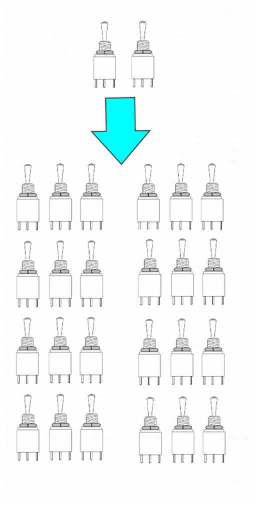
Instructions for the source-script „Virtual Switch“



Inhaltsverzeichnis

Introduction

General information

VirtualSwitch manages to trigger up to 24 switching functions with only two buttons.

This is achieved by differentiation in the "key time" (i.e. short, medium or long press) in combination with the gyros that may be integrated in transmitters.

The transmitter position (normal, left, right, forward) is determined via the gyros.

Transmitters without gyros can therefore only map six switching functions.

What can be triggered / switched ?

For safety-related reasons, only secondary functions should be triggered for the actual control of models.

e.g.:
- Announcement of telemetry values
- Announcement of timers
- Resetting of timers, single telemetry sensors (consumption, altitude...) or complete telemetry
- Switching ("toggling") of maximum two logical switches

Basic idea:

On the one hand, the use of the pushbuttons mounted in the bottom of the transmitter housing makes it possible to trigger many functions without taking your fingers off the stick.
In addition, the number of possible functions "relieves" switches & trimmers, if necessary, which are now available for other important functions.

Typically one owns several models of a class (competition gliders, scale gliders, motor models...) with the same requirement profile regarding telemetry sensors.
Now you can configure the complete class with one setup for announcements, resets etc.. without having to adopt this every time in the Ethos config.
With a central change in the script one can directly adjust all corresponding models.

One becomes more independent of the transmitter type:
The original version originated from the requirement to migrate models (under oTx) from a Horus X12s to an X-Lite pro quite quickly, without really having to deal with the switching functions.
For this purpose, a script was programmed for the X12, which mapped the required "simple" switching functions via the input devices possible there. Another script with the same name for the identical switching functions but with a different Hardware assignment was developed for the X Lite.

If you copied a model from the X12 to the X Lite pro, you had all these functions immediately available, despite the different HW configuration, without having to make any adjustments !
Trick was just the identical lua script name !

Notes in advance:

(1)

For proper function of the transmitter gyro, I assume that the transmitter has been calibrated correctly under Ethos.

I always calibrate in a way that the maximum gyro value is reached only with almost vertical position.

Ethos up to Rev 1.4.4 shows a bug in the calibration routine of the X18s/se.
Please note issue#2161 here !
https://github.com/FrSkyRC/ETHOS-Feedback-Community/issues/2161

(2)

The current Ethos Lua (Q4 2022) unfortunately does not yet provide all special functions as a Lua method. However, the functionality will be extended successively.

Telemetry reset is therefore currently only programmatically "reproduced" by dedicated resets of certain sensors.

Real timer resets are not possible, I just set the timer to 00:00:00.
The emulated "reset" therefore only works with up-counting timers.

If appropriate methods become available through Ethos updates, the script will be extended accordingly.

How it works:

Four position levels (standard, left, right, front) multiplied by six possible switching states per position result in 24 possible functions.
Of course you don't have to use all functions, depending on your taste you should configure the script in a way, that you don't have to think over and over again, which combination triggers what.

The script reads a configuration file to determine which key duration in combination with which transmitter position triggers which function.

Now the transmitter gyro and the probe states are permanently queried.

The transmitter has a certain dead range in the "standard position", so that a gyro event is not detected with every wobble.
As soon as a "threshold" is exceeded for the transmitter's banking (approx. 30deg left/rear, or slightly forward), the transmitter position is reassigned, a short sound signal is emitted.
You have to go back to neutral to redefine the "position", a direct slew from "forward" to "left" is not immediately supported, because it turned out to be difficult to handle.
Optionally, a sound signal can also be emitted for reaching the "standard" position again.
(Is off by default, bothered me...)

