



rt.buffer Reference Manual

Scannex Electronics Limited

2022-07-25

Table of Contents

Introduction	2
Useful Bookmarks	3
IoT Operations	4
Update Mechanism	5
update.txt Keywords	5
User Process	7
Configuration Options	7
Stopping the User Process	7
Lua Modifications	8
Lua Compiler and Virtual Machine changes	8
Lua "Basic Functions"	8
Coroutine Manipulation - coroutine	8
Modules - package	8
String Manipulation - string	8
Table Manipulation - table	9
Mathematical Functions - math	9
Input and Output Facilities - io	9
Operating System Facilities - os	9
Debug Library - debug	10
boot.lua	11
Lua Library Extensions	12
Library adc : ADC Library	13
Library adc adco : ADC Objects	16
ADC Object Table ‘adct’	18
ADC Callback	19
Library alm : Alarm Library	20
Alarm Table ‘at’	24
Alarm Callbacks	24
Library alm ao : Alarm Objects	26
Library app : Application Utilities	30
Memory efficient dynamic blocks	30
Library bit : Logical Bit Functions	32
Library cfg : Configuration	37
Library cfg.key Secret Keys	39
Library coro : Coroutine Extensions	41
Library crc : CRC Calculations	46

Library dpc : Digital Pulse Counters	48
Library evt : Event Library	50
Event Callback Functions	53
Event Callback Names	54
Event Callback Examples	55
Library fc : File Controls	56
Library float : Floating Point Utilities	66
Library float fao : Float Array Object	75
Library float mpco : Multi-Point Conversion Object	82
Library float polyo : Polynomial Object	85
Library float rro : Running Regression Object	86
Library float rso : Running Statistics Object	87
Library gps : GPS/GNSS Library	89
GPS Location gtb	93
GPS Callback	93
Library hash : Hash Calculations	94
Hash Table Structure	96
Library hydrins : HydrINS Support	97
Hydrins Table ht	100
Library iot : Internet Of Things	101
Default Functions	111
IoT Callbacks	113
Library job : Job Scheduler	116
Job Schedule Strings	116
Schedule Examples	117
Scheduled Jobs	117
Scheduled Job Callbacks	119
UTC Based Jobs	120
UTC Based Job Callbacks	122
Library jobq : Job Queue	123
Typical usage	124
Using the global job queue	124
Library jobq jqo : Job Queue Objects	125
Library kvc : Key Value Container	128
Library kvc kvco : Key-Value Container Objects	129
Library modbus : MODBUS functions	131
modbus.cmd tfmt & rfmt Format Strings	139
Library modem : Modem interaction	140

Library nmea : NMEA Functions	144
Library out : Digital Outputs	146
Library pwr : Power Control	149
Development Functions	154
Library rt : rt.buffer Utility Functions	155
Library sdi12 : SDI-12 Protocol	166
SDI-12 Table Structure	171
Library ser : Serial Ports	172
Library ser cp : Comport Objects	175
Receive Until Table rxut	187
Library smp : Sample Data Store	188
Structure of ftable	188
Sample Data Store Functions	190
Prebuilt Function Methods	192
CSV Formatting Table csv	194
CSV Formatting Callback .fmtf	194
Library sms : SMS Text Messaging	196
SMS Received callback	200
Library struct : Structure Handling	201
Struct format strings	203
Struct Examples	204
Library twi : TWI Hardware Interface	205
Alternative to the TWI add parameter for v3.11+	207
Library usb : USB functions	208
Library utc : Time Functions	209
Library util : Utility functions	215
LCD and User Interface Library Extensions	228
Library key : Keyboard Library	229
Key callback functions	232
Library lcd : LCD Library	234
Hardware control	235
Drawing functions	240
Font and Text functions	244
Font names	247
Library ui : User Interface LCD Library	248
Boilerplate code	249
UI Element : Menu	250
UI Element : Values with editing	252

UI Element: Number Editor	253
UI Element: List Picker	255
UI Element: Settings and Hooks	257
Preference settings	263
UI Callbacks	268
Configuration Variables	270
Config c.alm : Alarm	271
Config c.arc : Archive Parameters	272
Config c.cell : Cellular	273
Config c.iot : IoT Parameters	276
Config c.lua : Lua Configuration	280
Config c.site : Site Deployment	281
Config c.term : Terminal Controls	282
Config c.tls : Security Settings	283
Config c.toa : Terminal Over the Air settings	286
Config c.user : User IoT settings	288
Information Variables	289
Info i.app.XXXXX : Application	290
Info i.cell : Cellular Information	291
Modem activity	291
Modem timings	292
Info i.fw : Firmware Information	295
Info i.iot : IoT Information	296
Info i.lua : Lua Information	297
Info i.rt : rt.buffer Information	298
Info i.smp.XXXX : Samples	300
Info i.pwr : Power Information	301
Terminal Commands	303
Terminal: Internal Debug Commands	304
Terminal: External Devices	306
Terminal: General Functions	308
Terminal: Lua Core	309
Terminal: Modem	312
Terminal: NAND Flash	314
BootLoader Terminal Commands	317
LED Sequences	319
Example Lua App	320
The Main Sample	321

Keeping Tally	322
Linking the Schedule into the Configuration	323
Updating the Schedule Immediately	324
Adding Diagnostic Information	325
Adding Pressure Readings	326
Adding ADC Scaling	327
Adding file headers and footers	328
Full Listing	329
Terminal Over the Air (TOA) Details	330
Sequence of Connection	331
CONNECT header	332
CONNECT header with Authentication	335
HTTP POST Protocol	337
Custom Headers	338
Example - Sending Diagnostics	342
rt.buffer sends	342
Server replies	342
Example - Requesting Updates	343
rt.buffer sends	343
Server replies	343
Crypographic Hash Calculations	344
Initialisation	344
Verifying rt.buffer Request	344
Generating Response	345
Software Licenses	346
Operating System	347
Cortex Support Libraries (CMSIS)	348
Atmel Files	349
Lua License	350
Libstruct Library	351
eLua additions	352
zlib Compression Library	353
Cryptographic Functions	354
U8G2 LCD Library	355
Index	357

Data	Author	Release
2017-10-11	MP	For firmware 1.00
2017-10-27	MP	For firmware 1.01
2017-11-10	MP	For firmware 1.02
2017-12-06	MP	For firmware 1.03
2018-01-26	MP	For firmware 1.10
2018-03-08	MP	For firmware 1.11
2018-06-13	MP	For firmware 1.30 (documentation format change)
2018-10-26	MP	For firmware 2.00
2019-03-22	MP	For firmware 3.00
2019-08-19	MP	For firmware 3.10
2020-03-11	MP	For firmware 3.21
2020-05-22	MP	For firmware 3.31
2021-02-23	MP	For firmware 3.40
2021-03-12	MP	For firmware 3.41
2021-04-09	MP	For firmware 3.42
2021-06-21	MP	For firmware 3.43 (and corrected Lua syntax highlighting)
2022-07-25	MP	For firmware 4.00 (and index improvements)

Introduction



See also the "rt.buffer Getting Started Manual".

Lua 5.1.5 has been chosen, along with numerous patches and extensions that allow it to run in a very resource-constrained environment.

Lua in the rt.buffer has about 80k bytes of memory available to it, and this has required using short names for functions and variables (because even though code is compiled into p-code, the tables and functions are still referred to by their string name).

A combination of event-driven (aka "Reactor Pattern") programming and loop programming allows for a flexible combination of ultra-low-power operation, and ease of programming for sequential transmit/receive style operations.

A real-time-operating system (RTOS) provides multi-tasking support, and Lua is called from three different tasks in the RTOS.

1. [Main Lua Loop](#) Task
2. [Event Callback](#) Task
3. [Modem Cellular Delivery](#) Task



Think of the 'Event Callback` Task as an "interrupt handler" in traditional coding.

It is important to be aware that the Event Callback Task must NEVER perform long-running operations. If you try, they will be aborted. Instead, callbacks issued from within this task should signal to the Main Lua Loop Task (or otherwise queue up requests in a Lua table).

There are also callbacks that occur within the Modem/3G Delivery Task—and these can take a reasonable time (minutes). However, the task can still choose to abort the running Lua callback.

Most of the system architecture has been designed to be completely flexible. Consequently, the process of designing a suitable Lua-based App may take some time and skill to be successful.

This document aims to provide a reference for the many Lua library extensions that have been written to complement the Lua core.

Useful Bookmarks

These links may be helpful:

- USB and Terminal
 - [Terminal Commands](#)
 - [BootLoader Terminal Commands](#)
- [c.](#) and [i.](#) values
 - [Configuration Variables](#)
 - [Information Variables](#)
- Scheduling
 - [Job Schedule Strings](#)
 - [Schedule Examples](#)
- [LED Sequences](#)

IoT Operations

"Internet-of-Things" methods.

Primarily the IoT core provides services to push collected data and alerts to your central servers. Additionally, the the rt.buffer can be managed remotely through the [Update Mechanism](#) - allowing firmware upgrades, diagnostic gathering, etc, to be handled centrally and remotely.

A [User Process](#) can also be glued in, so that special actions can occur on a specific event (like the magnet triggering).

This section outlines some key details for using these services.

See also:

- [Configuration Variables](#):
 - [c.cell](#) - for connection details and modem timings
 - [c.iot](#) - for server details
 - [c.user](#) - for the [User Process](#) settings
- [Lua Library Extensions](#)
 - [iot.*](#) - for IoT actions
 - [gps.time](#) - for the GPS service timing setting
 - [sms.rxt](#) - for the SMS service timing setting

Update Mechanism

The update mechanism works by pulling an `update.txt` file from the IoT server, and parsing the lines of the ASCII file.

When completed, the `update.txt` file will be renamed on the server as `update.YYYYYMMDDHHMMSS.txt` (using the `rt.buffer`'s UTC time that the update was completed).

The commands and available options are listed below.

update.txt Keywords



If the `c.iot_gz=0`, then the following transfers will not use gzip compression, and will not use the `.gz` extension.

cellinfo

Perform a cellular survey and post the results to the IoT server.

The file will be named `cellinfo.{i.rt_sn}.txt.gz`

diag

Post a diagnostics dump file onto the IoT server.

The diag file will be named `diag.{i.rt_sn}.txt.gz`

COMMAND

COMMAND:Lua `expression`

Execute a Lua command. This is syntactically the same as the USB command `lx`. However, the results of the Lua command will be thrown away.

Complex expressions can be executed, with a line length up to 128 characters.

```
lx job.utc(os.time()+6000, function() iot.go('user') end)
```

If an App has utility functions, for example to make configuration changes to the connected device, you can call these too. They will be executed from the context of the modem task.

```
lx CallMyFunction(10.34, 34.56)
```

getfile SOURCE,TARGET

SOURCE: The full path and name of the file on the rt.buffer

TARGET: The (path and) filename of where to save the file on the IoT server

Post a file from the flash file system onto the IoT server.

Listing 1. gz will be added to the target filename, if appropriate.

```
getfile /Logs/system.log,/temp/system.log
```

setfile SOURCE,TARGET

SOURCE: Source filename to ask from the IoT server

TARGET: The full path of where to save the file on the rt.buffer

Pull a file from the IoT server and save into the flash file system.



The source file cannot be gzip or zlib compressed. The rt.buffer does not have the RAM to decompress data.

```
setfile /temp/config.mine.txt,/Config/config.txt
```



If updating the `/Config/config.txt` file be **very** careful you do not change the IoT settings, or break the cellular settings... otherwise you could lose control of the rt.buffer!

NAME.blf

NAME: Filename (and path) on the IoT server to pull

Pulls a firmware image from the IoT server and burns.

```
../rt.buffer.Lua.1.02.blf
```

c.KEY = VALUE

c.KEY: Config parameter

VALUE: Config value

Make a configuration change

User Process

The user process is typically triggered by the magnet.

The rt.buffer will connect to the IoT server and repeatedly send the config and info sets. This should enable a back-end process to present the data to a live web-page, so any engineer can view the current information on a phone, tablet or PC.

The process will pause, by default 15s, and send again. This will continue for a default time of 10 minutes. The settings `c.user_ins` & `c.user_onm` override the values.

Configuration Options

The process will post the data to a file named `"user.{i.rt_sn}.txt.gz"` in the folder specified by `c.iot_url`
`c.iot_user`.

If the `c.iot_user` value is not present, then `"Update/{c.site_name}"` is used (same logic as the `c.iot_upd` value).

The function `"doUser"` will be called on each loop, allowing the Lua App to include additional values (like live ADC readings) in the information tree.

Stopping the User Process

The back-end process can abort the user mechanism by leaving a file named: `"user.{i.rt_sn}.stop"`

When the rt.buffer detects this file, it deletes it and quits the User Process.

Lua Modifications

This section documents the additions to the standard Lua core that provide the functionality required for the {rpbuffer} product.

Lua Compiler and Virtual Machine changes

There are some basic internal changes to the Lua core, including:

1. Reading from a table that doesn't exist returns NIL, and does not generate a run-time error. Attempting to write to a NIL table will still generate a run-time error.
 - `lvm.c` modified
2. C/C++ style comments are supported with the `//` tag and `/.../` tag. This is in addition to the regular Lua comment marker `--`.
 - `llex.c:llex` function modified
3. Strings can use the C/C++ hex markers. e.g. `'\xee\xff'` is 0xee + 0xff.
 - `llex.c:read_string` modified
4. The os functions `execute`, `exit`, `getenv`, `setlocale`, and `tmpname` removed.

Lua "Basic Functions"

Fully included.

Coroutine Manipulation - `coroutine`

Fully included

Modules - `package`

NOT included. The memory requirements were too large.

However, you can still use Lua's `loadfile` and `loadstring`.

String Manipulation - `string`

Fully included.

Table Manipulation - `table`

Fully included.

Mathematical Functions - `math`

Fully included, using double-precision floating point internally.

Input and Output Facilities - `io`

Mostly included.

Not included

- `io.popen`
- `io.tmpfile`
- `file:setvbuf`

Operating System Facilities - `os`

Mostly included.

Not included

- `os.execute`
- `os.getenv`
- `os.tmpname`

Added

```
ok = os.move(fromfile, destfile)
```

Move a file

- Parameters**
- `fromfile`: Filename of existing file
 - `destfile`: Filename of target

- Returns**
- `ok`: True if successful

Description Move a file



See the more flexible `fc.move` function (which supports wildcard operations).

Debug Library - debug

NOT included.

boot.lua

If the file `/Lua/boot.lua` exists, the rt.buffer will execute this file before loading the application Lua file.



Be careful what you code you put in this file!

Lua Library Extensions

There is a large set of Lua library extensions embedded in the firmware. The libraries and their functions are constantly being improved and added to.

Many of the libraries have default behaviour that can be overridden using tables, variables, and function overrides.

Library `adc` : ADC Library

This library provides a coroutine-friendly way to take multiple ADC measurements, and average them.

`adc.def()` v3.00

Set ADC default parameters

Parameters None

Returns None

Description Sets the default parameters of 1 sample and no 'off' measurement.

`done,n,tot = adc.done()` v3.00

Check for complete

Parameters None

Returns

- `done`: Whether complete (boolean)
- `n`: Number of samples taken
- `tot`: Total number of samples to take

Description Checks whether the ADC sequence has completed.

You can create a loop if needed, or use the `onAdc` callback:

```
while not adc.done()
do coro.w(100)
end
```

```
uv,ts,ouv = adc.get([m, [c, [raw]]]) v3.00
```

Get ADC sample

Parameters

- **m**: Slope for the ADC value
- **c**: Offset for the ADC value
- **raw**: Whether to use raw ADC values (true), or microvolts (false=default)

Returns

- **uv**: Microvolts.
Range ± 0 -1500000uV (± 0 to 1.5V)
(3.0V is the reference voltage)
- **ts**: Timestamp of the first sample
- **ouv**: Microvolts when the power was off.

Description

Get average pressure reading in microvolts, optionally calculating $y=mx+c$



hardware iss2/iss3 ADC is LTC2485, 24-bit delta-sigma, that has V-REF = 3.0V.



hardware iss4 is TI ADS122C04/ADS112C04, V-REF = 3.0V.

```
adc.go() v3.00
```

Start ADC

Parameters

None

Returns

- **ok**: Whether sampling started

Description

Starts the ADC measurement process.

Each sample takes 170ms, so an 8-sample average will take 1.36s to complete.

```
adco = adc.obj(n, [adct]) v4.00
```

Get ADC object

- Parameters**
- **n**: Slot number (0-7)
 - **adct**: (optional) [configuration table](#)

- Returns**
- **adco**: ADC object

Description Provides access to an ADC object for measurement and control.

```
n,moff = adc.set([n], [moff]) v3.00
```

Set or get ADC parameters

- Parameters**
- **n**: Number of samples to average (default=1)
 - **moff**: True to measure the 'off' reading as well

- Returns**
- **n**: Number of samples to average
 - **moff**: True to measure the 'off' reading as well

Description Set or get the ADC parameters.

When triggered, the ADC task will read **n** samples. If **moff** is true, then before enabling the ADC power an 'off' reading will be taken.

Library `adc` | `adco` : ADC Objects

These functions work on the object returned from `adc.obj()`.

For the sake of documentation, these are shown to refer to 'adco'

```
adct,ave,std,ts,working,ndone = adco:get() v4.00
```

Get ADC Object settings

Parameters None

Returns

- `adct`: ADC [table](#) of settings
- `ave`: mean value
- `std`: standard deviation
- `ts`: timestamp of first sample
- `working`: whether still sampling
- `ndone`: number of samples completed

Description Returns the readings and values.

```
adco:go() v4.00
```

Start ADC Object conversion

Parameters None

Returns None

Description Starts the sampling process.

```
prevstate = adco:pwr([docontrol]) v4.00
```

Get/set ADC Object power control

- Parameters**
- `docontrol`: whether to control the reference power
- Returns**
- `prevstate`: the setting before updating the setting
- Description** Returns (or modifies) whether to control the reference power during this ADC measurement.

```
adco:set(adct) v4.00
```

Set ADC Object settings

- Parameters**
- `adct`: ADC [table](#) of settings
- Returns** None
- Description** Update or change the settings for the ADC object.

```
ave,var,std,skew,kurt,num = adco:stat() v4.00
```

Get ADC Object Statistics

- Parameters** None
- Returns**
- `ave`: mean value
 - `var`: variance
 - `std`: standard deviation
 - `skew`: skewness
 - `kurt`: kurtosis
 - `num`: number of data points
- Description** Returns the detailed statistics.

ADC Object Table 'adct'

adct v4.00

ADC Table Settings

Parameters

- [adct.vp](#): Positive channel (0-3)
- [adct.vn](#): Negative channel (0-3). If not provided then uses single-ended measurement on [.vp](#)
- [adct.g](#): Gain (1,2,4,8,16,32,64,128). Default=2
- [adct.n](#): Number of samples to take
- [adct.t](#): Interval (milliseconds) between samples
- [adct.m](#): Slope for conversion
- [adct.c](#): Offset for conversion
- [adct.raw](#): (boolean) = whether to read the ADC raw value
- [adct.pwr](#): (boolean) = whether to measure with power applied
- [adct.bcs](#): (boolean) = whether to use Burnout Current Sources. NOTE: This affects measurement accuracy, but if the sensor is not connected or shorted it will go to full-scale.

Description

As used by [adc.set](#) and returned by [adc.get](#)

ADC Callback

```
function onAdc() v3.00  
function onAdc(slot) v4.00
```

ADC completion callback

Parameters • `slot`: The ADC slot that completed

Returns None

Description Called when the ADC completes all readings.



You will need to enable the callback event with `evt.en('onAdc')`

Library `alm` : Alarm Library

The alarm library allows for tracking counts on 32 distinct alarms. The alarms can be updated with bitfields, or individually.

```
alm.clr(n)
alm.clr( {bitfield} )
```

Alarm clear

- Parameters**
- `n`: Alarm number, 1-32
 - `bitfield`: The bitfield number is placed in the first element of a table

Returns None

Description Clear the Counts.

```
aa, at = alm.get(n)
ab, at = alm.get( {bitfield} )
```

Alarm get

- Parameters**
- `n`: Alarm number, 1-32
 - `bitfield`: The bitfield number is placed in the first element of a table

- Returns**
- `aa`: Alarm active (true if active)
 - `ab`: Alarm bits that are active
 - `at`: Alarm table of values

Description Returns a table with the alarm parameters and counts for each alarm specified.

```
local ab, at = alm.get( {0x3} )
-- ab = 1
-- at = {1={co=2,cf=0,lo=2,lf=1,alm=true}, 2={co=0,cf=0,lo=2,lf=1} }
```

```
ao = alm.new() v3.43
```

Create alarm object

Parameters None

Returns • [ao](#): Alarm object

Description Return a new alarm object.



Alarm objects are Lua tables with metamethods for method functions.

```
alm.set(n, limiton, limitoff)
alm.set( {bitfield}, limiton, limitoff)
alm.set(n, limiton, limitoff, valueon, valueoff) v3.42
```

Alarm set

Parameters

- [n](#): Alarm number, 1-32
- [limiton](#): Number of sequential 'ons' before alarmed
- [limitoff](#): Number of sequential 'offs' before inactive.
- [bitfield](#): The bitfield number is placed in the first element of a table
- [valueon](#): The floating point value for the alarmed state. $value \geq valueon$
- [valueoff](#): The floating point value for the idle state. $value < valueoff$

Returns None

Description Set the parameters for an individual alarm 'n', or a range of alarms '{bitfield}'

When using [valueon](#) and [valueoff](#) the code will handle both positive and negative slopes (in which case it becomes $value \Leftarrow valueon$ and $value > valueoff$).

```
alm.set( {0xff}, 10, 1) -- set alarms 1-8
alm.set( 10, 2, 2) -- set alarm 2 only
```

```
txt=alm.txt(n)
```

Alarm get text

Parameters

- **n**: alarm number

Returns

- **txt**: text

Description Return the text for the alarm, as set in `c.alm_txt##`

e.g. `alm.txt(2)` will return the value from `c.alm_txt02`

```
alm.upd(n, ison)
alm.upd( {bitfield, bitvalues} )
alm.upd(n, value) v3.42
```

Alarm update

Parameters

- **n**: Alarm number, 1-32
- **ison**: Boolean indicating whether on or off
- **bitfield**: The bitfield number is placed in the first element of a table
- **bitvalues**: A bitfield representation of the alarms covered by bitfield, where 0=off, 1=on
- **value**: Floating point value to compare. See [alm.set](#)

Returns None

Description Update the alarm counters, and trigger the alarm callbacks as required.

```
alm.upd(2, true)
alm.upd( {0xff, 0x5a} )
```

```
alr,hys = alm.val(n) v3.43
```

Alarm value

- Parameters**
- **n**: Alarm number, 1-32
- Returns**
- **alr**: true if active/alerted; false if inactive
 - **hys**: true if co>0; false if cf>0; nil if in hysteresis zone
- Description** Returns state details about the specified alarm.

Alarm Table ‘at’

at

Alarm Table

- Parameters**
- `at.co`: Counts on
 - `at.cf`: Counts off
 - `at.lo`: Limit on
 - `at.lf`: Limit off
 - `at.alm`: true if in the alarmed state
 - `at.vo`: Value on ^{v3.42}
 - `at.vf`: Value off ^{v3.42}

Description As returned by `alm.get(...)`

Alarm Callbacks

There are two types of callbacks made from the library:

`function onAlm##(state)`

Alarm individual callback function

- Parameters**
- `state`: true if alarmed, false if inactive

Returns None expected

Description Callback for an individual alarm, where `##` is the two-digit decimal number of the alarm:

```
function onAlm02(state)
  if state
  then -- do something
  else -- do something else
  end
end
```

function onAlm(n, state)

Alarm global callback function

Parameters

- **n**: Alarm number, 1-32
- **state**: true if alarmed, false if inactive

Returns

None expected

Description

A callback for all alarms. For each alarm, the `onAlm##` is called first, followed by the `onAlm` callback (allowing a mix of specific and general handling).



You can use the function `alm.txt(n)` within the callback to pick out a name for the alarm.

Library `alm` | `ao` : Alarm Objects

The alarm objects are Lua tables that contain the alarm state and settings.

They have the same field names as the Alarm Table ‘at’ but include some additional fields that hold functions for callback and compare.

- `ao.lo` = Limit on
- `ao.lf` = Limit off
- `ao.co` = Counts on
- `ao.cf` = Counts off
- `ao.vo` = Value on
- `ao.vf` = Value off
- `ao.v` = last number value passed to `ao:upd`
- `ao.alm` = true if in the alarmed state; false unalarmed; nil initial state
- `ao.fc` = function callback
- `ao.cmp` = compare function



Added in firmware 3.43

`ao:clr()`

Alarm Object clear

Parameters None

Returns None

Description Clear the Counts. i.e. `ao.co=0` & `ao.cf=0`


```
ao:set(limiton, limitoff, [valueon, [valueoff]], [fc])
```

Set Alarm Object values

- Parameters**
- **limiton**: Number of sequential 'ons' before alarmed
 - **limitoff**: Number of sequential 'offs' before inactive.
 - **valueon**: The floating point value for the alarmed state. value >= valueon
 - **valueoff**: The floating point value for the idle state. value < valueoff
 - **fc**: Function callback

Returns None

Description Sets the parameters for the alarm.

This is really just a syntactic helper to give compatibility to the original `alm.set` method.

The table fields of `ao` will be assigned the various values passed as parameters.



If a parameter is `nil` then the table field assignment is skipped.

```
sc,alm = ao:upd(ison)
sc,alm = ao:upd(value)
```

Alarm Object update

- Parameters**
- **ison**: Boolean indicating whether on or off
 - **value**: Floating point value to compare. See `ao:set`

- Returns**
- **sc**: State Change > +1=alarmed, 0=nochange, -1=unalarmed
 - **alm**: Alarm state > true=alarmed, false=unalarmed

Description Update the alarm counters, and trigger the alarm callback (`ao.fc`) as required.

```
alr,hys = ao:val()
```

Alarm Object value

Parameters None

Returns • `alr`: true if active/alerted; false if inactive
 • `hys`: true if `ao.co`>0; false if `ao.cf`>0; nil if in hysteresis zone

Description Returns state details about the alarm object.

```
ao.fc = function(ao, st)
```

Alarm Object state callback

Parameters • `ao`: The alarm object is passed to the function
 • `st`: The state (boolean)

Returns None

Description The function `ao.fc` is called in the context of `ao:upd` when the alarm state changes.

```
function mycb(ao, state)
  if state
  then -- do something urgent
  end
end

local ao = alm.new()
...
-- various ways to set ao.fc:
ao:set(5, 10, mycb)           -- Boolean type alarm
ao:set(5, 10, 0.5, 0.3, mycb) -- Value type alarm
ao.fc = mycb                  -- Manual assignment
```

ao.cmp = function(ao, v)

Alarm Object compare function

- Parameters**
- **ao**: The alarm object is passed to the function
 - **v**: The value (number)
- Returns**
- **ac**: Alarm change > +1 increments ao.co; -1 increments ao.cf; 0 no change
- Description** The function **ao.cmp** is called when using **ao:upd(NUMBER)** to decide how to adjust the alarm counters.

The function should return +1/0/-1 to consider the alarm 'on'/'nochange'/'off'.

```
-- Emulates the default behaviour in pure Lua:
function mycmp(ao, v)
  if ao.v0 > ao.vf
  then
    if v >= ao.v0
    then return 1
    elseif v < ao.vf
    then return -1
    else return 0
    end
  else
    if v <= ao.v0
    then return 1
    elseif v > ao.vf
    then return -1
    else return 0
    end
  end
end

local ao = alm.new()
...
ao.cmp = mycmp -- Override default firmware compare
```

Library `app` : Application Utilities

The `app` library provides some helpful functions related to the Lua app.

Memory efficient dynamic blocks

You can add special commented markers within the Lua app file, and dynamically retrieve these at runtime.

The tags are in the form:

```
--[[TAG]text--]]--

--[[TAG]
multi-line text
is here
--]]--
```

```
rv,... = app.eval(tag) v2.00
```

App evaluate block

Parameters

- `tag`: Tag name

Returns

- `rv`: Lua return values...

