# *HoloLens*
# *Vuforia Studio*
# *Creo Illustrate*

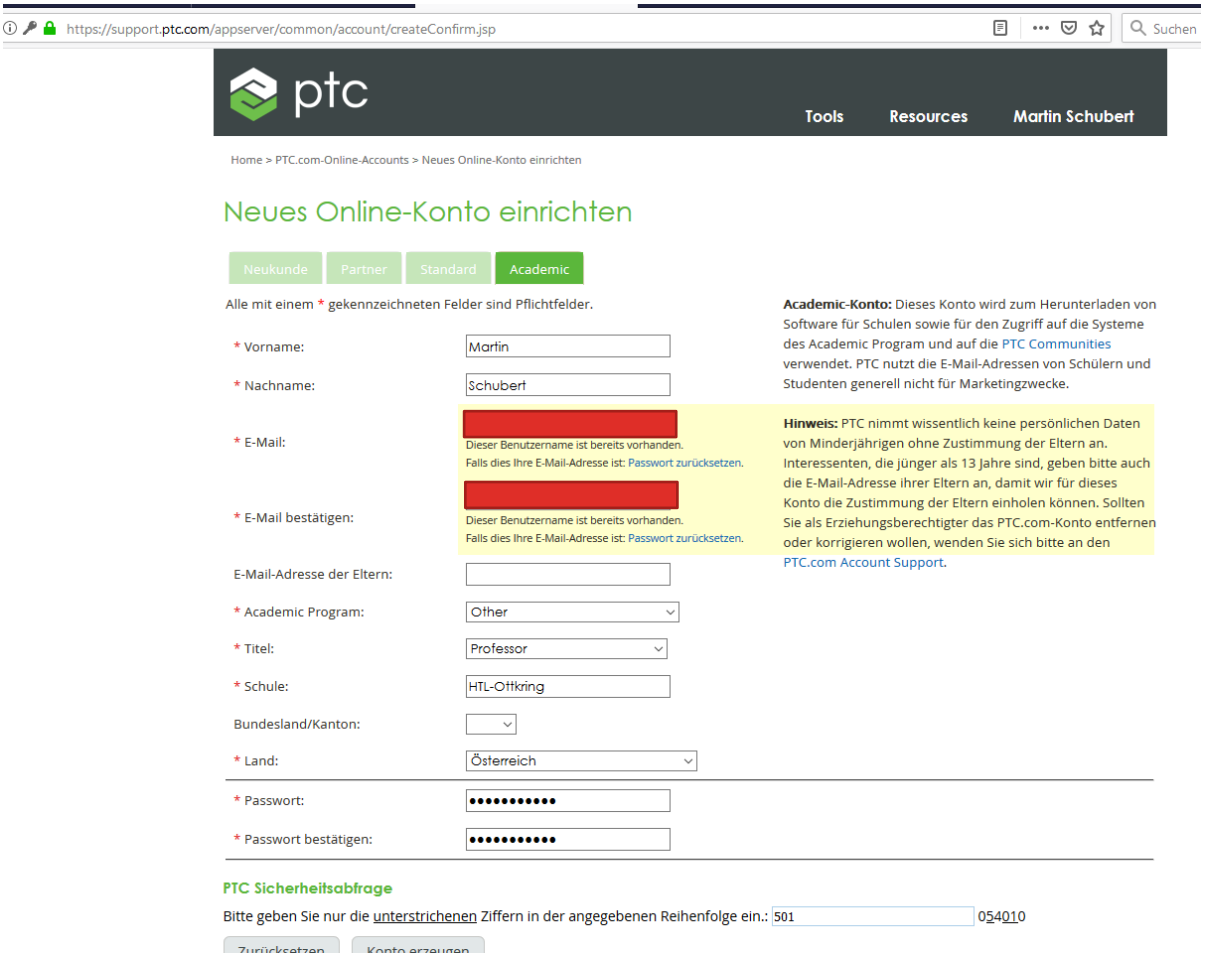
P.Fürli, M.Schubert

# Inhalt

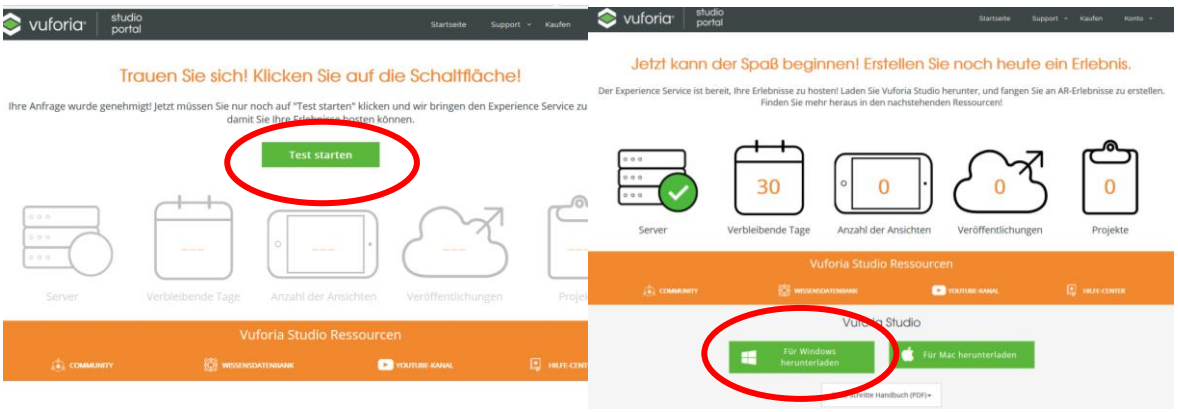
# Vorbereitung

## Installation von Vuforia Studio

Anlegen eines PTC-Accounts:




Download und Installation der Software via https://studio.vuforia.com

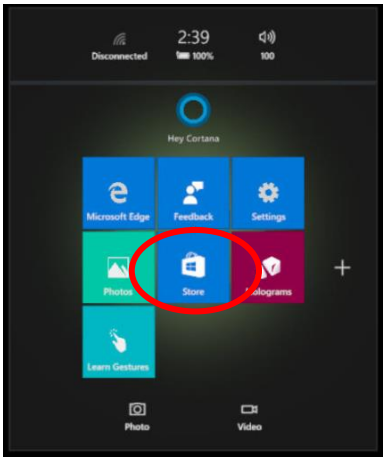


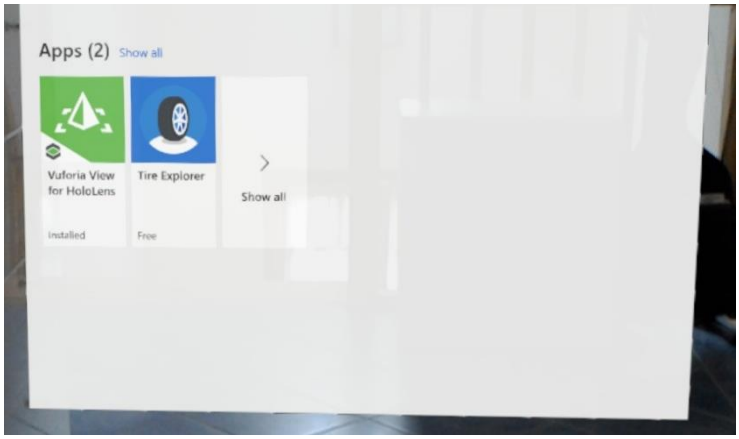
## Einrichten der Hololens

Die Betriebssystemsoftware sollte auf dem neuesten Stand sein.

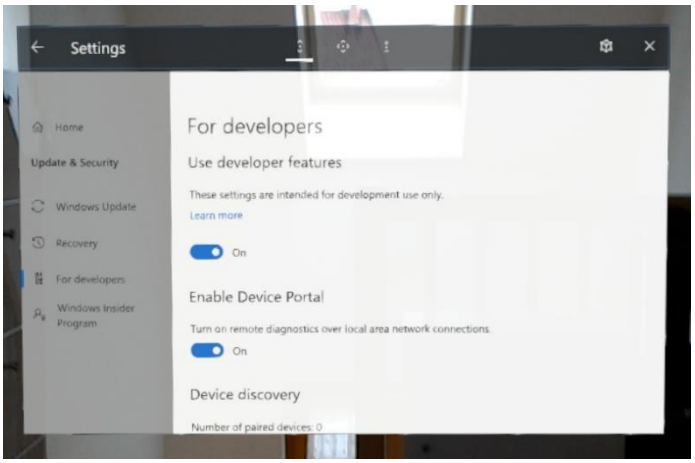
Die Hololens muss bereits mit einem WLAN mit Internetanbindung verbunden sein.

im Store nach Vuforia suchen und die App installieren

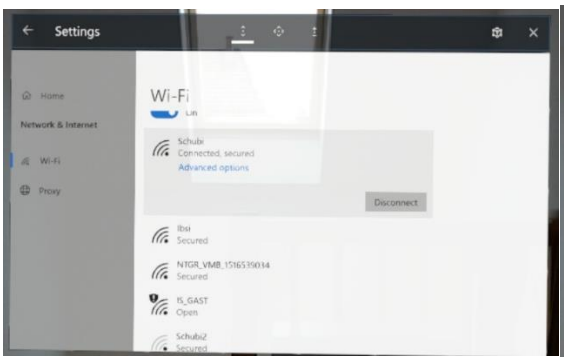


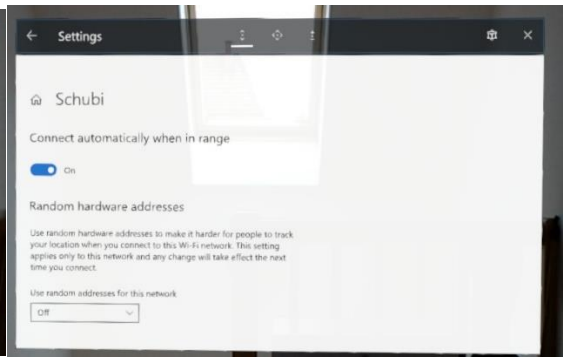




Nach erfolgreicher Installation unter Settings den Eintrag For developers anwählen und *developer features* und *device portal* aktivieren:



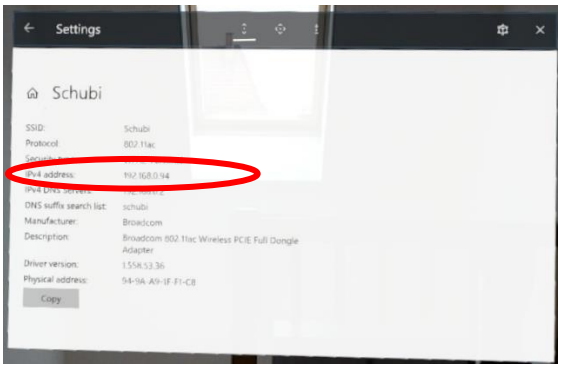

Ermittlung der IP-Adresse (dieser Schritt ist für die Verwendung mit Vuforia Studio nicht erforderlich)



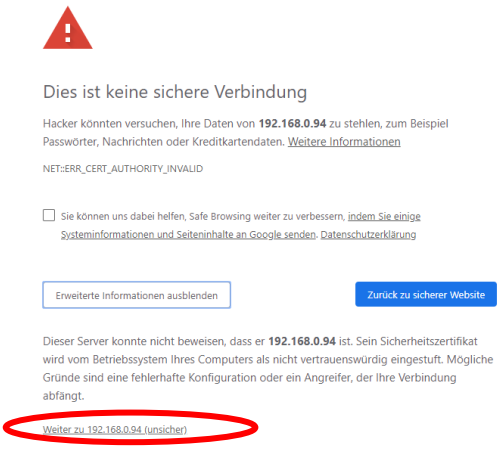




Beim aktiven Netzwerk *Advanced options* anklicken und im sich öffnenden Fenster nach unten scrollen.

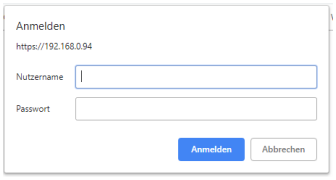
Hier findet man die IP-Adresse der HoloLens:



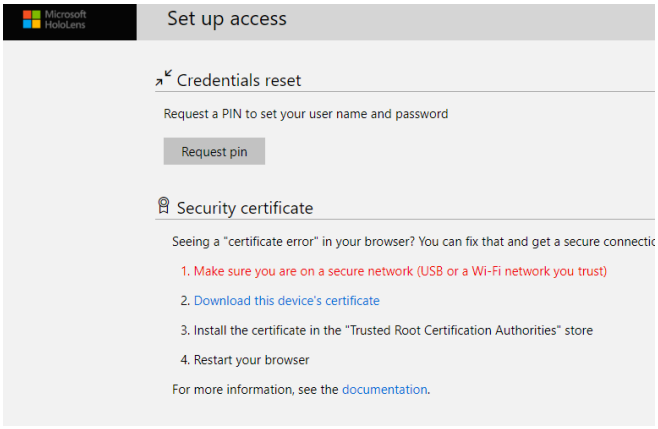

Nun kann man auf einem Rechner (der im selben Netzwerk sein muss) diese IP in den Browser eintragen. Warnungen bezüglich der Sicherheit ignorieren – bzw. bestätigen, dass es OK ist.



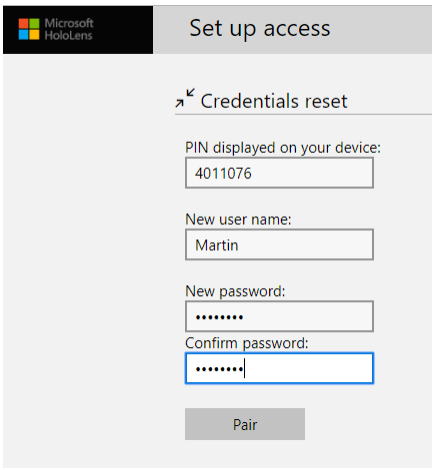

Nun werden ein Name und Passwort abgefragt. Wenn man das nicht weiß, dann mindestens drei Mal Name und Passwort leer lassen und Anmelden drücken.



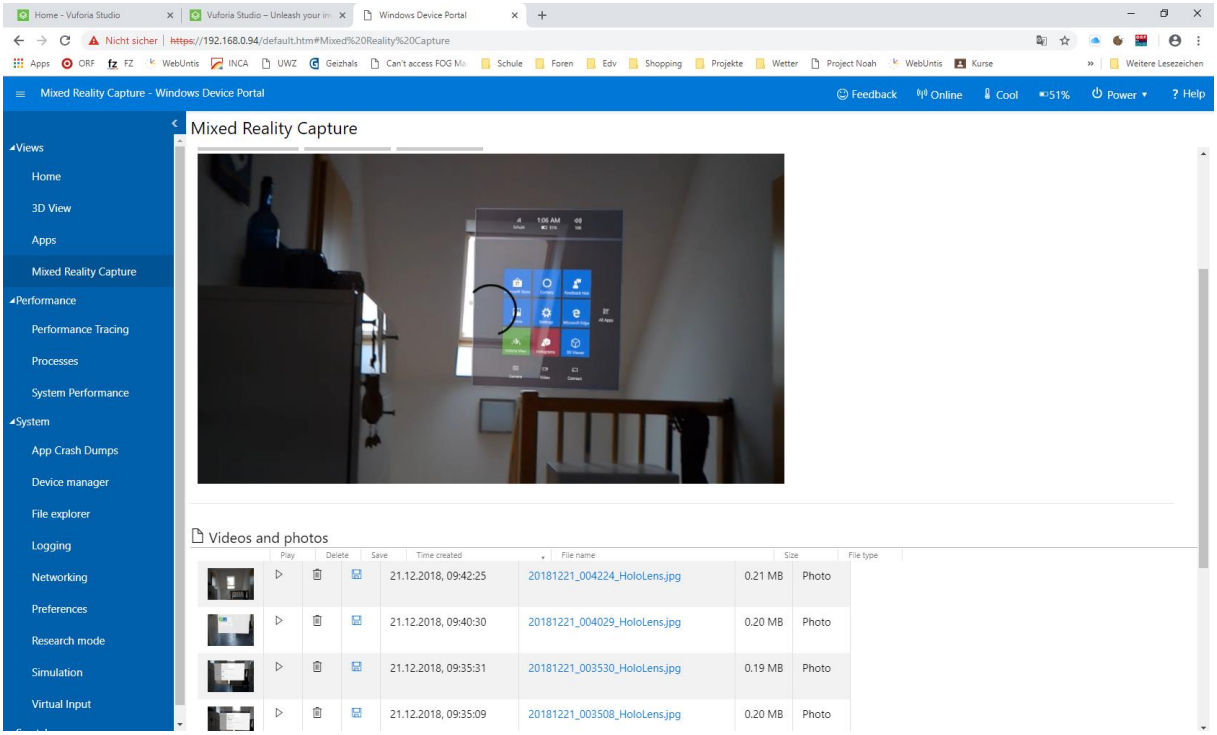

… es öffnet sich diese Seite




Request PIN anklicken und den in der HoloLens angezeigten Pin im folgenden Fenster eintragen:




Nun kann man sich mit dem angegeben Namen und Passwort in das WEBIF der HoloLens einloggen. Hier kann man zum Beispiel den Livestream im Browser anzeigen lassen, oder auch Screenshots herunterladen:

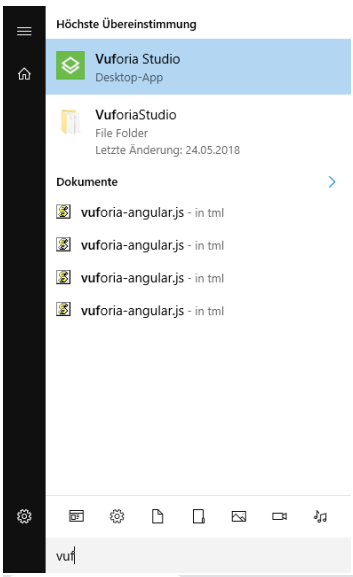


# Arbeiten mit Vuforia Studio

## Erstes Projekt „Wagenheber“

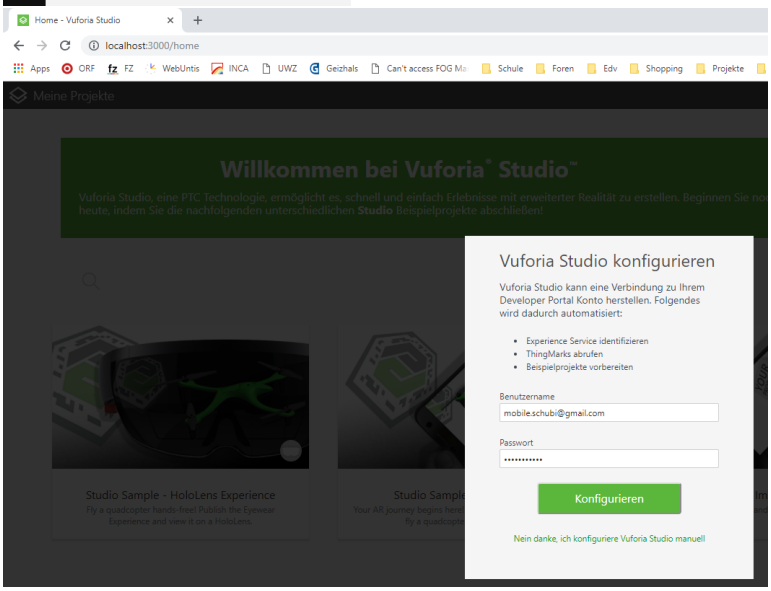
### Grundlagen

Vuforia Studio läuft als Webapplikation am lokalen Rechner und wird daher in einem Browser dargestellt. Es ist ratsam **ausschließlich Google Chrome zu verwenden!**



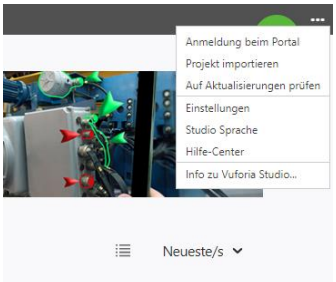

Nach dem Start öffnet sich dieses Fenster:

Durch Anklicken von Öffnen startet der Standardbrowser des Systems. Wenn das nicht Google Chrome ist, dann den Link aus der Adressleiste kopieren und Chrome einfügen!

