-Widerstand handelt.

Wie würde das nun bei unserer Schaltung aussehen? Bauen wir zunächst die Variante mit Pulldown-Widerstand ($100k\Omega$):



Dann die Variante mit Pullup-Widerstand (ebenfalls $100k\Omega$):



1.2.6.1 Übungsaufgaben „Input external Pull Resistor"

	Bauen Sie Ihre Schaltung um, so dass sie mit einem Pulldown-Widerstand funktioniert. Überlegen Sie ob Sie das Programm anpassen müssen und führen Sie die Anpassungen entsprechend aus.	

	Bauen Sie Ihre Schaltung um, so dass sie mit einem Pullup-Widerstand funktioniert. Überlegen Sie ob Sie das Programm anpassen müssen und führen Sie die Anpassungen entsprechend aus.	

1.2.7 Interner Pullup-Widerstand

Video-Link 11: https://youtu.be/-EHohJ6aumo

Nun, es ist ein zusätzliches Schaltungsteil, welches wir in unserer Schaltung berücksichtigen. Glücklicherweise lassen sich Widerstände sehr klein herstellen und man kann sie gut direkt auf dem Silizium-Chip integrieren. Genau das ist bei unserem Arduino passiert: Dort gibt es einen eingebauten Pull-Up-Widerstand.

Aber warum hat dann unsere Schaltung beim Einlesen des Einganges nicht ohne externe Schaltung funktioniert? Das ist, weil wir uns aussuchen können, ob wir den internen Pull-Widerstand verwenden möchten, oder nicht. Der Unterschied liegt in der Definition des Pins. Ersetzen wir die Zeile:

```
pinMode(SWITCH_IN, INPUT);
```

mit

```
pinMode(SWITCH_IN, INPUT_PULLUP);
```

ersetzen, so wird der interne Pullup-Widerstand aktiv – und wir können die Schaltung ohne Widerstand betreiben.

1.2.7.1 Pflichtaufgaben „Input with internal Pullup Resisior"

	Bauen Sie Ihre Schaltung wieder ohne Widerstand auf. Und stellen sie das Programm auf den internen Pullup-Widerstand um. Überlegen Sie ob Sie das Programm anpassen müssen und führen Sie die Anpassungen entsprechend aus.	

1.2.8 Speicherschaltung

Video-Link 12: https://youtu.be/T2yuLsvUtDM

Bisher hat das Programm, bzw. der ganze Controller, wenig Sinn. Das gleiche Ergebnis könnte man mit praktisch den gleichen Bauteilen, nur wesentlich billiger, nämlich ohne Controller und ohne Programm erreichen. Außerdem macht auch das Verhalten nur sehr eingeschränkt Sinn. Wo kann man schon so etwas brauchen?

Deswegen werden wir jetzt das Verhalten unseres Programms ein wenig anpassen. Was wir wollen:

Beim 1. Tastendruck soll die LED angehen. Beim 2. Tastendruck soll die LED ausgehen.

1.2.8.1 Pflichtaufgaben „Memory On/Off Light"

	Programmieren sie Ihr Programm um: Der Eingang soll mit internem Pullup-Widerstand betrieben werden. Das Verhalten soll so wie oben beschrieben funktionieren.	

1.2.9 Der Serielle Monitor

Video-Link 13: https://youtu.be/Gzk1Vqj_5-U

Na, das war für die meisten gar nicht so einfach. Vor allem das Suchen nach dem Fehler gestaltet sich gar nicht einfach. Es wäre schön, wenn man ein bisschen in den Programmablauf hineinblicken könnte.

Zum Glück gibt es bei unserem Arduino diese Möglichkeit. Man kann sich „Nachrichten" schreiben. Und zwar auf dem sogenannten „Seriellen Monitor". Seriell deswegen, weil wir mit einem Universal Serial Bus (USB) verbunden sind. Und „Monitor", weil dieser anzeigt, was sich auf dem seriellen Bus tut. Und wenn wir uns auf der seriellen Schnittstelle Daten zukommen lassen, können wir diese mit dem Seriellen Monitor anzeigen.

Um Daten auf einer seriellen Schnittstelle auszugeben, muss diese zunächst vom Programm „initialisiert" werden. Das klingt wilder als es ist, es ist nämlich nur ein Befehl notwendig und dieser lautet:

```
Serial.begin(9600);
```

Genaugenommen wird hier die `begin`-Methode des Objekts `Serial` aufgerufen. Der Wert in Klammer ist die Kommunikationsgeschwindigkeit in Zeichen pro Sekunde („baud") – auch genannte Baud-Rate. Im obigen Beispiel nehmen wir 9600 Zeichen pro Sekunde, wobei ein Zeichen immer aus einem Byte (also 8 Bit) besteht. Normalerweise steht diese Initialisierung im `setup()`-Teil des Programms.

Um dann etwas auf der seriellen Schnittstelle auszugeben können wir die Methoden

```
Serial.print(text);
```

```
Serial.println(text);
```

verwenden. Dabei versucht `print` so viel wie möglich als lesbaren Text anzuzeigen. Zahlen wie `integer` oder `float` werden z.B. in eine Zeichenkette umgewandelt.

Damit können wir an einer uns passend erscheinenden Stelle im Programm irgendwelche Ausgaben machen. Sei es zum Beispiel, um festzustellen, dass wir einen bestimmten Programmzweig durchlaufen:

```
Serial.println("Bin jetzt bei Position 1.");
```

Oder wir können uns auch Werte von Variablen ausgeben (Man beachte die Verwendung von `print` und `println`):

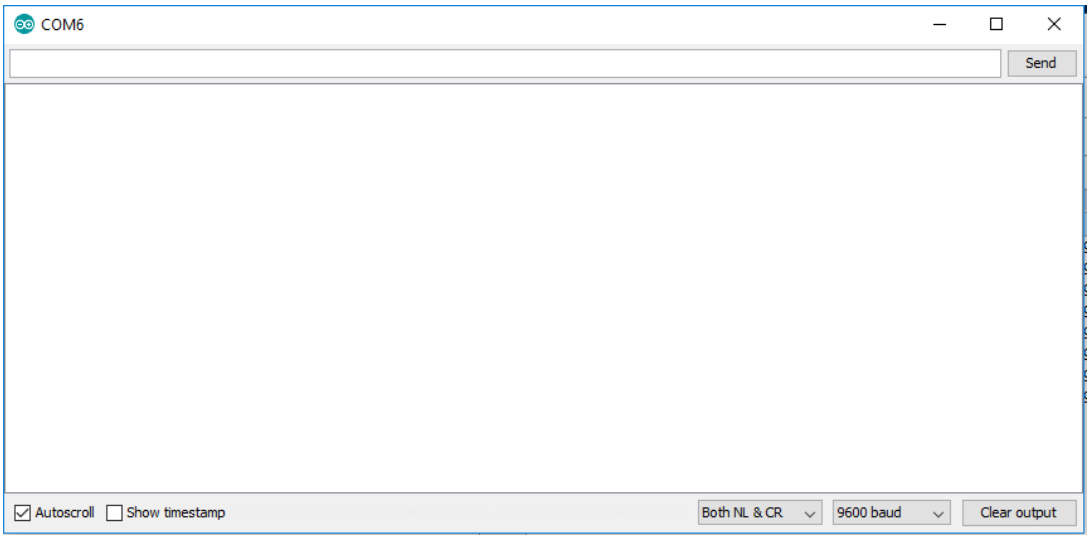
```
Serial.print("i = ");
Serial.println(i);
```

Nehmen wir einfach ein Beispiel her. Wir verwenden jetzt einfach das obige Programm und erweitern es um die serielle Schnittstelle:

```
#define LED_OUT 2
#define SWITCH_IN 4

void setup() {
  Serial.begin(9600);
  pinMode(LED_OUT, OUTPUT);              // define LED_OUT as output
  pinMode(SWITCH_IN, INPUT_PULLUP);      // define SWITCH_IN as input
}

void loop() {
  int readVal = 0;
  static bool pressed;
  static bool lit;

  readVal = digitalRead(SWITCH_IN);      // Read status of Switch

  if (!readVal) {                        // If pressed we read LOW
    Serial.print("Button pressed.");     // Write this on Serial
    if (!pressed) {                 // button was not pressed before
      lit = !lit;                   // change status of LED
      Serial.print("-> new, so turn LED ");
      Serial.println(lit);          // Output this on Serial
    } else Serial.println("");      // just make an ENTER
  }

  pressed = !readVal; // store information of switch for next cycle

  if (lit) {
    digitalWrite(LED_OUT, HIGH);      // turn the LED on
  } else {
    digitalWrite(LED_OUT, LOW);       // turn the LED off
  }
}
```

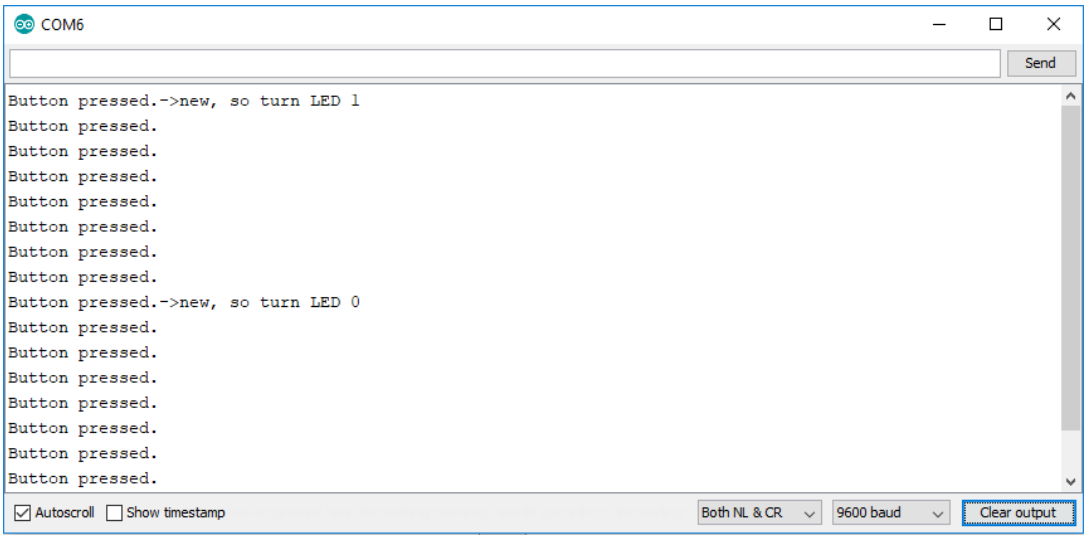
Wenn wir dieses Programm laden, dann funktioniert es so wie vorher. Wir merken eigentlich keinen Unterschied. Wo finden wir nun unsere Ausgaben auf der Seriellen Schnittstelle? Dazu müssen wir den Serial Monitor öffnen. Dieser befindet sich rechts oben auf unserem Arduino-IDE-Fenster ().

Ein Klick auf dieses Symbol öffnet den Seriellen Monitor:



Rechts unten müssen wir noch die passende Übertragungsgeschwindigkeit einstellen. Weil wir im Programm 9600 Baud gewählt haben (bei der Methode `Serial.begin(9600)`) müssen wir hier auch 9600 wählen. Haben wir eine falsche Baud-Rate erscheinen nur seltsame Zeichen auf dem Monitor.

Passt alles zusammen, sollten wir im Fenster des seriellen Monitors die entsprechenden Ausgaben erkennen können:



1.2.9.1 Übungsaufgabe „Serial Monitor LED Output"

	Stellen sie obig beschriebenes Programm nach. Sehen sie sich die Ausgaben des Seriellen Monitors an. Erklären sie, warum es hin und wieder zu seltsamen Effekten kommt (kurzes Flackern).	

1.2.10 Zeitfunktion 1 (Ganglicht einfach)

Video-Link 14: https://youtu.be/hvWQMBIBkI8

Wir wollen nun unsere Ein- und Ausgänge anders verknüpfen. Der Hardware-Aufbau bleibt gleich, nur das Verhalten soll anders sein. Mit einer Softwareänderung kann man einfach ein ganz anders Verhalten erzeugen. Das ist ein großer Vorteil von Steuerungen wie diesen: mit dem gleichen Hardwareaufbau lassen sich unterschiedliche Ergebnisse erzielen. Die Logik liegt rein in der Software.

Was soll geschehen? Wenn man den Knopf betätigt, soll die LED angehen. Nach einer einstellbaren Zeit soll sie von alleine wieder ausgehen. Wir wollen also eine Art Ganglicht programmieren.

Bei dem Wort „warten" fällt einem einfach eine Verzögerung ein. Unser Arduino kennt den Befehl

```
delay(timeMs);
```

Dabei wartet das Programm die angegeben Anzahl an Millisekunden. Klingt perfekt. Also versuchen wir unser Programm entsprechend anzupassen:

```
#define LED_OUT 2
#define SWITCH_IN 4
#define GANG_TIME 10000

void setup() {
  Serial.begin(9600);
  pinMode(LED_OUT, OUTPUT);              // define LED_OUT as output
  pinMode(SWITCH_IN, INPUT_PULLUP);      // define SWITCH_IN as input
}

void loop() {
  int readVal = digitalRead(SWITCH_IN);

  if (!readVal) {
    Serial.print("Button pressed.");
    digitalWrite(LED_OUT, HIGH);      // turn the LED on
    Serial.print(" Wait for ");
    Serial.print(((float)GANG_TIME)/1000);
    Serial.println(" s");
    delay(GANG_TIME);
  }

  digitalWrite(LED_OUT, LOW);       // turn the LED off
}
```

Damit sollte die Aufgabenstellung eigentlich erfüllt sein.

1.2.10.1 Übungsaufgabe „Corridor light simple"

	Bauen und programmieren sie oben erwähntes Ganglicht mit dieser einfachen Methode. Versuchen Sie Schwachstellen zu erkennen und nennen Sie diese.	

1.2.11 Zeitfunktion 2 (Ganglicht komplex)

Video-Link 15: https://youtu.be/zoGSqkC3gl4

Der größte Nachteil an der Funktion `delay()` ist, dass der Controller in der Wartezeit gar nichts macht. Es werden keine Eingänge verarbeitet und keine sonstigen Entscheidungen getroffen. Einen Mitarbeiter, der während einer Wartezeit nichts anderes machen kann als warten, würde man wahrscheinlich nicht lange einen Kollegen nennen dürfen. Es muss also besser gehen als einfach stupide zu warten. Unser Arduino weiß auch, wie viele Millisekunden seit dem Start des Programms vergangen sind. Dazu kann man die Funktion

```
time = millis();
```

verwenden. Zurück kommt ein Zahlenwert vom Typ `unsigend long`. Also ein Wert zwischen `0` und `4294967295`. Ist unsere Steuerung länger als diesen Maximalwert in Betrieb (`4294967295ms` entsprechen etwa 50 Tage – genau 49,71 Tage), liefert die Funktion wieder kleinere Werte zurück.

Genau diese Funktion wollen wir nun in einem adaptierten Programm nutzen:

```
#define LED_OUT 2
#define SWITCH_IN 4
#define GANG_TIME 10000

void setup() {
  Serial.begin(9600);
  pinMode(LED_OUT, OUTPUT);              // define LED_OUT as output
  pinMode(SWITCH_IN, INPUT_PULLUP);      // define SWITCH_IN as input
}

void loop() {
  int readVal = 0;
  static unsigned long pressTime;
  static bool pressed;

  readVal = digitalRead(SWITCH_IN);

  if (!readVal) {
    Serial.print("Button pressed.");
    if (!pressed) {
      pressTime = millis();
      Serial.print("-> new, save millis: ");
      Serial.println(pressTime);
      delay (200);
    } else Serial.println("");
  }
  pressed = !readVal;
  if (( millis() - pressTime) < GANG_TIME) {
    Serial.print("Wait Time: ");
    Serial.print(((float)(pressTime + GANG_TIME -
                  millis()))/1000.0);
    Serial.println(" s");
    digitalWrite(LED_OUT, HIGH);      // turn the LED on
  } else {
    digitalWrite(LED_OUT, LOW);       // turn the LED off
  }
}
```

Wo ist der Unterschied in der Funktion? Hier können wir während der Laufzeit etwas machen. In diesem Fall ist das einzige, das passiert, die Ausgabe der Wartezeit auf dem Seriellen Monitor. Genau in der Zeile ist auch ein sogenannter Type-Cast:

```
((float)(pressTime + GANG_TIME - millis())) / 1000.0
```

Die Anweisung `(float)` wandelt das Ergebnis in den Typ `float` um. Hier wird also die Rechnung `pressTime + GANG_TIME - millis()` im Typ `unsigend long` durchgeführt, dann in eine Gleitkommazahl vom Typ `float` umgewandelt und danach durch 1000 dividiert.

1.2.11.1 *Übungsaufgabe „Corridor light advanced"*

	Bauen und Programmieren sie oben erwähntes Ganglicht mit dieser besseren Methode.	

1.2.11.2 *Pflichtaufgabe „Corridor light turn-off"*

	Erweitern Sie das Programm um folgende Funktion: das Ganglicht soll mit einem weiteren Druck auf den Taster während der Laufzeit auch aktiv abgeschaltet werden.	

1.2.12 Personalisieren von Einstellungen

Video-Link 16: https://youtu.be/3FUaEGSUr-8

Stellen wir uns vor, wir wollen unser Ganglicht personalisieren. Nicht jeder will, dass die gleiche Zeitdauer beleuchtet wird. Das ist bei einem Ganglicht vielleicht nicht so wichtig, aber stellen wir uns vor, wir wollen in einem Ofen Brot backen. Dort braucht man unterschiedliche Zeitdauern für große und kleine Brote. Sehen wir uns unser Programm an: Mit der Zeile

```
#define GANG_TIME 10000
```

definieren wir die Zeitdauer – in diesem Beispiel also 10000 ms, also 10 Sekunden. Wenn wir die Zeit ändern wollen, dann müssen wir nur diese Programmzeile ändern und schon leuchtet das Licht eine andere Zeit.

Wir müssen also das Programm ändern, danach übersetzen, in den Arduino einspielen und dann wieder starten – reichlich kompliziert.

Wir können natürlich auch eine Variable bemühen und diese ändern. Ich habe deswegen folgendes Programm geschrieben:

```
#define LED_OUT 2
#define SWITCH_IN 4
#define GANG_TIME 10000
#define LEARN_MODE 1000

void setup() {
  Serial.begin(9600);
  pinMode(LED_OUT, OUTPUT);              // define LED_OUT as output
  pinMode(SWITCH_IN, INPUT_PULLUP);      // define SWITCH_IN as input
}
```

```
void loop() {
  int readVal = 0;
  static unsigned long pressTime; // time the button was pressed
  static unsigned long pressLearn; // time no button was pressed
  static unsigned long waitTime = GANG_TIME; // wait time for delay
  static bool pressed = false; // button pressed already last cycle
  static bool learnMode = false;  // learn mode for delay time

  readVal = digitalRead(SWITCH_IN);

  if (!readVal) {
    Serial.print("Button pressed.");
    if (!pressed) {
      if (learnMode) {
        waitTime = millis() - pressTime;
        learnMode = false;
        Serial.print(" Newly stored delay time :");
        Serial.print((float)waitTime/1000.0);
        Serial.println(" s");
      } else {
        pressTime = millis();
        Serial.print("-> new, save millis: ");
        Serial.println(pressTime);
      }
      delay (200);
    } else {
      if (( millis() - pressLearn) > LEARN_MODE) {
        Serial.println("-> Enter learn Mode. Stopping countdown.");
        learnMode = true;
        digitalWrite(LED_OUT, LOW);
        delay (200);
        digitalWrite(LED_OUT, HIGH);
        delay (200);
      } else Serial.println("");
    }
  } else pressLearn = millis();

  pressed = !readVal;

  if (learnMode) {
    Serial.print("New Time: ");
    Serial.print(((float)(millis() - pressTime))/1000.0);
    Serial.println(" s");
  } else {
    if (( millis() - pressTime) < waitTime) {
      Serial.print("Wait Time: ");
      Serial.print(((float)(pressTime + waitTime -
                            millis())))/1000.0);
      Serial.println(" s");
      digitalWrite(LED_OUT, HIGH);      // turn the LED on
    } else {
      digitalWrite(LED_OUT, LOW);       // turn the LED off
    }
  }
}
```

Was passiert hier? Im `setup`-Teil ist nichts anders. Der Unterschied ist im `loop`-Teil. Zunächst haben wir ein paar neue Variablen:

```
static unsigned long waitTime = GANG_TIME;
```

Hier wird die Wartezeit gespeichert. Im restlichen Programm wird anstatt `GANG_TIME` jetzt immer nur `waitTime` verwendet. Und weil es sich jetzt statt um eine Compiler-Konstante um eine Variable handelt kann diese auch geändert werden. Initialisiert (bei Strom ein oder Reset) wird diese Variable nach wie vor mit der Konstante `GANG_TIME`.

Jetzt müssen wir uns nur mehr eine Möglichkeit schaffen, den Wert der Variablen auch zu verändern. Das geschieht folgendermaßen: Bleibt man länger auf dem Knopf (länger als `LEARN_MODE`, also hier 1000ms) schaltet die Steuerung in einen Lernmodus. Das Licht bleibt dann an, bis es durch einen Tastendruck ausgeschalten wird. Die Zeit, die zwischen der Aktivierung des Lern-Modus und dem Tastendruck vergangen ist, wird dann als neue Verzögerungszeit verwendet.

Dazu wird immer die Zeit gespeichert, wenn die Taste nicht gedrückt ist. Das passiert in der Zeile

```
else pressLearn = millis();
```

Wenn die Taste jedoch gedrückt wird, bleibt der Wert in `pressLearn` unverändert. Man kann also den Wert mit der aktuellen Zeit vergleichen und wenn der Unterschied größer als `LEARN_MODE` ist, dann schaltet man in den Lernmodus. Der Vergleich passiert hier:

```
if ((millis() - pressLearn) > LEARN_MODE) {
  Serial.println("-> Enter learn Mode. Stopping countdown.");
  learnMode = true;
  digitalWrite(LED_OUT, LOW);
  delay (200);
  digitalWrite(LED_OUT, HIGH);
  delay (200);
}
```

Dabei blinkt die LED mit 200ms, damit auch klar ist: Jetzt haben wir in den Lernmodus gewechselt.

Befindet sich die Steuerung im Lern-Modus, geht die LED nicht mehr automatisch aus. Stattdessen zeigt der serielle Monitor an, wieviel Zeit bereits vergangen ist:

```
if (learnMode) {
  Serial.print("New Time: ");
  Serial.print(((float)(millis() - pressTime))/1000.0);
  Serial.println(" s");
}
```

Erst wenn wieder eine Taste gerückt wird, wird die neue Zeit in `waitTime` gespeichert:

```
if (learnMode) {
  waitTime = millis() - pressTime;
  learnMode = false;
  Serial.print(" Newly stored delay time :");
  Serial.print((float)waitTime/1000.0);
  Serial.println(" s");
}
```

1.2.12.1 *Übungsaufgabe „Corridor light adjust"*

	Bauen und Programmieren Sie oben erwähntes Ganglicht mit dieser Methode. Zeigen Sie, dass Sie mit diesem Programm die Zeit verändern können. Versuchen Sie, Schwachstellen zu erkennen und nennen Sie diese.	

1.2.13 Speichern im nicht flüchtigen Speicher

Video-Link 17 (Speichervarianten): https://youtu.be/Tt16v7lJ4NE Video-Link 18 (Arduino EEPROM): https://youtu.be/PZoDGxIDVS4

Stellen wir uns vor, wir haben ein Radio und dieses ist auf unseren Lieblingssender eingestellt. Und immer, wenn wir ihn neu aufdrehen, müssen wir unseren Lieblingssender neu suchen – schon etwas nervig, oder? Bei solchen Einstellungen wie im vorigen Kapitel wäre es schön, wenn diese „nicht flüchtig", auch einen Reset oder Spannung Aus, überleben würden.

Dafür bietet unser Arduino einen sogenannten Flash-Bereich. Der funktioniert praktisch wie ein USB-Flash-Drive, nur ist er eben gleich eingebaut. Es handelt sich dabei um den sogenannten EEPROM[9]. Eine Speicherstelle kann im Durchschnitt ungefähr 100.000mal beschrieben werden. Danach gibt es seitens des Herstellers keine Garantie mehr, dass der richtige Wert drinnen steht – der EEPROM wird dann „vergesslich". Ein Verhalten, dass übrigens auch SSD-Platten haben. Deswegen gibt es dort ausgeklügelte Algorithmen, die eine gleichmäßige „Abnutzung" der Speicherstellen sicherstellen sollen („Wear-Leveling").

Aber, um einen Einstellwert zu speichern, ist das gut genug. Nehmen wir an, wir ändern den Wert jeden Tag, dann würden wir das länger als 273 Jahre[10] machen können.

Um dieses EEPROM nutzen zu können, benötigen wir eine sogenannte „Bibliothek" oder auch „Library". Das ist ein Programmteil, denn irgendjemand geschrieben hat und uns zur Verfügung stellt. Wir borgen uns die Funktion dieses Programmteils aus – daher der Name. Um eine Bibliothek zu verwenden, benötigen wir das `#include` statement. _Und die verwendete Bibliothek muss natürlich auf unserem Computer zur Verfügung stehen. Die für das EEPROM ist standardmäßig dabei und heißt EEPROM.h. Unsere erste Codezeile ist nun also

```
#include <EEPROM.h>
```

Damit binden wir die Bibliothek in unser Programm ein. Eine Beschreibung der Funktionen der Bibliothek findet man auch im Anhang (4.1.4.3). Unser obiges Programm aus Kapitel 1.2.12 ändert sich nun also folgendermaßen:

[9] EEPROM: Electrical Ereaseable Programmable Read Only Memory: Kann elektrisch gelöscht und wieder beschrieben werden. Vorgänger waren:
ROM: Read Only Memory – kam so aus der Fabrik und fertig. Wenn man eine Änderung machen wollte musste man sich einen neuen ROM kaufen.
PROM: Programmable Read Only Memory – Man kauft leere Speicherblöcke, kann diese programmieren, aber nicht mehr löschen. Wenn man eine Änderung wollte, dann benötigte man einen neuen leeren Baustein.
EPROM: Eraseable Programable Read Only Memory – Hier kann man den Speicherinhalt wieder löschen. Das geschah mit UV-Licht. Es gab Lichtkammern, in die man den Speicher einlegte. Nach ein paar Minuten (10-15) „Sonnenstudio" war der Speicher wieder leer.

[10] Will man 1x pro Sekunde den Wert ändern, so ergibt sich eine Lebensdauer von nicht einmal 28 Stunden! Es ist also schon wichtig hier hauszuhalten.

```
#include <EEPROM.h>

#define LED_OUT 2
#define SWITCH_IN 4
#define GANG_TIME_MAX 3600000
#define LEARN_MODE 1000
#define WAIT_TIME_ADDR 0

unsigned long waitTime;     // wait time for delay ms

void setup() {
  Serial.begin(9600);
  pinMode(LED_OUT, OUTPUT);              // define LED_OUT as output
  pinMode(SWITCH_IN, INPUT_PULLUP);      // define SWITCH_IN as input
  EEPROM.get(WAIT_TIME_ADDR, waitTime);
  if (waitTime > GANG_TIME_MAX) waitTime = GANG_TIME_MAX;
}
void loop() {
  int readVal = 0;
  static unsigned long pressTime; // time the button was pressed
  static unsigned long pressLearn; // time no button was pressed
  static bool pressed = false; // button pressed already last cycle
  static bool learnMode = false;  // learn mode for delay time

  readVal = digitalRead(SWITCH_IN);

  if (!readVal) {
    Serial.print("Button pressed.");
    if (!pressed) {
      if (learnMode) {
        waitTime = millis() - pressTime;
        EEPROM.put(WAIT_TIME_ADDR, waitTime);
        learnMode = false;
        Serial.print(" Newly stored delay time :");
        Serial.print((float)waitTime/1000.0);
        Serial.println(" s");
      } else {
        pressTime = millis();
        Serial.print("-> new, save millis: ");
        Serial.println(pressTime);
      }
      delay (200);
    } else {
      if (( millis() - pressLearn) > LEARN_MODE) {
        Serial.println("-> Enter learn Mode. Stopping countdown.");
        learnMode = true;
        digitalWrite(LED_OUT, LOW);
        delay (200);
        digitalWrite(LED_OUT, HIGH);
        delay (200);
      } else Serial.println("");
    }
  } else pressLearn = millis();

  pressed = !readVal;
```

```
if (learnMode) {
    Serial.print("New Time: ");
    Serial.print(((float)(millis() - pressTime))/1000.0);
    Serial.println(" s");
  } else {
    if (( millis() - pressTime) < waitTime) {
      Serial.print("Wait Time: ");
      Serial.print(((float)(pressTime + waitTime -
                           millis())))/1000.0);
      Serial.println(" s");
      digitalWrite(LED_OUT, HIGH);      // turn the LED on
    } else {
      digitalWrite(LED_OUT, LOW);       // turn the LED off
    }
  }
}
```

1.2.13.1 *Übungsaufgabe „Corridor light adjust EEPROM"*

	Bauen und Programmieren Sie oben erwähntes Ganglicht mit dieser Methode. Suchen und erklären Sie die Unterschiede zum vorherigen Kapitel.	

1.2.14 Analoge Ausgaben

Video-Link 19 (PWM): https://youtu.be/ABSYgU00c3A Video-Link 20 (Analog Out): https://youtu.be/1nZBLZo_wYM

Wir können nun digitale Ausgänge schalten. Oftmals reicht das auch, aber manchmal möchte man Dinge nicht nur ein- und ausschalten, sondern in deren Intensität einstellen (analog einem Wert, also einen Analogausgang). Ein gutes Beispiel ist die Helligkeitssteuerung der Hintergrund-beleuchtung des Smartphones. Dieses wird nicht einfach ein- und wieder ausgeschalten, sondern die Helligkeit passt sich automatisch an die Umgebungslichtstärke an. Ist es hell, leuchtet das Display heller. Ist es dunkel so ist auch das Display dunkler, damit es uns nicht die Augen „einhaut".

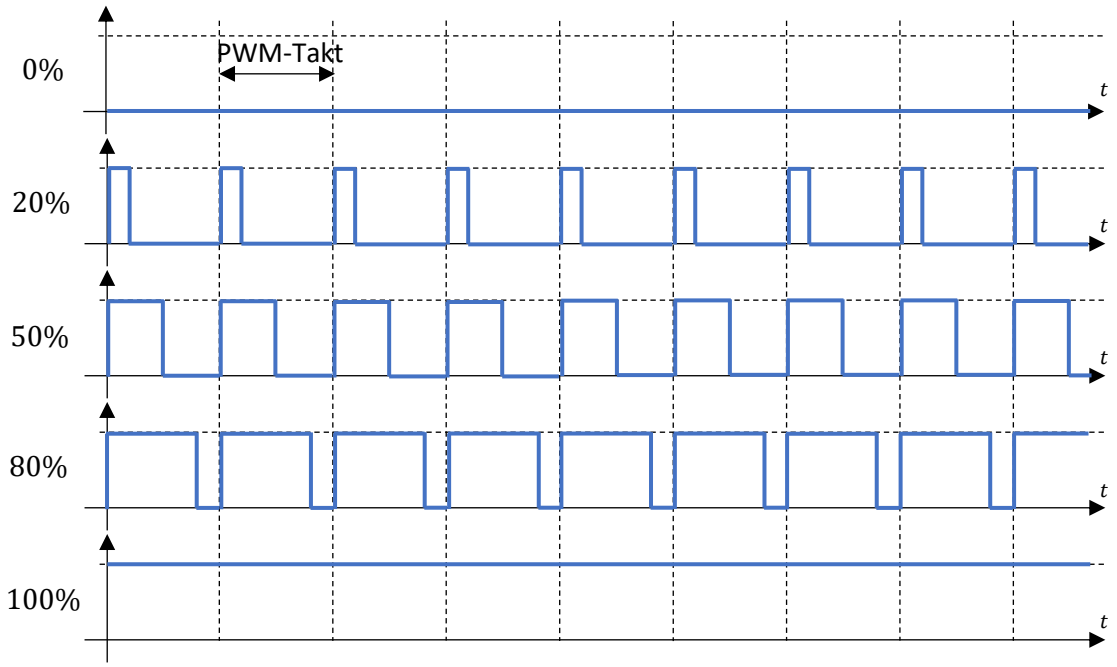
Unser Arduino kann jedoch Ausgänge nicht wirklich „halb" ansteuern. Entweder der Ausgang ist da oder eben nicht. Manche Ausgänge können jedoch mit der sogenannten „Pulsweitenmodulation" angesteuert werden. Was bedeutet das? Nun, entweder ist der Ausgang ein- oder ausgeschalten. Man kann den Ausgang jedoch kurz einschalten und danach kurz wieder aus. Dann wieder ein und wieder aus und immer so weiter. Je nachdem, welcher Anteil nun vom eingeschalteten Zustand und welcher Anteil vom ausgeschalteten Zustand eingenommen wird, ist der Ausgang im Mittel mehr oder weniger eingeschalten.

PWM erfolgt normalerweise mit einer fixen Frequenz, die sich als Kehrwert des PWM Takts versteht. Unser Arduino Uno hat üblicherweise die Frequenz 490Hz. Dein Ein-/Ausschalten-Spiel startet also etwa alle 2ms. Manche PINs haben sogar einen Takt von 980Hz und sind somit noch schneller[11].

Arduino UNO	Pin	3	5	6	9	10	11
	Frequenz	490 Hz	980 Hz	980 Hz	490 Hz	490 Hz	490 Hz

Zur besseren Verdeutlichung hier nun ein Bild einer typischen PWM. Die Prozentangeben spiegeln den Anteil des eingeschalteten Zustands wider.

[11] Wirklich schnell ist das jedoch nicht. Oftmals werden PWM Signale im kHz Bereich verwendet.

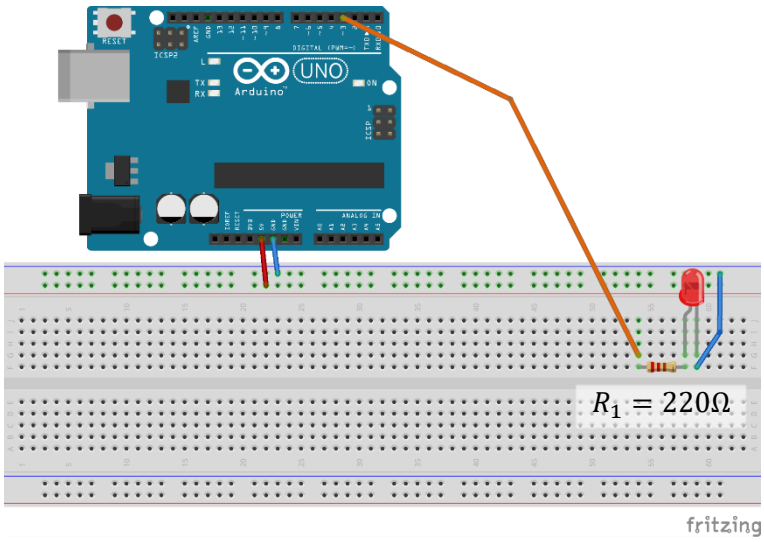


Der passende Befehl für das Ansteuern eines PWM Ausganges ist `analogWrite`. Wir müssen dem Befehl nur sagen, auf welchem Pin und welcher Prozentanteil eingeschalten werden soll:

```
analogWrite(pin, value);
```

`pin`	`int`	Pin Nummer des Pins der beschrieben werden soll. Nur bei PWM-Pins hat das einen Effekt.
`value`	`int`	Ein Wert zwischen `0` (0%) und `255` (100%)

Wir verwenden zunächst folgenden Hardware-Aufbau:



Dazu schreiben wir ein einfaches Programm, welches die LED mittels PWM ansteuert. Wir werden erkennen, was das wirklich bedeutet:

```
#define LED_OUT 3
#define WAIT_TIME 10

void setup() {
  pinMode(LED_OUT, OUTPUT);       // define LED_OUT as output
}

void loop() {
  static int val = 0;

  for (val = 0; val < 255; val++) {
    analogWrite(LED_OUT, val);
    delay (WAIT_TIME);
  }

  for (val = 255; val > 0; val--) {
    analogWrite(LED_OUT, val);
    delay (WAIT_TIME);
  }
}
```

1.2.14.1 *Übungsaufgabe „Analog Out"*

	Bauen und Programmieren Sie oben erwähnte Ansteuerung. Erklären Sie die Funktionsweise des Programms.	

1.2.14.2 *Pflichtaufgabe „Analog Out"*

	Erweitern Sie die Schaltung mit einem Taster. Wird der Taster gedrückt und gehalten, soll sich die Helligkeit der LED ändern. Nach dem Loslassen und neuerlichen gedrückt halten soll sich die Helligkeit in die andere Richtung ändern. Also z.B. beim ersten Mal halten soll die LED immer heller werden. Wenn man dann loslässt und erneut den Taster hält, soll die LED dunkler werden.	

1.2.14.3 *Zusatzaufgabe „Analog Out turn-off"*

	Der Taster soll folgende Funktion haben: Wird er nur kurz gedrückt, so soll die LED ein- und ausgeschalten werden können. Und zwar mit der Helligkeit, die zuletzt verwendet wurde.	

1.2.15 Analoge Eingänge

Video-Link 21 (Spannungsteiler): https://youtu.be/CevOz2EN77A Video-Link 22 (Potentiometer): https://youtu.be/5oySfAXBJNI
Video-Link 23 (A/D-Konverter): https://youtu.be/CevOz2EN77A Video-Link 24 (Analog Input): https://youtu.be/5oySfAXBJNI

Genauso wie es analoge (einem wert entsprechende) Ausgänge gibt, gibt es auch analoge Eingänge. Es gibt also Eingänge, die sind nicht einfach da oder nicht da, sondern können auch „ein bisschen" da sein. Wieviel dieses bisschen ist, hängt von der Stärke des Eingangssignals ab. Unser Arduino hat mehre Spannungseingänge (`A0-A5`). Üblicherweise liegt dort eine Spannung zwischen $0 - 5\ V$ an. Also genauso wie bei unseren digitalen Eingängen, mit dem Unterschied, dass nicht ab etwa $3V$ auf „HIGH" umgeschaltet wird, sondern, dass man vom analogen Eingang verschiedene Nummern bekommt. Was sind das für Nummern? Nun, je kleiner die Spannung, desto kleiner die Nummer. Je größer die Spannung, desto größer die Nummer. Liegen $0V$ an, so bekommt man einen Wert von 0. Liegen hingegen 5V an, so bekommt man eine maximale Nummer. Die maximale Nummer ist abhängig von der sogenannten „Auflösung". Bei uns beträgt die Auflösung $10bit$. Das beduetet, dass der Spannungswert mit einer Binärzahl mit $10bit$ dargestellt wird:

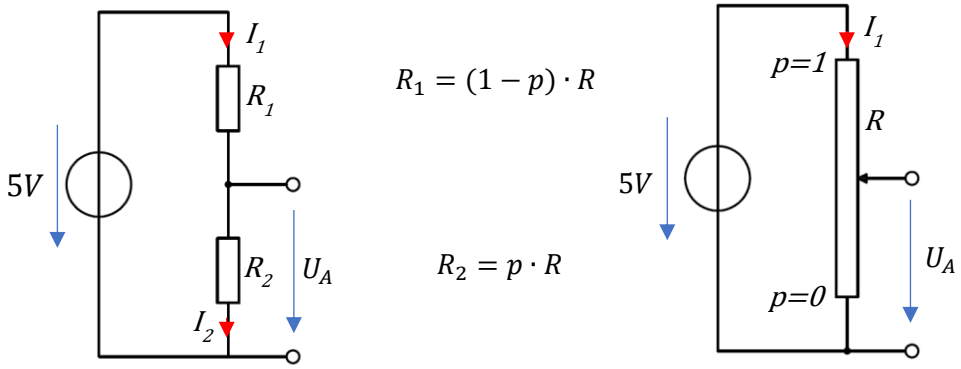
Spannung	Binärzahl	Dezimalzahl
$0V$	$00\ 0000\ 0000_{BIN}$	0_{DEZ}
$5V$	$11\ 1111\ 1111_{BIN}$	1023_{DEZ}

Schneller und einfacher lässt sich der maximale Wert auch angeben, wenn man sich einfach überlegt, wieviel Kombinationen man mit der entsprechenden Bit-Anzahl darstellen kann. Hier wären das:

$$2^{10} = 1024$$

Da auch 0 eine Kombination ist, ist die maximale Zahl um eins weniger als die so ausgerechnete. Wobei wir wieder auf 1023 kommen[12].

Um unseren Analogeingang zu testen, benötigen wir noch eine Spannung, die wir als Eingang verwenden können. Nun, unser Arduino stellt `5V` und `3,3V` zur Verfügung. Das ist ja ganz nett, aber wenn wir schon testen wollen, dann mit vielen unterschiedlichen Werten. Deswegen benötigen wir eine entsprechende Schaltung. Wir verwenden hier den sogenannten „Spannungsteiler".



Auf der linken Seite sehen wir den „typischen" Spannungsteiler mit zwei Widerständen. Wie der Name schon sagt, teilt ein Spannungsteiler eine Spannung in zwei Teile. Beide Teile gemeinsam ergeben wieder die ursprüngliche Spannung. Warum, wieso und was die Voraussetzungen sind, lernt

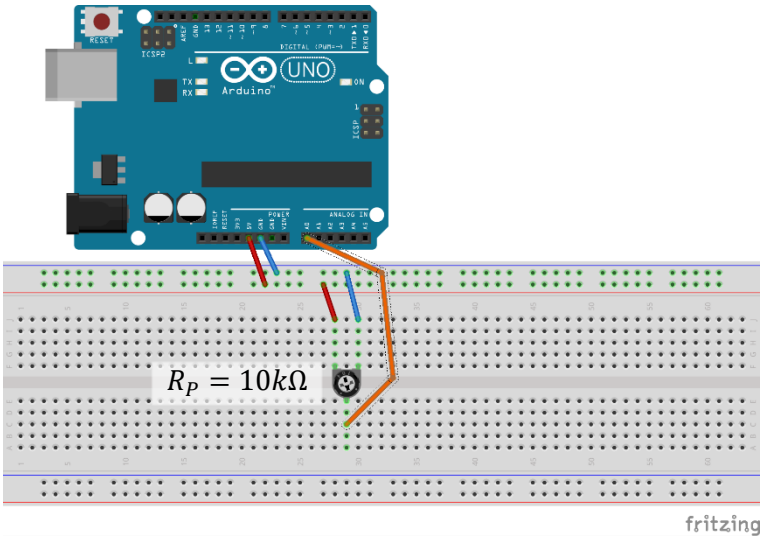
[12] Andere Analog/Digital-Konverter haben andre Bitbreiten. Sehr üblich sind $12Bit$ (Wertebereich $0 - 4095$). Hochwertige Standard-Konverter haben auch schon 14 $(0 - 16383)$ oder 15 Bit $(0 - 32767)$

man in Elektrotechnik. Für uns ist jetzt einmal nur wichtig, dass ein Spannungsteiler die ursprüngliche Spannung von $5V$ aufteilt in einen Teil U_A, und den Rest auf die $5V$. U_A wird dabei umso größer, je größer R_2 im Verhältnis zu R_1 ist. Im Extremfall, wenn $R_1 = 0\Omega$ ist R_2 viel größer und es wird $U_A = 5V$. Im anderen Extremfall, wenn $R_2 = 0\Omega$ wird $U_A = 0V$. Bei Werten dazwischen stellt sich eine Spannung zwischen 0 und $5V$ ein. Sind z.B. beide Widerstände gleich groß, so ist $U_A = {}^1/_2 \cdot 5V = 2{,}5V$.

Wichtig ist, dass die beiden Widerstände in Summe zumindest 1000Ω haben sollten. Sonst wird der $5V$-Ausgang überlastet.

Gut, wir bekommen also Spannungen zwischen 0 und $5V$. Wir können unseren Eingang testen. Unpraktisch dabei ist, dass wir immer Widerstände stecken müssen. Es wäre doch schön, wenn wir das stufenlos hinbekommen. Genau zu diesem Zweck gibt es ein sogenanntes Potentiometer (rechte Seite des oberen Bildes). Dieses hat einen fixen Widerstand. Mit einem Schleifkontakt kann man den Widerstand in zwei Teile zerlegen. Bewegt man den Schleifkontakt ändert man das Verhältnis der beiden Teile – das ist genau das, was wir wollen! Deswegen verwendenden wir genau ein solches Potentiometer.

Wir haben ein $10k\Omega$-Potentiometer in unserem Starter-Set. Damit machen wir folgenden Hardware-Aufbau:



Die äußeren Anschlüsse liegen auf $5V$ (`5V`) und $0V$(`GND`). Der Schleiferanschluss wird auf den Eingang `A0` verdrahtet. Damit können wir den Spannungswert zwischen 0 und $5V$ auf unseren Analogeingang bringen.

Jetzt noch zum Programm. Der Befehl, den wir brauchen ist:

```
int value = analogRead(pin);
```

`pin`	`int`	Pin-Nummer des Pins der gelesen werden soll (`0`=`A0`).
`value`	`int`	Ein Wert zwischen `0` (0%) und `1023` (100%)

Damit können wir ein einfaches Programm schreiben:

```
#define ANA_00_IN 0

void setup() {
  Serial.begin(9600);
}

void loop() {
  int readVal = analogRead(ANA_00_IN);

  Serial.print("Analogwert: ");
  Serial.print(readVal);
  Serial.print(" (");
  Serial.print(((float)readVal)/10.23);
  Serial.print(" %, ");
  Serial.print(((float)readVal)*5.0/1023.0);
  Serial.println(" V)");
}
```

Was macht das Programm? Es gibt einfach den eingelesenen Wert auf dem seriellen Monitor aus. Und zwar als Wert, als Prozent und die gemessene Spannung. Probeausgabe:

```
Analogwert: 173 (16.91 %, 0.85 V)
```

1.2.15.1 Übungsaufgabe „Analog In"

	Bauen und Programmieren Sie oben erwähntes einlesen des Analogwertes. Erklären Sie die Funktionsweise des Programms.	

1.2.15.2 Pflichtaufgabe „Analog In"

	Erweitern Sie die Schaltung derart, dass eine LED angesteuert wird. Dabei soll die LED bei einem kleinen Analogwert dunkel, und bei einem hohen Analogwert hell leuchten.	

1.2.15.3 Zusatzaufgabe „Analog In, adjustable"

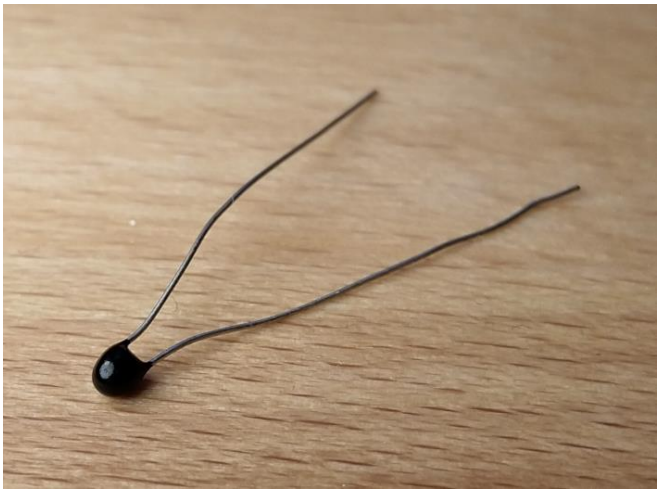
	Der Wert für Minimale und Maximale Helligkeit soll einstellbar sein.	

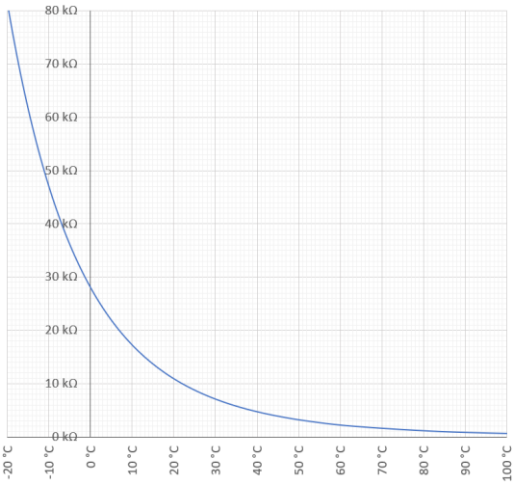
Video-Link 25 (Serial Plotter): https://youtu.be/LPVWyHBPlho

1.2.16 Der NTC als Temperatursensor

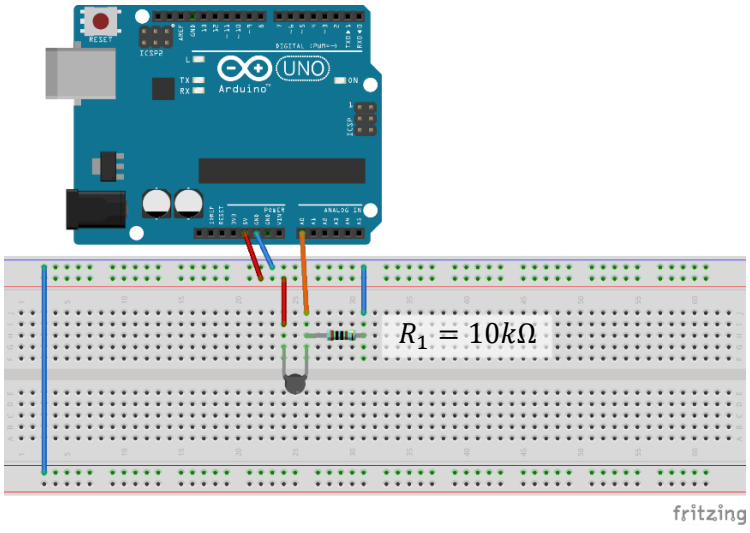
Video-Link 26 (Bauelement): https://youtu.be/eifkygfDF3Y Video-Link 27 (Programm): https://youtu.be/FZRVupykrQU

NTC bedeutet „Negative Temperature Coefficient". In Elektrotechnik hört man, dass sich der Widerstand eines Materials mit der Temperatur ändert. „Normale" metallische Leiter haben dabei einen positiven Temperaturkoeffizienten. Das bedeutet, dass der Widerstand höher wird, wenn die Temperatur steigt und umgekehrt. Oftmals wird dieser Effekt bei der Temperaturmessung verwendet und bildet den gängigen Bereich der „Widerstandsthermometer"[13].





Verschiedenste Materialen wurden getestet und im Bereich der Halbleiter wurden dann auch tatsächlich Materialen gefunden, die einen negativen Temperaturkoeffizienten aufweisen (der Widerstand wird größer, wenn das Material kälter wird). Leider sind diese Materialien sehr „nichtlinear" (siehe Diagramm)[14]. Das bedeutet, die Widerstandsänderung muss umständlich in Temperatur umgewandelt werden. Zum Glück kann das ja unser Arduino für uns erledigen. Machen wir einen entsprechenden Aufbau (Der Widerstand hat einen Wert von $10k\Omega$):



[13] In Automatisierungstechnik werden wir mehr über Temperaturmessung und deren Methoden erfahren.
[14] Natürlich sind zwecks der einfacheren Handhabung Sensoren mit linearer Charakteristik wünschenswert.

Die Umrechnung zwischen dem eingelesenen Wert und der tatsächlichen Temperatur wird so programmiert:

```
int readVal = analogRead(ANA_00_IN);
float temp = log(10000.0 * ((1024.0 / readVal - 1)));
temp = 1 / (0.001129148 +
              (0.000234125 + (0.0000000876741 * temp * temp ))
           * temp )
       - 276.15;
```

Genau diese Formel rechnet den eingelesenen Wert (`0..1023`) in die momentane Temperatur in °C um.

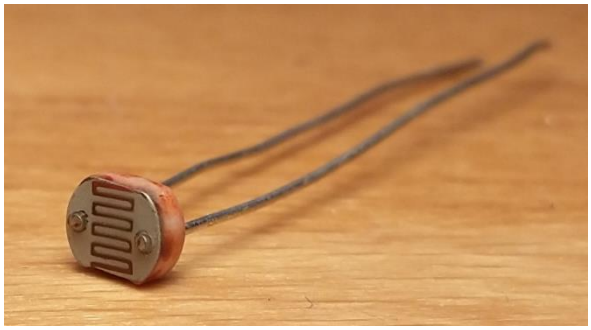
1.2.16.1 Pflichtaufgabe "Temperature Sensor"

	Aufbau der oben angeführten Schaltung. Die gemessene Temperatur soll am Serial Monitor ausgegeben werden.	

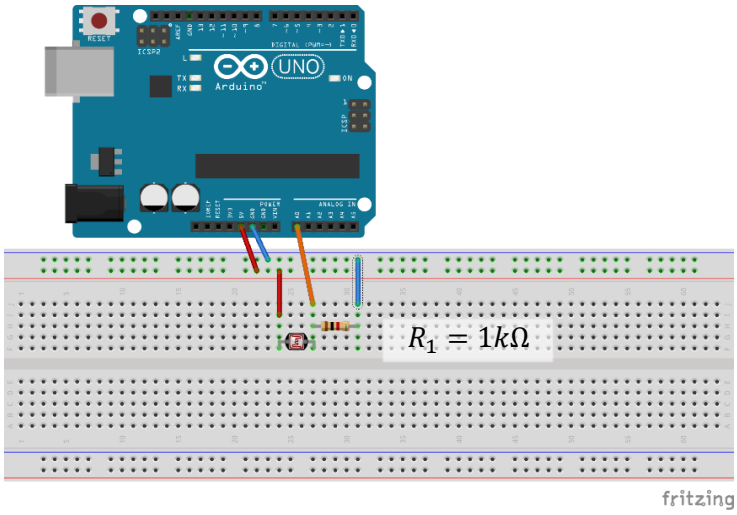
1.2.17 Helligkeitssensor

Video-Link 28 (Bauelement): https://youtu.be/ohtV-rIrAog Video-Link 29 (Programm): https://youtu.be/vLajecdz-Es

Der in unserm Set beinhaltete Helligkeitssensor ist eigentlich nichts anderes als ein Widerstand, dessen Wert sich mit der Photonendichte (Lichtintensität) ändert (Photowiderstand). Je mehr Photonen den Ladungsträgern „helfen" zu fließen, desto geringer ist der Widerstand. Bei Dunkelheit ist der Widerstandswert irgendwo bei $200k\Omega$. Im grellen Licht geht es auf wenige Ohm herunter.



Genau diesen Effekt wollen wir ausnutzen. Wir werden einen Spannungsteiler aufbauen, wobei einer der Widerstände unser Photowiderstand ist. Dabei ist zu beachten: die minimale Last soll bei etwa $1k\Omega$ liegen. Wenn der Widerstand bis auf wenige Ohm sinken kann, dann wählen wir eben den 2. Widerstand mit $1k\Omega$. Es wäre noch schön, wenn der gemessene Wert mit steigendem Lichteinfall höher wird. Deswegen werden wir den Photowiderstand „oben", als R_1, verwenden. Genau dann wird nämlich der unter Widerstand im Verhältnis immer größer und der eingelesene Wert steigt.



Wir machen also den Hardwareaufbau (wir ersetzten praktisch nur das Potentiometer durch diesen Spannungsteiler, wenn Sie die LED noch aufgebaut haben, dann lassen Sie das auch so).

Von $5V$ (`5V`) geht es über den Photowiderstand und einen $1k\Omega$-Widerstand auf $0V$ (`GND`). An der Verbindungsstelle geht es zum Analogeingang (`A0`).

Wenn es also finster ist, dann sollte dort ein kleiner Wert zu finden sein. Ist es hell, so wird der Wert höher.

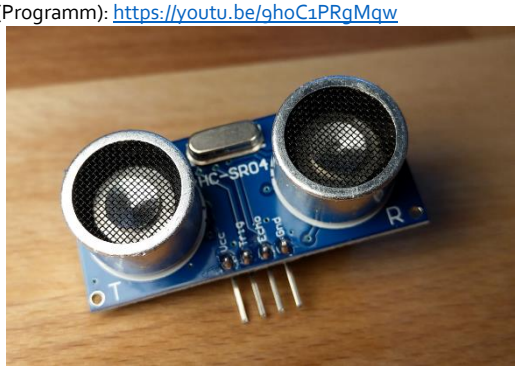
Ohne Änderung des Programms (gleiches Programm wie bei 1.2.15) sollten wir die Funktion zumindest im Seriellen Monitor beobachten können.

1.2.17.1 Pflichtaufgabe „Photoncell"

	Bauen Sie den Helligkeitssensor wie beschrieben auf. Verwenden Sie auch die LED so wie im vorigen Kapitel. Beschrieben Sie den Effekt und vergleichen Sie diesen mit der Steuerung der Displayhelligkeit auf ihrem Smartphone. Funktioniert das genauso oder genau verkehrt?	

1.2.18 Abstandssensor

Video-Link 30 (Bauelement): https://youtu.be/k1YEiX85cZU Video-Link 31 (Programm): https://youtu.be/9hoC1PRgMqw

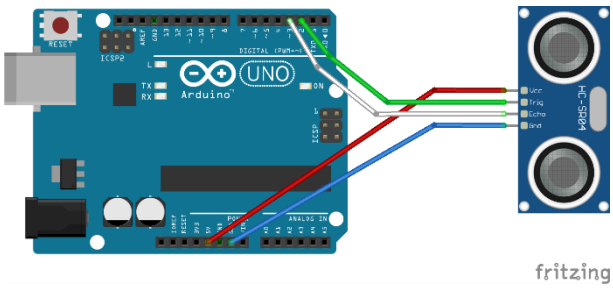


Abstände kann man auf vielfältige Weise messen. Eine Möglichkeit ist ein Maßband. Das lässt sich aber schlecht automatisieren. Deswegen verwenden wir hier einen anderen Ansatz: Wir erzeugen einen Ton und warten auf das entsprechende Echo. Je länger wir auf das Echo warten müssen, desto weiter muss die reflektierende Oberfläche entfernt sein. Und weil wir dabei auf blödes Gepiepse verzichten wollen, verwenden wir eben eine Wellenlänge, die wir als Menschen nicht mehr wahrnehmen können – Ultraschall. Der entsprechend Ultraschallsensor in unserm Starterkit ist der `HC-SR04`. Dieser zeichnet sich durch getrennte Sende- und Empfangseinrichtungen aus. Es gibt einen dezidierten Lautsprecher (mit T gekennzeichnet) und ein Mikrofon (mit R gekennzeichnet)[15]. Der Sensor benötigt eine Spannungsversorgung ($3.3V$ wären

[15] Es gibt auch Sensoren mit kombinierter Technik. Diese sind kleiner, weil es jedoch Zeit benötigt, um von Lautsprecher auf Mikrofonmodus umzuschalten, können diese erst ab einer gewissen Entfernung messen.

genug, $5V$ machen aber auch nichts). Die Messung kann über ein Signal auf dem `TRIG`-Pin gestartet werden. Danach liegt auf dem `ECHO`-Pin so lange ein Signal an, wie es dauert, ein Echo zu empfangen. Wenn wir also die Zeit des `ECHO`-Pins messen, wissen wir die Laufzeit des Signals. Ist die Schallgeschwindigkeit bekannt, so kann daraus auf die Entfernung geschlossen werden.

Am Arduino benötigen wir also zwei Pins: einen Ausgang, um die Messung zu starten und einen Eingang, dessen Dauer wir messen müssen. Bauen wir eine entsprechende Schaltung auf.



Und versuchen wir ein einfaches Programm dazu zu schreiben:

```
#define TIGGER_PIN    2
#define ECHO_PIN      3

unsigned long get_duration()
{
  digitalWrite(TIGGER_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TIGGER_PIN, LOW);
  delayMicroseconds(2);
  return pulseIn(ECHO_PIN, HIGH, 10000L);
}

void setup() {
  Serial.begin(9600);
  pinMode(TIGGER_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
}

void loop() {
  unsigned long duration = get_duration();

  if (duration != 0) {
    Serial.print("Gemessene Dauer : ");
    Serial.print(duration);
    Serial.println(" µs");
  } else {
    Serial.println("! kein Echo !");
  }

  delay(500);
}
```

Wirklich neu sind bei der ganzen Sache nur zwei Dinge:

```
delayMicroseconds(timeMs);
```

Parameter:

`timeMs` `unsigned long` Wartezeit des Programms in Mikrosekunden. Kein anderer Code wird ausgeführt! Praktisch das gleiche wie `delay` nur eben mit feiner Unterteilung.

Die Messung der Pulsdauer des Eingangs erledigt ein einziger Befehl:

```
unsigned long duration = pulseIn(pinNo, value[, timeout]);
```

Parameter:

`pinNo` `int` Nummer des Pins der gelesen werden soll

`value` `int` Typ des Pulses, der gemessen werden soll:
`HIGH` ein Pulsdauer von `GND` auf `+5V` soll gemessen werden
`LOW` ein Pulsdauer von `+5V` auf `GND` soll gemessen werden

`timeout` `unsingned long` Timeout für das Warten auf den Anfang des Pulses in ms. Defaultwert: 1000ms

`duration` `unsingned long` Gemessene Pulsdauer am angegebenen Pin in μs

1.2.18.1 Pflichtaufgaben „Distance Measurement"

	Bauen Sie die Schaltung auf und verwenden Sie das Programm als Basis. Es soll jedoch nicht die Dauer, sondern die Entfernung des reflektierenden Gegenstandes in cm ausgegeben werden. Überlegen Sie, wie Sie diese bei UltraSCHALLimpulsen berechnen könnten.	

1.2.18.2 Zusatzaufgabe „Temperaturkompensation"

	Die Messmethode ist Temperaturempfindlich (eine andere Temperatur führt zu einem anderen Messergebnis). Versuchen Sie die gemessene Entfernung mit einer gemessenen Temperatur zu korrigieren.	

2 Virtual- und Augmented-Reality-Systeme

Playlist-Link: https://youtube.com/playlist?list=PLVut1tKPvtvP7toNsmlonWPwJQjhYq-pB

2.1 Allgemeines

2.1.1 Wo liegt der Unterschied zwischen VR und AR?

Video-Link 1: https://youtu.be/q_tcmoNsRlM

Bei einer Virtual Reality Anwendung kommt alles aus dem Computer: Dreidimensional dargestellte Bilder, Geräusche, physikalische Eigenschaften der dargestellten Dinge unter Umständen sogar physikalische Einwirkungen auf den Körper (Wärme, Wind, ...) ... All diese Dinge werden in Echtzeit berechnet und über eine geeignetes HMI[16] dem Benutzer präsentiert. Der Benutzer taucht also in eine rein am Computer generierte Welt ein und lässt die echte Realität hinter sich. Je nach Ausgereiftheit und Komplexität des verwendeten HMIs funktioniert das mehr oder weniger gut.

Bei der Augmented-Reality wird er Benutzer nicht von der wirklichen Welt abgekoppelt. Ziel ist es viel mehr zusätzliche Informationen direkt in das Gesichtsfeld zu bringen. Es werden also nur Teile der präsentierten Realität im Computer generiert und diese möglichst nahtlos in die echte Realität integriert – die Realität wird „erweitert"[17]. Auch hier werden spezielle HMIs benutzt und auch hier gilt, dass das Verschmelzen von Realität und virtuellen Inhalten umso besser gelingt je komplexer das HMI ist. Im Gegensatz zu VR-Systemen muss ein AR-System seine Umgebung auch erfassen können, um dann zu berechnen wie die virtuellen Dinge eingeblendet werden müssen, um eine schlüssige Darstellung zu generieren. Das ist mehr Aufwand wie bei reinen VR-Devices, womit sich der Preisunterschied erklärt.

Der Grad der Virtualität kann im Realitäts-Virtualitäts-Kontinuum nach Paul Milgram et al. Erfasst werden:

Realitäts-Virtualitäts-Kontinuum (RV)

[18]

Hier ein paar Beispiele zur Verdeutlichung der Begriffe:

- Reine Realität ist ein Mensch, der zum Einkaufen in den Supermarkt geht.
- Erweiterte Realität sind Brillengläser, auf deren Innenseite ein Computer den Einkaufszettel des Benutzers projiziert; und zwar so, dass beim Benutzer der Eindruck entsteht, der Einkaufszettel sei an die Wand des Supermarktes angeschrieben. Die Wirklichkeit wird hier um virtuelle Informationen angereichert.
- Erweiterte Virtualität ist ein Computerspiel, das über einen VR-Helm gespielt wird und das die Türsprechanlage auf die Kopfhörer überträgt, wenn es an der Tür klingelt. Die Virtualität wird hier um reelle Informationen angereichert.
- Virtuelle Realität ist ein Computerspiel, das über einen VR-Helm gespielt wird und nicht auf die reale Außenwelt des Benutzers reagiert.

[16] HMI: human Machine Interface - Benutzerschnittstelle
[17] Engl. Augmented: vergrößert, steigern, verbessern, anreichern
[18] Von Aera - Eigenes Werk, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=78080198

2.2 Virtual Reality

2.2.1 Anforderungen

Video-Link 2: https://youtu.be/klyEDREwnKo

Beim Erstellen einer virtuellen Welt kann man einige Anforderungen definieren, die erfüllt werden sollten. Einige mögliche Anforderungen sind zum Beispiel:

- Immersion:
  Immersion beschreibt die Einbettung des Nutzers in die virtuelle Welt. Die Wahrnehmung der eigenen Person in der realen Welt wird vermindert und der Nutzer fühlt sich mehr als Person in der virtuellen Welt. Dies lässt sich durch eine anspruchsvolle und spannende Gestaltung der virtuellen Welt erreichen, z. B. durch eine hohe Anzahl der möglichen Aktionen in dem System.
- Plausibilität und Interaktivität:
  Eine virtuelle Welt wird als plausibel angesehen, wenn die Interaktion in ihr logisch und stimmig ist. Dies betrifft einerseits das Gefühl des Nutzers, dass die eigenen Aktionen auf die virtuelle Umgebung Einfluss haben, jedoch auch, dass die Ereignisse in der Umgebung die Sinne des Nutzers beeinflussen, er also in der Welt agieren kann. Durch diese Interaktivität wird die Illusion geschaffen, dass das, was zu passieren scheint, tatsächlich passiert.
- Wiedergabetreue:
  Wiedergabetreue wird erreicht, wenn die virtuelle Umgebung genau und naturgetreu gestaltet wird. Dies geschieht, wenn die virtuelle Welt Eigenschaften einer natürlichen Welt abbildet, sie erscheint dem Nutzer dann glaubwürdig.

2.2.2 Hardware

Video-Link 3: https://youtu.be/EotSrulNeaw

Um ein Gefühl der Immersion zu erzeugen, werden zur Darstellung virtueller Welten spezielle Ausgabegeräte namens Virtual-Reality-Headsets benötigt. Bekannt sind vor allem die Oculus Rift, HTC Vive. Ebenfalls möglich sind Großbildleinwände und die CAVE.