Description

Find the `tag` and evaluates, equivalent to `return app.txt(tag)`

```
rv,... = app.exec(tag) v2.00
```

App execute block

Parameters

- `tag`: Tag name

Returns

- `rv`: Lua return values...

Description Find the `tag` and execute, like Lua's `dostring`.



Useful for dynamic function definitions.

```
tb = app.lines(tag) v2.00
```

App fetch lines

Parameters

- `tag`: Tag name

Returns

- `tb`: Table of strings

Description Find the `tag` and return as a table of strings.



Useful for internationalisation strings.

```
txt = app.txt(tag) v2.00
```

App fetch text block

Parameters

- `tag`: Tag name

Returns

- `txt`: Text block, if found.

Description Find the `tag` and return as a raw string.

Library `bit` : Logical Bit Functions

The bit module provides useful bit operation functions.



Bit functions by Reuben Thomas <rrt@sc3d.org>

<http://luaforge.net/projects/bitlib>

(Extensions for bit-wise string handling and nibble/byte swap added by Scannex.)

```
v = bit.al(a,b) v3.44
v = bit.alshift(a, b) v3.44
```

Bit arithmetic shift right

Parameters

- `a`: Value to shift (signed 32-bit)
- `b`: Number of bits to shift

Returns

- `v`: Result

Description Returns `a` left shifted arithmetically `b` places.

```
v = bit.ar(a,b)
v = bit.arshift(a, b)
```

Bit arithmetic shift right

Parameters

- `a`: Value to shift (signed 32-bit)
- `b`: Number of bits to shift

Returns

- `v`: Result

Description Returns `a` right shifted arithmetically `b` places.

```
v = bit.band(v1, ...)
v = bit.band(txt)
```

Bitwise AND

Parameters

- `v1`: Value 1, etc
- `txt`: String

Returns

- `v`: Result

Description Returns the bitwise AND of the `v's, or string

```
v = bit.bnot(a)
```

Bitwise NOT

Parameters

- `a`: Value

Returns

- `v`: Result

Description Returns one's complement of `a`.

```
v = bit.bor(w1, ...)
v = bit.bor(string)
```

Bitwise OR

Parameters

- `v1`: Value 1, etc
- `txt`: String

Returns

- `v`: Result

Description Returns the bitwise OR of the `v's, or string

```
v = bit.bxor(w1, ...)
v = bit.bxor(string)
```

Bitwise XOR

Parameters

- `v1`: Value 1, etc
- `txt`: String

Returns

- `v`: Result

Description Returns the bitwise XOR of the ``v`'s, or string

```
v = bit.cast(a)
```

Bit cast

Parameters

- `a`: Value

Returns

- `v`: Result

Description Cast `a` to the internally used integer type.

```
v = bit.ls(a, b)
v = bit.lshift(a, b)
```

Bitwise left shift

Parameters

- `a`: Value to shift (unsigned 32-bit)
- `b`: Number of bits to shift

Returns

- `v`: Result

Description Returns `a` left shifted `b` places.


```
v = bit.rs(a, b)
v = bit.rshift(a, b)
```

Bitwise right shift

Parameters

- **a**: Value to shift (unsigned 32-bit)
- **b**: Number of bits to shift

Returns

- **v**: Result

Description Returns **a** right shifted **b** places.

```
v = bit.swap16(a)
```

Bit swap bytes

Parameters

- **a**: Value

Returns

- **v**: Result

Description Returns **a** with the LSB and MSB bytes exchanged.

```
v = bit.swap32(a)
```

Bit swap

Parameters

- **a**: Value

Returns

- **v**: Result

Description Returns **a** with each of the four bytes reversed.

```
v = bit.swap8(a)
```

Bit swap nibbles

Parameters

- **a**: Value

Returns

- **v**: Result

Description

Returns **a** with the nibbles exchanged.

Library `cfg` : Configuration

The `cfg` Library provides helpful routines for reading and writing values into the configuration tree.

```
v = cfg.get(key, [t], [d]) v1.00
```

Config_get

Parameters

- `key`: Config key name
- `t`: (optional) table for replacing the # characters in key
- `d`: (optional) default value

Returns

- `v`: Result

Description

Reads a value from the config tree.

The '`c.`' prefix is not required, and will be ignored if supplied.

If `key` contains hash (#) characters then each hash is replaced with a value from the table `t`.

If `d` is supplied, and the config tree does not have the key named, then `v=d`.

`v` will be NIL, a string, or a number (i.e. its natural type).

```
local t = cfg.get('adc_c', 0.001)
local t = cfg.get('d_#_#', {1, 'm'}) -- returns value of c.d_1_m
```

```
v = cfg.getb(key, [t], [d]) v1.00
```

Config get boolean

Description As `cfg.get`, but converts to boolean (or nil).

```
v = cfg.getn(key, [t], [d]) v1.00
```

Config get number

Description As `cfg.get`, but converts to number (or nil).

```
v = cfg.gets(key, [t], [d]) v1.00
```

Config get string

Description As `cfg.get`, but converts to string (or nil).

```
ok = cfg.set(key, [t], v) v1.00
```

Config set

Parameters

- `key`: Config key name
- `t`: (optional) table for replacing the # characters in key
- `v`: Value to set

Returns

- `ok`: True if succeeded

Description Sets `c.key=v`

If `key` contains hash (#) characters, then each hash is replaced with a value from the table `t`.

```
cfg.set('d_#_#', {1, 'm'}, 'Testing')
cfg.set('adc_c', 1.234)
```

```
v = cfg.table(key, [t, [d]]) v1.00
```

Config read as table

Parameters

- **key**: Config key name
- **t**: (optional) table for replacing the # characters in key
- **d**: (optional) default table

Returns

- **v**: Resulting table, or NIL

Description

Reads a config sub-tree as a Lua table.

If **key** contains hash (#) characters, then each hash is replaced with a value from the table **t**.

If **d** is supplied, then this forms the starting point of the **v** result. The table **d** is "deep-copied" to **v**, and then any values within the config sub-tree will replace the entries within **v** - effectively combining the two.

```
local t = cfg.table('') -- gets the whole tree
local t = cfg.table('adc', {}, {m=1,c=0})
```

Library **cfg.key** Secret Keys

The **cfg.key** library provides control over write-only keys that are used for cryptographic functions.

Keys are stored in a hidden directory **/_keys**. However, the keys themselves are encrypted with a device-specific key. The files cannot be used in any other device.



Added in firmware v3.00

ok = cfg.key.chk(kn) v3.00

Check a secret key

- Parameters**
- **kn**: Key name
- Returns**
- **ok**: True if a key has been set
- Description** Checks existence of a key.

cfg.key.set(kn, txt) v3.00

Set a secret key

- Parameters**
- **kn**: Key name
 - **txt**: Plaintext secret
- Returns** None
- Description** Saves the key.

cfg.key.wipe(kn) v3.00

Erase a secret key

- Parameters**
- **kn**: Key name
- Returns** None
- Description** Erases the key.

Key names

- **iot** : The IoT secret for http locking
- **toa** : The Terminal Over-the-Air shared secret

Library `coro` : Coroutine Extensions

Lua includes a mechanism to run 'coroutines' - also known as cooperative multi-tasking. The `rt.buffer` firmware extends that library to make it simple to run scheduled tasks.

Any of the `onXXX` callback functions and the UI functions absolutely require that code runs very quickly. The `coro` library provides a very convenient way to queue longer running tasks and execute them in the context of the Lua Loop task.

Up to 16 coroutine functions can be queued at any one time.

Depending on how the functions are written, they will execute strictly sequentially, or can cooperatively multitask (if you use the `coro.w()` or `coro.yld()` functions).



Added in firmware v2.00



Lua library functions that delay or wait, such as the serial routines or `rt.ms`, will block **all** coroutines from running during the delay. Once the Lua library function returns then it is possible to call `coro.w` or `coro.yld` to allow other coroutines to run.

```
ok = coro.add([dup], func, [user])
ok = coro.add([id], func, [user])
```

Coro add function to queue

Parameters

- `dup`: True to allow multiple instances of `func`
- `id`: Identifier for the function
- `func`: Lua function
- `user`: User value (string, number, or table) that is passed to `func`

Returns

- `ok`: True if added to the queue

Description

Adds a Lua function, using the supplied id (or the function itself).

By default, the function will not be queued again if it is already pending or running. e.g. `coro.add(MyFuncName)` will only allow a single run of `MyFuncName` at a time.

The `id` is useful to associate the same function with different activities. So, the id could be a string, or a number. You could use the id for a process name, e.g. `smpl`, or `prog` - so that one function can be queued for each particular activity.

The `user` value can be used to queue up parameters for the function. So, for example, the function may reconfigure a connected device, and the `user` parameter may be a table of values that are used

to reconfigure. This approach allows a single function to perform variable actions.

```
v = coro.id()
```

Coro get ID

Parameters None

Returns • **v**: RTOS task and Lua coroutine identifier

Description Helpful function that can differentiate different Lua tasks and RTOS tasks.

The locking functions rely on this id value.

```
n = coro.kill()
n = coro.kill(idfn)
```

Coro kill routine

Parameters • **idfn**: The specific **idfn** of the coroutine to kill. (function or ID name as passed to [coro.add](#))

Returns • **n**: Number of coroutines that were terminated

Description Abort all coroutines, or the specified routine.

```
n = coro.n()
n = coro.n(fn) v3.00
```

Coro number of routines running

Parameters • **fn**: Function

Returns • **n**: Number of queued coroutines

Description Return the number of coroutines currently queued.

If **fn** is defined, then **n** is the number of iterations of **fn** that are still running.


```
coro.yld()
coro.w(ms)
```

Coro yield or wait

Parameters • `ms`: Milliseconds to wait

Returns None

Description Yields the current coroutine, allowing the next coroutine task to execute immediately. If `ms` is provided, the current coroutine will not resume until the delay has expired.

If there are no other pending coroutines, the `coro.yld` will return immediately and execution will continue.



The actual delay may be *longer* than the `ms` value.



`coro.yld` and `coro.w` are identical.



v3.00 allows `coro.w` to be called from the main Lua context (e.g. the terminal). In this case it will execute `rt.ms` automatically.



Don't use `coro.w` inside a function referenced by `rt.call`. Use the `coro.except` instead.

`r = coro.wu(ti, [ft], func)` ^{v3.31}

Coro wait until

Parameters

- `ti`: Millisecond interval to wait
- `ft`: Future time (as returned by `rt.msf`)
- `func`: Function to call each loop

Returns

- `r`: Not false means quit the loop

Description

Provides a simple way of waiting until something happens in a coro-friendly way.

If the `func` returns non-null & non-zero then the `coro.wu` returns the result, allowing you to quit a simple `while..do..end` loop.

```
-- 500ms intervals, checking received data
while not coro.wu(500, function() return dp:rxw() > 10 end) do end

-- Max 30s with 200ms intervals, calling MyTest
function MyTest(ti,ft)
  -- (ti and ft are passed into the function)
  if dp:rxw() > 10 then return 'yes' end
  return false
end

local ft = rt.msf(30000)
while not coro.wu(200, ft, MyTest) do end
```

```
local onErr = coro.except(func, v)
local onKill = coro.except(func, v)
local onErr, onKill = coro.except(func, v) v3.31
```

Coro exception object

- Parameters**
- `func`: Lua function to call
 - `v`: Parameter to pass
- Returns**
- `onErr`: Exception for error handling
 - `onKill`: Exception for `coro.kill` on this coroutine
- Description** Creates a special Lua exception handler that **must** be saved local.

The exception handler gives your code a chance to tidy up on exceptions.

If there is a Lua error, the `onErr` will be called after the firmware error handling. If `coro.kill` causes this routine to abort, then `onKill` will be called.



`onKill` is only called as a result of `coro.kill`. For normal exit of a coroutine you must manually call.



Version 3.31 returns two copies of the handler, allowing a single line assignment to both local variables.

```
function OnMyExcept(func, user, cause)
    -- Cleanup
    -- cause == 1 means error
    -- cause == 2 means killed
end

function MyCoro(v)
    local onErr, onKill = coro.except(OnMyExcept, v)
    -- do stuff
end

coro.add(MyCoro)
```

Library `crc` : CRC Calculations

CRC calculations for CRC-8 (c8), CRC-16 (c16), CRC-32 (c32), and CRC-Xmodem (xm), and custom CRC values.

```
cv = crc.c8(txt) / crc.c16(txt) / crc.c32(txt) / crc.xm(txt)
```

CRC calculate

Parameters

- `txt`: String to CRC

Returns

- `cv`: CRC value

Description Perform CRC-8, CRC-16, CRC-32, or CRC-Xmodem on a string.

Short-hand for using `crc.new` / `crc.upd` / `crc.fin`

```
local v = crc.c32('Testing')
```

```
ct = crc.new(size)
ct = crc.new(txt)
ct = crc.new(ci, ref, polyrev, xor)
```

CRC new object

Parameters

- `size`: Number of bits = 8, 16, 32
- `txt`: Name of crc, e.g. 'xmodem', 'crc32'
- `ci`: Initial CRC value
- `ref`: Boolean value. True to reflect the bits
- `polyrev`: Bit-reversed polynomial
- `xor`: Final XOR value

Returns

- `ct`: CRC table structure

Description Initialise a new CRC object table.

```
--[[ Sample CRC Custom Values
CRC-32: crc.new(0xffffffff, false, 0xedb88320, 0xffffffff)
CRC-16/Modbus: crc.new(0xffff, false, 0xa001, 0)
CRC-16/SDI-12: crc.new(0, false, 0xa001, 0)
CRC-16: crc.new(0xffff, true, 0x8408, 0)
CRC-16/Xmodem: crc.new(0, true, 0x8408, 0)
CRC-16/Kermit: crc.new(0, false, 0x8408, 0)
CRC-16/DNP: crc.new(0, false, 0xa6bc, 0xffff)
]]--
```

cv = ct:fin()

CRC finalise

Parameters None

Returns • [cv](#): CRC value

Description Finish CRC and return CRC value.

```
local c = crc.new('xmodem')
c:upd('Testing')
local v = c:fin()
```

ct:upd(txt) ct:upd(fhandle, [length])

CRC update

Parameters • [txt](#): String to CRC
 • [fhandle](#): File handle
 • [length](#): Optional number of bytes (otherwise does all)

Returns None

Description Update CRC.

```
local c = crc.new(32)
c:upd('Testing')
```

Library `dpc` : Digital Pulse Counters

```
pc1,pc2 = dpc.dis()
```

Digital pulse disable

Parameters None

Returns

- `pc1`: Pulse count 1 before disabled
- `pc2`: Pulse count 2 before disabled

Description Disable pulse counter input.

```
l1, l2 = dpc.lvl() v1.00  
l = dpc.lvl(n) v3.00
```

Digital pulse current level

Parameters

- `n`: Which input - 1 or 2

Returns

- `l1`: True if Pulse 1 is at 0V/GND
- `l2`: True if Pulse 2 is at 0V/GND
- `l`: Level (when n>0)

Description Return the current level of the digital pulse inputs.

```
pc1,pc2 = dpc.rst()
pc = dpc.rst(n) v3.00
```

Digital pulse reset

- Parameters**
- **n**: Which input - 1 or 2
- Returns**
- **pc1**: Pulse count 1 before reset
 - **pc2**: Pulse count 2 before reset
 - **pc**: Pulse count (when n>0)
- Description** Reset digital pulse counts (and enables).

```
pc1,pc2 = dpc.val()
pc = dpc.val(n) v3.00
```

Digital pulse get value

- Parameters**
- **n**: Which input - 1 or 2
- Returns**
- **pc1**: Pulse count 1
 - **pc2**: Pulse count 2
 - **pc**: Pulse count (when n>0)
- Description** Query the pulse count values.

Library `evt` : Event Library

```
evt.en(evts)
evt.di(evts)
```

Event enable or disable

Parameters • `evts`: Event name strings

Returns None

Description Enable or disable event callbacks.

A group of the events are always enabled, and cannot be disabled.

```
evt.en('onDevRx,onPL1')
```

```
evt.init(evts) v3.10.0171
```

Event initialise

Parameters • `evts`: Event name strings (blank = all)

Returns None

Description Fires the events if they're active. Useful to kick off an event on Lua script start, etc.

Applies to `onPL2`, `onPL1`, `onDevCon`, `onEngCon`, `onDevCts`, `onEngCts`, and `onMag` only

evt.lsig()

Event signal

Parameters	None
Returns	None
Description	Signal the event loop.

It is important that the main Lua Loop Task is asleep for most of its life. The `evt.lsig()` can be called from an event callback function to wake up a sleeping loop process.



If you use the `coro` library, do NOT use `evt.lwait` or `evt.lsig` (they are used by the library internally)

ok = evt.lwait([timeout])

Event wait

Parameters	• <code>timeout</code>: Number of milliseconds to wait. If NIL, then waits forever. • <code>ok</code>: True if signalled. False if timed-out
-------------------	---

Description	Wait for the signal from the event callbacks.
--------------------	---

It is important for power-consumption that the main Lua Loop Task is asleep for most of its life. The `evt.lsig()` can be called from an event callback function to wake up a sleeping loop process that has called `evt.lwait()`



If you use the `coro` library, do NOT use `evt.lwait` or `evt.lsig` (they are used by the library internally)

evt.sig(evts)

Event signal trigger

Parameters

- `evts`: Event name strings

Returns

None

Description

Trigger a callback (for testing and development).

Event Callback Functions

Event callbacks are run in a separate "Lua Events Task" inside the rt.buffer. To make use of a callback, you must declare a function within the App, and enable the named function with `evt.en(...)`

The callbacks are passed two or three parameters:

```
function eventcallbackname(name, idx)
function eventcallbackname(name, idx, value)
```

Event callback function

Parameters

- **name**: Event name, e.g. "onSec"
- **idx**: Event index number, 1-32
- **value**: Optional value relating to the callback

Returns

None expected

Event Callback Names

- For time-based callback events, use the job library.
- Always enabled
 - `onMag` - magnet is closed (default is `iot._onMag`)
 - `onMagOff` - magnet is released
 - `onConfig` - configuration change occurred
 - `onEngCon` - engineer port is connected or disconnected
 - `onDevCon` - device port is connected or disconnected
 - `onUnload` - called when the script is being unloaded ^{v1.00}
- Require enabling
 - `onEngCts` - Engineer CTS line changed
 - `onDevCts` - Device CTS line changed
 - `onEngRx` - data is being received
 - `onDevRx` - data is being received
 - `onPL1` - digital pulse input 1 is low
 - `onPL2` - digital pulse input 2 is low
 - `onPH1` - digital pulse input 1 is high
 - `onPH2` - digital pulse input 2 is high



Note that some of the events (like the digital pulse events) will dramatically increase power consumption if used with high-frequency devices.

i.e. do not enable the pulse events unless they will occur relatively infrequently (e.g. door closure contacts, etc)

Event Callback Examples

```
function onMag()  
  -- Magnet triggered!  
  iot._onMag() -- call the original default  
  -- Do some other things  
end  
  
function onDevCon(n,idx,v)  
  if ser.dev():okr()  
  then  
    -- connected!  
  else  
    -- disconnected :-(  
  end  
end  
  
evt.en('onMag,onDevCon') -- enable these two (onMag already on)
```

Library `fc` : File Controls

```
name = fc.add(filename, ext1, [ext2...])
```

File add extension

Parameters

- `filename`: Base filename
- `ext1`: Extension to add
- `ext2`: ...More extensions (if required)

Returns

- `name`: Full filename

Description

Add extension(s) to filename (dots not required)

```
nb = fc.append(fhandle, filename, [txt])
```

File append contents

Parameters

- `fhandle`: File handle
- `filename`: Filename of existing file to append to fhandle
- `txt`: optional string to add after the file is appended

Returns

- `nb`: Number of bytes added to fhandle

Description

Writes the contents of filename to the filehandle (at the current position).

```
attr = fc.attr(fname) v4.00
attr,was = fc.attr(fname, change) v4.00
```

File attributes

- Parameters**
- **fname**: Filename
 - **change**: Change attribute string
 - **attr**: Attribute string
 - **was**: Previous attribute string

Description Query or change file attributes.

- **A** = archive
- **a** = not archive
- **H** = hidden
- **h** = not hidden

```
nb = fc.cache([blocks, [mode]]) v1.00-2.xx
```

File set cache

- Parameters**
- **blocks**: Number of blocks (2112byte) cache to assign
 - **mode**: Cache mode string:
 - 'r' = Read,
 - 'w' = Write-back,
 - 't' = Write-Thru

Returns • **nb**: Number of bytes allocated to the cache

Description Sets up the file system cache.



'w' is the mode usually required - where data is lazy written to the volume.

The default is a single block of write-back cache to speed up writing.

i.e. `fc.cache(1, 'w')`



Memory is taken from the Lua memory area.

fc.chkupd() v1.00

File check update

Parameters None**Returns** None**Description** Triggers a check on the config.txt and Lua App files.**fc.clean([blocks, [sectors]])** v1.00

File clean flash

Parameters

- **blocks**: =2 (optional) Number of flash blocks to clean
- **sectors**: =0 (optional) Number of sectors

Returns None**Description** Prepare the flash file system for faster writing. The call does internal housekeeping to free up the requested number of blocks/sectors.

This is helpful when you expect a large file write to take place that must occur without stopping.

```
fc.clean(2, 0) -- make space for about 200k
```



```
ok = fc.copy(fname, tname) v1.00
ok,count,total = fc.copy(fwildcard, twildcard) v3.20
```

File copy

- Parameters**
- **fname**: Source file name
 - **fwildcard**: Source file using `*` and/or `?` wildcards
 - **tname**: Target file name
 - **twildcard**: Target directory or `wildcarded path - the` is replaced by each found filename.
- Returns**
- **ok**: True if copied
 - **count**: Number of files copied successfully
 - **total**: Number of files found.

Description Copies a file or files.

```
fc,fs = fc.del([compare], dirpattern, [isdir], [UTCfrom, [UTCto]], [func])
```

File delete

- Parameters** compare: dirpattern: isdir: UTCfrom: UTCto:
- **func**: Same structure as `fc.find`
- Returns**
- **fc**: filecount
 - **fs**: filesize (if isdir==false)

Description Deletes selected files from the directory.

```
ext,name = fc.ext(filename)
```

File split extension

- Parameters**
- `filename`: Full filename string
- Returns**
- `ext`: Extension (without the dot)
 - `name`: Filename
- Description** Remove first extension from filename (without the dot).

```
n = fc.find([compare], dirpattern, [isdir], [UTCfrom, [UTCto]], [func])
```

File find

- Parameters**
- `compare`: How to compare the files `'='` first file that matches (default) `'=='` last file that matches `'!'` first file that does not match `'!!!'` last file that does not match
 - `dirpattern`: Filename and pattern, e.g. `'/Send/*.txt'`
 - `isdir`: boolean. Set to true to find directories rather than files. (Files are default) UTCfrom:
 - `UTCto`: Also adds file's modified time to comparison, such that:
 $\text{UTCfrom} \leq \text{filemtime} < \text{UTCto}$
 - `func`: optional Lua function that's called for each found file (to allow for moving, table creation, etc)
- Returns**
- `n`: Filename of found file
- Description** Finds the file or directory in directory that conforms to the wildcard pattern.

```
-- template for fc.find's func parameter:
function fc_find_func(filename, dir, creation, modified, filesize, attribs)
    -- do something, e.g. populate table etc
end

-- e.g.
t = {}
fc.find('/Lua/*e*.lua',
    function(fn,dir,ct,mt,fs,attr)
        t[fn] = fs
    end) -- Collects file sizes into table
```

```
sc,so = fc.gz(fname, [tname], [del]) v3.43
so,sc = fc.gz(fname, [tname], [del], [decomp]) v4.00
```

File gzip

- Parameters**
- **fname**: Source file name
 - **tname**: Target file name (optional)
 - **del**: Delete source (optional, default=false)
 - **dcomp**: Decompress file. If **fname** ends with **.gz** then this is implied. ^{v4.00}

- Returns**
- **sc**: Size of compressed file
 - **so**: Size of original file

Description Compresses a file with gzip.



Adds **.gz** to the target filename.



It is not possible to decompress inside the rt.buffer - it does not have enough RAM.

```
ok = fc.js() v3.00
```

Journalling supported

Parameters None

- Returns**
- **ok**: True if journalling supported

Description Checks whether the internal file system supports journalling.

```
ok,res... = fc.jx(fn, ...) v3.00
```

Execute with `journalling`

- Parameters**
- `fn`: Function to call
 - `...`: Parameter list
- Returns**
- `ok`: True if ok
 - `res`: Results (depending on `fn` return values)
- Description** Calls a function `fn` with `journalling` begin/end.



This is an advanced function to support special cases. Normally, the default operation will be perfectly fine.

```
ok = fc.lv(fhandle, luavnames, [txt])
```

File write Lua value tree

- Parameters**
- `fhandle`: File handle
 - `luavnames`: Lua value names
 - `txt`: optional string to append after the list
- Returns**
- `ok`: True if evaluated some Lua values
- Description** Outputs the Lua value trees to the filehandle.

(Same syntax as the USB `LV` command)

If `txt` is assigned, also writes this afterwards.

```
fc.lv(fh, 'c,i', '\r\n')
```

```
fn = fc.mdf(dir, name)
```

File join dir and file

- Parameters**
- **dir**: Directory path
 - **name**: Filename
- Returns**
- **fn**: Full directory + filename
- Description** Join directory + filename.

```
ok,rc = fc.mkd(dir)
```

File make directory

- Parameters**
- **dir**: Directory to create
- Returns**
- **ok**: True if success
 - **rc**: Internal result code
- Description** Create a directory, and all sublevels if required.

```
ok = fc.move(fname, tname) v3.20  
ok,count,total = fc.move(fwildcard, twildcard) v3.20
```

File move

- Parameters**
- **fname**: Source file name
 - **fwildcard**: Source file using `\*` and/or `?` wildcards
 - **tname**: Target file name
 - **twildcard**: Target directory or `\*` wildcarded path - the `\*` is replaced by each found filename.
- Returns**
- **ok**: True if copied
 - **count**: Number of files copied successfully
 - **total**: Number of files found.
- Description** Moves a file or files.

```
ok,rc = fc.rmd(dir)
```

File remove directory

Parameters

- `dir`: Directory to remove

Returns

- `ok`: True if success
- `rc`: Internal result code

Description Remove a directory and all contents (can take a long time to run if there are lots of contents).

```
dir,name = fc.sdf(filename)
```

File split dir and file

Parameters

- `filename`: Full filename

Returns

- `dir`: Directory path
- `name`: Just filename and extensions

Description Separate directory and filename.

```
fc.sfwm(safe) v1.00-2.xx  
fc.sfwm(mode)
```

File set file write mode

Parameters

- `safe`: Boolean.
true = safe and slow
false = medium
- `mode`: Mode string. 'fast', 'med', 'safe'

Returns None

Description Scannex debug facility to set the file volume's file write mode.

The default mode is 'safe'.



Do not use this unless necessary! Using non-safe file system could corrupt data!

fc.sync() v1.00

File sync to flash

Parameters None**Returns** None**Description** Scannex debug facility to synchronise and flush the NAND file system.**utc = fc.utc(filename)**

File get UTC creation

Parameters • **filename**: Filename of existing file**Returns** • **utc**: Integer UTC file creation time.**Description** Returns the creation date of the filename or directory.

```
local s = utc.txt( fc.utc('/Config/config.txt') )
```

ok = fc.wc(filename, pattern)

File wildcard compare

Parameters • **filename**: Filename to compare against
 • **pattern**: Wildcard pattern**Returns** • **ok**: true if filename matches the pattern**Description** Does a wildcard compare.

Library `float` : Floating Point Utilities

The float library provides conversion between Lua's numbers and IEEE-754 binary representations. Both single precision (32-bit) and double precision (64-bit) numbers are supported, in integer and ASCII formats.



