SimLink

User Guide Rev 0.1

Table of Contents

What is SimLink?

SimLink is a software interface for XPlane and P3D, designed to allow devices to communicate over standard protocols and interfaces (USB, Ethernet, the Internet) to control a flight simulator. This is achieved using “messages” transmitted back and forth between the host computer and remote devices, such as an Arduino.

How does it work?

SimLink accepts packets from connected devices, which we call “events.” These device-side events are internally mapped and transformed into a simulator-side event, which is triggered on receipt of a message.

Your First Event

In order to start using SimLink to control your simulator, you must first connect some sort of electrical device (switch, pushbutton, rotary encoder, potentiometer, or even just two bare wires you touch together) to something that can read and interpret their state.

A few definitions

1. State - One of:
    a. A switch being either on or off
    b. A button being pushed or not pushed (also on or off)
    c. A rotary encoder being turned
    d. Potentiometer’s current position
    e. Etc.
2. Event - An action that is transmitted between a device and SimLink, or between SimLink and the simulator
3. Device - An Arduino, ESP8266, Nucleo, etc.
4. Message/Packet - One singular event being transmitted over the wire
5. Write Event - An event that causes a change to a value in the simulator. From device to simulator
6. Read Event - An event that reads a value from the simulator. From simulator to device

Step 1 - Arduino Wiring

We are going to make a switch that either raises or lowers the landing gear of our aircraft, depending on the state of this switch. This guide is not an exhaustive guide on how to interface with toggle switches using your Arduino, as there are plenty of the sort on the Internet. However, some rudimentary foundation will be laid.

You will need a few things:

1. An Arduino, ESP8266, Nucleo, or similar
2. A USB cable, to connect the above
3. A toggle switch
4. Some wire (Ethernet cable cut open works well)
5. A soldering iron (For making a permanent connection to the switch)

Connecting the toggle switch to the Arduino

A full guide on connecting a momentary pushbutton to an Arduino is available at https://www.arduino.cc/en/tutorial/button

For the sake of our aircraft, we are going to be following the above guide, except where it tells you to add a push button, we are going to use a switch.

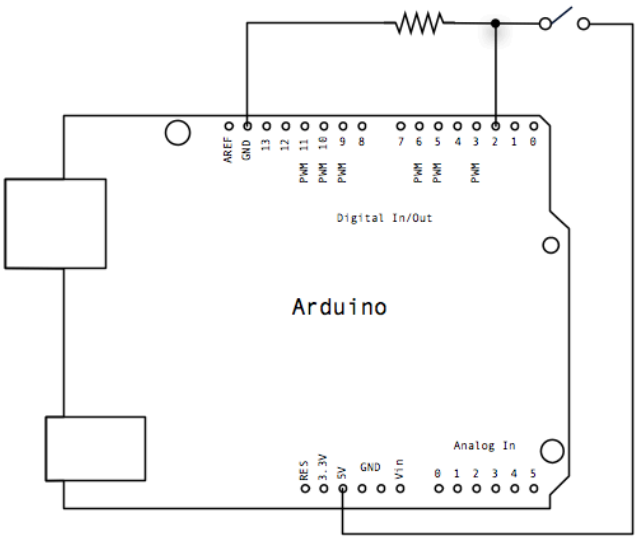


Image source: https://www.arduino.cc/en/uploads/Tutorial/button_schem.png

Please follow the above guide to connect your switch, and confirm it works using either code you can write, or the code referenced below.

Step 2 - Arduino Code

There is some sample code you can use available from the SimLink website. This code will take your switch, wired up to pin 2, and turn it into a usable event that can be transmitted over the wire.

