Script for the

Lecture and the exercises

about

Augmented reality with real-time data

from

Heinz PETERSCHOFSKY

2020-1-DE02-KA202-007393

Krems and St. Pölten, 2023

Co-funded by the Erasmus+ Programme of the European Union

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.

Table of Content

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.

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.

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

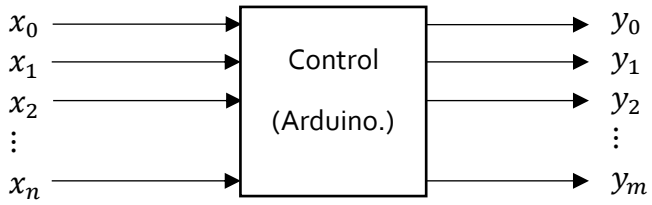
Playlist: https://www.youtube.com/playlist?list=PLVut1tKPvtvNNhOVwQmlc74szsikRsIDk

1.1 General

1.1.1 What is an Arduino?

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

"Arduino" refers to a single board controller. All the important components are housed on a small piece of circuit board. In terms of function, it is actually quite simple:



x_n ... Inputs to the control (either digital or analog)
y_m ... Outputs from the control (either digital or "PWM")

Whether and when inputs are read in, or whether and when outputs are written and what the sequence is, is stored in the controller with a "program". This program defines how the controller should react to different states (which outputs are set). With the outputs, the control can influence anything in the real world.

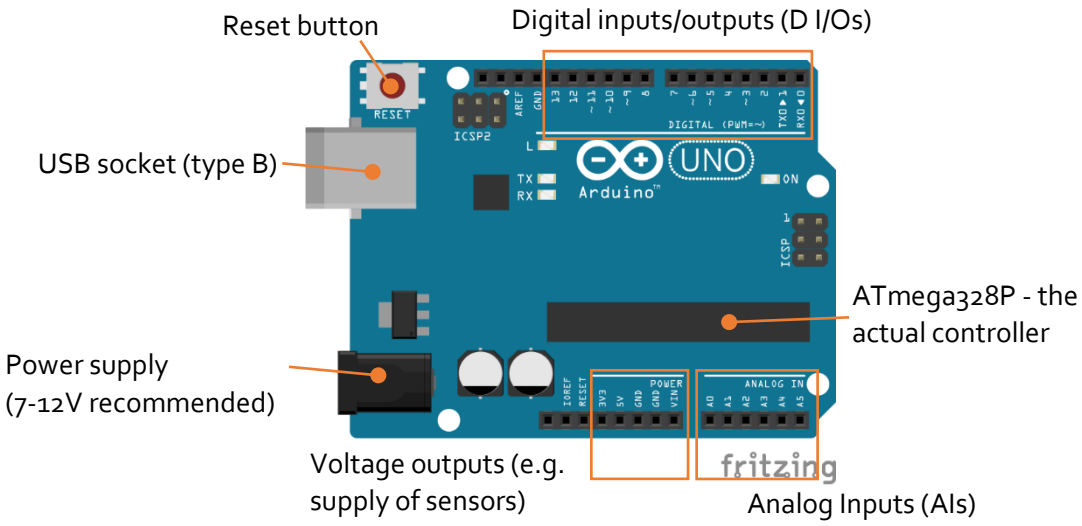
Our Arduino is supposed to be able to read things in the environment and react to them according to a program. In this case, the "program" is actually a piece of software that can respond to the change in inputs. The program is stored in memory in the Arduino and is replaceable. To swap it out, all we need is a computer and we can simply overwrite the program in the program memory - not much more complicated than writing to a USB flash drive.

So, our Arduino has inputs and outputs and a modifiable program that resides in a memory. That makes it essentially what is called a "Programmable Logic Controller" ("PLC").

Starting with simple things, we will try to learn all the functions of the Arduino.

1.1.2 The Arduino hardware

We use an "Arduino UNO" for this purpose. This board is well suited for the entry into the Arduino world. You can try out practically all things with it. However, it is too "weak" for complex control tasks and a bit too big for small solutions.



Microcontroller	ATmega328P	Practically the whole Arduino is included in this chip memory
Logic level	5V	5V means logical "1" and 0V means logical "0" (TTL[1] -level)
Input voltage	7-12V recommended 6-20V maximum	level of the power supply, is also supplied with power via the USB socket.
Maximum current per I/O pin	20 mA	
Maximum current at 3.3V	50 mA	
Program memory (Flash)	32 kB	Built in ATmega328P
Working memory (SRAM)	2 kB	Built in ATmega328P
EEPROM	1 kB	Built in ATmega328P
Clock rate	16 MHz	
Dimensions (lxw)	68.6 mm x 53.4 mm	
Weight	25 g	

Details about the UNO as well as information about other products can also be found on the Arduino homepage: www.arduino.cc.

[1] TTL: Transistor-Transistor-Logic

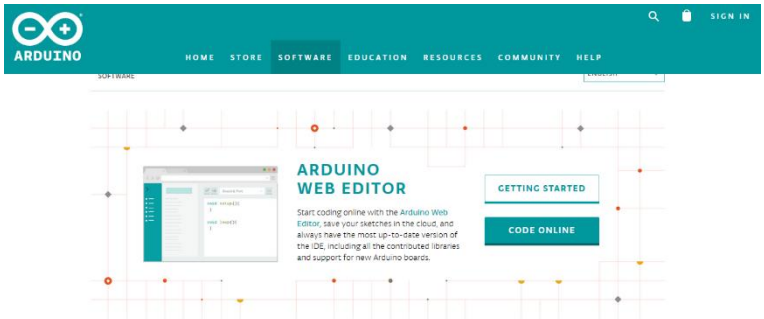
1.1.3 Setting up the software

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

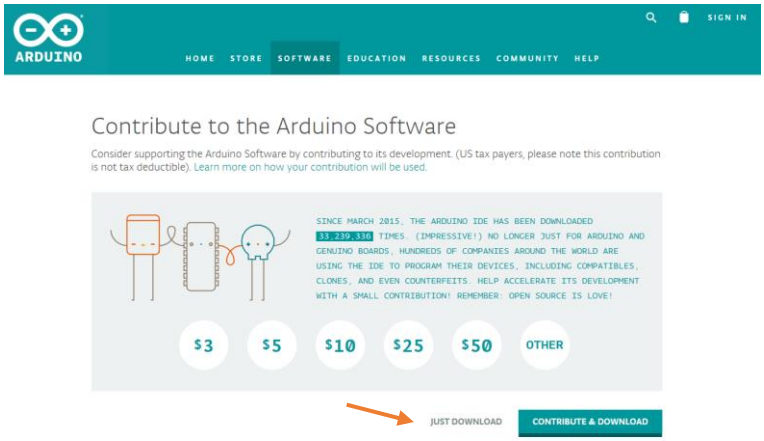
1.1.3.1 Download

The programming software "Arduino IDE[2] " can be downloaded from the Arduino homepage:

In the "Software" menu, select the "Downloads" page.



Selection of the "Windows Installer"

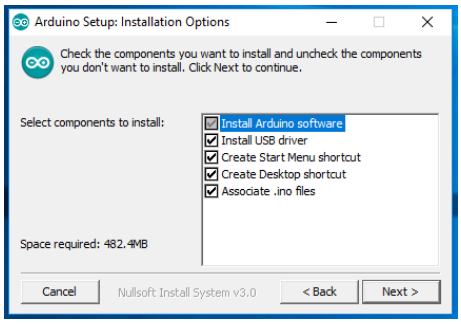


Who wants can donate or click on "Just download".

[2] IDE: Integrated Development Environment: Combines a text editor and the necessary things to translate the program (compiler) and bring it to the Arduino (loader).

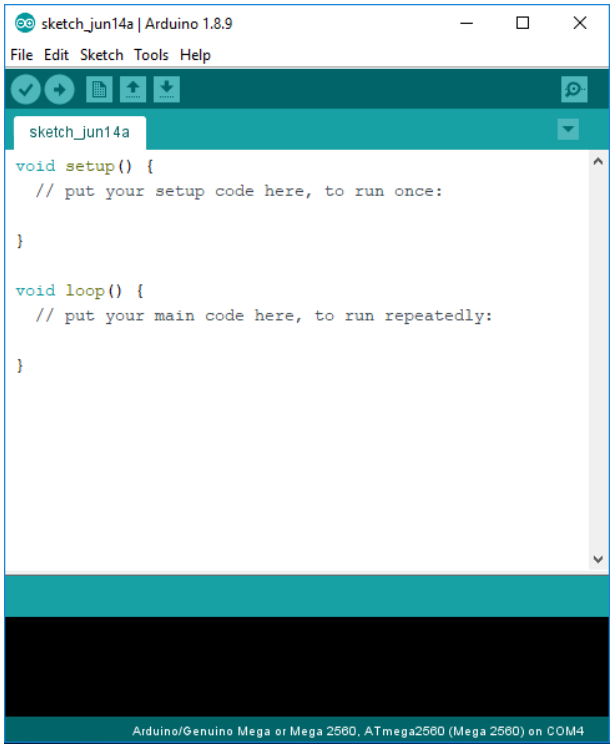
1.1.3.2 Install

After the download is done, the IDE can be installed. For this I always activate all options. If someone doesn't want to have an icon on the desktop or doesn't want to open *.INO files automatically with the IDE, please deselect the corresponding options.



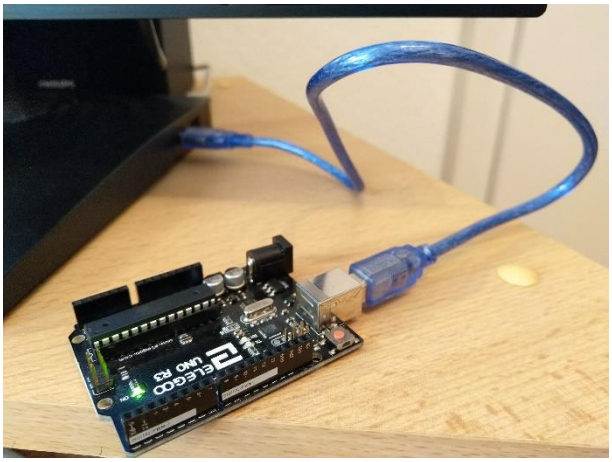
1.1.3.3 Launch

After the installation is complete, the Arduino IDE can be started. After the start it should look like this:



1.1.3.4 Connection settings

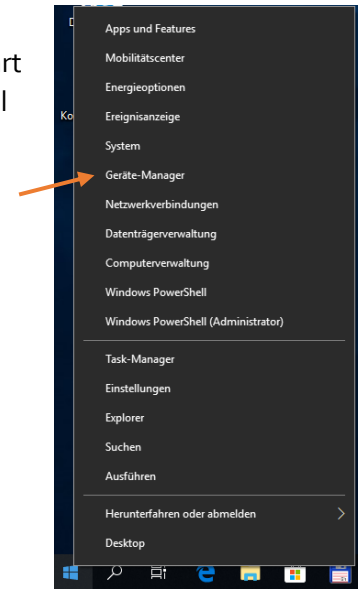
Video Link 3: https://youtu.be/5M2wrfL5oPY

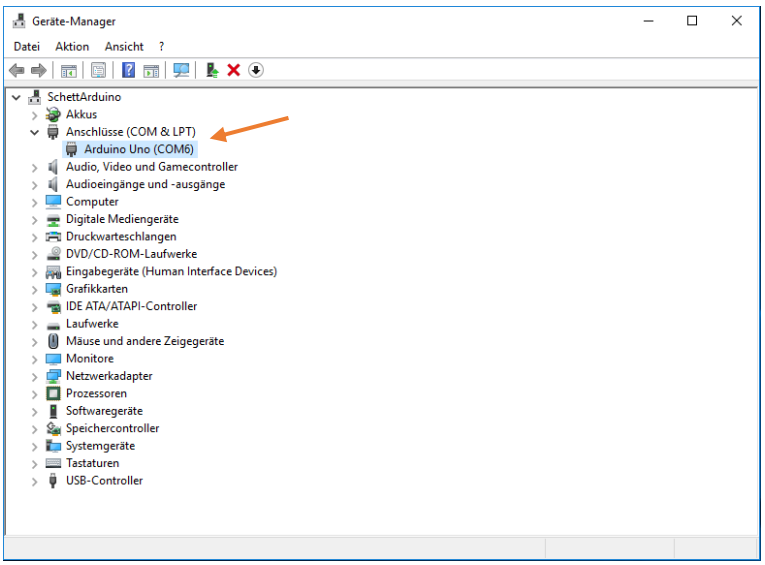


Now that we have the IDE installed, we must get to introduce our Arduino and the IDE. To do this, we need to connect our board to the computer for the first time. The starter kit already contains a suitable USB cable. Therefore, it should be no problem to connect the Arduino to a free USB port on the computer.

Immediately the LEDs light up and the Arduino is ready for operation. In the best of all cases, the appropriate driver is also loaded on the computer - at least the appropriate driver should have been installed with the installation of the IDE. On the computer, if everything is correct, we now find an additional COM port[3] . How can we now find out whether a suitable virtual COM port has been created?

On Windows computers, the so-called "Device Manager" can be called up. To do this, click the RIGHT mouse button on the Windows icon (Start menu). A selection menu opens. If you click on "Device Manager", it will open. There you should find the COM port under "COM&LPT".





Conveniently, the number of the COM port is also included - in the example it is COM6. Exactly this number must be communicated to the IDE, because how else should the program know where it "must" go?

[3] COM: Communication Port: Originally a serial interface, which was also present as a "plug" in real hardware (in the form of a mostly 9-pin socket). Nowadays, a "virtual" COM port is created for many USB devices, which is only "there" as long as the device is plugged in. For programs it doesn't matter whether they access an actual hardware COM port or a virtual COM port. The Windows drivers do the correct "translation".

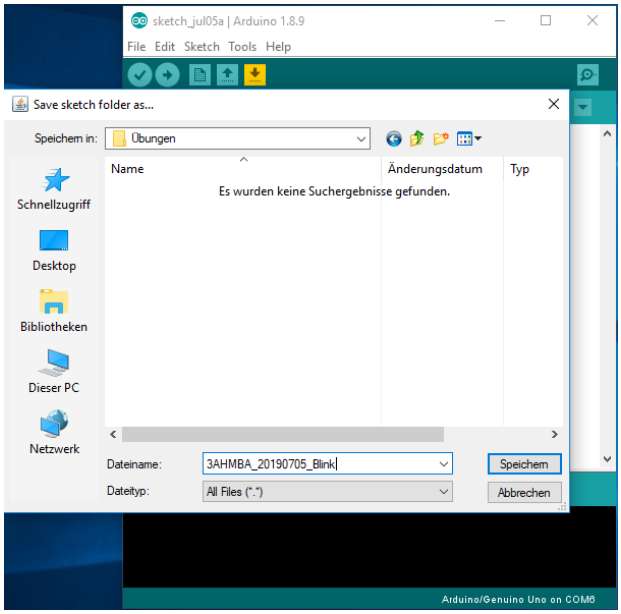
1.2 Exercises

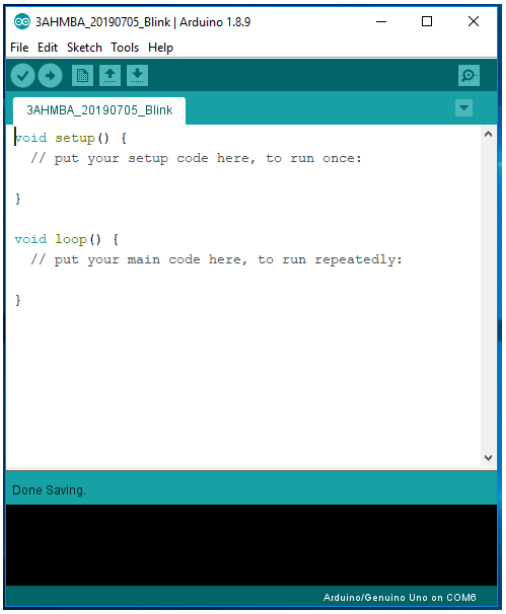
1.2.1 Upload test

Video Link 4: https://youtu.be/VZIgZnGPeN4

Now that we have set up our software so that we can get started, we want to see if we can upload a program to our Arduino.

First, we save our "Sketch" (this is the Arduino name for the program) under a different name. As usual, under "File" and "Save as..." pick an appropriate place and give the file a name. I would suggest using some format. In the example we see in the file name the class (3AHMBA), the creation date (20190705) and the name.





The programming language used is practically C++. A small reference of the language can be found in the appendix, or also on www.arduino.cc.

Basically, we now see two standard functions:

`setup()`

The instructions in Setup are executed after a reset (when e.g. the voltage was connected, or also when the reset button was pressed, or a new program was loaded). This program part then runs exactly 1x and then no more. The purpose behind this is to carry out any initializations. For example, you can specify here whether pin 5 is to be an input or an output - but more about that later.

`loop()`

These instructions are executed "over and over again". Whenever the execution of `loop()` is finished, it starts again from the beginning - hence the name "loop". The purpose behind this is to let the control instructions run through again and again. To query all necessary inputs again and again, and then to derive the necessary actions from them again and again. The "intelligence" of the control is therefore represented here.

We will look into it, and it will then become more and more logical for us what to write where.

What do we want our program to do now? We want to make the built-in LED blink. In a certain rhythm. Actually, we are concerned with the upload and not with the function.

That is why we are now adapting our program accordingly:

```
#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
}
```

What's happening here? We want to turn an LED on and off. So, we want to write to an output. But to which one? Fortunately, the IDE offers us a few constants. One of them is `LED_BUILTIN`. It contains the number of the output for the internal LED. You can define such constants yourself. We did this in our program with the instruction `#define.` There we define the constant `LED_ONOFF_TIME` and set it to a value of `1000` - why we will see below.

First, let's look at `setup()`. There is only a single line:

`pinMode` is a command that defines what an input or output should be. In this case, it sets the PIN numbered `LED_BUILTIN` as the output (`OUTPUT`). All options would be:

`OUTPUT`, `INPUT`, `INPUT_PULLUP`

More about `INPUT` and `INPUT_PULLUP` later.

Good, so now we have an output. We have to describe it, we do this periodically in the function `loop()`:

`digitalWrite` is a command that sets an output (switches `+5V` to it) or resets it (switches `0V` to it). You have to tell the command the PIN number of the output and whether it should set it (`HIGH`) or reset it (`LOW`). Alternatively, `1` (`HIGH`) or `0` (`LOW`) can be written.

The first line in `loop()` switches the LED on (sets the output to `HIGH`). The third line switches the LED off (set the output to `LOW`). In between there are the `delay functions`:

These simply wait. As argument you must pass the wait time in milliseconds to the command. In this example it is `LED_ONOFF_TIME` ms (so 1000ms), so 1 second.

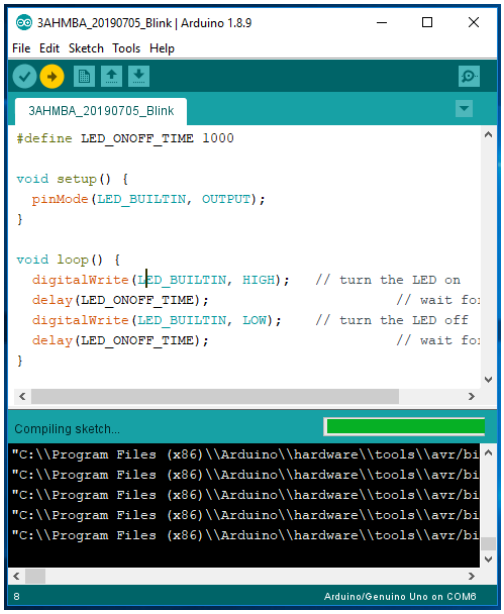
So, the LED is turned up, then it lights up for one second, and then it is turned down again. Then the program pauses again for 1 second and is then off. And because `loop()` is always repeated, the LED is also turned up again. So, the LED flashes with 0.5 Hz.

Now it's time to upload. An age-old rule still applies:

"Save often, Save early!"

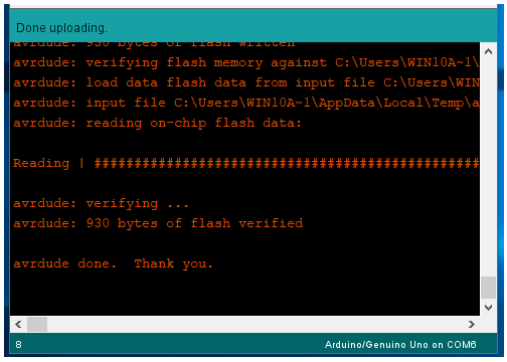
Therefore, before the upload attempt, click on the "Save" icon ( ) in the menu bar. After that you can click on the "Upload" icon. This is an arrow that points to the right for some reason. As if the right would mean up, but anyway, that's where we press it:

What happens now? The compiler is started, and the C++ code is translated into machine language. This translation could also be started with the hook symbol. The output area at the bottom of the window shows the current status.



After successful compilation the upload starts automatically. It will try to connect to the Arduino and copy the generated machine code to the flash memory of our controller.

After successful transfer, a completion message is displayed in the output area:



Error messages are also output in the output area. This allows you to see where the error is.

In the best of all cases, the built-in LED flashes at 2-second intervals - all correct.

1.2.1.1 *Exercises "Blink Intern*

	Create a program as described above, which controls the LED on the Arduino, load it on your Arduino and show it.	

1.2.1.2 *Mandatory tasks "Blink Intern*

	Modify the program so that the LED flashes in a different, asymmetrical rhythm.	

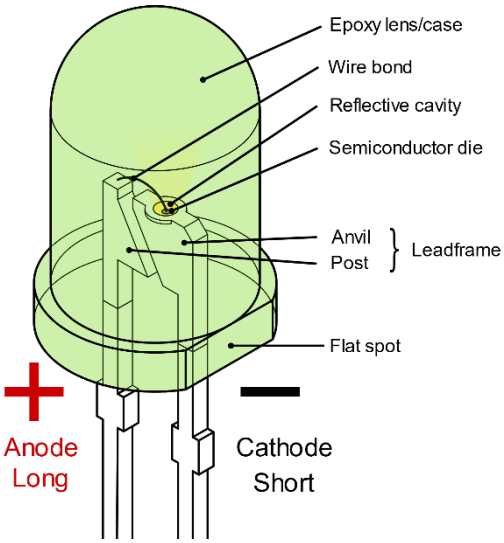
1.2.2 Connecting an LED

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

We want to control real outputs with our Arduino. A good way to tell if an output is controlled or not is to put a light emitting diode or LED[4] on the output. Then this lights up when the output is controlled.

But how do you connect something like that? Let's take a look at how an LED works:

In principle, the more voltage is applied to an LED, the more current flows through the LED. The more current through the LED, the brighter the LED shines. So far so logical. The problem is the characteristic curve of the LED: If you apply the voltage in the wrong direction, nothing ever happens - no current, no glow. Therefore, we have to pay attention to the polarity - where do we connect plus and where minus? The connections are marked: The +-terminal (anode) is a little longer. The –-connection (cathode) is shorter and has a flattening at the housing (mnemonic: a minus is "flatter" than a plus sign):



[5]

And if you apply the voltage in the right direction, then almost nothing happens at low voltages. At "suitable" voltages the LED shines well and at a slightly higher voltage so much current flows that the LED is destroyed. This happens quite fast, because an LED (like all diodes) starts to conduct

[4] LED: Light Emitting Diode

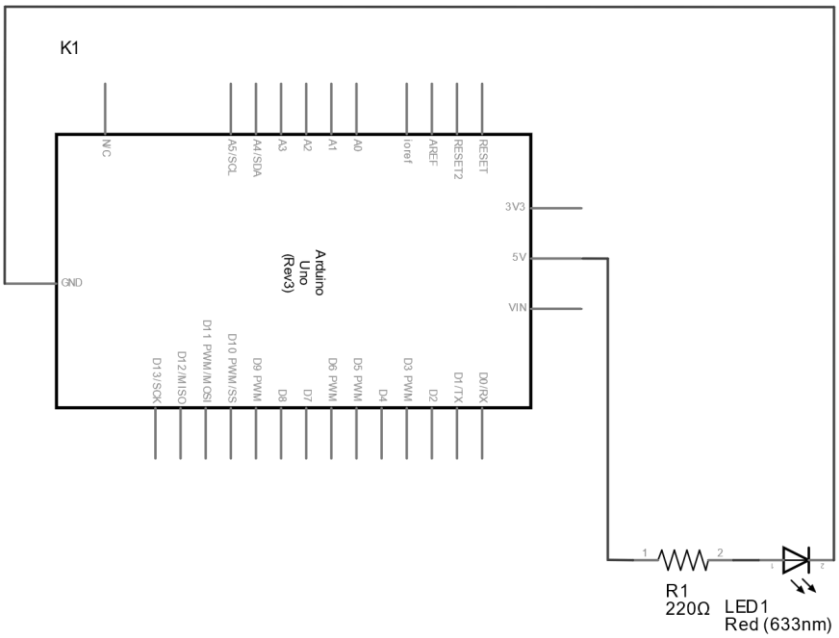
[5] Source: By Inductiveload - Own work by uploader, drawn in Solid Edge and Inkscape., Public domain, https://commons.wikimedia.org/w/index.php?curid=6431789

current at a certain "forward voltage". So, if we don't have a suitable voltage available, the LED is immediately destroyed.

To make matters worse, red, green, blue, and white LEDs all have different flux voltages. Also, two LEDs of the same color have different values (manufacturing dispersion). This doesn't make things any easier. So, you need something that limits the current appropriately.

In the simplest case, this is practically a "bottleneck" for the current, which ensures that the current flow is throttled - as in a water pipe. Such a bottleneck is called a "resistor". One speaks of operating an LED with a "series resistor". The series resistor depends on how much voltage is available in total. The more voltage, the more must be choked. With the voltage available to us of $U_{TTL} = 5V$ and the used light emitting diodes a series resistor of $R = 220\Omega$.

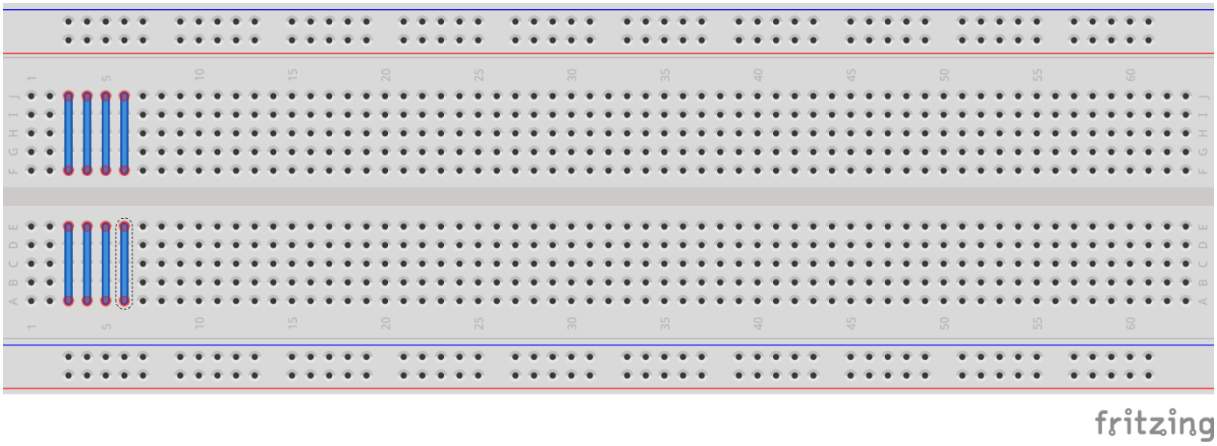
So, we're supposed to build something like this:



How do you actually recognize the resistance value? They are all round things with two knuckles without a number. The secret is the color code. Each axially wired resistor has a color code in the form of colored rings. This means that it can be read from all sides, without it mattering whether the writing happens to be on top at the moment. The color code is structured according to tables (see appendix).

The best way to do this is to use our "breadboard". This breadboard is for circuits that are not yet fully baked and for verifying circuit variants. You can assemble and try out circuits there completely without tools - ideal for us. So, we will use that to build our circuits.

Breadboards, or "plug boards" have a series of holes. Some of these holes are connected to each other, others are isolated from each other. In the next picture you can see how the holes are connected to each other.

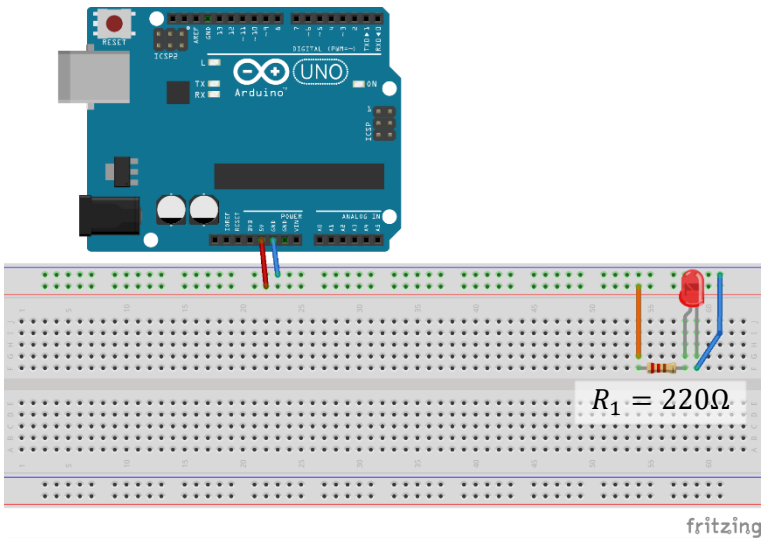


The outer rows are marked with red and blue. These are plated through. Each row by itself. These are to be used to bring a power supply anywhere on the board.

At the middle area there are many holes, where all holes between each other are connected. In the picture, this is highlighted with a few blue lines - but all rows are constructed in the same way.

The "feet" of our electronic components fit exactly into these holes. The standardized grid dimension of $1/10\ inch$ $(2{,}54mm)$ is to thank.

With this, we now make our LED light up by building the following circuit:



As soon as the Arduino is powered (e.g. via the USB socket) the LED should light up.

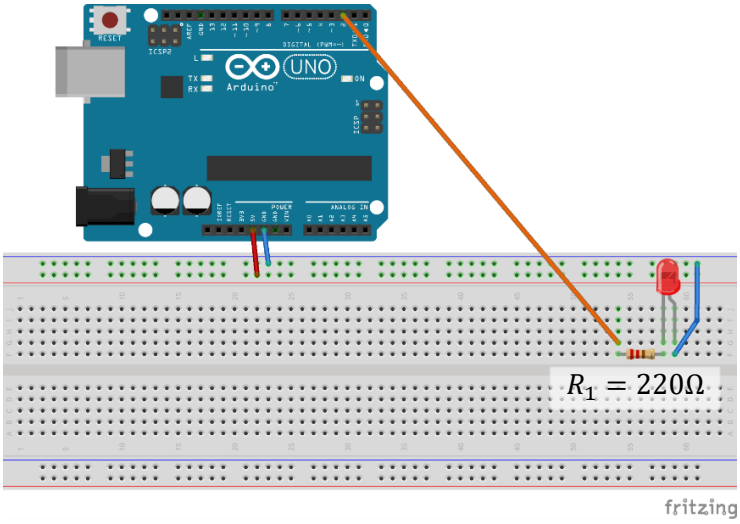
1.2.2.1 Exercise "LED connect

	Rebuild the circuit described above, so that the LED lights and nothing "burns off".	

1.2.3 Control of a digital output

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

Instead of leaving the LED always on, we want to switch it on and off again with the controller. To do this, we "marry" the previous two exercises. First, we rebuild the circuit. It should look like this:



And from the first lesson we take the program, save it under a different name and change it so that it is not the internal LED that is controlled, but the external 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
}
```

You can see that the changes are very small: The pin for the LED has changed from `LED_BUILTIN` to the (newly defined) constant `LED_PIN` - done. The external LED is already blinking - in a color that we like.

1.2.3.1 Exercise "Digital Output

	Create a circuit as described above, including a program that controls the external LED, upload it to your Arduino and demonstrate it.	

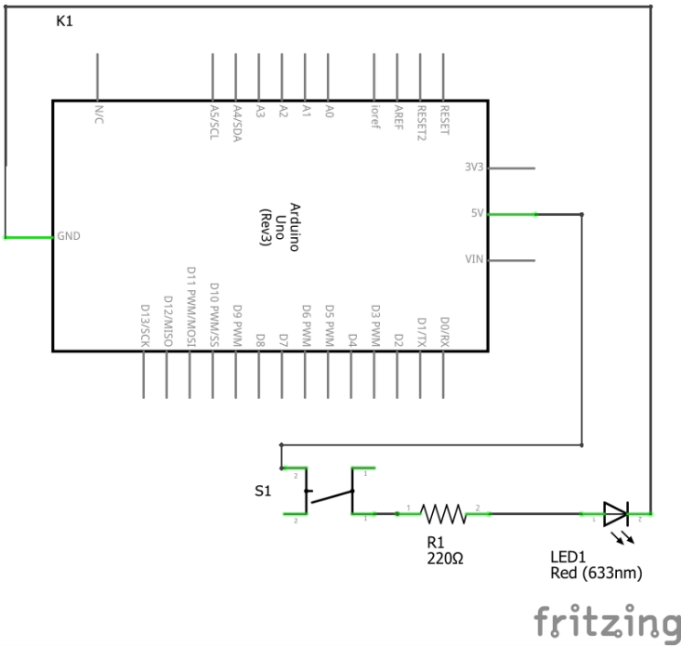
1.2.3.2 Mandatory task "Digital Output

	Install a second LED. This should always light up when the first is not lit and vice versa. Modify the program accordingly.	

1.2.4 Using a push button

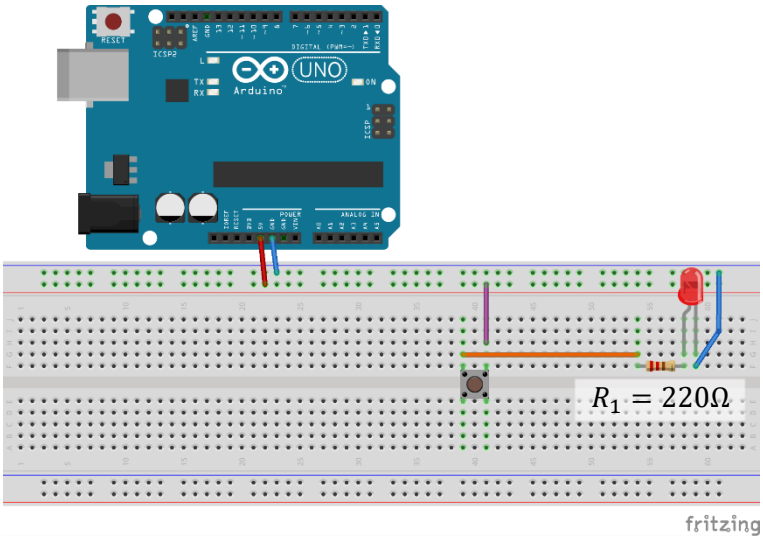
Video Link 8: https://youtu.be/L3gwYoOS7PQ

A lamp that is always on is of course not exactly what one imagines by sustainability. Turning off the lights is a natural desire. What you have to do is to interrupt the circuit. This is best done with a contact. The contact is either closed (in which case current can flow) or open (in which case no current flows). We have such a contact in the form of a "push button". A push button does not latch - as long as you press it, the contact is closed - if you release it, the contact opens again[6] .
We will now integrate the pushbutton into our test circuit:



With the push button we should be able to switch the LED on and off. If the pushbutton is pressed, the LED lights up. If the contact is open, the LED is off. The inconvenience is that we must keep the button pressed if we want our LED to light up.

On the breadboard it looks like this:



[6] In contrast, there are also "switches". These have two stable states and you switch the contact. The actuating force can disappear again afterwards.

1.2.4.1 *Exercise "LED pushbutton*

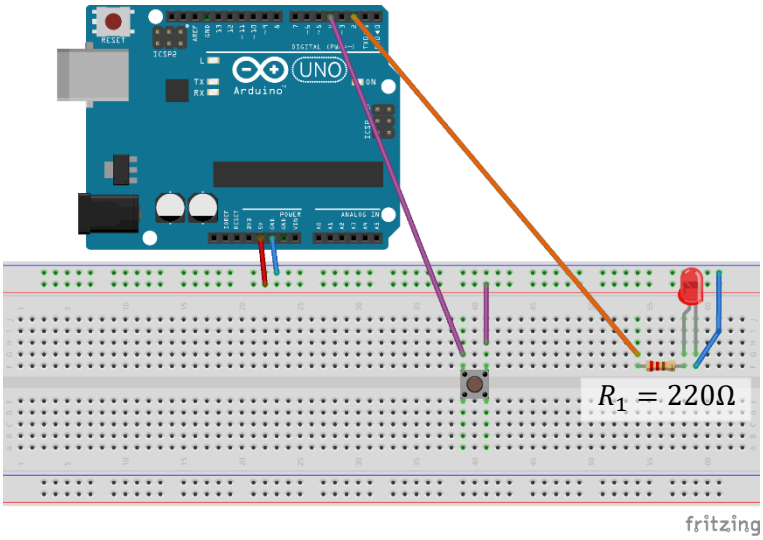
	Rebuild the circuit described above so that the LED lights up and can be switched.	

1.2.5 Reading a digital input

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

Now we want to reprogram this behavior. For this we have to read the state of a switch for the first time and set a corresponding reaction. So, we want the button to either bring us 5V to an input or not. If we read in logic 1, then we set an output and the LED lights up. If we read in logic 0, then we turn the LED off again.

To do this, we rebuild the circuit a little:



The switch brings us either 5V to pin 4 or pin 4 hangs in the air. The LED is again controlled via pin 2. So, our input is pin 4 and our output is pin 2 again. We design our program accordingly:

```
#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
  }
}
```

This time we have defined two pins in the `setup()`. One as usual as output (via the constant `LED_OUT` this is pin number 2) and one as input (via the constant `SWITCH_IN` pin number 4). So exactly how we connected them.

In `loop()` the input is read first. This is done with the command `digitalRead`. You only need to specify the corresponding pin number and you get the status of this pin. If there is a voltage of 5V you will read `HIGH` (1). If there is a voltage of 0V, `LOW` (0) will be read.

The `if` statement distinguishes `TRUE` or `FALSE` in operation. However, any value not equal to `0` is interpreted as `TRUE`. And the value `0` as `FALSE`. This means that the output is set as soon as `HIGH` (1) is read in, and the output is cleared as soon as `LOW` (0) is read in. The LED switches accordingly.

That should make it work, right?

1.2.5.1 Mandatory tasks "LED push button controller

	Rebuild the circuit described above, program the program and test for proper operation. Does the program work as expected, or does it sometimes show "strange" behavior?	

1.2.6 Pulldown and pullup

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

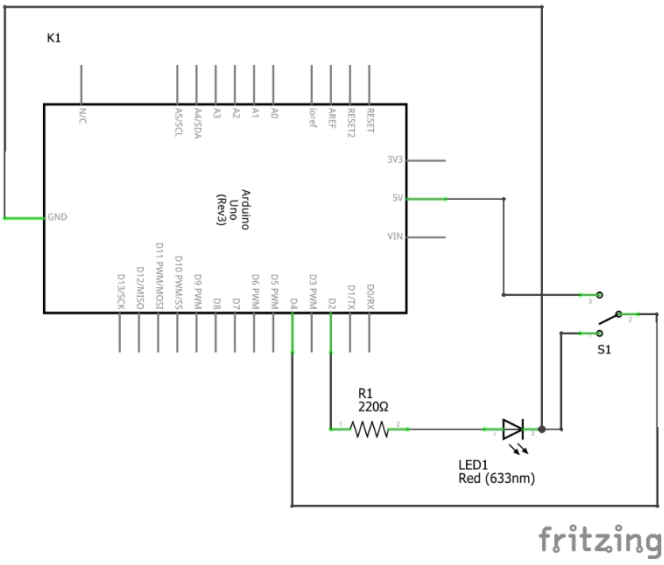
Why does the circuit in the previous exercise not work optimally and always shows strange LED flickering? The problem is the open position; that is, when the input is not connected to `+5V`, but simply "hangs in the air".

Thus, the potential of the input is not defined. And if the potential is not defined, the potential difference and thus the voltage between the input and the reference potential (ground, or `GND`) is also not defined.

Sometimes it works. Sometimes the input wire happens to be a little "charged" and the LED won't go off. When you tap the wire, the LED flashes - these are all signs of such a potential problem.

Let's look at possible solutions:

- We do not simply switch "off" but back and forth between `+5V` and `GND`:

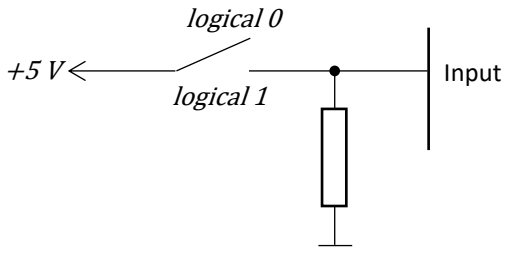


This would be a relatively good solution. Such a contact is called "changeover contact" because it can change between two connections[7] . The potential is always defined and no unnecessary energy is consumed. Only at the short switching time the potential is not defined. The big disadvantage of this method, however, is the additional wire needed from `GND` to the switch. This requires space and increased effort in routing and planning. Therefore, such solutions are not used, especially for integrated circuits. In other industries, such as the automotive sector, this is a standard solution[8] .

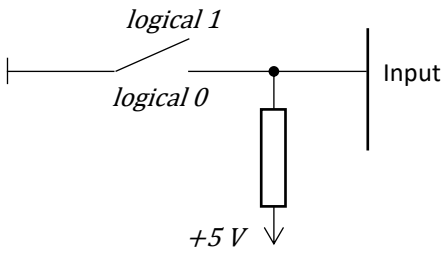
[7] The pushbutton used above has a so-called "normally open contact" - this means that when it is pressed, it is closed, i.e. conductively connected. In addition to "normally open" and "changeover" contacts, there are also "normally closed" contacts. These are normally connected and open when actuated. In English, this is called NO (normally open) or NC (normally closed).

[8] Sometimes you can observe strange light plays at the rear lights of cars. On one side, other lights shine instead of the brake light, but only dimly. Or instead of the turn signal tail light and brake light flash around. Then it's a ground problem. At this taillight the `GND` wire is broken and the current runs back through another light (which is switched to `GND` by the changer). Thus, both shine, but only weaker, because the voltage of usually 12V in the car remains the same.

- Pull resistors:



Pull-down circuit with pull-down resistor



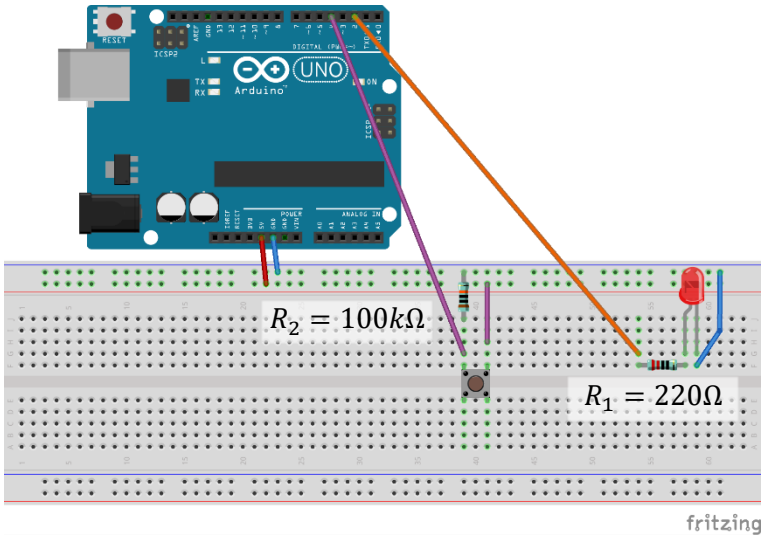
Pull-up circuit with pull-up resistor

This is where additional resistors are used. Let's have a look at the pulldown resistor: This is connected between the input and `GND`. But this also means, if you bring `+5V` to the input with the switch, a current runs through the resistor. The resistor must be big enough, that only a small current flows, which doesn't disturb us (somewhere in the higher $k\Omega$-range, we will use $100k\Omega$ use.
Alright, so when the switch is closed the 5V are at the input and a small current runs away through the pulldown resistor. If the switch is now opened, any charge on the conductor is dissipated to ground via the resistor. The input is pulled to ground in a defined way.
This is exactly the behavior we want: If the switch is closed, we definitely hang on `+5V`. If the switch is open, we are defined to 0V - fits! So a pulldown resistor pulls the potential down to the reference potential - hence the name.
In the case of pullup, the resistor pulls the potential up to the `HIGH` level. So, it hangs between the input and `+5V`. The contact now connects the input to `GND` instead of `+5V`. This pulls the input to 0V in a defined way when the switch is pressed. So now we have the reaction reversed: When the button is pressed, the input is cleared. But this is unimportant, the main thing is, that we get sure, that the switch was pressed. What `LOW` and `HIGH` really mean is defined by our program.

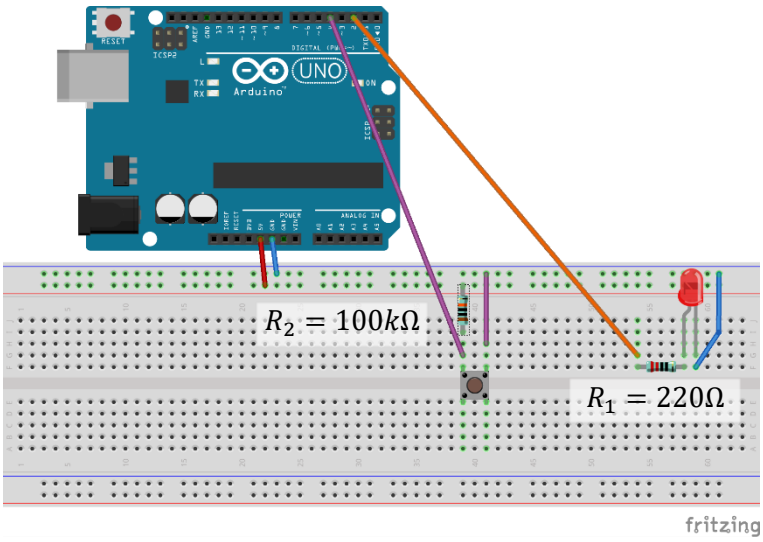
Normally pull resistors are used. Why? Because you need a whole component more and not only a line. Yes, that's true, but a resistor is very small to produce (and therefore cheap), and `HIGH` and `LOW` levels are available at the input anyway, the effort is limited.

In addition, such a resistor can be integrated in the chip. This means that you can't see anything from the outside. From the data sheet you can then find out whether a pull resistor is installed or not. So, the inputs with the pull resistors work "transparent" for us. What we already need to know is whether it is a pull-up or a pull-down resistor.

What would this look like in our circuit? Let's first build the variant with pull-down resistor ($100k\Omega$):



Then the variant with pullup resistor (also $100k\Omega$):



1.2.6.1 Exercises "Input external Pull Resistor

	Rebuild your circuit to work with a pull-down resistor. Consider whether you need to adjust the program and make the adjustments accordingly.	

	Rebuild your circuit to work with a pull-up resistor. Consider whether you need to adjust the program and make the adjustments accordingly.	

1.2.7 Internal pull-up resistor

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

Well, it is an additional circuit part which we consider in our circuit. Fortunately, resistors can be made very small, and you can integrate them well directly on the silicon chip. That's exactly what happened with our Arduino: There is a built-in pull-up resistor there.

But then why didn't our circuit work without external circuitry when we read the input? It's because we can choose to use the internal pull resistor or not. The difference is in the definition of the pin. Let's replace the line:

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

with

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

the internal pullup resistor becomes active - and we can operate the circuit without a resistor.

1.2.7.1 Mandatory tasks "Input with internal Pullup Resistor"

	Rebuild your circuit without resistor. And change the program to the internal pull-up resistor. Consider whether you need to adjust the program and make the adjustments accordingly.	

1.2.8 Memory circuit

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

So far, the program, or rather the whole controller, has little sense. The same result could be achieved with practically the same components, only much cheaper, namely without controller and without program. In addition, the behavior also makes only very limited sense. Where can you need something like that?

That's why we're going to adjust the behavior of our program a little bit now. What we want:

The LED should light up when the first button is pressed. The LED should go out when the 2nd button is pressed.

1.2.8.1 Mandatory tasks "Memory On/Off Light

	Reprogram your program: The input is to be operated with internal pull-up resistor. The behavior should work as described above.	

1.2.9 The Serial Monitor

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

Well, that wasn't so easy for most people. Especially the search for the error is not easy at all. It would be nice if you could see a little bit into the program flow.

Fortunately, our Arduino has this possibility. You can write "messages" to each other. And this on the so-called "serial monitor". Serial because we are connected to a Universal Serial Bus (USB). And "monitor" because this shows what's happening on the serial bus. And when we send data to ourselves on the serial port, we can display it with the Serial Monitor.

To output data on a serial interface, it must first be "initialized" by the program. This sounds wilder than it is, because only one command is necessary, and this is:

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

Strictly speaking, the `begin` method of the `Serial` object is called here. The value in brackets is the communication speed in characters per second ("baud") - also called baud rate. In the above example we take 9600 characters per second, where one character always consists of one byte (i.e. 8 bits). Normally this initialization is in the `setup()` part of the program.

To then output something on the serial port we can use the methods

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

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

to do this. In doing so, `print` tries to display as much as possible as readable text. For example, numbers like `integer` or `float` are converted to a string.

This allows us to do some output at a point in the program that seems appropriate to us. For example, to determine that we are running through a certain program branch:

```
Serial.println("Am now at position 1.");
```

Or we can also print out values of variables (note the use of `print` and `println`):

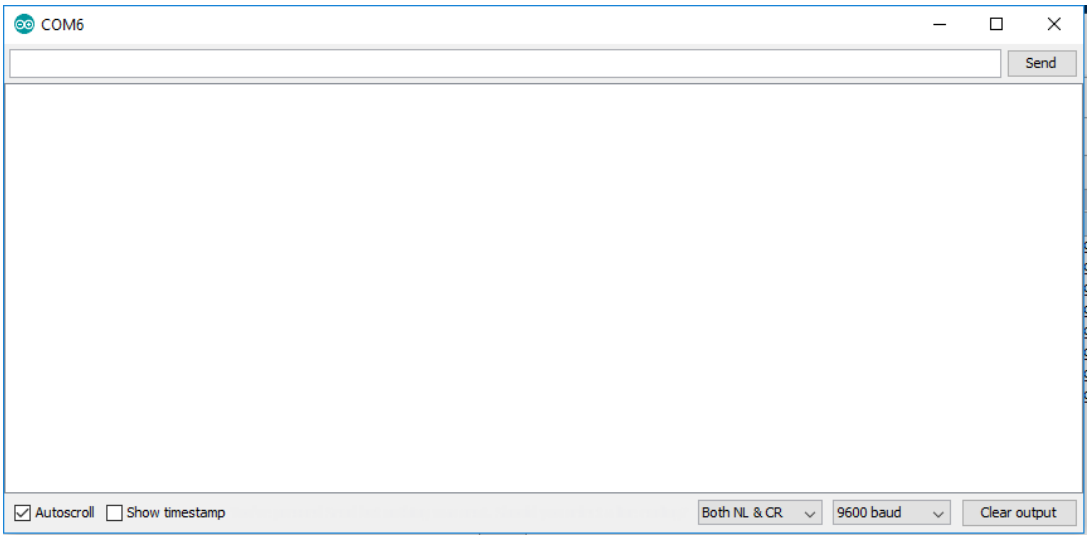
```
Serial.print("i = ");
Serial.println(i);
```

Let's just take an example here. We now simply use the above program and extend it with the serial interface:

```
#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
  }
}
```

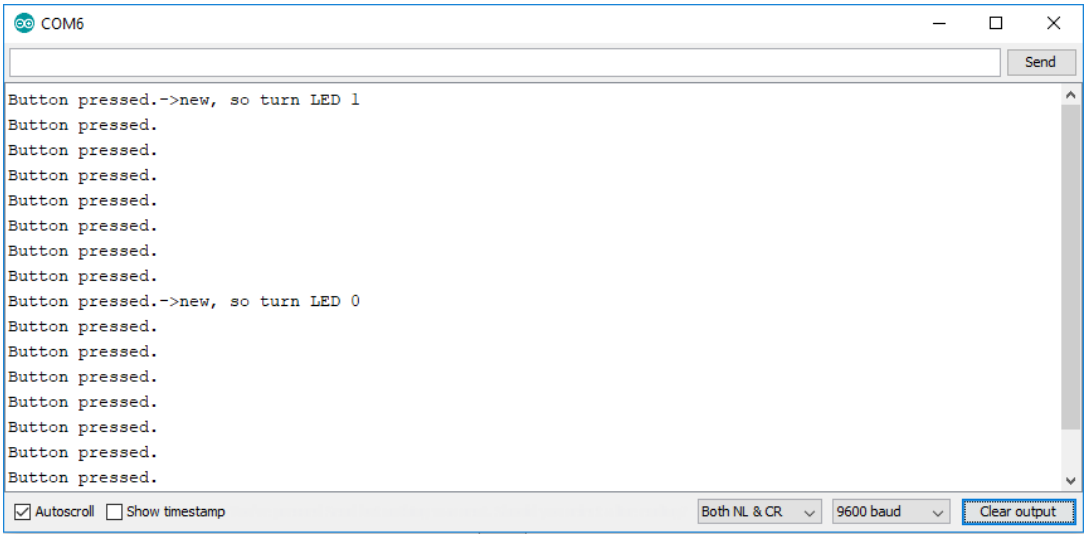
When we load this program, it works the same as before. We don't really notice any difference. Now where do we find our output on the serial port? To do this, we need to open the Serial Monitor. This is located on the top right of our Arduino IDE window ( ).

Clicking on this icon opens the Serial Monitor:



At the bottom right we have to set the appropriate transmission speed. Because we have selected 9600 baud in the program (with the method `Serial.begin(9600)`) we must also select 9600 here. If we have a wrong baud rate, only strange characters appear on the monitor.

If everything fits together, we should be able to see the corresponding output in the serial monitor window:



1.2.9.1 Exercise "Serial Monitor LED Output

	Run the program described above. Look at the output of the serial monitor. Explain why strange effects (short flickering) occur from time to time.	

1.2.10 Time function 1 (simple aisle light)

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

We now want to link our inputs and outputs differently. The hardware structure remains the same, only the behavior should be different. With a software change, you can simply create a completely different behavior. That's a big advantage of controllers like these: you can get different results with the same hardware setup. The logic is purely in the software.

What should happen? When you press the button, the LED should turn on. After an adjustable time, it should go off again by itself. So, we want to program a kind of aisle light.

The word "wait" simply brings to mind a delay. Our Arduino knows the command

```
delay(timeMs);
```

The program waits the specified number of milliseconds. Sounds perfect. So, we try to adapt our program accordingly:

```
#define LED_OUT 2
#define SWITCH_IN 4
#define AISLE_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) AISLE_TIME)/1000);
    Serial.println(" s");
    delay(AISLE_TIME);
  }

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

This should actually complete the task.

1.2.10.1 Exercise "Corridor light simple"

	Build and program the aisle light mentioned above using this simple method. Try to identify weak points and name them.	

1.2.11 Time function 2 (complex aisle light)

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

The biggest disadvantage of the `delay()` function is that the controller does nothing during the waiting time. No inputs are processed, and no other decisions are made. An employee who can do nothing but wait during a wait would probably not be allowed to be called a colleague for long. So, it has to do better than just waiting mindlessly. Our Arduino also knows how many milliseconds have passed since the program started. For this you can use the function

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

to use. A numerical value of the type `unsigend long` is returned. If our controller is in operation longer than this maximum value (`4294967295ms` correspond to about 50 days - exactly 49.71 days), the function returns smaller values again.

We now want to use exactly this function in an adapted program:

```
#define LED_OUT 2
#define SWITCH_IN 4
#define AISLE_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) < AISLE_TIME) {
    Serial.print("Wait Time: ");
    Serial.print(((float)(pressTime + AISLE_TIME -
                  millis()))/1000.0);
    Serial.println(" s");
    digitalWrite(LED_OUT, HIGH); // turn the LED on
  } else {
    digitalWrite(LED_OUT, LOW); // turn the LED off
  }
}
```

Where is the difference in function? Here we can do something during runtime. In this case, the only thing that happens is the output of the wait time on the serial monitor. Exactly in the line is also a so-called type cast:

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

The instruction `(float)` converts the result into the type `float`. So here the calculation `pressTime + AISLE_TIME - millis()` is performed in the type `unsigend long`, then converted to a floating point number of the type `float` and then divided by 1000.

1.2.11.1 *Exercise task "Corridor light advanced*

	Build and program the above aisle light with this better method.	

1.2.11.2 *Compulsory "Corridor light turn-off" task*

	Extend the program by the following function: the aisle light should also be actively switched off during runtime by pressing the button again.	

1.2.12 Personalize settings

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

Let's imagine we want to personalize our aisle light. Not everyone wants the same amount of time lit. This may not be so important for an aisle light, but let's imagine we want to bake bread in an oven. There you need different durations for large and small loaves. Let's look at our program: With the line

```
#define AISLE_TIME 10000
```

we define the time duration - so in this example 10000 ms, so 10 seconds. If we want to change the time, then we only have to change this program line and the light shines another time.

So we have to change the program, then translate it, put it into the Arduino and then start it again - quite complicated.

Of course, we can also trouble a variable and change it. Therefore, I have written the following program:

```
#define LED_OUT 2
#define SWITCH_IN 4
#define AISLE_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 = AISLE_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
    }
  }
}
```

What happens here? Nothing is different in the `setup` part. The difference is in the `loop` part. First, we have a few new variables:

```
static unsigned long waitTime = AISLE_TIME;
```

Here the waiting time is stored. In the rest of the program now only `waitTime` is used instead of `AISLE_TIME`. And because it is now a variable instead of a compiler constant, it can also be changed. Initialized (at power on or reset) this variable is still initialized with the constant `AISLE_TIME`.