Be aware that converting from single to double, and vice-versa, can introduce apparent 'errors'. There are helpful articles on the Internet that highlight the technical reasons why this happens. Lua is double-precision (64-bit) internally.

```
fao = float.array(size) v4.00
```

Create new float array object

Parameters

- `size`: Number of elements in the array

Returns

- `fao`: [Float Array Object](#)

Description Create a fix-size float array for statistics and FFT/DWT style operations.

```
iv = float.d2v(fp)
```

Double to integer

Parameters

- `fp`: Floating point that needs converting

Returns

- `iv`: Integer 64-bit IEEE-754


```
ah = float.d2x(fp)
```

Double to hex

- Parameters**
- `fp`: Floating point that needs converting
- Returns**
- `ah`: ASCII-Hex representation of 64-bit IEEE-754
- Description** Convert from Lua number to 64-bit float

```
fp = float.dp(fv, tv) v3.42
```

Trim float value

- Parameters**
- `fv`: Floating point that needs trimming
 - `tv`: Trim value. e.g. 0.001 trims to 3dp
- Returns**
- `fp`: Floating point result.
- Description** Trims and rounds a floating point number.

```
local v = float.dp(123.456, 0.01) -- > 123.46
local v = float.dp(123.456, 10)   -- > 120
```

```
iv = float.f2v(fv)
```

Float to integer value

- Parameters**
- `fv`: Floating point that needs converting
- Returns**
- `iv`: Integer 32-bit IEEE-754
- Description** Convert from Lua number to 32-bit integer

```
ah = float.f2x(fv)
```

Float to hex

- Parameters**
- `fv`: Floating point that needs converting
- Returns**
- `ah`: ASCII-Hex representation of 32-bit IEEE-754
- Description** Convert from Lua number to 32-bit ASCII-hex float

```
ok = float.isfin(fv) v3.42
```

Check if finite

- Parameters**
- `fv`: Value to check
- Returns**
- `ok`: True if finite
- Description** Compare with C library `isfinite`

```
ok = float.isinf(fv) v3.42
```

Check if infinite

- Parameters**
- `fv`: Value to check
- Returns**
- `ok`: True if infinite
- Description** Compare with C library `isinf`

```
ok = float.isnan(fv) v3.42
```

Check if a number

Parameters

- **fv**: Value to check

Returns

- **ok**: True if NaN

Description Compare with C library [isnan](#)

```
ok = float.isnor(fv) v3.42
```

Check if a normal number

Parameters

- **fv**: Value to check

Returns

- **ok**: True if a normal number

Description Compare with C library [isnormal](#)

```
ave,var,std,skw,kur,count = float.mean(v,[tbl]...) v3.42
```

Get running statistics values

Parameters

- **v**: Value
- **tbl**: Table of values

Returns

- **ave**: Mean average
- **var**: Variance
- **std**: Standard Deviation
- **skw**: Skewness
- **kur**: Kurtosis
- **count**: Number of sample

Description Calculates mean, variance, etc from a set of values and/or table-of-values.

```
mpco = float.mpc(tx, ty) v2.00  
mpco = float.mpc(tx, ty, [uv], [ov]) v3.41
```

Create multi-point-conversion object

- Parameters**
- **tx**: Table of x-values
 - **ty**: Table of y-values
 - **uv**: Under value (y when x is too low)
 - **ov**: Over value (y when x is too high)

- Returns**
- **mpco**: [Multi-Point Conversion Object](#)

Description Creates a new MPC object that can be called to perform a conversion.

See: [\[mpco:y\]](#)



Stores the **tx** and **ty** in a memory-efficient C structure in the Lua RAM area.



Use [float.rf](#) or [float.rd](#) to load raw IEEE values from a file.

```
polyo = float.poly(m0,m1,m2...) v2.00
```

Calculate n-order polynomial object

- Parameters**
- **m0,m1,m2...**: polynomial multipliers

- Returns**
- **polyo**: [Polynomial Object](#)

Description Creates a new polynomial object that can be called to perform a conversion. You can use any number of values.

e.g. `float.poly(5, 0.5, 0.1) = 5 + 0.5x + 0.1x^2`

See: [\[polyo:y\]](#)



Stores the values in a memory-efficient C structure in the Lua RAM area.

```
t = float.rf(fh, n) v2.00
t = float.rd(fh, n) v2.00
```

Read floats/doubles from file

- Parameters**
- **fh**: File handle
 - **n**: Number of values to read

- Returns**
- **t**: Table of floating numbers

Description Reads raw IEEE single float 4-byte (**rf**) or double float 8-byte (**rd**) into a table.

```
rro = float.regr() v3.42
```

Create a running regression object

Parameters None

- Returns**
- **rro**: Running Regression Object

Description Creates an object that can calculate slope, intercept, and correlation on X,Y.

See: https://www.johndcook.com/blog/running_regression/

```
local mo = float.regr()
local t = {5,8,4,5}
mo:push(2,5)
mo:push(6,1)
mo:push(t)
print(mo:get()) -- -0.54,7.06,-0.32,4
```

Because many methods return the object itself you can chain calls:

```
local mo = float.regr()
local t = {5,8,4,5}
print(mo:push(2,5):push(6,1):push(t):get()) -- -0.54,7.06,-0.32,4
```

rso = float.stat() v3.42

Create a running statistics object

Parameters None**Returns** • **rso**: Running Statistics Object**Description** Creates an object that can calculate mean, variance, standard-deviation, skewness, and Kurtosis.See: https://www.johndcook.com/blog/skewness_kurtosis/

```

local mo = float.stat()
local t = {5,8,4,5}
mo:push(2,5,6)
mo:push(1)
mo:push(t)
print(mo:get()) -- 4.25,5.07,2.25,0.15,-0.79,8

```

Because many methods return the object itself you can chain calls:

```

local mo = float.stat()
local t = {5,8,4,5}
print(mo:push(2,5,6):push(1):push(t):get()) -- 4.25,5.07,2.25,0.15,-0.79,8

```

```
fp = float.v2d(iv)
```

Integer to double

Parameters

- **iv**: Integer 64-bit IEEE-754 value

Returns

- **fp**: Floating point value

```
fp = float.v2f(iv)
```

Integer value to float

Parameters

- **iv**: Integer 32-bit IEEE-754 value

Returns

- **fp**: Floating point value

Description Convert from 32-bit integer to Lua number

```
n = float.wf(fh, t) v2.00
```

```
n = float.wd(fh, t) v2.00
```

Write floats/doubles to file

Parameters

- **fh**: File handle
- **t**: Table of numbers

Returns

- **n**: Count of values written

Description Writes raw IEEE single float 4-byte (**wf**) or double float 8-byte (**wd**) into a file.

```
fp = float.x2d(ah)
```

Hex to double

- Parameters**
- [ah](#): ASCII-Hex 64-bit string
- Returns**
- [fp](#): Floating point value
- Description** Convert from 64-bit ASCII-hex float to Lua number

```
fp = float.x2f(ah)
```

Hex to float

- Parameters**
- [ah](#): ASCII-Hex 32-bit string
- Returns**
- [fp](#): Floating point value
- Description** Convert from 32-bit ASCII-hex float to Lua number

Library float | fao : Float Array Object

```
sv = fao:add(fav) v4.00
sv = fao:sub(fav) v4.00
sv = fao:mul(fav) v4.00
sv = fao:div(fav) v4.00
```

add/sub/mul/div values

Parameters • fav: Float Array or Value

Returns • sv: Sum of all entries in fao

Description Modified the fao entries with add/sub/mul/div, by either referencing another float-array object, or a single value.

e.g.

```
local sv = fa1:sub(fa2) -- fa1[idx] = fa1[idx] - fa2[idx]
local sv = fa1:sub(4.56) -- fa1[idx] = fa1[idx] - 4.56
```



Modifies the values in the array

```
fao:clr() v4.00
```

Clear

Parameters None

Returns None

Description Clears all entries.

```
sv = fao:copy(fav) v4.00
```

Copy values

Parameters

- **fav**: Float Array or Value

Returns

- **sv**: Sum of all entries in **fao**

Description Copies the entries from the float array **fav**, or set to the value **fav**.

```
sv = fao:ft(wa, [wo], [wi]) v4.00  
svt = fao:ft(wa, [wo], [wit]) v4.00
```

Fourier Transform (FFT/DFT)

Parameters

- **wa**: Fourier wave array object
- **wo**: (optional) offset into the wave array (default=0). **true** = 1/4 wave (for cosine)
- **wi**: (optional) increment through wave array (default=1.0)
- **wit**: (optional) table of increments. Negative values will use cosine, positive will use sine.

Returns

- **sv**: Sum value
- **svt**: Table of results

Description Calculate a Fourier style multiply-and-sum with the wave array being repeated as the values of **fao** are iterated.



This is a building block of FFT, not the whole process. It allows calculation of certain harmonics, or of compound wave arrays (where the harmonics have been pre-summed).

Using the **wit** table of increments allows DFT calculations in one call.

e.g.

```
local wa = float.array(256)  
wa:sin()  
local svt = fao:ft(wa, nil, {1,-1,2,-2,4,-4,8,-8})
```

Calculates sin(x), cos(x), sin(2x), cos(2x), etc

```
v = fao:get(idx) v4.00
```

Get an array entry

Parameters

- `idx`: Index of the entry to get (1..count)

Returns

- `v`: Value

Description Retrieve a value from the float array.

Returns an out of bounds error if `idx` is not within 1..count.



`idx=1` is the oldest entry



Using `v = fao[n]` will wrap `n` to the number of entries. (i.e. a modulus index access)

```
n = fao:push(v) v4.00
```

Push value

Parameters

- `v`: Value to push

Returns

- `n`: Number of entries in `fao`

Description Push a value into the float array.

If the array is already full, the oldest value is shuffled off to make space.

```
v = fao:set(idx, v) v4.00
```

Update an array entry

Parameters

- **idx**: Index of the entry to set (1..count)
- **v**: Value to set

Returns

- **v**: Original value

Description Update a value in the float array.

Returns an out of bounds error if **idx** is not within 1..count.



idx=1 is the oldest entry

```
fao:sin() v4.00
```

Fill with sin values

Parameters None

Returns None

Description Fill the array with sin values (a single cycle).

(Cosine can be achieved by a 1/4 offset into the sin array.)

```
n = fao:size() v4.00
```

Get array size

Parameters None

Returns • **n**: Size of the float array

Description Returns the total size of the float-array.



To find the current count, use `n = #fao`

```
sv = fao:sqrt() v4.00
```

Square root entries

Parameters None

Returns • **sv**: Sum of all the updated entries in the float array

Description Update each of the entries so that `v = sqrt(v)`.



Modifies the values in the array

```
rso = fao:stat([idx],[n],[rso]) v4.00
```

Calculate stats

Parameters • **idx**: (optional) index to start (default=1, modulus)
 • **n**: (optional) count of values (default #fao)
 • **rso**: (optional) Running Stats Object to feed values into

Returns • **rso**: The results

Description Push the values from `fao` into either an existing, or a new, running stats object.



Use `?fa:stat():get()` to return the values.

```
sv = fao:sum() v4.00
```

Sum all values

Parameters None

Returns • **sv**: Sum of all entries in the float array

Description Add all entries in the array

```
t = fao:tab() v4.00
```

Convert to table

Parameters None

Returns None

Description Return the array as a table.

```
sv = fao:wt(wa, [dao], [wi]) v4.00
```

Wavelet Transform (DWT/CWT)

Parameters • **wa**: Wavelet array object

- **dao**: (optional) offset from the end of **fao** data. (default = `#fao/2`).
- **wi**: (optional) wavelet increment (default=1.0)

Returns • **sv**: Sum value

Description Calculate a wavelet style multiply-and-sum with a single run of the wave array. The optional **wi** parameter allows stretching or compressing of a single wavelet array object across the data values of **fao**.

The **wa** wave array should be a wavelet (e.g. Mexican hat, Haar, Daubechies, etc)

```
sv = fao:xs([dao],[dai],ra,[ro],[ri],n) v4.00
```

Multiply-and-sum two arrays

Parameters

- [dao](#): (optional) Data array offset (default=0, modulus)
- [dai](#): (optional) Data array increment (default=1.0)
- [ra](#): Reference Array object
- [ro](#): (optional) offset into referebce (default=0)
- [ri](#): (optional) increment through reference data (default=1.0)
- [n](#): (optional) count of values to process (default #fao)

Returns

- [sv](#): Sum value

Description

Iterates through the [fao object](#) and multiplies by the values in [ra](#).

This is the underlying calculation used for both [fao:ft](#) and [fao:wt](#).

Library `float` | `mpco` : Multi-Point Conversion Object

`mpco:oob(uv,ov)` ^{v3.41}

Set out-of-bound values

- Parameters**
- `mpco`: Multi-Point-Conversion Object (returned from [`float.mpc`])
 - `uv`: Under value
 - `ov`: Over value

Returns None

Description Sets the under and over values.

`count,uv,xf,yf,xl,yl,ov = mpco:range()` ^{v3.41}

Get range

- Parameters**
- `mpco`: Multi-Point-Conversion Object (returned from [`float.mpc`])

- Returns**
- `count`: Number of x,y pairs in the object
 - `uv`: Under value
 - `xf`: First x-value
 - `yf`: First y-value
 - `xl`: Last x-value
 - `yl`: Last y-value
 - `ov`: Over value

Description Returns some key details about the `mpco` object.


```
x,y = mpco:xy(index) v3.41
```

Get x,y pair

- Parameters**
- `mpco`: Multi-Point-Conversion Object (returned from [`float.mpc`])
 - `index`: Index of x,y set (1-based)

- Returns**
- `x`: x value
 - `y`: y value

Description Returns a single x,y pair from the set.



If `index < 1` then `x = xf-1` and `y=uv`



If `index > count` then `x = xl + 1` and `y=ov`

```
xt,yt,uv,ov = mpco:xyt() v3.41
```

Get x and y tables

- Parameters**
- `mpco`: Multi-Point-Conversion Object (returned from [`float.mpc`])

- Returns**
- `xt`: Table of x-values
 - `yt`: Table of y-values
 - `uv`: Under value
 - `ov`: Over value

Description Returns the two tables that were used to create the object.

yv,idx = mpc:y(xv) v2.00

Convert from x to y

Parameters

- **mpco**: Multi-Point-Conversion Object (returned from `[float.mpc]`)
- **xv**: X-value to convert

Returns

- **yv**: Y-value conversion
- **idx**: Index (effectively insertion point of xv)

Description

Finds the pair of X-values that **xv** is within, and performs linear slope conversion to find **yv**.



Returns nil if **xv** is out-of-bounds, or the **uv** or **ov** values (if set in `mpco:oob`)



Uses the **tx** and **ty** tables passed to `[float.mpc]`.

```
local m = float.mpc( {10,20,30}, {100,200,300} )
local y = m:y(15)  -- > 150
local y = m:y(1)   -- > nil
local y = m:y(301) -- > nil
```

Library `float` | `polyo` : Polynomial Object

```
m = polyo:c(idx) v3.41
```

Return coefficient

- Parameters**
- `polyo`: Polynomial-Conversion Object (returned from `[float.poly]`)
 - `idx`: Index of coefficient, ≥ 1
- Returns**
- `m`: coefficient value
- Description** Returns the coefficient.

```
m0,m1,m2,m3... = polyo:terms() v3.41
```

Return terms

- Parameters**
- `polyo`: Polynomial-Conversion Object (returned from `[float.poly]`)
- Returns**
- `m0...`: coefficient values
- Description** Returns the coefficients as a set.

```
yv = polyo:y(xv) v2.00
```

Convert x to y

- Parameters**
- `polyo`: Polynomial-Conversion Object (returned from `[float.poly]`)
 - `xv`: X-value to convert
- Returns**
- `yv`: Y-value conversion
- Description** Performs the n-order polynomial conversion of `xv`

```
local p = float.poly(5, 1, 0.5, 0.01) -- 5 + x + 0.5x^2 + 0.01x^3
local y = p:y(123) -- > 26301.17
```

Library float | rro : Running Regression Object

```
rro = rro:add(rro2) v3.42
```

Add running regression objects

Parameters • `rro2`: Running regression object

Returns • `rro`: Running Regression Object

Description Effectively pushes the information from `rro2` to the `rro` object.

```
rro = rro:clear() v3.42
```

Clear running regression object

Parameters None

Returns • `rro`: Running Regression Object

Description Clears the regression object.

```
rro = rro:push(x,y, [x,y]...) v3.42  
rro = rro:push(tbl, ...) v3.42
```

Push values into running regression object

Parameters • `x,y`: Values for x & y
 • `tbl`: Table of values

Returns • `rro`: Running Regression Object

Description Updates the `rro` object with X,Y pairs.

Can include a mixture of tables and value as parameters.

Library float | `rso` : Running Statistics Object

```
rso = rso:add(rso2) v3.42
```

Add running stats objects

Parameters • `rso2`: Running stats object

Returns • `rso`: Running Statistics Object

Description Effectively pushes the information from `rso2` to the `rso` object.

```
rso = rso:clear() v3.42
```

Clear running stats object

Parameters None

Returns • `rso`: Running Statistics Object

Description Clears the statistics.

```
ave,var,std,skw,kur,count = rso:get() v3.42
```

Get running statistics values

Parameters None

Returns • `ave`: Mean average
 • `var`: Variance
 • `std`: Standard Deviation
 • `skw`: Skewness
 • `kur`: Kurtosis
 • `count`: Number of sample

Description Pulls the current set of statistics from the running object.

```
slo,itr,cor,count = rso:get() v3.42
```

Get running regression values

Parameters None

Returns

- `slo`: Slope
- `itr`: Intercept
- `cor`: Correlation
- `count`: Number of sample

Description Pulls the current set of statistics from the running regression object.

```
rso = rso:push(v, v...) v3.42  
rso = rso:push(tbl, ...) v3.42
```

Push values into running stats object

Parameters

- `v`: Value
- `tbl`: Table of values

Returns

- `rso`: Running Statistics Object

Description Updates the `rso` object.

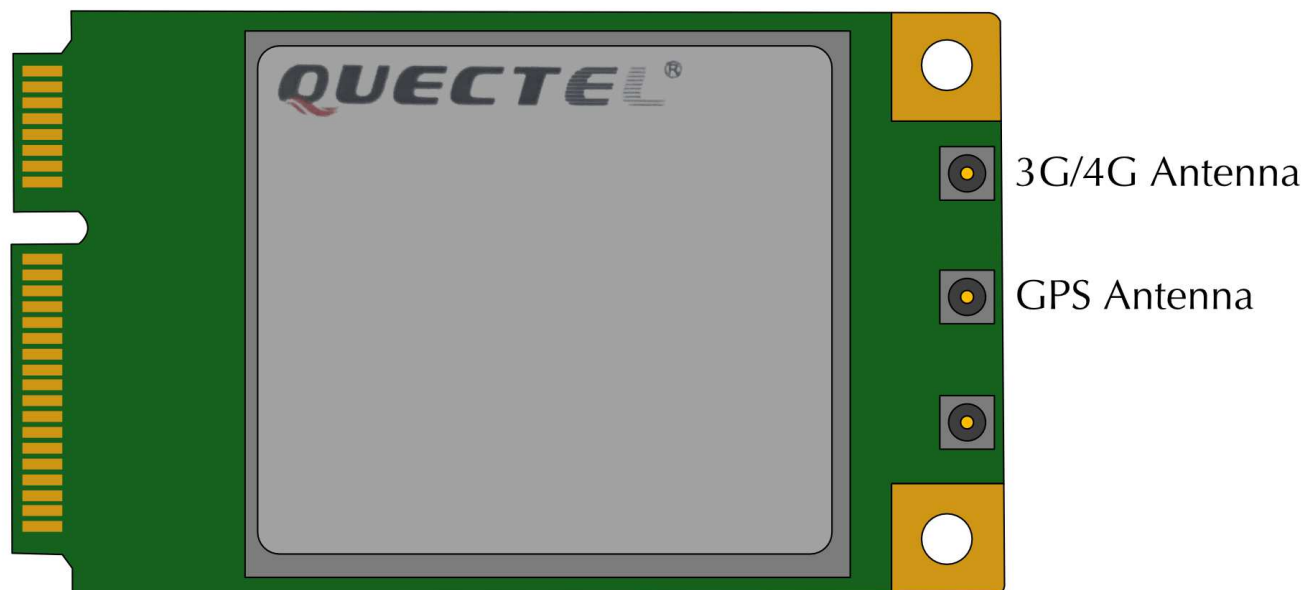
Can include a mixture of tables and value as parameters.

Library `gps` : GPS/GNSS Library

This library provides an interface to the GPS/GNSS location hardware in the Quectel module modem.



You must connect a suitable GPS/GNSS/GLONASS antenna to the modem.



```
en,st = gps.act() v3.10
```

Get GPS activity

Parameters None

Returns

- `en`: True if enabled
- `st`: State = 0:idle, 1:waiting, 2:querying

Description Get the GPS status.

```
d = gps.dist([d]) v3.10
```

Set/Get GPS fix distance

Parameters • **d**: Fix distance, in meters (default: 50)

Returns • **d**: Fix distance in meters

Description Sets or gets the fix distance.

```
t = gps.fixt([t]) v3.10
```

Set/Get GPS fix time

Parameters • **t**: GPS fix maximum time, in seconds (default: 30)

Returns • **t**: GPS fix maximum time.

Description Set or get the GPS fix maximum time.

```
t = gps.gap([t]) v3.10
```

Set/Get GPS interval

Parameters • **t**: Interval time, in seconds (default: 15)

Returns • **t**: Interval time, in seconds

Description Set or get the interval time between GPS location queries.

gps.go() v3.10

Start GPS

Parameters	None
Returns	None
Description	Starts the GPS modem process.

Equivalent to `iot.go('gps cho')`

gtb = gps.loc() v3.10

Get GPS location

Parameters	None
Returns	• <code>gtb</code>: Table of values
Description	Get the last GPS location data.

lc = gps.rst() v3.10

Reset the GPS counter

Parameters	None
Returns	• <code>lc</code>: Last count
Description	Reset the count, so the next value of <code>gtb.idx</code> will be 1.

```
t = gps.time([t]) v3.10
```

Set/Get overall GPS duration

Parameters

- **t**: Time, in seconds (default: 120)

Returns

- **t**: Time, in seconds

Description Set or get the overall time the GPS module should be alive.



A value of -1 will indicate to keep the GPS running all the time the modem is connected.



It usually takes about 45 seconds to obtain a first GPS location.

GPS Location *gtb*

As returned by `alm.get(...)`

- `.idx` = Index of the reading (see `gps.rst()`)
- `.utc` = UTC sent by GPS satellite
- `.alv` = `rt.buffer` alive time of this reading
- `.sat` = Number of satellites
- `.lon` = Longitude
- `.lat` = Latitude
- `.alt` = Altitude (in meters)
- `.hdop` = Horizontal dilution of precision (lower numbers better)
- `.cog` = Ground heading based on true north (ddd.mm)
- `.spkm` = Speed over ground (km/hr)

GPS Callback

```
function onGPS() v3.10
```

GPS location available callback

Parameters None

Returns None

Description Called when a GPS location has arrived.



The event is always enabled.

Library `hash` : Hash Calculations

Performs MD5, SHA1, SHA256, and HMAC-SHA256 hash functions of text or file contents.

```
hv = hash.md5(txt)
hv = hash.sha1(txt)
hv = hash.sha256(txt)
hv = hash.hmac256(txt, secret) v3.00
hv = hash.hmac256(txt, secret, keyname) v3.31
```

Hash string immediately

- Parameters**
- `txt`: String to hash
 - `secret`: Secret string for HMAC-SHA-256 ^{v3.00}
 - `keyname`: Name of the key that stored `cfg.key.set`

Description Calculate hash of direct string.



The HMAC-SHA-256 function will use concatenate the `secret` text with the value of the `keyname`.

```
ht = hash.new(method, secret, [keyname])
```

Create new hash object

- Parameters**
- `method`: Which hash = `'sha1'`, `'sha224'`, `'sha256'`, `'md5'`, `'hmac256'`
 - `secret`: Secret string (for `hmac256`)
 - `keyname`: Name of the key that was stored with `cfg.key.set` (for `hmac256`) ^{v3.31}

Returns

- `ht`: Hash table structure

```
ha = hash.txt(hv, [gap])
```

Convert hash value to text

- Parameters**
- [hv](#): Hash Value, as returned from `ht:fin()`
 - [gap](#): optional gap string. Default is a colon ':'

- Returns**
- [ha](#): ASCII representation

Description Convert hash value into a human-readable format.

```
hv = hash.val(ha)
```

Convert hash string to value

- Parameters**
- [ha](#): ASCII representation of hash (non-hex digits are ignored)

- Returns**
- [hv](#): Hash Value.

Description Converts ASCII hex to string

```
hv = ht:fin()
```

Hash finalise

Description Produces hash value.



Use [hash.txt\(hv\)](#) to convert to ASCII

```
ht:upd(txt)
ht:upd(fileobject, [length])
```

Hash update

Description Updates the hash with text or file contents

Hash Table Structure

The hash table is returned from a call to `hash.new`.



There should not be any real need to use the table directly in an app.

- `.m` = Method (1, 5, 224, 256, -256 = HMAC-SHA-256)
- `.c` = Context for hash (opaque internal value)
- `.b` = Number of bytes processed
- `.op` = OPAD for HMAC-SHA-256

```
local h = hash.new('sha1')
h:upd('Testing')
local v = h.b -- should be 7
```

Library `hydrins` : HydrINS Support

```
res, txt = hydrins.cmd(cp, cmdtxt, [timeout])
res, txt, ht = hydrins.cmd(cmdtxt, [timeout]) v3.20
```

Send command to HydrINS

- Parameters**
- `cp`: Com Port Object
 - `cmdtxt`: The command to send (# is not required)
 - `timeout`: Timeout in milliseconds

- Returns**
- `res`: True if ACK'd by the HydrINS
 - `txt`: Response string
 - `ht`: HydrINS type (20 or 21)

Description Send a command to HydrINS.

```
...
if hydrins.wake(dp, 10, 20)
then
  hydrins.cmd(dp, '086;966')
end
```

```
ht = hydrins.parse(txt)
ht = hydrins.parse(txt, [prm]) v2.00
```

HydrINS parse text

- Parameters**
- `txt`: The received HydrINS data. MUST include the 6-byte binary header. The "W" wakeup character is optional.
 - `prm`: The #086 value. When present, parses HydrINS v1 format.

- Returns**
- `ht`: HydrINS table as described

Description Parse a HydrINS line into a table.

```
res = hydrins.run(cp)
res, ht = hydrins.run() v3.20
```

Tell HydrINS to run

Parameters

- `cp`: Com Port Object

Returns

- `res`: True if ACK'd
- `ht`: HydrINS type (20 or 21)

Description Issues the #028 command to put into run mode.

Short-hand for:

```
hydrins.cmd(cp, '028')
```

```
ht = hydrins.rx(cp, [timeout])
ht = hydrins.rx(cp, [timeout], [prm]) v2.00
ht = hydrins.rx([timeout], [prm]) v3.20
```

HydrINS receive

Parameters

- `cp`: Com Port Object (as returned by `ser.dev()`)
- `timeout`: Optional timeout, in milliseconds
- `prm`: The #086 value. When present, parses HydrINS v1 format.

Returns

- `ht`: HydrINS table of values, or NIL if not received

Description Read and parse a HydrINS record from the Com Port.


```
res = hydrins.wake(cp, [delaysec, [trysec]])
res, ht = hydrins.wake([delaysec, [trysec]]) v3.20
```

Wake up HydrINS

Parameters

- **cp**: Com Port Object
- **delaysec**: Number of seconds to delay before starting (default 0)
- **trysec**: Number of seconds to keep trying (default 130s)

Returns

- **res**: True if successfully woke up the HydrINS into command mode and received an ACK.
- **ht**: HydrINS type (20 or 21)

Description

Try and wake up the HydrINS.



