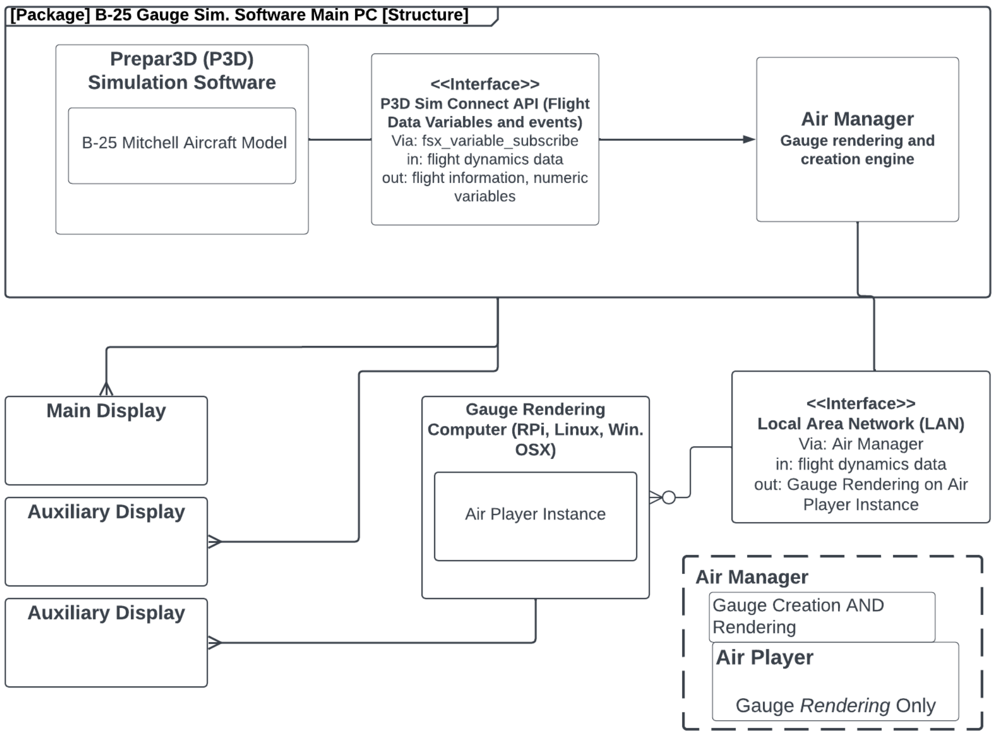
Sim Documentation

2022-2023 Design Team
UCCS

Table of Contents

Introduction

The intention of this document and its supporting code and referenced documentation is intended to equip the user with a comprehensive solution for creating a simulation interface between a professional flight simulator and a B-25 Mitchell cockpit. To accomplish this goal, the flight simulator itself needs to be able to work with multiple GPU's to allow for the required number of monitors to be utilized, along with external input sources such as switches and buttons being interpolated through Arduinos. A diagram of these different parts is shown in the diagram below.


[Package] B-25 Gauge Sim. Software Main PC [Structure]
Prepar3D (P3D) Simulation Software
B-25 Mitchell Aircraft Model
<<Interface>>
P3D Sim Connect API (Flight Data Variables and events)
Via: fsx_variable_subscribe
in: flight dynamics data
out: flight information, numeric variables
Air Manager
Gauge rendering and creation engine
Main Display
Auxiliary Display
Auxiliary Display
Gauge Rendering Computer (RPi, Linux, Win. OSX)
Air Player Instance
<<Interface>>
Local Area Network (LAN)
Via: Air Manager
in: flight dynamics data
out: Gauge Rendering on Air Player Instance
Air Manager
Gauge Creation AND Rendering
Air Player
Gauge Rendering Only

In the following sections of this document, each aspect of the simulator is discussed in detail as to how they were created, how to change and edit what has been provided, and guides on how to integrate each software package and common hardware item. Accompanying code has also been provided along with this document including custom gauges for the B-25, and Arduino input-output code. The gauges should be easily integrated into the final simulator in their current form, and the Arduino code will provide a strong code base for when the wiring is being completed at a later stage.

Air Manager

Gauge Creation Guide:

Objective:

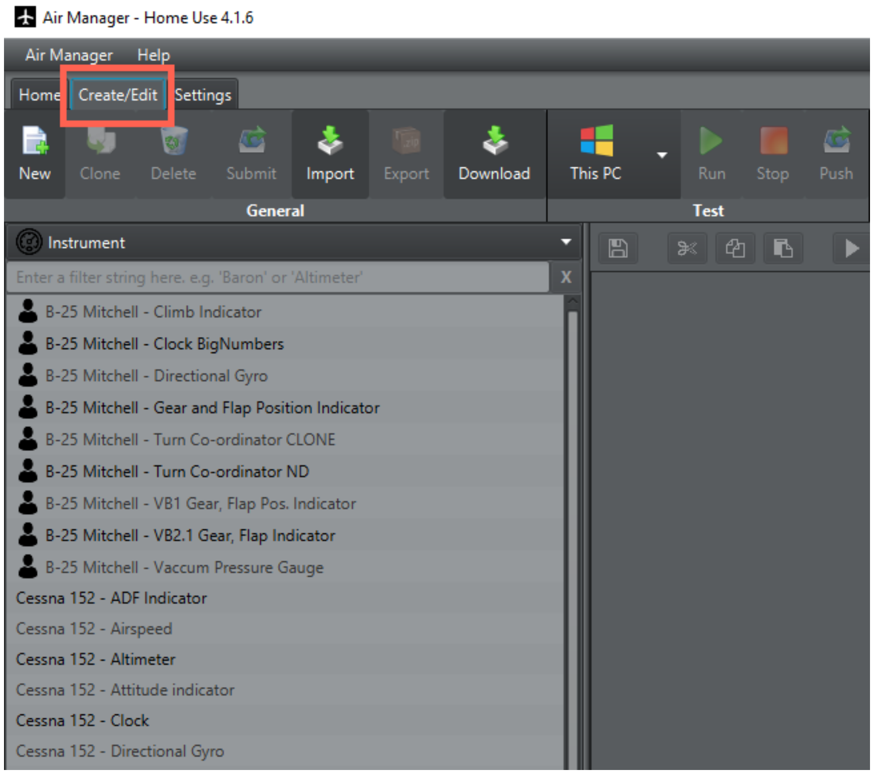
The purpose of this documentation is to record how to make custom gauges that one would want to display on a screen using Air Manager.

Equipment used:

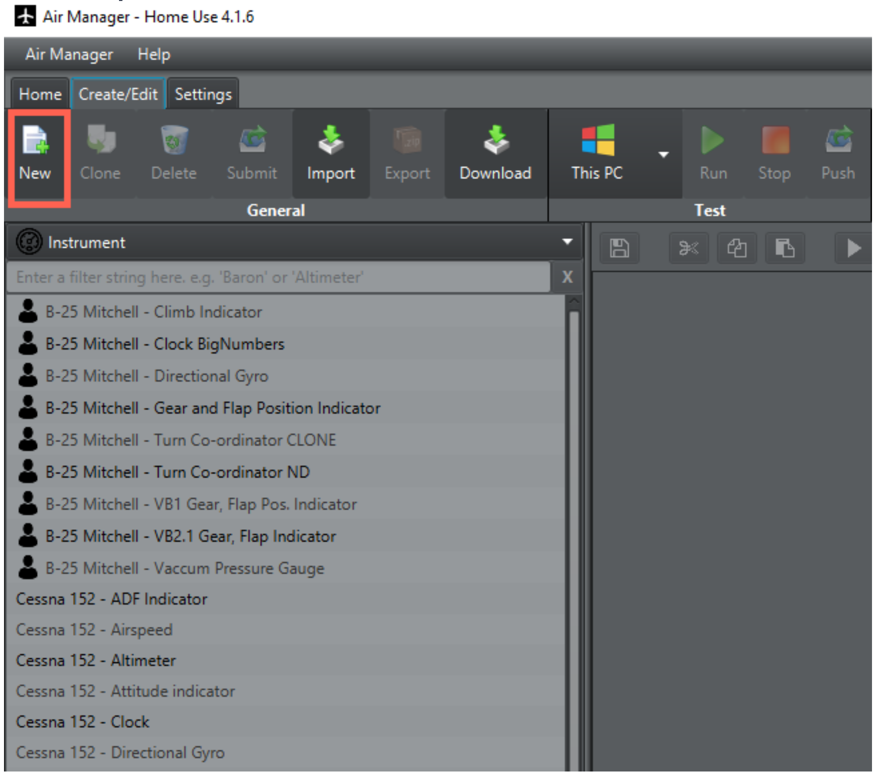
- Air Manager
- Computer
- Gimp 2.0
- VS Code (optional)

Steps:

1. Turn on Air Manager.
2. Go to the Create/Edit tab.

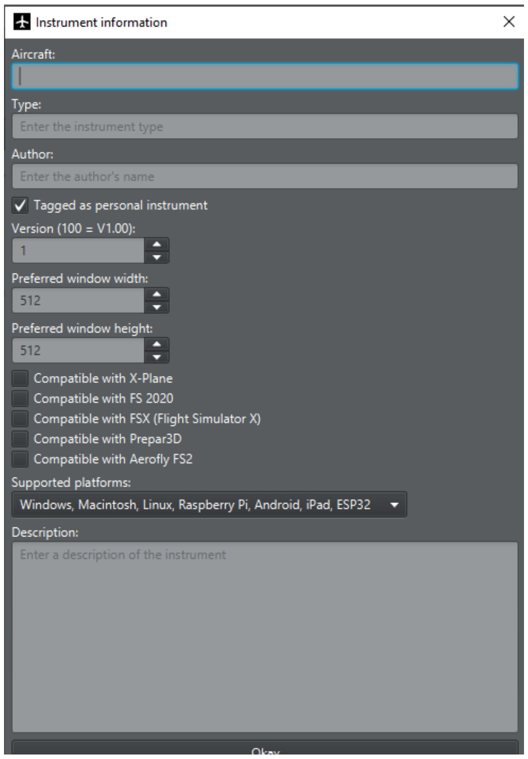

Air Manager - Home Use 4.1.6
Air Manager
Help
Home
Create/Edit
Settings
New
Clone
Delete
Submit
Import
Export
Download
This PC
Run
Stop
Push
General
Test
Instrument
Enter a filter string here. e.g. 'Baron' or 'Altimeter'
X
B-25 Mitchell - Climb Indicator
B-25 Mitchell - Clock BigNumbers
B-25 Mitchell - Directional Gyro
B-25 Mitchell - Gear and Flap Position Indicator
B-25 Mitchell - Turn Co-ordinator CLONE
B-25 Mitchell - Turn Co-ordinator ND
B-25 Mitchell - VB1 Gear, Flap Pos. Indicator
B-25 Mitchell - VB2.1 Gear, Flap Indicator
B-25 Mitchell - Vaccum Pressure Gauge
Cessna 152 - ADF Indicator
Cessna 152 - Airspeed
Cessna 152 - Altimeter
Cessna 152 - Attitude indicator
Cessna 152 - Clock
Cessna 152 - Directional Gyro

3. Click New at the top left.



B-25 Mitchell - Climb Indicator
B-25 Mitchell - Clock BigNumbers
B-25 Mitchell - Directional Gyro
B-25 Mitchell - Gear and Flap Position Indicator
B-25 Mitchell - Turn Co-ordinator CLONE
B-25 Mitchell - Turn Co-ordinator ND
B-25 Mitchell - VB1 Gear, Flap Pos. Indicator
B-25 Mitchell - VB2.1 Gear, Flap Indicator
B-25 Mitchell - Vaccum Pressure Gauge
Cessna 152 - ADF Indicator
Cessna 152 - Airspeed
Cessna 152 - Altimeter
Cessna 152 - Attitude indicator
Cessna 152 - Clock
Cessna 152 - Directional Gyro

4. Fill out all the information that is needed when the new window is created.



Instrument information

Aircraft:

Type: Enter the instrument type

Author: Enter the author's name

✓ Tagged as personal instrument

Version (100 = V1.00): 1

Preferred window width: 512

Preferred window height: 512

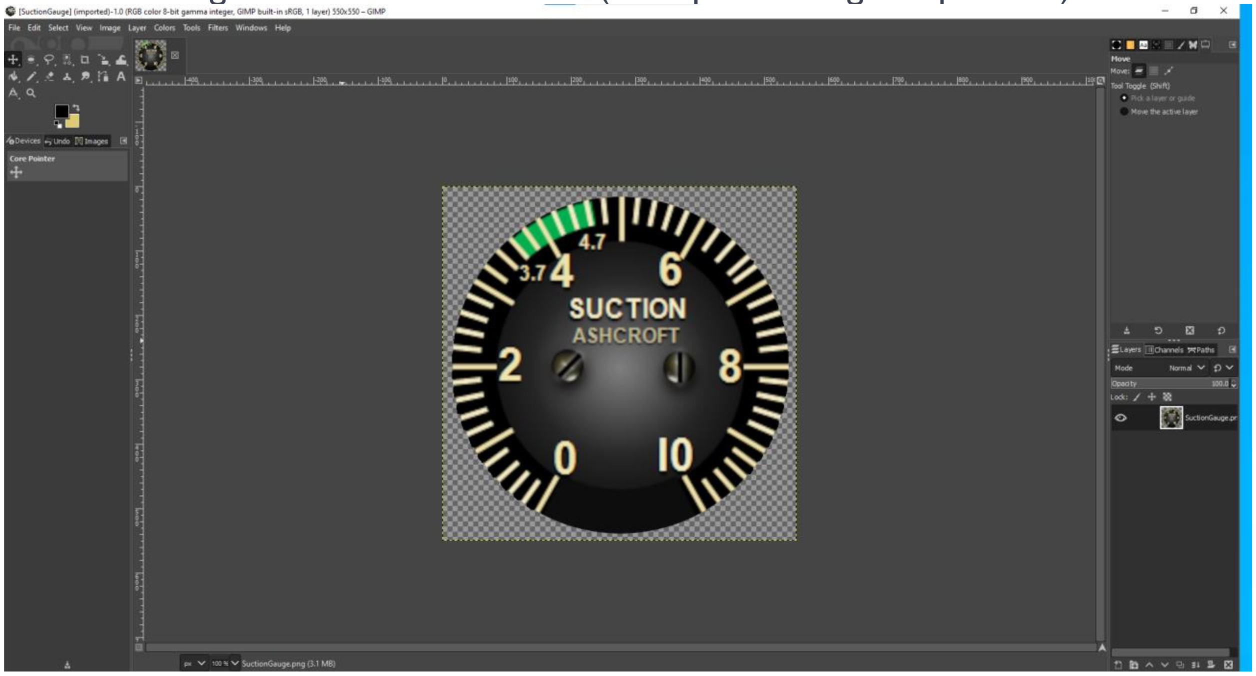
☐ Compatible with X-Plane
☐ Compatible with FS 2020
☐ Compatible with FSX (Flight Simulator X)
☐ Compatible with Prepar3D
☐ Compatible with Aerofly FS2

Supported platforms: Windows, Macintosh, Linux, Raspberry Pi, Android, iPad, ESP32

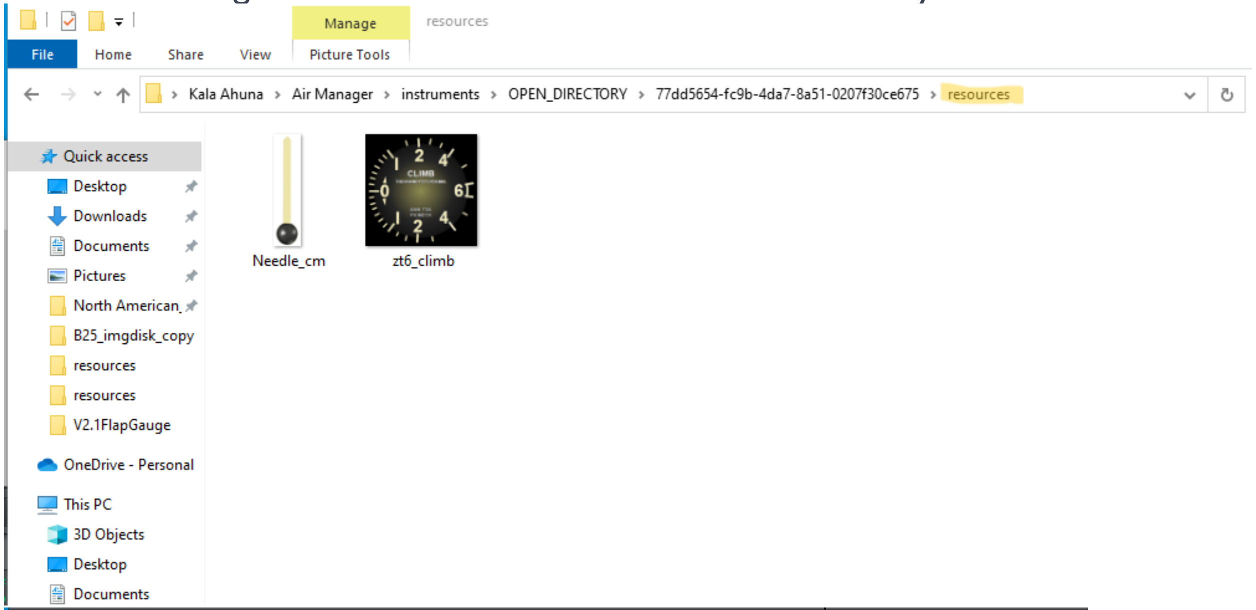
Description: Enter a description of the instrument

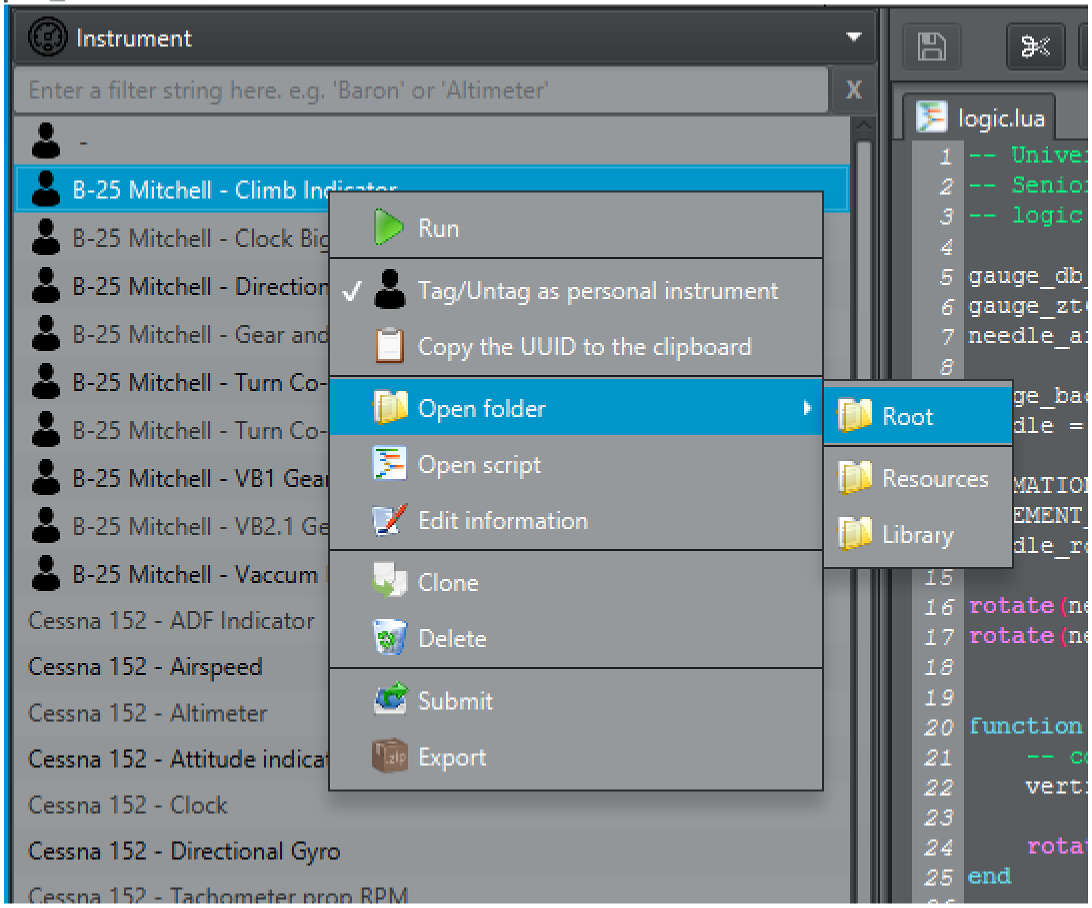
Okay

5. Know what you want your gauge to look like. It helps to have an image that you know you want to manipulate. Use Gimp or any other photo manipulation tool that you feel comfortable using. Photo must be a .PNG (Example of using Gimp below)