 [19]

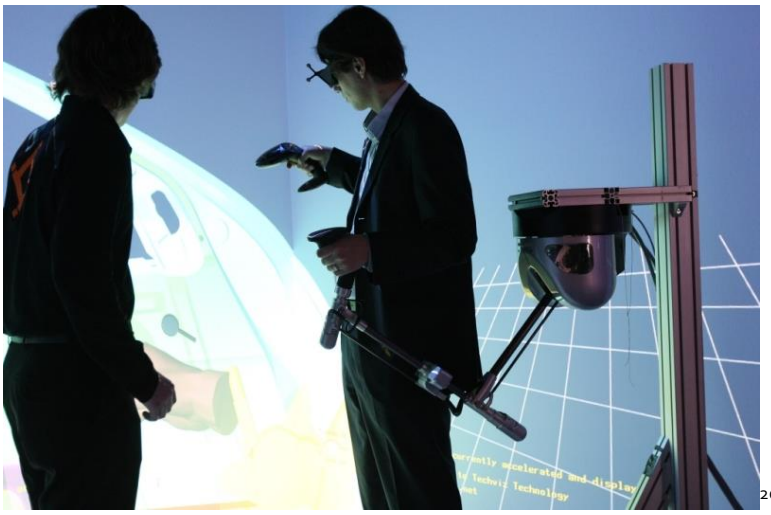
3D-Powerwall

Um einen räumlichen Eindruck zu vermitteln, werden zwei Bilder aus unterschiedlichen Perspektiven erzeugt und dargestellt (Stereoprojektion). Um das jeweilige Bild dem richtigen Auge zuzuführen, existieren verschiedene Technologien. Man unterscheidet aktive (z. B. Shutterbrillen) und passive Technologien (z. B. Polfilter oder Infitec).

[19] Von Der ursprünglich hochladende Benutzer war TMP-Mediagroup in der Wikipedia auf Deutsch - Übertragen aus de.wikipedia nach Commons mithilfe des CommonsHelper., CC BY-SA 2.0 de, https://commons.wikimedia.org/w/index.php?curid=11918646

Für die Interaktion mit der virtuellen Welt werden spezielle Eingabegeräte benötigt. Zu nennen sind hier unter anderem 3D-Maus, Datenhandschuh und Flystick sowie das omnidirektionale Laufband, mit welchem das Gehen im virtuellen Raum durch reale Gehbewegungen gesteuert wird.

Der Flystick wird zur Navigation mit einem optischen Trackingsystem genutzt, wobei Infrarot-Kameras durch Erfassung von Markern am Flystick permanent die Position im Raum an das VR-System melden, damit sich der Benutzer ohne Verkabelung frei bewegen kann. Optische Trackingsysteme können auch für die Erfassung von Werkzeugen und kompletten Menschmodellen eingesetzt werden, um diese innerhalb des VR-Szenarios in Echtzeit manipulieren zu können.

 [20]

Interaktion mit Flystick und 6DoF Force-feedback in Mehrseitenprojektion

Einige Eingabegeräte vermitteln dem Benutzer eine Kraftrückkopplung auf die Hände oder andere Körperteile (Force Feedback), so dass der Mensch sich durch die Haptik und Sensorik als weitere Sinnesempfindung in der dreidimensionalen Welt orientieren und realitätsnahe Simulationen durchführen kann.

2.2.3 Software

Man benötigt zur Erzeugung virtueller Realität speziell für diesen Zweck entwickelte Software. Diese Programme müssen komplexe dreidimensionale Welten in Echtzeit, d. h. mit mindestens 25 Bildern pro Sekunde, in Stereo (getrennt für linkes und rechtes Auge) berechnen können. Dieser Wert variiert je nach Anwendung – eine Fahrsimulation beispielsweise erfordert mindestens 60 Bilder pro Sekunde, um Übelkeit (Simulatorkrankheit) zu vermeiden.

In den 1990er Jahren waren die Rechenleistung als auch die Hardware meist noch nicht ausreichend für den produktiven Einsatz und für realistische Simulationen, deshalb wurden hier meist spezielle Grafikworkstations eingesetzt. Zu Anfang dieses Jahrtausends sind, durch deutlich leistungsfähigere Rechner und Grafikprozessoren, die Möglichkeiten der Interaktion in den Szenarien deutlich gestiegen.

Für die Modellierung von dreidimensionalen, virtuellen Objekten kommen Programme wie Maya, 3D Studio Max, Blender, SketchUp, Softimage XSI, Cinema 4D, LightWave 3D und andere CAD- oder 3D-Programme zur Anwendung. Zusätzliche Software wird für die Bild- und Tonbearbeitung benötigt. Um die dort modellierten Objekte zu interaktiven Simulationen zusammenzusetzen, nutzt man Autorensysteme, wie z. B. World Tool Kit oder World Up.

[20] Von HAPTION - Eigenes Werk, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=15291213

2.2.4 Anwendungen im technischen Bereich

Video-Link 4: https://youtu.be/cqGWOeCytQY

Virtuelle Realität lässt sich in vielen Bereichen einsetzen. Ein sehr bekanntes Einsatzgebiet ist die Pilotenausbildung in Flugsimulatoren. Auch in der Industrie wird diese Technologie verstärkt eingesetzt, vor allem zur Erstellung von virtuellen Prototypen, Produktionsplanungen, virtuellem Training, für ergonomische Bewertungen und räumliche Studien in der Geologie.

 [21]

Flugsimulator mit einer CAVE

Weitere Einsatzgebiete sind Visualisierungen in Architektur, Medizin, Chemie, Energie und Edutainment (z. B. Virtual Cultural Heritage). Der therapeutische Einsatz von virtueller Realität wird unter dem Stichwort virtuelle Rehabilitation untersucht. In der Industrie werden sowohl „Powerwall" als stereoskopische 3D-Wand als auch Mehrseitenprojektionen wie CAVE zum vollständigen Eintauchen in die grafische Simulation genutzt. In den letzten Jahren haben sich in Deutschland und Frankreich eine Reihe von Firmen etabliert, die Virtual-Reality-Software für Industrieunternehmen anbieten, etwa ICIDO, VISENSO und andere.

Virtuelle Realität kann natürliche Arbeitssysteme simulieren. Beschäftigte erleben realitätsnah in einer virtuellen Arbeitsumgebung den Umgang mit simulierten Anlagen, Maschinen und Arbeitsmitteln. Die virtuelle Arbeitsumgebung erscheint dabei in ihrer natürlichen Größe, technische Prozesse laufen kontinuierlich und in Echtzeit. Bewegungen in dieser Umgebung lassen sich von Maschinen und/oder Personen direkt steuern. Perspektive, Blickwinkel und Akustik ändern sich abhängig davon, wo der Mensch steht und wie er sich bewegt[22].

[21] Von TheGrenzebachGroup - Eigenes Werk, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=28172269

[22] An der Schule nutzten wir z.B. im Roboterlabor die Möglichkeit des VR mit einem entsprechenden Headset.

 [23]

Üben von Fallschirmsprüngen an einem Simulator

Mit VR lassen sich alle Phasen des Produktlebenszyklus simulieren, analysieren und optimieren: von der Konstruktion über den Einsatz bis hin zur Entsorgung. Mit VR im Arbeitsschutz kann man

- die Gebrauchstauglichkeit (Usability) von Produkten und Prozessen bereits während ihrer Entwicklung und Konstruktion überprüfen und verbessern. Das vermeidet Fehlentwicklungen und nachträglichen Veränderungsaufwand.
- Gestaltungslösungen zur Mensch-System-Interaktion und ihren Einfluss auf menschliches Verhalten systematisch und empirisch untersuchen. Dies reduziert Umbauten an Maschinen und aufwendige Feldstudien.
- potenziell gefährliche Produkte, Prozesse und Schutzkonzepte gefahrlos testen. Dies verhindert tatsächliche Gefährdungen während der Mensch-System-Interaktion.
- Ursache-Wirkungs-Beziehungen nach Unfällen an und mit Produkten ermitteln. Dies spart materiellen, personellen, zeitlichen und finanziellen Aufwand für Vor-Ort-Untersuchungen

[23] Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=611156

2.3 Augmented Reality

Video-Link 5: https://youtu.be/5yj6CJOcERg

Unter erweiterter Realität (englisch augmented reality kurz AR) versteht man die computergestützte Erweiterung der Realitätswahrnehmung. Diese Information kann alle menschlichen Sinnesmodalitäten ansprechen. Häufig wird jedoch unter erweiterter Realität nur die visuelle Darstellung von Informationen verstanden, also die Ergänzung von Bildern oder Videos mit computergenerierten Zusatzinformationen oder virtuellen Objekten mittels Einblendung/Überlagerung. Bei Fußball-Übertragungen ist erweiterte Realität beispielsweise das Einblenden von Entfernungen bei Freistößen mithilfe eines Kreises oder einer Linie.

Im Realitäts-Virtualitäts-Kontinuum sind die erweiterte Realität (augmented reality, AR) und erweiterte Virtualität (augmented virtuality) Teil der sogenannten gemischten Realität (mixed reality). Während der Begriff Augmented Virtuality kaum von der Fachwelt benutzt wird, werden Augmented Reality und Mixed Reality, selten auch Enhanced Reality, meist synonym verwendet.

Unter einem AR-System (kurz ARS) versteht man das System der technischen Bestandteile, die nötig sind, um eine Augmented-Reality-Anwendung aufzubauen: Kamera, Trackinggeräte, Unterstützungssoftware usw.

2.3.1 Anwendungen

Video-Link 6: https://youtu.be/cqGWOeCytQY

Erweiterte Realität könnte in praktisch allen Bereichen des Alltags zum Einsatz kommen. Monteure könnten sich den nächsten Arbeitsschritt direkt in ihr Sichtfeld einblenden lassen; Soldaten oder Katastrophenhelfer könnten sich Ziele und Gefahrenzonen im Gelände anzeigen lassen und Designer könnten mit tatsächlich und virtuell anwesenden Kollegen am selben dreidimensionalen Modell arbeiten.

[24]

Mit fortschreitender Technologie lassen sich futuristische Anwendungsszenarien erschließen: Elektronische Geräte, die nur virtuell existieren, aber auf echte Berührungen reagieren, künstliche Sinneserweiterungen wie den „Röntgenblick" und Computerspiele in freiem Gelände.

Ein Beispiel für eine AR-Anwendung sind die in Echtzeit eingeblendeten virtuellen Marken bei Sportübertragungen: Verschiedene Entfernungen der Konkurrenten beim Skispringen, Weitwurf usw. (Man beachte, dass dieses Beispiel oft keine Augmented-Reality-Anwendung nach der obigen Definition ist, da manchmal das interaktive Element fehlt.)

Mögliche Anwendungsgebiete sind:

[24] Von Quest Visual, Inc.ZTebaykina at en.wikipedia - Word Lens demoTransferred from en.wikipedia by Ronhjones, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=18310222

- Hilfestellung bei komplexen Aufgaben, v. a. in Konstruktion, Wartung und Medizin:
  Durch Anzeigen von Zusatzinformationen kann eine Hilfestellung bei komplexen Aufgaben geschehen. Zum Beispiel werden für einen Mechaniker die Teile eines Gerätes „beschriftet", und er bekommt Arbeitsanweisungen.
  In der Medizin kann erweiterte Realität genutzt werden, um die Darstellung nicht sichtbarer Elemente zu ermöglichen. Zum Beispiel kann dies intraoperativ geschehen, als „Röntgenblick" für den Operateur, basierend auf vorheriger Tomographie oder aktuellen Bilddaten von Ultraschallgeräten oder offenen Kernspintomografen.
- Industrielle Anwendungen:
  Mit Augmented Reality können digitale Planungsdaten effizient mit vorhandenen realen Geometrien abgeglichen werden. Die Technik ermöglicht ferner den breiten Einsatz von digitalen Absicherungsmethoden bei der Kombination von digitalen Daten mit realen Prototypen bzw. Konstruktionen.
- Navigation:
  Erweiterte Realität kann grundsätzlich die Navigation im Gebäude (bei der Wartung von Industrieanlagen), im Freien (für Militär oder Katastrophenmanagement), im Auto (Projektion von Navigationshinweisen an die Windschutzscheibe, so dass beispielsweise Abbiegehinweise auf der Fahrbahn erscheinen) oder im Flugzeug (Head-Up-Displays in Kampfflugzeugen sind eine der frühesten AR-Anwendungen überhaupt) genutzt werden.
- Digitalkameras:
  Bei Digitalkameras mit Live-View-Suchern und -Bildschirmen kann zusätzlich zum Motiv errechnete Information eingeblendet werden, wie zum Beispiel für erkannte Gesichter, für scharf gestellte Kanten oder für fehlerhaft belichtete Bildbereiche. Zur Ausrichtung von Bild- oder Motivkanten können Gitterlinien oder elektronische Wasserwaagen eingeblendet werden.
- Kunst:
  AR in der bildenden Kunst ermöglicht es Objekten oder Orten, künstlerische multidimensionale Erfahrungen und Interpretationen der Realität auszulösen. Das nicht fertiggestellte Denkmal des Künstlers Benno Elkan wurde virtuell rekonstruiert und im Museum für Kunst und Kulturgeschichte Dortmund ausgestellt. Mit einer Smartphone-App kann das virtuelle Denkmal von allen Seiten betrachtet werden.
- Militär und Katastrophenmanagement:
  Im Bereich Militär und Katastrophenmanagement können tragbare Systeme verwendet werden, die etwa Freund und Feind oder Brandherde anzeigen.
- Hydrologie, Ökologie, Geologie:
  Systeme können für die Prospektion, für die Darstellung und die interaktive Analyse von Karten und Geländemerkmalen genutzt werden, um beispielsweise Bodenschätze auszubeuten.
- Architektur:
  Erweiterte Realität eignet sich ebenfalls für die Visualisierung von Architektur. So können zerstörte historische Gebäude oder auch zukünftige Architekturprojekte dargestellt werden.
- Simulation:
  Auch für Flug- und Fahrsimulatoren kann erweiterte Realität eingesetzt werden.
- Zusammenarbeit verteilter Teams:
  Die Zusammenarbeit örtlich verteilter Teams kann erleichtert werden. Zum Beispiel durch Video-Konferenzen mit realen und virtuellen Teilnehmern. Aber auch die gemeinsame Arbeit an simulierten 3D-Modellen wird so unterstützt.

- Werbung:
  Zunehmend setzen Unternehmen in ihrer Werbung auf AR-Komponenten, um dem Kunden einen Mehrwert zu bieten. So veröffentlichte beispielsweise die Möbelhauskette IKEA 2013 einen Katalog, in dem ausgewählte Möbelstücke per Smartphone-App eingescannt und virtuell an einen beliebigen Platz in der Wohnung projiziert werden konnten.
- Lernen:
  Erweiterte Realität verfügt auch im Lernbereich über ein großes Potenzial. Es besteht die Möglichkeit, insbesondere durch mobile Applikationen, digitale Layer auf reale Welten zu projizieren und nahtlos in die Realität zu integrieren. Besonders Applikationen mit Animationen erlauben ein interaktives Erschließen laufender Prozesse. Somit werden vor allem abstrakte Konzepte, welche in traditionellen Lernformen teilweise nur einseitig betrachtet werden können, greifbarer.
  Ein Beispiel für AR-Lernen durch selbständiges Erkunden ist die App „Timetraveler Berlin Wall", welche historische Ereignisse in die Umgebung der heutigen Welt integriert.
  Kritiker befürchten, dass bei einem unreflektierten Einsatz von AR-Anwendungen lineare Wahrnehmungsmodi eingeübt werden, die eine ‚natürliche', dynamische Wahrnehmung und Interaktion mit der Welt unterbinden.
- Fernsehen und Sport:
  Einige Fernseh- und Sportübertragungen setzen auf visuelle Informationsgrafiken, die in Form einer Erweiterung der Realität auf den Bildschirm projiziert werden und z. B. Informationen über das Spiel vermitteln.
  AR kann auch als Erweiterung für das Ausüben einer Sportart dienen, so wird beim Augmented Climbing beispielsweise die Kletterwand durch Lichtprojektionen zu einem Spiel.

2.3.2 Herausforderungen

Video-Link 7: https://youtu.be/lTj5UcUfVBM

Ein Problem ist die technische Belastung bei erweiterter Realität, besonders die Nachführung der Bilder bei Bewegungen. Auch die Sensoren werden durch die Bewegung beeinträchtigt. So gibt es Rauschen, Drift und Abschattung des Trackingsystems (beispielsweise bei GPS, INS). Eine Kombination von beispielsweise GPS mit Trägheits- und optischer Navigation ist daher bei fortgeschrittenen Systemen üblich.

Ein weiteres Problem stellt die Energieversorgung dar. Die momentan verfügbaren Akkus reichen noch nicht aus, um mobile Augmented-Reality-Systeme längere Zeit (mehr als ein paar Stunden) zu versorgen. Auch die Verfügbarkeit von Daten, Authoring und hohe Komplexität von Daten können zu Problemen führen.

Um die Einbettung der virtuellen Szene in die reale Szene möglichst überzeugend zu leisten, sind Daten notwendig, die die Umgebung auch in ihrer Geometrie beschreiben. Darauf aufbauend können dann virtuelle Schnitte durch reale Objekte gezeichnet und die Verdeckung der virtuellen Objekte durch die realen Objekte berechnet werden. Diese Geometriedaten sind jedoch nicht immer verfügbar oder aktuell.

Die vollständige Integration virtueller Objekte in reale Szenen erfordert das Ausblenden von Hintergrundteilen, damit die Objekte nicht durchsichtig erscheinen. Systeme, die die direkte Sicht vollständig durch Kamerabilder ersetzen (EyeTap) haben dieses Problem nicht, sind aber für viele Anwendungen ungeeignet.

2.3.3 Zukunft

Als zukünftige Anwendungen werden einige Beispiele hier aufgeführt. Zum einen kann eine Erweiterung von PC-Betriebssystemoberflächen in die reale Umwelt geschehen. Programmfenster und Icons werden dann als virtuelle Geräte im realen Raum dargestellt und durch Blicke oder Zeigen mit dem Finger bedient. Dies kann generell zum Ersatz herkömmlicher Bildschirme (Ersatz von Handy- und Navigatorbildschirmen und Einblendung der Informationen direkt in die Umwelt, beispielsweise von Leitlinien direkt auf die Fahrbahn, sowie Erweiterungen, wie beispielsweise „Röntgenblick" zur Darstellung verdeckter Ziele), Gerätebedienfelder, oder zu völlig neuen Gerätetypen führen.

Außerdem kann erweiterte Realität für multimediale Anwendungen, wie pseudo-holografische virtuelle Bildschirme, virtuelle „Holodecks", virtuelles Surround-Kino genutzt werden. Aber auch zur Verschönerung der alltäglichen Umwelt, wie durch die Darstellung virtueller Pflanzen, Tapeten, Ausblicke, Kunstwerke, Dekorationen, Beleuchtung usw., wären Anwendungen denkbar. Bei allgemeiner Verbreitung von AR-Systemen könnte man auch virtuelle Schaufenster, Plakate oder Verkehrsschilder nutzen.

Möglich wäre zudem ein Zusammenwachsen von Virtual und Augmented Reality, sodass der User auf einem Endgerät zwischen den Formen wechseln kann.

2.4 Vuforia Studio

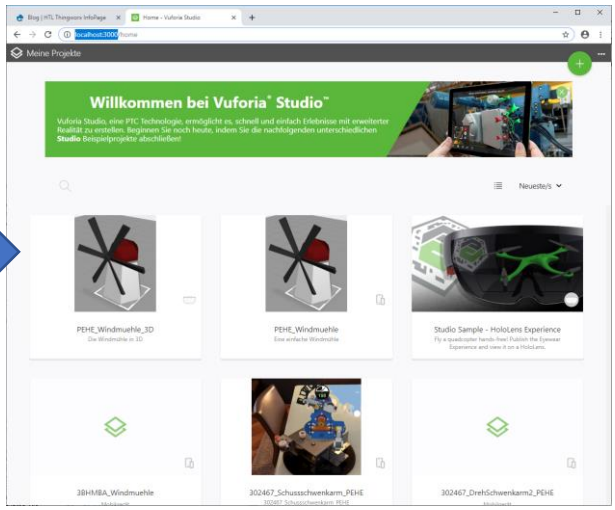
2.4.1 Überblick

Video-Link 8: https://youtu.be/52dkOzo8JUY

Das Softwarepakt Vuforia Studio von PTC ist speziell auf die Erstellung von AR-Inhalten ausgelegt. Wir werden im Unterricht Vuforia verwenden, um unsere eigene Augmented Reality zu schaffen. Dazu müssen wir Vuforia auf unserm PC installieren.

Die Anwendung lädt die erforderlichen Dinge aus dem Internet herunter und installiert Vuforia Studio auf dem PC. Nach erfolgreicher Installation (hier kann man nicht viel einstellen, das installiert sich so wie es will). Kann Vuforia Studio auch schon gestartet werden.

Vuforia Applikation:
erzeugt Webserver mit Vuforia Funktionalität



Webbrowser:
stellt die von Vuforia generierte Webseite dar – hier kann man Vuforia bedienen. Die Adresse ist:
localhost:3000

Zunächst ist das Ergebnis ernüchternd. Nach ein paar Sekunden erschient ein Fenster und das war's. Dann gibt es noch ein Button „Öffnen" welcher den Webbrowser startet. Warum ist das so?

Nun, Vuforia Studio startet auf unserem Rechner einen Webserver. Der Webbrowser ersetzt das Programmieren einer eignen Anzeige-Applikation. Und mit dem Webbrowser wird die von diesem Server zur Verfügung gestellte Webseite dargestellt. Die Seite ist über den Port 3000 zu erreichen und weil es sich um eine Seite auf unserm lokalen Rechner handelt, ist die richtige Adresse im Webbrowser: localhost:3000

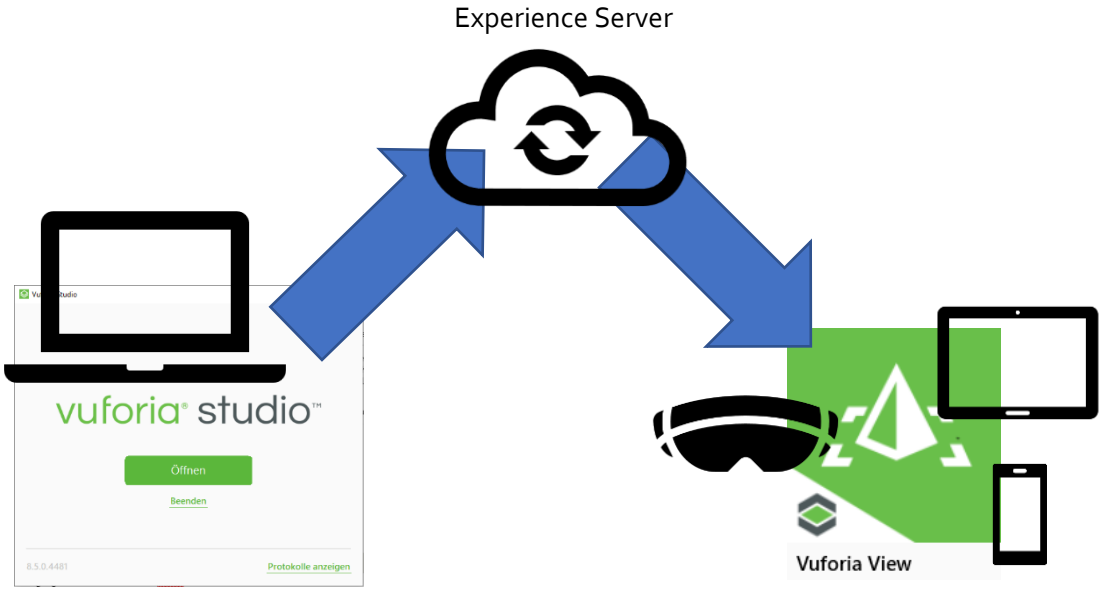
Beim Druck auf den Button „Öffnen" (der nichts anderes ist als ein Hyperlink[25]) wird der Standard-Webbrowser gestartet und die oben genannte Seite dargestellt. Zu beachten ist, dass PTC offiziell nur Google Chrome unterstützt. Bei anderen Webbrowsern kommt dann der Hinweis, dass es sich um einen nicht unterstützten Browser handelt. Erfahrungen haben gezeigt, dass es tatsächlich zu Funktionseinschränkungen bei Verwendung eines anderen Browsers kommen kann. Ist es nicht erwünscht Google Chrome als Standrad-Browser zu verwenden kann man Chrome trotzdem installieren und dann muss man dort eben die Adresse eingeben und kann nicht einfach bequem auf Öffnen klicken – wenn man sich ein Bookmark anlegt, ist es praktisch kein Mehraufwand.

[25] Ein Hyperlink ist nichts anderes als ein „normaler" Internetlink.

2.4.2 Funktionsprinzip

Im Vuforia Studio wird ein Projekt erstellt. Im Prinzip muss man irgendwie einen Referenzpunkt in der Realität schaffen. Das soll heißen: Man gibt in Vuforia Studio einen Bezugspunkt an. Danach baue man seine virtuellen Objekte rund um diesen Bezugspunkt ein. Damit ist festgelegt, wie und in welcher Orientierung sich diese virtuellen Objekte befinden. Wenn man später den Bezugspunkt in der Realität festlegt, hat man auch festgelegt wo die virtuellen Dinge eingeblendet werden. Dieser Bezugspunkt ist also das Bindeglied zwischen der Projektierung und der Realität. Im Vuforia-Sprachgebrauch nennt sich solch ein Referenzpunkt „Target". Die virtuellen Objekte müssen als 3D-Modell verfügbar sein. Vuforia unterstützt nur das PTC-Format `pvz`. Glücklicherweise können andere Formate (z.B. SolidWorks) importiert werden.

Wir definieren also ein Target und bauen unsere Virtuellen Objekte darum auf. Dann muss diese Information noch irgendwie zu unserem HMI kommen, welches uns die virtuellen Objekte tatsächlich in die Realität einblendet. Das geschieht mit der sogenannten „Veröffentlichung". Wir nehmen also unsere so erstellte „Experience" (so nennt Vuforia ein Projekt) und veröffentlichen die Daten auf einem Experience-Server. Mit unserem Gerät, mit dem wir die Experience ansehen wollen, verbinden wir uns mit dem Server, laden die Experience herunter und können sie genießen. Dazu benötigt man auf dem Gerät eine Applikation deren Name „Vuforia View" ist. Diese kann die Daten auf dem Experience-Server verstehen und erledigt die Darstellung.



Die Applikation Vuforia View ist dabei für Windows 10, iOS, Android und die Microsoft HoloLens verfügbar. Entsprechende Geräte werden je nach Ausstattung unterstützt. Ein Beispiel, bei dem eine einfaches Modell einer Windmühle auf einem Android-Smartphone verwendet wurde, ist im Bild auf der Seite zu sehen. Die virtuelle Windmühle wurde auf den realen Tisch platziert.

Wir benötigen also einen Experience-Server, auf dem wir unsere Projekte veröffentlichen können. Dieser muss von der Schule zur Verfügung gestellt werden. Die Adresse und die Login-Daten sind beim unterrichtenden Lehrer zu besorgen. Im Schuljahr 2021/22 lautete die richtige Adresse[26]:

https://example.twx.htl.schule

Username: **AdminUser**

Passwort: **TWXpasswort**

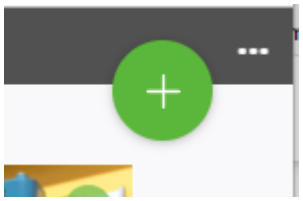
Da wir alle auf einem Experience-Sever veröffentlichen müssen wir uns an Namenskonventionen halten. Alle benennen Ihre Experience nachfolgendem Schema:

KLASSE_NACHNAME_VORNAME

Also z.B.: **3AHMBA_PETERSCHOFSKY_HEINZ**

2.4.3 Erstes Projekt für ein Handheld-Device (Handy)

Video-Link 9: https://youtu.be/ooTlFPyPego



Das Vuforia Studio lernen wir am besten mit einem eigenen Projekt. Zuerst starten wir ganz einfach, indem wir einfach ein Objekt von uns in die reale Welt projizieren lassen. Dazu müssen wir eine neue Experience erstellen. Das geschieht mit dem +-Icon im rechten oberen Rand. Wird dieses ausgewählt müssen wir uns zunächst für einen Projekttyp entschieden, Dabei haben wir folgende Möglichkeiten:

2D-Eyewear
Brillen, die einfach eine zweidimensionale Zusatz-information einblenden können.

3D-Eyeware
3D-Brillen, wie zum Beispiel die später verwendete Microsoft HoloLens

Mobile
Handheld Devices wie Tablets und Smartphones

Wir wollen zunächst ein Projekt für unser Smartphone erstellen. Deswegen verwend en wir die Option „Mobile". Dabei werden wir aufgefordert einen Projektnamen zu vergeben. Wir werden die

[26] Sollte diese Adresse nicht mehr funktionieren wurde der Server geändert. Dan sind die Login-Daten beim unterrichtenden Lehrer zu besorgen.

oben genannte Namenskonvention befolgen und vergeben einen entsprechenden Namen. Im Beispiel verwende ich **302467_3AHMBA_PETERSCHOFSKY_HEINZ**.

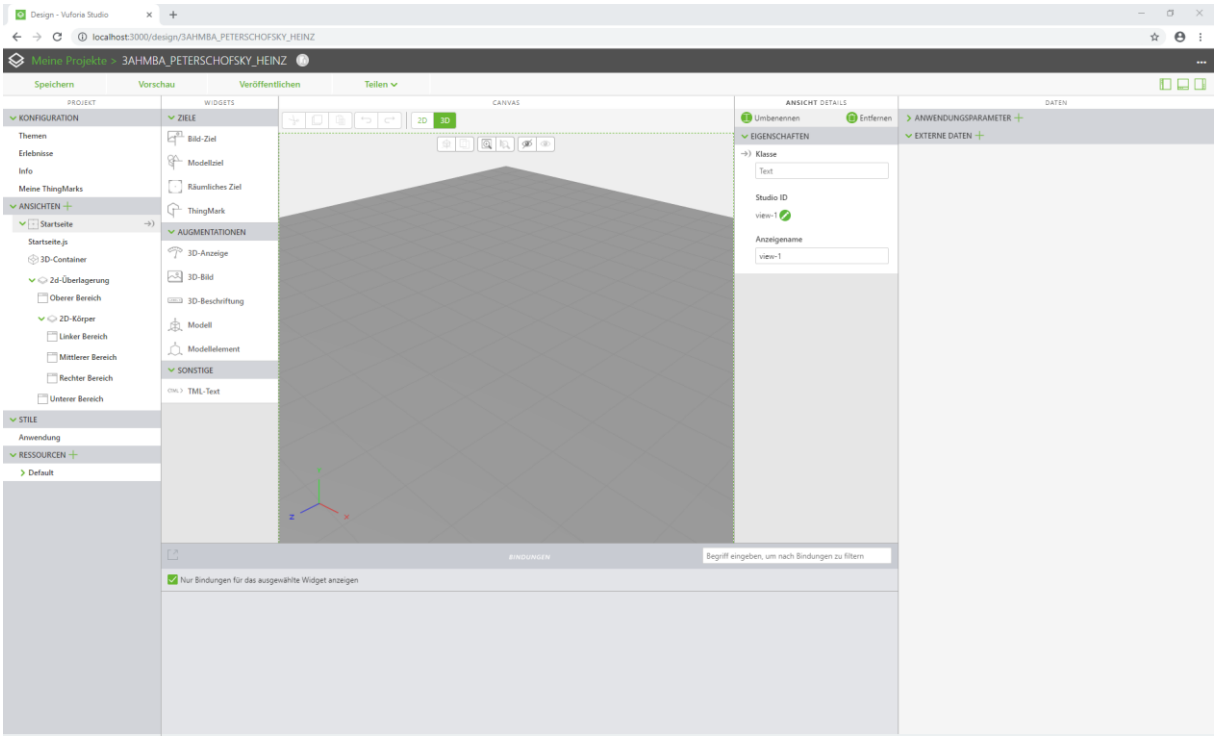
Wir werden ebenso aufgefordert eine Experience Service URL einzugeben – das ist der Link zu unserem Experience Server am Port 8443. Mit oben genanntem Link schreiben wir dort also hinein:

https://example.twx.htl.schule:8443

Danach kann der Knopf „Erstellen" betätigt werden und das Projekt wird angelegt. Alle Projekte werden an einem Ort abgelegt, und zwar im Ordner „Eigene Dateien" des aktuellen Benutzers. Sollte das Default-Verzeichnis der Eigenen Dateien dabei woanders sein wird dies außer Acht gelassen. Das dürfte irgendwo fix programmiert sein. Es handelt sich also immer um das Verzeichnis:

```
c:\Users\Benutzer\Documents\VuforiaStudio\Projects\
```

Nach dem Anlegen sieht man die Benutzeroberfläche des Vuforia Studios. In der Mitte erscheint der „Canvas". Dort werden die virtuellen Objekte dargestellt und manipuliert. Links gibt es einen Abschnitt mit dem Namen „Projekt". Das ist praktisch das Hauptmenü. Unmittelbar daneben gibt es den „Widgets"-Bereich. Der kann verwendet werden, um neue Objekte einzufügen. Im rechten Bereich gibt es noch Detail-Informationen.



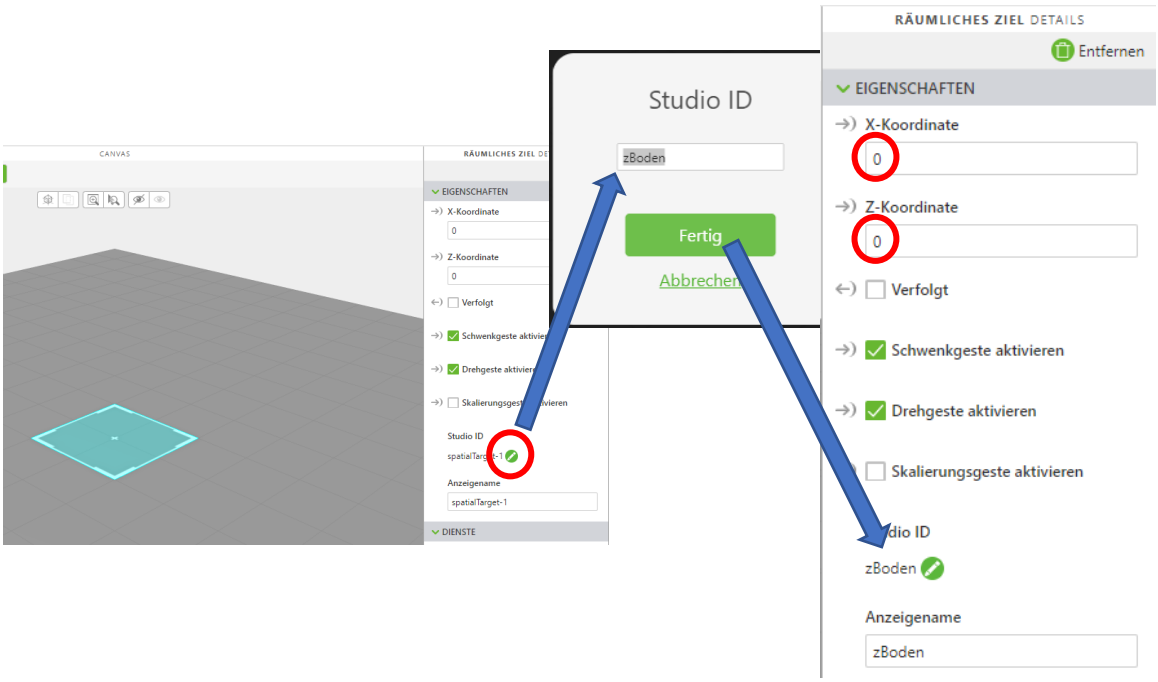
Das Erste was hinzugefügt werden muss ist ein sogenanntes „Target". Also der Referenzpunkt, der der „Ursprung" der virtuellen Einblendungen ist. Im „Widgets"-Bereich gibt es dafür die Rubrik „Ziele". Dabei gibt es folgende Zielarten:

- **Bild-Ziel:** Versucht ein Bild zu erkennen. Wird das dort angegebene Bild in der Realität erkannt so werden die virtuellen Teile entsprechend hinzugefügt. Dabei funktionieren Bilder mit starken Kontrasten - wie z.B. technische Zeichnungen - am besten. Damit könnte man also eine Zeichnung filmen und schon entsteht über der Zeichnung das 3D-Modell des Teils (oder auch der Einbauort des Teils).
- **Modellziel:** Versucht eine dreidimensionale Struktur zu finden. Wird diese Struktur erkannt so werden die virtuellen Teile hinzu projiziert. Man könnte also z.B. den Sockel eines Teils filmen und wenn der Sockel als Modellziel erkannt wird, wird der Teil virtuell dazu projiziert.
- **Räumliches Ziel:** Ist praktisch eine ebene Fläche. In der Realität sucht man sich eine Fläche aus und sagt: Das ist die, welche mit dem projektierten räumlichen Zeil übereinstimmt. Dann werden die virtuellen Objekte entsprechend dargestellt.
- **ThingMark:** Ist ein 2D-Code. Wird dieser erkannt (z.B. auf einem Ausdruck) werden die virtuellen Dinge entsprechend hinzuprojiziert. Jeder ThingMark hat eine Nummer. Es muss der ThingMark erkannt werden der projektiert ist. Außerdem wird die Skalierung angepasst. Im Projekt sagt man z.B. der ThingMark ist $10cm$ breit. Ist er in der Realität aber mit einer Breite von $11cm$ ausgedruckt so wird das Modell auch auf 110% skaliert[27].

Wir werden für unseren Zweck einmal räumliches Ziel verwenden. Wir ziehen das Räumliche Ziel vom Widgets-Bereich in den Canvas-Bereich und lassen es dort los.

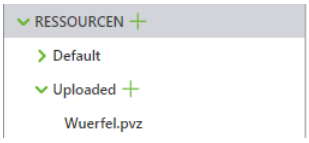
Damit „liegt" das Ziel jetzt einmal auf unserm virtuellen Boden. Es sollte auch bereits ausgewählt sein. Dann werden die Eigenschaften des Ziels im Details-Bereich angezeigt. Wir werden den Namen das Ziels noch anpassen. Das geschieht mit dem grünen Bleistift-Symbol im Details-Bereich (jetzt steht dort wahrscheinlich „`spatialTarget-1`". Wir werden dieses Zeil jetzt „`zBoden`" nennen. Das ist gleichzeitig die sogenannte Studio-ID. Das wird später wichtig, wenn wir ein paar Sachen programmieren wollen.

[27] Bei Verwendung der Microsoft HoloLens erfolgt zusätzlich eine Verschiebung des Modells. Hier ist also besonders auf die richtige Markengröße zu achten.

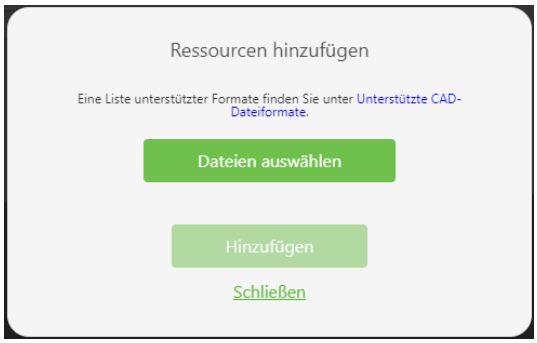


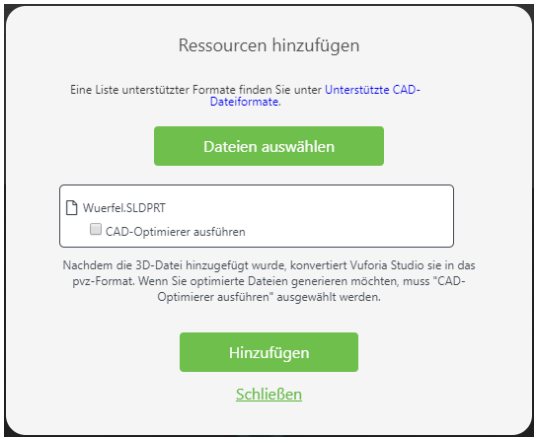
Außerdem wollen wir darauf achten, dass das Zeil im Ursprung liegt. Die Maßeinheiten sind hier immer Meter. Wenn wir also irgendwo 1,1 eingaben, dann entspricht das einem Abstand von $110cm$.

Nachdem wir also ein Ziel definiert haben wollen wir ein Modell hinzufügen. Dazu benötigen wir ein 3D-Objekt. Dieses Objekt kann z.B. auch ein Assembly sein. Ich würde jedoch einmal klein anfangen und ein kleines, einfaches Objekt nehmen. Bei mir ist dies ein kleiner roter Spielewürfel, den ich mir in SolidWorks modelliert habe. Den muss ich jetzt irgendwie in das Vuforia-Studio bekommen.

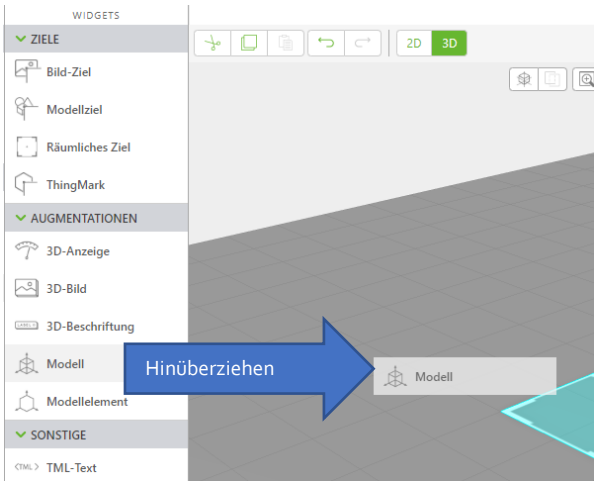


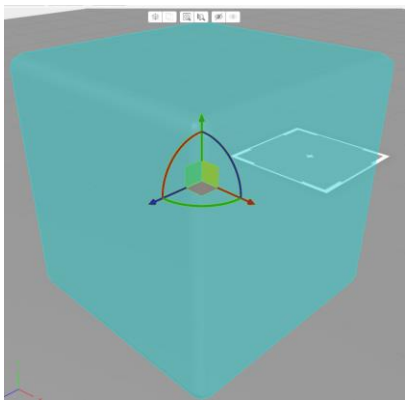
Dinge, die „von außen" kommen heißen dort „Ressourcen" und können links unten gefunden werden. Dort gibt es noch nichts, deswegen müssen wir dort das „+"-Symbol drücken. Es öffnet sich der „Ressourcen hinzufügen"-Dialog. Mit „Dateien auswählen" können wir die passende Datei mit dem 3D-Modell hinzufügen. Wenn wir die Dateien jetzt hinzugefügt haben, gibt es noch die Möglichkeit eine „CAD-Optimierer" auszuführen. Dieser legt statt eines Modells deren drei an und reduziert bei zweien davon die Polygonzahl. Bei sehr großen Modellen kann dies sinnvoll sein, bei unseren kleinen Objekten ist dies egal. Wir werden den CAD-Optimierer also nicht ausführen. Mit dem Klick auf „Hinzufügen" wird das Modell in ein pvz-Datei umgewandelt. Das funktioniert ganz gut, nur wenn Teile nicht gefunden werden, dann werden die auch nicht konvertiert. Besonders bei Standard-Teilen ist dies oft ein Problem. Man erkannt eine solche Schwierigkeit aber relativ schnell. Nach erfolgreicher Konvertierung findet man die neue Ressource links unten eingeblendet.

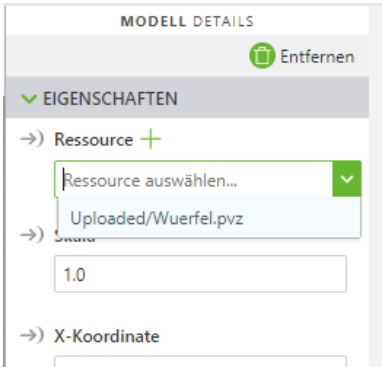




Jetzt müssen wir diesen Würfel nur mehr anzeigen. Dazu fügen wir in unseren Canvas ein „Modell" hinzu, indem wir es wieder einfach hineinziehen. Dort erscheint dann ein Modellplatzhalter (ein großer Würfel mit abgerundeten Ecken).

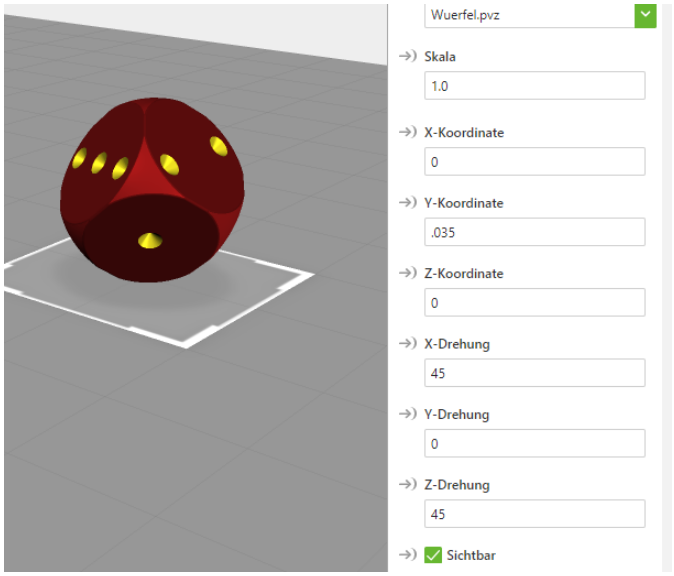






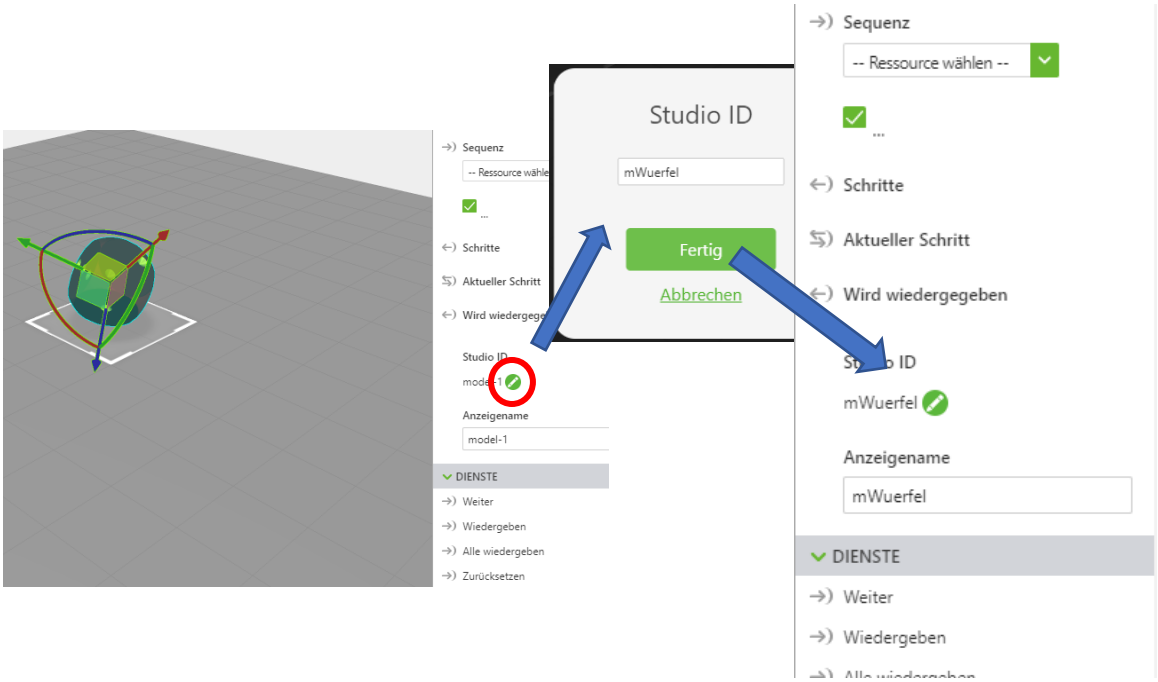
Wir müssen diesem Platzhalter nun unsere Ressource zuweisen. Das passiert im Eigenschaftsbereich ganz oben. Ist dies passiert, so erschient auch schon unser Modell und nicht mehr der Platzhalter. Man kann sofort erkennen, ob der Import des Modells erfolgreich war. Fehlen Teile so konnten die entsprechenden Daten nicht gefunden werden. Das kann bei Assemblies passieren, wo die Part-Dateien nicht zugänglich sind. Passiert so etwas muss man ein wenig herumprobieren. Normalerweise findet man einen Weg. Jetzt ist auch die Zeit mit den Werten für X-, Y- und Z-Koordinate sowie der Drehung zu spielen. Das Ergebnis sieht man unmittelbar auf dem Bildschirm.

Ich habe meinen Würfel auf den Koordinaten $(x; y; z) = (0; 0{,}035; 0)$ mir der Drehung $(rx; ry; rz) = (45°; 0°; 45°)$ gesetzt. Damit sieht mein Würfel aus, wie wenn er auf einer Spitze stehen würde.

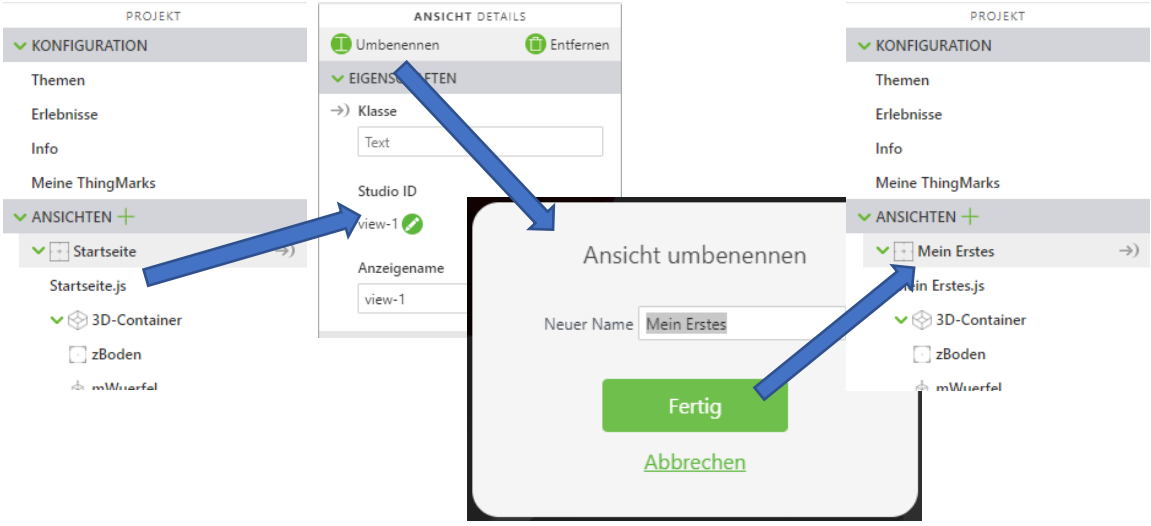


Obwohl wir in dieser Übung nur ein Modell verwenden werden, benennen wir dieses – in Zukunft haben wir es unter Umständen mit mehr als einem Modell zu tun, und dann haben wir keine Ahnung mehr ob sich hinter „`model-12`" jetzt der Deckel oder das Lager verbirgt. Deswegen werden wir

das sauber durchziehen. Unser Modell heißt normalerweise jetzt „`model-1`". Wie benennen wir dieses um? Im rechten Bereich – unter den Koordinaten gibt es wieder die „Studio-ID". Diese ändern wir mit einem Klick auf das gründe Bleistift-Symbol. Genauso wie bei der Namensvergabe des Ziels. Ich verwende „`mWuerfel`" (beachte keine Umlaute!)

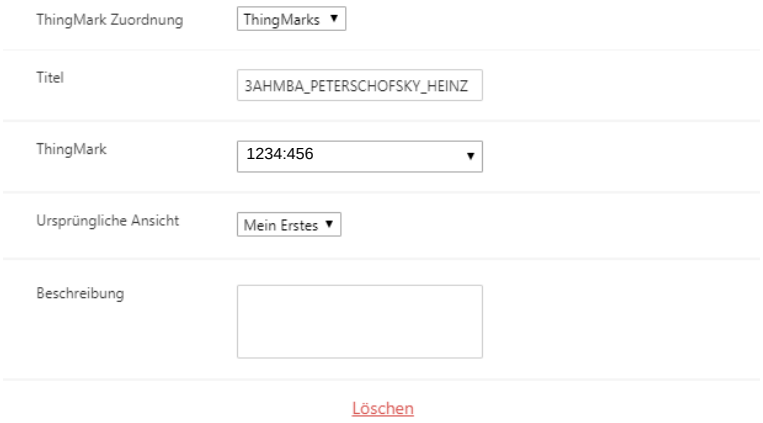


Jetzt geben wir unserer „Ansicht" noch einen passenden Namen. Ansichten sind praktisch „Unterprojekte". In jedem Projekt können verschiedene Ansichten verwendende werden. Wir werden die Ansichten verwenden, um unsere verschiedenen Übungen in einem Projekt abzuhandeln. Im Moment heißt die einzige Ansicht vermutlich „Startseite". Um diese umzubenennen klicken wir im linken Bereich auf die Ansicht. Im Details-Bereich werden dann die Details der Ansicht dargestellt. Dort können wir auf „Umbenennen" klicken.



Es öffnet sich ein Dialogfenster und wir wählen einen besseren Namen. Ich wähle hier einfach „`Mein Erstes`". Mit dem Klick auf „Fertig" wird die Ansicht umbenannt.

Das reicht uns für diese erste Übung. Wir wollen dieses Projekt jetzt als Experience zur Verfügung stellen. Wir haben ja bereits den Experience-Server eingetragen. Jetzt ist es Zeit noch ein paar Daten zu ergänzen und dann die Sache zu Veröffentlichen. Was müssen wir noch ergänzen? Im Projekt Baum unter Erlebnisse legen wir jetzt ein ThingMark fest unter dem wir unsere Experience finden wollen. Außerdem geben wir eine kurze Beschreibung. Jedes ThingMark hat eine Nummer, und diese muss im entsprechenden Feld eingetragen werden.

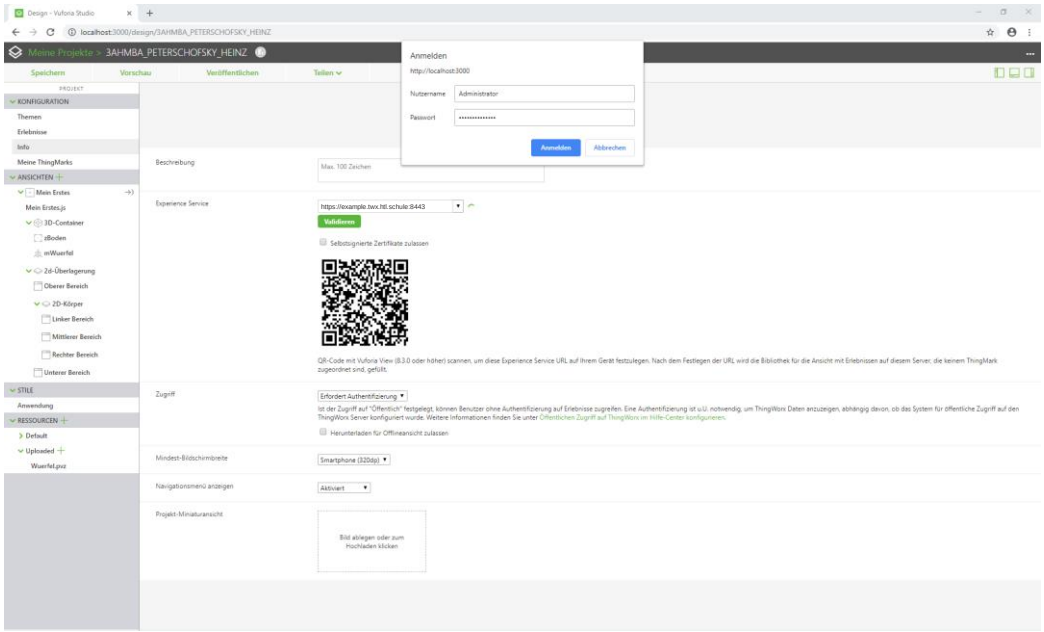


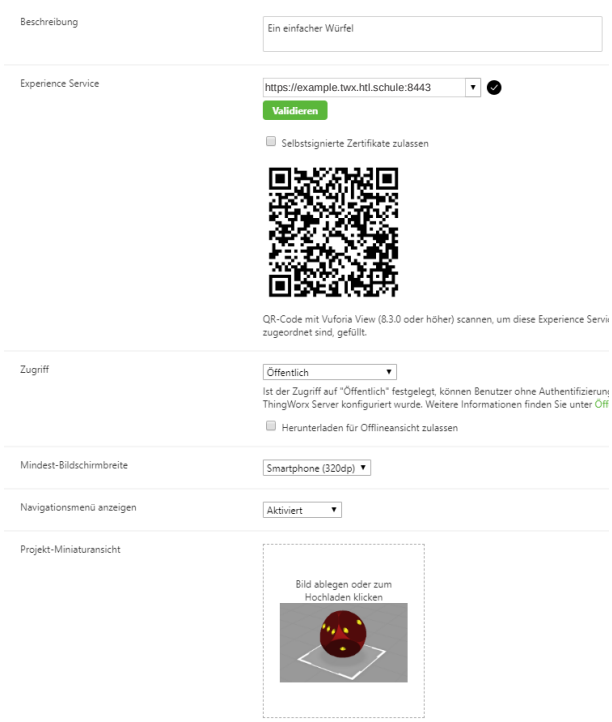
Nun können wir unsere Experience (Projekt) zur Verfügung stellen. Dazu muss der entsprechende Experience-Server jedoch erreichbar sein. Deswegen müssen wir noch den Experience-Server testen. Dazu wechseln wir zum Bereich „Info". Dort versuchen wir zunächst den Experience-Server zu erreichen, indem wir auf „Validieren" klicken. Damit erreichen wir eine Anmelde-Maske, in der wir den Usernamen und das Passwort eingeben müssen. Zur Erinnerung: Im Moment lauten die Daten:

Username: **AdminUser**

Passwort: **TWXpasswort**

Diese Information eingegeben und auf Anmelden gedrückt.





Nach erfolgreicher Validierung erschient neben der Experience-URL ein Häkchen – die Verbindung und die Logindaten stimmen. Danach sollten wir noch eine aussagekräftige Beschreibung und ein ebensolches Bild in der Projektminiaturansicht hinzufügenden. Ich habe mich für einen Screenshot per Snipping-Tool entschieden.

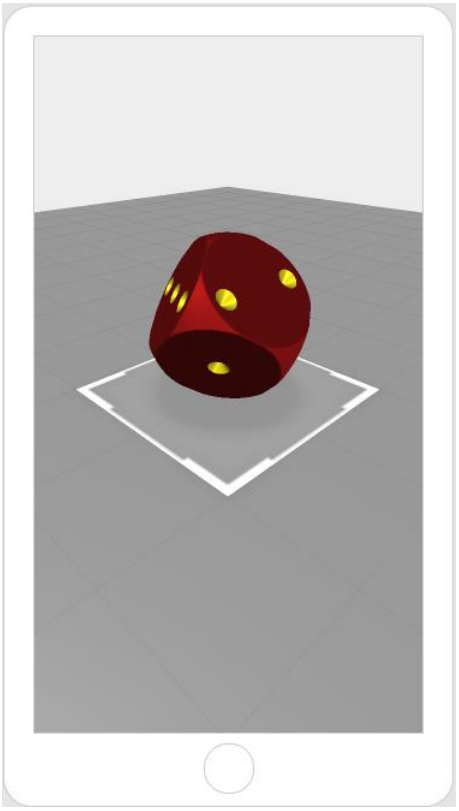
Der Zugriff kann auf „Öffentlich" gestellt werden. Das bedeutet, dass jeder, der den vorhin eingegeben ThingMark hat auch die Experience vom Sever laden darf. Mit „Erfordert Authentifizierung" müsste man Username und Passwort angeben. Das ist im Moment übertrieben.

Damit sind wir bereit zur Veröffentlichung. Man kann vorher noch auf „Vorschau" am oberen Balken klicken. Es wird ein neuer Tab geöffnet, der das Projekt simuliert. Bei komplexeren Projekten macht dieses Feature mehr Sinn. Jetzt sieht es irgendwie fad aus und unterschiedet sich kaum von der Projektierungsumgebung.

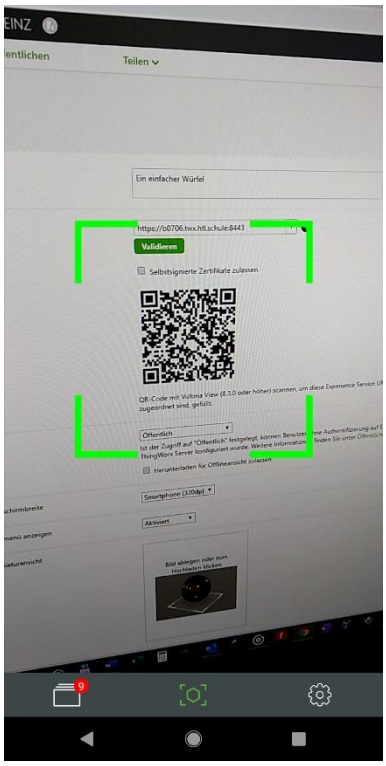
Also versuchen wir es: Klicken wir auf „Veröffentlichen"! Es erscheint kurz ein Balken mit einer Fortschrittsanzeige und dann ist alles schon wieder vorbei.

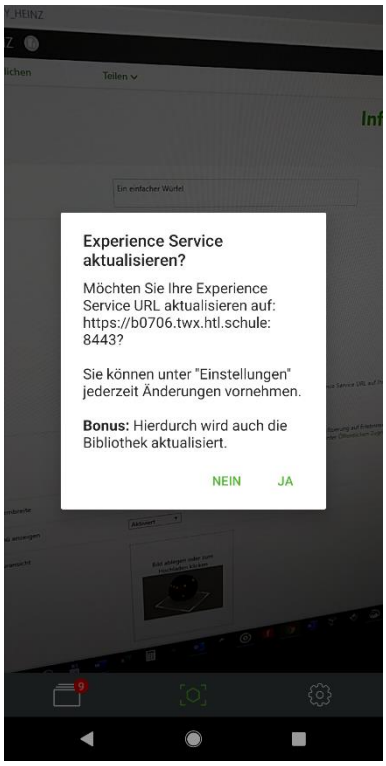
Offensichtlich ist etwas passiert. Wie geht es jetzt weiter?

Auf unserem Rechner sind wir fertig. Alles ist getan. Kommen wir zu einem Smartphone oder Tablet. Dort installieren wir Vuforia View (im App-Store suchen).

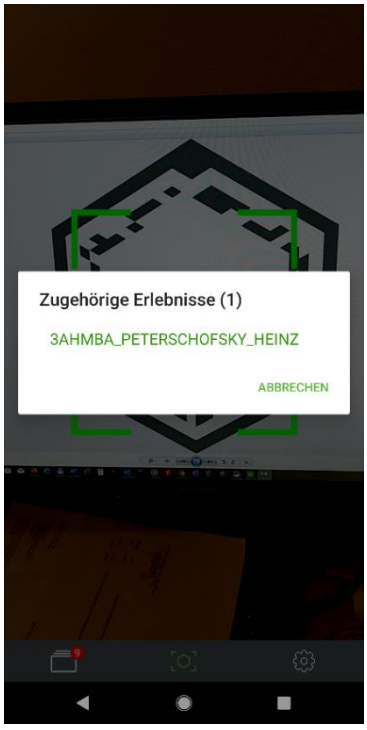


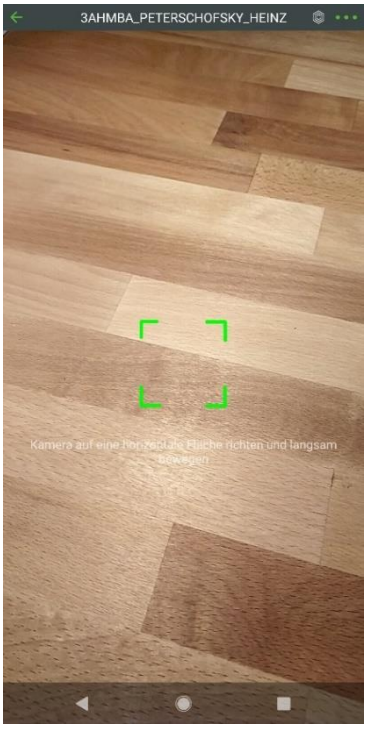
Nach der Installation von Vuforia View können wir die App öffnen. Die App meldet sich mit einem grünen Rahmen und dem Live-Bild der Kamera. Das Erste, was wir machen müssen ist der App irgendwie mitzuteilen, wie der Experience-Server erreicht werden kann. Dies geschieht zum Glück relativ einfach. Im Info-Bereich des Vuforia Studios gibt es einen QR-Code. Diesen kann man mit der App scannen. Deswegen gibt es auch diesen grünen Rahmen. Bringt man den QR-Code in den Rahmen wird nachgefragt, ob man tatsächlich den Experience-Server setzen will. Da kann man in diesem Fall bejahen.

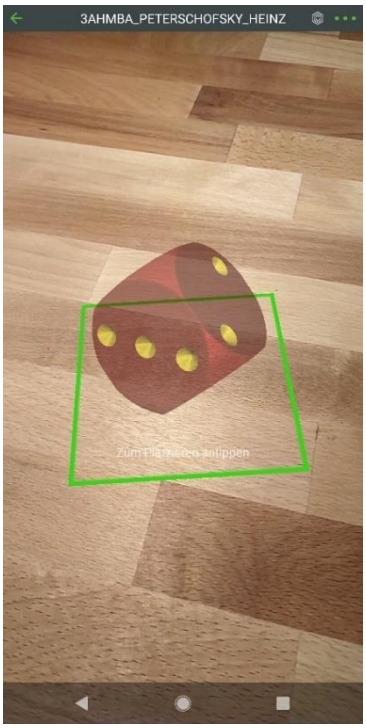




Damit ist der Experience-Server gesetzt. Jetzt muss nur mehr mitgeteilt werden welche Experience geladen werden soll. Wir haben als Identifikationsmethode ThingMark gewählt und das zu unserer Klasse passende ThingMark verwendet. Scannen wir jetzt dieses ThingMark so bekommen wir eine Auflistung welche Experiences unter diesem ThingMark gespeichert sind. Wir wählen das passende aus und suchen uns eine ebene Fläche. Jetzt können wir uns praktisch aussuchen, wo unser Flächenziel ist. Mit einem Klick legen wir die Position fest und dann wird auch schon unser Objekt in die reale Umgebung eingebettet. Selbst wenn wir unser Smartphone bewegen, bleibt das Ding dort, wo es ist. Je nach Telefonmodell funktioniert es besser oder schlechter, aber man bekommt auf alle Fälle eine Vorstellung des Objekts.







