Lua Scripting in OpenTX
Reference Guide

Contents

Introduction

OpenTX 2.0 added support for Lua (current version 5.2.2) user scripts.

Lua is a lightweight multi-paradigm programming language designed as a scripting language. More at Wikipedia

There are several types of Lua scripts used in openTX. More general information about Lua scripts can be found on page http://www.open-tx.org/lua-instructions.html

Lua scripts must be placed on SD card in correct folders and have an extension `.lua`. Maximum Lua script file name length is TODO characters. The script folders have been reorganized in OpenTX 2.0.3. The folder structure looks like this:

- `/SCRIPTS/WIZARD/` - For the Wizard script
- `/SCRIPTS/MIXES/` - For model scripts
- `/SCRIPTS/FUNCTIONS/` - For function scripts
- `/SCRIPTS/«modelname»/telemXX.lua` - For telemetry scripts
- `/SCRIPTS/TEMPLATES/` - For template scripts

Lua Standard Libraries	**Included**
package	no
coroutine	no
table	no
io	no
os	no
string	no
bit	Future 2.1.0?
math	Available from 2.0.0
debug	no

Model Scripts

WARNING
Do not use Lua model scripts for controlling any aspect of your model that could cause a crash if script stops executing.

General description

Each model can have several model scripts associated with it. These scripts are run periodically for entire time that model is selected/active. These scripts behave similar to standard OpenTX mixers but at the same time provide much more flexible and powerful tool.

Typically model scripts take several values as inputs, do some calculation or logic processing based on them and output one or more values. Each run of scripts should be as short as possible. Exceeding certain script execution runtime will result in script being forcefully stopped and disabled.

See also:

- Lua One-time Scripts describes one-time running general scripts
- Lua Function Scripts
- Lua Telemetry Scripts
- Lua Script Reference detailed reference of OpenTX Lua implementation and interface
- Lua Script Examples some example scripts with comments

Examples of typical use of model scripts

- replacement for complex mixes that are **not critical to model function**
- complex processing of inputs and reaction to their current state and/or their history
- filtering of telemetry values
- automatic detection of number of battery cells and setting of low battery threshold
- automatic announcing of maximum altitude for each DLG throw
- see also Lua Script Examples

Limitations of model scripts

- Should not display anything on LCD screen.
- Can't wait for user input via dialog.
- Should not exceed maximum allowed runtime/ number of instructions.
- Standard OpenTX mixes are run every XX milliseconds in a very deterministic way (guaranteed execution) while model scripts are run from another thread with less priority. Their execution period is around 30ms and is not guaranteed!
- A script could be disabled/killed anytime due to several causes like (error in script, not enough free memory, etc...)

Anatomy of model script

Location of model scripts

Place them on SD card in folder `/SCRIPTS/MIXES/`

Lifetime of model script

- script is loaded from SD card when model is selected
- script **init** function is called
- script **run** function is periodically called (inside GUI thread, period cca 30ms)
- script is stopped and disabled if it misbehaves (too long runtime, error in code, low memory)
- all model scripts are stopped while one-time script is running (see Lua One-time scripts)

Script interface definition

Every script must include a **return** statement at the end, that defines its interface to the rest of OpenTX code. This statement defines:

- script **inputs** (optional)
- script **outputs** (optional)
- script **init** function (optional)
- script **run** function

For example:

```
-- script body would be here

return { input=inputs, output=outputs, run=run_func, init=init_func }
```

This example defines:

- *inputs* table (array) as **input** values to model script
- *outputs* table as **output** of model script
- *run_func()* function as **periodic execution function** that takes inputs as parameters and returns outputs table
- *init_func()* function as function that is **called one time** when script is loaded and begins execution.

Parameters **init**, **input** and **output** are optional. If model script doesn't use them, they can be omitted from return statement. Example without **init** and **output**:

```
local inputs = { { "Aileron", SOURCE }, { "Ail. ratio", VALUE, -100, 100, 0 } }

local function run_func(ail, ratio)
    -- do some stuff
    if (ail > 50) and ( ratio < 40) then
        playFile("foo.wav")
    end
end

-- script that only uses input and run
return { run=run_func, input=inputs }
```

Script initialization

If defined, **init** function is called right after the script is loaded from SD card and begins execution. Init is called **only once** before the run function is called for the first time.

local <init_function_name>()

called once before first call to run function

`Parameters:`	`none`
`Returns:`	`none`

Script execution

The **run** function is the function that is periodically called for the entire lifetime of script. Syntax of run function is different between model scripts and one-time scripts.
local <run_function_name>([first input, [second input], …])

`Parameters:`	`<>`	zero or more input values, their names are arbitrary, their meaning and order is defined by the input table
`Returns:`	`none`	if output table is empty (i.e. script has no output)
	`values`	(comma separated list of values) list of output values, their order and meaning is defined by the output table

One-Time Scripts

General description

These scripts start when called upon by a specific radio function or when the user selects them from a contextual menu. They do their task and are then terminated and unloaded. Please note that all persistent scripts are halted during the execution of one time scripts. They are automatically restarted once the one time script is finished. This is done to provide enough system resources to execute the one time script.

See also:

- Lua Model Scripts describes continuously running model scripts
- Lua Script Reference detailed reference of OpenTX Lua implementation and interface
- Lua Script Examples some example scripts with comments

Examples of typical use of one-time scripts

- All kind of wizards to set up/edit model settings. The official **model wizard** is one example of such script
- Replacement for templates
- Games

Limitations of one-time scripts

- When running all other Lua scripts are halted.
- Should not exceed maximum allowed runtime/ number of instructions.

Anatomy of one-time script

Location of one-time scripts

Place them anywhere on SD card, the folder `/SCRIPTS/` is recommended. The only exception is official model wizard script, that should be put into `/SCRIPTS/WIZARD/` folder - that way it will start automatically when new model is created.

Lifetime of a one-time script

- script is executed when user selects `Execute` on a script file from SD card browser screen.
- script executes until:
    - it returns value different from 0
    - is forcefully closed by user by long press of EXIT key
    - is forcefully closed by system if if it misbehaves (too long runtime, error in code, low memory)

Script interface definition

Every script must include a **return** statement at the end, that defines its interface to the rest of OpenTX code. This statement defines:

- script **init** function (optional)
- script **run** function

For example:

```
-- script body would be here

return { run=run_func, init=init_func }
```

This example defines:

- run_func() function as periodic execution function that takes a key press event as parameter and returns some value
- init_func() function as function that is called one time when script is loaded and begins execution.

Parameter init is optional.

Script initialization

see script initialization

Script execution

The **run** function is the function that is periodically called for the entire lifetime of script. Syntax of run function is different between model scripts and one-time scripts.

local <run_function_name>(event)

```
Parameters:   event    number that contains currently active key press
                       code.
                       If no key is pressed the value is 0.
                       event contains two distinct fields:

                           * what happened (key up, key down, long key
                       press, etc)
                           * which key is/was pressed

                       The actual values of event are usually not
                       important inside Lua script, the event is mainly
                       used as one of parameters for the popupInput()
                       function.

Returns:      0        script will continue execution (run function will
                       be called again

              !=0      script is terminated (ends execution)
```

Example Script

Function Scripts

TODO (blank in Wiki)

Telemetry Scripts

General description

These scripts are used for building customized telemetry screens. Theoretically it is possible to have up to 7 custom telemetry screens, all written in Lua. It is possible to use different scripts on a per model basis.

Anatomy of telemetry script

Location of telemetry scripts

Place them on SD card in the folder /SCRIPTS/«modelname»/telemX.lua where X is a number from 0 to 6. Example: /SCRIPTS/Extra/telem0.lua would be first custom telemetry screen for model Extra.

Lifetime of telemetry script

- script is loaded from SD card and executed when the model is loaded.
- script **init** function is called
- script **background** function is periodically called when custom telemetry screen **is not visible**
- script **run** function is periodically called when custom telemetry screen **is visible**
- script is stopped and disabled if it misbehaves (too long runtime, error in code, low memory)
- all telemetry scripts are stopped while one-time script is running (see Lua One-time scripts)

Script interface definition

Every script must include a **return** statement at the end, that defines its interface to the rest of OpenTX code. This statement defines:

- script **init** function (optional)
- script **background** function
- script **run** function

For example:

```
-- script body would be here

return { run=run_func, init=init_func, background=bckgrnd_func }
```

This example defines:

- *bckgrnd_func()* function as **periodic execution function** that is periodically called when custom telemetry screen **is not visible**
- *run_func()* function as **periodic execution function** that is periodically called when custom telemetry screen **is visible**
- *init_func()* function as function that is **called one time** when script is loaded and begins execution.

Parameter **init** is optional.

Script initialization

see script initialization

Script execution

The **run** or **background** function is the function that is periodically called for the entire lifetime of script. Which one is called depends on the visibility of custom telemetry screen:

- not visible - the **background** function is called. Script should not draw to screen. Usually used to store/process telemetry data.
- visible - **run** function is called. Script should draw its screen.

local <background_function_name>()

```
Parameters:   none

Returns:      none
```

local <run_function_name>(event)

```
Parameters:   event    number that contains currently active key press code. If
                       no key is pressed, the value is 0
                       Event contains two distinct fields:
                         * what happened (key up, key down, long key press, etc).
                         * which key is/was pressed.

Returns:      none
```

Examples

- Telemetry Screen #1
- Telemetry Screen #2

Lua Script Reference

General Syntax

Local vs. Global

All Lua model scripts in OpenTX exist in same Lua environment. This means, that they share global functions and variables. Using global variable with the same name from two different scripts could lead to unpredictable results. Therefore the use of global variables and functions should be avoided!

TODO: how to share data between scripts

Warning

Even variables defined inside local functions without **local** keyword are global. For example if we have two scripts, script1:

```
local function run()
    global_var = 7   -- this one is GLOBAL
    local local_var = "foo"   -- this one is local to script1,
                              -- but visible in all script1 functions
end
```

and script 2:

```
local function run()
    if global_var == 7 then   -- here global_var is already defined from script1
        playFile("Whoopsie.wav")
    end
    if local_var == "foo" then   -- local_var here is nil, because it was not yet assigned
        print("We don't get here, local_var is nil")
    end
end
```

If we execute both scripts, we would hear "Whoopsie.wav" (if it is present on SD card). This means that variable **global_var** is actually global even if it is first defined in some local function in other script.