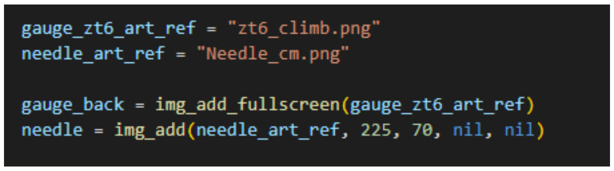

[SuctionGauge] (imported)-1.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 1 layer) 550x550 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
SUCTION
ASHCROFT
SuctionGauge.png (3.1 MB)

6. Put the saved image in gauges resources directory of your Air Manager gauge. Refer to the second image below on how to access the root directory.

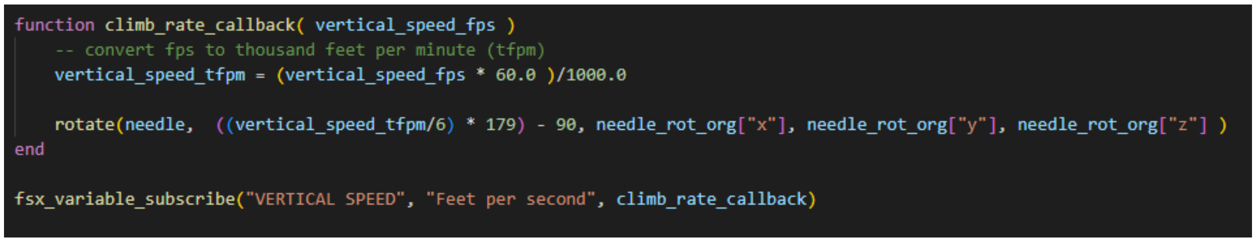



Instrument
Enter a filter string here. e.g. 'Baron' or 'Altimeter'
B-25 Mitchell - Climb Indicator
B-25 Mitchell - Clock Big
B-25 Mitchell - Direction
B-25 Mitchell - Gear and
B-25 Mitchell - Turn Co-
B-25 Mitchell - Turn Co-
B-25 Mitchell - VB1 Gear
B-25 Mitchell - VB2.1 Ge
B-25 Mitchell - Vaccum
Cessna 152 - ADF Indicator
Cessna 152 - Airspeed
Cessna 152 - Altimeter
Cessna 152 - Attitude indicat
Cessna 152 - Clock
Cessna 152 - Directional Gyro
Cessna 152 - Tachometer prop RPM
Run
Tag/Untag as personal instrument
Copy the UUID to the clipboard
Open folder
Open script
Edit information
Clone
Delete
Submit
Export
Root
Resources
Library
logic.lua

7. Begin coding the gauge Lua is the coding language. (The images below are examples of the source code).

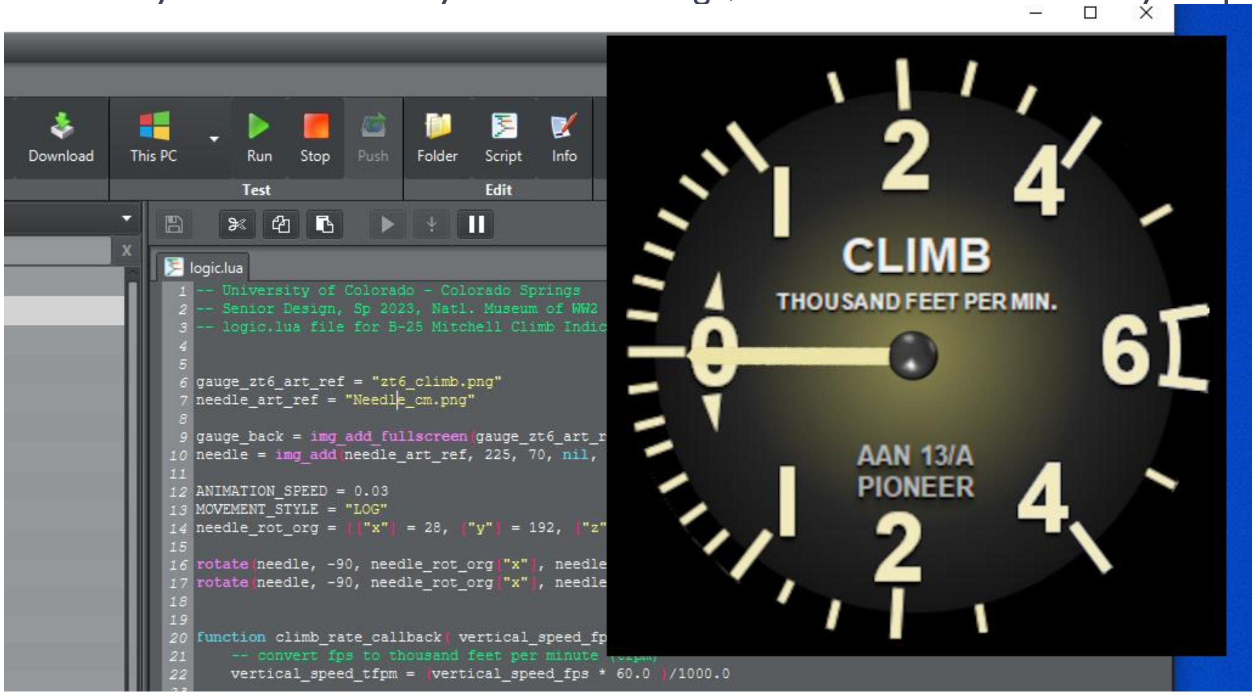


```lua
gauge_zt6_art_ref = "zt6_climb.png"
needle_art_ref = "Needle_cm.png"

gauge_back = img_add_fullscreen(gauge_zt6_art_ref)
needle = img_add(needle_art_ref, 225, 70, nil, nil)
```



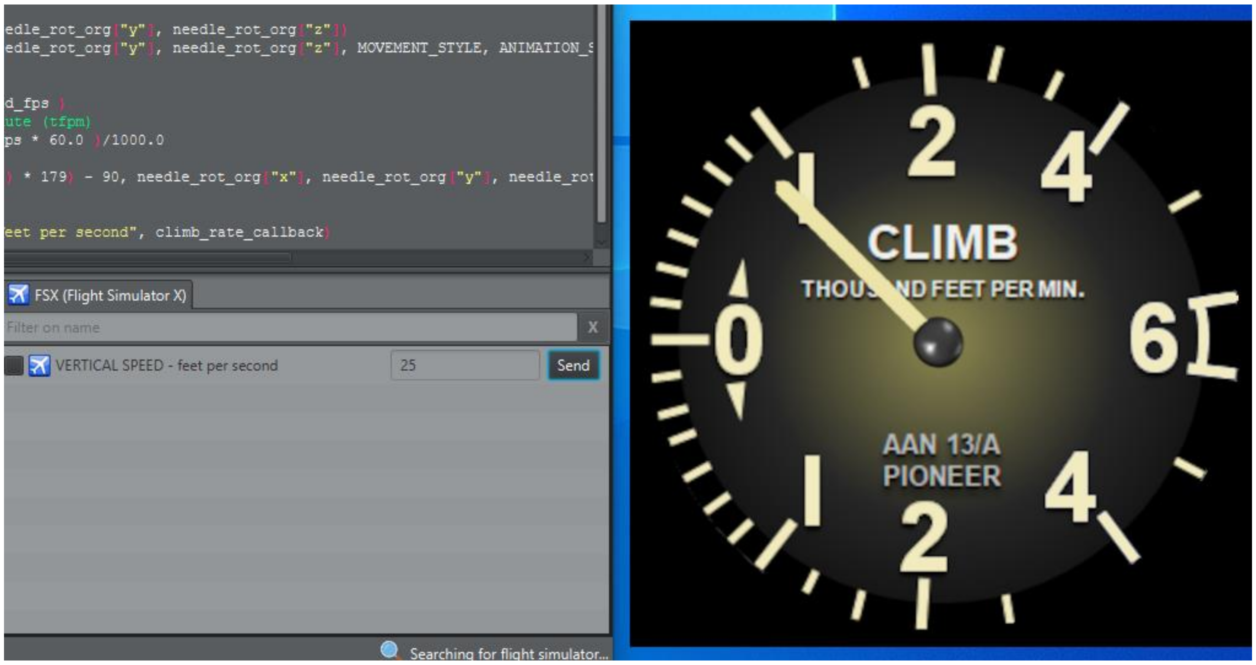
```lua
function climb_rate_callback( vertical_speed_fps )
    -- convert fps to thousand feet per minute (tfpm)
    vertical_speed_tfpm = (vertical_speed_fps * 60.0 )/1000.0

    rotate(needle,  ((vertical_speed_tfpm/6) * 179) - 90, needle_rot_org["x"], needle_rot_org["y"], needle_rot_org["z"] )
end

fsx_variable_subscribe("VERTICAL SPEED", "Feet per second", climb_rate_callback)
```