You can make use of the `ht.cycle` value to work out a suitable combination of `delaysec` + `trysec`:

```
local ht = hydrins.rx(dp)
...
if ht
then
  if hydrins.wake(dp, ht.cycle-2, 4) -- 2s before next cycle
  end
```

Hydrins Table ht

ht

Hydrins Table `ht`

Parameters

- `ht.ver`: Hydrins version (1, 2 or 2.1)
- `ht.params`: Parameters 16-bit field
- `ht.alarms`: Alarms 16-bit field
- `ht.cycle`: Cycle time, in seconds
- `ht.tests`: Self tests 16-bit field (for Hydrins 2.1)
- `ht.water`: True if in water
- `ht.air`: True if in air
- `ht.PULSES`: HydrINS pulse count value
- `ht.VPAV .VPAVu`: For each, there is a value and optional units. i.e. `ht.TOTm` is the value, and `ht.TOTmu` is the unit string.
- `ht.SPV .SPVu`: (ditto)
- `ht.VMAV .VAMVu`: (ditto)
- `ht.SMV .SMVu`: (ditto)
- `ht.FAV .FAVu`: (ditto)
- `ht.SVG .SVGu`: (ditto)
- `ht.TOTp .TOTpu`: (ditto)
- `ht.TOTm .TOTmu`: (ditto)
- `ht.TOTn .TOTnu`: (ditto)
- `ht.B1p`: Battery 1 percentage
- `ht.B1v`: Battery 1 voltage (if present)
- `ht.B2p`: Battery 2 percentage (if present)
- `ht.B2v`: Battery 2 voltage (if present)
- `ht.BU`: Batteries used (if present)
- `ht.BF`: Batteries fitted (if present)
- `ht.TEMP .TEMPu`: temperature & units (for Hydrins 2.1 if enabled)
- `ht.NAMUR`: (for Hydrins 2.1 if enabled)

Description

Table of values for Hydrins

Library `iot` : Internet Of Things

The `iot` library provides utilities for sending data via FTP, and FTPS to the central IOT server.

- FTP default port is 21;
- FTPS (**implicit** TLS/SSL) default port is 990.
- *You can override the port number with the standard `[:xxx]` in the URL.*

```
txt = iot.abort([reasons])
```

IoT abort modem requests

Parameters

- `reasons`: String containing text (as in `iot.go`)

Returns

- `txt`: String representation of reasons that are still outstanding.

Description Cancel the modem requests.



Use with CAUTION. You may prevent the `rt.buffer` from ever connecting to the IoT server!

```
ok = iot.arch(filename, [archive]) v3.20
```

Archive file

Parameters

- `filename`: Filename to archive
- `archive`: True to archive. False to just delete. Default is `true`

Returns

- `ok`: True if successful.

Description Archive a file.



Does not currently support wildcard filenames.

iot.aup()

IoT abort user process

Parameters	None
Returns	None
Description	Abort the User Process (if active).

min = iot.azt([min]) ^{v3.10}

Set modem anti-cell-zombie timeout

Parameters	min: Anti-cell-zombie time, in minutes (default: 720 = 12hrs)
Returns	min: Time, in minutes.
Description	Set/Get the modem's anti-cell-zombie timer value.

Many cells will 'zombie' a device if it has been connected for a while.

Going through a brief disconnect-connect phase periodically ensures the cell keeps the Internet connection available.

See also ``iot.scmv`` - to keep modem online permanently.

txb,rx,nsll = iot.bytes([reset]) ^{v3.40}

IoT get byte counts

Parameters	reset: Pass <code>true</code> to reset the counters
Returns	txb: Number of plaintext transmitted bytes rx: Number of plaintext received bytes nsll: Number of ssl connections
Description	Returns TX and RX byte counters for the modem.



These are *plaintext* byte counts sent to the modem module for TCP connections, so the over-the-air byte count can be larger if using SSL, which will be dependent on certificate sizes, crypto settings etc. `nsll` can help estimate total over-the-air count.

```
ok = iot.cert(cafile, [certfile, keyfile]) v1.00  
ok = iot.cert([use])
```

IoT set PKI certificate

Parameters

- **cafile**: Local filename of the CA.pem file
- **certfile**: (optional) Local filename of the client certificate
- **keyfile**: (optional) Local filename of the matching client private key
- **use**: Passing 'false' will erase the CA, cert, and key

Returns

- **ok**: If successful

Description

Copies the local file to the modem for use as a CA certificate check, and optionally uploads the client certificate and key.

(Can only be called within the context of the modem task.)

```
iot.cert('/config/ca.pem')  
iot.cert('/config/ca.pem', '/config/client.pem', '/config/client.key')  
iot.cert(false) -- stop using CA/Cert/Key
```

iot.cho()

IoT clear hold off

Parameters

None

Returns

None

Description

Clear hold off.



Use this sparingly to avoid over-use of modem.

```
ok = iot.diag([target], [opts])
```

Send diagnostics info to IoT server

- Parameters**
- **target**: Optional target filename
 - **opts**: String containing options of what is to be sent:
 - 'info' = Information tree
 - 'config' = Configuration tree
 - 'app' = Lua App source code

- Returns**
- **ok**: Whether ok

Description Sends the diagnostics dump to the IoT server.

```
iot.diag() -- everything
iot.diag('my-file.diag')
iot.diag(_, 'config app') -- not info
```

```
ok = iot.dorun()
```

IoT allowed to run

Parameters None

- Returns**
- **ok**: Whether modem can still remain online

Description Check whether we should still be online, or whether we are about to hangup.

```
txt = iot.flg()
```

Get IoT reason string

Parameters None

Returns • **txt**: String containing the outstanding reasons for connecting to IoT

Description Returns a string compatible with iot.go values.

```
ht = iot.gho()
```

IoT get hold off

Parameters none

Returns • **ht**: The hold-off time in seconds

Description Return the current hold off time, in seconds

```
iot.go([reasons])
```

Trigger IoT connection

Parameters • **reasons**: optional reasons for triggering IoT

- 'info' = perform cellular survey ^{v1.01}
- 'data' = send data
- 'update' = perform update process
- 'time' = synchronise time
- 'ntp' = (same as 'time')
- 'user' = perform user actions (like magnet)
- 'term' = connect terminal OTA ^{v1.10}
- 'toa' = connect terminal OTA ^{v1.10}
- 'test' = test delivery
- 'diag' = send diag on update ^{v3.00}
- 'sms' = SMS transmission ^{v1.10}
- 'data time update' = default options

Returns None

Description Trigger connect and send.



The survey results are saved in [/Logs/cellinfo.txt](#). This process may take between 2 and 5 minutes (perhaps longer if there is no SIM installed).



The modem may be in a 'hold-off' period, but will connect when that period expires.

```
iot.hum([kill])
```

IoT hang up modem

Parameters • **kill**: true to perform nasty, immediate, power kill.

Returns None

Description Hang up the modem. Without the 'kill' option, the modem handler will let all tasks close down gracefully.

With the **kill** option, the modem power is cut immediately, and it can take a few minutes for the modem handler to timeout and clean up.

```
stop = iot.isf(name)
```

IoT is there a stop file

Parameters • **name**: Root name of the file to look for

Returns • **stop**: True if there is a '.stop'

Description Check whether there's a request to stop from the IoT server.

If found, the file is deleted from the server.

```
local stp = iot.isf('mine') -- looks for 'mine.{i.rt_sn}.stop'
```



```
ok,txt = iot.modem(cmd, [timems]) v3.31
ok,txt = iot.modem(cmd, [table]) v3.31
```

Send modem command

- Parameters**
- **cmd**: AT command string to send (without the 'AT')
 - **timems**: Wait time in milliseconds
 - **table**: Table of options
 - .time = timems
 - .look = (string) prefix to look for
 - .case = (boolean) case sensitive
 - .xok = (boolean) expecting OK
 - .prompt = (boolean) expecting '>' (special case)

- Returns**
- **ok**: Whether successful
 - **txt**: Result string

Description Allows sending additional 'AT' commands to the modem during configuration.



Can only be called within the context of `iotInit` or other `iotXXX` callbacks.

```
mv = iot.scmv([mv]) v3.10
```

Set modem stay-connected millivolt

- Parameters**
- **mv**: Millivolt setting to keep modem alive (default: 0)

- Returns**
- **mv**: Millivolt level

Description Setting `iot.scmv` with a non-zero value will keep the modem alive and ignore the `c.cell_mto`, all the time the power supply is $\geq$ `mv`.

As soon as the power supply drops below `mv` then the timings in `c.cell_mto` are followed.



The modem will still disconnect after the 'anti-cell-zombie' timeout (see `iot.azt`). However, if `sms.rxt(-1)` or `gps.time(-1)` then the modem will reconnect after a short pause.

```
iot.scmv(10000) -- keep online while above 10V
iot.azt(720)    -- 12hr anti-cell-zombie time
sms.rxt(-1)    -- keep SMS reception going
```

```
fc,fs = iot.send(srcptn, [options, [arch]])
```

Send files to IoT server

Parameters

- **srcptn**: File path pattern (wildcarded) for choosing what to send
- **options**: optional overrides for the transfers
 - ' ' = send plain
 - 'tfr' = use temp-file-rename process
 - 'gz' = use .gz compression
 - 'zlib' = use .zlib stream compression
 - _ (NIL/not present) = send using tfr + gz
- **arch**: Whether to archive the file when sent (default = true)

Returns

- **fc**: Count of files sent
- **fs**: Number of bytes sent

Description

Transfer files from the rt.buffer directory area that match the filename pattern.

Copies according to source ASCII-sort name.

- MUST only be called within the context of the modem process, within the iotData callback function.

```
iot.send('/Send/*.mine.txt')
iot.send('/Send/*.more.dat', ' ') -- no compression, direct
```

iot.set(uri)

Set IoT server URL

Parameters	• uri: Absolute or relative URL for the server
Returns	None
Description	Sets the address relative to the previous call to iot.set (i.e. appends this path to the previous full URL).

```
iot.set('ftp://user:pass@ftp.scannex.com:21/target')
-- or --
iot.set('/NewDirectory/Here')
```

Uses the scheme and server details from the previous call to iot.set

```
iot.set('Relative/Path/Here')
```

This function makes use of [\[rt.exp\]](#) to provide for Lua variable expansion:

```
-- include the Site Name and Serial Number:
iot.set('Relative/Path/{c.site_name}-{i.rt_sn}')
```

See [\[rt.exp\]](#) for details on the expansion mechanism.

iot.sho(value)

IoT set modem hold off

Parameters • [value](#): The hold off time required in seconds.

Returns None

Description Set modem hold off. Only applies if the current hold off is less than value.

ok = iot.trim() ^{v3.20}

Trim archive folder

Parameters None

Returns • [ok](#): True if successful.

Description Trim the archive folder, using the [c.arc_fc](#) and [c.arc_kb](#) values.

Default Functions

function iot._test

IoT default test callback

Description Default IoT Test function. Effectively:

```
function iot._data()  
  iot.set(c.iot_data)  
  iot.send('/Config/*.tst')  
end
```

function iot._onJob11

IoT default Job11 callback

Description Default onJob11 function. Effectively:

```
function iot._onJob11()  
  iot.go('update time')  
end
```

function iot._onJob12

IoT default Job12 callback

Description Default onJob12 function. Effectively:

```
function iot._onJob12()  
  smp.cut()  
  iot.go('data')  
end
```

function iot._data

IoT default data callback

Description Default IoT Data function. Effectively:

```
function iot._data()  
  iot.set(c.iot_data)  
  iot.send('/Send/*')  
end
```

function iot._user

IoT default user callback

Description Default IoT User function.

function iot._onMag

IoT default magnet callback

Description Default Magnet event handler. Effectively:

```
function iot._onMag()  
  iot.go('user update')  
  iot.cho()  
end
```

IoT Callbacks

The IoT callbacks are suitable for transferring data using the [iot](#) library commands. You can also use these commands to interact with the modem (e.g. issuing AT commands via the [modem](#) library commands).

```
function iotData()
```

IoT data callback

Description Called when the rt.buffer should send data to the central server.

If this function is not defined, then all files within the /Send directory are sent.

- Within this callback you should make use of `iot.set` and `iot.send` (and not much else).

```
function iotData()
  iot.send('/Send/*.dat')
  iot.set('Another/Path')
  iot.send('/Send/*.csv')
end
```

```
function iotIdle() v3.40
```

IoT Modem Idle callback

Description Called in between jobs while the modem is powered.

function `iotInit()` v3.31

IoT Modem Initialise callback

Description Called when the modem is powered, but before the SIM is initialised, allowing for additional configuration sequences using `iot.modem` or `modem.XXX`.

function `iotOff()` v3.40

IoT Modem Complete callback

Description Called just *before* powering off.

function `iotOffline()` v3.40

IoT Modem Offline callback

Description Called just *before* going offline.

function `iotOnline(ok)` v3.40

IoT Modem Online callback

Description Called when online connection has finished. `ok` is a boolean defining whether successful.

function `iotReg(ok)` v3.40

IoT Modem Initialise callback

Description Called when registration has finished. `ok` is a boolean defining whether successful.

function iotSetup() v3.40

IoT Modem Starting callback

Description Called when the modem is powered and SIM initialised allowing for additional configuration sequences using `iot.modem` or `modem.XXX`

function iotTest()

IoT test callback

Description Called when the rt.buffer should send test file(s) to the central server.

If this function is not defined, then all *.tst files in /Config directory are sent, and not archived.

function iotUpd()

IoT update callback

Description Called within the context of the Update mechanism.

This Lua callback occurs *after* the firmware has performed its processes.

function iotUser()

IoT user callback

Description Called when the rt.buffer should perform the user interaction (e.g. when the magnet is triggered).

The default function repeatedly sends a diagnostic dump to 'user.SERIAL.txt' in the c.iot_user directory (which is the c.iot_url by default). It checks for a 'user.SERIAL.stop' file - if this exists it will delete the file on the server and quit the loop. Otherwise, it will pause 15s, and loop.

The pause can be set with `c.user_ins`, and the total loop time set with `c.user_onm`.

Something like this:

```
function iotUser()
  local et = (c.user_onm or 2) * 60 + i.rt_alv
  local iv = (c.user_ins or 15)
  while (i.rt_alv < et) and iot.dorun()
  do
    iot.diag( rt.exp('user.{i.rt_sn}.txt'), 'config info' )
    if iot.isf('user') then break end
    rt.ms(1000 * iv)
  end
end
```

Library job : Job Scheduler

The job scheduler provides an efficient mechanism to handle repeat events - like sampling, triggering delivery etc.

There are 12 jobs, and each job can have up to 8 time slots (so that different times of the day, and/or days of the week, can have different frequencies).

When the job needs running, the Lua Event Task will execute onJob#. e.g. Job 1 will execute the Lua callback function `onJob1`.

`Job12` is reserved for the Data Delivery schedule, and defaults to 23:00:00 each day.

`Job11` is reserved for the Update mechanism schedule, and defaults to 23:00:00 each day.

Job Schedule Strings

The `rt.buffer` has a flexible job scheduler that is used throughout the firmware. There are up to twelve jobs that can be used for different purposes (e.g. deliver data; contact update server). Each job can have eight different time slots (e.g. for different times throughout the week).

A schedule is programmed with a string that has the following structure:

- The primary value is the interval (default is 24 hours)
 - `00:00:15` = every 15 seconds
- The `@` specifies a time range (default start time is 00:00:00, and default end time is 24:00:00)
 - `@23:00` = at (or from) 11 pm
 - `@6:00-23:30` = between 6:00am (inclusive) and 11:30pm.
- The `#` specifies the day bit-field
 - Where: 1 = Sunday, 2 = Monday, 4 = Tuesday, 8 = Wednesday, 16 = Thursday, 32 = Friday, 64 = Saturday
 - `#62` = weekdays
 - `#65` = weekends
 - `#d` = weekdays, Mon-Fri
 - `#e` = weekends, Sat+Sun
 - `#a` = all days (#127)
- A comma (,) separates the entries



The end time is *exclusive*. So, for example, `2:00@12:00-18:00` will fire at 12:00, 14:00, and 16:00 only. This method allows for clarity when changing the interval, say to 0:00:01.

Schedule Examples

- `00:00:30@8-18, 00:05:00`
 - every 30s between 8am and 6pm, and 5 minutes outside those times
- `00:00:30@8-18#2,00:00:45@8-18#60,00:05:00`
 - every 30s between 8am and 6pm on Monday; every 45s between 8am and 6pm on Tue-Fri, and 5 minutes otherwise.
- `4@8-18,12`
 - every 4 hours between 8am and 8pm, and every 12hrs otherwise (00:00:00 & 12:00:00)
- `@12:30`
 - an event that occurs each day at 12:30pm

Scheduled Jobs

```
job.set(jobn, txt, [variance])
```

Set job schedule

Parameters

- `jobn`: Job number, 1-12
- `txt`: The job schedule string
- `variance`: Set to true to apply delivery variance to the schedule.

Returns

None

Description

Sets the parameters for a job. Not all parameters are required for an entry, and you may have up to 8 time slots defined for each job. Spaces are optional, as are leading zeroes.

```
txt,var = job.txt(jobn)
```

Get job schedule

- Parameters**
- `jobn`: Job number, 1-12
- Returns**
- `txt`: Job schedule string (long-hand format)
 - `var`: Whether variance applied
- Description** Returns the textual representation and whether variance is applied for the job.

```
var = job.var(jobn, [variance])
```

Set or get job variance

- Parameters**
- `jobn`: Job number, 1-12
 - `variance`: If present, sets the variance flag for the job
- Returns**
- `var`: Whether variance is applied on this schedule.
- Description** Sets or queries the variance flag for a job.

Scheduled Job Callbacks

```
function onJob##(jobn, utc, utcj)
```

Job callback prototype

Parameters

- **jobn**: The job number, 1-12
- **utc**: The actual UTC time value. Note that this may be slightly behind the current UTC time if there has been a backlog of jobs.
- **utcj**: The UTC the job sees (if variance is applied)

Returns

None

Description

This is the function prototype of the job callback, i.e. for onJob1 etc.

The function name should match the job number:

```
function onJob1(n, u, j)
  -- n = 1
  -- u + j = time
  -- do stuff here...
end
```

However, with Lua, the following is also possible:

```
function Foo(n,u,j)
  -- stuff
end

onJob1 = Foo
onJob2 = Foo
```

UTC Based Jobs

The UTC based jobs are useful for one-time events that need to occur at a certain time, such as triggering a rendezvous to base at a certain time.

There are 16 UTC slots that can be used.

```
job.utc(jobu, utc)
```

Set a UTC based job

- Parameters**
- `jobu`: Job number, 1-16
 - `utc`: Time the job should run

Returns None

Description Sets the Job UTC slot to the UTC time provided

Calls the function defined for the job number:

```
onJobU##( ) ①
```

① where '`##`' is the jobu number

```
function onJobU5()
-- do something
end

job.utc(5, utc.cvt('2017-09-01 16:00'))
```

jobu = job.utc(utc, function)

Set a UTC based job function

Parameters

- **utc**: Time the job should run
- **function**: Lua function that is called at utc

Returns

- **jobu**: The job number assigned, or NIL if no free jobs

Description

Find a spare UTC job slot, and glue the supplied function to the onJobUxx event

e.g. Assuming slot 5 is free, the function will be linked to onJobU5. On execution, the onJobU5 function is unlinked, and therefore the function originally provided may be garbage-collected (i.e. if it was an unnamed function).

```
job.utc(os.time()+120, function() --[[ do stuff ]]- end)
job.utc(os.time()+120, MySpecialFunction)
```

utc,tmp = job.utc(jobu)

Query UTC job slot

Parameters

- **jobu**: Job number, 1-16

Returns

- **utc**: The time the job is scheduled for, or NIL if blank.
- **tmp**: True if the job is a temporary one (see [\[job.utc\]\(utc, function\)](#))

Description

Query Job UTC slot number.

UTC Based Job Callbacks

```
function onJobU##()
```

UTC Job callback prototype

Parameters None

Returns None

Description Called when the UTC job runs.

e.g.

```
function onJobU4()  
  -- do something  
end
```



The name of the function must match the UTC job number, 1-16

Library `jobq` : Job Queue

The job queue library provides a useful way to schedule functions to run sequentially.



Added in firmware v3.43

This is useful when devices need data extracting, configuration changes, and diagnostics to be run. Other tasks can post to the job queue and the function callbacks will be run strictly in order.

There is also a global job queue `JQ` that can be used in situations where you might need a simple function call: e.g. `coro.add(jobq.run)` — will run all jobs in JQ as a coroutine. To use this global queue, call the `jobq.XXX` functions without the first table parameter. (You don't need to use `job.new` for the global one - it is created automatically.)



The job queue is atomic - you can safely add and pop from any job queue in any Lua context.

```
jgo = jobq.new()
```

Create a new Job Queue Object

Returns

- `jgo`: Returned Job Queue Object

Description

Creates a new job queue.

You can have multiple job queue objects that are serviced by different parts of your Lua app.



A Job Queue Object is just an indexed table with metamethods. Consequently you can count the number of pending jobs with the standard Lua `#` operator. e.g.
`count = #jgo`

Typical usage

Generally, you will use the methods `jobq.new` and `jmq:run`

```
mjq = jobq.new()

function MyJob1(a1,a2)
  -- so something with a1 & a2
end

function MyJob2()
  -- do work
end

...
mjq:add(MyJob1, {42, 'Interesting'})
mjq:add(MyJob2)

...
mjq:run() -- runs all jobs
```

Of course, you can wrap the run into a coroutine:

```
coro.add( jobq.run, mjq ) -- effectively runs jobq.run(mjq) == mjq:run()
```

Using the global job queue

You can use the global job queue `JQ`:

```
jobq.add(MyJob1)
jobq.add(MyJob2)
...
coro.add( jobq.run )
```



Note the subtle difference of ‘.’ vs ‘:’)

Library `jobq` | `jqr` : Job Queue Objects

```
ok = jqr:add(func, argt, [always])
```

Add callback to job queue

- Parameters**
- `func`: Lua function
 - `argt`: Table of arguments that will be passed to `func`
 - `always`: true means always add. Default is `false` (ie only add if not already in the queue)

- Returns**
- `ok`: True if added to the queue

Description Scans the `jqr` for a matching `func` + `argt` set.

The callback is queued if the combination is not already in the queue, or if `always` is `true`.

```
count = jqr:del()
count = jqr:del(func, all)
count = jqr:del(func, argt)
```

Delete jobs from the queue

- Parameters**
- `func`: Lua function
 - `argt`: Argument table
 - `all`: (bool) whether to remove all instances of `func`

- Returns**
- `count`: Number of jobs deleted

Description Deletes either all jobs (`jqr:del()`), or a specific set of jobs from the queue.

`ok = jqo:is(func, [argt])`

Check if callback in job queue

Parameters	• <code>func</code>: Lua function • <code>argt</code>: (optional)Table of arguments
Returns	• <code>ok</code>: True if exists in the queue
Description	Scans the <code>jqo</code> for a matching <code>func</code> + <code>argt</code> set. If <code>argt</code> is not provided, then checks if there is any <code>func</code> queued.

`foo, argt = jqo:pop()`

Pop a job from the queue

Parameters	None
Returns	• <code>foo</code>: Function callback • <code>argt</code>: Argument table
Description	Pops a job from the queue.

Use `unpack` to convert `argt` into a set of arguments.

```

mqj = jobq.new()
...
local foo, argt = mqj:pop()
if (foo) then foo(unpack(argt)) end

```

ok,rv... = jqo:poprun()

Pop a job from the queue

Parameters None

Returns

- **ok**: If the job ran correctly (boolean)
- **rv...**: Set of arguments returned by the function

Description Pops a job from the queue and runs it.

This is a simpler way of straight calling one job.

```
mjq = jobq.new()
...
local res = { mjq:poprun() }
-- res[1] has 'ok' return value
```

count = jqo:run()

Run all jobs in the queue

Parameters None

Returns

- **count**: Number of jobs successfully run

Description Pops each job from the queue and runs it, looping until all are run.

```
mjq = jobq.new()
...
local count = mjq:run()
```

Library `kvc` : Key Value Container

The `kvc` library can be used to save Lua RAM for sets of values. It uses the same internals as the `c` configuration tree.

```
kvco = util.kvc(nb,ni) v2.00
```

Create new key-value container object

Parameters

- `nb`: Number of bytes for the data
- `ni`: Number of index entries

Returns

- `kvco`: KVC object

Description

Return a KVC object. RAM is taken from the Lua memory for the KVC object, the data area, and the index block.

RAM will be recovered during garbage collection when the `kvco` object becomes unused.

```
local k = util.kvc(2048,64)
k.my_value=1.234
k.my_setting='Test'
```

Library `kvc` | `kvco` : Key-Value Container Objects

```
nb = kvco:_free v2.00
```

Query KVC object free bytes

Parameters None

Returns • `nb`: Number of bytes.

Description Returns the number of bytes free in the data area of the `kvco` object.



Not a function, but a pseudo variable.

```
nb = kvco:_sizei v2.00
```

Query KVC object index capacity

Parameters None

Returns • `nb`: Number of bytes.

Description Returns the number of bytes used by the index area of the `kvco` object.



Not a function, but a pseudo variable.

```
nb = kvco:_size v2.00
```

Query KVC object byte size

Parameters None

Returns • `nb`: Number of bytes.

Description The total number of bytes in the data area of the `kvco` object.



Not a function, but a pseudo variable.

```
ni = kvco:_freei v2.00
```

Query KVC object free index entries

Parameters None

Returns • **ni**: Number of index entries free

Description Returns the number of index entries free in the **kvco** object.



Not a function, but a pseudo variable.

```
ok = kvco:_load(filename,[clear]) v2.00
```

Load data from file to KVC object

Parameters • **filename**: Name of the text file to load from
 • **clear**: Whether to clear the kvco before loading (default=true)

Returns • **ok**: If successful

Description Loads the file into the **kvco** object.

```
ok = kvco:_save(filename) v2.00
```

Save data from KVC object to file

Parameters • **filename**: Name of the text file to save to

Returns • **ok**: If successful

Description Saves the contents of the **kvco** object into the text file.

Library `modbus` : MODBUS functions

The MODBUS library provides raw frame transmission and reception, as well as a higher-level command processor that can frame and de-frame data.

Only supports MODBUS RTU on serial - either RS-232 or RS-485 (with an adapter).

```
rt,txt = modbus.cmd(cp, adr, fnc, [tfmt, ttab, [rfmt, ms]])
```

Send MODBUS command

Parameters

- `cp`: COM Port object
- `adr`: Slave address
- `fnc`: Function number
- `tfmt`: Transmit format string
- `ttab`: Transmit data table
- `rfmt`: Receive format string
- `ms`: Receive time out

Returns

- `rt`: Table of values (as specified by `rfmt`)
- `txt`: Remaining string (if any)

Description

Send a MODBUS RTU request, and read the reply.

The indexed table 'ttab' is used, along with `tfmt`, to create a data string that is passed to `modbus.tx`. The reply formatting is optional.

For example to write to register 1000:

```
modbus.cmd(dp, 1, 16, 'WWBW', {1000, 1, 2, 7878})
-- register 1000 will have 2 bytes written = 7878
```

`Modbus.rx` is called, and any reply is pulled apart using the `rfmt` specifiers to create a list of results.

For example, the Read Holding Registers (0x03) function will return a byte-count, followed by $n \times$ 16-bit values (`rfmt` = 'BWWW' for 3 values):



Use `modbus.rhr` for Read Holding Registers. This example provided for instruction only.

```

t = modbus.cmd(dp, 1, 3, 'WW', {2, 3}, 'BWWW')
if t
then
  -- t[1] = Number of bytes
  -- t[2] = register 0002
  -- t[3] = register 0003
  -- t[4] = register 0004
  reg1, reg2, reg3 = unpack(t)
end

```



Note the use of Lua's `unpack(t)` to turn a table into a list of values.

```
tab = modbus.fs2t(tfmt, txt) v1.00
```

Unpack string to table

Parameters

- `tfmt`: Format string
- `txt`: Binary block of ModBus data to unpack

Returns

- `tab`: Lua table of values

Description

Utility function (used by `modbus.cmd` etc) to unpack a binary block received by ModBus and convert to a table of values.

```
txt = modbus.ft2s(tfmt, ttab) v1.00
```

Pack table to string

Parameters

- `tfmt`: Format string
- `ttab`: Table of values to pack

Returns

- `txt`: ModBus packed block to send

Description

Utility function (used by `modbus.cmd` etc) to pack a set of values into a binary string block ready to send.

```
modbus.rdi(cp, adr, fc, nc, [ms]) v1.30
```

Read discrete inputs

- Parameters**
- `cp`: COM Port object
 - `adr`: Slave address
 - `fc`: First coil to read
 - `nc`: Number of coils to read
 - `ms`: Receive time out
- Returns**
- `tab`: Table of booleans (one for each input)
 - `err`: ModBus error

Description Read discrete inputs (function 02)

```
tab = modbus.rhr(cp, adr, regs, regc, [rfmt, [ms]]) v1.00
```

Read MODBUS holding registers

- Parameters**
- `cp`: COM Port object
 - `adr`: Slave address
 - `regs`: First register to read
 - `regc`: Count of registers to read
 - `rfmt`: Receive format string
 - `ms`: Receive time out
- Returns**
- `tab`: Table of values
- Description** Wrapper function for `modbus.cmd` that performs a ModBus **Read Holding Registers** (3_{dec} / 0x03) function.

```
local t = modbus.rhr(dp, 1, 28, 6, 'W+') -- read DVP readings
```

```
tab = modbus.rir(cp, adr, regs, regc, [rfmt, [ms]]) v1.30
```

Read input registers

Parameters

- `cp`: COM Port object
- `adr`: Slave address
- `regs`: First register to read
- `regc`: Count of registers to read
- `rfmt`: Receive format string
- `ms`: Receive time out

Returns

- `tab`: Table of values

Description Read Input Registers (function 04).