2.4.3.1 *Pflichtaufgabe „Augmented Model"*

	Erstellen Sie so wie oben beschrieben Ihre Augmented Reality mit einem Modell Ihrer Wahl. Veröffentlichen Sie diese und führen Sie Ihr Modell vor.	

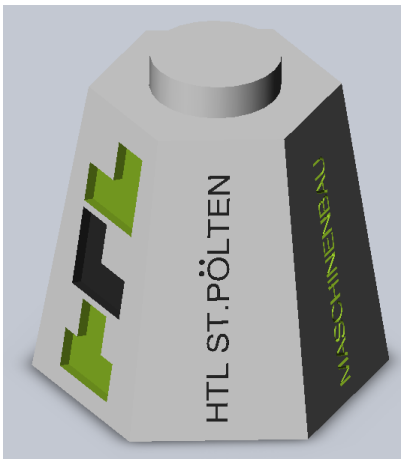
2.4.4 Mehrteiliges Modell für 2D-Device

Video-Link 10: https://youtu.be/aObZ54X_clw

Wir wollen uns nicht länger mit einem einteiligen Modell begnügen. Wenn wir nämlich Dinge animieren wollen, dann müssen wir mehrere Modelle anzeigen bzw. diese gegeneinander verdrehen oder verschieben. Dadurch entsteht der Effekt der Animation. Als ein Beispiel eines solchen mehrteiligen Modells habe ich mich für eine Windmühle entschieden. Ich will nämlich zwei Bewegungen ausführen: Ich möchte, dass der Propeller des Windrades sich drehen kann, außerdem möchte ich das Dach „in den Wind" drehen können[28].

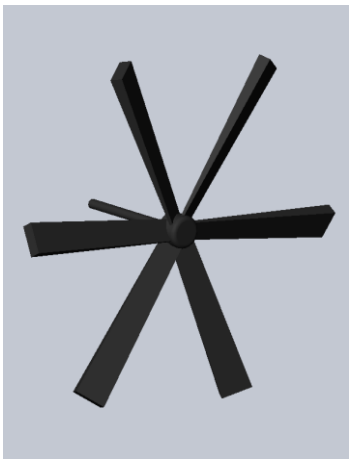
Ich habe damit begonnen die benötigten Dinge in SolidWorks zu zeichnen. Damit hatte ich drei Teile:

[28] Andere Beispiele wären einfach ein Windrad oder auch ein Kran, den man schwenken kann und eine Laufkatze vorwärts und rückwärts bewegen kann. Ich bin für Vorschläge offen.

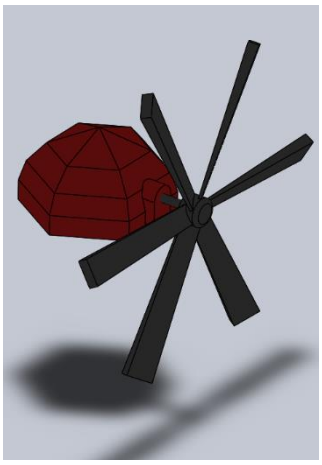


Gebäudeteil


Dachteil


Windrad

Zu beachten ist, dass die Ursprünge der einzelnen Teile gut zueinander passen. Wollen wir z.B., dass das Dachteil genau um die richtige Achse dreht, so muss der Ursprung auch auf dieser Achse liegen. Bei mir wäre das der Mittelpunkt der sechseckförmigen Grundfläche. Bei Rotation um zwei Achsen ist das praktisch nicht möglich. Deswegen habe ich aus dem Dachteil und dem Windrad ein Assembly gemacht und diese beiden Dinge bereits zusammengefügt.

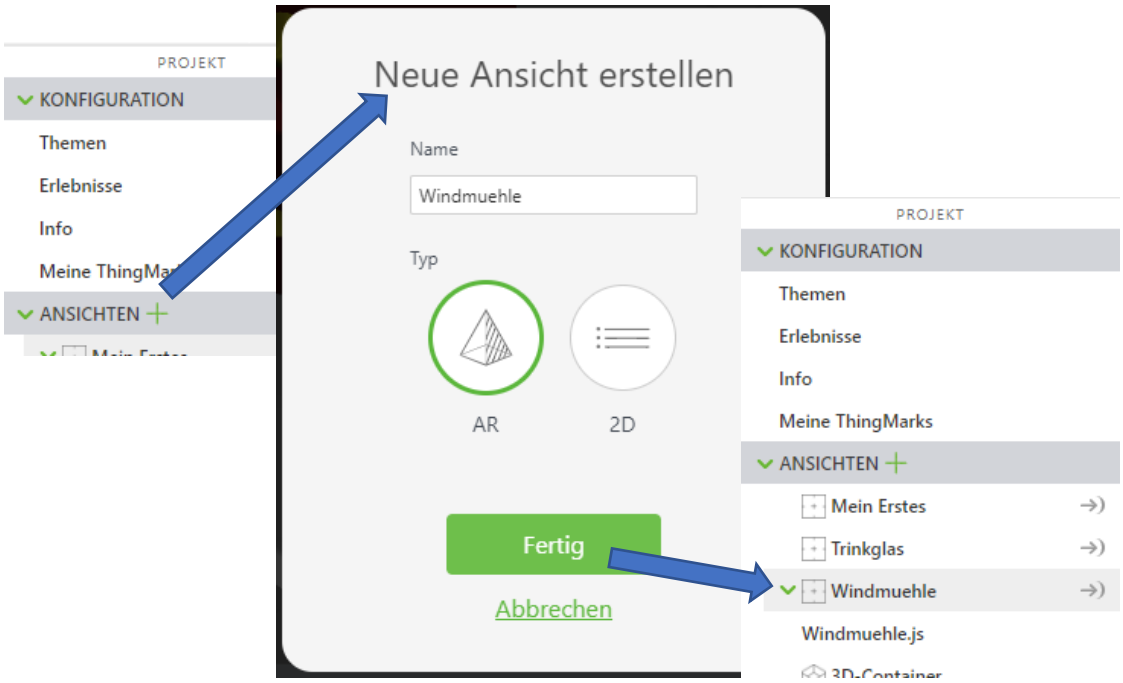


So konnte ich es wagen mein Projekt zu erweitern. Ja: Erweitern, denn wir machen kein neues Projekt, sondern wir erweitern unser bisheriges um eine sogenannte „Ansicht". Diese Ansichten sind so etwas wie „Unterprojekte".

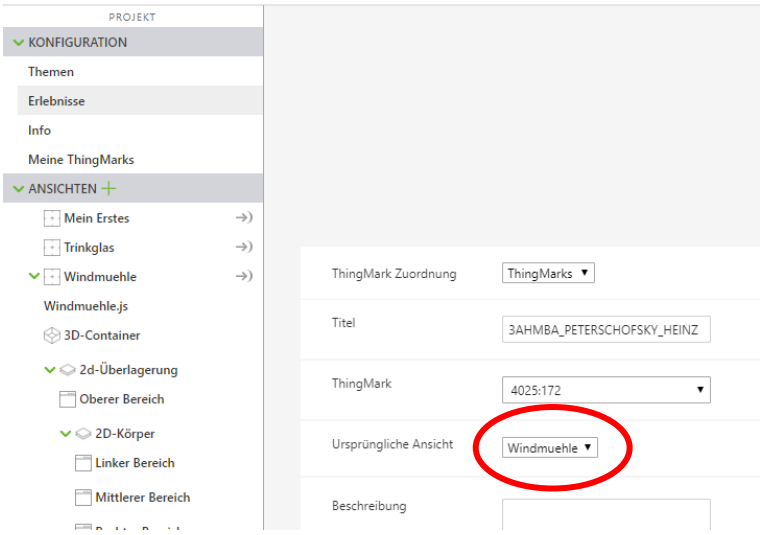
Um eine neue Ansicht zu erstellen klickt man unter „Projekt" auf das „+"-Symbol neben dem Wort „Ansichten".

Dadurch öffnet sich ein Dialogfenster. Dort wählen wir „AR" aus und geben der neuen Ansicht einen Namen. Ich nenne Sie bei mir „`Windmuehle`" (beachte: Keine Umlaute!). Mit einem Klick auf den Button „Fertig" wird die Ansicht erstellt und kann benutzt werden[29]:

[29] Bitte beachten Sie: Ich habe zuvor breites eine Ansicht mit dem Namen Trinkglas erstellt, deswegen scheint diese hier auf – bei Ihnen fehlt diese Zeile. Das ist in Ordnung so.

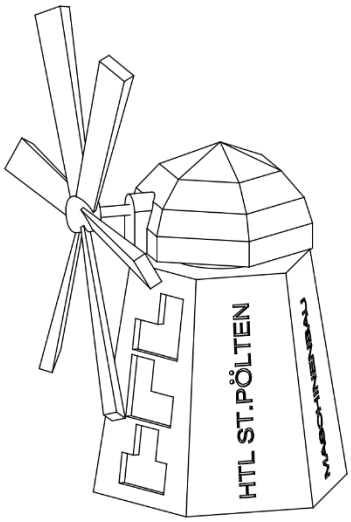


Wir haben also jetzt eine neue View. Wenn wir das Projekt veröffentlichen, bekommen wir deswegen auch die Möglichkeit zwischen den Views zu wechseln – aber das werden wir später sehen. Unter Umständen ist es nervig, wenn man immer wieder auf die richtige View wechseln muss. Deswegen bietet uns Vuforia die Möglichkeit uns eine Standard-View auszusuchen. Dazu wechseln wir auf die Erlebnisse-Einstellseite und wählen dort unter „Ursprüngliche Ansicht" unsere neue View „Windmuehle" aus.

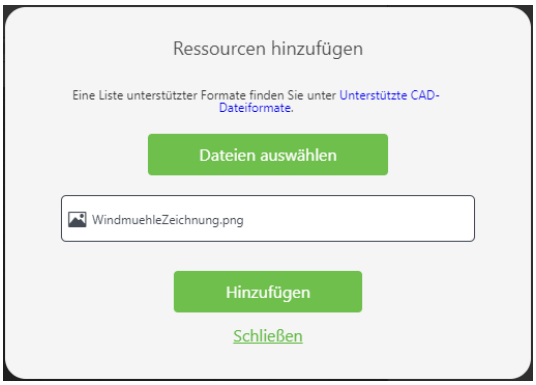


Das bedeutet, dass wir, wenn wir unsere Experience öffnen, automatisch diese View angezeigt bekommen – zu allen anderen Views der Experience müssen wir wechseln.

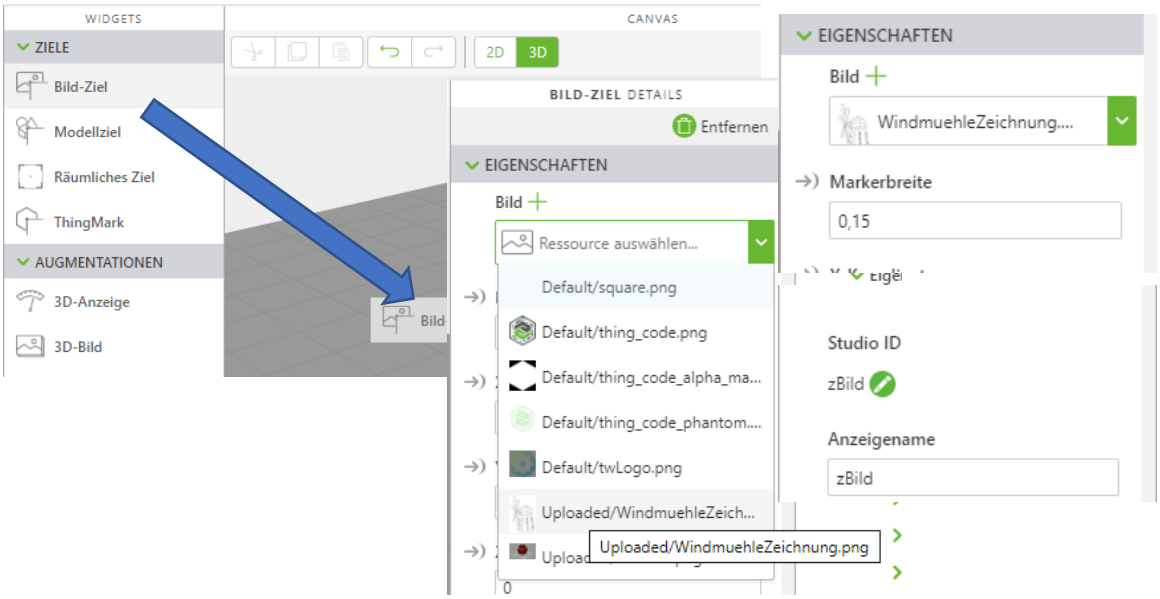
Gut, also legen wir los: Wir werden dieses Mal ein Bild-Ziel verwenden. Ich habe mir in SolidWorks eine Zeichnung des geplanten fertigen Zusammenbaus erstellt und dieses als Bild exportiert. Dieses will ich nun als Bild-Ziel verwenden. Das bedeutet, dass wenn irgendwo die Zeichnung der Windmühle herumliegt und wir mit unserem Device darauf zielen, soll die Windmühle darüber erscheinen.



Zunächst müssen wir die Zeichnung als Ressource hinzufügen. Nun, das funktioniert wieder genauso wie mit Modellen: Das „+"-Symbol neben den Ressourcen Schriftzug öffnet das Dialogfenster. Wir wählen dort die entsprechende Bilddatei aus.

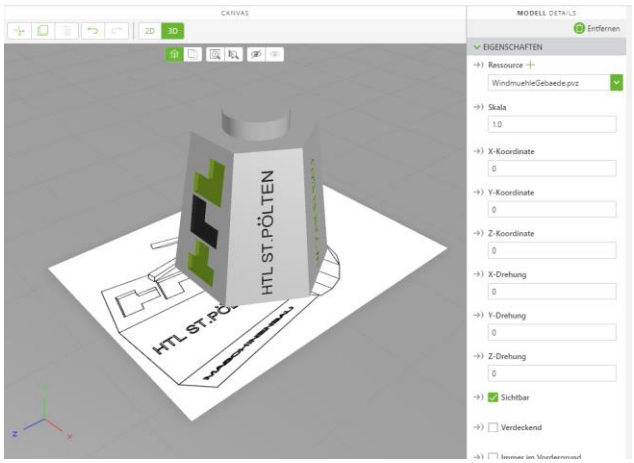


Damit ist sehen wir die Zeichnung in den Ressourcen unter „Uploaded". Jetzt fügen wir ein Bild-Ziel hinzu: Das Bild-Ziel in den Canvas hineinziehen und loslassen. Dann unter Bild-Ziel Details unsere neue Zeichnung als Ressource wählen.



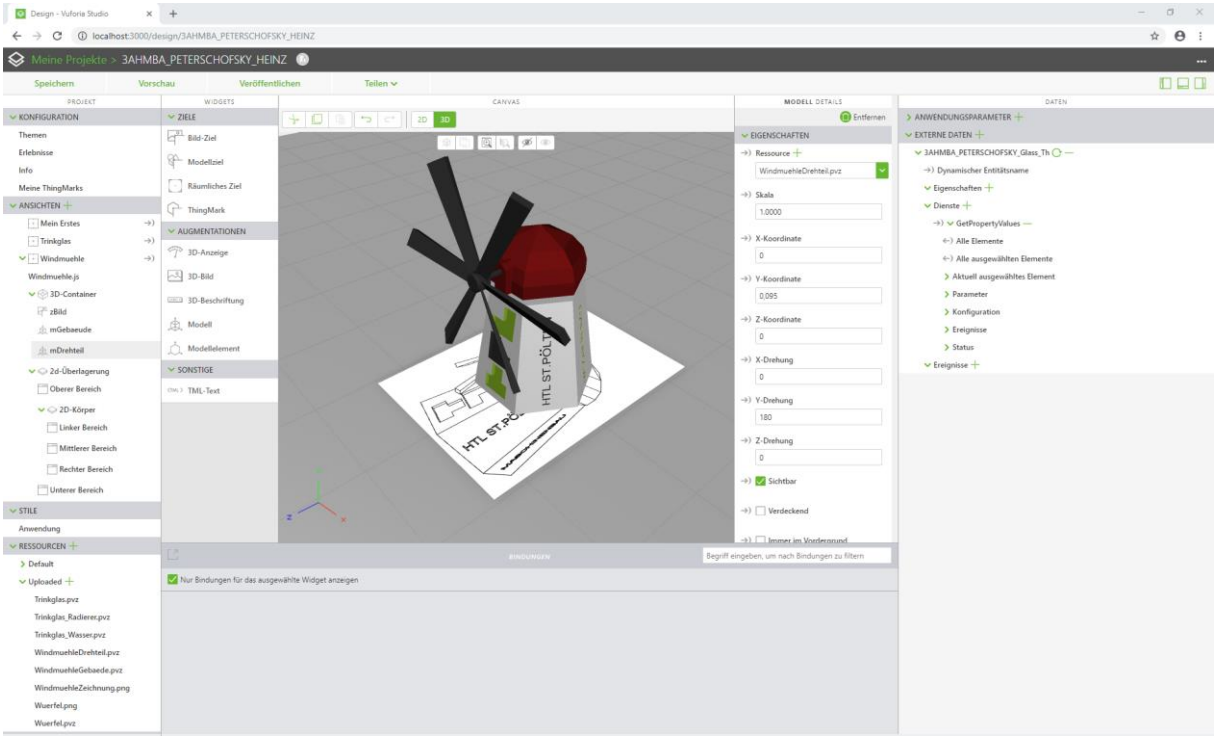
Jetzt müssen wir noch eingeben wie breit unsere Zeichnung ist. Die Windmühle ist bei mir etwa 15cm breit, deswegen gebe ich bei Markerbreite „0.15" ein (zur Erinnerung: Wir haben hier Meter als Einheit). Außerdem nenne ich mein Ziel wieder entsprechend und setzte die Studio- ID auf „`zBild`". Damit haben wir unser Ziel definiert – es wird auch im Canvas entsprechend angezeigt, sogar mit dem richtigen Bild und im Maßstab.

Jetzt können wir beginnen unsere Modelle hinzuzufügen. Ich beginne wieder mit dem Hinzufügen der Ressourcen. Ich wähle als eine Ressource den Gebäude-Teil der Windmühle und als andere Ressource das Assembly aus Dach und Flügelrad. Danach füge ich ein Modell hinzu und wähle dort als Ressource das Gebäude-Teil.

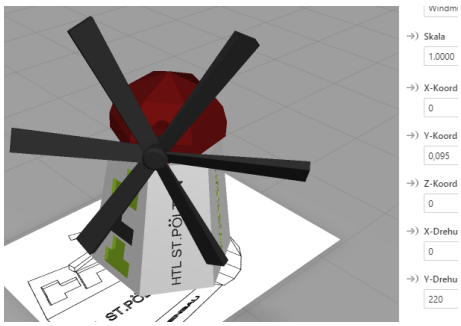


Mit den Koordinaten $(x; y; z) = (0; 0; 0)$ steht das Ding genau auf der Zeichnung und wir geben dem Modell einen entsprechenden Namen – ich wähle „`mGebaeude`".

Jetzt fügen wir den nächsten Teil hinzu: Wieder ein Modell und dieses Mal das Assembly mit dem Dach und dem Flügelrad. Die Koordinaten sind $(x; y; z) = (0; 0.095; 0)$ und die Rotationen sind $(rx; ry; rz) = (0; 180; 0)$. Außerdem nenne ich das Modell noch „`mDrehteil`" und mein Fenster sieht so aus:

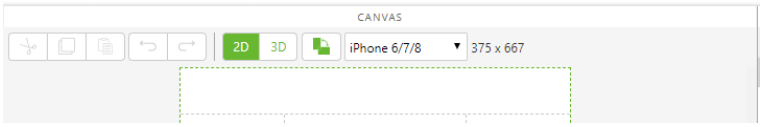


Na, das sieht ja schon sehr vielversprechend aus. Im Prinzip haben wir jetzt ein zweiteiliges Modell. Wir können auch ausprobieren was passiert, wenn wir die Y-Drehung verändern. Ändern wir den Wert dort auf z.B. 220 Grad, so sehen wir wie dich das Dach mit dem Flügelrad um die y-Achse dreht. Das gelingt so gut, weil der Ursprung des Assembly richtig gewählt wurde. Die x-, y- und z-Achsen sind nämlich die vom Assembly. Liegt dessen Ursprung „irgendwo" dreht sich unser Teil auch um dieses „Irgendwo".

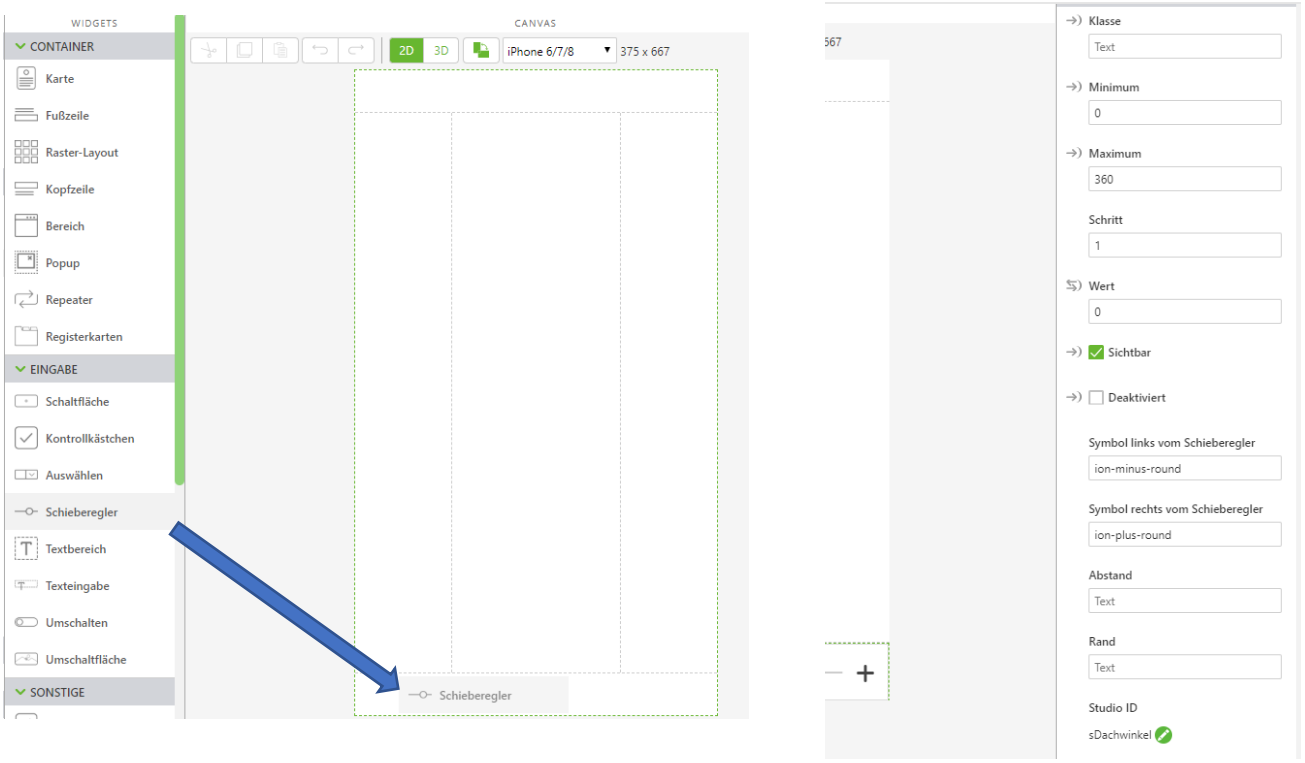


Sind alle Achsen richtig gewählt so lässt sich unser Modell also ganz nett animieren. Jetz sollte man dies auch von unserem Smartphone aus während der Anzeige der Experience können und nicht nur hier, in der Entwurfsphase. Dazu werden wir ein Bedienelement benötigen. Glücklicherweise hat unser Smartphone einen Touchscreen als Bendienoberfläche. Dort können wir uns verschiedenste Dinge einblenden lassen. Ich denke an einen Schieberegeler, mit dem man die y-Drehung des Dachteils einstellen kann.

Dafür können wir im oberen Bereich bei Canvas diesen auf „2D" umschalten. Dann erschit eine weiße Fläche und man kann sich ein Beispieldevice auswählen (im Bild steht da `iPhone 6/7/8`). Wichtig ist zu wissen: Es ist egal welcher Telefon/Tablet dort ausgewählt wird – es wird immer auf allen Geräten funktionieren.



Der Hintergrund des Knopfes ist, dass man sein Layout auf verschiedensten Bildschirmgrößen ansehen kann, um zu beurteilen, ob das Ding denn noch bedienbar bleibt. Beginnen wir einmal mit unserem Schieberegler.



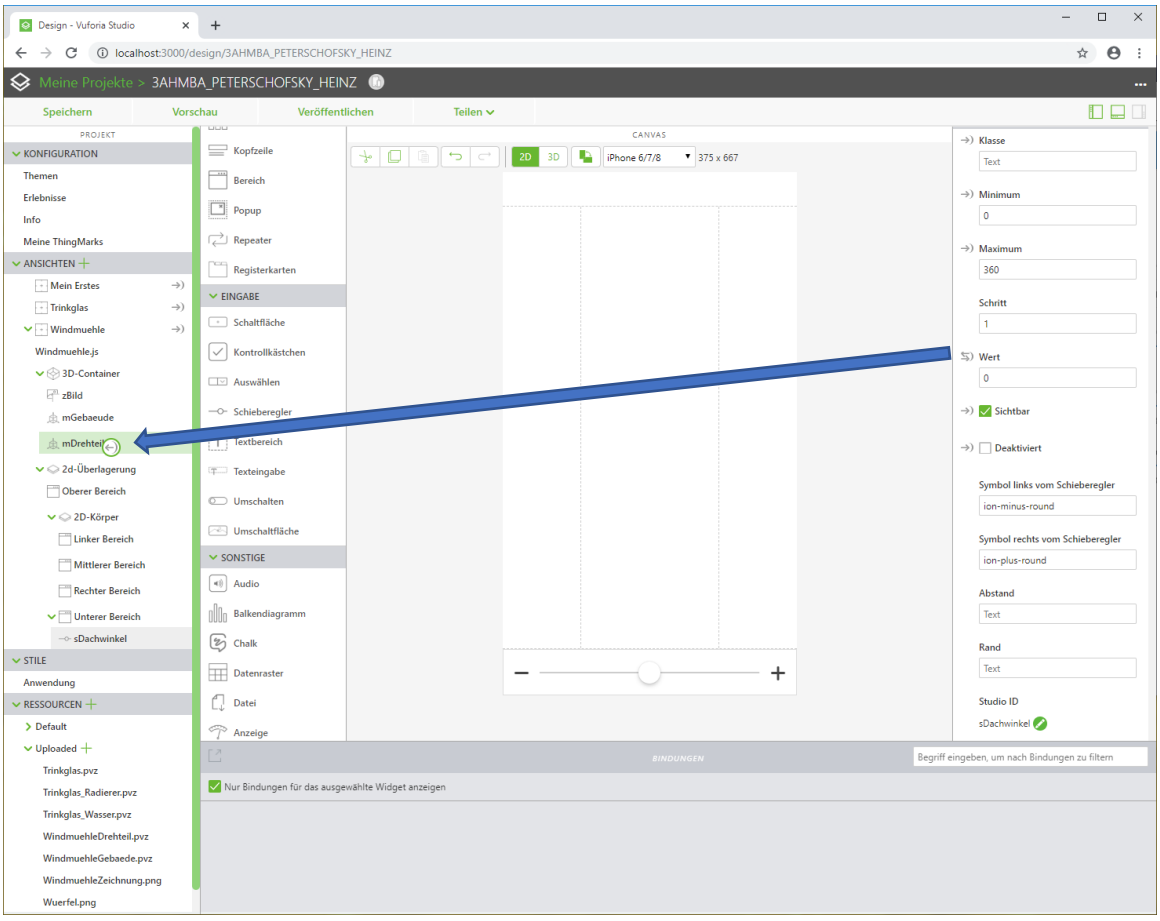
Wie üblich ziehen wir das entsprechende Widget dorthin, wo wir es haben wollen. Ich habe mich entschieden den Regler im unteren Bereich anzuzeigen, also zieh ich ihn dort hin. Er wird auch gleich mit seinen Default-Einstellungen dort angezeigt. Wie üblich sollte man auch den Namen wieder anpassen – meiner hier heißt „`sDachwinkel`". Außerdem soll es ein Drehwinkel sein und ich stelle ein: Minimaler Wert: 0; Maximaler Wert: 360; Schritt: 1; Wert: 0

Wie unschwer zu erahnen handelt es sich bei diesen Dingen um folgende Einstellungen:

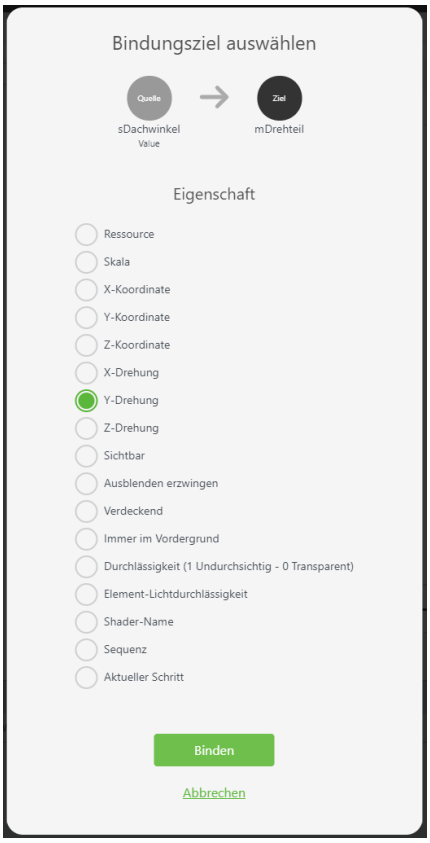
- **Wert:** Ein Schieber hat einen Zahlenwert, der seine Position festlegt. Das ist der aktuelle Wert des Schiebers. Dort ist gerade „das Kugerl". Es wird die aktuelle Position des Bedienelements wiedergegeben.
- **Minimaler Wert:** Das ist der Wert am linken Ende des Schiebers.
- **Maximaler Wert:** Das ist der Wert am rechten Ede des Schiebers.
- **Schritt:** Das ist die minimal erlaubte Änderung des Schieberwertes. Wenn der Schieber bewegt wird, dann ändert sich dessen Wert in genau diesen Schritten.
- **Symbol links/rechts:** Hier kann man die angezeigten Symbole ändern. Dies geschieht über den sogenannten Ionicons-Code. Es gibt ganze Kataloge, aber nicht alle werden unterstützt. Eine Auswahl findet sich im Anhang 4.3.4.

Mit den hier getroffenen Einstellungen kann man also Werte zwischen 0 und 360 einstellen. Man kann den Wert nur in ganzen Zahlen verändern und zum Startzeitpunkt soll der Wert 0 betragen. Das sollte für eine Gradeingabe doch passen: $0° - 360°$, gradweise positionierbar.

Der Schieber ist jetzt hier, aber machen tut er noch nichts. Nur weil sich sein Wert verändert, hießt noch nicht, dass sich das Dach verdreht. Man muss den Wert an den Drehwinkel des Daches „binden". Dazu muss man nur diese beiden Pfeilchen neben dem Wort „Wert" mit der Maus nehmen und auf das Objekt ziehen, auf welches der Wert verbunden werden soll. Ich will damit den y-Drehwinkel meines Objekts `mDrehteil` verändern und ziehe den Wert deswegen dorthin. Hier kann man erkennen, wie gut es war, dass wir unseren Dingen einen lesbaren, interpretierbaren Namen geben haben.



Lässt man den Wert dort los erschient ein Auswahldialog. Dort kann man auswählen mit welcher Eigenschaft des Objekts man den Wert verbinden will. Es kommt eine ganz schöne Liste – man kann also viele Dinge von Modellen manipulieren. Wir wollen hier die Y-Drehung verändern, deswegen wählen wir diese auch in der Liste aus und drücken „Binden".

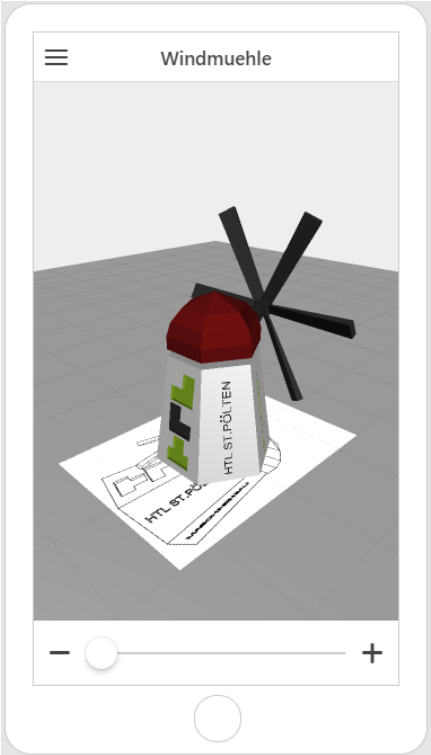


Im unteren Bereich wird dann auch sogleich die Bindung angezeigt:

Mit einem Klick auf das Mistkübel-Symbol könnten wir diese Bindung auch wieder loswerden. Im Bild kann man erkenne, dass der Wert des Sliders mit der Eigenschaft ry des Modells Drehteil verknüpft ist.

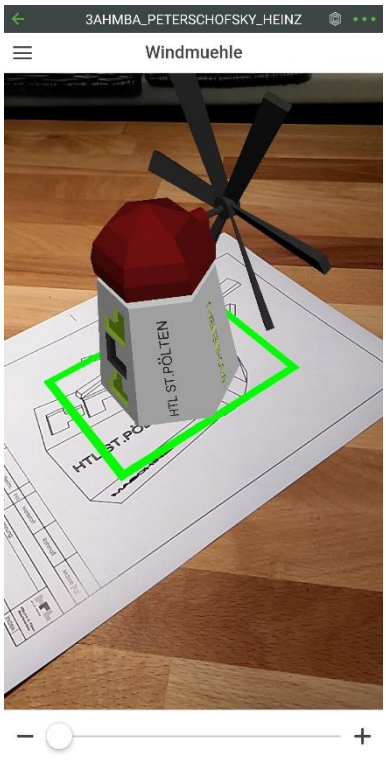
Das sollte es gewesen sein. Wir können unsere Experience einmal testen, indem wir auf „Vorschau" klicken. Und tatsächlich: Es erscheint unsere Windmühle und im unteren Bereich ein Slider. Wenn wir diesen bewegen, so bewegt sich auch das Dach mit dem Flügelrad der Windmühle – wunderbar, so soll es aussehen.

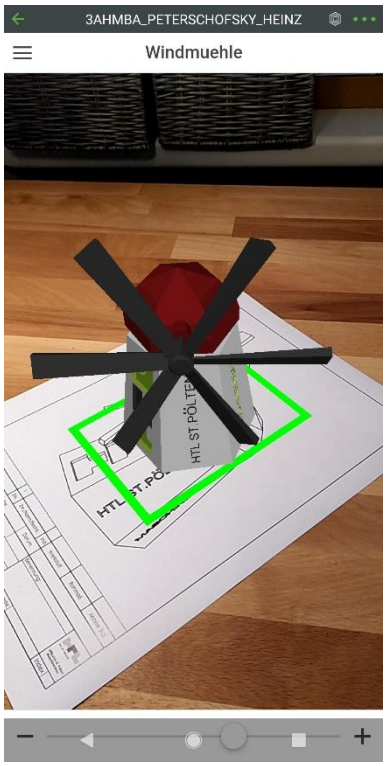
Dreht sich ein Teil um eine seltsame Achse, so passte entweder die Achszuordnung nicht (x-, y-, z-Achse irgendwie falsch) oder der Ursprung im Modell sitzt an einer falschen Stelle.



Wir können unsere Experience veröffentlichen und auf unserm Smartphone bzw. Tablet ansehen. Dazu benötigen wir natürlich einen Ausdruck der Zeichnung, die wir hier als Zeichungsziel verwendet haben. Theoretisch müsste es sogar gehen unser Modell mit der Zeichnung „wegzutragen".

Alternativ würde auch einfach eine Anzeige der Zeichnung auf dem Bildschirm funktionieren. Dann steht unser Teil eben aus dem Bildschirm heraus.





2.4.4.1 *Pflichtaufgabe „Simple animated Model"*

	Erstellen Sie so wie oben beschrieben Ihre animierte Augmented Reality mit einem Modell Ihrer Wahl. Veröffentlichen Sie diese und führen Sie Ihr Modell vor.	

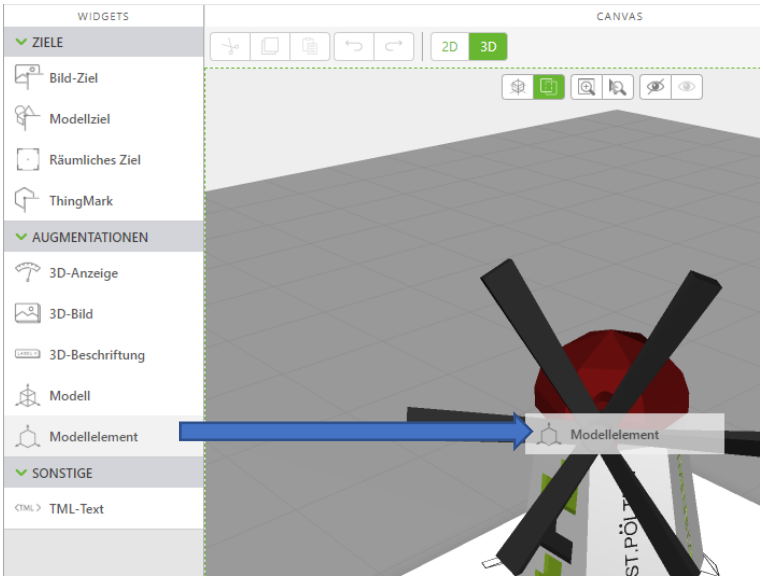
2.4.5 Automatische Animation

Video-Link 11: https://youtu.be/NRckM3p42Ok

Einfach nur das Dach meiner Windmühle zu animieren ist nur die halbe Miete, denn eigentlich will man ja auch, dass sich der Rotor dreht. Natürlich könnte man einen zweiten Slider einbauen, wir wollen hier aber mehr: Wir wollen, dass man die Rotation des Rotors ein- und ausschalten kann. Der Rotor soll sich dann drehen und nicht mehr aufhören. Der Drehwinkel des Rotors soll sich also automatisch erhöhten[30].

Eine erste Schwierigkeit ergibt sich, wenn wir uns überlegen welchen Teil wir rotieren lassen wollen. Einfach das Modell `mDrehteil` zu rotieren bringt uns nichts, denn dann würde das ganze Dach mitrotieren – das war vorhin angenehm, jetzt stört es aber. Wir müssen hier ein sogenanntes „Modellelement" verwenden. Vuforia erkennt nämlich Untergruppen in unseren Assemblies und wir können uns eine solche Untergruppe als Modellelement auswählen[31].

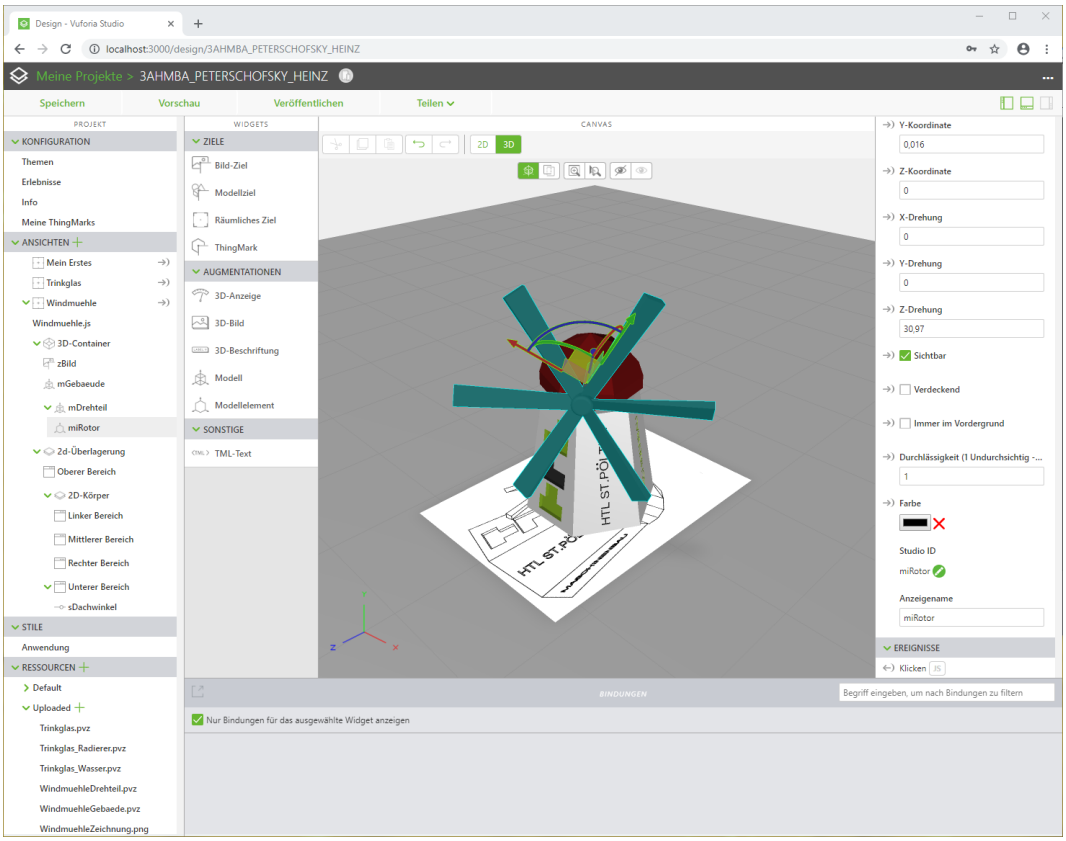
Das funktioniert wieder relativ einfach, indem wir aus den Widgets das Modelelement wählen und dieses auf das entsprechende Modell darauf ziehen (Umschalten auf 3D-Ansicht nicht vergessen!). Das Zielen ist etwas unglücklich, aber es funktioniert erstaunlich gut.



Ob wir auch das richtige Element unseres Assemblies getroffen haben, sehen wir unmittelbar darauf, denn das Modellelement wird hervorgehoben. Im Baum rechts ist es ebenfalls neu hinzugekommen (logischerweise unter dem Modell `mDrehteil`. Wir geben dem neuen Modellelement noch einen schönen Namen (ich wähle `miRotor`).

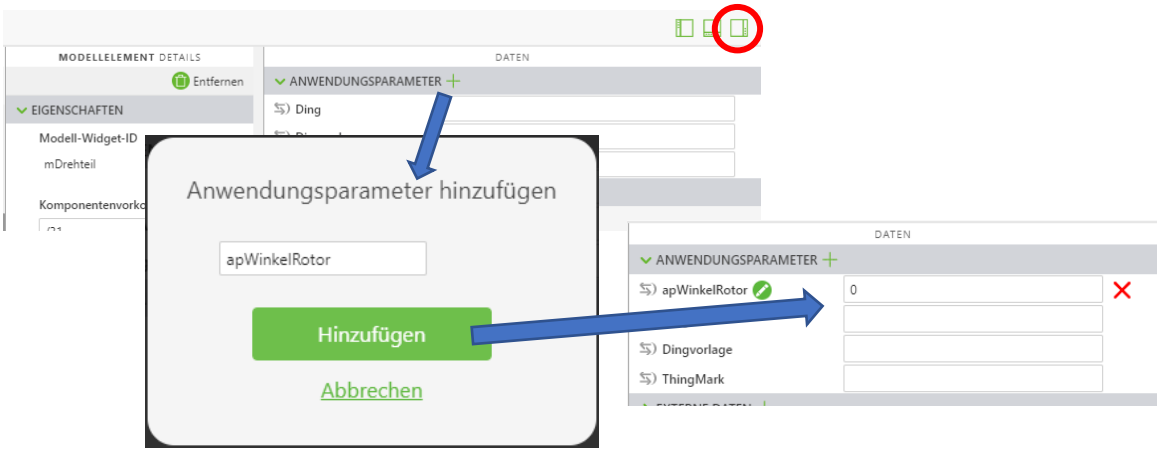
[30] Natürlich werden wir es so machen, dass der Rotor von 359° wieder auf 0° wechselt und auch von 0° auf 359° statt den Winkel über alle Grenzen hinweg größer werden zu lassen.

[31] Das geht nicht in beliebiger Feinheit. Als Faustregelt gilt: Je komplizierter das Assembly desto wahrscheinlicher ist es, dass nicht alles korrekt erkannt wird.



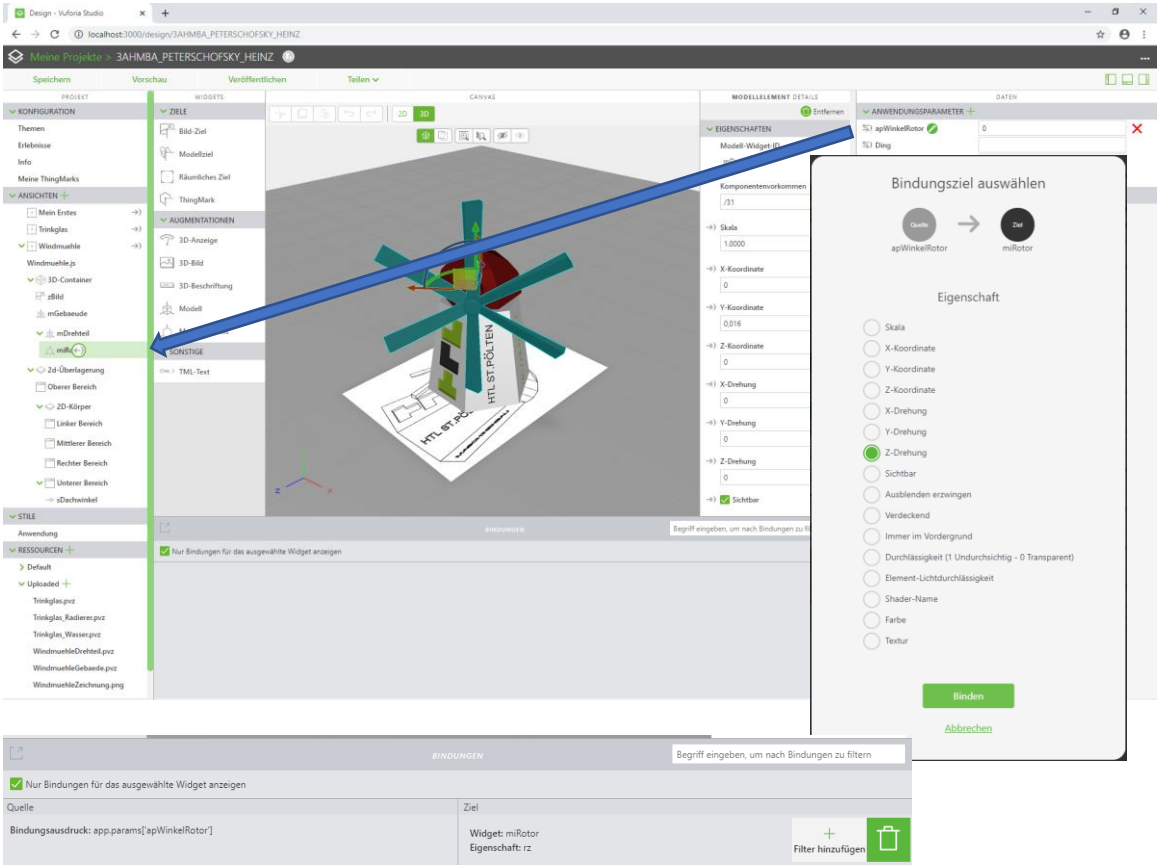
Praktischerweise haben wir auch gleich die Achsen unseres Modellelements eingezeichnet. Die blaue Z-Achse ist die gültige Drehachse. Wenn wir also Wind simulieren wollen, dann müssen wir die Z-Drehung von `miRotor` manipulieren.

Es hat sich gezeigt, dass es eine gute Idee ist, sogenannte „Application Parameter" oder auch „Anwendungsparameter" anzulegen. Diese Anwendungsparameter kann man sich wie Variablen in einem Programm vorstellen. Man kann ihnen einen Wert geben, sie manipulieren und diesen Wert als Eigenschaft für ein Objekt verwenden. Obwohl es in diesem Beispiel nur beschränkt Sinn macht legen wir uns einen solchen Anwendungsparameter an: Wir wollen den Drehwinkel des Flügelblattes über einen solchen steuern. Sollte der Daten-Bereich ganz rechts nicht eingeblendet sein kann man ihn mit einem Klick auf das rechte Fenster-Symbol (unten rot eingeringelt) anzeigen lassen.



Zunächst drücken wir das „+"-Symbol neben den Anwendungsparametern. Als Namen wähle ich `apWinkelRotor` und drücke hinzufügen. Dann ist dieser Parameter angelegt und ich gebe ihn

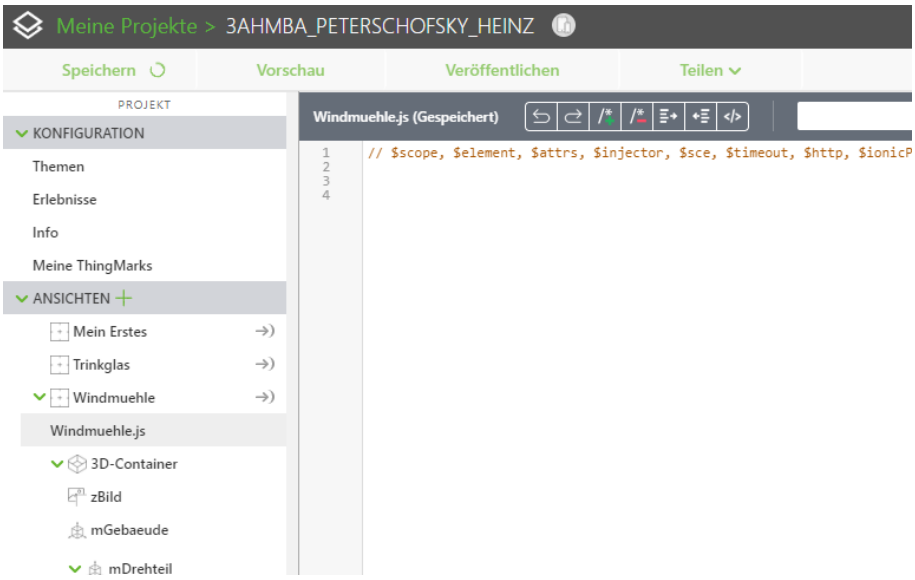
noch den Wert 0. Der macht jetzt einmal noch nichts. Ich binde den Wert des Parameters wieder mit dem Ziehen der beiden Pfeilchen auf mein Modelitem `miRotor`, und zwar mit der z-Drehung.



Diese Bindung wird untern wieder angezeigt. Was immer jetzt in `apWinkelRotor` drinnen steht wird als Z-Drehwinkel für `miRotor` übernommen.

Wir fügen noch einen weiteren Anwendungsparameter hinzu. Dieser soll beinhalten wie schnell sich der Rotor dreht. Ich füge also den Parameter `apWindStaerke` hinzu und setze ihn auf den Wert 1.

Jetzt müssen wir nur mehr den Anwendungsparameter `apWinkelRotor` manipulieren. Dazu müssen wir wieder etwas Programmieren – dieses Mal in der Sprache JavaScript. Im linken Projekt-Baum befindet sich ganz oben, gleich unter der Ansicht der Eintrag „`Windmuehle.js`“. Dort kann man etwas programmieren, also drücken wir drauf.



Es öffnet sich ein Text-Editor. Hier können wir ein Programm schrieben. Dieses lautet:

```
var timerId = -1;         // timer id
var angleIncrement = 1;   // degrees
var timingInterval = 50; // milliseconds

$scope.spinRotor = function() {        // function to calculate the
  $scope.app.params.apWinkelRotor =    // new rotation
    $scope.app.params.apWinkelRotor -
    $scope.app.params.apWindStaerke
  $scope.app.params.apWinkelRotor =
    $scope.app.params.apWinkelRotor % 360; // only between 0 and 359
}

$scope.timerFuncSpin = function() {      // calc new rotation
  if (!$scope.app.params.apWinkelRotor)  // app parameter check
    $scope.app.params.apWinkelRotor = 0; // set a default value
  if (!$scope.app.params.apWindStaerke)  // check for other para
    $scope.app.params.apWindStaerke = angleIncrement;

  $scope.$apply($scope.spinRotor()); // apply the new values
}

$scope.fStartWind = function() {    // function to start wind
  if (timerId > -1) clearInterval(timerId);

  timerId = setInterval($scope.timerFuncSpin, // start timer and
                        timingInterval);      // call every xx ms
}


$scope.fStopWind = function() {    // function to stop wind
  clearInterval(timerId);          // delete the timer
  timerId = -1;                    // and it's id
}
```

Und was macht das bitte?

Also die Funktion `$scope.spinRotor()` addiert immer wenn sie aufgerufen wird den Wert von `apWindStaerke` zu `apWinkelRotor`. Danach wird noch durch 360 dividiert und der Rest genommen (Modulo-Operation). Somit bleibt der Wert von `apWinkelRotor` immer zwischen 0 und 359.

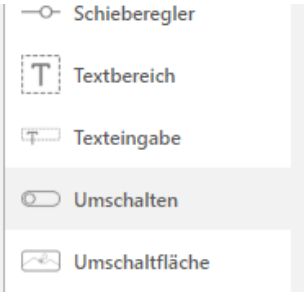
Die Funktion `$scope.timerFuncSpin()` schaut nur ob die beiden Applikationsparameter `apWinkelRotor` und `apWindStaerke` vorhanden sind. Sind sie das nicht wird ein Default-Wert gesetzt. Das kann eigentlich nur passieren, wenn wir zwar den Parameter angelegt haben, aber keinen Wert zugewiesen haben – der Fall wäre dann aber hier abgefangen. Danach wird die

Funktion `$scope.spinRotor()` aufgerufen, und zwar mit einem `$scope.$apply`. Das bedeutet, sobald die Funktion durchgelaufen ist, wird das Bild neu berechnet – sonst würde nämlich nichts passieren.

Die Funktion `$scope.fStartWind()` schaut zu Beginn ob bereits ein Timer gesetzt ist. Ist dies der Fall wird dieser gelöscht. Das ist ebenfalls nur zur Sicherheit, sollte eine Fehlbedienung vorliegen. Jetzt ist der Timer auf alle Fälle abgedreht – also können wir ihn aufdrehen. Mit `setInterval` wird ein Timer periodisch aufgerufen. Wir nennen die Funktion `$scope.timerFuncSpin()` und den Wert `timingInterval`. Die Funktion `$scope.timerFuncSpin()` wird also ab hier alle $50ms$ auferufen. Das Rad dreht sich.

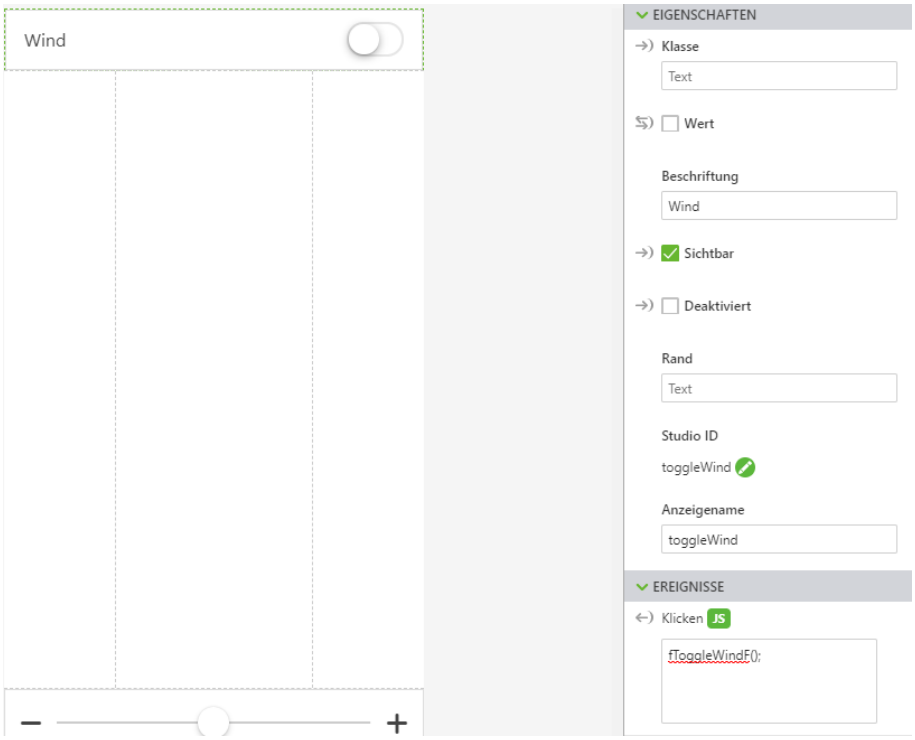
Die Funktion `$scope.fStopWind()` löscht den Timer wieder – das Rad bleibt stehen.

Wir brauchen also nur `fStartWind()` aufrufen und der Rotor wird sich bewegen. Wenn wir danach `fStopWind()` aufrufen wieder der Rotor wieder stehen bleiben.



Um dies zu bewerkstelligen bauen wir uns einen Toggle-Button in unsere 2D-Oberfläche ein. Ein Umschalten auf 2D und wir verwenden dort das entsprechende Widget „Toggle" oder „Umschalten" und ziehen es in den oberen Bereich.

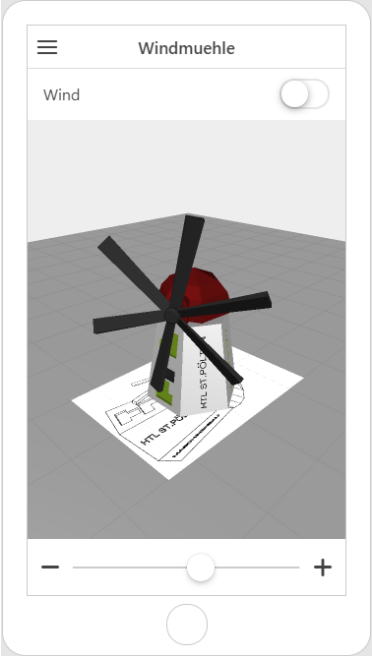
Als Beschriftung wählen ich „Wind" und die Studio-ID setze ich auf `toggleWind`. Jetzt haben wir einen Schalter, den man ein und ausschalten kann. Was wir noch benötigen ist eine Funktion, die wir bei jedem Mal klicken aufrufen. Leider können wir nicht zwei unterschiedliche Funktionen je nach Status (eingeschalten/ausgeschalten) aufrufen, sondern nur eine. Wir überlegen uns also einen weiteren Namen – ich wähle `fToggleWindF()`.

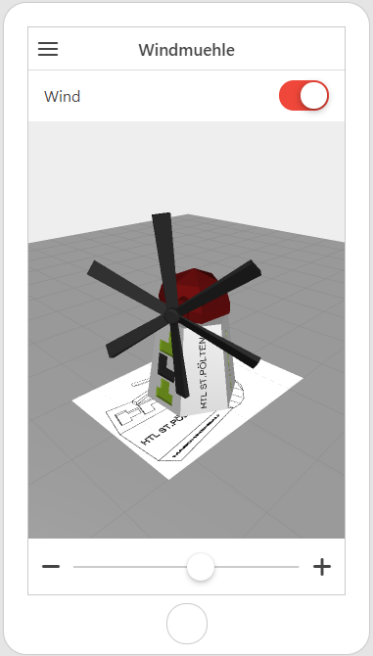


Diese müssen wir aber noch programmieren. In „Windmuehle.js" fügen wir deswegen folgenden Code hinzu:

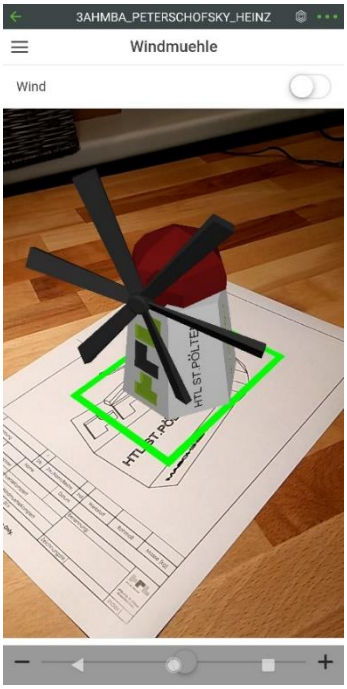
```
$scope.fToggleWindF = function() { // function to start/stop turning
  if ($scope.view.wdg['toggleWind']['value']) $scope.fStartWind();
  else $scope.fStopWind();
}
```

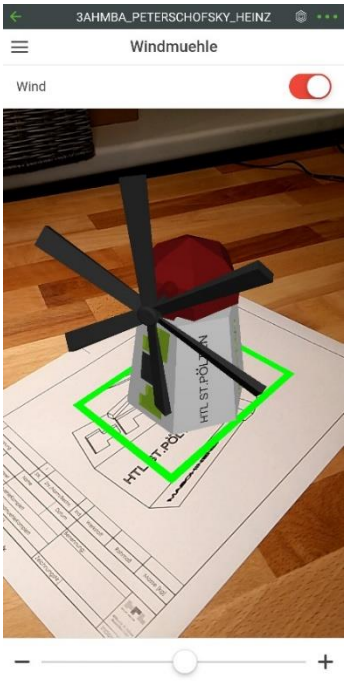
Was macht das? Wenn der Wert des Widgets `toggleWind` `TRUE` ist, wird die Bewegung gestartet – ansonsten gestoppt. Damit sollte das funktionieren. Das Versuchen wir jetzt mit der Vorschau – und wirklich: Es geht! Die Rotorblätter bewegen sich, wenn wir den Schalter einschalten und stoppen, wenn der Schalter aus ist.





Es ist also einen Versuch wert die Experience zu veröffentlichen und mit unserem Smartphone anzusehen. Auch hier scheint es zu funktionieren.





2.4.5.1 *Pflichtaufgabe „Advanced animated Model"*

	Erstellen Sie so wie oben beschrieben Ihre animierte Augmented Reality mit einem Modell Ihrer Wahl und automatischer Animation. Veröffentlichen Sie diese und führen Sie Ihr Modell vor.	

2.4.5.2 *Zusatzaufgaben „Wind adjustable"*

	Modifizieren Sie die Experience so, dass die Animationsgeschwindigkeit der automatischen Bewegung mit einem Schieberegler langsamer und schneller gemacht werden kann. Der Schieberegler soll nur eingeblendet sein, wenn die Animation läuft.	

2.4.6 Projekt für ein 3D-Device (Microsoft HoloLens)

Video-Link 12: https://youtu.be/CtqglwcVido

Wir haben vermutlich gemerkt, dass die vorher gemachten Dinge – je nach Endgerät – einmal besser und einmal schlechter funktionieren. Wir sind also abhängig von der Implementation der Sensoren in das jeweilige Smartphone oder Tablet. Das ist wie schon im allgemeinen Teil beschrieben wurde in generelles Problem von AR. Wir wollen doch nicht vergessen, dass es sich immer noch um ein Telefon handelt.

Bei Devices welche explizit Augmented Reality als Anwendungsfall haben sieht das anders aus. Wir werden im Unterricht die Microsoft HoloLens verwenden. Mittlerweile ist diese in zwei Genrationen verfügbar. Die Bedienung der HoloLens ist zwar etwas gewöhnungsbedürftig, aber es funktioniert nach ein paar Versuchen ganz gut.

Wir werden erkennen: Das Bild, welches uns die HoloLens bietet, ist nicht mit den der Smartphones oder Tablets vergleichbar. Das Bild steht ruhig vor uns. Wir können uns wirklich uneingeschränkt bewegen und haben die Hände frei – dafür gibt es aber auch keine Bedienoberfläche wo wir einfach unsere Köpfe hinlegen können.

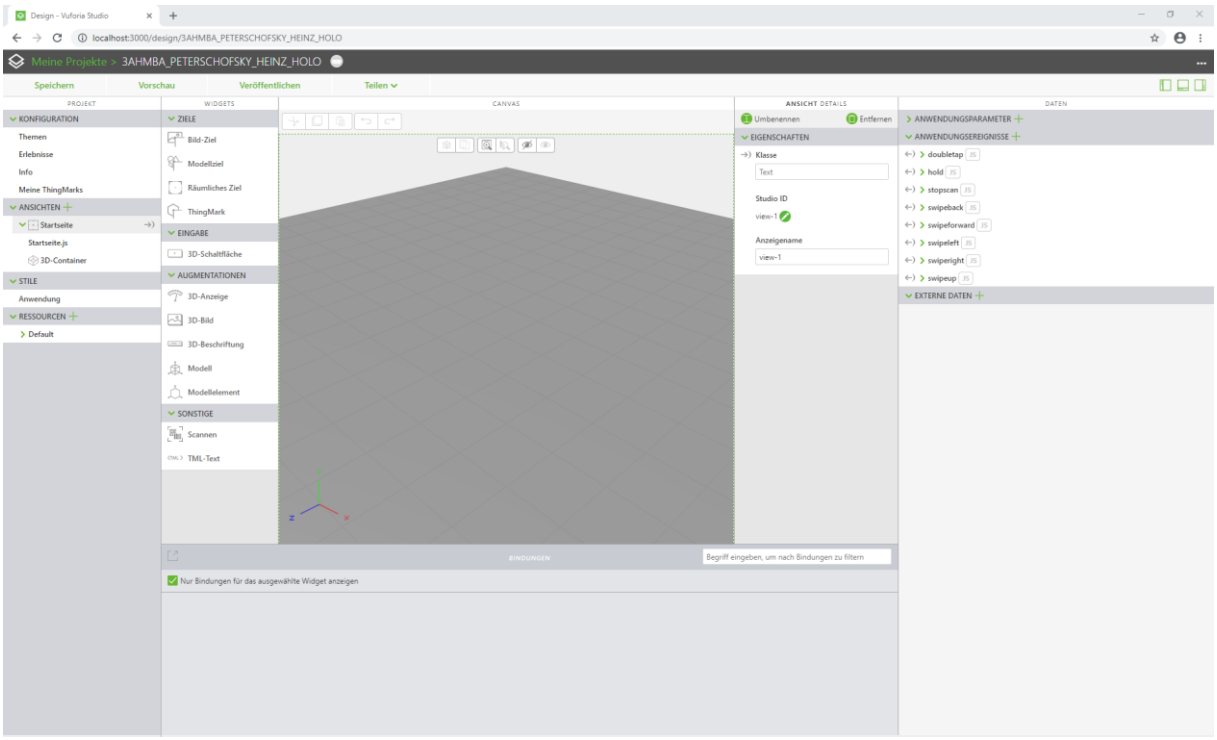
Wir werden das jetzt einfach einmal versuchen – also erzeugen wir ein neues Projekt. Nur dieses Mal wählen wir „3D Eyewear – Default".