Local Variables

Script can have any number (limited by memory usage) of local variables, their value is preserved between each call to run function. They are defined as:

```
local simple_number = 4
local some_table = {1, 2, 120}
```

Local variables are only visible to the script that defined them. Two scripts can define a local variable with the same name. These two variables don't share anything, each script has his own instance of variable.

Local Functions

Script can have any number (limited by memory usage) of local functions:

```
local function some_function(a, b, c)
```

```
    local value1 = a + b * c
    return value1
end
```

Local functions are only visible to the script that defined them. Two scripts can define a local function with the same name. These two functions don't share anything, each script has his own instance of function.

Inputs Syntax

Input are only used in model scripts. However the same number format is returned by function ***getValue()***.

Number Format

Inputs are analogue values from openTX that are converted to 16 bit signed integers before they reach Lua scripts.

Analogue values such as sticks and sliders have value in percent **multiplied by 10.24**:

Aileron Stick Value	Input Value to Script
0%	0
60.6%	620
100.0%	1024
-100.0%	-1024

Switches (real and logical) are represented as:

Switch Position	Input Value to Script
down (-100%)	-1024
middle	0
up (100%)	1024

Telemetry values are returned as proper values:

Telemetry Value	Input Value to Script
altitude 120.5m	120.5
A1 voltage 5.47V	5.47
Consumption 1260mAh	1260

Types of Inputs

Source

Source type provides current value of selected OpenTX variable (stick position, slider, channel). User assigns assigns actual source for this input in Custom script menu. Source can be any value OpenTX knows about (inputs, channels, telemetry values, switches, custom functions,...).

Syntax: { name, SOURCE }
Example: { "Aileron", SOURCE }

Defines SOURCE type input with name Aileron. Name length is limited to TODO.

Value

Value type provides constant value that user sets in Custom script menu

Syntax: { name, VALUE, min, max, default }
Example: { "Ratio", VALUE, -100, 100, 0 }

Defines VALUE input with name Ratio that has limits -100 and 100 and default value of 0. Name length is limited to TODO.

Outputs Syntax

Outputs are only used in model scripts.

Syntax: { name1, name2 }
Example: { "Calc", "Out" }

Output name is limited to four characters.

Number Format

Outputs are 16 bit signed integers when they leave Lua script and are then **divided by 10.24** to produce output value in percent:

Output from Script	Output as seen from OpenTX
0	0%
996	97.2%
1024	100%
-1024	-100%

Lua General Functions

getTime()

Returns the time since the radio was started in multiple of 10ms

```
Parameters:   none

Returns:      number   Number of 10ms ticks since the radio was started

Status:       current  Introduced in 2.0.0
```

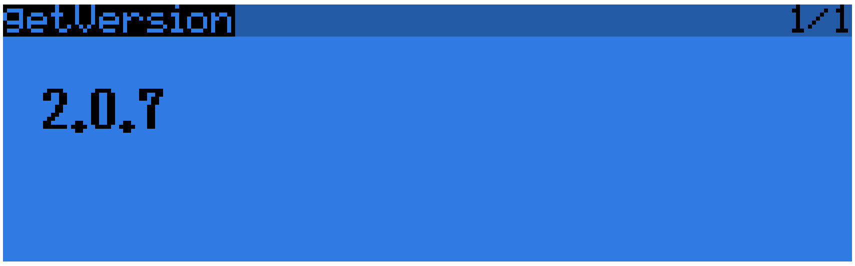
getVersion()

Returns OpenTX version

```
Parameters:   none

Returns:      value    (string) Version (i.e. 2.0.0)

Status:       current  Introduced in 2.0.0
```

Example: Display Version Number

```lua
local function run_func(event)
  lcd.lock()
  lcd.clear()
  lcd.drawScreenTitle("getVersion",1,1)

  lcd.drawText(10,20,getVersion(),MIDSIZE)

  if event == EVT_EXIT_BREAK then   -- Test for Exit Key
    return 1  -- Exit
  else
    return 0
  end
end

return { run=run_func }
```



getVersion 1/1

2.0.7

getValue(source)

Returns the value of a source

```
Parameters:   source   can be a constant (i.e STICK_RUDDER)
                       or a string name (i.e. “altitude”)

Returns:      number   value of source or nil if value is not available.

Status:       current  Introduced in 2.0.0
```

supported source constants

name
MIXSRC_FIRST_INPUT
MIXSRC_Rud
MIXSRC_Ele
MIXSRC_Thr
MIXSRC_Ail
MIXSRC_SA
MIXSRC_SB
MIXSRC_SC
MIXSRC_SD
MIXSRC_SE
MIXSRC_SF
MIXSRC_CH1

This may be a beter reference point until 2.0.6 http://www.rcgroups.com/forums/showpost.php?p=28897780&postcount=298

Supported source names are:

name	value	example
"altitude"	barometric altitude in meters	120.56
"altitude-max"	max barometric altitude in meters	120.56
"vario"	vario speed in m/s	120.56
"tx-voltage"	Tx voltage in V	120.56

"rpm"	RPMs	120.56
"latitude"	GPS latitude in degrees, North is positive	45.5667
"longitude"	GPS longitude in degrees, East is positive	120.5677
"pilot-latitude"	first GPS value (usually pilot position) format same as "latitude"	-12.567
"pilot-longitude"	first GPS value (usually pilot position) format same as "longitude"	-0.567

Names are case sensitive eg getValue("altitude")

playFile(path)

Plays a file from the SD card

```
Parameters:   path     full path to wav file (i.e. “/SOUNDS/en/system/tada.wav”)

Returns:      none     value of source or nil if value is not available.

Status:       current  introduced in 2.0.0
```

Introduced in 2.1.0

If you use a relative path, will append current language to the path.

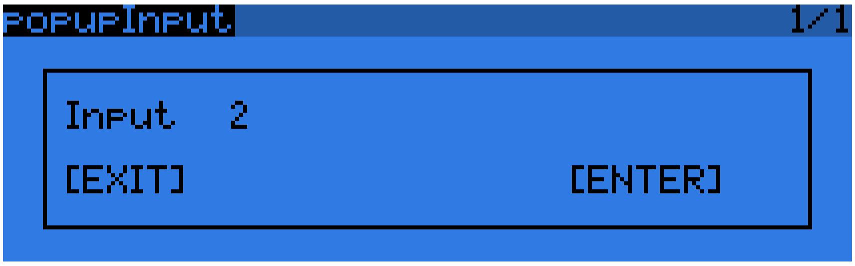
popupInput(title, event, input, min, max)

Raises a popup on screen that allows uses input

Parameters:	title	(string) Text to display
	event	(number) the event variable that is passed in from the Run Function (key pressed)
	input	(number) value that can be adjusted by the +/- keys
	min	(number) min value that input can be decremented by the - key
	max	(number) max value that input can be decremented by the - key
Returns:	result	(string) “OK” ENT pressed (string) “CANCEL” EXIT pressed (number) the result of the input adjustment
Status:	current	introduced in 2.0.0

Example: Pressing +/- will increment/decrement number.

```
local result = 0
local swtch = 0

local function run_func(event)
  lcd.lock()
  lcd.clear()
  lcd.drawScreenTitle("popupInput",1,1)

  result = popupInput("Input", event, swtch, -10, 10)
  lcd.drawNumber(62,24,swtch,0)
  if result == "OK" then
    return 0 -- ignore
  elseif result == "CANCEL" then
    return 1 -- exit program
  else
    swtch = result   -- it is number
    return 0
  end

end

return { run=run_func }
```



Input 2

getGeneralSettings()

Returns a table containing battMin and battMax

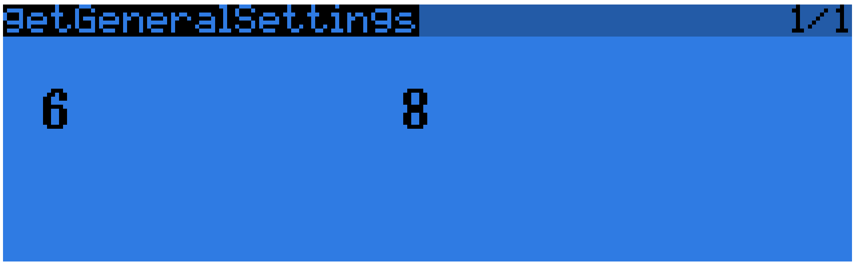
Parameters:	none		
Returns:	value	(general-table)	Min and Max battery value
Status:	Current	Introduced 2.0.6	

General Table Format

battMin	(number)	Minimum Battery
battMax	(number)	Maximum Battery

Example: Display Min and Max Battery volatge

```
local settings=getGeneralSettings()
lcd.drawNumber(10,20,settings.battMin,LEFT+MIDSIZE)
lcd.drawNumber(100,20,settings.battMax,LEFT+MIDSIZE)
```



```
getGeneralSettings                1/1

 6              8
```

playNumber(number, unit, att)

```
Parameters:   number   (integer)

              unit     (integer)

              att      (integer)

Returns:      0

Status:       Current  Introduced 2.0.0
```

Unit table

Unit	Sound	Description
0		
1	Volts	0115.wav
2	Amperes	0118.wav
3	Meters per Second	0120.wav
4		
5	Speed	KMH or Knots depending on radio setting (imperial/metric)
6	Height	Number is meters. Converted to feet for Imperial.
7	Temperature	Number is celcius. converted to F if radio is in Imperial.
8	Percent	
9	Millions	
10	MH	
11	Watts	
12	DB	
13	Feet	
14	Speed	KMH or Knots depending on radio setting (imperial/metric)
15	Hours	
16	Minutes	
17	Seconds	
18	RPM	

19	Gee	
20	Degrees	
21		
22		
23	Point Zero	
24	Point two	
25	Point four	
26	Point six	
27	Point eight	

Attr

ATTR	
PREC1	Add decimal place to number. ie if number is 58, will announce 5.8

defaultStick(channel)

Get stick that is assigned to a channel. See Default Channel Order in General Settings