A rough guide on writing the code from scratch:

1. Create two variables, `state` and `previousState`
2. Open a serial connection with `Serial.begin(115200)`
3. On every loop, store the state of the pin connected to your switch in state. This can be done using the `digitalRead(2)` function
4. If `state` is not equal to `previousState`:
    a. If `state == 1` and `previousState == 0`, then `Serial.println("A02H")`
    b. Otherwise, `Serial.println("A02L")`
5. Set `previousState` equal to `state`

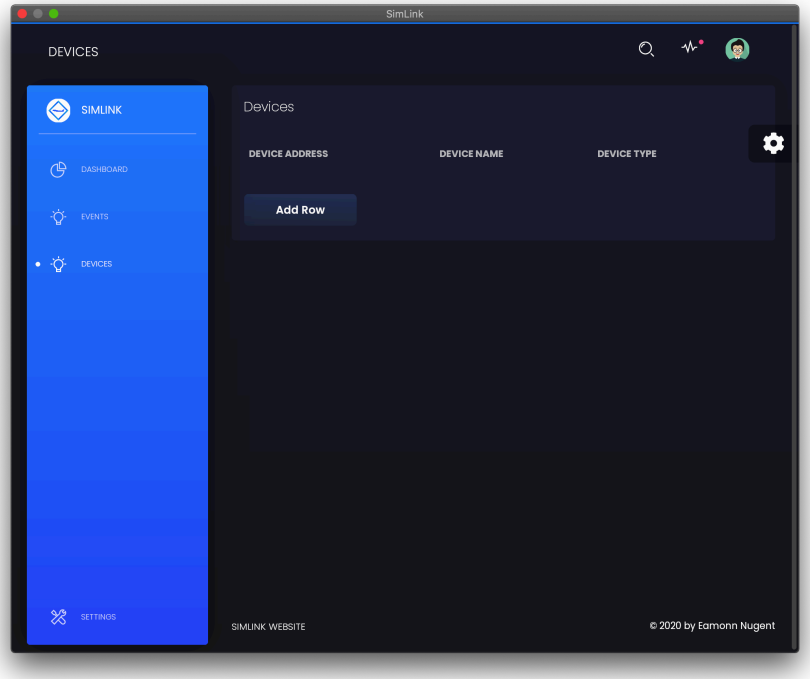
The act of printing a value over the Serial connection is what is read by SimLink. We can read `A02H` or `A02L` and interpret it as the switch being toggled, telling the simulator to raise or lower the landing gear.

Step 3 - SimLink

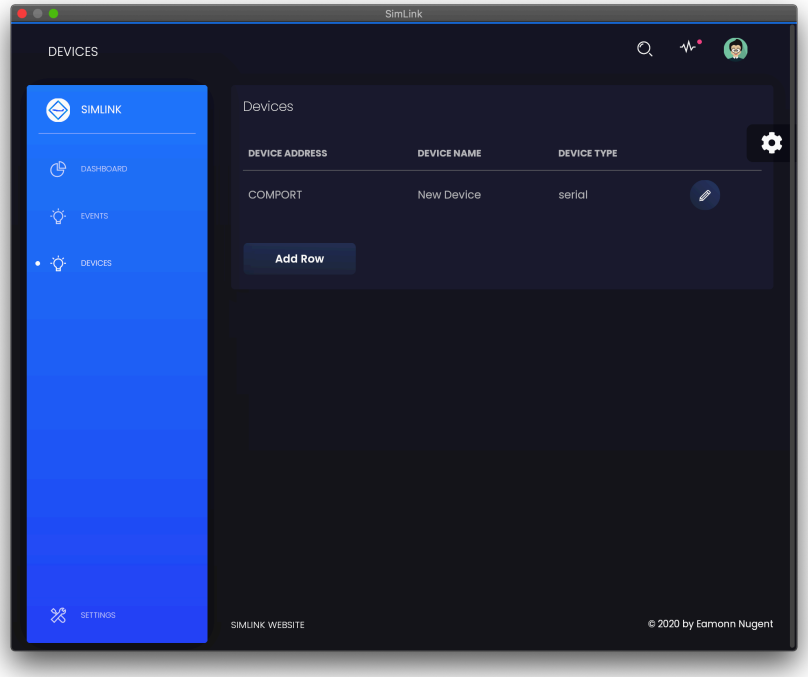
Device Configuration

In order for SimLink to properly control the simulator, it must first be told of the existence of the device. To accomplish this:

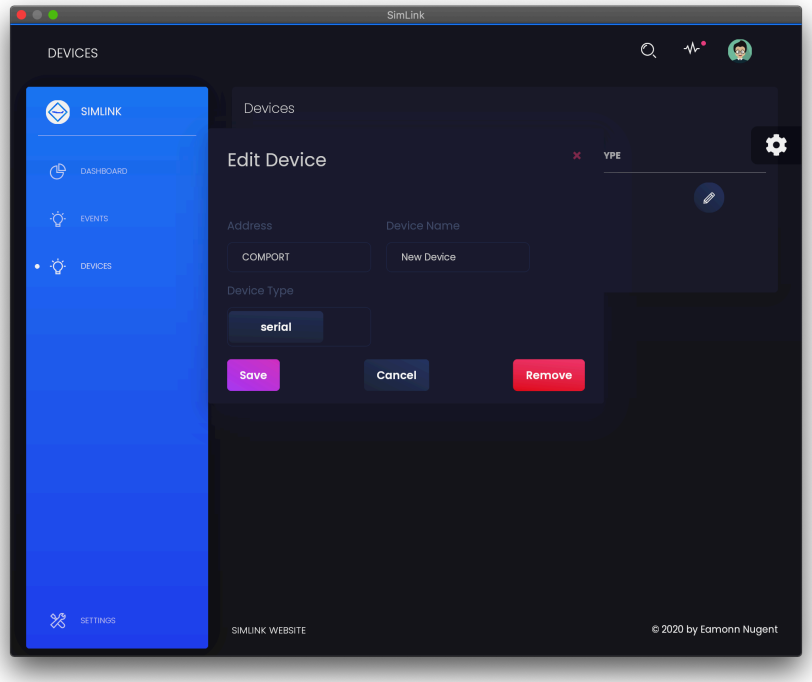
Open the "Devices" tab



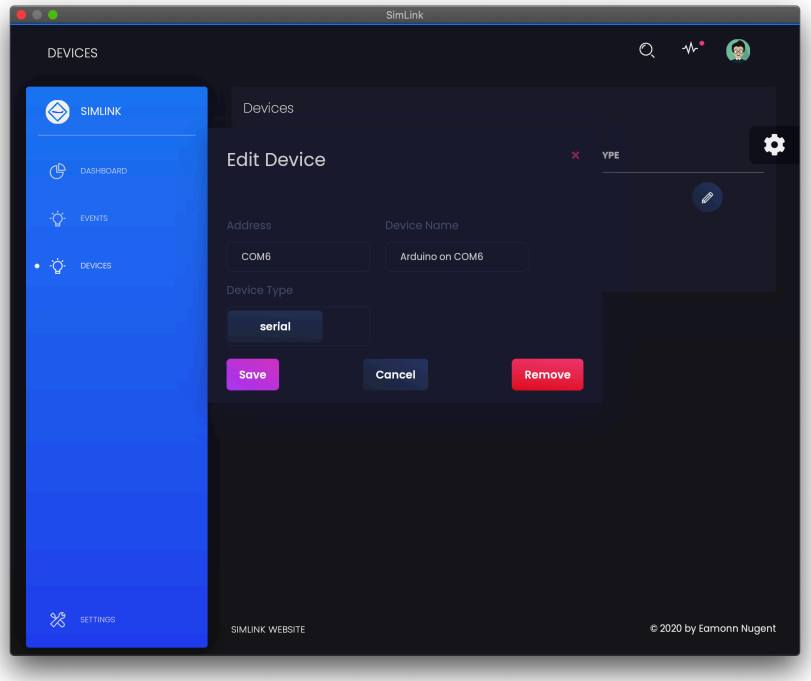
Click “Add Row”



Click on “Edit” next to our new row



Change COMPORT to the COM port of the device (for example, COM6 on Windows, /dev/ttyusb0 on Linux), give it a name, and select whether it is a Network or Serial device (ours is serial)