Now we just need to create a way to change the value of the variable as well. This happens as follows: If you stay longer on the button (longer than `LEARN_MODE`, so here 1000ms) the controller switches into a learning mode. The light then remains on until it is switched off by pressing the button. The time that has elapsed between the activation of the learning mode and the button press is then used as the new delay time.

For this purpose, the time is always stored when the key is not pressed. This happens in the line

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

However, if the key is pressed, the value in `pressLearn` remains unchanged. So, you can compare the value with the current time and if the difference is greater than `LEARN_MODE`, then you switch to learn mode. The comparison happens here:

```
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);
}
```

At the same time the LED flashes with 200ms, so that it is also clear: Now we have changed to the learning mode.

If the controller is in learn mode, the LED no longer goes out automatically. Instead, the serial monitor shows how much time has passed:

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

Only when a key is moved again, the new time is stored in `waitTime`:

```
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 *Exercise "Corridor light adjust"*

	Build and program above aisle light using this method. Show that you can change the time with this program. Try to identify weak points and name them.	

1.2.13 Saving to non-volatile memory

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

Let's imagine we have a radio and it's tuned to our favorite station. And every time we turn it up again, we have to search for our favorite station again - a bit annoying, isn't it? With such settings as in the previous chapter, it would be nice if these "non-volatile", would also survive a reset or voltage off.

For this purpose, our Arduino offers a so-called flash area. It works practically like a USB flash drive, only it is built in. This is the so-called EEPROM[9] . A memory location can be written on average about 100,000 times. After that there is no guarantee from the manufacturer that the correct value is inside - the EEPROM then becomes "forgetful". A behavior that SSD disks also have, by the way. That is why there are sophisticated algorithms that are supposed to ensure an even "wear" of the memory locations ("wear leveling").

But, to store a setting value, that's good enough. Suppose we change the value every day, we would be able to do that for more than 273 years .[10]

To be able to use this EEPROM, we need a so-called "library". This is a part of a program that someone has written and made available to us. We borrow the function of this program part - hence the name. To use a library we need the `#include` statement. And of course, the library we use must be available on our computer. The one for the EEPROM is included by default and is called EEPROM.h. So, our first line of code is now

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

With this we integrate the library into our program. A description of the functions of the library can also be found in the appendix (4.1.4.3). Our above program from chapter 1.2.12 now changes as follows:

[9] EEPROM: Electrical Ereaseable Programmable Read Only Memory: can be electrically erased and rewritten. Predecessors were:
ROM: Read Only Memory - came out of the factory and that was it. If you wanted to make a change you had to buy a new ROM.
PROM: Programmable Read Only Memory - You buy empty memory blocks, you can program them, but you can't erase them. If you wanted a change, you needed a new empty block.
EPROM: Eraseable Programable Read Only Memory - Here you can erase the memory content again. This was done with UV light. There were light chambers, in which one put the memory. After a few minutes (10-15) "sun studio" the memory was empty again.

[10] If you want to change the value 1x per second, this results in a lifetime of not even 28 hours! It is therefore important to be careful here.

```
#include <EEPROM.h>

#define LED_OUT 2
#define SWITCH_IN 4
#define AISLE_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 > AISLE_TIME_MAX) waitTime = AISLE_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 Exercise "Corridor light adjust EEPROM"

	Build and program above aisle light using this method. Find and explain the differences from the previous chapter.	

1.2.14 Analogue outputs

Video link 19 (PWM): https://youtu.be/ABSYgUooc3A Video-Link 20 (Analog Out): https://youtu.be/1nZBLZo_wYM

We can now switch digital outputs. Often this is enough, but sometimes you don't just want to switch things on and off, you want to adjust their intensity (analog to a value, so an analog output). A good example is controlling the brightness of the smartphone's backlight. This is not simply switched on and off again, but the brightness automatically adjusts to the ambient light intensity. If it is bright, the display shines brighter. If it is dark, the display is also darker so that it does not "hit" our eyes.

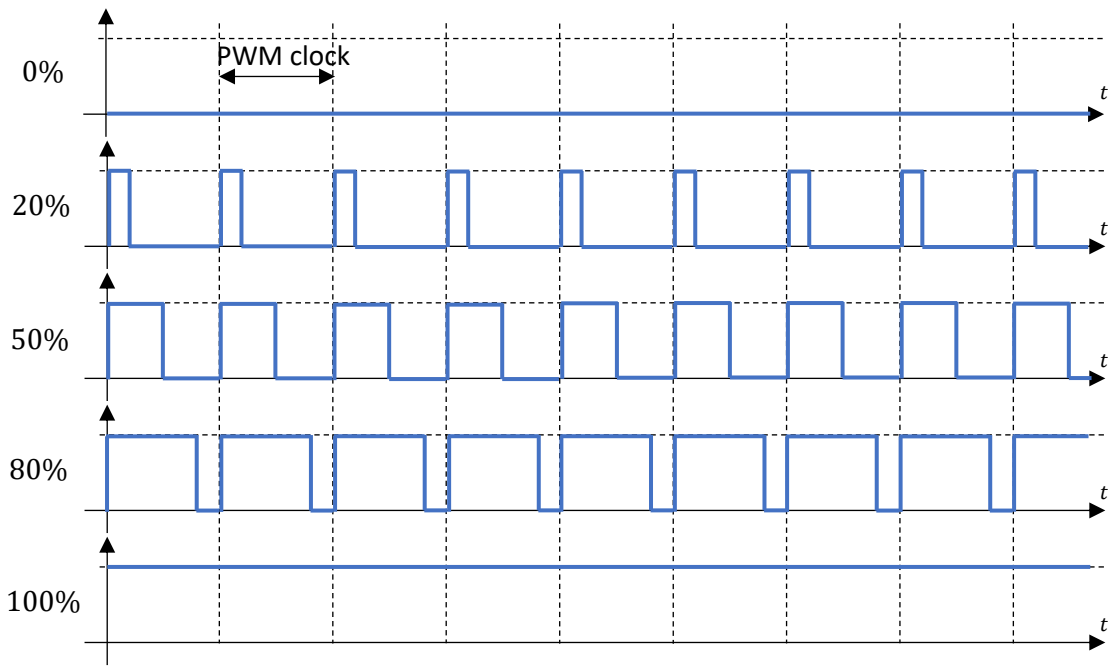
However, our Arduino cannot really "half" control outputs. Either the output is there or not. However, some outputs can be controlled with the so-called "pulse width modulation". What does that mean? Well, either the output is on or it is off. But you can switch the output on briefly and then off again briefly. Then on again and off again and so on. Depending on which part is taken by the on state and which part is taken by the off state, the output is on average more or less on.

PWM is usually done with a fixed frequency, which is the reciprocal of the PWM clock. Our Arduino Uno usually has the frequency 490Hz. So your on/off game starts about every 2ms. Some PINs even have a clock of 980Hz and are therefore even faster[11] .

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

[11] However, this is not really fast. PWM signals in the kHz range are often used.

For better clarification, here is a picture of a typical PWM. The percentages reflect the share of the switched-on state.

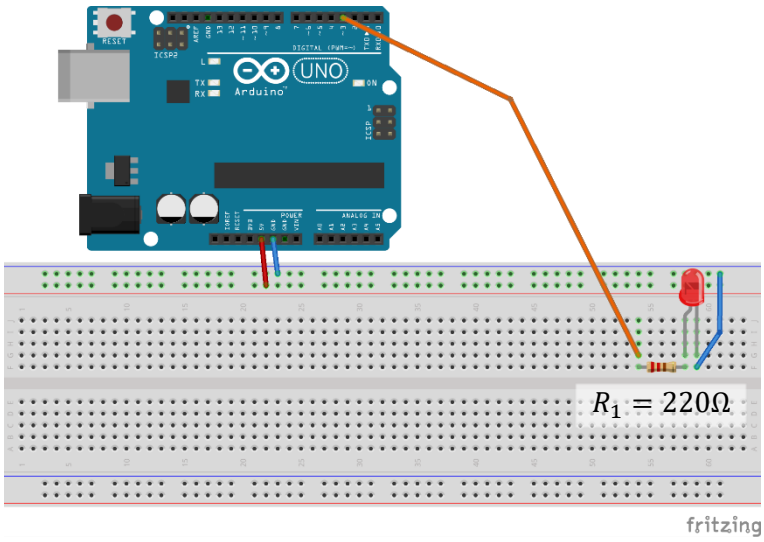


The appropriate command for driving a PWM output is `analogWrite`. We only have to tell the command on which pin and which percentage should be switched on:

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

`pin`	`int`	Pin number of the pin to be written. Only with PWM-pins this has an effect.
`value`	`int`	A value between `0` (0%) and `255` (100%)

We first use the following hardware setup:



To do this, we'll write a simple program that drives the LED using PWM. We will see what this really means:

```
#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 Exercise "Analog Out

	Build and program the above mentioned controller. Explain how the program works.	

1.2.14.2 Mandatory task "Analog Out

	Extend the circuit with a push button. If the button is pressed and held, the brightness of the LED should change. After releasing the button and holding it down again, the brightness should change in the other direction. For example, the first time the button is held down, the LED should become brighter and brighter. If you then release and hold the button again, the LED should become darker.	

1.2.14.3 Additional task "Analog Out turn-off

	The button should have the following function: If it is pressed only briefly, the LED should be able to be switched on and off. And this with the brightness that was last used.	

1.2.15 Analog inputs

Video link 21 (Voltage divider): https://youtu.be/CevOz2EN77A
Video-Link 22 (Potentiometer): https://youtu.be/5oySfAXBJNI
Video link 23 (A/D converter): https://youtu.be/CevOz2EN77A
Video link 24 (Analog Input): https://youtu.be/5oySfAXBJNI

Just as there are analog outputs (corresponding to a value), there are analog inputs. So there are inputs that are not just there or not there, but can also be there "a little bit". How much that little bit is depends on the strength of the input signal. Our Arduino has several voltage inputs (`A0-A5`). Usually there is a voltage between $0 - 5\ V$ is present. So just like our digital inputs, with the difference that not from about $3V$ to "HIGH", but that you get different numbers from the analog input. What are these numbers? Well, the smaller the voltage, the smaller the number. The higher the voltage, the higher the number. If you have $0V$ is present, you get a value of 0. If, on the other hand, 5V are present, you get a maximum number. The maximum number depends on the so-called "resolution". In our case the resolution is $10bit$. This means that the voltage value is represented by a binary number with $10bit$ is displayed:

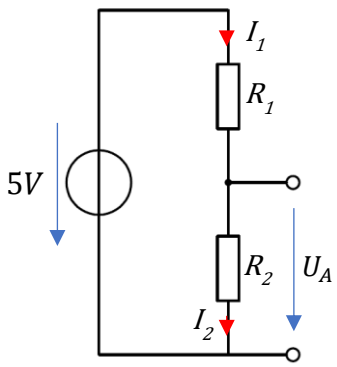
Voltage	Binary number	Decimal
$0V$	$00\ 0000\ 0000_{BIN}$	0_{DEZ}
$5V$	$11\ 1111\ 1111_{BIN}$	1023_{DEZ}

It is also quicker and easier to specify the maximum value if you simply consider how many combinations you can represent with the corresponding number of bits. Here these would be:

$$2^{10} = 1024$$

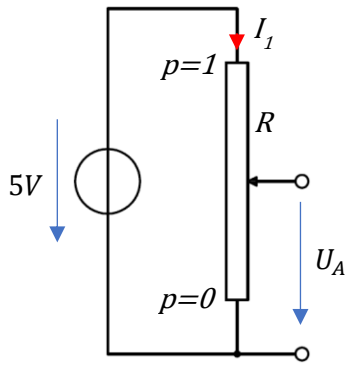
Since also 0 is also a combination, the maximum number is one less than the number calculated in this way. Whereby we come back to 1023[12] .

To test our analog input, we still need a voltage that we can use as input. Well, our Arduino provides `5V` and `3.3V`. That's nice, but if we want to test, then with many different values. That's why we need an appropriate circuit. We use the so called "voltage divider" here.



$$R_1 = (1 - p) \cdot R$$

$$R_2 = p \cdot R$$



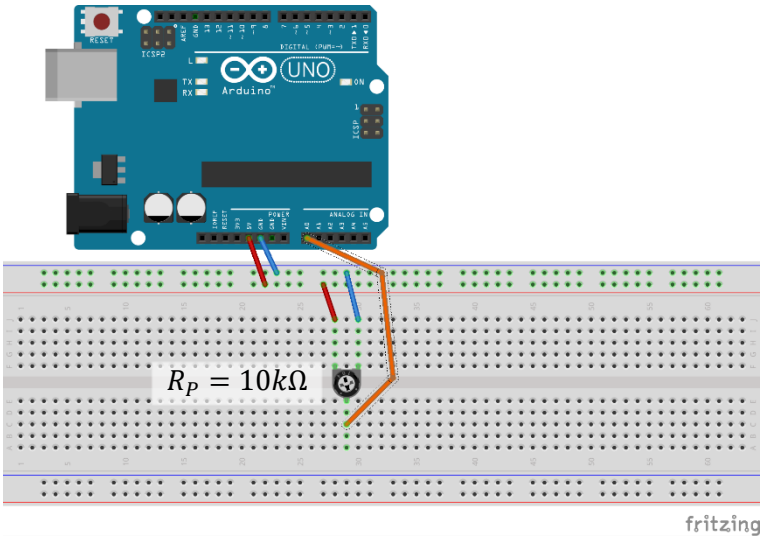
[12] Other analog/digital converters have different bit widths. Very common are $12Bit$ (value range $0 - 4095$). High quality standard converters also have 14 ($0 - 16383$) or 15 Bit ($0 - 32767$)

On the left side we see the "typical" voltage divider with two resistors. As the name suggests, a voltage divider divides a voltage into two parts. Both parts together result in the original voltage again. Why, and what the requirements are, you learn in electrical engineering. For us it is only important that a voltage divider divides the original voltage of $5V$ into one part U_A and the rest to the $5V$. U_A becomes the larger, the larger R_2 in relation to R_1 is. In the extreme case when $R_1 = 0\Omega$ is R_2 is much larger and it becomes $U_A = 5V$. In the other extreme case, if $R_2 = 0\Omega$ becomes $U_A = 0V$. For values in between, there is a voltage between 0 and $5V$ occurs. If, for example, both resistors are of the same value, then $U_A = {}^{1}/_{2} \cdot 5V = 2{,}5V$.

It is important that the two resistors in sum should have at least 1000Ω. Otherwise, the $5V$-output will be overloaded.

All right, so we get tension between 0 and $5V$. We can test our input. The impractical thing is that we always have to plug in resistors. It would be nice if we could do this stepless. Exactly for this purpose there is a so-called potentiometer (right side of the upper picture). This has a fixed resistance. With a sliding contact you can split the resistance into two parts. If you move the sliding contact, you change the ratio of the two parts - that's exactly what we want! That's why we use exactly such a potentiometer.

We have a $10k\Omega$-potentiometer in our starter set. With it we make the following hardware setup:



The outer connections are on $5V$ (`5V`) and $0V$(`GND`). The wiper connection is wired to input `A0`. This allows us to set the voltage value between 0 and $5V$ to our analog input.

Now, about the program. The command we need is:

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

`pin`	`int`	Pin number of the pin to be read (`0`=`A0`).
`value`	`int`	A value between `0` (0%) and `1023` (100%)

With this we can write a simple program:

```
#define ANA_00_IN 0

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

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

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

What does the program do? It simply outputs the read value on the serial monitor. Namely, as a value, as a percentage, and the measured voltage. Sample output:

```
Analog value: 173 (16.91 %, 0.85 V)
```

1.2.15.1 Exercise "Analog In"

	Build and program above mentioned read in the analog value. Explain how the program works.	

1.2.15.2 Mandatory task "Analog In"

	Expand the circuit in such a way that an LED is controlled. The LED should be dark when the analog value is low and bright when the analog value is high.	

1.2.15.3 Additional task "Analog In, adjustable"

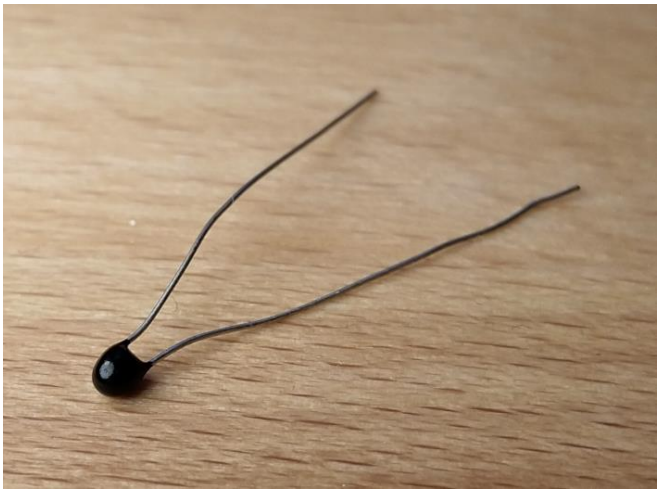
	The value for minimum and maximum brightness should be adjustable.	

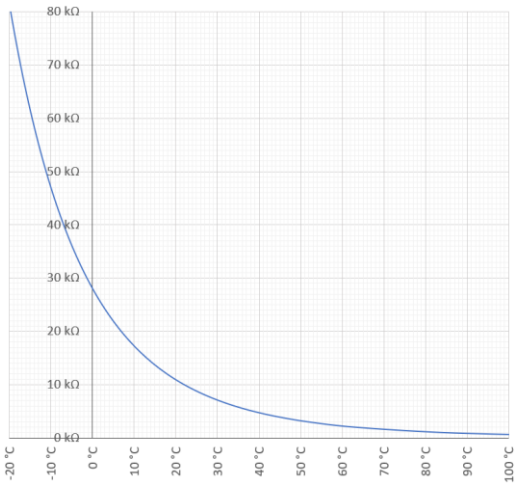
Video link 25 (Serial Plotter): https://youtu.be/LPVWyHBPlho

1.2.16 The NTC as temperature sensor

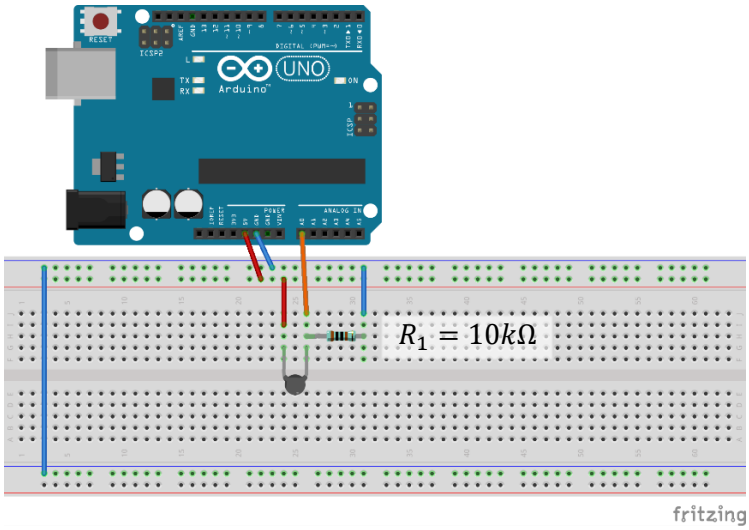
Video link 26 (component): https://youtu.be/eifkygfDF3Y Video-Link 27 (program): https://youtu.be/FZRVupykrQU

NTC means "Negative Temperature Coefficient". In electrical engineering, one hears that the resistance of a material changes with temperature. "Normal" metallic conductors have a positive temperature coefficient. This means that the resistance becomes higher as the temperature rises and vice versa. This effect is often used in temperature measurement and forms the common field of "resistance thermometers"[13] .





Various materials were tested and in the field of semiconductors, materials were actually found that have a negative temperature coefficient (the resistance increases as the material gets colder). Unfortunately, these materials are very "non-linear" (see diagram)[14] . This means the change in resistance must be awkwardly converted to temperature. Fortunately, our Arduino can do that for us. Let's make an appropriate setup (the resistor has a value of. $10k\Omega$):



[13] In automation technology we will learn more about temperature measurement and its methods.
[14] Of course, sensors with linear characteristics are desirable for ease of use.

The conversion between the read-in value and the actual temperature is programmed in this way:

```
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;
```

Exactly this formula converts the read-in value (`0..1023`) into the current temperature in $°C$ converted.

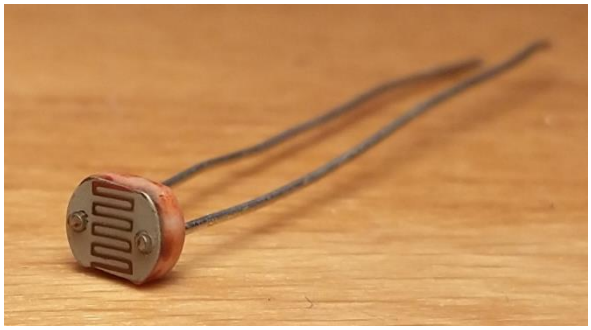
1.2.16.1 Mandatory task "Temperature Sensor"

	Assemble the NTC as described. Print out the measured temperature value on the serial monitor.	

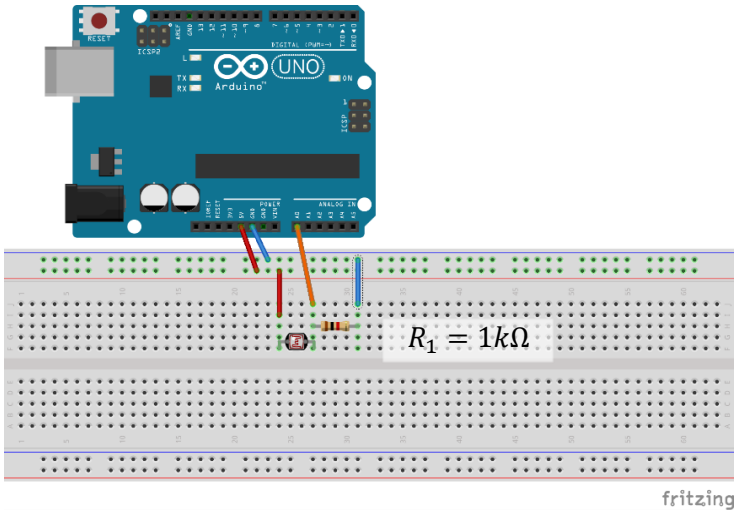
1.2.17 Brightness sensor

Video link 28 (component): https://youtu.be/ohtV-rIrAog Video-Link 29 (program): https://youtu.be/vLajecdz-Es

The brightness sensor included in our set is actually nothing more than a resistor whose value changes with the photon density (light intensity) (photo resistance). The more photons "help" the charge carriers to flow, the lower the resistance. In darkness the resistance value is somewhere around $200k\Omega$. In bright light it goes down to a few Ohms.



This is exactly the effect we want to exploit. We will build a voltage divider, where one of the resistors is our photoresistor. Please note: the minimum load should be around $1k\Omega$. If the resistor can go down to a few ohms, then we just choose the 2nd resistor with $1k\Omega$. It would be still nice if the measured value becomes higher with increasing light incidence. Therefore, we will use the photoresistor "above", as R_1, we will use. Exactly then the lower resistance becomes larger and larger in relation and the read value increases.



So we're going to do the hardware setup (we're basically just replacing the potentiometer with this voltage divider, if you still have the LED set up then leave that as well).

From $5V$ (`5V`) it goes via the photoresistor and a $1k\Omega$-resistor to $0V$ (`GND`). At the junction it goes to the analog input (`A0`).

So, if it is dark, there should be a small value there. If it is bright, the value will be higher.

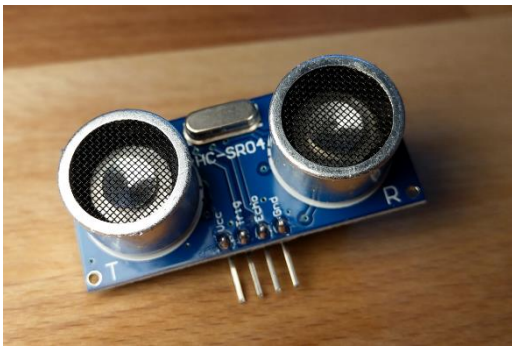
Without changing the program (same program as with 1.2.15) we should be able to observe the function at least in the serial monitor.

1.2.17.1 Mandatory task "Photoncell"

	Assemble the brightness sensor as described. Also use the LED as described in the previous chapter. Describe the effect and compare it with the display brightness control on your smartphone. Does it work the same way or exactly the wrong way?	

1.2.18 Distance sensor

Video link 30 (component): https://youtu.be/k1YEiX85cZU Video-Link 31 (program): https://youtu.be/ghoC1PRgMqw

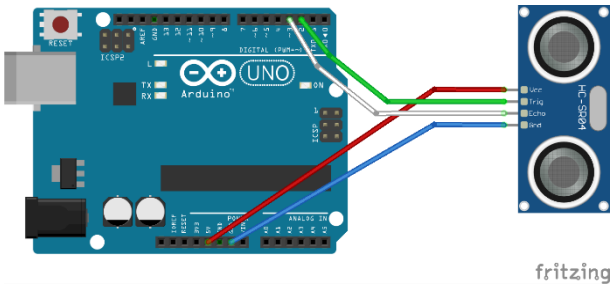


There are many ways to measure distances. One possibility is a tape measure. But that is difficult to automate. That's why we use a different approach here: We generate a sound and wait for the corresponding echo. The longer we have to wait for the echo, the further away the reflecting surface has to be. And because we want to do without stupid beeping, we use a wavelength that we as humans can no longer perceive - ultrasound. The corresponding ultrasonic sensor in our starter kit is the `HC-SR04`. It is characterized by separate transmitting and receiving devices. There is a dedicated loudspeaker (marked T) and a microphone (marked R)[15] . The sensor needs a power supply ($3.3V$ would be enough, $5V$ but also do nothing). The measurement can be started by a signal on the `TRIG` pin. After that, there is a signal

[15] There are also sensors with combined technology. These are smaller, but because it takes time to switch from speaker to microphone mode, they can only measure from a certain distance.

on the ECHO pin for as long as it takes to receive an echo. So, if we measure the time of the ECHO pin, we know the propagation time of the signal. If the speed of sound is known, we can infer the distance.

So, on the Arduino we need two pins: an output to start the measurement and an input whose duration we need to measure. Let's build a corresponding circuit.



And let's try to write a simple program to do this:

```
#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("Measured duration : ");
    Serial.print(duration);
    Serial.println(" µs");
  } else {
    Serial.println("! no echo ! ");
  }

  delay(500);
}
```

There are only two things that are new about the whole thing:

```
delayMicroseconds(timeMs);
```

Parameters:

`timeMs`	`unsigned long`	Waiting time of the program in microseconds. No other code is executed! Practically the same as `delay` but with fine subdivision.

The measurement of the pulse duration of the input is done by a single command:

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

Parameters:

`pinNo`	`int`	Number of the pin to be read
`value`	`int`	Type of the pulse to be measured: `HIGH` a pulse duration from `GND` to `+5V` is to be measured `LOW` a pulse duration from `+5V` to `GND` is to be measured
`timeout`	`unsingned long`	Timeout for waiting for the start of the pulse in ms. Default value: 1000ms
`duration`	`unsingned long`	Measured pulse duration at the specified pin in μs

1.2.18.1 Mandatory tasks "Distance Measurement

	Build the circuit and use the program as a basis. However, not the duration, but the distance of the reflecting object in cm should be output. Consider how you could calculate this for ultra-sound pulses.	

1.2.18.2 Additional task "Temperature compensation

	The measurement method is temperature sensitive (a different temperature will give a different measurement result). Try to correct the measured distance with a measured temperature.	

2 Virtual and augmented reality systems

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

2.1 General

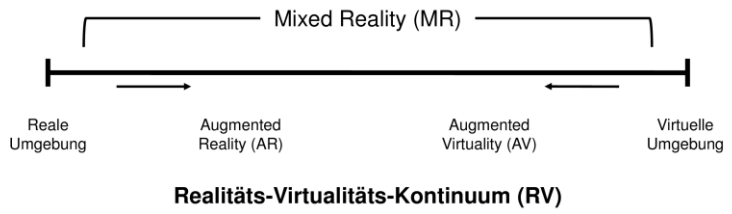
2.1.1 What is the difference between VR and AR?

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

In a virtual reality application, everything comes from the computer: three-dimensional images, sounds, physical properties of the displayed things, possibly even physical effects on the body (heat, wind, ...) ... All these things are calculated in real time and presented to the user via a suitable HMI[16]. The user is thus immersed in a purely computer-generated world and leaves real reality behind. Depending on the sophistication and complexity of the HMI used, this works more or less well.

With augmented reality, the user is not disconnected from the real world. The goal is to bring much more additional information directly into the field of vision. Thus, only parts of the presented reality are generated in the computer, and these are integrated as seamlessly as possible into the real reality - reality is "augmented"[17] . Special HMIs are used here as well, and again, the more complex the HMI, the more successful the merging of reality and virtual content.
Unlike VR systems, an AR system must also be able to sense its environment and then calculate how the virtual things need to be blended in to generate a coherent representation. This is more effort than with pure VR devices, which explains the price difference.

The degree of virtuality can be captured in the reality-virtuality continuum according to Paul Milgram et al:



[18]

Here are a few examples to clarify the concepts:

- Pure reality is a person who goes to the supermarket to shop.
- Augmented reality are eyeglass lenses on the inside of which a computer projects the user's shopping list in such a way that the user gets the impression that the shopping list is written on the wall of the supermarket. Reality is here enriched with virtual information.
- Augmented virtuality is a computer game played through a VR helmet that transmits the door intercom to the headphones when the doorbell rings. Virtuality is enriched here with real information.
- Virtual reality is a computer game played through a VR helmet that does not react to the user's real outside world.

[16] HMI: Human Machine Interface
[17] Augmented: enlarged, increase, improve, enrich.
[18] By Aera - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=78080198

2.2 Virtual reality

2.2.1 Requirements

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

When creating a virtual world, you can define some requirements that should be met. Some possible requirements are for example:

- Immersion:
  Immersion describes the embedding of the user in the virtual world. The perception of oneself in the real world is reduced and the user feels more like a person in the virtual world. This can be achieved by a challenging and exciting design of the virtual world, e.g. by a high number of possible actions in the system.
- Plausibility and interactivity:
  A virtual world is considered plausible if the interaction in it is logical and coherent. On the one hand, this concerns the user's feeling that his own actions have an influence on the virtual environment, but also that the events in the environment influence the user's senses, i.e. that he can act in the world. This interactivity creates the illusion that what seems to be happening is actually happening.
- Fidelity:
  Fidelity is achieved when the virtual environment is designed to be accurate and lifelike. This happens when the virtual world reproduces characteristics of a natural world, it then appears believable to the user.

2.2.2 Hardware

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

To create a feeling of immersion, special output devices called virtual reality headsets are needed to display virtual worlds. Well known are especially the Oculus Rift, HTC Vive. Also possible are large screens and the CAVE.

[19]

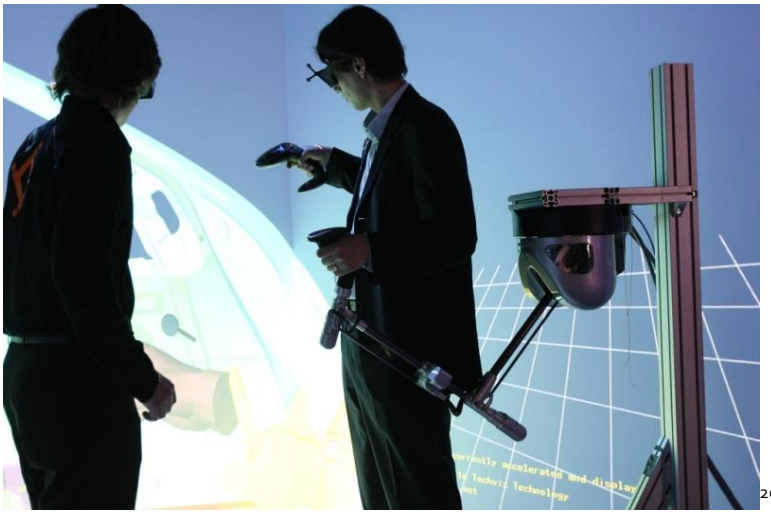
3D powerwall

To convey a spatial impression, two images are generated and displayed from different perspectives (stereo projection). Various technologies exist for delivering the respective image to the correct eye. A distinction is made between active (e.g. shutter glasses) and passive technologies (e.g. polarizing filters or Infitec).

[19] By The original uploading user was TMP-Mediagroup in Wikipedia in German - Transferred from de.wikipedia to Commons using CommonsHelper., CC BY-SA 2.0 en, https://commons.wikimedia.org/w/index.php?curid=11918646

Special input devices are required for interaction with the virtual world. These include a 3D mouse, data glove and flystick, as well as the omnidirectional treadmill, with which walking in virtual space is controlled by real walking movements.

The Flystick is used for navigation with an optical tracking system, whereby infrared cameras permanently report the position in space to the VR system by detecting markers on the Flystick, so that the user can move freely without cabling. Optical tracking systems can also be used to capture tools and complete human models in order to manipulate them in real time within the VR scenario.


[20]

Interaction with Flystick and 6DoF Force-feedback in multi-page projection

Some input devices provide the user with force feedback to the hands or other parts of the body (force feedback), so that humans can orient themselves in the three-dimensional world and perform realistic simulations through haptics and sensory perception as a further sensory sensation.

2.2.3 Software

Software developed specifically for this purpose is needed to create virtual reality. These programs must be able to calculate complex three-dimensional worlds in real time, i.e. with at least 25 images per second, in stereo (separately for left and right eye). This value varies depending on the application - a driving simulation, for example, requires at least 60 frames per second to avoid nausea (simulator sickness).

In the 1990s, the computing power as well as the hardware were usually not yet sufficient for productive use and for realistic simulations, which is why special graphics workstations were usually used here. At the beginning of this millennium, the possibilities of interaction in the scenarios have increased significantly due to much more powerful computers and graphics processors.

For the modeling of three-dimensional, virtual objects, programs such as Maya, 3D Studio Max, Blender, SketchUp, Softimage XSI, Cinema 4D, LightWave 3D and other CAD or 3D programs are used. Additional software is needed for image and sound processing. To assemble the objects modeled there into interactive simulations, one uses authoring systems, such as World Tool Kit or World Up.

2.2.4 Applications in the technical field

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

Virtual reality can be used in many areas. A very well-known field of application is pilot training in flight simulators. This technology is also increasingly used in industry, especially for the creation of

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

virtual prototypes, production planning, virtual training, for ergonomic assessments and spatial studies in geology.


[21]

Flight simulator with a CAVE

Other areas of application are visualizations in architecture, medicine, chemistry, energy and edutainment (e.g. virtual cultural heritage). The therapeutic use of virtual reality is being investigated under the heading of virtual rehabilitation. In industry, both "Powerwall" as a stereoscopic 3D wall and multi-sided projections such as CAVE are used for full immersion in graphic simulation. In recent years, a number of companies offering virtual reality software for industrial companies have been established in Germany and France, such as ICIDO, VISENSO and others.

Virtual reality can simulate natural work systems. Employees experience close-to-reality handling of simulated systems, machines and work equipment in a virtual work environment. The virtual working environment appears in its natural size, technical processes run continuously and in real time. Movements in this environment can be directly controlled by machines and/or people. Perspective, viewing angle and acoustics change depending on where the person is standing and how they move[22] .


[23]

Practice parachute jumps on a simulator

[21] By TheGrenzebachGroup - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=28172269
[22] At the school, for example, we used the possibility of VR with an appropriate headset in the robotics lab.
[23] Public domain, https://commons.wikimedia.org/w/index.php?curid=611156

With VR, all phases of the product life cycle can be simulated, analyzed and optimized: from construction to use to disposal. With VR in occupational safety, you can

- Check and improve the usability of products and processes already during their development and construction. This avoids undesirable developments and the need for subsequent changes.
- Systematically and empirically investigate design solutions for human-system interaction and their influence on human behavior. This reduces modifications to machines and costly field studies.
- safely test potentially hazardous products, processes and protection concepts. This prevents actual hazards during human-system interaction.
- Determine cause-effect relationships after accidents on and with products. This saves material, personnel, time and financial expenditure for on-site investigations.

2.3 Augmented reality

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

Augmented reality (AR) is the computer-based extension of reality perception. This information can address all human sensory modalities. However, augmented reality is often only understood as the visual representation of information, i.e. the addition of computer-generated information or virtual objects to images or videos by means of superimposition. In football broadcasts, for example, augmented reality is the superimposition of distances for free kicks using a circle or a line.

In the reality-virtuality continuum, augmented reality (AR) and augmented virtuality are part of the so-called mixed reality. While the term augmented virtuality is hardly used by experts, augmented reality and mixed reality, rarely also enhanced reality, are mostly used synonymously.

An AR system (ARS for short) is the system of technical components necessary to build an augmented reality application: Camera, tracking devices, support software, etc.

2.3.1 Applications

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

Augmented reality could be used in virtually all areas of everyday life. Assembly workers could have the next work step displayed directly in their field of vision; soldiers or disaster relief workers could have targets and danger zones in the field displayed and designers could work on the same three-dimensional model with colleagues who are actually and virtually present.

[24]

As technology advances, futuristic application scenarios can be developed: electronic devices that exist only virtually but respond to real touch, artificial sensory enhancements such as "X-ray vision" and computer games in open terrain.

An example of an AR application are the virtual markers superimposed in real time during sports broadcasts: Different distances of competitors in ski jumping, long throw, etc. (Note that this example is often not an augmented reality application according to the definition above, as sometimes the interactive element is missing).

[24] From 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

Possible areas of application are:

- Assistance with complex tasks, especially in construction, maintenance and medicine: Additional information can be displayed to assist with complex tasks. For example, the parts of a device are "labeled" for a mechanic, and he receives work instructions.
  In medicine, augmented reality can be used to enable the display of elements that are not visible. For example, this can be done intraoperatively, as an "X-ray view" for the surgeon, based on previous tomography or actual image data from ultrasound machines or open MRI scanners.
- Industrial applications: Augmented reality can be
  used to efficiently match digital design data with existing real geometries. The technology also enables the broad use of digital validation methods when combining digital data with real prototypes or designs.
- Navigation:
  Augmented reality can basically be used for navigation in buildings (for maintenance of industrial plants), outdoors (for military or disaster management), in cars (projection of navigation instructions on the windshield so that, for example, turn-by-turn directions appear on the roadway) or in airplanes (head-up displays in fighter planes are one of the earliest AR applications ever).
- Digital cameras:
  Digital cameras with live view viewfinders and screens can display additional calculated information about the subject, such as faces detected, edges in focus, or incorrectly exposed areas of the image.
  Grid lines or electronic spirit levels can be displayed to help align the edges of the image or subject.
- Art:
  AR in the visual arts enables objects or places to trigger artistic multidimensional experiences and interpretations of reality. The unfinished monument of the artist Benno Elkan was virtually reconstructed and exhibited in the Museum of Art and Cultural History Dortmund. Using a smartphone app, the virtual monument can be viewed from all sides.
- Military and disaster management:
  In the military and disaster management sector, portable systems can be used to indicate friend or foe or the source of fires.
- Hydrology, ecology, geology:
  systems can be used for prospecting, for the display and interactive analysis of maps and terrain features, for example to exploit mineral resources.
- Architecture:
  Augmented reality is also suitable for the visualization of architecture. Destroyed historical buildings or future architectural projects can be depicted.
- Simulation:
  Augmented reality can also be used for flight and driving simulators.
- Collaboration of distributed teams: The collaboration of
  locally distributed teams can be facilitated. For example, through video conferences with real and virtual participants. But also the joint work on simulated 3D models is supported in this way.
- Advertising:
  Increasingly, companies are using AR components in their advertising to offer customers added value. In 2013, for example, the furniture store chain IKEA published a catalog in

which selected pieces of furniture could be scanned via smartphone app and virtually projected onto any place in the home.

- Learning:
  Augmented reality also has great potential in the field of learning. It is possible, especially through mobile applications, to project digital layers onto real worlds and to integrate them seamlessly into reality. Especially applications with animations allow an interactive exploration of running processes. Thus, especially abstract concepts, which in traditional forms of learning can sometimes only be viewed from one side, become more tangible.
  An example of AR learning through independent exploration is the app "Timetraveler Berlin Wall", which integrates historical events into the environment of today's world.
  Critics fear that if AR apps are used without reflection, linear modes of perception will be practiced that prevent a 'natural', dynamic perception of and interaction with the world.
- Television and sports:
  Some television and sports broadcasts rely on visual information graphics projected onto the screen in the form of an augmentation of reality to convey information about the game, for example.
  AR can also serve as an extension for practicing a sport; in augmented climbing, for example, the climbing wall
  becomes a game through light projections.

2.3.2 Challenges

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

One problem is the technical burden of augmented reality, especially the tracking of images as they move. Sensors are also affected by motion. Thus there is noise, drift and shadowing of the tracking system (for example with GPS, INS). A combination of, for example, GPS with inertial and optical navigation is therefore common in advanced systems.

Another problem is the energy supply. The currently available batteries are not yet sufficient to power mobile augmented reality systems for longer periods (more than a few hours). The availability of data, authoring and high complexity of data can also lead to problems.

In order to embed the virtual scene in the real scene as convincingly as possible, data is required that also describes the geometry of the environment. Based on this, virtual sections through real objects can then be drawn and the occlusion of the virtual objects by the real objects can be calculated. However, this geometry data is not always available or up-to-date.

The complete integration of virtual objects into real scenes requires the hiding of background parts so that the objects do not appear transparent. Systems that completely replace the direct view with camera images (EyeTap) do not have this problem, but are unsuitable for many applications.

2.3.3 Future

As future applications, some examples are given here. On the one hand, an extension of PC operating system interfaces into the real environment can happen. Program windows and icons are then displayed as virtual devices in real space and operated by looking or pointing with the finger. This can generally lead to the replacement of conventional screens (replacing mobile phone and navigator screens and superimposing information directly into the environment, for example from guidelines directly onto the roadway, as well as extensions, such as "X-ray vision" to display hidden targets), device control panels, or to completely new types of devices.

Furthermore, augmented reality can be used for multimedia applications, such as pseudo-holographic virtual screens, virtual "holodecks", virtual surround cinema. But applications could also be envisaged for the beautification of everyday environments, such as through the display of virtual plants, wallpaper, vistas, artwork, decorations, lighting, etc. If AR systems become more widespread, virtual shop windows, billboards or traffic signs could also be used.

It would also be possible to merge virtual and augmented reality so that the user can switch between the forms on a terminal device.

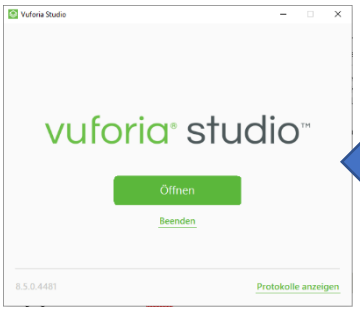
2.4 Vuforia Studio

2.4.1 Overview

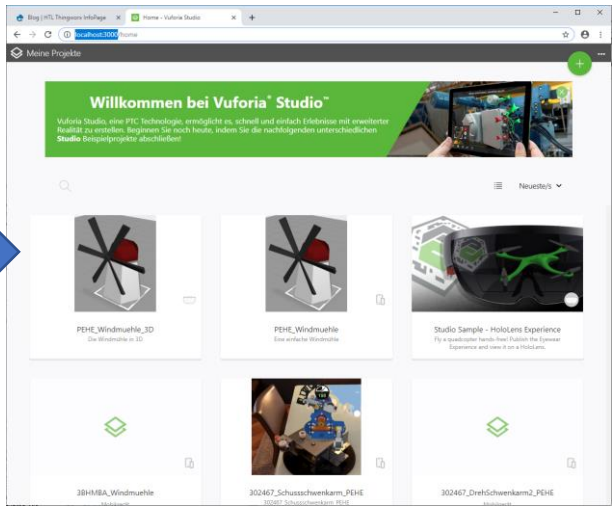
Video Link 8: https://youtu.be/52dkOzo8JUY

PTC's Vuforia Studio software package is specifically designed for AR content creation. We will be using Vuforia in class to create our own augmented reality. To do this, we need to install Vuforia on our PC.

The application downloads the necessary things from the Internet and installs Vuforia Studio on the PC. After successful installation (there is not much you can set here, it installs itself as it wants). Vuforia Studio can also be started already.



Vuforia application: creates web server with Vuforia functionality.



Web browser: displays the web page generated by Vuforia - this is where you can operate Vuforia. The address is: localhost:3000

At first the result is sobering. After a few seconds a window appears and that's it. Then there is a button "Open" which starts the web browser. Why is that?

Well, Vuforia Studio starts a web server on our computer. The web browser replaces the need to program your own display application. And the web browser is used to display the web page provided by this server. The page is accessible via port 3000 and because it is a page on our local computer, the correct address in the web browser is

localhost:3000

Pressing the "Open" button (which is nothing more than a hyperlink[25]) will launch the default web browser and display the above page. It should be noted that PTC officially only supports Google Chrome. For other web browsers, you will then see the message that it is an unsupported browser.

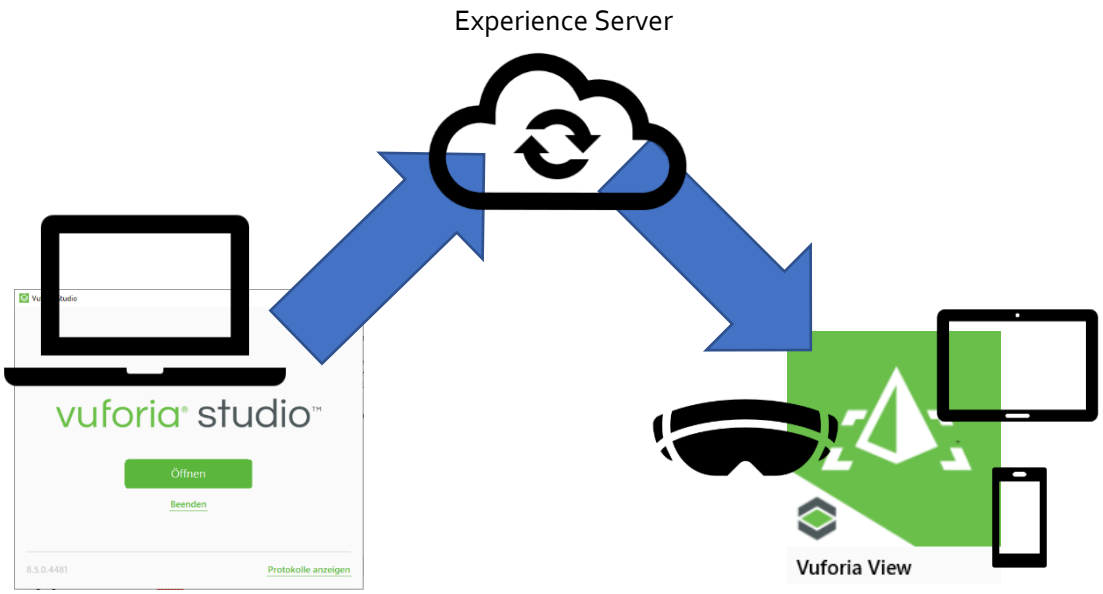
Experience has shown that there may indeed be functional limitations when using another browser. If you don't want to use Google Chrome as your default browser, you can still install Chrome and then you have to enter the address there and can't just comfortably click on Open - if you create a bookmark, it's practically no extra effort.

[25] A hyperlink is nothing more than a "normal" Internet link.

2.4.2 Functional principle

A project is created in the Vuforia Studio. In principle, you must somehow create a reference point in reality. This means: You specify a reference point in Vuforia Studio. Then you build your virtual objects around this reference point. This determines how and in which orientation these virtual objects are located. When you later define the reference point in reality, you have also defined where the virtual things will be inserted. This reference point is therefore the link between the projection and the reality. In Vuforia parlance, such a reference point is called a "target". The virtual objects must be available as a 3D model. Vuforia only supports the PTC format `pvz`. Fortunately, other formats (e.g. SolidWorks) can be imported.

So we define a target and build our virtual objects around it. Then somehow this information has to get to our HMI, which actually shows us the virtual objects in reality. This is done with the so-called "publication". So we take the "Experience" we've created (that's what Vuforia calls a project) and publish the data to an Experience server. With our device, with which we want to view the Experience, we connect to the server, download the Experience and can enjoy it. To do this, we need an application on the device whose name is "Vuforia View". This application can understand the data on the experience server and does the display.



The Vuforia View application is available for Windows 10, iOS, Android and Microsoft HoloLens. Corresponding devices are supported depending on the equipment. An example using a simple model of a windmill on an Android smartphone can be seen in the image on the page. The virtual windmill was placed on the real table.

We therefore need an experience server on which we can publish our projects. This must be provided by the school. The address and the login data have to be obtained from the teacher. As an example, the following access data should be used[26] :

https://example.twx.htl.schule

Username: **AdminUser**

Password: **TWXPassword**

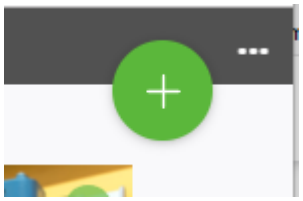
Since we all publish to an experience sever, we have to follow naming conventions. Everyone names their experience according to the following scheme:

CLASS_SURNAME_GIVENAME

So for example: **3AHMBA_PETERSCHOFSKY_HEINZ**

2.4.3 First project for a handheld device (mobile phone)

Video Link 9: https://youtu.be/ooTlFPyPeqo



The best way to learn Vuforia Studio is with a project of our own. First, we start simple by simply having an object of ours projected into the real world. To do this, we need to create a new Experience. This is done with the + icon in the upper right margin. If this is selected, we must first decide on a project type, where we have the following options:

2D Eyewear that is simply a two-dimensional add-on information

3D Eyeware 3D eyewear later used Microsoft HoloLens

Mobile eyewear handheld devices such as tablets and Show smartphones.

We want to create a project for our smartphone first. Therefore we use the option "Mobile". We will be prompted for a project name. We will follow the naming convention above and assign an appropriate name. In the example I use **3AHMBA_PETERSCHOFSKY_HEINZ**.

[26] Please request the real login data from your teacher.

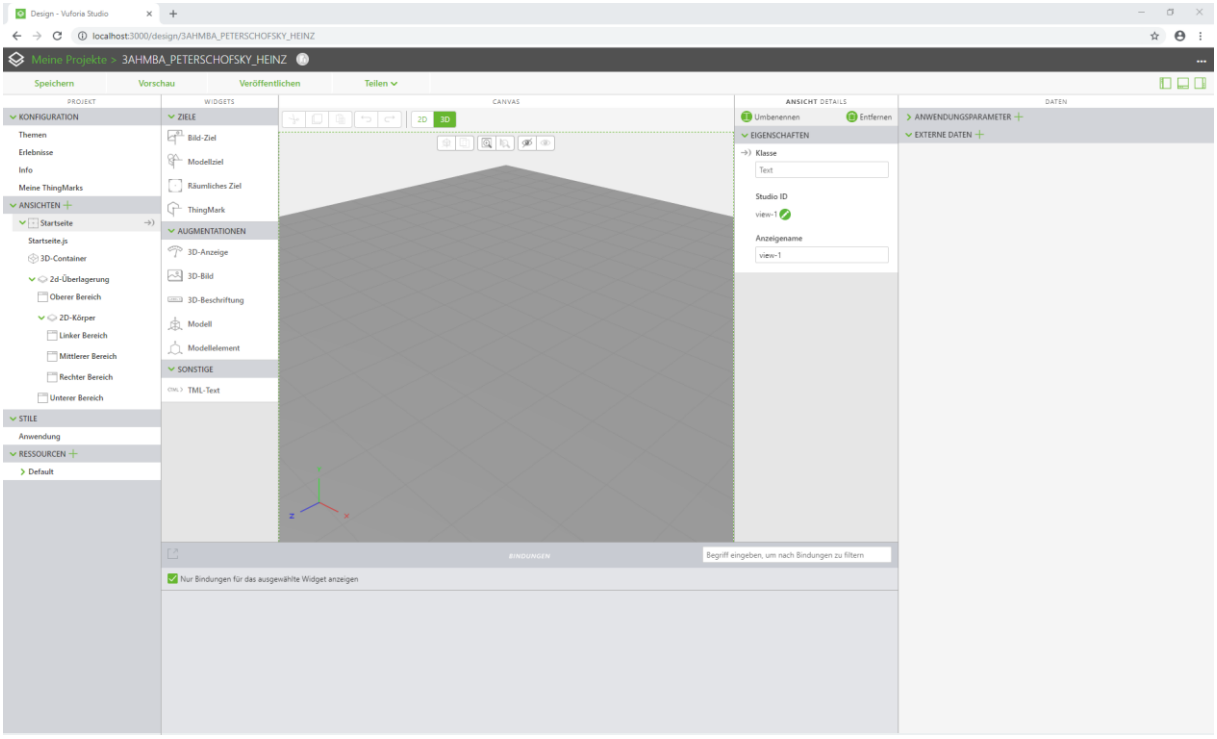
We are also prompted to enter an Experience Service URL - this is the link to our Experience Server on port 8443, so with the above link we write in there:

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

After that the button "Create" can be pressed and the project will be created. All projects are stored in one place, namely in the folder "My Documents" of the current user. If the default directory of the My Documents is somewhere else, this will be ignored. This should be programmed somewhere. It is therefore always the directory:

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

After creating, you will see the user interface of the Vuforia Studio. The "canvas" appears in the middle. This is where the virtual objects are displayed and manipulated. On the left, there is a section called "Project". This is practically the main menu. Right next to it is the "Widgets" section. This can be used to insert new objects. In the right section there is detailed information.



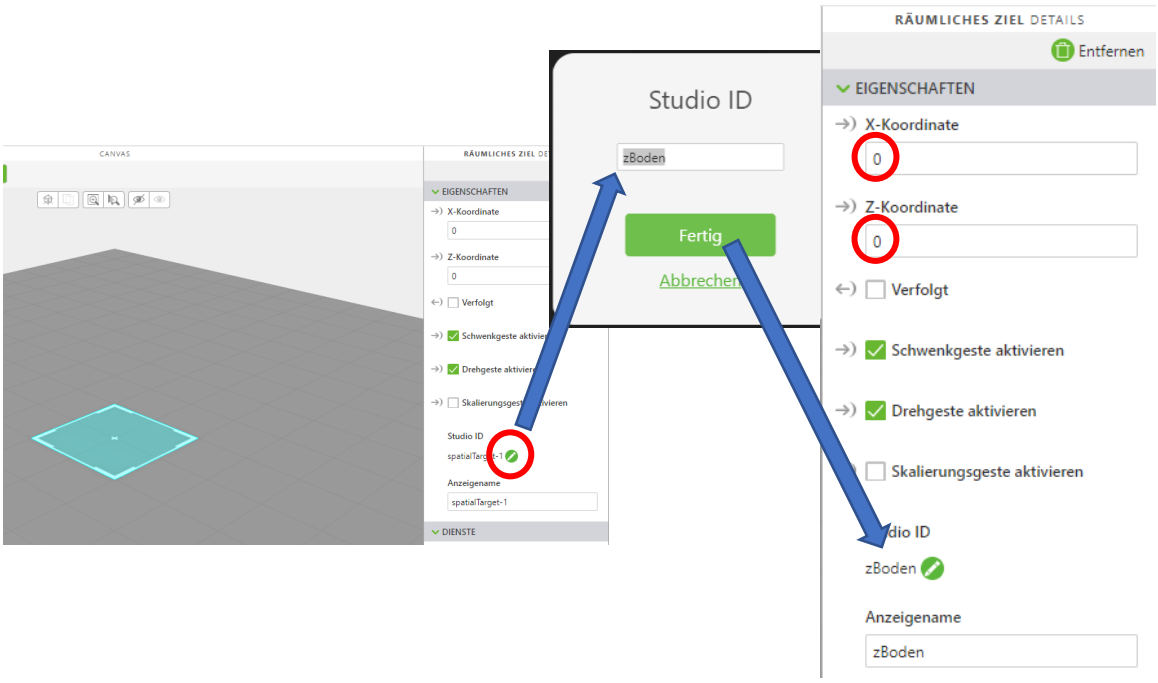
The first thing that needs to be added is a so-called "target". This is the reference point that is the "origin" of the virtual overlays. In the "Widgets" area there is a section called "Targets". Thereby there are the following target types:

- **Image target:** Attempts to recognize an image. If the image specified there is recognized in reality, the virtual parts are added accordingly. Images with strong contrasts - like technical drawings - work best. So you could film a drawing and the 3D model of the part (or even the installation location of the part) is created over the drawing.
- **Model target:** Attempts to find a three-dimensional structure. If this structure is recognized, the virtual parts are projected to it. For example, you could film the base of a part and if the base is recognized as a model target, the part is virtually projected to it.
- **Spatial target:** Is practically a flat surface. In reality, you select a surface and say: This is the one that corresponds to the projected spatial target. Then the virtual objects are displayed accordingly.
- **ThingMark:** Is a 2D code. If this is recognized (e.g. on a printout), the virtual things are projected accordingly. Each ThingMark has a number. The ThingMark that is projected must be recognized. In addition, the scaling is adjusted. In the project one says e.g. the ThingMark is $10cm$ wide. But if it is printed in reality with a width of $11cm$ the model will also be scaled to 110%[27] .

We will use spatial target once for our purpose. We drag the Spatial Target from the Widgets area to the Canvas area and drop it there.

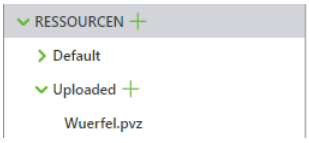
With this, the target now "lies" once on our virtual ground. It should also already be selected. Then the properties of the target are displayed in the details area. We will change the name of the target. This is done with the green pencil icon in the details area (now it probably says "`spatialTarget-1`"). We will now name this row "`zFloor`". This is also the so-called studio ID. This will be important later when we want to program some stuff.