Parameters:	number	(number) Channel Number
Returns:	value	(number) Stick assigned to this channel
Status:	Current	Introduced 2.0.0

defaultChannel(stick)

Get channel assigned to stick. See Default Channel Order in General Settings

```
Parameters:   stick    (number) Stick Number

Returns:      nil

              value    (number) Channel Number

Status:       Current  Introduced 2.0.0
```

killEvents(key-event)

Removes key-event from event

```
Parameters:   key-event    (number) Key events to remove (mask)

Returns:      nil

Status:       Current      introduced in 2.0.0
```

Key Events	comments
EVT_MENU_BREAK	
EVT_PAGE_BREAK	
EVT_PAGE_LONG	
EVT_ENTER_BREAK	
EVT_ENTER_LONG	
EVT_EXIT_BREAK	
EVT_PLUS_BREAK	
EVT_MINUS_BREAK	
EVT_PLUS_FIRST	
EVT_MINUS_FIRST	
EVT_PLUS_RPT	
EVT_MINUS_RPT	

getFieldInfo(fieldname)

Gets detailed information about field.

```
Parameters:   fieldname    (string) short field name. see LUA Source List

Returns:      value        (field-table) details of field

Status:       Current      2.0.8
```

Field Table Format

```
        id               (number) Field index

        name             (string) Short Name

        desc             (string) Long description for field
```

playDuration(duration,playtime)

```
Parameters:   duration     (integer)

              playtime     (boolean)

Returns:      none

Status:       Planned      2.1.0
```

playTone(frequency, length, pause, attr, frequnecyInc)

```
Parameters:   frequency     (integer) Frequency of tone, in Hertz

              length        (integer) Length of tone in ms

              pause         (integer)

              attr          (integer) See table below

              frequencyInc  (integer)

Returns:      none

Status:       Planned       2.1.0
```

Attr	**Description**
PLAY_NOW	Play immediately
PLAY_BACKGROUND	Place in background queue

Lua Model Functions

Please note that writing (even the same value) to model settings will cause the a write to EEPROM. This could occur within 5 seconds of the change or when model is unloaded or radio switched off.

model.getTimer(timer)

Returns model timer

```
Parameters:   timer    (number) timer number

Returns:      nil      unknown timer number.

              value    (timer-table) timer data

Status:       current  introduced in 2.0.0
```

Timer Table Format

```
        mode            (number) timer trigger source: off, abs, stk,  stk%,
                        sw/!sw, !m_sw/!m_sw

        start           (number) start value [seconds], 0 for up timer, 0> down
                        timer

        value           (number) current value [seconds]

        countdownBeep   (number) countdown beep
                        (0-silent, 1-beeps, 2-voice)

        minuteBeep      (boolean) minute beep

        persistent      (number) persistent timer
```

Example:

```
--get timer data into tim1
tim1 = model.getTimer(1)
--access returned values as tim1.<value>
if tim1.value > 0 then
    --do something
end
```

model.setTimer(timer, data)

Sets model timer

```
Parameters:   timer    (number) timer number

              data     (timer-table) new timer data. See

Returns:      none

Status:       current  introduced in 2.0.0
```

see model.getTimer(timer) for timer table format

model.getInputsCount(input)

Returns number of lines for given input

```
Parameters:   input    (unsigned number) input number (0 -> max inputs - 1)

Returns:      value    (unsigned number) number of configured lines for given
                       input.

Status:       current  introduced in 2.0.0
```

model.getInput(input, line)

Returns input data for given input and line number

```
Parameters:   input    (unsigned number) input number (0 -> max inputs - 1)

              line     (unsigned number) input line (0 -> max lines - 1)

Returns:      value    (input-table) input data

Status:       current  introduced in 2.0.0
```

Input-Table Format

```
        name             (string) input line name

        source           (number) input source index

        weight           (number) input weight

        offset           (number) input offset
```

model.insertInput(input, line, value)

Inserts an Input at specified line

```
Parameters:   input    (unsigned number) input number (0 -> max inputs - 1)

              line     (unsigned number) input line (0 -> max lines - 1)

              value    (input-table) see model.getInput(input, line)

Returns:      none

Status:       current  introduced in 2.0.0
```

model.deleteInput(input, line)

Delete line from specified input

```
Parameters:   input    (unsigned number) input number (0 -> max inputs - 1)

              line     (unsigned number) input line (0 -> max lines - 1)

Returns:      none

Status:       current  introduced in 2.0.0
```

model.deleteInputs()

Delete all Inputs

```
Parameters:   none

Returns:      none

Status:       current  introduced in 2.0.0
```

model.defaultInputs()

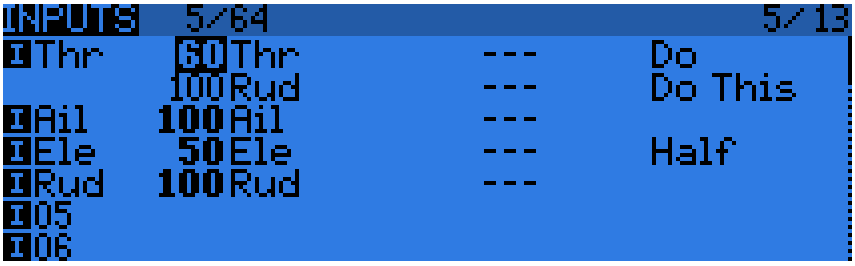
Set all inputs to Defaults.

```
Parameters:   none

Returns:      none

Status:       current   introduced in 2.0.0
```

Example:

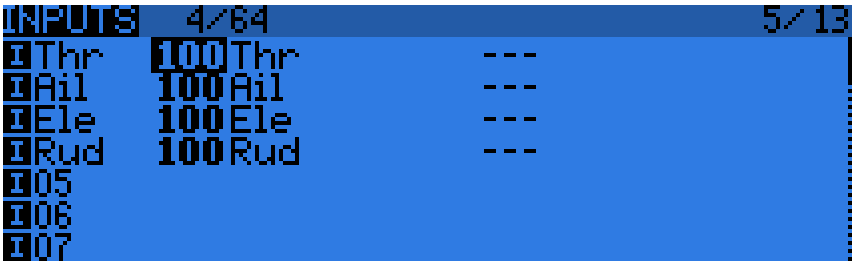
```
local function run_func(event)
  model.defaultInputs()
  return 1
end

return { run=run_func}
```

Radio before script is run



```
INPUTS  5/64                              5/13
I Thr    60 Thr          ---      Do
        100 Rud          ---      Do This
I Ail   100 Ail          ---
I Ele    50 Ele          ---      Half
I Rud   100 Rud          ---
I 05
I 06
```

Radio after script is run



```
INPUTS  4/64                              5/13
I Thr   100 Thr          ---
I Ail   100 Ail          ---
I Ele   100 Ele          ---
I Rud   100 Rud          ---
I 05
I 06
I 07
```

model.getMixesCount(channel)

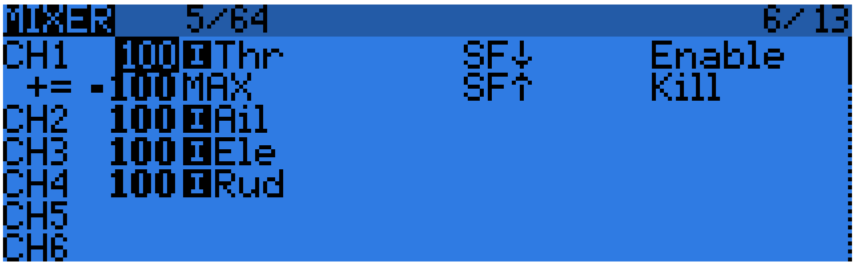
Get the number of Mixer lines that the specified Channel has

```
Parameters:   channel   (number) Channel number to look up. Zero numbered (i.e.
                        CH1 is 0)

Returns:      value     (number) number of line

Status:       current   introduced in 2.0.0
```

Example

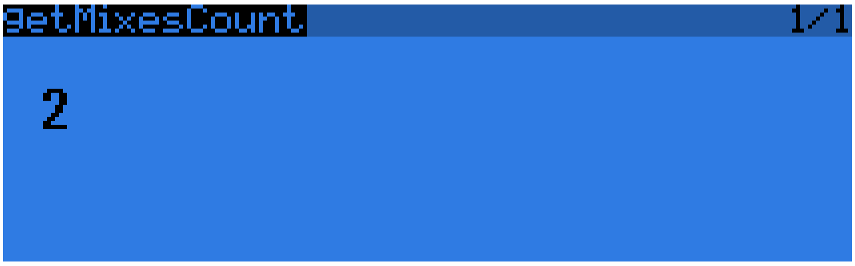
```
mix = model.getMixesCount(0)
lcd.drawNumber(10,20,mix,LEFT+MIDSIZE)
```

Radio Configuration



```
MIXER     5/64                          6/13
CH1   100 Thr              SF↓       Enable
  += -100 MAX              SF↑       Kill
CH2   100 Ail
CH3   100 Ele
CH4   100 Rud
CH5
CH6
```

Result



```
getMixesCount                           1/1

  2
```

model.getMix(channel, line)

Get configuration for specified Mix

Parameters:	channel	(number) Channel number to look up. Zero numbered (i.e. CH1 is 0)
	line	(number) line number of Mix. Zero numbered.
Returns:	value	(mix-table) line details
	nil	invalid parameters
Status:	current	introduced in 2.0.0

Mix-Table Format

name	(string)
source	(number)
weight	(number) Weight value or gVar1..9 = 4096..4114, -gVar1..9 = 4095.. 4087
offset	(number) Offset value or gVar1..9 = 4096..4114, -gVar1..9 = 4095.. 4087
switch	(number) Switch Number
multiplex	(number) 0=ADD, 1=MULTIPLY, 2=REPLACE

model.insertMix(channel, mix, value)

Insert a mixer line into Channel

Parameters:	channel	(number) Channel number to look up. Zero numbered (i.e. CH1 is 0)
	line	(number) line number to insert. Existing line will be moved down one line
	value	(mix-table) see model.getMix(channel, line)
Returns:	nil	
Status:	current	introduced in 2.0.0

model.deleteMix(channel, mix)