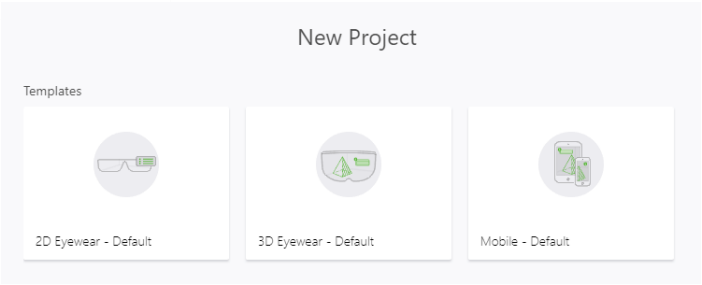




Hier mit dem PTC-Account anmelden.

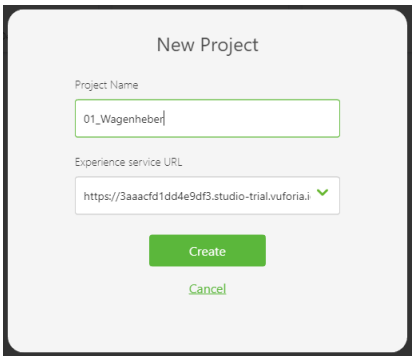
Die Willkommensseite öffnet sich – nun die Sprache auf Englisch umstellen:

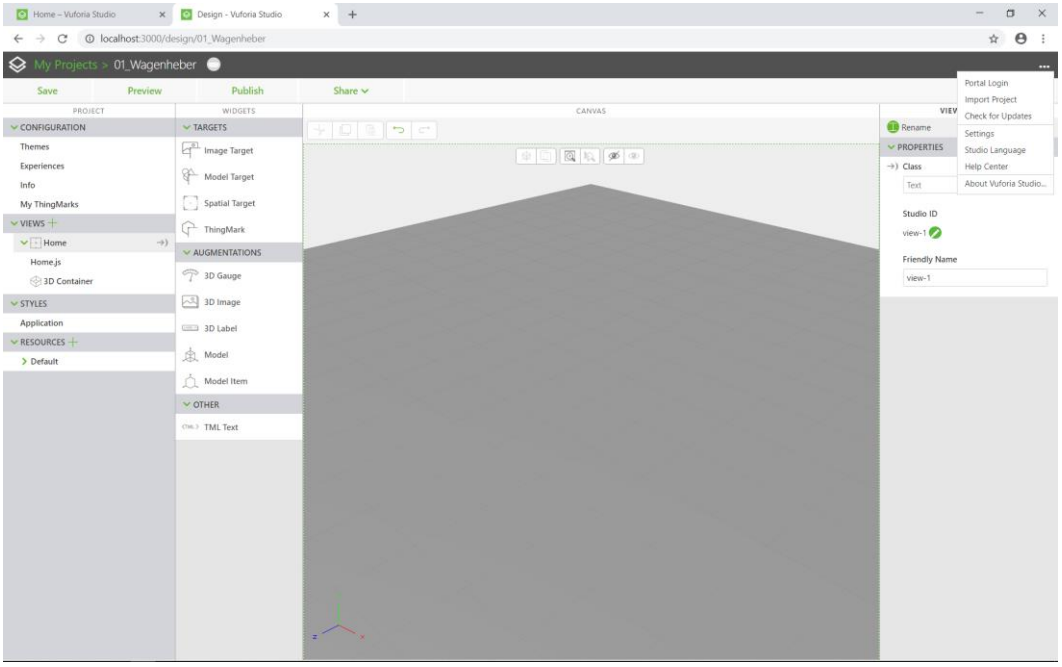


Mit dem Plus-Symbol rechts oben ein neues Projekt erstellen:




… für die HoloLens 3D Eyeware auswählen.

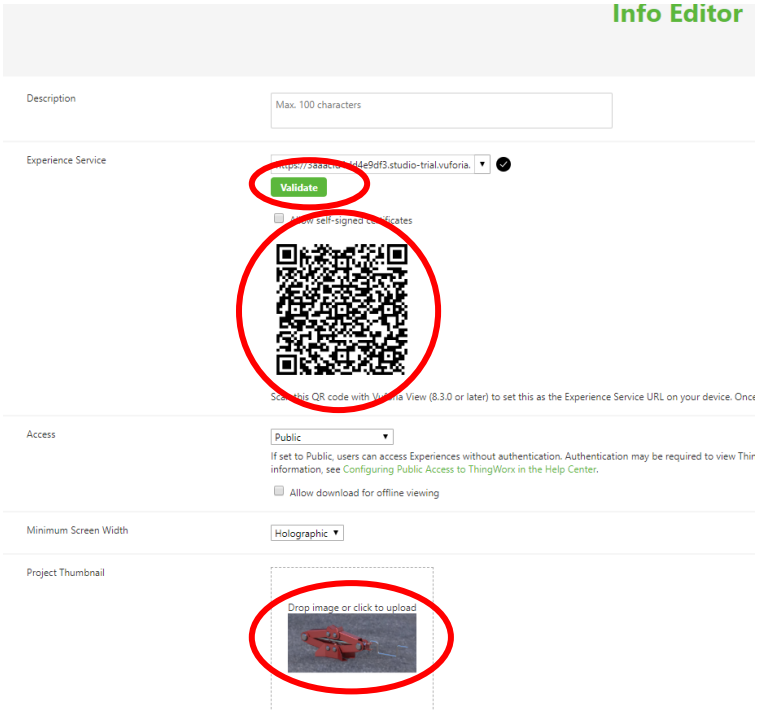



Viele Funktionen in der Oberfläche lassen sich per „drag and drop“ ausführen.

Zuerst überprüfen ob wir mit dem richtigen Experience Service arbeiten:

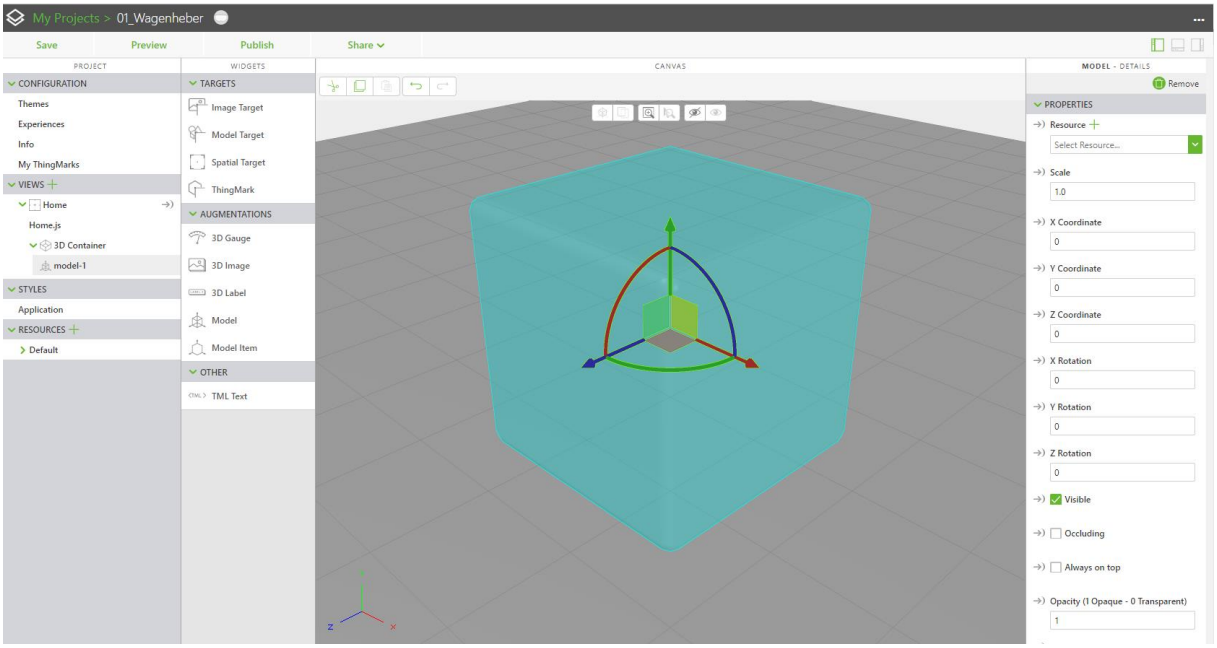
Project→Info:




- Mittels Validate den Status überprüfen (Anmeldung mit PTC-Account erforderlich) – gegeben Falls richtigen Server auswählen.
- Der Angezeigte QR-Code enthält den Link zum Server, der von der View-App auf der HoloLens benötigt wird!
- Hier kann auch gleich ein Vorschaubild für das Projekt hochgeladen werden.
- Access sollte auf Public stehen.

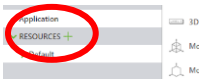
Model hinzufügen – mittels „drag&drop“:

- Das Model ist vorerst nur ein Platzhalter – bzw. Hülle. Es muss erst mit Daten „befüllt“ werden:



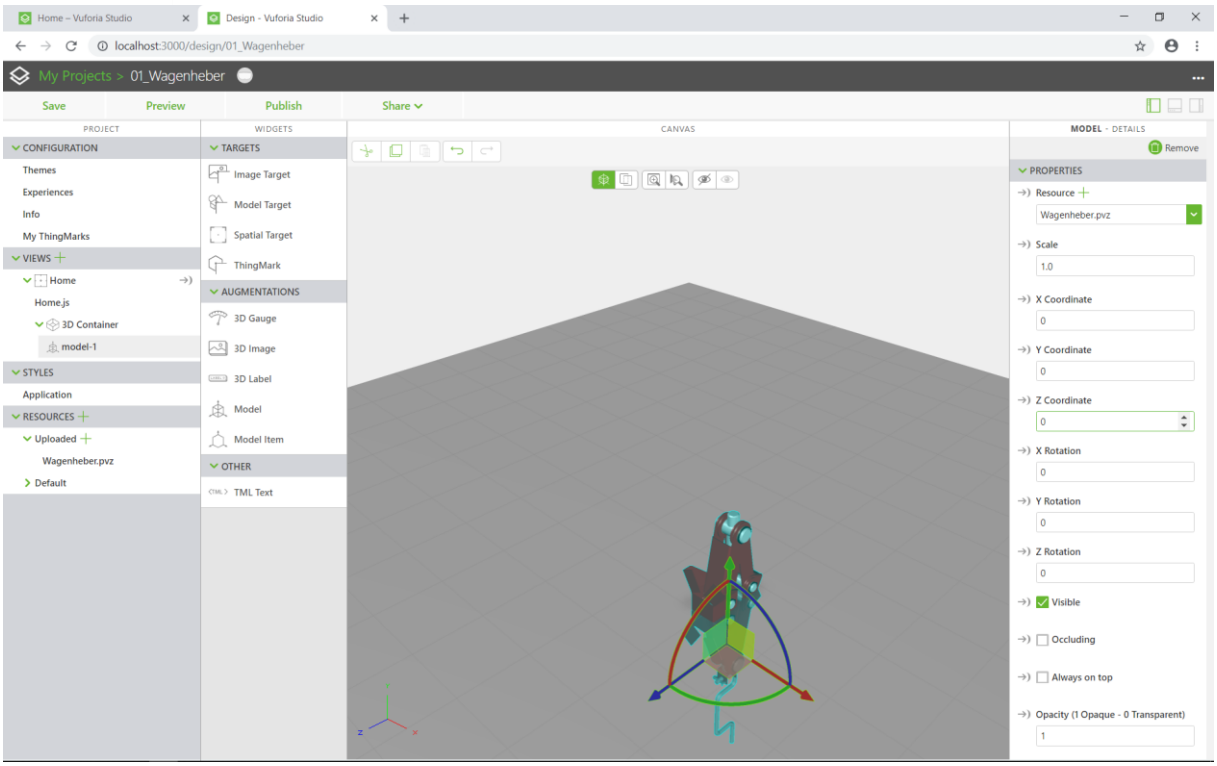

Die eingespielten Daten (3D-Daten, Bilder, …) werden über Resources hinzugefügt. Dies kann entweder vor der Erstellung des Model geschehen (linker Menübereich) oder während der Model Generierung (rechts oben)

Add Resources:



PVZ hinzufügen…


CAD Optimizer für große Dateien – es werden 4 Größen erzeugt.

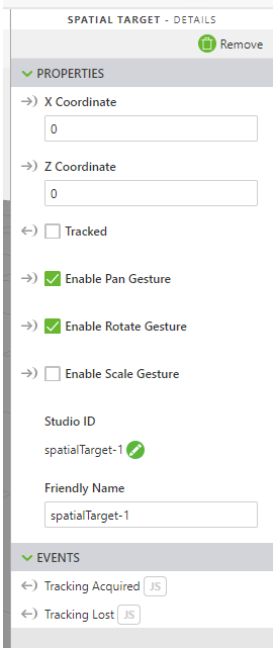



Es wird immer das oberste Koordinatensystem in der Baugruppe für die Einbaubedingungen verwendet.
Die X, Y, Z Koordinaten sind in Meter angegeben.

Zurzeit gibt es Probleme, wenn die Baugruppe Unterbaugruppen enthält. Es ist sinnvoll die Baugruppen in Einzelteile zu verschmelzen.

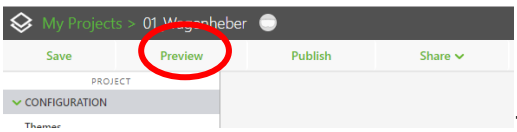
Nun muss man noch ein Target einfügen (mittels drag&drop) – wir verwenden vorerst „Spatial Target“, dies lässt ein freies Platzieren der AR im Raum zu.

Die gewünschten Properties im rechten Bereich einstellen:




Vor dem Publizieren der AR kann man noch unter Preview die Funktion im Browser testen:

Letzter Schritt ist das veröffentlichen der AR mittels Publish:

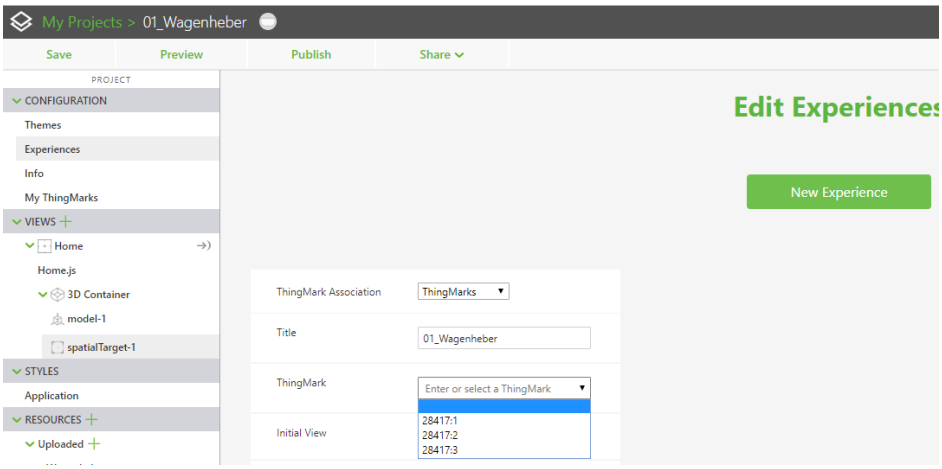



→

ACHTUNG Popups zulassen:

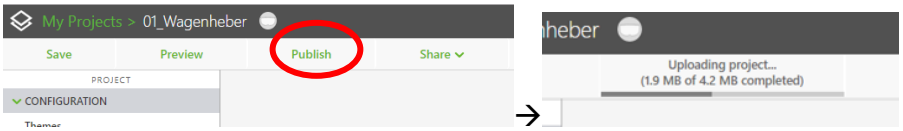
**OPTIONAL:**

Damit die AR in der HoloLens auch ohne Sprachsteuerung aufgerufen werden kann ist es gut unter Configuration→ Experiences noch ein Thingmark anzugeben:




Wenn ein Thingmark angegeben wurde wird diese AR nicht in der library angezeigt, sondern kann nur durch Scannen das Thingmarks geladen werden!

Letzter Schritt ist das veröffentlichen der AR mittels Publish:




Nachdem der Upload abgeschlossen wurde kann man in der HoloLens die Vuforia View App aufrufen.