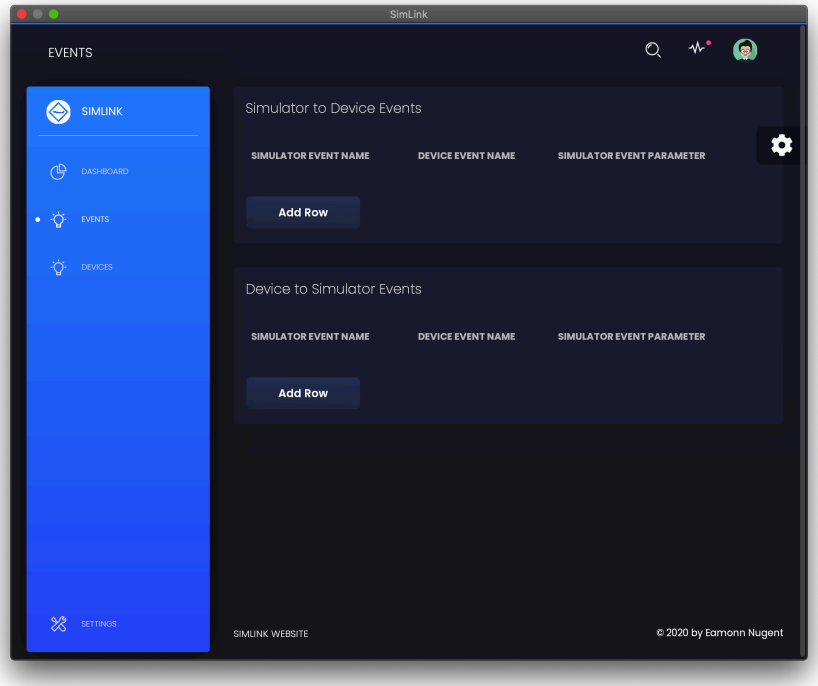
Click Save

SimLink will automatically try to connect to your device. If you don't see an error, it worked!

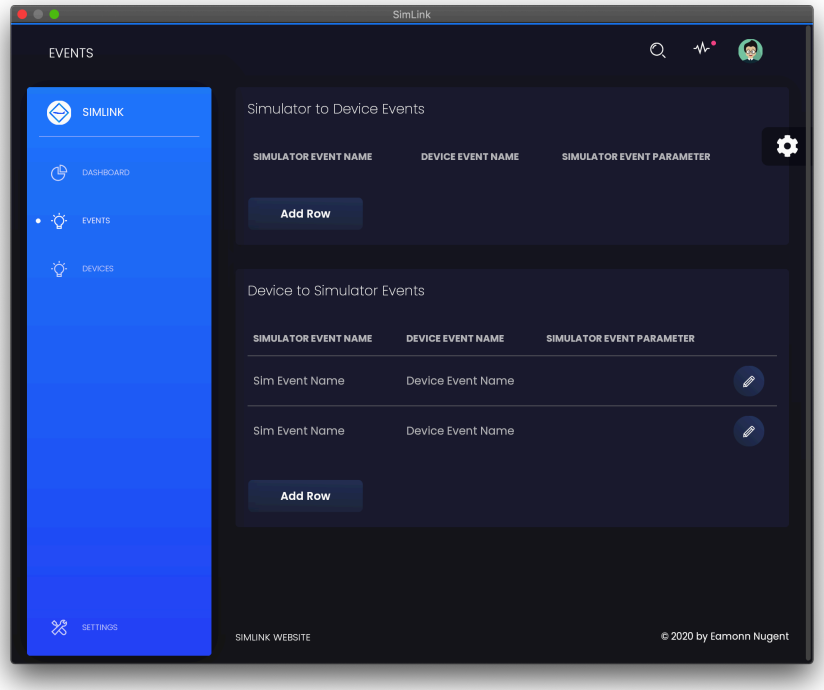
Event Configuration

SimLink now needs to know how to handle your brand new switch. It needs two events to be defined - one for `A02H`, one for `A02L` - one for each possible state of the switch.