Delete mixer line from specified Channel

Parameters:	channel	(number) Channel number to look up. Zero numbered (i.e. CH1 is 0)
	line	(number) line number to delete. Existing lines will be moved up one line
Returns:	nil	
Status:	current	introduced in 2.0.0

model.deleteMixes()

Removes **ALL** lines from **ALL** channels

```
Parameters:   none

Returns:      none

Status:       current  introduced in 2.0.0
```

model.getLogicalSwitch(switch)

Get Logical Switch parameters

```
Parameters:   switch    (number) Logical Switch Number

Returns:      value     (switch-table)

Status:       current  introduced in 2.0.0
```

Switch-Table Format

```
        func             (number)

        v1               (number)

        v2               (number)

        v3               (number)

        and              (number)

        delay            (number)

        duration         (number)
```

model.setLogicalSwitch(switch, value)

Set Logical Switch parameters

```
Parameters:   switch    (number) Logical Switch Number

              value     (switch-table) See model.getLogicalSwitch(switch).

Returns:      none

Status:       current  introduced in 2.0.0
```

model.getCustomFunction(function)

Get Special Functions

Parameters:	function	(number)	Special Function Number
Returns:	value	(function-table)	
Status:	current	introduced in 2.0.0	

Function-Table Format

switch	(number) Switch Number
func	(number) Action Number
name	(string) Name of track to play, only returned if Action is play track or sound.
value	(number)
mode	(number)
param	(number)
active	(number) 0 = !Enabled, 1 = Enabled

Function Table. Caution, table behavior changes depending on the function. Need to determine a good way to show this information. The sound orientated functions in particular affect the usages of fields.
Unused variables will contain values from a previous function i.e. if you change a function and value is no longer used, it will still contain the value from the old function.

Action	Parameter	func	value	mode	param
Safety CHx	125..-125	0	Parameter Value		Channel 0-21
Trainer		1			0
Trainer RUD		1			1
Trainer ELE		1			2
Trainer THR		1			3
Trainer AIL		1			4
Instant Trim		2			
Play Sound		10			
Reset	Timer1	3	0		
Reset	Timer2	3	1		
Reset	All	3	2		
Reset	Telemetry	3	3		
Set Timer 1	0..n	4	Parameter Value		0
Set Timer 2	0..n	4			1
Vario		18			
Play Value	- - -	12			
Start Logs	0.0..25.5	20	Parameter value * 10		
Volume	source	6	See source table		
Backlight		21			
Background Music					
Background Music Pause		17			
Adjust GV1..9	Value	5	Parameter Value	0	GV 0..8
Adjust GV1..9	Source	5	See source table	1	GV 0..8
Adjust GV1..9	GVAR	5	GV 0.8	2	GV 0..8
Adjust GV1..9	Increment	5	0 = -1, 1 = +1	3	GV 0..8

model.setCustomFunction()

```
Parameters:   function   (number) Special Function Number

              value      (function-table) See model.getCustomFunction(function)

Returns:      none

Status:       current    introduced in 2.0.0
```

model.getOutput(index)

Get servo details

```
Parameters:   index      (number) Channel Number (Zero Numbered)

Returns:      value      (output-table)

              nil

Status:       current    2.0.0
```

Output-Table Format

```
        name             (string) Channel Name

        min              (number) Minimum % * 10

        max              (number) Maximum % * 10

        offset           (number) Subtrim * 10

        ppmCenter        (number) Offset from PPM Center. 0 = 1500

        symetrical       (number) Linear Subtrim 0 = Off, 1 = On

        revert           (number) Direction 0 = ---, 1 = INV

        curve            (number) Curve number 0..31 = curve(1)..(32), -2..-33
                         = !curve(1)..(32)
                         Nil if no curve set
```

model.setOutput(index, value)

Set servo properties

```
Parameters:   index      (number) Channel Number (Zero Numbered)

              value      (output-table) see model.getOutput(index)

Returns:      nil

Status:       current   introduced in 2.0.0
```

model.getInfo()

Get current Model information

```
Parameters:   none

Returns:      value    (model-table) Current Model information

Status:       current  introduced in 2.0.6
```

Model-Table Format

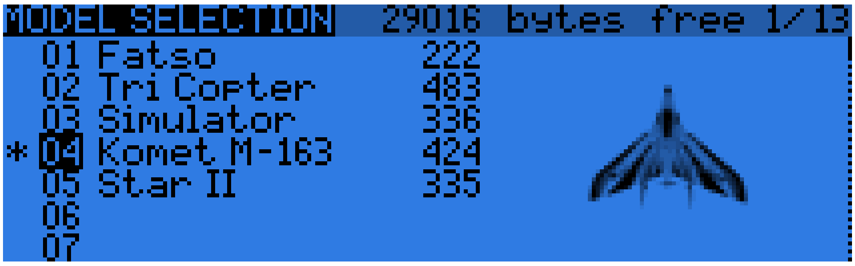
```
        name              (string) model name

        id                (number) receiver number
```

Example:

Get the Model name and Number of current Model

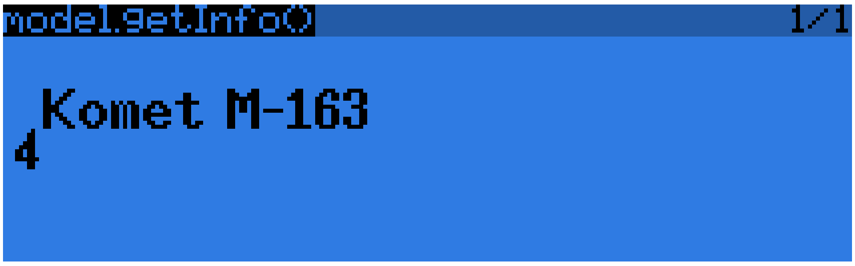
```
modelinfo = model.getInfo()
lcd.drawText(10,20,modelinfo.name,MIDSIZE)
lcd.drawNumber(10,30,modelinfo.id,MIDSIZE)
```

Current Models in Radio



```
MODEL SELECTION  29016 bytes free 1/13
  01 Fatso          222
  02 Tri Copter     483
  03 Simulator      336
* 04 Komet M-163    424
  05 Star II        335
  06
  07
```

Script Output



```
model.getInfo()                     1/1

 Komet M-163
4
```

model.setInfo

Set the current Model Name and Number

```
Parameters:   value    (model-table) Current Model information

Returns:      none

Status:       current  introduced in 2.0.6
```

Example:

model.getGlobalVariable(gvar, flightmode)

Get value of Gvar for specified Flight Mode

Parameters:	gvar	(number) gVar number
	flightmode	(number) flight mode
Returns:	value	(number) gVar value. If value is > 1024 then gVar is getting its value from another flightmode. Subtract 1025 to get the actual flightmode number.
Status:	current	introduced in 2.0.0

model.setGlobalVariable(gvar, flightmode, value)

Parameters:	gvar	(number)	gVar number
	flightmode	(number)	flight mode
	value	(number)	value of GVAR
Returns:	none		
Status:	current	introduced in 2.0.0	

model.getTelemetryChannel(idx)

```
Parameters:   idx           (integer) Channel number. A1..A4 (zero numbered)

Returns:      value         (telemetry-table)

              nil

Status:       current       introduced in 2.0.8
```

Telemetry-Table Format

```
        range            (number) Range

        offset           (number) Offset

        alarm1           (number) Low Alarm

        alarm2           (number) Critical Alarm

        unit             (integer) see Unit table
```

Unit Table

Unit	index
Volts (V)	0
Amps (A)	1
Speed (m/s or ft/s)	2
Raw (-)	3
Speed (km/h or miles/h)	4
Meters (m or ft)	5
Temp ()	6
Fule (%)	7
mAmps (mA)	8

model.setTelemetryChannel(idx, value)

Parameters:	idx	(integer) Channel number
	value	(telemetry-table) see model.getTelemetryChannel(idx)
Returns:	none	
Status:	current	introduced in 2.0.8

Lua Display Functions

lcd.lock()

Prevents main OpenTX code from modifying LCD screen. This lock is reset every time script is run and must be set again if script wants LCD to be locked on each iteration.

```
Parameters:   none

Returns:      none

Status:       current  introduced in 2.0.0
```

lcd.clear()

Clears the LCD screen

```
Parameters:   none

Returns:      none

Status:       current  introduced in 2.0.0
```

lcd.drawPoint(x, y)

Draws a single pixel at (x,y) position

```
Parameters:   x         (integer)  x position in pixels

              y         (integer)  y position in pixels

Returns:      none

Status;       current  introduced in 2.0.0
```

Note: Taranis has an LCD display width of 212 pixels and height of 64 pixels. Position (0,0) is at top left. Y axis is negative, top line is 0, bottom line is 63.

lcd.drawLine(x1, y1, x2, y2)

Draws a line from (x1,y1) to (x2,y2)

```
Parameters:   <x1,y1>  (integer) start position. See lcd.drawPoint()

              <x2,y2>  (integer) end position. See lcd.drawPoint()

Returns:      none

Status:       current  introduced in 2.0.0
```

lcd.drawRectangle(x, y, width, height)

Draws a rectangle from top left corner (x,y) of specified width and height

Parameters:	<x,y>	(integer) top left text position. See lcd.drawPoint()
	width	(integer) width in pixels
	height	(integer) height in pixels
Returns:	none	
Status:	current	introduced in 2.0.0

lcd.drawText(x, y, text, att)

Draws a text beginning at (x,y)

```
Parameters:    <x,y>     (integer) top left text position. See lcd.drawPoint()

               text      (string) text to display.

               att       text attributes

Returns:       none

Status:        current   introduced in 2.0.0
```

Text Attributes:
All att values can be combined together using the + character. ie BLINK + DBLSIZE. See the Appendix for available characters in each font set.

value	font	companion version	Note
0	normal font		
DBLSIZE	double size font		
MIDSIZE	mid sized font		
SMLSIZE	small font		
INVERS	inverted display		
BLINK	blinking text		
XXLSIZE	jumbo font	2.0.6	
LEFT	left justify	2.0.6	Only for drawNumber