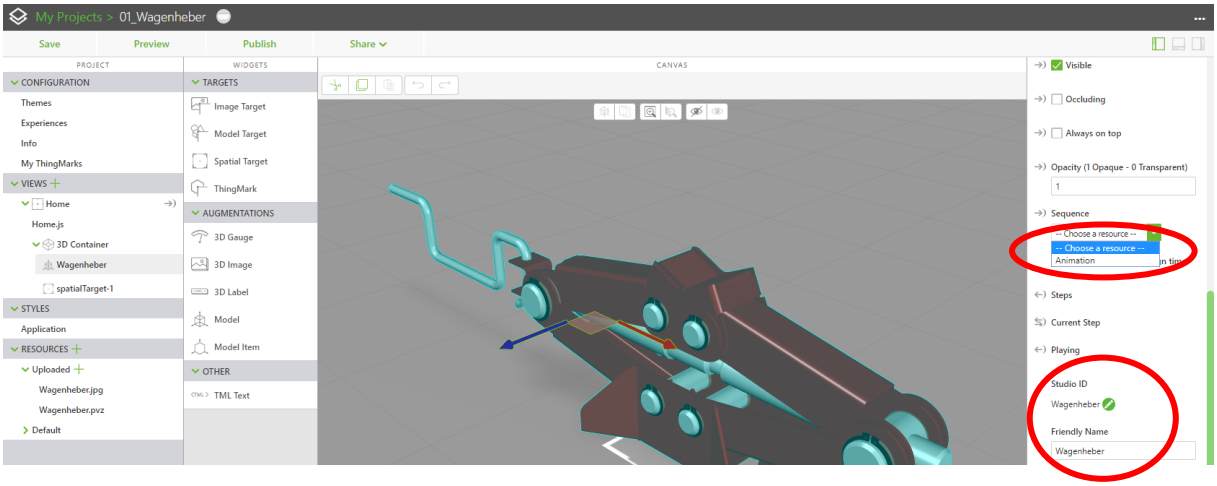
Erster Schritt ist dabei das Scannen des QR-Codes (enthält die Server-URL)

Nun kann mit den Sprachkommandos „Hey View“ … warten auf Signalton …“Show library“ und der Auswahl der gewünschten AR das Teil am Boden platzieren.

## Animation hinzufügen

Dafür muss im pvz-File bereits eine Animation vorhanden sein.

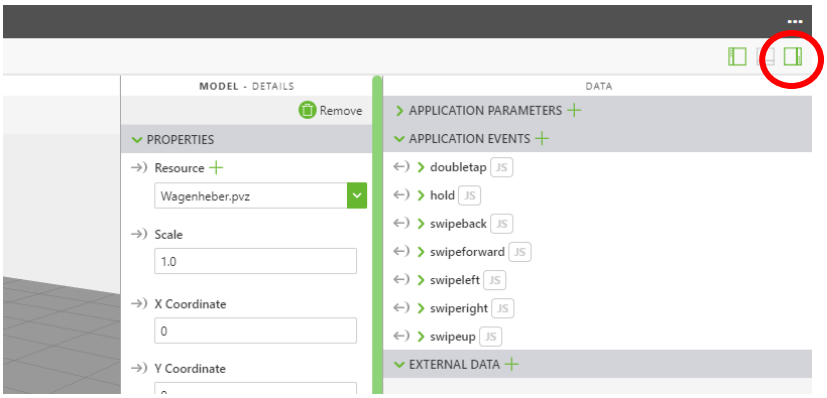
Im 3D-Container das entsprechende Model anwählen und bei den Properties (rechts) nach unten scrollen und bei Sequence die Animation auswählen:
ACHTUNG – Programm genehmigt sich eine kurze Watezeit 😊



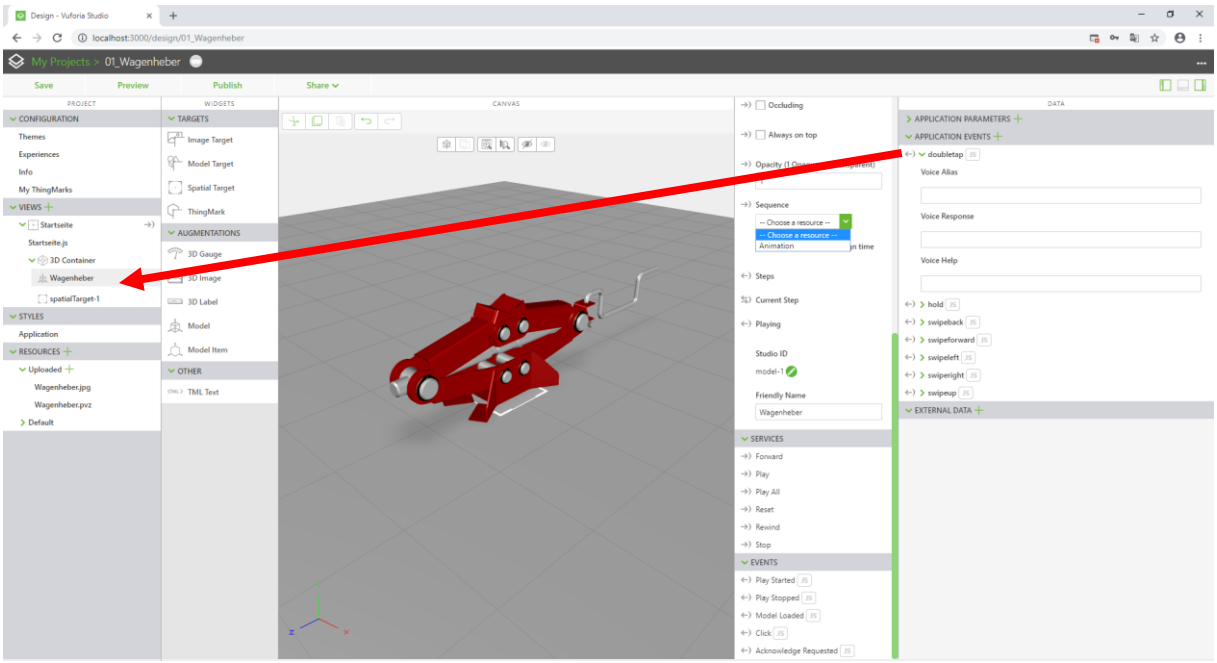

Gleichzeitig kann man auch noch die Einträge Studio ID und Friendly Name anpassen (das ist nicht zwingend erforderlich)

Die Animation soll mit einem Doppelklick gestartet werden und mit Swiperight wieder in den Ausgangszustand gebracht werden.

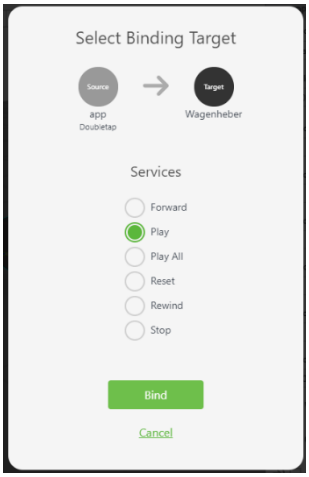
Dazu das Menü Data einblenden:



Dann den doubletab nach Wagenheber ziehen:

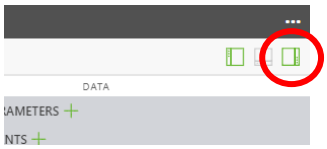


Nun das Binding auf Play setzen:

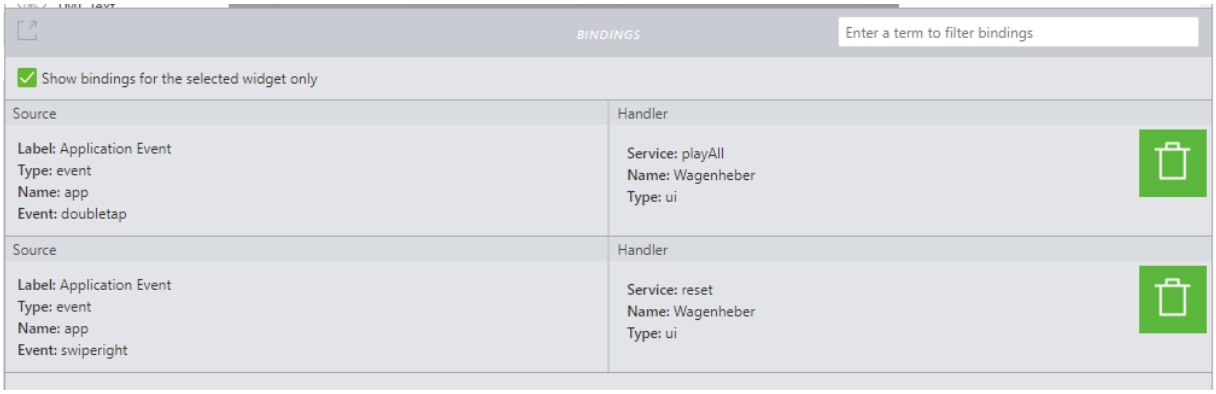


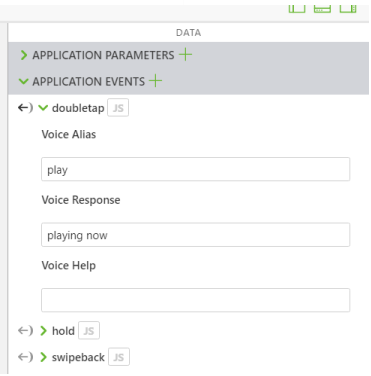

Selben Vorgang noch für swiperight mit dem Binding Reset ausführen.

Die vorhanden Bindings im Projekt können im unteren Fensterbereich angezeigt werden:




Folgende Bindings sollten nun vorhanden sein:





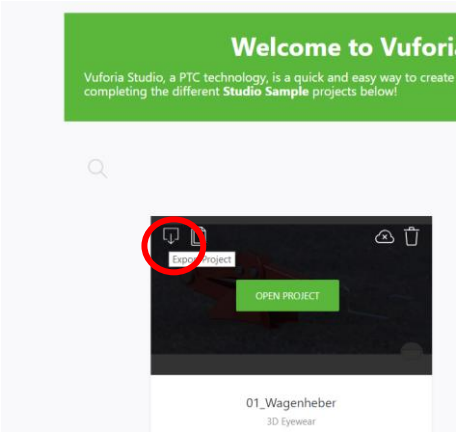

Mit den Einstellungen bei Voice kann man noch einen Sprachbefehl bzw. eine Sprachausgabe festlegen.

Also z.B. Sprachbefehl bei doubletap "play" und bei swiperight "reset".

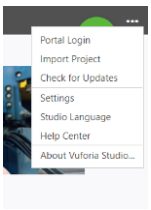
Das Projekt wie gewohnt sichern bzw. publizieren.

## Export und Import

Für die spätere Verwendung kann das gesamte Projekt aus der Übersichtsseite heraus exportiert werden:




Das so erstellte Zip-File kann auch wieder importiert werden:

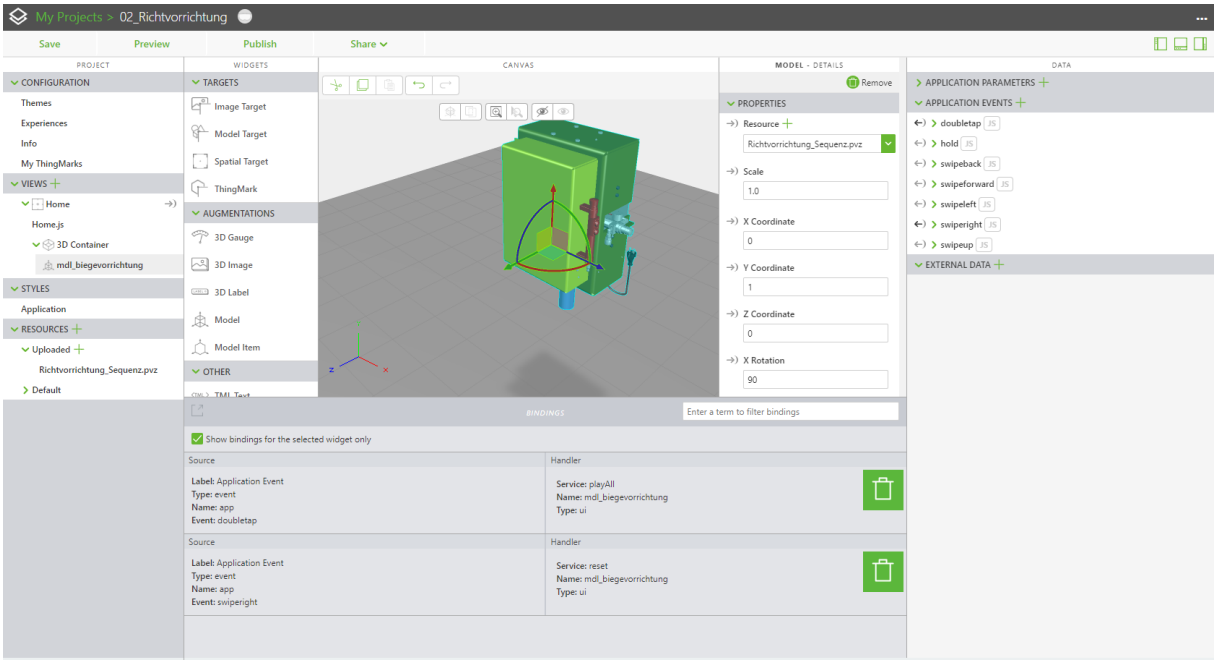


# Projekt „Richtvorrichtung“

## Vorbereitungen

Das vorbereitet pvz-File wieder wie im 1. Beispiel einfügen und die Animation wieder wie vorher einstellen.

Das Ergebnis sollte so aussehen:




**Wichtig!**

Immer zwischendurch unter Preview die Funktion kontrollieren. Wenn es hier nicht funktioniert, dann geht es auch sicher nicht mit der Hololens!

Für die weiteren Schritte gibt es mehrere Ansätze:

1.) Anlegen von Application Parameters
2.) Direktes Ansprechen der Objekte in JS

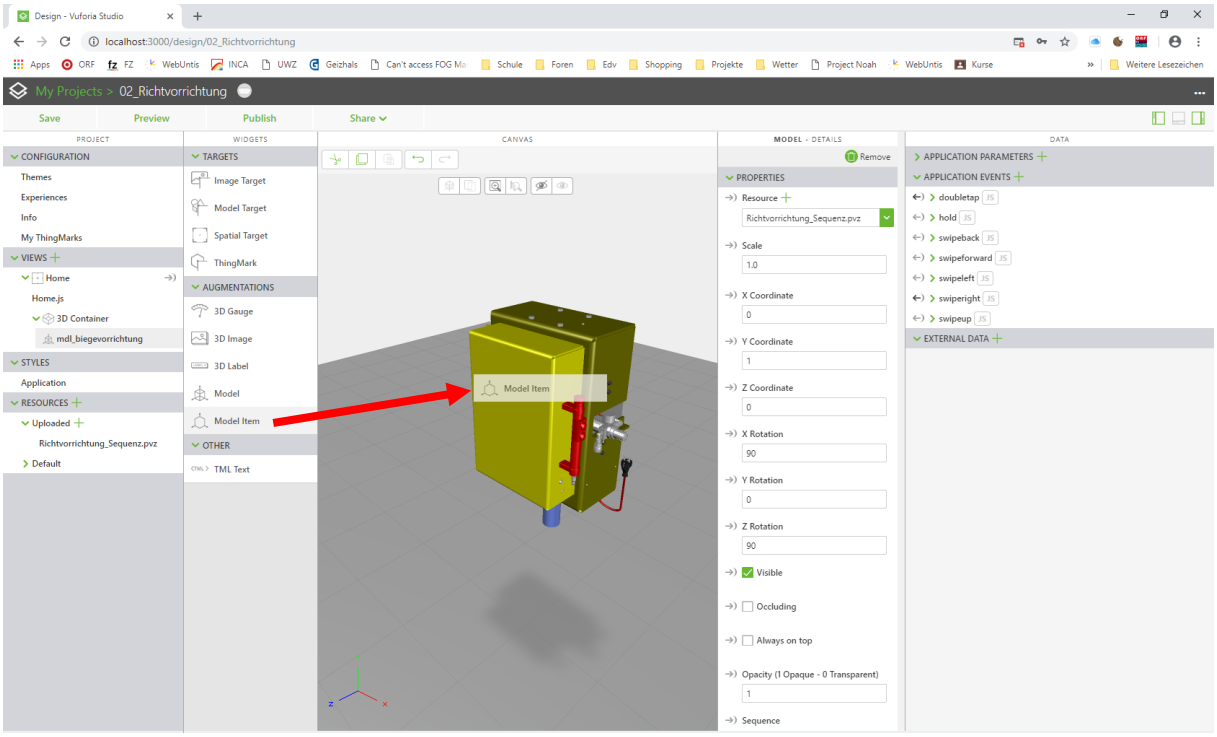
## Objekteigenschaften ändern

Damit ist es möglich die Eigenschaften einzelne Elemente während der Simulation gezielt zu verändern. Dazu gehören z.B. Verschiebung, Transparenz, Rotation,…

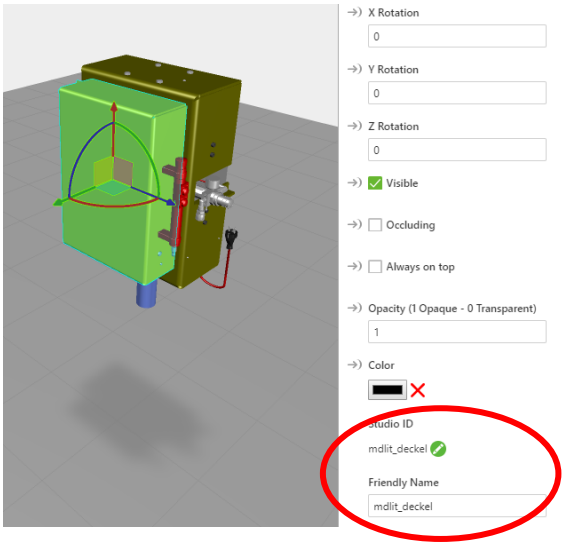
Durch den Einsatz von JavaScript (JS) ist es erreichbar beliebige Abhängigkeiten zu schaffen.