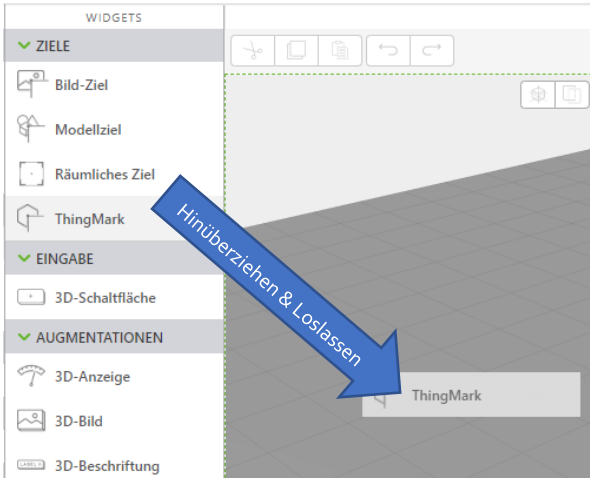
Also Projektname verwende ich dieses Mal **3AHMBA_PETERSCHOFSKY_HEINZ_HOLO** (weil ja mein Default-Name 3AHMBA_PETERSCHOFSKY_HEINZ schon belegt ist (vom 2D-Projekt).

Wie schon beim ersten Projekt werden wir aufgefordert eine Experience Service URL einzugeben – das ist der Link zu unserem Experience Server am Port 8443. Mit oben genanntem Link schreiben wir dort also hinein:

https://example.twx.htl.schule:8443

Noch den Knopf Erstellen klicken und wir bekommen wieder die altbekannte Oberfläche. Am ersten Block sieht gar nichts anders aus, im Detail gibt es aber schon ein paar Unterschiede: Der 2D-Bereich ist verschwunden – kein Wunder, es gibt je dieses Mal auch keine 2D-Oberfläche. Dafür gibt es ein paar zusätzliche Widgets (z.B. 3D-Anzeige oder 3D-Bild). Im Prinzip hat sich jedoch nichts geändert und wir sollten keinerlei Probleme haben eine Experience zu erstellen.

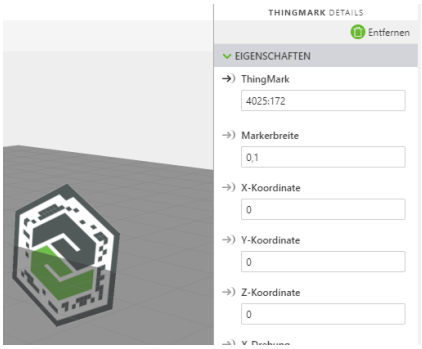




Als erstes fügen wir wieder ein Ziel hinzu, Dieses Mal werden wir ein ThingMark als Ziel verwenden.

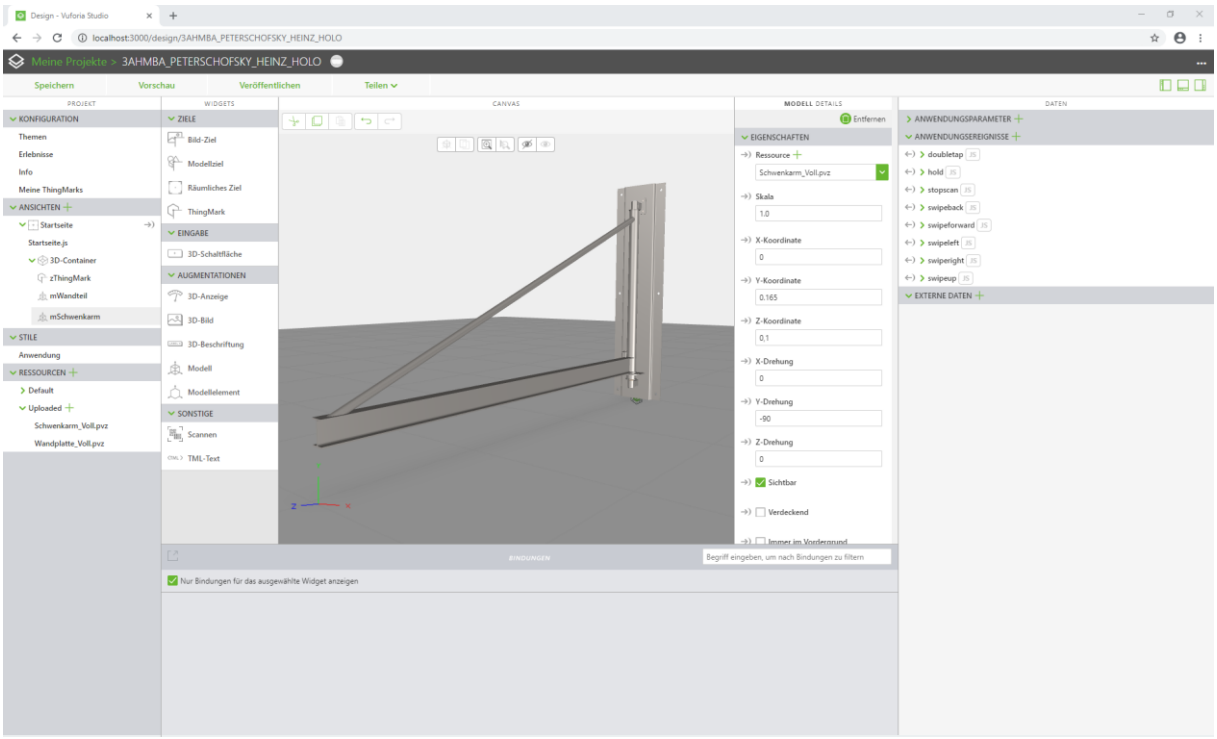
Eigentlich funktioniert das genauso wie beim Bildziel aus Kapitel 2.4.4, nur müssen wir anstatt der Bildressource einfach eine ThingMark-Nummer angeben.

Die Markenbreite, die hier einzustellen ist, ist praktisch die Schlüsselweite des sechseckigen ThingMarks. Ich verwende hier $0{,}10$ - also $10cm$. Außerdem möchte ich, dass das ThingMark senkrecht steht, also setze ich alle Drehungen auf 0. Jetzt noch die Studio-ID auf einen vernünftigen Wert gesetzt (`zThingMark`) und wir haben unser Ziel platziert.



Im Beispiel verwende ich wieder ein zweiteiliges Modell eines Wandkrans. Ich füge also die benötigten Ressourcen hinzu und mache mir zwei Modelle und nenne sie entsprechend.

Positioniert und gedreht wird so, dass die zwei Dinge zusammenpassen. Eigentlich sollte jeder in der Lage sein diese Schritte selbsttätig durchzuführen. Deswegen hier ein Screenshot wie es in etwas aussehen sollte.



Zum Veröffentlichen gehen wir wieder genauso vor wie unter Kapitel 2.4.3 beschrieben. Also unter Erlebnisse das ThingMark einstellen und eine kurze Beschreibung hinzufügen. Unter Info eine Beschreiung hinzufügen, den Experience-Server validieren und den Zugriff auf Öffentlich stellen. Als Projekt-Miniaturansicht ein geeignetes Bild auswählen und wir sind startklar um zu Veröffentlichen.

Das können wir machen und es einmal mit der HoloLens ausprobieren. Als Ziel können wir das angegeben ThingMark ausdrucken. Das ist „zufällig" genau 10cm breit und passt somit genau für diesen Zweck. Wir werden gleich den Unterschied erleben. Das Ding steht ruhig vor uns.

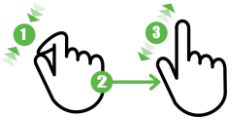
2.4.6.1 Pflichtaufgabe „3D-Augmentend Device"

	Erstellen Sie so wie oben beschrieben Ihre Augmented Reality für die Microsoft HoloLens mit einem Modell Ihrer Wahl. Veröffentlichen Sie diese und führen Sie Ihr Modell vor.	

2.4.7 User-Eingaben mit der Microsoft HoloLens

Video-Link 13: https://youtu.be/cKmu19Am_GY

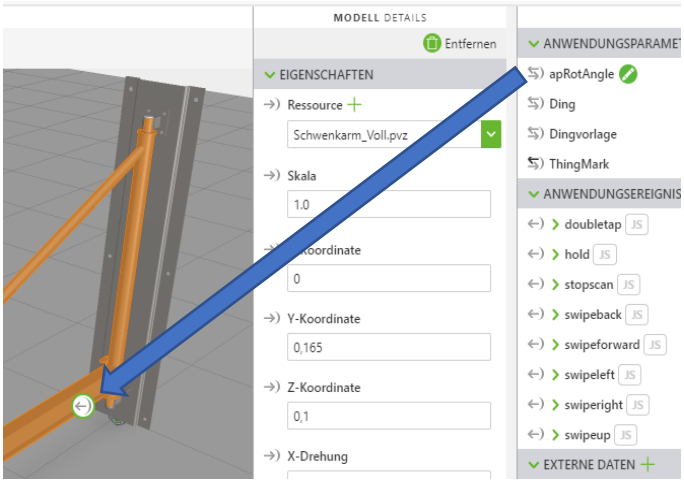
Was uns bei der HoloLens schon abgeht ist die Möglichkeit der Eingabe. Es gibt schlicht und ergreifend keine Oberfläche, wo man etwas platzieren könnte. Es gibt die Möglichkeit von Gestiken. spezielle Bewegungsabläufe können Aktionen auslösen. Die von der HoloLens unterstützten Abläufe sind in folgender Tabelle zusammengefasst.

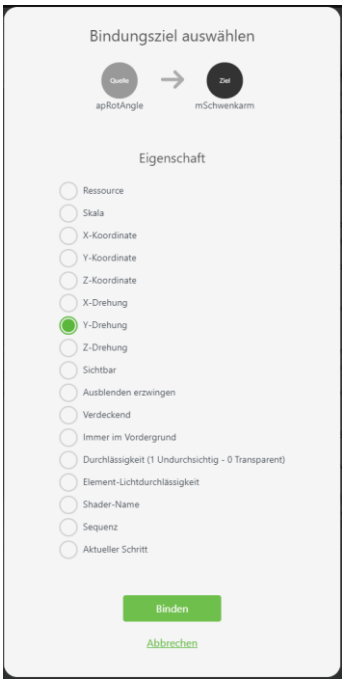
Name	Kurztitel	Symbol	Erklärung
doubletab	Doppelklick		Den Zeigefinder zweimal hintereinander auf den Daumen bewegen.
hold	Halten		Den Zeigefinger auf den Daumen legen und dort belassen
swipeback	nach hinten ziehen		Den Zeigefinger auf den Daumen legen, danach zum Körper ziehen und danach den Zeigefinger wieder austrecken.
swipeforward	nach vorne ziehen		Den Zeigefinger auf den Daumen legen, danach vom Körper ziehen und danach den Zeigefinger wieder austrecken.
swipeleft	nach links ziehen		Den Zeigefinger auf den Daumen legen, danach nach links ziehen und danach den Zeigefinger wieder austrecken.
swiperight	nach rechts ziehen		Den Zeigefinger auf den Daumen legen, danach nach rechts ziehen und danach den Zeigefinger wieder austrecken.
swipeup	nach oben ziehen		Den Zeigefinger auf den Daumen legen, danach nach oben ziehen und danach den Zeigefinger wieder austrecken. Defaultmäßig löst diese Geste einen Rescan aus.

Als Übung wollen wir folgendes realisieren: Ein Swiperight soll den Kran nach rechts schwenken. Ein Swipeleft soll den Kran nach links schwenken. Ein Doubletap soll der Kran wieder in die Mitte bringen.

Wie schon in Kapitel 2.4.5 verwenden wir einen Application-Parameter. Wir fügen also einen solchen hinzu, nennen ihn `apRotAngle` und geben einen Defaultwert von −90.

Danach binden wir den Parameter noch mit der Y-Drehung des Drehteils.





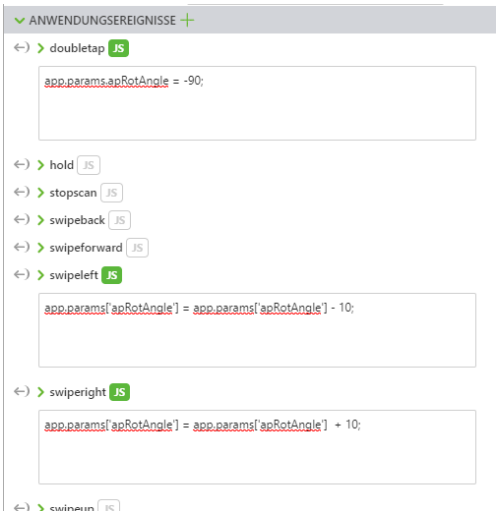
Wenn man neben den Gestiken auf die „JS"-Schaltfläche drückt, so öffnet sich ein kleines JavaScript-Fenster. Dort können wir unsere Manipulation direkt hineinschreiben[32]. Dabei haben wir prinzipiell zwei Möglichkeiten auf die Anwendungsparameter zuzugreifen. Entweder wir schreiben dort

```
app.params.apRotAngle = -90;
```

oder

```
app.params['apRotAngle'] = -90;
```

Das Ergebnis ist das Gleiche (im Screenshot rechts sind beide Varianten verwendet). Die Art des Zugriffs sollte egal sein.



[32] Alternativ könnten wir auch eine Funktion aufrufen, die wir im Startseite.js definieren – ganz ähnlich wie wir es in Kapitel 2.4.5 gemacht haben.

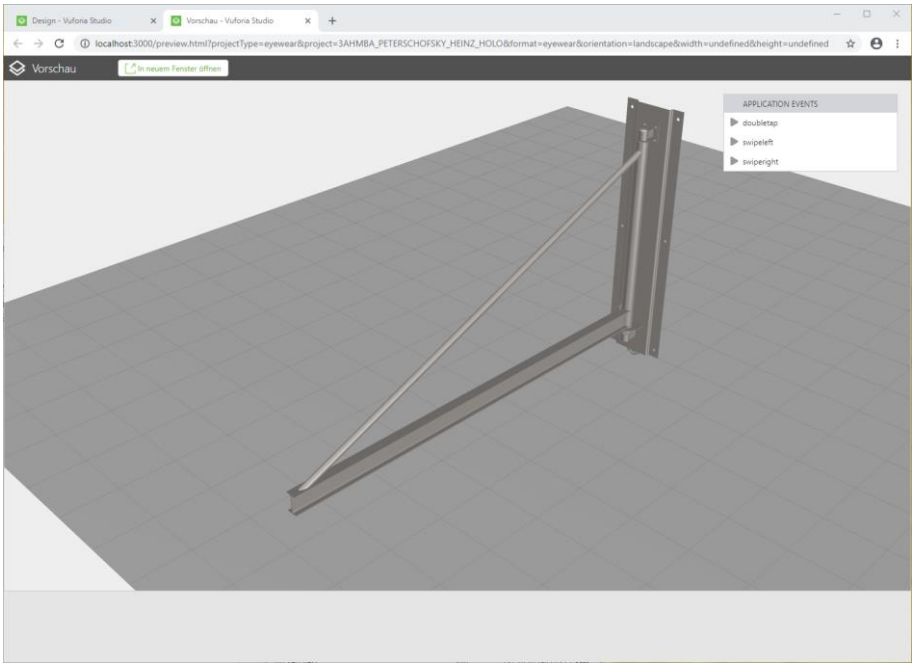
Wir wollen also beim Doubletap den Wert wieder auf -90 setzen. Bei Swipeleft soll der Wert um 10 verringert werden:

```
app.params['apRotAngle'] = app.params['apRotAngle'] - 10;
```

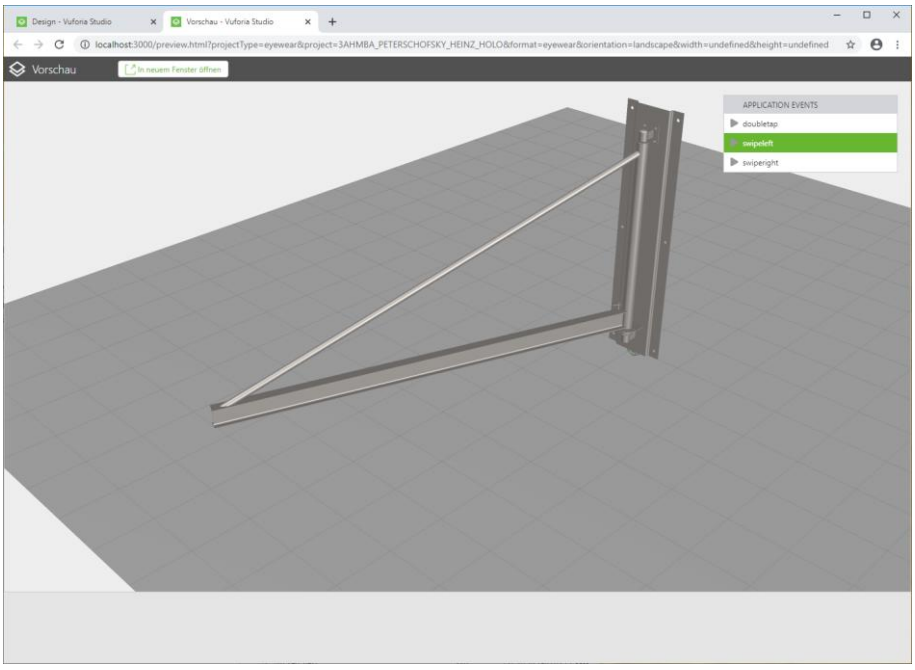
Und bei Swiperight soll der Wert um 10 erhöht werden:

```
app.params['apRotAngle'] = app.params['apRotAngle']  + 10;
```

Wir können das Ganze jetzt wieder mit der Vorschau ausprobieren. Es bringt natürlich nichts, wenn wir vor dem Bildschirm herumfuchteln. Deswegen sind die Gestiken rechts oben zur Auswahl dargestellt.

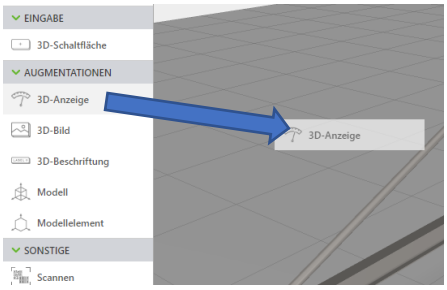


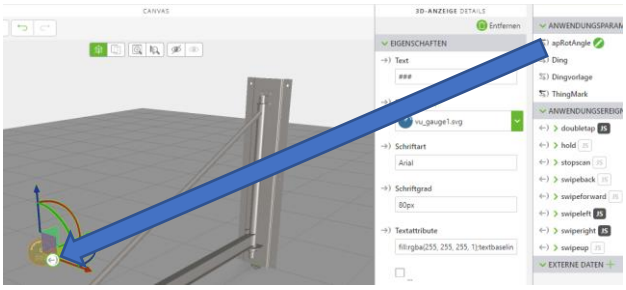
Ein Klick auf ein Event und es wird ausgeführt. Versuchen wir z.B. Swipeleft und wir werden sehen, der Kran dreht sich in die richtige Richtung:



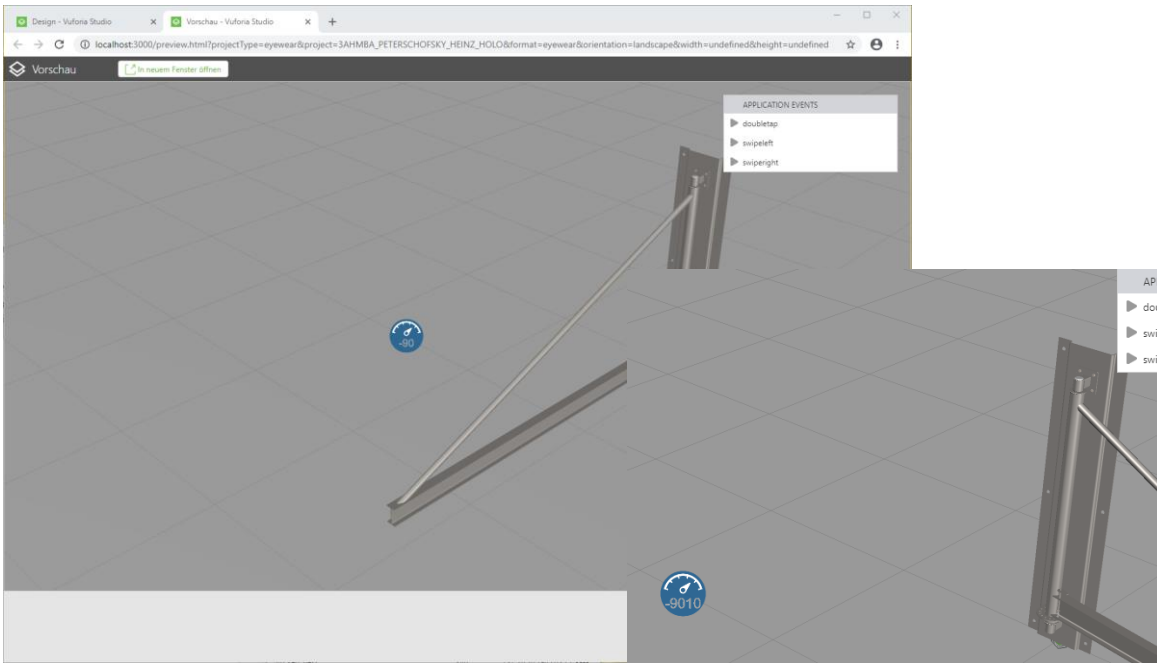
Ganz zu Beginn zeigt unser Kran manchmal ein komisches Verhalten. Er springt bei ersten Swiperight irgendwo hin. Passiert vorher ein Swipeleft oder ein Doubletap, so ist alles in Ordnung.

Um das Verhalten zu analysieren wollen wir uns den Wert auf einer Anzeige darstellen lassen. Dazu verwenden wir das Widget 3D-Anzeige. Dieses ziehen wir wieder in den Canvas und verbinden es mit dem Applikationsparameter, und zwar mit der Eigenschaft „Text".

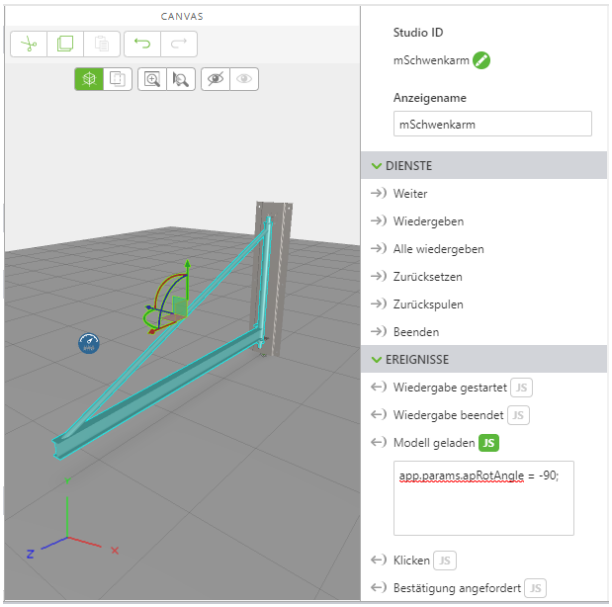




Jetzt sollte der Wert des Parameters angezeigt werden. Verwenden wir also die Vorschau und sehen nach – Tastsächlich zeigt die Anzeige jetzt -90 an. Jetzt sind wir gespannt und drücken „swiperight". Siehe da: Die Anzeige wechselt auf -9010!



Das gibt uns einen Hinweis auf das, was passiert: Offensichtlich wird die Zahl als Zeichenkette interpretiert und es wird einfach die Zeichenkette 10 angefügt. Wenn wir weiter „swiperight" drücken, so erhalten wir tatsächlich: -901010, -90101010, …



Das kann bei JavaScript leider passieren, dass ein Typ nicht automatisch korrekt erkannt wird. Nun, die praktikabelste Lösung, die mir eingefallen ist, ist beim Laden des Modells den korrekten Wert zu setzen. Wenn wir das Modell `mSchwenkarm` auswählen, so gibt es das Ereignis „Modell geladen". Dort fügen wir den gleichen Aufruf wie bei Doubletap ein – dann funktioniert das sehr gut.

Wir können das in der Vorschau überprüfen. Tatsächlich wechselt der Wert jetzt ordnungsgemäß von −90 auf −80. Passt, Problem gelöst – Die Anzeige können wir wieder löschen.

Wir können die Experience veröffentlichen und mit der realen Hardware ausprobieren.

2.4.7.1 *Pflichtaufgabe „3D-Device Input"*

	Erstellen Sie so wie oben beschrieben Ihre Eingabe für die Augmented Reality mit einem Modell Ihrer Wahl. Veröffentlichen Sie diese und führen Sie Ihr Modell vor.	

2.4.7.2 *Zusatzaufgaben „3D-Device Input limited"*

	Modifizieren Sie die Experience so, dass sich der Drehwinkel nur in einem sinnvollen Bereich verändern lässt. Es sollen keine „Überdrehungen" und somit Kollisionen möglich sein.	

2.4.8 Weitere Eingabemöglichkeiten

Video-Link 14: https://youtu.be/YqujtR6Y6oc

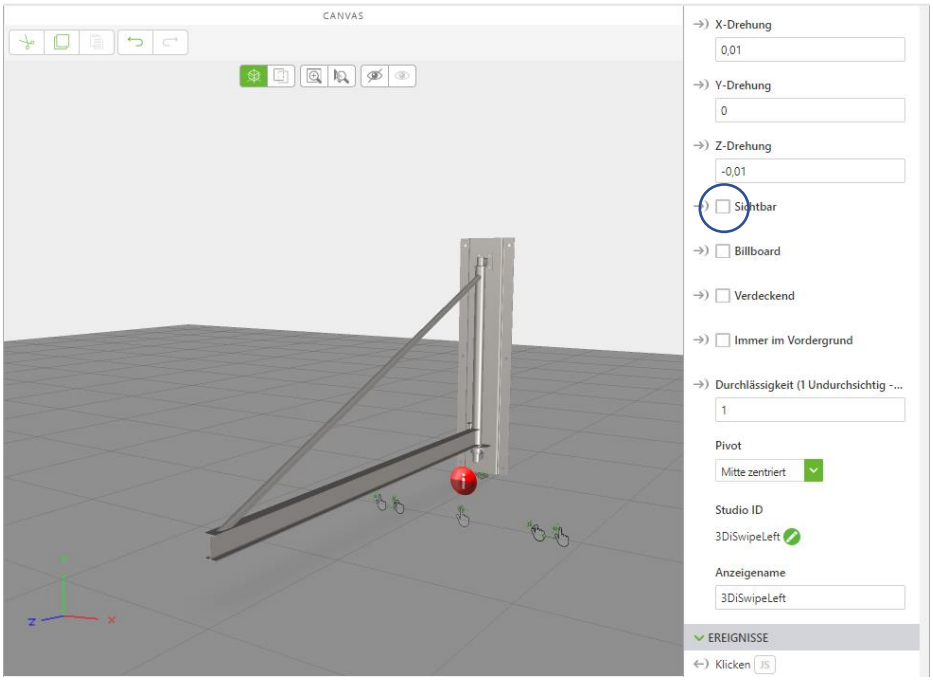
Es gibt die Möglichkeit 3D-Bilder oder -Beschriftungen als Schaltflächen auszuführen. Bei der HoloLens 2 gibt es auch die Möglichkeit eine 3D-Schaltfläche (Knopf) zu platzieren. Genau das wollen wir probieren: Wir werden uns Information anzeigen lassen, die uns ein wenig Erklärung zur Bedienung geben. Dazu wollen wir ein 3D-Bild verwenden. Also ziehen wir zunächst das zugehörige Widget in den Canvas. Auch dort müssen wir eine Ressource wählen – nun, das kennen wir ja schon vom Bildziel aus Kapitel 2.4.4. Ich habe ein Bild mit einem „i"-Symbol gemacht, das möchte ich einblenden. Ich stelle mir vor, dass wir eine Erklärung mit einem Klick einblenden können. Also verwende ich das Symbol als Ressource und platziere es schön unter meinem Kran. Dann nehme ich mir noch ein paar 3D-Bilder und verwende dort Standard-Bilder Doubletap, Swipeleft und Swiperight. Dazu muss man keine Ressource hochladen, sondern einfach eine Standard-Graphik wählen[33].

[33] Diese entsprechen den Graphiken in der Tabelle der Anwendungsereignisse weiter oben im Skriptum.

Ich vergebe die Studio-IDs:

`3DiIButton` `3DiDoubleTap` `3DiSwipeLeft` `3DiSwipeRight`

Die Sichtbarkeit von den Standrad-Symbolen drehen wir ab (Hacken wegklicken). Diese sollen erst sichtbar werden, wenn wir auf unser „i" klicken.



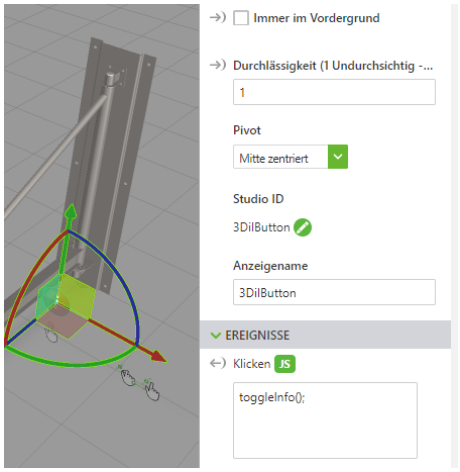
Um die Sichtbarkeit zu steuern müssen wir wieder ein JavaScript programmieren. Ganz ähnlich wie wir das schon in Kapitel 2.4.5 gemacht haben. Wir klicken also im Projektbaum auf „Startseite.js" und fügen dort folgenden Code ein:

```
$scope.toggleInfo = function() {
  $scope.app.view['Startseite'].wdg['3DiDoubleTap']['visible'] =
     !$scope.app.view['Startseite'].wdg['3DiDoubleTap']['visible'];
  $scope.app.view['Startseite'].wdg['3DiSwipeLeft']['visible'] =
     $scope.app.view['Startseite'].wdg['3DiDoubleTap']['visible'];
  $scope.app.view['Startseite'].wdg['3DiSwipeRight']['visible'] =
     $scope.app.view['Startseite'].wdg['3DiDoubleTap']['visible'];
}
```

Wir definieren uns also eine Funktion mit dem Namen `toggleInfo`. Das Einzige, was diese Funktion macht ist, dass die Sichtbarkeit von `3DiDoubleTap` invertiert[34] wird (also was nicht sichtbar war wird sichtbar und umgekehrt). Die Sichtbarkeiten von `3DiSwipeLeft` und `3DiSwipeRight` werden dann der Sichtbarkeit von `3DiDoubleTap` nachgeführt.

[34] Bitte das „ ! " bei der Zuweisung beachten!

Immer wenn wir die Funktion aufrufen werden, also die 3D-Bilder `3DiDoubleTap`, `3DiSwipeLeft` und `3DiSwipeRight` aus- bzw. eingeblendet. Diese Funktion muss also nur mehr aufgerufen werden, wenn wir `3DiIButton` klicken. Es gibt bei diesem Bild auch ein Ereignis „Klicken". Dort füllen wir ein JavaScript ein und rufen unsere Funktion auf – mittlerweile ein Klacks für uns.



Das können wir jetzt in der Vorschau probieren und sieh da: Es klappt!

Also veröffentlichen und „in Echt" probieren.

2.4.8.1 *Pflichtaufgabe „3D-Image Input"*

	Erstellen Sie so wie oben beschrieben Ihre Eingabe für die Augmented Reality mit einem Modell Ihrer Wahl. Veröffentlichen Sie diese und führen Sie Ihr Modell vor.	

2.4.8.2 *Zusatzaufgaben „3D-Image Input enhanced"*

	Modifizieren Sie die Experience so, dass beim Klick auf die Symbole Doubletap, Swipeleft/-right die gleiche Aktion passiert wie bei der Aktion selbst.	

3 Internet of Things

Playlist-Link: https://youtube.com/playlist?list=P LVut1tKPvtvP-qYca6ecv-Fa2ugt_UpHP

3.1 Allgemeines

Video-Link 1: https://youtu.be/mqYFD1VMdKc

Oftmals ist es wünschenswert Daten, welche irgendwo in einer Anlage bzw. Maschine entstehen zur Verfügung zu haben. Manchmal benötigt ein anderer Anlagenteil die Informationen, ein Andres Mal möchte man diese Messwerte dazu verwenden, um sie zentral darzustellen. Manchmal möchte man vielleicht auch eine Augmented Reality Darstellung mit momentanen Werten anreichern – z.B. indem man die aktuellen Messwerte der Anlage direkt ins Gesichtsfeld einblendet. Oftmals ist nicht nur der aktuelle Wert interessant, sondern auch der zeitliche Verlauf über die letzten Minuten oder Stunden.

Allen diesen Dingen ist gemeinsam, das die Daten irgendwie vom Ort des Entstehens (Sensor bzw. Messeinrichtung) zum Anzeigeort gebracht werden müssen. Dazu ist eine Art „Datenvermittler" notwendig. Ein Server der von allen Teilnehmern erreicht werden kann. Einige Teilnehmer legen ihre aktuellen Messwerte auf dem Server an. Andere Teilnehmer holen sich die Daten ab. Dabei ist irrelevant wer zu welchem Zeitpunkt die Daten schreibt bzw. abholt. Jeder Teilnehmer kann seine Daten veröffentlichen und andere Daten beziehen – es gibt also normalerweise keine „exklusiven" Rollen.

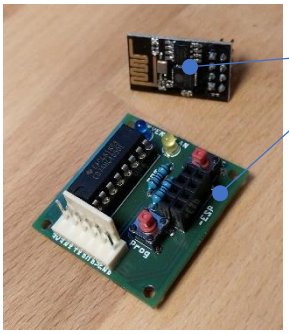
Dadurch ist es möglich Daten innerhalb von Systemen und via Internet auch über Systemgrenzen hinaus auszutauschen – das sogenannte Internet of Things – die totale Vernetzung.

3.2 Anforderungen

- Ein Server, der die Daten entgegennehmen kann und zur Verfügung stellt
- Der Server muss von allen Teilnehmern erreichbar sein
- Die Kommunikation muss sicher sein (nur berechtigte Teilnehmer sollen Daten ablegen bzw. empfangen dürfen)
- Übertragung soll trotz Zeitverzögerung in der Kommunikation zuverlässig funktionieren.

3.3 Anpassung unsers Arduino-Kits

Leider ist unser Arduino Uno nicht netzwerkfähig. Es gibt sogenannte WLAN- und Netzwerk-Shields, welche diese Funktionalität hinzufügen. Diese belegen aber immer ein paar IO-Pins und man ist nicht sehr flexibel. Es gibt Controller die WLAN-fähig sind und ebenfalls mit der Arduino IDE programmiert werden können. Einer dieser Controller ist z.B. der ESP8266. Diesen gibt es in verschiedensten Ausführungen. Die einfachste ist der ESP8266-1. Er hat eine serielle Schnittstelle, zwei General-Purpose-IOs und kann entweder als WLAN-Accesspoint oder WLAN-Client arbeiten.



Komponenten

ESP8266-1 Controller

Adapterplatte

Anschlüsse:

3V3	Spannungsversorgung +3,3V
RX	Receive-Leitung (zu ESP)
TX	Transmit-Leitung (von ESP)
D0	GPIO #0 (von ESP)
D2	GPIO #2 (von ESP)
GND	Spannungsversorgung GND

Zusammengebaut

Da der ESP mit $3{,}3V$ arbeitet müssen alle Signale zum ESP auch auf High-Level maximal $3{,}3V$ betragen. Für die notwendige Level-Anpassung der RX-Leitung wird eine Adapterplatte verwendet. Diese bietet folgende Features:

- Ein IC vom Typ `CD74HC4050E` wird verwendet um den Pegel des `RX` PINs auf $3{,}3V$ zu begrenzen. Dies funktioniert im Gegensatz zu Spanungsteilern auch bei hohen Frequenzen sehr zuverlässig.
- Die notwendigen Verbindungen zu den Chip-Select und -Enable-PINs (`CH_PD`) sind intern gemacht. Das bedeutet: Sobald der PIN `3V3` mit Spannung versorgt wird, wird das Programm im ESP gestartet.
- Der Reset-PIN des ESP ist auf einen Taster gezogen. Ein Druck auf den Knopf `Reset` genügt, um den ESP neu zu starten.
- Um den ESP zu programmieren, muss `GPIO0` beim Neustart auf `GND` gezogen werden. Um diese einfach zu bewerkstelligen ist ein zweiter Knopf „`Prog`" eingebaut. Dieser verbindet `GPOI0` mit `GND`. Dadurch kann das ESP-Modul in der Adapter-Platte programmiert werden.
- Für beide IOs gibt es eine LED. Dabei ist die LED mit der Beschriftung `WLAN` mit `GPIO0` und jene mit der Beschriftung `SERVER` mit `GPIO2` verbunden. Sinn der Beschriftung ist der Einsatzzweck hier: `WLAN` würde bedeuten der ESP ist per WLAN verbunden (bzw. es gibt WLAN-Verkehr). `SERVER` bedeutet der IoT-Server wurde kontaktiert bzw. es gibt Datenaustausch mit dem Server.

Das prinzipielle Vorgehen ist, dass auf dem ESP ein Programm läuft, welches sich mit dem WLAN verbindet und den IoT-Server kontaktiert. Man muss dabei also verschiedenste Informationen vom Arduino zum ESP bringen und umgekehrt. Das geschieht mit einer seriellen Schnittstelle, ganz genau so wie wir es in Kapitel 2.4.1 gemacht haben. Wir sagen dem ESP also z.B. die SSID und das Passwort des WLANs und der ESP verbindet sich damit. Dann sagen wir Ihm noch einen IoT-Server und er verbindet sich mit dem. Dann nach und nach neue Werte und diese werden vom ESP auf den genannten IoT-Server gebracht. Gibt es eine Nachricht vom Server für uns, so wird uns der ESP entsprechend informieren.

Diese ESP-Programme unterschieden sich natürlich je nach IoT-Server. Das sehen wir aber im Folgenden. Im Moment sind zwei Varianten verfügbar: für MQTT-Server (siehe Kapitel 3.4) und für Tingworx-Server (sieh Kapitel 3.5).

3.4 Message Queuing Telemetry Transport (MQTT)

Video-Link 2: https://youtu.be/FjVwyLIW7oM

MQTT ist ein offenes Nachrichtenprotokoll für Machine-to-Machine-Kommunikation (MMI), dass die Übertragung von Telemetriedaten[35] in Form von Nachrichten zwischen Geräten ermöglicht, trotz hoher Verzögerungen oder beschränkter Netzwerke. Es gibt keine Einschränkungen für Geräte. Es können also Sensoren und Aktoren oder auch Mobiltelefone, embedded Systems oder ganz normale Computer sein.

Dabei gibt seinen sogenannten „Broker". Dies ist nichts anderes als der zentrale Server, welcher die Daten vermittelt (daher auch die Bezeichnung). Dabei müssen die Daten nicht irgendwelche Ziffern sein, sondern können ganz allgemeine Dinge sein. Vom Bit bis zu ganzen Videos oder anderer Dateien.

Solche Daten haben einen Namen und einen Inhalt. Der Name wird dabei „Topic" genannt und ist hierarchisch strukturiert. Das sieht so ähnlich wie ein Ordnerpfad aus. Ein Beispiel eines solchen Topics wäre `Haus/Wohnzimmer/Couch/Belegung` oder etwas ernsthafter auch `Pneumatik/Windkessel/Druck`.

Clients können an einem Broker solche Topics senden oder auch abonnieren. Empfängt der Broker ein Topic so kann er diese Nachricht speichern (auf Wunsch des Clients). Der Broker hat also immer die gesamte Datenlage seiner Clients. Hat ein anderer Client ein Topic abonniert, dann wird die entsprechende Nachricht an diesen Client weitergeleitet – das können dann natürlich auch mehrere sein. Genauso können mehrere Clients dasselbe Topic senden (z.B. mehrere Bedienstellen für ein und denselben Antrieb).

Deswegen ist MQTT eine interessante Möglichkeit für die Automatisierung und IoT. Seit 2013 wird MQTT deswegen auch als Protokoll für IoT standardisiert. Prinzipiell gibt es MQTT, welches nur für TCP/IP Netzwerke funktioniert. Mit MQTT-SN[36] ist eine Erweiterung für Sensornetzwerke die nicht über TCP/IP funktionieren (z.B. ZigBee) verfügbar.

Nachrichten bestehen also immer aus einem Topic und einem Inhalt. Dabei kann der Nachricht ein Quality Of Service zugewiesen werden. Folgend QoS sind möglich:

- **At most once (Level QoS 0):** Die Nachricht wird einmal gesendet – fertig. Bei einem Verbindungsabbruch kommt die Nachricht möglicherweise nicht an.
- **At least once (Level QoS 1):** Die Nachricht wird solange gesendet bis deren Empfang bestätigt wird. Das bedeutet, das eine Nachricht möglicherweise öfter empfangen wird, weil die entsprechende Bestätigung nicht „durchkommt".
- **Exactly once (Level QoS 2):** Die Nachrichtkommt exakt einmal an. Das gilt auch bei Verbindungsabbrüchen oder -unterbrechungen.

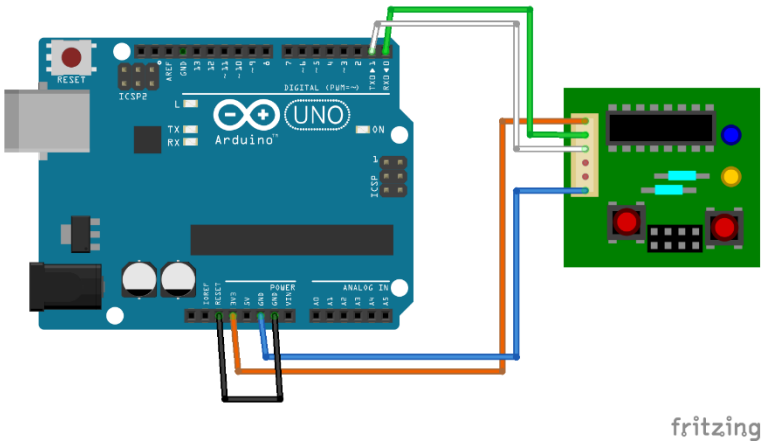
Der Client kann für seine Nachricht auch ein sogenanntes „Retain-Flag" setzen. Das bedeutet der Broker soll doch bitte diese Nachricht speichern. Liegt eine gespeicherte Nachricht zu einem Topic vor und ein Client abonniert dieses Topic wird die gespeicherte Nachricht übermittelt und nicht erst wenn eine neue Nachricht zu diesem Topic eintrifft.
Außerdem ist es Clients möglich einen „Last Will" als Nachricht zu definieren. Falls die Verbindung zum Client verloren geht wird diese Nachricht an die Abonnenten gesendet.

3.4.1 Kommunikation mit MQTT

Video-Link 3: https://youtu.be/bKwWlMEQZqI

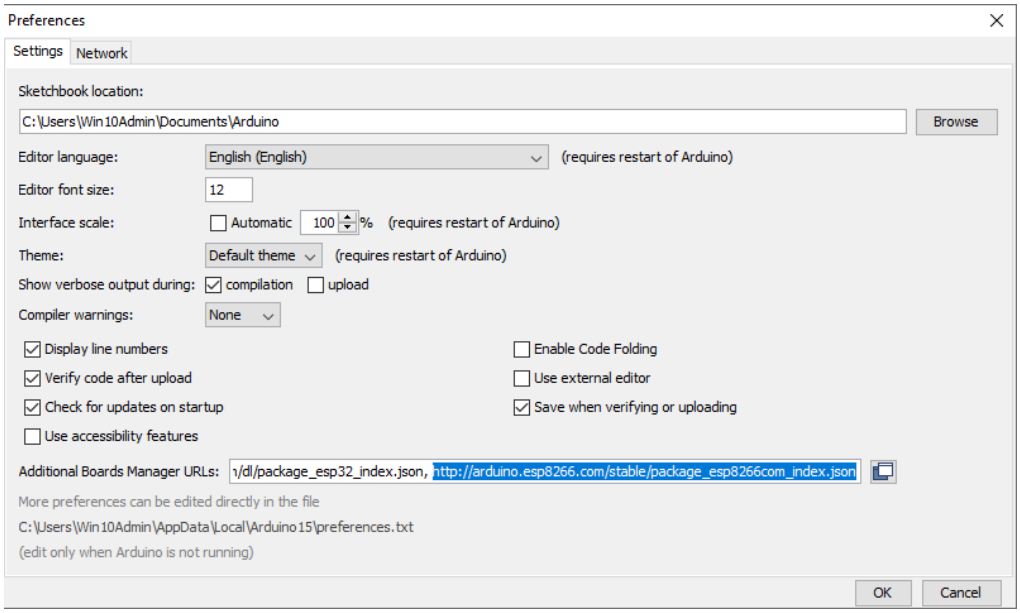
[35] Telemetrie: „Fernmessung"; von altgriechisch: Tele „fern" und Metron „Maß")
[36] SN: Sensor Networks.

Um unseren Arduino fit für die Kommunikation mit einem MQTT-Broker zumachen müssen wir Ihm Netzwerkfähigkeit bescheren. Wie schon in 3.3 beschrieben verwenden wir dazu einen eigenen WLAN-fähigen Microcontroller: einen ESP8266-1 mit einem entsprechenden Adapter-Board. Diesen schließen wir zunächst wie im gezeigten Schema an unseren Arduino an.



Die Verbindung zwischen `RESET` und `GND` ist kein Fehler. Dadurch wird unser Arduino in den USB-Adapter-Modus geschalten. Der ganze USB-Verkehr wird zu den Pins 0 (RXD) und 1 (TXD) geleitet. Wir können also mit dem Gerät kommunizieren, welches „hinter" dem Arduino hängt, ohne dass dieses einen USB-Anschluss braucht. Unser ESP hat keinen USB-Anschluss und so können wir ihn trotzdem erreichen und programmieren.

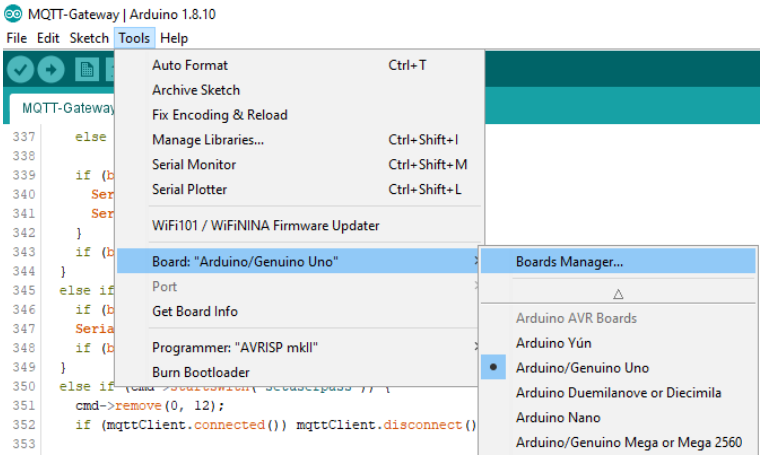
Zunächst müssen wir unserer Arduino IDE jedoch noch beibringen, dass es ein solches ESP-Board gibt. Dieses gibt es nämlich nicht standardmäßig. Wir müssen einen eigenen sogenannten „Board-Manager" installieren. Dazu öffnen wir die Einstellungen aus dem Datei-Menü:



Dort gibt es ein Eingabefeld mit zusätzlichen Board-Manger URLs. Dort ist folgende URL einzutragen (ist schon etwas eingetragen werden verschiede URLs mit Komma „,“ getrennt):

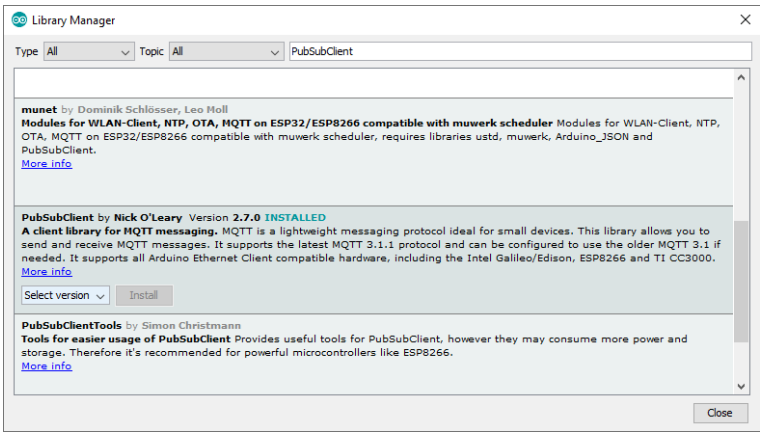
http://arduino.esp8266.com/stable/package_esp8266com_index.json

Damit können die Boards von dieser URL installiert werden, doch wie geht das? Man geht in das „Tools"-Menü und öffnen dort das Board-Untermenü und wählen den Boards-Manager:



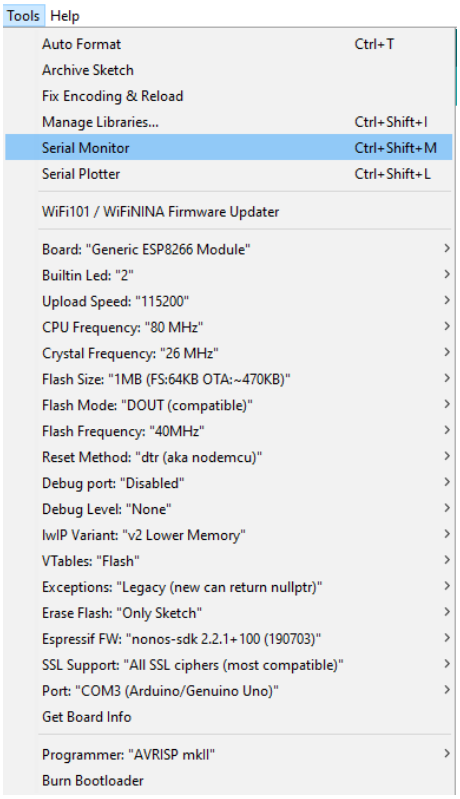
Dort kann man aus relativ vielen Quellen neue Board-Typen installieren. Wir benötigen den Typ ESP8266. Dort wählen wir die letzte Version und installieren diese:

Jetzt kennt die Arduino IDE den ESP8266-Controller. Wir wollen das notwendige Programm, um mit einem MQTT-Broker zu kommunizieren auf den ESP laden. Dazu wurde das Programm `MQTT-Gateway.ino` zur Verfügung gestellt. Dieses ist mit der Arduino IDE zu öffnen.



Was uns noch fehlt ist die Bibliothek `PubSubClient.h`[37]. Diese müssen wir mit Manage Libraries… einbinden. Diese Bibliothek benötigen wir, um die MQTT-Funktionalität abzudecken. Zu Erinnerung: Der ESP-Controller soll die Verbindung mit dem MQTT-Broker herstellen, also benötigen wir diese Bibliothek, um dies zu bewerkstelligen.

[37] Von Nick O'Learey – Achtung es gibt viele „Wrapper" die ähnlich heißen – wir verwenden das Original.

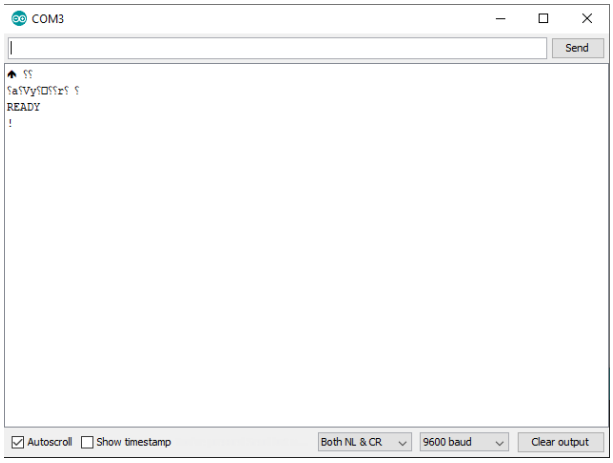


Haben wir diese Dinge alle installiert können wir das bereitgestellte Programm `MQTT-Gateway.ino` laden[38]. Als Board wählen wir „`Generic ESP8266 Module`" und der COM-Port ist der Port unseres Arduino.

Wir haben ein Board eingestellt, einen COM-Port und haben ein Programm. Von unserem Arduino sind wir gewohnt: Jetzt drücken wir auf Upload und das war's. Leider funktioniert das beim ESP nicht so einfach. Diesen müssen wir erst in den Programmiermodus versetzen. Dazu müssen wir folgende Reihenfolge einhalten:

1. Drücken der Taster `Prog`
2. Drücken der Taste `Reset`
3. Loslassen der Taste `Reset`
4. Loslassen der Taste `Prog`
5. Download des Programms

Um zu testen, ob das Programm auf dem Controller aufgebracht wurde können wir den seriellen Monitor öffnen. Die richtige BAUD-Rate ist 9600. Drücken wir jetzt kurz die Reset-Taste am Verbindungsboard sollten wir das Wort `READY` lesen können. Wenn wir „`?`" eingeben so sollte sich der Controller mit der Antwort „ ! " zurückmelden. Beide LEDs leuchten – das heißt wir sind weder mit dem WLAN noch mit dem MQTT-Broker verbunden.



[38] Man kann versuchen zu verstehen, was da drinnen passiert. Es ist relativ übersichtlich geschrieben. Nur die Funktion `ExecuteCmd()` hat eine gewisse Länge, das ist aber Großteiles immer gleich.

Video-Link 4: https://youtu.be/HQ5K3jFHgJo

Wir wollen uns jetzt einmal „händisch" zu unserem MQTT-Server verbinden. Dazu schalten wir uns zunächst alle möglichen Ausgaben vom ESP ein. Wir tippen in den Serial Monitor nacheinander die Befehle

```
startwarn
startinfo
startdebug
```

ein.

Wenn wir jetzt noch `getverbose` verwenden, bekommen wir zur Bestätigung die Ausgabe:

```
ERROR WARN. INFO. DEBUG (240)
  X     X     X     X
```

Die „X" kennzeichnen, dass die Ausgaben für Fehler, Warnungen, Informationen und Debug ausgegeben werden. Jeder Befehl wird jetzt mit einer Menge Rückmeldung quittiert. Versuchen wir uns zunächst mit dem WLAN der Schule zu verbinden. Dazu verwenden wir die Befehle:

```
setssid WifISSID
```

Rückmeldung:

```
Set WifiSSID: 'WifISSID'
OK
```

```
setpass WifiPassword
```

Rückmeldung:

```
Set WifiPass: 'WifiPassword'
OK
```

Sobald eine SSID und ein Passwort übermittelt wurden versucht der ESP eine Verbindung mit dem angegeben Netzwerk aufzubauen. Wenn nicht gleich eine Verbindung zustande kommt, versucht es der Controller immer wieder. Ist die SSID oder das Passwort falsch reicht es den Wert mit dem entsprechenden Befehl zu korrigieren. Die Ausgabe sollte ähnlich wie hier aussehen:

```
Try to connect to WiFi.
Connecting to: WifISSID
.....................Could not connect.
ERR
Try to connect to WiFi.
Connecting to: WifISSID
...... Connected!
OK
New MQTT-Client-ID: 'ESP8266Gateway_50:02:91:B0:17:D0'
```

Ist die Verbindung erfolgreich sollte die LED WLAN ausgegangen sein. Dies hat energiespartechnische Gründe. Beim Betrieb mit z.B. eine Batterie ist es unnötig, wenn hier im Normalzustand irgendwelche LEDs leuchten.

Der erste Schritt ist also geschafft – wir sind mit dem WLAN verbunden. Auch wurde bereits der zweite Schritt vorbreitet. Wir erkennen, dass bereits ein MQTT-Client-Name vergeben wurde. Dieser besteht aus einem Präfix und der MAC-Adresse der ESP-WLAN-Schnittstelle. Das sollte also einzigartig sein.

Jetzt müssen wir uns mit dem MQTT-Broker verbinden. Hier an der Schule läuft ein MQTT-Server auf einem Raspberry Pi. Dieser hat die Adresse `210.192.168.1`[39]. Also geben wir diese Information an den ESP weiter. Wir benutzen den Befehl:

[39] Bitte vergewissern Sie sich beim unterrichtenden Lehrer über die Gültigkeit der IP-Adresse.

```
sethost 210.192.168.1
```

Sofort versucht sich der Controller mit dem MQTT-Broker verbinden. Die Ausgabe wird wohl ähnlich wie im Folgenden gezeigt aussehen:

```
Set Host: '210.192.168.1'
OK
Connecting to MQTT-Broker '210.192.168.1'
Error connecting to MQTT-Broker '210.192.168.1' Reason code: '5'
ERR
Connecting to MQTT-Broker '210.192.168.1'
Error connecting to MQTT-Broker '210.192.168.1' Reason code: '5'
ERR
...
```

Alle paar Sekunden wird der Controller versuchen den MQTT-Broker zu erreichen. Er schafft es jedoch nicht. Es wird auch noch ein Reason-Code angegeben. Dieser bedeutet:

Code	Ursache
-4	Timeout (Keine Antwort des Servers)
-3	Verbindung zurückgesetzt (Netzwerkverbindung unterbrochen)
-2	Verbindung fehlgeschlagen (keine Netzwerkverbindung)
-1	Nicht Verbunden (Fehlerfreier Verbindungsabbau)
0	Verbunden
1	Protokollfehler (Versionskonflikt)
2	Client ID nicht akzeptiert
3	Verbindung nicht möglich
4	Username oder Passwort falsch
5	Nicht berechtigt

Wir sind offensichtlich nicht berechtigt auf den Broker zu verbinden. Der verwendete Broker verlangt nämlich Username und Passwort. Nun, das können wir übertragen. Die richtigen Dinge sind:

Username: `USER_STP` Passwort: `USERinnovativ`

Wir verwenden also die Befehle:

```
setusername USER_STP
setuserpass USERinnovativ
```

Die Ausgabe wird dann wohl so ähnlich aussehen:

```
Set Username: 'USER_STP'
OK
Set Userpass: 'USERinnovativ'
OK
Connecting to MQTT-Broker '210.192.168.1'
Connected!
OK
```

Zur Bestätigung ist auch die SERVER-LED ausgegangen. Hin und wieder blinken die LEDs. Das ist ein Zeichen, dass das Programm mit dem Server kommuniziert. Gratulation, wir haben es geschafft. Wir sind mit dem MQTT-Broker verbunden und können dort unsere Informationen veröffentlichen. Aber wie geht das?

Unser Programm unterstützt 50 verschiedene Topics, oder Properties. Diese können wir anlegen und einen Namen vergeben und danach brauchen wir nur mehr die Nummer und den Wert übertragen. Wir verwenden also die Befehle:

```
setpropname 0 3AHMBA/Peterschofsky/Text
```

Wir bekommen auch die Bestätigung:

```
Set Property Name #0 : '3AHMBA/Peterschofsky/Text'
OK
```

Wenn wir jetzt einen neuen Wert übertragen wollen so reicht jetzt nurmehr eine Referenz auf die Nummer:

```
setpropval 0 Hallo
```

Die Antwort folgt auf dem Fuße:

```
Set Property Value #0 : 'Hallo'
OK
```

Passiert dies aber auch wirklich? Wir können uns z.B. auf das Property verbinden (oder auch auf das des Nachbarn). Solch ein Topic zu abonnieren, funktioniert mit dem Befehl:

```
setsubscribe 0 3AHMBA/Peterschofsky/Text
```

Als Betätigung erhalten wir:

```
Set Subscribe Topic #0 : '3AHMBA/Peterschofsky/Text'
OK
```

Und etwas später taucht auch schon die Zeile

```
*#RCV#0#'Hallo'
```

auf. Der letzte Wert wurde uns übertragen. De ESP wiederholt nämlich den letzten übertragenen Wert in Zeitabständen. Also sammeln sich jetzt die Zeilen. Wir können jetzt z.B. schreiben:

```
setpropval 0 Servus
```

und wir erhalten folgende Ausgabe:

```
Set Property Value #0 : 'Servus'
OK
*#RCV#0#'Servus'
```

Um die Wiederholzeit zu ändern können wir auch den Befehl

```
setresendtime 60
```

verwenden. Dieser würde z.B. die Wiederholzeit auf 60 Sekunden setzen. Dadurch wird der letzte Wert nicht andauernd wiederholt, sondern eben nur alle 60 Sekunden. Der maximal einstellbare Wert ist 3600 Sekunden, also einmal pro Stunde. Man braucht dann aber keine Angst haben, dass Werte nicht übertragen werden. Sollte ein neuer Wert übertragen werden, so wird dieser sofort weggesendet. Dieses Verhalten würde jedoch beim andauernden Eintreffen von neuen Werten zu dauerhafter Übertragung führen. Deswegen ist auch eine Blockierzeit eingebaut. Das bedeutet, dass

zwei Messwerte nicht unmittelbar hintereinander übertragen werden, sondern erst nach Ablauf der Blockierzeit. Um die Blockierzeit einzustellen kann man z.B. den Befehl:

```
setblocktime 0.2
```

verwenden. Dieser würde z.B. diese Blockierzeit auf 0,2 Sekunden setzen. Schneller eintreffende Daten werden nicht weitergeleitet (erst nach Ablauf der Resend-Time).

Neben dem Befehl `setpropname` gibt es noch `setboolname` und `setvaluename`. Mit begleitend sind dann `setboolval` und `setvalue`. Insbesondere der Value-Teil ist interessant, um Messwerte zu übertragen:

Werte schwanken üblicherweise ein wenig. Wenn jede Änderung übertragen wird, so ist dies unnötig. Deswegen kann man bei den Value-Werten auch eine Übertragungsschwelle festlegen. Man kann also immer einen neuen Wert zum ESP übertragen und erst wenn der neue Wert um den Schwellwert anders ist als der zuletzt Übertragene erfolgt eine neuerliche Übermittlung zum MQTT-Broker. Dadurch werden kleine Änderungen nicht (erst mit dem Ablauf der Resend-Time), große Änderungen aber sofort übertragen.

Eine Auflistung der Befehle zum ESP-Gateway befindet sich im Anhang.

3.4.1.1 Übungsaufgabe „MQTT-Connect"

	Laden Sie so wie oben beschrieben die Software auf Ihren ESP-Controller. Verbinden Sie sich mit dem MQTT-Broker und veröffentlichen Sie ein Thema Ihrer Wahl. Sehen sie ob Sie dieses Thema auch selbst abonnieren können.	

3.4.1.2 Pflichtaufgabe „MQTT-Chat"

	Verwenden Sie ein allgemeines Topic und versuchen Sie mit einem Partner Ihrer Wahl über veröffentlichte MQTT-Topics zu „chatten".	

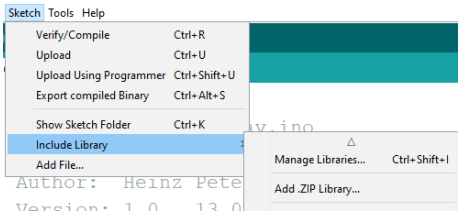
3.4.2 Übertragen aus einem Arduino-Programm

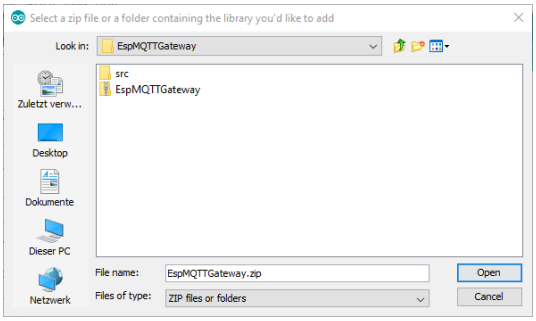
Video-Link 5: https://youtu.be/77nMC8lkdl4

Es ist ja ganz nett, dass wir uns hier verbinden können und irgendwelche Nachrichten veröffentlichen. Interessant wäre es, wenn wir ein Programm ablaufen lassen könnten und dann, wenn wir einen neuen Messwert haben diesen zu Übertragen. Wir wollen also aus einem Programm heraus oben genannte Befehle verwenden. Im Prinzip brauchen wir es nur wie in Kapitel 2.4. machen – wir erzeugen uns eine Serielle Schnittstelle, diese verbinden wir mit dem ESP und geben ihm darüber Befehle. Dem ESP ist es nämlich egal ob wir das direkt mit der Tastatur schrieben, oder ob diese Befehle von einem automatischen System kommen.

Tatsächlich habe ich für genau diesen Zweck eine kleine Library mit dem Namen „`EspMQTTGateway`" geschrieben. Prinzipiell könnte man natürlich mit `Serial.print` etc. arbeiten, doch das Timing ist relativ kritisch. Deswegen habe ich diese Funktionalität in einer Library versteckt. Diese Library ist als zip-Datei verfügbar. Um sie zu installieren müssen wir eine zip-Library hinzufügen. Dazu öffnen wir in der Arduino-IDE das Sketch-Menü und wählen dort aus dem Submenü „Include Library" den Eintrag „Add .ZIP Library...".

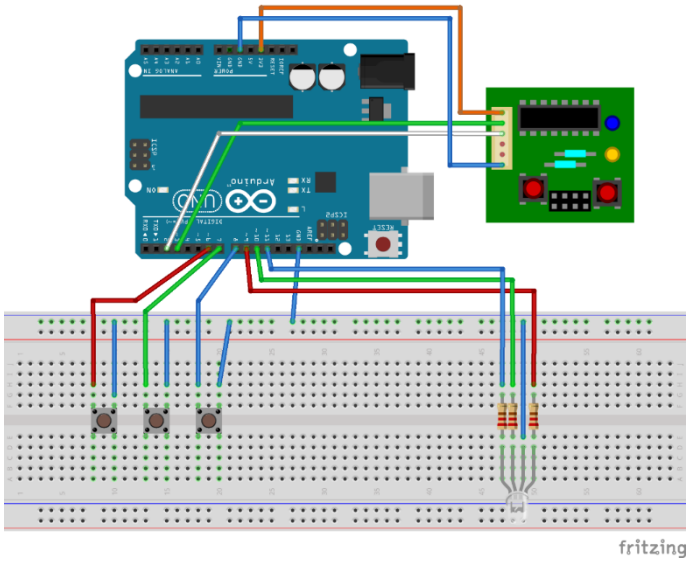
Es öffnet sich ein Öffnen-Dialog. Dort wählen wir die zip-Datei aus und wir sollten in der Lage sein die Library zu verwenden.