Same format as `modbus.rhr` function 03.

```
tab, err = modbus.rmc(cp, adr, fc, nc, [ms]) v1.30
```

Read multiple coils

Parameters

- `cp`: COM Port object
- `adr`: Slave address
- `fc`: First coil to read
- `nc`: Number of coils to read
- `ms`: Receive time out

Returns

- `tab`: Table of booleans (one for each coil)
- `err`: ModBus error

Description Read multiple coils (function 01)

```
dt,err = modbus.rsid(cp, adr, fmt, [ms]) v1.30
```

Read server ID

Parameters

- `cp`: COM Port object
- `adr`: Slave address
- `fmt`: In format string
- `ms`: Receive time out

Returns

- `dt`: Data table of results (or string if `fmt` not defined)
- `err`: ModBus error

Description

Read Server ID (function 17/0x11)

```
ok,adr,fnc,data = modbus.rx(cp, [ms])
```

Receive MODBUS RTU frame

Parameters

- `cp`: COM Port object
- `ms`: Optional timeout (default 250ms)

Returns

- `ok`: Whether the frame response is CRC'd ok
- `adr`: Address sent by the slave
- `fnc`: Function number sent by the slave
- `data`: String of remaining data (not including the CRC)

Description

Receive a MODBUS RTU frame.

modbus.set(cp, bps, [rs485])

Setup MODBUS on COM port

- Parameters**
- **cp**: COM Port object, e.g. from ser.dev()
 - **bps**: Baud and Protocol string
 - **rs485**: true or '485' = RS-485, false or '232'=RS-232

Returns None

Description Sets the baud-rate and protocol for the MODBUS.

```

local dp=ser.dev()
modbus.set(dp, '19200e8')

```

rx_t,tx_t = modbus.time()

Return MODBUS timings

Parameters None

- Returns**
- **rx_t**: Time spent receiving packet (ms)
 - **tx_t**: Time spend transmitting packet (ms)

Description Debug function to return the last timings.

```
modbus.tx(cp, adr, fnc, d1, d2)
modbus.tx(cp, adr, fnc, str)
```

Transmit MODBUS RTU frame

- Parameters**
- `cp`: COM Port object
 - `adr`: Slave address 1-127
 - `fnc`: Function number
 - `d1,d2`: Data bytes
 - `str`: String

Returns None

Description Sends a CRC'd MODBUS RTU frame on the COM port 'cp'.

```
modbus.tx(dp, 1,3, 0,1, 0,2) -- read two registers from reg 0001
```

```
tab,txt = modbus.wmr(cp, adr, regs, tfmt, ttab, [ms])
```

Write MODBUS multiple registers

- Parameters**
- `cp`: COM Port object
 - `adr`: Slave address
 - `regs`: First register to read
 - `tfmt`: Transmit format string
 - `ttab`: Transmit data table
 - `ms`: Receive time out

Returns

- `tab`: Table of values returned (should be just start register and count)
- `txt`: Remaining string (if any)

Description Wrapper function for `modbus.cmd` that performs a ModBus **Write Multiple Registers** (16_{dec} / 0x10) function.

```
modbus.wmr(dp,2,19, 'W',{5}) -- sets OverFLO average length=5
```

```
rt,txt=modbus.wsr(cp, adr, rno, rv, [ms]) v1.30
```

Write single register

Parameters

- `cp`: COM Port object
- `adr`: Slave address
- `rno`: Register Number
- `rv`: Register Value
- `ms`: Receive time out

Returns

- `rt`: Table of values (like `modbus.cmd`)
- `txt`: Remaining text, if any (like `modbus.cmd`)

Description

Write Single Register (function 06)

modbus.cmd tfmt & rfmt Format Strings

The format strings tell the command processor how to package data, and how to parse it.

- **B** = 8-bit unsigned byte
- **b** = 8-bit signed byte ^{v1.30}
- **C** = 8-bit character ^{v1.30}
- **W** = 16-bit unsigned word
- **w** = 16-bit signed byte ^{v1.30}
- **D** = 32-bit double unsigned word ^{v1.00}
- **d** = 32-bit double signed word ^{v1.30}
- **F** = 32-bit IEEE-754 floating point value ^{v1.00}
- **S** = string
- **SN** = string with length 'N' ^{v1.30}
- **+** = keep using the last parameter until no more data (only applicable at the end of the string) ^{v1.00}

Library `modem` : Modem interaction

The `modem` library includes commands to interact with the modem. All the commands must be run within the context of the modem task. Therefore, they should only be used within the `iotXXX` callback functions (e.g. `iotStart`)

```
clr = modem.clearfplmn([clr]) v3.40
```

Request Clear Forbidden PLMN List

Parameters

- `clr`: Set to `false` to *not* clear the fPLMN list. (Optional)

Returns

- `clr`: True if the clear request is pending.

Description Request that the modem handler query and clear the fPLMN during the next `iotSetup` phase.

If there is a non-blank fPLMN list in the SIM a log entry is created:

```
mdm    fPLMN cleared (32F40132F42332F40232F403)
```

(Showing the original fPLMN value)

```
ok,txt = modem.cmd(cmd, [timems]) v3.40
ok,txt = modem.cmd(cmd, [table]) v3.40
```

Send modem command

Parameters

- `cmd`: AT command string to send (without the 'AT')
- `timems`: Wait time in milliseconds
- `table`: Table of options
 - `.time` = timems
 - `.look` = (string) prefix to look for
 - `.case` = (boolean) case sensitive
 - `.xok` = (boolean) expecting OK
 - `.prompt` = (boolean) expecting '>' (special case)

Returns

- `ok`: Whether successful
- `txt`: Result string

Description Allows sending additional 'AT' commands to the modem during configuration.



Same as the `iot.modem` function

```
txt,blnk = modem.fplmn([clr]) v3.40
```

Read/Clear Forbidden PLMN List

- Parameters**
- `clr`: Set to `true` to clear the fPLMN list. (Optional)
- Returns**
- `txt`: The fPLMN value (before clearing)
 - `blnk`: (boolean)`true` if `txt` is only 'F' characters (i.e. blank)
- Description** Manages the SIM card's Forbidden Public Land Mobile Network list.

The firmware function queries the SIM STATUS for fileid 28539 and extracts the length. It then reads the binary file data and returns it.

If `clr` is `true` then the firmware function will update the SIM card EF with `F`s if the fPLMN list is not blank.



This function does not normally need to be used. Just use the `modem.clearfplmn()` function.

```
function iotSetup()
  if DoClearFPLMN ①
  then
    local txt,blnk = modem.fplmn(true)
    if not blnk
    then rt.log('fPLMN',txt) ②
    end
    DoClearFPLMN = nil
  end
end
```

① In a job set DoClearFPLMN=true so this will be called periodically on demand

② If the value is not blank then we log it to the /Logs/system.log file



This mechanism is already coded in firmware using the `modem.clearfplmn()` call.

modem.kill() ^{v3.40}

Kill modem

Parameters	None
Returns	None
Description	Forces the modem to be powered off.

txt = modem.simrb(fileid, [offset], len) ^{v3.40}

Read SIM FileID Binary

Parameters	• fileid: The decimal FileID (e.g. 28539) to read from the SIM card • offset: Optional offset • len: Length of the file to read
Returns	• txt: The ASCII-hex value returned from the SIM (e.g. 'FFFFFFFFFFFFFFFFFFFFFFFF')
Description	Returns the results from the <code>AT+CRSM=176,fileid,offset,offset, len</code> command.

tbl,len,orig = modem.sims(fileid) ^{v3.40}

Read SIM FileID STATUS

Parameters	• fileid: The decimal FileID (e.g. 28539) to read from the SIM card
Returns	• tbl: The status array as a table. See <code>modem.stot</code> • len: The integer value of the Tag '80' (decimal 128) • orig: The original string returned
Description	Returns the results from the <code>AT+CRSM=192,fileid</code> command.



See 3GPP 102.221 specifications

```
ok = modem.simwb(fileid, [offset], txt) v3.40
```

Write SIM FileID Binary

Parameters

- **fileid**: The decimal FileID (e.g. 28539) to write to the SIM card
- **offset**: Optional offset
- **txt**: The ASCII-hex data to write

Returns

- **ok**: Whether successful

Description

Returns the results from the `AT+CRSM=214,fileid,offseth,offsetl,len,txt` command.



The **len** is calculated from the length of the **txt** value.

```
tbl, len = modem.stot(txt) v3.40
```

Convert SIM STATUS to table

Parameters

- **txt**: The ASCII-hex data to convert.

Returns

- **tbl**: The status array as a table. The index is the decimal value of the field.
e.g. `tbl[128]='000C'` — length
- **len**: The integer value of the Tag '80' (decimal 128)

Description

Iterates the **txt** data and extracts the information based on 3GPP 102.221 specification.

The length field's value is also returned, because this is most frequently needed.

This function is used by `modem.sims`.

Library `nmea` : NMEA Functions

The NMEA library provides firmware functions that convert from ASCII to Lua table, and from Lua table to ASCII.

```
tab, special, ok = nmea.tab(txt)
```

Convert NMEA line to table

- Parameters**
- `txt`: ASCII NMEA line
- Returns**
- `tab`: Lua table of values
 - `special`: True if an AIS 'I' record
 - `ok`: True if the checksum is good

Description Convert a line of NMEA data to a table.

```
local tb = nmea.tab('$GPAAM,A,A,0.10,N,WPTNME*32')
```

```
--[[
  tb[1] = 'GPAAM'
  tb[2] = 'A'
  tb[3] = 'A'
  tb[4] = 0.10
  tb[5] = 'N'
  tb[6] = 'WPTNME'
]]--
```

```
txt = nmea.txt(tab, [special])
```

Convert table to NMEA string

Parameters

- **tab**: Table of values
- **special**: True if this is an AIS '!' record

Returns

- **txt**: ASCII output, with checksum and CRLF

Description

Convert Lua table to NMEA string.

```
local tx = nmea.txt( {'PQRS', 12.34, 'TEST'} )
```

Library `out` : Digital Outputs

The digital output library allows direct control of the four logic outputs on the expansion header HD3.

Additional plug-in PCBs can provide for signalling open-collector outputs, high-current drive outputs, etc - depending on the application.

```
on = out.get(n) v3.00
v = out.get() v3.00
```

Read output state

- Parameters**
- `n`: Which output to query (1-4)
- Returns**
- `on`: True if the output is enabled
 - `v`: Bit-field integer for all outputs
- Description** Query the output state.

```
out.init([def]) v1.10
```

Initialise outputs

- Parameters**
- `def`: Default values
- Returns** None
- Description** Call this once in the App to initialise the outputs.

If included, 'def' will set the startup state of the outputs. The default value is 0.

out.off(n) v1.10

Turn off output

Parameters

- **n**: Which output to turn off (1-4)

Returns None

Description Turn OFF a selected output.

out.on(n) v1.10

Turn on output

Parameters

- **n**: Which output to turn on (1-4)

Returns None

Description Turn ON a selected output.

```
mp, xp, ep = out.pwo() v2.00
out.pwo(v) v2.00
out.pwo(n, en) v2.00
mp, xp, ep = out.pwo(vs) v3.21
```

Control power outputs

Parameters

- **v**: Bitfield to set both outputs (D0=main, D1=daughter, D2=engineer)
- **vs**: String defining which outputs to enable ('d'=Device port, 'x'=Expansion port, 'e'=Engineer port)
- **n**: Which output to turn off (1=main, 2=daughter, 4=engineer)
- **en**: True to enable the output

Returns

- **mp**: Mainboard Power status
- **xp**: eXpansion board Power status
- **ep**: Engineer Power status

Description Either queries the two power outputs, or controls the power outputs.

Mainboard power output is 3.6V

Expansion power output is 12V with the RS485/switchout convertor PCB.

Engineer power output is 3.3V (with a nominal 0.3V Schottky diode drop), but will rise to 5V when the USB is connected (because it's the same pin!).



Overrides the `cp:pwo()` function! Use *either* `out.pwo` or `cp:pwo`



v3.21 always returns the `mp, xp, ep` set

```
out.set(v) v1.10  
out.set(n, ison)
```

Set outputs

- Parameters**
- `v`: Binary value to output (0-15)
 - `n`: Which output to change (1-4)
 - `ison`: Boolean value, true/false

Returns None

Description Set all the outputs, or control an individual output.

```
out.set(2, true) -- turn output #2 ON  
out.set(1, false) -- turn output #1 off
```

Library `pwr` : Power Control

`pwr.apo(n)`

Control ADC power output

Parameters

- `n`: 0 = Power off
1 = Power on

Returns None

Description Control power for power outputs for ADC reference.



The `rt.adc` command may override the ADC reference power output.

```
bn,mv = pwr.back([bn], [mv]) v3.00
bn,mv,rmv = pwr.back([bn], [mv], [rmv]) v3.31
```

Set backup battery

Parameters

- `bn`: Battery for backup (0=none, 1=BATT1, 2=BATT2, 3=EXT)
- `mv`: Minimum millivolts for the other inputs
- `rmv`: Restore millivolt setting

Returns

- `bn`: As above
- `mv`: As above
- `rmv`: As above

Description Sets which battery is the backup battery, and a minimum voltage.

e.g. for lead acid, `pwr.back(0,10800)` will enter backup mode at 10.8V

e.g. `pwr.back(0, 10800, 11500)` will enter backup mode at 10.8V and resume at 11.5V



Firmware v3.31 defaults `rmv` to (`mv` + 100mV).

pwr.boot()

Reboot device

Description Reboot the rt.buffer.**pc = pwr.dpc([mA, [V]])** ^{v1.00}

Set the coulomb consumption for device

Parameters

- **mA**: Current in milliamps
- **V**: Optional voltage measured at (3.6V default)

Returns

- **pc**: Power consumption in milli-coulombs (@3.6V)

Description Set or get the Device port power consumption.

To get the current value, pass no parameters.

pc = pwr.epc([mA, [V]]) ^{v1.00}

Set the coulomb consumption for engineer

Parameters

- **mA**: Current in milliamps
- **V**: Optional voltage measured at (3.3V default)

Returns

- **pc**: Power consumption in milli-coulombs (@3.3V)

Description Set or get the Engineer port power consumption.

To get the current value, pass no parameters.

```
sa,sl,cr,cw = pwr.gpm()
```

Get power manager metrics

Parameters None

Returns

- **sa**: Seconds alive
- **sl**: Seconds in ultra-low power
- **cr**: CPU cycles running
- **cw**: CPU cycles waiting

Description Get power manager metrics. Can be used to calculate power consumption:

- Effective CPU freq.cf = $(cr + cw) / (sa - sl)$
- CPU run timert = $sa * cr / (cr + cw)$
- Idle percentageip = $(sa - rt) / sa * 100$

```
pwr.off()
```

Turn off device

Description Enter deep sleep / off mode.

Wakes up on USB or Engineer Serial port re-connection.

```
pt,cpu = pwr.rsn()
```

Query the power reasons

Parameters None

Returns

- **pt**: Table of {"section"=count} indicating which power functions are in use.
- **cpu**: CPU frequency, in Hz

```
bc = pwr.sbc([ac, [v]])
```

Set battery coulomb capacity

- Parameters**
- **ac**: Battery capacity <200 = A/hr
>200 = Coulombs
 - **v**: Voltage of battery (default is 7.2V)

- Returns**
- **bc**: Battery coulombs

Description Set battery capacity for power consumption estimation.

To read the capacity, pass no parameters

```
pwr.sbc(26, 7.2) -- 26Ahr @ 7.2V = 93600C
pwr.sbc(5, 6) -- 5Ahr @ 6V = 15000C (@7.2V)
local c = pwr.sbc() -- read the value
```

```
pwr.scu(c) v1.00
```

Set coulombs used

- Parameters**
- **c**: Coulombs used

Returns None

Description Sets the number of coulombs used. Useful if replacing with a half-full battery.

```
mv,sv,ws,ta = pwr.tmv() v2.00
m1,s1,m2,s2,mE,sE,ta = pwr.tmv(true) v2.00
```

Query the transient minimum voltage

Parameters None

Returns

- **mv**: Minimum voltage (mV)
- **sv**: Start voltage (mV)
- **ws**: Which source. 1, 2, or E
- **m1,s1**: Minimum and start for Battery 1
- **m2,s2**: Minimum and start for Battery 2
- **mE,sE**: Minimum and start for External power
- **ta**: Time alive of measurement

Description Returns the Transient Minimum Voltage.



This value is obtained each time the modem is powered up. If no measurement has happened, `pwr.tmv` will return `0,0,'?'`

```
pwr.work(ms)
```

Allow hard work

Parameters

- **ms**: Time to perform hard work (0-2000) milliseconds.
0 = back to normal (low) power

Returns None

Description Runs at 96MHz for a time, before falling back to low power, 12MHz, running.

Development Functions

These functions are for internal testing at Scannex, and are not guaranteed to remain constant between firmware versions.

```
pwr._bv(boost, unboost)
```

```
pwr._bv() v1.03
```

Set the boost values

Description Set the boost millisecond timers.

Use boost=0 to disable auto-boost Pass no parameters to set the defaults.

```
pwr._iv(min, max)
```

```
pwr._iv() v1.03
```

Set the idle timers

Description Set the idle timers for ultra-low power mode.

Pass no parameters to set the defaults.

Library `rt` : `rt.buffer` Utility Functions

```
rt.print(...) v1.30
print(...) v1.30
```

Print string

Parameters (As for Lua's print function)

Description Print to the USB terminal, the TOA terminal, or USB log.



This is useful for debugging Lua functions when you execute the function from the USB terminal with `lx`.



`rt.print` will output nothing if not in a terminal context, but `print` will output to USB if not in a terminal.

```
uv,rs = rt.adc([phases], [m], [c])
```

Get ADC reading

Parameters

- `phases`: Choice of ADC phase (to save power).
 ' ' = do all phases (takes about 170ms)
 'L' = lock and power up
 'l' = lock, but don't change power
 'S' = Start fresh sample
 's' = Start sample
 'R' = Read sample
 'U' = Un-power and unlock
 'u' = Unlock only
- `m`: Slope for the ADC value
- `c`: Offset for the ADC value

Returns

- `uv`: Microvolts.
 Range ± 0 -1500000uV (± 0 to 1.5V)
 (3.0V is the reference voltage)
- `rs`: Raw Sample Value

Description Get pressure reading in microvolts, optionally calculating $y=mx+c$



Use the new `adc.go()` and `adc.get()` library calls.



The ADC is LTC2485, 24-bit delta-sigma, that has V-REF = 3.0V.

```

local p = rt.adc('LSRU') -- does the default
local p = rt.adc('LSRU') -- measures without powering up
local p = rt.adc(0.001, 500) -- apply m & c

```

i. = rt.atom(function, args[]) + Atomic function call

Parameters

- **function**: Lua function to execute
- **args**: Arguments that are passed to 'function'

Returns Varies (depending on the function)

Description Execute the function atomically. Syntactically the same as pcall, but wraps with a mutex so that data writes and file renames can occur atomically.



The function **MUST** be a short function, with no waits. Ideally it should return in milliseconds. Otherwise much of the rt.buffer will be blocked!



Use coroutines where possible!

str = rt.bin(txt) v1.00

Convert ASCII hex to binary

Parameters

- **txt**: Human-readable ASCII Hex to convert to binary string. Can include separators between hex values

Returns

- **str**: Binary string

Description Convert a string of ASCII-hex into a binary representation.

```
b = rt.bool(v, [dflt]) v1.01
```

Convert boolean value

Parameters

- **v**: Value to check
- **dflt**: (optional) default value

Returns

- **b**: Boolean result

Description

Returns true or false based on the value of v.

- If v is a Lua boolean type, then b = v.
- If v is a Lua number, then b = (v != 0)
- If v is a Lua string, then a non-case-sensitive comparison occurs:
 - b = true if v is 'y', 't', or starts with "true" or "yes"
 - b = false if v is 'n', 'f', or starts with "false" or "no"
 - b = dflt or false if v has any other value.
- If v is NIL then b = dflt or false

```
i. = rt.call(function, args[]) + Protected call
```

Description

Non-atomic version of rt.atom

The same as Lua's pcall, just presented under the rt group.

rt.coldboot() v3.42

Reboot device

Parameters None**Returns** None**Description** Reboots the device as soon as possible.**v = rt.cyc()** v1.03

Get Cortex cycle counter

Parameters None**Returns** • **v**: Cortex cycle counter**Description** Returns the 32-bit CPU cycle counter.**v = rt.dct(t)** v1.00

Deep copy table

Parameters • **t**: Table to deep copy**Returns** • **v**: Resulting copy of table**Description** Deep-copies a Lua table. Any changes made to **v** will therefore be independent of the original table **t**.

using the Lua standard `v=t` will only result in a single table in memory. Any changes to `v` will affect `t` as well. That's why a "deep copy" is needed to produce a replica of the original table that is independent.



Only the Lua types: `LUA_TNIL`, `LUA_TBOOLEAN`, `LUA_TNUMBER`, `LUA_TSTRING`, and `LUA_TLIGHTUSERDATA` are handled.

```
sn,cn,pn,dt = rt.dev() v3.00
```

Get device details

Parameters None

Returns

- **sn**: Serial number
- **cn**: Company name
- **pn**: Product name
- **dt**: Description text

Description Read the device details (as shown in the USB enumeration).

```
ok = rt.dorun()
ok = rt.ok() v3.31
```

Still allowed to run

Parameters None

Returns

- **ok**: True if still keep running.

Description Check whether Lua can still run in this task.



Only useful within the context of Lua Loop Task and Modem Task. (i.e. do not use within the Lua Event Task - it will time out!)

```
txt = rt.exp(txt)
```

Expand a string

- Parameters**
- `txt`: String to expand
- Returns**
- `txt`: Expanded string
- Description** Expand string using `{..}` to evaluate Lua values.

This function forms a key part of the `iot.set` method.

```
local s = rt.exp('Data-{c.site_name}-{i.rt_sn}.csv')
```

Because the text within the `{..}` is evaluated and executed within Lua, you can even use expressions:

```
local s = rt.exp('Data-{i.rt_sn * 12345 + 34.56}')
```

```
txt = rt.hex(str, [gap]) v1.00
```

Convert binary to ASCII hex

- Parameters**
- `str`: Binary string to convert
 - `gap`: (optional) Gap character
- Returns**
- `txt`: Human readable ASCII hex string
- Description** Convert a binary string into human readable ASCII hex.

rt.led(en) v1.00

Enable debug LED sequence

Parameters

- **en**: True to enable the 5/4 LED pulse

Returns None

Description Scannex debug facility to flash LED in the 5/4 pulse sequence.

rt.log(module, txt)

Write log entry

Parameters

- **module**: Module string
- **txt**: Text to log

Returns None

Description Log to the /Logs/system.log file.

rt.luaboot() v3.42

Reboot Lua

Parameters None

Returns None

Description Reboots Lua as soon as possible.

```
ms,ma,mr = rt.mag([ms]) v2.00.0144
```

Set magnet delay

- Parameters**
- **ms**: Time the magnet has to be active (milliseconds)
- Returns**
- **ms**: Time the magnet has to be active
 - **ma**: True if the magnet is considered on (i.e. onMag has been called)
 - **mr**: True if the magnet IO line is active (i.e. the pre-timer value)
- Description** Controls the delay on the magnet before **onMag** is called

```
res = rt.ms(value)
ms = rt.ms() v1.00
```

Delay milliseconds

- Parameters**
- **value**: Delay time in milliseconds
- Returns**
- **res**: Reason - 0=OK, 1=TaskTimeOut, 2=LuaAbort
 - **ms**: System millisecond counter
- Description** Delay number of milliseconds. The value is limited to half the Lua timeout value for the calling task.

Or, with no parameters, returns the system time in milliseconds.



The actual time will be in the range of 'value-1 < actual ≤ value'. For small delays use **rt.us**


```
fv = rt.msf(offset) v2.00
```

Calculate millisecond finish time

Parameters

- **offset**: Offset time in milliseconds

Returns

- **fv**: Future value, in milliseconds

Description Calculate an end time. Use with [rt.msx](#) to determine when expired.

```
ok = rt.msx(fv) v2.00
```

Check millisecond expiry

Parameters

- **fv**: Future value - the result from [rt.msf](#)

Returns

- **ok**: True if expired

Description Determine when a timeout has expired.

```
b1,b2,ext = rt.mv([id, [smp]])
```

Get millivolt readings

Parameters

- **id**: (Optional) ID. 0 = all three values
1 = Battery 1
2 = Battery 2
3 = External
- **smp**: (Optional) Set to false to prevent sampling and just read the last reading.

Returns

- **b1**: Batt1 Voltage (mV)
- **b2**: Batt2 Voltage (mV)
- **ext**: External Voltage (mV)

Description Get millivolt readings

```
mv, which = rt.mvh() v3.42
```

Get highest millivolt

Parameters None

Returns

- **mv**: Highest input (mV)
- **which**: Which input - 1=Batt1, 2=Batt2, 3=Ext

Description Get the highest millivolt reading.

```
rt.setblf(filename) v1.20
```

Set a BLF file

Parameters

- **filename**: name of file to be loaded into the BLF area

Description Load a file into the BLF area, ready for the boot-loader to burn. Use `pwr.boot()` to reboot in code.

```
c = rt.temp() v1.00
```

Get temperature

Parameters None

Returns

- **c**: Temperature in degrees Celcius

Description Returns the temperature from the LTC2485. Note, this can take 160ms to sample.

rt.us(value) v2.00

Delay microseconds

Parameters • **value**: Delay time in microseconds (1 to 5000)**Returns** None**Description** Delay number of microseconds.

Library `sdi12` : SDI-12 Protocol

The SDI-12 protocol requires a hardware adapter (either internal or external) to provide the SDI-12 signal levels.

The library provides a set of low-level protocol commands, as well as a pair of high-level function (`sdi12.go` and `sdi12.done`) that handle the complete protocol as a state-machine.



Ensure you call `sdi12.set` to setup the Comport before calling other SDI-12 library calls.

Example:

```
dp = ser.dev()
sdi12.set(dp)
dp.pwo(true)

VRS = {{'2I'}, {'2CC'}} -- Get ID and measurements

function QueryExample()
  sdi12.go(dp, VRS)
  while not sdi12.done(t)
  do
    rt.ms(250)
  end
end

QueryExample()
print(VRS[1].v, VRS[2].v[1])
```

```
res = sdi12.ack(cp, dev)
```

Check ACK

Parameters

- `cp`: Comport object
- `dev`: Device number

Returns

- `res`: Result - false or address

Description Use the ACK active command. Returns `false` if the device is not there, else returns the device number.

```
local dn = sdi12.ack(dp, '?') -- Query a single device
```

```
sdi12.brk(cp)
```

Send break

Parameters

- `cp`: Comport object

Returns None

Description Outputs a 12ms break sequence.

```
res = sdi12.chg(cp, dev, devto)
```

Change device address

Parameters

- `cp`: Comport object
- `dev`: Device number
- `devto`: Device address to change to

Returns

- `res`: false = failed; else address of device

Description Attempt to change the device address

```
ok,txt,crc = sdi12.crc(txt)
```

CRC calculation

- Parameters**
- `txt`: String to CRC check
- Returns**
- `ok`: True if the CRC matches
 - `txt`: The text without the CRC
 - `crc`: The CRC string

Description Check, or generate a CRC for SDI-12

```
local _,txt,crc = sdi12.crc('1Testing---') -- calculate CRC
local ok = sdi12.crc('1TestingDU}') -- verify
```

```
ok = sdi12.done(sdit)
```

Check whether sequence complete

- Parameters**
- `sdit`: SDI-12 table structure
- Returns**
- `ok`: True if the state-machine is complete
- Description** Polls the SDI-12 protocol state machine to work through the commands in the table.

```
sdi12.go(cp, sdit)
```

Run SDI-12 sequence

- Parameters**
- `cp`: Comport object
 - `sdit`: SDI-12 command table
- Returns**
- `sdit`: The table, ready to be passed to `sdi12.done`
- Description** Initialises the SDI-12 protocol table and starts the process.

```
idt = sdi12.id(cp, dev)
```

Query ID

- Parameters**
- `cp`: Comport object
 - `dev`: Device number
- Returns**
- `idt`: ID table
`idt[0]`: Address
`idt[1]`: SDI Version
`idt[2]`: Company (8 characters)
`idt[3]`: Model (6 characters)
`idt[4]`: Version (3 characters)
`idt[5]`: Serial number
- Description** Queries the ID of the device.

```
sdi12.mrk(cp)
```

Send mark

- Parameters**
- `cp`: Comport object
- Returns** None
- Description** Waits the required 8.33ms.

```
ok = sdi12.set(cp, [mb])
```

Setup COM port for SDI-12

Parameters

- `cp`: Comport object
- `mb`: Mainboard. Set to false if using internal daughter board.