8. When you feel that what you have is enough, feel free to run it and see your progress.



```lua
-- University of Colorado - Colorado Springs
-- Senior Design, Sp 2023, Natl. Museum of WW2
-- logic.lua file for B-25 Mitchell Climb Indic


gauge_zt6_art_ref = "zt6_climb.png"
needle_art_ref = "Needle_cm.png"

gauge_back = img_add_fullscreen(gauge_zt6_art_r
needle = img_add(needle_art_ref, 225, 70, nil,

ANIMATION_SPEED = 0.03
MOVEMENT_STYLE = "LOG"
needle_rot_org = {["x"] = 28, ["y"] = 192, ["z"

rotate(needle, -90, needle_rot_org["x"], needle
rotate(needle, -90, needle_rot_org["x"], needle


function climb_rate_callback( vertical_speed_fp
    -- convert fps to thousand feet per minute
    vertical_speed_tfpm = (vertical_speed_fps * 60.0 )/1000.0
```

9. You can also test to see if your gauge works by inputting test values once you hit the run button.



```
edle_rot_org["y"], needle_rot_org["z"])
edle_rot_org["y"], needle_rot_org["z"], MOVEMENT_STYLE, ANIMATION_S

d_fps )
ute (tfpm)
ps * 60.0 )/1000.0

) * 179) - 90, needle_rot_org["x"], needle_rot_org["y"], needle_rot

eet per second", climb_rate_callback)
```

10. Once your custom gauge is up to your satisfaction, you can move onto your next gauge or if that was your last one, you can be done. It is also recommended, but not necessary that you add a preview image.

References:

- Adding Images to Gauges:
  - (Adding Images to Guages)
  - (Adding Full Size Images for Guages)
- Manipulating Images (needles, indicators, etc.):
  - (Creating Rotating Needles)
  - (Moving Indicator Images)
- Referencing Variables:
  - Note: P3D shares the same data link subscription API as Flight Sim X. But the variable names are different between P3D and FSX.
    - (Connecting Variables to Guage Behavior/Animation)
    - (Prepar-3D Flight Dynamics Variables)

Air Player integration

Objective:

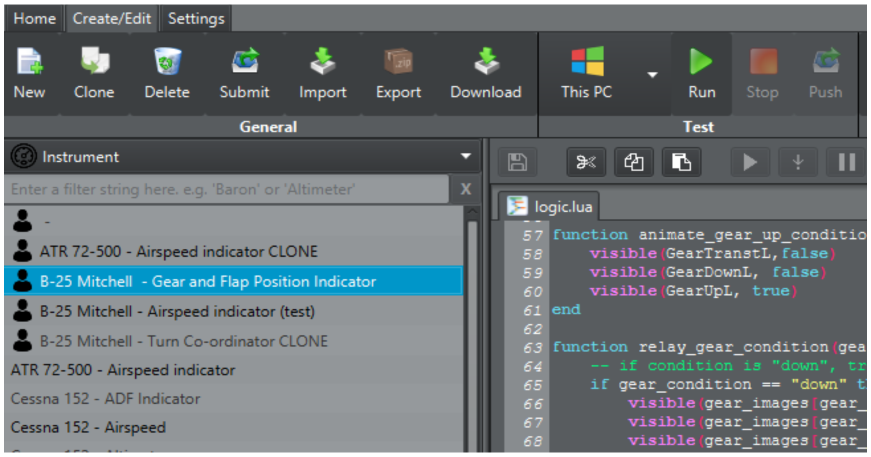
To use Prepare3d and Air Manager on a computer to send a custom gauge to a Raspberry Pi with Air Manager on it and have it display the desired gauge layout on the Pi's screen.

Equipment used:

Raspberry Pi 4b, with 32-bit OS
Screen Interfaced with Raspberry Pi
keyboard and Mouse
Prepare3d
Air Player
Air Manager

Steps:

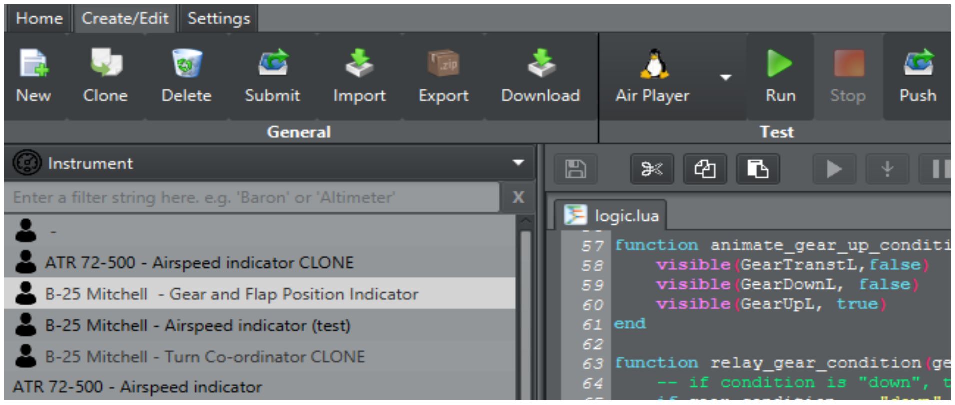
1. Download Air Player on the Pi you wish to use.
2. Have Prepare3d and Air Manager on and open on the PC. Make sure they are connected via a LAN network.
3. Go to the Create/Edit tab on Air Manager.
4. When you have your custom gauge the way you want it to, save it and you're ready to move on.
5. Have your custom gauge file selected and change "This PC" to "Air Player" (Images below).



-
ATR 72-500 - Airspeed indicator CLONE
B-25 Mitchell - Gear and Flap Position Indicator
B-25 Mitchell - Airspeed indicator (test)
B-25 Mitchell - Turn Co-ordinator CLONE
ATR 72-500 - Airspeed indicator
Cessna 152 - ADF Indicator
Cessna 152 - Airspeed

```
function animate_gear_up_conditio
    visible(GearTranstL, false)
    visible(GearDownL, false)
    visible(GearUpL, true)
end

function relay_gear_condition(gea
    -- if condition is "down", tr
    if gear_condition == "down" t
        visible(gear_images[gear_
        visible(gear_images[gear_
        visible(gear_images[gear_
```

6. With your file selected, select the "Push" button that is three buttons right of "Air Player"



-
ATR 72-500 - Airspeed indicator CLONE
B-25 Mitchell - Gear and Flap Position Indicator
B-25 Mitchell - Airspeed indicator (test)
B-25 Mitchell - Turn Co-ordinator CLONE
ATR 72-500 - Airspeed indicator

```
function animate_gear_up_conditi
    visible(GearTranstL,false)
    visible(GearDownL, false)
    visible(GearUpL, true)
end

function relay_gear_condition(ge
    -- if condition is "down", t
```

7. When ready, select the “Run” button to see the gauge on the screen connectePi (You might have to adjust the position of gauge to where you want.)

References:

(Sim Innovations Main Page)
(Air Player Installation)
(Air Player Raspberry Pi Manual)
(Air Manager User Manual)

Physical-Digital interfacing

Interfacing with Prepar3D and Air Manager

When creating physical/digital interfacing using Arduinos, there are two options of integration, both of which will be documented. The first is using the Arduino Leonardo or other 32u4 or SAMD processor boards. These specific model of Arduino processors can send keystroke commands directly to a computer. These keystrokes can be any legal Extended ASCII character and act the same as if a user had pressed the same key on a keyboard. By combining this with custom key binds in Prepar3D, inputs can be effectively simulated. The only limitation on the number of boards used in this method is the number of USB ports present upon the computer running Prepar3D. This method is best applied for Boolean inputs, such as switches and buttons. Variable inputs such as levers can also be simulated as well. **There is no upper limit** to the number of Arduino's that can be networked in this way, save for the physical data transfer limitations of the computer that they are integrated with.

The second option is by interfacing with AirManager using their custom library *SiMessagePort*. This allows for direct values to be passed between AirManager and the attached Arduino. The boards that are supported by this program are the Arduino MEGA 2560, Uno, Micro, Leonardo and Nano. It is important to note that unlike the previous option, **only a single Arduino can be interfaced in this fashion**, so the Arduino MEGA 2560 is the best option for the breadth of inputs that it provides. It is important to note that this is an output medium rather than an input medium and is intended to be used to trigger lights or movement in the cockpit. The specific code does not change from board to board, only the number of inputs available to the coder. Both methods (Prepar3D and AirManager integration) are 100% compatible and can be used in tandem with each other. Both require a physical USB to µUSB cable connection with the machine that they are interfacing with.

Hardware design decisions

Arduino Leonardo Vs. Other Arduino Boards Vs Premade Pedals, flight controls

- Low current
    - Arduino triggers off of voltage, not current
    - Simple circuits desired
- Control yokes would be better created by modifying existing flight sim hardware rather than from scratch
- Anything variable is a potentiometer
- Anything binary is either a button or a switch
- Common grounding is necessary

Arduino Code Examples, Schematics, and application notes for Arduino Leonardo

This document assumes a working knowledge of the Arduino programming language and general programming convention. For those not aquatinted with either, it is recommended that they reference the "Arduino Tutorials" link in the references subsection. When implementing circuits like the examples provided below or otherwise, it is important to review the electrical characteristics of the device in the datasheet provided by the manufacturer of the microprocessor. MOST IMPORTANTLY, current and voltage maximums for the board must be followed strictly to avoid damage to the Arduino. For the Arduino Leonardo, the input or output of a single pin cannot exceed 20mA, and the combined input of all pins cannot exceed 100mA. Similarly, the board cannot handle input voltage levels exceeding 12V. The voltage thresholds for HIGH (V_{IH}) and LOW (V_{IL}) voltage states are also variable from processor to processor, with the Leonardo having a V_{IH} of between 2 and 5V (It is recommended that the voltage in be as low as possible) and a V_{OH} of GND. Common ground is highly recommended when integrating Arduino circuits.