Das sind wir prinzipiell auch, nur benötigt diese Library auch die Library „`Timeout.h`". Die können wir ganz genauso installieren.

Der Hardwareaufbau, den wir verwenden sieht wie folgt aus[40]:



[40] Im Vergleich zu 3.4.1 müssen wir also nur die Brücke zwischen `Reset` und `GND` entfernen.

Jetzt, nach der Installation der Libraries sollte folgendes Programm „verstanden" werden und zu kompilieren sein:

```
#include "SoftwareSerial.h"
#include "EspMQTTGateway.h"
#include "Timeout.h"

#define RX_PIN                2
#define TX_PIN                3

#define RED_PIN     9
#define GREEN_PIN  10
#define BLUE_PIN   11

#define RED_SWITCH     6
#define GREEN_SWITCH   7
#define BLUE_SWITCH    8

String SSID = "WifISSID";
String PASS = "WifiPassword";
String MQTTHost = "210.192.168.1";
String MQTTUser = "USER_STP";
String MQTTPass = "USERinnovativ";

SoftwareSerial espPort(RX_PIN, TX_PIN);
EspMQTTGateway *mqttClient;

Timeout to(100);

bool red, green, blue;

void mqttReceived(int topic, String tag, float value)
{
        switch (topic) {
        case 0: //red
                red = (value > 0.0);
                break;
        case 1: //green
                green = (value > 0.0);
                break;
        case 2: // blue
                blue = (value > 0.0);
                break;
        }
}

void setup() {
        Serial.begin(115200);
        Serial.println("MQTT-Tester is starting up ...");

        red = 0; green = 0; blue = 0;

        pinMode(RED_PIN, OUTPUT);
        pinMode(GREEN_PIN, OUTPUT);
        pinMode(BLUE_PIN, OUTPUT);

        pinMode(RED_SWITCH, INPUT_PULLUP);
        pinMode(GREEN_SWITCH, INPUT_PULLUP);
        pinMode(BLUE_SWITCH, INPUT_PULLUP);

        mqttClient = new EspMQTTGateway(&espPort, &Serial, &mqttReceived);
        mqttClient->SetVerbose(16+32+64+128);
}
```

```
void loop() {
        if (!mqttClient->WifiSsidTransfered()) mqttClient->SetWifiSsid(SSID);
        if (!mqttClient->WifiPassTransfered()) mqttClient->SetWifiPass(PASS);
        if (!mqttClient->MqttHostTransfered()) mqttClient->SetMqttHost(MQTTHost);
        if (!mqttClient->MqttUserTransfered()) mqttClient->SetMqttUser(MQTTUser);
        if (!mqttClient->MqttPassTransfered()) {
                mqttClient->SetMqttPass(MQTTPass);
                mqttClient->SetBoolName(0, "meinname/led/rot");
                mqttClient->SetBoolName(1, "meinname/led/gruen");
                mqttClient->SetBoolName(2, "meinname/led/blau");
                mqttClient->Subscirbe(0, "anderer/led/rot");
                mqttClient->Subscirbe(1, "anderer/led/gruen");
                mqttClient->Subscirbe(2, "anderer/led/blau");
        }

        if (to.TimedOut()) {
                to.SetNow();
                if (mqttClient->MqttConnected()) {
                        mqttClient->SetBool(0, !digitalRead(RED_SWITCH));
                        mqttClient->SetBool(1, !digitalRead(GREEN_SWITCH));
                        mqttClient->SetBool(2, !digitalRead(BLUE_SWITCH));
                }
        }

        mqttClient->Update();

        digitalWrite(RED_PIN, red);
        digitalWrite(GREEN_PIN, green);
        digitalWrite(BLUE_PIN, blue);
}
```

Sehen wir uns den Code einmal genauer an. Beginnen wir so wie der Programmablauf bei `setup()`:

Der erste Aufruf der interessant ist lautet

```
mqttClient = new EspMQTTGateway(&espPort, &Serial, &mqttReceived);
mqttClient->SetVerbose(16+32+64+128);
```

Zuerst wird ein neues Objekt vom Typ `EspMQTTGateway` angelegt. Wir übergeben dort die Adresse der Software Serial zum ESP-Port (`&espPort`[41]), die Adresse der seriellen Schnittstelle für die Ausgaben der Bibliothek (`&Serial`) und auch den Zeiger zu der Funktion `mqttReceived` (`&mqttReceived`). Damit ist das Objekt angelegt. Zunächst setzen wir noch alle möglichen Bits und erfreuen uns an vielen Ausgaben am Debug-Port (welcher bei uns der „normale" Serielle Port ist und wir es damit am Seriellen Monitor betrachten können).

Wenden wir uns nun der referenzierten Funktion `mqttReceived` zu. Dort wird immer, wenn der empfangene Wert > 0.0 ist die Boole'schen Variablen `red`, `green` und `blue` entweder auf `true` oder `false` gesetzt. Welche Variable behandelt wird hängt von der Nummer des empfangenen abonnierten Topics ab. Immer wenn wir also z.B. das Thema 2 empfangen und der empfangene Wert > 0.0 ist so wird der Wert von `blue` auf `true` gesetzt. Was das für einen Sinn hat sehen wir, wenn wir uns die `loop`-Funktion ansehen:

Es wird sichergestellt, dass die richtigen Wi-Fi und MQTT-Parameter übertragen wurden. Dabei wird der Wert nur zum ESP übertragen, wenn er nicht schon übertragen wurde. Was hat das für einen Sinn? Die Bibliothek prüft in regelmäßigen Abständen, ob die Verbindung zum MQTT-Server und zum WLAN in Ordnung sind.

[41] Das &-Symbol kennzeichnet, dass wir nicht das Objekt selbst übergeben, sondern nur eine Adresse, oder auch einen „Pointer" bzw. Zeiger auf das Objekt. Wir geben der Bibliothek nur eine Adresse und die arbeitet dort indem Sie auf die Adresse zugreift und nicht bloß auf eine Kopie des seriellen Objekts.

Bricht z.B. aus irgendeinem Grund die Verbindung zum MQTT-Server ab so wird von der Bibliothek angenommen, dass sie möglicherweise mit falschen Parametern arbeitet und setzt die Verbindung zu MQTT zurück. Dadurch würden die MQTT-Parameter hier wieder auf den ESP übertragen werden. Das gleiche gilt sinngemäß für die Wi-Fi-Parameter. Übertragungsfehler zwischen Arduino und ESP können so korrigiert werden.

Wir veröffentlichen also die Themen[42]:

```
0: meinname/led/rot
1: meinname/led/gruen
2: meinname/led/blau
```

Und wir abonnieren die Themen[43]:

```
0: anderer/led/rot
1: anderer/led/gruen
2: anderer/led/blau
```

Jetzt wird also auch klar warum wir beim Empfang von z.B. Topic 1 die Variable `green` manipulieren – weil wir genau dort das Thema zur grünen LED abonniert haben.

Sobald wir mit dem MQTT-Server verbunden sind beginnen wir unsere Themen zu veröffentlichen. Wir können die Werte relativ oft an den ESP übertragen (hier können wir das mit dem `Timeout`-Objekt `to` aussuchen). Der sendet ohnehin erst wenn sich der Wert ändert, oder wenn die Weiderholzeit abgelaufen ist. Wir verursachen also höchstens viel Datenverkehr zwischen Arduino und ESP, aber nicht am Netzwerk. Dabei übertragen wir nun auf den einzelnen Topics die eingelesenen Schalter – bei uns tut sich dabei nichts. Bei allen die unser Topic aber abonniert haben geht die entsprechende LED an. Warum das so ist sieht man in den letzten drei Zeilen des Programms. Davor gibt es noch die Zeile

```
mqttClient->Update();
```

Dort wird einfach der Bibliothekscode aufgerufen. Lässt man diesen Aufruf weg, so kann nichts empfangen werden. Dann wären die abonnierten Themen nutzlos. Außerdem funktioniert der Check der ESP-Verbindung nicht mehr. Kurz: Es ist notwendig dies 1x pro loop aufzurufen.

3.4.2.1 Übungsaufgabe „MQTT-Transfer"

	Laden Sie so wie oben beschrieben die Software auf Ihren ESP-Controller. Versuchen Sie mit einem Kollegen sich gegenseitig die Lichter aufzudrehen.	

3.4.2.2 Pflichtaufgabe „MQTT-Analog"

	Übertragen Sie keine binäre Werte, sondern Analoge. Übertragen Sie die Anteile von Rot Grün und Blau so, dass sich eine Mischfarbe beim Kollegen einstellt.	

[42] Der erste Teil ist durch den eigenen Namen zu ersetzen. Ich würde als z.B. das Thema „`peterschofsky/led/rot`" veröffentlichen.
[43] Auch hier ist der erste Teil mit dem Namen eins Klassenkammeraden zu ersetzen. Wenn ich z.B. die Werte von Prof. Tiefenbacher abonnieren will, so abonniere ich „`tiefenbacher/led/rot`".

3.5 Thingworx

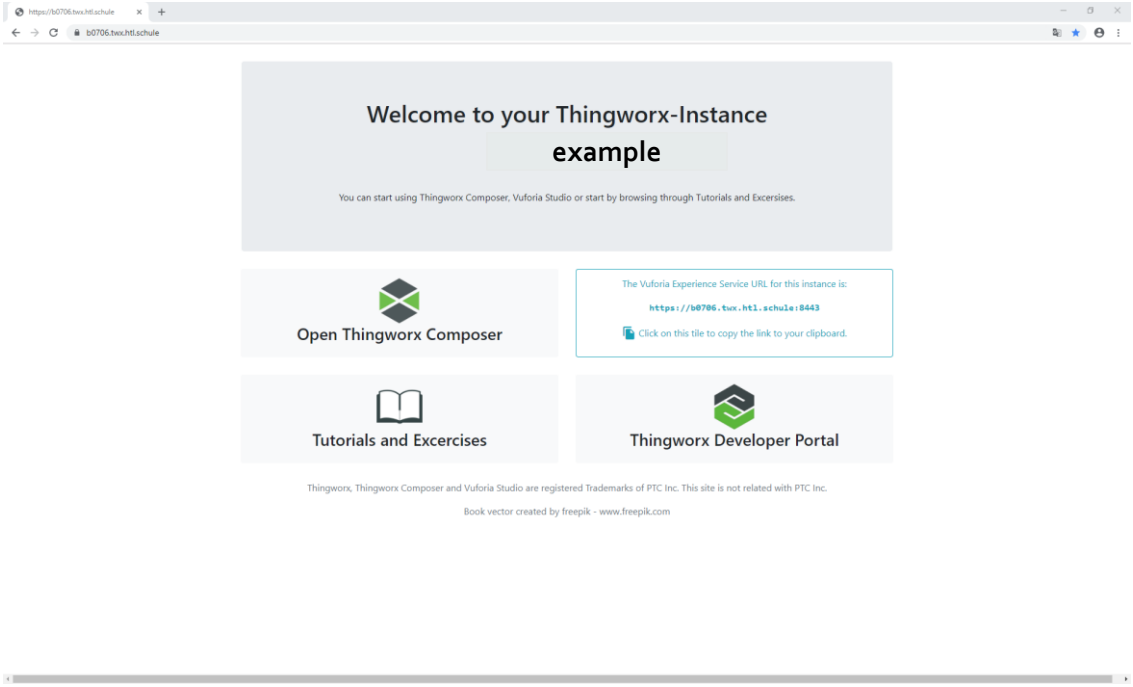
3.5.1 Allgemeines

Video-Link 6: https://youtu.be/McAREiTvEog

Thingworx ist eine IoT-Plattform[44] von PTC, welche gut integrierte Schnittstellen zu den anderen PTC-Produkten bietet. Es muss also irgendwo einen Server geben, der die Daten entgegennimmt und bei Bedarf weitergeben kann. Dieser Server heißt bei dieser Plattform Thingworx-Instance und ist unter derselben Adresse wie der Server für das Publishen von Vuforia-Experiences erreichbar (Siehe Kapitel 2.4).

https://example.twx.htl.schule

Tippt man diese Zeile in die Adressleisten eines Browsers[45] ein so erscheint der Startbildschirm:

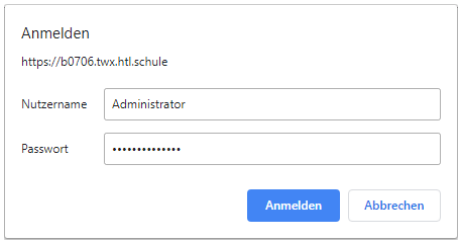


Das ist praktisch der gleiche Server, der uns auch bei Vuforia als Experience-Server gedient hat. Tatsächlich enthält der hellblau geschriebene Bereich die Experience-URL des Servers. Mit einem Klick auf diese Schrift wird die Experience-URL in das Clipboard kopiert. Das ist als Erleichterung für Vuforia-Studio gedacht und hat eigentlich nichts mit Thingworx zu tun.

Thingworx bedient man mit dem „Composer". Dieser ist mit einem Klick auf „Open Thingworx Composer" erreichbar. Es erschient wieder ein Anmelde-Dialog, in dem wir wieder unsere Anmeldedaten eingeben:

Username: **AdminUser**

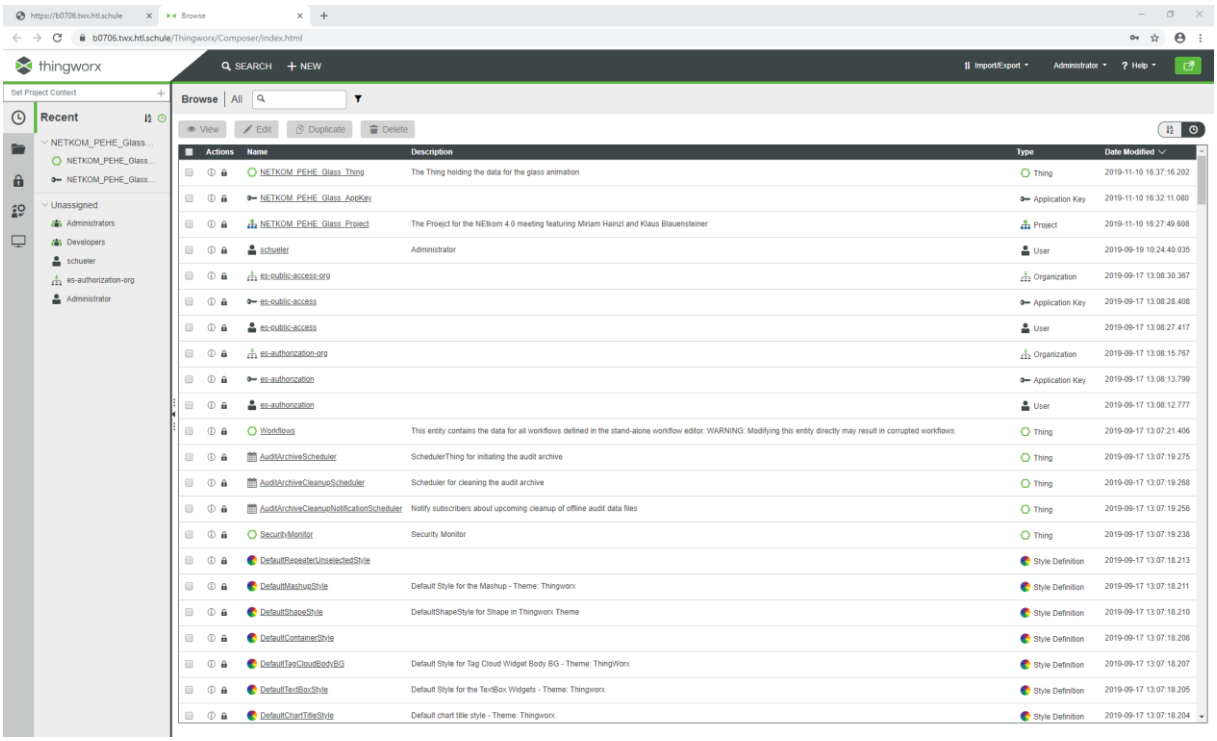
Passwort: **TWXpasswort**



[44] Zu Internet of Things siehe auch 3.1 Allgemeines

[45] Auch hier wird offiziell von PTC nur Goolge Chorme unterstützt. Alle anderen Browser können Darstellungsfehler und Funktionseinschränkungen nach sich ziehen.

Danach lädt der Thingworx-Composer. Auf den ersten Blick ein vollkommen unübersichtliches Fenster mit einem Sammelsurium aus seltsamen Namen und vielen, vielen Links.



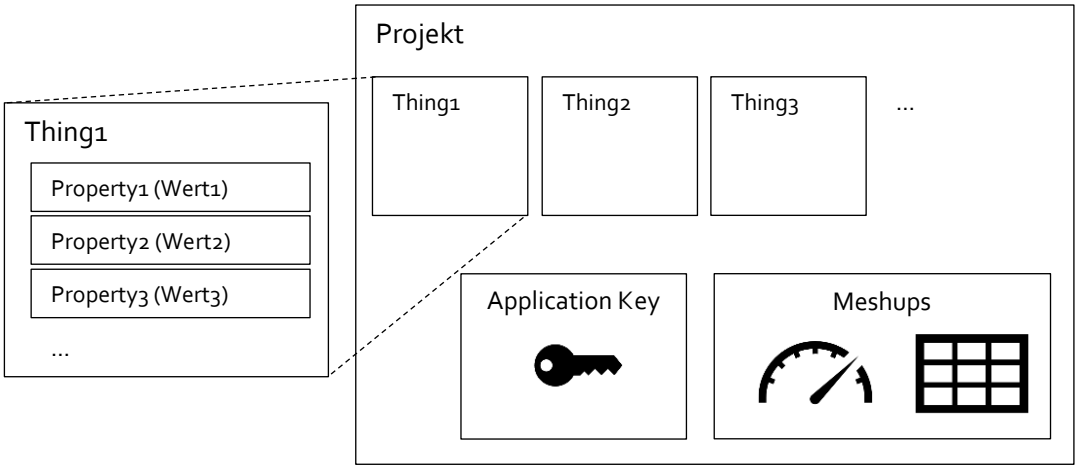
Die Daten auf solch einem Server sind in sogenannten „Things" organisiert. Ein Thing ist im wahrsten Sinn des Wortes ein Ding, welches die Daten sammeln kann, bereithält und bei Bedarf zur Verfügung stellt. Das ist das zentrale Element.

Ein oder mehrere Things werden in einem Projekt zusammengefasst. Auf Ebene der Projekte sind auch die Berechtigungen und die Darstellungssicht auf die Daten der Things angesiedelt.

Für die Berechtigung enthält ein Projekt einen sogenannten „Application Key". Wer diesen kennt, darf seine Daten in den Things ablegen. Wer die Daten abholen darf wird direkt im Thing geregelt.

Für die Darstellung enthält ein Projekt ein oder mehrere „Meshups"[46]. Dies sind praktisch Webseiten, welche die Daten darstellen können. Dort gibt es z.B. Analog- bzw. Digitalanzeigen, aber auch Balken- und Liniendiagramme. Außerdem könnte man Bedienelemente hinzufügen, Knöpfe, Regler, …

[46] Ein weiter verbreiteter Name ist auch „Dashboard" – also das Instrumentenbrett.



Zusammengehalten wird das also alles durch das Projekt. Wir werden uns in den nächsten Kapiteln damit beschäftigen, wie wir ein funktionierendes Thingworx-Projekt mit einem Thing und einem Application Key anlegen. Dann werden wir diese Daten in einem Meshup darstellen und danach werden wir diese Daten in Vuforia Studio verwenden, um eine entsprechende Darstellung zu generieren.

Als Namenskonvention verwenden wir:

KLASSE_NACHNAME_Projekt_

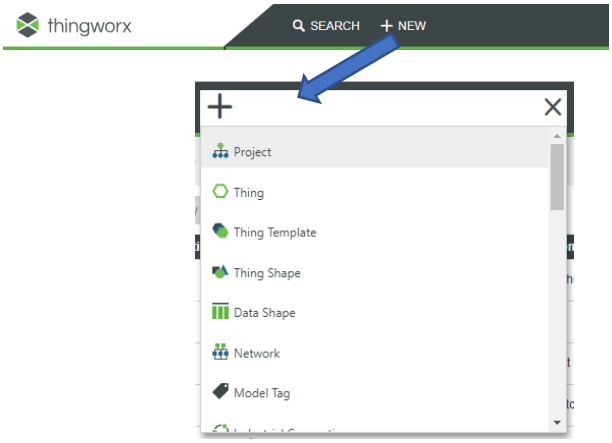
Projekt:	**KLASSE_NACHNAME_Projekt_Pr**
Thing:	**KLASSE_NACHNAME_Projekt_Th**
Application Key:	**KLASSE_NACHNAME_Projekt_Ak**
Mashup:	**KLASSE_NACHNAME_Projekt_Mu**

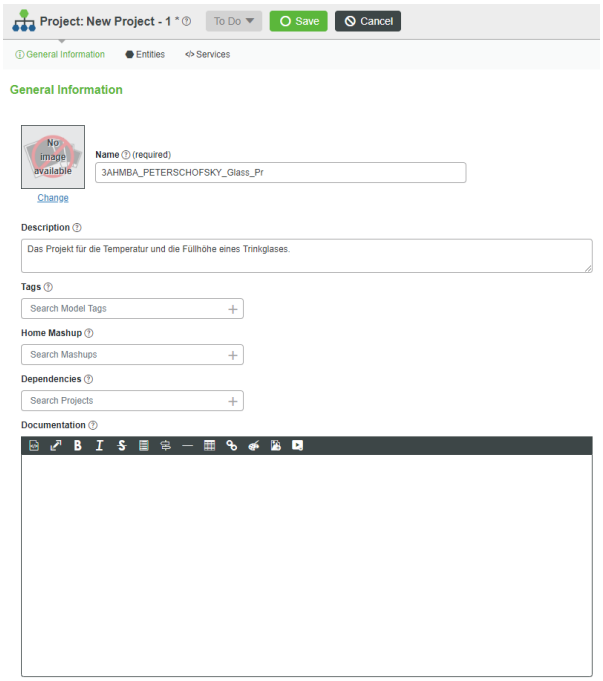
3.5.2 Anlegen unseres ersten Thingwork-Projekts

Video-Link 7: https://youtu.be/v1feQBBfoVc

3.5.2.1 Das leere Projekt

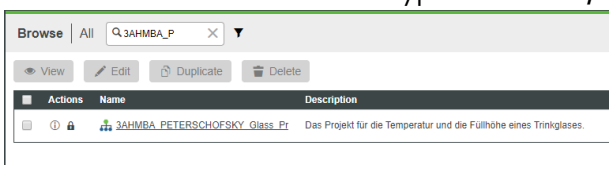
Da alles von einem Projekt zusammengehalten wird, legen wir dieses auch als Erstes an. Wir halten uns an die Namenkonvention. Ich werde hier z.B. `3AHMBA_PETERSCHOFSKY_Glass_Pr` verwenden. Zum Hinzufügen eines Projekts klickt man zunächst auf `+NEW` in der oberen Leiste. Aus dem folgenden Menü wählen wir „Project" aus.

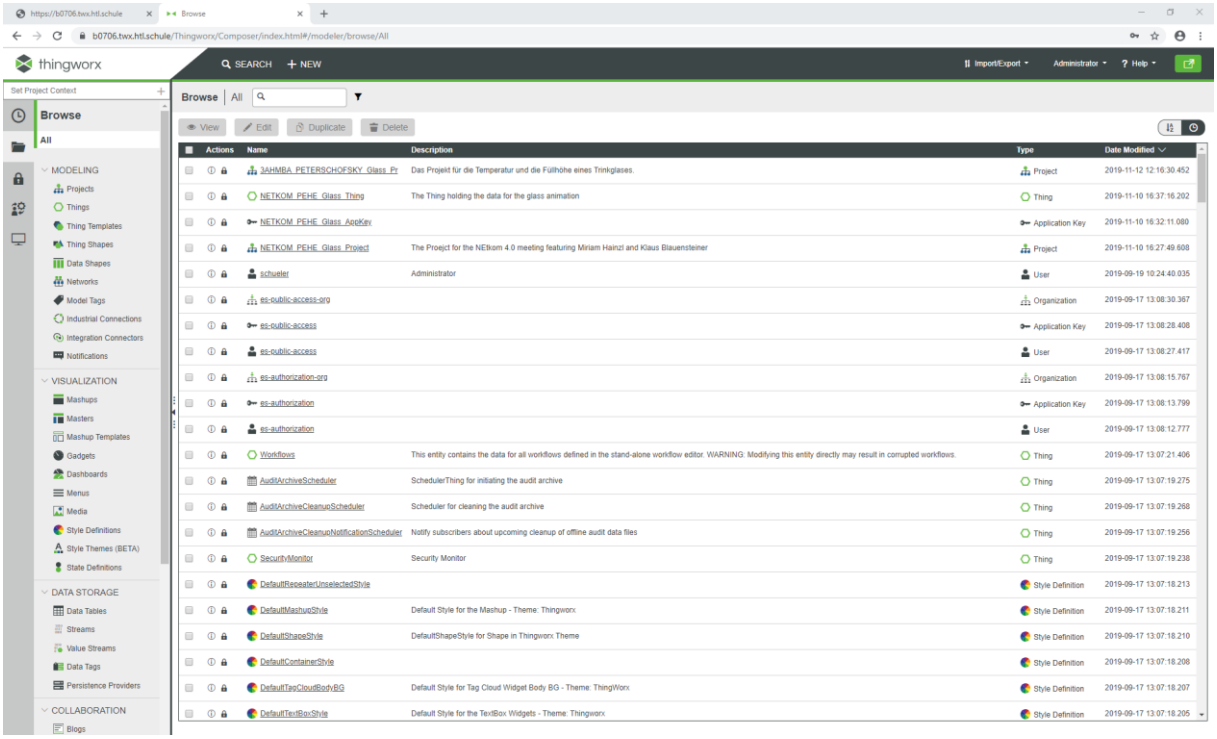




Im folgenden Dialogfeld fülle ich die Informationen aus. Wichtig ist eigentlich nur der Projektname. Der Vollständigkeit halber füllen wir auch noch die Description aus. Ein Klick auf „Save" und das Projekt ist erstellt.

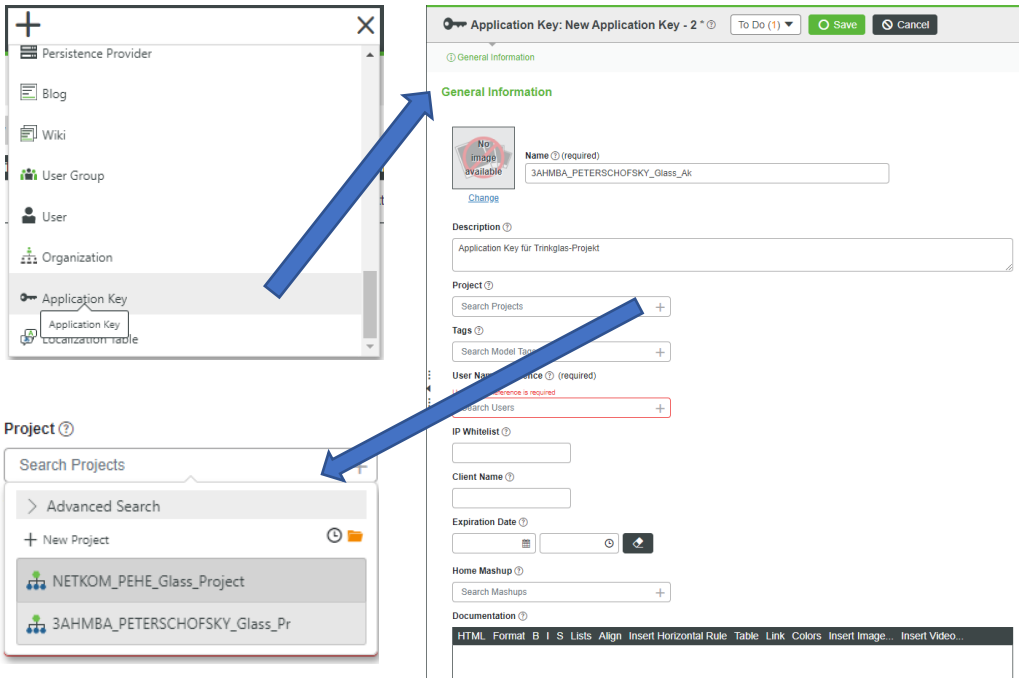
Dieses Projekt ist im Moment eine leere Hülle mit einem Namen und einer Beschreibung, die wir noch mit Leben füllen müssen. Dass das Projekt auch wirklich angelegt wurde erkennt man, wenn man ganz links auf das Ordner-Symbol klickt. Dadurch wird das „Inhaltsverzeichnis" des Thingworx-Servers angezeigt. Defaultmäßig sind alle möglichen Dinge dargestellt (Projekte, Things, Application Keys, Mashups, User, Usergruppen, Farbschemen, ...). Der neueste Eintrag ist immer oben – das bringt schon einmal ein bisschen Übersicht. Wenn man nur bestimmte Typen sehen will, so kann man auch den Filter verwenden. Oben in der Leiste kann man die Anfangsbuchstaben eingeben und es werden plötzlich nur mehr Dinge angezeigt, die mit diesen Buchstaben anfangen. Das bringt ordentlich Ordnung ins Chaos.





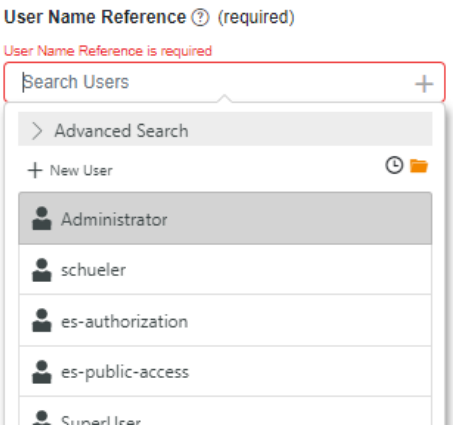
3.5.2.2 *Der zugehörige Application Key*

Eigentlich wollen wir ja unbedingt ein Thing anlegen, aber zuerst kümmern wir uns um die Zugriffsrechte – die kann man sonst leicht vergessen. Also ist das Nächste was wir machen einen Application Key anzulegen. Das geht wieder genauso wie beim Projekt nur dieses Mal wählen wir eben Application Key. Wieder halten wir uns an die Namenskonvention – das bedeutet bei mir also `3AHMBA_PETERSCHOFSKY_Glass_Ak`.

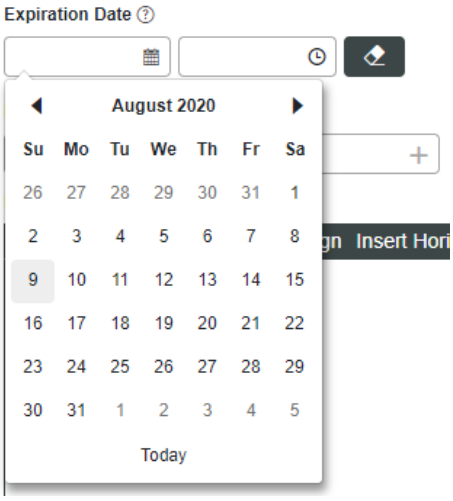


Bei der Projektzuordnung kann man auf das „+"-Symbol klicken, dann öffnet sich eine Liste mit allen auf diesem Thingworx-Server verfügbaren Projekten. Hier wählen wir natürlich das Projekt, welches

wir zuvor angelegt haben (ich hier also `3AHMBA_PETERSCHOFSKY_Glass_Pr`). Damit ist dieser Application Key dem Projekt zugeordnet. Jetzt müssen wir noch einen User zuordnen.



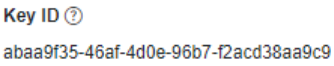
Im Prinzip legt der User fest, was der Application Key alles darf. Darf der zugeordnete User nichts, so kann man mit dem Application Key auch nichts machen. Darf der zugeordnete User alles, so kann man mit dem Application Key auch alles machen. Weil wir keinen anderen User haben verwenden wir einfach „Administrator" – das dürfte keine Probleme beim Zugriff geben! Wir bekommen jedoch eine Warnung von Thingworx, dass das keine gute Idee ist. Nun, das stimmt natürlich. Für unsere kleine Lernanwendung akzeptieren wir es trotzdem und schließen die Warnung mit „Yes".



Wichtig ist es noch ein Expiration Date anzugeben. Standardmäßig wird dieses mit einer Woche angesetzt – wir wollen jedoch, dass unser Application Key länger funktioniert, weswegen wir uns ein Datum in ferner Zukunft geben. Irgendwo in den Sommerferien müsste reichen[47].

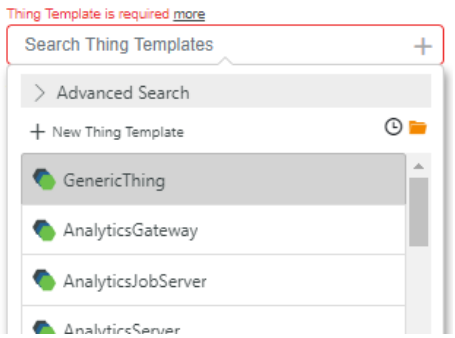
Es ist ein beliebter Fehler mit einem abgelaufenen Application Key zu arbeiten. Plötzlich bekommt man immer Not Authorized-Responses. Ganz nach dem Motto: Gestern hat es noch funktioniert, aber ich habe gar nichts gemacht.

Damit sind wir fertig und können „Save" drücken. Nach dem erfolgreichen Speichern gibt es plötzlich eine Key-ID. Das ist der Application Key. Wird dieser in den Anfragen mitgesendet so gibt man sich als „Wissender" zu erkennen und die Daten werden geschrieben.



3.5.2.3 Das Thing

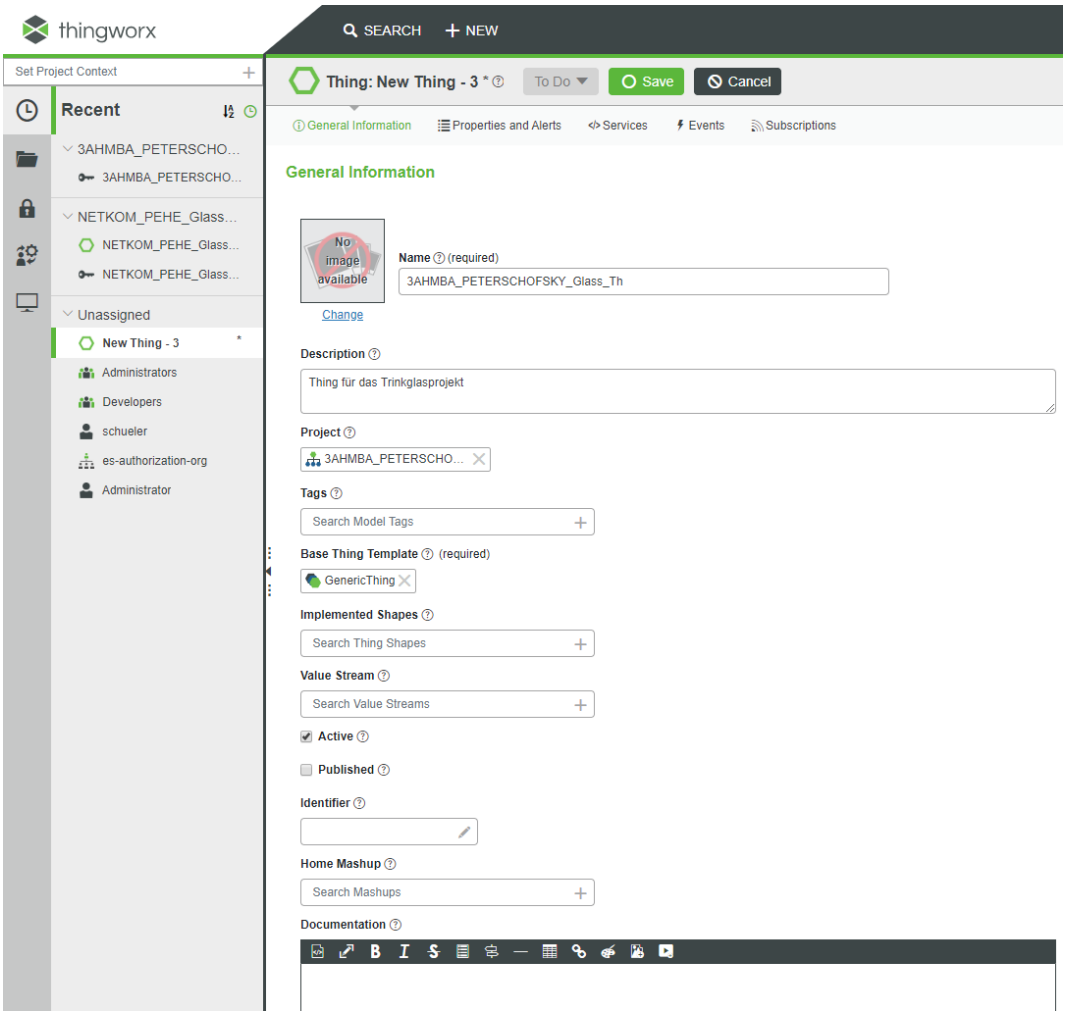
Jetzt endlich können wir unser Thing anlegen. Endlich kommen wir dahin, wo wir auch wirklich unsere Daten speichern. Nun, wir wollen ein Trinkglas machen. Und wir wollen uns die folgenden Dinge anlegen: Füllhöhe, Inhalt und Temperatur. Ein neues Thing anzulegen, startet wieder genauso wie vorhin. Nur dieses Mal wählen wir eben Thing aus.



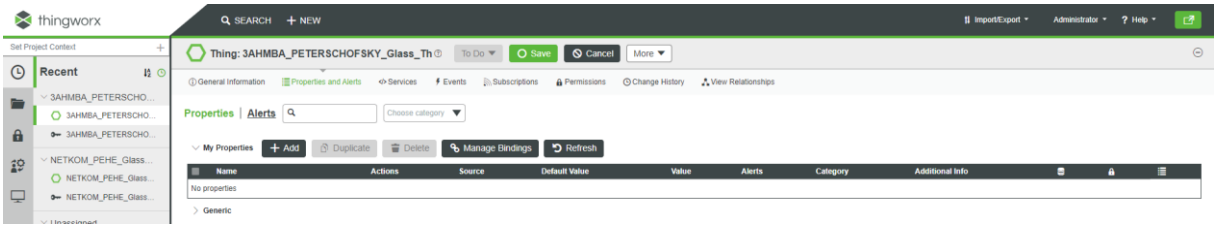
Wieder müssen wir einen Namen vergeben (hier `3AHMBA_PETERSCHOFSKY_Glass_Th`) der der Konvention entspricht. Als Projekt wählen wir wieder unser Projekt. Wir müssen noch ein „Template" wählen. Wir wählen aus der Liste „`Generic Thing`".

Damit sieht unsere Thingdefinition so ähnlich wie auf dem Bild aus und wir können auf „Save" drücken:

[47] Der Thingworx-Server wird ohnehin im neuen Schuljahr neu aufgesetzt. Länger bringt also nichts.

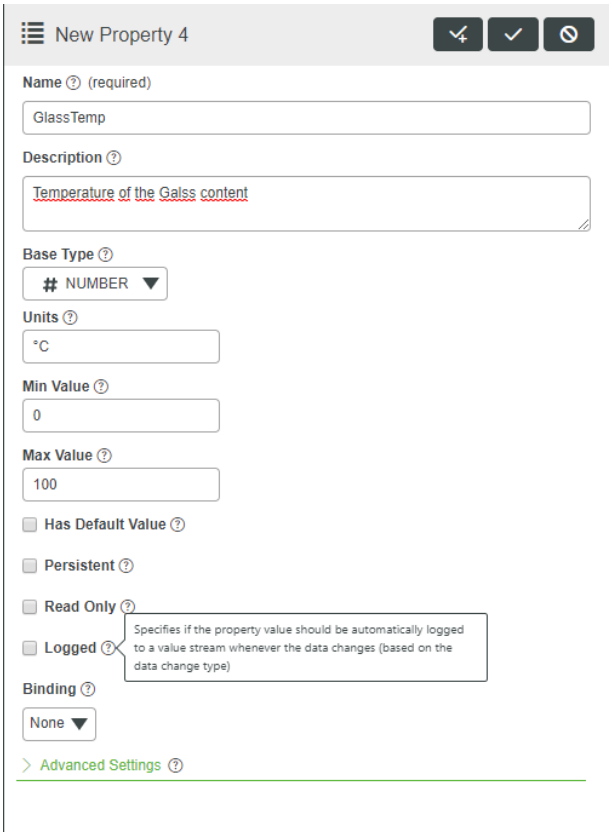


Das Thing erscheint links im Projektbaum, es hat einen Namen, ist dem richtigen Projekt zugeordnet, ansonsten aber leer. Wir wollen jetzt unsere Daten hinzufügen. Diese werden hier Properties (also Eigenschaften) genannt. Wir klicken oben auf „Properties and Alerts" und sehen eine leere Liste.

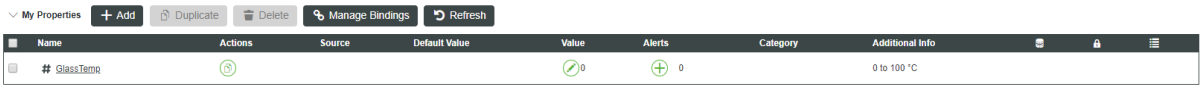


Um eine Eigenschaft hinzuzufügen drücken wir auf „`+Add`". Auf der rechten Seite wird ein Frame eingefügt. Dort kann man jetzt eine neue Eigenschaft anlegen. Wir kümmern uns zunächst um die Temperatur und geben deswegen bei Namen „`GlassTemp`" ein. Als Description wählen wir „`Temperature of the Glass content`" und als Base Type wählen wir „`Number`". Dann können wir Units mit „`°C`" Min Value mit „`0`" und Max Value mit „`100`" setzen. Weiters wählen wir „`Persistent`" und „`Logged`" aus.

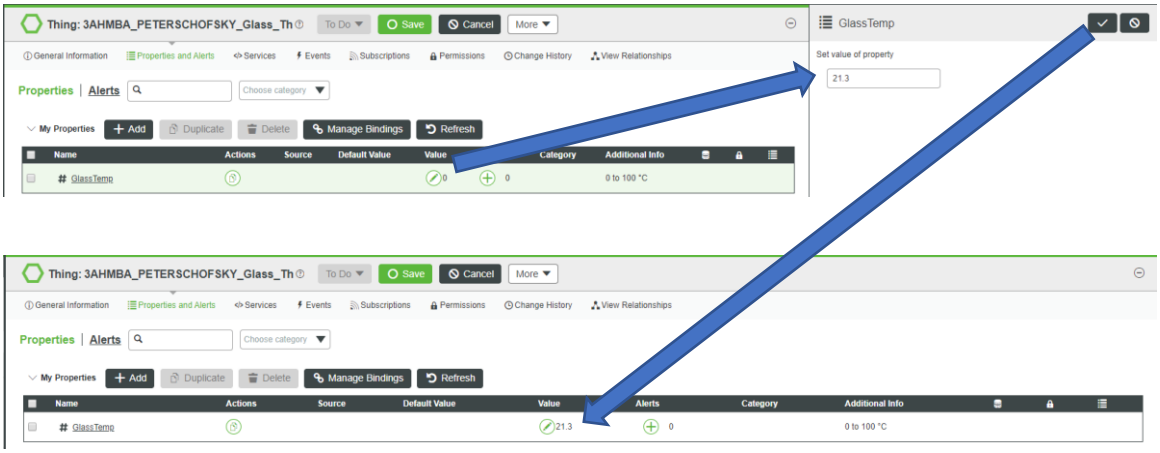
Unser neues Property sollte also in etwa so aussehen:



Mit einem Klick auf den Haken[48] oben in der Mitte speichern wir das Property. Es scheint in der Liste auf. Jetzt müssen wir noch einmal das Thing mit „Save" speichern. Jetzt hat die Eigenschaft auch einen Wert, nämlich 0.



Mit dem Bleistiftsymbol neben den Wert kann man diesen ändern. Hier ändern wir den Wert auf $21{,}3°C$. Achtung auf die Komma-Schreibweise – es ist ein „ . " Und nicht ein „ , "!



Funktioniert das? Dann ist es gelungen – wir haben unser erstes Thing erstellt. Mit einem zugehörigen Projekt und einer entsprechenden Berechtigung.

[48] Der Haken mit dem Plus würde gleich wieder das nächste Property hinzufügen.

3.5.2.4 *Übungsaufgabe „Simple"*

	Erstellen Sie so wie oben beschrieben Ihr Thing.	

3.5.2.5 *Pflichtaufgabe „Extedned Thing"*

	Fügen Sie noch die Eigenschaften für den Füllstand und das Füllvolumen hinzu. Überlegen Sie sich geeignete Grenzen und zeigen Sie Ihr Thing.	

3.5.3 Schreiben/Lesen von Daten in das Thing

Video-Link 8: https://youtu.be/sTKUigmElPY

Das Schreiben von Daten in unser Thing erfolgt mit Aufruf eines zugehörigen Service. Dieses Service heißt „`UpdatePropertyValues`"[49]. Es gibt verschiedenste Möglichkeiten diesen Service aufzurufen. Es hat sich gezeigt, dass die Methode einen speziellen http-Request zu senden relativ zuverlässig funktioniert. Dieser Request sieht wie folgt aus (Alle Werte zwischen <...> müssen angepasst werden:

```
PUT /Thingworx/Things/<Thingname>/Properties/* HTTP/1.1
Host: <server-IP>
Content-Type: application/json
appKey: <application key>

{
"<Proprty#1>":"<Value#1>"[,
"<Proprty#2>":"<Value#2>"[,
...]]
}
```

Angepasst auf das Beispiel aus dem vorigen Kapitel o würde der entsprechende Request also lauten:

```
PUT /Thingworx/Things/3AHMBA_PETERSCHOFSKY_Glass_Th/Properties/* HTTP/1.1
Host: 69b34.twx.htl.schule
Content-Type: application/json
appKey: abaa9f35-46af-4d0e-96b7-f2acd38aa9c9

{
"GlassTemp":"25.1"
}
```

Der obere Request würde das Property „`GlassTemp`" den Wert „`25.1`" geben. Die entsprechende Antwort des Servers würde lauten:

```
HTTP/1.1 200
Connection: keep-alive
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
```

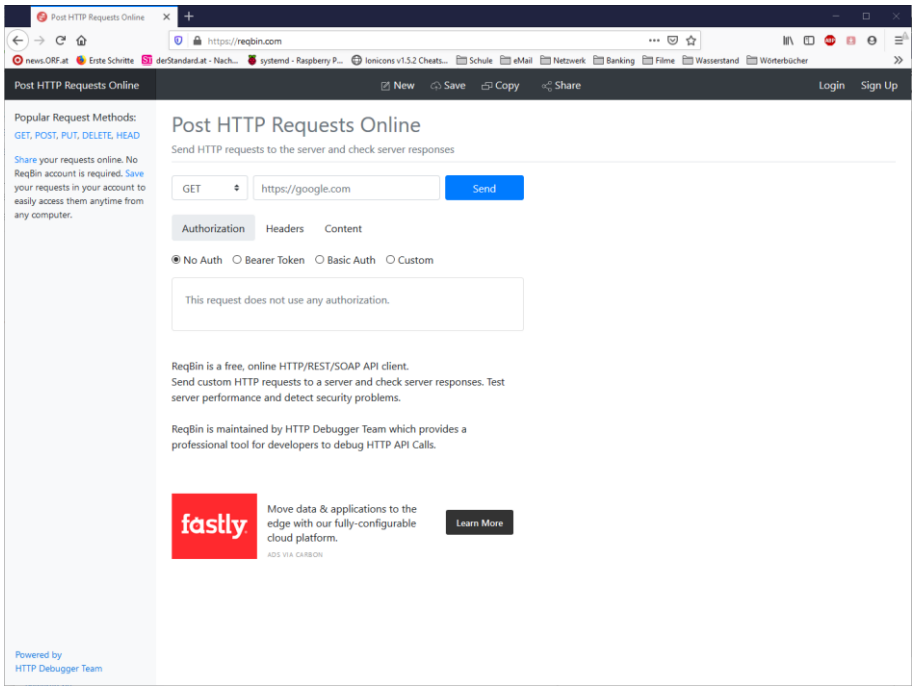
[49] Man kann sich auch seinen eigenen Service schreiben. Damit wäre es möglich nicht nur einfach Werte in die Properties zu schreiben, sondern z.B. diese auch umzurechnen (z.B.: Temperatur von °C in °F oder umgekehrt) oder mehrere Properties gleichzeitig aufzufüllen (z.B. hier die Füllhöhe und daraus das Füllvolumen berechnen und beide gleichzeitig schreiben). Das wollen wir in diesem Rahmen aber bleiben lassen.

```
Content-Security-Policy: frame-ancestors 'self'
X-Frame-Options: SAMEORIGIN
Cache-Control: no-store, no-cache,post-check=0, pre-check=0
Content-Type: text/html;charset=UTF-8
Date: Tue, 12 Nov 2019 15:00:59 GMT
Expires: 0
Server: nginx
Pragma: no-cache
Content-Encoding:
Transfer-Encoding: chunked
```

Schaut alles ganz wild aus, was aber wichtig ist, ist folgender Text: `HTTP/1.1 200`

Das bedeutet Response Code `200`, was auch `OK` heißt. So ein Code ist ein Sogenannter http-Statuscode. Eine Auflistung der Statuscodes befindet sich im Anhang. Relativ häufig werden wir noch mit dem Statuscode `401` kämpfen. Das bedeutet `Not Authorized`. Vermutlich gibt es keinen, oder einen abgelaufenen oder einen falschen Application Key.

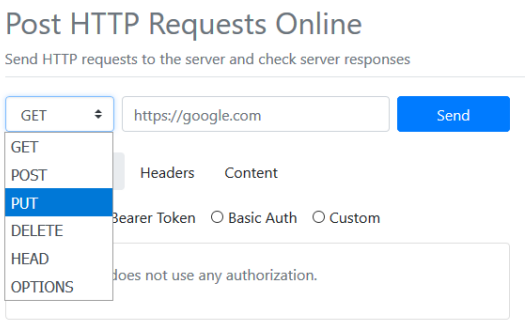
Jetzt stehen wir nur mehr vor der Herausforderung wie wir diesen http-Request wegsenden können. Nun, es gibt eine Menge Tools zum Versenden und Testen von http-Requests. Man kann verwenden was man will. Wenn man Datenbedenken hat, sollte man ein Programm verwenden, welches lokal am Rechner installiert ist[50]. Da es bei uns hier egal ist und wir nicht mit sensiblen Daten zu tun haben greifen wir auf ein Online-Tool- zurück. Ich verwende hier https://reqbin.com/.

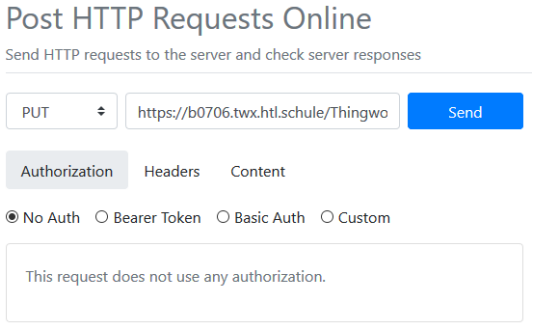


[50] Beispiele wären Postman oder Putty.

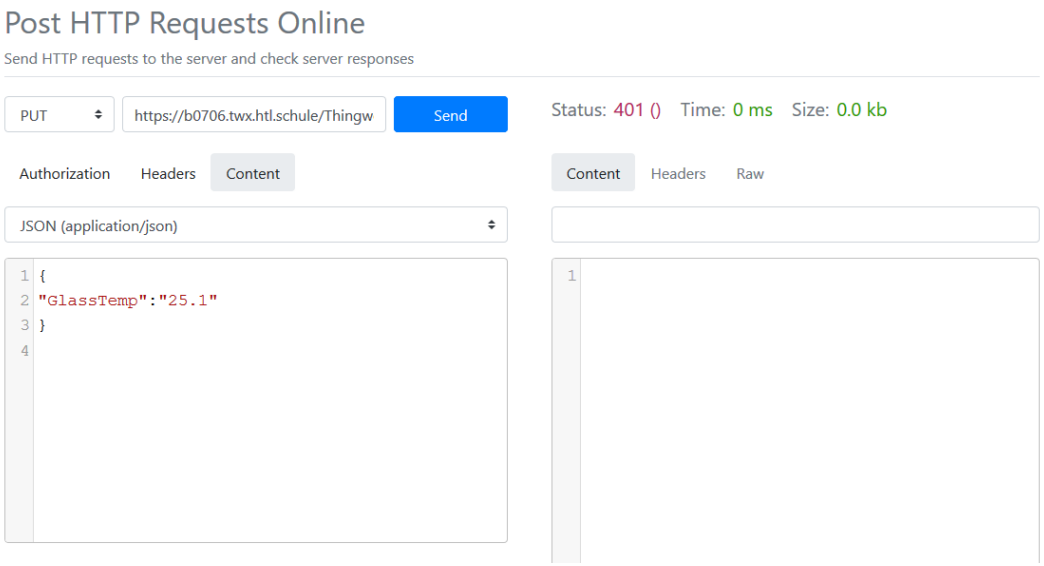
Hier können wir uns einfach einen entsprechenden http-Request zusammenbauen. Nun, zunächst benötigen wir einen PUT-Request, also stellen wir dies ein. Außerdem soll es ein `https`-Request an den Server `example.twx.htl.schule` an die URL `/Thingworx/Things/3AHMBA_PETERSCHOFSKY_Glass_Th/Properties/*` werden. Das können wir ganz einfach in das Eingabefeld daneben schreiben:

```
https://example.twx.htl.schule/Thingworx/Things/3AHMBA_PETERSCHOFSKY_Glass_Th/Properties/*
```





Der Inhalt soll ja ein neuer Wert für unsere Properties sein. Wir hatten ja das Property „`GlassTemp`" und dieses wollen wir mit einem neuen Wert versorgen. Sagen wir die Temperatur soll sich auf 25,1°C erhöht haben. Also schalten wir auf die Schaltfläche Content und geben dort die entsprechenden Informationen ein (der richtige Typ `JSON` sollte bereits eingestellt sein):



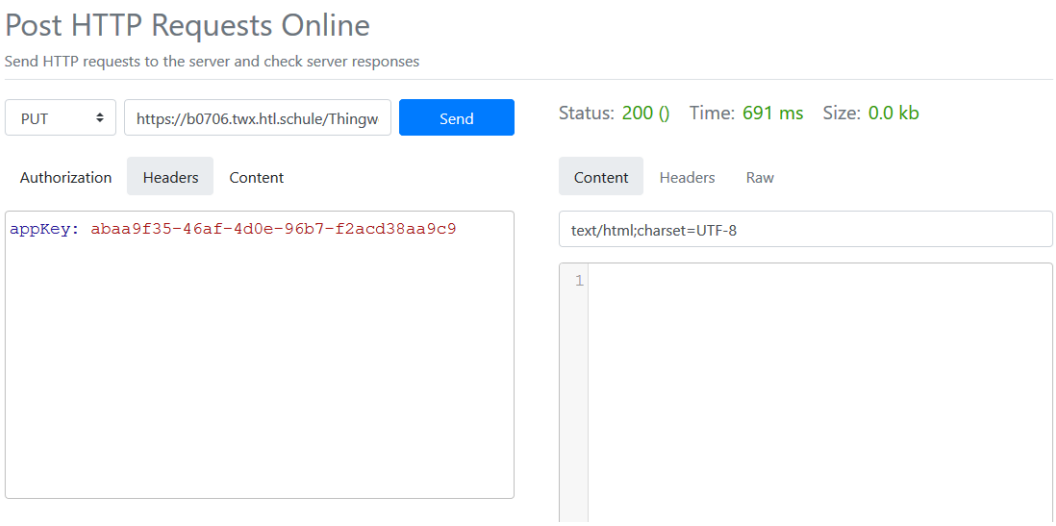
Wenn wir den Request so wegsenden sollten wir eine Antwort vom Sever erhalten, welche jedoch den Code `401` enthält. Die gute Nachricht: Der Server ist erreichbar und redet mit uns. Die schlechte Nachricht: Er lässt uns nichts ändern.

Das ist auch kein Wunder, denn immerhin haben wir Aufwand betrieben einen Application Key anzulegen und den haben wir noch gar nicht in unserem http-Request verwendet. Wir werden diesen Key also hinzufügen. Dazu wechseln wir auf den Bereich „Headers" und fügen die Zeile

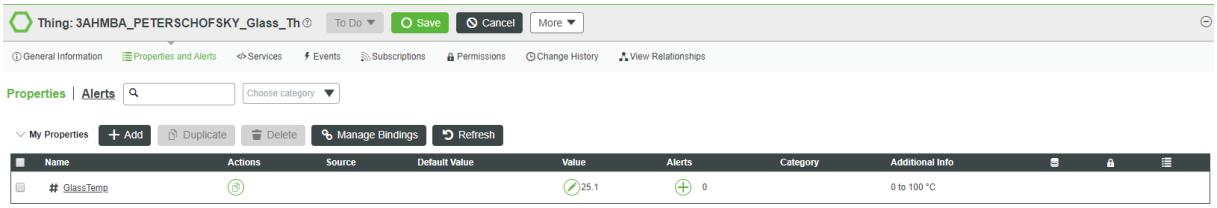
```
appKey: abaa9f35-46af-4d0e-96b7-f2acd38aa9c9
```

hinzu. ACHTUNG: Jeder hat seinen eignen App-Key! Deswegen die RICHTIGE Nummer verwenden!

Senden wir dies nun weg sollten wir tatsächlich einen `200`-Status-Code erhalten. Der Server hat unsere Anfrage akzeptiert!



Ein Blick in die entsprechende Property in Thingworx macht uns sicher: Der Wert hat sich tatsächlich auf die angegeben $25{,}1°C$ geändert (eventuell muss auf Refresh geklickt werden). Es funktioniert also!



Von welchem Gerät dieser http-Request kommt ist vollkommen egal! Wenn wir es also schaffen ein Gerät zu bauen, welches in der Lage ist entsprechende http-Requests zu senden bringen wir die Daten vom Gerät in unseren Thingworx-Server. Wir haben dann ein Messgerät, welches seine Werte in die Cloud überträgt und zur Verfügung stellt. Es muss nur den richtigen Application Key verwenden und die Sache läuft. Überträgt das Gerät zyklisch, so ist immer der aktuelle Messwert online verfügbar und kann irgendwo auf dieser Welt verwendet werden.

Wie holen wir nun die Daten wieder aus der Datenbank ab? Nun, auch das geht mit einem http-Request. Dieser sollte folgende Form haben:

```
GET /Thingworx/Things/<Thingname>/Properties/ HTTP/1.1
Host: <server-IP>
appKey: 1f3f94f9-0c53-433c-9e61-cf03beebeb15
```

Sehr einfach, oder? Versuche wir den Request auf unseren Fall anzupassen:

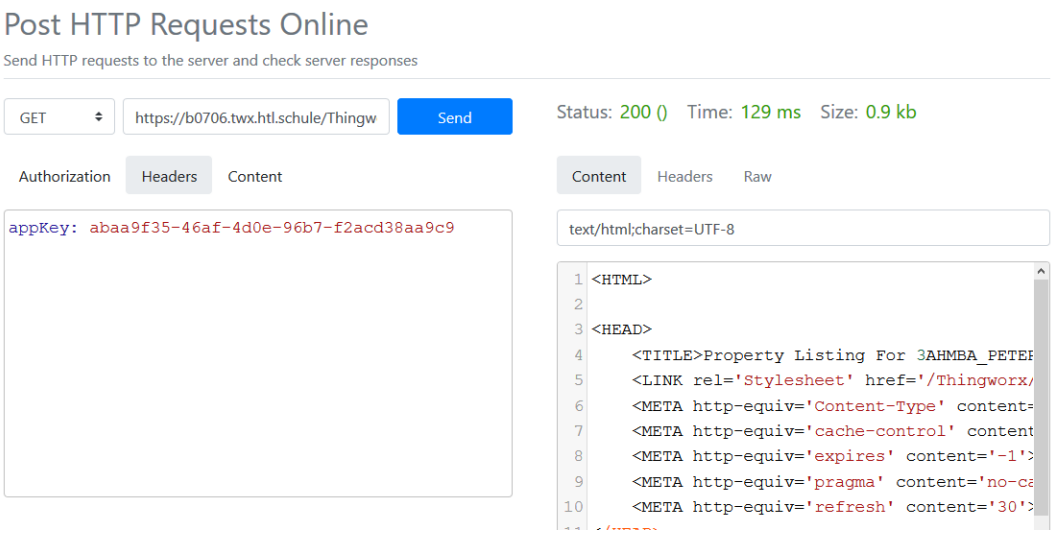
```
GET /Thingworx/Things/3AHMBA_PETERSCHOFSKY_Glass_Th/Properties/ HTTP/1.1
Host: example.twx.htl.schule
appKey: abaa9f35-46af-4d0e-96b7-f2acd38aa9c9
```

Um diesen Request auszuführen verwenden wir wieder https://reqbin.com/.

Also stellen wir GET-ein und verwenden folgende Zeile:

```
https://example.twx.htl.schule/Thingworx/Things/3AHMBA_PETERSCHOFSKY_Glass_Th/Properties/
```

In den Header müssen wir wieder den appKey angeben. Der Content kann leer bleiben. Sendet man diesen Request so erhalten wir eine positive Antwort, nur dieses Mal stet auch etwas im Content:



Bei mir ist diese Antwort:

```
<HTML>
<HEAD>
    <TITLE>Property Listing For 3AHMBA_PETERSCHOFSKY_Glass_Th</TITLE>
    <LINK rel='Stylesheet' href='/Thingworx/css/thingworxapi.css' type='text/css'></LINK>
    <META http-equiv='Content-Type' content='text/html'></META>
    <META http-equiv='cache-control' content='no-cache, no-store'></META>
    <META http-equiv='expires' content='-1'></META>
    <META http-equiv='pragma' content='no-cache, no-store'></META>
    <META http-equiv='refresh' content='30'></META>
</HEAD>
<BODY>
    <IMG SRC="/Thingworx/images/ThingworxLogo.png" />
    <BR/>
    <H1>Property Listing For 3AHMBA_PETERSCHOFSKY_Glass_Th</H1>
    <TABLE>
        <TR>
            <TH>name</TH>
            <TH>value</TH>
        </TR>
        <TR>
            <TD>description</TD>
            <TD>Thing f&uuml;r das Trinkglasprojekt</TD>
        </TR>
        <TR>
            <TD>GlassTemp</TD>
            <TD>25.1</TD>
        </TR>
        <TR>
            <TD>name</TD>
            <TD>3AHMBA_PETERSCHOFSKY_Glass_Th</TD>
        </TR>
        <TR>
            <TD>tags</TD>
            <TD></TD>
        </TR>
        <TR>
            <TD>thingTemplate</TD>
            <TD>GenericThing</TD>
        </TR>
    </TABLE>
</BODY>
</HTML>
```

Versteht das einer? Nun, zumindest kommt unser Property „`GlassTemp`“ mit dem aktuellen Wert `25.1` darin vor – also es schienen tatsächlich irgendwie die Daten enthalten zu sein. Das ganze

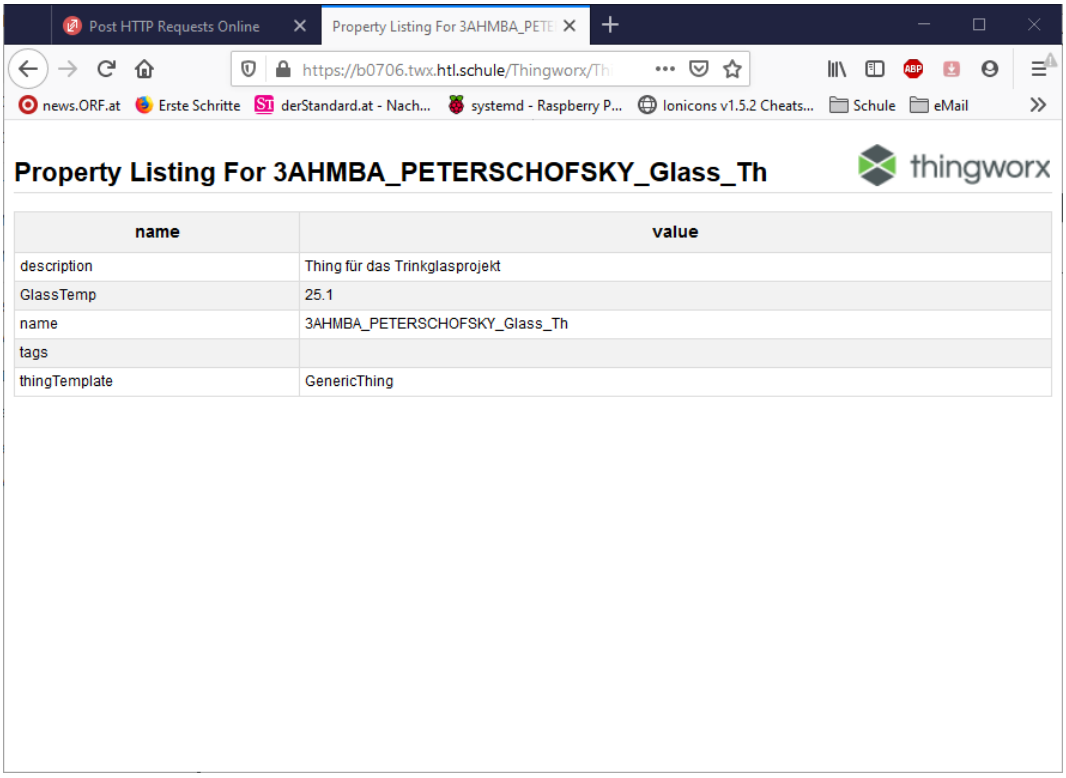
drum herum ist nicht nur unnötiges Beiwerk, das ist die Beschreibungssprache HTML[51]. Diese Sprache wird von allen Webbrowsern verstanden – das ist eine Webpage in Textfrom!

Was liegt also näher als die Adresse

```
https://example.twx.htl.schule/Thingworx/Things/3AHMBA_PETERSCHOFSKY_Glass_Th/Properties/
```

gleich in den Webbrowser einzugeben?