Returns

- `ok`: True if ok

Description

Configures the Comport object to 1200E7 and configures the RTS for direction.

```
txt = sdi12.t(cp, cmd)
```

Transparent command

Parameters

- `cp`: Comport object
- `cmd`: Command string (the '!' is optional)

Returns

- `txt`: Reply (or false if no reply)

Description

Performs a transparent SDI-12 command.

```
local dp = ser.dev()
local txt = sdi12.t(dp, '2!')
```


SDI-12 Table Structure

The table starts as an indexed set of tables with commands:

```
sdi12 = { {'1C5'}, {'2C1'}, {'2C2'}, {'3CC'} }
```

Once `sdi12.go` and `sdi12.done` have completed, there will be additional fields within the table:

- `sdi12.c` = Comport object
- `sdi12[#].t` = Millisecond alive time when ready. False if the device did not respond
- `sdi12[#].n` = Number of samples (if a measurement command)
- `sdi12[#].v` = Table of results, or a string result

Library `ser` : Serial Ports

```
isen,wasen = ser.bias([enable]) v4.00
```

Set RS485 bias control

Parameters

- `enable`: true to enable bias pull-up

Returns

- `isen`: is bias now enabled
- `wasen`: was bias enabled before change

Description For hardware iss4 (only) this allows script control of the RS485+ pull-up resistor. This allows for better power management if the RS485 bus is live while idling.

```
cp = ser.dev()
cp = ser.eng(prot, [rsmode], [main], [hwflow]) v3.31
```

Get the device COM port object

Parameters

- `prot`: Protocol string, e.g. `19200N8`
- `rsmode`: 232 = RS232, 485 = RS485
- `main`: true for the mainboard / false for daughterboard
- `hwflow`: Hardware flow string. `rx` = RX (RTS) flow, `tx` = TX (CTS) flow, `rxtx` = both

Returns

- `cp`: Com Port object

Description Return a COM port userdata value for the device serial port.

```
local dp = ser.dev()
dp:tx('Hello!')
```

```
cp = ser.eng([try])
cp = ser.eng(prot, [rsmode], [main], [hwflow]) v3.31
```

Get the engineer COM port object

- Parameters**
- **try**: Set to true to try and grab the engineer port. If it is in use by the terminal, then cp will be nil.
 - **prot**: Protocol string, e.g. `19200N8`
 - **rsmode**: 232 = RS232, 485 = RS485
 - **main**: true for the mainboard / false for daughterboard
 - **hwflow**: Hardware flow string. `rx` = RX (RTS) flow, `tx` = TX (CTS) flow, `rxtx` = both

- Returns**
- **cp**: Com Port object

Description Return a COM port userdata value for the engineer serial port.



This implicitly disables the terminal function, unless `try` is true.

```
local dp = ser.eng()
dp:tx('Hello Engineer!')
```

```
ok,code = ser.exp() v3.20
```

Check expansion board

Parameters None

- Returns**
- **ok**: Whether expansion board detected
 - **code**: Code for board. 0=none, 1=CTS, 2=RX, 3=RX+CTS

Description Checks whether an expansion board is present.

- RS485 board returns `true,2`
- LED test board returns `true,3`

```
b = ser.term([en]) v1.00
b = ser.term(prot)
```

Enable or disable terminal over COM port

- Parameters**
- **en**: (optional) boolean value to control serial terminal
 - **prot**: (optional) Enable terminal with the given protocol

- Returns**
- **b**: Boolean result. True if serial terminal enabled.

Description Controls the engineer serial port to terminal logic.

When Lua reboots, the engineer serial port defaults to be used by the terminal.



You must have RX/TX and RTS/CTS connected to work as a terminal! The terminal will not start until RTS is asserted. If RTS is unasserted for 10 seconds the rt.buffer assumes the terminal has disconnected and will de-energise its RS232 port.

To use the engineer serial port for other uses, call:

```
ser.term(false) -- let Lua use the port
ep = ser.eng() -- also lets Lua use the engineer port
--...and the serial port can then be used within Lua.
```

Library `ser` | `cp` : Comport Objects

These functions work on the object returned from `ser.dev()` and `ser.eng()`.

For the sake of documentation, these are shown to refer to 'cp'

`cp:brk(ms)`

Send break

Parameters

- `ms`: Number of milliseconds for the break sequence

Returns None.

Description Send break on the line.

```
nb,ti = cp:cap(file, [ms]) v1.00
nb,ti = cp:cap(fh, [ms])
nb,ti = cp:cap(file/fh, tab)
```

Capture data to file

Parameters

- `file`: Filename to overwrite and create
- `fh`: File handle to write to
- `ms`: Time out to close in milliseconds (default 1000)
- `tab`: Parameter table.
 - `tab.ms` = timeout, in milliseconds
 - `tab.ft` = first byte timeout (default is 2x .ms)
 - `tab.nb` = number of bytes to capture
 - `tab.hw` = set to true to do hardware at 96MHz

Returns

- `nb`: Number of bytes captured
- `ti`: Timed out flag

Description Captures serial bytes directly to a file. Allows faster capturing than doing this directly in Lua.



The write will occur from the the file handle current position.

```
ok,txt,err = cp:cmd([txt], [tab])
```

Handle a command sequence

Parameters

- **txt**: ('\\r\\n') String to send
- **tab**: Table of values:
 - **.pt** = (0) pause time in seconds
 - **.ss** = (\\r\\n) send string (overridden by txt)
 - **.wt** = (5000) wait time, in milliseconds
 - **.dt** = data time, in milliseconds (default is .wt)
 - **.ls** = (>) look for string
 - **.es** = (NIL) error string
 - **.nt** = (5) number of times to try
 - **.tt** = (120) try time, in seconds (.nt must be 0)
 - **.re** = (false) Ignore echo of send string

Returns

- **ok**: True if we got the device's attention
- **txt**: The text returned (without the prompt)
- **err**: If the error string was found, this return value has the string

Description

Send a command string to the device until it responds, or times out.



Text is limited to 2048 bytes.

```
local dp = ser.dev()
local ok,txt = dp:cmd('$\\r\\n', {ls='>', wt=1500, nt=10})

local dp = ser.dev()
local ok = dp:cmd('$\\r\\n') -- wait for '>' prompt with defaults

local dp = ser.dev()
local t = {ss='$\\r\\n$\\r\\n', ls='ERROR\\r\\n>', tt=240}
local ok,txt = dp:cmd(t)
```

```
value = cp:cts()
```

Get CTS line

Parameters None

Returns • **value**: True if the CTS line is asserted.

Description Return the CTS status

```
nb = cp:fifo([size])
```

Set the receive FIFO

Parameters • **size**: Number of bytes for the Serial Port DMA buffer + (min 128)

Returns • **nb**: Number of bytes actually reserved

Description Change the buffering for the serial port. The size defaults to 1024 bytes on Lua reboot.

Memory is taken from the Lua RAM area, split into 4 parts. Two parts at a time are queued with the hardware DMA controller. Therefore, the buffer time for a given FIFO size will = $\text{size} / 2 \times \text{character-time}$. e.g. at 115200 baud, with `cp:fifo(4096)` will give a time of $4096/2 \times 87\mu\text{s} = 178\text{ms}$.

```
ok = cp:lck([ms]) v2.00
```

Lock the COM port

Parameters • **ms**: Time to keep the lock enabled (default = 10000ms, max=30000)

Returns • **ok**: True if we have locked the COM port.

Description Lock the COM port. The lock keeps track of the number of times `cp:lck` was called.

```

dp = ser.dev()

if dp:lck(2000)
then
  -- do stuff short than 2000ms
  dp:ulck()
else
  -- do something else
  -- someone else is using it!
end

```

```
ok = cp:okr()
```

Check whether signals are valid

Parameters None

Returns • **ok**: true if there is a valid signal level on the serial port.

Description Check whether the MAX-chip is detecting a valid signal.

```

local dp = ser.dev()
if dp:okr()
then
  dp:tx('You are connected :-)')
end

```



```
txt,baud,par,db,sb = cp:prot([actual]) v3.31
```

Get the protocol settings

Parameters

- **actual**: Pass `true` to return the actual baud rate

Returns

- **txt**: Protocol string
- **baud**: Baud rate
- **par**: Parity `N`, `E`, `O`, `S`, `M`
- **db**: Databits
- **sb**: Stop bits

Description Return the current protocol settings for the port

```
ison = cp:pwo(ison)
```

Set COM port power output

Parameters

- **ison**: `true` = enable power output
`false` = turn off power output

Returns

- **ison**: `True` if the COM port power output is on

Description Set power output on the COM port.

```
local dp=ser.dev()
dp:pwo(true) -- enable the 3.6V output
```

```
cp:pwrx(ison, [ms])
cp:rxpw(ison, [ms]) v3.31
```

Enable power for receiving

- Parameters**
- **ison**: true = enable receiving
false = disable receiving
 - **ms**: Optional time to keep the RX power enabled. Zero disables the timer. (ison is ignored)

Returns None

Description Set the receive power. Enabling receiving will keep the CPU alive. You **MUST** disable both pwt_x and pwr_x for both ports to enter low power sleep mode.

This is only needed if pwt_x is not called, but you need to receive on a USART

```
cp:pwtx(ison, [ms])
cp:txpw(ison, [ms]) v3.31
```

Enable power for transmitting

- Parameters**
- **ison**: true = enable transmission
false = disables transmission
 - **ms**: Optional time to keep the TX power enabled. Zero disables the timer. (ison is ignored)

Returns None.

Description Set the transmit power.



Enabling transmission on the serial chips requires about 8mA. Only enable the transmit power when absolutely necessary!

```
local dp=ser.dev()
dp:pwtx(0, 250) -- keep TX on for 250ms
-- OR --
dp:pwtx(true) -- use TX
...
dp:pwtx(false) -- finished
```

```

mode = cp:rs([mode]) v1.00
mode = cp:rs([mode], [main]) v1.10
mode,main = cp:rs([mode], [main]) v3.31
mode,main = cp:rs(485) v4.00

```

Set the RS232/RS485 mode

Parameters

- **mode**: 232 or 485
- **main**: true or false

Returns

- **mode**: Current mode - 232 or 485

Description Set the RS232 or RS485 mode of the Device COM Port.

The 'main' value defines whether the main PCB electronics are used for serial communications, or whether the add-on PCB is used.

```

local dp = ser.dev()
dp:rs(485, false) -- use add-on board in RS485 mode
dp:rs(485) -- use add-on board in RS485 (fw v4.00+)
dp:rs(485, true) -- use the main PCB in RS485 mode (inline adapter)
dp:rs(232, true) -- default = RS232 on the main PCB

```

cp:rts(value)

Set RTS Line

Parameters

- **value**: Boolean or integer value

Returns None

Description Set the RTS line. 0/false unasserts; 1/true asserts.



If value is false, this overrides receive flow control. i.e. RTS flow control only operates when value = true.

```
txt,b,to = cp:rx([ms, [nbytes]])
```

Receive data

- Parameters**
- **ms**: Time to wait, in milliseconds. Default = 5ms
 - **nbytes**: Maximum number of bytes to read. Default = 2k. Maximum = 8k
- Returns**
- **txt**: String received
 - **b**: Number of bytes in txt
 - **to**: true if timed-out
- Description** Wait for receive data.

```
nb = cp:rxv([ms]) v1.00
```

Receive and empty data

- Parameters**
- **ms**: Time to wait, in milliseconds. Default = 100ms
- Returns**
- **nb**: Number of bytes flushed
- Description** Flush receive data.

```
cp:rxv(ison)
```

Set receive flow control

- Parameters**
- **ison**: True if RTS should unassert when receive buffers are full.
- Returns** None
- Description** Set RX flow control (uses RTS line).



If ison is true, this also calls :pwtx(true) and :rts(true) Therefore, power will be consumed all the time that :rxv(true) is enabled - because the RS232 chip has to be enabled to assert RTS! And you will see the DeviceTx enabled in the □PMM□ terminal command.

cp:rxto(bits)

Set receive timeout value

Parameters	bits: Number of data bits to consider a receive timeout for the internal DMA controls
Returns	None
Description	Set RX time out in bits

cp:rxtx(ison, [ms]) ^{v3.31}

Enable power for receiving and transmitting

Parameters	ison: true = enable receiving & transmitting false = disable receiving & transmitting ms: Optional time to keep the RX/TX power enabled. Zero disables the timer. (ison is ignored)
Returns	None
Description	Sets both the receive & transmit power. Enabling will keep the CPU alive. You MUST disable for both ports to enter low power sleep mode.

txt,b,ti,fs = cp:rxu(rxut, [ms])

Receive until

Parameters	rxut: Receive Until Table ms: Timeout in milliseconds
Returns	txt: The received data b: Number of bytes in txt ti: true if timed-out fs: Find state: 0=didn't find start -1=found start but no end 1=found end (OK)
Description	Receive Until.

See the following information the 'rxut' table structure.

```
b = cp:rxw()
```

Number of RX bytes waiting

Parameters None

Returns • **b**: Number of bytes waiting

Description Returns number of bytes waiting on port

```
cp:set(txt)
cp:set(baud, [parity, [databits, [stopbits]]])
cp:set(txt, [rsmode], [main], [hwflow]) v3.31
```

Setup COM port

Parameters

- **txt**: Baud rate and protocol string
e.g. '19200n8'
- **stop bits are defined with a suffix**:
'.' = 1, e.g. '19200n8.'; ',' = 1.5, e.g. '19200n8,'; ':' = 2, e.g. '19200n8:'
- **baud**: Baud rate, from 300-115200
- **parity**: Parity, 'n', 'e', 'o', 'm', 's'
- **stopbits**: 1, 1.5, 2
- **rsmode**: 232 = RS232, 485 = RS485
- **main**: true for the mainboard / false for daughterboard
- **hwflow**: Hardware flow string. **rx** = RX (RTS) flow, **tx** = TX (CTS) flow, **rxtx** = both

Returns None

Description Set the protocol.

```
local dp=ser.dev()
dp:set('300n8:')
dp:set(300, 'n', 8, 2) -- same as above
```

b,to,txt = cp:tx(txt, [ms])

Transmit data

Parameters

- **txt**: The text to send
- **ms**: Timeout for sending

Returns

- **b**: Number of bytes sent
- **to**: true if timed-out
- **txt**: Remaining part of txt that is yet to be sent

Description

Send data.



If you don't see data being output, check whether you have transmit flow control and the CTS line status, and whether cp:pwtx has been enabled.

```
local dp=ser.dev()
dp:pwtx(1)
dp:tx('Welcome!')
dp:wtx(100)
dp:pwtx(0)
```

cp:txf(ison)

Set transmit flow control

Parameters

- **ison**: True if CTS should be asserted before data is sent

Returns

None

Description

Set TX flow control (uses CTS line).

cp:txtg(bits)

Set transmit time guard gap

Parameters	bits: Number of data bits to pause between each transmitted byte
Returns	None
Description	Set TX time guard in bits. The time-guard slows down transmission of data for the benefit of slow devices.

```
ok = cp:ulck()
cp:ulck(true) v2.00
```

Unlock the COM port

Parameters	true: This form will unlock all counts
Returns	ok: True if we still owned the COM port (i.e. false indicates the COM port timer has expired, etc)
Description	Unlock the COM port (before the timeout completes). You will need to call cp:ulck the same number of times that cp:lck was called in order to unlock before the timeout.

to = cp:wtx(ms)

Wait for TX to complete

Parameters	ms: Number of milliseconds to wait
Returns	to: true if timed out before data was sent
Description	Wait for transmission to complete.

Receive Until Table `rxut`

As used by the `cp:rxu` method.

- `st = 0`
 - start timeout, in milliseconds, for flushing. If > 0 then the function will wait for a pause until beginning the scanning.
- `ss = ''`
 - start string or table-of-strings
- `bn = 0`
 - minimum number of bytes (the number of bytes before the et/es are started)
- `bx = 256`
 - maximum number of bytes (256 is the largest)
- `et = 0`
 - end timeout in milliseconds.
- `es = ''`
 - end string or table-of-strings.

```
local txt,to = dp:rxu( {ss='begin'} )
```

```
local txt,to = dp:rxu( {ss={'this', 'that'}} ) -- waits for this or that
```

Useful Examples

```
NMEA= {ss='$',es='\r\n',bn=4}
ASCII= {es='\n'} or {es='\r\n'} or {es={'\r','\n'}}
HydrINS= {ss='W',es='\r\n',bn=8}
```

Library `smp` : Sample Data Store

The sample data store library provides a convenient way to handle periodic writes of sample data. The writing is handled 'atomically'. When a sample store reaches a certain size (e.g. 128k), or when it's time to send what has been saved, the sample store is "cut" and the file moved to the /Send directory.

The sample data store is also suitable for collecting bulk data from a device, and choosing to either save or abandon the data (e.g. on a communication drop-out).

By default, the sample data store works with a single file. However, it is possible to use the table structures defined to save to multiple sample stores - for example where there are multiple connected devices, or where selected data needs to be extracted to a secondary file.

Structure of `fTable`

- `.n = 'AppName.data.txt'`
 - Filename of the current file (".tmp" is added automatically)
 - Default is 'AppName.data.txt'
- `.f = nil`
 - Current file object (NIL if closed)
- `.r`
 - String or Function for renaming when cutting the file.
 - If this is nil (or not present), then the default function will use the form "YYYYMMDD-HHMMSS-SerialNo-SiteName-data.txt"
- `.h`
 - String or Function for the header of the file
 - `function(fTable, file)` - return nothing, do the writes within the function
- `.t`
 - String or Function for the tail of the file
 - `function(fTable, file)` - return nothing, do the writes within the function
- `.bx = 131072`
 - Maximum number of bytes for the file
 - 0 means unlimited (not recommended)
 - Default is 128k (131072 bytes)
- `.secret = ''`
 - The prefix for the HMAC-SHA-256 tail function (if used)

- `.cfgkey = 'data'`
 - The name of the key store for the suffix for HMAC-SHA-256 tail function (if used).
 - The HMAC secret becomes `.secret` + value-of(`.cfgkey`)
- `.gz = 0`
 - Set to 1 to gzip the files into the `/Send/` directory.
 - Included in firmware v3.43

Sample Data Store Functions

```
ov = smp.cnt([v])
```

Set or get counter

Parameters

- **v**: New value for the `i.smp.cnt`

Returns

- **ov**: The old value

Description The `i.smp.cnt` value is 'hidden' in C++. The value will be incremented and applied to `i.smp.cnt` on the next call to `smp.save`.

```
txt = smp.csv(valuetable, [inctime])
```

Convert table to CSV text

Parameters

- **valuetable**: table of values to turn into CSV
- **inctime**: optional boolean. Set to false to not include the time as the first value.

Returns

- **txt**: returned string in CSV format

Description Creates a CSV line with UTC as the first field in the form

"YYYY-MM-DD HH:MM:SS" (default timeformat = '%Y-%m-%d %H:%M:%S')

Strings are quoted with quote (default double quotes)

The line ends with lineend (default [CR][LF])

The CSV values are controlled by a global table "`csv`". The `valuetable` can also include a field `.csv` to point to an alternative CSV formatting table.

The `valuetable` can also include a field `.fmtf` to perform formatting function callbacks where more complex formatting is needed.

```
local t = {1, 2, 3, i.cell_csq}
```

```
smp.save( smp.csv(t) )
```

smp.cut([ftable], [killfile])

Cut the sample file

- Parameters**
- **ftable**: optional. If not present, uses global table "ft"
 - **killfile**: optional boolean. If true, the cut file will be discarded.

Returns None

Description Split off the file and rename.



Use the killfile option when downloading bulk data from a device and the process failed.

```
function onJob2() -- do this on a schedule
  smp.cut()
end
```

ftable = smp.save([ftable], txt)

Save sample to flash

- Parameters**
- **ftable**: optional. If not present, uses global table "ft"
 - **txt**: text to save to the sample file

Returns • **ftable**: table that was passed (for convenience)

Description Atomically saves a single block of data to the data store



Also flashes LED to indicate data being stored

```
local t = 'this is a sample\r\n'

smp.save(t)
```

Prebuilt Function Methods

These functions save you coding them in pure Lua.

`smp.r_ft`

Default rename function callback

Description Default rename function

Renames the file according to YYYYMMDDHHMMSS-serial-site_name-filename

(Uses the `fc.utc` and `utc.fn` formatting functions to use the file's creation time)

```
ft = {r=smp.r_ft}
```

`smp.t_crc32`

Tail function for CRC32

Description Tail function that adds CR/LF+CRC32 of the file on close

```
ft = {t=smp.t_crc32}
```

```
ft = {r=smp.r_ft, t=smp.t_md5} - two options
```

smp.t_md5

Tail function for MD5

Description Tail function that adds CR/LF+MD5 of the file on close

```
ft = {t=smp.t_md5}
```

smp.t_sha1 ^{v3.31}

smp.t_sha256 ^{v3.31}

smp.t_hmac256 ^{v3.31}

Tail function for hashes

Description Tail function that adds secure hash to the end of the file on close.

Each of these add the hash name, a colon, a tab, and the hash string.

e.g.

hmac-sha256:
e6:11:f2:e9:53:08:9d:b9:81:d1:f6:aa:a6:4a:9b:81:02:b3:45:b0:9f:1b:68:44:ad:d2:f4:75:ca:81:0f:3c



the HMAC-SHA-256 function uses the secret that is set by `cfg.key.set('data', XXXX)` for the secret.

CSV Formatting Table `csv`

The global table 'csv' defines the formatting for the `smtp.csv` function.

- `csv.s` = `' , '`
 - Separator string override.
- `csv.q` = `'\"'`
 - Quote character for strings.
- `csv.tf` = `'%Y-%m-%d %H:%M:%S'`
 - Time format. If this is blank (i.e. ""), the time is NOT placed at the beginning of the line. You'll need to add the time yourself into the table. Use standard C strftime format strings.
- `csv.le` = `'\r\n'`
 - Line ending string
- `csv.str` = `false`.
 - Set to true to represent all fields as strings

CSV Formatting Callback `.fmtf`

If the table field `.fmtf` is defined, this can point to a function that handles formatting of the table data for CSV output.

The function is passed:

- The original table
- The index value (1-based)
- The value to format

The first return parameter is the value (string or number) for inclusion.

The second return value of the function can control how the CSV quotation is handled. +1 = Use quotation; 0 = use default; -1 = do not quote


```
function myfmt(tbl, idx, v)
  --tbl    = table
  --idx    = index (1-based)
  --v      = value, equivalent to tbl[idx]
  -- Can format any way you like...
  -- return values dictate the CSV style:
  -- v,-1 = do not quote
  -- v,0  = use auto/default quote
  -- v,1  = use quote marks
  return v,0
end

local t = {1,2,3,4}
t.fmtf = myfmt
local txt = util.tcsv(t)
```

Library `sms` : SMS Text Messaging

The SMS library allows for the transmission of 8-bit binary, or 16-bit UCS2 encoded PDU SMS messages, and the reception of 7-bit GSM, 8-bit, and 16-bit UCS2 encoded PDU SMS messages.

Up to 16 SMS messages are queued up while the modem is offline, and the messages are sent when the modem connects. Incoming messages are handled asynchronously, and the `rt.buffer` can be programmed to remain connected to the network for a minimum time - to allow for delivery of the SMS messages.

```
gsm = sms.ag7(txt) v3.00
```

Convert ASCII to GSM-7bit

Parameters • `txt`: ASCII text

Returns • `gsm`: GSM 7-bit encoded

Description Encodes ASCII.

```
tv = sms.dec(PDU) v1.10
```

Decode SMS PDU

Parameters • `PDU`: Binary PDU block

Returns • `tv`: Table of values

- `.num` = originating number
- `.utc` = UTC time
- `.tzm` = Time zone minute offset
- `.pid` = Protocol identifier
- `.dcs` = data coding scheme (0=7-bit GSM, 4=8-bit, 8=UCS2)
- `.msg` = message ASCII
- `.ucs` = raw UCS2 (if encoding is UCS2)

Description Decodes a PDU frame into usable fields.



This only decodes `SMS-DELIVER` message received through `onSMS` callback. It will not decode the output from `sms.enc`. Use web services, e.g. <https://www.smspdu.be> to decode.

```
PDU = sms.enc(msg, [number], [options]) v1.10
```

Encode SMS PDU

Parameters

- **msg**: Message string (max 140 octets)
- **number**: (optional) target phone number. If not present, then `c.cell_sms` is used
- **options**: (optional) table of optional values
 - .mti** = Message Type. 0x01 or 0x11 are acceptable
 - .mr** = Message Reference
 - .vp** = Validity period. e.g. 0xaa = 4-days (See 3GPP TS 123 040 section 9.2.3.12.1)
 - .pid** = Protocol identifier
 - .ucs** = true (sets UCS2 encoding, and `msg` is assumed to be two-byte Unicode) **.bin** = set to `true` to encode as binary

Returns

- **PDU**: Binary output

Description

Encodes a simple ASCII or binary message into a PDU frame suitable for sending.

```
fc,mc = sms.free() v1.10
```

Check free SMS entries

Parameters

None

Returns

- **fc**: Number of free messages available
- **mc**: Number of messages currently queued for transmission.

Description

Returns the message queue counters.

```
txt = sms.g7a(gsm) v3.00
```

Convert GSM-7bit to ASCII

Parameters • `gsm`: GSM 7-bit encoded

Returns • `txt`: ASCII text

Description Decodes to ASCII.

```
sms.go([DoItNow]) v1.10
```

Trigger SMS exchange

Parameters • `DoItNow`: (optional) whether to connect immediately.

Returns None

Description Tells the modem handler that we need to perform SMS transmission on the next connection phase.

DoItNow is assumed to be true (i.e. clear hold off and dial immediately), so 'sms.go(false)' will just arm the modem handler for the next phase.

```
v = sms.rxt([secs]) v1.10
```

Set receive timer

Parameters • `secs`: Number of seconds to stay online (minimum). Set to 0 to disable the receive wait process.

Returns • `v`: Current receive time

Description Sets, or reads, the receive time. This is the minimum number of seconds the modem will remain registered before powering off. The delay is required to provide time for the cellular base-station to send the messages to the rt.buffer.



A value of -1 will enable SMS all the time the modem is online. `sms.rxt(-1)` v3.10

```
ok = sms.snd(PDU, [DoItNow]) v1.10
```

Send PDU frame

- Parameters**
- **PDU**: SMS PDU frame. Use `sms.enc` to encode
 - **DoItNow**: Whether to transmit immediately. Pass the value `false` to simply queue for later
- Returns**
- **ok**: True if the message was queued.
- Description** Queue the PDU frame and connect the modem.

```
ok = sms.txt(msg, [number], [DoItNow]) v1.10
```

Send text

- Parameters**
- **msg**: ASCII message
 - **number**: (optional) target phone number. If not present, uses `c.cell_sms`
 - **DoItNow**: Whether to transmit immediately.
 - **Ascii**: (true) Set to `false` to tell `sms.txt` that `msg` is already UTF-8 encoded
- Returns**
- **ok**: True if the message was queued
- Description** Combined function, similar to `sms.snd(sms.enc(msg, number, DoItNow))`.