Special Characters

Hex	Decimal	Function	Example
0x1D	29	Tab	local string = 'hello\31\110world' tab inserted btween hello and world
0x1E	30	Newline	local string = 'hello\30world' world will print on next line
0x1F	31	x co-ord prefix.	local string = 'hello\31\110world' world will print from x=110
< 0x20			all other codes will insert an extended space

lcd.drawSwitch(x, y, switch, att)

Draws a text representation of switch at (x,y)

Parameters:	<x,y>	(integer) top left text position. See lcd.drawPoint()
	switch	(integer) number of switch to display, negative number displays negated switch
	att	(integer) text attribute See lcd.drawText(x, y, text, att)
Returns:	none	
Status:	current	introduced in 2.0.0

Note: Testing shows that as of 2.0.8, only the SMLSIZE BLINK & INVERS attribute works correctly.

lcd.drawPixmap(x, y, path)

Draws a bitmap at (x,y)

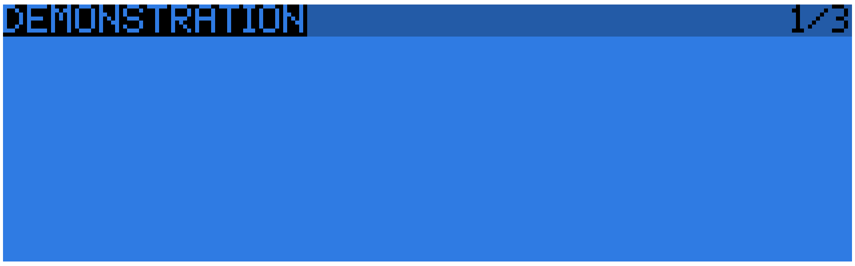
Parameters:	<x,y>	(integer) top left text position. See lcd.drawPoint()
	path	(string) full path to the bitmap on SD card (i.e. "/BMP/test.bmp")
Returns:	none	
Status:	current	introduced in 2.0.0

lcd.drawScreenTitle(title, idx, cnt)

Draws a title bar

```
Parameters:   title    (string) text for the title

              idx      (integer) page number

              cnt      (integer) total number of pages. Only used as indicator on
                       the right side of title bar. (i.e. idx=2, cnt=5, display
                       “2/5”)

Returns:      none

Status:       current  introduced in 2.0.0
```

Example: lcd.drawScreenTitle("DEMONSTRATION",1,3)



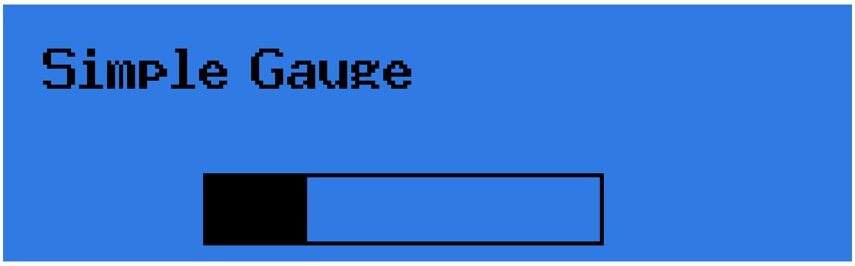
DEMONSTRATION 1/3

lcd.drawGauge(x1, y1, w, h, fill, maxfill)

Draws a simple gauge that is filled based upon fill value.

```
Parameters:   <x1,y1>   (integer) start position. See lcd.drawPoint()
              w         (integer) width in pixels
              h         (integer) height in pixels
              fill      (integer) amount of fill to apply
              maxfill   (integer) total value of fill
Returns:      none
Status:       current   introduced 2.0.6
```

Example: lcd.drawGauge(50, 42, 100, 18, 25, 100)



lcd.drawChannel(x, y, source, att)

Draw the value of a source. Equivalent to lcd.drawText(x, y, model.getValue(source), att)

`Parameters:`	`<x,y>`	`(number) top left text position. See lcd.drawPoint()`

	`source`	`can be a constant (i.e STICK_RUDDER)`
		`or a string name (i.e. “altitude”). See getValue(source)`

	`att`	`text attributes. See lcd.drawText(x, y, text, att)`
`Returns:`	`none`	
`Status:`	`current`	`Introduced 2.0.6`

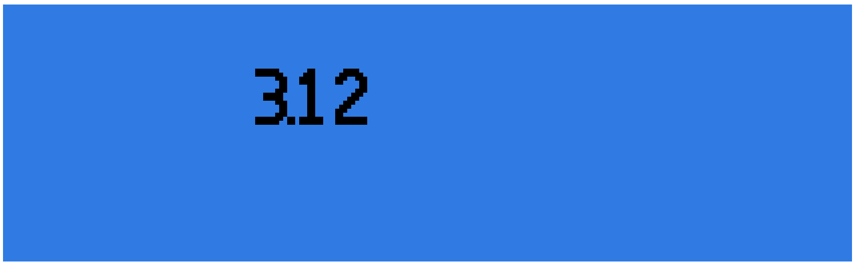
lcd.drawNumber(x, y, number ,att)

Draw a number on the display

```
Parameters:    <x,y>     (integer) top left text position. See lcd.drawPoint()

               number    (integer) value to display

               att       text attributes. See lcd.drawText(x, y, text, att)

Returns:       none

Status:        current   introduced 2.0.0
```

To display a floating point number, use the PREC1 or PREC2 attributes.

lcd.drawNumber(62,15,312,DBLSIZE + PREC2 + LEFT)



3.12

lcd.drawTimer(x, y, value, att)

Display a value formatted as time

`Parameters:`	`<x,y>`	`(integer) top left text position. See` [`lcd.drawPoint()`]()
	`value`	`(timer) A timer value`
	`att`	`text attributes. See` [`lcd.drawText(x, y, text, att)`]()
`Returns:`	`none`	
`Status:`	`current`	`Introduced 2.0.6`

lcd.getLastPos()

Returns the last X position from previous output

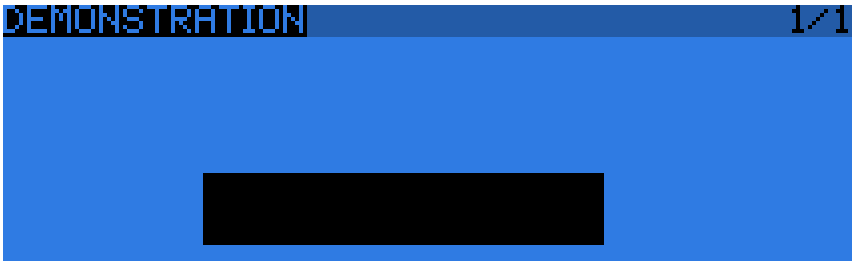
```
Parameters:   none

Returns:      number   (integer) X position

Status:       current  Introduced 2.0.6
```

lcd.drawFilledRectangle(x, y, w, h, att)

Draw a rectangle on the screen as a solid block

Parameters:	<x,y>	(number) top left text position. See lcd.drawPoint()
	width	(number) width in pixels
	height	(number height in pixels
	att	text attributes. See lcd.drawText(x, y, text, att)
Returns:	none	
Status:	current	Introduced 2.0.0

Example: lcd.drawFilledRectangle(50,42,100,18,0)

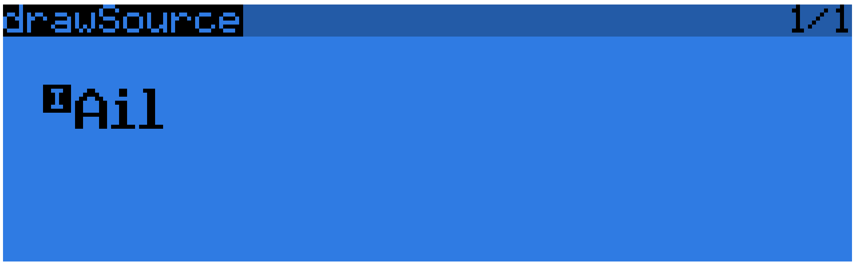


lcd.drawSource(x, y, source, att)

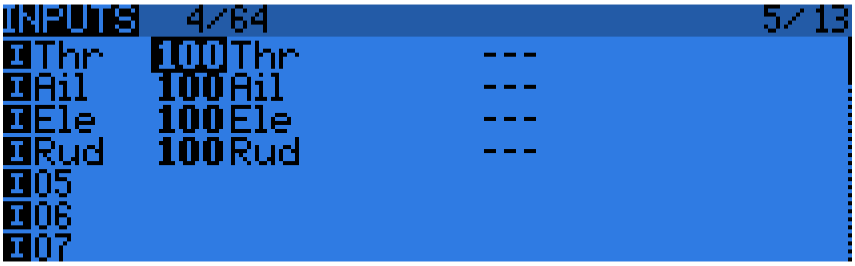
Displays the name of the corresponding input as defined by the source.

```
Parameters:    <x,y>     (number) top left text position. See lcd.drawPoint()

               source    (number) Input index number

               att       text attributes. See lcd.drawText(x, y, text, att)

Returns:       none

Status:        current   Introduced 2.0.0
```

Example: lcd.drawSource(10,20,2,LEFT+MIDSIZE)



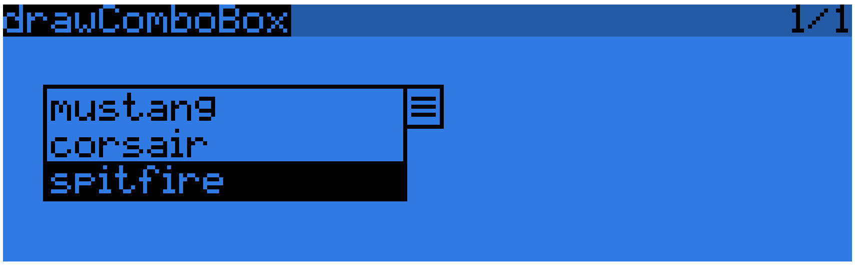
Radio configuration Screen, showing that Input 02 is AIL



lcd.drawCombobox(x, y, w, list, idx, flag)

Parameters:	<x,y>	(integer) top left text position. See lcd.drawPoint()
	w	(integer) width of comboBox
	value	(combo-list) A array of items to display in the combo box
	idx	(integer) Index of entry to highlight
	flag	(integer) 0 Collapsed, 1 = Expanded
Returns:	none	
Status:	current	Introduced 2.0.0

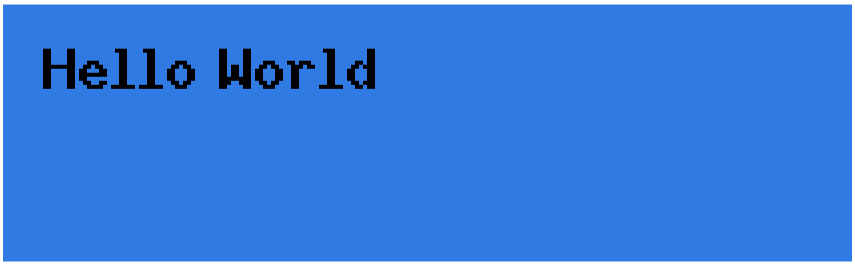
Example:
table[1] = "mustang"
table[2] = "corsair"
table[3] = “spitfire”
lcd.drawCombobox(10,20,100,table,2,1)



Script Examples

One-Time Script Example: Hello World

A simple demonstration script to display “Hello World” on the screen. Script closes when you use the momentary switch SH.