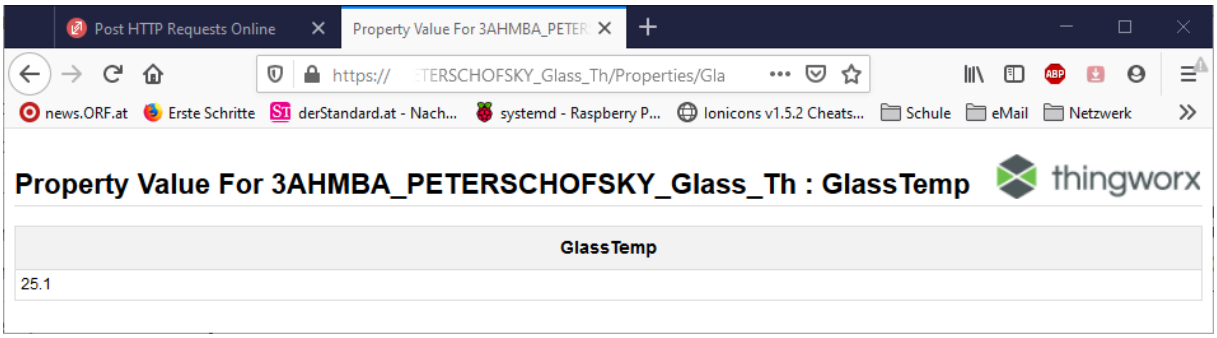
Machen wir das, so fordert uns der Browser auf die Zugangsdaten einzugeben (wir haben ja wieder keinen appKey vom Browser aus). Geben wir diese ein so erschient eine Webseite, die die Information wesentlich übersichtlicher darstellt.



Sind mehrere Properties vorhanden, so werden diese alle aufgelistet. Will man dezidiert auf genau ein Property zugreifen kann man auch z.B. die Zeile

```
https://example.twx.htl.schule/[...]/Properties/GlassTemp
```

verwenden.



[51] Hyper Text Markup Language

3.5.3.1 *Übungsaufgabe „Change"*

	Erstellen Sie so wie oben beschrieben einen http-Request für die Änderung der Daten in Ihrem Thing.	

3.5.3.2 *Pflichtaufgabe „Change Data"*

	Erweitern Sie den Request für die Eigenschaften des Füllstandsund des Füllvolumens!	

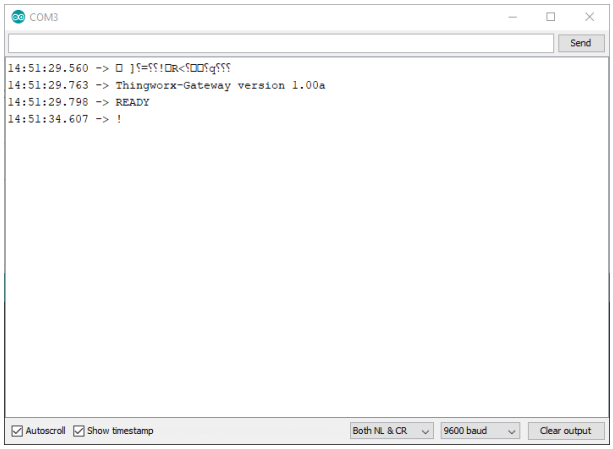
3.5.4 Verbindung zwischen Arduino und Thingworx

Video-Link 9: https://youtu.be/cflsgaegzvw

Prinzipiell haben wir hier das gleiche Problem wie wir schon bei der Verbindung mit MQTT. Das wurde ja in Kapitel o behandelt. Nun, hier gibt es praktisch das gleiche „Rezept": Ein Programm auf unserm ESP8266-Board und wir sprechen mit diesem per Serial Interface.

Dieses Mal verwenden wir das Programm `ThingWorker.ino` und spielen dieses auf unseren ESP. Das Vorgehen ist ident mit dem in Kapitel 3.4.1, nur eben mit dem anderen Programm.

Die Befehle sind wieder relativ ähnlich. Auch hier können wir nach dem Reset die Verbindung mit der Eingabe eines „?" überprüfen und auch hier bekommen wir „!" zurück – es geht!



Wir wollen uns jetzt wieder „händisch" zu unserem Thingworx-Server verbinden. Dazu schalten wir uns zunächst alle möglichen Ausgaben vom ESP ein. Wir tippen in den Serial Monitor nacheinander die Befehle

```
startwarn
startinfo
startdebug
```

ein.

Wenn wir jetzt noch `getverbose` verwenden bekommen wir zur Bestätigung die Ausgabe:

```
ERROR WARN. INFO. DEBUG (240)
  X     X     X     X
```

Die „X" kennzeichnen, dass die Ausgaben für Fehler, Warnungen, Informationen und Debug ausgegeben werden. Jeder Befehl wir jetzt mit einer Menge Rückmeldung quittiert.

Versuchen wir uns zunächst mit dem WLAN der Schule zu verbinden. Dazu verwenden wir die Befehle:

```
setssid WifISSID
```

Rückmeldung:

```
Set WifiSSID: 'WifISSID'
OK
```

```
setpass WifiPassword
```

Rückmeldung:

```
Set WifiPass: 'WifiPassword'
OK
```

Sobald eine SSID und ein Passwort übermittelt wurden versucht der ESP eine Verbindung mit dem angegeben Netzwerk aufzubauen. Wenn nicht gleich eine Verbindung zustande kommt, versucht es der Controller immer wieder. Ist die SSID oder das Passwort falsch reicht es den Wert mit dem entsprechenden Befehl zu korrigieren. Die Ausgabe sollte ähnlich wie hier aussehen:

```
Try to connect to WiFi.
Connecting to: WifISSID
.....................Could not connect.
ERR
Try to connect to WiFi.
Connecting to: WifISSID
...... Connected!
OK
```

Ist die Verbindung erfolgreich sollte die LED WLAN ausgegangen sein. Dies hat energiespartechnische Gründe. Beim Betrieb mit z.B. eine Batterie ist es unnötig, wenn hier im Normalzustand irgendwelche LEDs leuchten.

Der erste Schritt ist also wieder einmal geschafft – wir sind mit dem WLAN verbunden. Jetzt geht es weiter mit dem Thingworx-Server. Wir müssen die host-Adresse senden, und danach den Application Key und das Thing. Also fangen wir an und geben nacheinander ein:

```
sethost example.twx.htl.schule
```

Rückmeldung:
```
Set Host: 'example.twx.htl.schule'
OK
```

```
setappkey abaa9f35-46af-4d0e-96b7-f2acd38aa9c9
```

Rückmeldung:
```
Set App Key: 'abaa9f35-46af-4d0e-96b7-f2acd38aa9c9'
OK
```

```
setthing 3AHMBA_PETERSCHOFSKY_Glass_Th
```

Rückmeldung:
```
Set Thing: '3AHMBA_PETERSCHOFSKY_Glass_Th'
OK
```

Damit haben wir also Server, Application Key und Thing festgelegt. Kommen wir zu den Properties.

Dort gibt es ganz genau so wie bei der MQTT-Variante wieder die drei Varianten:

- Value-Property: Numerischer Wert, hier kann auch ein Übertragungsschwelle angegeben werden.
- Boole'sches Property: Wahrheitswert
- Allgemeines Property: Kann jeden Wert beinhalten.

Ich will gleich einmal unsere Glastemperatur als numerischen Wert hinzufügen und gebe deswegen ein:

```
Setvaluename 0 GlassTemp
```

Rückmeldung:
```
Set Value Name #0 : 'GlassTemp'
OK
```

Danach noch die Übertragungsschwelle:

```
setvalth 0 0.25
```

Rückmeldung:
```
Set Value Threshold #0 : '0.25'
OK
```

Damit haben wir festgelegt, dass wir in unser Thing `3AHMBA_PETERSCHOFSKY_Glass_Th` mit dem Property `GlassTemp` einen Wert hinschrieben wollen. Noch tut sich nichts, erst wenn wir mit

```
setval 0 25.3
```

einen Wert setzen beginnen plötzlich mehr Ausgaben. Die Rückmeldung ist klar:

```
Set Value #0 : '25.30'
OK
```

Danach folgen in regelmäßigen Abständen die Ausgabe:

```
{
"GlassTemp":"25.300"
}
reply was:
==========
Response: 200
headers received:
HTTP/1.1 200
Server: nginx
Date: Sun, 02 Feb 2020 14:26:31 GMT
Content-Type: text/html;charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Content-Security-Policy: frame-ancestors 'self'
X-Frame-Options: SAMEORIGIN
Expires: 0
Cache-Control: no-store, no-cache
Cache-Control: post-check=0, pre-check=0
Pragma: no-cache



==========
closing connection
```

Zuerst sehen wir den zum Server übertragenen JSON[52]-Block und danach die Antwort des Servers. Hier sehen wir den Response-Code: `200`. Das ist gut, der Server hat unseren Wert verstanden. Dieses Mal müssen wir keinen http-Request zusammenbauen. Den macht der ESP für uns in der geeigneten Form. Wir müssen nur sagen was wir übertragen wollen.

Immer wenn die Übertragungszeit abgelaufen ist, so werden einfach ALLE eingeben Properties übertragen. Genau wie beim MQTT-Teil können wir Zeiten für die Weiderholung eingeben. Alle Befehle des ESP-Gateways für Thingworx sind im Anhang verfügbar (sieh Kapitel 4.5.2).

3.5.4.1 Übungsaufgabe „Change Data ESP"

	Erstellen Sie so wie oben beschrieben das Programm für das ESP-Gateway für die Änderung der Daten in Ihrem Thing.	

3.5.4.2 Pflichtaufgabe „Change all Data ESP"

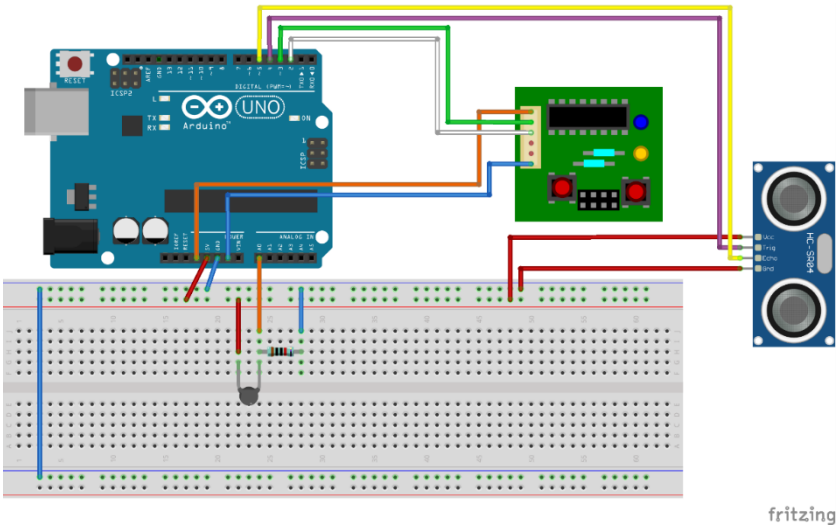
	Erweitern Sie das Programm für die erweiterten Eigenschaften des Füllstands und des Füllvolumens!	

[52] JSON: JavaScript Object Notation, ein Datenformat, das die Informationen in einer speziellen Form enthält.

3.5.5 Versorgen des Things mit Live-Daten

Video-Link 10: https://youtu.be/cXgQxogFqvQ

Wie schon beim MQTT-Thema ist es ja ganz nett, wenn wir uns mit Kommandozeilenparameter die Werte ändern können, wenn diese aber aus realen Messungen stammen, so wirkt das Ganze noch einmal um einiges besser. Wir wollen also einen Pegel und eine Temperatur messen und die Werte dann zum Thingworx-Server übertragen. Nun, es gibt für unseren Arduino wieder eine Bibliothek. Diese nennt sich analog der MQTT-Bibliothek „`EspThingworxGateway`". Wir verwenden also wieder eine zip-Datei um die Bibliothek zu installieren. Auch diese benötigt die Bibliothek `Timeout.h`[53].



Mit den installierten Bibliotheken und folgendem Hardwareaufbau können wir uns daran machen die Dinge zu programmieren.

Wir verwenden also den NTC um die Temperatur zu messen und das Ultraschallmodul um einen Abstand festzustellen. Entsprechend sieht das Programm aus:

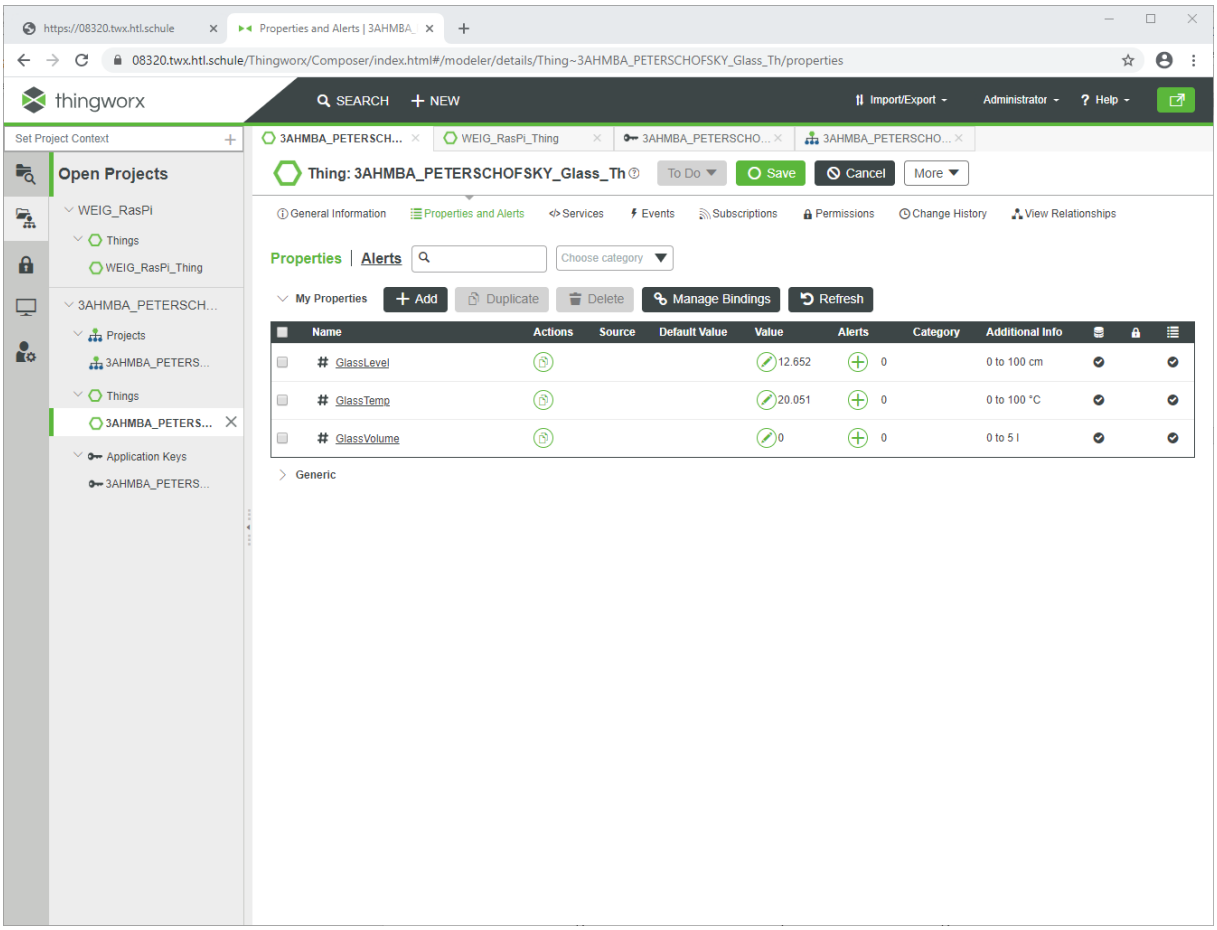
```
#include "SoftwareSerial.h"
#include "EspThingworxGateway.h"
#include "Timeout.h"

#define TRIGGER_PIN    4
#define ECHO_PIN       5
#define SPEED_OF_SOUND  343.0
#define MAX_DIST         25.0

String SSID = "WifISSID";
String PASS = "WifiPassword";
String TwxHost = "69b34.twx.htl.schule";
String AppKey = "abaa9f35-46af-4d0e-96b7-f2acd38aa9c9";
String Thing = "3AHMBA_PETERSCHOFSKY_Glass_Th";

SoftwareSerial espPort(2, 3);
EspThingworxGateway *twxClient;
Timeout *changeData = new Timeout(500);

unsigned long get_duration()
{
  digitalWrite(TRIGGER_PIN, LOW);  delayMicroseconds(2);
  digitalWrite(TRIGGER_PIN, HIGH); delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);  delayMicroseconds(2);
  return pulseIn(ECHO_PIN, HIGH, 10000L);
}
```

[53] Warum und wieso steht auch im Kapitel o beschrieben.

```
void setup() {
        Serial.begin(115200);
        Serial.println("Schetter Thingworx-Tester is starting up ...");
        pinMode(TRIGGER_PIN, OUTPUT);
        pinMode(ECHO_PIN, INPUT);
        twxClient = new EspThingworxGateway(&espPort, &Serial);
        twxClient->SetVerbose(16+32+64+128);
}

void loop() {
  unsigned long duration = get_duration();
  float distance;
  int readVal = analogRead(0);
  float temp = log(10000.0 * ((1024.0 / readVal - 1)));

  temp = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * temp * temp)) * temp)
        - 276.15;

        if (!twxClient->WifiSsidTransfered()) twxClient->SetWifiSsid(SSID);
        if (!twxClient->WifiPassTransfered()) twxClient->SetWifiPass(PASS);
        if (!twxClient->ThingworxHostTransfered()) twxClient->SetThingworxHost(TwxHost);
        if (!twxClient->AppkeyTransfered()) twxClient->SetAppkey(AppKey);
        if (!twxClient->ThingTransfered()) {
                twxClient->SetThing(Thing);
                twxClient->SetValueName(0, "GlassLevel");
                twxClient->SetValueThreshold(0, 0.5, 2);
                twxClient->SetValueName(1, "GlassTemp");
                twxClient->SetValueThreshold(2, 0.2, 2);
        }

   if (duration != 0) distance = MAX_DIST - duration * SPEED_OF_SOUND / 20000;

        if (changeData->TimedOut()) {
                if (twxClient->ThingworxConnected()) {
                  twxClient->SetValue(0, distance, 3);
         twxClient->SetValue(1, temp, 3);
                }
                changeData->SetNow();
        }
        twxClient->Update();
}
```

Die meisten Dinge kennen wir ja schon. Es ist praktisch eine Mischung aus den Kapiteln 1.2.16 Der NTC als Temperatursensor und 1.2.18 Abstandssensor. Insbesondere der Abstandssensor ist etwas erweitert, weil wir nicht die Dauer, sondern wirklich die Distanz berechnen. Außerdem nehmen wir an wir messen von oben in das Gefäß. Je kleiner der Abstand wird, desto voller ist also unser Gefäß. Angereichert ist das Ganze mit der Übertragung auf den Thingworx-Server. Im Prinzip sollte das ganze selbsterklärend sein. Es funktioniert ganz ähnlich wie in Kapitel 3.4.2 – sogar noch ein wenig einfacher, weil wir ja keinen Rückkanal haben und von der Thingworx-Plattform keine Daten empfangen. Eine Auflistung aller Methoden für die Klasse `EspThingworxGateway` findet sich im Anhang.

Wenn wir unser Thing im Thingworx-Composer öffnen, erkennen wir, dass wir immer neue, frische Werte zur Verfügung haben. Die Messung läuft also und überträgt die Werte in die Properties des gewählten Things.



3.5.5.1 *Übungsaufgabe „Automatic Data Transfer"*

	Erstellen Sie so wie oben beschrieben das Programm mit den Messungen.	

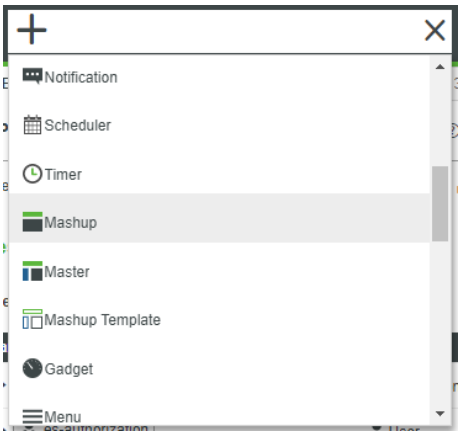
3.5.5.2 *Pfilchtaufgabe „Automatic Volume Transfer"*

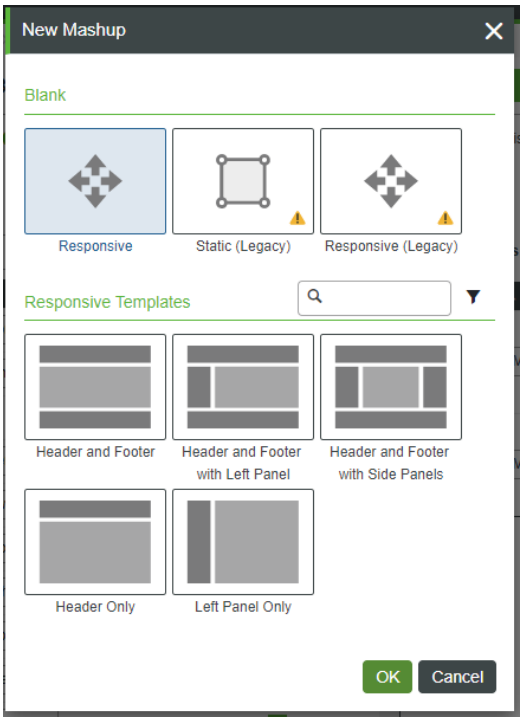
	Erweitern Sie das Programm um die Eigenschaft des Füllvolumens. Überlegen Sie wie sie dieses berechnen können und übertragen Sie den Wert sinnvoll.	

3.5.6 Anzeigen von Daten in einem Mashup

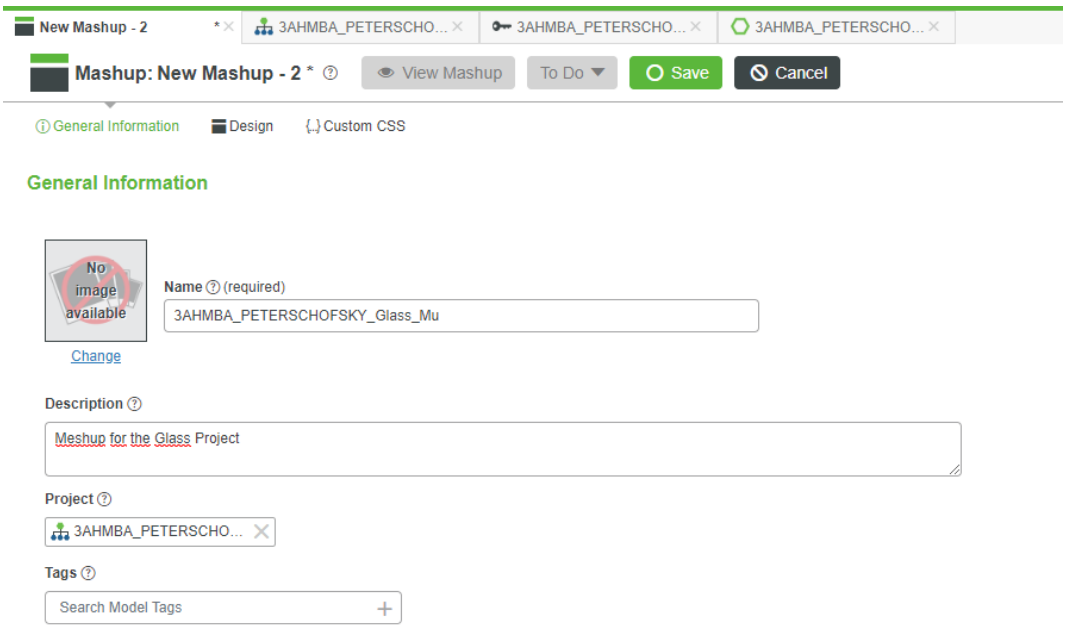
Video-Link 11: https://youtu.be/DZINXvExyhk

Immer wenn wir uns die Daten des Things ansehen wollen, dann müssen wir entweder die Seite des Things im Thingworx-Composer öffnen, oder die entsprechende URL eingeben. Eine gesammelte übersichtliche Darstellung wäre hilfreich. Und genau so etwas gibt es auch. Dies wird in Thinworx „Mashup" genannt. Wie üblich können wir solch ein Mashup zu unserem Projekt hinzufügen. Wieder versuchen wir diese mit dem +-Symbol in der oberen Leiste und dieses Mal wählen wir „Mashup" aus.

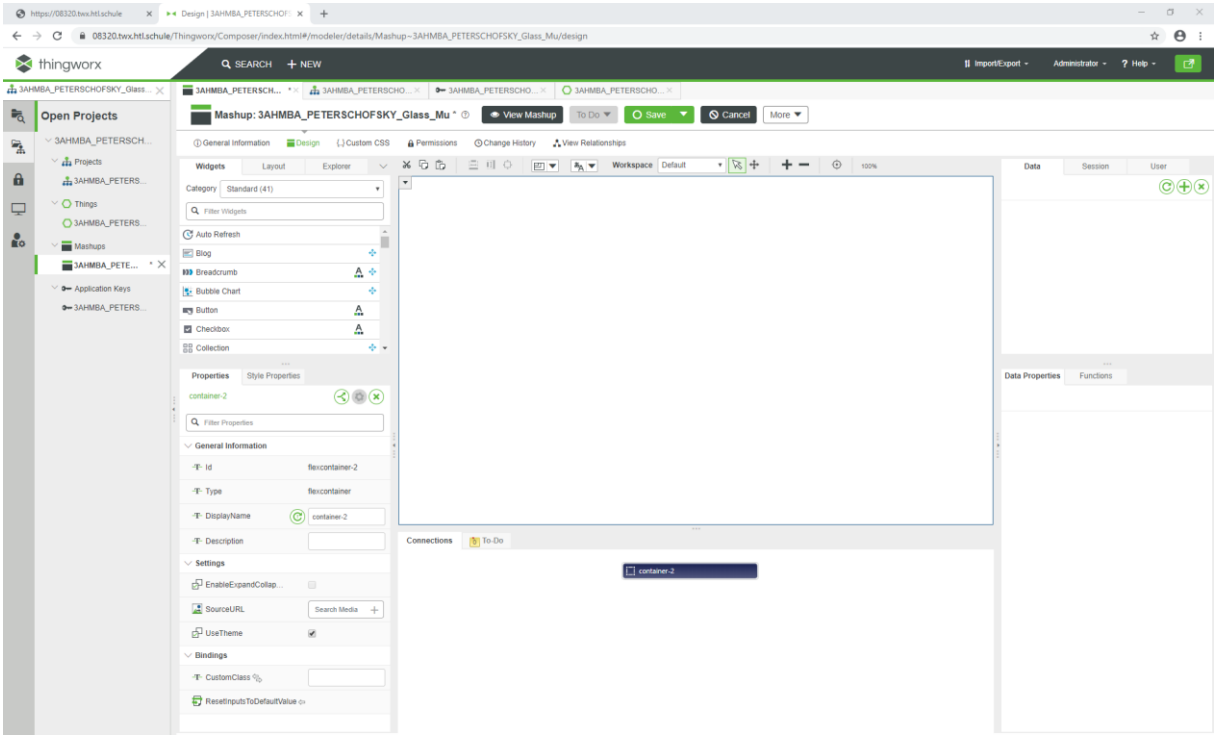




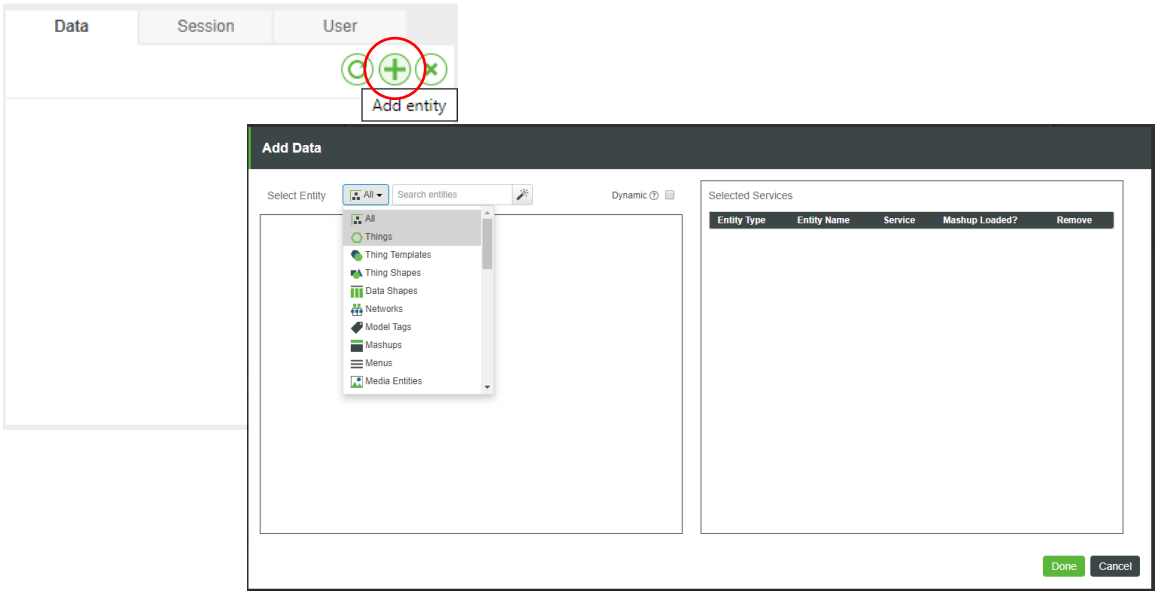
Aus dem erscheinenden Menü wählen wir „Responsive" aus und drücken OK. Wir werden die geeigneten Daten entsprechend der Namenskonvention eintragen. Ich wähle also für den Namen des Mashups `3AHMBA_PETERSCHOFSKY_Glass_Mu`.



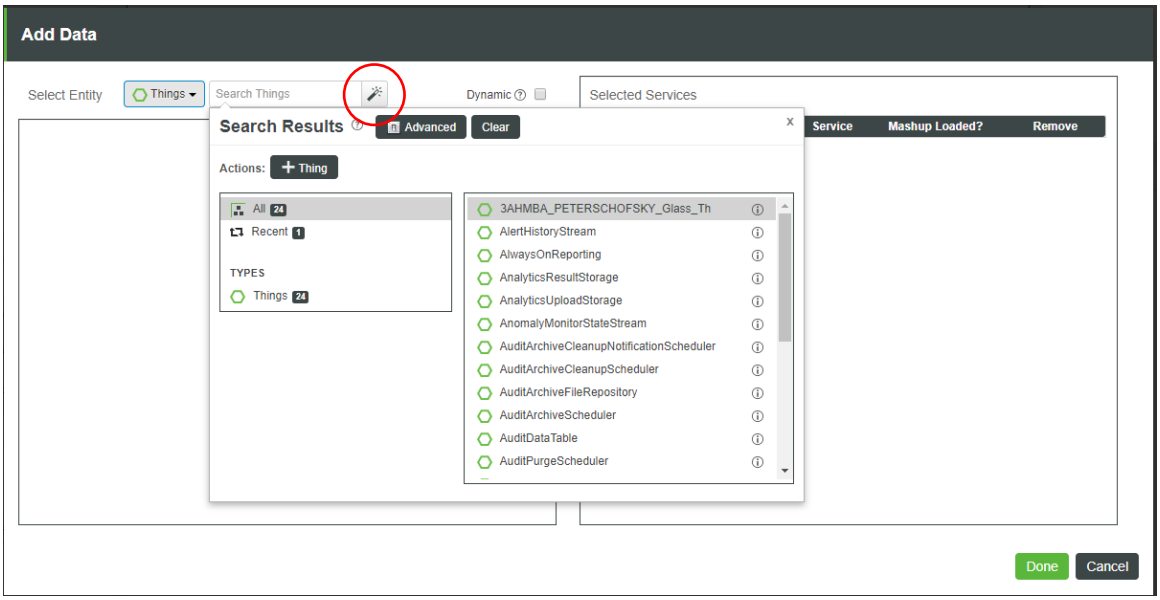
Wenn wir danach „Save" klicken, ist das Mashup angelegt. Wenn wir danach auf „Design" umschalten, so erhalten wir die Oberfläche, um ein Mashup anzulegen.



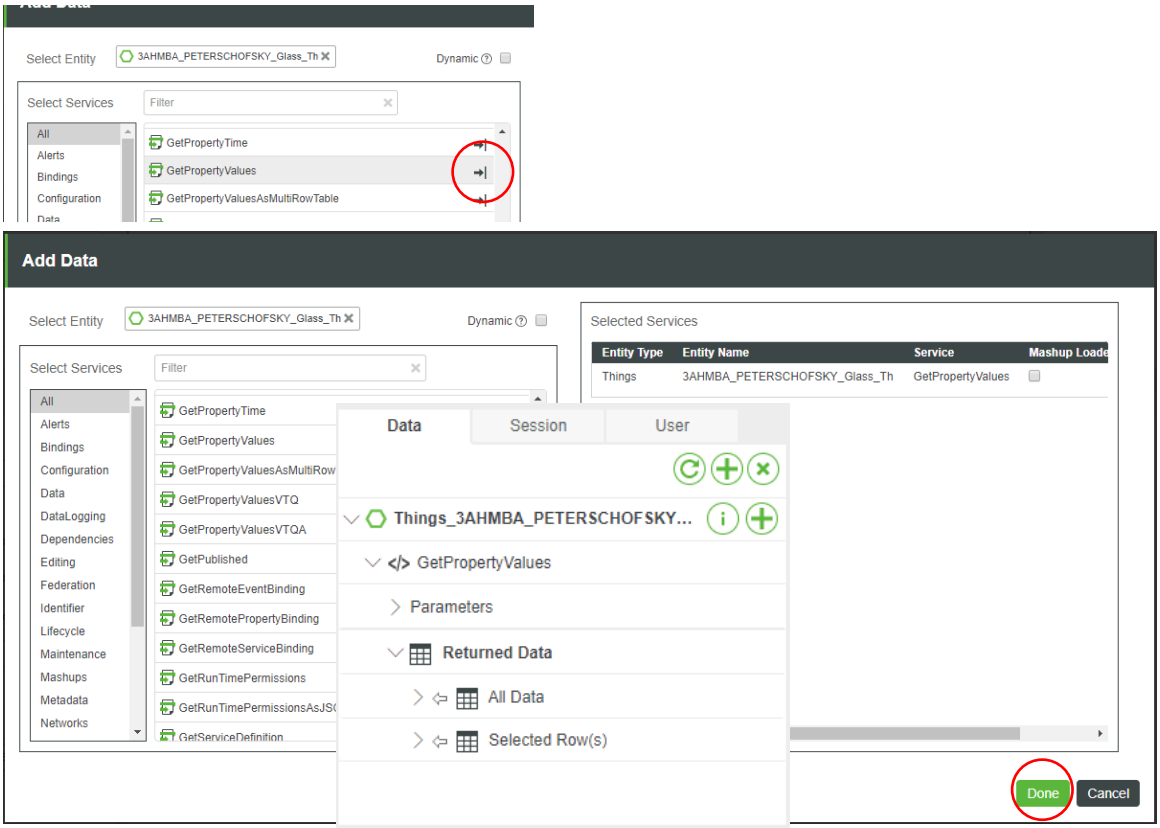
Zunächst müssen wir uns um die „Verbindung" zum richtigen Thing kümmern. Wir wollen von einem Thing Daten laden. Das geschieht über ein sogenanntes „Service". Ganz im rechten Bereich haben wir einen Bereich mit drei Tabs (Data, Session und User). Dort wählen wir Data aus und klicken danach auf das +-Symbol.



Danach öffnet sich ein Dialog-Overlay. Dort wählen wir zunächst aus dem Dropdown-Menü „Things" aus und drücken den Zauberstab. Dadurch sehen wir eine Liste von allem möglichen Things auf unserem Server. Dank der Namenskonvention sollte es einfach möglich sein unser Thing herauszufinden. Wir drücken auf das gewählte Thing darauf und wir bekommen eine Liste von allen möglichen Services, die dieses Thing bietet.

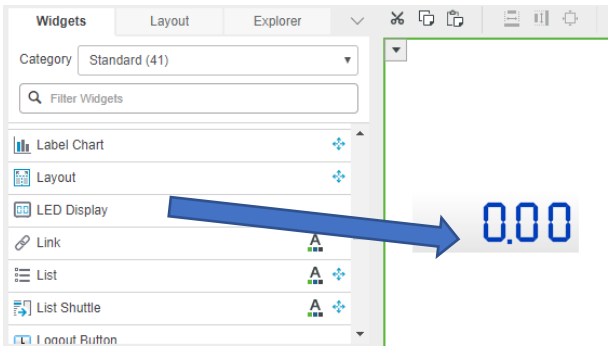


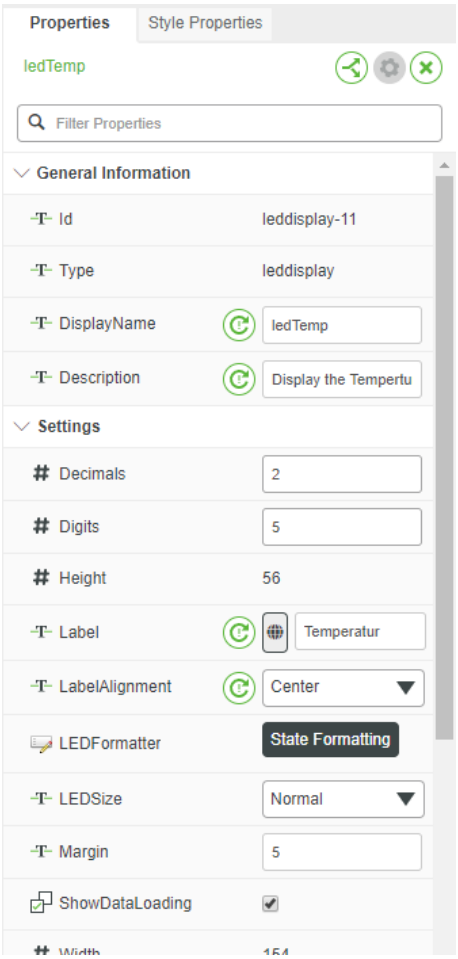
Nun, wir interessieren uns für die Properties und deren Daten. Das richtige Service dafür heißt „`GetPropertyValues`". Dieses suchen wir in der Liste und drücken auf das Pfeil-Symbol. Dadurch wir das Service im rechten Bereich angezeigt. Das bedeutet dieses Service wurde ausgewählt. Damit sind wir vorerst zufrieden und wir wählen „Done".



Der Data-Bereich im Hauptfenster hat sich verändert. Dort sehen wir jetzt die entsprechenden Services. Wir haben nur eines ausgewählt, also erscheint dort auch nur eines. Damit meinen wir, wir können die Daten aus den Properties auslesen. Das wollen wir jetzt probehalber probieren indem wir die Daten anzeigen. Ein Element, um numerische Werte anzuzeigen ist das Widget „LED-Display".

Um dieses in unsere Anzeige zu bekommen ziehen wir diese einfach von Widgets-Bereich in den Workspace (ganz genau so wie es auch in Vuforia-Studio funktioniert.

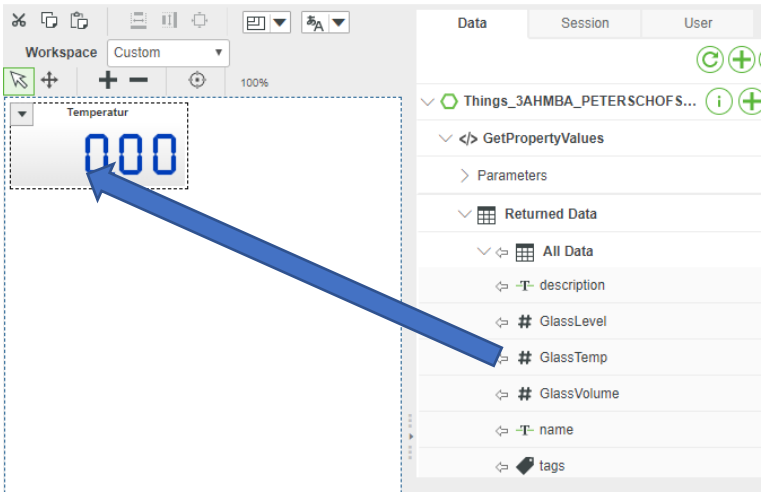


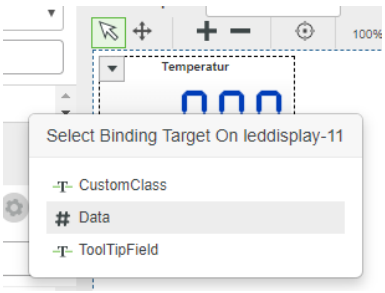


Probehalber wollen wir versuchen die Temperatur in dieser Anzeige darzustellen im linken unteren Bereich bei den Properties füllen wir deswegen ein paar Dinge aus, dass wir die Übersicht bewahren.

Als DispalyName verwende ich hier „`ledTemp`" und wähle eine aussagekräftige Description. Außerdem setze ich ein Label und schreibe dort „`Temperatur`". Dadurch bekommt die LED-Anzeige einen Text über der Zahl. Mir gefällt es gut, wenn die Beschriftung in der Mitte ist, weswegen ich bei LabelAlignment „`Center`" wähle.

Damit reicht mir die Darstellung meiner LED-Anzeige und ich kann beginnen mir den Wert auszuwählen, der dargestellt werden soll. Wen man im Data-Bereich rechts den Baum „aufmacht" und dann aus der Struktur unter All Data den Wert `GlassTemp` findet so kann man diesen auf die LED-Anzeige ziehen. Danach erscheint eine Auswahlbox, wo man auswählen kann, welche Eigenschaft der LED-Anzeige wir an den Wert binden wollen – dort wählen wir natürlich „`Data`".

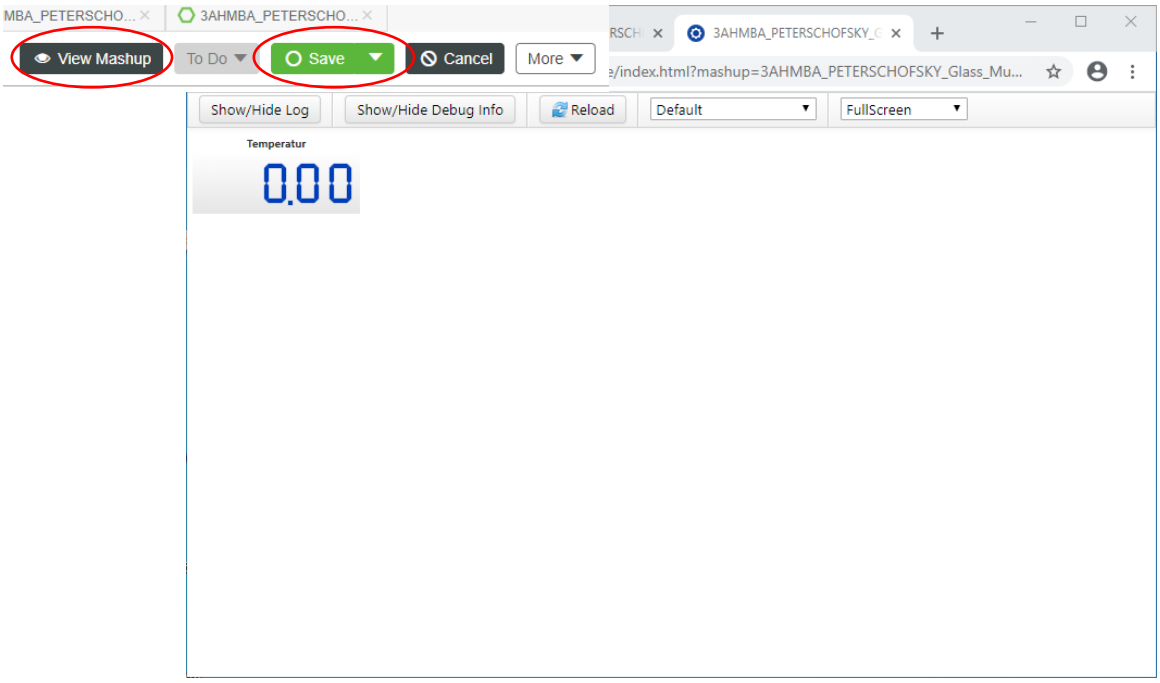




Im unteren Bereich sieht man auch sofort die Verbindung. Dabei wird dort immer das aktuelle Element in der Mitte dargestellt. Von links kommen die Datenquellen, nach rechts gehen die

Ausgänge. Mit dem roten x-Symbol könnte man die Verbindung wieder löschen. Das wollen wir jetzt natürlich nicht.

Um unser Werk zu betrachten können wir oben zunächst auf „Save" und danach auf View Mashup drücken.



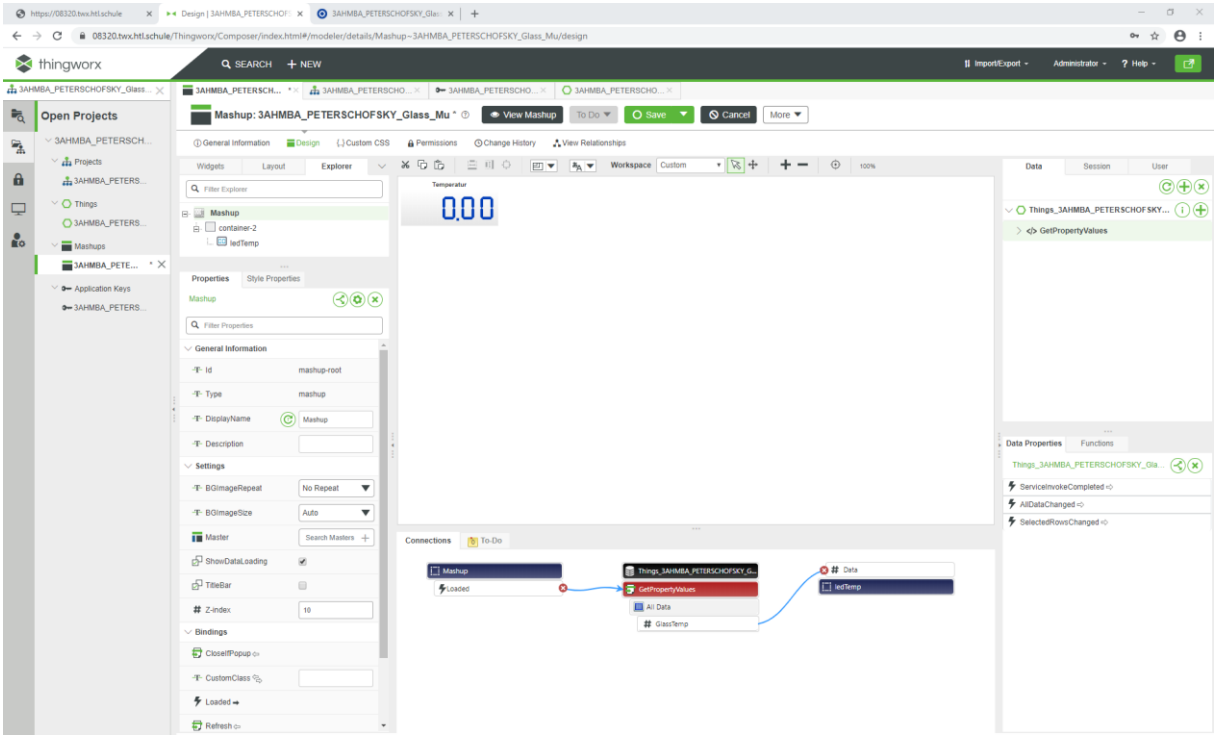
Das Ergebnis ist ernüchternd. Es steht immer 0,00 drinnen, obwohl im Thing ganz klar andere Werte vorhanden sind. Irgendwie finden die Daten nicht richtig hierher.

Der Grund ist, dass das Service zwar angelegt wurde, aber nie aufgerufen wird. Wir können das ändern. Wir wollen die Daten laden, wann das Mashup aufgerufen wurde. Dazu wählen wir im linken Bereich „Explorer" aus und wählen dort das Mashup aus. Dann erschienen direkt darunter die Properties des Mashups. Eine Property ist „`Loaded`". Diese binden wir mit dem Service `GetPropertyValue`.

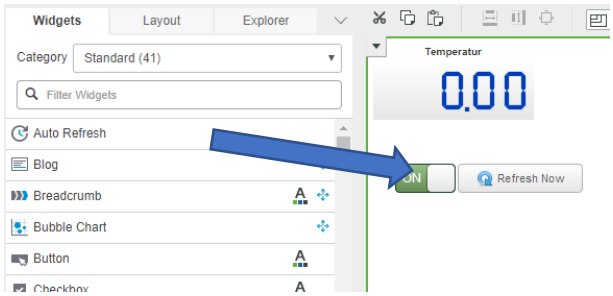
Das Vorgehen ist wieder ident: Wir ziehen den kleinen Pfeil neben Loaded auf das Service und lassen dort los. Sofort werden im unteren Bereich die Verbindungen aktualisiert und wir erkennen die Kausalitätskette:

- Sobald das Mashup geladen ist wird der Event `Loaded` ausgelöst.
- Der Event `Loaded` löst das Service `GetPropertyValue` aus. Es werden also neue Daten abgefrufen.
- Diese neuen Daten finden ihren Weg zur Eigenschaft `Data` der LED-Anzeige.

Das klingt doch gut.

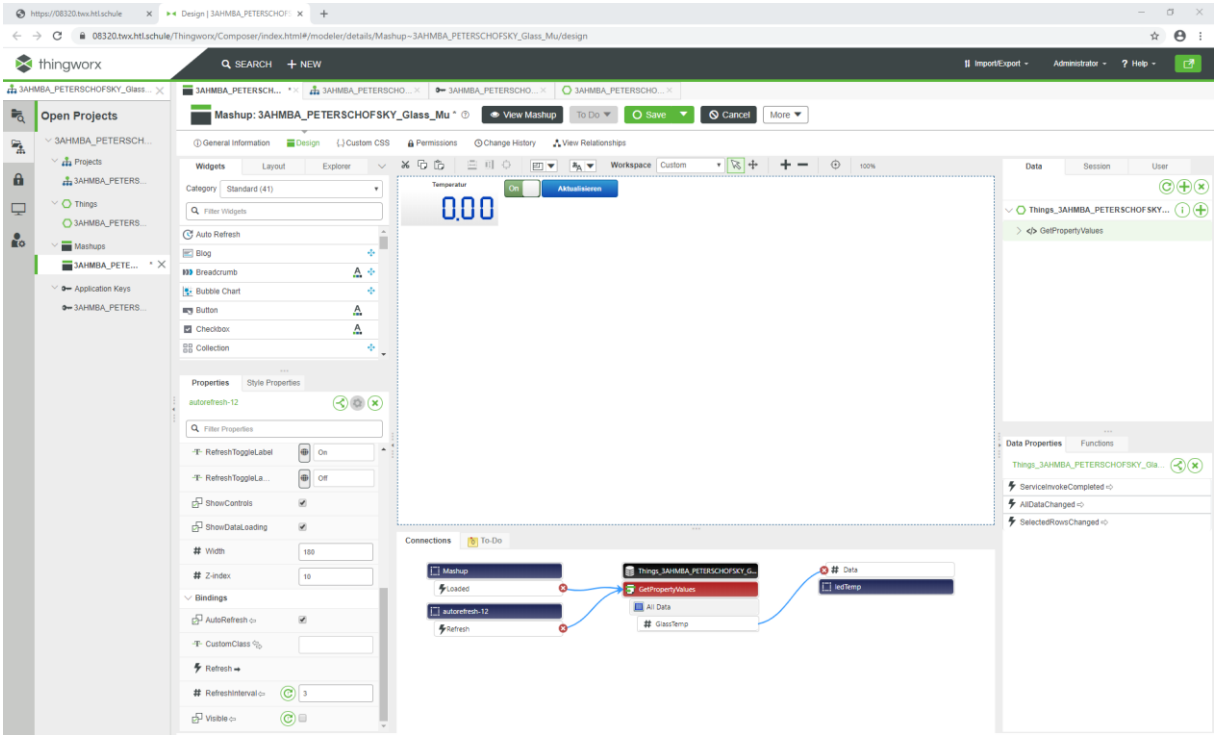


Also probieren wir es aus. Und tatsächlich steht dort jetzt der Temperaturwert. Wir haben nur ein Problem: Die Daten ändern ich nicht. Nur wenn wir Reload drücken bekommen wir einen neuen Temperaturwert. Livedaten sehen anders aus. Dazu benötigen wir einen automatischen Update. Und so ähnlich hießt auch das Widget was wir benötigen: „Auto Refresh". Dieses ziehen wir uns in den Workspace.



Was wir erhalten ist ein Element mit zwei Bedienelementen: Einen Schiebeknopf mit dem man die automatische Aktualisierung ein- und ausschalten kann und einen Knopf, mit dem eine manuelle Aktualisierung angestoßen werden kann. In den Properties des Widgets kann man verschiedenste Sachen einstellen. Ich habe folgende Dinge verändert:

- Label: hier verwende ich „`Aktualisieren`"
- RefreshInterval: Hier verwende ich 3 – ich möchte alle 3 Sekunden den aktuellen Wert anzeigen.
- Visible: Das hacke ich weg. Ich möchte eigentlich nicht, dass man das Element auf der Oberfläche sieht. Das soll im Hintergrund passieren.
- Refresh: Das ist das Ereignis und das zeihe ich wieder auf das Service `GetPropertyValues`. Also immer, wenn jetzt das Refresh-Ereignis ausgelöst wird (entweder automatisch durch das RefreshInterval oder mit einem Knopfdruck) werden jetzt neue Daten geladen.



Probieren wir es aus: Tatsächlich, der Wert ändert sich. Wenn wir den Temperaturfühler berühren so steigt die Temperatur an und die Anzeige ändert sich. Passt, so soll das sein.

3.5.6.1 Übungsaufgabe „Basic Data Display"

	Erstellen Sie so wie oben beschrieben ein Mashup für die Temperatur.	

3.5.6.2 Pflichtaufgabe „Basic Data Display"

	Erweitern Sie die Anzeige um die Eigenschaften für des Füllstands und des Füllvolumens!	

3.5.6.3 Zusatzaufgabe „Beautiful Mashup"

	Versuchen Sie nicht nur Numerische Anzeigen. Es gibt da Dinge, wie z.B. die Widgets „Gauge" oder auch „Label Chart". Erstellen Sie ein Mashup welches diese Dinge beinhaltet.	

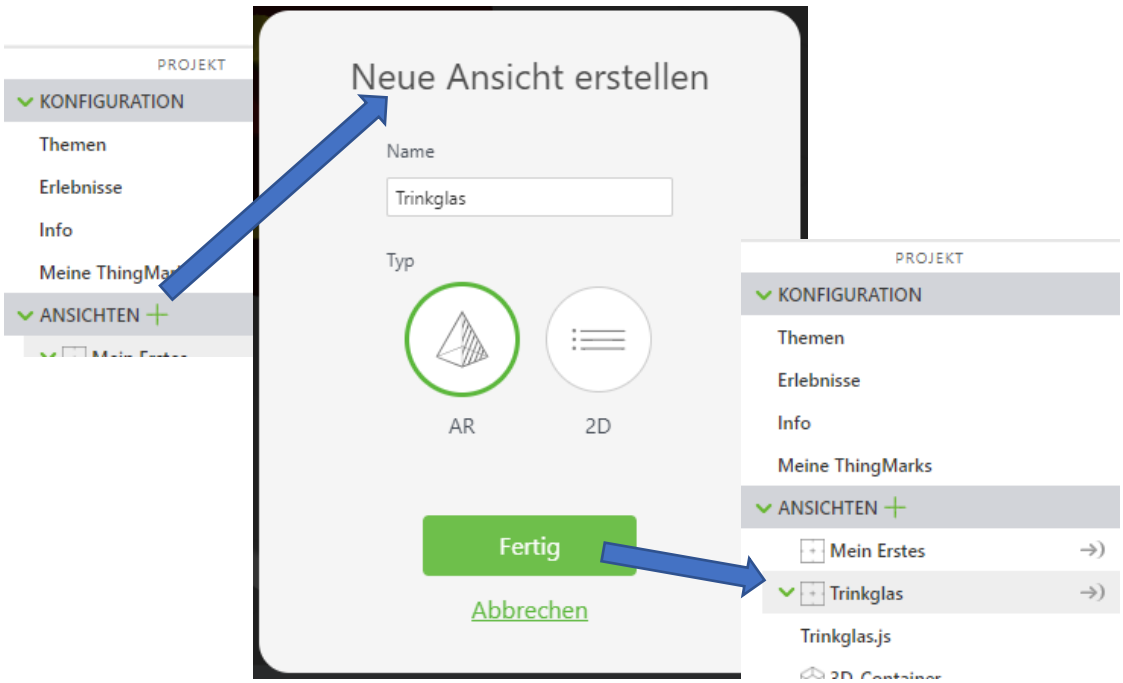
3.5.7 Verwenden von Things in Vuforia-Experiences

3.5.7.1 Darstellung eines Messwertes

Video-Link 12: https://youtu.be/KSmg_Lwvld4

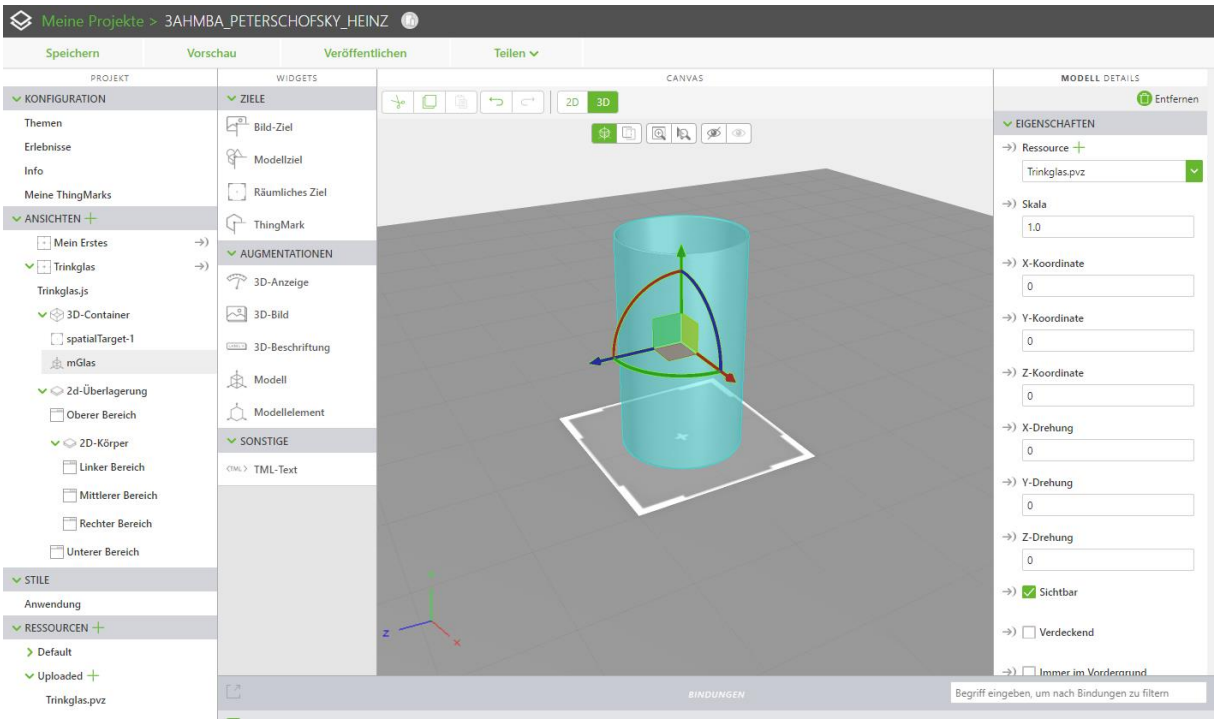
Manchmal ist es wünschenswert aktuelle Werte in die Augmentierte Welt einzublenden. Es ist möglich Daten aus Thingworx in Vuforia View zu verwenden. Glücklicherweise benötigen wir hierfür keine http-Requests, denn es gibt eine eingebaute Schnittstelle, die relativ einfach zu bedienen ist. Sehen wir uns das wieder anhand eines Beispiels an.

Wir nehmen unser Projekt her und fügen wieder eine neue Ansicht hinzu. Diese nennen wir „Trinkglas" und wählen „AR" aus. Danach sollte die Ansicht generiert worden sein und wir können loslegen.

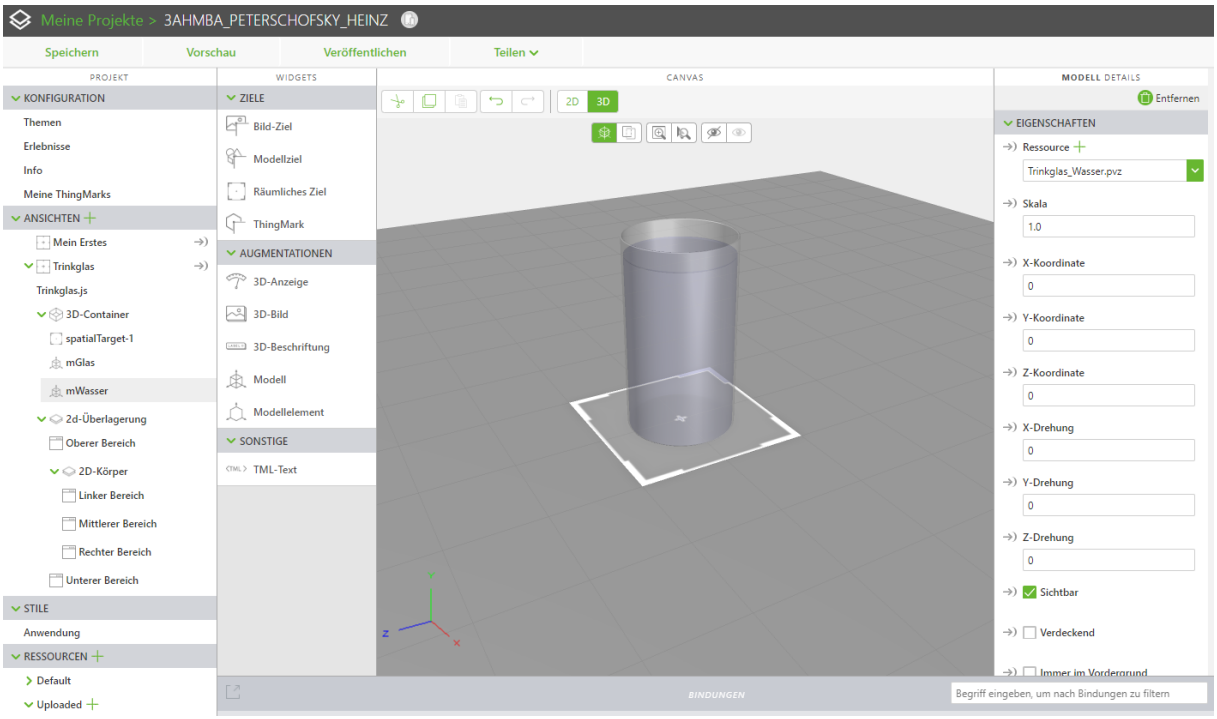


Wir fügen ein Ziel hinzu, danach ein Modell. Ich habe mir ein schönes Trinkglas gezeichnet importiere dieses als Ressource und verwende es als Modell.

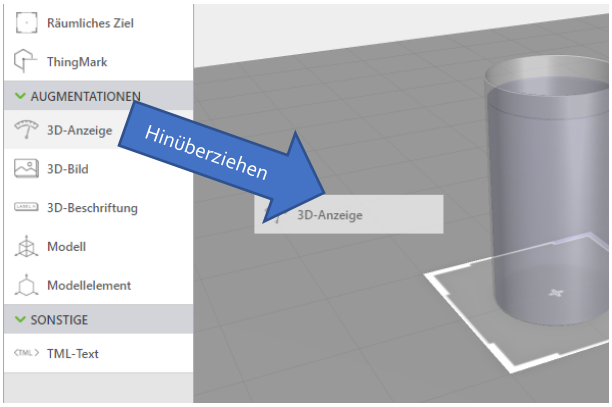
Weil es bei mir entsprechend gezeichnet ist, kann ich das Glas auf $(x; y; z) = (0; 0; 0)$ stellen und es sieht aus, wie wenn es am Tisch stehen würde.



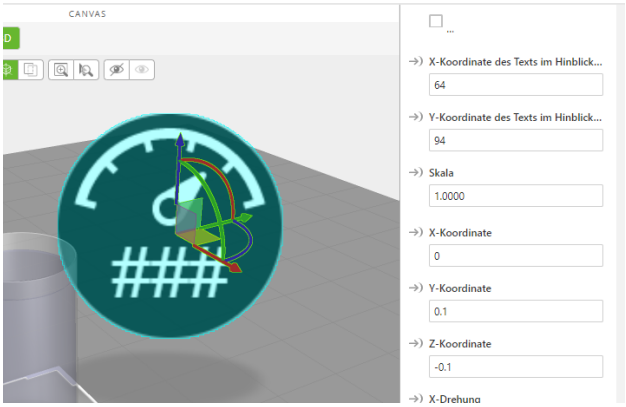
Was wir noch benötigen ist eine Wasserfüllung. Deswegen habe ich mir auch noch einen einfachen Blauen Zylinder gezeichnet, welcher den leeren Bereich des Glases fast vollständig ausfüllen kann (mit der maximalen Füllhöhe). Die Idee ist, dass wir die y-Koordinate mit der Füllhöhe anpassen und somit das Glas aussieht, wie wenn es verschiedene Füllstände hat. Ich füge also ein zweites Modell hinzu und positioniere es ebenfalls auf $(x; y; z) = (0; 0; 0)$. Damit sieht das Glas nun gefüllt aus.



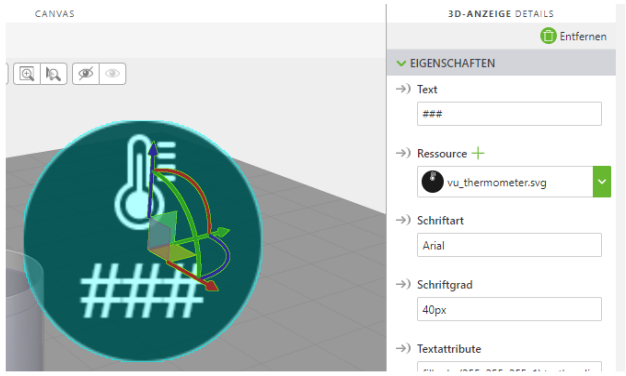
Sieht ja schon ganz gut aus.