Besides the typical announcement and reset functions, there is also the possibility to switch two logical switches.

For this always two bits of the so called "source variable" are used.
The source variable can take values from 0 to 3:

0 >> LSW 1 off	LSW 2 off
1 >> LSW 1 on	LSW 2 off
2 >> LSW 1 off	LSW 2 on
2 >> LSW 1 on	LSW 2 on

Um die LSW‘s unter Ethos auzuwerten sind Einstellungen eines LSW‘s nötig, die unter dem Kapitel Konfiguration ausgeführt werden.

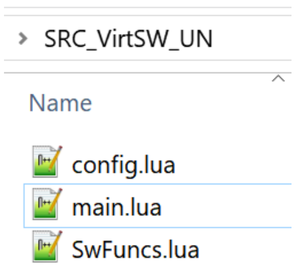
Operating:

It is a so called "source script", a bit analog to the oTx mixer script.
Therefore no UI / operation is possible

Installation:

(1) Copy files & folders

After downloading the zip file it is necessary to unzip it.
Now a folder is available, which must be copied into the script directory of the transmitter:



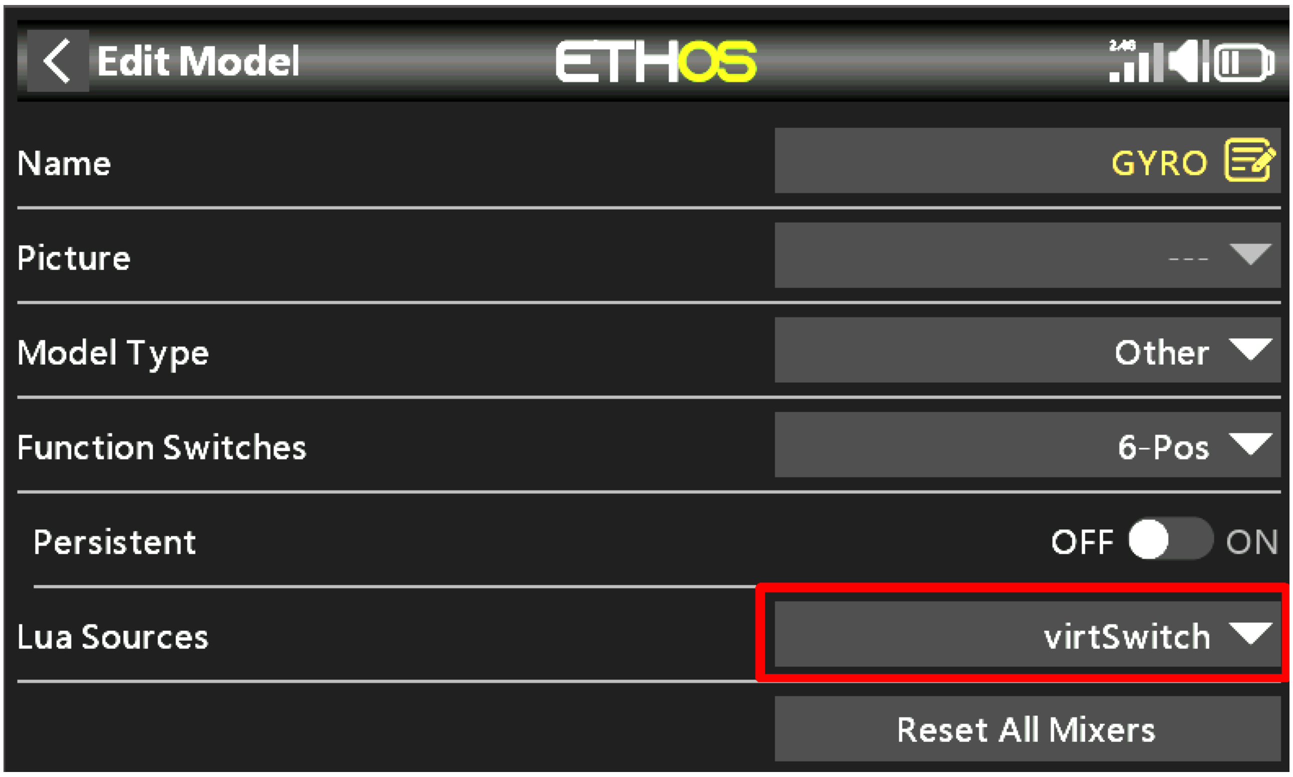
SRC_VirtSW_UN

Name

config.lua
main.lua
SwFuncs.lua

(2) set up source script

Source scripts are always activated in model settings (see also "Edit Model").