Hello World

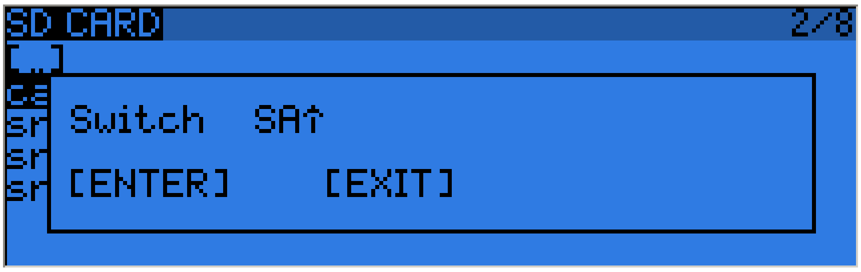
- To install, copy the code below to a file called HelloWorld.lua
- Place the file in the SCRIPTS folder of the SD card
- Browse the SD card contents to the SCRIPTS folder and press enter on the HelloWorld.lua entry
- Execute script
- Use the Momentary Switch SH to close the script

```lua
local function init_func()
  local switch_value = 0
end

local function run_func(event)
  lcd.lock()
  lcd.clear()
  lcd.drawText(10,10,"Hello World",MIDSIZE)
  switch_value = getValue(99)

  if switch_value > 100 then
    return 1  -- Exit
  else
    return 0
  end
end

return { run=run_func, init=init_func }
```

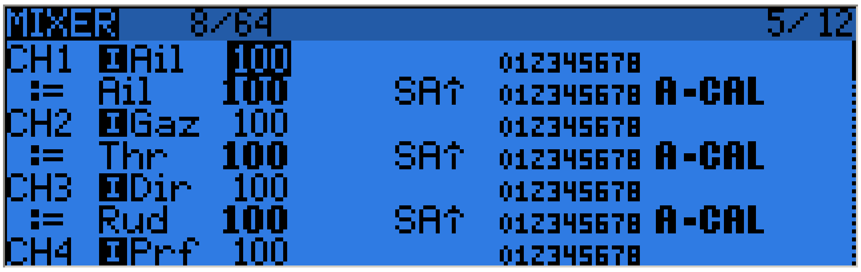
One-Time Script Example: Generic Template

This is a template that I made for somebody who wanted to use a switch in order to check the limits of each channel (without any dual rates, expos, offsets etc. applied).

The inserted lines in the mixes are named "A-CAL", which allows the user either to change the switch by calling again the Lua script with another switch, or to remove all inserted lines by selecting "---" as a switch.



```
SD CARD                                     2/8
[..]
ca
sr  Switch   SA↑
sr
sr  [ENTER]      [EXIT]
```

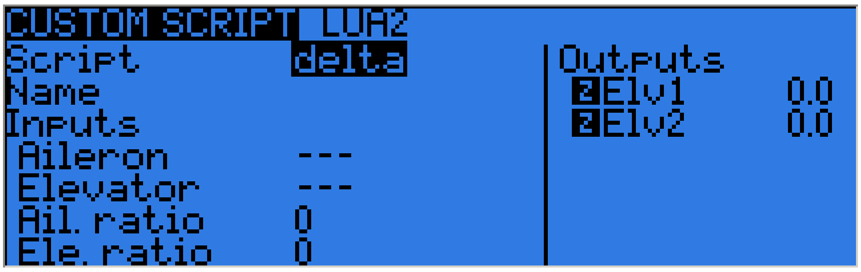


```
MIXER     8/64                              5/12
CH1  I Ail  100             012345678
 :=    Ail  100      SA↑    012345678 A-CAL
CH2  I Gaz  100             012345678
 :=    Thr  100      SA↑    012345678 A-CAL
CH3  I Dir  100             012345678
 :=    Rud  100      SA↑    012345678 A-CAL
CH4  I Prf  100             012345678
```

```lua
local swtch = 0

local function init()
  for channel = 0, 32, 1 do
    for idx = 0, model.getMixesCount(channel), 1 do
      mix = model.getMix(channel, idx)
      if mix ~= nil and mix.name == "A-CAL" then
        swtch = mix.switch
      end
    end
  end
end

local function removeTemplate()
  for channel = 0, 32, 1 do
    for idx = 0, model.getMixesCount(channel), 1 do
      mix = model.getMix(channel, idx)
      if mix ~= nil and mix.name == "A-CAL" then
        model.deleteMix(channel, idx)
      end
    end
  end
end

local function applyTemplate()
  for channel = 0, 32, 1 do
    count = model.getMixesCount(channel)
    if count > 0 then
      first_mix = model.getMix(channel, 0)
      mix_source = first_mix["source"]
      if mix_source >= 1 and mix_source <= 32 then
        input = model.getInput(mix_source-1, 0)
        mix = { name="A-CAL", source=input["source"], weight=100, switch=swtch, multiplex=REPLACE }
        model.insertMix(channel, count, mix)
      else
        mix = { name="A-CAL", source=mix_source, weight=100, switch=swtch, multiplex=REPLACE }
        model.insertMix(channel, count, mix)
      end
    end
  end
end

local function run(event)
  lcd.lock()
  result = popupInput("Switch", event, swtch, -SWSRC_LAST, SWSRC_LAST)
  lcd.drawSwitch(62, 24, swtch, 0);
  if result == "OK" then
    removeTemplate()
    if swtch ~= 0 then
      applyTemplate()
    end
    return 1
  elseif result == "CANCEL" then
    return 1
  else
    swtch = result
    return 0
  end
end

return { init=init, run=run }
```

Model Script Example: Delta Mixer

This example shows how to setup a single delta mix, with its configuration pages (2 sources with a weight on each). It should not be used for real model! This example has four inputs and two outputs:



```
CUSTOM SCRIPT LUA2
Script        delta     | Outputs
Name                    |  Elv1      0.0
Inputs                  |  Elv2      0.0
 Aileron       ---      |
 Elevator      ---      |
 Ail. ratio    0        |
 Ele. ratio    0        |
```

```lua
-- this is comment in LUA

-- inputs definition, here we use variable named inp.
--
local inp = {
             -- first input: user defined input that will be
             -- displayed as "Aileron" in setup screen
              { "Aileron", SOURCE },
              { "Elevator", SOURCE },
            -- third input: user defined constant value
              { "Ail. ratio", VALUE, -100, 100, 0 },
              { "Ele. ratio", VALUE, -100, 100, 0 }
            }

-- outputs definition
local out = { "Elv1", "Elv2" }

-- periodic run function
-- order of parameters follows inp definition,
-- first parameter is "Aileron", second is "Elevator", etc...
local function run_func(input1, input2, ratio1, ratio2)
  -- remember to use local modifier for variables,
  -- if omitted variable value1 would be GLOBAL!

  -- input1 has current value of input that is selected
  -- under "Aileron" in script setup screen
  local value1 = (input1 * ratio1) / 100
  local value2 = (input2 * ratio2) / 100
  -- again, REMEMBER local
  local elevon1 = value1 + value2
  local elevon2 = value1 - value2

  -- now return outputs
  -- elevon1 returned for "Elv1", elevon2 for "Elv2"
  return elevon1, elevon2
end

-- declaration of interface (we do not use init in this example)
-- this is where we link our local variables and funcitons to the OpenTX
-- run_func() is defined as run function
-- inp table is defined as input
-- out table is defined as output
return { run=run_func, input=inp, output=out}
```

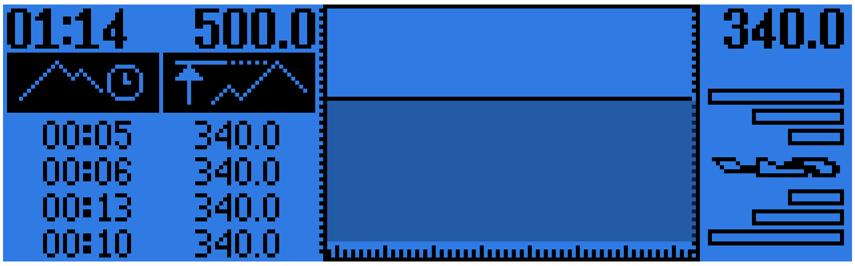
See also:

Lua Scripting in OpenTX

- Inputs syntax
- Outputs syntax

Telemetry Script Example: Screen #1

This is the first Lua telemetry screen script. It can be used to add an additional telemetry screen to any model. OpenTX firmware version 2.0.4 or greater is needed to use the script. You can edit the script yourself to change what information that is displayed.