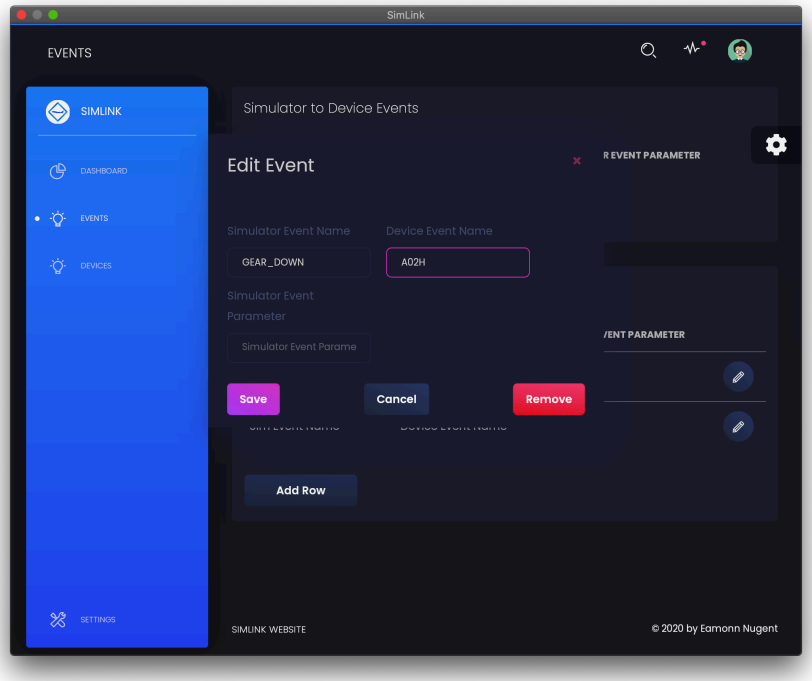
Open up the Events page of SimLink



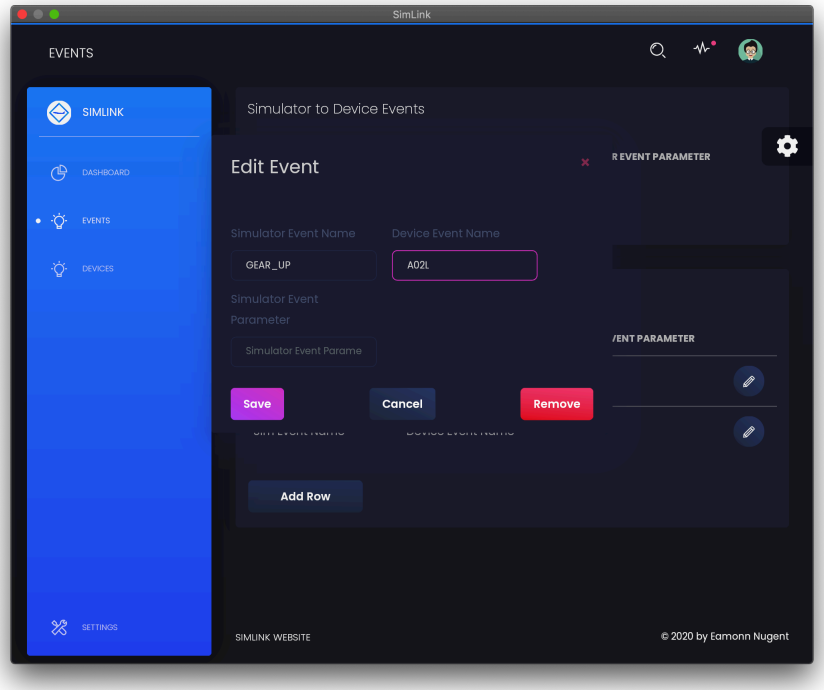
Click "Add Row" twice to create two new events under Device to Simulator Events