[27] When using the Microsoft HoloLens, the model is also shifted. Here, special attention must be paid to the correct brand size.

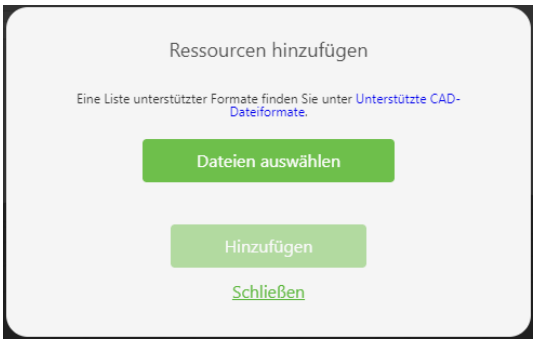


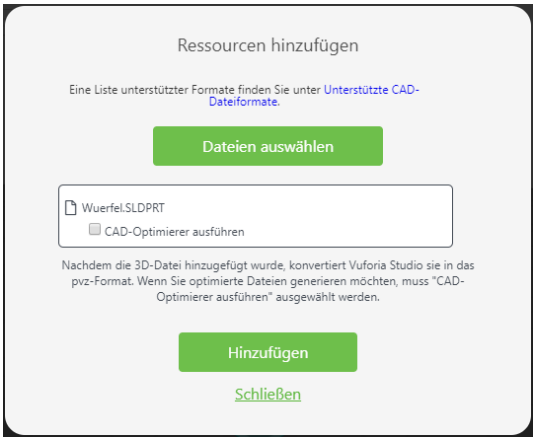
Also, we want to make sure that the target is at the origin. The units of measurement here are always meters. If we enter somewhere 1,1 somewhere, then this corresponds to a distance of $110cm$.

After we have defined a target, we want to add a model. For this we need a 3D object. This object can also be an assembly, for example. But I would start small and use a small, simple object. In my case this is a small red game cube, which I have modeled in SolidWorks. Now I have to get it into the Vuforia Studio somehow.

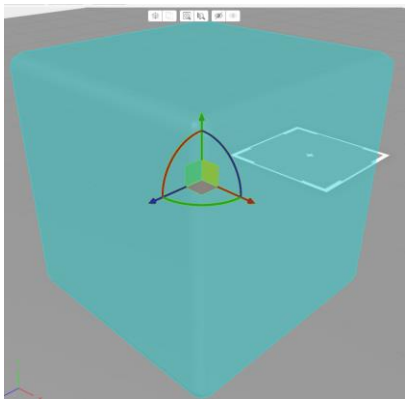


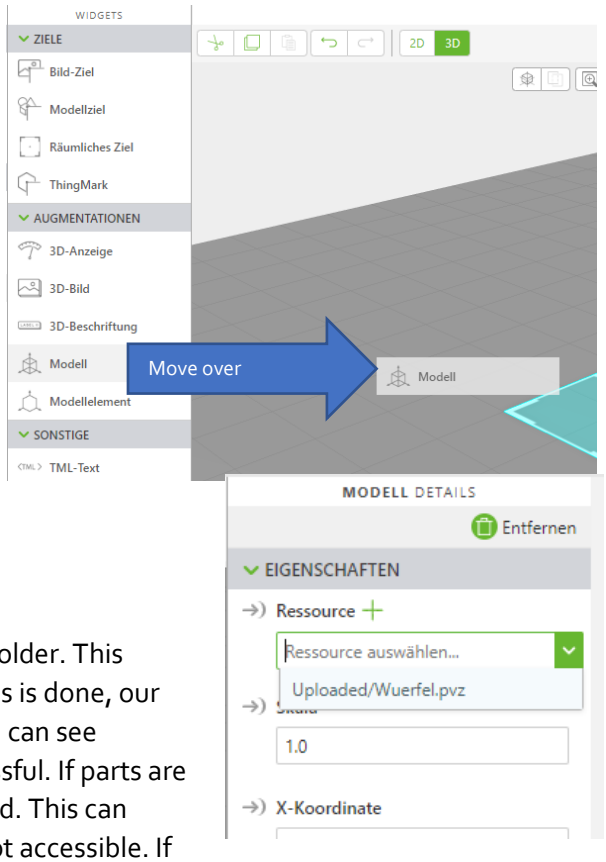
Things that come "from outside" are called "resources" and can be found in the lower left corner. There is nothing there yet, so we have to press the "+" symbol there. The "Add resources" dialog opens. With "Select Files" we can add the appropriate file with the 3D model. If we have added the files now, there is still the possibility to run a "CAD Optimizer". This creates three models instead of one and reduces the polygon count of two of them. For very large models this can be useful, for our small objects it doesn't matter. So we will not run the CAD optimizer. Clicking on "Add" converts the model to a pvz file. This works quite well, only if parts are not found, then they will not be converted. Especially with standard parts this is often a problem. But you can recognize such a difficulty relatively fast. After successful conversion you will find the new resource displayed in the lower left corner.





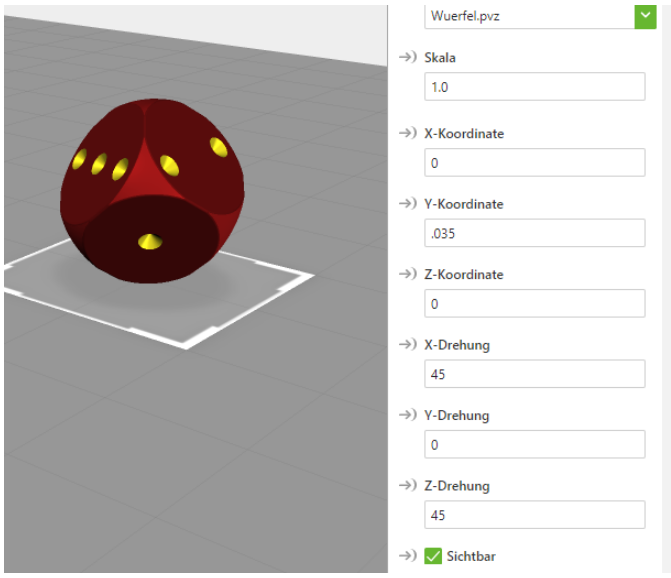
Now we just need to display this cube. To do this, we add a "model" to our canvas by simply dragging it back in. A model placeholder will then appear there (a large cube with rounded corners).





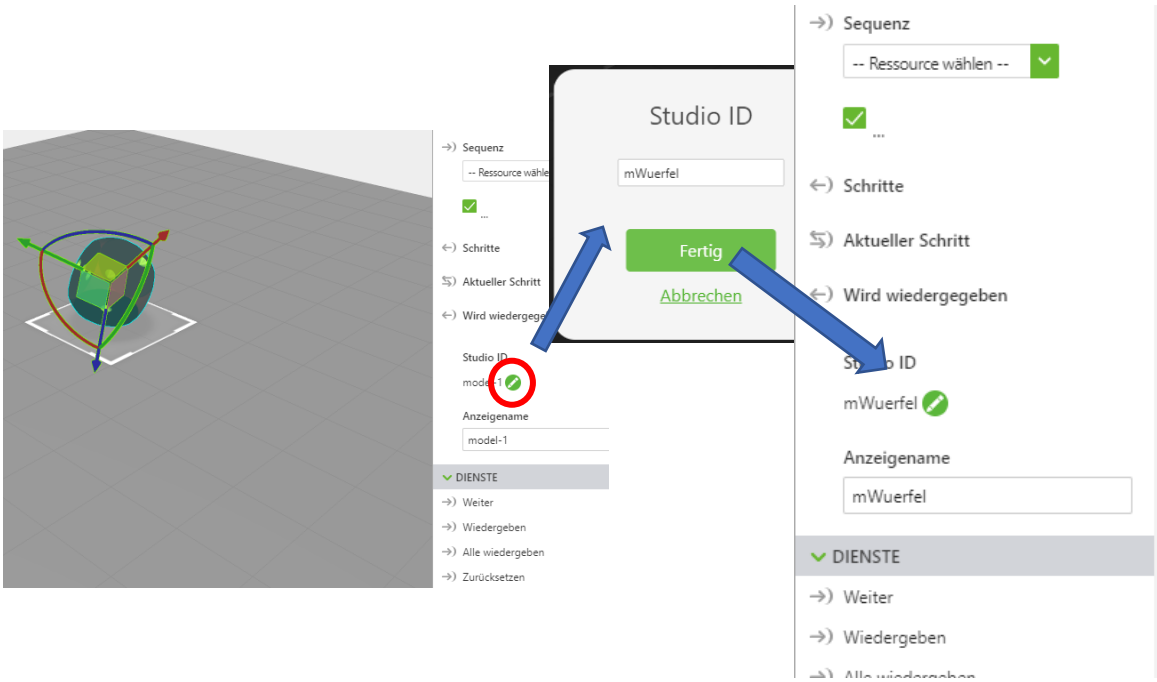
We now need to assign our resource to this placeholder. This happens in the properties area at the top. Once this is done, our model appears and no longer the placeholder. You can see immediately if the import of the model was successful. If parts are missing, the corresponding data could not be found. This can happen with assemblies where the part files are not accessible. If this happens you have to try a little bit. Usually you will find a way. Now is also the time to play with the values for X-, Y- and Z-coordinate as well as the rotation. The result can be seen immediately on the screen.

I've set my cube to the coordinates $(x; y; z) = (0; 0{,}035; 0)$ with the rotation $(rx; ry; rz) = (45°; 0°; 45°)$ the rotation. This makes my cube look like it would be balanced on a tip.

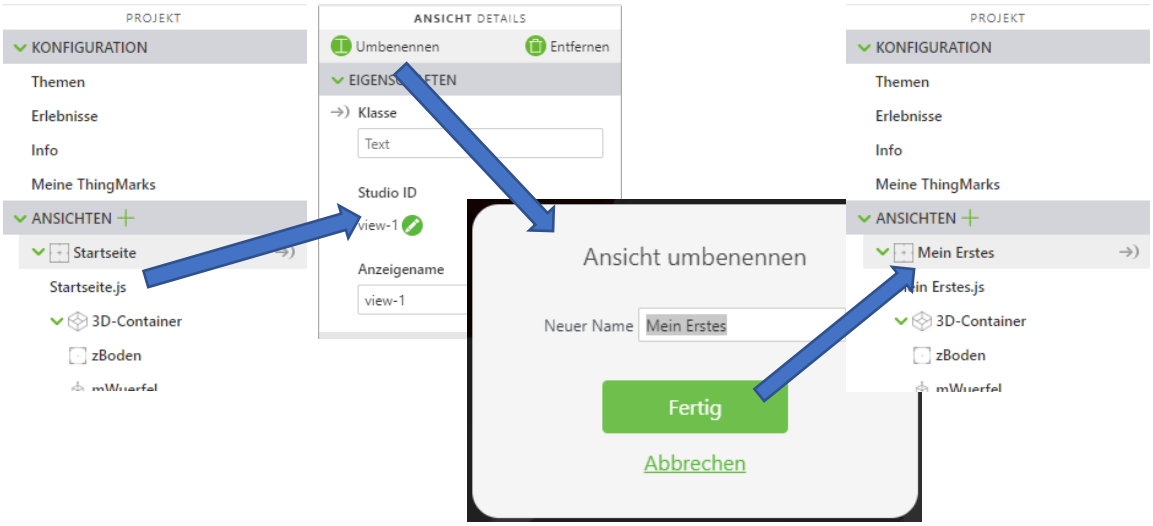


Although we will only use one model in this exercise, we will name it - in the future we may have to deal with more than one model, and then we will have no idea whether "`model-12`" means the lid or the bearing. That's why we're going to do this cleanly. Our model is now normally called "`model-1`". How do we rename it? In the right area - under the coordinates there is again the "Studio-ID". We

change this by clicking on the green pencil icon. Just like when we name the target. I use "`mWuerfel`"[28] (note no umlauts or other special characters!)



Now let's give our "view" an appropriate name. Views are effectively "sub-projects". Different views can be used in each project. We will use the views to handle our various exercises in one project. At the moment, the only view is probably called "Home". To rename it, we click on the view in the left pane. The Details pane will then show the details of the view. There we can click on "Rename".

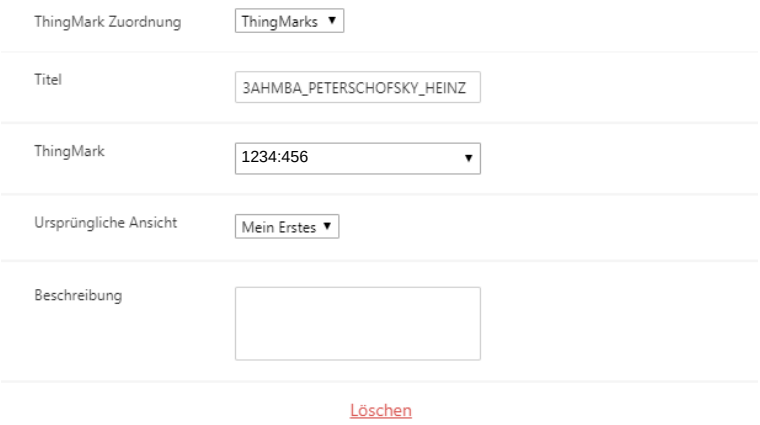


A dialog box opens and we choose a better name. I simply choose "`My First`" here. With the click on "Done" the view is renamed.

That's enough for us for this first exercise. We now want to make this project available as an experience. We have already registered the experience server. Now it's time to add some more data and then publish it. What do we have to add? In the project tree under experiences we now define a

[28] Würfel is German for cube or in this case: dice.

ThingMark under which we want to find our experience. We also give a short description. Each ThingMark has a number, and the number can be used in the corresponding field.

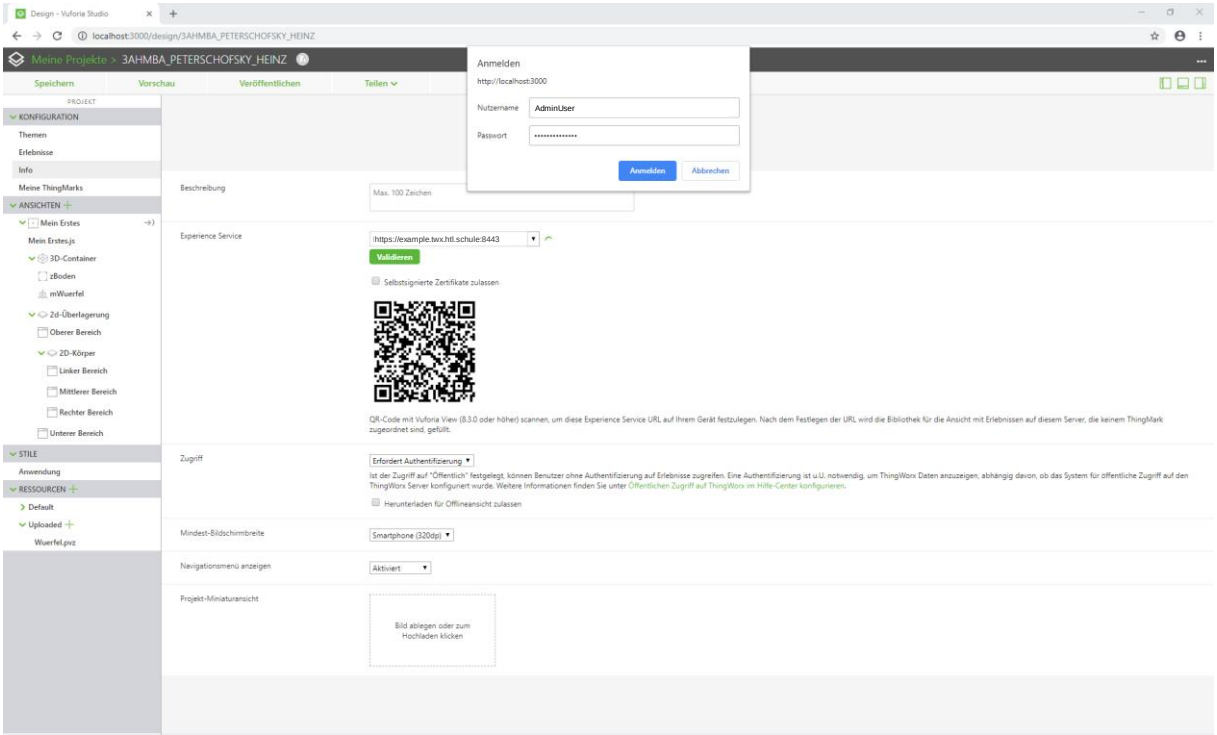


Now we can make our experience (project) available. To do this, however, the corresponding experience server must be accessible. Therefore, we still have to test the experience server. To do this, we switch to the "Info" area. There we first try to reach the experience server by clicking on "Validate". This will take us to a login screen where we have to enter the username and password. As a reminder: With the above example data this would be:

Username: **AdminUser**

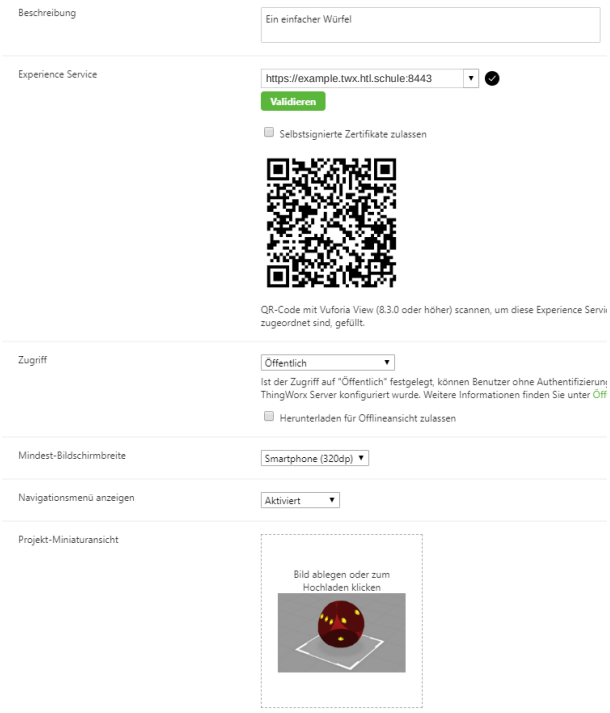
Password: **TWXPassword**

Entered this information and pressed Login.



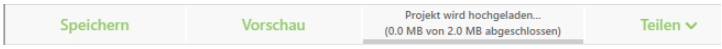
After successful validation, a check mark appears next to the Experience URL - the connection and login data are correct. After that we should add a meaningful description and an image in the project thumbnail. I decided to take a screenshot using the snipping tool.

The access can be set to "Public". This means that everyone who has the ThingMark entered earlier can also load the experience from the server. With "Requires Authentication" you would have to enter username and password. This is overkill at the moment.



With this we are ready to publish. You can still click on "Preview" at the top bar beforehand. This will open a new tab that simulates the project. For more complex projects, this feature makes more sense. Now it looks kind of bland and hardly different from the projecting environment.

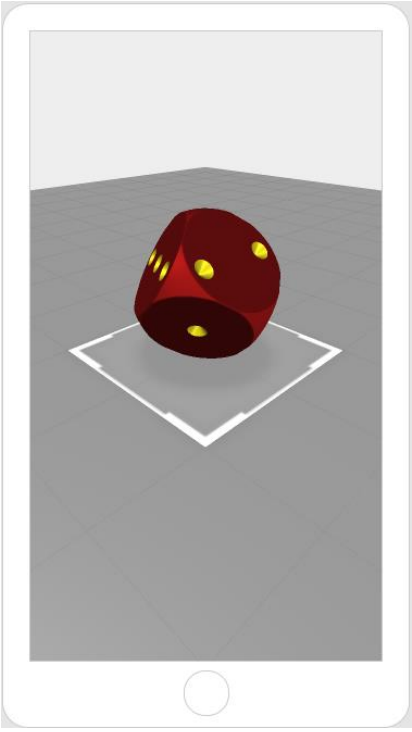
So let's try it: Let's click on "Publish"! A bar with a progress indicator appears briefly and then it's all over again.

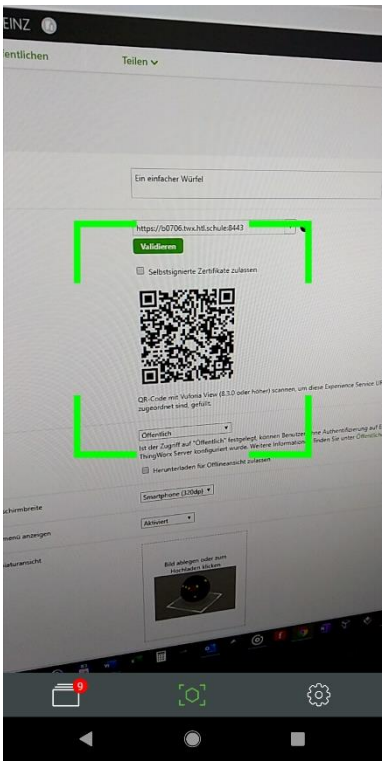


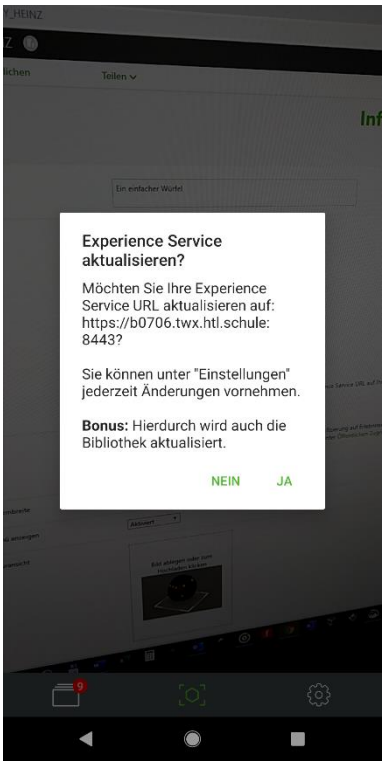
Obviously, something's happened. What happens now?

On our computer, we are done. Everything is done. Let's move on to a smartphone or tablet. There we install Vuforia View (search in the app store).

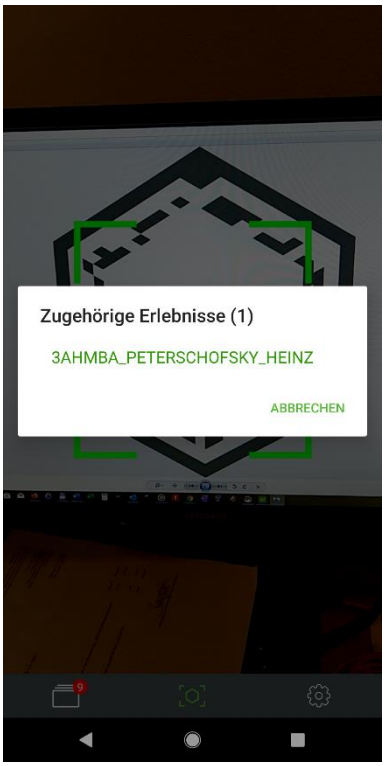
After installing Vuforia View, we can open the app. The app will announce itself with a green frame and the live image of the camera. The first thing we have to do is somehow tell the app how to reach the Experience server. Fortunately, this is done relatively easily. In the info section of the Vuforia Studio, there is a QR code. You can scan this with the app. That's why there's this green frame. If you bring the QR code into the frame, you will be asked if you really want to set the experience server. In this case, you can say yes.

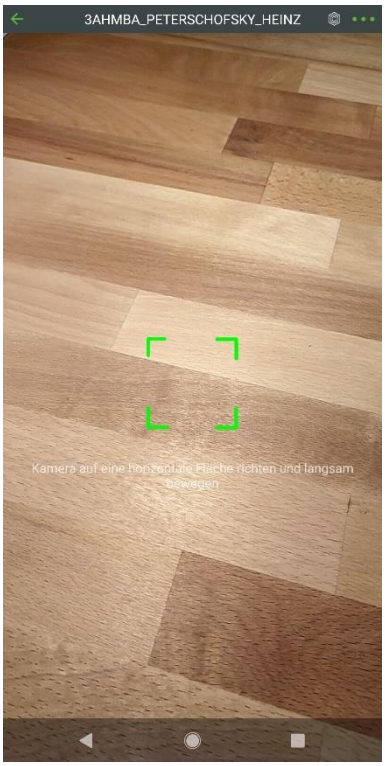


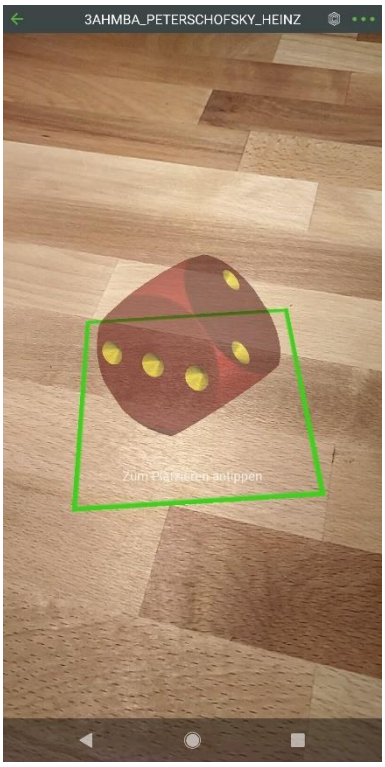

Teilen
Ein einfacher Würfel
https://b0706.twx.htl.schule:8443
Validieren
Selbstsignierte Zertifikate zulassen


Experience Service aktualisieren?
Möchten Sie Ihre Experience Service URL aktualisieren auf: https://b0706.twx.htl.schule:8443?
Sie können unter "Einstellungen" jederzeit Änderungen vornehmen.
Bonus: Hierdurch wird auch die Bibliothek aktualisiert.
NEIN
JA

The experience server is now set. Now we only have to specify which experience should be loaded. We have chosen ThingMark as identification method and used the ThingMark matching our class. We can display this on the screen or print it. If we now scan this ThingMark we get a list of all Experiences stored under this ThingMark. We select the appropriate one and look for a flat area. Now we can practically choose where our area target is. With one click we set the position and then our object is already embedded in the real environment. Even if we move our smartphone, the thing stays where it is. Depending on the phone model, it works better or worse, but you definitely get an idea of the object.







2.4.3.1 *Mandatory task "Augmented Model*

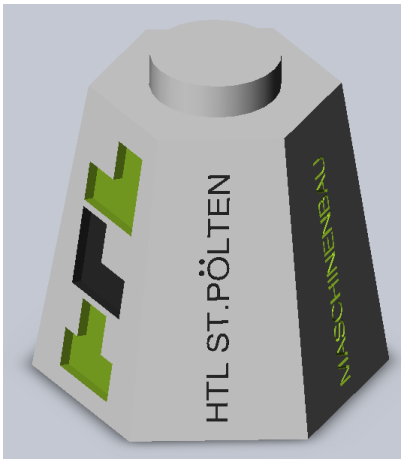
	Create your augmented reality with a model of your choice as described above. Publish it and demonstrate your model.	

2.4.4 Multi-part model for 2D device

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

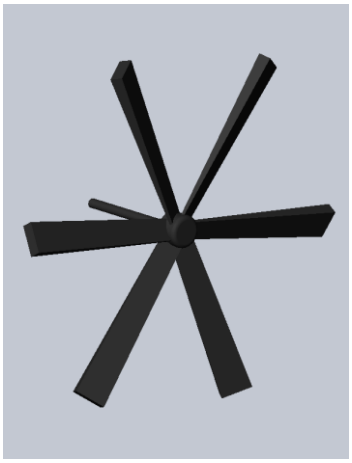
We no longer want to settle for a one-piece model. In fact, if we want to animate things, we need to display multiple models, or rotate or move them against each other. This creates the effect of animation. As an example of such a multi-part model, I have chosen a windmill. You see, I want to make two movements: I want the windmill's propeller to be able to turn, and I also want to be able to turn the roof "into the wind"[29] .

I started by drawing the things I needed in SolidWorks. With that I had three parts:



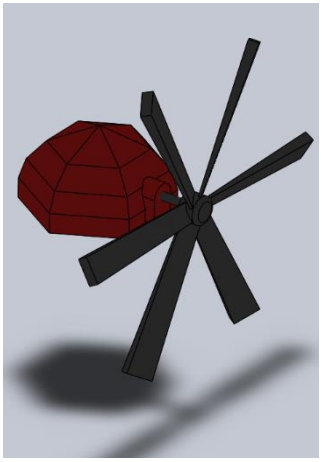
Building part

Roof section



Wind turbine

It should be noted that the origins of the individual parts fit well together. For example, if we want the roof part to rotate about exactly the right axis, the origin must also lie on that axis. In my case, that would be the center of the hexagonal base. With rotation around two axes this is practically not possible. That's why I made an assembly out of the roof part and the windmill and put these two things together already.

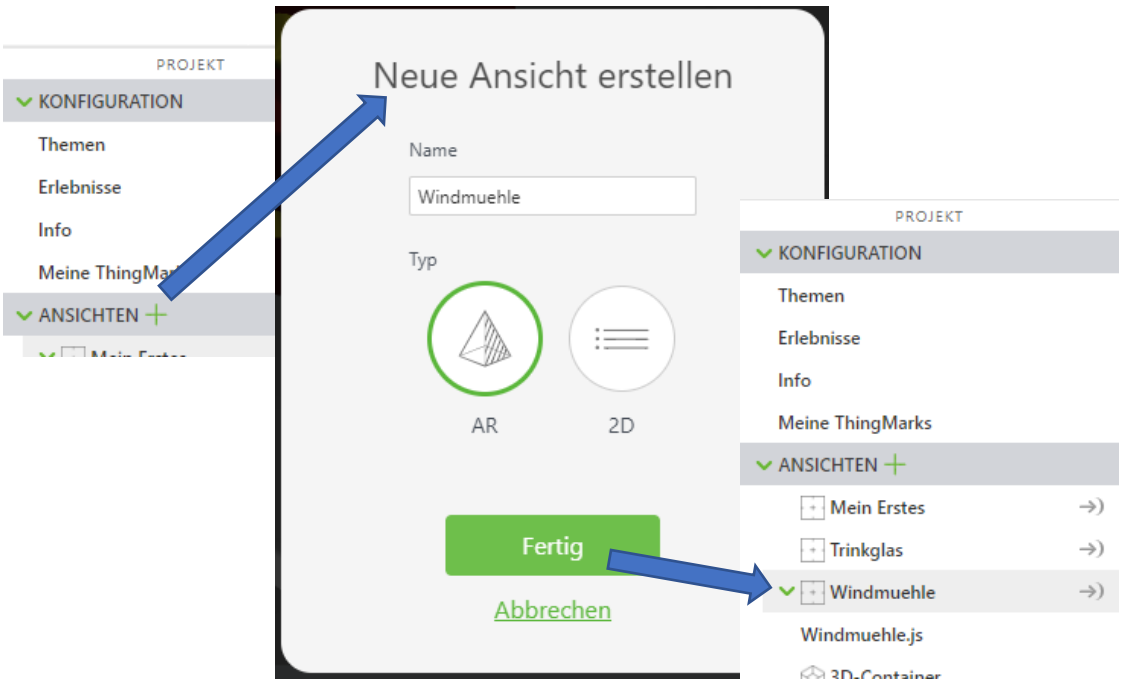


So I could dare to extend my project. Yes: extend, because we don't make a new project, but we extend our previous one with a so called "view". These views are something like "subprojects".

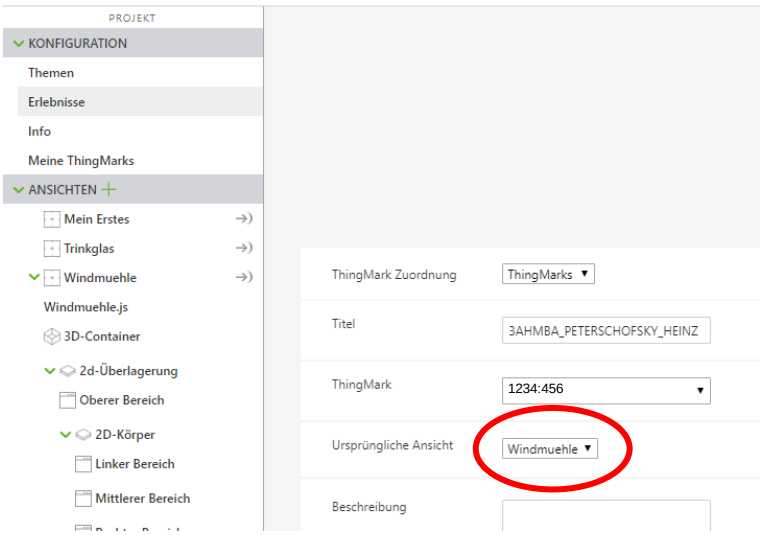
To create a new view, click on the "+" symbol next to the word "Views" under "Project".

[29] Other examples would simply be a windmill, or even a crane that you can swing and move a trolley forward and backward. I am open to suggestions.

This opens a dialog box. There we select "AR" and give the new view a name. I call it "`Windmuehle`" (note: no umlauts!). With a click on the button "Done" the view is created and can be used[30].



So we have a new view now. When we publish the project, we will get the possibility to switch between the views - but we will see that later. It can be annoying to have to switch to the right view over and over again. That's why Vuforia offers us the possibility to choose a default view. To do this, we switch to the experiences settings page and select our new view "Windmuehle" under "Original View".

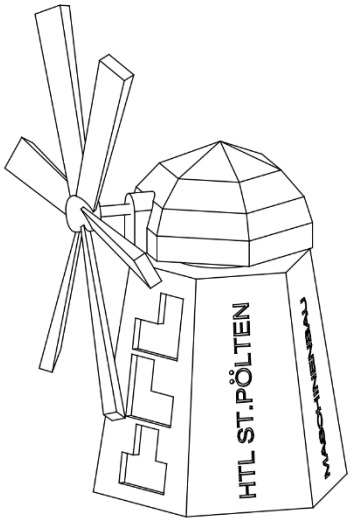


This means that when we open our Experience, we automatically get this view - we have to switch to all other views of the Experience.

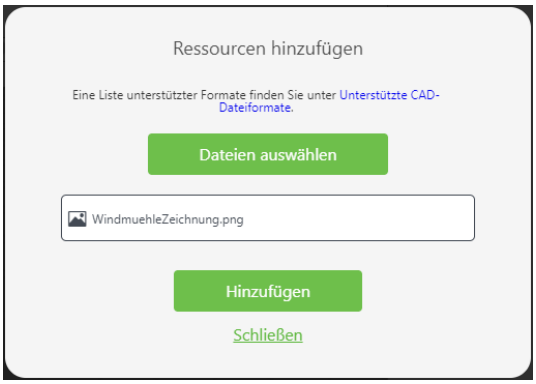
Alright, so let's get started: we're going to use an image target this time. I have created a drawing of the planned finished assembly in SolidWorks and exported it as an image. I now want to use this as

[30] Please note: I created a view with the name Trinkglas before. That's why it shows up here - you're missing that line. That is fine.

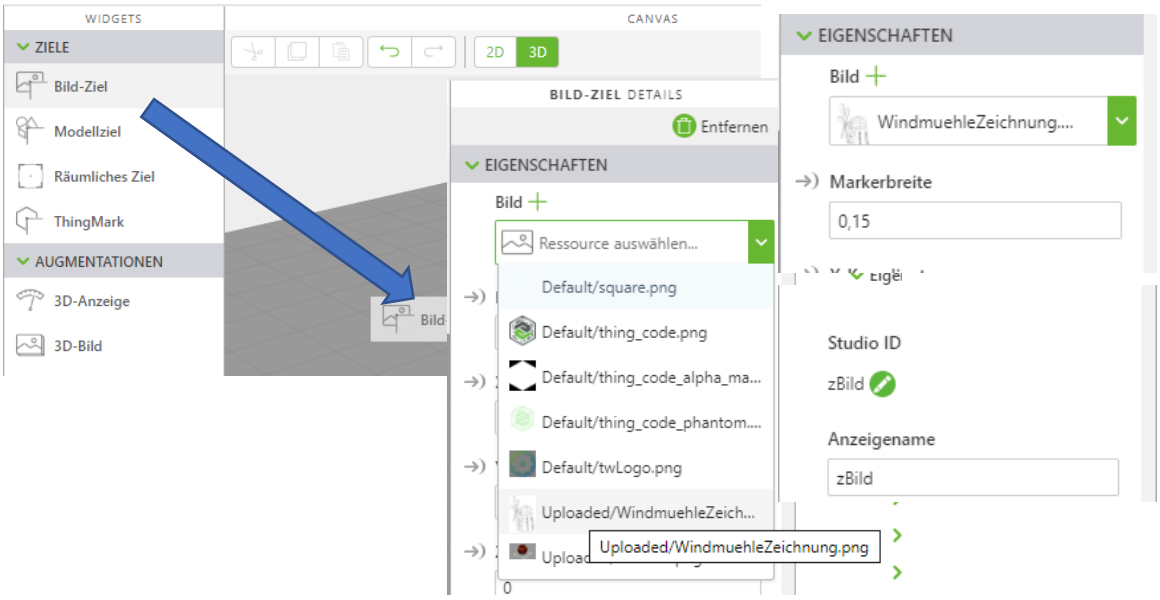
the image target. This means that if the drawing of the windmill is lying around somewhere and we aim at it with our device, the windmill should appear on top of it.



First, we need to add the drawing as a resource. Now, this works again the same way as with models: The "+" symbol next to the resources lettering opens the dialog box. We select the corresponding image file there.

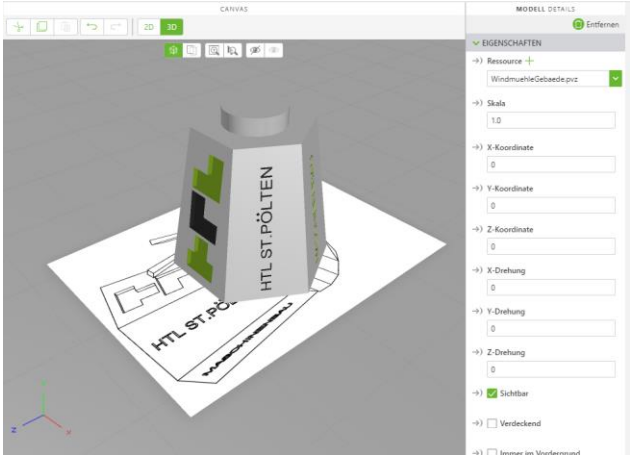


Now we see the drawing in the resources under "Uploaded". Now we add an image target: Drag the image target into the Canvas and release it. Then, under Image Target Details, select our new drawing as a resource.



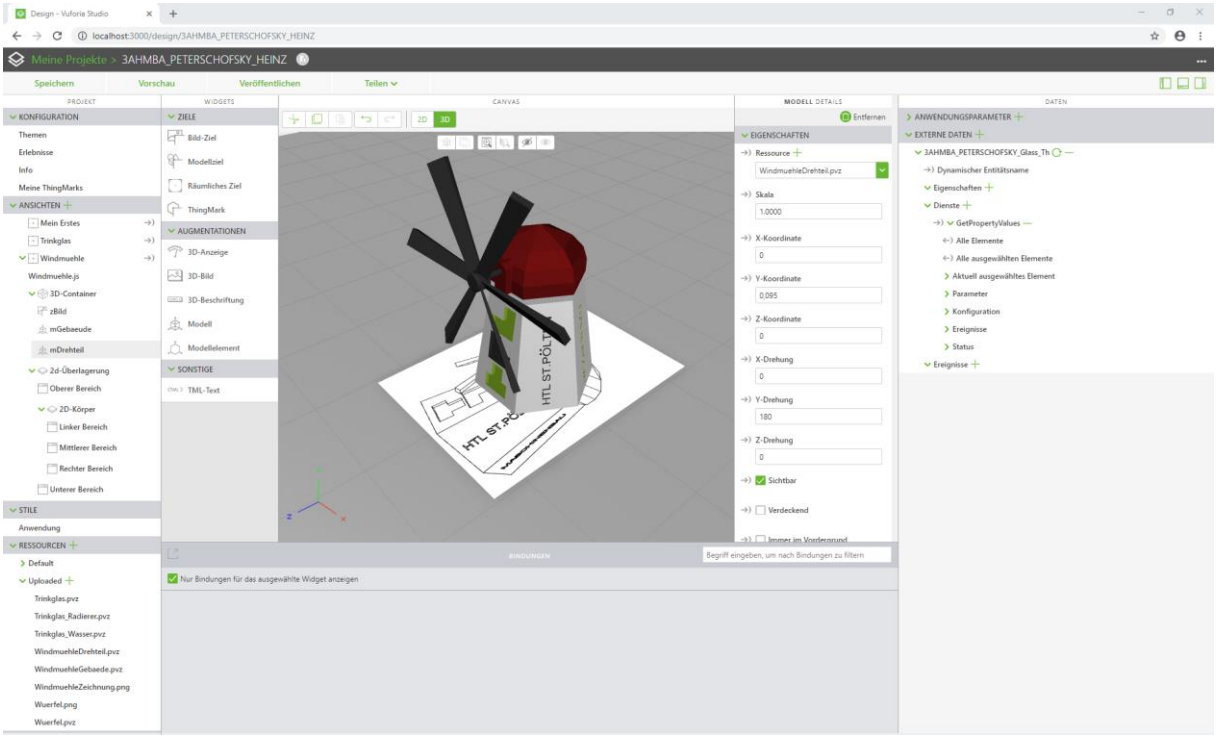
Now we have to enter the width of our drawing. The windmill is about 15cm wide, so I enter "0.15"(remember: we have meters as unit here). I also name my target accordingly and set the Studio ID to "`zPicture`". With this we have defined our target - it will also be displayed in the canvas accordingly, even with the correct image and in scale.

Now we can start adding our models. I start again by adding the resources. I choose as one resource the building part of the windmill and as another resource the assembly of roof and impeller. After that I add a model and select the building part as resource there.

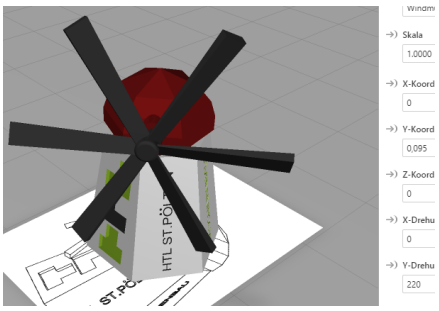


With the coordinates $(x; y; z) = (0; 0; 0)$ the thing is exactly on the drawing and we give the model an appropriate name - I choose "`mGebaeude`"[31].

Now we add the next part: Again a model and this time the assembly with the roof and the impeller. The coordinates are $(x; y; z) = (0; 0.095; 0)$ and the rotations are $(rx; ry; rz) = (0; 180; 0)$. Also, I still call the model "`mDrehteil` "[32] and my window looks like this:



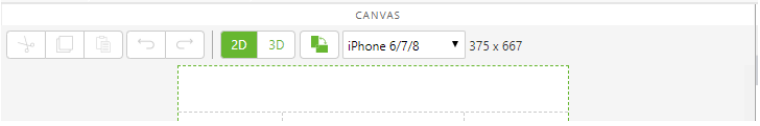
Well, this already looks very promising. In principle, we now have a two-part model. We can also try out what happens when we change the Y-rotation. If we change the value there to e.g. 220 degrees, we can see how the roof with the impeller rotates around the y-axis. This works so well because the origin of the assembly was chosen correctly. The x-, y- and z-axes are those of the assembly. If its origin lies "somewhere", our part also rotates around this "somewhere".



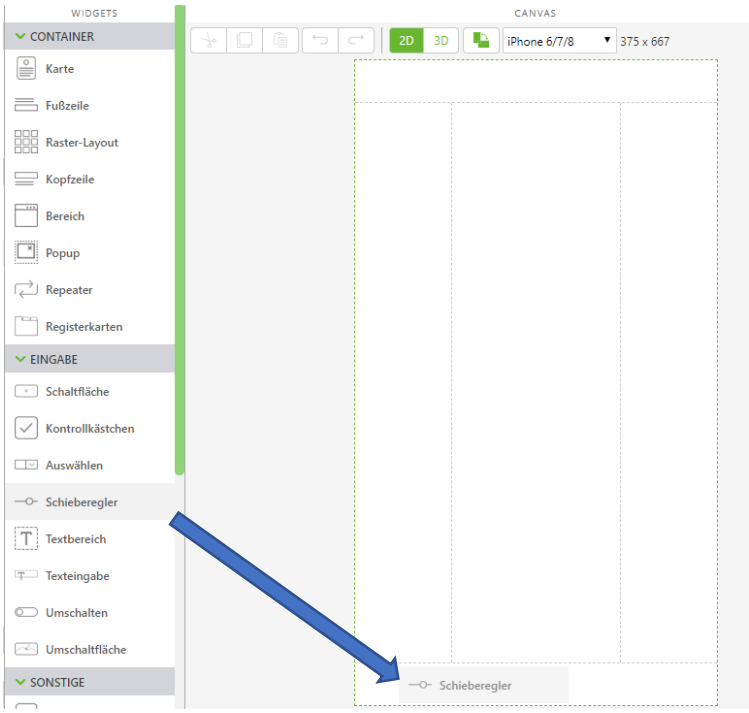
[31] Gebäude: German for buolding
[32] Drehteil: German for turning part

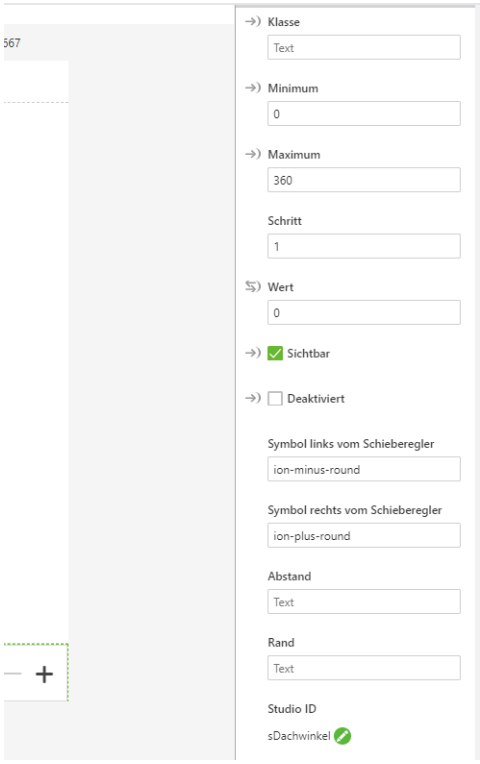
If all axes are selected correctly, our model can be animated quite nicely. Now we should be able to do this from our smartphone while viewing the experience and not just here, in the design phase. For this we will need a control element. Fortunately, our smartphone has a touchscreen as a serving interface. We can have a wide variety of things pop up there. I'm thinking of a slider that you can use to adjust the y-rotation of the roof section.

To do this, we can switch the canvas in the upper area to "2D". Then a white area appears and you can select a sample device (in the picture it says `iPhone 6/7/8`). It is important to know: It doesn't matter which phone/tablet is selected there - it will always work on all devices.



The background of the button is that you can view your layout on different screen sizes to judge if the thing remains usable. Let's start with our slider.





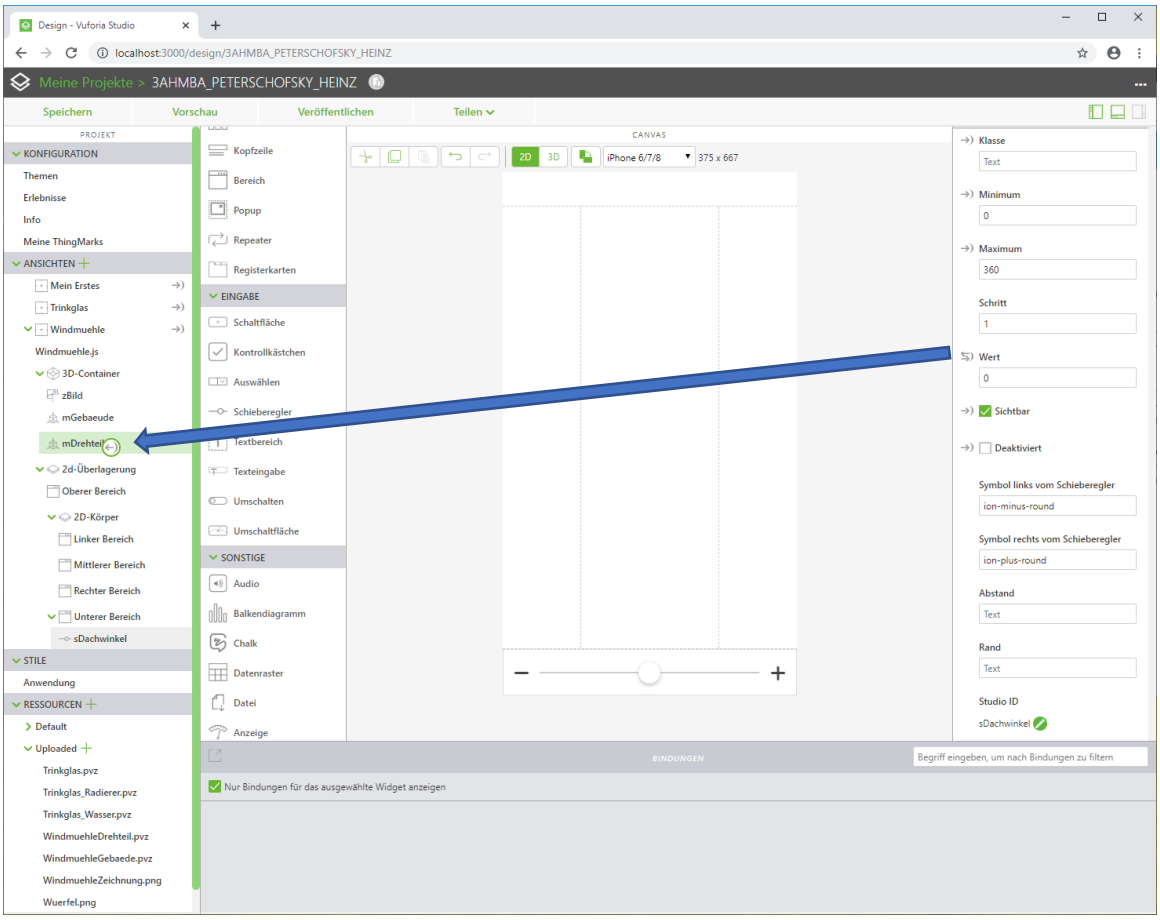
As usual, we drag the appropriate widget to where we want it. I decided to display the slider in the lower area, so I drag it there. It will also be displayed there with its default settings. As usual, you should also adjust the name again - mine here is called "`sDachwinkel`". Also, it should be a rotation angle and I set: Minimum value: 0; Maximum value: 360; Step: 1; Value: 0

As you can easily guess, these things are the following settings:

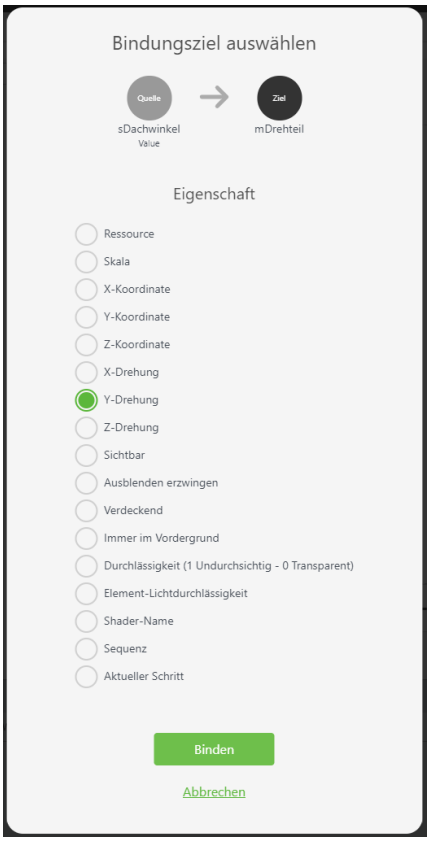
- **Value:** A slider has a numerical value that determines its position. This is the current value of the slider. There is just "the pug". The current position of the control element is displayed.
- **Minimum value:** This is the value at the left end of the slider.
- **Maximum value:** This is the value at the right end of the slider.
- **Step:** This is the minimum allowed change of the slider value. When the slider is moved, its value changes in exactly these steps.
- **Icon left/right:** Here you can change the displayed icons. This is done via the so-called Ionicons code. There are whole catalogs, but not all are supported. A selection can be found in the appendix 4.3.4.

With the settings made here, values between 0 and 360 can be set. You can only change the value in whole numbers and at the start time the value should be 0 at the start time. That should fit for a degree input: 0° – 360°, positionable by degrees.

The slider is here now, but it doesn't do anything yet. Just because its value changes, does not mean that the roof rotates. One must "bind" the value to the rotation angle of the roof. To do this, you just have to take these two arrows next to the word "value" with the mouse and drag them onto the object to which the value should be bound. I want to change the y-angle of my object `mDrehteil` and therefore drag the value there. Here you can see how good it was that we gave our things a readable, interpretable name.



If you release the value there, a selection dialog appears. There you can select with which property of the object you want to connect the value. There is quite a list - so you can manipulate many things of models. Here we want to change the Y-rotation, so we select it in the list and press "Bind".

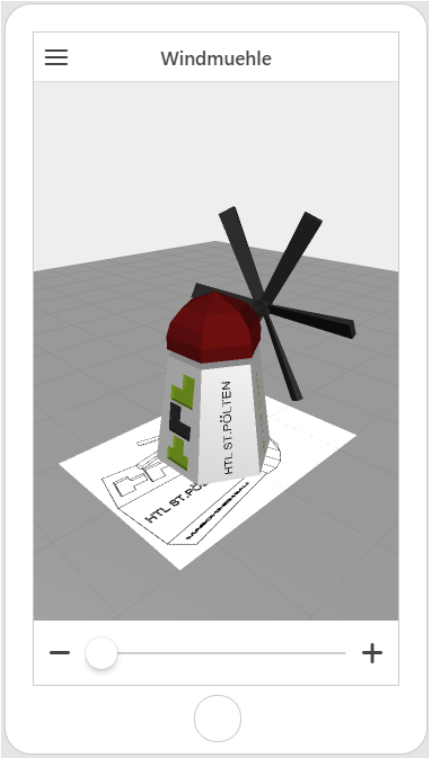


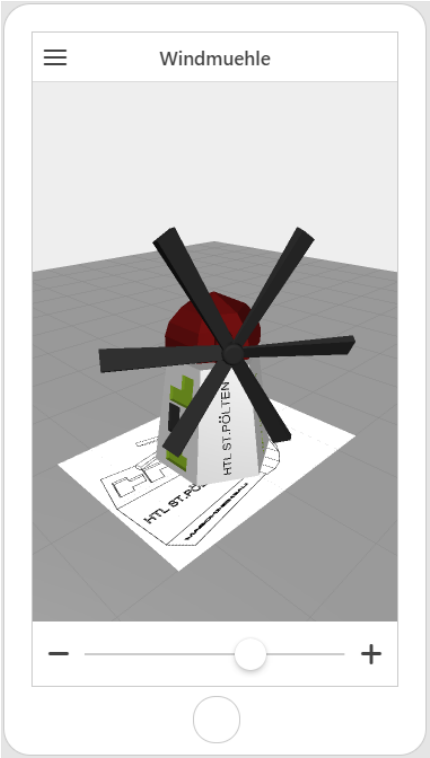
The binding is then also immediately displayed in the lower area:

We could also get rid of this binding by clicking on the trashcan icon. In the picture you can see that the value of the slider is linked to the property ry of the model Turned Part.

That should be it. We can test our experience once by clicking on "Preview". And indeed: Our windmill appears and a slider in the lower area. When we move it, the roof with the windmill's impeller also moves - wonderful, that's how it should look.

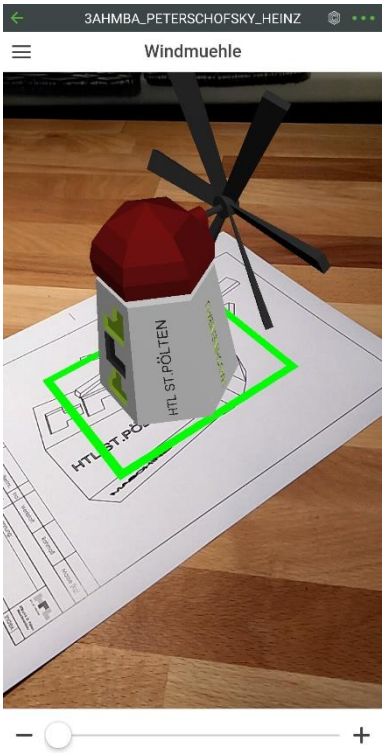
If a part rotates around a strange axis, either the axis assignment was not correct (x-, y-, z-axis somehow wrong) or the origin in the model is in a wrong place.

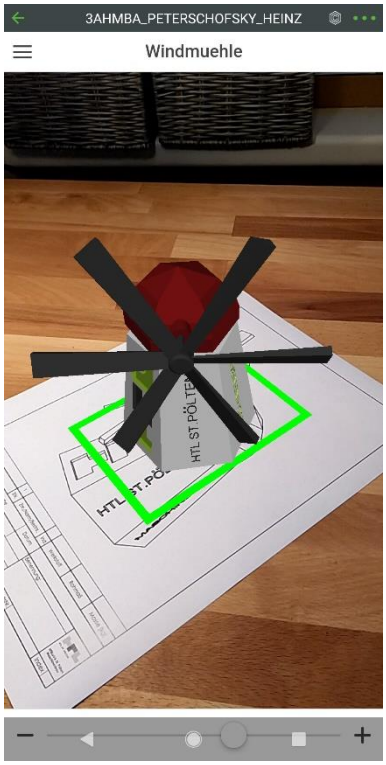




We can publish our experience and view it on our smartphone or tablet. To do this, of course, we need a printout of the drawing that we have used here as a drawing target. Theoretically, it should even be possible to "carry away" our model with the drawing.