- Interfacing with Prepar-3D Best Practice
  - When creating custom key binds, it is imperative that the checkbox labeled "On Release" is UNCHECKED. For some reason, Prepar-3D will not read the Arduino's keyboard inputs if this is enabled.
  - Do not connect the Arduino(s) until after Prepar-3D has been fully launched and loaded. Else the key binds will cause whatever currently selected program to execute them instead.
  - Prepar-3D can take any singular or double sequence of button presses

- Key Presses

The Arduino Leonardo Sends keypresses through the Arduino “keyboard” library. This library provides several useful functions, which are provided along with a brief description below:

Command:	Description:
Keyboard.begin()	Opens communication with connected computer
Keyboard.end()	Ends connection with connected computer
Keyboard.press()	Sends a keystroke and will continue to hold down the key. Can be coded to send either ascii values or have their characters directly typed into the function.
Keyboard.release()	Releases held key

The possible range of keypresses is near limitless. By default, the extended ascii library, plus additional special keys are included. The names of these function keys have been provided in the table below. An even more extended library of international and specialty characters is possible to integrate as well, however the integration is considered outside of the scope of this project and therefore has been omitted. Additional information can be found in references under “Keyboard modifiers and special keys.”

Key	**Hexadecimal value**	**Decimal value**	**Notes**
KEY_LEFT_CTRL	0x80	128	
KEY_LEFT_SHIFT	0x81	129	
KEY_LEFT_ALT	0x82	130	Option (⌥) on Mac
KEY_LEFT_GUI	0x83	131	OS logo, Command (⌘) on Mac
KEY_RIGHT_CTRL	0x84	132	
KEY_RIGHT_SHIFT	0x85	133	
KEY_RIGHT_ALT	0x86	134	also AltGr, Option (⌥) on Mac
KEY_RIGHT_GUI	0x87	135	OS logo, Command (⌘) on Mac
KEY_TAB	0xB3	179	
KEY_CAPS_LOCK	0xC1	193	
KEY_BACKSPACE	0xB2	178	
KEY_RETURN	0xB0	176	
KEY_MENU	0xED	237	
KEY_INSERT	0xD1	209	
KEY_DELETE	0xD4	212	
KEY_HOME	0xD2	210	
KEY_END	0xD5	213	
KEY_PAGE_UP	0xD3	211	
KEY_PAGE_DOWN	0xD6	214	
KEY_UP_ARROW	0xDA	218	
KEY_DOWN_ARROW	0xD9	217	
KEY_LEFT_ARROW	0xD8	216	

KEY_RIGHT_ARROW	0xD7	215	
KEY_NUM_LOCK	0xDB	219	
KEY_KP_SLASH	0xDC	220	
KEY_KP_ASTERISK	0xDD	221	
KEY_KP_MINUS	0xDE	222	
KEY_KP_PLUS	0xDF	223	
KEY_KP_ENTER	0xE0	224	
KEY_KP_1	0xE1	225	
KEY_KP_2	0xE2	226	
KEY_KP_3	0xE3	227	
KEY_KP_4	0xE4	228	
KEY_KP_5	0xE5	229	
KEY_KP_6	0xE6	230	
KEY_KP_7	0xE7	231	
KEY_KP_8	0xE8	232	
KEY_KP_9	0xE9	233	
KEY_KP_0	0xEA	234	
KEY_KP_DOT	0xEB	235	
KEY_ESC	0xB1	177	
KEY_F1	0xC2	194	
KEY_F2	0xC3	195	
KEY_F3	0xC4	196	
KEY_F4	0xC5	197	
KEY_F5	0xC6	198	
KEY_F6	0xC7	199	
KEY_F7	0xC8	200	
KEY_F8	0xC9	201	
KEY_F9	0xCA	202	
KEY_F10	0xCB	203	
KEY_F11	0xCC	204	
KEY_F12	0xCD	205	
KEY_F13	0xF0	240	
KEY_F14	0xF1	241	
KEY_F15	0xF2	242	
KEY_F16	0xF3	243	
KEY_F17	0xF4	244	
KEY_F18	0xF5	245	
KEY_F19	0xF6	246	
KEY_F20	0xF7	247	
KEY_F21	0xF8	248	
KEY_F22	0xF9	249	
KEY_F23	0xFA	250	
KEY_F24	0xFB	251	
KEY_PRINT_SCREEN	0xCE	206	Print Screen or PrtSc / SysRq
KEY_SCROLL_LOCK	0xCF	207	
KEY_PAUSE	0xD0	208	Pause / Break

- Buttons (Push Switches)

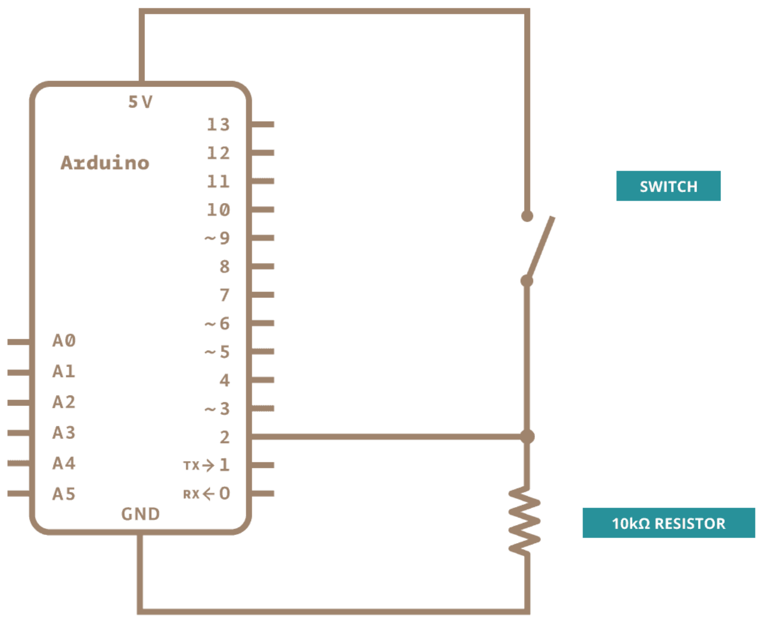
Pullup Vs Pulldown in Arduino:

Arduino µcontrollers are internally wired as pullup resistors. Meaning that they will throw FALSE (also equivalent to 0 or LOW in the Arduino language) and TRUE (also equivalent to 1 or HIGH in the Arduino code) when connected to VCC on any configured input pin. This is why when working with complex Arduino-driven circuits it is important to have a common ground across the entire circuit.

Example Code:

```
#include <Keyboard.h>

const int buttonPin = 2;  // the number of the pushbutton pin
int buttonState = 0;  // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
  //start keyboard connection
  Keyboard.begin();
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == HIGH) {


        Keyboard.releaseAll();
        Keyboard.press(KEY_LEFT_CTRL);
        Keyboard.press('e');
        delay (500);
        Keyboard.releaseAll();
    //}
    delay(2000);


  } else {
    delay(100);
  }
}
```

Diagram of associated circuit:

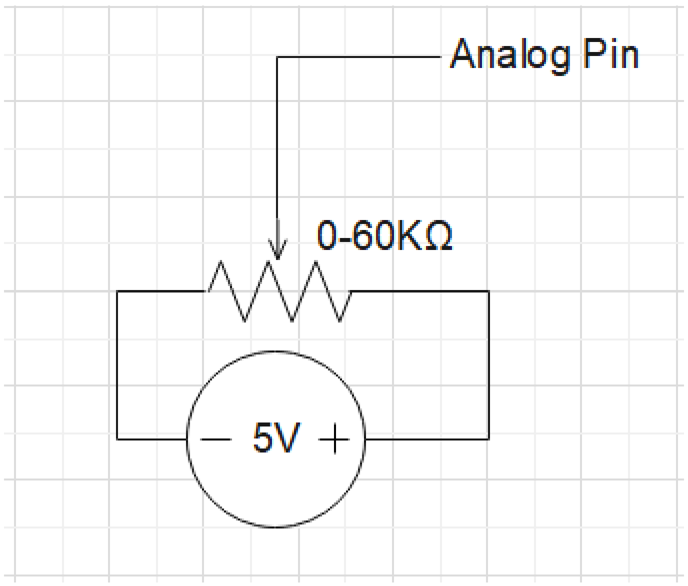


- Potentiometers

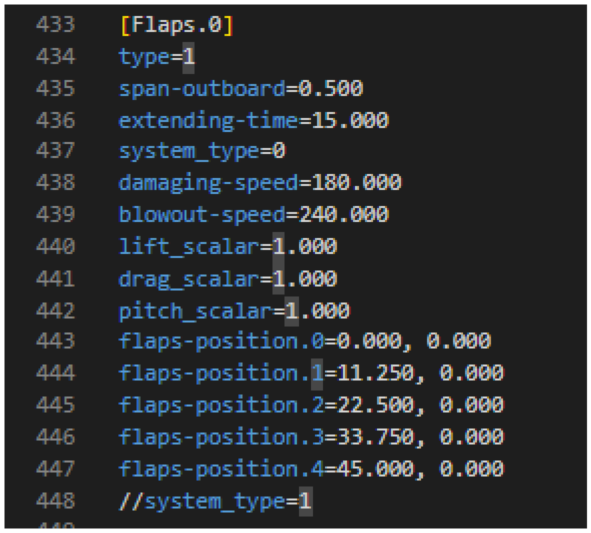
The Arduino works on a linear percentage scale of read voltage between 0 and VCC, returning a value between 0 and 1023. This code then takes that value, turns it into a percentage of flaps (0-45 degrees), finds the closest preprogrammed flap step, and then sends the appropriate number of key inputs.

```
#include<Keyboard.h>

int potPin = A0; // potentiometer is connected to analog 0 pin
int currentStep = 0; // variable used to store the value coming from the sensor
float percent; // variable used to store the chosen flap percentage value (0-45 deg)
int sensorValue;//value the analog pin reads (integers 0-1023)
float percentVals[] = {0, 11.25, 22.5, 33.75, 45}; //stored value steps (0-45 deg)
//for more precise steps, this array could be generated by a for loop.

void setup() {
  Keyboard.begin();
}

void loop() {
 // put your main code here, to run repeatedly:
  sensorValue = analogRead(A0);
  percent = sensorValue * (45.0 / 1023.0);//turn percent value to degrees
  if(percent> percentVals[currentStep]){//if number is greater than current step
    if(percent-percentVals[currentStep] > abs(percent-percentVals[currentStep+1])){
      //comparing which percent step is closer
        currentStep++;//increase current step
        Keyboard.press(KEY_F7);//extend flaps to next step
        delay (500);
      Keyboard.releaseAll();
    }
  }
 else if (percent < percentVals[currentStep]){ //if number is less than current step
    if(abs(percent-percentVals[currentStep]) > abs(percent-percentVals[currentStep-1])){
      //comparing which percent step is closer
        currentStep--;//decrease current step
        Keyboard.press(KEY_F6);//retract flaps to next step
        delay (500);
        Keyboard.releaseAll();
    }
  }
}
```

Associated Potentiometer Circuit:



In Prepar3D, for everything that can be incremented and decremented through keypresses (such as flaps, trim, or throttle) the number of increments and what angle/value they go to is preprogrammed in the aircrafts .cfg file. For example, below is the code defining the flap characteristics for the B-25.



```
[Flaps.0]
type=1
span-outboard=0.500
extending-time=15.000
system_type=0
damaging-speed=180.000
blowout-speed=240.000
lift_scalar=1.000
drag_scalar=1.000
pitch_scalar=1.000
flaps-position.0=0.000, 0.000
flaps-position.1=11.250, 0.000
flaps-position.2=22.500, 0.000
flaps-position.3=33.750, 0.000
flaps-position.4=45.000, 0.000
//system_type=1
```

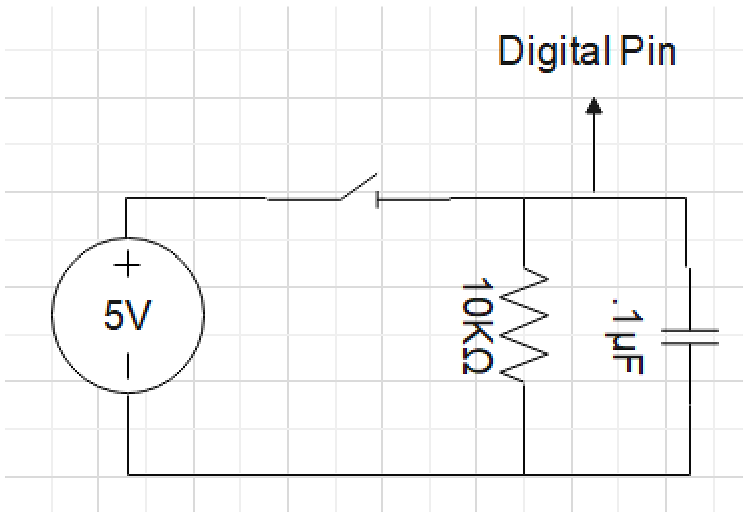
When looking at the definition of flaps-position.n (where n is an integer), each number entered is the angle in degrees the flaps will extend to, and each step is one that will be cycled to when the “increment/decrement flaps” key bind is pressed. Additional steps can be programmed in and there is no upper limit of flap positions possible.

- Switches

The code for switches is almost the same as that for a button. In this circuit a capacitor is included to decouple to ground to remove bounce but is not necessary if an appropriate delay is already programmed into the code.

```
#include <Keyboard.h>

const int buttonPin = 2;  // the number of the pushbutton pin
int buttonState = 0;  // variable for reading the pushbutton status
char ctrlKey = KEY_LEFT_CTRL;

void setup() {
  // initialize the LED pin as an output:
 // pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
  //start keyboard connection
  Keyboard.begin();
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == HIGH) {


        Keyboard.releaseAll();
        Keyboard.press(KEY_F8);
        delay (500);
        Keyboard.releaseAll();

    delay(100);


  } else {
       Keyboard.releaseAll();
        Keyboard.press(KEY_F5);
        delay (500);
        Keyboard.releaseAll();
  }
}
```

Associated Switch circuit



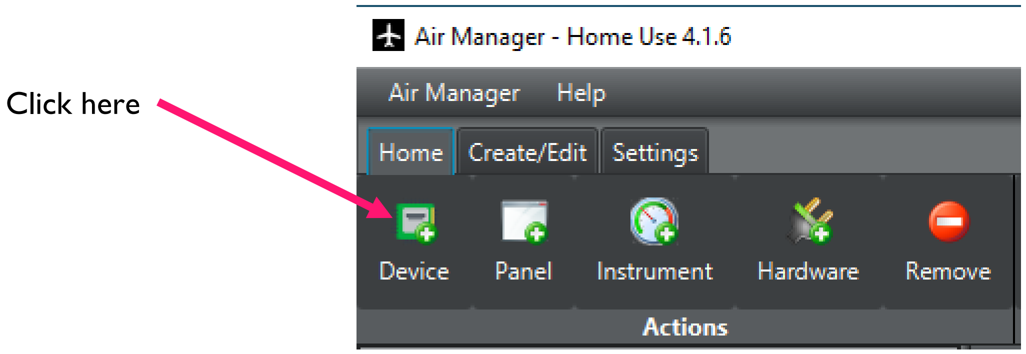
Arduino/Air manager interfacing

Air manager can flash code to the Arduino and automatically interface with the Arduino to simulate button, switch, and potentiometer inputs. This ability is limited only to a singular connected Arduino which must be specifically called in the code of air manager. The following table displays the compatible board types and their internal ID's in Air-managers code.

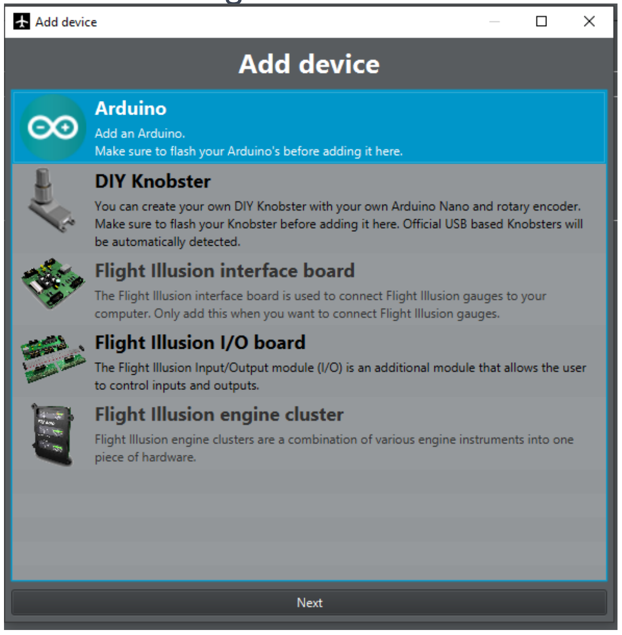
Board Type	ID
Mega 256	ARDUINO_MEGA_2560_X
Nano	ARDUINO_NANO_X
Uno	ARDUINO_UNO_X
Micro	ARDUINO_MICRO_X
Leonardo	ARDUINO_LEONARDO_X

The 'X' after the board definition defines which data channel will be used. A total of 16 data lines are available for use, defined as lines A through P.

To add an Arduino to Air-player the following steps need to be performed:
Navigate to Home > Device



Select the Arduino device you will be using from the list



Arduino
Add an Arduino.
Make sure to flash your Arduino's before adding it here.

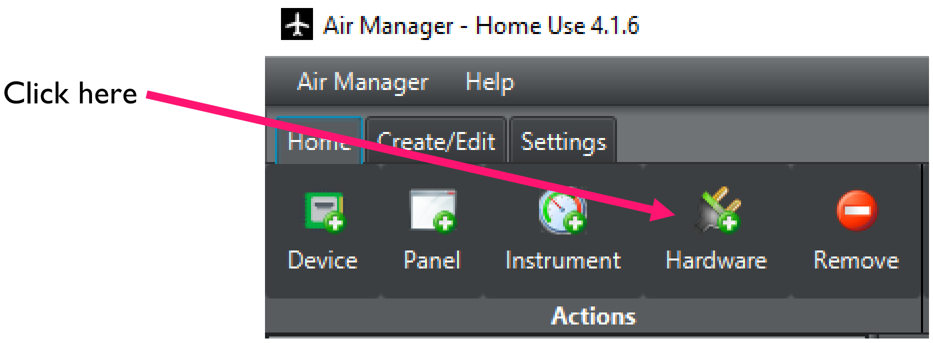
DIY Knobster
You can create your own DIY Knobster with your own Arduino Nano and rotary encoder. Make sure to flash your Knobster before adding it here. Official USB based Knobsters will be automatically detected.

Flight Illusion interface board
The Flight Illusion interface board is used to connect Flight Illusion gauges to your computer. Only add this when you want to connect Flight Illusion gauges.