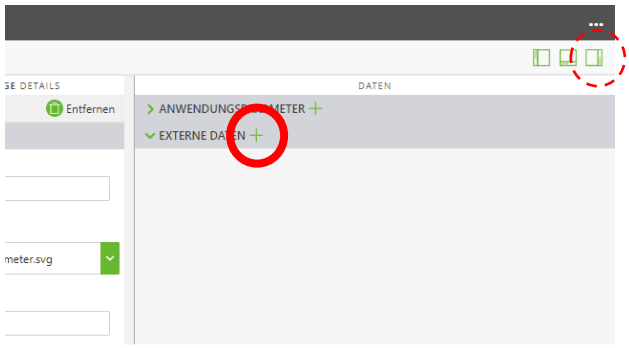
Jetzt wollen wir uns zunächst die Wassertemperatur anzeigen lassen. Dazu gibt es in der 3D-Ansicht unter Augmentation eine „3D-Anzeige". Wir nehmen also die 3D-Anzeige und ziehen sie in den Canvas.



Diese Anzeige sieht nun relativ groß aus. Setzen wir sie zunächst auf vernünftige Koordinaten. Ich setze sie neben das Glas, und zwar auf die Koordinaten $(x; y; z) = (0; 0.1; -0.1)$.

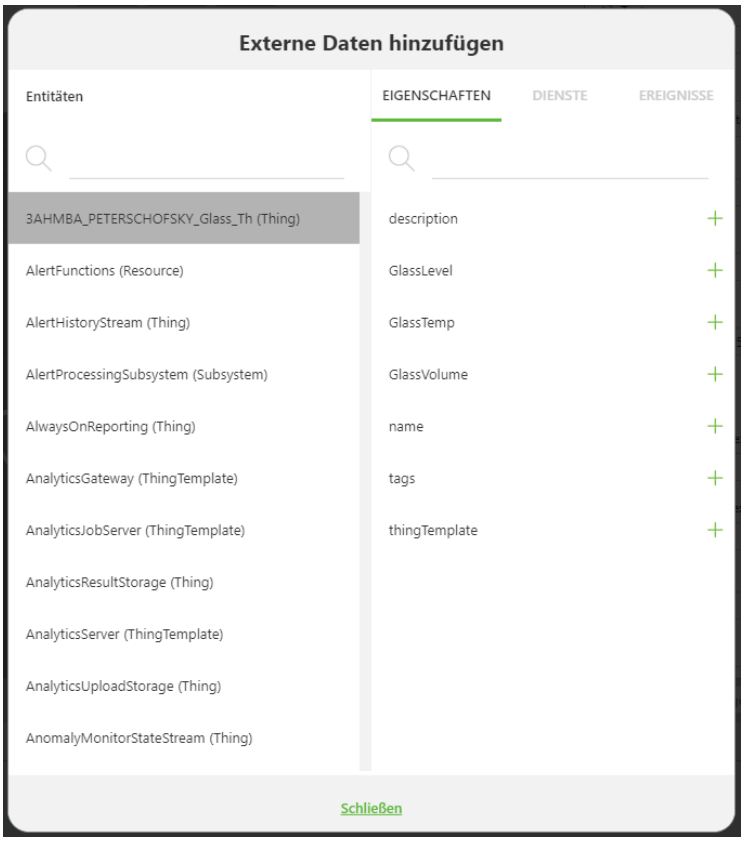


Außerdem möchte ich, dass ein Temperatursymbol erscheint. Unter Ressource kann man sich ein geeignetes Symbol auswählen.

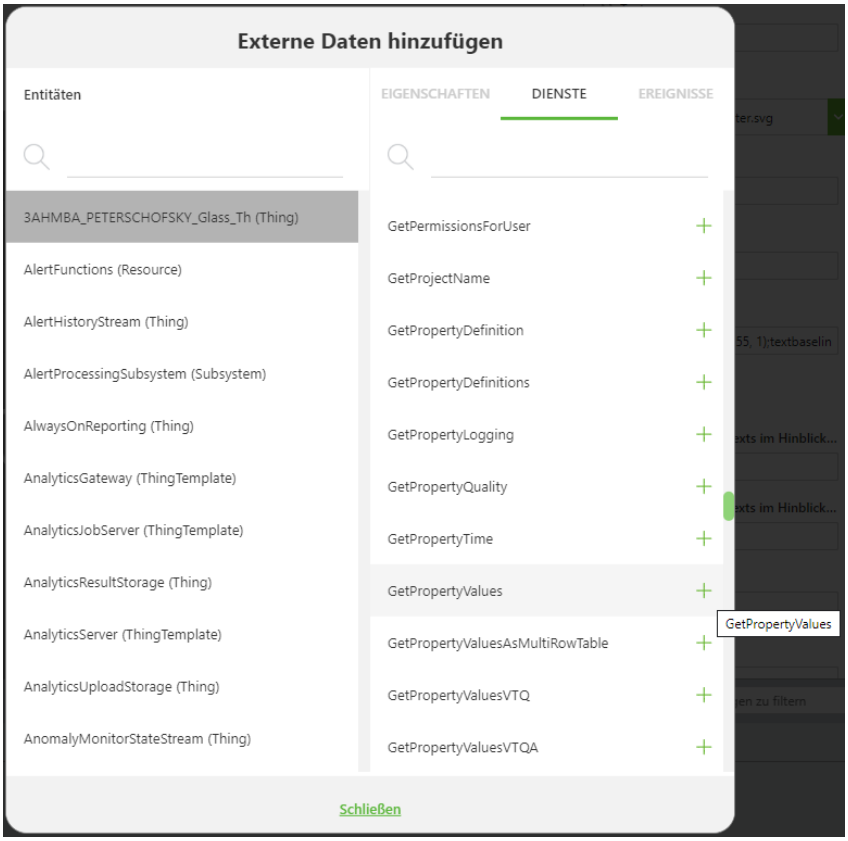


Was wir jetzt noch irgendwie erledigen müssen ist, dass der Text nicht „###" enthält, sondern den aktuellen Messwert, den wir ja aus Thingworx auslesen müssen. Dazu können wir im rechten Bildschirmbereich unter DATEN auf des „+" neben EXTERNE DATEN klicken. Sollte der Datenbereich nicht da sein, kann man ihn rechts oben unter den drei Punkten einblenden.

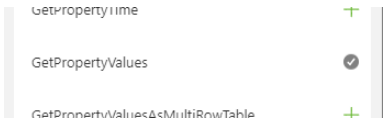
Es erscheint ein Dialog. Da unser Experience-Server und der Thingworx-Server der gleiche ist bekommen wir alle Things die am Tingworx-Server liegen aufgelistet – das ist schon einmal praktisch. Wir suchen uns unser Thing heraus und klicken darauf (bei mir war das `3AHMBA_PETERSCHOFSKY_Glass_Th`).

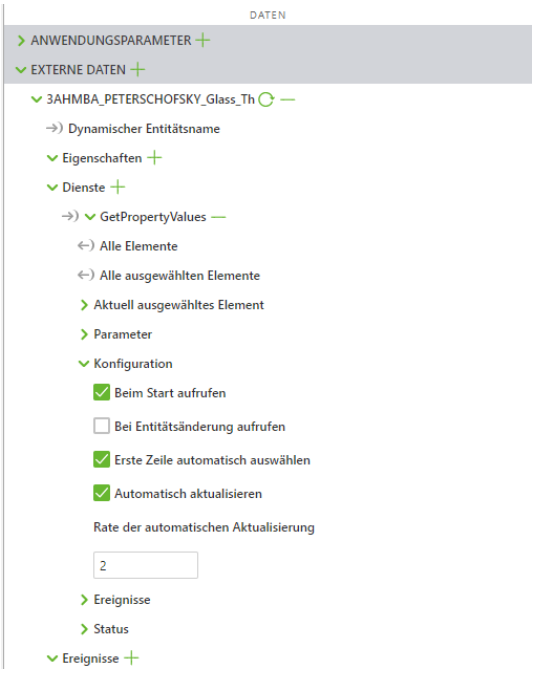


Da steht auch schon was von unseren Properties. Aber ACHTUNG, Falle! Da gehen wir nicht hinein. Wir wollen nämlich nicht direkt etwas von den Properties, sondern wollen einen Dienst starten. Den Dienst, den wir suchen ist `GetPropertieValues`. Wir wechseln also auf DEINSTE und suchen uns diesen heraus:



Nachdem wir das „+" Symbol daneben geklickt haben ändert sich dieses in ein graues Häkchen. Wir haben den Dienst abonniert. Das ist im Moment alles was wir von unserm Thingworx-Server wollen, wir können auf Schließen drücken.





Zunächst müssen wir uns noch aussuchen wann wir die Daten haben wollen. Das geschieht auf der rechten Seite bei den Externen Daten. Die haben sich jetzt verändert. Dort finden wir nämlich jetzt unseren abonnierten Dienst. Im Baum unter GetPropertyValues > Konfiguration stellen wir ein paar Sachen ein:

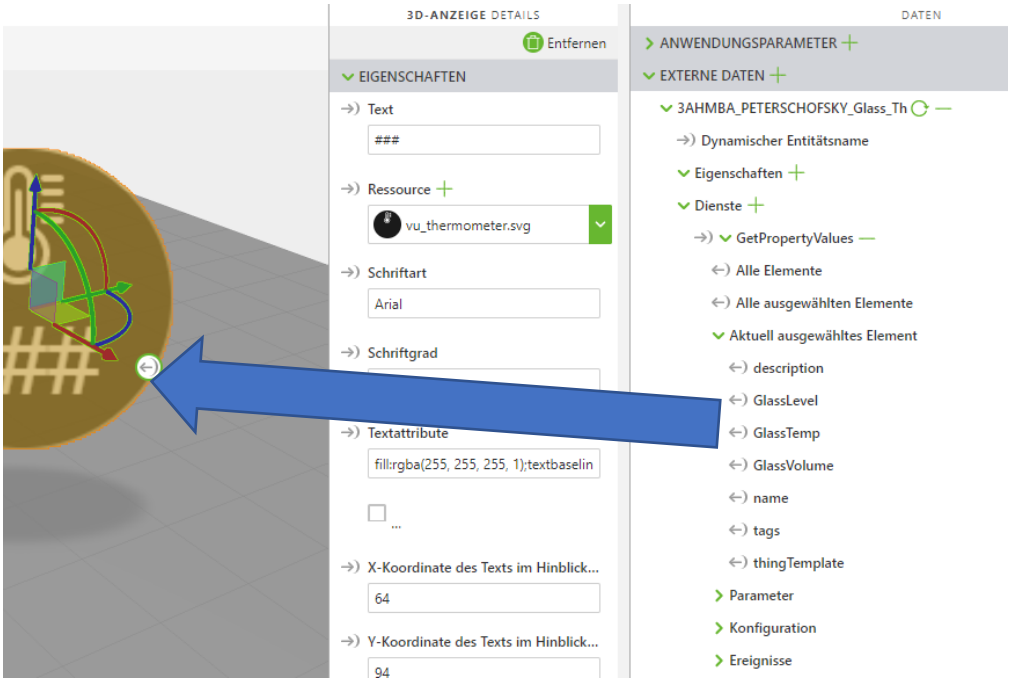
Wir wollen, dass beim Start eine Abfrage erfolgt

Wir wollen eine Automatische Aktualisierung

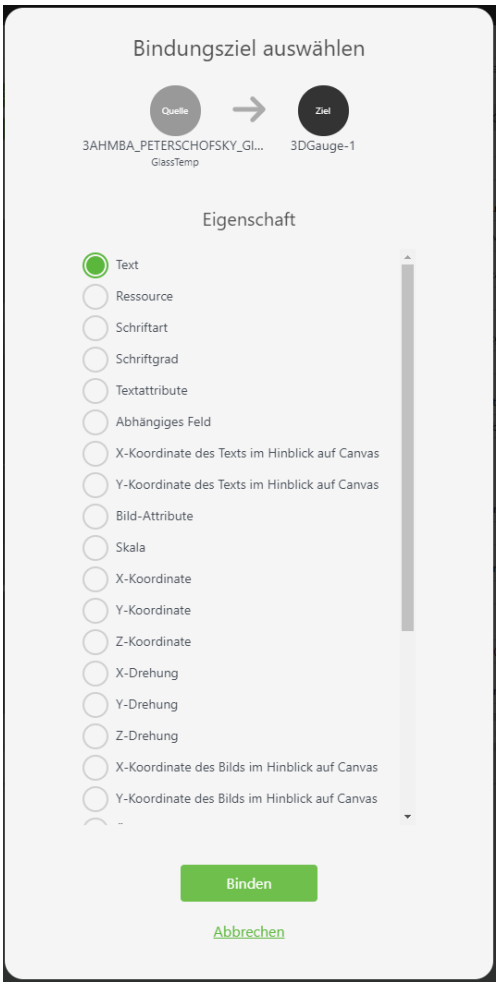
Wir wollen diese Aktuleideurng alle 2 Sekunden.

Ist alles so eingestellt, sollte es so ähnlich wie auf dem Bild aussehen.

Was jetzt noch fehlt, ist, dass wir irgendwie sagen müssen welchen Wert unsere Anzeige zeigen soll. Dazu öffnen wir bei den Externen Daten den Zweig „Aktuell ausgewähltes Element". Dort finden wir unsere Properties wieder – wir nehmen die Temperatur (Also `GlassTemp`) und ziehen es auf die Anzeige. Dort lassen wir es los und wir erhalten eine Auswahl an Dingen, die wir an diesen Wert binden können – eine ganz schöne Liste!



Natürlich wählen wir hier den Text aus. Wir wollen ja die Temperatur anzeigen. Theoretisch könnten wir die Anzeige auch verschieben (einfach die Koordinaten verändern). Das wäre die Idee zum Wasserpegel verändern.



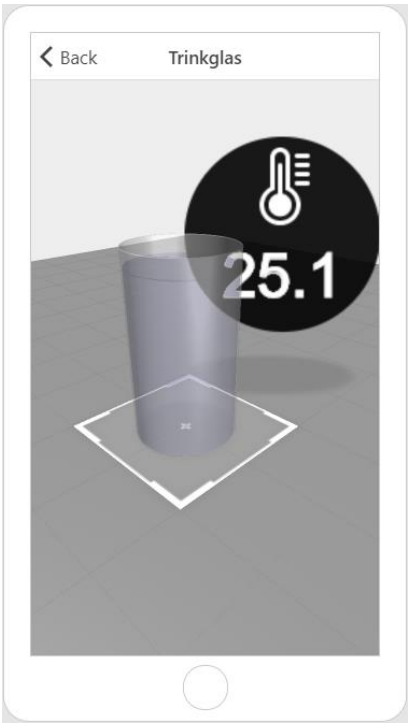
Zunächst aber beschränken wir uns auf die Anzeige des Messwertes, also wählen wir Text aus und drücken „Binden".

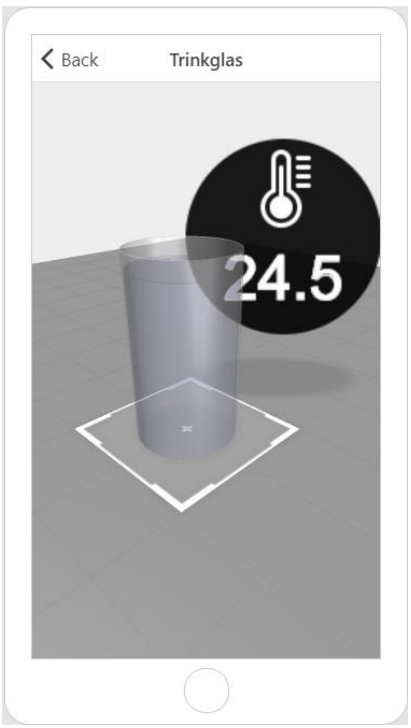
Im unteren Bereich erscheint dann auch sofort die Bindungsausdruck, wenn die 3D-Anzeige ausgewählt wurde.

Dabei fällt auf, dass die 3D-Anzeige noch immer den Namen „`3DGauge-1`" hat. Den ändern wir wieder auf einen besseren Namen. Ich verwende „`3DGTemp`".

Das war es auch schon. Wir können auf Vorschau klicken. Tatsächlich, die Temperatur erschient in der 3D-Anzeige!

Ändern wir die Temperatur am Thingworx-Server, so ändert sich auch die Anzeige. Die Verbindung scheint zu funktionieren. Wir können unsere Experience veröffentlichen.



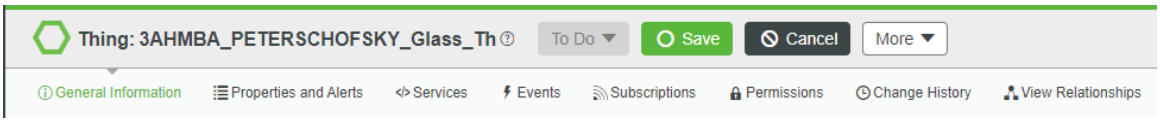


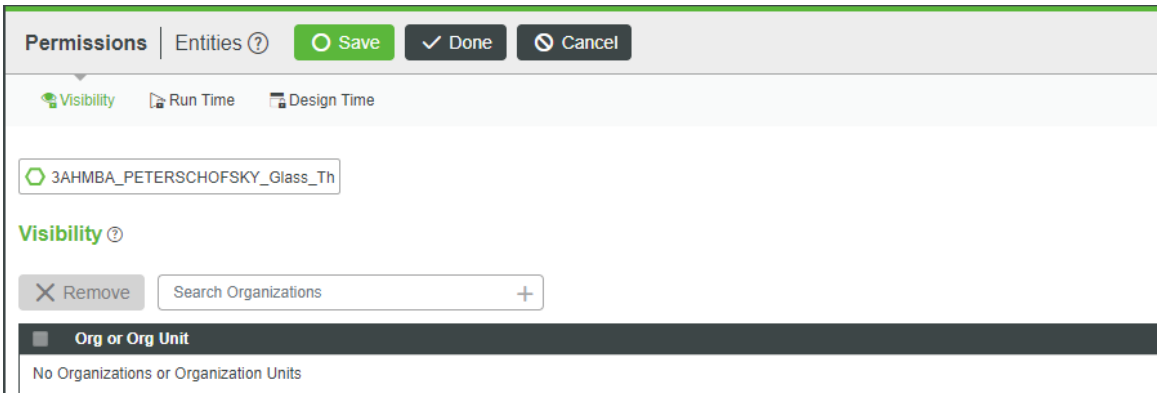
Wenn wir die Experience mit unserem Smartphone öffnen, dass sollte dies auch hier funktionieren. Tatsächlich bekommt man jedoch keine Werte! Der Grund sind die Rechte. Unsere Experience ist öffentlich – wir müssen keine Logindaten eingeben, um die Experience zu starten. Deswegen sind wir auch nicht eingelogged und haben auch keinen Application Key eingegeben – also bekommen wir keine Daten. So einfach ist das.

Prinzipiell gibt es zwei Möglichkeiten damit umzugehen:

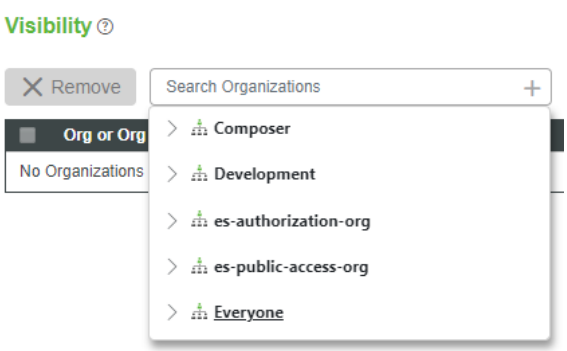
- Die Experience beim Veröffentlichen nicht öffentlich machen. Dann benötigt man zum Öffnen Username und Passwort und erhält Daten.
- Den Lese-Zugriff auf die Daten öffentlich machen. Wenn man der Öffentlichkeit erlaubt die Daten zu lesen, dann bekommt die Experience ebenfalls die notwendigen Daten.

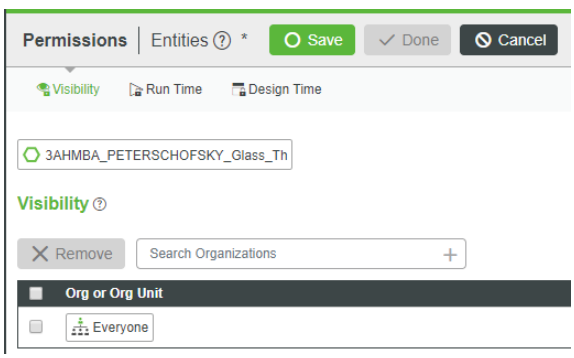
Wir werden den zweiten Ansatz wählen. Also öffnen wir das zugehörige Thing in Thingworx-Composer. Mit einem Klick auf „Permissions" in der oberen Leiste bekommen wir die Zugriffsrechte auf unser Thing präsentiert.



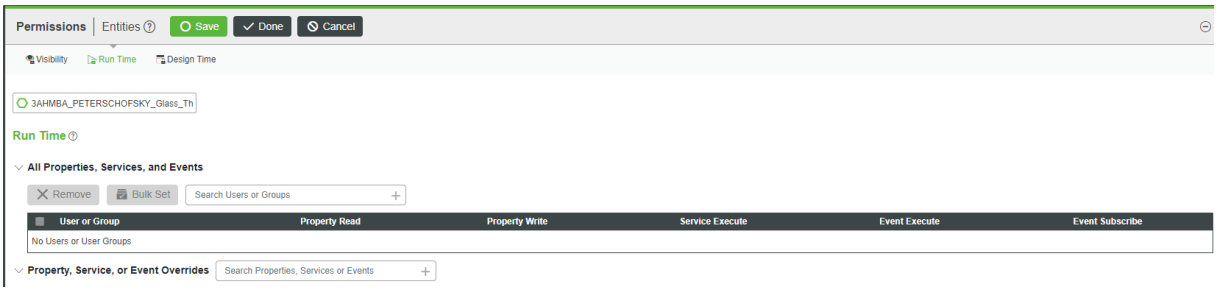


Zunächst ist das Problem, dass unser Service von außen gar nicht sichtbar ist. Wir tragen also bei der Sichtbarkeit „`Everyone`" ein und drücken danach auf „Save".

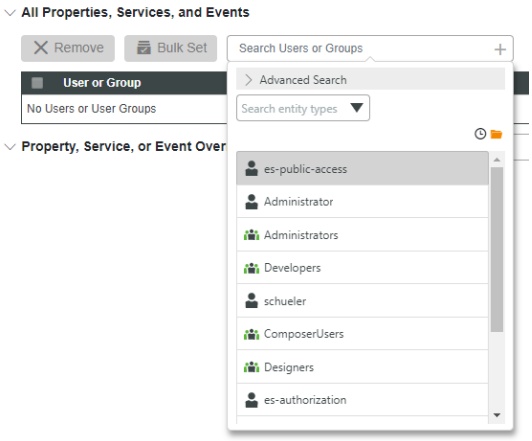




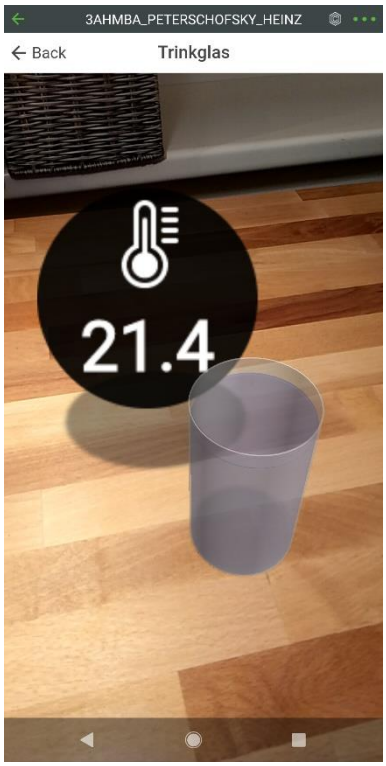
Dadurch ist das Service zwar sichtbar, aber wir haben nach wie vor keine Rechte. Wir müssen die Zugriffsrechte für die Run Time ändern. Denn zur Laufzeit wollen wir die Daten haben. Also wechseln wir dorthin. Im Moment ist noch nichts eingetragen – das wollen wir ändern.



Der User, den wir hinzufügen müssen, heiß „`es-public-access`". Diesem User erlaben wir genau zwei Dinge: Ein (oder mehrere) Properties zu lesen und ein Service zu starten. Damit ist sichergestellt, dass man den Wert zwar bekommt (über das Service `GetPropertyValues`), aber nicht beschreiben kann (weil wir ja Property Write nicht erlauben).



Der Versuch macht uns sicher: Es scheint zu funktionieren – wir sehen einen Wert und dieser ändert sich bei einer Änderung.



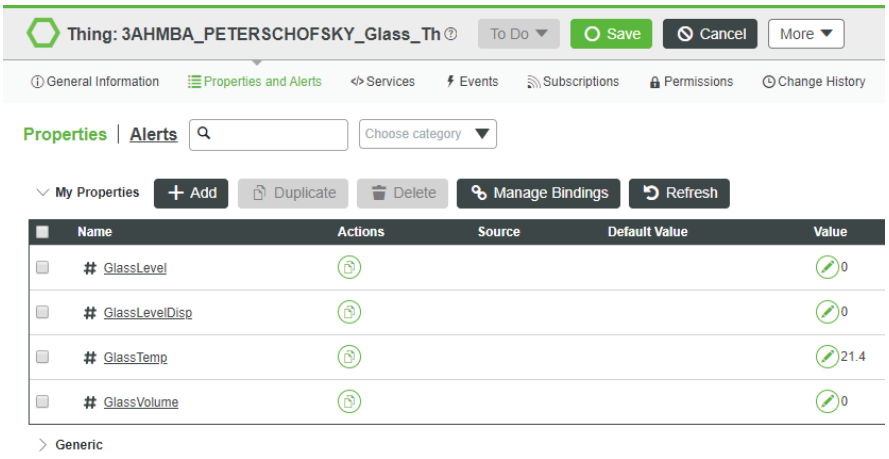
3.5.7.2 *Pflichtaufgabe „Show augmented value"*

	Stellen Sie ebenfalls die Temperatur in ihrer Augmentation dar. Erweitern Sie die Anzeige auch um den Inhalt des Glases. Wählen Sie ein geeignetes Piktogramm und platzieren Sie es sinnvoll.	

3.5.7.3 *Animieren eines Objekts*

Video-Link 13: https://youtu.be/q8vvcCuQYLY

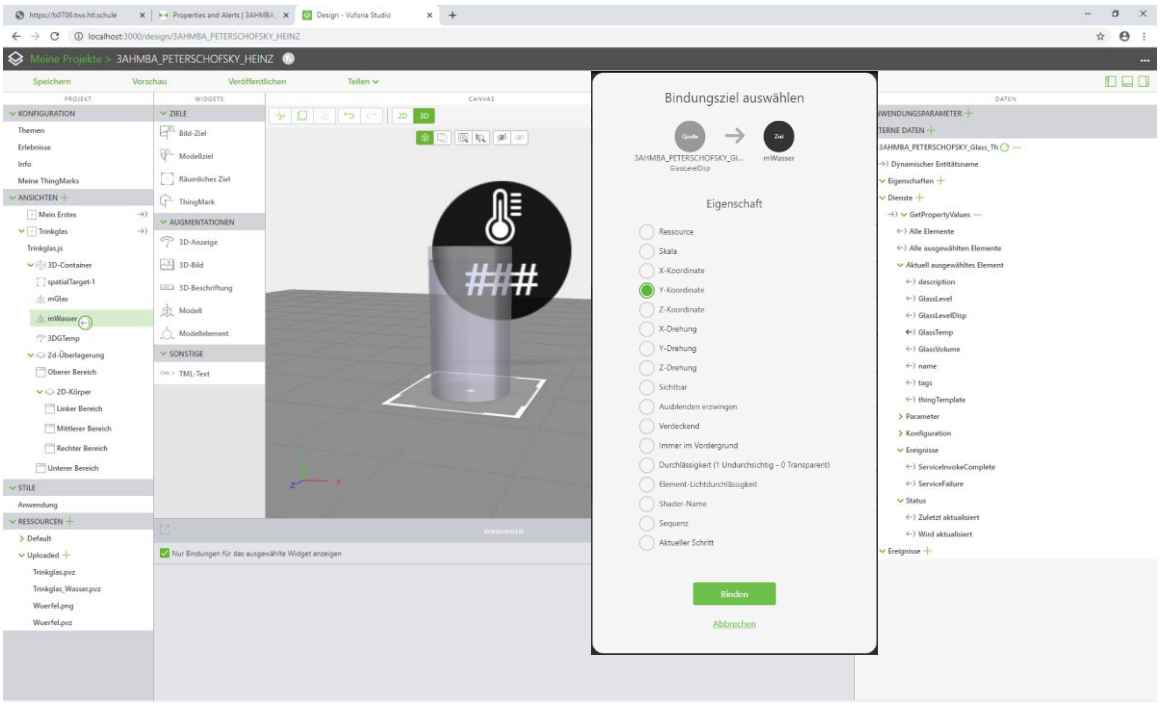
Leider gibt es keine einfache Möglichkeit den Wert, den wir aus Thingworx bekommen vorher zu manipulieren. Das bedeutet, wenn wir einfach die Füllhöhe nehmen und den Wasserblock in y-Richtung damit verschieben funktioniert es nicht richtig. Was wir benötigen ist bereits die korrekte y-Koordinate in Metern. Wir benötigen also ein neues Property, wo wir direkt die y-Koordinate bekommen. Nun, erweitern wir unser Thing entsprechend. Fügen wir ein Property mit dem Namen „`GlassLevelDisp`" hinzu.

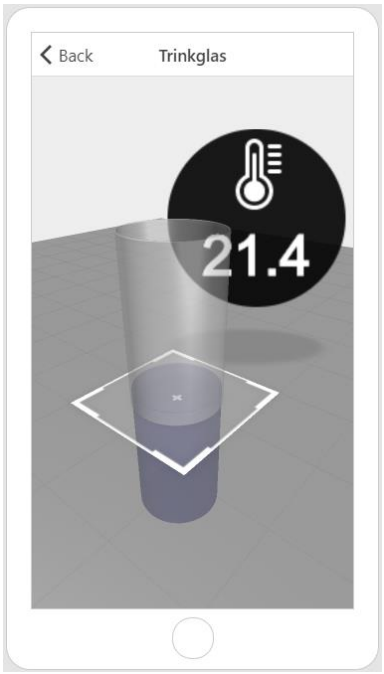


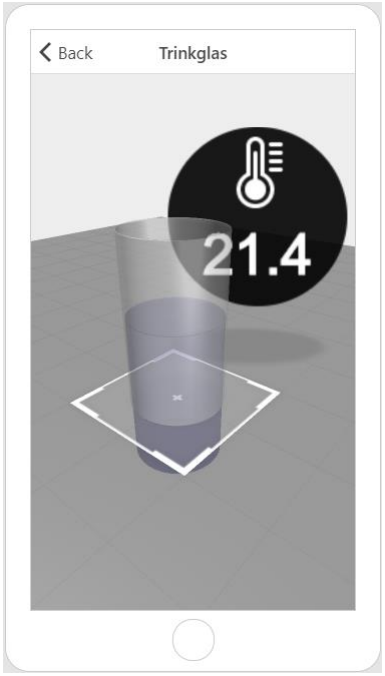
Diese Property sollten wir auch in Vuforia Studio zur Verfügung haben. Ist dies nicht der Fall, so kann der Update-Pfeil neben dem Thing gedrückt werden. Ist das Thing in Thingworx erfolgreich gespeichert worden sollte spätestens jetzt die neue Eigenschaft in Vuforia Studio verfügbar sein.

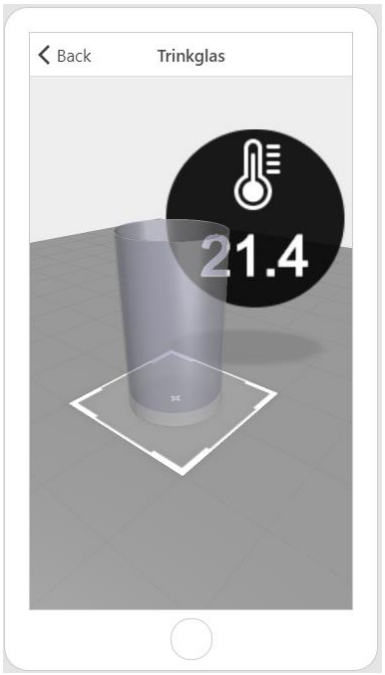
Diese ziehen wir jetzt auf das Wasser-Modell, welches bei mir „`mWasser`" heißt (denn dieses soll sich ja verändern) und binden es an die Y-Position. Dadurch wird sich die Wassersäule in y-Richtung verschieben. Aber wo ist jetzt eigentlich Null? Nun, mein Glas hat einen $10mm$ dicken Bode und mein Wasserzylinder ist $100mm$ hoch. Das bedeutet eine y-Koordinate von $y = -0.09$ wäre ein „trockener" Boden. Und bei $y = 0.01$ sind wir komplett voll.

Also probieren wir es mit der Vorschau aus was passiert.







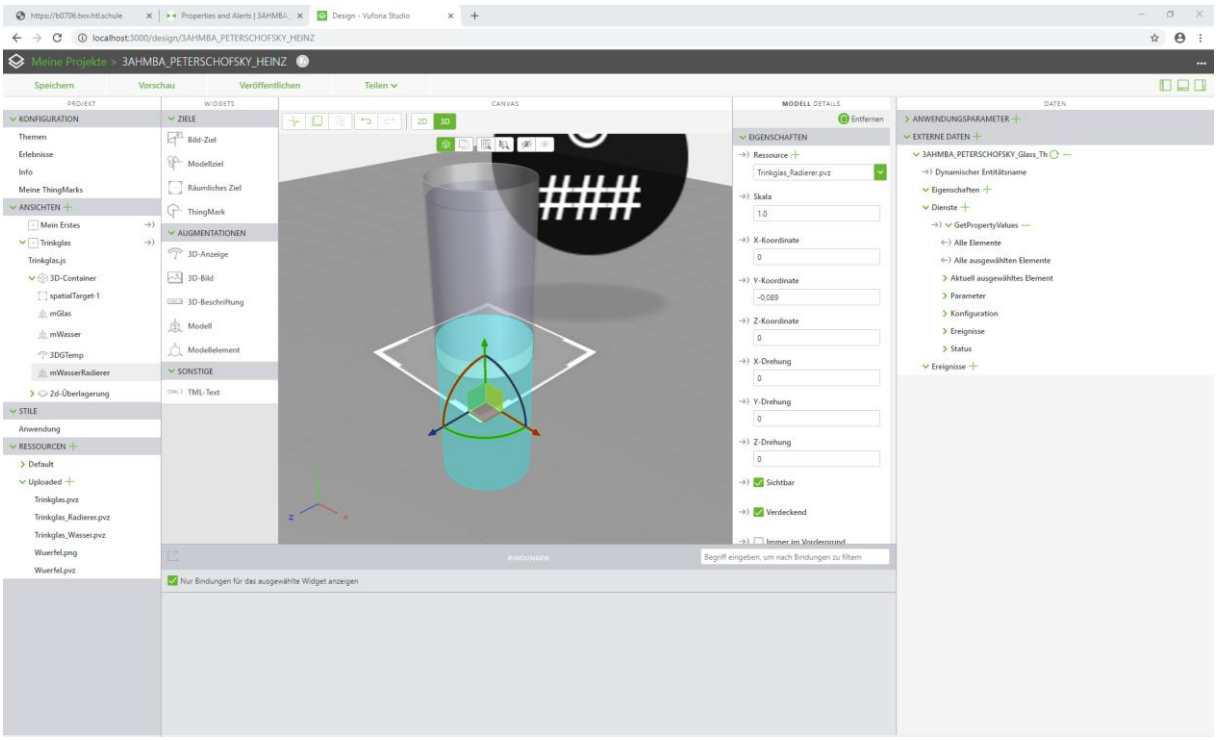


$GlassLevelDisp = -0{,}09$ $GlassLevelDisp = -0{,}04$ $GlassLevelDisp = +0{,}01$

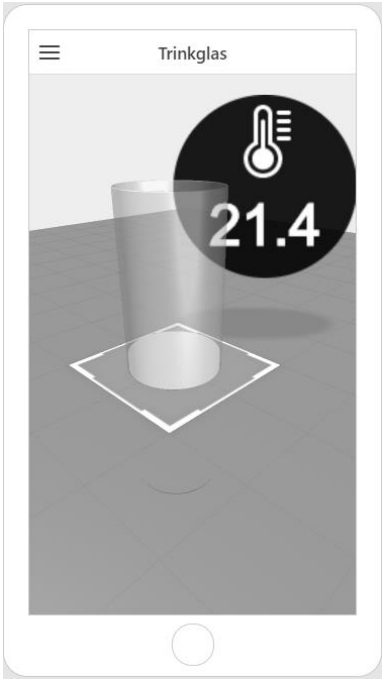
Tatsächlich, es bewegt sich – sieht aber nicht so aus, wie wenn es so gehören würde. Die Wassersäule wird nämlich nicht weniger, sondern schiebt sich einfach durch das Glas hindurch nach unten. Nun, so soll das nicht bleiben. Wir benötigen eine „Radierer" der uns den unteren Teil des Wassers weglöscht. Dieser Radierer kann ebenfalls ein Modell sein, welches unsichtbar aber verdeckend wird. Diesen Radierer platzieren wir genau unter dem Wasser und wenn es nach unten geht wird der untere Teil von einem unsichtbaren Objekt verdeckt – ein Superschmäh!

Also frisch ans Werk und einen geeigneten Radierer gebaut. Er muss etwas mehr Durchmesser als das Wasser haben, aber etwas weniger als das Glas – sonst wird der unter Teil des Glases gleich ebenfalls wegradiert. Bei mir ist das eine Kopie des Wasser-Modells nur ist es um $1mm$ höher und

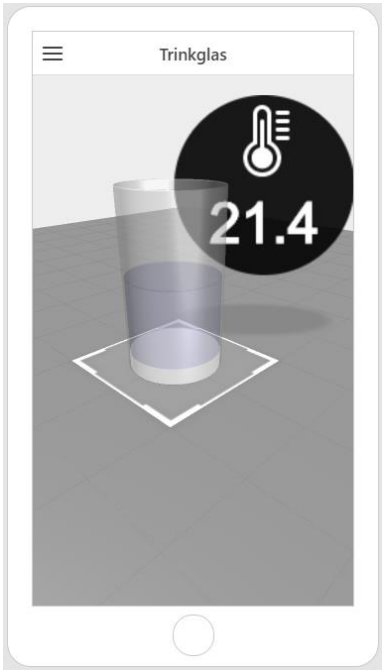
hat $0.5mm$ mehr Durchmesser. Ich füge eine neue Ressource hinzu, ein neues Modell, wähle die Ressource und positioniere es entsprechend. Danach wähle ich es noch als sichtbar und verdeckend.



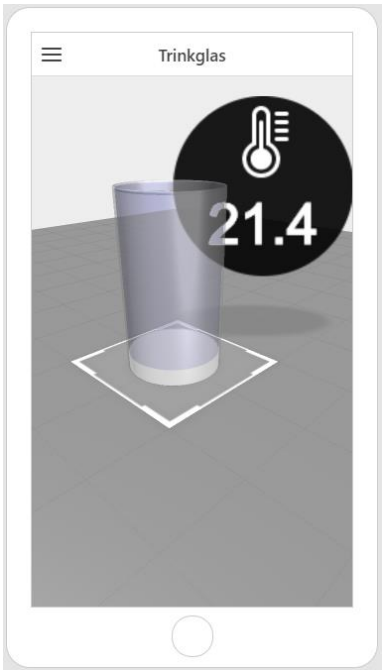
Probieren wir es aus. Tatsächlich, das Wasser verschwindet. Das sieht ja jetzt ganz gut aus – nur die Tatsache, dass unser Radierer auch den Glasboden wegradiert ist etwas störend. Da ist immer so ein hässlicher weißer Sockel.



$GlassLevelDisp = -0{,}09$

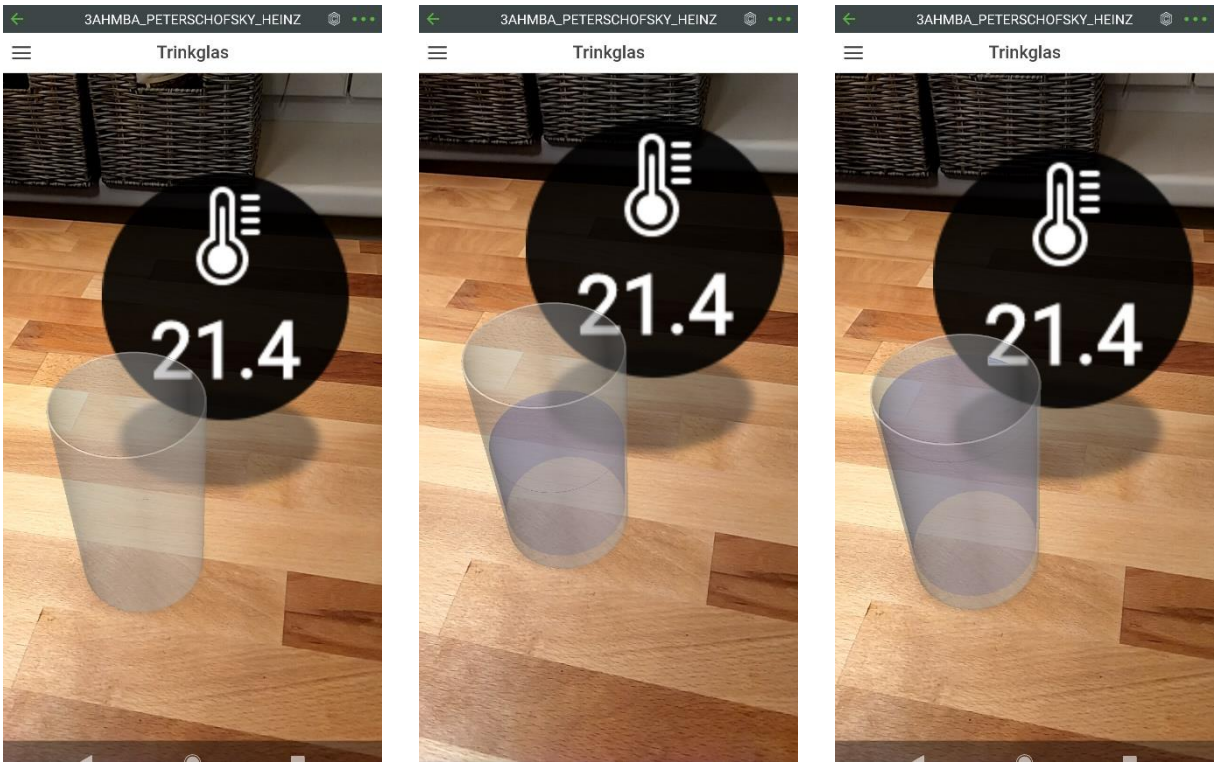


$GlassLevelDisp = -0{,}04$



$GlassLevelDisp = +0{,}009$

Aber, es ist zumindest so gut, dass wir uns das Ergebnis ja einmal mit einem „echten" Mobilgerät ansehen und beurteilen können:



$GlassLevelDisp = -0{,}09$ $GlassLevelDisp = -0{,}04$ $GlassLevelDisp = 0$

Na bitte, das sieht doch ganz gut aus. Zum Glück passen Vorschau und „echt" nicht immer 100% zusammen!

3.5.7.4 Pflichtaufgabe „Fillanimation"

	Stellen Sie ebenfalls die Füllhöhe in Ihrer Augmentation dar. Versuchen Sie den momentanen Inhalt des Glases an der Wasseroberfläche darzustellen.	

4 Anhang

4.1 Arduino Referenz

4.1.1 Sprachstruktur

4.1.1.1 Kommentare

```
// Das ist ein Kommentar in dieser Zeile. Der startet mit //.

/* Das ist ein Kommentar über mehrere Zeilen.
   Da gibt es einen Anfang mit /*
   Und ein Ende, welches mit */ gekennzeichnet ist */
```

4.1.1.2 Variablen

Variable bestehen aus einem Typen und einen Namen:

```
type variablenname;
```

Mögliche Typen sind:

Ganzzahlen:

`byte`	mit 8-Bit (also Werte von 0 bis `255`) Eine Konstante wird ganz einfach als Zahl angegeben z.B.: `24`
`int`	am Arduino mit 16-Bit (also Werte von `-32768` bis `32767`) Eine Konstante wird ganz einfach als Zahl angegeben z.B.: `5`
`unsigned int`	am Arduino mit 16-Bit (also Werte von `0` bis `65535`) Eine Konstante wird ganz einfach als Zahl angegeben z.B.: `125`
`long`	am Arduino mit 32-Bit (also Werte von `-2147483648` bis `2147483647`) Eine Konstante wird ganz einfach als Zahl angegeben z.B.: `-275`
`unsigned int`	am Arduino mit 32-Bit (also Werte von `0` bis `4294967295`) Eine Konstante wird ganz einfach als Zahl angegeben z.B.: `55`

Kommazahlen (Gleitpunktdarstellung):

`float`	am Arduino 4 Byte, 6-7 Nachkommastellen und Hochzahl. Wertebereich: `-3.4028235E+38` bis `3.4028235E+38` Eine Konstante muss mit Kommapunkt angegeben werden. z.B..: `5.0`

Zeichen:

`char`	enthält den Code von genau einem Zeichen. Es kommt die ASCII-Tabelle um Einsatz. Eine Konstante wird in einfachem Hochkomma angegeben. Also z.B. `'a'`
`String`	enthält eine Zeichenkette. Unterscheidet sich vom ANSI-C-String. Ist eine Objektlasse mit einer Fülle von Methoden. Bitte diesbezüglich auf www.arduino.cc nachzusehen. Eine Konstante wird in doppeltem Hochkomma angegeben. Also z.B. `"Blinde Nuss"`

Wahrheitsangaben (Bool'scher Ausdruck)

`bool`	kann entweder `TRUE` oder `FALSE` sein, einfach eine Wahrheitsaussage.

4.1.1.3 Objekte

Objekte sind etwas komplexer als einfache Variable, obwohl die Definition eigentlich gleich aussieht. Auch haben Objekte einen Typ und einen Namen. Diese haben jedoch nicht nur einen Wert, sondern auch „Methoden". Das sind praktisch in die Variable „eingebaute" Funktionen.

Ein Beispiel ist das vorhin erwähnte Objekt `String`. Das hat einen Wert, aber auch eingebaute Funktionen. Eine dieser Funktionen ist z.B. `startsWith()`.

Ein Codebeispiel zum besseren Verständnis:

```
String name;  // Name von einer Person
…
if (name.startsWith("Stefan")) {
  // Hier machen wir etwas, wenn der Name mit "Stefan" anfängt
}
…
```

Dabei muss das `startsWith()` nirgendwo definiert werden, es ist bereits in der Klasse „`String`" eingebaut – praktisch.

4.1.1.4 Funktionen

Definition einer Funktion:

```
type funktionsname([type1 parameter1][, type2 parameter2 [,…]]) {
     //Code der Funktion
}
```

`type`: Jede Funktion hat einen Rückgabewert. Dieser kann auch vom Typ `void` sein. Ein Typ für den Rückgabewert muss aber vor dem Funktionsnamen angegeben werden.

`funktionsname`: Eine Funktion hat einen Namen, mit der die Funktion aufgerufen werden kann. Dieser besteht aus Buchstaben und Ziffern. Beginnen muss der Funktionsnamen immer mit einem Buchstaben.

Funktionen können (müssen aber nicht) Übergabeparameter aufweisen. Jeder Parameter hat einen Typ und einen Namen – genau wie bei Variablen.

Die geschwungenen Klammern {..} fassen einen Anweisungsblock zusammen. Hier kennzeichnen sie den Umfang der Funktion. Besteht die Funktion nur aus einem einzigen Befehl, könnte man die Klammern auch weglassen – das wäre aber sehr selten. Häufiger kommt das bei `if`-Anweisungen vor. Dort werden die zugehörigen Anweisungen auch mit {..} gruppiert (siehe dort)

Aufruf einer Funktion:

```
variablenname = funktionsname([type1 parameter1][, type2 parameter2
[,…]]);
```

4.1.1.5 Kompilezeit-Konstanten

Zum Zeitpunkt der Programmerstellung können bereits Konstanten vergeben werden. Bevor das Programm übersetzt wird, werden dann die entsprechenden Stellen mit dem Wert der Kontanten ersetzt (wie ein Ersetzen-Befehl in einem Dokument). Dadurch nehmen solche Kontanten im Hauptspeicher des Arduino keinen Platz ein. Sie sind Teil des Programms und deswegen im Programmspeicher. Das ist oft hilfreich, wenn man Werte öfter als einmal benötigt und man nicht sicher ist, ob man das in zukünftigen Versionen ändern will.

Syntax:

```
#define KONSTANE WERT
```

Immer wenn im Programm irgendwo die Zeichenkette `KONSTANTE` steht, wird sie vor dem Übersetzen mit `WERT` ersetzt.

Ein Beispiel wäre die Pin-Nummer eines LED-Signals. Man muss dann nicht immer Pin Nummer 3 schreiben, sondern definiert sich eine Konstante. Ändert sich die LED-Nummer später braucht man nur den Wert der Konstanten anpassen und nicht bei jeder Codestelle:

```
#define ledPin 3

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);    // turn the LED on
  delay(1000);                   // wait for a second
  digitalWrite(ledPin, LOW);     // turn the LED off
  delay(1000);                   // wait for a second
}
```

4.1.1.6 If-Statement

Syntax:

```
if (bedingung){
  …          // Code der ausgeführt wird wenn Bedingung = TRUE
} else {
  …          // Code der ausgeführt wird wenn Bedingung = FASLE
}
```

`bedingung` `bool` Ein Wahrheitsausdruck. Entweder ein Ergebnis eines Vergleichs, einer bool'schen Operation oder einer Berechnung. Alle Werte ungleich o werden als `TRUE` gewertet. Alle Werte gleich o als `FALSE`.

mögliche Vergleichsoperatoren sind:

`!=`	nicht gleich
`<`	Kleiner als
`<=`	Kleiner oder Gleich als
`==`	Gleich
`>`	Größer als
`>=`	Größer oder Gleich als

mögliche bool'sche Operatoren sind

`!`	logisches NICHT
`&&`	logisches UND
`||`	logisches ODER

4.1.2 Befehle

4.1.2.1 pinMode

Syntax:

```
pinMode(pinNo, mode);
```

Parameter:

`pinNo`	`int`	Nummer des Pins der eingestellt werden soll
`mode`	`int`	Modus für den PIN:
	`OUTPUT`	soll als Ausgang verwendet werden
	`INPUT`	soll als Eingang verwendet werden
	`INPUT_PULLUP`	soll als Eingang mit dem eingebauten Pullup-Widerstand verwendet werden.

4.1.2.2 digitalWrite

Syntax:

```
digitalWrite(pinNo, status);
```

Parameter:

`pinNo`	`int`	Nummer des Pins der geschrieben werden soll
`status`	`int`	Möglichkeiten für den PIN:
	`HIGH`	der PIN wird auf logisch 1 gesetzt (`+5V`), alternativ auch `1`
	`LOW`	der PIN wird auf logisch 0 gesetzt (0V), alternativ auch `0`

4.1.2.3 digitalRead

Syntax:

```
int value = digitalRead(pinNo);
```

Parameter:

`pinNo`	`int`	Nummer des Pins der gelesen werden soll
`value`	`int`	Möglichkeiten für das Ergebnis:
	`HIGH`	der PIN ist auf logisch 1 gesetzt (`+5V`), alternativ auch `1`
	`LOW`	der PIN ist auf logisch 0 gesetzt (0V), alternativ auch `0`

4.1.2.4 *pulseIn*

Syntax:

```
unsigned long duration = pulseIn(pinNo, value[, timeout]);
```

Parameter:

`pinNo` `int` Nummer des Pins der gelesen werden soll

`value` `int` Typ des Pulses, der gemessen werden soll:
`HIGH` ein Pulsdauer von `GND` auf `+5V` soll gemessen werden
`LOW` ein Pulsdauer von `+5V` auf `GND` soll gemessen werden

`timeout` `unsingned long` Timeout für das Warten auf den Anfang des Pulses in ms. Defaultwert: 1000ms

`duration` `unsingned long` Gemessene Pulsdauer am angegebenen Pin in μs

4.1.2.5 *analogWrite*

Syntax:

```
analogWrite(pinNo, value);
```

Parameter:

`pinNo` `int` Nummer des Pins der geschrieben werden soll (muss PWM-fähig sein)

`value` `int` Tastverhältnis der Pulsweitenmodulation (PWM). Grenzen sind:
`0` PWM ist auf immer aus
`255` PWM ist auf immer ein

4.1.2.6 *analogRead*

Syntax:

```
int value = analogRead(pinNo);
```

Parameter:

`pinNo` `int` Nummer des Pins der gelesen werden soll (Pin `A0` ist `0` , `A1` ist `1` usw.)

`value` `int` Wert des Einganges. Üblicherweise bei Arduino 10 Bit Auflösung. Das bedeutet der Eingang wird in $2^{10} = 1024$ Teile unterteilt. Wir bekommen also Werte zwischen 0 und 1023.

4.1.2.7 *delay*

Syntax:

```
delay(timeMs);
```

Parameter:

`timeMs`	`unsigned long`	Wartezeit des Programms in Millisekunden. Kein anderer Code wird ausgeführt!

Alternative:

```
delayMicroseconds(timeMs);
```

Parameter:

`timeMs`	`unsigned long`	Wartezeit des Programms in Mikrosekunden. Kein anderer Code wird ausgeführt!

4.1.2.8 *Millis*

Syntax:

```
time = millis();
```

Parameter:

`time`	`unsigned long`	Anzahl der Millisekunden, die seit dem Einschalten vergangen sind. Der maximale Datenbereich ist zu berücksichtigen. Die Zahl wiederholt sich etwa alle 50 Tage (fängt wieder bei 0 an).

4.1.2.9 *Micros*

Syntax:

```
time = micros();
```

Parameter:

`time`	`unsigned long`	Anzahl der Mikrosekunden, die seit dem Einschalten vergangen sind. Der maximale Datenbereich ist zu berücksichtigen. Die Zahl wiederholt sich etwa alle 70 Minuten (fängt wieder bei 0 an).

4.1.2.10 bitSet

Syntax:

```
bitSet(x, n);
```

Parameter:

x	byte	Variabe, in der ein Bit gesetzt werden soll
n	unisgend int	Bitnummer des Bits das gesetzt werden soll. o ist das least significant bit (LSB, das „rechteste“)

4.1.2.11 bitClear

Syntax:

```
bitClear(x, n);
```

Parameter:

x	byte	Variabe, in der ein Bit gelöscht werden soll
n	unisgend int	Bitnummer des Bits das gelöscht werden soll. o ist das least significant bit (LSB, das „rechteste“)

4.1.2.12 bitRead

Syntax:

```
int value = bitRead(x, n);
```

Parameter:

x	byte	Variabe, in der ein Bit gesetzt werden soll
n	unisgend int	Bitnummer des Bits das gesetzt werden soll. o ist das least significant bit (LSB, das „rechteste“)
value	int	Wert des angefragten Bits (0 oder 1)

4.1.2.13 bitWrite

Syntax:

```
bitWirte(x, n, b);
```

Parameter:

x	byte	Variabe, in der ein Bit geschrieben werden soll
n	unisgend int	Bitnummer des Bits das geschrieben werden soll. o ist das least significant bit (LSB, das „rechteste“)
b	int	Wert des zu schreibenden Bits (0 oder 1)

4.1.2.14 *shiftOut*

Syntax:

```
shiftOut (dataPin, clockPin, bitOrder, value);
```

Parameter:

`dataPin`	`int`	Pin- Nummer des Datenkanals (dort werden die Bits von `value` nach der Reihe hingeschrieben).
`clockPin`	`int`	Pin-Nummer des Clock-Kanals. Immer wenn ein neues Datenbit auf `dataPin` liegt wechselt `clockPin` von `0` auf `1`.
`bitOrder`	`int`	legt fest in welcher Reichenfolge der Wert von `value` auf den `dataPin` hinausgeschoben wird. Möglichkeiten sind: `MSBFIRST` Most significant bit first (das „linkste" zuerst) `LSBFIRST` Least significant bit first (das „rechteste" zuerst)
`value`	`byte`	Wert der auf die Pins seriell hinausgeschrieben wird

4.1.2.15 *tone*

Syntax:

```
tone(pin, freq[, dur]);
```

Parameter:

`pin`	`int`	Pin, auf dem der Ton ausgegeben werden soll (muss PWM fähig sein)
`freq`	`unisgend int`	Frequenz des auszugebenden Tones.
`dur`	`unsigned long`	Länge des erzeugten Tones. Optional. Bei Weglassen wird der Ton bis zum Aufruf vom `noTone` abgespielt.

4.1.2.16 *noTone*

Syntax:

```
noTone(pin);
```

Parameter:

`pin`	`int`	Pin, auf dem der Ton gestoppt werden soll

4.1.2.17 *isnan*

Syntax:

```
bool nan = isnan(value);
```

Parameter:

`value`	`float`	Wert, der auf eine Nummer überprüft werden soll
`nan`	`bool`	Liefert `true`, wenn `value` keine Zahl ist (not a number, oder `nan`)

4.1.2.18 attachInterrupt

Syntax:

```
attachInterrupt(intNumber, ISR, mode);
```

Parameter:

`intNumber` `int` Interrupt Nummer des Interrupts der einer Funktion zugewiesen werden soll

`ISR` Name der aufzurufenden Funktion

`mode` `int` legt fest auf welche Ereignisse des Eingangs reagiert werden soll

- `LOW` Triggert den Interrupt wenn der Pin `LOW` ist.
- `CHANGE` Triggert den Interrupt wenn der Pin wechselt.
- `RISING` Triggert den Interrupt wenn der Pin von `LOW` auf `HIGH` geht
- `FALLING` Triggert den Interrupt wenn der Pin von `HIGH` auf `LOW` geht.

4.1.2.19 detachInterrupt

Syntax:

```
detachInterrupt(intNumber);
```

Parameter:

`intNumber` `int` Interrupt Nummer des Interrupts der aufgehoben werden soll.

4.1.2.20 digitalPinToInterrupt

Syntax:

```
int intNumber = digitalPinToInterrupt(pin);
```

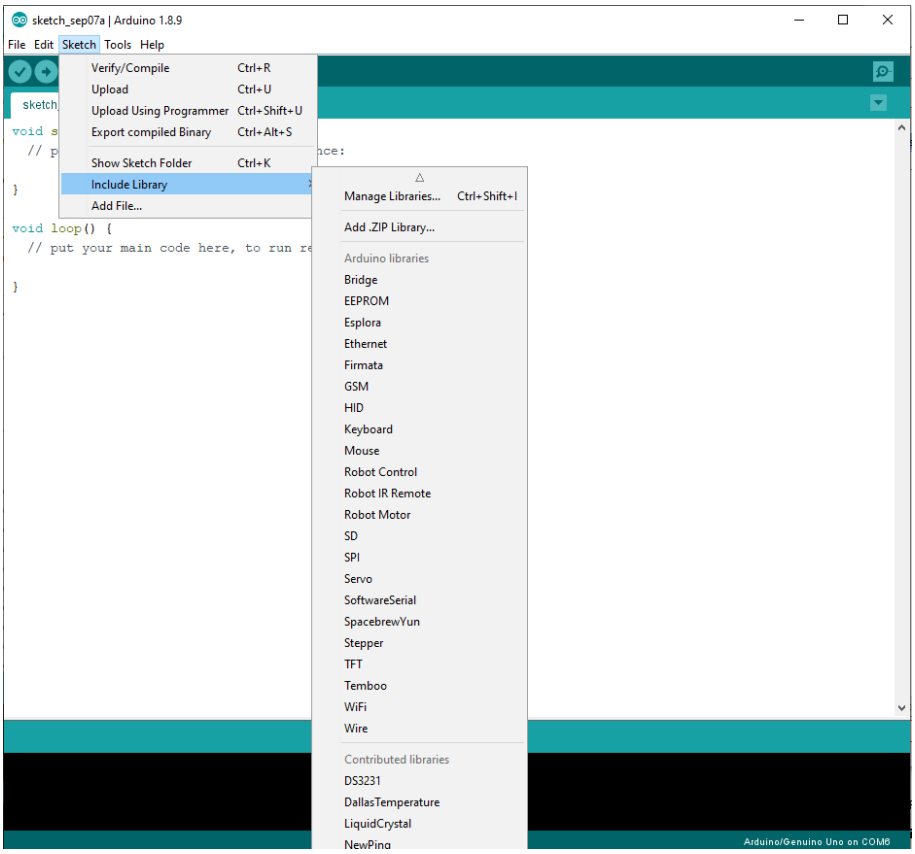
Parameter:

`pin` `int` Pin, dem ein Interrupt zugewiesen werden soll. Daraus wird die entsprechende Interruptnummer, die man bei attachInterrupt und detachInterrupt benötigt, je nach verwendetem Board berechnet.

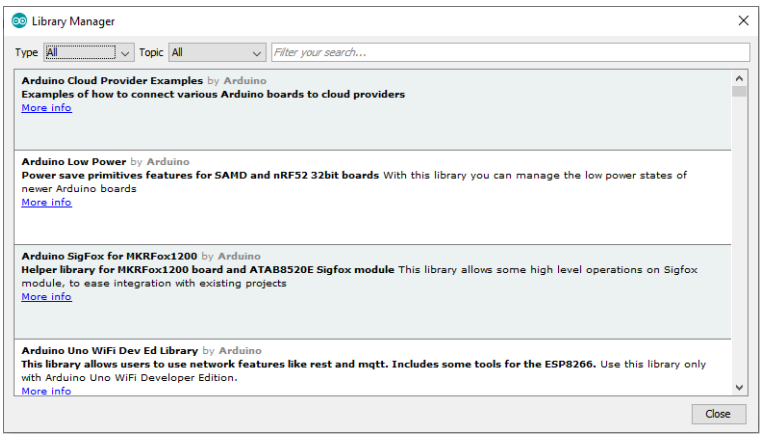
`intNumber` `int` Interrupt Nummer des Interrupts der einer Funktion zugewiesen werden soll

4.1.3 Hinzufügen von Libraries

Manchmal ist es notwendig, fremden Code in sein Projekt einzubinden. Das macht durchaus Sinn, denn oftmals muss ein Sensor etc. speziell angesprochen werden und sehr oft haben sich Leute, die sich mit diesem Thema intensiver beschäftigt haben, sogenannte Bibliotheken (Libraries) dazu geschrieben. Diese beinhalten den notwendigen Code, um eine gewisse Thematik abzudecken. Im Folgenden installieren wir eine Library, welche dazu dient, Infrarot-Signale von einer Fernbedienung zu empfangen. Wir gehen dabei Schritt für Schritt vor. Oftmals findet man alle notwendigen Libraries genau auf diesem Weg.

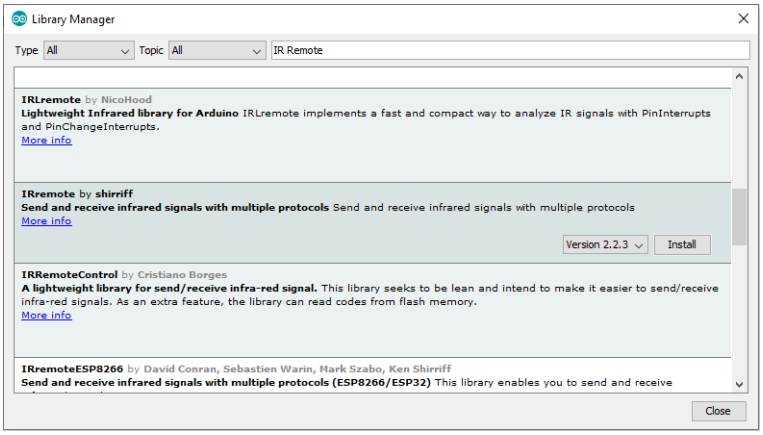


Ist die Arduino IDE gestartet, findet man im Menü „Sketch" den Eintrag „Include Library". Dort werden schon viele installierte Libraries aufgelistet. Ganz oben ist gibt es jedoch den Eintrag „Manage Libraries..." – diesen wählen wir aus. Dann öffnet sich der Library Manger.

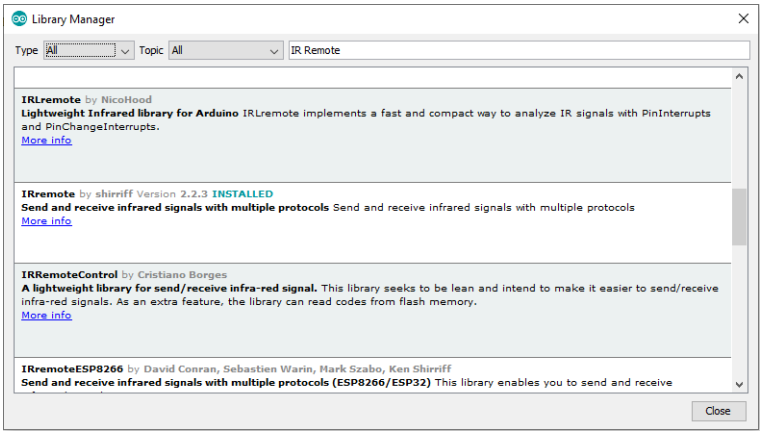


Dort kann man dann versuchen, nach passenden Libraries zu suchen. Wir wollen ja IR-Signale empfangen, also geben wir dort als Suche einmal „IR Remote" ein. Das schränkt die Liste schon

erheblich ein. Aus diesem Katalog kann man dann eine geeignet erscheinende Bibliothek auswählen. Wir suchen uns die Library „IRremote" von einem gewissen „shirriff" aus – die habe ich nämlich immer verwendet:



Es erscheint dort ein „Install"-Knopf. Drückt man diesen, wird die ausgewählte Version der Library installiert. In der Liste der Libraries wird dies auch entsprechend gekennzeichnet.



Und damit sind wir auch schon fertig. Die Bibliothek ist auf unserm Gerät und kann mit einem `include`-Statement verwendet werden.

4.1.4 Wichtige Objekte

4.1.4.1 Serial-Objekt

`Serial.begin(speed)` Initialisierung der seriellen Schnittstelle

`speed`	`long`	Übertragungsrate
	übliche Raten:	`300`, `1200`, `2400`, `4800`, `9600`, `19200`, `38400`, `57600`, `74880`, `115200`

`Serial.print(text[,format])` Ausgabe einer Zeichenkette auf der seriellen Schnittstelle.