Über sogenannte Model-Items kann man einzelne Bauteile „ansprechen“

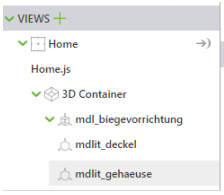
Dazu wird ein Modellitem auf das gewünschte Bauteil gezogen:



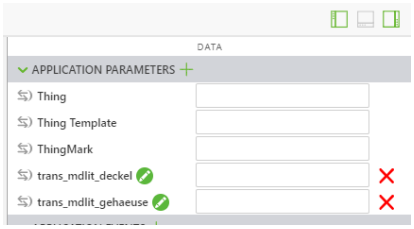

Sofort mit einem Namen versehen:



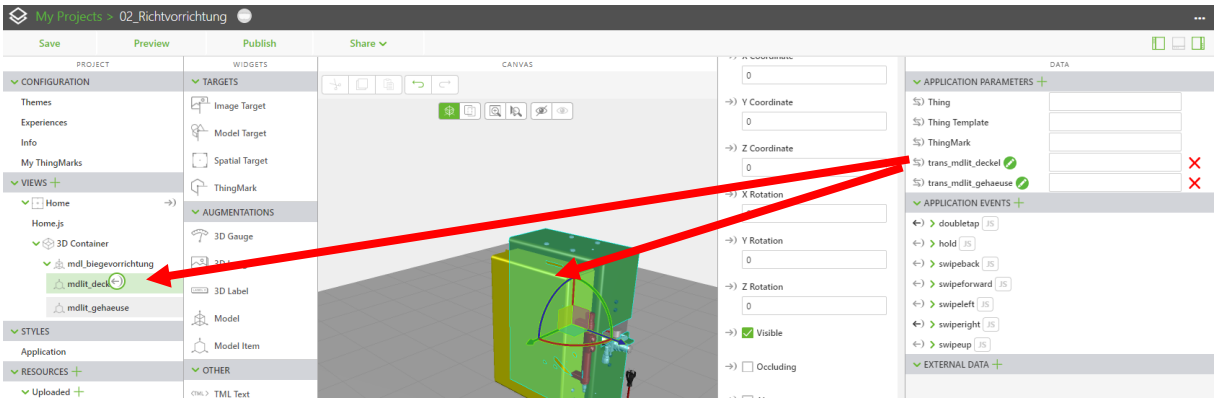
Auch das Gehäuse einem ModellItem zuweisen:



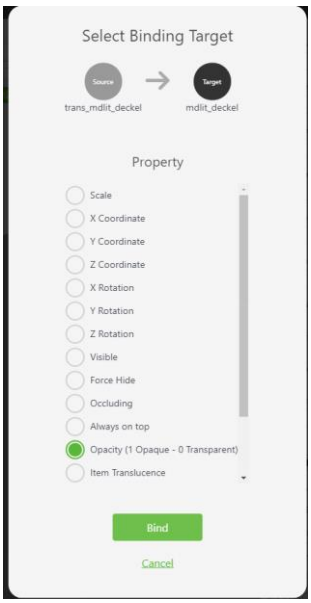

Weiters werden Application Paramter erzeugt. Sie ermöglichen den gezielten Zugriff auf die Eigenschaften der vorher angelegten Modellitems.




Die Application Parameter nun auf das gewünschte Moelltem ziehen (kann im 3D-Bereich **oder** in der Auflistung geschehen:



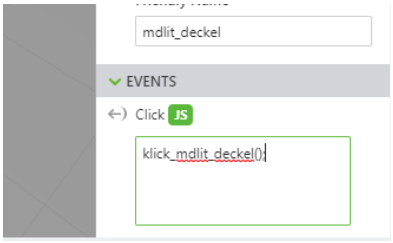

Danach wird das entsprechende Binding (also auf welche Eigenschaft sich der Parameter beziehen soll) angegeben:



Jetzt muss nur noch der Code geschrieben werden, der den Zusammenhang zwischen dem Anklicken des Teils und dem dadurch zuzuweisenden Wert herstellt:

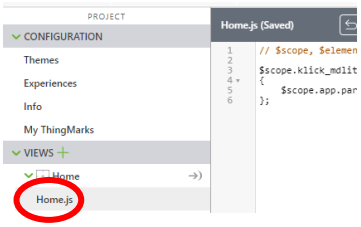
Das entsprechende Event findet man im Modellitem „mdlit_deckel“:

Dort fügt man einen Funktionsaufruf ein:




Man könnte den Code auch direkt hier hineinschreiben, der Vorteil einer Funktion liegt aber darin, dass der Code aller Funktionen zentral in einem Fenster bearbeitet werden kann.

Der Code wird unter Home→ Home.js eingefügt:



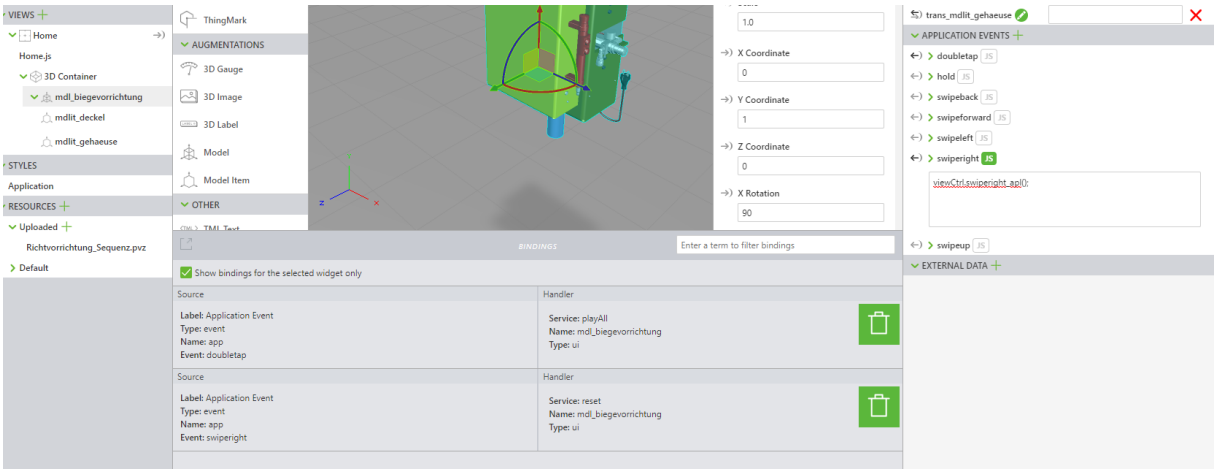

```
$scope.klick_mdlit_deckel = function()              // Funktionsaufruf DECKEL
{
    $scope.app.params.trans_mdlit_deckel = 0.3;     // Transparenz auf 0.3 setzen
};
```

Wenn man will, dass bei jedem Klick die Transparenz wechselt, kann man folgenden Code verwenden:

```
$scope.klick_mdlit_deckel = function()              // Funktionsaufruf DECKEL
{
    if ($scope.app.params.trans_mdlit_deckel < 1)
    {
        $scope.app.params.trans_mdlit_deckel = 1;     // Transparenz auf 1 setzen
    }
    else
    {
      $scope.app.params.trans_mdlit_deckel = 0.3;     // Transparenz auf 0.3 setzen
    }
};
```

Dieselben Schritte sind sinngemäß beim Gehäuse durchzuführen.

Weiters kann man beim swipright noch einen JS-Code hinzufügen, der die Transparenz unabhängig vom Ist-Wert auf zurücksetzt (Das Binding mit dem reset von mdl_biegevorrichtung wir dadurch nicht aufgehoben).




Bei diesem Funktionsaufruf muss noch viewCtrl vorangesetzt werden da es aus dem app-Objekt aufgerufen wird

```
viewCtrl.swiperight_apl();
```

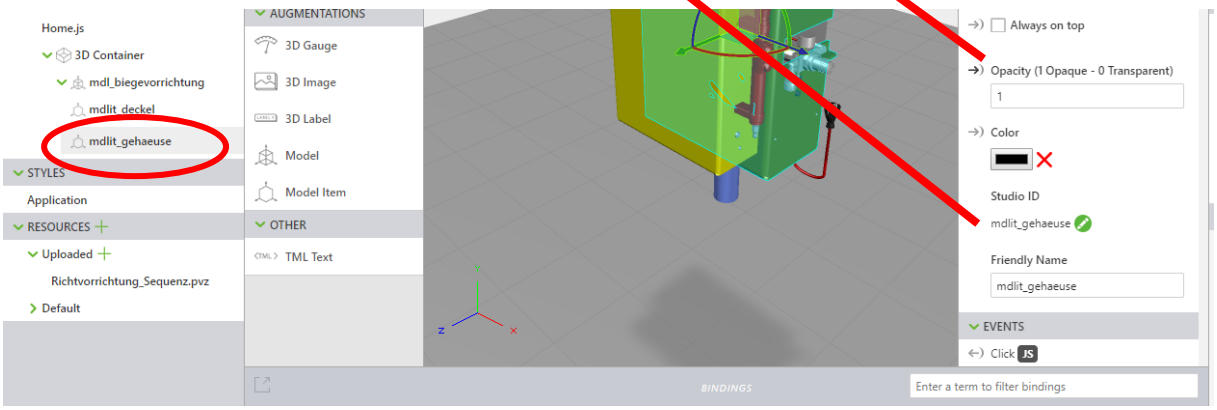
In home.js muss man folgendes ergänzen:

```
$scope.swiperight_apl = function()              // Funktionsaufruf SWIPERIGHT
{

        $scope.app.params.trans_mdlit_deckel = 1;     // Transparenz auf 1 setzen

        $scope.app.params.trans_mdlit_gehaeuse = 1;     // Transparenz auf 1 setzen

};
```

Alternative Methode - direktes ansprechen von Objekten:

Man kann auf das erzeugen Application Parametern verzichten und stattdessen folgende Syntax im JS-Code verwenden:

```
$scope.klick_mdlit_gehaeuse = function()        // Funktionsaufruf KLICK GEHAEUSE
{
    if ($scope.view.wdg['mdlit_gehaeuse'].opacity < 1)
    {
      $scope.view.wdg['mdlit_gehaeuse'].opacity = 1;
    }
    else
    {
      $scope.view.wdg['mdlit_gehaeuse'].opacity = 0.3;
    }
};
```




Dieser Code erreicht die selbe Funktionalität wie auf Seite17, ohne dass man dafür einen Application Parameter anlegen muss.

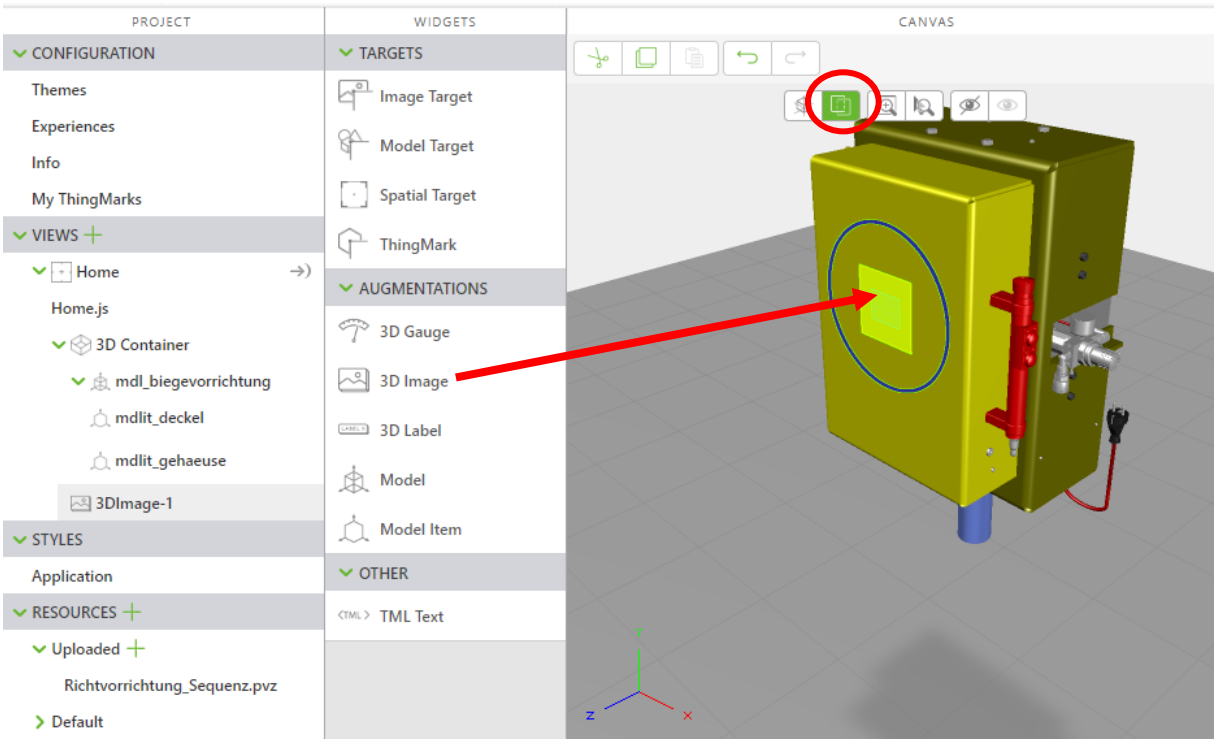
Ein mischen beider Varianten innerhalb einer Funktion funktioniert nur bedingt, da der Application Parameter bei einer Änderung der Eingenschaft nicht aktualisiert wird.

## Zusatzinformationen einfügen

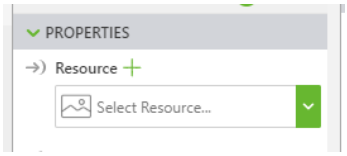
Im Folgenden wird ein Infoblatt eingefügt, auf dem z.B. die Bedienung und die Funktionen der AR beschrieben sind.

Ein 3D Image wird in den Modellbereich gezogen, dabei ist es praktisch die Funktion „mate to surface“ zu aktivieren, sie ermöglicht das gezielte platzieren auf einer Oberfläche des Modells.

Zuerst fügen wir das Symbol für die Schaltfläche ein:

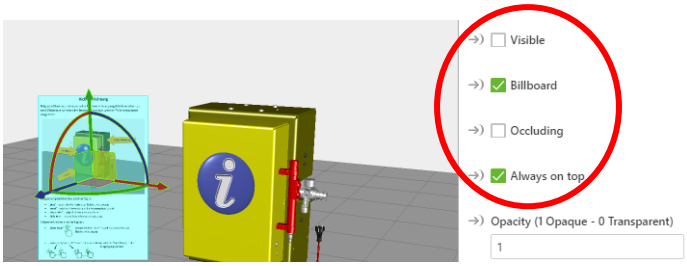



Nun noch über Resource+ die gewünschte Datei hinzufügen:




Dann das eigentliche Infoblatt:
(Hier die Funktion „mate to surface“ nicht einschalten)

Folgende Parameter einstellen:



Im JS-Code müssen noch folgende Funktionen eingebaut werden:

- Ausblenden Info Button wenn Animation gestartet
- Transparenz setzen wenn Deckel Transparent

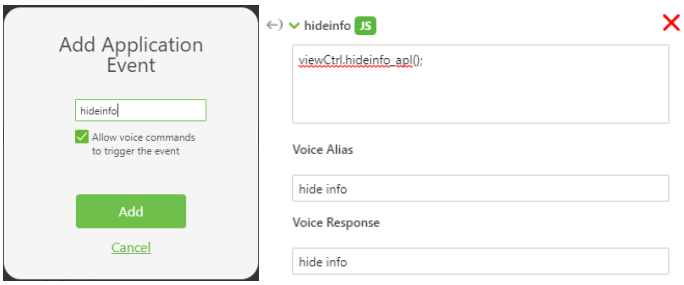
Der gesamte Code sieht nun so aus: (Deckel wird über Applicationparameter gesteuert, der Rest direkt)

```
$scope.klick_mdlit_deckel = function()          // Funktionsaufruf KLICK DECKEL
{
  if ($scope.app.params.trans_mdlit_deckel < 1)
    {
      $scope.app.params.trans_mdlit_deckel = 1;       // Transparenz auf 1 setzen
      $scope.view.wdg['img_ibutton'].opacity = 1;
    }
  else
    {
      $scope.app.params.trans_mdlit_deckel = 0.3;       // Transparenz auf 0.3 setzen
      $scope.view.wdg['img_ibutton'].opacity = 0.3;
    }
};
$scope.klick_mdlit_gehaeuse = function()          // Funktionsaufruf KLICK GEHAEUSE
{
    if ($scope.view.wdg['mdlit_gehaeuse'].opacity < 1)
    {
      $scope.view.wdg['mdlit_gehaeuse'].opacity = 1;
    }
    else
    {
      $scope.view.wdg['mdlit_gehaeuse'].opacity = 0.3;
    }
};
$scope.swiperight_apl = function()            // Funktionsaufruf SWIPERIGHT = RESET
{
      $scope.app.params.trans_mdlit_deckel = 1;       // Transparenz auf 1 setzen
      $scope.app.params.trans_mdlit_gehaeuse = 1;     // Transparenz auf 1 setzen
      $scope.view.wdg['img_ibutton'].visible = true;  // Infobutton anzeigen
};
$scope.klick_img_ibutton = function()
{
  $scope.view.wdg['img_info'].visible =! $scope.view.wdg['img_info'].visible;  // Toggle Infotext
};
$scope.doubletap_apl =function ()           // Aufruf wenn Animation gestartet wird
{
  $scope.view.wdg['img_ibutton'].visible = false;   // Infobutton ausblenden
};
```

## Sprachsteuerung

Für die Sprachsteuerung werden noch weitere Application Events angelegt

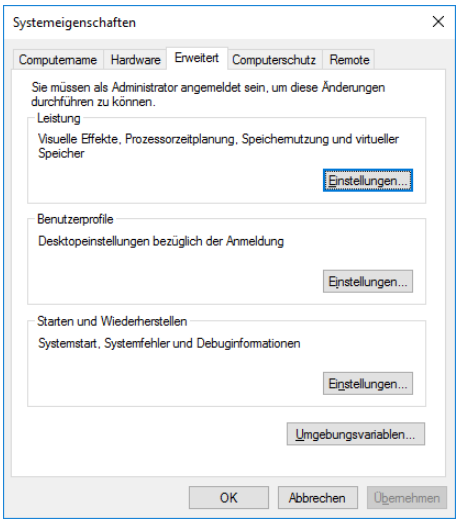
und mit entsprechendem Code in Home.js erzeugen

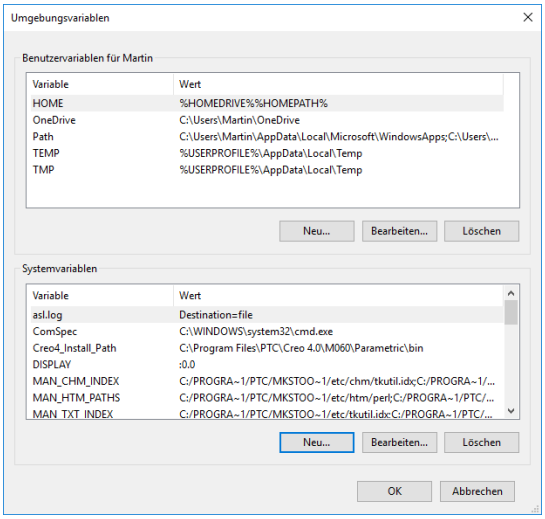


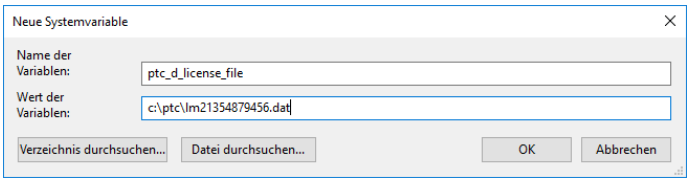
# Arbeiten mit Illustrate

## Installation

Nach der Installation kann es erforderlich sein, die Umgebungsvariable *ptc_d_license_file* zu erstellen und als Wert den Pfad zum Lizenzfile anzugeben:

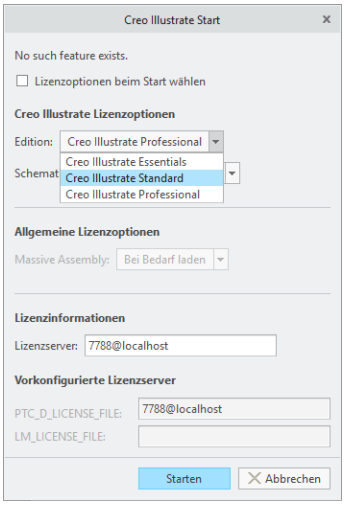



Wenn man einen Lizenzserver verwendet ist dieser Schritt nicht notwendig.