However, `msg` can be longer than 140 characters - this function will queue up multiple PDUs.



This sends the message GSM-7-bit characters. For UCS2/UTF-16 you need to use `sms.enc` and `sms.snd`

SMS Received callback

The SMS callback occurs asynchronously at any time while the modem is registered (i.e. not just during the SMS transmission phase).

The service message center may require the rt.buffer to be connected and registered for some time (e.g. at least 30seconds) in order to send any queued SMS messages. Use the 'sms.rxt' function to enable minimum-waiting-times.

```
function onSMS(smsc, pdu) v1.10
```

SMS callback function

Parameters

- **smsc**: The Service Message Center number
- **pdu**: The binary PDU received.

Description

You should use 'sms.dec' to decode the received PDU frame and do something meaningful with them.

e.g.

```
sms.rxt(30) -- set the minimum online time
function onSMS(nu, pdu)
  local t = sms.dec(pdu) -- decode the frame into a table
  local s = t.num .. ' : ' .. t.msg -- make a message
  usb.log('SMS:'..s)
end
```

Library `struct` : Structure Handling

The `struct` module adds C style packing and unpacking (for byte-wide values, not bits).



See:

<http://www.inf.puc-rio.br/~roberto/struct/>

<https://github.com/dubiousjim/luafiveq/blob/master/src/struct.c>

<https://github.com/dubiousjim/luafiveq/blob/master/BENEFITS-LUA>

```
txt = struct.pack(fmt, d1, d2, ...)
```

Pack structure

Parameters

- `fmt`: Structure Format string
- `d1`: Data
- `d2`: ...more data

Returns

- `txt`: Packed data

Description

Returns a string containing the values `d1`, `d2`, etc. packed according to the format string `fmt`.

```
nb = struct.size(fmt)
```

Calculate size of format

Parameters

- `fmt`: Structure Format string

Returns

- `nb`: Number of bytes required

Description

Returns the size of a string formatted according to the format string `fmt`. The format string should contain neither the option `s` nor the option `c0`.

```
d1,d2,... = struct.unpack(fmt, txt, [idx])
```

Unpack structure

Parameters

- `fmt`: Structure Format string
- `txt`: Packed data
- `idx`: Start of reading (default is 1)

Returns

- `d1`: Data 1
- `d2`: Data 2 etc

Description

Returns the values packed in string `s` according to the format strong `fmt`. An optional `i` marks where in `s` to start reading (default is 1). After the read values, this function also returns the index in `s` where it stopped reading, which is also where you should start to read the rest of the string.

Struct format strings

Here are the formatting codes. Initially endianness is set to native and alignment is set to none (!1).

- ">" use big endian
- "<" use little endian
- "!" use machine's native alignment
- "!n" set the current alignment to n (a power of 2)
- " " ignored
- "x" padding zero byte with no corresponding Lua value
- "xn" padding n bytes
- "Xn" padding n align (default to current or native, whichever is smaller)
- "b/B" a signed/unsigned char/byte
- "h/H" a signed/unsigned short (native size)
- "l/L" a signed/unsigned long (native size)
- "i/I" a signed/unsigned int (native size)
- "in/In" a signed/unsigned int with n bytes (a power of 2)
- "f" a float (native size)
- "d" a double (native size)
- "s" a zero-terminated string
- "cn" a sequence of exactly n chars corresponding to a single Lua string. An absent n means 1. The string supplied for packing must have at least n characters; extra characters are ignored.
- "c0" this is like "cn", except that the n is given by other means: When packing, n is the actual length of the supplied string; when unpacking, n is the value of the previous unpacked value (which must be a number). In that case, this previous value is not returned.
- "(" stop capturing values
- ")" start capturing values
- "=" current offset

Struct Examples

To match a C structure

```
/* C demo structure */
struct Str {
    char b;
    int i[4];
};
```

in Linux/gcc/x86 (little-endian, max align 4), use "<!4biiii"

To pack and unpack Pascal-style strings

```
sp = struct.pack("Bc0", string.len(s), s)
```

```
s = struct.unpack("Bc0", sp)
```

In the latter command, the length (read by the element "B") is not returned.

To pack a string in a fixed-width field with 10 characters padded with blanks

```
x = struct.pack("c10", s .. string.rep(" ", 10))
```

Library `twi` : TWI Hardware Interface

```
a1,a2... = twi.qy()
ok = twi.qy(addrf,[addt])
```

Query TWI bus

- Parameters**
- `addrf`: Device address to start
 - `addt`: Device address to finish

- Returns**
- `a1`: Addresses
 - `ok`: If found

Description Queries the TWI bus.

```
res = twi.rd(addr,[txt])
res = twi.rd(addr,num,[v])
```

Read TWI

- Parameters**
- `addr`: Device address
 - `txt`: String to send during the read phase
 - `num`: Number of bytes to read
 - `v`: Output value while reading (default = 0xff)

- Returns**
- `res`: String bytes read. False if fails.

Description Reads from the TWI bus.



Use the Lua `string.bytes(...)` to pull the reply apart.

```
res = twi.wr(add,txt)
res = twi.wr(add,b1,b2,b3)
```

Write TWI

Parameters

- **add**: Device address
- **txt**: String to write b#: Bytes to send

Returns

- **res**: String bytes read. False if fails.

Description

Writes to the TWI device.



You can mix and match the string and byte parameters.

Alternative to the TWI `add` parameter for v3.11+

In firmware 3.11+ the `add` address parameter is more flexible, allowing addressing of memory-type TWI/I2C devices.

When `add` is a table it can specify the device address, the internal address and length, as well as timing values for the transfer.

Simplest forms:

- `{add, iad}` = Address device `add` with internal address `iad` and length 1 (addresses 0-255)
- `{add, iad, ial}` = Address device `add` with internal address `iad` and length `ial`

```
twi.wr( {28, 16}, 'Test')
twi.wr( {28, 16, 1}, 'Test')    -- Same as above
```

Alternatively, a named table index can be used with the following details:

- `dev`= I2C device address
- `addr`= Internal device address
- `alen`= Internal device address length (default 1)
- `tw`= Write time (ms)
- `tr`= Read time (ms). Increasing this allows the device more time to gather the data to send back.
- `tb`= Inter-byte time (ms)
- `ts`= Stop-bit time (ms)



Setting the timing values to zero will disable them.



The timing settings will remain in place for the next call to `twi.rd` or `twi.wr`.

```
twi.wr( {dev=28, addr=16, alen=1}, 'test') -- Same as previous example
```

```
local t1 = {dev=28, addr=16, ts=200}
twi.wr(t1, 'test')                        -- Same with timing added
```

Library `usb` : USB functions

```
usb.log([sub], txt)
```

Send log string to USB

- Parameters**
- `sub`: Optional sub section string.
 - `txt`: Text to output to USB log stream

Returns None

Description Output a string for the USB log. Useful for debugging Lua Apps.

```
n, vp = usb.on()
```

Check USB connected

Parameters None

- Returns**
- `cn`: USB Connected (at the protocol level)
 - `vp`: Voltage present on the USB port

Description Returns information about the USB port.

Library `utc` : Time Functions

```
utc = utc.a2utc(alv) v3.10
```

Convert alive time to UTC

Parameters

- `alv`: Alive time

Returns

- `utc`: Converted UTC time.

Description Uses the `rt.buffer`'s current clock to convert from an alive-time to UTC.

```
txt = utc.fmt(format, [t])
```

Format UTC

Parameters

- `format`: C strftime format string
- `t`: UTC time value (integer). Default is now.

Returns

- `txt`: Formatted time string.

Description Convert time into formatted text.

```
txt = utc.fn([t])
```

Filename UTC

Parameters

- `t`: UTC time value (integer). Default is now.

Returns

- `txt`: String of time in form 'YYYYMMDDHHMMSS'

Description Returns a time string that is suitable for filenames.

utc.job(...)

Description Convenience function - same as job.utc

utc = utc.now() v4.00

Return current UTC

Parameters None

Returns • **utc**: Current UTC time

Description Return the current UTC time.

utcoff,utcon = utc.offt() v4.00

Get power off time

Parameters None

Returns • **utcoff**: UTC time the power was cut
• **utcon**: UTC time the power was restored

Description Returns the off/on time from the RTC chip (if present).

If no RTC (hardware iss2/iss3) this returns `nil, 'No RTC'`


```
ok = utc.ok()
```

Check if UTC is sensible

Parameters none

Returns • `ok`: true if the time is sensible.

Description Returns true if the time is sensible (i.e. > 2016-01-01 00:00:00)

Use this to determine when we've got a good time before continuing with data collection.

```
utc,utcoff,batok = utc.rtc() v4.00
ok = utc.rtc(true) v4.00
```

Get/Set Real Time Clock

Returns • `utc`: Current UTC

- `utcoff`: UTC the power was last turned off
- `batok`: true if the battery is good
- `ok`: true if the sync from CPU to RTC was ok.

Description Returns the current settings from the RTC (Real Time Clock) chip that is on hardware iss4.

When used as `utc.rtc(true)` this syncs the current CPU time back to the RTC chip.

utc.set([utc], [tzm])

Set UTC time and timezone

- Parameters**
- **utc**: UTC time integer, or string
 - **tzm**: optional time zone offset, in minutes

Returns None

Description Sets the real time clock.

```

utc.set('2017-01-01 10:00:00') -- just the time
utc.set(_, 120) -- just the Timezone (+2hr)
utc.set(1234567, 120) -- integer UTC and timezone

```

wd,spm,ds = utc.split(t)

Split UTC time

- Parameters**
- **t**: Time value (integer) or string

- Returns**
- **wd**: UTC time at 00:00:00 on this day
 - **spm**: Seconds past midnight
 - **ds**: Days since 1970-01-01

Description Split the UTC time value into components.

```
spm = utc.spm(t)
```

Get seconds past midnight

Parameters

- **t**: Time value (integer) or string (Default is now.)

Returns

- **spm**: Seconds past midnight

Description Calculate the seconds-past-midnight. i.e. the UTC time value with the day-portion subtracted.

```
txt = utc.txt([t])
```

Text of UTC

Parameters

- **t**: UTC time value (integer). Default is now.

Returns

- **txt**: String of time in form 'YYYY-MM-DD HH:MM:SS'

Description Convert time into text.

```
tzom, tzos = utc.tz()
```

Get time zone

Parameters None

Returns

- **tzom**: Time zone offset in minutes
- **tzos**: Time zone offset in seconds

Description Returns the time-zone offset (as retrieved by the 3G modem).

```
alv = utc.utc2a(utc) v3.10
```

Convert UTC to alive time

Parameters

- `utc`: UTC time.

Returns

- `alv`: Alive time

Description Uses the `rt.buffer`'s current clock to convert from a UTC time to an alive-time.

```
t = utc.val(txt)
```

Convert string to UTC

Parameters

- `txt`: string time to convert

Returns

- `t`: UTC time value

Description Convert text to either full date or seconds-past-midnight

```
local v = utc.val('08:00')
local v = utc.val('2017-06-03 12:15:00')
```

```
wd,ds = utc.wds(t)
```

Get whole days

Parameters

- `t`: Time value (integer) or string

Returns

- `wd`: UTC time at 00:00:00 on this day
- `ds`: Days since 1970-01-01

Description Calculate the whole-day portion of the UTC time value.

Library `util` : Utility functions

```
t = util.add(t1, t2, ...)
```

Add tables

Parameters

- `t1`: Table
- `t2`: Table ...

Returns

- `t`: combined table

Description

Combines two or more tables into a single table. If keys in `t2` are present in `t1`, then `t2` takes precedence.

Effectively uses Lua's "pairs" on each table in turn. Lua equivalent for each table:

```
--[[ Equivalent Lua
t = {}
for k,v in pairs(t1) do t[k]=v
]]--
```

```
t = util.addi(tv1, tv2, ...)
```

Add indexed tables

Parameters

- `tv1`: Table or value
- `tv2`: Table or value ...

Returns

- `t`: Combined table

Description

Combines two or more tables (or values) into a single table.

Effectively uses Lua's "ipairs" on each table in turn. Lua equivalent for each table:

```
--[[ Equivalent Lua
t = {}
for k,v in ipairs(tv1) do table.insert(t, v)
]]--
```

Example:

```

local t1 = {1,2,3}
local t2 = {5,6,7}
local t3 = util.addi(t1, 4, t2, 8, 9)
-- t3 = {1,2,3,4,5,6,7,8,9}

```

```
u = util.au16(a, [be]) v3.00
```

Convert ASCII to UTF-16

Parameters

- **a**: ASCII text
- **be**: Set to true for Big-Endian encoding (default)

Returns

- **u**: UTF-16 encoded text

Description Simple conversion into double character sequences.

```
u = util.au8(a) v3.00
```

Convert ASCII to UTF-8

Parameters

- **a**: ASCII text

Returns

- **u**: UTF-8 encoded text

Description Simple conversion of 0x00-0x7f and 0x80-0xff into double character sequences.

```
t = util.bits(v) v3.44
t = util.bits(v, F00) v3.44
```

Create bits table

- Parameters**
- **v**: Number (up to 32-bits wide)
 - **F00**: `function(bitno,mask,count)` to call
- Returns**
- **t**: Table of t[] = bit
- Description** Creates a table entry for each bit that is set in **n**.

Additionally, can call a function for each bit that's set in **n**.

Example:

```
X = util.bits(42)
-- X[1] = 1
-- X[2] = 3
-- X[3] = 5
```

```
t = util.btab(v, tab, [nils]) v3.44
```

Create table from table with bits

- Parameters**
- **v**: Number (up to 32-bits wide)
 - **tab**: Table to extract from (1-based)
 - **nils**: Include nil values from **tab** (defaults to false)
- Returns**
- **t**: Table of t[] generated from **tab**
- Description** Creates a table from **tab** for each bit that is set in **n**.



Bit zero in **v** corresponds to **tab[1]**

Example:

```
Y = {'zero', 'one', 'two', 'three', 'four', 'five', 'six'}
X = util.btab(42, Y)
-- X[1] = 'one'
-- X[2] = 'three'
-- X[3] = 'five'
```

```
same = util.cmpi(t1, t2) v3.43
```

Compare indexed tables

- Parameters**
- **t1**: Table
 - **t2**: Table

- Returns**
- **same**: true if they are equivalent

Description Compares two indexed tables.



Only compares the first level tables, and not the contents of subtables:

```
xx = {1,2,3}
xy = {1,2,3}
util.cmpi({9,8,7,xx}, {9,8,7,xx}) --> true
util.cmpi({9,8,7,xx}, {9,8,7,xy}) --> false
```

```
cs = util.codesize(MYFUNC) v2.00
cs = util.codesize(nil, true) v2.00
cs = util.codesize(nil, function(cs,fn) print(fn,cs) end) v2.00
```

Estimate Lua codesize

- Parameters**
- **MYFUNC**: Function name to profile
 - **cs**: Number of bytes (for callback)
 - **fn**: Function name (for callback)

- Returns**
- **cs**: Number of bytes

Description Calculate the RAM impact of a function, or all top-level functions.

The format `util.codesize(nil, true)` will print the individual function sizes to the console. Providing a function for the second parameter allows customisation of the results.


```
t,rtxt = util.csvt(txt, [opt]) v3.00
```

Convert CSV text to table

- Parameters**
- **txt**: ASCII line to parse
 - **opt**: Optional table of parameters
 - **.s** = Separator character (,)
 - **.q** = Quote character (")
 - **.str** = Boolean value. If true then all fields treated as string

- Returns**
- **t**: Table of values
 - **rtxt**: Remaining text (for incomplete quotation etc)

Description This function converts an ASCII text line to a Lua table.

```
t1,t2 = util.cuti(t, idx)
```

Cut indexed table

- Parameters**
- **t**: Table that needs cutting in two
 - **idx**: Index of first entry to be in second half

- Returns**
- **t1**: First half
 - **t2**: Second half

Description Cuts a table in half at the 'idx' point.

If idx=1, then t1 will be empty, and t2==t

Idx can be negative to indicate "n-th point from the end", so if idx=-1, then t1==t, and t2 will be empty.

```
local t1 = {1,2,3,4,5,6,7,8,9}
local t2,t3 = util.cuti(t1, 6)
-- t2 = {1,2,3,4,5}
-- t3 = {6,7,8,9}
```

```
v = util.dct(t) v2.00
```

Deep copy table

Parameters

- **t**: Table to deep copy

Returns

- **v**: Resulting copy of table

Description Deep-copies a Lua table. Any changes made to **v** will therefore be independent of the original table **t**.



using the Lua standard `tv=t` will only result in a single table in memory. Any changes to `tv` will affect `t` as well. That's why a "deep copy" is needed to produce a replica of the original table that is independent.



Only the Lua types: LUA_TNIL, LUA_TBOOLEAN, LUA_TNUMBER, LUA_TSTRING, and LUA_TLIGHTUSERDATA are handled.

```
idx,found = util.find(t, v, [func]) v2.00
```

Find

Parameters

- **t**: Table of values
- **v**: Value to find
- **func**: Optional comparison function

Returns

- **idx**: Index (effectively the insertion point)
- **found**: Whether found in the list

Description Binary search through the ordered table.



Uses the first entry of the table to determine whether string or number.

```

local idx,found = util.find( {1,2,3,4,5}, 3) -- > 3, true
local idx,found = util.find( {1,2,3,4,5}, 3.1) -- > 3,false
local idx,found = util.find( {'apple','orange','pear'}, 'orang') -- > 2,false
local idx,found = util.find( {'apple','orange','pear'}, 'oranges') -- > 3,false
local idx,found = util.find( {'apple','orange','pear'}, 'oranges',
    function compare(a,b) ①
        if a<b
        then return -1
        elseif a==b
        then return 0
        else return 1
        end) -- > 3,false

```

① Comparison function

```
n = util.iter(F00,start,[depth]) v2.00
```

Iterate table

Parameters

- **F00**: Function to call. `n = F00(o)` is the prototype.
- **start**: Starting object or table
- **depth**: (optional) table depth. Default = 3

Returns

- **n**: Count or total

Description

The function **F00** can return a value that is summed to form the total **n**.

```
kvco = util.kvc(nb,ni) v2.00
```

Create key-value container object

- Parameters**
- **nb**: Number of bytes for the data
 - **ni**: Number of index entries

- Returns**
- **kvco**: KVC object

Description Return a KVC object.

See [util.kvc](#)

```
t = util.remi(t, idx, [last])
```

Remove indexed table

- Parameters**
- **t**: Table that needs cutting in two
 - **idx**: Entry to remove, or first entry to remove
 - **last**: Optional last entry to remove

- Returns**
- **t**: Resulting table

Description Returns a table that is 't' minus the entry at idx, or minus entries between idx and last (inclusive)

```
local t1 = {1,2,3,4,5,6,7,8,9}
local t2 = util.remi(t1, 6)
-- t2 = {1,2,3,4,5,7,8,9} (#6 removed)
local t3 = util.remi(t1, 6, -2)
-- t3 = {1,2,3,4,5,9} (#6-8 removed)
```

```
txt = util.tcsv(t) v3.42
```

Convert table to CSV text

Parameters

- **t**: Table of values
- **.csv** = CSV parameter table
- **.fmtf** = Format function(t, idx, v)

Returns

- **txt**: ASCII line

Description Converts a table of values to a formatted CSV output.

If present **.csv** can point to a table of alternative CSV format parameters. Otherwise the global **csv** table will be used. See **smp.csv** table definitions.

See the **.fmtf** table information for formatting function callback.

```
ty = util.tia(ta, tb) v2.00
```

Append index table

Parameters

- **ta**: Table A
- **tb**: Table B

Returns

- **ty**: Resultant table

Description Append two indexed tables.

```
t = rt.tia({1,2,3},{ 'A', 'B' })
---> t = {1,2,3, 'A', 'B' }
```

```
ty = util.tio(ta, tb) v2.00
```

OR indexed tables

- Parameters**
- `ta`: Table A
 - `tb`: Table B
- Returns**
- `ty`: Resultant table
- Description** OR together two indexed tables.



Useful for applying defaults to a table.

```
t = util.tio({1,_,3},{'A','B','C'})
--> t = {1,'B',3}
```

```
u = util.u16u8(u, [be]) v3.00
```

Convert UTF-16 to UTF-8

- Parameters**
- `u`: UTF-16 encoded text
 - `be`: Set to true for Big-Endian encoding (default)
- Returns**
- `u`: UTF-8 text
- Description** Converts Unicode 16-bit characters with either big or little endianness to UTF-8 encoding.

```
a,ok = util.u8a(u, [d]) v3.00
```

Convert UTF-8 to ASCII

- Parameters**
- **u**: UTF-8 encoded text
 - **d**: Text to replace undefined characters (default `?`)

- Returns**
- **a**: ASCII text
 - **ok**: True if the decoding was ok

Description Converts Unicode code points 0x0000 to 0x00ff into ASCII.

Invalid UTF-8 code points will return `ok` as false.

```
u,ok = util.u8u16(u, [be]) v3.00
```

Convert UTF-8 to UTF-16

- Parameters**
- **u**: UTF-8 encoded text
 - **be**: Set to true for Big-Endian encoding (default)

- Returns**
- **u**: UTF-16 text
 - **ok**: True if the decoding was ok

Description Converts UTF-8 to Unicode 16-bit characters with either big or little endianness.

```
v = util.use(fn, [force]) v3.42
u1,u2,... = util.use() v3.42
```

Include Lua library

Parameters	• fn: Filename of Lua file • force: Force re-load (set to <code>true</code>)
Returns	• v: true if found and loaded; false if already used; nil,error on error • u1,u2...: Uses file name entries
Description	Allows load and run of Lua libraries. Similar to Lua's <code>require</code> but designed for embedded use.

`fn` does not need to have a full path. `.lua` is appended if no extension is found. `/Lua` is prefixed if `fn` does not start with `/` (allows for relative module names)

```
txt = util.zdec(txt) v4.00
```

Z-base-32 decode

Parameters	• txt: z-base-32 encoded string
Returns	• txt: Decoded string
Description	Convert a z-base-32 string to string/binary data.


```
txt = util.zenc(txt) v4.00
```

Z-base-32 encode

- Parameters**
- `txt`: Text string to convert
- Returns**
- `txt`: z-base-32 encoded string
- Description** Convert a string, or binary string, to z-base-32

See: <https://github.com/cryptii/cryptii/blob/main/src/Encoder/Base32.js>

LCD and User Interface Library Extensions

Library `key` : Keyboard Library

The keyboard library provides scripting support for up to 8 push-to-make switches that are connected to an IO-expander on the I2C bus.

```
bf,td = key.get()
v,td = key.get(keyno)
```

Get key state

- Parameters**
- `keyno`: Which key to check (1-8)
- Returns**
- `v`: (boolean) whether the key is pressed (if `keyno` is provided)
 - `bf`: (number) bit-field of the pressed keys (D0 = key #1)
 - `td`: Time keys have been pressed (milliseconds)

Description Get the state of an individual key, or all keys as a bitfield.

```
v = key.ign(bf)
v = key.ign(keyno, ign)
```

Ignore keys

- Parameters**
- `bf`: Number bitfield. If a bit is set then the key is ignored.
 - `keyno`: Key number (1-8)
 - `inv`: Whether to ignore (boolean)

Returns

- `v`: Bitfield of ignore field.

Description Set which keys to ignore.



Removed in v3.00

```
ok = key.inj(keyno) v2.00
```

Inject keystroke

- Parameters**
- **keyno**: Key number 1-8
- Returns**
- **ok**: True if injected
- Description** Injects a keystroke into the keyboard buffer.

```
v = key.inv(bf)
v = key.inv(keyno, inv)
```

Set key inversion

- Parameters**
- **bf**: Number bitfield. If a bit is set then the key is inverted.
 - **keyno**: Key number (1-8)
 - **inv**: Whether to invert (boolean)
- Returns**
- **v**: Bitfield of inversion field.
- Description** Set which keys to invert.

```
key.kre(keyno, en)
```

Set key repeat enable

- Parameters**
- **keyno**: Key number 1-8
 - **en**: Whether to act on key repeats (true=repeat)
- Returns** None
- Description** Set the Key Repeat Enable. If an individual key has repeat enabled, then there will be a callback at the repeat interval all the time the key is kept pressed.

```
rt,it = key.kri(rt,[it])
```

Set keyboard repeat idle

- Parameters**
- **rt**: Repeat time (milliseconds)
 - **it**: Idle time (milliseconds)
- Returns**
- **rt**: Repeat time (milliseconds) Default = 400ms
 - **it**: Idle time (milliseconds) Default = 10000ms (10sec)
- Description** Set the Key Repeat Idle time and repeat interval time.

```
ok = key.ok()
```

Check keyboard hardware

- Parameters** None
- Returns**
- **ok**: True if the keyboard hardware was found.
- Description** Check whether the keyboard is connected.

Key callback functions

```
function onKey#(keyno, isdown, t)
```

Individual key callback

- Parameters**
- **keyno**: Key number 1-8.
 - **isdown**: True if the key is pressed.
 - **t**: Time, in milliseconds, that the key has been held down for.

Returns None

Description Callback for an individual key. Called after the [onKey](#) general callback.

```
function onKey2(kn, id, td)
  -- do something
end
```

```
function onKey(keyno, isdown, t)
```

Key global callback

- Parameters**
- **keyno**: Key number 1-8. Or -1 for the Idle callback.
 - **isdown**: True if the key is pressed.
 - **t**: Time, in milliseconds, that the key has been held down for.

Returns None

Description Called when any key is pressed. This function is called first.

When the keyboard is idle, the [onKey](#) function is called with keyno=0 (allowing for a single function to handle all events). The [onKeyIdle](#) callback (if defined) is called after this function.

function onKeyId()

Keyboard idle callback

Parameters	None
Returns	None
Description	Called when the keyboard has been idle.

See [key.kri](#) - Idle Time.

Library `lcd` : LCD Library



The LCD library was included in firmware v2.00. None of the LCD functions are in previous releases.

The `lcd` library provides low-level functions for displaying text, icons, and graphics.

Two LCD boards are supported (both from Electronic Assembly <http://www.lcd-module.com>):

- EA DOGXL-160
 - 160x104 monochrome
- EA DOGL-128 or EA DOGM-128
 - 128x64 monochrome

The LCD board is an optional attachment that also includes support for 8 keyboard switches.

See also the `key` library.



The LCD library will not function unless the keyboard library detects the keyboard chip!



The coordinate system has (0,0) as the top-left pixel.



The LCD library is a modified and improved version of the U8G2 library.

Hardware control

```
ison = lcd.act()
```

Check if the LCD is on

Parameters None

Returns • `ison`: true if the LCD is currently on

Description Find out if the LCD is powered.

```
lcd.c(col)
```

Set LCD draw colour

Parameters • `col`: Colour - 0=WHITE, 1=BLACK

Returns None

Description Set the current drawing colour.

```
lcd.ers()
```

Erase the LCD buffer

Parameters None

Returns None

Description Erases the LCD buffer.

lcd.go([ison])

Display the current LCD buffer

Parameters • `ison`: Whether to power on the LCD too (default=true)

Returns None

Description Display the current LCD buffer.

```
w,h = lcd.size() v4.00  
w,h = lcd.gss()
```

Get the LCD screen size

Parameters None

Returns • `w`: Width in pixels
 • `h`: Height in pixels

Description Get the LCD screen size dimensions.



X-coordinates will be from 0 to (`w-1`); Y-coordinates will be from 0 to (`h-1`)

v = lcd.idle(v)

Set or get LCD idle time

Parameters • `v`: Idle time, in milliseconds

Returns • `v`: Idle time, in milliseconds

Description Set, or get, the LCD idle timeout.



Limited to >1500ms and <120000ms (2 minutes)

```
ok = lcd.init(width, [rot], [eng])
```

Initialise LCD hardware

- Parameters**
- **width**: Pixel width of LCD, either 160 or 128 (default=160)
 - **rot**: Optional rotation 0, 90, 180, 270 (default=0)
 - **eng**: True to route LCD to engineer port

Description Initialise the LCD hardware.