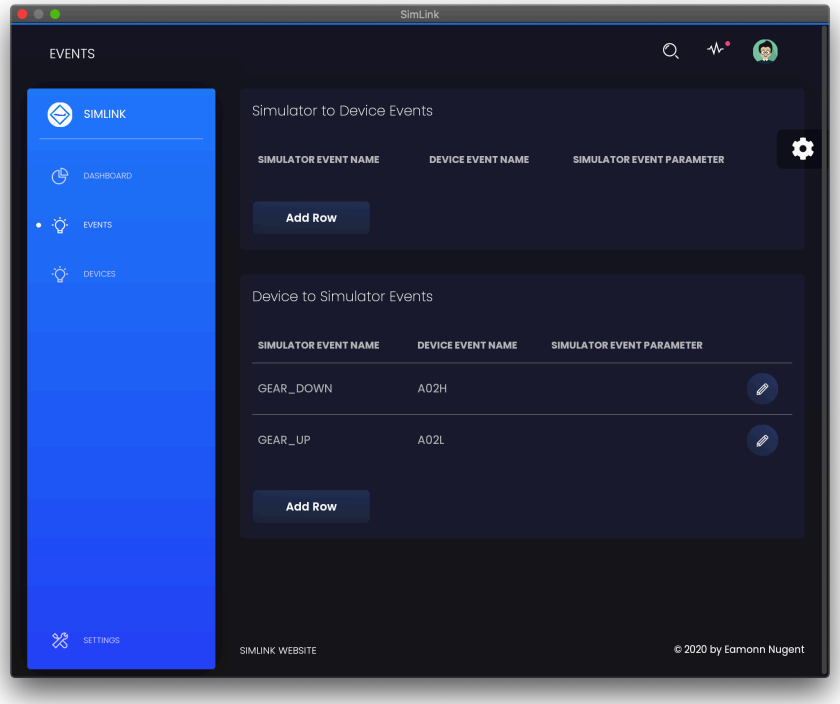
Click Edit next to one of the events



Fill in the values. To lower the gear in SimConnect-based platforms, you can type “GEAR_DOWN” in the “Simulator Event Name” window. Put “A02H” in the Device Event Name window



Do the same for the next event. GEAR_UP will raise the gear, and we want it for event A02L. Then click Save



You now have your events saved

Step 4 - Boot up your simulator!

Launch your simulator and try it out! With any luck, you should be able to flick a switch, and SimLink will toggle your landing gear for you!

A note about landing gear

It is recommended to test SimLink using the default aircraft. In Prepar3D (and other SimConnect-based platforms), the landing gear lever is often overridden on non-default aircraft, and the events listed above won't work. For example, PMDG uses custom code on their airplanes, which causes the landing gear lever to not properly activate.

Also, if you're tearing your hair out, asking why it isn't working - make sure you're in the air. Some planes won't let you retract your landing gear while you're still on the ground.

Other Simulator Events

Want to do something more powerful with your simulator than just control your landing gear? No problem! You can use *any* SimConnect event ID or XPlane dataref to control your aircraft - including third party ones!

Here is the master list of default SimConnect events:
http://www.prepar3d.com/SDKv3/LearningCenter/utilities/variables/event_ids.html

Here are the default XPlane datarefs (yes, you can make a button that sets your engine on fire):
https://developer.x-plane.com/datarefs/

Third-party aircraft addons

SimLink supports third-party aircraft natively. As long as the switch, button, knob, or dial is controlled using a SimConnect event or X-Plane dataref, it is able to be controlled.

SimConnect (FS2020/Prepar3D/FSX)

In order to control your third-party aircraft, you must first know what the event IDs are within SimConnect. This may require some coding knowledge, as often, you need to look through the SDK that comes with the plane to determine the event ID. They are almost always numeric (f.ex, `67328`), and often mathematically calculated, rather than written straight-out.

In order for these third-party events to be recognized by SimConnect, the simulator event ID must be in the format of hash symbol, then the numeric ID of the event. For example: `#67328` for the event mentioned above.