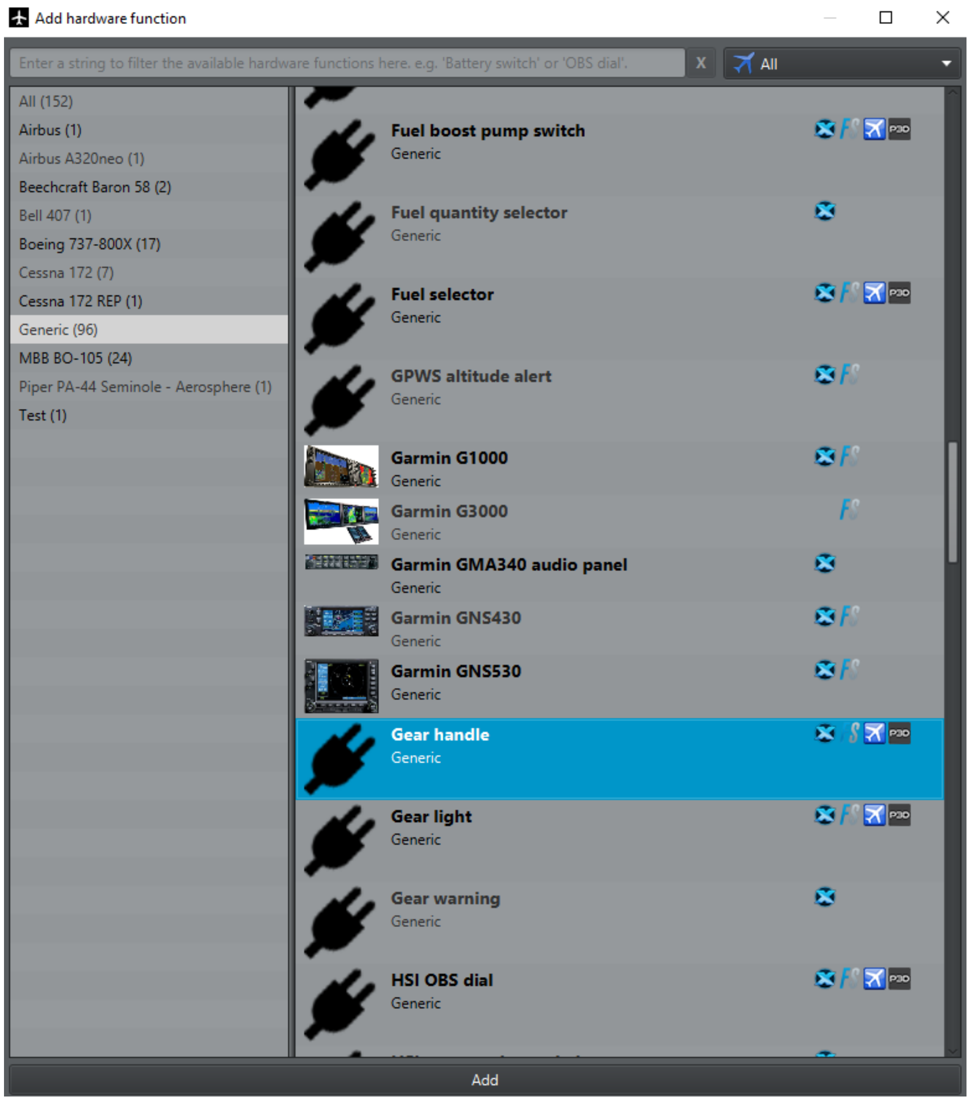
Flight Illusion I/O board
The Flight Illusion Input/Output module (I/O) is an additional module that allows the user to control inputs and outputs.

Flight Illusion engine cluster
Flight Illusion engine clusters are a combination of various engine instruments into one piece of hardware.

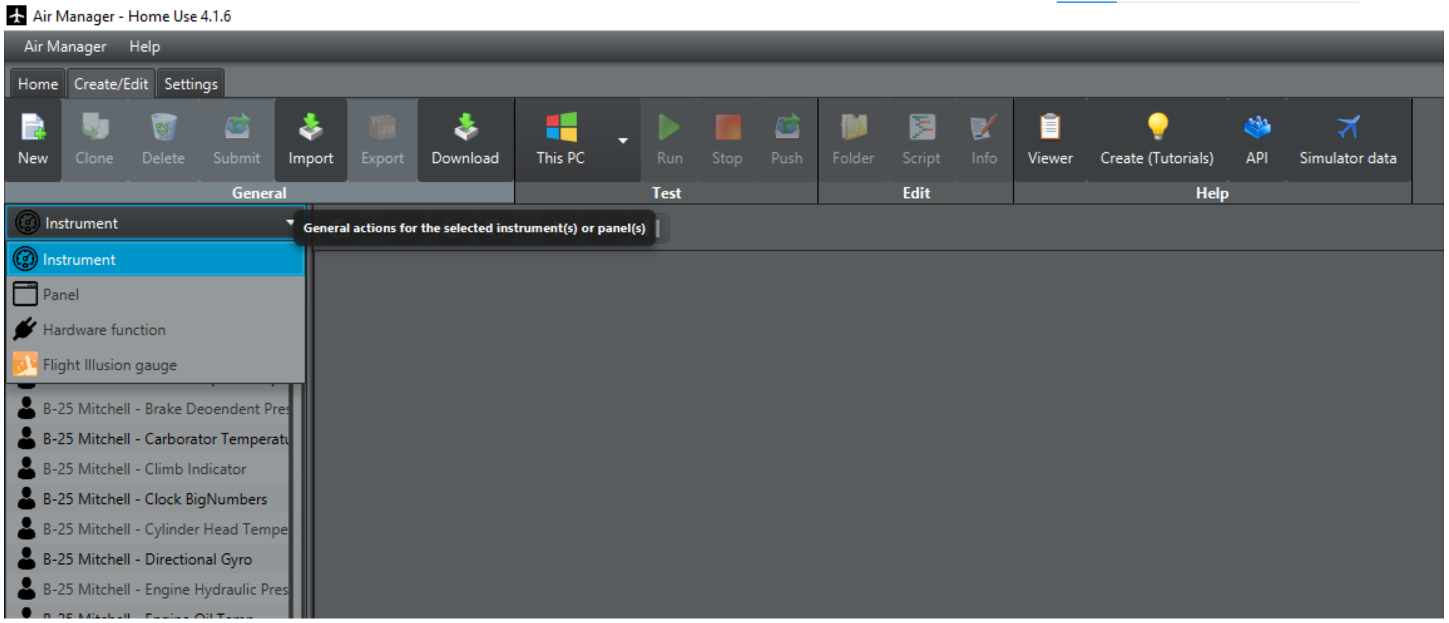
Navigate to home > Hardware



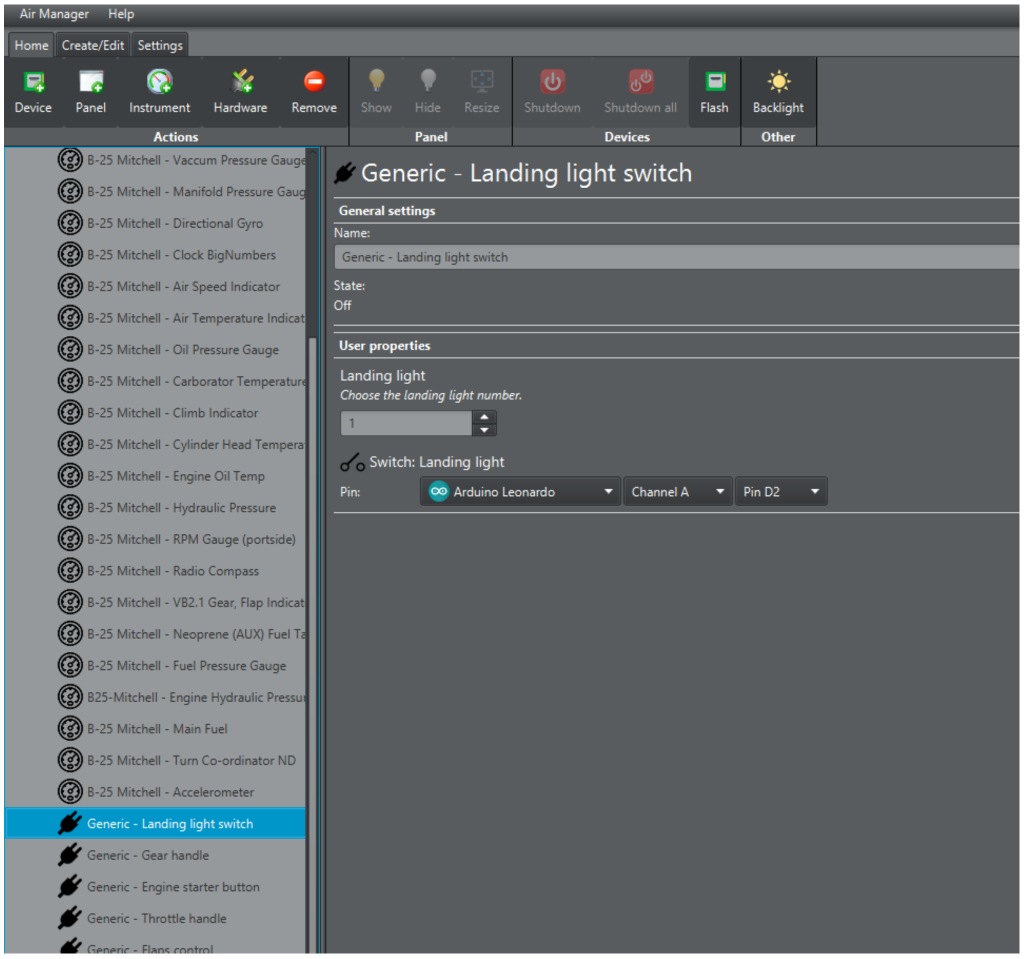
Navigate to the "generic" tab and choose the functionality you would prefer.



If the creation of custom hardware interactions is preferred, the code of each hardware function can be observed and modified under the ‘Create/Edit’ tab, with ‘hardware function’ selected.