Alternatively, simply displaying the drawing on the screen would work. Then our part just stands out of the screen.





2.4.4.1 *Compulsory task "Simple animated Model*

	Create your animated augmented reality with a model of your choice as described above. Publish it and demonstrate your model.	

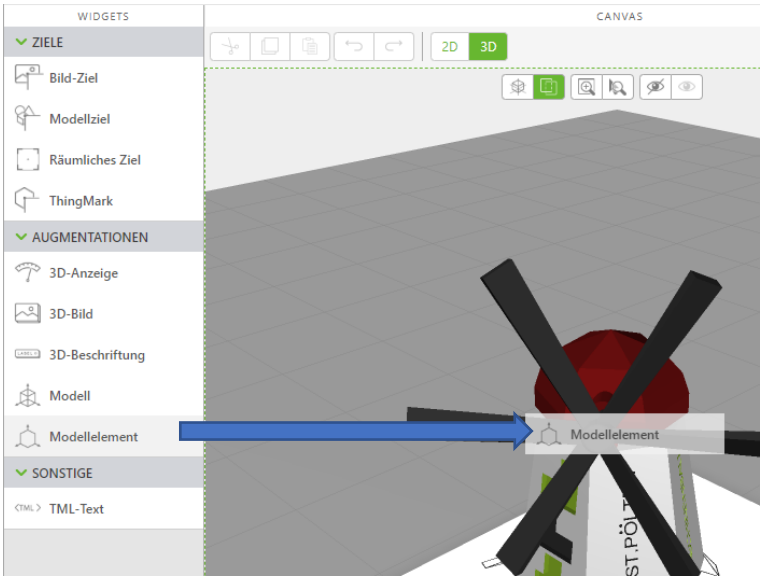
2.4.5 Automatic animation

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

Simply animating the roof of my windmill is only half the battle, because you actually want the rotor to rotate as well. Of course you could add a second slider, but we want more here: We want you to be able to turn the rotation of the rotor on and off. The rotor should then rotate and not stop. The rotation angle of the rotor should increase automatically[33] .

A first difficulty arises when we consider which part we want to rotate. Simply rotating the model `Drehteil` does us no good, because then the whole roof would rotate with it - that was pleasant before, but now it disturbs. We have to use a so-called "model element" here. Vuforia recognizes subgroups in our assemblies and we can select such a subgroup as a model element[34] .

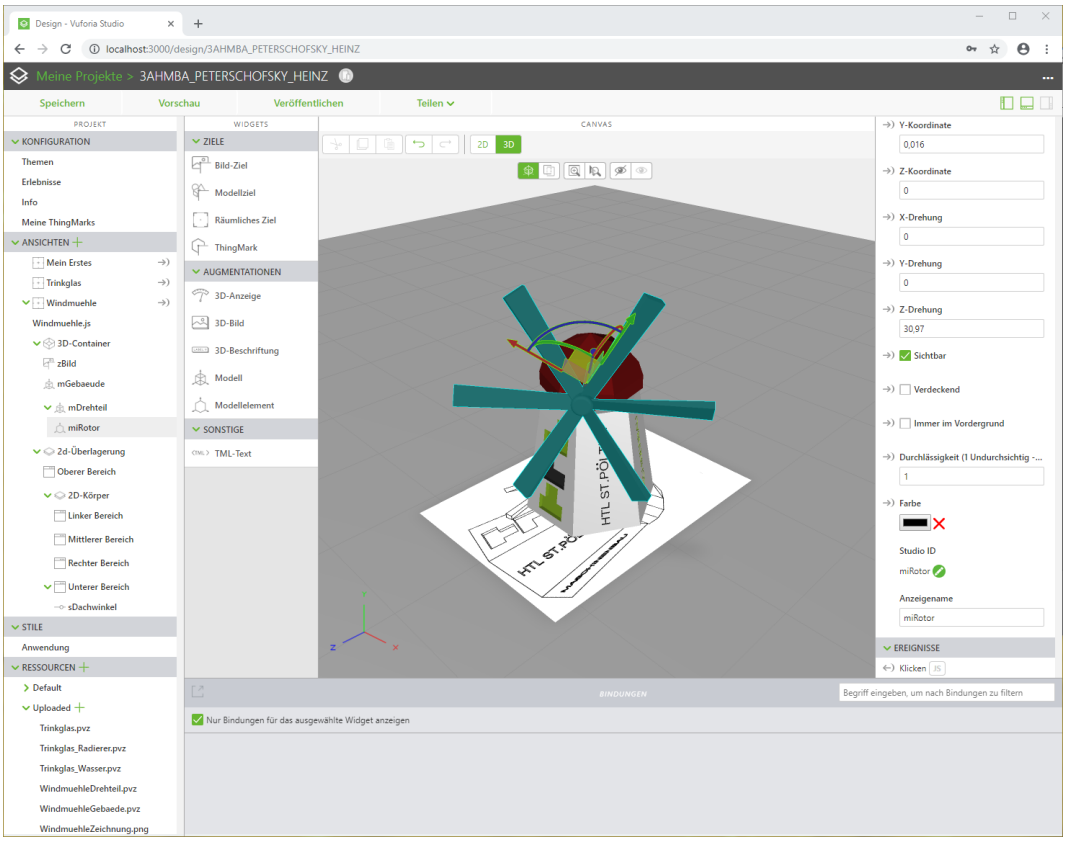
Again, this works relatively easily by selecting the fashion element from the widgets and dragging it onto the corresponding model on it (don't forget to switch to 3D view!). The aiming is a bit awkward, but it works surprisingly well.



Whether we have also hit the right element of our assembly, we can see immediately afterwards, because the model element is highlighted. In the tree on the right, it has also been newly added (logically under the model `mDrehteil`. We give the new model element a nice name (I choose `miRotor`).

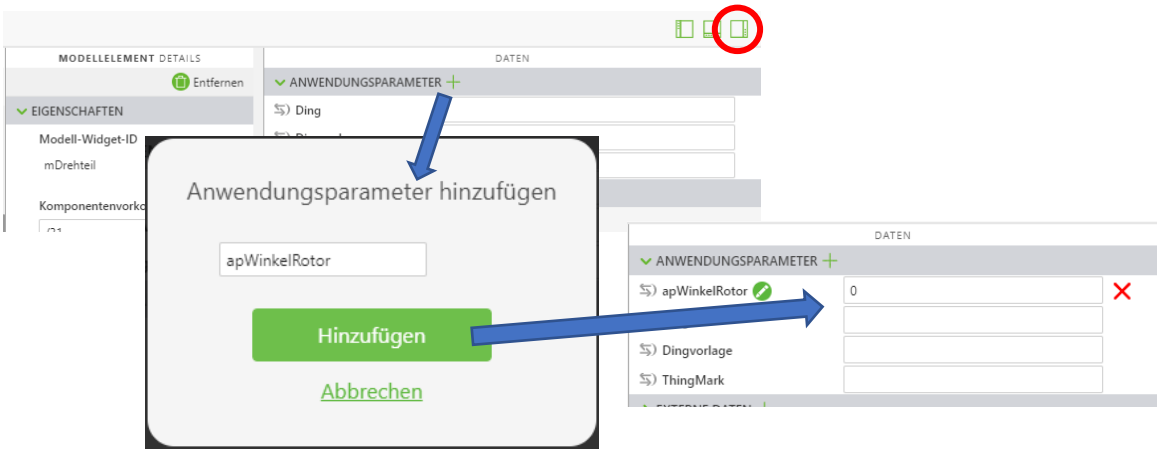
[33] Of course, we're going to make it so that the rotor from 359° again to 0° and also from 0° to 359° instead of letting the angle increase across all boundaries.

[34] You can't do this with arbitrary subtlety. As a rule of thumb, the more complicated the assembly, the more likely it is that not everything will be recognized correctly.

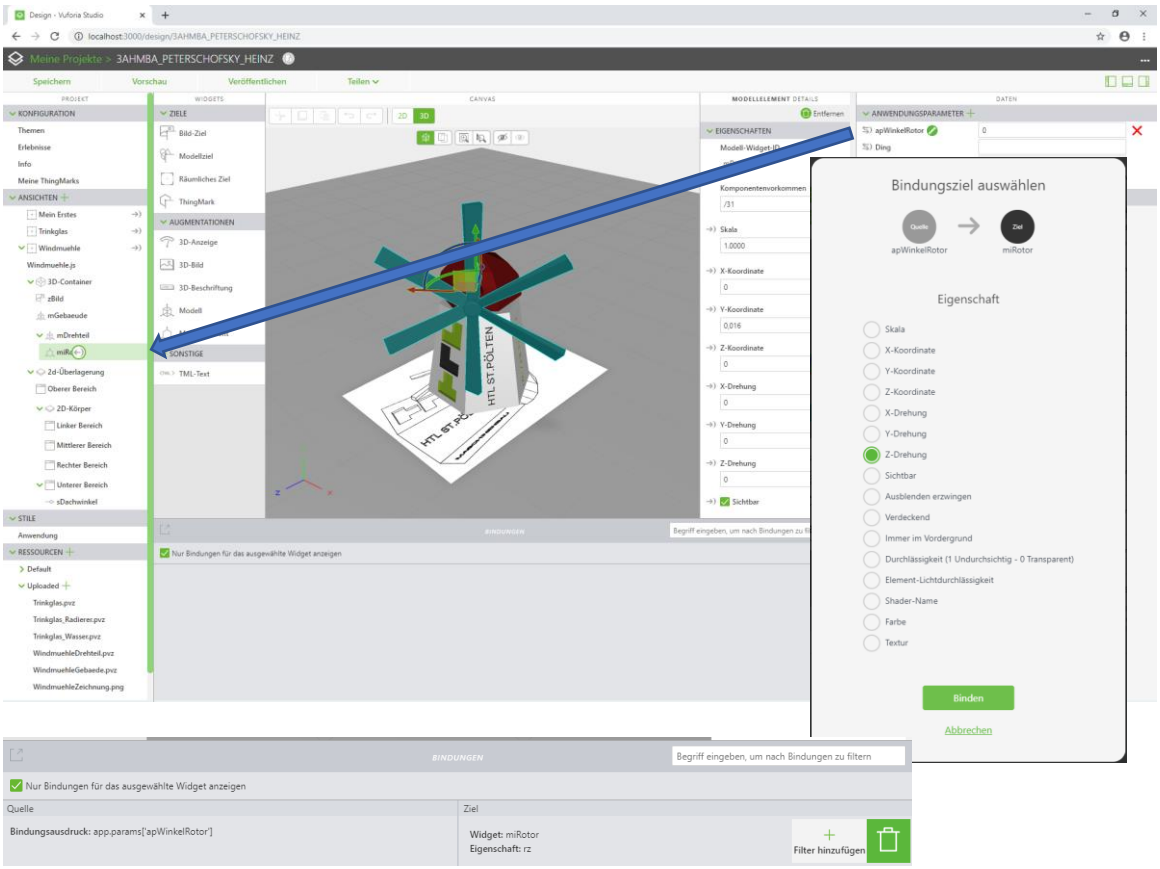


Conveniently, we have also drawn in the axes of our model element. The blue Z-axis is the valid rotation axis. So if we want to simulate wind, we have to manipulate the Z-rotation of `miRotor`.

It has been shown that it is a good idea to create so-called "application parameters". These application parameters can be thought of as variables in a program. You can give them a value, manipulate them, and use that value as a property for an object. Although it only makes limited sense in this example, we will create such an application parameter: We want to use one to control the rotation angle of the blade. If the data area on the far right is not displayed, you can display it by clicking on the right-hand window icon (ringed in red at the bottom).



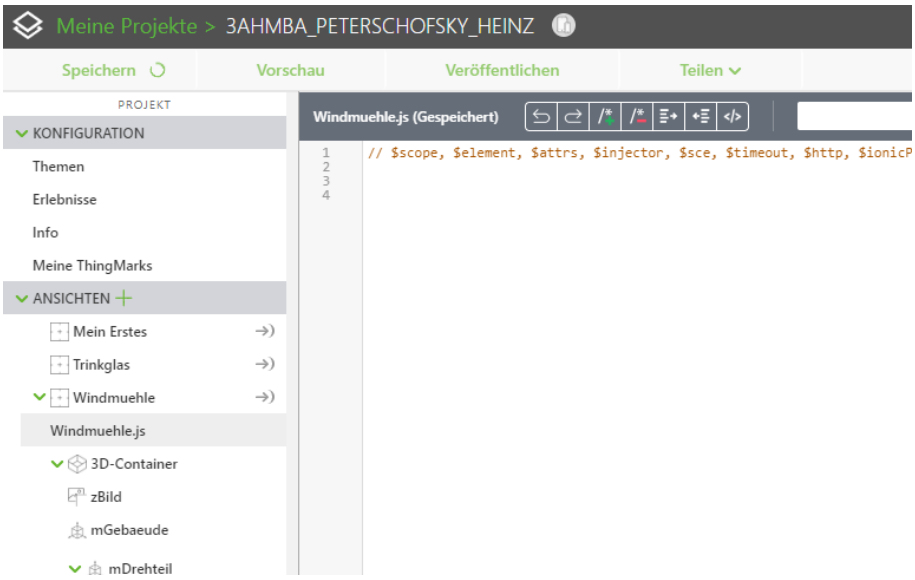
First we press the "+" symbol next to the application parameters. As name I choose `apWinkelRotor` and press add. Then this parameter is created and I give it the value 0. It does nothing yet. I bind the value of the parameter again by dragging the two arrows to my model item `miRotor`, namely with the z-rotation.



This binding is displayed again. Whatever is now inside `apWinkelRotor` will be taken as the Z-rotation angle for `miRotor`.

We add one more application parameter. This should contain how fast the rotor turns. So I add the parameter `apWindStaerke` and set it to the value 1.

Now we just need to manipulate the application parameter `apWinkelRotor`. To do this, we need to do some programming again - this time in the JavaScript language. In the left project tree, at the very top, just below the view, there is the entry "`Windmuehle.js`". There you can program something, so we press on it.



A text editor opens. Here we can write a program. This is:

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

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

$scope.timerFuncSpin = function() {       // calc new rotation
  if (!$scope.app.params.apAngleRotor)    // app parameter check
    $scope.app.params.apAngleRotor = 0;   // set a default value
  if (!$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
}
```

And what does that do?

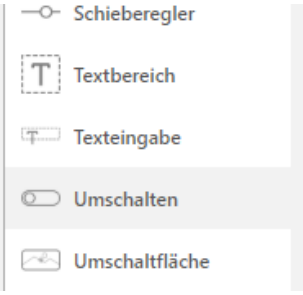
So the function `$scope.spinRotor()` adds the value of `apWindStaerke` to `apWinkelRotor` whenever it is called. After that it divides by 360 and the remainer is taken (modulo operation). Thus the value of `apWinkelRotor` always remains between 0 and 359.

The function `$scope.timerFuncSpin()` only checks whether the two application parameters `apWinkelRotor` and `apWindStaerke` are present. If they are not, a default value is set. This can actually only happen if we have created the parameter but not assigned a value - but the case would be caught here. After that the function `$scope.spinRotor()` is called, with a `$scope.$apply`. This means that as soon as the function has run, the image is recalculated - otherwise nothing would happen.

The function `$scope.fStartWind()` checks at the beginning if a timer is already set. If this is the case, it is deleted. This is also just to be on the safe side, should there be a misoperation. Now the timer is definitely turned off - so we can turn it up. With `setInterval` a timer is called periodically. We call the function `$scope.timerFuncSpin()` and the value `timingInterval`. So the function `$scope.timerFuncSpin()` will be called every $50ms$ called. The wheel is spinning.

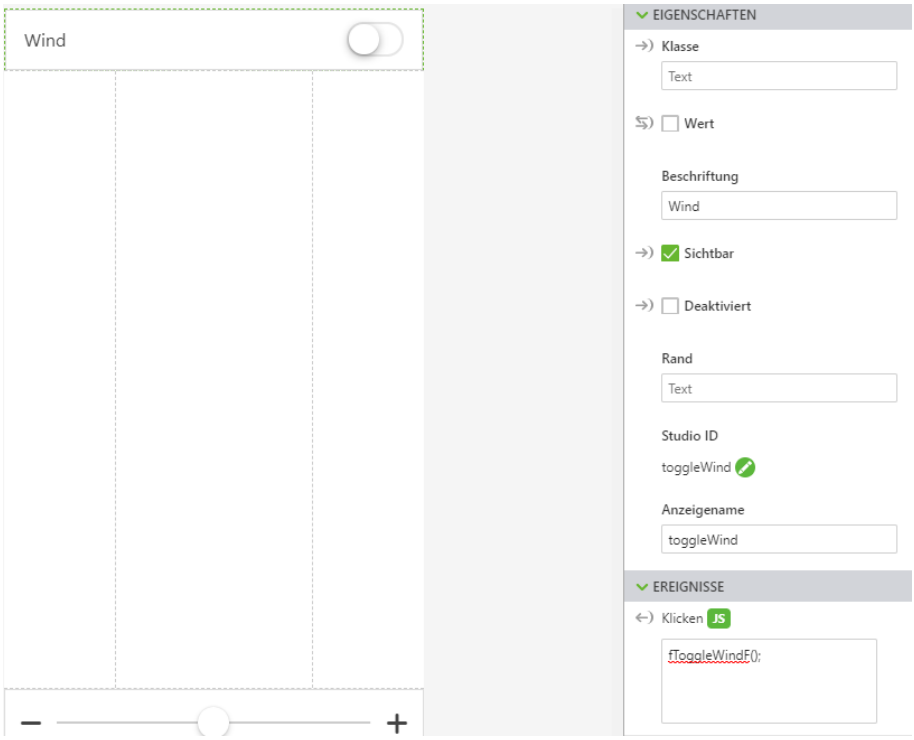
The function `$scope.fStopWind()` clears the timer again - the wheel stops.

So we only need to call `fStartWind()` and the rotor will move. If we call `fStopWind()` afterwards, the rotor will stop again.



To accomplish this, we're going to build a toggle button into our 2D interface. A toggle to 2D and we use the corresponding widget "Toggle" there and drag it to the upper area.

I choose "Wind" as the label and set the Studio ID to `toggleWind`. Now we have a switch that can be toggled on and off. What we still need is a function that we call every time we click it. Unfortunately, we can't call two different functions depending on the status (on/off), we can only call one. So we think of another name - I choose `fToggleWindF()`.

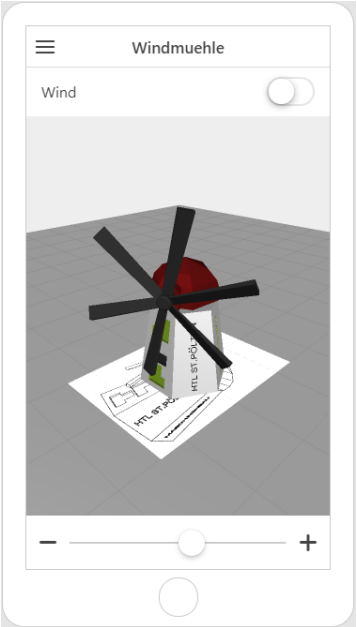


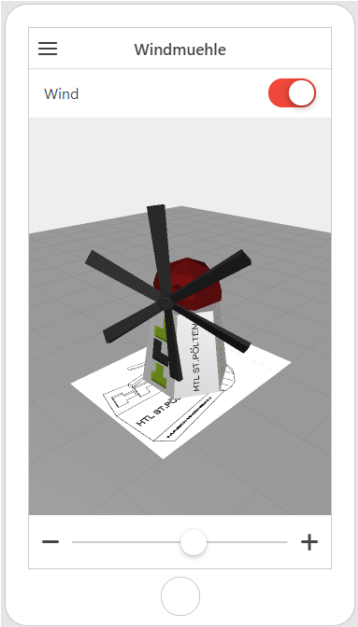
B

ut we still have to program this. In "Windmuehle.js" we therefore add the following code:

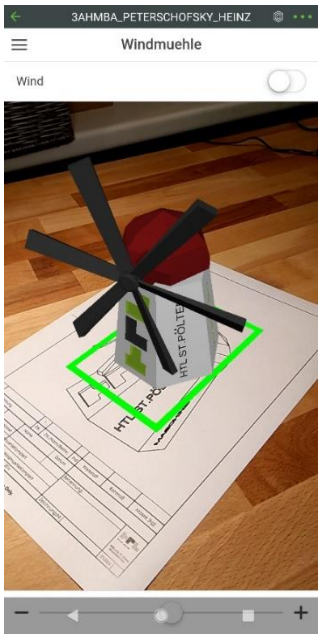
```
$scope.fToggleWindF = function() { // function to start/stop turning
  if ($scope.view.wdg['toggleWind']['value']) $scope.fStartWind();
  else $scope.fStopWind();
}
```

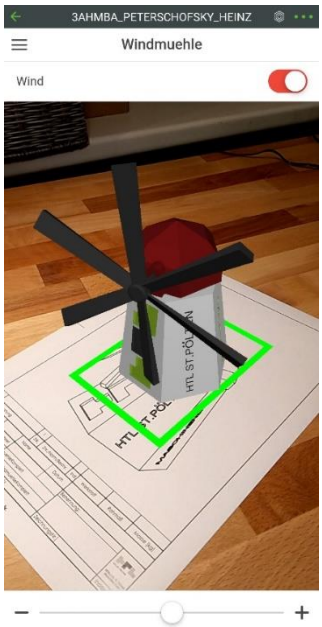
What does it do? If the value of the widget `toggleWind` is `TRUE`, the movement is started - otherwise stopped. With this, it should work. Let's try this now with the preview - and really: it works! The rotor blades move when we turn the switch on and stop when the switch is off.





So it's worth a shot to post the Experience and view it with our smartphone. Again, it seems to work.





2.4.5.1 Mandatory task "Advanced animated Model

	Create your animated augmented reality as described above with a model of your choice and automatic animation. Publish it and demonstrate your model.	

2.4.5.2 *Additional tasks "Wind adjustable*

	Modify the Experience so that the animation speed of the automatic movement can be made slower and faster with a slider. The slider should only be shown when the animation is running.	

2.4.6 Project for a 3D device (Microsoft HoloLens)

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

We have probably noticed that the things we did before - depending on the end device - work better and worse. So we are dependent on the implementation of the sensors in the respective smartphone or tablet. As already described in the general part, this is a general problem of AR. Let's not forget that it is still a phone.

With devices that explicitly have augmented reality as a use case, things look different. We will use the Microsoft HoloLens in class. In the meantime, this is available in two generations. The operation of the HoloLens takes a little getting used to, but it works quite well after a few attempts.

We will realize: The image that the HoloLens offers us is not comparable to those of smartphones or tablets. The image stands still in front of us. We can really move around without restrictions and have our hands free - but there is also no user interface where we can simply put our heads down.

We'll just give this a try now - so let's create a new project. Only this time we choose "3D Eyewear - Default".

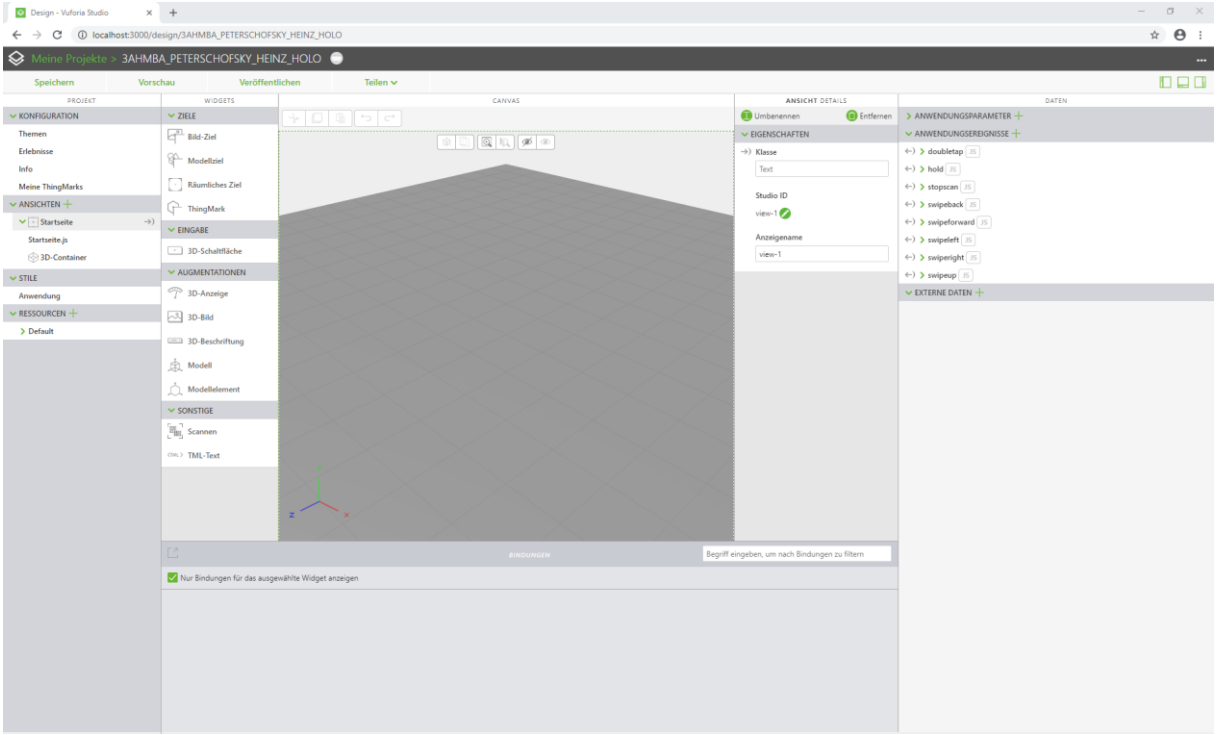
So project name I use this time **3AHMBA_PETERSCHOFSKY_HEINZ_HOLO** (because my default name 3AHMBA_PETERSCHOFSKY_HEINZ is already occupied (from 2D project).

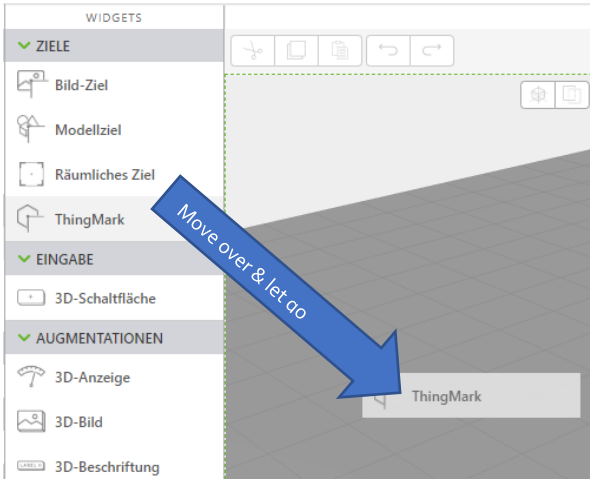
As with the first project, we are prompted for an Experience Service URL - this is the link to our Experience Server on port 8443, so with the above link we write in there:

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

Click the Create button and we get the familiar interface again. At the first block nothing looks different, but in detail there are a few differences: The 2D area is gone - no wonder, there is no 2D interface this time. Instead, there are a few additional

widgets (e.g. 3D display or 3D image). In principle, however, nothing has changed and we should have no problems creating an experience.

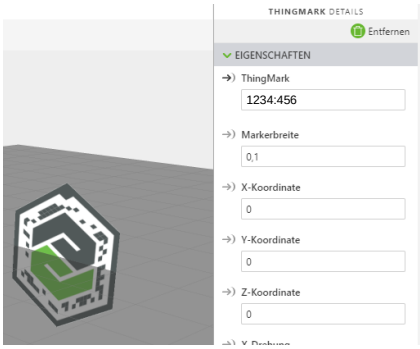




The first thing we'll do is add a target again, this time we'll use a ThingMark as the target.

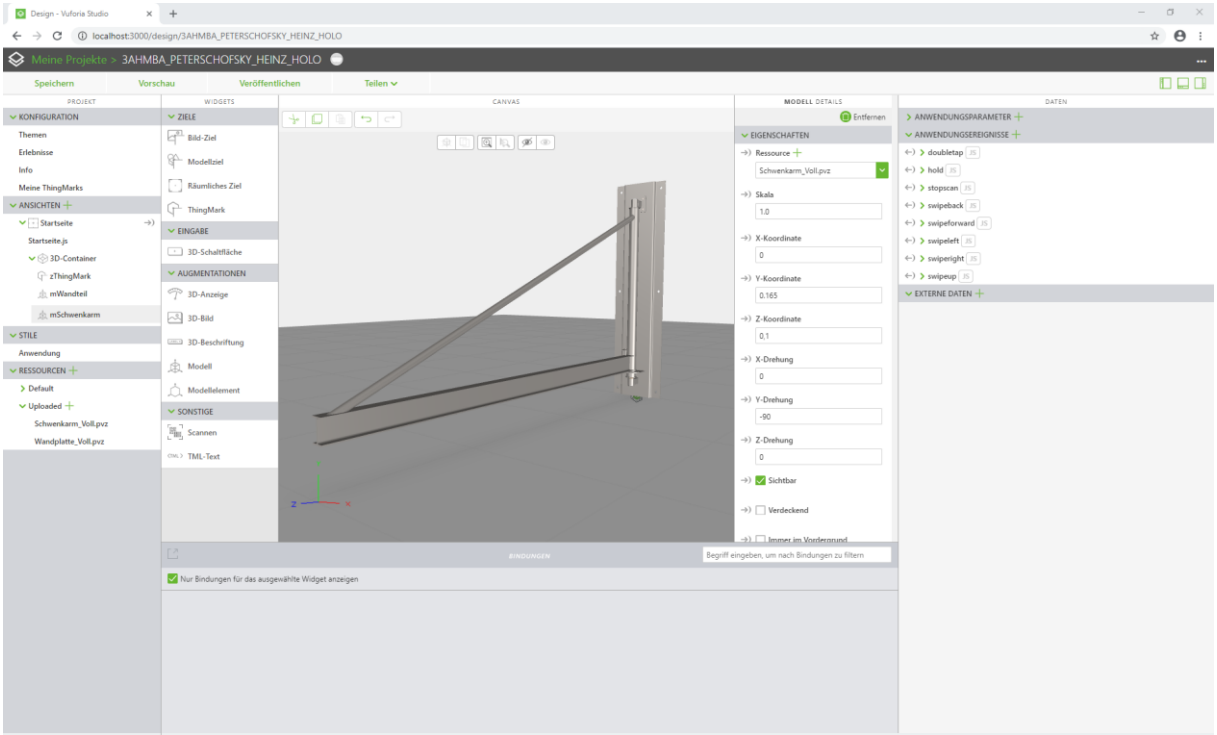
Actually, this works just like the image target from chapter 2.4.4 but instead of the image resource we just have to specify a ThingMark number.

The mark width to be set here is practically the key width of the hexagonal ThingMark. I use here 0,10 - so $10cm$. Also, I want the ThingMark to be vertical, so I set all rotations to 0. Now, set the Studio ID to a reasonable value (`zThingMark`) and we have our target placed.



In the example, I again use a two-part model of a wall crane. So I add the resources I need and make myself two models and name them accordingly.

Positioned and rotated so that the two things fit together. Actually, everyone should be able to perform these steps automatically. Therefore here a screenshot how it should look in something.



For publishing we proceed again in the same way as described in chapter 2.4.3 chapter. So under Experiences set the ThingMark and add a short description. Under Info add a description, validate the experience server and set the access to Public. Select a suitable image as project thumbnail and we are ready to publish.

We can do that and try it once with HoloLens. As a target we can print ThingMark given. This "happens" to be exactly 10cm wide and thus fits exactly for this purpose. We're about to experience the difference. The thing stands quietly in front of us.

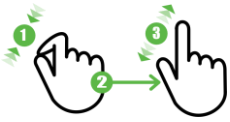
2.4.6.1 Mandatory task "3D augmenting device

	Create your augmented reality for Microsoft HoloLens with a model of your choice as described above. Publish it and demonstrate your model.	

2.4.7 User input with the Microsoft HoloLens

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

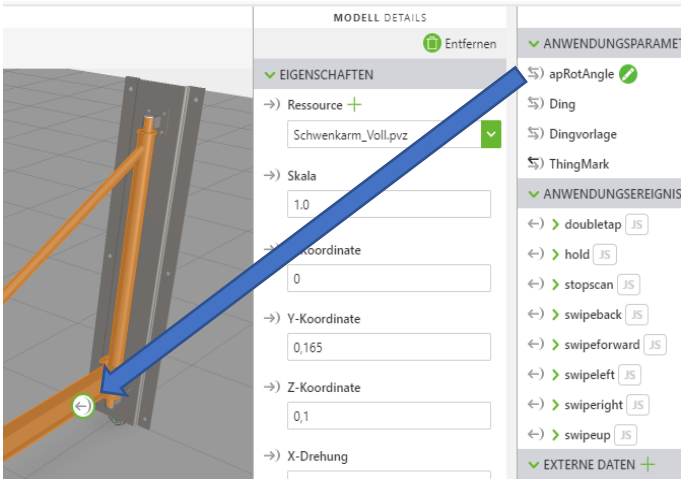
What we already miss with the HoloLens is the possibility of input. There is simply no surface where you can place something. There is the possibility of gestures. Special motion sequences can trigger actions. The sequences supported by HoloLens are summarized in the following table.

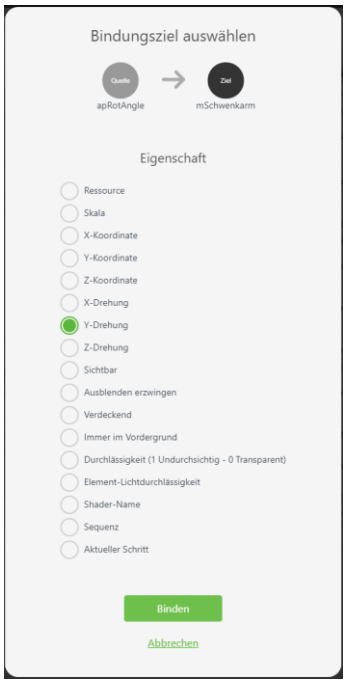
Name	Short title	Symbol	Explanation
doubletab	Double-click		Move the index finger to the thumb twice in succession.
hold	Hold		Place the index finger on the thumb and leave it there
swipeback	back		Place the index finger on the thumb, then pull it towards the body and then extend the index finger again.
swipeforward	pull forward		Place the index finger on the thumb, then pull it away from the body, and then extend the index finger again.
swipeleft	pull to the left		Place the index finger on the thumb, then pull it to the left and then extend the index finger again.
swiperight	pull to the right		Place the index finger on the thumb, then pull it to the right and then extend the index finger again.
swipeup	pull up		Place the index finger on the thumb, then pull it upwards and extend the index finger again. By default, this gesture triggers a rescan.

As an exercise we want to realize the following: A swiperight is to swing the crane to the right. A Swipeleft should swing the crane to the left. A Doubletap should bring the crane back to the middle.

As in chapter 2.4.5 we use an Application parameter. So we add one, name it `apRotAngle` and give it a default value of −90.

After that we still bind the parameter with the Y-rotation of the turning part.





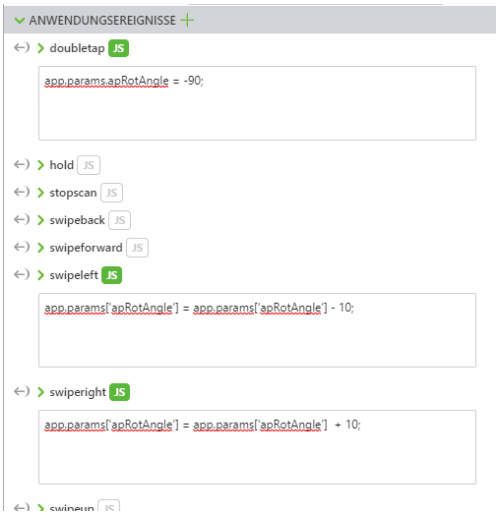
If you press the "JS" button next to the gestures, a small JavaScript window opens. There we can write our manipulation directly into it[35] . In principle, we have two possibilities to access the application parameters. Either we write there

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

or

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

The result is the same (both variants are used in the screenshot on the right). The type of access should not matter.



[35] Alternatively, we could call a function that we define in the startup.js - much like we did in chapter 2.4.4 have done.

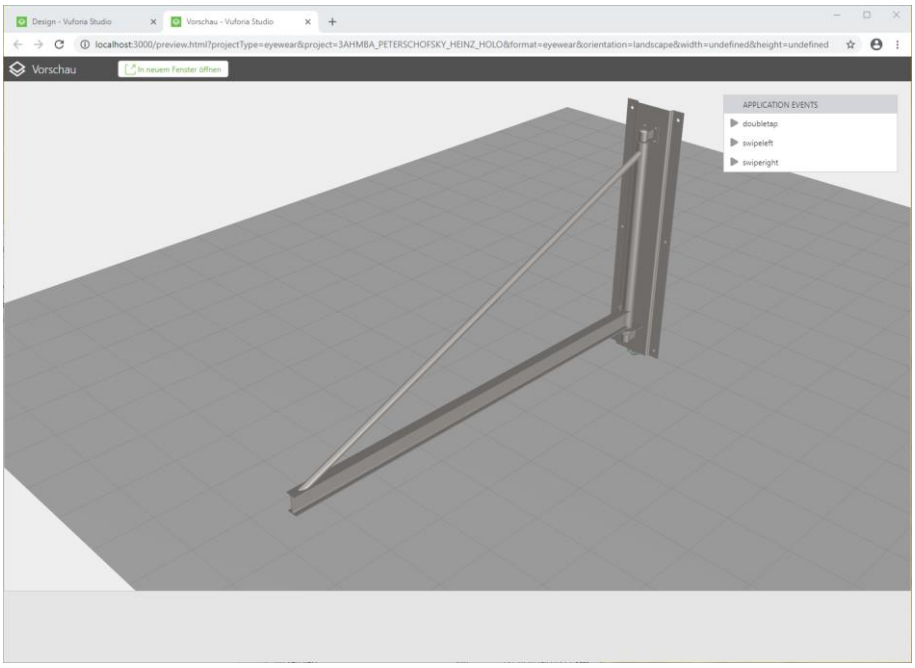
So for doubletap we want to set the value back to -90. For Swipeleft we want to decrease the value by 10 for swipeleft:

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

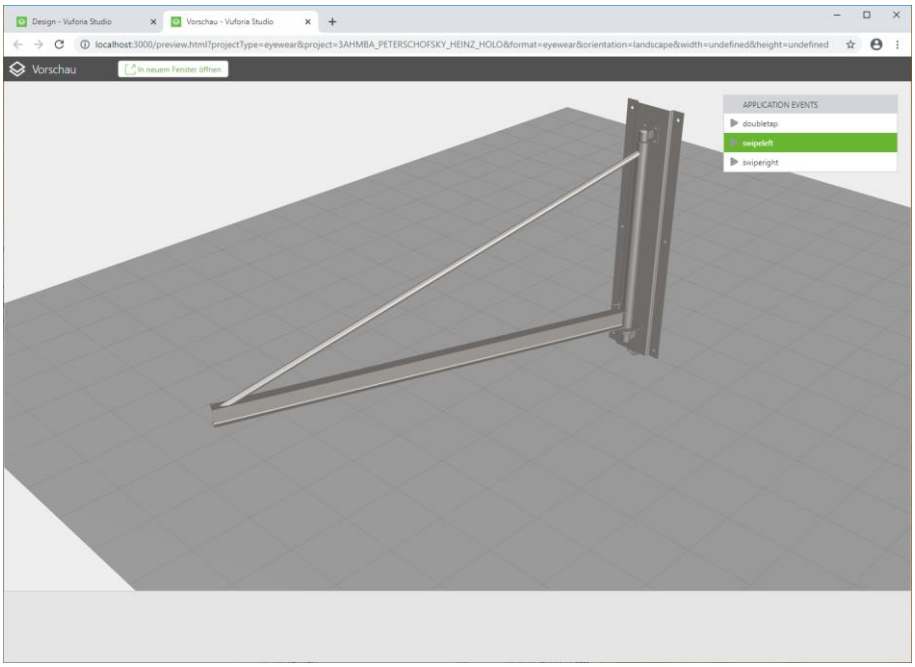
And for Swiperight, the value is to be increased by 10 be increased:

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

We can now try the whole thing out again with the preview. Of course, it's no use fidgeting around in front of the screen. That's why the gestures are shown in the upper right corner for selection.

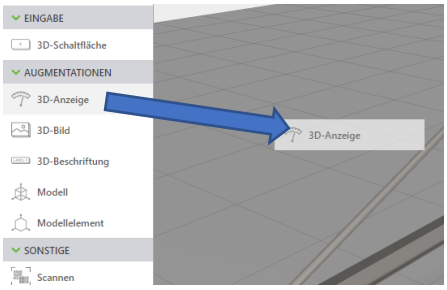


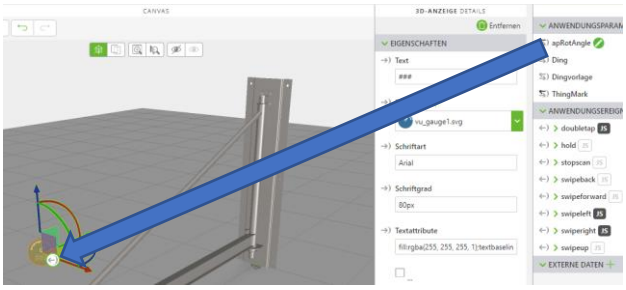
One click on an event and it will be executed. For example, let's try Swipeleft and we will see the crane turns in the right direction:



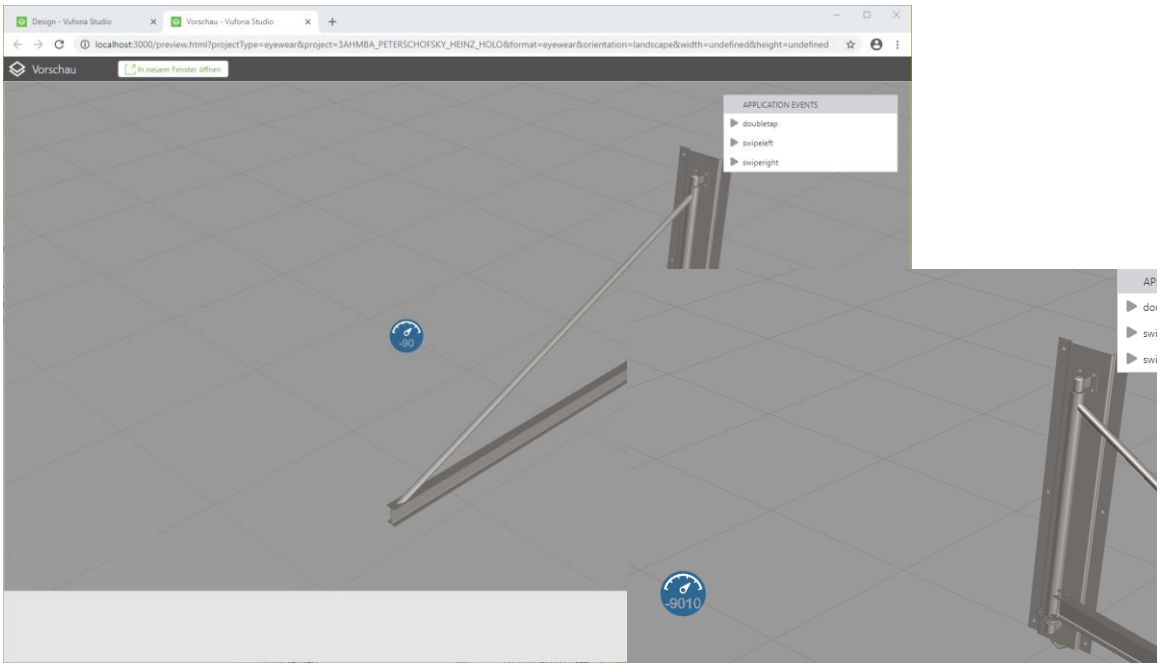
At the very beginning our crane sometimes shows a strange behavior. It jumps at the first swiperight somewhere. If a swipeleft or a doubletap happens before, everything is fine.

To analyze the behavior, we want to display the value on a screen. To do this, we use the 3D Display widget. We drag it back into the canvas and connect it with the application parameter, namely with the property "Text".

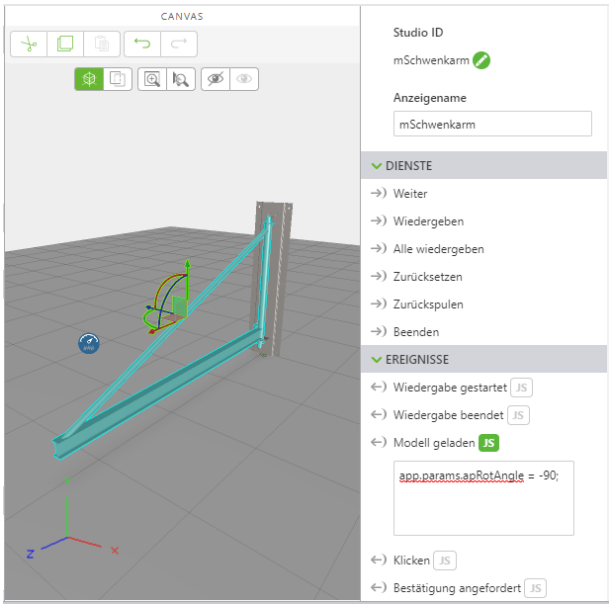




Now the value of the parameter should be displayed. So let's use the preview and see - In fact, the display now shows -90 shows. Now we are excited and press "swiperight". Lo and behold: The display changes to -9010!



This gives us a clue as to what is happening: obviously, the number is interpreted as a string and the string 10 is simply appended. If we keep pressing "swiperight", we actually get: -901010, -90101010, ...



Unfortunately, this can happen with JavaScript, that a type is not automatically recognized correctly. Well, the most feasible solution I came up with is to set the correct value when loading the model. When we select the model `mSchwenkarm`, there is the event "Model loaded". There we add the same call as for doubletap - then it works very well.

We can check this in the preview. In fact, the value now properly changes from -90 to -80. Fits, problem solved - We can delete the display again.

We can publish the experience and try it out with real hardware.

2.4.7.1 *Mandatory task "3D-Device Input*

	Create your input for augmented reality with a model of your choice as described above. Publish it and demonstrate your model.	

2.4.7.2 *Additional tasks "3D-Device Input limited*

	Modify the Experience so that the angle of rotation can only be changed within a reasonable range. No "over-rotations" and thus collisions should be possible.	

2.4.8 Further input options

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

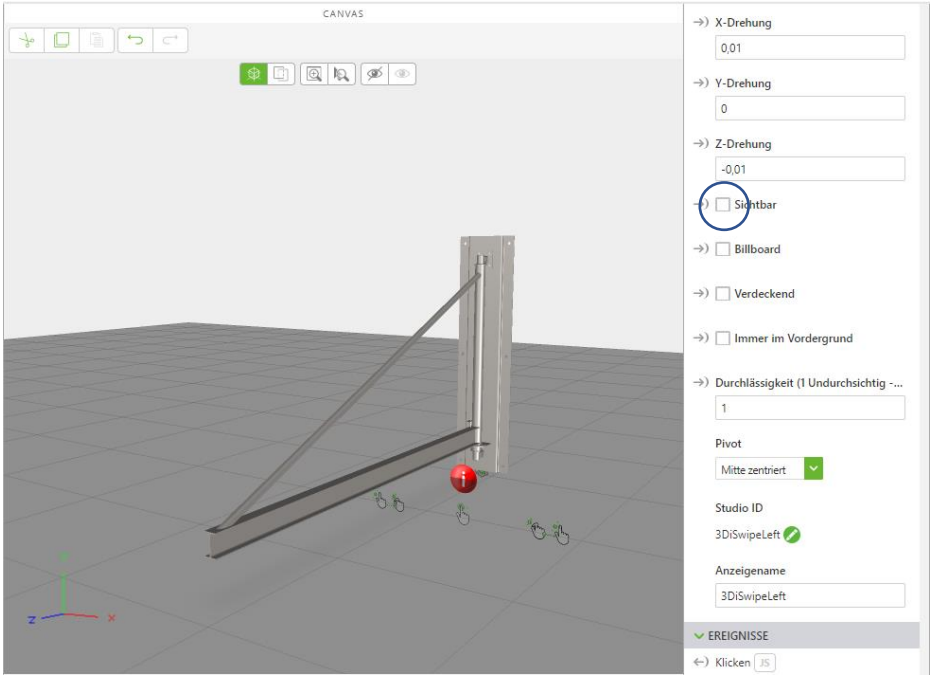
There is the possibility to execute 3D images or labels as buttons. With HoloLens 2 there is also the possibility to place a 3D button. This is exactly what we want to try: We will display information that gives us a little explanation on how to use it. For this we want to use a 3D image. So first we drag the associated widget into the canvas. There, too, we have to select a resource - well, we already know that from the image target from chapter 2.4.4. I made an image with an "i" symbol, I want to show that. I imagine that we can fade in an explanation with one click. So I use the icon as a resource and place it nicely under my crane. Then I'll grab some 3D images and use standard Doubletap, Swipeleft and Swiperight images there. No need to upload a resource for this, just choose a default graphic[36] .

[36] These correspond to the graphs in the table of application events earlier in the script.

I'm assigning studio IDs:

`3DiButton` `3DiDoubleTap` `3DiSwipeLeft` `3DiSwipeRight`

We turn off the visibility of the stationary wheel symbols (click away the check mark). These should only become visible when we click on our "i".



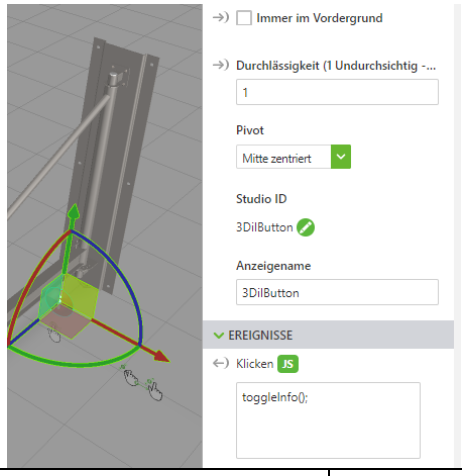
To control the visibility we have to program a JavaScript again. Quite similar to what we already did in chapter 2.4.5 chapter. So we click on "Startpage.js" in the project tree and insert the following code:

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

So we define ourselves a function called `toggleInfo`. The only thing this function does is invert the visibility of `3DiDoubleTap`[37] (so what was not visible becomes visible and vice versa). The visibilities of `3DiSwipeLeft` and `3DiSwipeRight` will then track the visibility of `3DiDoubleTap`.

[37] Please note the " ! " in the assignment!

Whenever we call the function, the 3D images `3DiDoubleTap`, `3DiSwipeLeft` and `3DiSwipeRight` are faded in and out. So this function only needs to be called again when we click `3DiIButton`. There is also a "Click" event on this screen. There we fill in a JavaScript and call our function - by now a snap for us.



Now we can try this in the preview and look: It works!

So publish it and try it "for real".

2.4.8.1 *Mandatory task "3D Image Input*

	Create your input for augmented reality with a model of your choice as described above. Publish it and demonstrate your model.	

2.4.8.2 *Additional tasks "3D-Image Input enhanced*

	Modify the Experience so that when you click the Doubletap, Swipeleft/-right icons, the same action happens as the action itself.	

3 Internet of Things

Playlist link: https://youtube.com/playlist?list=PLVut1tKPvtvP-qYca6ecv-Fa2ug1_UpHP

3.1 General

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

Often it is desirable to have data available that is generated somewhere in a plant or machine. Sometimes another part of the plant needs the information, other times you want to use these measured values to display them centrally. Sometimes you might also want to enrich an augmented reality display with current values - e.g. by displaying the current measured values of the plant directly in the field of view. Often, it is not only the current value that is interesting, but also the temporal progression over the last minutes or hours.

What all these things have in common is that the data must somehow be brought from the point of origin (sensor or measuring device) to the point of display. For this purpose, a kind of "data mediator" is necessary. A server that can be reached by all participants. Some participants store their current measured values on the server. Other participants fetch the data. It is irrelevant who writes or fetches the data at which time. Each participant can publish his data and obtain other data - so there are normally no "exclusive" roles.

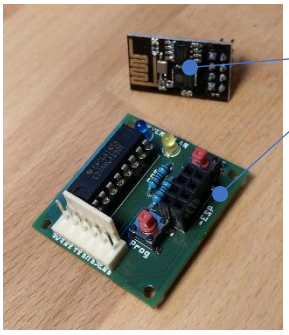
This makes it possible to exchange data within systems and via the Internet beyond system boundaries - the so-called Internet of Things - total networking.

3.2 Requirements

- A server that can receive and provide the data
- The server must be accessible by all participants
- Communication must be secure (only authorized participants should be allowed to store or receive data)
- Transmission should work reliably despite time delay in communication.

3.3 Customization of our Arduino kit

Unfortunately our Arduino Uno is not network capable. There are so-called WLAN and network shields, which add this functionality. But these always occupy a few IO pins and you are not very flexible. There are controllers that are WLAN capable and can also be programmed with the Arduino IDE. One of these controllers is the ESP8266. This controller is available in different versions. The simplest is the ESP8266-1, which has a serial interface, two general purpose IOs and can work either as a WLAN access point or WLAN client.



Components

ESP8266-1 Controller

Adapter plate

Connections:

3V3	Power supply +3.3V
RX	Receive line (to ESP)
TX	Transmit line (from ESP)
D0	GPIO #0 (from ESP)
D2	GPIO #2 (from ESP)
GND	Power supply GND

Assembled

Since the ESP works with $3{,}3V$ all signals to the ESP have to be at maximum $3{,}3V$ at high level. An adapter plate is used for the necessary level adjustment of the RX line. This offers the following features:

- An IC of the type `CD74HC4050E` is used to limit the level of the `RX` PIN to $3{,}3V$ to the maximum level. In contrast to voltage dividers, this works very reliably even at high frequencies.
- The necessary connections to the chip select and enable PINs (`CH_PD`) are made internally. This means: As soon as PIN `3V3` is supplied with voltage, the program is started in the ESP.
- The reset PIN of the ESP is drawn on a button. Pressing the `Reset` button is sufficient to restart the ESP.
- To program the ESP, `GPIO0` must be pulled to `GND` at restart. To make this easy a second button "`Prog`" is built in. This connects `GPOI0` with `GND`. This allows the ESP module to be programmed in the adapter board.
- There is one LED for both IOs. The LED with the label `WLAN` is connected to `GPIO0` and the one with the label `SERVER` to `GPIO2`. The sense of the labeling is the purpose here: `WLAN` would mean the ESP is connected via WLAN (or there is WLAN traffic). `SERVER` means the IoT server has been contacted or there is data exchange with the server.

The basic procedure is that a program runs on the ESP, which connects to the WLAN and contacts the IoT server. This means that various types of information have to be transferred from the Arduino to the ESP and vice versa. This is done with a serial interface. We tell the ESP e.g. the SSID and the password of the WLAN and the ESP connects to it. Then we tell it another IoT server and it connects to that. Then gradually new values and these are brought by the ESP to the said IoT server. If there is a message from the server for us, the ESP will inform us accordingly.

Of course, these ESP programs differed depending on the IoT server. However, we will see that in the following. At the moment, two variants are available: for MQTT servers (see chapter 3.4) and for Tingworx servers (see chapter 0).

3.4 Message Queuing Telemetry Transport (MQTT)

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

MQTT is an open messaging protocol for machine-to-machine communication (MMI) that enables the transmission of telemetry data[38] in the form of messages between devices, despite high delays or limited networks. There are no restrictions on devices. So it can be sensors and actuators or even mobile phones, embedded systems or just normal computers.

This gives its so-called "broker". This is nothing other than the central server, which mediates the data (hence the name). The data does not have to be any digits but can be quite general things. From bit to whole videos or other files.

Such data has a name and a content. The name is called "Topic" and is structured hierarchically. This looks something like a folder path. An example of such a topic would be `house/living room/couch/occupancy` or more seriously `pneumatics/wind boiler/pressure`.

Clients can send such topics to a broker or subscribe to them. If the broker receives a topic, it can save this message (at the client's request). Thus, the broker always has the entire data situation of its clients. If another client has subscribed to a topic, then the corresponding message is forwarded to this client - of course this can be several clients. In the same way, several clients can send the same topic (e.g. several control stations for one and the same drive).

That is why MQTT is an interesting option for automation and IoT. Since 2013, MQTT has also been standardized as a protocol for IoT. In principle, there is MQTT, which only works for TCP/IP networks. With MQTT-SN[39] an extension for sensor networks that do not work via TCP/IP (e.g. ZigBee) is available.

Messages therefore always consist of a topic and a content. Thereby a Quality Of Service can be assigned to the message. The following QoS are possible:

- **At most once (Level QoS 0):** The message is sent once - done. If the connection is lost, the message may not arrive.
- **At least once (level QoS 1):** The message is sent until its reception is confirmed. This means that a message may be received more than once because the corresponding confirmation does not "get through".
- **Exactly once (Level QoS 2):** The message arrives exactly once. This also applies to disconnections or interruptions.

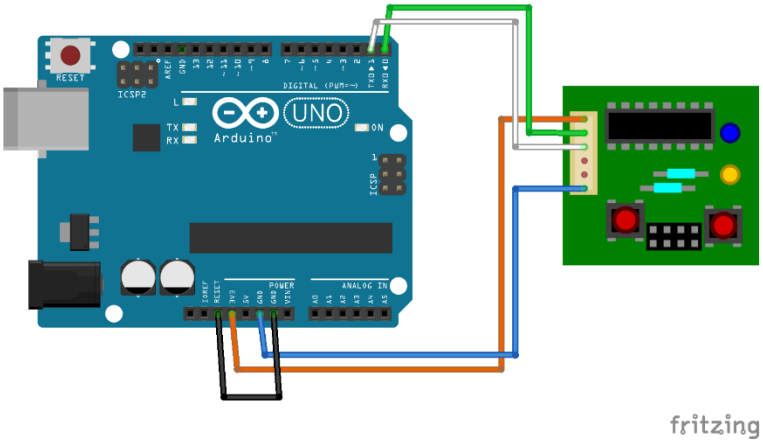
The client can also set a so-called "retain flag" for its message. This means that the broker should please save this message. If there is a saved message for a topic and a client subscribes to this topic, the saved message is transmitted and not only when a new message arrives for this topic.
It is also possible for clients to define a "Last Will" as a message. If the connection to the client is lost, this message is sent to the subscribers.

[38] Telemetry: "remote measurement"; from ancient Greek: tele "far" and metron "measure")
[39] SN: Sensor Networks.

3.4.1 Communication with MQTT

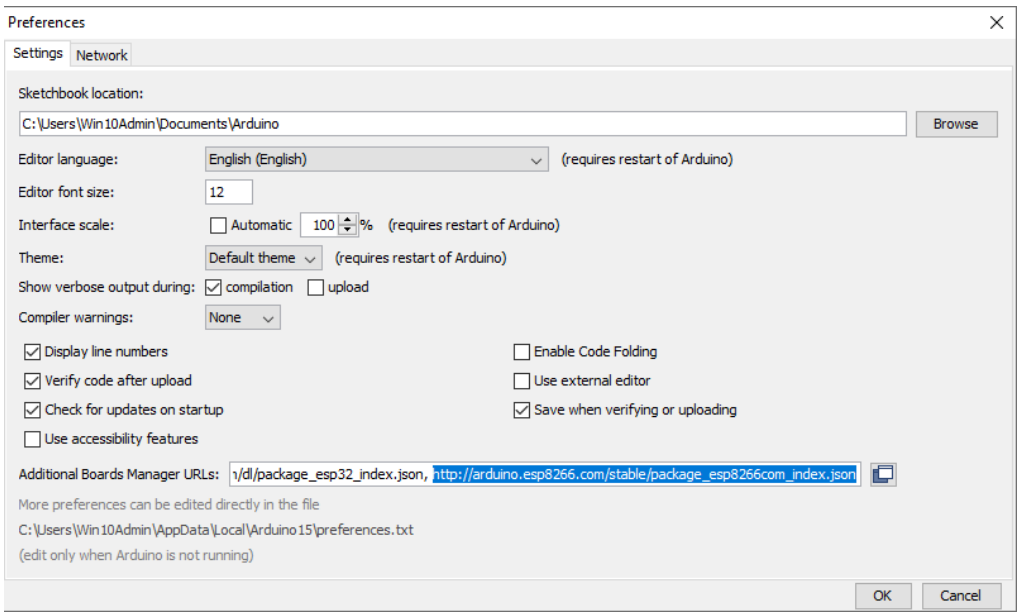
Video Link 3: https://youtu.be/bKwWlMEQZqI

To make our Arduino fit for the communication with a MQTT broker we have to give it network capability. As already described in 3.3 we use our own WLAN-enabled microcontroller: an ESP8266-1 with a corresponding adapter board. First, we connect it to our Arduino as shown in the diagram.



The connection between `RESET` and `GND` is not an error. This switches our Arduino into USB adapter mode. All USB traffic is routed to pins 0 (RXD) and 1 (TXD). So we can communicate with the device hanging "behind" the Arduino without it needing a USB connection. Our ESP doesn't have a USB port, so we can still reach it and program it.

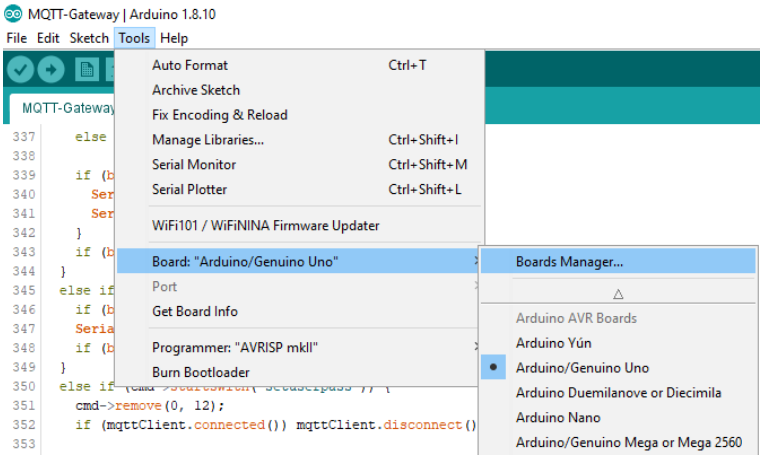
But first we have to teach our Arduino IDE that there is such an ESP board. This is because it does not exist by default. We have to install our own so-called "board manager". To do this, we open the settings from the file menu:



There is an input field with additional board manager URLs. There the following URL is to be entered (is already something entered different URLs with comma "," separated):

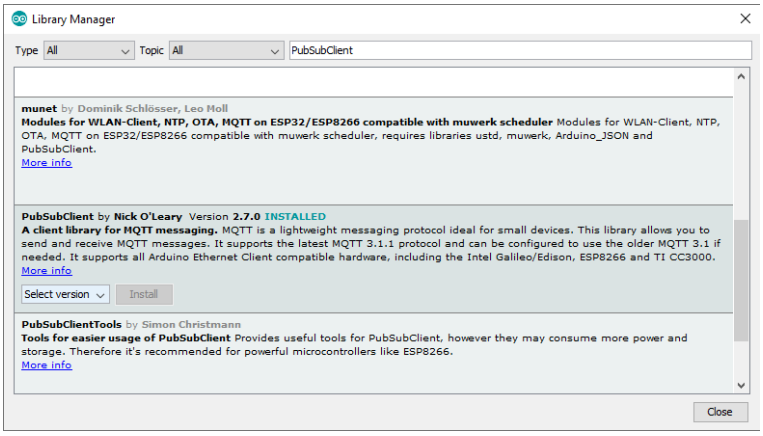
http://arduino.esp8266.com/stable/package_esp8266com_index.json

So the boards can be installed from this URL, but how does that work? One goes into the "Tools" menu and open there the board submenu and select the boards manager:



There you can install new board types from relatively many sources. We need the type ESP8266. There we choose the latest version and install it:

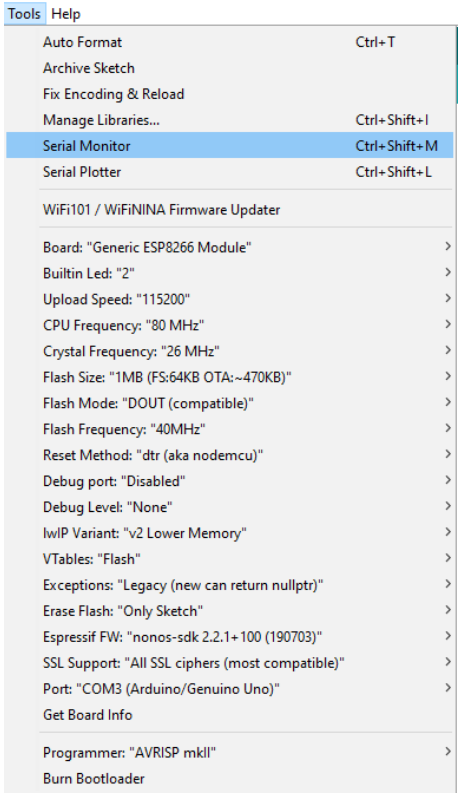
Now the Arduino IDE knows the ESP8266 controller. We want to load the necessary program to communicate with a MQTT broker on the ESP. For this purpose the program `MQTT-Gateway.ino` has been provided. This is to be opened with the Arduino IDE.



What we are still missing is the library `PubSubClient.h`[40] . We have to include this with Manage Libraries.... We need this library to cover the MQTT functionality. Remember: The ESP controller is supposed to connect to the MQTT broker, so we need this library to do that.

[40] By Nick O'Learey - beware there are many "wrappers" that are similarly named - we use the original.

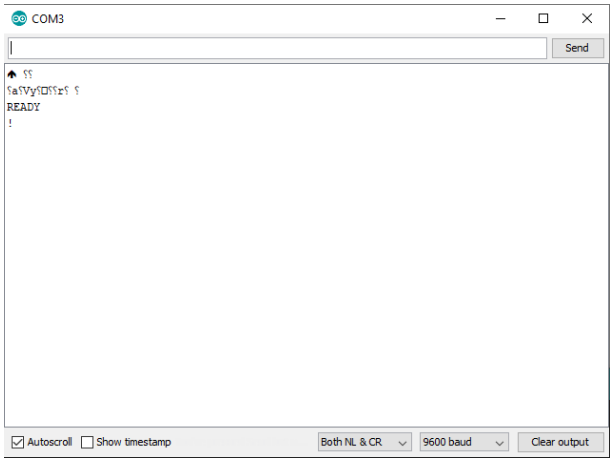
Once we have installed all these things we can load the provided program `MQTT-Gateway.ino`[41]. As board we choose "`Generic ESP8266 Module`" and the COM port is the port of our Arduino.



We have set a board, a COM port and have a program. From our Arduino we are used to: Now we press upload and that's it. Unfortunately, this does not work so easily with the ESP. First we have to put it into programming mode. To do this, we have to keep to the following sequence:

1. Pressing the `Prog` button
2. Pressing the `Reset` button
3. Releasing the `Reset` button
4. Releasing the `Prog` key
5. Download the program

To test if the program has been applied to the controller we can open the serial monitor. The correct BAUD rate is 9600. If we now press the reset button on the connection board we should be able to read the word `READY`. If we type "`?`" the controller should respond with "`!`". Both LEDs are lit - this means we are neither connected to the WLAN nor to the MQTT broker.



[41] You can try to understand what's going on in there. It is written relatively clearly. Only the function `ExecuteCmd()` has a certain length, but that is mostly always the same.

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

We now want to connect to our MQTT server "manually". To do this, we first switch on all possible outputs from the ESP. We type into the Serial Monitor one after the other the commands