X-Plane

Thankfully, X-Plane datarefs are much more standard. Most default functions that X-Plane has native datarefs for are followed, however, there are often custom ones defined. There are two main ways (plus a third secret one) to discover datarefs

Option 1

Open the Datarefs.txt file within the addon aircraft folder. Usually, these will be human-readable event names that you can look through, then copy + paste to your heart's desire

Option 2

There are dataref tools available online to discover, edit, and read datarefs. Here is one I've personally liked:
https://github.com/leecbaker/datareftool

This tool can be used by filtering for changed datarefs, triggering the event in the simulator, and then looking for what has changed. This can be cumbersome, but effective, in determining how the plane works internally.

Option 3

Read the documentation! Many add-on aircraft will document their datarefs for developers to view. Simply see if there is an SDK or similar package to learn from, and grab the refs from there.

Lua Scripting

SimLink also supports modifying events on-the-fly for Lua scripting. For example, want to have a three way switch, where when you push it up, it increases the flaps, instead of setting the flaps to a specific position? Want a trim switch? The primary way of accomplishing this is to transmit the event from the device to SimLink, then have a Lua event intercept it and rewrite or re-fire events as needed. You could even have a single button that resets the entire aircraft to cold-and-dark, or runs through the full startup checklist. The possibilities are endless.

Interface

The SimLink Lua interface is of a relatively simple format. All events are transmitted to all Lua scripting file entrypoints (of the function header defined below), and the function can either return `true`, stating that it “takes over” handling of the event, or `false`, that the script does not explicitly handle that event.

Script Definition

All Lua scripts must be placed in the `Scripts` directory in the SimLink folder. The folder is in the following locations for different operating systems:

Windows

`C:\Users\<your-user>\Documents\SimLink\Scripts`

Mac

`/Users/<your-user>/Documents/SimLink/Scripts`

Linux

`/home/<your-user>/Documents/SimLink/Scripts`

Entrypoint

All Lua scripts must contain the following function header:
function handle_write(command, parameter)

The function must:

- Return `true` if the event should *not* be sent to the simulator by SimLink, but instead be sent by a `trigger_write(event, parameter)` call
- Return `false` if the event *should* be sent to the simulator, and is *not* handled by the Lua script

Triggering write events from Lua

Lua scripts can trigger write events to the simulator in the course of their execution with a single function call. The function is defined in the SimLink code that loads your Lua script, and should *not* be defined elsewhere in your script. The function is named `trigger_write`, and accepts two arguments: the `command`, and the `parameter`.

Command

The `command` argument is the name of the in-simulator event to be fired. For example, it could be `GEAR_UP` in SimConnect, or `sim/cockpit/switches/gear_handle_status` in X-Plane.

Parameter

The `parameter` argument is the value to be sent to the simulator. For SimConnect, this is an integer. For X-Plane, this is a string of the format `datatype:value`.

X-Plane Parameters

The following datatypes are implemented in the X-Plane plugin:

- float
- int
- boolean
- failure_enum

Note: failure_enum is an alias for int

For example, to trigger a fire on engine 1, you would transmit:
Command: `sim/operation/failures/rel_engfir0`
Parameter: `failure_enum:6`

Troubleshooting

SimLink Logs

SimLink generates a log file which can be helpful in debugging issues. For any support requests, please ensure that the log file is attached.

Log Location

Windows

`C:\Users\<your-user>\Documents\SimLink\simlink.log`

Mac

`/Users/<your-user>/Documents/SimLink/simlink.log`

Linux

`/home/<your-user>/Documents/SimLink/simlink.log`

SimLink crashes when I open it

SimLink will automatically exit if it does not detect a valid license. This is determined by contacting the SimLink authentication service, via opening a web browser. If SimLink crashes, it

is likely due to the browser not being opened properly. Make sure SimLink is not quarantined by any antivirus, and has proper permissions.

In addition, it is possible for SimLink to crash if the executable does not have valid permissions, is not able to open the log or configuration files, or if it detects invalid configuration. Try removing the `simlink.json` file in SimLink's folder. If that causes it not to crash, there likely has been corruption of the main config file.