Under the corresponding form entry, at the bottom, ("Lua Sources") the "virtSwitch" script has to be activated:



Edit Model ETHOS

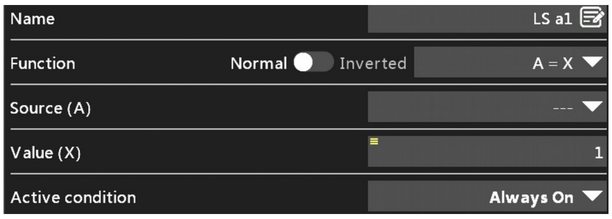
Name	GYRO
Picture	---
Model Type	Other
Function Switches	6-Pos
Persistent	OFF ON
Lua Sources	virtSwitch
	Reset All Mixers

(3, optional) Set up logical switches

If the script is also to be used to set up logical switches, these must be defined within Ethos.
Altogether four LSW's are needed.
Two for preprocessing, two for the "actual" switching functions.

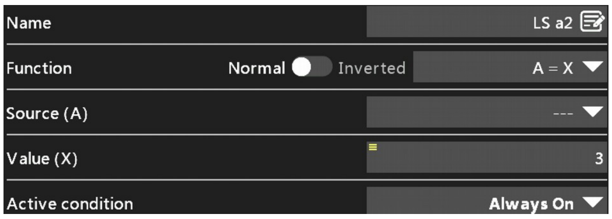
(1) LSW preprocessing:
The first log.switch, "LS A", is active when "the value of the script" is 1 or 3.
Accordingly, a "LSW a1" is defined for "value =1" and another "LSW a2" for "value =3
"LS A" is active at the "boolean condition": "LS a1" or "LS a2", that makes already three LSW's

LS a1:



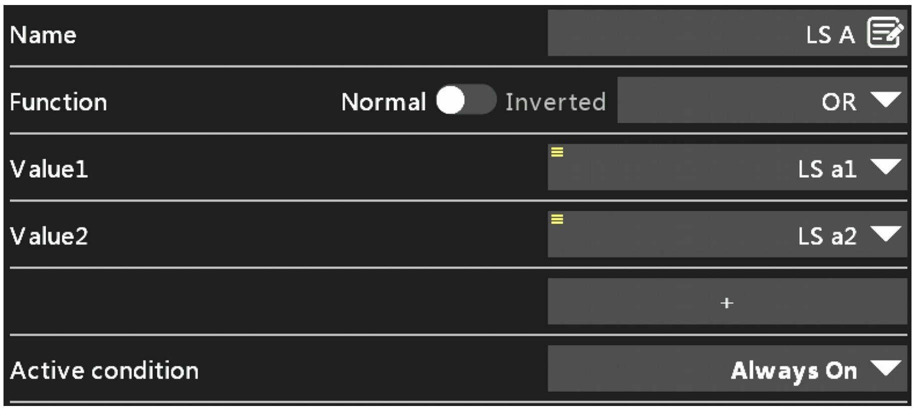
Name	LS a1
Function (Normal / Inverted)	A = X
Source (A)	---
Value (X)	1
Active condition	Always On

LS a2:



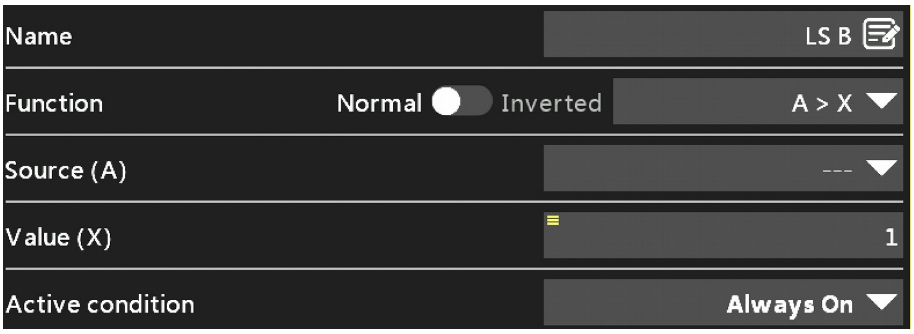
Name	LS a2
Function (Normal / Inverted)	A = X
Source (A)	---
Value (X)	3
Active condition	Always On

LS A (actual LSW for further use):



Name	LS A
Function (Normal / Inverted)	OR
Value1	LS a1
Value2	LS a2
	+
Active condition	Always On

LSW2 is simply defined as "Value > 1



Name	LS B
Function (Normal / Inverted)	A > X
Source (A)	---
Value (X)	1
Active condition	Always On

Configuration:

Introduction:

The configuration is done by editing files.
Essentially, the button assignment must be defined depending on the transmitter position.

This is done via entries in the file "config.lua".
The desired functions are entered there.

In the main program "main.lua", users experienced in working with computers can also make settings for the desired timing of the buttons (short/medium/long) or the threshold values for how far a transmitter must be swiveled to suit their personal needs.

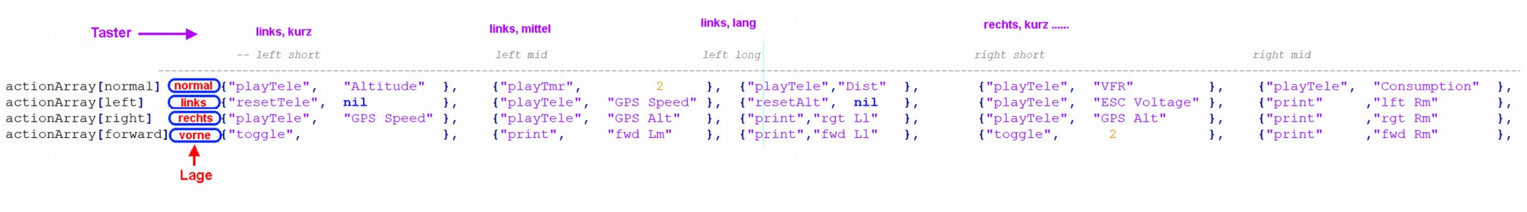
Maintain Config Table

If you open the "config.lua" file with a text editor, you can easily recognize a paragraph with table-like entries

A row corresponds to a transmitter attitude (normal, left...) a column corresponds to a button state ("left,short" or "left,medium" or "right,long" etc..), depending on which button you hold down and for how long.

Inside the braces (the "cell") the desired function is entered, followed by a parameter separated using a comma.

A telemetry announcement has the function "playTele", the desired sensor in quotation marks as text to be entered. (Note: the English term of the sensors must be entered, regardless of which language the transmitter has been configured to).



```
                            -- left short                      left mid                 left long                          right short                    right mid
                      ------------------------------------------------------------------------------------------------------------------------------------------------------------------
actionArray[normal]  {"playTele",    "Altitude"  },    {"playTmr",         2      },  {"playTele","Dist"  },        {"playTele",  "VFR"          },    {"playTele",  "Consumption"  },
actionArray[left]    {"resetTele",  nil          },    {"playTele",  "GPS Speed" },  {"resetAlt",  nil    },        {"playTele",  "ESC Voltage"  },    {"print"     ,"lft Rm"        },
actionArray[right]   {"playTele",    "GPS Speed" },    {"playTele",  "GPS Alt"   },  {"print","rgt Ll"    },        {"playTele",  "GPS Alt"      },    {"print"     ,"rgt Rm"        },
actionArray[forward] {"toggle",                  },    {"print",     "fwd Lm"    },  {"print","fwd Ll"    },        {"toggle",      2            },    {"print"     ,"fwd Rm"        },
```