```
startwarn
startinfo
startdebug
```

in the Serial Monitor.

If we use `getverbose` now, we get the output for confirmation:

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

The "X" indicates that the output for errors, warnings, information and debug is output. Each command is now acknowledged with a lot of feedback. Let's first try to connect to the school's WLAN. For this we use the commands:

```
setssid WifISSID
```

Feedback:

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

```
setpass WfiPassword
```

Feedback:

```
Set WifiPass: 'WfiPassword'
OK
```

As soon as an SSID and a password have been transmitted, the ESP attempts to establish a connection with the specified network. If a connection is not established immediately, the controller tries again and again. If the SSID or the password is wrong, it is sufficient to correct the value with the corresponding command. The output should look similar to this:

```
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'.
```

If the connection is successful, the WLAN LED should go out. This has energy-saving reasons. When operating with e.g. a battery, it is unnecessary for any LEDs to light up here in the normal state.

So the first step is done - we are connected to the WLAN. The second step has also already been prepared. We can see that an MQTT client name has already been assigned. This consists of a prefix and the MAC address of the ESP WLAN interface. So this should be unique.

Now we need to connect to the MQTT broker. Here at the school, an MQTT server is running on a Raspberry Pi. This has the address `210.192.168.1`[42]. So we pass this information to the ESP. We use the command:

```
sethost 210.192.168.1
```

Immediately, the controller will attempt to connect to the MQTT broker. The output will probably look similar to what is shown below:

```
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
...
```

Every few seconds, the controller will try to reach the MQTT broker. However, it does not succeed. There is also a reason code given. This means:

Code	Cause
-4	Timeout (No response from the server)
-3	Connection reset (network connection interrupted)
-2	Connection failed (no network connection)
-1	Not Connected (Error-free disconnection)
0	Connected
1	Protocol error (version conflict)
2	Client ID not accepted
3	Connection not possible
4	Username or password incorrect
5	Not authorized

We are obviously not authorized to connect to the broker. The broker used requires a username and password. Well, we can transfer that. The right things are:

Username: `USER_STP` Password: `USERinnovativ`

So we use the commands:

```
setusername USER_STP
setuserpass USERinnovativ
```

The output will probably look something like this:

```
Set Username: 'USER_STP'
OK
 Set Userpass: 'USERinnovativ'.
OK
Connecting to MQTT broker '210.192.168.1'.
Connected!
OK
```

[42] Please verify the validity of the IP address with the teacher who is teaching you.

To confirm this, the SERVER LED has also gone out. Now and then the LEDs flash. This is a sign that the program is communicating with the server. Congratulations, we did it. We are connected to the MQTT broker and can publish our information there. But how does it work?

Our program supports 50 different topics, or properties. We can create these and give them a name, and then all we need to do is transfer the number and value. So we use the commands:

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

We're also getting confirmation:

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

If we want to transfer a new value now, only a reference to the number is sufficient:

```
setpropval 0 Hello
```

The answer follows on the heels:

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

But does this really happen? For example, we can connect to the property (or even to the neighbor's). Subscribing to such a topic works with the command:

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

As actuation we get:

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

And a little later the line

```
##RCV#0# 'Hello'
```

on. The last value was transmitted to us. De ESP namely repeats the last transmitted value in time intervals. So now the lines accumulate. We can now write, for example:

```
setpropval 0 Servus
```

and we get the following output:

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

To change the repeat time we can also use the command

```
setresendtime 60
```

to use. This would, for example, set the repeat time to 60 seconds. This means that the last value is not repeated continuously, but only every 60 seconds. The maximum value that can be set is 3600 seconds, i.e. once per hour. One does not need to have fear then, however, that values are not transferred. If a new value is transmitted, it is immediately sent away. However, this behaviour would lead to permanent transmission if new values were to arrive continuously. For this reason, a

blocking time is also built in. This means that two measured values are not transmitted immediately one after the other, but only after the blocking time has elapsed. To set the blocking time, you can use the command:

```
setblocktime 0.2
```

to use. This would, for example, set this blocking time to 0.2 seconds. Faster incoming data will not be forwarded (only after the resend time has expired).

Besides the `setpropname` command, there are also `setboolname` and `setvaluename`. Accompanying then are `setboolval` and `setvalue`. Especially the value part is interesting to transfer measured values:

Values usually fluctuate a little. If every change is transmitted, this is unnecessary. For this reason, a transmission threshold can also be defined for the value values. So you can always transmit a new value to the ESP and only if the new value is different from the last transmitted value by the threshold value, a new transmission to the MQTT broker takes place. As a result, small changes are not transmitted (only after the resend time has elapsed), but large changes are transmitted immediately.

A list of commands for the ESP gateway can be found in the appendix.

3.4.1.1 Exercise "MQTT Connect

	Load the software on your ESP controller as described above. Connect to the MQTT broker and publish a topic of your choice. See if you can subscribe to this topic yourself.	

3.4.1.2 Mandatory task "MQTT Chat

	Use a general topic and try to "chat" with a partner of your choice about published MQTT topics.	

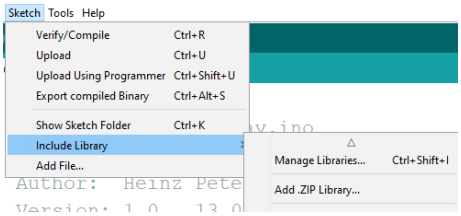
3.4.2 Transferred from an Arduino program

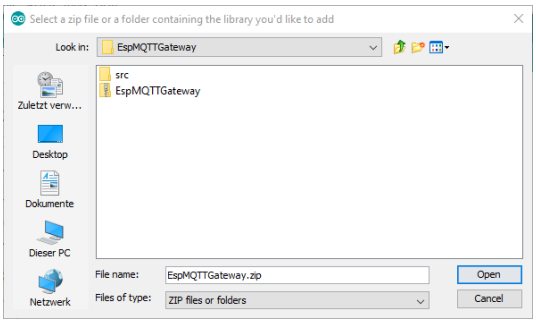
Video Link 5: https://youtu.be/77nMC8lkdl4

It is quite nice that we can connect here and publish any messages. It would be interesting if we could run a program and then when we have a new reading to transmit it. So we want to use above commands from a program. We create a serial interface, connect it to the ESP and give it commands. The ESP doesn't care if we write directly with the keyboard or if these commands come from an automatic system.

In fact, I wrote a small library called "`EspMQTTGateway`" for exactly this purpose. In principle you could of course work with `Serial.print` etc., but the timing is quite critical. Therefore I hid this functionality in a library. This library is available as a zip file. To install it, we have to add a zip library. To do this, we open the Sketch menu in the Arduino IDE and select "Add .ZIP Library..." from the "Include Library" submenu.

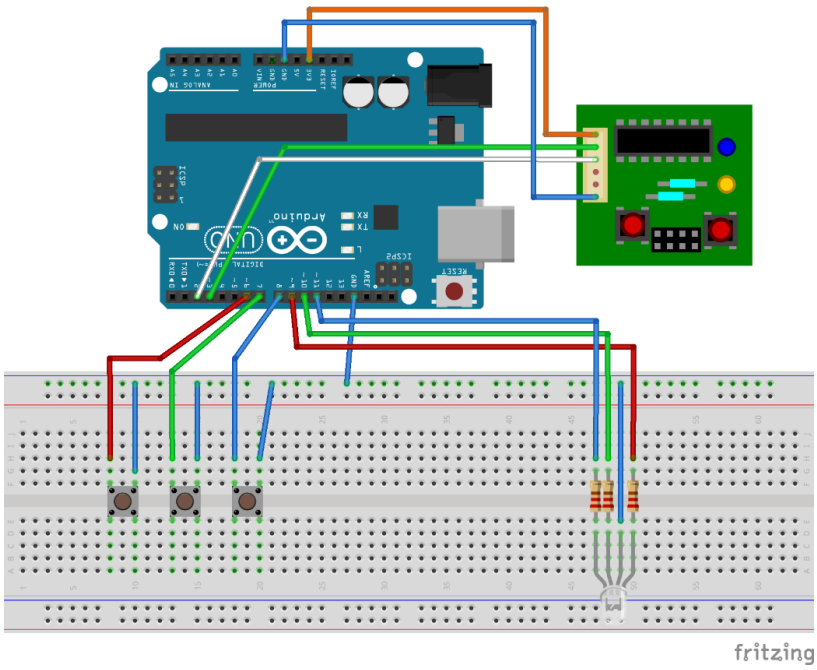
An open dialog will open. There we select the zip file and we should be able to use the library.





In principle we are, but this library also needs the library "`Timeout.h`". We can install this in exactly the same way.

The hardware setup we use looks like this[43] :



[43] Compared to 0 we only have to remove the jumper between `Reset` and `GND`.

Now, after installing the libraries the following program should be "understood" and compile:

```
#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 = "WfiPassword";
String MQTTHost = "210.192.168.1";
String MQTTUser = "USER_STP";
String MQTTPass = "USERinnovative";

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, "myname/led/red");
                mqttClient->SetBoolName(1, "myname/led/green");
                mqttClient->SetBoolName(2, "myname/led/blue");
                mqttClient->subscirbe(0, "other/led/red");
                mqttClient->subscirbe(1, "other/led/green");
                mqttClient->subscirbe(2, "other/led/blue");
        }

        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);
}
```

Let's take a closer look at the code. Let's start like the program flow at `setup()`:

The first call which is interesting is

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

First we create a new object of the type `EspMQTTGateway.` We pass there the address of the software Serial to the ESP port (`&espPort`[44]), the address of the serial interface for the outputs of the library (`&Serial`) and also the pointer to the function `mqttReceived` (`&mqttReceived`). Now the object is created. First we set all possible bits and enjoy many outputs at the debug port (which is the "normal" serial port in our case, so we can watch it on the serial monitor).

Let us now turn to the referenced function `mqttReceived`. There, whenever the received value > 0.0 is the Boolean variables `red`, `green` and `blue` are set to either `true` or `false`. Which variable is handled depends on the number of the received subscribed topic. So for example, whenever we receive topic 2 and the received value is > 0.0 the value of `blue` is set to `true`. We can see what the point of this is by looking at the `loop()` function:

It is ensured that the correct Wi-Fi and MQTT parameters have been transmitted. In doing so, the value is only transmitted to the ESP if it has not already been transmitted. What is the purpose of this? The library checks at regular intervals whether the connection to the MQTT server and the WLAN are OK.

If, for example, the connection to the MQTT server is lost for some reason, the library assumes that it may be working with incorrect parameters and resets the connection to MQTT. This would cause

[44] The & symbol indicates that we are not passing the object itself, but only an address, or a pointer to the object. We just give the library an address and it works there by accessing the address and not just a copy of the serial object.

the MQTT parameters to be transferred back to the ESP here. The same applies mutatis mutandis to the Wi-Fi parameters. Transmission errors between Arduino and ESP can be corrected in this way.

So we publish the topics[45] :

```
0: myname/led/red
1: myname/led/green
2: myname/led/blue
```

And we subscribe to the topics[46] :

```
0: other/led/red
1: other/led/green
2: other/led/blue
```

So now it becomes clear why we manipulate the variable `green` when receiving e.g. Topic 1 - because that's exactly where we subscribed to the topic about the green LED.

As soon as we are connected to the MQTT server we start publishing our topics. We can transmit the values relatively often to the ESP (here we can choose that with the `timeout` object to). The only time it sends is when the value changes anyway, or when the timeout time has expired. So we cause at most a lot of data traffic between Arduino and ESP, but not at the network. Now we transfer the read-in switches to the individual topics - nothing happens in our case. But for all those who have subscribed to our topic, the corresponding LED is on. You can see why in the last three lines of the program. Before that there is the line

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

There simply the library code is called. If you omit this call, nothing can be received. Then the subscribed topics would be useless. In addition, the check of the ESP connection no longer works. In short: It is necessary to call this 1x per loop.

3.4.2.1 Exercise "MQTT Transfer

	Load the software onto your ESP controller as described above. Try with a colleague to turn on each other's lights.	

3.4.2.2 Mandatory task "MQTT analogue

	Do not transmit binary values, but analog values. Transfer the proportions of red, green and blue in such a way that a mixed color appears on the colleague.	

[45] The first part is to be replaced by the own name. I would publish as e.g. the topic "`peterschofsky/led/red`".

[46] Here, too, the first part is to be replaced with the name of a classmate. For example, if I want to subscribe to the values of Prof. Tiefenbacher, I subscribe to "`tiefenbacher/led/rot`".

3.5 Thingworx

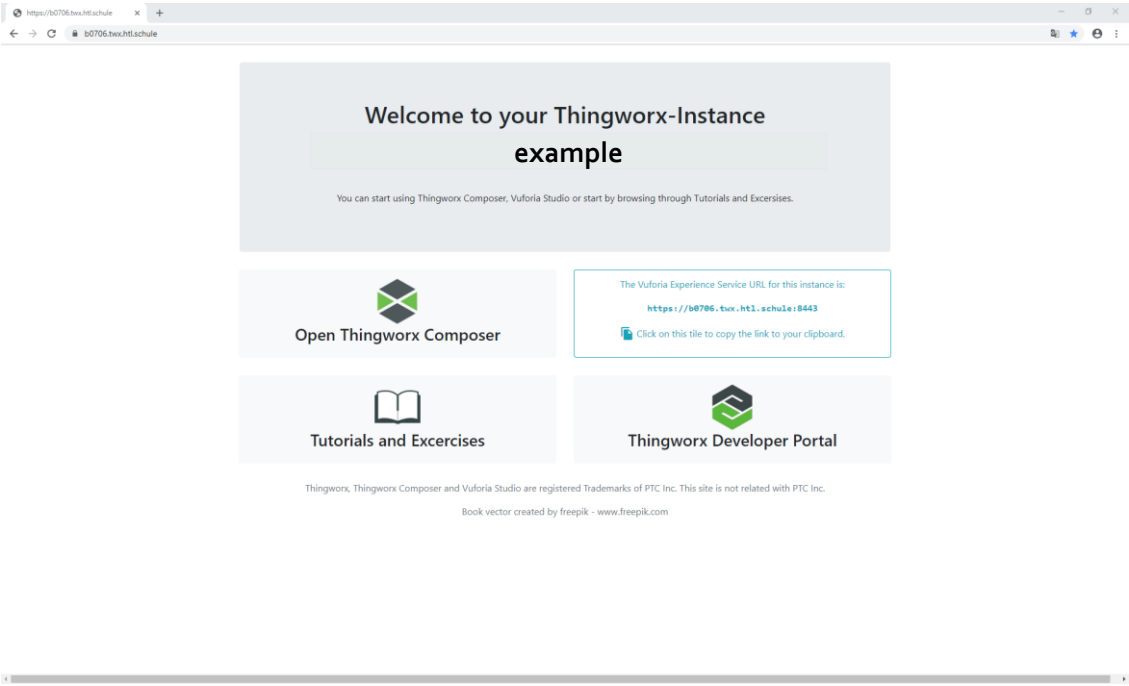
3.5.1 General

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

Thingworx is an IoT platform[47] from PTC, which offers well-integrated interfaces to the other PTC products. So there must be a server somewhere that receives the data and can pass it on if needed. This server is called Thingworx-Instance on this platform and can be reached at the same address as the server for publishing Vuforia-Experiences (see chapter 2.4).

https://example.twx.htl.schule

If you type this line into the address bars of a browser[48] the start screen appears:

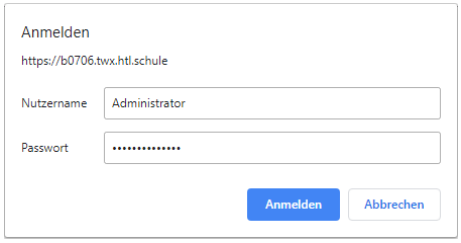


This is virtually the same server that served as our experience server at Vuforia. In fact, the area written in light blue contains the experience URL of the server. Clicking on this font copies the Experience URL to the clipboard. This is intended as a convenience for Vuforia Studio and actually has nothing to do with Thingworx.

Thingworx is operated with the "Composer". This can be reached by clicking on "Open Thingworx Composer". A login dialog appears again, in which we enter our login data:

Username: **AdminUser**

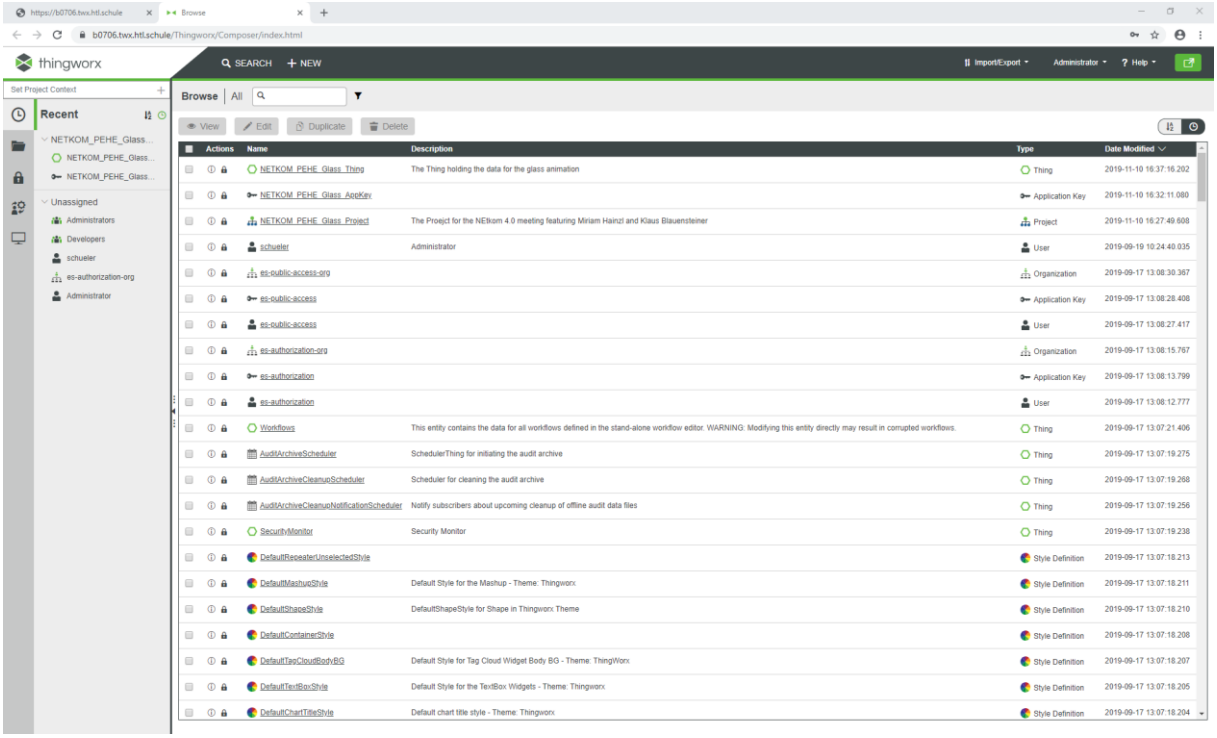
Password: **TWXPassword**



[47] For Internet of Things see also 3.1 General

[48] Again, only Goolge Chorme is officially supported by PTC. All other browsers can cause display errors and functional limitations.

After that, the Thingworx composer loads. At first glance, a completely confusing window with a hodgepodge of strange names and many, many links.



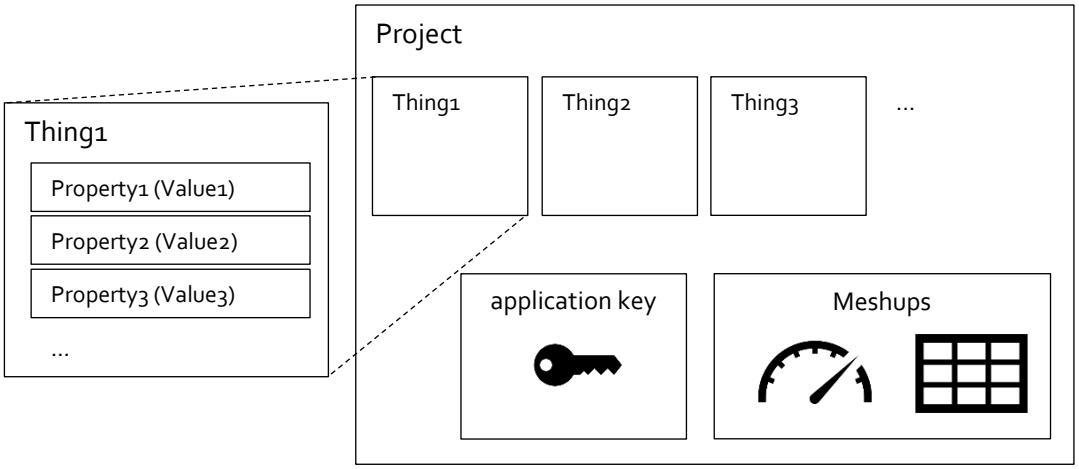
The data on such a server is organized in so-called "Things". A Thing is literally a thing that can collect the data, keep it ready and make it available when needed. This is the central element.

One or more Things are combined in a project. The permissions and the display view of the Things data are also located at the project level.

For the authorization, a project contains a so-called "Application Key". Whoever knows this key is allowed to store his data in the Things. Who is allowed to fetch the data is regulated directly in the Thing.

For display purposes, a project contains one or more "mashups"[49] . These are practically web pages which can display the data. There are e.g. analog or digital displays, but also bar and line charts. In addition, one could add control elements, buttons, sliders, ...

[49] A more common name is also "dashboard" - meaning the instrument panel.



So it's all held together by the project. In the next few chapters, we'll look at how to create a working Thingworx project with a Thing and an Application Key. Then we'll represent that data in a mashup, and then we'll use that data in Vuforia Studio to generate an appropriate representation.

As a naming convention, we use:

KLASSE_NACHNAME_Projekt_

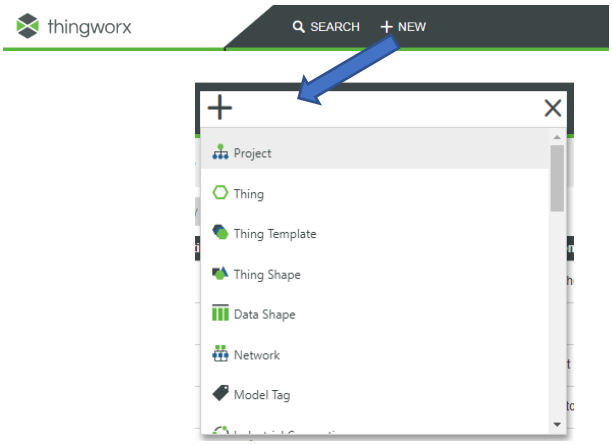
Project:	**KLASSE_NACHNAME_Projekt_Pr**
Thing:	**KLASSE_NACHNAME_Projekt_Th**
Application Key:	**KLASSE_NACHNAME_Projekt_Ak**
Mashup:	**KLASSE_NACHNAME_Projekt_Mu**

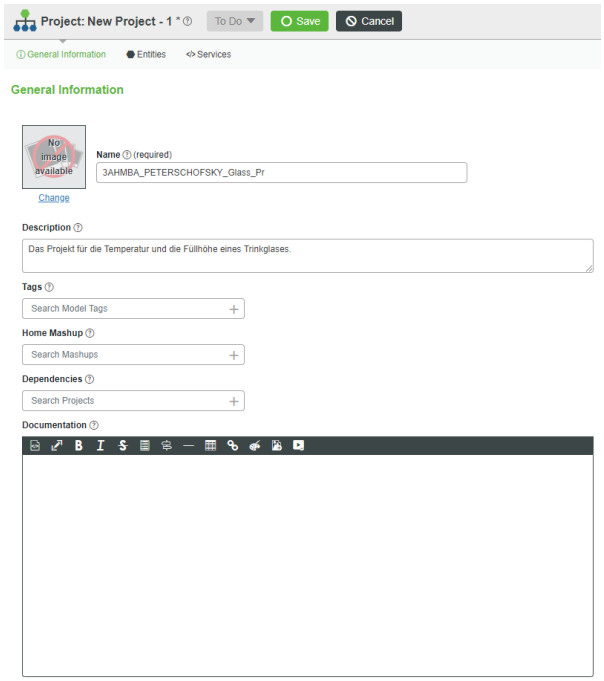
3.5.2 Creating our first Thingwork project

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

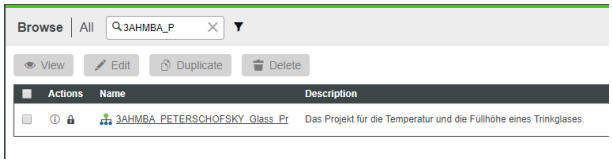
3.5.2.1 The empty project

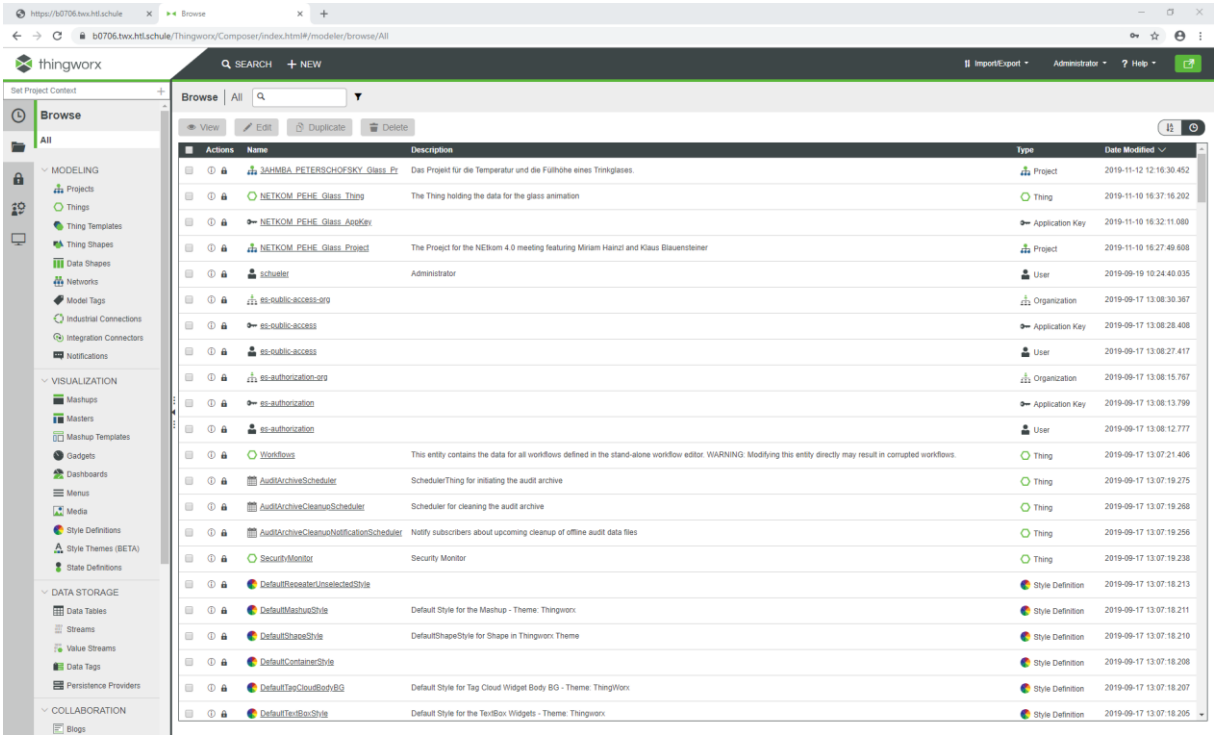
Since everything is held together by a project, we also create this first. We stick to the naming convention. For example, I will use `3AHMBA_PETERSCHOFSKY_Glass_Pr` here. To add a project, first click on `+NEW` in the top bar. From the following menu we select "Project".





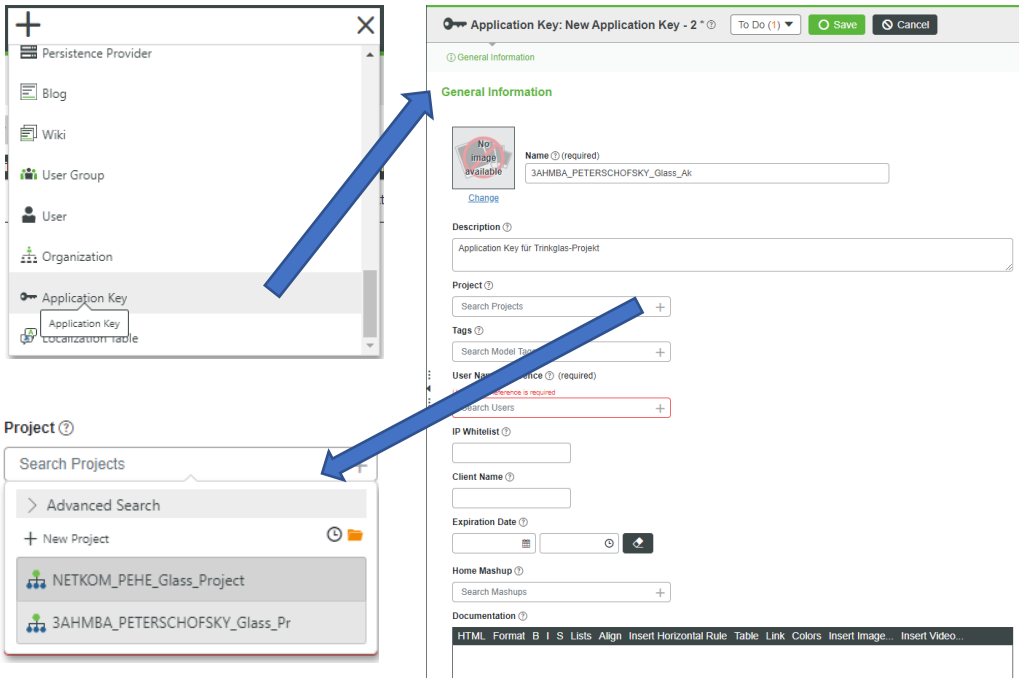
In the following dialog box I fill in the information. Actually, only the project name is important. For the sake of completeness we also fill in the description. A click on "Save" and the project is created. At the moment this project is an empty shell with a name and a description, which we still have to fill with life. You can see that the project has really been created by clicking on the folder symbol on the far left. This will display the "table of contents" of the Thingworx server. By default, all kinds of things are displayed (projects, things, application keys, mashups, users, user groups, color schemes, ...). The newest entry is always on top - that gives you a bit of an overview. If you only want to see certain types, you can also use the filter. At the top of the bar, you can enter the first letters and suddenly only things that start with these letters will be displayed. This brings order into the chaos.





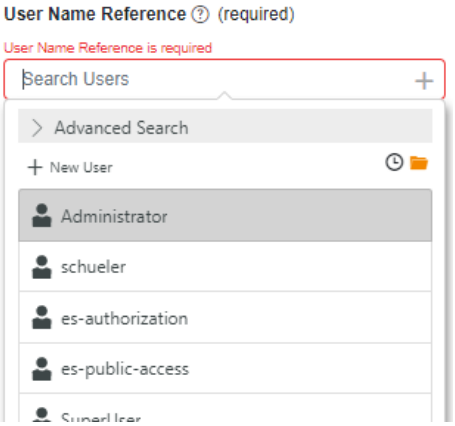
3.5.2.2 *The corresponding Application Key*

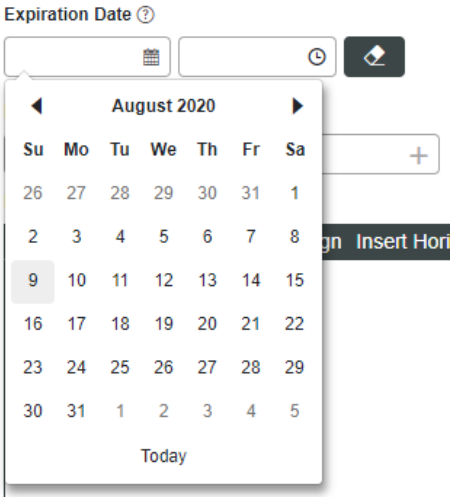
Actually, we really want to create a Thing, but first we take care of the access rights - otherwise it's easy to forget them. So the next thing we do is to create an Application Key. This is the same as for the project, but this time we choose Application Key. Again we stick to the naming convention - so for me that means `3AHMBA_PETERSCHOFSKY_Glass_Ak`.



In the project assignment you can click on the "+" symbol, then a list opens with all projects available on this Thingworx server. Here we select of course the project, which we created before (I here thus `3AHMBA_PETERSCHOFSKY_Glass_Pr`). Thus this Application Key is assigned to the project. Now we have to assign a user.

In principle, the user determines what the application key is allowed to do. If the assigned user is not allowed to do anything, you cannot do anything with the application key. If the assigned user is allowed to do everything, you can do everything with the application key. Because we don't have another user, we simply use "Administrator" - this should not cause any access problems! However, we get a warning from Thingworx that this is not a good idea. Well, that's true, of course. For our little learning application, we accept it anyway and close the warning with "Yes".

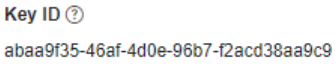




It is important to specify an expiration date. By default, this is set to one week - but we want our application key to work longer, so we give ourselves a date in the distant future. Somewhere in the summer holidays should be enough[50] .

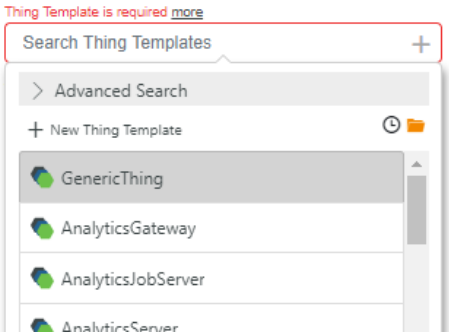
It is a popular mistake to work with an expired application key. Suddenly you always get Not Authorized responses. Just like the motto: Yesterday it still worked, but I didn't do anything.

With this we are done and can press "Save". After the successful save there is suddenly a Key-ID. This is the application key. If this is sent along in the requests, you identify yourself as the "knower" and the data is written.



3.5.2.3 *The Thing*

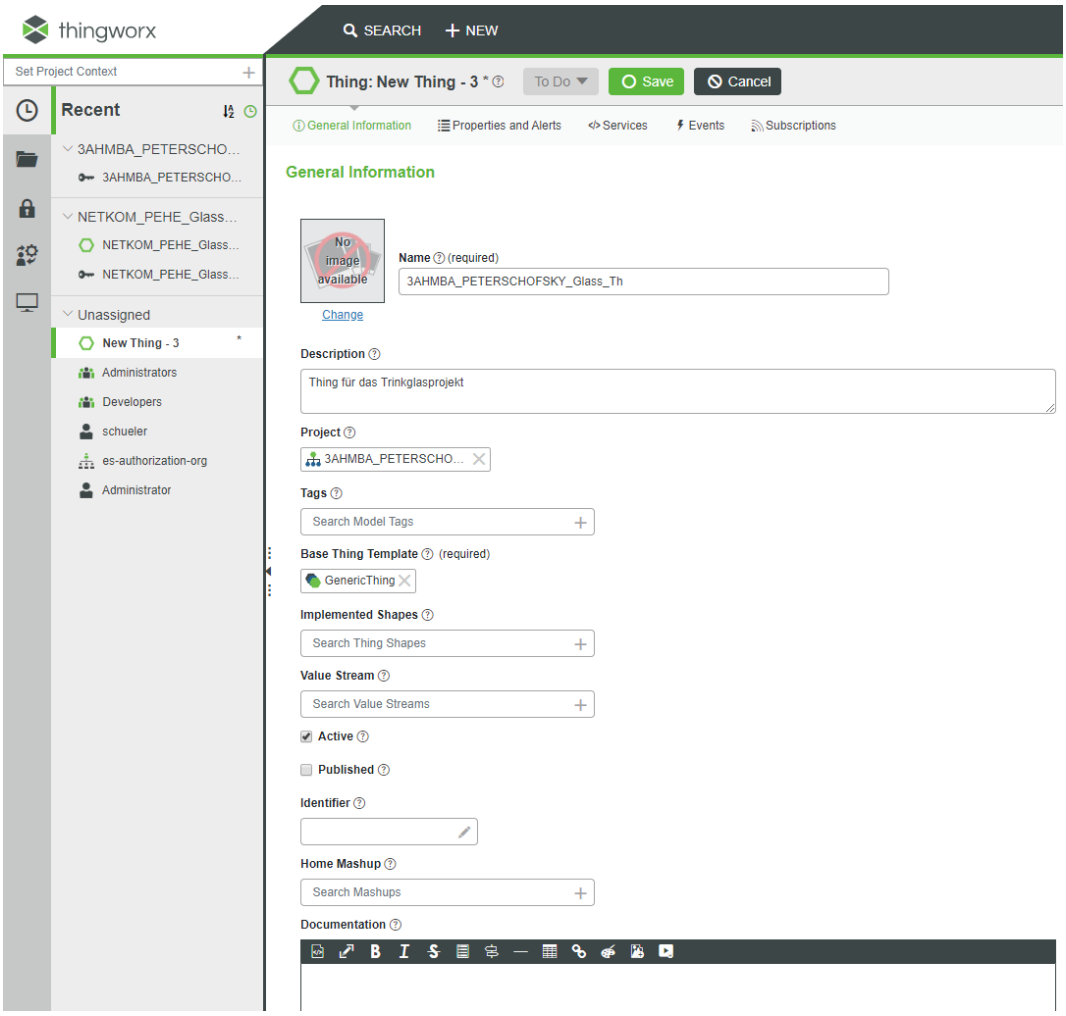
Now, finally, we can create our Thing. Finally we get to where we actually store our data. Now, we want to make a drinking glass. And we want to create the following things: Fill Level, Contents, and Temperature. Creating a new Thing starts the same way as before. Only this time, we're going to select Thing.



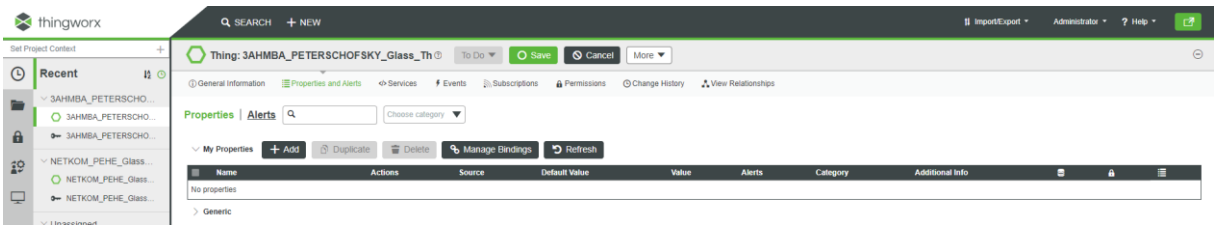
Again we have to assign a name (here `3AHMBA_PETERSCHOFSKY_Glass_Th`) which corresponds to the convention. As project we choose again our project. We still have to choose a "Template". We choose "`Generic Thing`" from the list.

[50] The Thingworx server will be rebuilt in the new school year anyway. So longer is useless.

With that, our Thing definition looks something like the picture and we can press "Save":

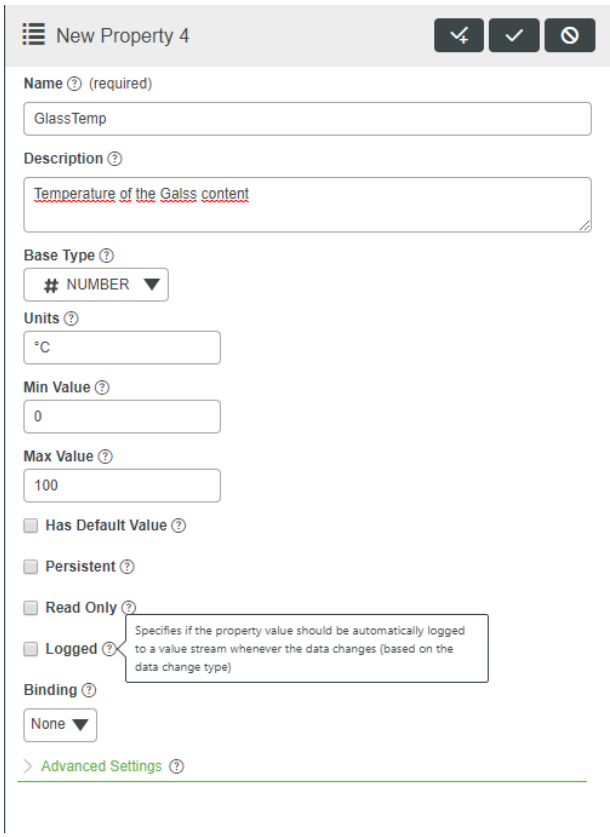


The Thing appears on the left in the project tree, it has a name, is assigned to the correct project, but is otherwise empty. We now want to add our data. These are called properties here. We click on "Properties and Alerts" at the top and see an empty list.

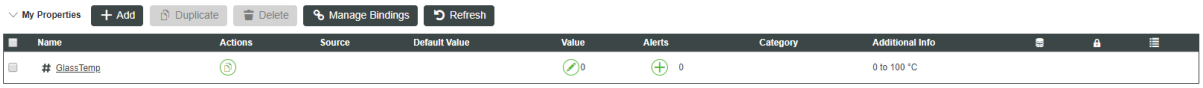


To add a property, we press "`+Add`". On the right side a frame is inserted. There you can now create a new property. First we take care of the temperature and therefore enter "`GlassTemp`" as name. As Description we choose "`Temperature of the Glass content`" and as Base Type we choose "`Number`". Then we can set Units with "`°C`", Min Value with "`0`" and Max Value with "`100`". Furthermore we select "`Persistent`" and "`Logged`".

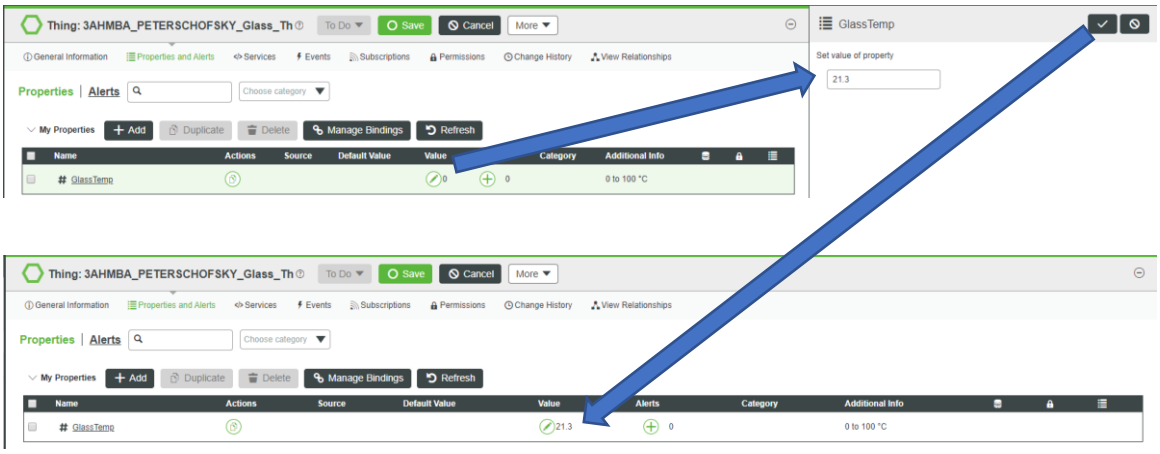
So our new property should look something like this:



With a click on the hook[51] in the top center we save the property. It appears in the list. Now we have to save the Thing again with "Save". Now the property also has a value, namely 0.



With the pencil symbol next to the value you can change it. Here we change the value to $21{,}3°C$. Pay attention to the comma notation - it's a " . " And not a " , "!



Does that work? Then it succeeded - we have created our first Thing. With an associated project and a corresponding permission.

[51] The hook with the plus would immediately add the next property.

3.5.2.4 Exercise "Simple

	Create your Thing as described above.	

3.5.2.5 Mandatory task "Extedned Thing"

	Add the properties for the fill level and the fill volume. Think about suitable limits and show your Thing.	

3.5.3 Writing/reading data to the Thing

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

Writing data to our Thing is done by calling an associated service. This service is called "`UpdatePropertyValues`"[52] . There are several ways to call this service. It has been shown that the method of sending a special http request works relatively reliably. This request looks like this (All values between <...> have to be adjusted):