`text` Hier können verschiedenste Daten verwendet werden, die danach möglichst als Text auf der seriellen Schnittstelle erscheinen sollen.

`format` Man kann angeben wie Zahlen geschrieben werden:

`BIN`	Ausgabe als Binärzahl
`OCT`	Ausgabe als Oktalzahl
`DEC`	Ausgabe als Dezimalzahl (default)
`HEX`	Ausgabe als Hexadezimalzahl

`Serial.println(text[,format])` Ausgabe einer Zeichenkette auf der seriellen Schnittstelle. Abgeschlossen wird mit einem Carriage Return (`'\r'`) und einem New Line (`'\n'`). (Also praktisch mit einem „Enter").

`text` siehe `print()`

`format` siehe `print()`

`Serial.write(text)` Ausgabe einer Zeichenkette auf der seriellen Schnittstelle.

`text` Im Unterschied zur `print`-Methode wird dabei eine Zahl nicht interpretiert, sondern ein entsprechendes Zeichen gesendet. Ein Wert von 123 ergibt zum Beispiel genau ein Zeichen, nämlich das aus der ASCII-Tabelle mit der Nummer 123 (`'{'`). Und nicht die Zeichenfolge `"123"`.

`bool Serial.available()` Gibt die Anzahl der Zeichen im Empfangs-Buffer der Seriellen Schnittstelle zurück. Ein Wert von `0` bedeutet es gibt nichts zu lesen. Der maximale Empfangs-Buffer ist `64` Byte.

`char Serial.read()` Auslesen eines Zeichens von der seriellen Schnittstelle. Es wird das erste Zeichen als Datentyp `int` zurückgegeben. Wird oft in Verbindung mit der `available`-Methode eingesetzt. Wird öfter hintereinander die `read`-Methode aufgerufen, erhält man immer das „nächste" Zeichen. Ein Zeichen gilt also als gelesen und wird aus dem Empfangs-Buffer der seriellen Schnittstelle gelöscht.

`char Serial.peek()` Auslesen eins Zeichens von der seriellen Schnittstelle. Es wird das erste Zeichen als Datentyp `int` zurückgegeben. Wird öfter hintereinander die `peek`-Methode aufgerufen, erhält man immer das „gleiche" Zeichen. Ein Zeichen gilt also als NICHT gelesen und bleibt im Empfangs-Buffer der seriellen Schnittstelle erhalten.

4.1.4.2 SoftwareSerial

Wenn keine Hardware UART zur Verfügung steht, kann auch ein Software UART definiert werden. Notwendig ist die Library „`SoftwareSerial.h`". Das Objekt hat die gleichen Methoden wie das normalen Serial-Objekt.

```
SoftwareSerial mySerial(RX_PIN, TX_PIN);
```

Parameter:

`RX_PIN`	`int`	Pinnummer für den Receive-Pin
`TX_PIN`	`int`	Pinnummer für den Transmit-Pin

4.1.4.3 EEPROM

Wird automatisch mit dem Einbinden der Bibliothek „`EEPROM.h`" verfügbar:

```
#include <EEPROM.h>
```

`EEPROM.read(add)` Lesen des Bytes an der angegeben Adresse.

`add`	`int`	Adresse beginn mit `0`

`EEPROM.write(add,val)` Schreiben des Bytes an der angegeben Adresse.

`add`	`int`	Adresse beginn mit `0`
`val`	`byte`	Zu schreibender Wert

`EEPROM.update(add,val)` Schreiben des Bytes an der angegeben Adresse. Der Schreibvorgang wird nur ausgeführt, wenn auch wirklich ein anderer Wert drinnen stehet – um Schreibzyklen zu sparen.

`add`	`int`	Adresse beginn mit `0`
`val`	`byte`	Zu schreibender Wert

`EEPROM.get(add,dat)` Lesen eines längeren Datentyps beginnend bei der angegebenen Adresse.

`add`	`int`	Adresse beginn mit `0`
`dat`	`*`	`Z`u schreibender Wert, kann irgendwas, z.B. ein `float` sein, oder sogar ein ganzer `struct`.

`EEPROM.put(add,dat)` Schreiben eines längeren Datentyps beginnend bei der angegebenen Adresse. Jedes Byte wird intern mit `EEPROM.update` beschrieben, um Schreibzyklen auf das EEPROM zu sparen.

`add`	`int`	Adresse beginn mit `0`
`dat`	`*`	`Z`u schreibender Wert, kann irgendwas, z.B. ein `float` sein, oder sogar ein ganzer `struct`.

`int EEPROM.length()` Liefert die Länge des EEPROM zurück. Verschiedene Boards haben verschiedene EEPROM-Speicherplätze.

4.1.5 ASCII-Tabelle (Auszug)

DEC	Zeichen
0	'\0' NULL
1	
2	
3	
4	
5	
6	
7	
8	
9	'\t' TAB
10	'\n' Line
11	
12	
13	'\r' Carr.
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	

DEC	Zeichen
32	' ' SAPCE
33	'!'
34	'"'
35	'#'
36	'$'
37	'%'
38	'&'
39	'''
40	'('
41	')'
42	'*'
43	'+'
44	','
45	'-'
46	'.'
47	'/'
48	'0'
49	'1'
50	'2'
51	'3'
52	'4'
53	'5'
54	'6'
55	'7'
56	'8'
57	'9'
58	':'
59	';'
60	'<'
61	'='
62	'>'
63	'?'

DEC	Zeichen
64	'@'
65	'A'
66	'B'
67	'C'
68	'D'
69	'E'
70	'F'
71	'G'
72	'H'
73	'I'
74	'J'
75	'K'
76	'L'
77	'M'
78	'N'
79	'O'
80	'P'
81	'Q'
82	'R'
83	'S'
84	'T'
85	'U'
86	'V'
87	'W'
88	'X'
89	'Y'
90	'Z'
91	'['
92	'\'
93	']'
94	'^'
95	'_'

DEC	Zeichen
96	'`'
97	'a'
98	'b'
99	'c'
100	'd'
101	'e'
102	'f'
103	'g'
104	'h'
105	'i'
106	'j'
107	'k'
108	'l'
109	'm'
110	'n'
111	'o'
112	'p'
113	'q'
114	'r'
115	's'
116	't'
117	'u'
118	'v'
119	'w'
120	'x'
121	'y'
122	'z'
123	'{'
124	'|'
125	'}'
126	'~'
127	

4.2 Widerstandsfarbcode

Kennzeichnung mit 4 Ringen:

Farbe	Widerstanswert			Toleranz
	1. Ring (Zehner)	2. Ring (Einer)	3. Ring (Muliplikator)	5. Ring
keine				$\pm 20\%$
Silber			10^{-2}	$\pm 10\%$
Gold			10^{-1}	$\pm 5\%$
Schwarz		0	10^{0}	
Braun	1	1	10^{1}	$\pm 1\%$
Rot	2	2	10^{2}	$\pm 2\%$
Orange	3	3	10^{3}	
Gelb	4	4	10^{4}	
Grün	5	5	10^{5}	$\pm 0{,}5\%$
Blau	6	6	10^{6}	$\pm 0{,}25\%$
Violett	7	7		$\pm 0{,}1\%$
Grau	8	8		$\pm 0{,}05\%$
Weiß	9	9		

Kennzeichnung mit 5 oder 6 Ringen:

Farbe	Widerstandswert				Toleranz	Temp. Koeffizent
	1. Ring (Hunderter)	2. Ring (Zehner)	3. Ring (Einer)	4. Ring (Muliplikator)	5. Ring	6. Ring
Silber				10^{-2}		
Gold				10^{-1}		
Schwarz		0	0	10^{0}		$200 \cdot 10^{-6}\ K^{-1}$
Braun	1	1	1	10^{1}	$\pm 1\%$	$100 \cdot 10^{-6}\ K^{-1}$
Rot	2	2	2	10^{2}	$\pm 2\%$	$50 \cdot 10^{-6}\ K^{-1}$
Orange	3	3	3	10^{3}		$15 \cdot 10^{-6}\ K^{-1}$
Gelb	4	4	4	10^{4}		$25 \cdot 10^{-6}\ K^{-1}$
Grün	5	5	5	10^{5}	$\pm 0{,}5\%$	
Blau	6	6	6	10^{6}	$\pm 0{,}25\%$	$10 \cdot 10^{-6}\ K^{-1}$
Violett	7	7	7		$\pm 0{,}1\%$	$5 \cdot 10^{-6}\ K^{-1}$
Grau	8	8	8		$\pm 0{,}05\%$	
Weiß	9	9	9			

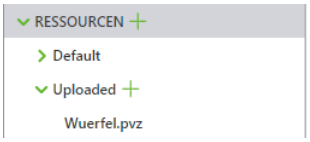
4.3 Vuforia Studio Referenz

4.3.1 Vorgehen bei der Projekterstellung in Vuforia Studio

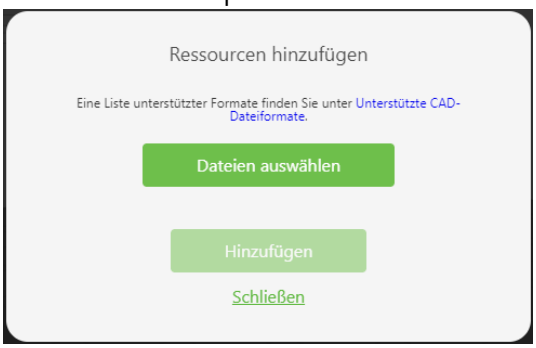
1. Projekt anlegen:
   Projektname Konvention: **KLASSE_NACHNAME_VORNAME<_HOLO>**
   Experience-URL: **https://example.twx.htl.schule:8443**
2. Ziel hinzufügen:
   Bild-Ziel: Versucht in der Realität das Bild zu finden
   Modell-Ziel: Versucht in der Realität das 3D-Modell zu finden
   Räumliches Ziel: Ebene Fläche, die ich in der Realität auswählen muss
   ThingMark: 2D-Code der in der Realität gesucht wird.
3. Virtuelle Objekte hinzufügen
4. Test des Experience-Services
5. Vorschau
6. Veröffentlichung

4.3.2 Importieren eins Modells

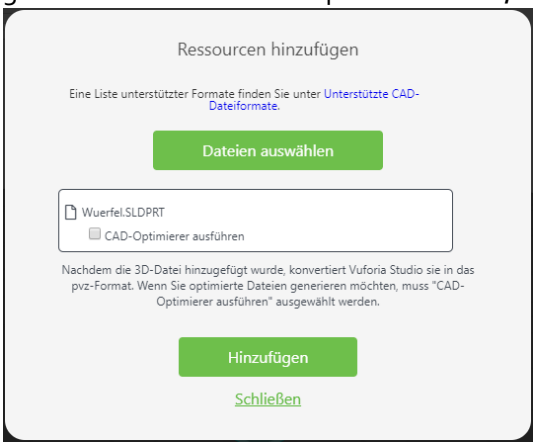
1. Drücken des „+"-Symbols links unten bei den Ressourcen



2. Auswahl der zu importierenden CAD-Datei



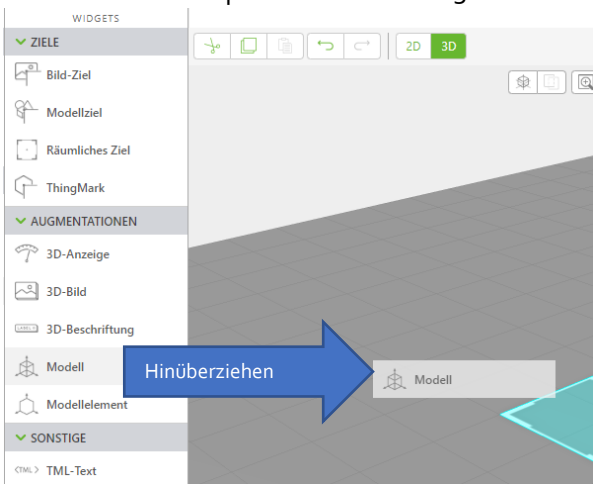
3. Bei großen detaillierten Modellen kann der CAD-Optimierer verwendet werden. Dieser versucht das CAD-Modell zu vereinfachen, ohne die Geometrie allzu sehr zu beeinflussen. Es gibt dann nicht nur eine importierte Datei, sondern 3 in verschiedenen Detailgrad.



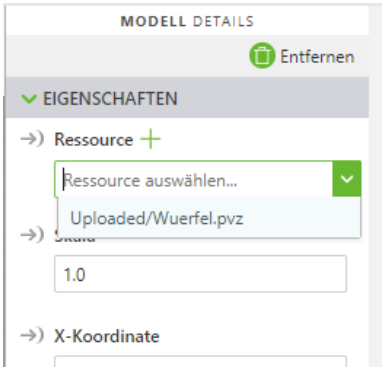
4. Import: Die Datei wird in eine pvz-Datei konvertiert und im Projektverzeichnis abgelegt.

4.3.3 Hinzufügen eines Modells

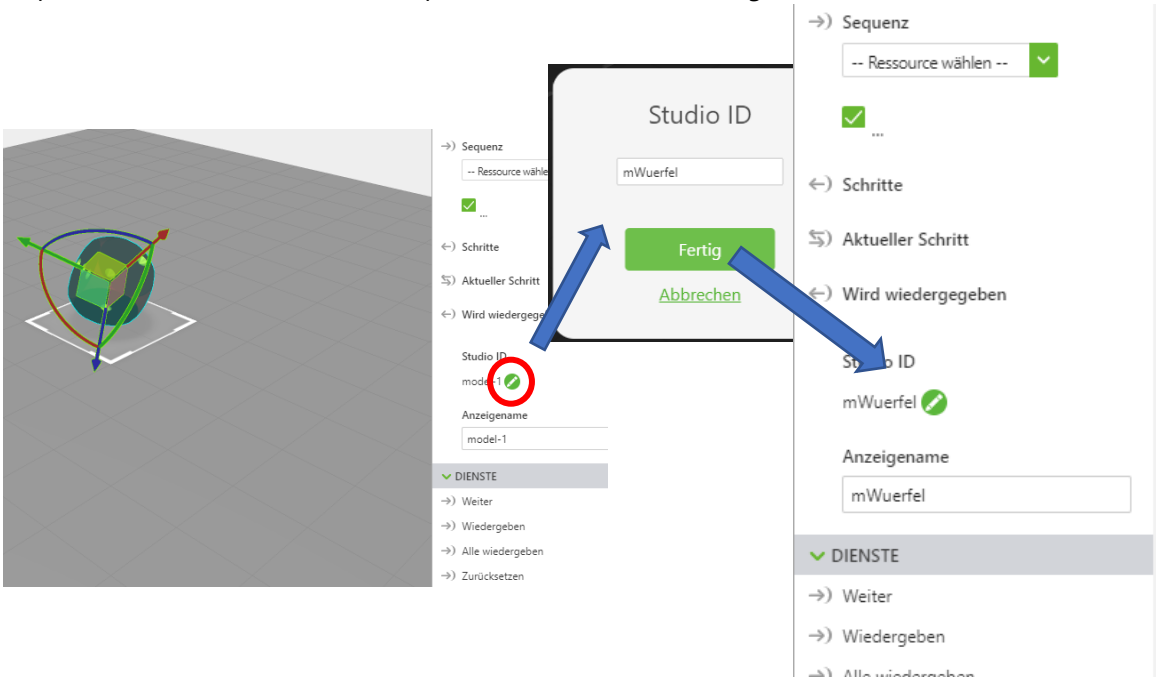
1. Ziehen des Modellplatzhalters in den 3D-Zeichenbereich (3D-Canvas)



2. Auswahl der zu verwendeten Ressource



3. Anpassen der Position und Drehung abhängig von der Ressource.
4. Anpassen der Studio-ID auf einen „passenden" Name (Achtung! Keine Umlaute!)



4.3.4 Ionicons -Codes für Symbole

	ion-minus-round		ion-plus-round
	ion-chevron-left		ion-chevron-right
	ion-arrow-return-right		ion-arrow-return-left

4.4 MQTT-Broker Mosquitto auf Raspberry installieren

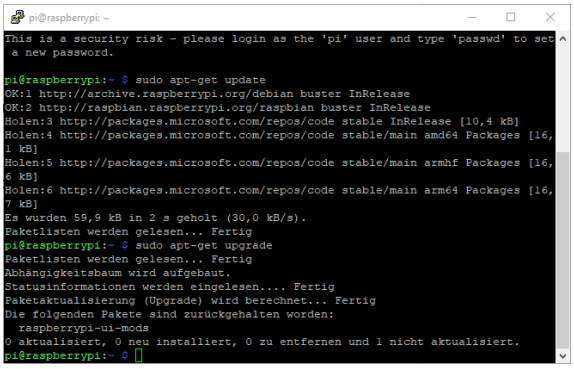
Video-Link x01: https://youtu.be/ID1voUOEdyc

In diesem Lehrgang verwenden wir als MQTT-Broker einen RaspberryPi-Einplatinencomputer. Voraussetzung für das Gelingen der folgenden Prozedur ist die Installation eines RaspberryOS und für den Zugriff die Verwendung von PuTTY[54].

Der hier verwendete MQTT-Broker „Mosquitto" ist Teil des Raspberry-Repository. Das ist eine Bibliothek die Software enthält, ganz ähnlich einem Play-Store, nur nicht so komfortabel zu bedienen. Zunächst müssen wir den Kataloginhalt der Bibliothek auf den aktuellen Stand bringen. Dazu öffnen wir mit PuTTY eine SSH-Verbindung und geben dort als Befehl

```
sudo apt-get update
```

ein[55].



Danach machen wir ein Upgrade auf die aktuellen Pakete. Dadurch werden alle installierten Pakete auf den neuesten Stand gebracht. Der entsprechende Befehl lautet

```
sudo apt-get upgrade
```

Damit sind wir also mit unserer Installation aktuell. Nun folgt die eigentliche Installation von Mosquitto. Dies installieren wir mit Hilfe dass Befehles

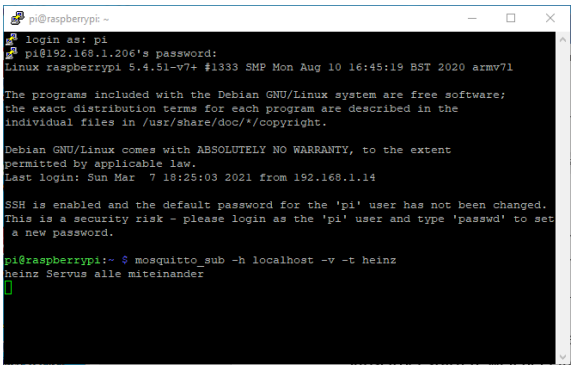
```
sudo apt-get install -y mosquitto mosquitto-clients
```

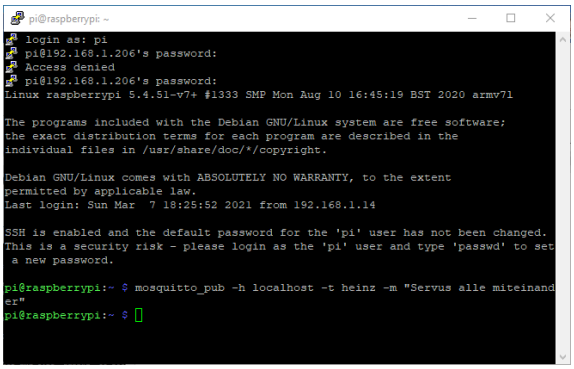
Danach sollte der Mosquitto-Server bereits laufen. Das können wir testen, indem wir einen MQTT-Subscirber starten, welcher ein gewisses Thema abonniert. Das kann mit dem Befehl passieren (Subscribed auf den Kanal `heinz`):

```
mosquitto_sub -h localhost -v -t heinz
```

Öffnet man ein zweites PuTTY-Fenster und ruft dort eine Veröffentlichung zum Thema `heinz` auf, so sollte dies im ersten Fenster erscheinen.

```
mosquitto_pub -h localhost -t heinz -m "Servus alle miteinander"
```





[54] Beide Dinge sind im Skriptum über Datenbanken beschrieben.

[55] Das bedeutet „Superuser do" (`sudo`) und Advanced Packaging Tool (`apt-get`), bitte hole die aktuelle Paketliste (`update`). Bei den anderen Befehlen ist es sehr ähnlich.

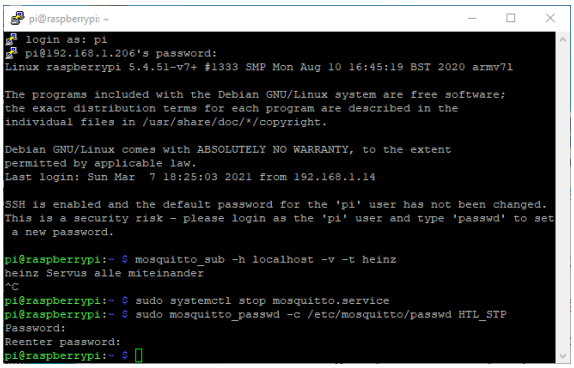
Die so installierte Konfiguration ist aber unsicher. Im Moment kann jeder Teilnehmer subscirben und veröffentlichen. Deswegen setzen wir noch ein Passwort für den Zugriff. Dazu stoppen wir zunächst den Server mit dem Befehl

```
sudo systemctl stop mosquitto.service
```

Jetzt müssen wir zunächst eine Passwort-Datei anlegen. Mosquitto unterstützt uns mit einem Dienstprogramm und wir können mit folgendem Befehl eine solche Datei anlegen (Dateiname ist hier `passwd`):

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd HTL_STP
```

Wobei `HTL_STP` bereits der Benutzername ist. Wir werden nun aufgefordert ein Passwort einzugeben (Achtung es erfolgt keine visuelle Rückmeldung der Eingabe). Ich wähle hier `USERinnovativ`.



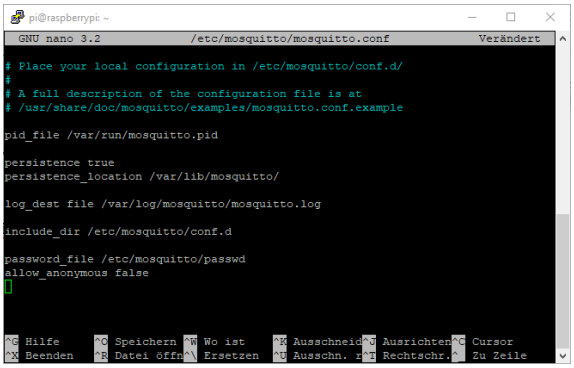
Nun müssen wir noch die Konfigurationsdatei umändern. Dazu öffnen wir diese mit dem Texteditor `nano`.

```
sudo nano /etc/mosquitto/mosquitto.conf
```

Dort fügen wir die Zeilen

```
password_file /etc/mosquitto/passwd
allow_anonymous false
```

hinzu.

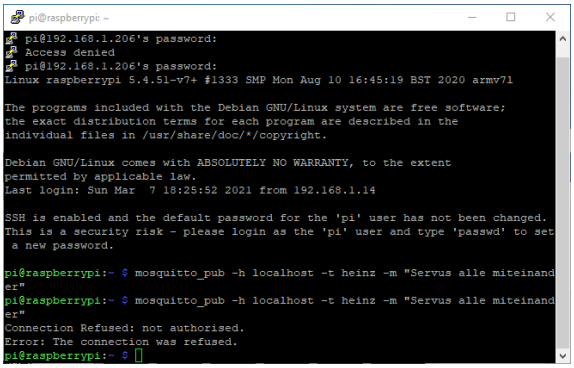


Mit `Strg+X` und `J` beenden und speichern wir die Datei.

Nun müssen wir den Server nur mehr wieder starten. Das geschieht mit dem Befehl

```
sudo systemctl start mosquitto.service
```

Als Test können wir versuchen wieder – genau wie vorher eine Meldung abzusetzen. Wir merken: Das geht jetzt nicht mehr. Der Server ist bereit.



4.5 Die ESP-Programme für IoT

4.5.1 Befehle für den MQTT-Gateway

Befehle zum MQTT-Gateway werden seriell übertragen. Dabei gibt es folgende Keywords und Responses. Die Responses hängen dabei von der eingestellten Informationsdichte ab. Es gibt 4 verschiedene Level: Fehler, Warnung, Information und Debug. Jedes Level kann einzeln abschalten werden. Die Farben in den erwähnten Responses gibt an, welcher Level aktiv sein muss, um den Response zu sehen.

Fehler: rot Warnung: orange Information: grün Debug: blau

Befehl	mögliche Antworten	Bedeutung
?	!	Überprüfung, ob die ESP-Verbindung funktioniert
reset	Resetting Module ... READY	Rücksetzen des Moduls, alle Daten werden gelöscht.
wifi	CON	Zeigt an, ob mit dem Wifi-Netzwerk verbunden ist.
	DIS	
mqtt	CON	Zeigt an, ob mit dem MQTT-Broker verbunden ist.
	DIS	
nowifi	Stopping Wifi. OK	Abdrehen der Wifi-Verbindung, selbst bei schon übertragenen SSID und Passwort
wifion	Starting Wifi. OK	Aktivieren der Wifi-Verbindung
startdebug	-	Aktiviert die Ausgabe des Levels debug (Bit 7)
stopdebug	-	Deaktiviert die Ausgabe des Levels debug (Bit 7)
startinfo	-	Aktiviert die Ausgabe des Levels info (Bit 6)
stopinfo	-	Deaktiviert die Ausgabe des Levels info (Bit 6)
startwarn	-	Aktiviert die Ausgabe des Levels warning (Bit 5)
stopwarn	-	Deaktiviert die Ausgabe des Levels warning (Bit 5)
starterr	-	Aktiviert die Ausgabe des Levels error (Bit 4)
stoperr	-	Deaktiviert die Ausgabe des Levels error (Bit 4)
setverbose <verb>	Set Verbose to : <verb> OK	Setzt den Informationsdichtelevel (Alternative zu den start/stop-Befehlen.
	Missing Argument. ERR	
getverbose	ERROR WARN. INFO. DEBUG (<verb>) X X X X	Liefert den momentanen Informationsdichtelevel zurück.
setbaud <rate>	Changing Baud-Rate to: '<rate>' OK	Wechselt die Baud-Rate des Moduls. Beim Reboot ist die Baud-Rate immer 9600. Gültig sind die üblichen Werte zwischen 300 und 115200.
	Invalid Baud-Rate. ERR	
	Missing Argument. ERR	
setblocktime <time>	Minimum time between Requests: '<time>' OK	Blockierzeit: Setzt die minimale Wartezeit bis wieder ein neuer Wert übertragen werden kann. Gültig ist 0 bis 30 Sekunden.
	Invalid Time given (valid: 0 - 30). ERR	
	Missing Argument. ERR	
getblocktime	Minimum time between Requests: '<time>' OK	Liefert die momentan eingestellte Zeit für die Blockierzeit in Sekunden.
setresendtime <time>	Time for resending values: '<time>' OK	Wiederholzeit: Setzt die Zeitspanne wann die Werte wieder an den MQTT-Broker gesendet werden ohne dass es einen neunen Wert gibt.
	Invalid Time given (valid: 0 - 3600). ERR	
	Missing Argument. ERR	
getresendtime	Time for resending values: '<time>' OK	Liefert die momentan eingestellte Zeit für die Wiederholzeit in Sekunden.
setssid <ssid>	Set WifiSSID: '<ssid>' OK	Setzt die Wifi-SSID. Wird <ssid> weggelassen wird die SSID gelöscht.
getssid	Get WifiSSID: '<ssid>' OK	Liefert die aktuell gesetzte SSID.
setpass <pw>	Set WifiPass: '<pw>' OK	Setzt das Wifi-Passwort. Wird <pw> weggelassen wird das Passwort gelöscht.
getpass	Get WifiPAss: '<pw>' OK	Liefert das aktuell gesetzte Wifi-Passwort.
sethost <ip>	Set Host: '<ip>' OK	Setzt die IP-Adresse des MQTT-Brokers.
gethost	Get Host: '<ip>' OK	Liefert die momentan gesetzte IP-Adresse des MQTT-Brokers.
setusername <name>	Set Username: '<name>' OK	Setzt den Username für die Verbindung zum, MQTT-Broker.

getusername	Get Username: '<name>' OK	Liefert den momentan gesetzten Username des MQTT-Brokers.
setuserpass <pw>	Set Userpass: '<pw>' OK	Setzt das Userpasswort für die Verbindung zum, MQTT-Broker.
getuserpass	Get Userpass: '<pw>' OK	Liefert das momentan gesetzten Userpasswort des MQTT-Brokers.
setsubsribe <#> <name>	Set Subscribe Topic #<#> : '<name>' OK	Abonniert ein Topic mit dem angegeben Namen. Es können bis zu 50 Topics abonniert werden, das bedeutet <#> muss zwischen 0 und 49 liegen.
	Invalid Subscribe Number. ERR	
	Missing Argument. ERR	
getsubsribe <#>	Get Subscribe Topic #<#> : '<name>' OK	Liefert das abonnierte Topic mit der angegebenen Nummer.
	Invalid Subscribe Number. ERR	
	Missing Argument. ERR	
setpropname <#> <name>	Set Property Name #<#> : '<name>' OK	Setzt den Namen des allgemeinen Topics mit der Nummer <#>. Es können 50 allgemeine Topics angegeben werden, das bedeutet <#> muss zwischen 0 und 49 liegen.
	Invalid Property Number. ERR	
	Missing Argument. ERR	
getpropname <#>	Get Property Name #<#> : '<name>' OK	Liefert den Namen des allgemeinen Topics mit der Nummer <#>.
	Invalid Property Number. ERR	
	Missing Argument. ERR	
setpropval <#> <val>	Set Property Value #<#> : '<val>' OK	Setzt einen neuen Wert für das allgemeine Topic mit der Nummer <#>. Wenn sich der neue Wert vom zuletzt Übertragenen unterschiedet so wird sofort eine Übertragung zum MQTT-Broker angestoßen.
	Invalid Property Number. ERR	
	Missing Argument. ERR	
getpropval <#>	Get Property Value #<#> : '<val>' OK	Liefert den letzten empfangenen Wert des angegebenen allgemeinen Topics mit der Nummer <#>.
	Invalid Property Number. ERR	
	Missing Argument. ERR	
setvaluename <#> <name>	Set Value Name #<#> : '<name>' OK	Setzt den Namen des Value-Topics mit der Nummer <#>. Es können 50 Value-Topics angegeben werden, das bedeutet <#> muss zwischen 0 und 49 liegen.
	Invalid Value Number. ERR	
	Missing Argument. ERR	
getvaluename <#>	Get Value Name #<#> : '<name>' OK	Liefert den Namen des Value-Topics mit der Nummer <#>.
	Invalid Value Number. ERR	
	Missing Argument. ERR	
setvalth <#> <th>	Set Value Threshold #<#> : '<th>' OK	Setzt die Übertragungsschwelle des Value-Topics mit der Nummer <#>. Wird ein neuer Wert empfangen so wird dieser mit dem letzten gesendeten verglichen. Nur wenn der Unterschied größer als der eingestellte Wert ist wird der neue Wert sofort auf den MQTT-Server veröffentlicht. Default-Wert ist 0,1.
	Invalid Value Number. ERR	
	Missing Argument. ERR	
getvalth <#>	Get Value Threshold #<#> : '<th>' OK	Liefert die Übertragungsschwelle des Value-Topics mit der Nummer <#>.
	Invalid Value Number. ERR	
	Missing Argument. ERR	
setval <#> <val>	Set Value #<#> : '<val>' OK	Setzt einen neuen Wert für das Value-Topic mit der Nummer <#>. Wenn sich der neue Wert vom zuletzt Übertragenen um mehr als die eingestellte Übertragungsschwelle unterschiedet so wird sofort eine Übertragung zum MQTT-Broker angestoßen.
	Invalid Value Number. ERR	
	Missing Argument. ERR	
getval <#>	Get Value #<#> : '<val>' OK	Liefert den letzten empfangenen Wert des angegebenen Value- Topics mit der Nummer <#>.
	Invalid Value Number. ERR	
	Missing Argument. ERR	
setboolname <#> <name>	Set Bool Name #<#> : '<name>' OK	Setzt den Namen des Boole'schen-Topics mit der Nummer <#>. Es können 50 Boole-Topics angegeben werden, das bedeutet <#> muss zwischen 0 und 49 liegen.
	Invalid Bool Number. ERR	

	Missing Argument. ERR	
getboolname <#>	Get Bool Name #<#> : '<name>' OK	Liefert den Namen des Boole-Topics mit der Nummer <#>.
	Invalid Bool Number. ERR	
	Missing Argument. ERR	
setboolval <#> <val>	Set Bool Value #<#> : '<val>' OK	Setzt einen neuen Wert für das Boole-Topic mit der Nummer <#>. Wenn sich der neue Wert vom zuletzt Übertragenen unterscheidet so wird sofort eine Übertragung zum MQTT-Broker angestoßen.
	Invalid Bool Number. ERR	
	Missing Argument. ERR	
getboolval <#>	Get Bool Value #<#> : '<val>' OK	Liefert den letzten empfangenen Wert des angegebenen Boole- Topics mit der Nummer <#>.
	Invalid Bool Number. ERR	
	Missing Argument. ERR	

4.5.2 Befehle für den Thingworx-Gateway

Befehle zum Thingworx-Gateway werden seriell übertragen. Dabei gibt es folgende Keywords und Responses. Die Responses hängen dabei von der eingestellten Informationsdichte ab. Es gibt 4 verschiedene Level: Fehler, Warnung, Information und Debug. Jedes Level kann einzeln abschalten werden. Die Farben in den erwähnten Responses gibt an, welcher Level aktiv sein muss, um den Response zu sehen.

Fehler: rot Warnung: orange Information: grün Debug: blau

Befehl	mögliche Antworten	Bedeutung
?	!	Überprüfung, ob die ESP-Verbindung funktioniert
reset	Resetting Module ... READY	Rücksetzen des Moduls, alle Daten werden gelöscht.
wifi	CON DIS	Zeigt an, ob mit dem Wifi-Netzwerk verbunden ist.
twx	CON DIS	Zeigt an, ob mit dem Thingworx-Server verbunden ist.
nowifi	Stopping Wifi. OK	Abdrehen der Wifi-Verbindung, selbst bei schon übertragenen SSID und Passwort
wifion	Starting Wifi. OK	Aktivieren der Wifi-Verbindung
startdebug	-	Aktiviert die Ausgabe des Levels debug (Bit 7)
stopdebug	-	Deaktiviert die Ausgabe des Levels debug (Bit 7)
startinfo	-	Aktiviert die Ausgabe des Levels info (Bit 6)
stopinfo	-	Deaktiviert die Ausgabe des Levels info (Bit 6)
startwarn	-	Aktiviert die Ausgabe des Levels warning (Bit 5)
stopwarn	-	Deaktiviert die Ausgabe des Levels warning (Bit 5)
starterr	-	Aktiviert die Ausgabe des Levels error (Bit 4)
stoperr	-	Deaktiviert die Ausgabe des Levels error (Bit 4)
setverbose <verb>	Set Verbose to : <verb> OK Missing Argument. ERR	Setzt den Informationsdichtelevel (Alternative zu den start/stop-Befehlen.
getverbose	ERROR WARN. INFO. DEBUG (<verb>) X X X X	Liefert den momentanen Informationsdichtelevel zurück.
setbaud <rate>	Changing Baud-Rate to: '<rate>' OK Invalid Baud-Rate. ERR Missing Argument. ERR	Wechselt die Baud-Rate des Moduls. Beim Reboot ist die Baud-Rate immer 9600. Gültig sind die üblichen Werte zwischen 300 und 115200.
setblocktime <time>	Minimum time between Requests: '<time>' OK Invalid Time given (valid: 0 - 30). ERR Missing Argument. ERR	Blockierzeit: Setzt die minimale Wartezeit bis wieder ein neuer Wert übertragen werden kann. Gültig ist 0 bis 30 Sekunden.
getblocktime	Minimum time between Requests: '<time>' OK	Liefert die momentan eingestellte Zeit für die Blockierzeit in Sekunden.
setresendtime <time>	Time for resending values: '<time>' OK Invalid Time given (valid: 0 - 3600). ERR Missing Argument. ERR	Wiederholzeit: Setzt die Zeitspanne wann die Werte wieder an den MQTT-Broker gesendet werden ohne dass es einen neunen Wert gibt.
getresendtime	Time for resending values: '<time>' OK	Liefert die momentan eingestellte Zeit für die Wiederholzeit in Sekunden.
setssid <ssid>	Set WifiSSID: '<ssid>' OK	Setzt die Wifi-SSID. Wird <ssid> weggelassen wird die SSID gelöscht.
getssid	Get WifiSSID: '<ssid>' OK	Liefert die aktuell gesetzte SSID.
setpass <pw>	Set WifiPass: '<pw>' OK	Setzt das Wifi-Passwort. Wird <pw> weggelassen wird das Passwort gelöscht.
getpass	Get WifiPAss: '<pw>' OK	Liefert das aktuell gesetzte Wifi-Passwort.
sethost <ip>	Set Host: '<ip>' OK	Setzt die URL des Thingworx-Servers.
gethost	Get Host: '<ip>' OK	Liefert die momentan gesetzte URL des Thingworx-Servers.
setappkey <name>	Set App Key: '<name>' OK	Setzt des Application Keys für den Zugriff auf das Thing.
getappkey	Get App Key: '<name>' OK	Liefert den momentan gesetzten Application Key für den Zugriff auf das Thing-.

setthing <thing>	Set Thing: '<thing>' OK	Setzt den Name des Things, welches die zu verändernden Properties beinhaltet.
getthing	Get Thing: '<thing>' OK	Liefert das momentan gesetzte Thing welches die zu verändernden Properties beinhaltet.
setpropname <#> <name>	Set Property Name #<#> : '<name>' OK Invalid Property Number. ERR Missing Argument. ERR	Setzt den Namen des allgemeinen Properties mit der Nummer <#>. Es können 50 allgemeine Properties angegeben werden, das bedeutet <#> muss zwischen 0 und 49 liegen.
getpropname <#>	Get Property Name #<#> : '<name>' OK Invalid Property Number. ERR Missing Argument. ERR	Liefert den Namen des allgemeinen Properties mit der Nummer <#>.
setpropval <#> <val>	Set Property Value #<#> : '<val>' OK Invalid Property Number. ERR Missing Argument. ERR	Setzt einen neuen Wert für das allgemeine Property mit der Nummer <#>. Wenn sich der neue Wert vom zuletzt Übertragenen unterschiedet so wird sofort eine Übertragung zum Thingworx-Server angestoßen.
getpropval <#>	Get Property Value #<#> : '<val>' OK Invalid Property Number. ERR Missing Argument. ERR	Liefert den letzten empfangenen Wert des angegebenen allgemeinen Properties mit der Nummer <#>.
setvaluename <#> <name>	Set Value Name #<#> : '<name>' OK Invalid Value Number. ERR Missing Argument. ERR	Setzt den Namen des Value- Properties mit der Nummer <#>. Es können 50 Value- Properties angegeben werden, das bedeutet <#> muss zwischen 0 und 49 liegen.
getvaluename <#>	Get Value Name #<#> : '<name>' OK Invalid Value Number. ERR Missing Argument. ERR	Liefert den Namen des Value- Properties mit der Nummer <#>.
setvalth <#> <th>	Set Value Threshold #<#> : '<th>' OK Invalid Value Number. ERR Missing Argument. ERR	Setzt die Übertragungsschwelle des Value- Properties mit der Nummer <#>. Wird ein neuer Wert empfangen so wird dieser mit dem letzten gesendeten verglichen. Nur wenn der Unterschied größer als der eingestellte Wert ist wird der neue Wert sofort auf den Thingworx-Server veröffentlicht. Default-Wert ist 0,1.
getvalth <#>	Get Value Threshold #<#> : '<th>' OK Invalid Value Number. ERR Missing Argument. ERR	Liefert die Übertragungsschwelle des Value- Properties mit der Nummer <#>.
setval <#> <val>	Set Value #<#> : '<val>' OK Invalid Value Number. ERR Missing Argument. ERR	Setzt einen neuen Wert für das Value- Property mit der Nummer <#>. Wenn sich der neue Wert vom zuletzt Übertragenen um mehr als die eingestellte Übertragungsschwelle unterschiedet so wird sofort eine Übertragung zum Thingworx-Servers angestoßen.
getval <#>	Get Value #<#> : '<val>' OK Invalid Value Number. ERR Missing Argument. ERR	Liefert den letzten empfangenen Wert des angegebenen Value- Properties mit der Nummer <#>.
setboolname <#> <name>	Set Bool Name #<#> : '<name>' OK Invalid Bool Number. ERR Missing Argument. ERR	Setzt den Namen des Boole'schen- Properties mit der Nummer <#>. Es können 50 Boole- Properties angegeben werden, das bedeutet <#> muss zwischen 0 und 49 liegen.
getboolname <#>	Get Bool Name #<#> : '<name>' OK Invalid Bool Number. ERR Missing Argument. ERR	Liefert den Namen des Boole- Properties mit der Nummer <#>.
setboolval <#> <val>	Set Bool Value #<#> : '<val>' OK Invalid Bool Number. ERR Missing Argument. ERR	Setzt einen neuen Wert für das Boole- Property mit der Nummer <#>. Wenn sich der neue Wert vom zuletzt Übertragenen unterscheidet so wird sofort eine Übertragung zum Thingworx-Server angestoßen.
getboolval <#>	Get Bool Value #<#> : '<val>' OK	

	`Invalid Bool Number.`
	`ERR`
	`Missing Argument.`
	`ERR`
		Liefert den letzten empfangenen Wert des angegebenen Boole- Properties mit der Nummer <#>.

4.5.3 Bibliotheken und deren Keywords

4.5.3.1 Timeout

Warten einer gewissen, frei zu definierender Zeit.

```
#include <Timeout.h>

Timeout to(unsigned long time)
```

Parameter:

`time` Wartezeit in ms für das Timeout.

Methoden:

`to.SetNow()` Startet den Timeout zu genau diesem Zeitpunkt.

`to.SetTimeout(unsigned long time)` Setzen einer neuen Wartezeit
- `time` Neue Wartezeit in ms

`bool to.TimedOut()` Abfrage, ob die Wartezeit schon vorbei ist
- `true` Wartezeit ist vorbei
- `false` Der letzte Aufruf von `SetNow` oder `SetTimeout` ist noch nicht einmal die Wartezeit her.

`unsigned long to.GetTimeoutValue()` Abfrage, der Wartezeit dieses Timeouts

`unsigned long to.GetElapsedTime()` liefert die Zeit seit dem letzten Aufruf von `SetNow` oder `SetTimeout`. Liefert maximal die Wartezeit.

`unsigned long to.GetRestTime()` liefert die restliche Zeit des laufenden Timeouts. Liefert minimal den Wert 0.

`unsigned long to.GetElapsedPortion()` liefert den restlichen Anteil des laufenden Timeouts in %. Ist der Timeout abgelaufen liefert die Funktion 100%.

4.5.4 Ansprechend des ESP MQTT-Gateways

```
#include <EspMQTTGateway.h>

EspMQTTGateway mqttClient(*SerialIF, *DebugIF, *callback);
```

Parameter:

`*SerialIF`	Pointer zu einem Hardware- oder Software-Serial-Objekt. Dort ist der ESP angeschlossen.
`*DebugIF`	Pointer zu einem Hardware- oder Software-Serial-Objekt. Dort wird die Bibliothek Meldungen ausgeben (Debug-Informationen zu einem Serial Monitor)
`*callback`	Pointer zu einer Callback-Funktion, falls ein abonniertes Topic empfangen wurde.

Die Callback-Funktion muss folgendes Format haben:

```
void mqttCallback(int topic, String tag, float value)
```

Parameter:

`topic`	Nummer des empfangen Topics.
`tag`	Inhalt des empfangen Topics als Zeichenkette.
`value`	Inhalt des empfangen Topics als Zahl.

Methoden:

`mqttClient.Update()` Aufruf des Clients und Abarbeitung von dessen Code (z.B. Nachsehen, ob ein neues Topic empfangen wurde oder auch abfragen ob der ESP erreichbar ist. Muss einmal jeden loop-Durchlauf aufgerufen werden.

`mqttClient.SetBlockTime(float value)` Setzen der Blockierzeit. Innerhalb dieser Zeit wird kein neuer Wert übertragen

`value`	Zeit der Sendeblockade in s $[0 - 30]$

`mqttClient.SetResendTime(float value)` Setzen der Wiederholzeit. Nach Ablauf dieser Zeit werden die aktuellen Werte übertragen – unabhängig davon, ob sie sich geändert haben oder nicht.

`value`	Zeit der Sendewiederholung in s $[0 - 3600]$

`mqttClient.SetVerbose(byte verbose)` Setzen der „Gesprächigekit". Je höher die Zahl ist desto mehr Output wird von der Bibliothek am eingestellten Debug-Interface generiert.

`verbose`	Bit 7 (128): Ausgabe von Debug-Informationen Bit 6 (64): Ausgabe von Informationslevel Bit 5 (32): Ausgabe von Warnungslevel Bit 4 (16): Ausgabe von Fehlerlevel

`mqttClient.SetWifiSsid(ssid)` Setzen der SSID auf dem ESP

`ssid`	die SSID mit der sich der ESP verbinden soll

`bool mqttClient.WifiSsidTranferred()` Auslesen, ob die SSID erfolgreich zum ESP übertragen wurde (dann kommt `true`).

`mqttClient.SetWifiPass(wifipass)` Setzen des WiFi-Passworts auf dem ESP
`wifipass` das Passwort des WiFi mit der sich der ESP verbinden soll

`bool mqttClient.WifiPassTranferred()` Auslesen, ob das WiFi-Passwort erfolgreich zum ESP übertragen wurde (dann kommt `true`).

`bool mqttClient.WifiConnected()` Auslesen, ob sich der ESP erfolgreich mit dem WiFi-Netzwerk verbunden hat (dann kommt `true`).

`mqttClient.SetMqttHost(host)` Setzen des Hosts (IP-Adresse) des MQTT-Brokers
`host` die Host-Adresse des MQTT-Brokers

`bool mqttClient.MqttHostTranferred()` Auslesen, ob die Host-Adresse des MQTT-Brokers erfolgreich zum ESP übertragen wurde (dann kommt `true`).

`mqttClient.SetMqttUser(user)` Setzen des Usernamens des MQTT-Brokers
`user` der Username des MQTT-Brokers

`bool mqttClient.MqttUserTranferred()` Auslesen, ob der Username des MQTT-Brokers erfolgreich zum ESP übertragen wurde (dann kommt `true`).

`mqttClient.SetMqttPass(pass)` Setzen des Userpassworts des MQTT-Brokers
`pass` das Userpasswort des MQTT-Brokers

`bool mqttClient.MqttPassTranferred()` Auslesen, ob das Userpasswort des MQTT-Brokers erfolgreich zum ESP übertragen wurde (dann kommt `true`).

`bool mqttClient.MqttConnected()` Auslesen, ob sich der ESP erfolgreich mit dem MQTT-Broker verbunden hat (dann kommt `true`).

`bool mqttClient.SetCommonName(int num, String name)`
Setzen des Namens des Topics mit der Nummer `num`.
`num` Nummer des Topics für den der Name ist
`name` Neuer Name des Topics

`bool mqttClient.DelCommonName(int num)`
Löschen des Namens des Topics mit der Nummer `num`.
`num` Nummer des Topics dessen Name zu löschen ist

`bool mqttClient.SetCommon(int num, String value)`
Setzen des Topics mit der Nummer `num`.
`num` Nummer des Topics für den der neue Wert ist
`value` Neuer Wert des Topics

`bool mqttClient.SetBoolName(int num, String name)`
Setzen des Namens des Boolwerts mit der Nummer `num`.
`num` Nummer des Boolwerts für den der Name ist
`name` Neuer Name des Boolwerts

`bool mqttClient.DelBoolName(int num)`
Löschen des Namens des Boolwerts mit der Nummer `num`.
`num` Nummer des Boolwerts dessen Name zu löschen ist

```
bool mqttClient.SetBool(int num, bool value)
```
Setzen des Boolwerts mit der Nummer `num`.

`num`	Nummer des Boolwerts für den der neue Wert ist
`value`	Neuer Wert des Boolwerts

```
bool mqttClient.SetValueName(int num, String name)
```
Setzen des Namens des Wertes mit der Nummer `num`.

`num`	Nummer des Wertes für den der Name ist
`name`	Neuer Name des Wertes

```
bool mqttClient.DelValueName(int num)
```
Löschen des Namens des Wertes mit der Nummer `num`.

`num`	Nummer des Wertes dessen Name zu löschen ist

```
bool mqttClient.SetValue(int num, float value[, int commas])
```
Setzen des Wertes mit der Nummer `num`.

`num`	Nummer des Wertes für den der neue Wert ist
`value`	Neuer Wert des Wertes
`commas`	Übertragungsgenauigkeit des Wertes (Default `2`)

```
bool mqttClient.SetValueThreshold(int num, float value[, int
commas])
```
Setzen der Übertragungsschwelle des Wertes mit der Nummer `num`.

`num`	Nummer des Wertes für die Übertragungsschwelle
`value`	Wert der Übertragungsschwelle
`commas`	Übertragungsgenauigkeit der Schwelle (Default `2`)

```
bool mqttClient.Subsribe(int num, String name)
```
Setzen des Topics, welches abonniert werden soll mit der Nummer `num`.

`num`	Nummer des Abos für den der Name ist
`name`	Topic welches abonniert werden soll

```
bool mqttClient.UnSubscribe(int num)
```
Löschen des Abonnements des Topics mit der Nummer `num`.

`num`	Nummer des Topics das nicht länger abonniert ist

4.5.4.1 Ansprechend des ESP Thingworx-Gateways

```
#include <EspThingworxGateway.h>

EspThingworxGateway twxClient(*SerialIF, *DebugIF);
```

Parameter:

`*SerialIF`	Pointer zu einem Hardware- oder Software-Serial-Objekt. Dort ist der ESP angeschlossen.
`*DebugIF`	Pointer zu einem Hardware- oder Software-Serial-Objekt. Dort wird die Bibliothek Meldungen ausgeben (Debug-Informationen zu einem Serial Monitor)

Methoden:

`twxClient.Update()` Aufruf des Clients und Abarbeitung von dessen Code (z.B. Nachsehen ob ein neues Topic empfangen wurde oder auch abfragen ob der ESP erreichbar ist. Muss einmal jeden loop-Durchlauf aufgerufen werden.

`twxClient.SetBlockTime(float value)` Setzen der Blockierzeit. Innerhalb dieser Zeit wird kein neuer Wert übertragen

`value` Zeit der Sendeblockade in s $[0 - 30]$

`twxClient.SetResentTime(float value)` Setzen der Wiederholzeit. Nach Ablauf dieser Zeit werden die aktuellen Werte übertragen – unabhängig davon ob sie sich geändert haben oder nicht.

`value` Zeit der Sendeblockade in s $[0 - 3600]$

`twxClient.SetVerbose(byte verbose)` Setzen der „Gesprächigkeit". Je höher die Zahl ist desto mehr Output wird von der Bibliothek am eingestellten Debug-Interface generiert.

`verbose` Bit 7 gesetzt: Ausgabe von Debug-Informationen
Bit 6 gesetzt: Ausgabe von Informationslevel
Bit 5 gesetzt: Ausgabe von Warnungslevel
Bit 4 gesetzt: Ausgabe von Fehlerlevel

`twxClient.SetWifiSsid(ssid)` Setzen der SSID auf dem ESP

`ssid` die SSID mit der sich der ESP verbinden soll

`bool twxClient.WifiSsidTranfered()` Auslesen, ob die SSID erfolgreich zum ESP übertragen wurde (dann kommt `true`).

`twxClient.SetWifiPass(wifipass)` Setzen des WiFi-Passworts auf dem ESP

`wifipass` das Passwort des WiFi mit der sich der ESP verbinden soll

`bool twxClient.WifiPassTranfered()` Auslesen, ob das WiFi-Passwort erfolgreich zum ESP übertragen wurde (dann kommt `true`).

`bool twxClient.WifiConnected()` Auslesen, ob sich der ESP erfolgreich mit dem WiFi-Netzwerk verbunden hat (dann kommt `true`).

`twxClient.SetThingworxHost(host)` Setzen des Hosts (IP-Adresse) des Thingworx-Server
`host` die Host-Adresse des Thingworx-Server

`bool twxClient.ThingworxHostTranfered()` Auslesen, ob die Host-Adresse des Thingwork-Servers erfolgreich zum ESP übertragen wurde (dann kommt `true`).

`twxClient.SetAppkey (key)` Setzen des Application Keys des Things
`key` der Application Key des Things

`bool twxClient.AppkeyTranfered()` Auslesen, ob der Application Keydes Tings erfolgreich zum ESP übertragen wurde (dann kommt `true`).

`twxClient.SetThing(thing)` Setzen des Thing-Namens
`thing` der Name des Things

`bool twxClient.ThingTranfered()` Auslesen, ob der Name des Things erfolgreich zum ESP übertragen wurde (dann kommt `true`).

`bool twxClient.ThingworxConnected()` Auslesen, ob sich der ESP erfolgreich mit dem Thingworx-Server verbunden hat (dann kommt `true`). Dies kann nur bei einem Sendeversuch aktualisiert werden, es kann also sein, dass der Thingworx-Server schon gar nicht mehr erreichbar ist. Beim nächsten nicht erfolgreichen Sendeversuch wird der Wert auf `fasle` gesetzt (und natürlich auf `true` wenn wieder erfolgreich übertragen wurde).

`bool twxClient.SetCommonName(int num, String name)`
Setzen des Namens des Properties mit der Nummer `num`.
`num` Nummer des Properties für den der Name ist
`name` Neuer Name des Properties

`bool twxClient.DelCommonName(int num)`
Löschen des Namens des Properties mit der Nummer `num`.
`num` Nummer des Properties dessen Name zu löschen ist

`bool twxClient.SetCommon(int num, String value)`
Setzen des Properties mit der Nummer `num`.
`num` Nummer des Properties für den der neue Wert ist
`value` Neuer Wert des Topics

`bool twxClient.SetBoolName(int num, String name)`
Setzen des Namens des Boolwerts mit der Nummer `num`.
`num` Nummer des Boolwerts für den der Name ist
`name` Neuer Name des Boolwerts

`bool twxClient.DelBoolName(int num)`
Löschen des Namens des Boolwerts mit der Nummer `num`.
`num` Nummer des Boolwerts dessen Name zu löschen ist

`bool twxClient.SetBool(int num, bool value)`
Setzen des Boolwerts mit der Nummer `num`.
`num` Nummer des Boolwerts für den der neue Wert ist
`value` Neuer Wert des Boolwerts

```
bool twxClient.SetValueName(int num, String name)
```

Setzen des Namens des Wertes mit der Nummer `num`.

`num`	Nummer des Wertes für den der Name ist
`name`	Neuer Name des Wertes

```
bool twxClient.DelValueName(int num)
```

Löschen des Namens des Wertes mit der Nummer `num`.

`num`	Nummer des Wertes dessen Name zu löschen ist

```
bool twxClient.SetValue(int num, float value[, int commas])
```

Setzen des Wertes mit der Nummer `num`.

`num`	Nummer des Wertes für den der neue Wert ist
`value`	Neuer Wert des Wertes
`commas`	Übertragungsgenauigkeit des Wertes (Default `2`)

```
bool twxClient.SetValueThreshold(int num, float value[, int commas])
```

Setzen der Übertragungsschwelle des Wertes mit der Nummer `num`.

`num`	Nummer des Wertes für die Übertragungsschwelle
`value`	Wert der Übertragungsschwelle
`commas`	Übertragungsgenauigkeit der Schwelle (Default `2`)

4.6 HTTP-Statuscodes

Ein HTTP-Statuscode wird von einem Server auf jede HTTP-Anfrage als Antwort geliefert. Auf der anfragenden Seite steht dabei ein Client wie beispielsweise ein Webbrowser. Der Server teilt durch den HTTP-Statuscode dem Client mit, ob die Anfrage erfolgreich bearbeitet wurde.

	Code	Nachricht	Bedeutung
1xx - Info	100	Continue	Die laufende Anfrage an den Server wurde noch nicht zurückgewiesen. Der Client kann nun mit der potenziell sehr großen Anfrage fortfahren.
	101	Switching Protocols	Wird verwendet, wenn der Server mit dem Wechsel zu einem anderen Protokoll einverstanden ist. Z.B. Wechsel von HTTP zu WebSocket.
	102	Processing	Wird verwendet, um ein Timeout zu vermeiden, während der Server eine zeitintensive Anfrage bearbeitet.
2xx – Successful Operation	200	OK	Die Anfrage wurde erfolgreich bearbeitet und das Ergebnis der Anfrage wird in der Antwort übertragen.
	201	Created	Die Anfrage wurde erfolgreich bearbeitet. Die angeforderte Ressource wurde vor dem Senden der Antwort erstellt.
	202	Accepted	Die Anfrage wurde akzeptiert, wird aber zu einem späteren Zeitpunkt ausgeführt. Das Gelingen der Anfrage kann nicht garantiert werden.
	203	Non-Authoritative Information	Der Server agiert als „Transforming Proxy", erhielt eine 200 OK Antwort von der Quelle und antwortet mit einem veränderten Dokument der Quelle.
	204	No Content	Die Anfrage wurde erfolgreich durchgeführt, die Antwort enthält jedoch bewusst keine Daten.
	205	Reset Content	Die Anfrage wurde erfolgreich durchgeführt; der Client soll das Dokument neu aufbauen und Formulareingaben zurücksetzen.
	206	Partial Content	Der angeforderte Teil wurde erfolgreich übertragen. Kann einen Client über Teil-Downloads informieren).
	207	Multi-Status	Die Antwort enthält ein XML-Dokument, das mehrere Statuscodes zu unabhängig voneinander durchgeführten Operationen enthält.
	208	Already Reported	Die Mitglieder einer WebDAV-Bindung wurden bereits zuvor aufgezählt und sind in dieser Anfrage nicht mehr vorhanden.
	226	IM Used	Der Server hat eine GET-Anforderung erfüllt, ist eine Darstellung des Ergebnisses einer Instanz-Manipulationen, bezogen auf die aktuelle Instanz.
3xx – Redirect	300	Multiple Choices	Die angeforderte Ressource steht in verschiedenen Arten zur Verfügung. Die Antwort enthält eine Liste der verfügbaren Arten.
	301	Moved Permanently	Die angeforderte Ressource steht ab sofort unter der im „Location"-Header-Feld angegebenen Adresse bereit (auch Redirect genannt).
	302	Found (Moved Temporarily)	Die angeforderte Ressource steht vorübergehend unter der im „Location"-Header-Feld angegebenen Adresse bereit. Die alte Adresse bleibt gültig.
	303	See Other	Die Antwort auf die durchgeführte Anfrage lässt sich unter der im „Location"-Header-Feld angegebenen Adresse beziehen.
	304	Not Modified	Der Inhalt der angeforderten Ressource hat sich seit der letzten Abfrage des Clients nicht verändert und wird deshalb nicht übertragen.
	305	Use Proxy	Die angeforderte Ressource ist nur über einen Proxy erreichbar. Das „Location"-Header-Feld enthält die Adresse des Proxys.
	306	(reserved)	306 wird nicht mehr verwendet, ist aber reserviert.
	307	Temporary Redirect	Die angeforderte Ressource steht vorübergehend unter der im „Location"-Header-Feld angegebenen Adresse bereit. Die alte Adresse bleibt gültig.
	308	Permanent Redirect	Die angeforderte Ressource steht ab sofort unter der im „Location"-Header-Feld angegebenen Adresse bereit, die alte Adresse ist nicht länger gültig.

4xx - Client Error	**400**	Bad Request	Die Anfrage-Nachricht war fehlerhaft aufgebaut.
	401	Unauthorized	Die Anfrage kann nicht ohne gültige Authentifizierung durchgeführt werden.
	402	Payment Required	Dieser Status ist für zukünftige HTTP-Protokolle reserviert.
	403	Forbidden	Die Anfrage wurde mangels Berechtigung des Clients nicht durchgeführt, bspw. Weil eine als HTTPS konfigurierte URL nur mit HTTP aufgerufen wurde.
	404	Not Found	Die angeforderte Ressource wurde nicht gefunden. Kann ebenfalls verwendet werden, um eine Anfrage ohne näheren Grund abzuweisen.
	405	Method Not Allowed	Die Anfrage darf nur mit anderen HTTP-Methoden (zum Beispiel GET statt POST) gestellt werden.
	406	Not Acceptable	Die angeforderte Ressource steht nicht in der gewünschten Form zur Verfügung.
	407	Proxy Authen. Required	Analog zum Statuscode 401 ist hier zunächst eine Authentifizierung des Clients gegenüber dem verwendeten Proxy erforderlich.
	408	Request Timeout	Innerhalb der erlaubten Zeitspanne keine vollständige Anfrage empfangen.
	409	Conflict	Die Anfrage wurde unter falschen Annahmen gestellt.
	410	Gone	Die angeforderte Ressource wird nicht länger bereitgestellt.
	411	Length Required	Die Anfrage kann ohne ein „Content-Length" nicht bearbeitet werden.
	412	Precondition Failed	Eine in der Anfrage übertragene Voraussetzung, zum Beispiel in Form eines „If-Match"-Header-Felds, traf nicht zu.
	413	Request Entity Too Large	Die gestellte Anfrage war zu groß, um vom Server bearbeitet werden zu können. Ein „Retry-After"-Header-Feld in der Antwort kann den Client darauf hinweisen, dass die Anfrage eventuell zu einem späteren Zeitpunkt bearbeitet werden könnte.
	414	UTI Too Long	Die URL der Anfrage war zu lang. Ursache ist oft eine Endlosschleife aus Redirects.
	415	Unsupported Media Type	Der Inhalt der Anfrage wurde mit ungültigem oder nicht erlaubtem Medientyp übermittelt.
	416	Requested Range Not Satisfiable	Der angeforderte Teil einer Ressource war ungültig oder steht auf dem Server nicht zur Verfügung.
	417	Expectation Failed	Verwendet im Zusammenhang mit einem „Expect"-Header-Feld. Das im „Expect"-Header-Feld geforderte Verhalten des Servers kann nicht erfüllt werden.
	420	Policy Not Fullfilled	In W3C PEP (Working Draft 21. November 1997) wird dieser Code vorgeschlagen, um mitzuteilen, dass eine Bedingung nicht erfüllt wurde.
	421	Misdirected Request	Die Anfrage wurde an einen Server gesendet, der nicht in der Lage ist, eine Antwort zu senden. Eingeführt in HTTP/2.
	422	Unprocessable Entity	Verwendet, wenn weder die Rückgabe von Statuscode 415 noch 400 gerechtfertigt wäre, eine Verarbeitung der Anfrage jedoch zum Beispiel wegen semantischer Fehler abgelehnt wird.
	423	Locked	Die angeforderte Ressource ist zurzeit gesperrt.
	424	Failed Dependency	Die Anfrage konnte nicht durchgeführt werden, weil sie das Gelingen einer vorherigen Anfrage voraussetzt.
	426	Upgrade Required	Der Server verlangt vom Client, dass er die Anfrage mit einem anderen Protokoll wiederholt. Ein Anwendungsfall ist das Umschalten auf HTTP mit Transport Layer Security.
	428	Precondition Required	Für die Anfrage waren nicht alle Vorbedingungen erfüllt. Dieser Statuscode soll Probleme durch Race Conditions verhindern, indem eine Manipulation oder Löschen nur erfolgt, wenn der Client dies auf Basis einer aktuellen Ressource anfordert (Beispielsweise durch Mitliefern eines aktuellen ETag-Headers).
	429	Too Many Requests	Der Client hat zu viele Anfragen in einem bestimmten Zeitraum gesendet.
	431	Request Header Fields Too Large	Die Maximallänge eines Headerfelds oder des Gesamtheaders wurde überschritten.
	451	Unavailable For Legal Reasons	Dieser Statuscode soll darauf hinweisen, dass die angeforderte Ressource aufgrund von gesetzlichen Bestimmungen (Copyrighteinschränkungen, Zensur etc., eventuell beschränkt auf ein bestimmtes Land) nicht verfügbar ist. Er wurde im Juni 2012 von Google-Mitarbeiter Tim Bray bei der IETF eingereicht und gilt seit dem 17. Dezember 2015 als angenommen. Bray schrieb am Ende seines Vorschlags „Thanks also to Ray Bradbury". In Anspielung auf dessen Roman Fahrenheit 451 schlug Bray die Nummer 451 für den Statuscode vor.

5xx – Server Error	**500**	Internal Server Error	Dies ist ein „Sammel-Statuscode" für unerwartete Serverfehler.
	501	Not Implemented	Die Funktionalität, um die Anfrage zu bearbeiten, wird von diesem Server nicht bereitgestellt. Ursache ist zum Beispiel eine unbekannte oder nicht unterstützte HTTP-Methode.
	502	Bad Gateway	Der Server konnte seine Funktion als Gateway oder Proxy nicht erfüllen, weil er seinerseits eine ungültige Antwort erhalten hat.
	503	Service Unavailable	Der Server steht temporär nicht zur Verfügung, zum Beispiel wegen Überlastung oder Wartungsarbeiten. Ein „Retry-After"-Header-Feld in der Antwort kann den Client auf einen Zeitpunkt hinweisen, zu dem die Anfrage eventuell bearbeitet werden könnte.
	504	Gateway Timeout	Der Server konnte seine Funktion als Gateway oder Proxy nicht erfüllen, weil er innerhalb einer festgelegten Zeitspanne keine Antwort von seinerseits benutzten Servern oder Diensten erhalten hat.
	505	HTTP Version Not Supported	Die benutzte HTTP-Version (gemeint ist die Zahl vor dem Punkt) wird vom Server nicht unterstützt oder abgelehnt.
	506	Variant Also Negotiates	Die Inhaltsvereinbarung der Anfrage ergibt einen Zirkelbezug.
	507	Insufficient Storage	Die Anfrage konnte nicht bearbeitet werden, weil der Speicherplatz des Servers dazu derzeit nicht mehr ausreicht.
	508	Loop Detected	Die Operation wurde nicht ausgeführt, weil die Ausführung in eine Endlosschleife gelaufen wäre. Definiert in der Binding-Erweiterung für WebDAV gemäß RFC 5842, weil durch Bindings zyklische Pfade zu WebDAV-Ressourcen entstehen können.
	509	Bandwidth Limit Exceeded	Die Anfrage wurde verworfen, weil sonst die verfügbare Bandbreite überschritten würde (inoffizielle Erweiterung einiger Server).
	510	Not Extended	Die Anfrage enthält nicht alle Informationen, die die angefragte Server-Extension zwingend erwartet.
	511	Network Authentication Required	Der Client muss sich zuerst authentifizieren, um Zugang zum Netzwerk zu erhalten.