List of possible functions

An expression in the above table like {"playTele", "Altitude" } represents a function that can be called by a button click.
Usually such an expression consists of a pair:

1. - the actual function (here "playTele"), this must be put in quotation marks
2. - usually as a second value a necessary parameter (texts must also be set in quotation marks).

If one does not want to use a corresponding key function like "left,long" in attitude "right", for example, a dummy function should be entered, like {"print", "right Ll"}, see example above.

If no parameter is needed, enter nil, >> look at the "resetAlt" function.

Calling Telemetry sensors is done by using their „origin english labels“, if you don't know them, I recommend the following procedure:

Copy model with as many sensors as possible as a working copy and change to it
Delete all sensors in the working copy
Change transmitter to english operation
Search for new sensors and note their names
Change the transmitter back to your language and switch to the original model .

the following functions are currently implemented:

playTele, sensor	Announcement of the sensor value "sensor" (sensor = text).
playTmr, timer	Announcement of the timer (timer = number)
resetTele, nil	reset telemetry (currently workaround)
resetAlt, nil	reset altitude
resetTmr, timer	reset timer (timer = number; set to value 0)
toggle, num	toggle a LSW (num = number 1 or 2; internal LSW number)

Note for Lua enthusiasts:

These "pairs" are "passed through" as function calls directly via the global namespace, this also allows calls to direct Lua methods/functions if desired (see "print") ...

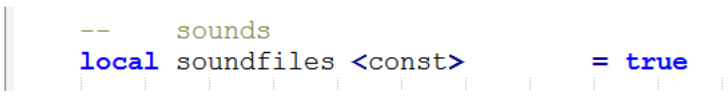
Sounds

I like it "short and direct", so by default, a telemetry value is reported only, without further voice announcements.

If you want to have a "prefix" like "altitude" , "distance", "receiver voltage" etc. included in the announcement, you can reconfigure this.

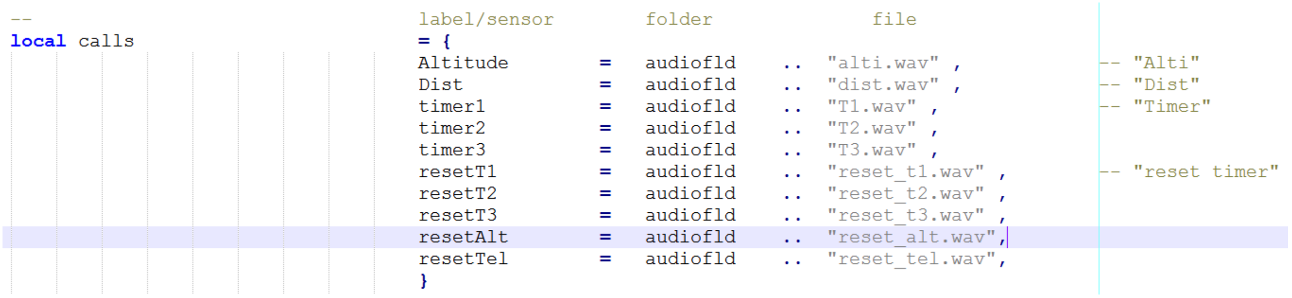
This is also done via the file "config.lua".

first the "constant" named soundfile has to be set to value "true":



```
--      sounds
local soundfiles <const>        = true
```