```
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> "[,
...]]
}
```

Adapted to the example from the previous chapter o the corresponding request would be

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

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

[52] You can also write your own service. This would make it possible not only to write values into the properties, but also to convert them (e.g.: temperature from °C to °F or vice versa) or to fill several properties at the same time (e.g. calculate the fill level here and the fill volume from it and write both at the same time). However, we do not want to do that in this context.

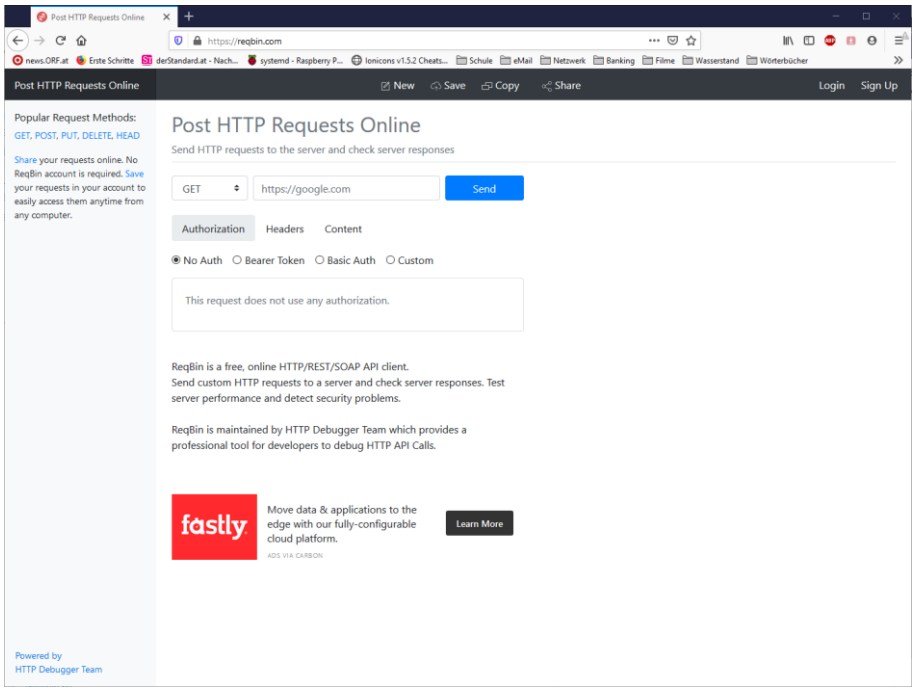
The top request would give the property "`GlassTemp`" the value "`25.1`". The corresponding response from the server would be:

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

It all looks wild, but what is important is the following text: `HTTP/1.1 200`

This means Response Code `200`, which also means `OK`. Such a code is a so-called http status code. A list of status codes can be found in the appendix. Relatively often we will still fight with the status code `401`. This means `Not Authorized`. Probably there is no, or an expired or a wrong application key.

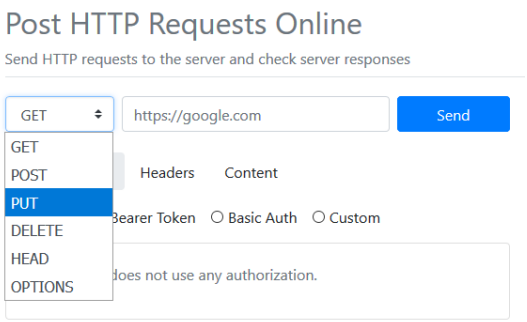
Now we are only faced with the challenge of how to send this http request away. Well, there are a lot of tools for sending and testing http requests. You can use whatever you want. If you have data concerns, you should use a program that is installed locally on your computer[53] . Since it doesn't matter here and we don't have to deal with sensitive data we use an online tool. I use https://reqbin.com/ here.

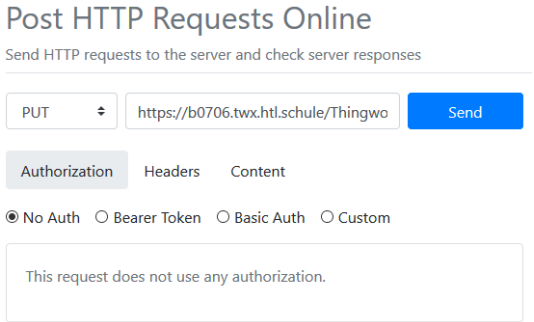


[53] Examples would be Postman or Putty.

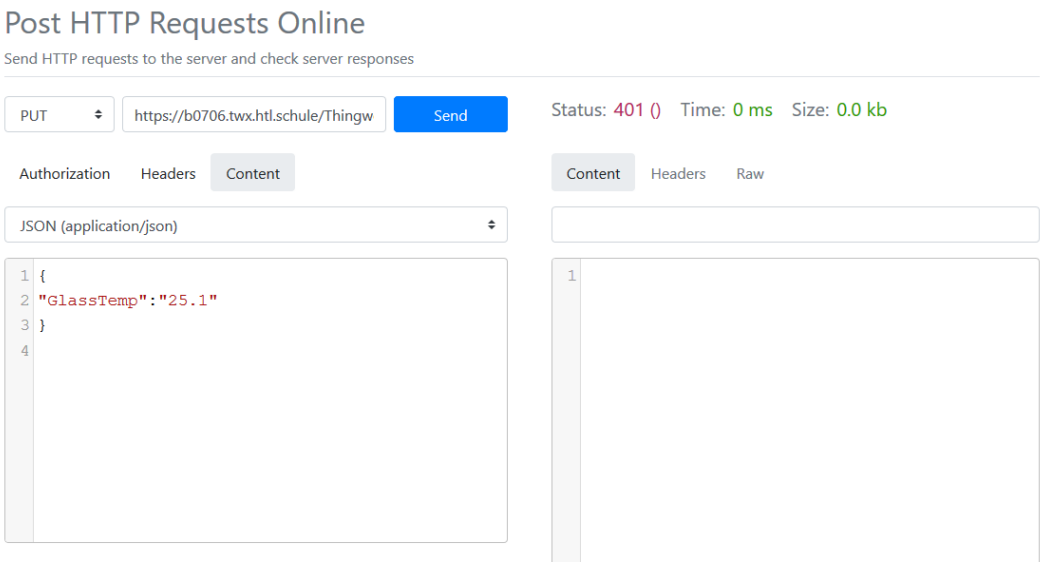
Here, we can just put together an appropriate http request. Well, first we need a PUT request, so let's set this up. Also, we want to send an `https request` to the server `example.twx.htl.schule` to the URL `/Thingworx/Things/3AHMBA_PETERSCHOFSKY_Glass_Th/Properties/*`. We can easily write that in the input field next to it:

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





The content should be a new value for our properties. We had the property "`GlassTemp`" and we want to provide this with a new value. Let's say the temperature should have increased to $25{,}1°C$ increased. So we switch to the Content button and enter the appropriate information there (the correct type `JSON` should already be set):



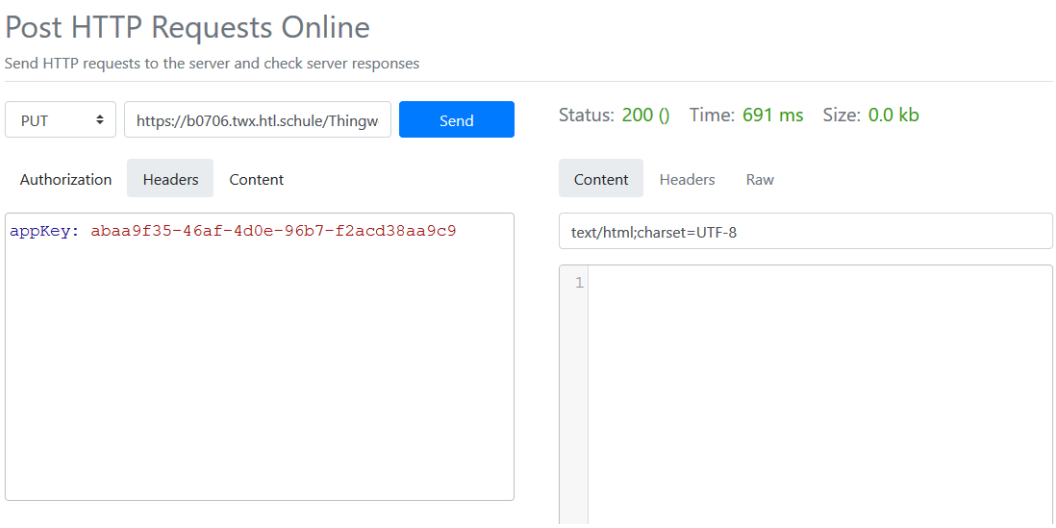
If we send the request away like this, we should receive a response from the server, but it contains the code `401`. The good news: The server is reachable and talks to us. The bad news: It won't let us change anything.

This is not surprising, because we have made an effort to create an application key and we have not yet used it in our http request. So we will add this key. To do this, we switch to the "Headers" section and add the line

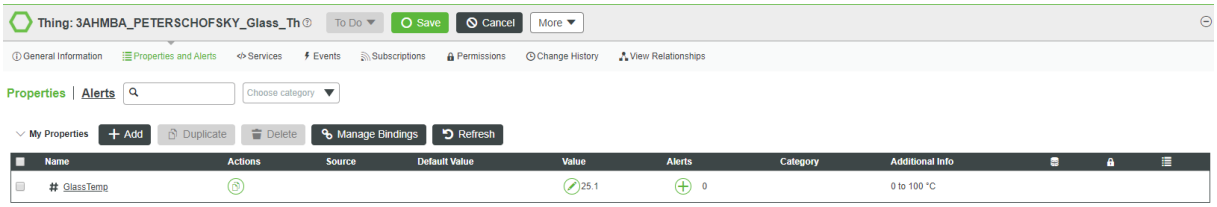
```
appKey: abaa9f35-46af-4d0e-96b7-f2acd38aa9c9
```

added. ATTENTION: Everybody has his own App-Key! Therefore use the CORRECT number!

If we send this away now we should actually get a `200` status code. The server has accepted our request!



A look into the corresponding property in Thingworx makes us sure: The value has indeed changed to the specified 25,1°C (you may have to click on Refresh). So it works!



It doesn't matter from which device this http request comes! If we manage to build a device that is able to send http requests, we bring the data from the device to our Thingworx server. We then have a meter that transmits its values to the cloud and makes them available. It just needs to use the right application key and things are up and running. If the device transmits cyclically, the current measured value is always available online and can be used anywhere in the world.

Now how do we retrieve the data from the database again? Well, that too can be done with an http request. This should have the following form:

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

Very simple, isn't it? Let's try to adapt the request to our case:

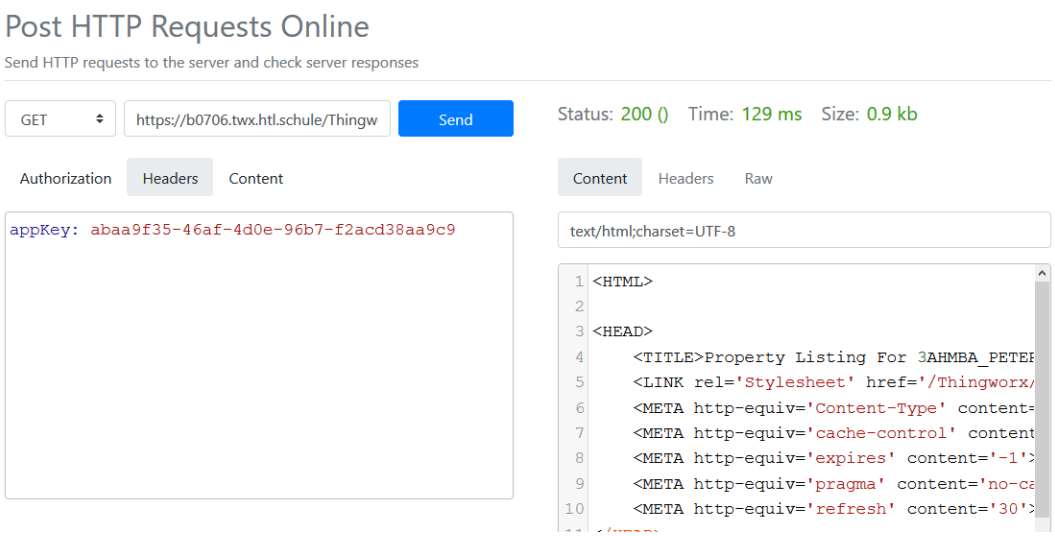
```
GET /Thingworx/Things/3AHMBA_PETERSCHOFSKY_Glass_Th/Properties/ HTTP/1.1
Host: example.twx.htl.schule
appKey: abaa9f35-46af-4d0e-96b7-f2acd38aa9c9
```

To execute this request, we again use https://reqbin.com/.

So we set GET and use the following line:

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

In the header we have to specify the appKey again. The content can remain empty. If we send this request we get a positive response, only this time there is also something in the content:



With me is this answer:

```
<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 the drinking glass project</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>
```

Does anyone understand this? Well, at least our property "`GlassTemp`" with the current value `25.1` occurs in it - so it actually seemed to contain the data somehow. The whole thing around it is

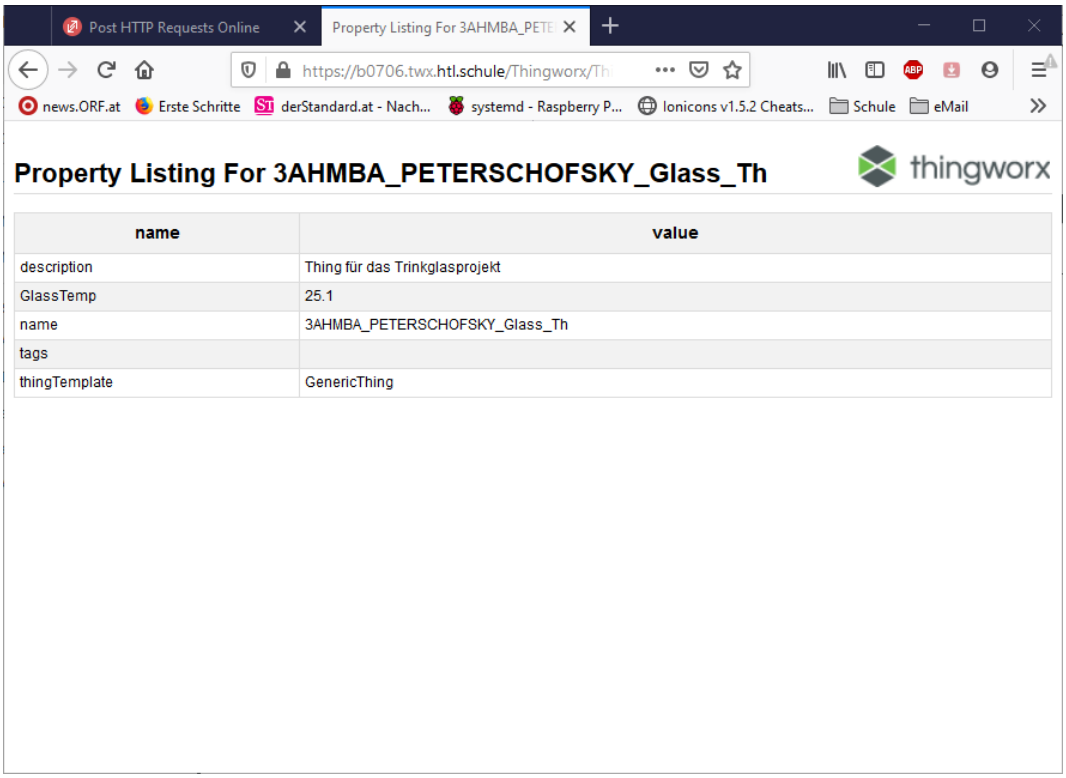
not just unnecessary stuff, this is the description language HTML[54] . This language is understood by all web browsers - this is a webpage in textfrom!

So what could be more obvious than the address

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

right into the web browser?

If we do this, the browser prompts us to enter the access data (again, we have no appKey from the browser). If we enter them, a web page appears that displays the information much more clearly.



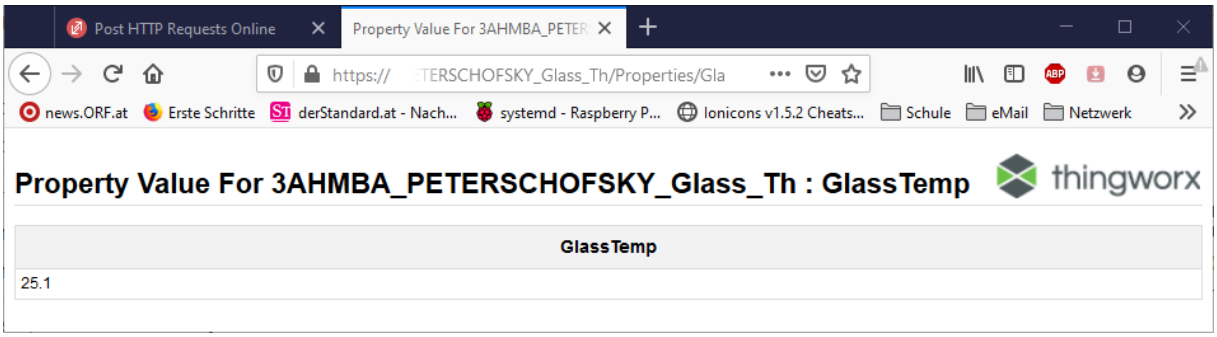
name	value
description	Thing für das Trinkglasprojekt
GlassTemp	25.1
name	3AHMBA_PETERSCHOFSKY_Glass_Th
tags	
thingTemplate	GenericThing

If several properties are available, they are all listed. If you want to access exactly one property you can also use e.g. the line

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

use.

[54] Hyper Text Markup Language



3.5.3.1 *Exercise "Change*

	Create an http request to change the data in your Thing as described above.	

3.5.3.2 *Pflicht task "Change Data*

	Expand the request for the properties of the fill level and the fill volume!	

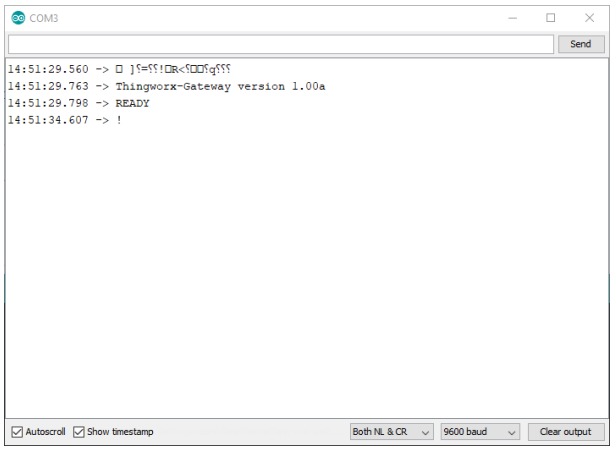
3.5.4 Connection between Arduino and Thingworx

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

In principle, we have the same problem here as we did when connecting with MQTT. This was discussed in chapter 0 dealt with. Well, here we have practically the same "recipe": A program on our ESP8266 board and we talk to it via serial interface.

This time we use the program `ThingWorker.ino` and play it on our ESP. The procedure is identical to the one in chapter 0 but with the other program.

The commands are again relatively similar. Again we can check the connection after the reset by entering a "?" and also here we get "!" back - it works!



We now want to connect to our Thingworx server "manually" again. To do this, we first switch on all possible outputs from the ESP. We type into the Serial Monitor one after the other the commands

```
startwarn
startinfo
startdebug
```

in the Serial Monitor.

If we use `getverbose` now, we get the output for confirmation:

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

The "X" indicates that the output for errors, warnings, information and debug will be printed. Each command is now acknowledged with a lot of feedback.

Let's first try to connect to the school's WLAN. For this we use the commands:

```
setssid WifISSID
```

Feedback:

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

```
setpass WfiPassword
```

Feedback:

```
Set WifiPass: 'WfiPassword'
OK
```

As soon as an SSID and a password have been transmitted, the ESP attempts to establish a connection with the specified network. If a connection is not established immediately, the controller tries again and again. If the SSID or the password is wrong, it is sufficient to correct the value with the corresponding command. The output should look similar to this:

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

If the connection is successful, the WLAN LED should go out. This has energy-saving reasons. When operating with e.g. a battery, it is unnecessary for any LEDs to light up here in the normal state.

So the first step is done once again - we are connected to the WLAN. Now we continue with the Thingworx server. We need to send the host address, and then the application key and the Thing. So we start and enter one after the other:

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

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

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

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

```
setthing 3AHMBA_PETERSCHOFSKY_Glass_Th
```

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

With this we have defined Server, Application Key and Thing. Let's move on to the properties.

There, just as with the MQTT variant, there are again the three variants:

- Value-Property: Numeric value, a transmission threshold can also be specified here.
- Boolean property: truth value
- General Property: Can contain any value.

I want to add our glass temperature as a numerical value right away and therefore enter:

```
Setvaluename 0 GlassTemp
```

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

Followed by the transmission threshold:

```
setvalth 0 0.25
```

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

With this we have determined that we want to write a value into our Thing `3AHMBA_PETERSCHOFSKY_Glass_Th` with the property `GlassTemp`. Nothing happens yet, only when we use

```
setval 0 25.3
```

put a value suddenly start spending more. The feedback is clear:

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

Thereafter, the output follows at regular intervals:

```
{
"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
```

First we see the JSON[55] block transmitted to the server and then the response from the server. Here we see the response code: `200`. That's good, the server understood our value. This time we don't have to assemble an http request. The ESP does that for us in the appropriate form. We just have to say what we want to transfer.

Whenever the transfer time has expired, simply ALL the properties entered are transferred. Just like with the MQTT part we can enter times for the retransmission. All commands of the ESP-Gateway for Thingworx are available in the appendix (see chapter 4.5.2).

3.5.4.1 Exercise "Change Data ESP

	Create the program for the ESP gateway for changing the data in your Thing as described above.	

3.5.4.2 Change all Data ESP" piling task

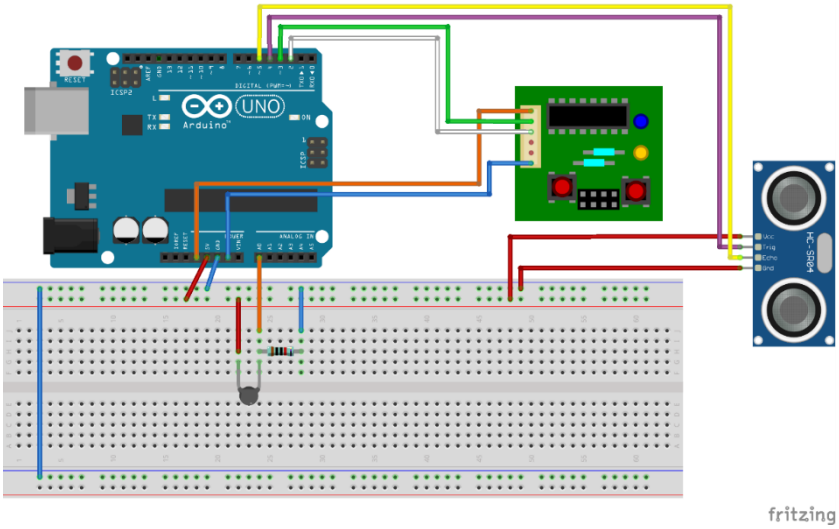
	Extend the program for the extended properties of the fill level and the fill volume!	

[55] JSON: JavaScript Object Notation, a data format that contains information in a special form.

3.5.5 Providing the thing with live data

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

As with the MQTT topic, it's nice to be able to change the values with command line parameters, but if they come from real measurements, the whole thing works a lot better. So we want to measure a level and a temperature and then transmit the values to the Thingworx server. Now, there is a library for our Arduino again. This is called "`EspThingworxGateway`", analogous to the MQTT library. So again, we use a zip file to install the library. This also needs the library `Timeout.h`[56].



With the libraries installed and the following hardware setup, we can get to programming things.

So we use the NTC to measure the temperature and the ultrasonic module to detect a distance. The program looks accordingly:

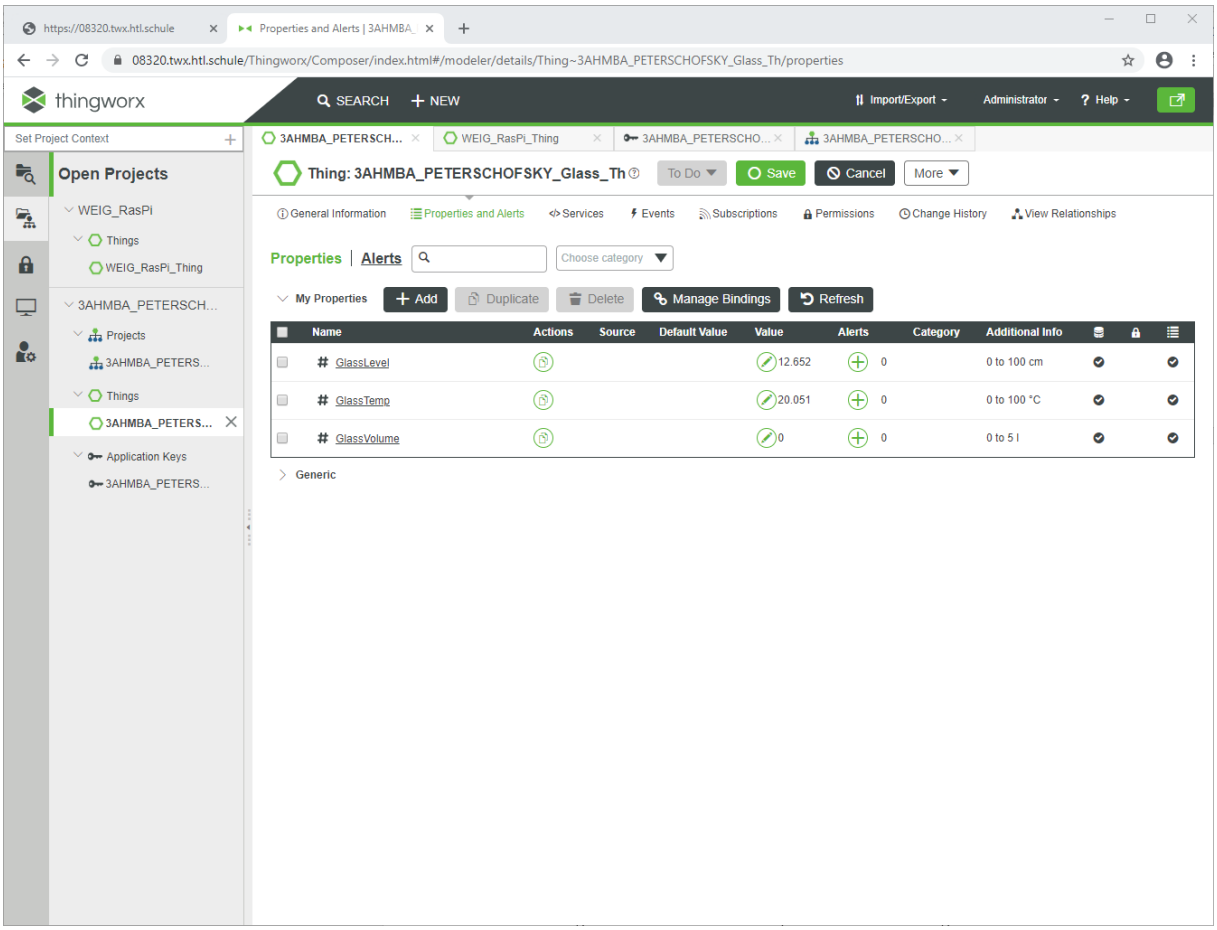
```
#include "SoftwareSerial.h"
#include "EspThingworxGateway.h"
#include "Timeout.h"

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

String SSID = "WifISSID";
String PASS = "WfiPassword";
String TwxHost = "69b34.twx.htl.school";
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);
}
```

[56] Why and why is also described in the chapter 0 described.