Beim Starten, noch als Edition „Creo Illustrate Standard“ wählen!



## Vorbereitung der Daten in Creo

Da zurzeit die Verwendung von Unterbaugruppen in Vuforia Studio Probleme verursacht, sollten Unterbaugruppen als Einzelteil eingebaut werden.

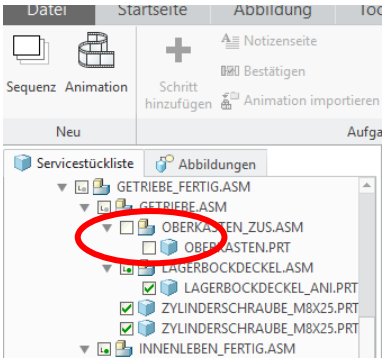
**Ablauf:**

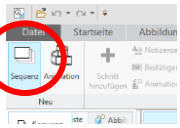
Unterbaugruppen als IGES speichern und als Teil wieder in die Hauptbaugruppe einbauen. Die originale Unterbaugruppe löschen.

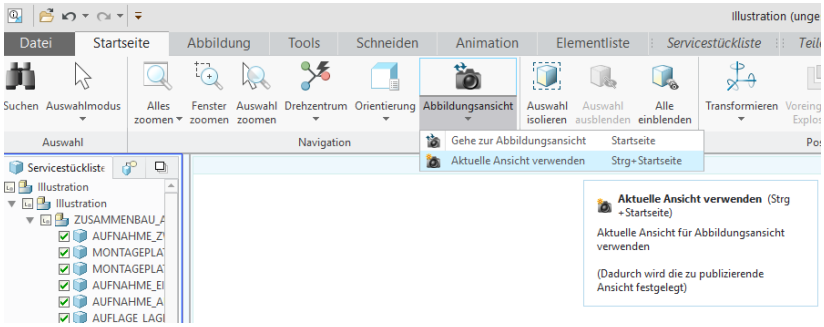
## Beispiel Getriebemontage

In Illustrate über die Schaltfläche Importieren die vorbereitete **Getriebemontage_Demontage.pvz** einbetten. Wenn man Verbinden wählt, dann werden Änderungen im Creomodell übernommen. Für die Aufbereitung mit Thingworx Studio ist allerdings das Einbetten sinnvoller.

Nach dem Öffnen den Oberkasten deaktivieren:

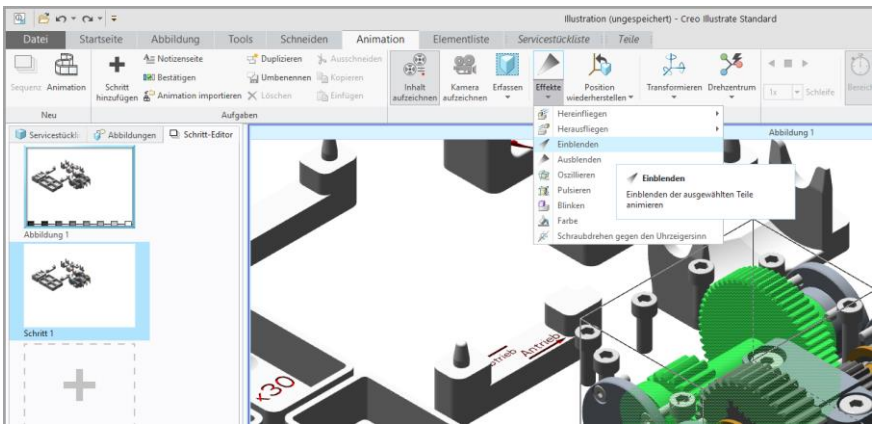



Im Menüband Animation die Schaltfläche Sequenz auswählen: 

Danach Im Menüband Startseite Abbildungsansicht – Aktuelle Ansicht verwenden anklicken: 

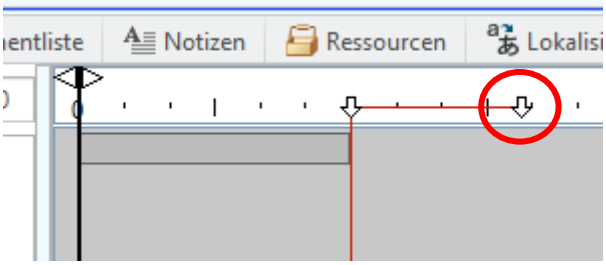

Dann unter Animation den ersten Schritt erzeugen:

Zwischenwelle anklicken und einblenden, da wir später den Effekt umkehren. D.h. aus dem Einblenden wird dann ein Ausblenden!




Nun die Welle in die Ablageposition bewegen – hier muss man sich vorher die genauen Koordinatenverschiebungen überlegen:

Die Zeit für die einzelnen Bewegungen können durch das Verschieben der Pfeile in der Zeitleiste angepasst werden. Dies ist vor allem bei sehr langen oder kurzen Wegen sinnvoll.

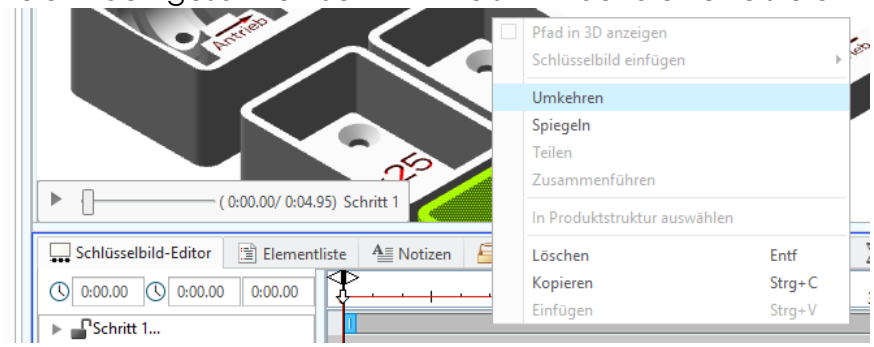



Die Schaltfläche Transformieren anwählen und folgende Verschiebungen nacheinander durchführen:

X: +70; Y: -105; Z: -98; X: -83

Dann die Zeit eventuell wieder etwas länger stellen und den Effekt „pulsieren“ auswählen.

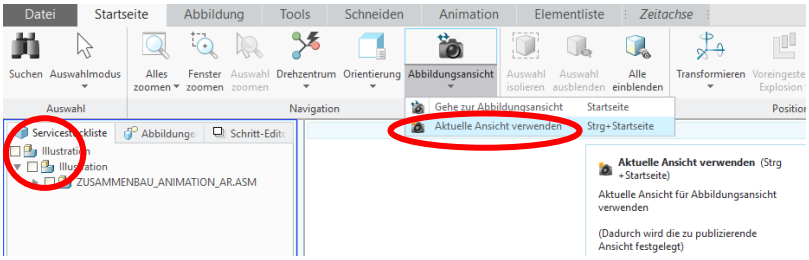
Zuletzt den gesamten Schritt mittels RMT auf die Zeitleiste umkehren:



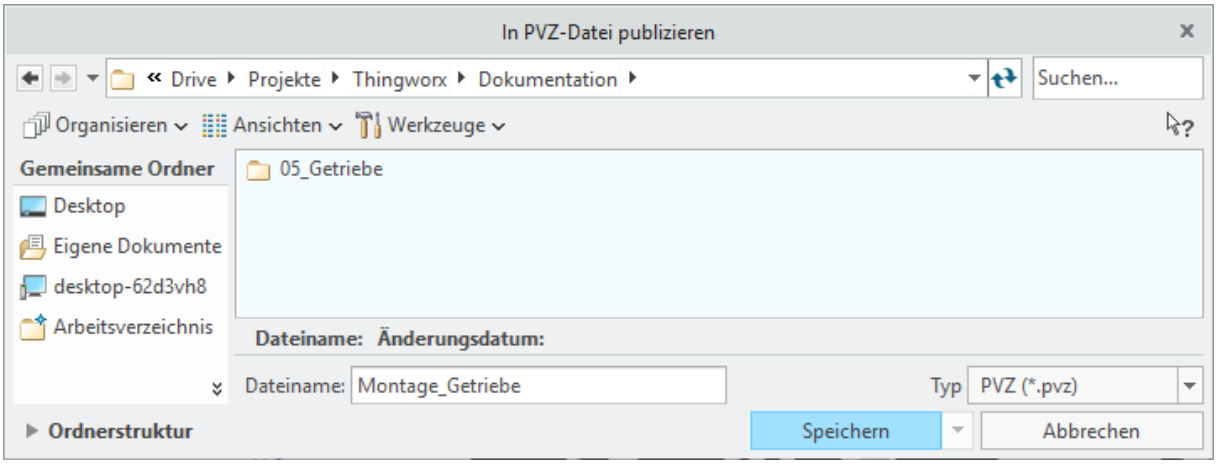
Dann einen neuen Schritt hinzufügen und den Vorgang sinngemäß für das nächste Bauteil erzeugen.

Auf diese Weise erhält man für jeden Arbeitsschritt eine eigene Sequenz, die dann in Vuforia Studio abgespielt werden kann.


Das fertige Projekt soll dann immer nur die aktuellen Arbeitsschritte in die Realität eiblenden, daher ist es sinnvoll, wenn man zu Beginn keines der Teile sieht. Daher die Abbildung (Montageablauf) im Schritteditor anklicken, dann auf Servicestückliste alle Teile abwählen und Abbildungsansicht – Aktuelle Ansicht verwenden:

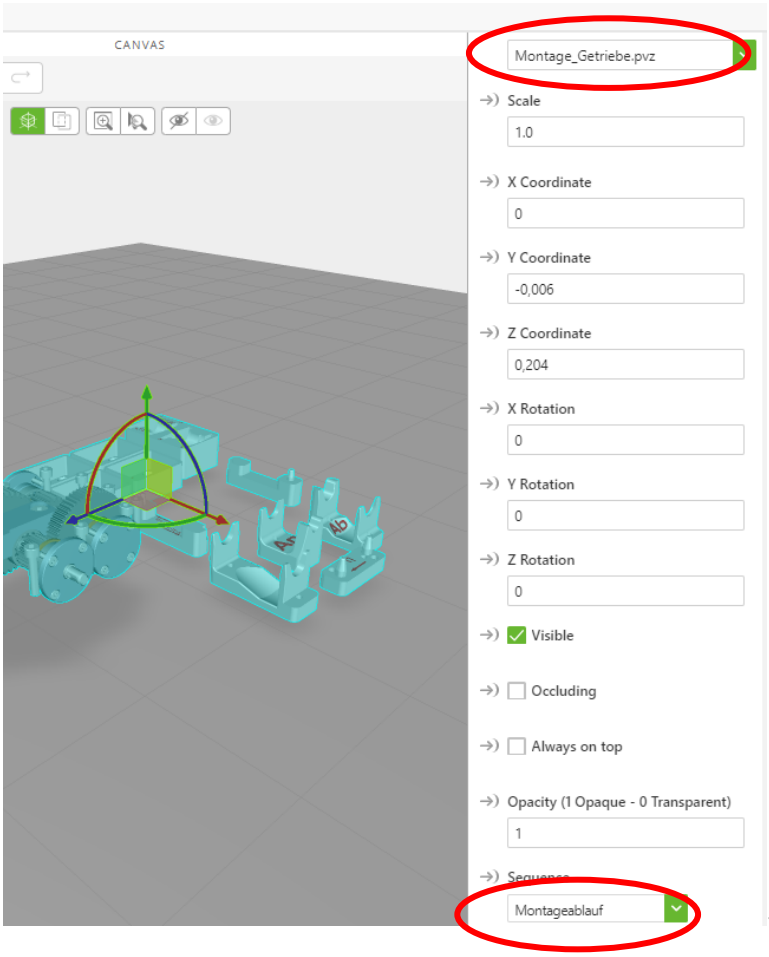



Abschließend als PVZ exportieren:



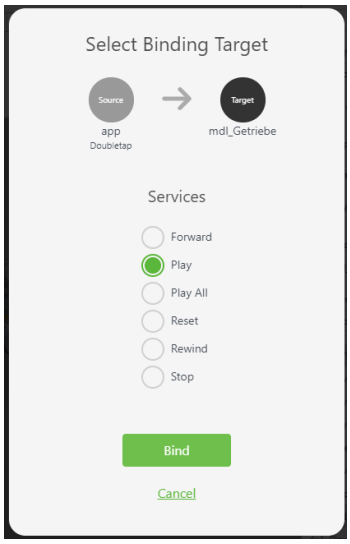
## Verwendung in Studio

Wie schon vorher wird in dem neuen Projekt ein Model erzeugt und die soeben generierte PVZ-Datei importiert.




- erzeugte Sequenz auswählen.

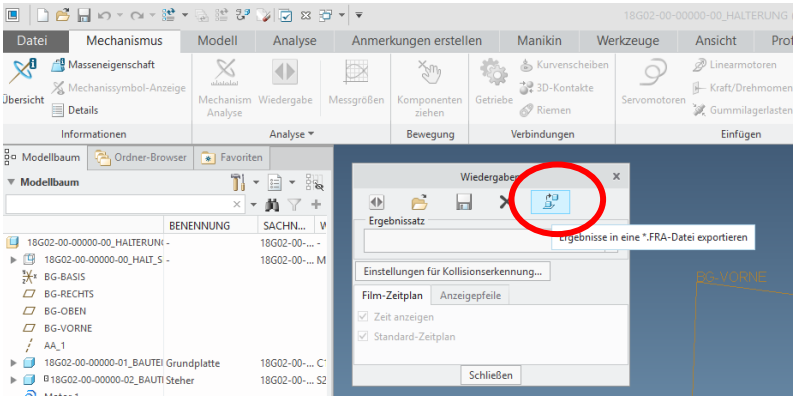
Damit die Sequenz gestartet wird, muss noch ein Binding hergestellt werden. Zum Beispiel mit einem Doppelklick:




Durch Auswahl von Play wird immer nur ein Schritt abgespielt.

## Animationen aus Creo Mechanismus importieren

Die Mechanismus-Analyse muss mit der Wiedergabesteuerung in CREO als fra-Datei exportiert werden:




Die Datei wird im Arbeitsverzeichnis gespeichert.

In Creo Illustrate wieder die folgenden Schritte durchführen:

Daten eingebettet importieren

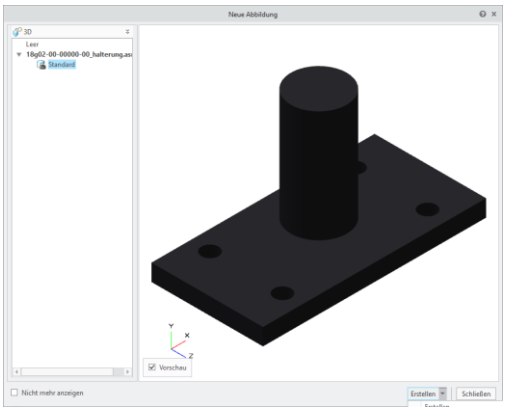
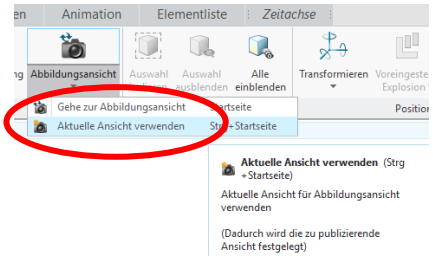
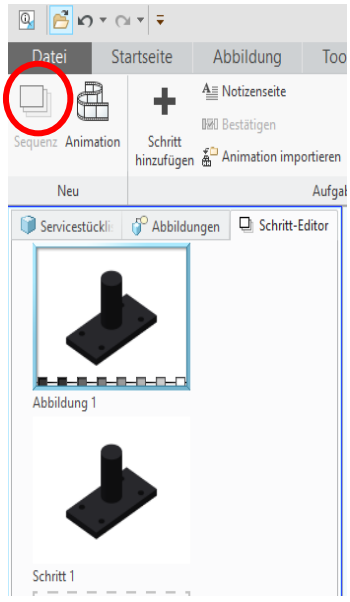


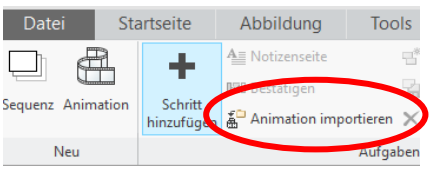
Abbildung der Ausgangssituation erstellen:




Dann Animation – Sequenz auswählen:

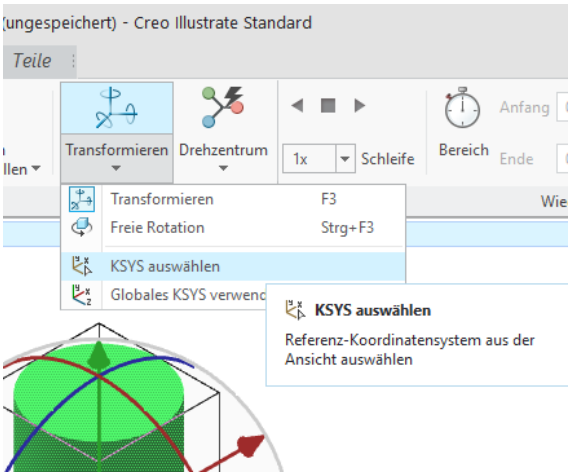
Nun in der Registrierkarte Animation auf „Animation importieren“ klicken und die soeben erzeugte fra-Datei laden:



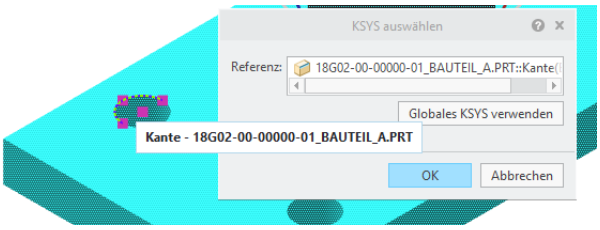

Der weiter Vorgang ist gleich wie vorher: - Publizieren und in Studio als Resource laden.