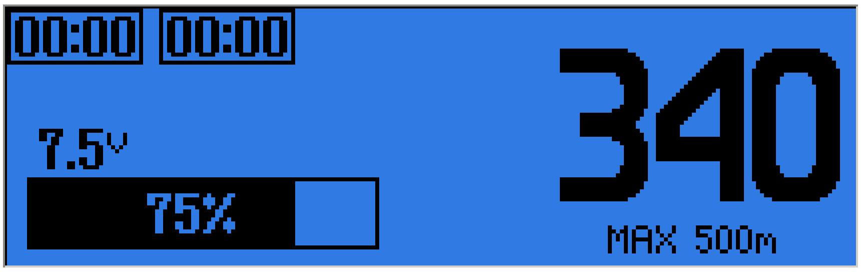
To Install Script:

- Download the telemetry screen script from here: [Download Link]
- Create a folder on the radio microSD card called SCRIPTS (if it does not already exist)
- Create a new subfolder in the SCRIPTS folder. Give the subfolder the same name as the model that will use the script.
- Place the script file in the folder. The path should read: /SCRIPTS/modelname/telem1.lua
- Create a subfolder in the modelname folder called BMP. The Path will become /SCRIPTS/modelname/BMP
- Place the the two bitmap files in the BMP folder (altitude-0.bmp and altitude-1.bmp)

That is it. The new telemetry screen should now automatically appear for the model.

Telemetry Script Example: Screen #2

This is the second Lua telemetry screen example script. It can be used to add an additional telemetry screen to any model. OpenTX firmware version 2.0.6 or greater is needed to use the script. The script will display two timers, the battery value and the altitude. The altitude is displayed using a new very large font. You can edit the script yourself to change what information that is displayed.



To Install Script:

- Download the telemetry screen script from here: Download Link
- Create a folder on the radio microSD card called SCRIPTS (if it does not already exist)
- Create a new subfolder in the SCRIPTS folder. Give the subfolder the same name as the model that will use the script.
- Place the script file in the folder. The path should read: /SCRIPTS/modelname/telem2.lua

That is it. The new telemetry screen should now automatically appear for the model.

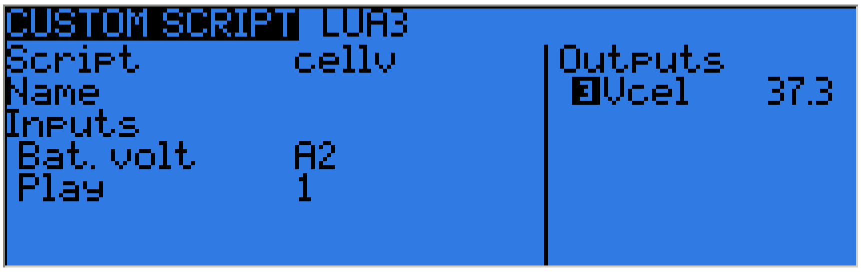
Model Script Example: Automatic Battery Cell Detection

This model script calculates the number of LiPo cells in connected battery and outputs a voltage of one cell. It can be used for automatic setting of low battery alarm in models where different cell count batteries are used interchangeably.

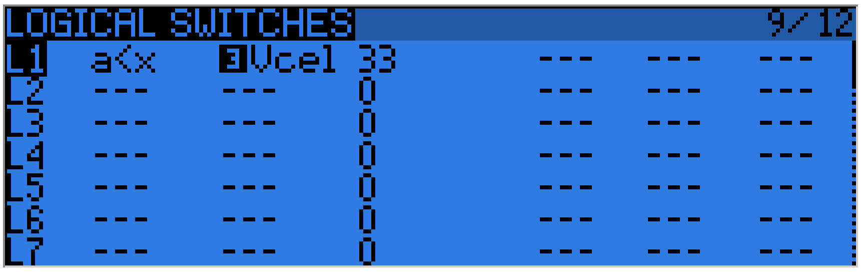
Script algorithm:

- waits for new battery (voltage change from zero to some value)
- calculates number of cells based on battery voltage
- outputs battery voltage divided by number of cells
- when new battery is detected, steps repeat

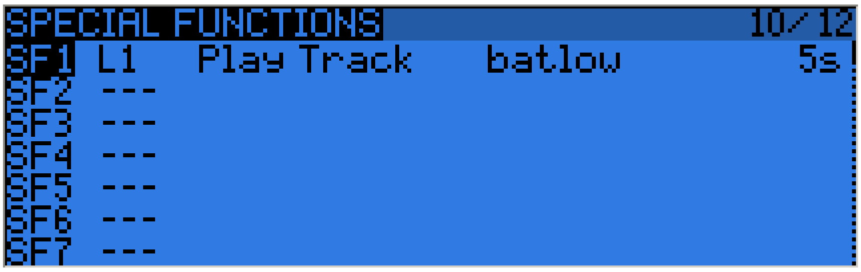
Setup screens:



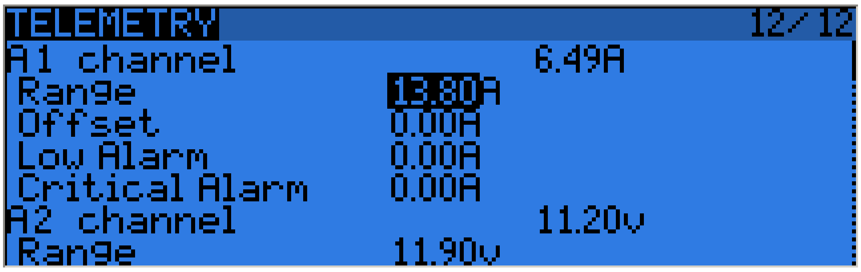
```
CUSTOM SCRIPT LUA3
Script         cellv          Outputs
Name                           3Vcel     37.3
Inputs
 Bat. volt     A2
 Play          1
```



```
LOGICAL SWITCHES                                9/12
L1  a<x    3Vcel 33       ---   ---   ---
L2  ---    ---   0        ---   ---   ---
L3  ---    ---   0        ---   ---   ---
L4  ---    ---   0        ---   ---   ---
L5  ---    ---   0        ---   ---   ---
L6  ---    ---   0        ---   ---   ---
L7  ---    ---   0        ---   ---   ---
```



```
SPECIAL FUNCTIONS                               10/12
SF1 L1   Play Track    batlow              5s
SF2 ---
SF3 ---
SF4 ---
SF5 ---
SF6 ---
SF7 ---
```



```
TELEMETRY                                       12/12
A1 channel                     6.49A
 Range                 13.80A
 Offset                0.00A
 Low Alarm             0.00A
 Critical Alarm        0.00A
A2 channel                     11.20v
 Range                 11.90v
```

This script must be placed onto SD card into folder /SCRIPTS/MIXES/. Script must be activated and its inputs and outputs set in Model settings Custom scripts page.

This example shows battery voltage (A2) is 11.2 Volts, detected cell count was 3 and calculated voltage for one cell is 3.73 Volts. Script output Vcel is multiplied by 10, so 3.73V is outputted as 37.3.

Vcel output is then used in Logical switch L1, which becomes true when cell voltage drops below 3.3V (remember value is multiplied by 10).

Special function is added to play battery low warning when L1 is true.

Script cellv.lua:

```lua
-- cell voltage calculator

local inputs = { {"Bat. volt", SOURCE}, {"Play", VALUE, 0, 1, 0} }
local outputs = { "Vcel" }
local wait_end = 0
local cell_count = 1
local state
local filtered_voltage = 0

--state functions forward declaration
local wait, no_battery, wait_to_stabilize, calc, done

function no_battery()
    -- wait for battery
    if filtered_voltage > 3 then
        state = wait_to_stabilize
        wait_end = getTime() + 200
        --print("wait " .. wait_end )
    end
end

function wait_to_stabilize()
    -- wait some time for battery voltage to stabilize
    if getTime() >= wait_end then
        state = calc
        --print("calc")
    end
end

function calc(play)
    -- calculate cell count
    cell_count = math.ceil(filtered_voltage / 4.25)  --this works up to 12 cells
    print("filtered_voltage: " .. filtered_voltage)
    print("cell count: " .. cell_count)
    -- play detected cell count
    if play > 0 then
        playNumber(cell_count, 0, 0)
        playFile("/SOUNDS/en/celdet.wav") -- wav says: "cell battery detected"
    end
    state = done
    --print("done")
end

function done()
    if filtered_voltage < 2 then
        state = no_battery
        --print("no_battery")
    end
end



local function run(voltage, play)
    filtered_voltage = filtered_voltage * 0.9 +  voltage * 0.1
    --if getTime() % 500 == 0 then print("v: " .. filtered_voltage) end
    if state == nil then state = no_battery end     --state initialization
    state(play)     --call state function
    if cell_count > 0 then
        return (voltage / cell_count) * 102.4
    end
    return 0
end

return { run=run, input=inputs, output=outputs }
```

Appendix

LUA Source List

This is a list of all index numbers that can be used as a SOURCE for getValue. As of 2.0.6 the names are not valid, only the number. This is currently under development so use with caution. Planned to be released in 2.0.8

number	name	description
1	input1	Input [I1]
2	input2	Input [I2]
3	input3	Input [I3]
4	input4	Input [I4]
5	input5	Input [I5]
6	input6	Input [I6]
7	input7	Input [I7]
8	input8	Input [I8]
9	input9	Input [I9]
10	input10	Input [I10]
11	input11	Input [I11]
12	input12	Input [I12]
13	input13	Input [I13]
14	input14	Input [I14]
15	input15	Input [I15]
16	input16	Input [I16]
17	input17	Input [I17]
18	input18	Input [I18]
19	input19	Input [I19]
20	input20	Input [I20]
21	input21	Input [I21]
22	input22	Input [I22]
23	input23	Input [I23]
24	input24	Input [I24]
25	input25	Input [I25]

26	input26	Input [I26]
27	input27	Input [I27]
28	input28	Input [I28]
29	input29	Input [I29]
30	input30	Input [I30]
31	input31	Input [I31]
32	input32	Input [I32]
75	rud	Rudder
76	ele	Elevator
77	thr	Throttle
78	ail	Aileron
79	s1	Potentiometer 1
80	s2	Potentiometer 2
81	s3	Potentiometer 3
82	ls	Left slider
83	rs	Right slider
85	cyc1	Cyclic 1
86	cyc2	Cyclic 2
87	cyc3	Cyclic 3
88	trim-rud	Rudder trim
89	trim-ele	Elevator trim
90	trim-thr	Throttle trim
91	trim-ail	Aileron trim
92	sa	Switch A
93	sb	Switch B
94	sc	Switch C
95	sd	Switch D
96	se	Switch E
97	sf	Switch F
98	sg	Switch G
99	sh	Switch H