```
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();
}
```

Most of the things we already know. It is practically a mixture of the chapters 1.2.16 The NTC as temperature sensor and 1.2.18 Distance sensor. Especially the distance sensor is a bit extended, because we don't calculate the duration, but really the distance. Also, let's assume we are measuring from the top of the vessel. So the smaller the distance becomes, the fuller our vessel is. The whole thing is enriched with the transfer to the Thingworx server. In principle, the whole thing should be self-explanatory. It works very similar to chapter 3.4.2 - even a little bit easier, because we don't have a return channel and don't receive any data from the Thingworx platform. A list of all methods for the `EspThingworxGateway` class can be found in the appendix.

When we open our Thing in the Thingworx composer, we see that we always have new, fresh values available. So the measurement runs and transfers the values to the properties of the selected Thing.



3.5.5.1 *Exercise "Automatic Data Transfer*

	Create the program with the measurements as described above.	

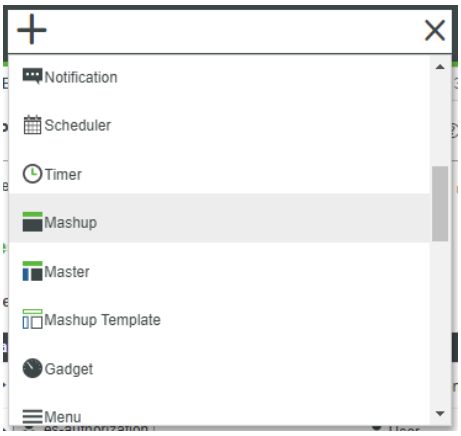
3.5.5.2 *Pflicht task "Automatic Volume Transfer*

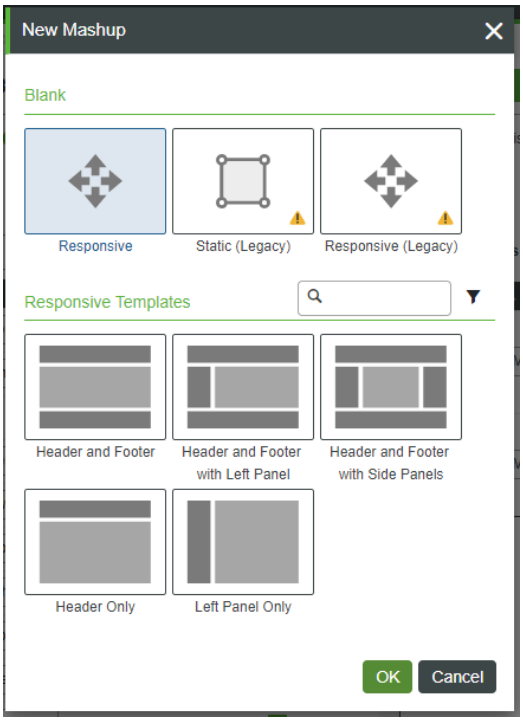
	Extend the program by the property of the filling volume. Think about how you can calculate this and transfer the value sensibly.	

3.5.6 Displaying data in a mashup

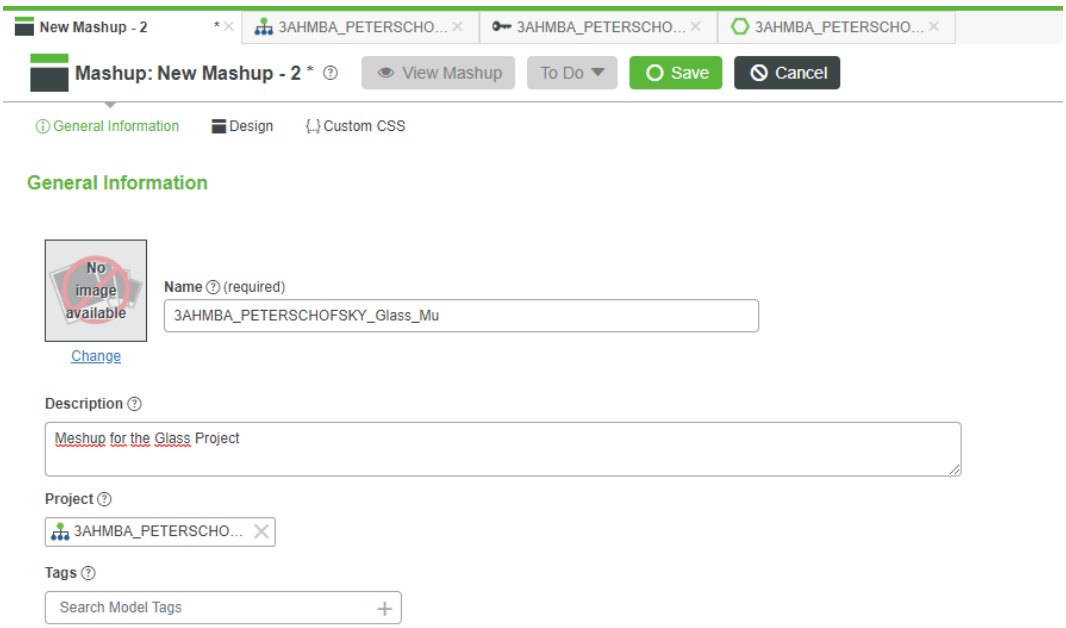
Video Link 11: https://youtu.be/DZINXvExyhk

Whenever we want to look at the Thing's data, we either have to open the Thing's page in the Thingworx composer, or enter the corresponding URL. A collected clear view would be helpful. And there is exactly such a thing. This is called a "mashup" in Thinworx. As usual, we can add such a mashup to our project. Again we try this with the + icon in the top bar and this time we select "Mashup".

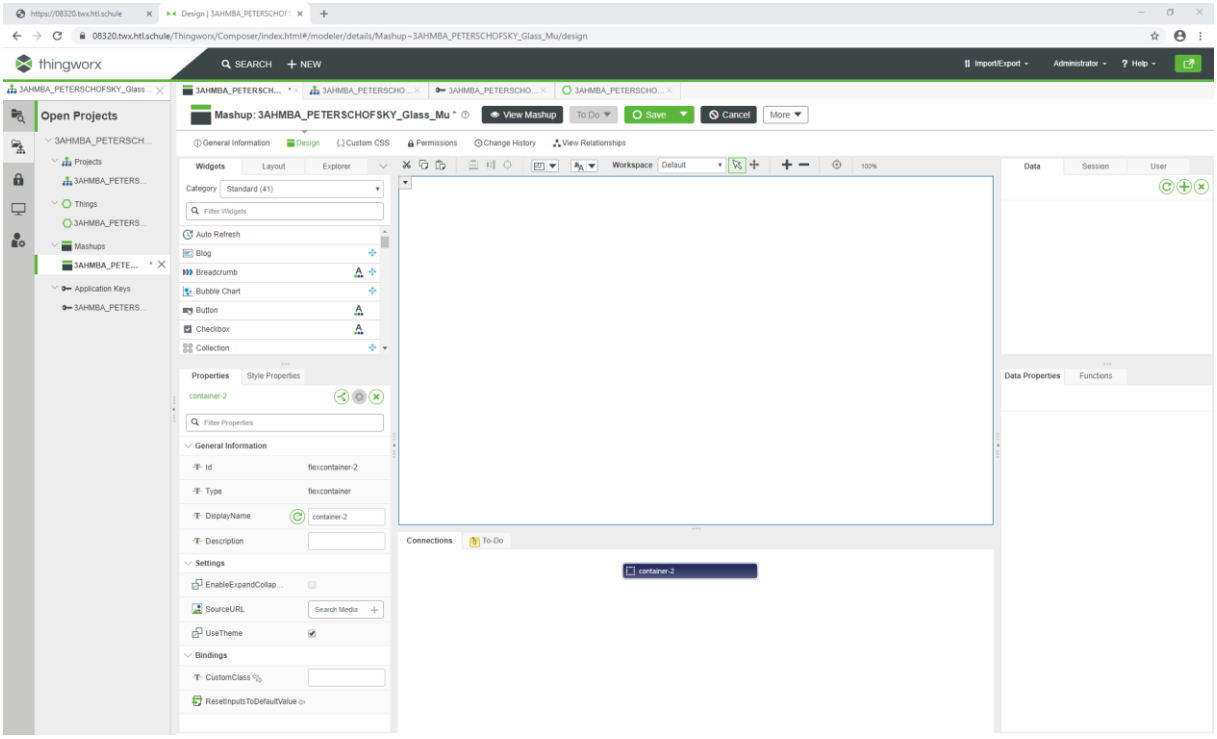




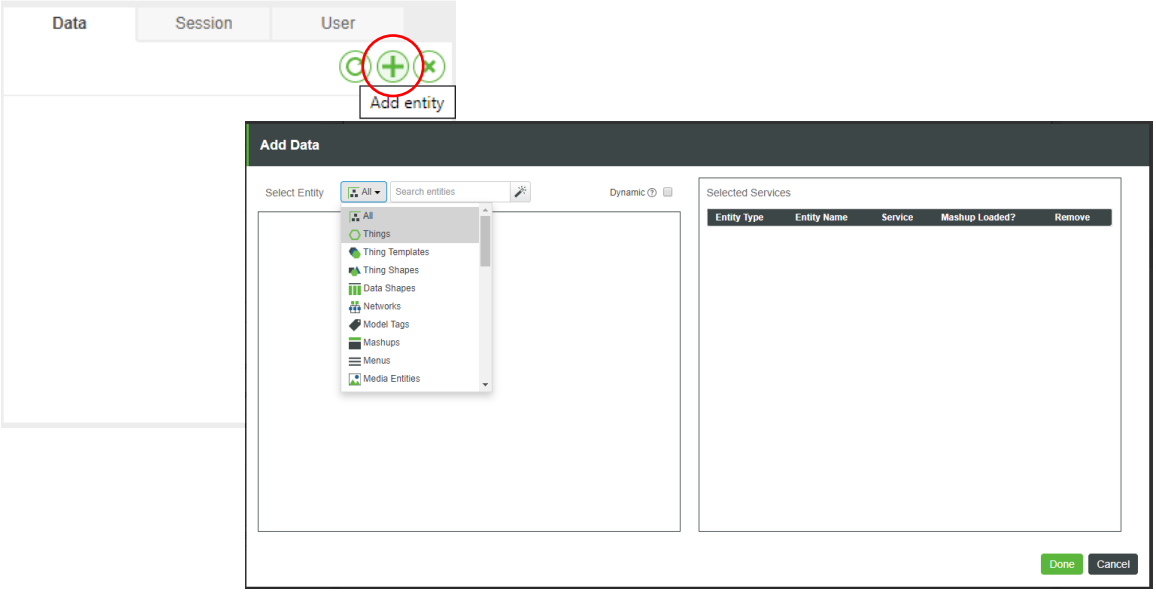
From the menu that appears, we select "Responsive" and press OK. We will enter the appropriate data according to the naming convention. So I choose for the name of the mashup `3AHMBA_PETERSCHOFSKY_Glass_Mu`.



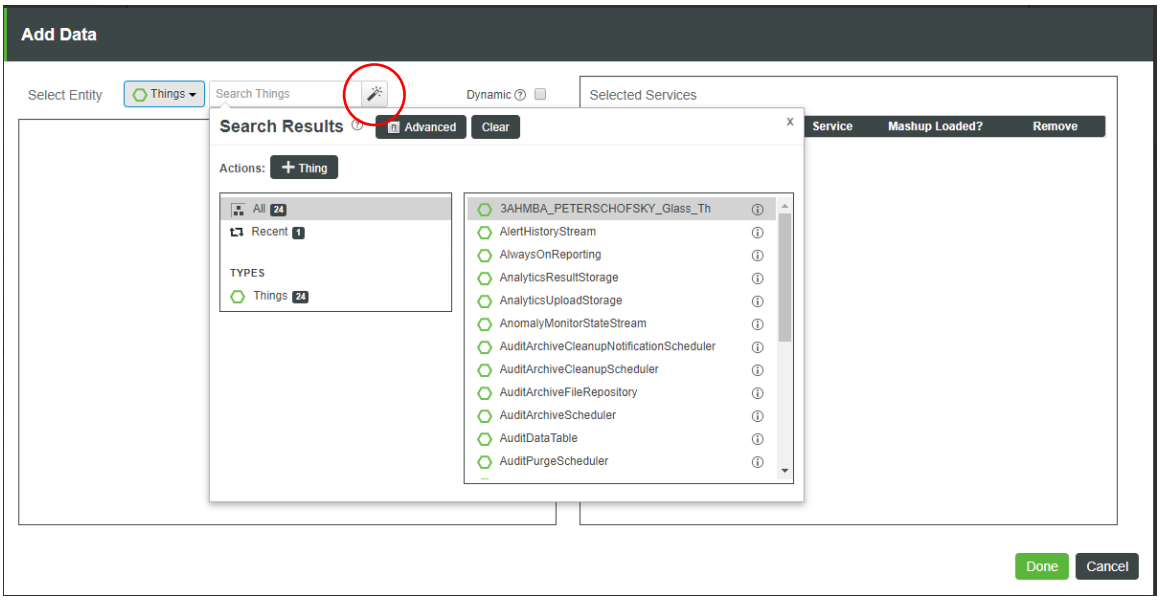
If we click "Save" afterwards, the mashup is created. If we switch to "Design" afterwards, we get the interface to create a mashup.



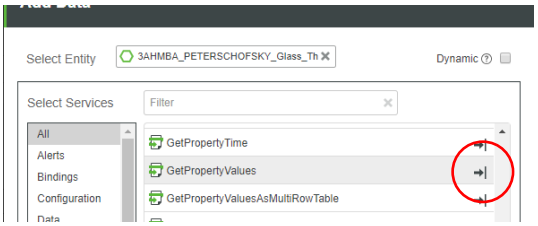
First we need to take care of the "connection" to the right Thing. We want to load data from a Thing. This is done via a so-called "Service". On the far right we have an area with three tabs (Data, Session and User). There we select Data and then click on the + symbol.

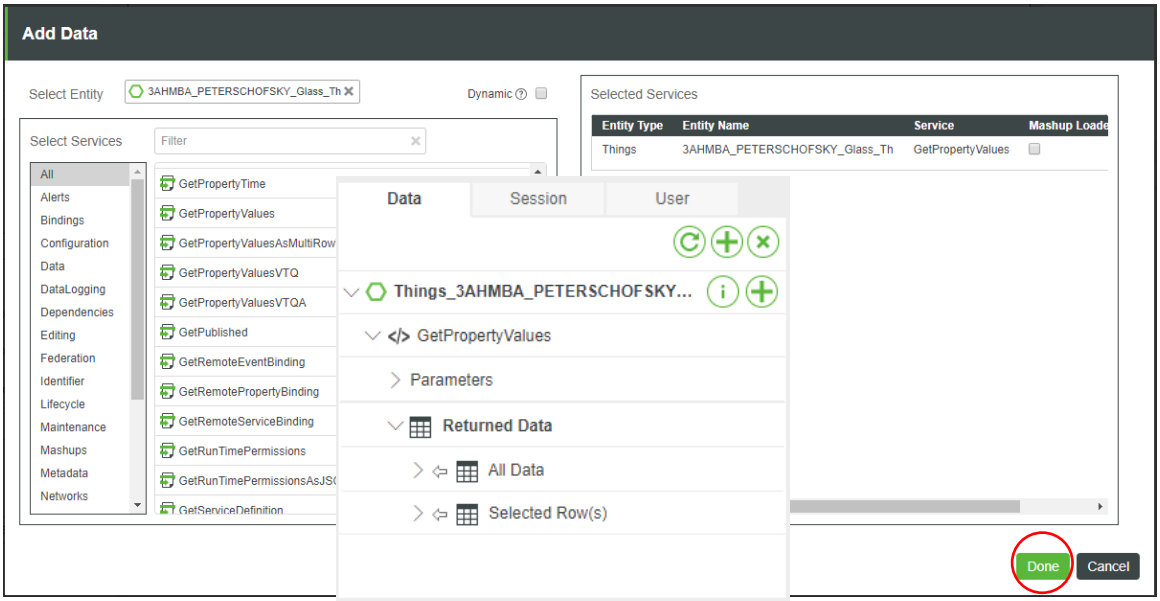


After that, a dialog overlay opens. There we first select "Things" from the dropdown menu and press the magic wand. This will show us a list of all possible Things on our server. Thanks to the naming convention, it should be easy to find our Thing. We press on the selected Thing and we get a list of all possible services that this Thing offers.



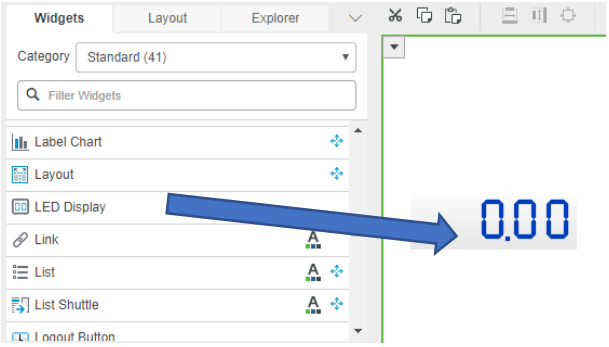
Well, we are interested in the properties and their data. The correct service for this is called "`GetPropertyValues`". We look for it in the list and press the arrow symbol. This will display the service in the right pane. This means that this service has been selected. We are satisfied with this for the time being and select "Done".

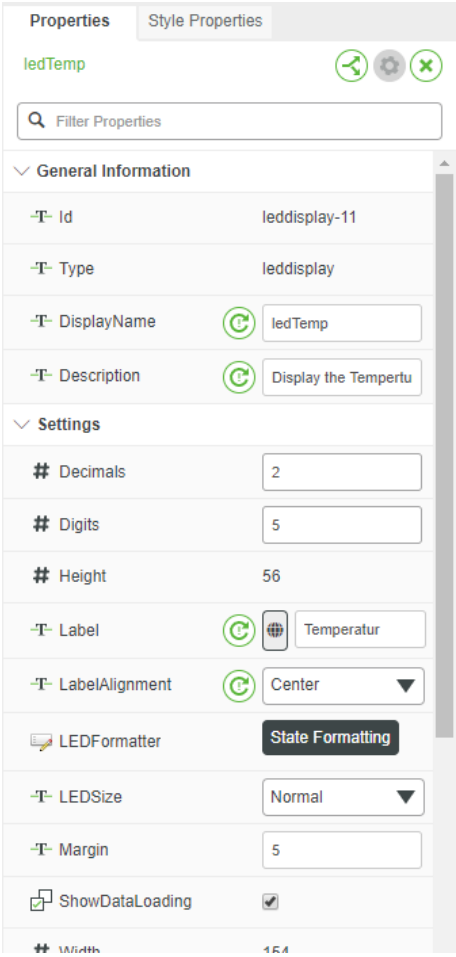




The data area in the main window has changed. There we now see the corresponding services. We have only selected one, so only one appears there. By this we mean we can read the data from the properties. Let's try that now on a trial basis by displaying the data. One element to display numerical values is the widget "LED Display".

To get this into our display, we simply drag it from the Widgets area into the Workspace (exactly the same way it works in Vuforia Studio.

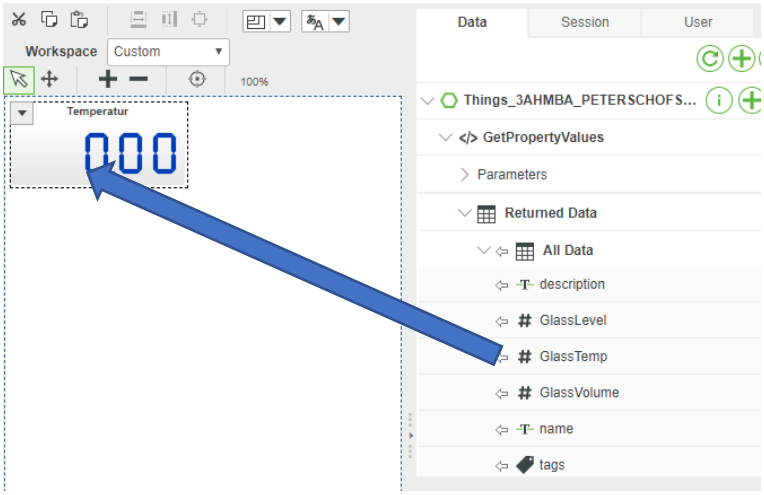


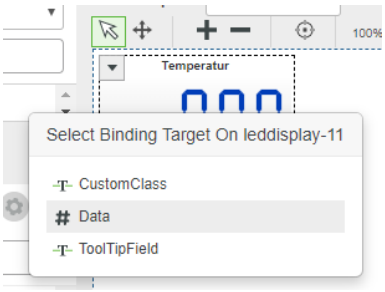


As a test we want to try to show the temperature in this display in the lower left area at the properties we therefore fill in a few things that we keep the overview.

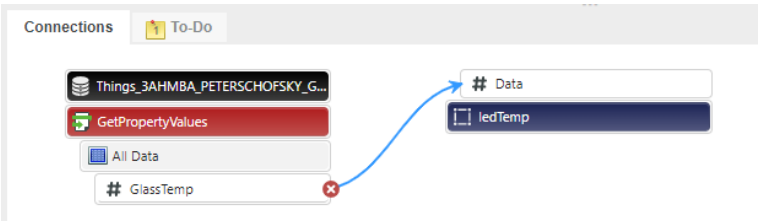
As DispalyName I use here "`ledTemp`" and choose a meaningful Description. I also set a label and write "`temperature`" there. This gives the LED display a text above the number. I like it when the label is in the middle, which is why I choose "`Center`" for LabelAlignment.

With it the representation of my LED display is sufficient for me and I can begin to select the value, which is to be represented. If you "open" the tree in the data area on the right and then find the value `GlassTemp in` the structure under All Data, you can drag it to the LED display. After that a selection box appears where you can choose which property of the LED display we want to bind to the value - there we choose of course "`Data`".

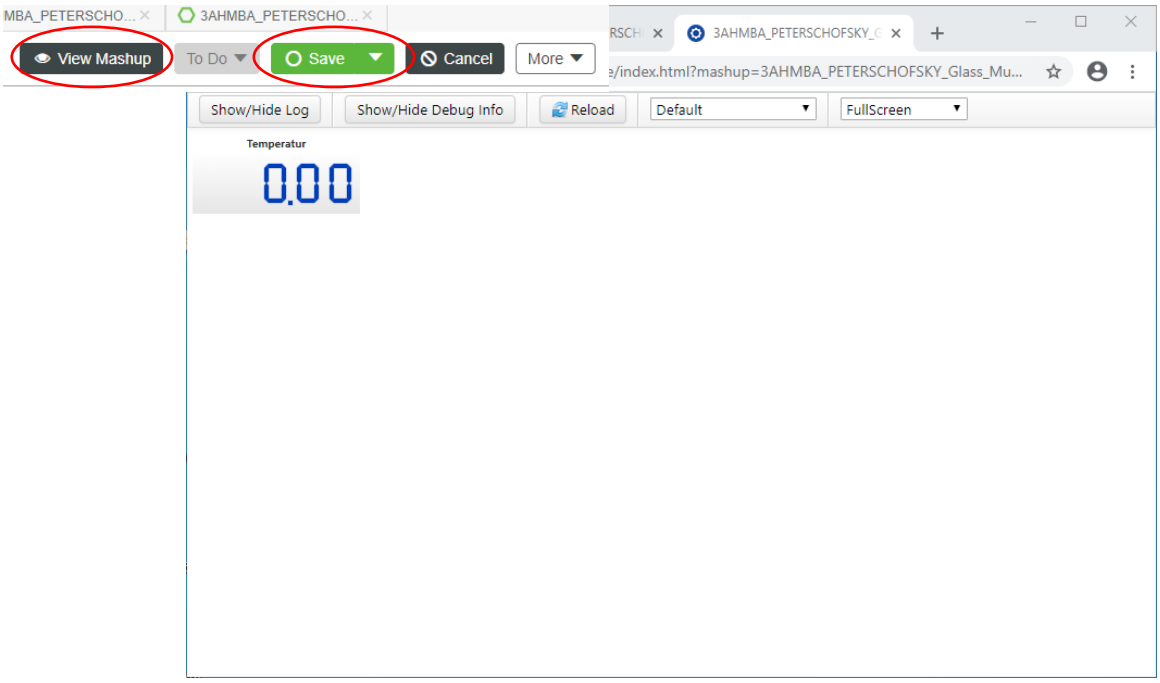




In the lower area you can also immediately see the connection. The current element is always displayed in the middle. The data sources come from the left, the outputs go to the right. With the red x-symbol you can delete the connection. Of course we don't want to do that now.



To view our work we can first press Save at the top and then View Mashup.



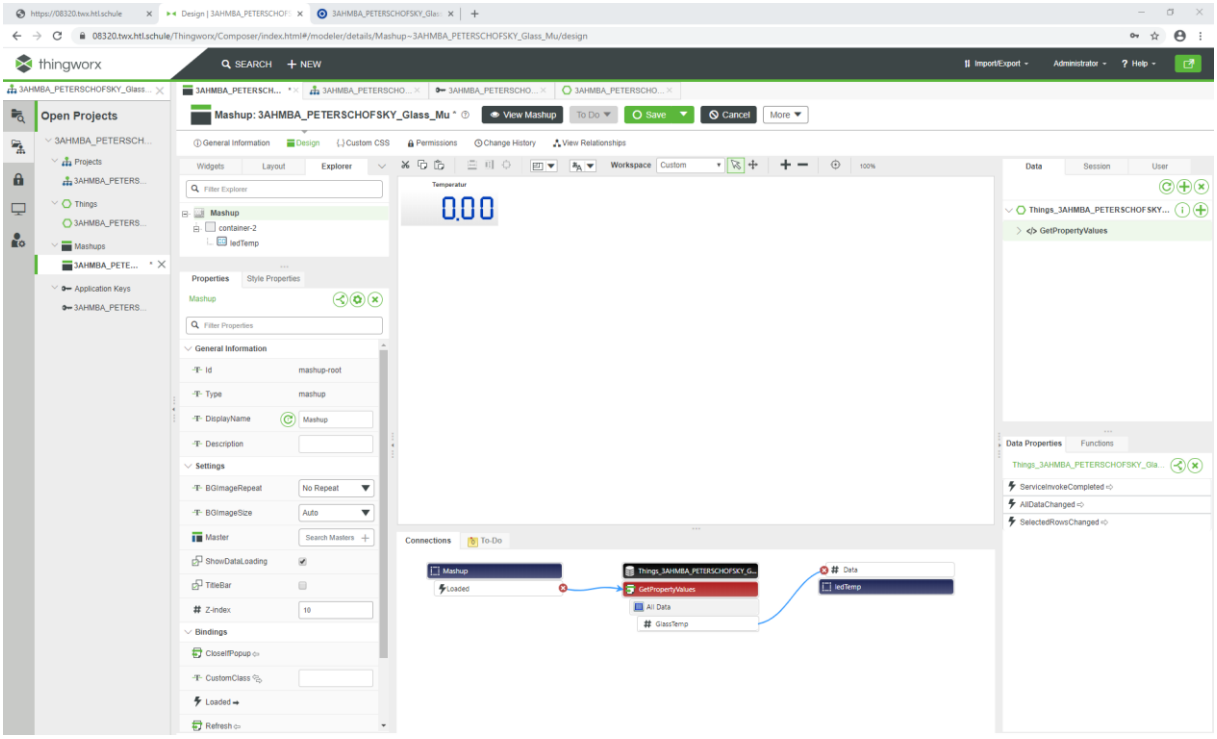
The result is sobering. It always says 0,00 inside, although in the Thing quite clearly other values are present. Somehow the data do not find here properly.

The reason is that the service has been created but is never called. We can change that. We want to load the data when the mashup was called. To do this, we select "Explorer" in the left pane and select the mashup there. Then, just below that, the properties of the mashup appeared. One property is "`Loaded`". We bind this with the service `GetPropertyValue`.

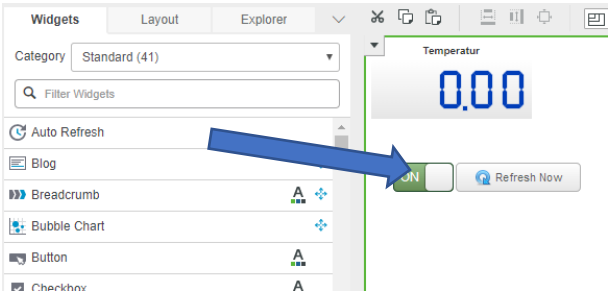
The procedure is again identical: we drag the small arrow next to Loaded to the service and release there. Immediately, the connections are updated in the lower area and we recognize the causality chain:

- As soon as the mashup is loaded the event `Loaded` is triggered.
- The `Loaded` event triggers the `GetPropertyValue` service. This means that new data is retrieved.
- This new data finds its way to the `Data` property of the LED display.

I like the sound of that.

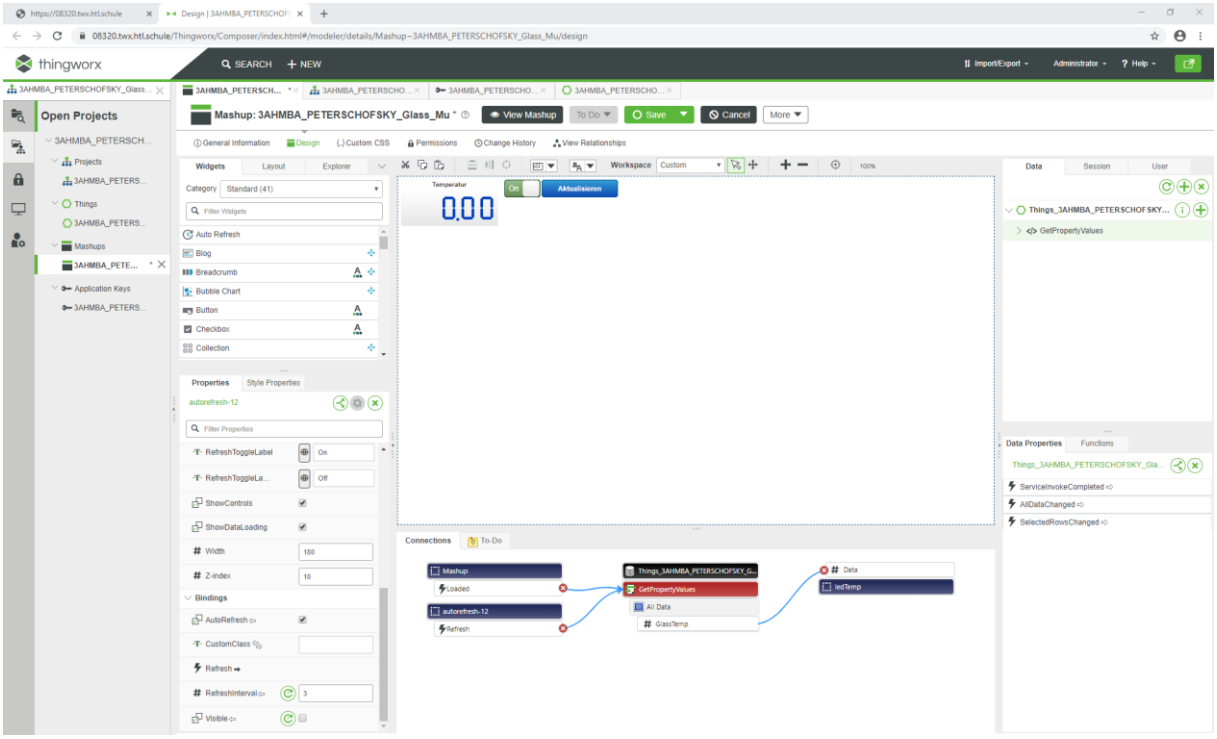


So we try it out. And indeed, there is now the temperature value. We only have one problem: I can't change the data. Only if we press Reload, we get a new temperature value. Live data looks different. For this we need an automatic update. And the widget we need is called "Auto Refresh". We drag this into the workspace.



What we get is an element with two controls: A slider button to toggle the automatic update on and off, and a button to trigger a manual update. In the properties of the widget you can set various things. I have changed the following things:

- Label: here I use "`Refresh`"
- RefreshInterval: Here I use 3 - I want to display the current value every 3 seconds.
- Visible: I'll chop that away. I don't really want the element to be seen on the surface. I want that to happen in the background.
- Refresh: This is the event and I draw this again to the `GetPropertyValues` service. So now whenever the Refresh event is triggered (either automatically by the RefreshInterval or with a button press) new data is loaded.



Let's try it: Indeed, the value changes. If we touch the temperature sensor, the temperature rises and the display changes. That's right, that's how it should be.

3.5.6.1 Exercise "Basic Data Display

	Create a mashup for the temperature as described above.	

3.5.6.2 Basic Data Display" piling task

	Extend the display by the properties for the filling level and the filling volume!	

3.5.6.3 Additional task "Beautiful Mashup

	Don't just try numeric displays. There are things like the "Gauge" and "Label Chart" widgets. Create a mashup that includes these things.	

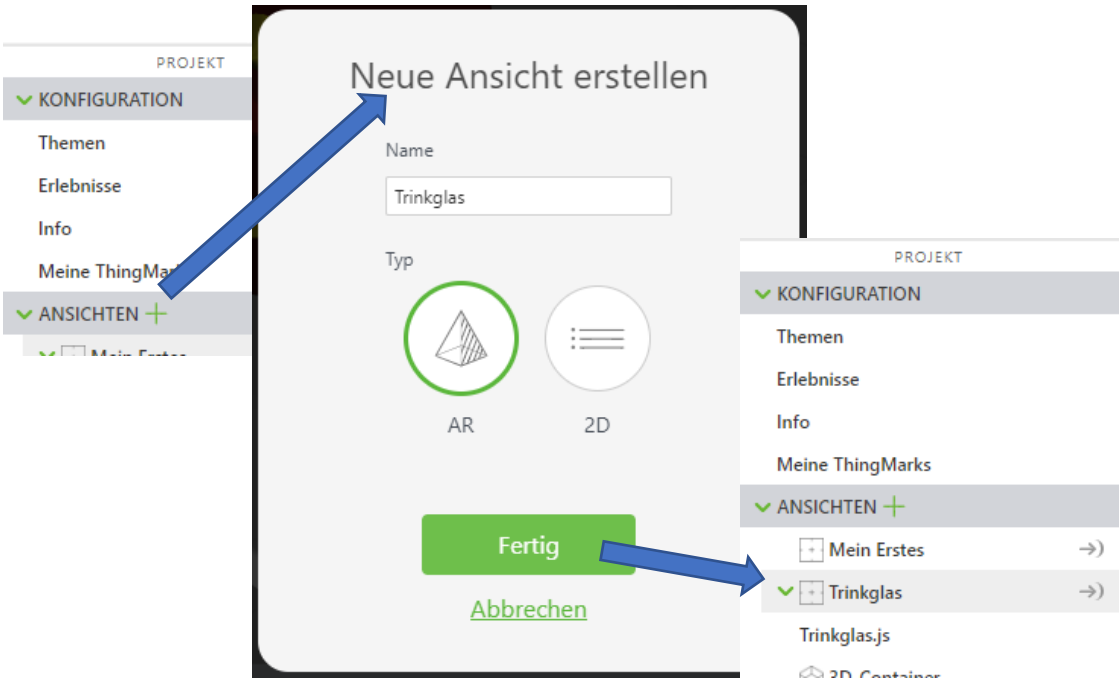
3.5.7 Using Things in Vuforia Experiences

3.5.7.1 Representation of a measured value

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

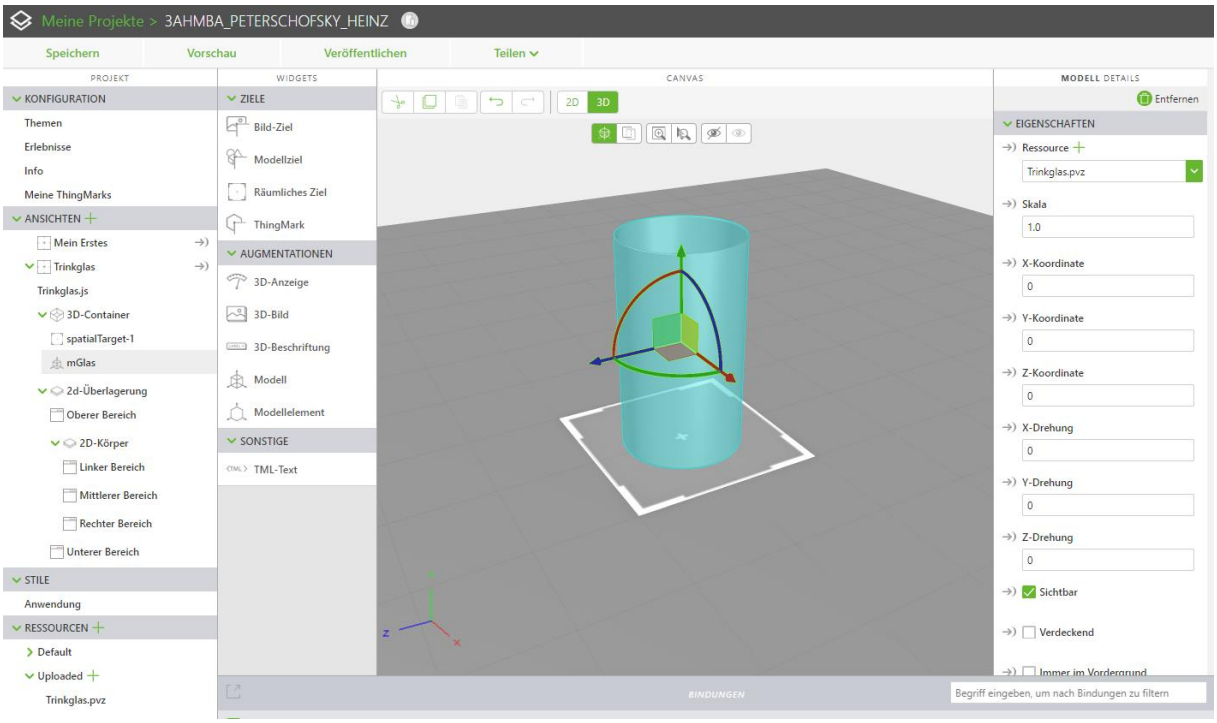
Sometimes it is desirable to show actual values in the augmented world. It is possible to use data from Thingworx in Vuforia View. Fortunately, we don't need http requests for this, because there is a built-in interface that is relatively easy to use. Let's look at it again with an example.

We take our project and add a new view again. We call this one "Drinking Glass" and select "AR". After that, the view should have been generated and we can get started.

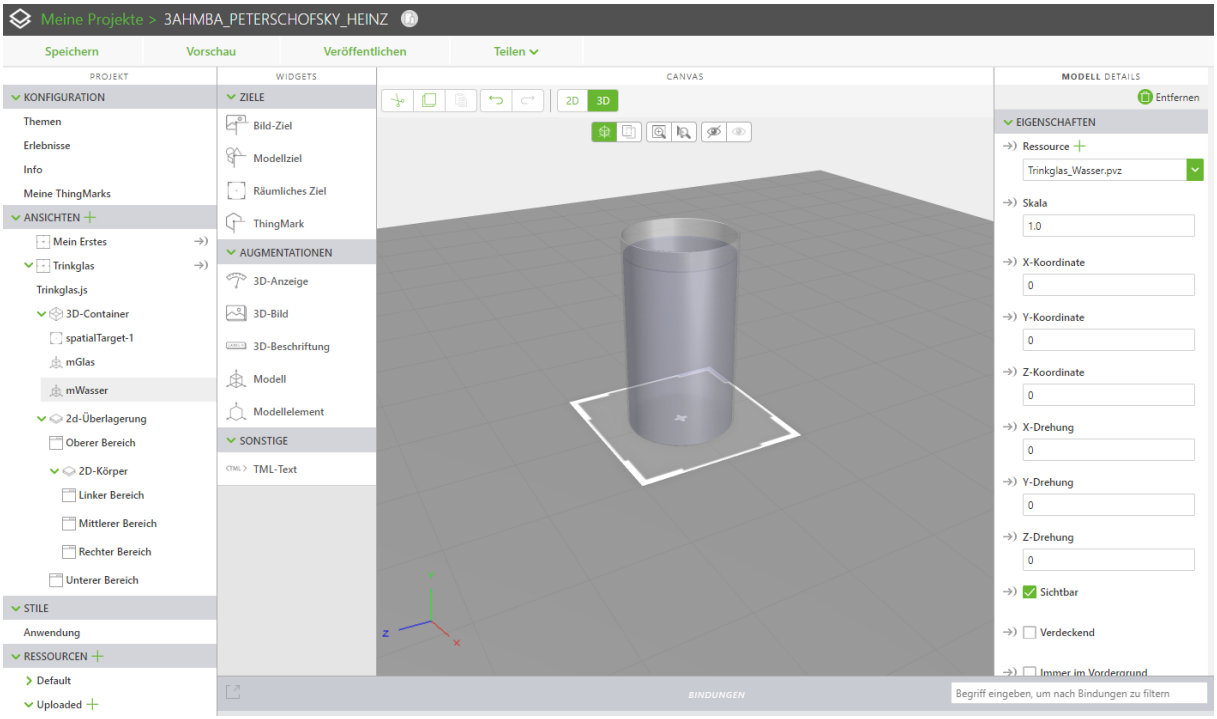


We add a target, then a model. I have drawn a nice drinking glass, import it as a resource and use it as a model.

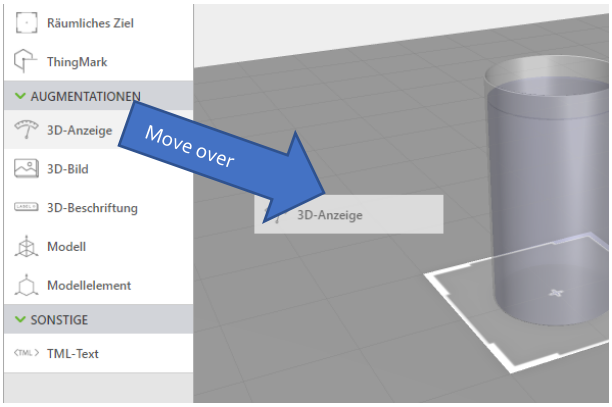
Because it's drawn accordingly on my end, I can put the glass on $(x; y; z) = (0; 0; 0)$ and it looks like it's standing on the table.



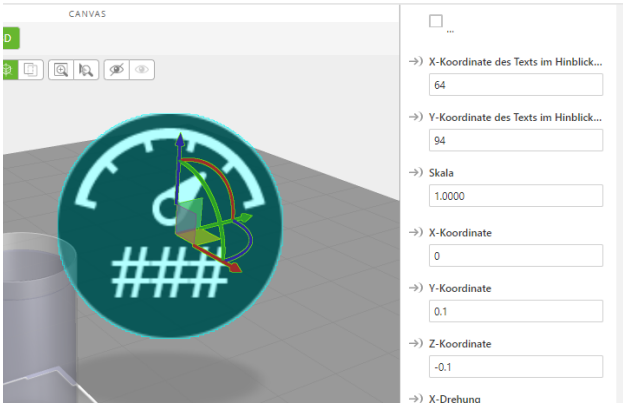
What we still need is a water filling. That's why I also drew myself a simple blue cylinder, which can almost completely fill the empty area of the glass (with the maximum fill level). The idea is that we adjust the y-coordinate with the fill level and so the glass looks like it has different fill levels. So I add a second model and position it also on $(x; y; z) = (0; 0; 0)$. With this, the glass now looks filled.



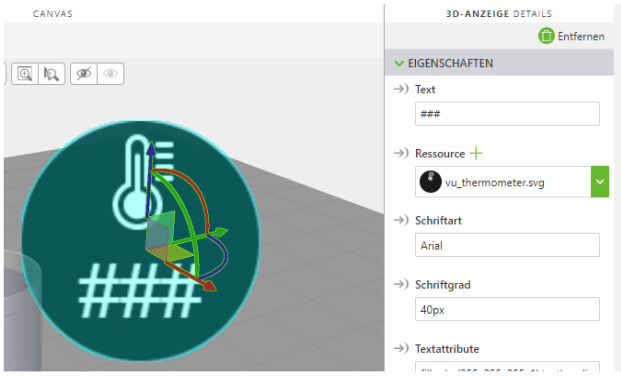
Looks pretty good already.



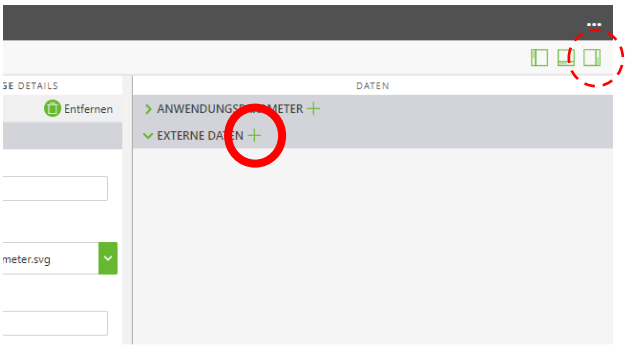
Now we first want to display the water temperature. To do this, there is a "3D Display" in the 3D view under Augmentation. So we take the 3D display and drag it into the canvas.



This display now looks relatively large. Let's set it to reasonable coordinates first. I'll put it next to the glass at the coordinates... $(x; y; z) = (0; 0.1; -0.1)$.

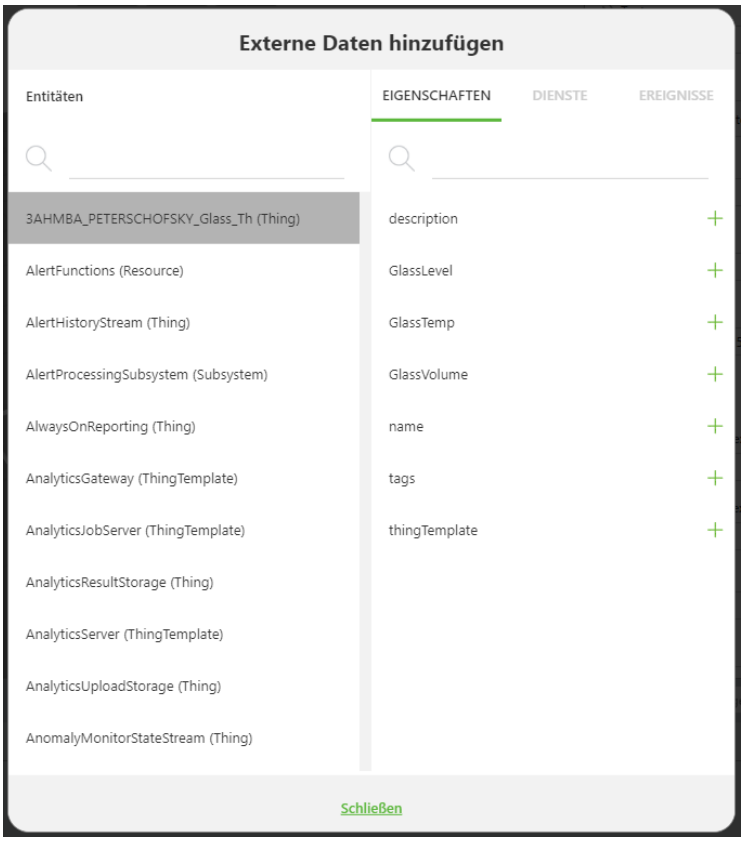


I would also like a temperature symbol to appear. Under resource you can choose a suitable symbol.

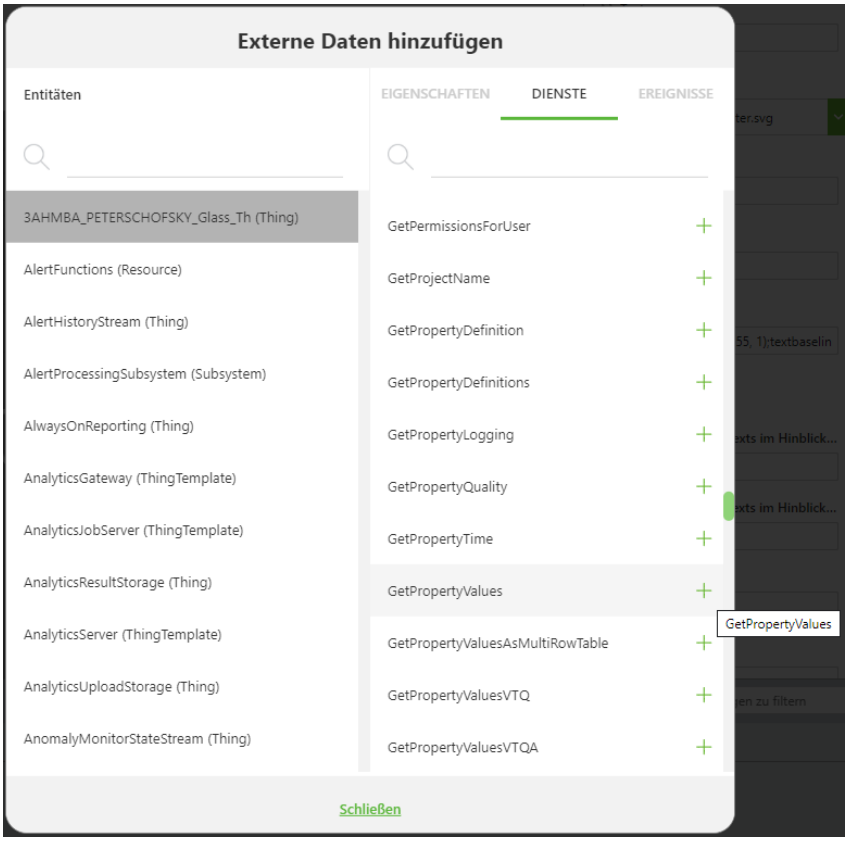


What we have to do now is that the text does not contain "###", but the current measured value, which we have to read out from Thingworx. To do this, we can click on the "+" next to EXTERNAL DATA in the right-hand area of the screen under DATA. If the data area is not there, you can show it in the upper right corner under the three dots.

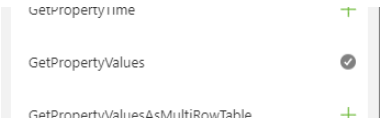
A dialog appears. Since our Experience server and the Thingworx server are the same, we get all Things listed that are on the Tingworx server - that's already practical. We choose our Thing and click on it (for me it was `3AHMBA_PETERSCHOFSKY_Glass_Th`).

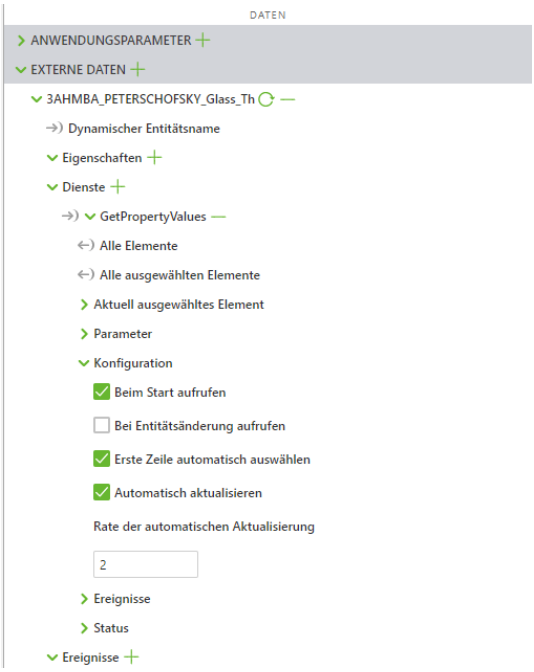


It also says something about our properties. But beware, trap! We won't go in there. We don't want something from the properties directly, but we want to start a service. The service we are looking for is `GetPropertieValues`. So we switch to SERVICES and look for it:



After we click the "+" icon next to it, it changes to a gray check mark. We have subscribed to the service. That's all we want from our Thingworx server for now, we can press close.





First we have to choose when we want to have the data. This happens on the right side at the external data. These have now changed. There we find our subscribed service. In the tree under GetPropertyValues > Configuration we set a few things:

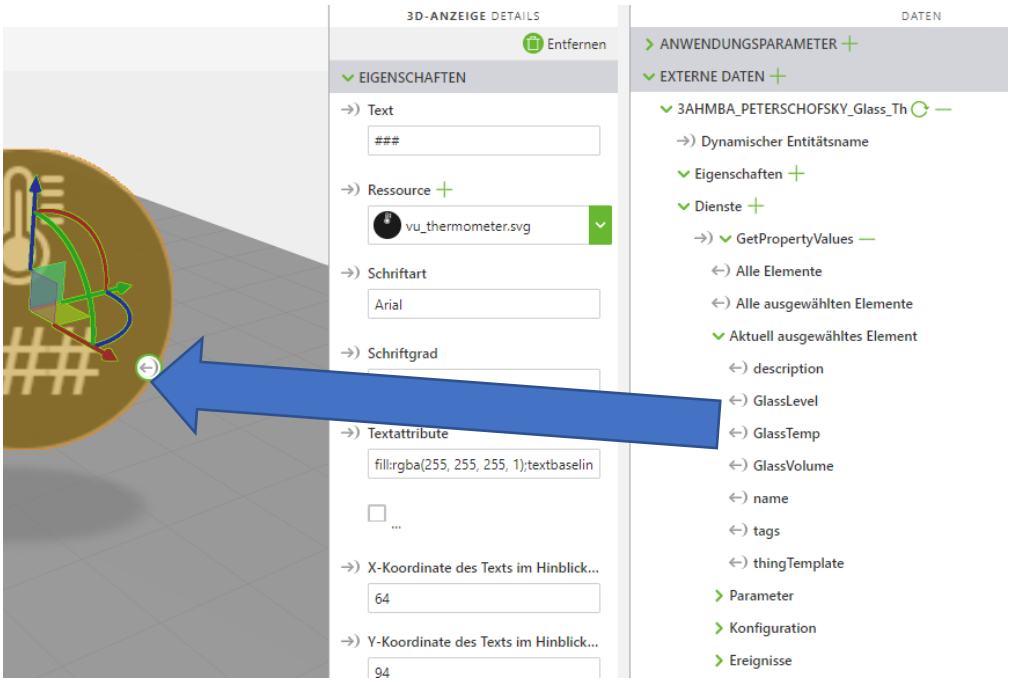
We want a query to be made at startup

We want an automatic update

We want this actuuleideurng every 2 seconds.

Once everything is set up like this, it should look something like the picture.

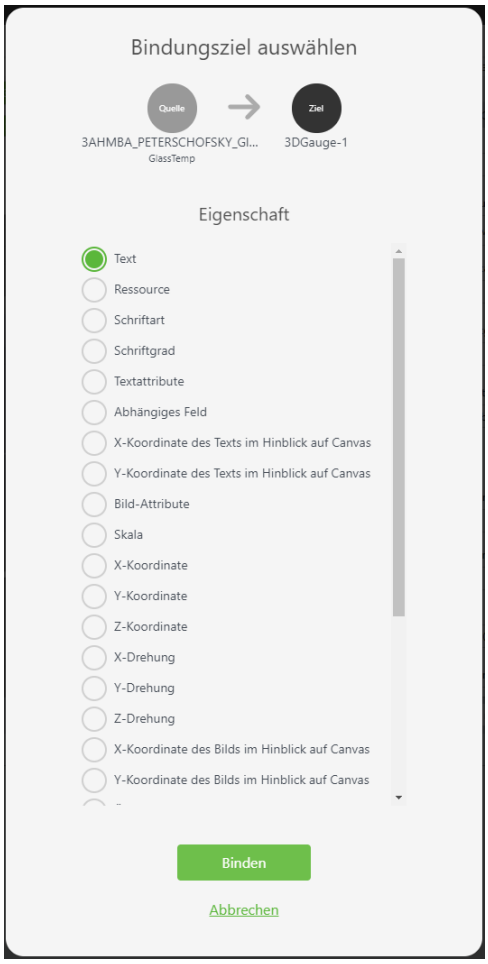
What is still missing now is that we have to tell somehow which value our display should show. To do this, we open the "Currently selected element" branch in the external data. There we find our properties again - we take the temperature (so `GlassTemp`) and drag it to the display. There we let it go and we get a selection of things we can bind to this value - quite a list!



Of course, we select the text here. After all, we want to display the temperature. Theoretically, we could also move the display (just change the coordinates). That would be the idea to change the water level.

But first we restrict ourselves to the display of the measured value, so we select text and press "Bind".

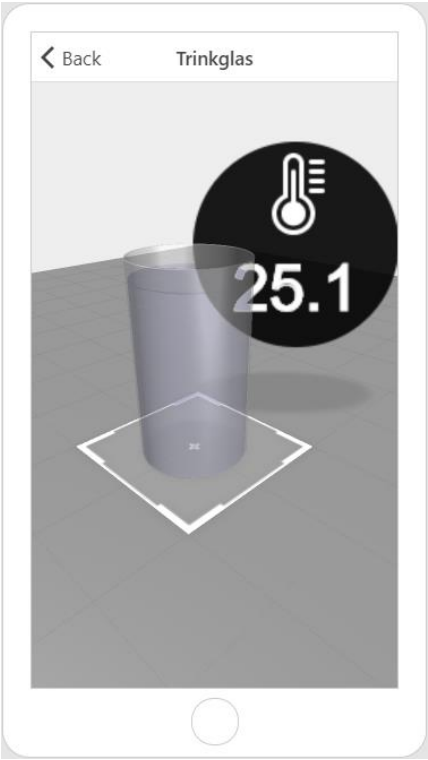
The binding printout will then also appear immediately in the lower area if the 3D display has been selected.

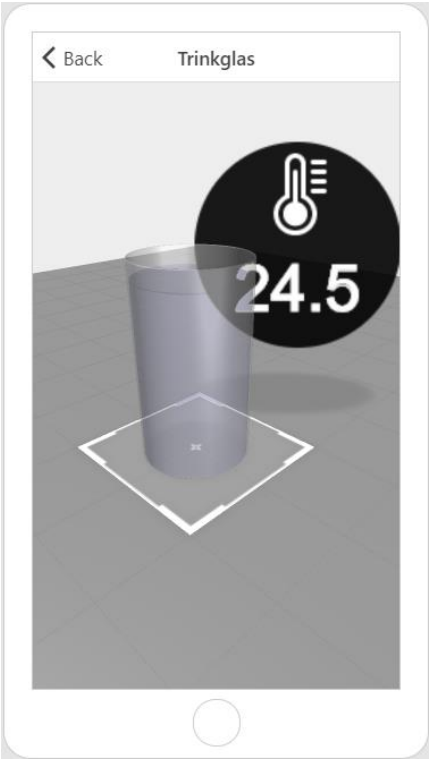


You will notice that the 3D display still has the name "`3DGauge-1`". Let's change that back to a better name. I use "`3DGTemp`".

That's about it. We can click on Preview. Indeed, the temperature appears in the 3D display!

If we change the temperature on the Thingworx server, the display also changes. The connection seems to be working. We can publish our experience.



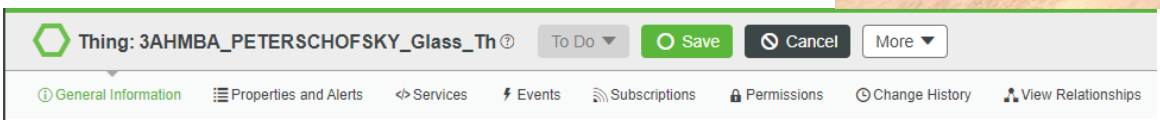


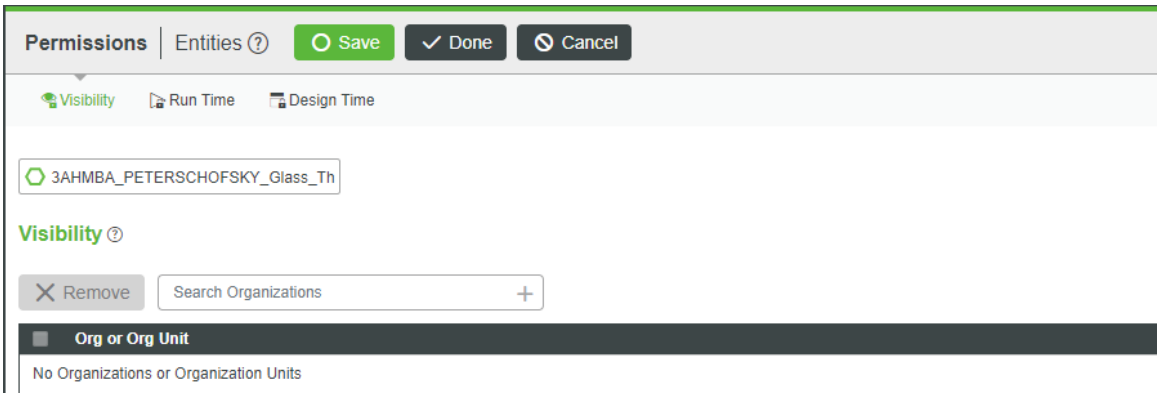
If we open the Experience with our smartphone, that should also work here. In fact, however, you get no values! The reason is the permissions. Our Experience is public - we don't have to enter any login data to start the Experience. That's why we're not logged in and we haven't entered an Application Key - so we don't get any data. It's as simple as that.

In principle, there are two ways to deal with this:

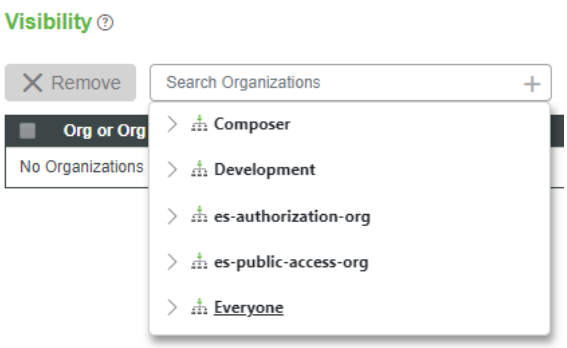
- Do not make the Experience public when publishing. Then you need username and password to open and receive data.
- Make the read access to the data public. If you allow the public to read the data, then the Experience also gets the necessary data.

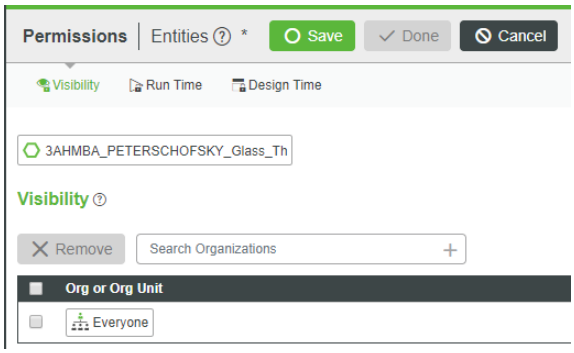
We will take the second approach. So let's open the associated Thing in Thingworx Composer. By clicking on "Permissions" in the top bar, we will be presented with the access permissions to our Thing.



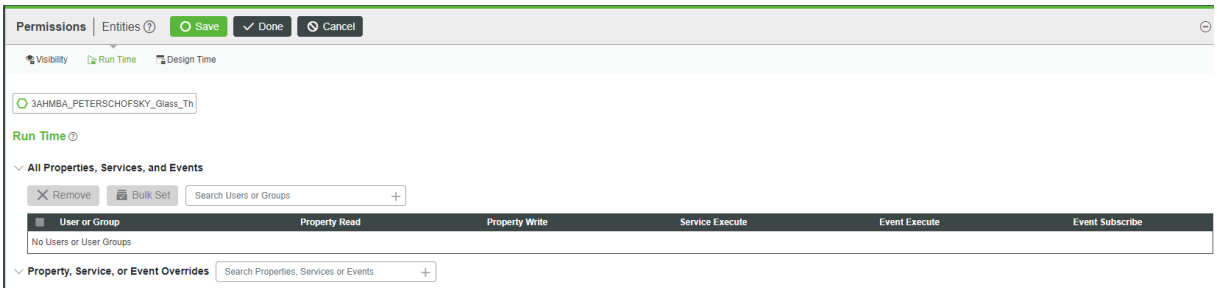


First of all, the problem is that our service is not visible from the outside. So we enter "`Everyone`" for the visibility and press "Save" afterwards.



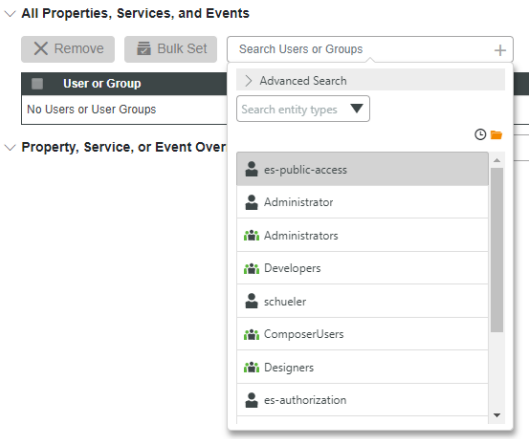


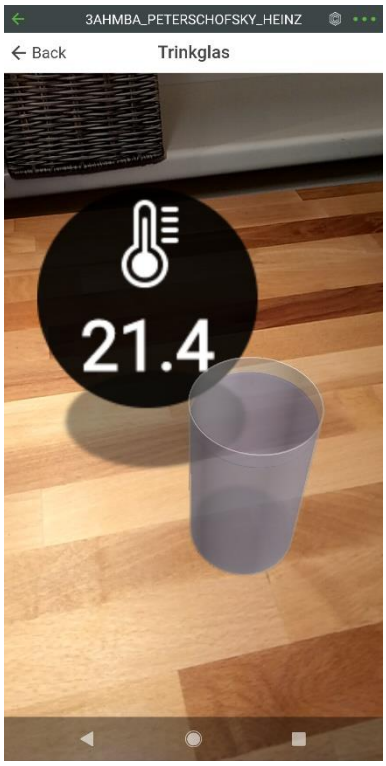
This makes the service visible, but we still have no rights. We need to change the run time access rights. Because at run time, we want the data. So we change there. At the moment nothing is entered - we want to change that.



The user we need to add is called "`es-public-access`". We give this user exactly two things: to read one (or more) properties and to start a service. This ensures that you can get the value (via the `GetPropertyValues` service), but you can't write to it (because we don't allow property writes).

Trying makes us sure: it seems to work - we see a value and it changes when we change it.





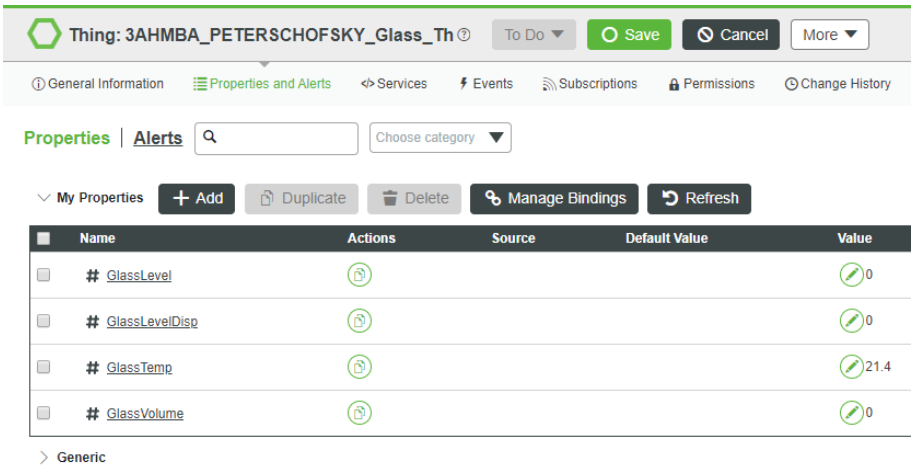
3.5.7.2 Mandatory task "Show augmented value

	Also display the temperature in its augmentation. Also expand the display to include the contents of the glass. Choose a suitable pictogram and place it sensibly.	

3.5.7.3 Animating an object

Video Link 13: https://youtu.be/q8vvcCuQYLY

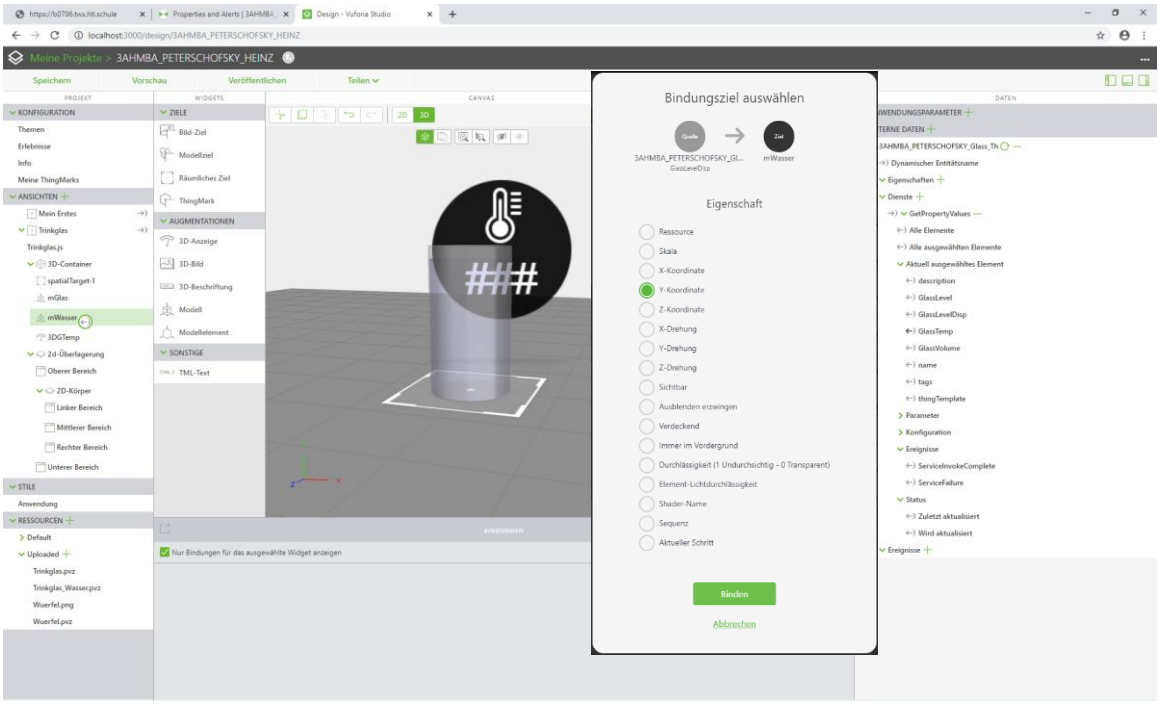
Unfortunately there is no easy way to manipulate the value we get from Thingworx beforehand. This means that if we just take the fill height and move the water block in y-direction with it, it won't work properly. What we need is already the correct y-coordinate in meters. So we need a new property where we can get the y-coordinate directly. Now, let's extend our Thing accordingly. Let's add a property called "`GlassLevelDisp`".

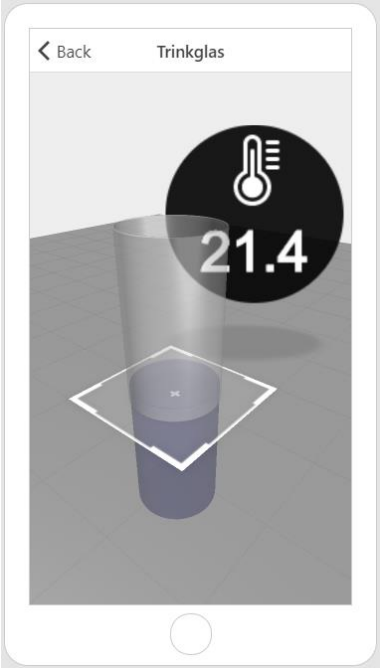


We should also have this property available in Vuforia Studio. If this is not the case, the update arrow next to the Thing can be pressed. If the Thing has been successfully saved in Thingworx, the new property should be available in Vuforia Studio by now at the latest.

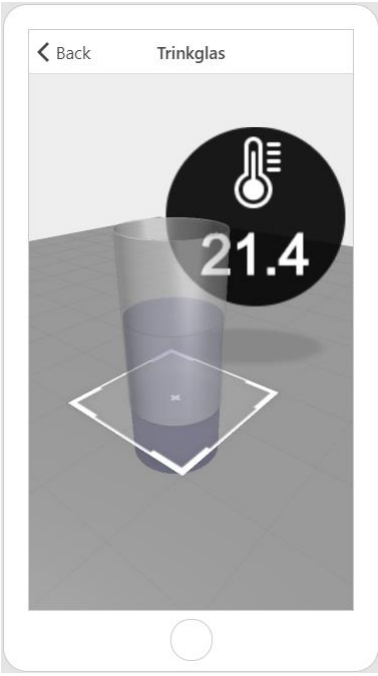
We now drag this onto the water model, which for me is called "`mWater`" (because this is supposed to change) and bind it to the Y-position. This will cause the water column to move in the y-direction. But where is actually zero now? Well, my glass has a $10mm$ thick bode and my water cylinder is $100mm$ high. That means a y-coordinate of $y = -0.09$ would be a "dry" bottom. And at $y = 0.01$ we're completely full.

So let's try previewing what happens.

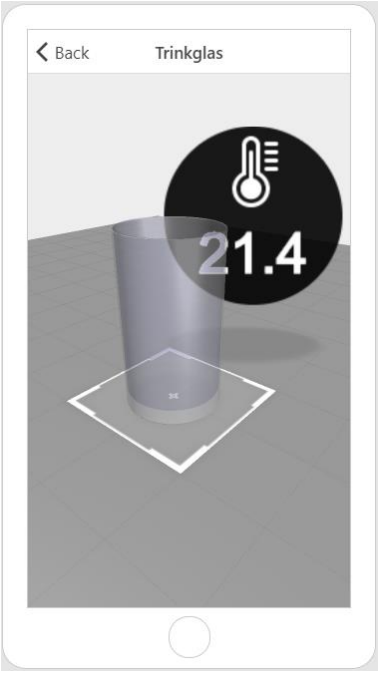




$GlassLevelDisp = -0{,}09$



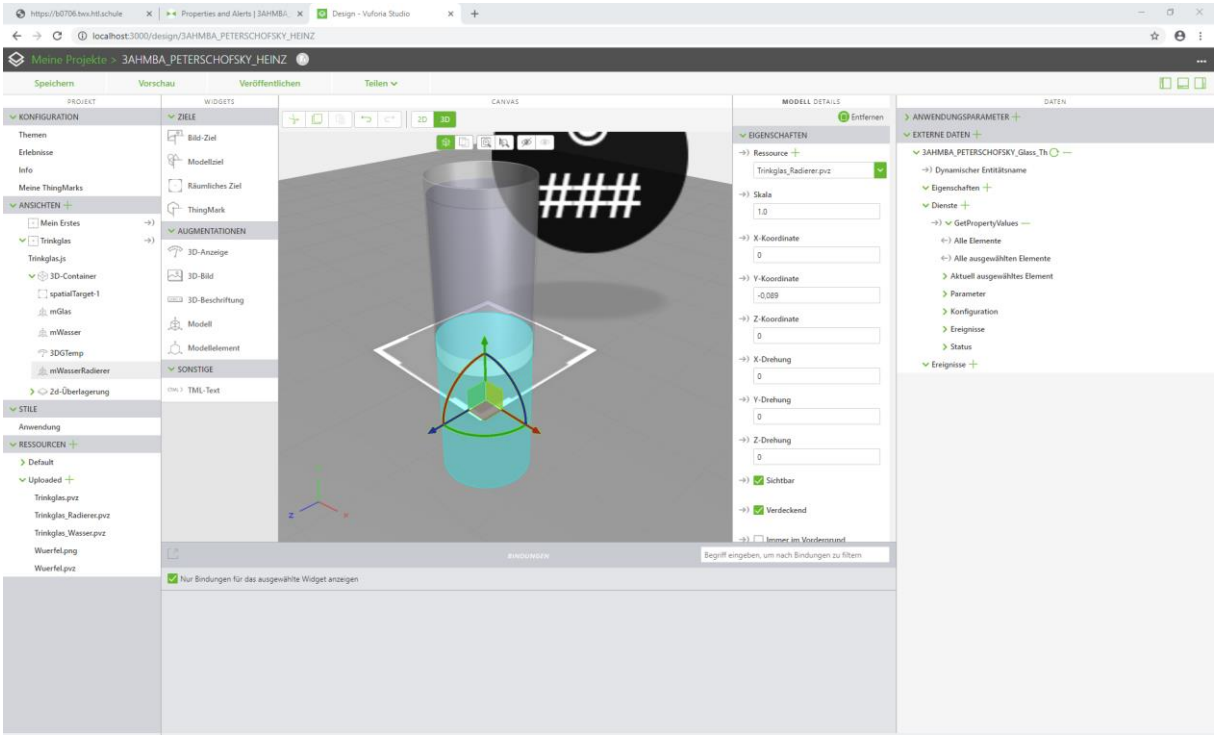
$GlassLevelDisp = -0{,}04$



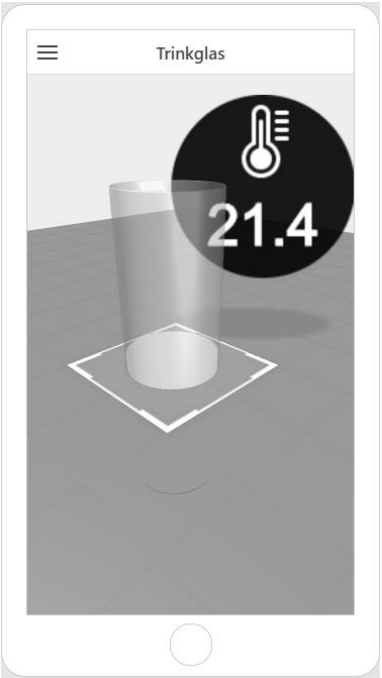
$GlassLevelDisp = +0{,}01$

In fact, it moves - but it doesn't look like it belongs that way. Indeed, the water column does not become less, but simply pushes down through the glass. Well, that's not how it's supposed to stay. We need an "eraser" which erases the lower part of the water. This eraser can also be a model, which becomes invisible but concealing. We place this eraser exactly under the water and when it goes down the lower part is covered by an invisible object - a super scam!

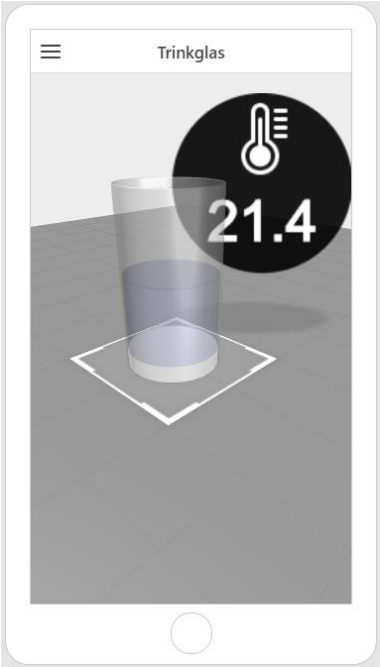
So fresh to the work and built a suitable eraser. It must have a little more diameter than the water, but a little less than the glass - otherwise the lower part of the glass is also erased. In my case it's a copy of the water model, only it's a bit $1mm$ higher and has $0.5mm$ more diameter. I add a new resource, a new model, select the resource and position it accordingly. After that I select it as visible and hidden.



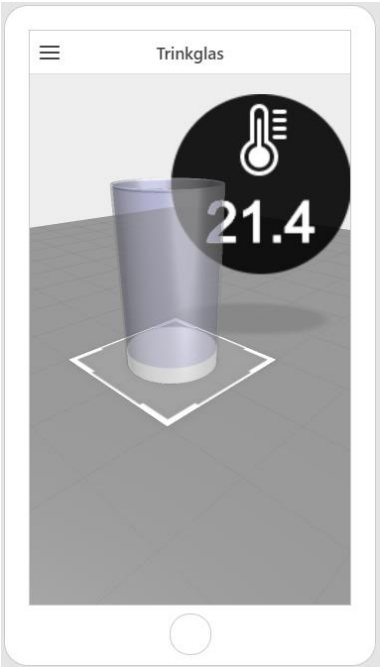
Let's try it. Indeed, the water disappears. That looks quite good now - only the fact that our eraser also erases the glass bottom is a bit disturbing. There is always such an ugly white base.



$GlassLevelDisp = -0{,}09$

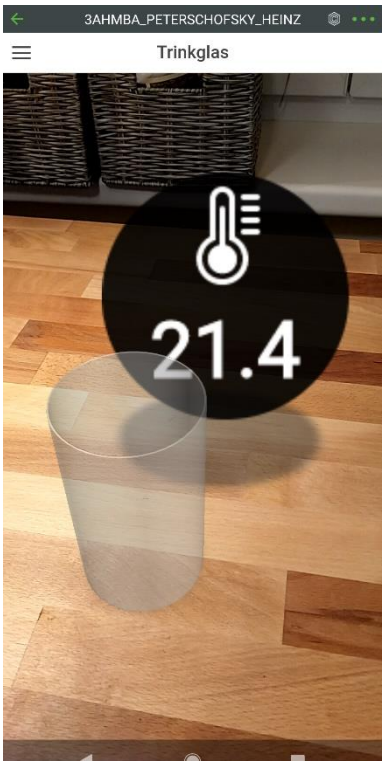


$GlassLevelDisp = -0{,}04$

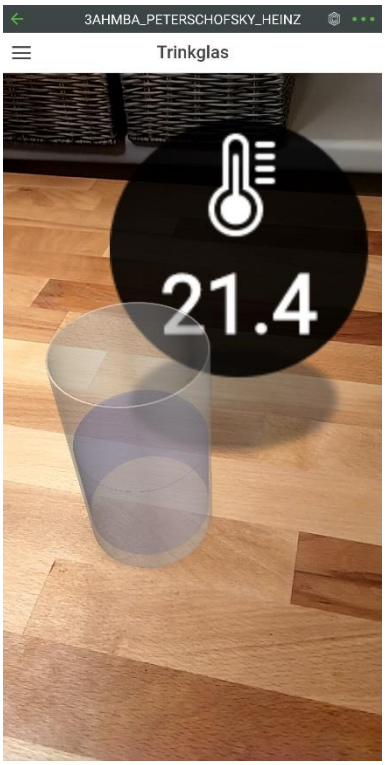


$GlassLevelDisp = +0{,}009$

But, it is at least so good that we can take a look and judge the result with a "real" mobile device:



$GlassLevelDisp = -0{,}09$



$GlassLevelDisp = -0{,}04$

$GlassLevelDisp = 0$

There you go, that looks pretty good. Fortunately, preview and "real" do not always match 100% together!

3.5.7.4 Mandatory task "Fillanimation

	Also represent the fill level in your augmentation. Try to represent the current contents of the glass at the water surface.	

4 Appendix

4.1 Arduino Reference

4.1.1 Language structure

4.1.1.1 Comments

```
// This is a comment in this line. It starts with //.

/* This is a comment over several lines.
   There is a beginning with /*
   And an end, which is marked with */ */
```

4.1.1.2 Variables

Variables consist of a type and a name:

```
type variablename;
```

Possible types are:

Integers:

`Byte`	with8-bit (i.e. values from 0 to `255`) A constant is simply specified as a number e.g.: `24`
`int`	on the Arduino with 16-Bit (so values from `-32768` to `32767`) A constant is simply given as a number e.g.: `5`
`unsigned int`	Arduino with 16-bit (i.e. values from `0` to `65535`) A constant is simply specified as a number e.g.: `125`
`long`	Arduino with 32-Bit (so values from `-2147483648` to `2147483647`) A constant is simply given as a number e.g.: `-275`
`unsigned int`	Arduino with 32-bit (i.e. values from `0` to `4294967295`) A constant is simply specified as a number e.g.: `55`

Decimal numbers (floating point representation):

`float`	Arduino 4 bytes, 6-7 decimal places and superscript. Value range: `-3.4028235E+38` to `3.4028235E+38` A constant must be specified with decimal point. e.g.: `5.0`

Characters:

`char`	the code of exactly one character. The ASCII table is used. A constant is specified in single apostrophe. So e.g. `'a'`
`String`	a character string. Differs from the ANSI C string. Is an object class with a wealth of methods. Please refer to www.arduino.cc for this. A constant is specified in double apostrophe. So e.g. `"blind nut"`.

Truth statements (Boolean expression)

`bool`	can be either `TRUE` or `FALSE`, simply a truth statement.

4.1.1.3 Objects

Objects are a bit more complex than simple variables, although the definition actually looks the same. Objects also have a type and a name. However, in addition to having a value, they also have "methods". These are effectively functions "built into" the variable.

An example is the `String` object mentioned earlier. This has a value, but also has built-in functions. One of these functions is e.g. `startsWith()`.

A code example for better understanding:

```
String name; // Name of a person
...
if (name.startsWith("Stefan")) {
  // Here we do something when the name starts with "Stefan".
}
...
```

The `startsWith()` does not have to be defined anywhere, it is already built into the class "`String`" - convenient.

4.1.1.4 Functions

Definition of a function:

```
type functionname([type1 parameter1][, type2 parameter2 [,...]]) {
     //Code of the function
}
```

`type`: Each function has a return value. This can also be of type `void`. However, a type for the return value must be specified before the function name.

`functionname`: A function has a name with which the function can be called. This consists of letters and numbers. The function name must always begin with a letter.

Functions can (but do not have to) have passing parameters. Each parameter has a type and a name - just like variables.

The curly brackets {..} summarize a statement block. Here they indicate the scope of the function. If the function consists of only one instruction, you could omit the brackets - but that would be very rare. This is more common with `if` statements. There, the corresponding statements are also grouped with {..} (see there).

Calling a function:

```
variable name = function name([type1 parameter1][, type2 parameter2
[,...]]);
```

4.1.1.5 Compile-time constants

At the time of program creation, constants can already be assigned. Before the program is compiled, the corresponding places are then replaced with the value of the constants (like a replace command in a document). Thus, such constants do not take up any space in the main memory of the Arduino. They are part of the program and therefore in program memory. This is often useful if you need values more than once and you are not sure if you want to change it in future versions.

Syntax:

```
#define CONSTANT VALUE
```

Whenever the character string `CONSTANTE` appears anywhere in the program, it is replaced with `VALUE` before translation.

An example would be the pin number of an LED signal. You don't always have to write pin number 3 but define a constant. If the LED number changes later, you only need to adjust the value of the constant and not at every code digit:

```
#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 (condition){
  ... // Code that is executed when condition = TRUE
} else {
  ... // Code that is executed when condition = FASLE
}
```

`condition` `bool` A truth expression. Either a result of a comparison, a boolean operation, or a calculation. All values not equal to 0 are evaluated as `TRUE`. All values equal to 0 as `FALSE`.

possible comparison operators are:

`!=`	not equal to
`<`	Less than
`<=`	Less than or equal to
`==`	Equal
`>`	Greater than
`>=`	Greater or Equal than

possible Boolean operators are

`!`	logical NOT
`&&`	logical AND
`||`	logical OR

4.1.2 Commands

4.1.2.1 *pinMode*

Syntax:

```
pinMode(pinNo, mode);
```

Parameters:

`pinNo`	`int`	Number of the pin to be set
`mode`	`int`	Mode for the PIN:
	`OUTPUT`	is to be used as output
	`INPUT`	is to be used as input
	`INPUT_PULLUP`	is to be used as input with the built-in pullup- resistor.

4.1.2.2 *digitalWrite*

Syntax:

```
digitalWrite(pinNo, status);
```

Parameters:

`pinNo`	`int`	Number of the pin to be written
`status`	`int`	Possibilities for the PIN:
	`HIGH`	the PIN is set to logical 1 (`+5V`), alternatively `1`
	`LOW`	the PIN is set to logical 0 (`0V`), alternatively also `0`

4.1.2.3 *digitalRead*

Syntax:

```
int value = digitalRead(pinNo);
```

Parameters:

`pinNo`	`int`	Number of the pin to be read
`value`	`int`	Possibilities for the result:
	`HIGH`	the PIN is set to logical 1 (`+5V`), alternatively `1`
	`LOW`	the PIN is set to logic 0 (`0V`), alternatively `0`

4.1.2.4 *pulseIn*

Syntax:

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

Parameters:

`pinNo` `int` Number of the pin to be read

`value` `int` Type of the pulse to be measured:
`HIGH` a pulse duration from `GND` to `+5V` is to be measured
`LOW` a pulse duration from `+5V` to `GND` is to be measured

`timeout` `unsingned long` Timeout for waiting for the start of the pulse in ms.
Default value: 1000ms

`duration` `unsingned long` Measured pulse duration at the specified pin in μs

4.1.2.5 *analogWrite*

Syntax:

```
analogWrite(pinNo, value);
```

Parameters:

`pinNo` `int` Number of the pin to be written (must be PWM-capable)

`value` `int` Pulse width modulation (PWM) duty cycle. Limits are:
`0` PWM is set to always off
`255` PWM is always on

4.1.2.6 *analogRead*

Syntax:

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

Parameters:

`pinNo` `int` Number of the pin to be read (pin `A0` is `0` , `A1` is `1` etc.)

`value` `int` Value of the input. Usually with Arduino 10 bit resolution. This means the input is divided into $2^{10} = 1024$ parts. So we get values between 0 and 1023.

4.1.2.7 *delay*

Syntax:

```
delay(timeMs);
```

Parameters:

`timeMs`	`unsigned long`	Waiting time of the program in milliseconds. No other code is executed!

Alternative:

```
delayMicroseconds(timeMs);
```

Parameters:

`timeMs`	`unsigned long`	Waiting time of the program in microseconds. No other code is executed!

4.1.2.8 *Millis*

Syntax:

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

Parameters:

`time`	`unsigned long`	Number of milliseconds that have passed since power-on. The maximum data range must be taken into account. The number repeats approximately every 50 days (starts again at 0).

4.1.2.9 *Micros*

Syntax:

```
time = micros();
```

Parameters:

`time`	`unsigned long`	Number of microseconds that have elapsed since power-on. The maximum data range must be taken into account. The number repeats approximately every 70 minutes (starts again at 0).

4.1.2.10 bitSet

Syntax:

```
bitSet(x, n);
```

Parameters:

x	byte	Variable in which a bit is to be set
n	unisgend int	Bit number of the bit to be set. 0 is the least significant bit (LSB, the "rightmost").

4.1.2.11 bitClear

Syntax:

```
bitClear(x, n);
```

Parameters:

x	byte	Variabe, in which a bit is to be deleted
n	unisgend int	Bit number of the bit to be deleted. 0 is the least significant bit (LSB, the "rightmost").

4.1.2.12 bitRead

Syntax:

```
int value = bitRead(x, n);
```

Parameters:

x	byte	Variable in which a bit is to be set
n	unisgend int	Bit number of the bit to be set. 0 is the least significant bit (LSB, the "rightmost").
value	int	Value of the requested bit (0 or 1)

4.1.2.13 bitWrite

Syntax:

```
bitWrite(x, n, b);
```

Parameters:

x	byte	Variable in which a bit is to be written
n	unisgend int	Bit number of the bit to be written. 0 is the least significant bit (LSB, the "rightmost").
b	int	Value of the bit to be written (0 or 1)

4.1.2.14 *shiftOut*

Syntax:

```
shiftOut(dataPin, clockPin, bitOrder, value);
```

Parameters:

`dataPin` `int` Pin number of the data channel (the bits of `value` are written there in sequence).

`clockPin` `int` Pin number of the clock channel. Whenever there is a new data bit on `dataPin`, `clockPin` changes from `0` to `1`.

`bitOrder` `int` determines in what order the value of `value` is pushed out to the `dataPin`. Possibilities are:
`MSBFIRST` Most significant bit first (the "leftmost" first).
`LSBFIRST` Least significant bit first (the "rightmost" first)

`value` `byte` value that is written out serially to the pins

4.1.2.15 *tone*

Syntax:

```
tone(pin, freq[, dur]);
```

Parameters:

`pin` `int` Pin on which the sound is to be output (must be PWM capable)

`freq` `unisgend int` Frequency of the sound to be output.

`major` `unsigned long` Length of the generated tone. Optional. If omitted, the tone will be played until `noTone` is called.

4.1.2.16 *noTone*

Syntax:

```
noTone(pin);
```

Parameters:

`pin` `int` Pin on which the sound is to be stopped

4.1.2.17 *isnan*

Syntax:

```
bool nan = isnan(value);
```

Parameters:

`value` `float` value to be checked for a number

`nan` `bool` Returns `true` if `value` is not a number (or `nan`).

4.1.2.18 attachInterrupt

Syntax:

```
attachInterrupt(intNumber, ISR, mode);
```

Parameters:

`intNumber` `int` Interrupt Number of the interrupt to be assigned to a function

`ISR` Name of the function to be called

`mode` `int` determines which events of the input are to be reacted to

- `LOW` Triggers the interrupt when the pin is `LOW`.
- `CHANGE` Triggers the interrupt when the pin changes.
- `RISING` Triggers the interrupt when the pin goes from `LOW` to `HIGH`.
- `FALLING` Triggers the interrupt when the pin goes from `HIGH` to `LOW`.

4.1.2.19 detachInterrupt

Syntax:

```
detachInterrupt(intNumber);
```

Parameters:

`intNumber` `int` Interrupt Number of the interrupt to be disabled.

4.1.2.20 digitalPinToInterrupt

Syntax:

```
int intNumber = digitalPinToInterrupt(pin);
```

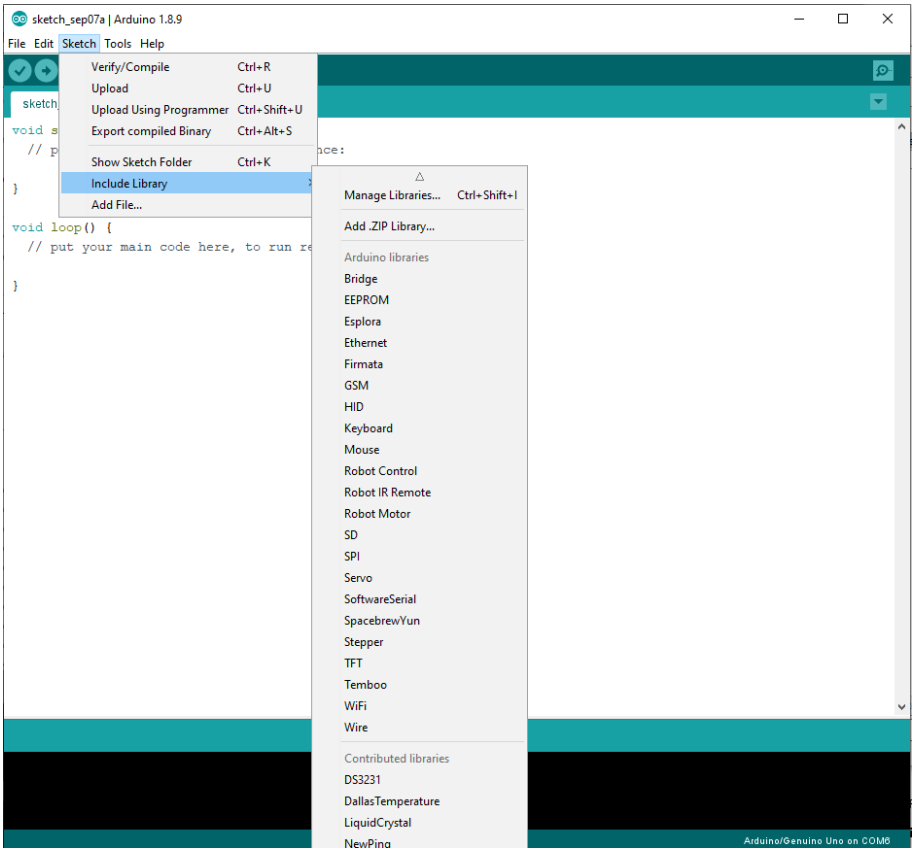
Parameters:

`pin` `int` Pin to which an interrupt is to be assigned. From this the corresponding interrupt number, which is needed for attachInterrupt and detachInterrupt, is calculated depending on the board used.

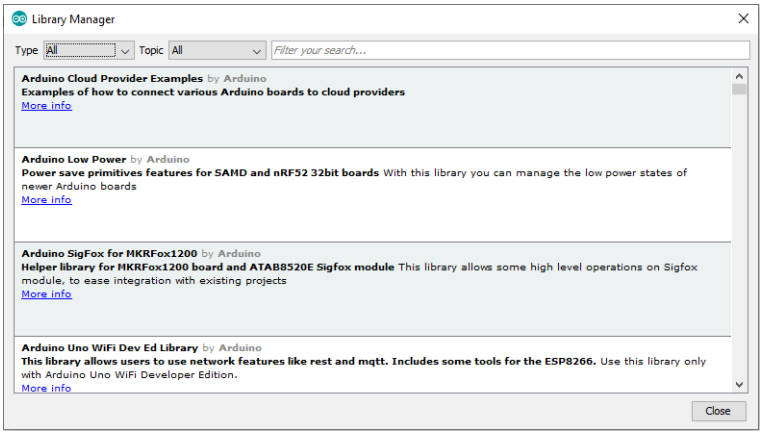
`intNumber` `int` Interrupt Number of the interrupt to be assigned to a function

4.1.3 Adding Libraries

Sometimes it is necessary to integrate foreign code into your project. This makes sense, because often a sensor etc. must be addressed specifically and very often people who have dealt with this topic more intensively have written so-called libraries for it. These contain the necessary code to cover a certain topic. In the following we will install a library which is used to receive infrared signals from a remote control. We will proceed step by step. Often you can find all necessary libraries exactly this way.

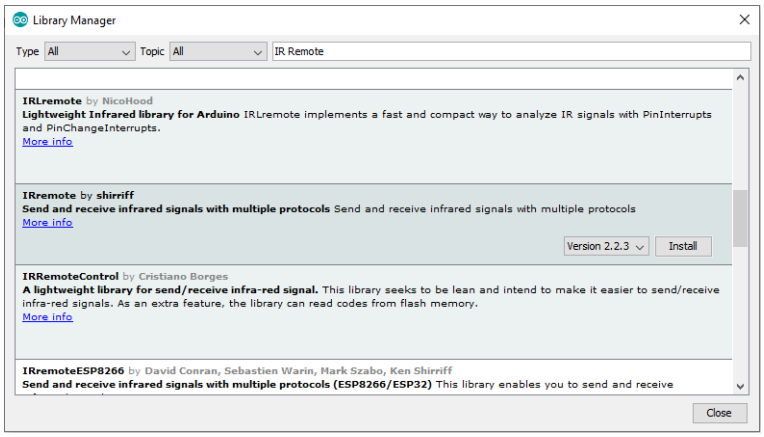


If the Arduino IDE is started, you will find the entry "Include Library" in the menu "Sketch". There are already many installed libraries listed. At the top there is the entry "Manage Libraries...". - we select it. Then the Library Manger opens.

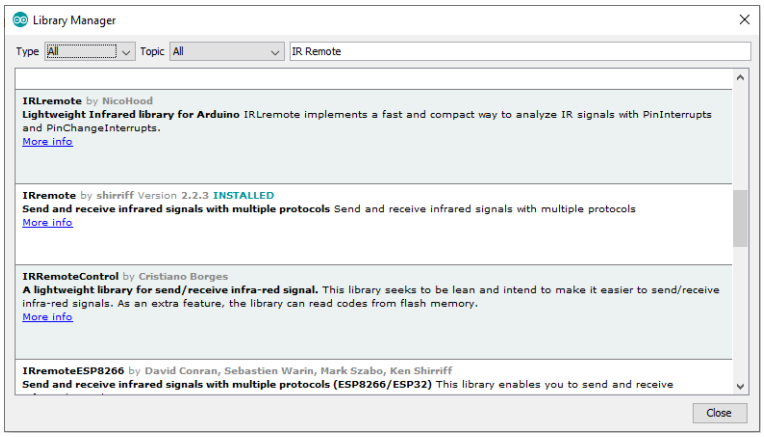


There you can try to search for suitable libraries. We want to receive IR signals, so we enter "IR Remote" as search. This limits the list considerably.

From this catalog you can select a suitable library. We choose the library "IRremote" from a certain "shirriff" - I have always used it:



An "Install" button appears there. If you press this button, the selected version of the library will be installed. This is also marked accordingly in the list of libraries.



And with that, we're done. The library is on our device and can be used with an `include` statement.

4.1.4 Important objects

4.1.4.1 Serial object

`Serial.begin(speed)` Initializes the serial interface

`speed` `longTransfer rate`

common rates: `300`, `1200`, `2400`, `4800`, `9600`, `19200`, `38400`, `57600`, `74880`, `115200`

`Serial.print(text[,format])` Output of a string on the serial interface.

`text` A wide variety of data can be used here, which should then appear as text on the serial interface if possible.

`format` You can specify how numbers are written:

- `BIN` Output as binary number
- `OCT` Output as octal number
- `DEC` Output as decimal number (default)
- `HEX` Output as hexadecimal number

`Serial.println(text[,format])` Output of a string on the serial interface. It is terminated with a carriage return (`'\r'`) and a new line (`'\n'`). (So practically with an "Enter").

`text` see `print()`

`format` see `print()`

`Serial.write(text)` Outputs a string on the serial interface.

`text` In contrast to the `print` method, a number is not interpreted, but a corresponding character is sent. For example, a value of 123 results in exactly one character, namely the one from the ASCII table with the number 123 (`'{'`). And not the character string `"123"`.

`bool Serial.available()` Returns the number of characters in the receive buffer of the serial port. A value of `0` means there is nothing to read. The maximum receive buffer is `64` bytes.

`char Serial.read()` Reads a character from the serial interface. The first character is returned as datatype `int`. Often used in conjunction with the `available` method. If the `read` method is called more than once in succession, the "next" character is always returned. A character is therefore considered to be read and is deleted from the receive buffer of the serial interface.

`char Serial.peek()` Reads a character from the serial interface. The first character is returned as datatype `int`. If the `peek` method is called more than once in succession, the "same" character is always returned. A character is therefore NOT read and remains in the receive buffer of the serial interface.

4.1.4.2 SoftwareSerial

If no hardware UART is available, a software UART can also be defined. The library "`SoftwareSerial.h`" is required. The object has the same methods as the normal serial object.

```
SoftwareSerial mySerial(RX_PIN, TX_PIN);
```

Parameters:

`RX_PIN`	`int`	Pin number for the receive pin
`TX_PIN`	`int`	Pin number for the transmit pin

4.1.4.3 EEPROM

Automatically becomes available with the inclusion of the library "`EEPROM.h`":

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

`EEPROM.read(add)`	Read the byte at the specified address.		
	`add`	`int`	Address start with `0`
`EEPROM.write(add,val)`	Write the byte at the specified address.		
	`add`	`intAddress`	start with `0`
	`val`	`byte`	Value to be written
`EEPROM.update(add,val)`	Write the byte at the given address. The write operation is only executed if there really is another value inside - to save write cycles.		
	`add`	`intAddress`	start with `0`
	`val`	`byte`	Value to write
`EEPROM.get(add,dat)`	Read a longer data type starting at the specified address.		
	`add`	`intAddress`	start with `0`
	`dat`	`*`	Value to write, can be anything, e.g. a `float`, or even an entire `struct`.
`EEPROM.put(add,dat)`	Write a longer data type starting at the specified address. Each byte is written internally with `EEPROM.update` to save write cycles to the EEPROM.		
	`add`	`int`	Address start with `0`
	`dat`	`*`	Value to write, may be anything, e.g. a `float`, or even a whole `struct`.
`int EEPROM.length()`	Returns the length of the EEPROM. Different boards have different EEPROM memory locations.		

4.1.5 ASCII table (excerpt)

DEC	Characters
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	Characters
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	Characters
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	Characters
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 Resistance color code

Marking with 4 rings:

Color	Resistance value			Tolerance
	1st ring (tens)	2nd ring (One)	3rd ring (multiplier)	5th ring
none				$\pm 20\%$
Silver			10^{-2}	$\pm 10\%$
Gold			10^{-1}	$\pm 5\%$
Black		0	10^{0}	
Brown	1	1	10^{1}	$\pm 1\%$
Red	2	2	10^{2}	$\pm 2\%$
Orange	3	3	10^{3}	
Yellow	4	4	10^{4}	
Green	5	5	10^{5}	$\pm 0{,}5\%$
Blue	6	6	10^{6}	$\pm 0{,}25\%$
Purple	7	7		$\pm 0{,}1\%$
Grey	8	8		$\pm 0{,}05\%$
White	9	9		

Marking with 5 or 6 rings:

Color	Resistance value				Tolerance	Temp. coefficient
	1st ring (hundreds)	2nd ring (tens)	3rd ring (One)	4th ring (multiplier)	5th ring	6th ring
Silver				10^{-2}		
Gold				10^{-1}		
Black		0	0	10^{0}		$200 \cdot 10^{-6}\ K^{-1}$
Brown	1	1	1	10^{1}	$\pm 1\%$	$100 \cdot 10^{-6}\ K^{-1}$
Red	2	2	2	10^{2}	$\pm 2\%$	$50 \cdot 10^{-6}\ K^{-1}$
Orange	3	3	3	10^{3}		$15 \cdot 10^{-6}\ K^{-1}$
Yellow	4	4	4	10^{4}		$25 \cdot 10^{-6}\ K^{-1}$
Green	5	5	5	10^{5}	$\pm 0{,}5\%$	
Blue	6	6	6	10^{6}	$\pm 0{,}25\%$	$10 \cdot 10^{-6}\ K^{-1}$
Purple	7	7	7		$\pm 0{,}1\%$	$5 \cdot 10^{-6}\ K^{-1}$
Grey	8	8	8		$\pm 0{,}05\%$	
White	9	9	9			

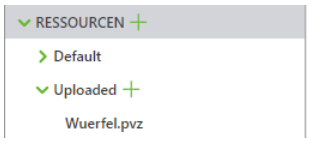
4.3 Vuforia Studio Reference

4.3.1 Project creation procedure in Vuforia Studio

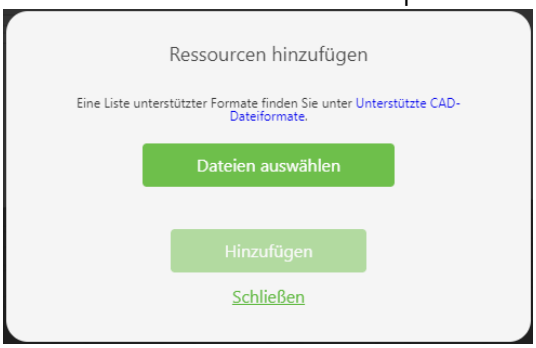
1. Create Project:
   Project Name Convention: **CLASS_SURNAME_GIVENAME<_HOLO>**
   Experience URL: **https://example.twx.htl.schule: 8443**
2. Add target:
   Image target: tries to find in reality the image.
   Model target: Try to find the 3D model in reality.
   Spatial target: plane surface to select in reality
   ThingMark: 2D code that is searched in reality.
3. Add virtual objects
4. Test of the Experience Service
5. Preview
6. Publication

4.3.2 Importing a model

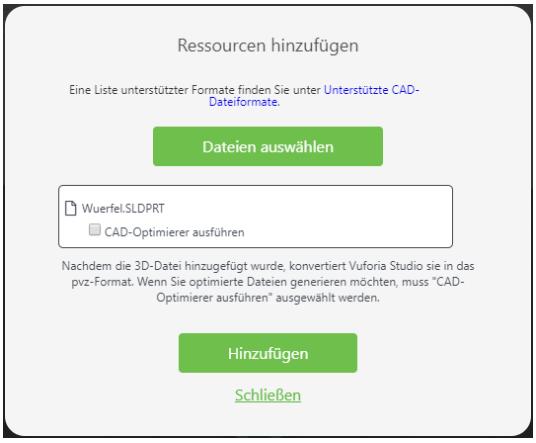
1. Pressing the "+" symbol at the bottom left of the resources



2. Selection of the CAD file to be imported



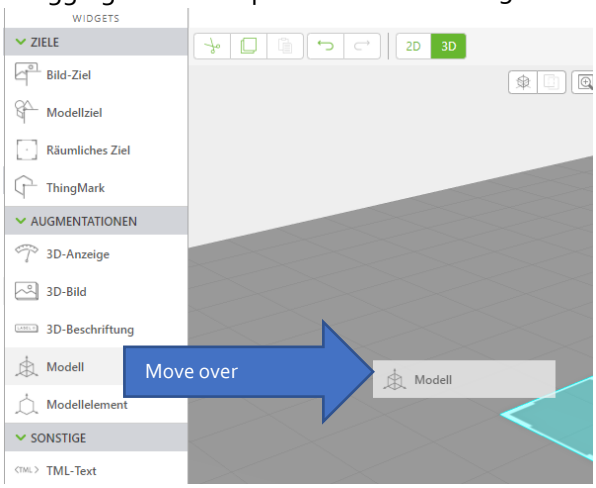
3. For large detailed models, the CAD optimizer can be used. This tries to simplify the CAD model without influencing the geometry too much. There is then not only one imported file, but 3 in different levels of detail.



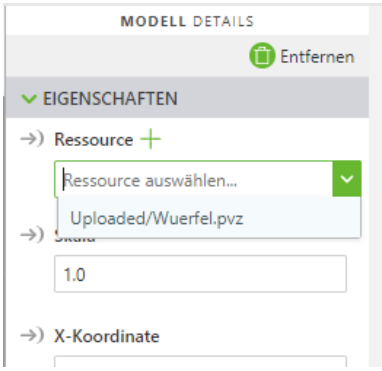
4. Import: The file is converted into a pvz file and stored in the project directory.

4.3.3 Adding a model

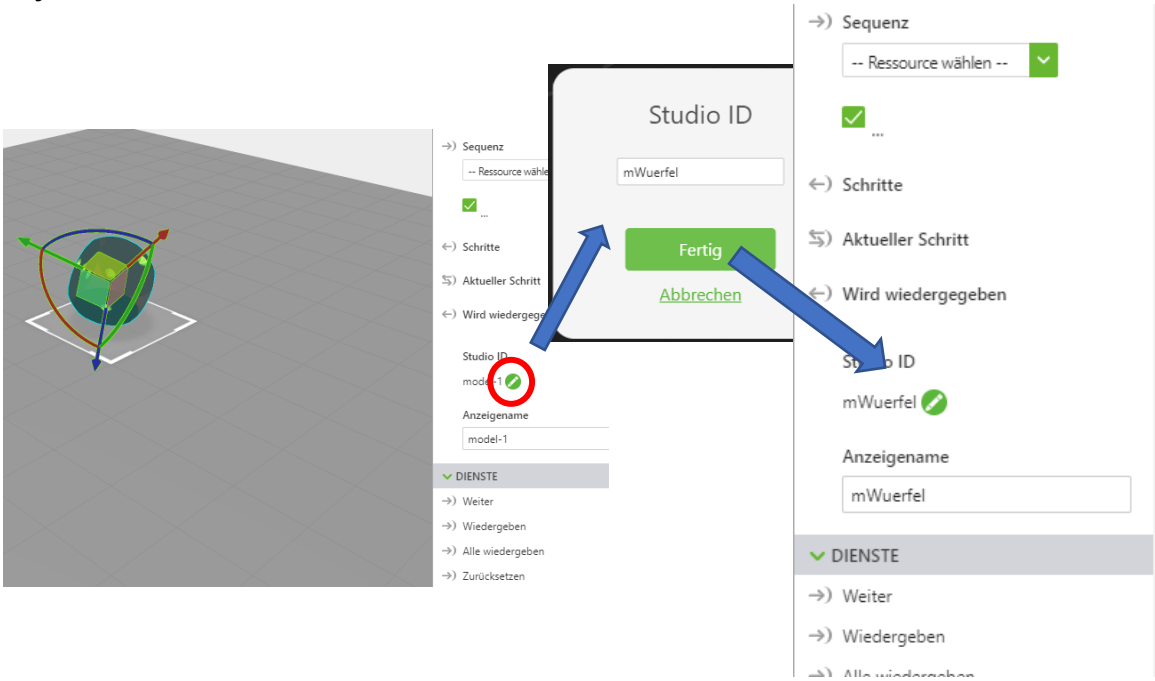
1. Dragging the model placeholder into the 3D drawing area (3D canvas)



2. Selection of the resource to be used



3. Adjust the position and rotation depending on the resource.
4. Adjust the Studio-ID to a "suitable" name (Attention! No umlauts!)



4.3.4 Ionicons -Codes for symbols

−	ion-minus-round	+	ion-plus-round
<	ion-chevron-left	>	ion-chevron-right
↱	ion-arrow-return-right	↰	ion-arrow-return-left

4.4 Install MQTT broker Mosquitto on Raspberry

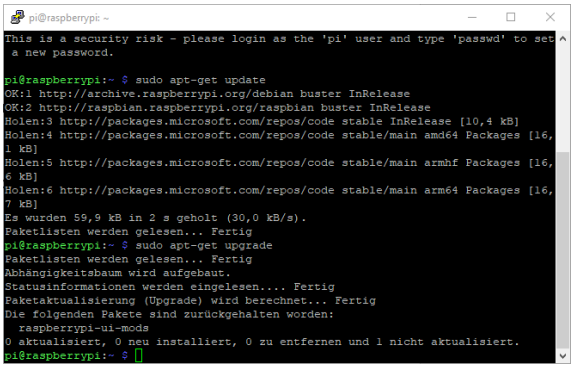
Video Link x01: https://youtu.be/ID1voUOEdyc

In this tutorial we use a RaspberryPi single board computer as MQTT broker. Prerequisite for the success of the following procedure is the installation of a RaspberryOS and for access the use of PuTTY[57] .

The MQTT broker "Mosquitto" used here is part of the Raspberry repository. This is a library that contains software, much like a Play store, only not as convenient to use. First, we need to bring the library's catalog contents up to date. To do this, we open an SSH connection with PuTTY and enter the command

```
sudo apt-get update
```

an[58] .



After that, we upgrade to the latest packages. This will bring all installed packages up to date. The corresponding command is

```
sudo apt-get upgrade
```

So we are up to date with our installation. Now follows the actual installation of Mosquitto. We install this with the help of the command

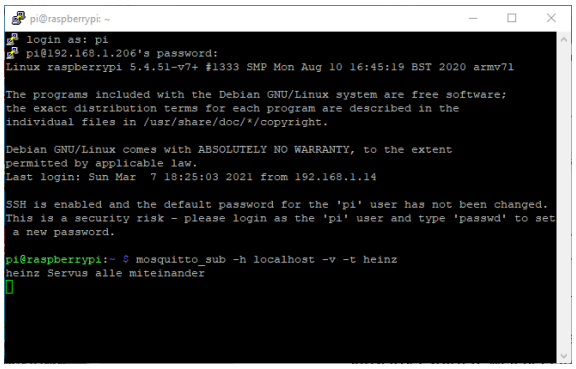
```
sudo apt-get install -y mosquitto mosquitto-clients
```

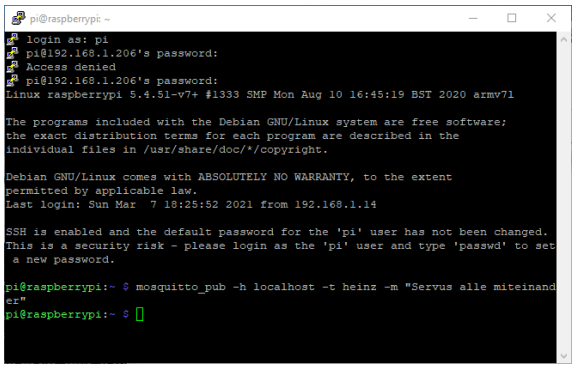
After that, the Mosquitto server should already be running. We can test this by starting an MQTT subscriber which subscribes to a certain topic. This can happen with the command (Subscribed to the channel `heinz`):

```
mosquitto_sub -h localhost -v -t heinz
```

If you open a second PuTTY window and call a publication on the topic `heinz` there, this should appear in the first window.

```
mosquitto_pub -h localhost -t heinz -m "Hello everyone"
```





[57] Both things are described in the script about databases.

[58] This means "superuser do" (`sudo`) and Advanced Packaging Tool (`apt-get`), please get the current package list (`update`). It is very similar for the other commands.

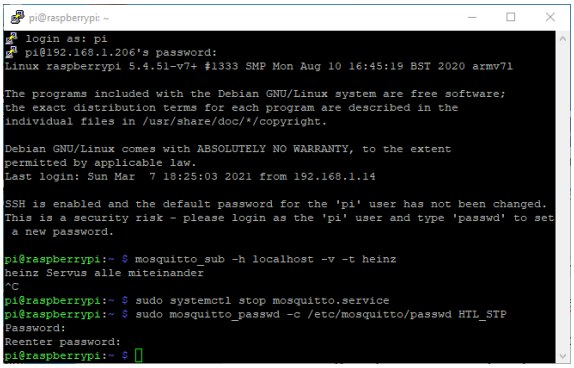
However, the configuration installed this way is insecure. At the moment every participant can subscirbe and publish. Therefore, we still set a password for access. To do this, we first stop the server with the command

```
sudo systemctl stop mosquitto.service
```

Now we have to create a password file first. Mosquitto supports us with a utility and we can create such a file with the following command (filename here is `passwd`):

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd HTL_STP
```

Where `HTL_STP` is already the user name. We are now prompted to enter a password (note that there is no visual feedback of the input). I choose `HTLinnovativ` here.

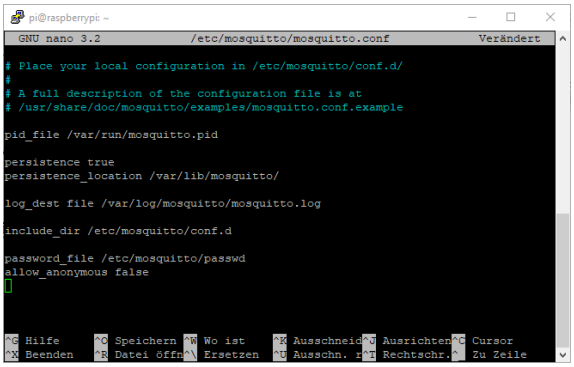


Now we have to change the configuration file. To do this, we open it with the text editor `nano`.

```
sudo nano
/etc/mosquitto/mosquitto.conf
```

There we add the lines

```
password_file /etc/mosquitto/passwd
allow_anonymous false
```



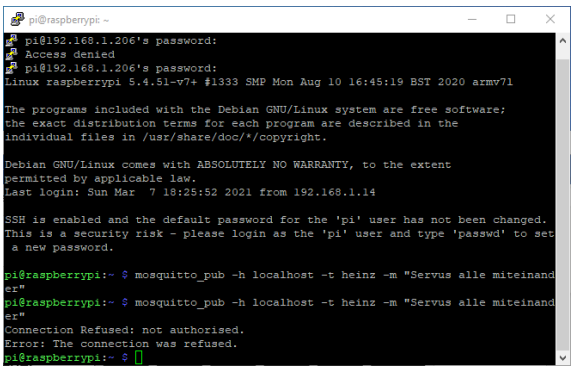
Added.

With `Ctrl+X` and `J` we end and save the file.

Now we just have to start the server again. This is done with the command

```
sudo systemctl start mosquitto.service
```

As a test, we can try to send a message again - just like before. We notice: We can't do that anymore. The server is ready.



4.5 The ESP programs for IoT

4.5.1 Commands for the MQTT Gateway

Commands to the MQTT gateway are transmitted serially. There are the following keywords and responses. The responses depend on the set information density. There are 4 different levels: Error, Warning, Information and Debug. Each level can be switched off individually. The colors in the mentioned responses indicate which level must be active to see the response.

Error: Red warning: orange Information: green debug: blue

Command	possible responses	Meaning
?	!	Checking whether the ESP connection is working
reset	Resetting Module ... READY	Reset of the module, all data will be deleted.
wifi	CON DIS	Indicates whether is connected to the Wifi network.
mqtt	CON DIS	Indicates whether connected to the MQTT broker.
nowifi	Stopping Wifi. OK	Turning off the Wifi connection, even with already transmitted SSID and password
wifion	Starting Wifi. OK	Activating the Wifi connection
startdebug	-	Activates the output of the level debug (bit 7)
stopdebug	-	Disables the output of the level debug (bit 7)
startinfo	-	Activates the output of the level info (bit 6)
stopinfo	-	Disables the output of the level info (bit 6)
startup warning	-	Activates the output of the level warning (Bit 5)
stopwarn	-	Deactivates the output of the level warning (bit 5).
starterr	-	Activates the output of the level error (Bit 4)
stoperr	-	Deactivates the output of the level error (bit 4).
setverbose <verb>	Set Verbose to : <verb> OK Missing Argument. ERR	Sets the information density level (alternative to the start/stop commands.
getverbose	ERROR WARN. INFO. DEBUG (<verb>) X X X X	Returns the current information density level.
setbaud <rate>	Changing Baud Rate to: '<rate>' OK Invalid Baud Rate. ERR Missing Argument. ERR	Changes the baud rate of the module. At reboot the baud rate is always 9600. Valid are the usual values between 300 and 115200.
setblocktime <time>	Minimum time between Requests: '<time> ' OK Invalid Time given (valid: 0 - 30). ERR Missing Argument. ERR	Blocking time: Sets the minimum waiting time until a new value can be transmitted again. Valid is 0 to 30 seconds.
getblocktime	Minimum time between Requests: '<time> ' OK	Returns the currently set time for the blocking time in seconds.
setresendtime <time>	Time for resending values: '<time> ' OK Invalid Time given (valid: 0 - 3600). ERR Missing Argument. ERR	Retry time: Sets the time span when the values are sent to the MQTT broker again without a new value.
getresendtime	Time for resending values: '<time> ' OK	Returns the currently set time for the repeat time in seconds.
setssid <ssid>	Set WifiSSID: '<ssid>' OK	Sets the Wifi SSID. If <ssid> is omitted, the SSID is deleted.
getssid	Get WifiSSID: '<ssid>' OK	Returns the currently set SSID.
setpass <pw>	Set WifiPass: '<pw>' OK	Sets the Wifi password. If <pw> is omitted, the password is deleted.
getpass	Get WifiPAss: '<pw>' OK	Returns the currently set Wifi password.
sethost <ip>	Set Host: '<ip>' OK	Sets the IP address of the MQTT broker.
gethost	Get Host: '<ip>' OK	Returns the currently set IP address of the MQTT broker.
setusername <name>	Set Username: '<name>' OK	Sets the username for the connection to the MQTT broker.
getusername	Get Username: '<name>' OK	Returns the currently set username of the MQTT broker.

setuserpass <pw>	Set Userpass: '<pw>' OK	Sets the user password for the connection to the MQTT broker.
getuserpass	Get Userpass: '<pw>' OK	Returns the currently set user password of the MQTT broker.
setsubsribe <#> <name>	Set Subscribe Topic #<#>: '<name>' OK	Subscribe to a topic with the specified name. You can subscribe up to 50 topics, that means <#> must be between 0 and 49.
	Invalid Subscribe Number. ERR	
	Missing Argument. ERR	
getsubsribe <#>	Get Subscribe Topic #<#>:'<name>' OK	Returns the subscribed topic with the specified number.
	Invalid Subscribe Number. ERR	
	Missing Argument. ERR	
setpropname <#> <name>	Set Property Name #<#>: '<name>' OK	Sets the name of the general topic with the number <#>. You can specify 50 general topics, which means <#> must be between 0 and 49.
	Invalid Property Number. ERR	
	Missing Argument. ERR	
getpropname <#>	Get Property Name #<#>: '<name> ' OK	Returns the name of the general topic with the number <#>.
	Invalid Property Number. ERR	
	Missing Argument. ERR	
setpropval <#> <val>	Set Property Value #<#>: '<val>' OK	Sets a new value for the general topic with the number <#>. If the new value differs from the last transmitted value, a transmission to the MQTT broker is triggered immediately.
	Invalid Property Number. ERR	
	Missing Argument. ERR	
getpropval <#>	Get Property Value #<#>: '<val>' OK	Returns the last received value of the specified general topic numbered <#>.
	Invalid Property Number. ERR	
	Missing Argument. ERR	
setvaluename <#> <name>	Set Value Name #<#>: '<name>' OK	Sets the name of the value-topic with the number <#>. 50 value-topics can be specified, which means <#> must be between 0 and 49.
	Invalid Value Number. ERR	
	Missing Argument. ERR	
getvaluename <#>	Get Value Name #<#>: '<name>' OK	Returns the name of the value topic with the number <#>.
	Invalid Value Number. ERR	
	Missing Argument. ERR	
setvalth <#> < th>	Set Value Threshold #<#>: '<th>' OK	Sets the transmission threshold of the value topic with the number <#>. If a new value is received, it is compared with the last value sent. Only if the difference is greater than the set value is the new value immediately published on the MQTT server. Default value is 0.1.
	Invalid Value Number. ERR	
	Missing Argument. ERR	
getvalth <#>	Get Value Threshold #<#>: '<th>' OK	Returns the transmission threshold of the value topic numbered <#>.
	Invalid Value Number. ERR	
	Missing Argument. ERR	
setval <#> < val>	Set Value #<#>: '<val>' OK	Sets a new value for the value topic with the number <#>. If the new value differs from the last transmitted value by more than the set transmission threshold, a transmission to the MQTT broker is triggered immediately.
	Invalid Value Number. ERR	
	Missing Argument. ERR	
getval <#>	Get Value #<#>: '<val>' OK	Returns the last received value of the specified Value-Topics with the number <#>.
	Invalid Value Number. ERR	
	Missing Argument. ERR	
setboolname <#> <name>	Set Bool Name #<#>: '<name>' OK	Sets the name of the Boolean topic with the number <#>. 50 Boolean topics can be specified, which means <#> must be between 0 and 49.
	Invalid Boolean Number. ERR	
	Missing Argument. ERR	
getboolname <#>	Get Bool Name #<#>: '<name>' OK	

	Invalid Boolean Number. ERR	Returns the name of the Boolean topic with the number <#>.
	Missing Argument. ERR	
setboolval <#> < val>	Set Bool Value #<#>: '<val>' OK	Sets a new value for the Boolean topic with the number <#>. If the new value differs from the last transmitted value, a transmission to the MQTT broker is triggered immediately.
	Invalid Boolean Number. ERR	
	Missing Argument. ERR	
getboolval <#>	Get Bool Value #<#>: '<val>' OK	Returns the last received value of the specified Boolean topic with the number <#>.
	Invalid Boolean Number. ERR	
	Missing Argument. ERR	

4.5.2 Commands for the Thingworx Gateway

Commands to the Thingworx Gateway are transmitted serially. There are the following keywords and responses. The responses depend on the set information density. There are 4 different levels: Error, Warning, Information and Debug. Each level can be switched off individually. The colors in the mentioned responses indicate which level must be active to see the response.

Error: Red Warning: orange information: green debug: blue

Command	possible responses	Meaning
?	!	Checking whether the ESP connection is working
reset	Resetting Module ... READY	Reset of the module, all data will be deleted.
wifi	CON DIS	Indicates whether is connected to the Wifi network.
twx	CON DIS	Indicates whether connected to the Thingworx server.
nowifi	Stopping Wifi. OK	Turning off the Wifi connection, even with already transmitted SSID and password
wifion	Starting Wifi. OK	Activating the Wifi connection
startdebug	-	Activates the output of the level debug (bit 7)
stopdebug	-	Disables the output of the level debug (bit 7)
startinfo	-	Activates the output of the level info (bit 6)
stopinfo	-	Disables the output of the level info (bit 6)
startup warning	-	Activates the output of the level warning (Bit 5)
stopwarn	-	Deactivates the output of the level warning (bit 5).
starterr	-	Activates the output of the level error (Bit 4)
stoperr	-	Deactivates the output of the level error (bit 4).
setverbose <verb>	Set Verbose to : <verb> OK Missing Argument. ERR	Sets the information density level (alternative to the start/stop commands.
getverbose	ERROR WARN. INFO. DEBUG (< verb>) X X X X	Returns the current information density level.
setbaud <rate>	Changing Baud Rate to: '<rate>' OK Invalid Baud Rate. ERR Missing Argument. ERR	Changes the baud rate of the module. At reboot the baud rate is always 9600. Valid are the usual values between 300 and 115200.
setblocktime <time>	Minimum time between Requests: '<time>' OK Invalid Time given (valid: 0 - 30). ERR Missing Argument. ERR	Blocking time: Sets the minimum waiting time until a new value can be transmitted again. Valid is 0 to 30 seconds.
getblocktime	Minimum time between Requests: '<time>' OK	Returns the currently set time for the blocking time in seconds.
setresendtime <time>	Time for resending values: '<time>' OK Invalid Time given (valid: 0 - 3600). ERR Missing Argument. ERR	Retry time: Sets the time span when the values are sent to the MQTT broker again without a new value.
getresendtime	Time for resending values: '<time>' OK	Returns the currently set time for the repeat time in seconds.
setssid <ssid>	Set WifiSSID: '<ssid>' OK	Sets the Wifi SSID. If <ssid> is omitted, the SSID is deleted.
getssid	Get WifiSSID: '<ssid>' OK	Returns the currently set SSID.
setpass <pw>	Set WifiPass: '<pw>' OK	Sets the Wifi password. If <pw> is omitted, the password is deleted.
getpass	Get WifiPAss: '<pw>' OK	Returns the currently set Wifi password.
sethost <ip>	Set Host: ' ip>' OK	Sets the URL of the Thingworx server.
gethost	Get Host: '<ip>' OK	Returns the currently set URL of the Thingworx server.
setappkey <name>	Set App Key: '<name>' OK	Sets the application key for accessing the Thing.
getappkey	Get App Key: '<name>' OK	Returns the currently set application key for accessing the Thing.

setthing <thing>	Set Thing: '<thing>' OK	Sets the name of the thing that contains the properties to be changed.
getthing	Get Thing: '<thing>' OK	Returns the currently set Thing which contains the properties to be changed.
setpropname <#> <name>	Set Property Name #<#>: '<name>' OK Invalid Property Number. ERR Missing Argument. ERR	Sets the name of the general property with the number <#>. 50 general properties can be specified, which means <#> must be between 0 and 49.
getpropname <#>	Get Property Name #<#>: '<name>' OK Invalid Property Number. ERR Missing Argument. ERR	Returns the name of the general property with the number <#>.
setpropval <#> < val>	Set Property Value #<#>: '<val>' OK Invalid Property Number. ERR Missing Argument. ERR	Sets a new value for the general property with the number <#>. If the new value is different from the last one transferred, a transfer to the Thingworx server is triggered immediately.
getpropval <#>	Get Property Value #<#>: '<val> ' OK Invalid Property Number. ERR Missing Argument. ERR	Returns the last received value of the specified general property numbered <#>.
setvaluename <#> <name>	Set Value Name #<#>: '<name>' OK Invalid Value Number. ERR Missing Argument. ERR	Sets the name of the value property with the number <#>. 50 value properties can be specified, which means <#> must be between 0 and 49.
getvaluename <#>	Get Value Name #<#>: '<name>' OK Invalid Value Number. ERR Missing Argument. ERR	Returns the name of the value property with the number <#>.
setvalth <#> < th>	Set Value Threshold #<#>: '<th>' OK Invalid Value Number. ERR Missing Argument. ERR	Sets the transmission threshold of the value property with the number <#>. If a new value is received, it is compared with the last value sent. Only if the difference is greater than the set value, the new value is immediately published on the Thingworx server. Default value is 0,1.
getvalth <#>	Get Value Threshold #<#>: '<th>' OK Invalid Value Number. ERR Missing Argument. ERR	Returns the transmission threshold of the value property with the number <#>.
setval <#> < val>	Set Value #<#>: '<val>' OK Invalid Value Number. ERR Missing Argument. ERR	Sets a new value for the value property with the number <#>. If the new value differs from the last transmitted value by more than the set transmission threshold, a transmission to the Thingworx server is triggered immediately.
getval <#>	Get Value #<#>: '<val>' OK Invalid Value Number. ERR Missing Argument. ERR	Returns the last received value of the specified value property with the number <#>.
setboolname <#> <name>	Set Bool Name #<#>: '<name>' OK Invalid Boolean Number. ERR Missing Argument. ERR	Sets the name of the Boolean property with the number <#>. 50 Boolean properties can be specified, which means <#> must be between 0 and 49.
getboolname <#>	Get Bool Name #<#>: '<name>' OK Invalid Boolean Number. ERR Missing Argument. ERR	Returns the name of the Boolean property with the number <#>.
setboolval <#> < val>	Set Bool Value #<#>: '<val>' OK Invalid Boolean Number. ERR Missing Argument. ERR	Sets a new value for the Boolean property with the number <#>. If the new value differs from the last one transferred, a transfer to the Thingworx server is triggered immediately.

getboolval <#>	Get Bool Value #<#>: '<val>' OK	Returns the last received value of the specified Boolean property with the number <#>.
	Invalid Boolean Number. ERR	
	Missing Argument. ERR	

4.5.3 Libraries and their keywords

4.5.3.1 Timeout

Waiting for a certain, freely definable time.

```
#include <Timeout.h>

Timeout to(unsigned long time)
```

Parameters:

`time` Waiting time in ms for the timeout.

Methods:

`to.SetNow()` Starts the timeout at this exact time.

`to.SetTimeout(unsigned long time)` Set a new wait time
`timeNew` wait time in ms

`bool to.TimedOut()` Query whether the waiting time is already over
`true` Waiting time is over
`false` The last call to `SetNow` or `SetTimeout` was not even the waiting time ago.

`unsigned long to.GetTimeoutValue()` Query the waiting time of this timeout

`unsigned long to.GetElapsedTime()` Returns the time since the last call to `SetNow` or `SetTimeout`. Returns at most the waiting time.

`unsigned long to.GetRestTime()` Returns the remaining time of the current timeout. Returns the value 0 as a minimum.

`unsigned long to.GetElapsedPortion()` returns the remaining portion of the running timeout in %. If the timeout has expired, the function returns 100%.

4.5.4 Responding of the ESP MQTT gateway

```
#include <EspMQTTGateway.h>

EspMQTTGateway mqttClient(*SerialIF, *DebugIF, *callback);
```

Parameters:

`*SerialIF` Pointer to a hardware or software serial object. The ESP is connected there.

`*DebugIF` Pointer to a hardware or software serial object. There the library will output messages (debug information to a serial monitor).

`*callback` Pointer to a callback function if a subscribed topic was received.

The callback function must have the following format:

```
void mqttCallback(int topic, String tag, float value)
```

Parameters:

`topic` Number of the received topic.

`tag` Content of the received topic as a string.

`value` Content of the received topic as a number.

Methods:

`mqttClient.Update()` Call the client and execute its code (e.g. check if a new topic was received or check if the ESP is reachable). Must be called once every loop.

`mqttClient.SetBlockTime(float value)` Set the blocking time. Within this time no new value is transmitted

`value` Time of the transmission block in s $[0 - 30]$

`mqttClient.SetResendTime(float value)` Sets the retry time. After this time has elapsed, the current values are transmitted - regardless of whether they have changed or not.

`value` Time of the send repetition in s $[0 - 3600]$

`mqttClient.SetVerbose(byte verbose)` Set the "talk verbose". The higher the number the more output is generated by the library at the set debug interface.

`verbose` Bit 7 (128): Output debug information.
Bit 6 (64): Output of information level
Bit 5 (32): Output of warning level
Bit 4 (16): Output of error level

`mqttClient.SetWifiSsid(ssid)` Set the SSID on the ESP

`ssid` the SSID the ESP should connect to

`bool mqttClient.WifiSsidTranferred()` Read whether the SSID was successfully transferred to the ESP (then `true`).

`mqttClient.SetWifiPass(wifipass)` Set the WiFi password on the ESP
`wifipass` the password of the WiFi the ESP should connect to

`bool mqttClient.WifiPassTranferred()` Read whether the WiFi password was successfully transferred to the ESP (then comes `true`).

`bool mqttClient.WifiConnected()` Read whether the ESP has successfully connected to the WiFi network (then comes `true`).

`mqttClient.SetMqttHost(host)` Set the host (IP address) of the MQTT broker
`host` the host address of the MQTT broker

`bool mqttClient.MqttHostTranferred()` Read whether the host address of the MQTT broker was successfully transferred to the ESP (then comes `true`).

`mqttClient.SetMqttUser(user)` Set the username of the MQTT broker
`user` the username of the MQTT broker

`bool mqttClient.MqttUserTranferred()` Read whether the username of the MQTT broker was successfully transferred to the ESP (then `true`).

`mqttClient.SetMqttPass(pass)` Set the user password of the MQTT broker
`pass` the user password of the MQTT broker

`bool mqttClient.MqttPassTranferred()` Read whether the user password of the MQTT broker was successfully transferred to the ESP (then `true`).

`bool mqttClient. MqttConnected()` Read whether the ESP has successfully connected to the MQTT broker (then comes `true`).

`bool mqttClient.SetCommonName(int num, String name)`
Set the name of the topic with the number `num`.
`num` Number of the topic for which the name is
`name` New name of the topic

`bool mqttClient.DelCommonName(int num)`
Delete the name of the topic with the number `num`.
`num` Number of the topic whose name is to be deleted

`bool mqttClient.SetCommon(int num, String value)`
Set the topic with the number `num`.
`num` Number of the topic for which the new value is
`value` New value of the topic

`bool mqttClient.SetBoolName(int num, String name)`
Set the name of the boolean value with the number `num`.
`num` number of the boolean value for which the name is
`name` New name of the boolean value

`bool mqttClient.DelBoolName(int num)`
Deletes the name of the boolean value with the number `num`.
`num` number of the boolean whose name is to be deleted

`bool mqttClient.SetBool(int num, bool value)`
Set the bool value with the number `num`.

num	number of the boolean value for which the new value is
value	New value of the boolean value

`bool mqttClient.SetValueName(int num, String name)`

Set the name of the value with the number `num`.

num	Number of the value for which the name is
name	New name of the value

`bool mqttClient.DelValueName(int num)`

Delete the name of the value with the number `num`.

num	Number of the value whose name is to be deleted

`bool mqttClient.SetValue(int num, float value[, int commas])`

Set the value with the number `num`.

num	number of the value for which the new value is
value	New value of the value
commas	Transmission accuracy of the value (default 2)

`bool mqttClient.SetValueThreshold(int num, float value[, int commas])`

Set the transmission threshold of the value with number `num`.

num	number of the value for the transmission threshold
value	value of the transmission threshold
commas	Transmission accuracy of the threshold (default 2)

`bool mqttClient.Subsribe(int num, String name)`

Set the topic to be subscribed to with the number `num`.

num	Number of the subscription for which the name is
name	Topic which should be subscribed

`bool mqttClient.UnSubscribe(int num)`

Unsubscribe the topic with the number `num`.

num	number of the topic that is no longer subscribed

4.5.4.1 Addressing the ESP Thingworx gateway

```
#include < EspThingworxGateway.h>

EspThingworxGateway twxClient(*SerialIF, *DebugIF);
```

Parameters:

`*SerialIF` Pointer to a hardware or software serial object. The ESP is connected there.

`*DebugIF` Pointer to a hardware or software serial object. There the library will output messages (debug information to a serial monitor).

Methods:

`twxClient.Update()` Call the client and execute its code (e.g. check if a new topic was received or if the ESP is reachable). Must be called once every loop run.

`twxClient.SetBlockTime(float value)` Set the blocking time. Within this time no new value is transmitted
`valueTime of` the transmission block in s $[0 - 30]$

`twxClient.SetResentTime(float value)` Sets the retry time. After this time has expired, the current values are transmitted - regardless of whether they have changed or not.
`valueTime of` the transmission block in s $[0 - 3600]$

`twxClient.SetVerbose(byte verbose)` Set the "verboseness". The higher the number the more output is generated by the library at the set debug interface.
`verbose` Bit 7 set: Output debug information.
Bit 6 set: Output of information level
Bit 5 set: Output of warning level
Bit 4 set: Output of error level

`twxClient.SetWifiSsid(ssid)` Set the SSID on the ESP
`ssid` the SSID the ESP should connect to

`bool twxClient.WifiSsidTranfered()` Read whether the SSID was successfully transferred to the ESP (then `true`).

`twxClient.SetWifiPass(wifipass)` Set the WiFi password on the ESP
`wifipass` the password of the WiFi the ESP should connect to

`bool twxClient.WifiPassTranfered()` Read whether the WiFi password was successfully transferred to the ESP (then comes `true`).

`bool twxClient.WifiConnected()` Read whether the ESP has successfully connected to the WiFi network (then comes `true`).

`twxClient.SetThingworxHost(host)` Set the host (IP address) of the Thingworx server
`host` the host address of the Thingworx server

`bool twxClient.ThingworxHostTranfered()` Read whether the host address of the Thingwork server was successfully transferred to the ESP (then `true`).

`twxClient.SetAppkey (key)` Set the application key of the thing
`key` the application key of the thing

`bool twxClient.AppkeyTranfered()` Read whether the application key of the ting was successfully transferred to the ESP (then `true`).

`twxClient.SetThing(thing)` Set the thing name
`thing` the name of the thing

`bool twxClient.ThingTranfered()` Read whether the name of the thing was successfully transferred to the ESP (then `true`).

`bool twxClient.ThingworxConnected()` Read whether the ESP has successfully connected to the Thingworx server (then `true`). This can only be updated on a send attempt, so it may be that the Thingworx server is already unreachable. At the next unsuccessful send attempt the value is set to `fasle` (and of course to `true` if the transmission was successful again).

`bool twxClient.SetCommonName(int num, String name)`
Set the name of the property with the number `num`.
`num` Number of the property for which the name is
`name` New name of the property

`bool twxClient.DelCommonName(int num)`
Deletes the name of the property with the number `num`.
`num` Number of the property whose name is to be deleted

`bool twxClient.SetCommon(int num, String value)`
Set the property with the number `num`.
`num` Number of the property for which the new value is
`value` New value of the topic

`bool twxClient.SetBoolName(int num, String name)`
Set the name of the boolean value with the number `num`.
`num` number of the boolean value for which the name is
`name` New name of the boolean value

`bool twxClient.DelBoolName(int num)`
Deletes the name of the boolean value with the number `num`.
`num` number of the boolean whose name is to be deleted

`bool twxClient.SetBool(int num, bool value)`
Set the boolean value with the number `num`.
`num` number of the boolean value for which the new value is
`value` New value of the boolean value

`bool twxClient.SetValueName(int num, String name)`

Set the name of the value with the number `num`.

`num`	Number of the value for which the name is
`name`	New name of the value

`bool twxClient.DelValueName(int num)`

Deletes the name of the value with the number `num`.

`num`	Number of the value whose name is to be deleted

`bool twxClient.SetValue(int num, float value[, int commas])`

Set the value with the number `num`.

`num`	Number of the value for which the new value is to be set.
`value`	New value of the value
`commas`	Transmission accuracy of the value (default `2`)

`bool twxClient.SetValueThreshold(int num, float value[, int commas])`

Set the transmission threshold of the value with the number `num`.

`num`	Number of the value for the transmission threshold
`value`	value of the transmission threshold
`commas`	Transmission accuracy of the threshold (default `2`)

4.6 HTTP status codes

An HTTP status code is delivered by a server in response to every HTTP request. The requesting side is a client such as a web browser. The server uses the HTTP status code to inform the client whether the request was processed successfully.

	Code	Message	Meaning
1xx - Info	100	Continue	The current request to the server has not yet been rejected. The client can now proceed with the potentially very large request.
	101	Switching Protocols	Used when the server agrees to switch to another protocol. E.g. change from HTTP to WebSocket.
	102	Processing	Used to avoid a timeout while the server is processing a time-intensive request.
2xx - Successful Operation	200	OK	The request was successfully processed and the result of the request is transmitted in the response.
	201	Created	The request was processed successfully. The requested resource was created before the response was sent.
	202	Accepted	The request has been accepted but will be executed at a later date. The success of the request cannot be guaranteed.
	203	Non-authoritative information	The server acts as a "transforming proxy", received a 200 OK response from the source and responds with a modified document from the source.
	204	No Content	The request was successful, but the response intentionally does not contain any data.
	205	Reset Content	The request was successful; the client should rebuild the document and reset form entries.
	206	partial content	The requested part was successfully transferred. Can inform a client about partial downloads).
	207	Multi-status	The response contains an XML document that contains several status codes for independently performed operations.
	208	Already Reported	The members of a WebDAV binding have been enumerated previously and are not present in this request.
	226	IM Used	The server has fulfilled a GET request, is a representation of the result of an instance manipulation, related to the current instance.
3xx - Redirect	300	multiple choices	The requested resource is available in different types. The response contains a list of the available types.
	301	Moved Permanently	The requested resource is now available at the address specified in the "Location" header field (also called a redirect).
	302	Found (Moved Temporarily)	The requested resource is temporarily available at the address specified in the "Location" header field. The old address remains valid.
	303	See Other	The response to the executed request can be obtained at the address specified in the "Location" header field.
	304	Not Modified	The content of the requested resource has not changed since the last client request and is therefore not transferred.
	305	Use Proxy	The requested resource is only accessible via a proxy. The "Location" header field contains the address of the proxy.
	306	(reserved)	306 is no longer used, but is reserved.
	307	Temporary Redirect	The requested resource is temporarily available at the address specified in the "Location" header field. The old address remains valid.
	308	Permanent Redirect	The requested resource is now available at the address specified in the "Location" header field, the old address is no longer valid.

4xx - Client Error	400	bad request	The request message was constructed incorrectly.
	401	Unauthorized	The request cannot be performed without valid authentication.
	402	Payment Required	This status is reserved for future HTTP protocols.
	403	Forbidden	The request was not executed due to a lack of authorization of the client, e.g. because a URL configured as HTTPS was only called with HTTP.
	404	Not Found	The requested resource was not found. Can also be used to reject a request without further reason.
	405	Method Not Allowed	The request may only be made using other HTTP methods (for example, GET instead of POST).
	406	Not Acceptable	The requested resource is not available in the requested form.
	407	Proxy Authen. Required	Analogous to the status code 401, an authentication of the client against the used proxy is required here first.
	408	request timeout	No complete request received within the allowed time period.
	409	Conflict	The request was made under false assumptions.
	410	Gone	The requested resource is no longer provided.
	411	Length Required	The request cannot be processed without a "Content-Length".
	412	Precondition Failed	A condition transmitted in the request, for example in the form of an "if-match" header field, did not apply.
	413	Request Entity Too Large	The request made was too large for the server to process. A "Retry-After" header field in the response can indicate to the client that the request might be processed at a later time.
	414	UTI Too Long	The URL of the request was too long. The cause is often an endless loop of redirects.
	415	Unsupported Media Type	The content of the request was transmitted with invalid or not allowed media type.
	416	Requested Range Not Satisfiable	The requested part of a resource was invalid or is not available on the server.
	417	Expectation Failed	Used in conjunction with an Expect header field. The server behavior requested in the "Expect" header field cannot be fulfilled.
	420	Policy Not Fullfilled	In W3C PEP (Working Draft 21 November 1997) this code is proposed to indicate that a condition has not been met.
	421	misdirected request	The request was sent to a server that is unable to send a response. Introduced in HTTP/2.
	422	Unprocessable Entity	Used when neither the return of status code 415 nor 400 would be justified, but processing of the request is rejected due to semantic errors, for example.
	423	Locked	The requested resource is currently locked.
	424	Failed Dependency	The request could not be performed because it requires the success of a previous request.
	426	Upgrade Required	The server requires the client to repeat the request using a different protocol. One use case is switching to HTTP with Transport Layer Security.
	428	Precondition Required	Not all preconditions were met for the request. This status code is intended to prevent problems caused by race conditions, in that manipulation or deletion only takes place if the client requests this on the basis of a current resource (for example, by supplying a current ETag header).
	429	Too Many Requests	The client has sent too many requests in a given time period.
	431	Request Header Fields Too Large	The maximum length of a header field or the total header was exceeded.
	451	Unavailable For Legal Reasons	This status code is to indicate that the requested resource is not available due to legal restrictions (copyright restrictions, censorship, etc., possibly limited to a certain country). It was submitted to the IETF in June 2012 by Google employee Tim Bray and has been considered accepted since December 17, 2015. Bray wrote at the end of his proposal "Thanks also to Ray Bradbury". Alluding to his novel Fahrenheit 451, Bray proposed the number 451 for the status code.

5xx - Server Error	**500**	Internal Server Error	This is a "collection status code" for unexpected server errors.
	501	Not Implemented	The functionality to process the request is not provided by this server. The cause is, for example, an unknown or unsupported HTTP method.
	502	Bad Gateway	The server could not fulfill its function as gateway or proxy because it received an invalid response on its part.
	503	Service Unavailable	The server is temporarily unavailable, for example due to overload or maintenance. A "Retry-After" header field in the response can alert the client to a time when the request might be processed.
	504	gateway timeout	The server was unable to perform its function as a gateway or proxy because it did not receive a response from servers or services it was using within a specified period of time.
	505	HTTP Version Not Supported	The HTTP version used (meaning the number before the dot) is not supported or rejected by the server.
	506	Variant Also Negotiates	The content agreement of the request results in a circular reference.
	507	Insufficient Storage	The request could not be processed because the server's memory is currently insufficient for this purpose.
	508	Loop Detected	The operation was not executed because the execution would have run into an infinite loop. Defined in the binding extension for WebDAV according to RFC 5842, because bindings can create cyclic paths to WebDAV resources.
	509	Bandwidth Limit Exceeded	The request was discarded because otherwise the available bandwidth would be exceeded (unofficial extension of some servers).
	510	Not Extended	The request does not contain all the information that the requested server extension necessarily expects.
	511	Network Authentication Required	The client must first authenticate to gain access to the network.