## Zusatzinfo „Scharniere“

Standardmäßig wird das Koordinatensystem des Bauteils als Drehzentrum verwendet. Um eine gezielte Drehbewegung um eine bestimmte Achse auszuführen ist es möglich ein anders KSYS auszuwählen:




Hier kann z.B. die Kante einer Bohrung ausgewählt werden:

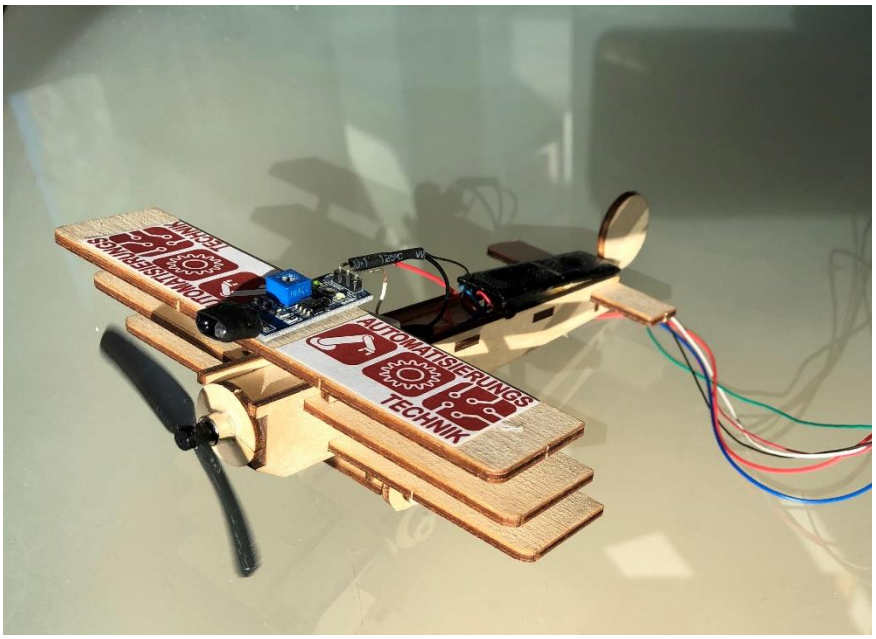



So ist es möglich eine Drehung um diese Achse zu erzeugen.

# Beispiele

## Modellflieger (Dreidecker – Fokker DR1)

Die vorliegenden CAD-Daten des Modellfliegers basieren auf einem Holzmodell von Conrad.




Folgende Messwerte werden erfasst und an den TWX-Server übermittelt:

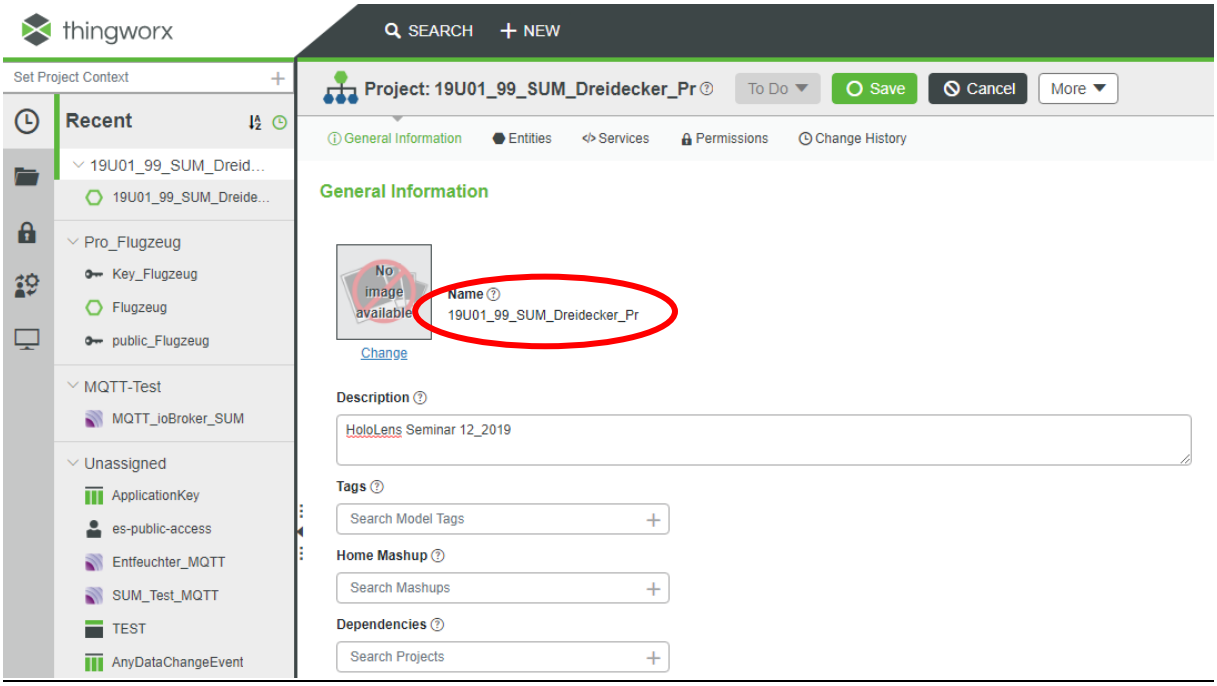
- Drehzahl (IR-Reflektionslichtschranke – Interrupt im ESP32)
- Temperatur und Feuchtigkeit (DHT22)
- Spannung der Solarzelle (AD-Wandler im ESP32)

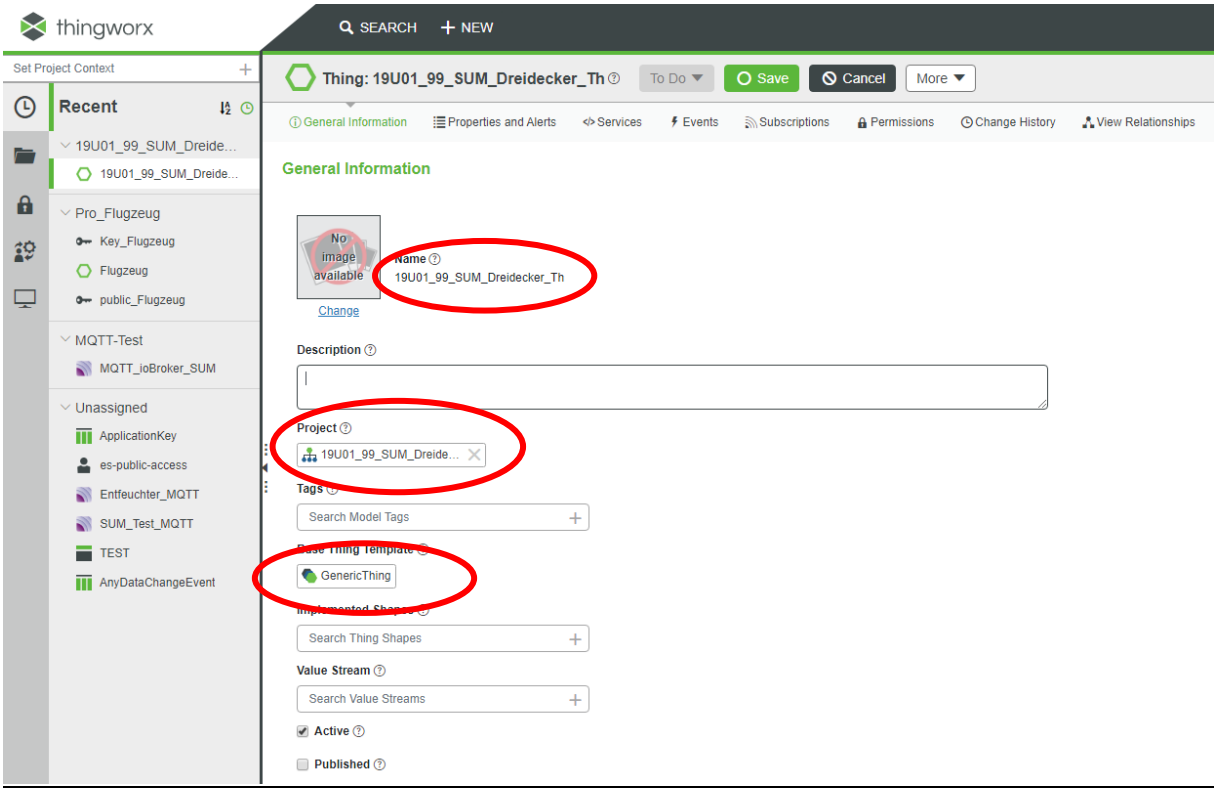
Zum Testen werden die entsprechenden Datenpunkte im TWX-Composer angelegt, aber nicht mit realen Werten versorgt.

Die erste Aufgabe besteht darin, den Propeller des Zwillings drehen zu lassen, wenn die Drehzahl > 0 ist.

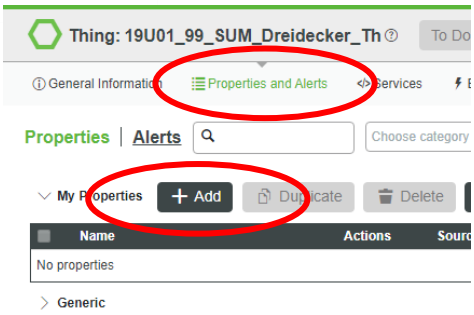
# Vorbereitung in TWX-Composer

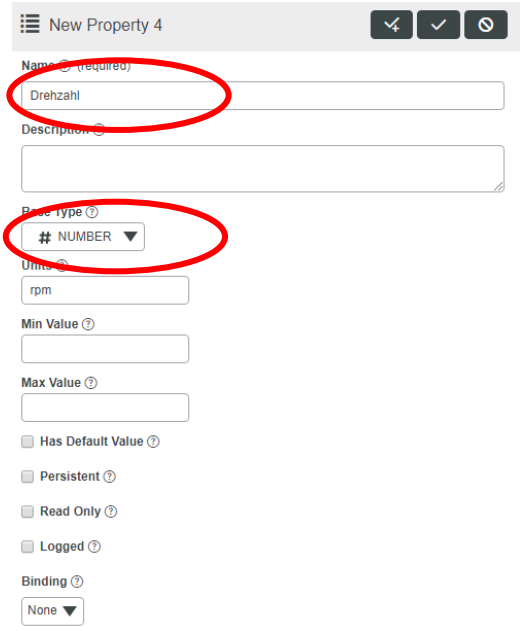
## Projekt und Thing anlegen:



Property Drehzahl hinzufügen:




Die angegeben Einheit hat keinen Einfluss auf die Funktion.

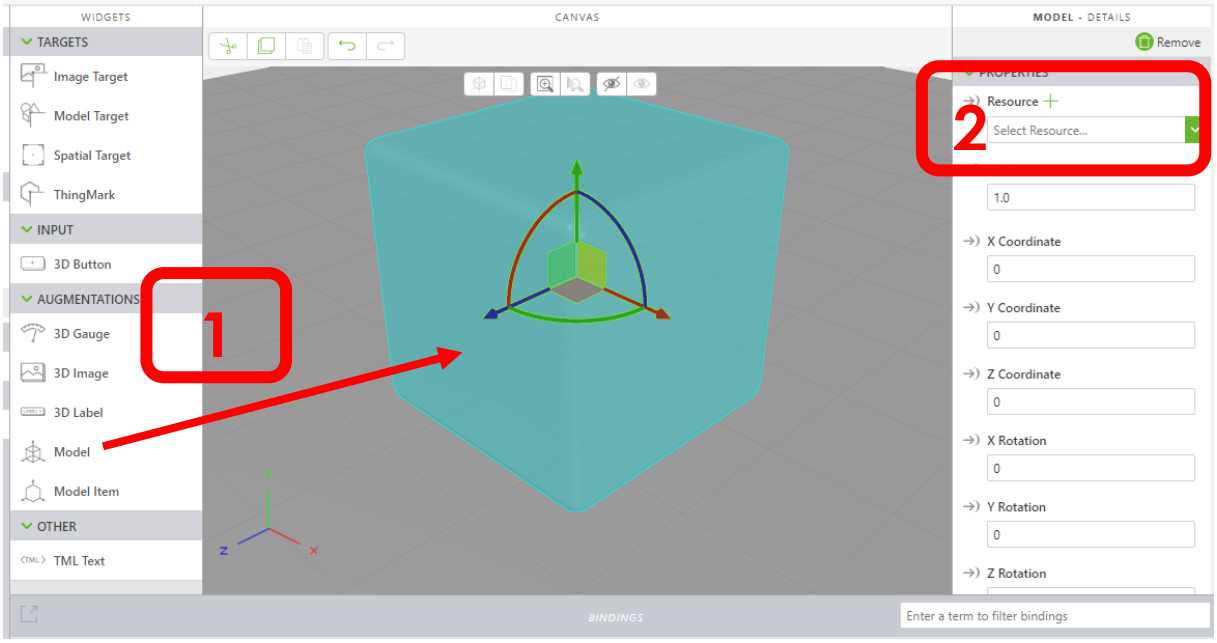
Das Thing danach speichern!

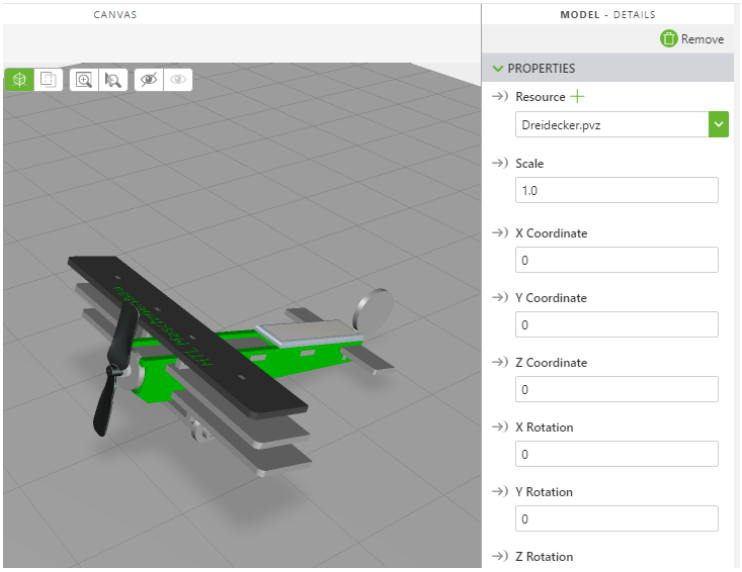
Um Daten an TWX senden zu können müsste auch noch ein Application Key erzeugt werden.

## Vuforia Studio

Anlegen eines neuen Projekts (3D-Eyewear)

Model in den Arbeitsbereich ziehen und Resource laden:

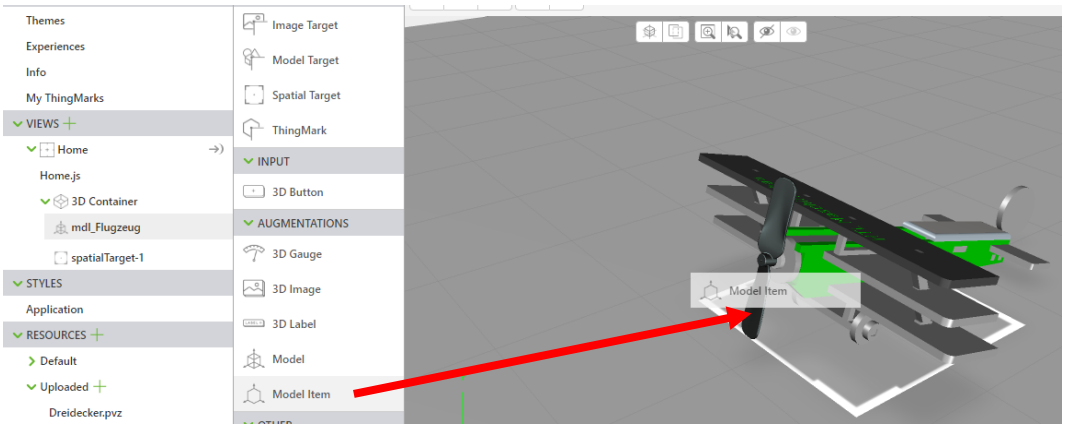




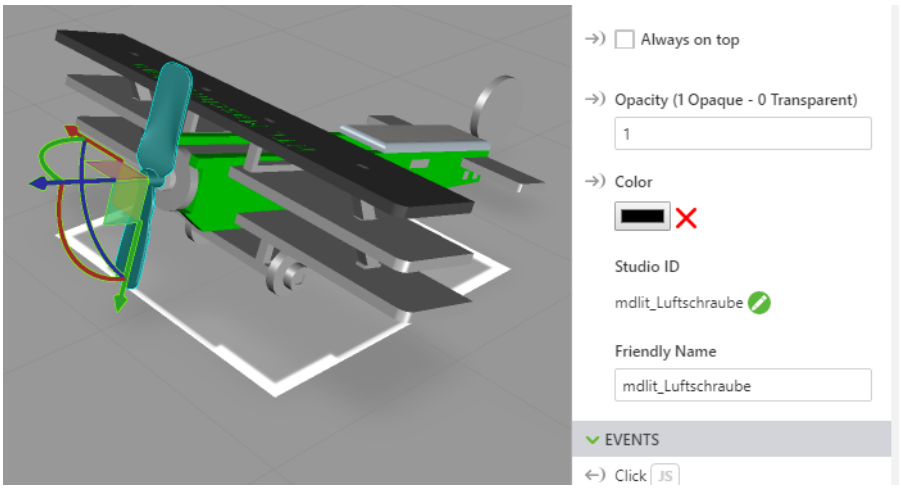

Koordinaten ggf. auf 0 setzen und Namen vergeben:


Als Target ein Spatial Target oder Thingmark wählen und in den Arbeitsbereich ziehen.