Air Manager - Home Use 4.1.6
Air Manager
Help
Home
Create/Edit
Settings
New
Clone
Delete
Submit
Import
Export
Download
This PC
Run
Stop
Push
Folder
Script
Info
Viewer
Create (Tutorials)
API
Simulator data
General
Test
Edit
Help
Instrument
General actions for the selected instrument(s) or panel(s)
Instrument
Panel
Hardware function
Flight Illusion gauge
B-25 Mitchell - Brake Deoendent Pres
B-25 Mitchell - Carborator Temperatu
B-25 Mitchell - Climb Indicator
B-25 Mitchell - Clock BigNumbers
B-25 Mitchell - Cylinder Head Tempe
B-25 Mitchell - Directional Gyro
B-25 Mitchell - Engine Hydraulic Pres

Once the desired hardware function(s) have been selected, you can click on each one individually to assign data channel, Arduino device type, and pin.


Air Manager
Help
Home
Create/Edit
Settings
Device
Panel
Instrument
Hardware
Remove
Show
Hide
Resize
Shutdown
Shutdown all
Flash
Backlight
Actions
Panel
Devices
Other
B-25 Mitchell - Vaccum Pressure Gauge
B-25 Mitchell - Manifold Pressure Gaug
B-25 Mitchell - Directional Gyro
B-25 Mitchell - Clock BigNumbers
B-25 Mitchell - Air Speed Indicator
B-25 Mitchell - Air Temperature Indicat
B-25 Mitchell - Oil Pressure Gauge
B-25 Mitchell - Carborator Temperature
B-25 Mitchell - Climb Indicator
B-25 Mitchell - Cylinder Head Tempera
B-25 Mitchell - Engine Oil Temp
B-25 Mitchell - Hydraulic Pressure
B-25 Mitchell - RPM Gauge (portside)
B-25 Mitchell - Radio Compass
B-25 Mitchell - VB2.1 Gear, Flap Indicat
B-25 Mitchell - Neoprene (AUX) Fuel Ta
B-25 Mitchell - Fuel Pressure Gauge
B25-Mitchell - Engine Hydraulic Pressu
B-25 Mitchell - Main Fuel
B-25 Mitchell - Turn Co-ordinator ND
B-25 Mitchell - Accelerometer
Generic - Landing light switch
Generic - Gear handle
Generic - Engine starter button
Generic - Throttle handle
Generic - Flaps control

Generic - Landing light switch

General settings

Name:
Generic - Landing light switch

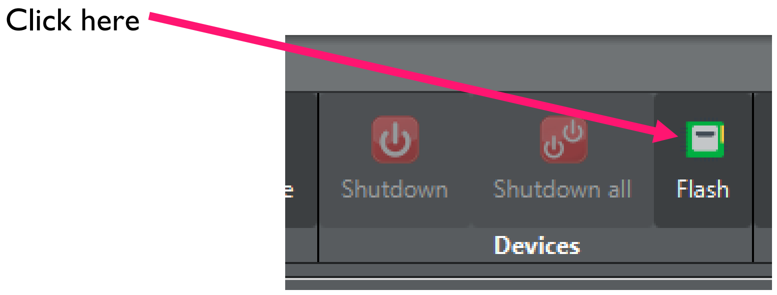
State:
Off

User properties

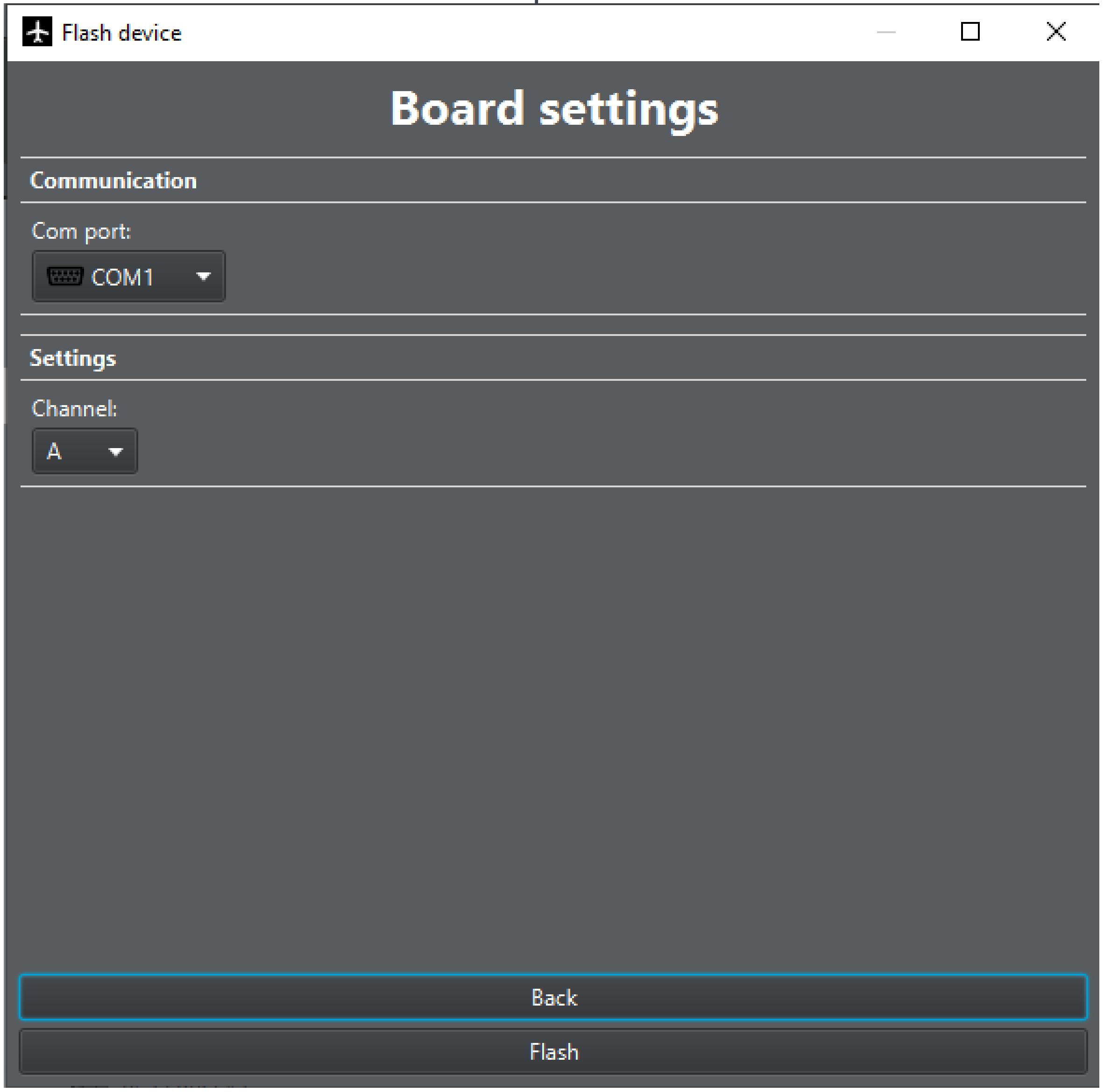
Landing light
Choose the landing light number.
1

Switch: Landing light
Pin: Arduino Leonardo | Channel A | Pin D2

Once these have been assigned, the code needs to be flashed to the Arduino. Click on 'Flash' on the right side of the screen



Next, select the com port that the Arduino is connected to on the computer, and the data channel it will be connected to. Afterwards the process will be automatic.


Flash device

Board settings

Communication

Com port: COM1

Settings

Channel: A

If done correctly, the functionality defined should interface seamlessly with prepar3D. Both potentiometers, switches, and buttons function as expected and no further work is required. For the physical circuits corresponding to the firmware defined here, circuits identical to those described in the previous section can be used, with the same parameters and constraints.

References:

- Arduino code bases
    - (Arduino Tutorials)
- Keyboard modifiers
    - (Arduino Leonardo Keyboard Modifiers List)
- Reading and modifying Prepar-3D configuration files
    - (Prepar 3D Config. File Reference)
- Air Manager Arduino integration
    - (Air Manager Arduino Reference)

References

Adding Full Size Images for Guages. 2023. <https://siminnovations.com/wiki/index.php?title=Img_add_fullscreen>.
Adding Images to Guages. 2023. <https://siminnovations.com/wiki/index.php?title=Img_add>.
Air Manager Arduino Reference. 2023. <https://siminnovations.com/wiki/index.php?title=Arduino >.
Air Manager User Manual. 2023.
<https://siminnovations.com/wiki/index.php?title=Air_Manager_4.x_User_Manual>.
Air Player Installation. 2023.
<https://siminnovations.com/wiki/index.php?title=Air_Player_4.x_Desktop_Installation#2._Unzip_the_downloaded_Air_Player_zip_file_to_a_new_AirPlayer_folder_on_your_Desktop_(~/Desktop/)>.
Air Player Raspberry Pi Manual. 2023.
<https://siminnovations.com/wiki/index.php?title=Air_Player_4.x_Raspberry_Pi_Manual>.
Arduino Leonardo Keyboard Modifiers List. 2023.
<https://www.arduino.cc/reference/en/language/functions/usb/keyboard/keyboardmodifiers/>.
Arduino Tutorials. 2023. <https://docs.arduino.cc/tutorials/>.
Connecting Variables to Guage Behavior/Animation. 2023.
<https://siminnovations.com/wiki/index.php?title=Fsx_variable_subscribe>.
Creating Rotating Needles. 2023. <https://siminnovations.com/wiki/index.php?title=Rotate>.
Moving Indicator Images. 2023. <https://siminnovations.com/wiki/index.php?title=Move>.
Prepar 3D Config. File Reference. 2023.
<http://www.prepar3d.com/SDKv3/LearningCenter/simobjects/aircraft_configuration_files.html>.
Prepar-3D Flight Dynamics Variables. 2023.
<http://www.prepar3d.com/SDKv3/LearningCenter/utilities/variables/simulation_variables.html#Simulation%20Variables>.
Sim Innovations Main Page. 2023. <https://siminnovations.com/wiki/index.php?title=Main_Page>.