In another table the corresponding sound file is assigned to the telemetry sensor (label):



```
--                          label/sensor      folder           file
local calls                 = {
                            Altitude      =   audiofld    ..  "alti.wav"  ,        -- "Alti"
                            Dist          =   audiofld    ..  "dist.wav"  ,        -- "Dist"
                            timer1        =   audiofld    ..  "T1.wav"  ,          -- "Timer"
                            timer2        =   audiofld    ..  "T2.wav"  ,
                            timer3        =   audiofld    ..  "T3.wav"  ,
                            resetT1       =   audiofld    ..  "reset_t1.wav"  ,    -- "reset timer"
                            resetT2       =   audiofld    ..  "reset_t2.wav"  ,
                            resetT3       =   audiofld    ..  "reset_t3.wav"  ,
                            resetAlt      =   audiofld    ..  "reset_alt.wav",
                            resetTel      =   audiofld    ..  "reset_tel.wav",
                            }
```

When calling a telemetry sensor or executing a function (reset...) this sound file will be called first. One must define only the file names to the labels, with which a preannouncement is desired.

If no sound file is defined, only the value is announced.

Sounds must be generated according to the individual requirements.
I recommend the application TTSAUTOMATE.
Associated psv files as an example to create your own sound files can be found in the folder "ttsautomate".

The sound files themselves belong in the corresponding ETHOS audio folder, directly under the folder for the respective language version (e.g. /audio/de).

Ethos uses
for the German language the setting "de-DE-Wavenet-E".
for the English language the setting "en-GB-standard A".

In all likelihood, the new TTSAutomate (in development, status Q4 2022) will support these languages.

additional options

If you are confident you can do further fine tuning in the main.lua:

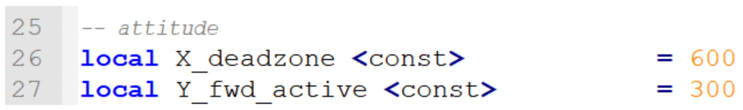
banking thresholds

It's possible to change the setting of the threshold value, from which transmitters banking a corresponding "Gyro Event" should be triggered.

Usually the transmitter delivers values between -1024 (e.g. vertical to one extreme position) and +1024 (from one to the other extreme position), whereby the extreme position depends on the calibration.
Neutral position is approx. 0, small offsets are possible.

For swivel left/right I define a "deadband" which defines the neutral position.
If the limit is exceeded, the script goes into the selected "Gyro Mode".

The deadzone is defined in the variable "X_deadzone".

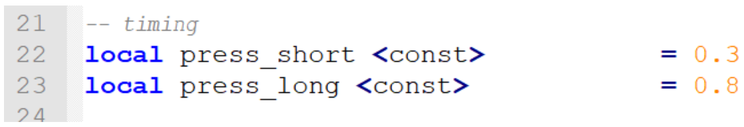


```
-- attitude
local X_deadzone <const>             = 600
local Y_fwd_active <const>           = 300
```

Similarly, "Y-fwd_active" defines the threshold for forward tilt.

Tactile times

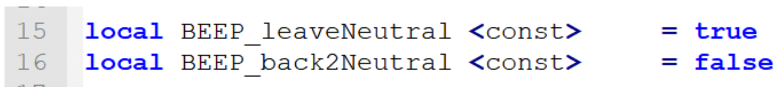
The parameters necessary for the "short/medium/long press" distinction are as follows:



```
-- timing
local press_short <const>            = 0.3
local press_long <const>             = 0.8
```

times < 0.3 seconds are thus defined as "short",
times >0.8 seconds as "long",
in between ergo "medium". These parameters can be adjusted individually.

Akustik-feedback



```
local BEEP_leaveNeutral <const>     = true
local BEEP_back2Neutral <const>     = false
```

These two parameters can each be set to "true" or "false".
The first parameter enables a signal tone when the gyro threshold is exceeded, the second when the neutral position is reached again.

Rev 0.8
unow, Dez 2022