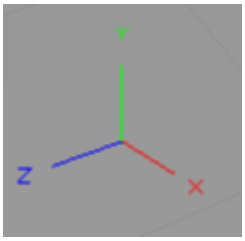
Nun muss die Luftschraube einem Modelitem zugewiesen werden:



Benennung ändern:




Die Achsen sind:



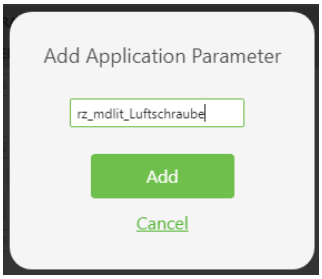

ROT X

GRÜN Y

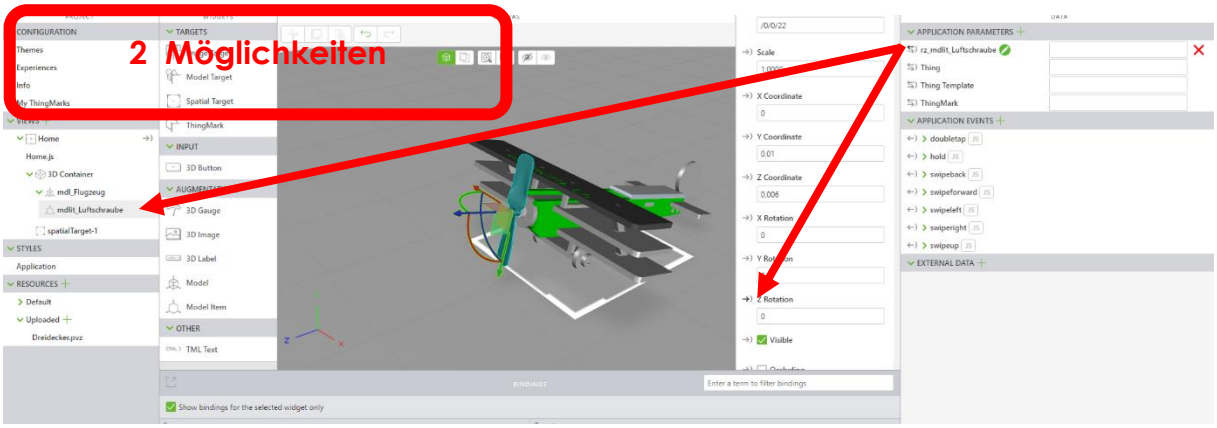
BLAU Z

Für die Drehung der Luftschraube ist daher die Z-Achse passend.

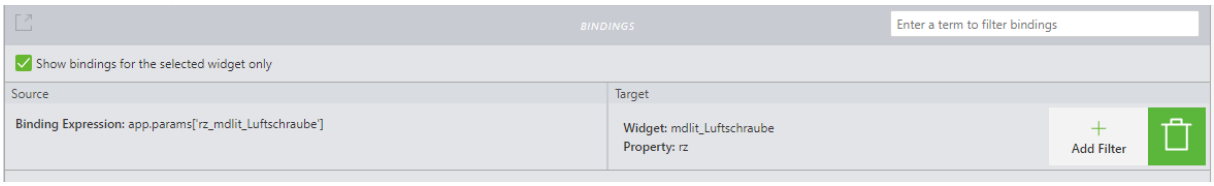
Nun wird ein Application Parameter erzeugt und Z Rotation zugewiesen:



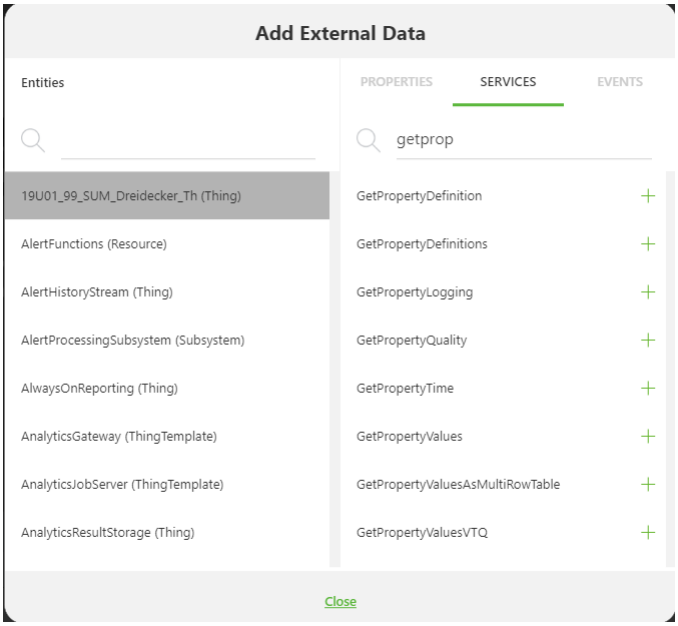

Dann den Parameter direkt auf Z Rotation ziehen, ***oder*** auf das Modelitem (Auswahldialog frägt dann gewünschte Property ab)



Folgendes Binding sollte angezeigt werden:

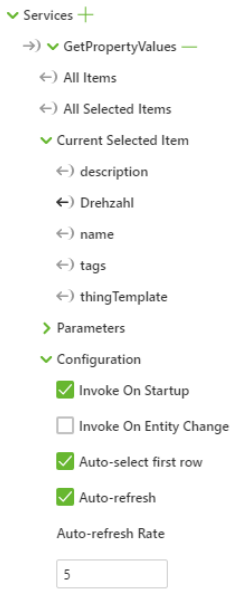



Einbinden der externen Daten:

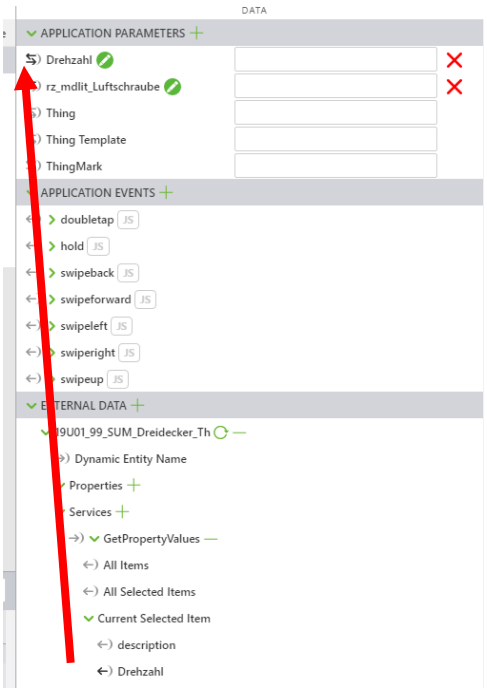



Den Service „GetPropertyValue“ hinzufügen.

Die Drehzahl sollte nun verfügbar sein. Wichtig ist das setzen der angezeichneten Optionen. Auto-refresh Rate ist das Aktualisierungsintervall in Sekunden.

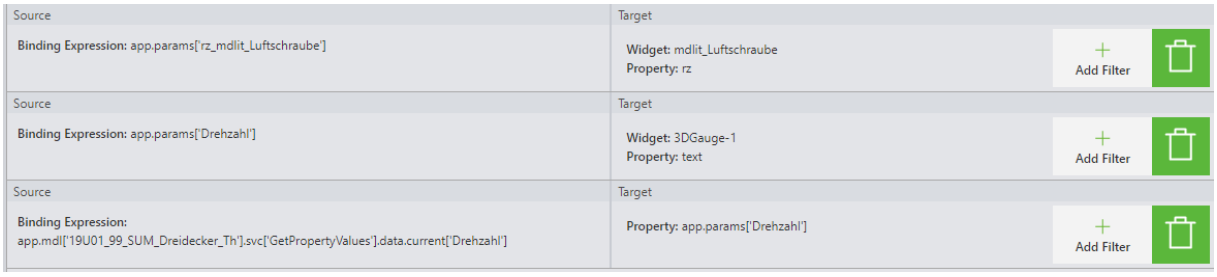


Für den Drehzahlwert wird auch noch ein Application Parameter erzeugt und ein Binding mit den externen Daten hergestellt:



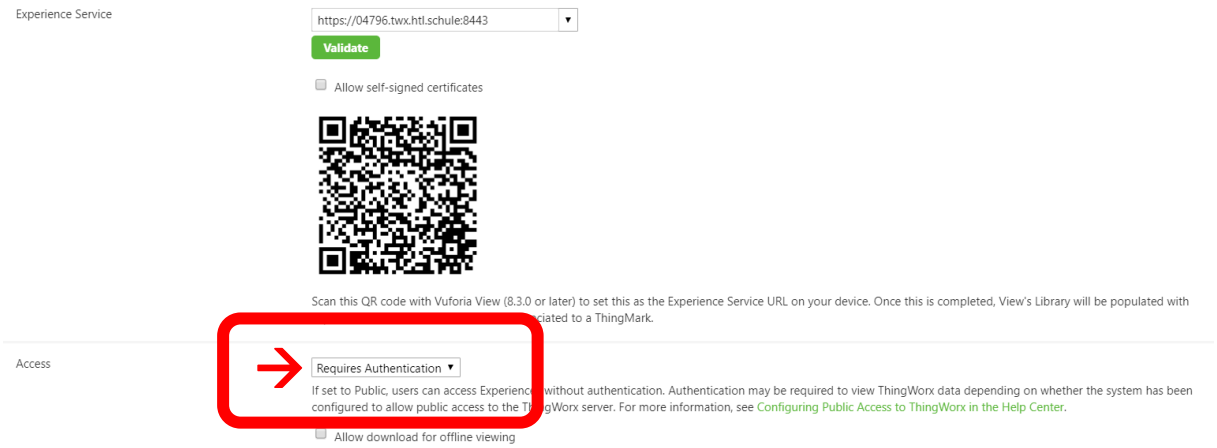

Zur Anzeige der Drehzahl kann diese auch mit einem 3DGauge verbunden werden.

Es gibt dann die folgenden Bindings:




ACHTUNG!

Die Übernahme der aktuellen Werte in ein Mobilgerät (HoloLens und Handy) funktioniert nur, wenn der Zugriff auf „Authentifizierung erforderlich“ gestellt wird.



## JavaScript für Luftschraube

Zum Steuern des Winkels der Luftschraube in Abhängigkeit des Drehzahlwerts ist es erforderlich auf Änderung des Drehzahlwertes zu reagieren. Wir öffnen daher den integrierten Editor in dem wir auf Home.js clicken.

Mit folgendem Code wird auf eine Änderung reagiert:

```
$scope.$watch("app.params['Drehzahl']",function()
{
**** MEIN CODE ***
})
```

Als ersten Test versuchen wir den Winkel der Luftschraube bei jeder Werteänderung um 10 zu erhöhen.

Der Code zur Addition von 10 lautet:

```
$scope.app.params.rz_mdlit_Luftschraube = $scope.app.params.rz_mdlit_Luftschraube +10;
```

Oder in kurzer Schreibweise:

```
$scope.app.params.rz_mdlit_Luftschraube += 10;
```

**ACHTUNG!**

Verwendet man diesen Code ohne weiter Maßnahmen funktioniert es nicht, da die Application Parameter noch nicht initialisiert sind und keinen definierten Wert haben.

Man muss daher prüfen ob die Variable bereits „gültig“ ist und einen Startwert zuweisen:

```
if (!$scope.app.params.rz_mdlit_Luftschraube){$scope.app.params.
rz_mdlit_Luftschraube = 0}
```

Der fertige Code hat dann folgende Form:

```
$scope.$watch("app.params['Drehzahl']",function()
{
          if (!$scope.app.params.rz_mdlit_Luftschraube){$scope.app.params.rz_mdlit_Luftschraube = 0}
          $scope.app.params.rz_mdlit_Luftschraube += 10;
          console.log("Drehwinkel: " + $scope.app.params.rz_mdlit_Luftschraube);
})
```

CODE → Schritt1.txt

Mit Preview nun die Funktionalität testen – bei Wertänderungen der Drehzahl im Composer sollte sich spätestens 5 Sekunden später die Luftschraube um 10 Grad weiter drehen.

## Loggen von Werten

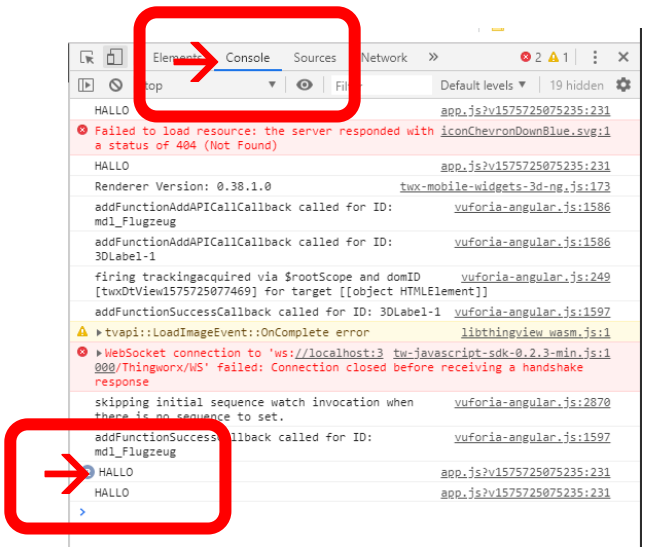
Bei der Entwicklung von neuem Code ist es oft hilfreich Werte oder Hinweise aus dem Programcode anzeigen zu lassen.

Der Eintrag wird im Code mit der folgenden Zeile ausgelöst:

```
console.log("HALLO");
```

Unter Chrome müssen die Entwicklertools aktiviert werden [Strg]+[Shift]+[i]

Dann Registrierkarte „Console“ umschalten:




Für unser Beispiel „Flugzeug“ sieht das dann wie folgt aus:

Damit sich die Luftschraube kontinuierlich dreht, muss der Drehwinkel kontinuierlich erhöht werden.

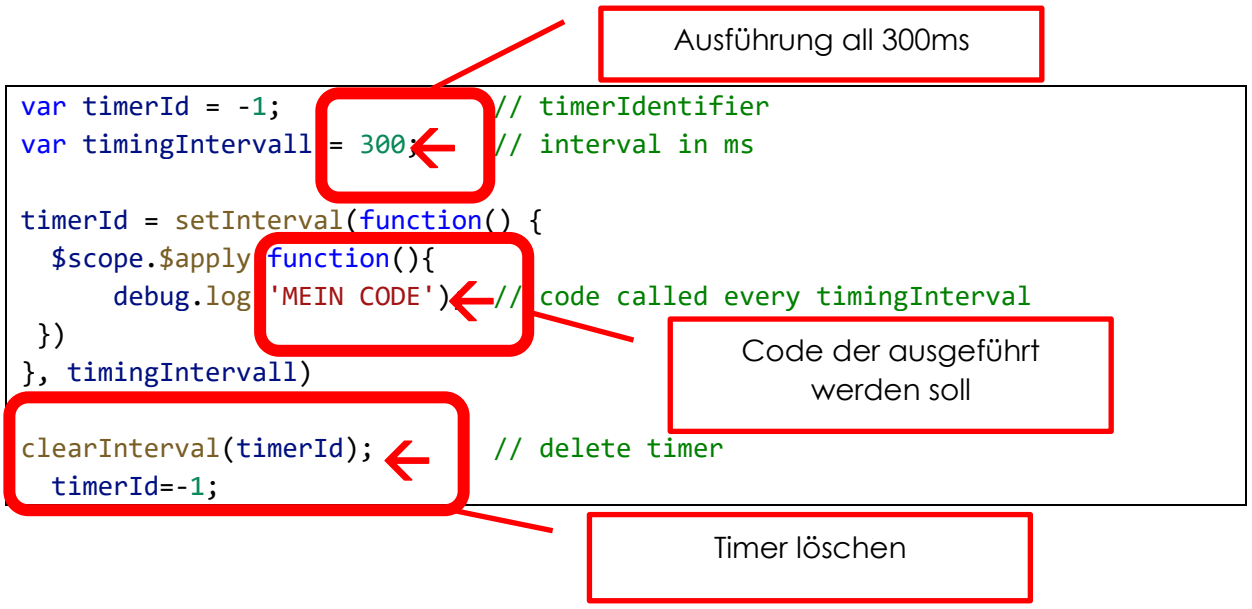
Hierfür verwenden wir einen Timer:

## Timer verwenden

Grundaufbau:

```
setInterval

clearInterval
```

Wenn der Timer gestartet wird, bekommt man als Rückgabewert einen sogenannten „timer identifier". Damit ist es später möglich diesen Timer auch wieder zu löschen.



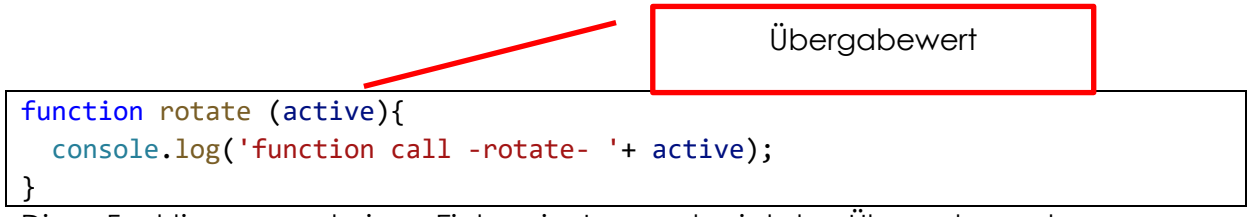

weitere Infos:

https://javascript.info/settimeout-setinterval

## Funktionen verwenden

Um die Übersichtlichkeit zu erhöhen kann man eigene Funktionen definieren und im Programmablauf aufrufen.

Wir definieren die Funktion rotate:




Diese Funktion erzeugt einen Eintrag im Log und zeigt den Übergabewert an.