100	ls1	Logical switch L1
101	ls2	Logical switch L2
102	ls3	Logical switch L3
103	ls4	Logical switch L4
104	ls5	Logical switch L5
105	ls6	Logical switch L6
106	ls7	Logical switch L7
107	ls8	Logical switch L8
108	ls9	Logical switch L9
109	ls10	Logical switch L10
110	ls11	Logical switch L11
111	ls12	Logical switch L12
112	ls13	Logical switch L13
113	ls14	Logical switch L14
114	ls15	Logical switch L15
115	ls16	Logical switch L16
116	ls17	Logical switch L17
117	ls18	Logical switch L18
118	ls19	Logical switch L19
119	ls20	Logical switch L20
120	ls21	Logical switch L21
121	ls22	Logical switch L22
122	ls23	Logical switch L23
123	ls24	Logical switch L24
124	ls25	Logical switch L25
125	ls26	Logical switch L26
126	ls27	Logical switch L27
127	ls28	Logical switch L28
128	ls29	Logical switch L29
129	ls30	Logical switch L30
130	ls31	Logical switch L31

131	ls32	Logical switch L32
132	trn1	Trainer input 1
133	trn2	Trainer input 2
134	trn3	Trainer input 3
135	trn4	Trainer input 4
136	trn5	Trainer input 5
137	trn6	Trainer input 6
138	trn7	Trainer input 7
139	trn8	Trainer input 8
140	trn9	Trainer input 9
141	trn10	Trainer input 10
142	trn11	Trainer input 11
143	trn12	Trainer input 12
144	trn13	Trainer input 13
145	trn14	Trainer input 14
146	trn15	Trainer input 15
147	trn16	Trainer input 16
148	ch1	Channel CH1
149	ch2	Channel CH2
150	ch3	Channel CH3
151	ch4	Channel CH4
152	ch5	Channel CH5
153	ch6	Channel CH6
154	ch7	Channel CH7
155	ch8	Channel CH8
156	ch9	Channel CH9
157	ch10	Channel CH10
158	ch11	Channel CH11
159	ch12	Channel CH12
160	ch13	Channel CH13
161	ch14	Channel CH14

162	ch15	Channel CH15
163	ch16	Channel CH16
164	ch17	Channel CH17
165	ch18	Channel CH18
166	ch19	Channel CH19
167	ch20	Channel CH20
168	ch21	Channel CH21
169	ch22	Channel CH22
170	ch23	Channel CH23
171	ch24	Channel CH24
172	ch25	Channel CH25
173	ch26	Channel CH26
174	ch27	Channel CH27
175	ch28	Channel CH28
176	ch29	Channel CH29
177	ch30	Channel CH30
178	ch31	Channel CH31
179	ch32	Channel CH32
180	gvar1	Global variable 1
181	gvar2	Global variable 2
182	gvar3	Global variable 3
183	gvar4	Global variable 4
184	gvar5	Global variable 5
185	gvar6	Global variable 6
186	gvar7	Global variable 7
187	gvar8	Global variable 8
188	gvar9	Global variable 9
189	tx-voltage	Transmitter battery voltage [volts]
190	clock	RTC clock [minutes from midnight]
196	timer1	Timer 1 value [seconds]
197	timer2	Timer 2 value [seconds]

198	swr	Transmitter antenna quality [less is better]
200	rssi	RSSI [more is better]
202	a1	A1 analogue value [units as configured]
203	a2	A2 analogue value [units as configured]
204	a3	A3 analogue value [units as configured]
205	a4	A4 analogue value [units as configured]
206	altitude	Variometer altitude [meters]
207	rpm	Rotational speed [revolutions per minute]
208	fuel	Fuel level [???]
209	temp1	Temperature 1 [degrees celsius]
210	temp2	Temperature 2 [degrees celsius]
211	gps-speed	GPS speed [???]
212	distance	GPS distance [meters]
213	gps-altitude	GPS altitude [meters]
214	cell-min	LiPo sensor - lowest current cell voltage [volts]
215	cell-sum	LiPo sensor - current summ of all cell voltages [volts]
216	vfas	Current sensor - voltage [volts]
217	current	Current sensor - current [ampers]
218	consumption	Current sensor - consumption [mili amper hours]
219	power	Current sensor - power [wats]
220	accx	G sensor - acceleration in X axis [g]
221	accy	G sensor - acceleration in Y axis [g]
222	accz	G sensor - acceleration in Z axis [g]
223	heading	GPS heading [degrees]
224	vertical-speed	Variometer vertical speed [m/s]
225	air-speed	Air speed [knots]
226	dte	Total energy [???]
232	a1-min	A1 analogue value minimum [units as configured]
233	a2-min	A2 analogue value minimum [units as configured]
234	a3-min	A3 analogue value minimum [units as configured]
235	a4-min	A4 analogue value minimum [units as configured]

236	altitude-min	Lowest altitude [meters]
237	altitude-max	Highest altitude [meters]
238	rpm-max	Highest rotational speed [revolutions per minute] [meters]
239	temp1-max	Highest temperature 1 [degrees celsius]
240	temp2-max	Highest temperature 2 [degrees celsius]
241	gps-speed-max	Highest GPS speed [???]
242	distance-max	Biggest GPS distance [meters]
243	air-speed-max	Highest air speed [knots]
244	cell-min-min	LiPo sensor - all time lowest cell voltage [volts]
245	cell-sum-min	LiPo sensor - all time lowest summ of all cell voltages [volts]
246	vfas-min	Current sensor - lowest voltage [volts]
247	current-max	Current sensor - highest current [ampers]
248	power-max	Current sensor - highest power [wats]

Character Maps

SMLSIZE

Font file: font_04x06.png for characters below 0xC0
: font_04x06_extra.png for characters above 0xc0

Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char
20		3A	:	54	T	6E	n		
21	!	3B	;	55	U	6F	o		
22	“	3C	<	56	V	70	p		
23	#	3D	=	57	W	71	q		
24	$	3E	>	58	X	72	r		
25	%	3F	?	59	Y	73	s		
26	&	40	º	5A	Z	74	t		
27	‘	41	A	5B	[	75	u		
28	(	42	B	5C	\	76	v		
29	)	43	C	5D	]	77	w		
2A	*	44	D	5E	^	78	x		
2B	+	45	E	5F	_	79	y		
2C	,	46	F	60	~	7A	z		
2D	-	47	G	61	a	7B	{		
2E	.	48	H	62	b	7C	|		
2F	/	49	I	63	c	7D	}		
30	0	4A	J	64	d	7E	→		
31	1	4B	K	65	e	7F	←		
32	2	4C	L	66	f				
22	3	4D	M	67	g				
34	4	4E	N	68	h			C0	↑
35	5	4F	O	69	i			C1	↓
36	6	50	P	6A	j			C2	↗
37	7	51	Q	6B	k			C3	↘
37	8	52	R	6C	l			C4	↙
39	9	53	S	6D	m			C5	↘

default

default character set (no constant so use att = 0)
Font file: font_05x07.png & font_05x07_extra.png

Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char
20		3A	:	54	T	6E	n		
21	!	3B	;	55	U	6F	o		
22	“	3C	<	56	V	70	p		
23	#	3D	=	57	W	71	q		
24	$	3E	>	58	X	72	r		
25	%	3F	?	59	Y	73	s		
26	&	40	º	5A	Z	74	t		
27	‘	41	A	5B	[	75	u		
28	(	42	B	5C	\	76	v		
29	)	43	C	5D	]	77	w		
2A	*	44	D	5E	^	78	x		
2B	+	45	E	5F	_	79	y		
2C	,	46	F	60	~	7A	z		
2D	-	47	G	61	a	7B	{		
2E	.	48	H	62	b	7C	|		
2F	/	49	I	63	c	7D	}		
30	0	4A	J	64	d	7E	→		
31	1	4B	K	65	e	7F	←		
32	2	4C	L	66	f				
22	3	4D	M	67	g			C0	↑
34	4	4E	N	68	h			C1	↓
35	5	4F	O	69	i			C2	↗
36	6	50	P	6A	j			C3	↘
37	7	51	Q	6B	k			C4	↙
37	8	52	R	6C	l			C5	↘
39	9	53	S	6D	m			C6	△

MIDSIZE

Font file: font_08x10.png

Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char
20		3A	:	54	T	6E	n		
21	!	3B	;	55	U	6F	o		
22	“	3C	<	56	V	70	p		
23	#	3D	=	57	W	71	q		
24	$	3E	>	58	X	72	r		
25	%	3F	?	59	Y	73	s		
26	&	40	º	5A	Z	74	t		
27	‘	41	A	5B	[	75	u		
28	(	42	B	5C	\	76	v		
29	)	43	C	5D	]	77	w		
2A	*	44	D	5E	^	78	x		
2B	+	45	E	5F	_	79	y		
2C	,	46	F	60	~	7A	z		
2D	-	47	G	61	a	7B	{		
2E	.	48	H	62	b	7C	|		
2F	/	49	I	63	c	7D	}		
30	0	4A	J	64	d	7E	→		
31	1	4B	K	65	e	7F	←		
32	2	4C	L	66	f				
22	3	4D	M	67	g				
34	4	4E	N	68	h				
35	5	4F	O	69	i				
36	6	50	P	6A	j				
37	7	51	Q	6B	k				
37	8	52	R	6C	l				
39	9	53	S	6D	m				

DBLSIZE

Font file: font_10x14.png

XXLSIZE

Uses font file font_22x38.png

Hex	Character				
2C	,				
2D	-				
2E	.				
2F	_				
30	0				
31	1				
32	2				
33	3				
34	4				
35	5				
36	6				
37	7				
38	8				
39	9				
3A	:				

Links to external documentation:

- OpenTX Web Site
- OpenTX Development Wiki
- Programming in Lua
- Lua 5.2 Reference Manual
- OpenTX University

Document Style

- Code examples are formatted using the Code Pretty Add-on for Google Docs.
- Where possible, each function should have examples with a screen shot.
- Functions and Constants should reference the version it has been added and if applicable, the version it was depreciated.

Acknowledgments