Der Funktionsaufruf erfolgt mit:

```
rotate(true);
```

bzw.:

```
rotate(false);
```

## Bedingungen verwenden

Um den Programmablauf steuern zu können braucht man noch die if und else Anweisungen:

```
if ($scope.app.params.Drehzahl > 0){
  rotate(true);
}
else {
  rotate (false);
}
```

Vergleichsoperatoren:

| | |
|---|---|
| == | ISTGLEIC |
| != | UNGLEICH |
| > | GRÖßER |
| >= | GRÖßERGLEICH |
| < | KLEINER |
| <= | KLEINERGLEICH |

**Weitere Infos:**

https://wiki.selfhtml.org/wiki/JavaScript/Operatoren/Vergleichsoperatoren

Hat man mehrere Bedingungen zu verknüpfen, so können die logischen Operatoren dazu verwendet werden:

| | |
|---|---|
| && | UND |
| \|\| | ODER |
| ! | NICHT |

**Weitere Infos:**

https://wiki.selfhtml.org/wiki/JavaScript/Operatoren/Logische_Operatoren

## Gesamter Code zum Drehen des Propellers

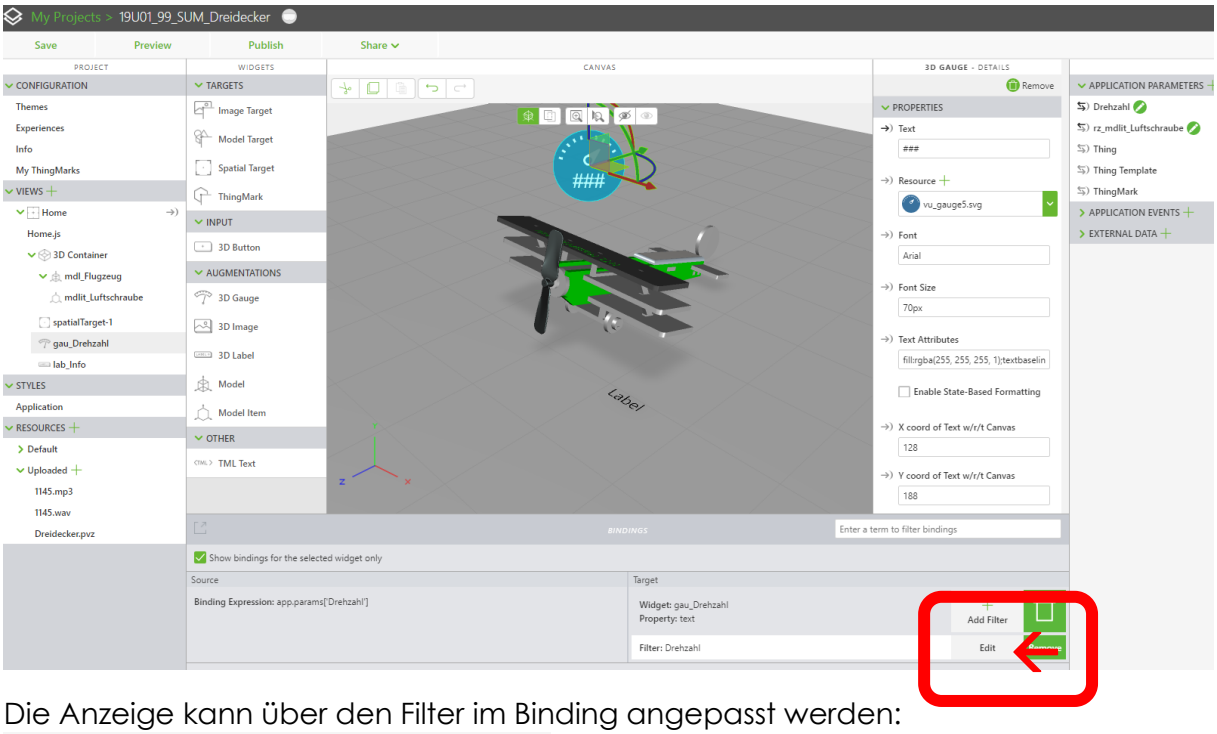
```
var timerId = -1;     //timer identifier
var timingIntervall = 300;  //miliseconds for timer
  if (!$scope.app.params.rz_mdlit_Luftschraube){$scope.app.params.rz_mdlit_Luftschraube = 0}  //  set
to startvalue

$scope.$watch("app.params['Drehzahl']",function() //  call function when value changes
{
  if ($scope.app.params.Drehzahl > 0){  //check propeller rotation
    rotate(true);
  }
  else {
    rotate (false);
  }
})

function rotate (active){
console.log('function call -rotate- '+ active);
if (active && timerId == -1){
 timerId = setInterval(function() {
   $scope.$apply(function(){              // code called every timingIntervall
       $scope.app.params.rz_mdlit_Luftschraube += 10;
  })
 }, timingIntervall)
}
if(!active){
 clearInterval(timerId);
   timerId=-1;
}
}
```

CODE → Schritt2.txt

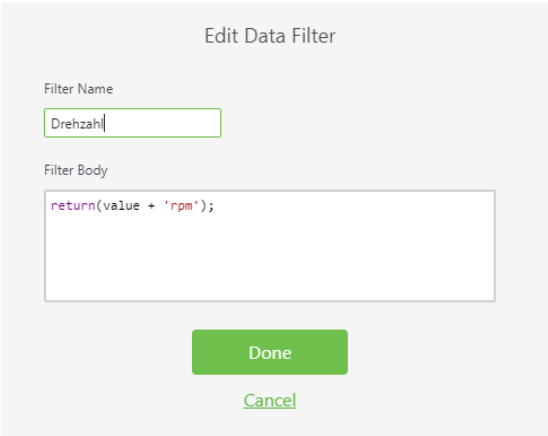
## Drehzahl und Statusanzeige

Wir fügen noch eine Anzeige für den Drehzahlwert und einen Infotext über den Zustand des Motors ein.

3D Gauge in den Arbeitsbereich ziehen und mit dem Applicationparameter „Drehzahl“ ein Binding herstellen.




Die Anzeige kann über den Filter im Binding angepasst werden:

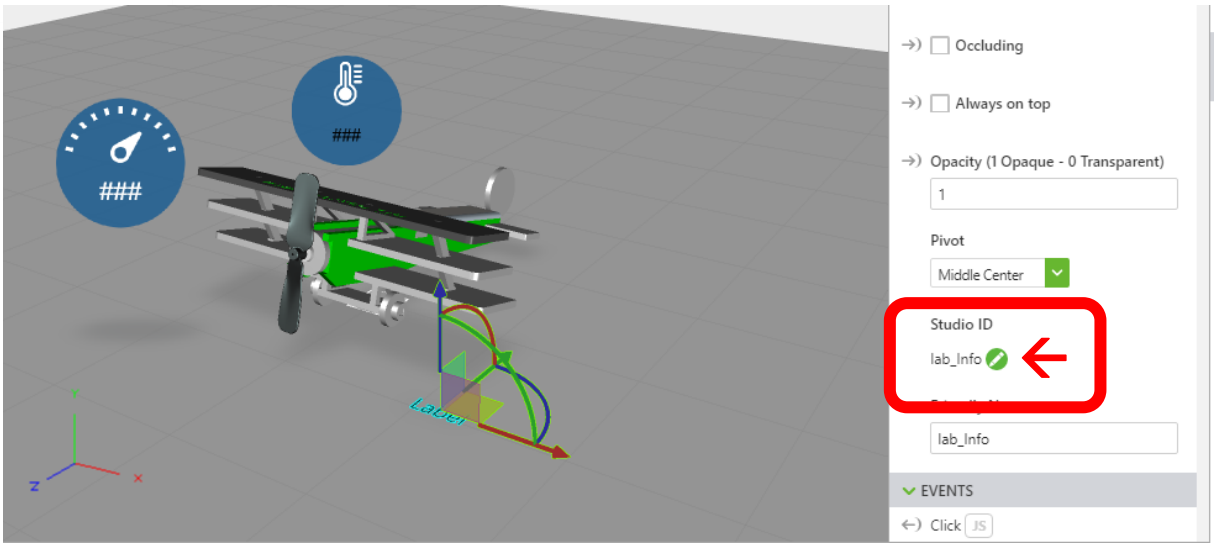



Wenn man Messwerte mit Nachkommastellen hat und diese runden möchte, dann kann man den Code erweitern: (funktioniert nur wenn Datentyp keine integer Wert ist)

```
Return (value.toFixed(2) + '°C');
```

Zum Testen Property „Temperatur“ anlegen und mit neuem Gauge binden.

Für den Infotext ein 3DLabel anlegen und als ID „lab_Info“ angeben:

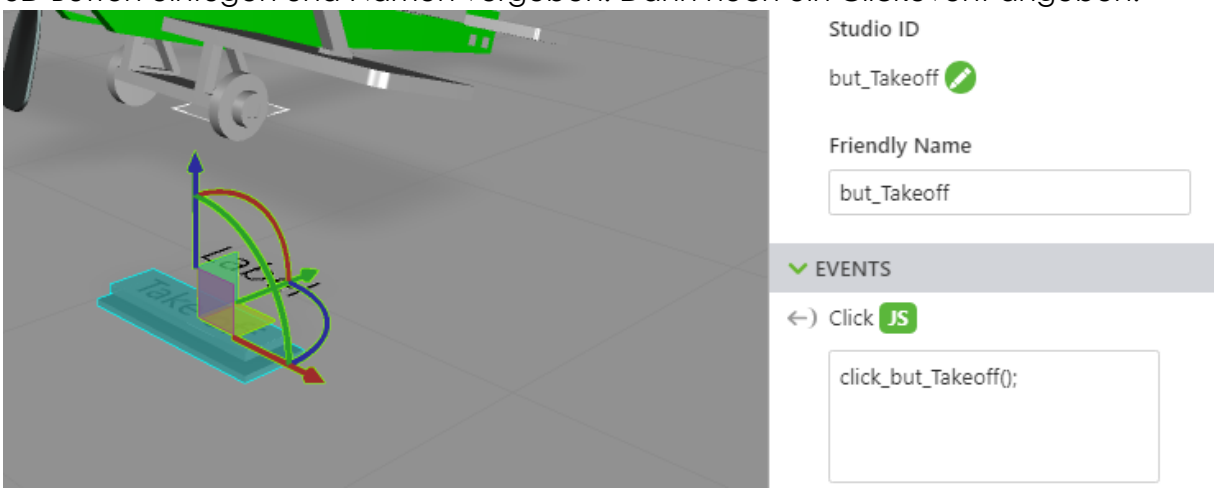



Im Code folgende Zeilen einfügen:

```
    var timerId = -1;           //timer identifier
    var timingIntervall = 30;   //miliseconds for timer
    if (!$scope.app.params.rz_mdlit_Luftschraube){$scope.app.para

$scope.$watch("app.params['Drehzahl']",function()
{
    if ($scope.app.params.Drehzahl > 0){                    //check p
        rotate(true);
        $scope.view.wdg['lab_Info'].text = 'Motor läuft';
    }
    else {
        rotate (false);
        $scope.view.wdg['lab_Info'].text = 'Motor aus';
    }
})

function rotate (active){
 console.log('function call -rotate- '+ active);
 if (active && timerId == -1){
```

Eine Schaltfläche „Take off“ soll nur sichtbar sein, wenn die Drehzahl > 0 ist.

3D Button einfügen und Namen vergeben. Dann noch ein Clickevent angeben:




Im Code die Sichtbarkeit steuern:

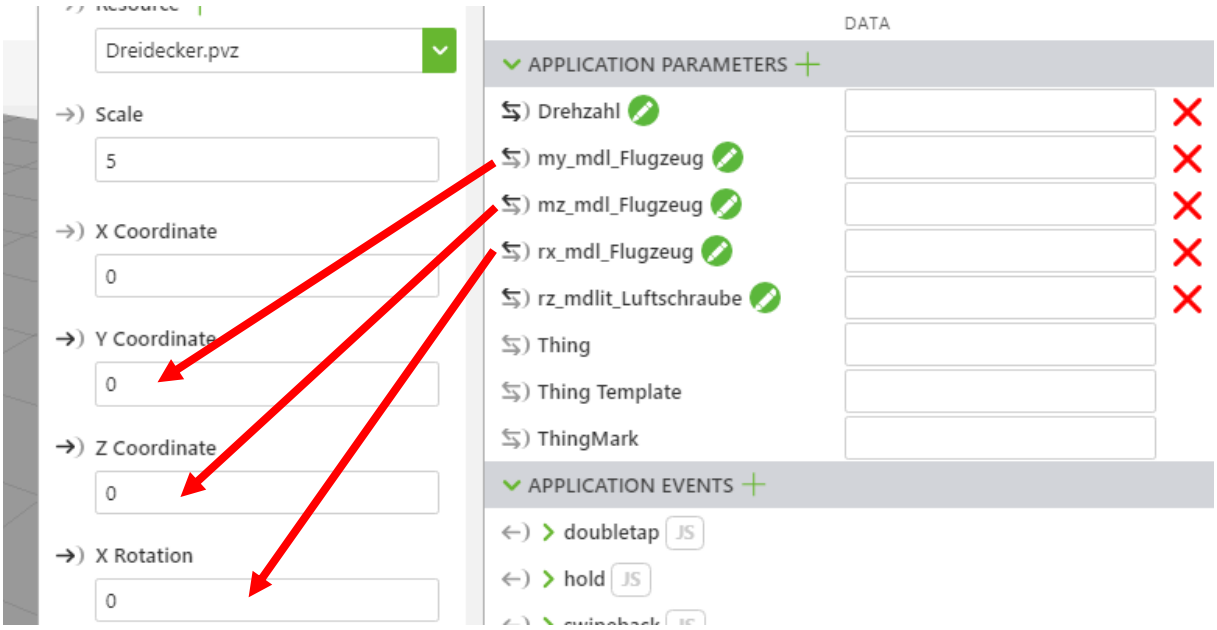
```
$scope.$watch("app.params['Drehzahl']",function()                    //
call function when value changes
{
    if ($scope.app.params.Drehzahl > 0){        //check propeller rotation
      rotate(true);
        $scope.view.wdg['but_Takeoff'].visible = true;
    }
    else {
      rotate (false);
        $scope.view.wdg['but_Takeoff'].visible = false;
    }
})
```

Mit der Schaltfläche soll nun ein Bewegungsvorgang für das ganze Flugzeug gestartet werden. Dazu führen wir einen weiteren Timer ein. Zur Unterscheidung benötigen wir eine neue timerID. Daher wird die vorhandene umbenannt:


Für die Bewegung wird eine neue Funktion erstellt, die im Wesentlichen gleich jener ist, die die Rotation der Luftschraube ausführt.

## Steigflug und Looping ausführen

Die Applicationparameter und die Bindings:




Mit dem Objekt Math können Berechnungen durchgeführt werden und Konstanten eingefügt werden:
Für den weiteren Ablauf benötigen wir

- Math.PI
- Math.sin(x)
- Math.cos(x)

Weiter Infos:

https://wiki.selfhtml.org/wiki/JavaScript/Objekte/Math

Das Flugzeug soll nun eine bestimmte Höhe im Steigflug erreichen und dann Loopings fliegen.

Wir legen zunächst folgende Variablen an:

```
var timerId_mov = -1;    //timer identifier


var timingIntervall_mov = 30; //miliseconds for timer (climb & looping)

var climb_mz = 0.01;       // climb parameter
var climb_my = 0.005;
```

und erstellen eine Funktion, die die Position des Flugzeugs ständig verändert:

```
function move (active){
console.log('function call -move- '+ active);
if (active && timerId_mov == -1){
 timerId_mov = setInterval(function() {
   $scope.$apply(function(){
        $scope.app.params.mz_mdl_Flugzeug += climb_mz;
        $scope.app.params.my_mdl_Flugzeug += climb_my;
  })
 }, timingIntervall_mov)
}
if(!active){                 // return home
 clearInterval(timerId_mov);
   timerId_mov=-1;
   $scope.app.params.mz_mdl_Flugzeug = 0;
  $scope.app.params.my_mdl_Flugzeug = 0;
}
}
```

CODE → Schritt3.txt

Der letzte Schritt ist das Verknüpfen der Funktion mit dem CLickereigniss des Buttons:

```
$scope.click_but_Takeoff = function () {
move(true);
}
```

Das Flugzeug sollte nun nach dem Betätigen des Buttons mit dem Steigflug beginnen.

Der weitere Code könnte folgendermaßen aussehen:

```
// $scope, $element, $attrs, $injector, $sce, $timeout, $http, $ionicPopup,
and $ionicPopover services are available
/*
  m.schubert 2019
    HTL Wiener Neustadt

*/

var timerId_rot = -1;      //timer identifier
var timerId_mov = -1;      //timer identifier
var timingIntervall_rot = 30; //miliseconds for timer


var timingIntervall_mov = 30; //miliseconds for timer (climb & looping)

var climb_mz = 0.01;       // climb parameter
var climb_my = 0.005;
```

```
var omega_loop = 0.4;     //looping rad/s
var r_loop = 1;          // radius looping
var starthight_loop = 0.3;    // starting loop at hight


var mzstart_loop = 0;
var mystart_loop = 0;
var phi_loop = 0;

  if
(!$scope.app.params.rz_mdlit_Luftschraube){$scope.app.params.rz_mdlit_Luftschr
aube = 0}  //  set to startvalue
if (!$scope.app.params.mz_mdl_Flugzeug){$scope.app.params.mz_mdl_Flugzeug = 0}
// mdl_pos z
  if (!$scope.app.params.my_mdl_Flugzeug){$scope.app.params.my_mdl_Flugzeug =
0}        // mdl_pos z
  if (!$scope.app.params.mz_mdl_Flugzeug){$scope.app.params.rx_mdl_Flugzeug =
0}        // mdl_rot x

$scope.$watch("app.params['Drehzahl']",function()                        //
call function when value changes
{
  if ($scope.app.params.Drehzahl > 0){        //check propeller rotation
    rotate(true);
      $scope.view.wdg['lab_Info'].text = 'Motor läuft';
      $scope.view.wdg['but_Takeoff'].visible = true;
  }
  else {
    rotate (false);
      $scope.view.wdg['lab_Info'].text = 'Motor aus';
      $scope.view.wdg['but_Takeoff'].visible = false;
  }
})

$scope.click_but_Takeoff = function () {
move(true);
}
$scope.click_but_Home = function () {
move(false);
}

function rotate (active){
console.log('function call -rotate- '+ active);
if (active && timerId_rot == -1){
 timerId_rot = setInterval(function() {
   $scope.$apply(function(){              // code called every timingIntervall
     if ($scope.app.params.rz_mdlit_Luftschraube >=
360){$scope.app.params.rz_mdlit_Luftschraube =0}
    $scope.app.params.rz_mdlit_Luftschraube += 10;
  })
 }, timingIntervall_rot)
```

```
}
if(!active){
 clearInterval(timerId_rot);
   timerId_rot=-1;
}
}

function move (active){
console.log('function call -move- '+ active);
if (active && timerId_mov == -1){
 timerId_mov = setInterval(function() {
   $scope.$apply(function(){
  if ($scope.app.params.my_mdl_Flugzeug<starthight_loop){ // climb to desired
hight
        $scope.app.params.mz_mdl_Flugzeug += climb_mz;
        $scope.app.params.my_mdl_Flugzeug += climb_my;
        mzstart_loop = $scope.app.params.mz_mdl_Flugzeug;
        mystart_loop = $scope.app.params.my_mdl_Flugzeug;
      }
      else{
        phi_loop=phi_loop + (omega_loop*timingIntervall_rot/1000);
//looping
          $scope.app.params.mz_mdl_Flugzeug = mzstart_loop +
r_loop*Math.cos(phi_loop + Math.PI*3/2);
          $scope.app.params.my_mdl_Flugzeug = mystart_loop +
r_loop*Math.sin(phi_loop + Math.PI*3/2) + r_loop;
          $scope.app.params.rx_mdl_Flugzeug = -1*phi_loop/Math.PI*180;    //
rotate airplane
      }
  })
 }, timingIntervall_mov)
}
if(!active){                // return home
 clearInterval(timerId_mov);
   timerId_mov=-1;
   $scope.app.params.mz_mdl_Flugzeug = 0;
  $scope.app.params.my_mdl_Flugzeug = 0;
  $scope.app.params.rx_mdl_Flugzeug = 0;
}
}
```