RAM for the LCD buffer is taken from the Lua working RAM. The DOGXL160 requires 2k, the DOGL128 requires 1k.



DOGL128 should have KEY6+KEY7 linked. The dimensions of the LCD are detected in fw v4.00+

```
lcd.off()
```

Turn off the LCD

Parameters None

Returns None

Description Turns off the LCD.

ok = lcd.ok()

Find out if LCD hardware present

Parameters None

Returns • **ok**: true if the LCD hardware was found

Description Check whether the LCD and keyboard is connected.

lcd.on(ison)

Control LCD state

Parameters • **ison**: true if enabled

Returns None

Description Turn the LCD on, or off.



The LCD consumes about 700uA while on.

lcd.save(filename)

Save the LCD to a bitmap

Parameters • **filename**: File to save the bitmap to.

Returns None

Description Save an NetPBM file **P4** style bitmap of the current LCD frame buffer.



Use the extension **.pbm** to be compatible with Photoshop, etc.

lcd.sdc(c)

Set LCD display contrast

Parameters • **c**: Contrast, 0-255

Returns None

Description Set the display contrast.

Drawing functions

```
lcd.box(ft, x,y, w,h, [r])
```

Draw LCD box

Parameters

- **ft**: Fill type:
 - 'f' = filled
 - 'h' = hollow
 - 'r' = rounded hollow
- **x,y**: Starting point
- **w**: Width in pixels
- **h**: Height in pixels
- **r**: Radius, in pixels

Returns

None

Description

Draw a box, or a block.

```
lcd.cir(ft, seg, x,y, rx, [ry])
```

Draw LCD circle or ellipse

Parameters

- **ft**: Fill type
 - 'f' = filled
 - 'h' = hollow
- **seg**: Which segments to draw:
 - '*' or nil = all
 - 'L' = upper left
 - 'R' = upper right
 - 'l' = lower left
 - 'r' = lower right
- **x,y**: Centre
- **rx**: Radius for width
- **rh**: Radius for height (=rx if not present)

Returns

None

Description Draw a circle, or ellipse, or segment.

```
lcd.l(x1,y1, x2,y2)
```

Draw LCD line

Parameters

- `x1,y1`: Starting point
- `x2,y2`: End coordinate

Returns None

Description Draws a line from (x1,y1) to (x2,y2) in the current drawing colour.

```
lcd.lh(x, y, w)
```

Draw LCD horizontal line

Parameters

- `x,y`: Starting point
- `w`: Width of line, in pixels

Returns None

Description Draws a horizontal line in the current drawing colour.

```
lcd.lv(x, y, h)
```

Draw LCD vertical line

Parameters

- `x,y`: Starting point
- `h`: Height of line, in pixels

Returns None

Description Draws a vertical line in the current drawing colour.

lcd.p(x, y)

Draw LCD pixel

- Parameters**
- **x**: X coordinate. 0-159
 - **y**: Y coordinate. 0-103

Returns None

Description Set a pixel to the current drawing colour.

ok,x,y,w,h = lcd.pbm(x,y, filename, [invert])

Draw LCD PBM bitmap from file

- Parameters**
- **x,y**: Position (top left corner)
 - **filename**: Name of the PBM file
 - **invert**: Whether to invert drawing

- Returns**
- **ok**: True if loaded and drawn
 - **x,y**: New position (bottom right corner)
 - **w,h**: Width and height of the bitmap

Description Draw a NetPBM [P4](#) binary bitmap.



Photoshop refers to this as "Portable Bit Map", and saves as a [.pbm](#) file.

lcd.tri(ft, x1,y1, x2,y2, x3,y3)

Draw LCD triangle

Parameters

- **ft**: Fill type
 - 'f' = filled
 - 'h' = hollow
- **x1,y1**: Start corner
- **x2,y2**: Middle corner
- **x3,y3**: End corner

Returns

None

Description

Draw a triangle.

lcd.xbm(x,y, w,h, txt)

Draw LCD XBM bitmap string

Parameters

- **x,y**: Position
- **w**: Bitmap width
- **h**: Bitmap height
- **txt**: XBM encoded string

Returns

None

Description

Draw a bitmap on the LCD frame buffer.

Font and Text functions

```
fa,fd = lcd.f(name)
```

Set the LCD font

Parameters • `name`: Font name string

Returns • `fa`: Font ascent
 • `fd`: Font descent

Description Choose a font to write with.

```
lcd.fr(rot)
```

Set LCD font rotation

Parameters • `rot`: Rotation angle, 0, 90, 180, 270

Returns None

Description Set the font rotation angle.

```
lcd.ft(tr)
```

Set LCD font transparency

Parameters • `ft`: Whether transparent font. 1=transparent; 0=opaque

Returns None

Description Set the font drawing method.

```
fa,fd = lcd.gad()
```

Get the LCD ascent and descent metrics

Parameters None

Returns

- **fa**: Font ascent
- **fd**: Font descent

Description Get the current font's ascent and descent metrics.

```
lcd.gly(x,y, gn)
```

Draw LCD glyph

Parameters

- **x,y**: Coordinate
- **gn**: Glyph number

Returns None

Description Draw a Unicode glyph from the current font.

```
w,fa,fd = lcd.gtw(txt)
```

Get LCD text width

Parameters

- **txt**: UTF-8 text to measure

Returns

- **w**: Width in pixels
- **fa**: Font ascent
- **fd**: Font descent

Description Measure the width of a UTF-8 string in the currently selected font.

```
x,y,w,h = lcd.t(x,y, txt)
```

Draw LCD text

- Parameters**
- `x,y`: Coordinate for the baseline
 - `txt`: UTF-8 text to print

- Returns**
- `x`: Updated x position
 - `y`: Updated y position
 - `w`: Width of text
 - `h`: Height of text

Description Draw the `txt` at the position `x,y` with the currently selected font.



You can throw away all, or some, of the return values.

```
local x = 10
x = lcd.t(x, 10, 'Hello ')
x = lcd.t(x, 10, 'world!')

local y, _
y = 10
_, y = lcd.t(10, y, 'Line 1')
_, y = lcd.t(10, y, 'Line 2')
```

Font names

Single size

- "gt" Glasstown (modified u8g2_font_glasstown_nbp_t_all)

X11 fonts

- "4x6"
- "5x8"
- "6x10"
- "6x12"
- "7x13"
- "7x13b" Bold
- "7x13e" Emphasised
- "9x15"
- "10x20"

ProFont

- "pf10"
- "pf11"
- "pf12"
- "pf15"
- "pf17"
- "pf22"
- "pf29"

Icon and special fonts

- "batt6"
- "cell6"
- "i1" OpenIconic x 1
- "i2" OpenIconic x 2
- "i4" OpenIconic x 4
- "sk7" Soft key icons
- "uf" Unifont symbols

Library **ui** : User Interface LCD Library



The LCD library was included in firmware v2.00. None of the LCD functions are in previous releases.

The **ui** library provides high level user interface functions for keyboard and LCD operation.

See also: **lcd** and **key** libraries.

The UI library is designed to be extremely memory efficient. The LCD frame buffer will take just over 2k (for an EA-DOG160XL LCD). Only the currently viewed UI element will be loaded into Lua's RAM - with everything else existing in comments within the Lua App file.

The UI will enter power save after a brief period, and can be instantly woken with a press of a key.



The UI is called from the context of the LuaEvents task. Consequently, ensure any functions do not block - otherwise you will hold up the rest of the system. (Best to trigger flags that are then executed within the Lua Loop task.)

Boilerplate code

The code here shows the smallest UI code required. Additional information is provided in the following chapters.

```
--[[Mmain1]] ①
Nsample
[Main menu]
Welcome!
Alive {i.pwr_alv} sec
]]--

--[[Nsample]] ②
my_value
[Choose a value]
Something {2} units
99
-99
]]--

lcd.init(160) ③
ui.go()
ui.push('Mmain')
```

- ① The UI element for page 1 of `Mmain`
- ② The UI element for a number editor `Nsample`, with a 2 decimal-place value between -99 and +99
- ③ The initialisation sequence for an EA-DOG160XL display

UI Element : Menu

The prefix **M** is for menu. The UI core will try and access **Mname1** to begin with, and pressing the **DOWN** button will sequence through the digits until no more are available in the file.

```
--[[Mname1] ①
Next      ②
  (OR)
/execfunc ②
  (OR)
.         ③
[Title]   ④
CommentText ⑤
Label {%.1f,my_value} ⑥
Label-{c.lua_app} {%.1f,my_value}} Units ⑦
Label {*%.1f,my_value} Units ⑧
/funcname ⑨
]]-- ⑩
```

- ① The first menu item for level **name**
- ② The UI element to push, or Lua function to call, when **SET** is pressed.
- ③ Use a dot when there is no **Next** or function, and there is no **[Title]**
- ④ (optional) Title string
- ⑤ A line that is shown as a comment
- ⑥ A line that is shown with a label and right-justified small number
- ⑦ A line with label, formatted value, and units. Note the double **{** to allow **rt.exp** expansion of Label and Units
- ⑧ The ***** denotes a line with big formatted value.
- ⑨ A Lua function hook for the line
- ⑩ The end of the UI element.

Example

Menu with: * static text on page 1 that leads to a number editor * static text on page 2 that leads to a list picker * dynamic text on page 3 showing two values


```
--[[Mmain1]
Nvalue
[Edit value]
Press SET/RIGHT to edit the value.
]]--

--[[Mmain2]
Litems
[Choose list]
Press SET/RIGHT to choose from the list
]]--

--[[Mmain3]
.
[Display only]
Alive {*%.0f,i.pwr_alv}
Serial {i.rt_sn}
]]--
```

UI Element : Values with editing

A UI menu that has no [Next](#) or [/execfunc](#) can be used to have in-place-editing values, saving several separate pages and glue-code.



If you accidentally assign a [Next](#) or [/execfunc](#) then the UI will never enter the edit mode. (Even if you have assigned the in-place editing `write;dps;min;max` values)

```
--[[Mname2] ①
.           ②
[Title]     ③
Label {%1f,my_value;my_value;1;-10} ④
Label {%1f,my_value;/WriteFunc;1;-100;200} Units ⑤
Label {*%1f,my_value} Units ⑥
Jump {;>Ltest} ⑦
Call {;/CallFunc} ⑧
]]-- ⑨
```

- ① The second menu item for level [name](#)
- ② Use a dot when there is no [Next](#) or function, and there is no [\[Title\]](#)
- ③ (optional) Title string
- ④ Edit value: write to [my_value](#); 1 decimal place; minimum=-10
- ⑤ Edit value: write with function [WriteFunc\(VALUE\)](#); 1 decimal place; minimum=-100; maximum=200
- ⑥ View only big value.
- ⑦ A label that will jump to the UI element [Ltest](#)
- ⑧ Lua call to [CallFunc\(uiname, uilevel\)](#)
- ⑨ The end of the UI element.



Using negative decimal places will increment/decrement in tens, hundreds, etc. e.g. -3 will use 1000 as a delta.

UI Element: Number Editor

The prefix **N** is for number-editor. The number editor can show an optional title, and specify optional maximum and minimum values.

```
--[[Nname] ①
Read ②
  (OR)
Read,Write ②
  (OR)
/readfunc,/writefunc ②
[Title] ③
Label {dp} Units ④
Maximum ⑤
Minimum ⑥
]]-- ⑦
```

- ① The UI element name
- ② Either read; read+write variables. Can also be a Lua function name prefixed with /
- ③ (optional) Title string
- ④ Label, decimal places, and Units
- ⑤ (optional) Maximum value
- ⑥ (optional) Minimum value
- ⑦ End of the UI element.

Example 1

Number editor that reads `my_val` and shows to 1 decimal place. When saved, the variable `my_val` is updated:

```
--[[None]
my_val
[Edit my_val]
Value {1} cm
]]--
```

Example 2

Number editor that reads `my_val` and shows to 1 decimal place. When saved, the variable `other_val` is updated:

```
--[[Ntwo]
my_val,other_val
[Edit my_val, save other_val]
Value {1} cm
]]--
```

Example 3

Number editor that uses a function `getmyval` to obtain the value. When saved, the function ``setmyval`` is called to store the value:

```
--[[Nthree]
/getmyval,/setmyval
[Edit with max + min]
Value {1} cm
100
0
]]--

function getmyval()
    return my_val
end

function setmyval(v)
    my_val = v
end
```

UI Element: List Picker

The prefix **L** is for the list-picker.

```
--[[Lname] ①
Read ②
  (OR)
Read,Write ②
  (OR)
/readfunc,/writefunc ②
[Title] ③
/listfunc ④
  (OR)
item1 ④
item2
item3
etc
]]-- ⑤
```

- ① The UI element name
- ② Either read; read+write variables. Can also be a Lua function name prefixed with /
- ③ (optional) Title string
- ④ Either a Lua function name for a software list, or the list of items
- ⑤ End of the UI element.

Example1

List picker that reads and writes to the variable `yn_answer`:

```
--[[Lyesno]
yn_answer
[Decide]
No
Yes
]]--
```

Example 2

List picker that uses functions `getyn` and `setyn` to read and write the values. Also uses a Lua function to generate the list contents - in this example the list is taken from a Lua table.

```
--[[Lsoft]
/getyn,/setyn
[Decide]
/ynlist
]]--

thelist = {"No", "Yes"}

function getyn()
    return yn_answer
end

function setyn(v)
    yn_answer = v
end

function ynlist(n)
    if not n then return #thelist end
    return thelist[n]
end
```

UI Element: Settings and Hooks

The UI elements can be expanded with additional modifiers. Place these before the `[Title]` value.

:hook:funcname

Draw hook

Parameters • **funcname**: Name of a Lua function

Description Add a draw hook that is called at the end of each LCD update.

You can use this to draw a bitmap for the selected list index, etc.

Example draw hook:

```
function MyDrawHook(name,menuidx,value)
  --name      = UI name, e.g. 'Mtest1'
  --menuidx   = UI index
  --value     = value in the UI.
  --          Menu or list index, or Number
end

--[Mname1] ①
/dothis    ②
:hook:MyDrawHook ③
:key:GO!   ④
:skms:2000 ⑤
[Title]    ⑥
Label {%.1f,my_value}
Label {%.1f,my_value} Units
L-{c.somelabel} {{"%.1f,my_value"}} {c.someunits}
]]-- ⑧
```

- ① The UI name
- ② Lua function call
- ③ Hook to display
- ④ The SET button will show `GO!`
- ⑤ You must hold the SET button down for 2 seconds

:sket:text

SET key override

Parameters

- **text**: String to use for the SET key

Description Override the key text. (See the example in :hook:funcname)

:skms:time

SET key milliseconds

Parameters

- **time**: The time delay, in milliseconds

Description Add a delay to the SET key. (See the example in :hook:funcname)

ui._bar

Default uiBar handler

Description Default [uiBar](#) handler for the UI.

ui._key

Default onKey handler

Description Default [onKey](#) handler for the UI. This has been made available so the default handler can be intercepted and overridden.

ui._cell

Default uiCell handler

Description Default [uiCell](#) handler for the UI. Draws a large timer and big icons for the status of the cellular modem.

ui.box(title, [y])

Draw UI box

Parameters

- [title](#): Title text
- [y](#): Top y-coordinate

Description Draws the optional title box, and main box (according to the UI preferences set).

`ui.go()`

Start the UI

Parameters None

Description Start the UI. Applies the defaults, hooks the [onKey](#) callback, and begins the UI.

`ui.key(kn,txt,[tfon])`

Draw soft key

Parameters

- [kn](#): Key number, 1 to 4
- [txt](#): String to display
- [tfon](#): True if a text font should be used

Returns None

Description Draw a soft-key (useful in the display hook callback)

`name = ui.pop()`

Pop UI element

Parameters None

Returns

- [name](#): Element name

Description Pops a UI element off the stack.

ui.popa([all])

Pop all UI

Parameters • [all](#): True to pop everything, false to pop everything except the top-level

Description Pops all UI elements off the stack.

ui.push(name)

Push UI element

Parameters • [name](#): Element name

Description Push a UI element onto the stack. The element is loaded from either the current Lua App, or to filename set with [ui.set.i8n](#).

ui.ssk(txt)

SET key icon

Parameters • [txt](#): String to use for the [SET](#) key

Returns None

Description Set the text or icon for the [SET](#) key.

name,val,index = ui.top()

Get the top UI element

Parameters None

Returns • [name](#): Name of the UI element
• [val](#): Value of the stack
• [index](#): Stack pointer for the element

Description Accesses the top element on the UI stack.

```
txt = ui.txt()
```

Get UI element text

Parameters None

Returns • `txt`: The text loaded by `app.txt`

Description Provides access to the full text within the UI element.

```
y = ui.val(y, label, value, units, [big])
```

Show values on LCD

Parameters • `y`: Y-position to draw the line

- `label`: The label text
- `value`: The value text
- `units`: The units text
- `big`: True to show large value text

Description Draws a single row of label/value/units.

The formatting is different depending on the contents of `value` and `units`.

```
ui.yes()
ui.yes(txt,[fontname]) v3.00
ui.yes(func) v3.00
```

Show YES tick

Parameters • `txt`: Text for the popup

- `fontname`: Font to use (default 'i4')
- `func`: Function to call for drawing

Returns None

Description Show the UI confirmation 'tick'.

If `func` is used, then the function can perform LCD drawing functions. The screen will already be blank on entry, and be drawn on exit.

Preference settings

The preferences can control the look and feel of the UI.

```
ui.set.bklt(enable) v4.00
```

Control LCD backlight

Parameters • `enable`: true to enable backlight

Returns None

Description Specify the backlight control.

KEY8 is controlled as an output to enable backlight. It is automatically turned on/off from the UI.

Calling `ui.set.bklt(false)` disables this control.

```
tc,bc,kc = ui.set.c(tc,bc,kc)
```

Set UI colours

Parameters • `tc`: Title colour
 • `bc`: User box colour
 • `kc`: Softkey colour

Returns • `tc`: Title colour
 • `bc`: User box colour
 • `kc`: Softkey colour

Description Control the colour of UI elements.

```
fn = ui.set.i8n(filename)
```

Set UI filename

Parameters

- `filename`: Filename to load UI elements from

Returns

- `fn`: Current filename

Description Set the internationalisation file. By default the UI elements are loaded from the current Lua App file.



Make sure the chosen `filename` contains **all** the elements referred to in the App!

```
v = ui.set.idle([n])
```

Set UI idle time

Parameters

- `n`: Number of seconds the UI should show when idle

Returns

- `v`: Current value

Description Sets, or gets, the current UI idle time.



Use a time of -1 to show the UI always.

```
ui.set.idle(-1) -- Show always
```

ui.set.km(bk,uk,dk,sk)

Set UI key mapping

Parameters

- **bk**: Back key number (1)
- **uk**: Up key number (2)
- **dk**: Down key number (3)
- **sk**: Set key number (4)

Description

Override the Key Mapping. The soft-keys will be rearranged to match the key assignments.



Key 1 is on the left of the display; Key 4 is on the far right of the display.

```
ui.set.km(4,3,2,1) -- reverse
```

```
ui.set.km(1,3,2,4) -- swap UP / DOWN
```

kr = ui.set.kr(kr)

Set UI key repeat

Parameters

- **kr**: Key repeat time

Returns

- **kr**: Key repeat time

Description

Set the Key Repeat value.

ui.set.onebtn(enable) v4.00

Parameters

- **enable**: true to use single button UI

Returns None

Description When `ui.set.onebtn(true)` is called, only KEY3 is required. This is suitable for display-only UI (i.e. not for on-screen editing functions).



The DOGL128 defaults to requiring a single button interface.

tr,br,kr = ui.set.r(tr,br,kr)

Set UI rounded corners

Parameters

- **tr**: Title round value (pixels)
- **br**: User box round value (pixels)
- **kr**: Softkey round value (pixels)

Returns

- **tr**: Title round value (pixels)
- **br**: User box round value (pixels)
- **kr**: Softkey round value (pixels)

Description Control whether UI elements have rounded corners.

ui.set.showsk(enable) v4.00

Parameters • **enable**: true to enable soft-keys

Returns None

Description Control whether soft-keys appear on the LCD.



The DOGL128 defaults to having no soft-keys

ui.set.ui(xt, ft)

Set UI font positions

Parameters

- **xt**: Table of X-positions
- **xt.t**: X-pos for Text
- **xt.l**: X-pos for Labels
- **xt.u**: X-pos for Units
- **xt.vl**: X-pos for left justified Values (no units)
- **xt.vr**: X-pos for right justified Values (with units)
- **ft**: Table of Font names
- **ft.t**: Font for Text
- **ft.l**: Font for Labels
- **ft.u**: Font for Units
- **ft.s**: Font for Value small size
- **ft.m**: Font for Value medium size
- **ft.b**: Font for Value big size

Description Overrides one, or more, of the X-positions, and/or the fonts used for UI components.



Apply settings **after** calling **ui.go**

UI Callbacks

```
h = uiBar()
```

uiBar callback

Parameters None

Returns • **h**: Height of the status bar (the default `ui._bar` returns a height of 8 pixels)

Description An override callback for the top line status bar.

```
show = uiCell()
```

uiCell callback

Parameters None

Returns • **show**: Return `false` to prevent the call to `lcd.go`. Any other value will show the contents of the LCD buffer.

Description An override callback for the cellular status LCD display.

The default `uiCell` function should be sufficient for most requirements.



You do not have to call `lcd.go` - this is handled in the firmware.

uiSmp()

uiSmp callback

Parameters None

Returns None

Description An override callback for the sample LCD display.

This callback is invoked if [smp.save](#) is called - allowing you to display important values from the last saved sample.



You do not have to call [lcd.go](#) - this is handled in the firmware.

Configuration Variables

The rt.buffer stores the configuration variables inside an efficient C++ `key=value` memory structure that has about 6k of space.

The contents of this store are saved and read from the file `/Config/config.txt`

The `config.txt` file is updated 500ms after configuration changes have stopped happening. Conversely, editing config.txt through the USB HID interface will trigger a reload of the config.txt into RAM.

However, to make life much easier inside Lua, the values can be queried and modified directly as a Lua global table, called `c`.

Within the C++ structure, the keys are organised in ASCII order, with the underscore (`_`) character representing the branches of a tree. This method enables simple 'wildcard' style erasing and querying of sets of values, and is also compatible with Lua's syntax.

```
c.my_value = 123
c.my_other = 'A string'
c.my_more = 'A complex \r\n string\twith\x00controlcodes' ①

c.mynot = 'Not the same tree'
c.my_other=_ -- erases just one key-value
c.my=_ -- erases all c.my* key-values. But c.mynot remains
```

① See https://en.wikipedia.org/wiki/Escape_sequences_in_C for examples of escaped characters. e.g. `\t` = TAB, `\r` = CR, `\x00` = NULL, etc

Some of configuration names are fixed and used within the firmware. However, your Lua App can choose to make use of additional configuration parameters as needed.

The following sections list the firmware-fixed configuration names and their settings.

Config `c.alm` : Alarm

```
c.alm_txt01 = ''  
c.alm_txt02 = ' ' ...
```

Alarm text names

Description Text names for the alarms 01-32.



(Developers) Can be retrieved in Lua with `alm.txt(...)` function.

Config `c.arc` : Archive Parameters

The `/Archive/` folder saves copies of the files that are sent. When the file count is exceeded, or the total bytes is exceeded, the oldest files are removed from the folder.

- The folder is "flat" - any files that do not begin with `/Send/` are mapped so that the `/` characters are replaced by `^` symbols.
e.g. `/MyFolder/myfile.txt` is archived as `/Archive/MyFolder^myfile.txt`

If you create sub-folders within `/Archive/` then these folders will not be scanned, nor curated (i.e. they will stay there and not be pruned).

```
c.arc_fc = 32 v1.00
```

Archive number of files

Description Maximum number of files to keep in the `/Archive/` directory.

```
c.arc_gz = 0 v3.43
```

Archive use gzip

Description Set to 1 to gzip files are they are moved into the `/Archive/` directory.



There are routes for compressing. See `smp ftable.gz`, and `fc.gz`

```
c.arc_kb = 4096 v1.00
```

Archive maximum kbytes

Description Maximum number of kilo-bytes to keep in the `/Archive/` directory.

Config `c.cell` : Cellular

```
c.cell_apn = 'Internet'
```

Internet APN name

Description APN name for the Internet connection

```
c.cell_auth = -1
```

Override APN authentication method

Description (optional) Override the APN authentication method

- `-1` = Auto (depending on `c.cell_user` + `c.cell_pass`)
- `0` = None
- `1` = PAP only
- `2` = CHAP only
- `3` = PAP/CHAP

```
c.cell_hof = 10
```

Hold off failure time

Description (optional) hold off failure time, in minutes.

```
c.cell_hos = 2
```

Hold off success time

Description (optional) hold off success time, in minutes.

```
c.cell_init = '' v3.31
```

Modem init sequence

Description (optional) Modem initialisation sequence.



Do not include the "AT", just the command.
Separate commands with a control character, e.g. '\r'

```
c.cell_mto = 15
```

Maximum time online

Description (optional) Maximum time online, minutes.

```
c.cell_pass = ''
```

Internet password

Description (optional) Password for the Internet connection

```
c.cell_pin = ''
```

SIM PIN code

Description (optional) PIN number for the SIM card

```
c.cell_reg = 120
```

Cell registration time limit

Description (optional) Set the time limit for registration, in seconds.


```
c.cell_sms = ''
```

Default SMS number

Description (optional) Default SMS target number. Should be in international format (i.e. starting with "+" and the country code)

```
c.cell_user = ''
```

Internet username

Description (optional) Username for the Internet connection

Config `c.iot` : IoT Parameters

```
c.iot_data = ''
```

Data process URL or path

Description (optional) Data process URL or path.

Uses Lua expansions.

```
c.iot_gz = 1
```

Use gzip compression

Description (optional) Use GZIP compression and naming.

```
c.iot_job = '@23:00'
```

Job schedule

Description (optional) Job string for Data delivery (and Update).



(Developers) Sets both [onJob11](#) for Update, and [onJob12](#) for Data delivery, unless [c.iot_ujob](#) is set - in which case [c.iot_job](#) does just data.

```
c.iot_ntp = 'time.apple.com'
```

NTP server

Description (optional) NTP server name.

```
c.iot_tfr = 1
```

Use Temp-File-and-Rename for FTP

Description (optional) Use Temp File and Rename for FTP.

Set to 0 to disable this method and send the file directly.



`c.iot_tfr=1` requires the user FTP account support renaming. However, this approach is **STRONGLY** recommended as it ensures files are completely transferred. Just make sure your back-end processes ignore `*.tmp` files (It is also a good idea to purge very old `*.tmp` files, e.g. over 1 month old.)

```
c.iot_tz = 60 v3.00
```

Time-zone override

Description (optional) Timezone override, in minutes.



When set, there is no DST (Daylight Saving Time) applied.

```
c.iot_ujob = _
```

Update job schedule

Description (optional) Update job string. If not present, then `c.iot_job` sets both update and data job strings.

```
c.iot_upd = 'Update/{c.site_name}'
```

Update URL or path

Description (optional) Update URL or path for the update mechanism, relative to c.iot_url (although can be an absolute URL too!)

Uses Lua expansions - see [rt.exp](#).

```
c.iot_url = ''
```

Base URL

Description (required) Base URL for IOT.



This uses Lua expansions.



Required!

```
c.iot_url = 'ftp://mp:pm@ftp.scannex.com/Upload/{c.site_name}-{i.rt_sn}'
```

```
c.iot_user = ''
```

User process URL or path

Description (optional) User process URL or path.

Uses Lua expansions.

```
c.iot_var = 3600
```

Push variance

Description Server variance (seconds).

Jobs that are scheduled can be skewed by a combination of c.iot_var and the serial number of the rt.buffer. This ensures that your server does not get 'hit' by many rt.buffer devices all at the same time, but are spread over time.

Config `c.lua` : Lua Configuration

```
c.lua_app = ''
```

Lua app name

Description The App name to run.



This configuration value is **required** for the `rt.buffer` to work. And the corresponding `.lua` app must already be in the `/Lua` folder.

Config `c.site` : Site Deployment

```
c.site_loc = ''
```

Site location string

Description Optional Site location details. Can be used by the App.

```
c.site_name = ''
```

Site name

Description (optional) rt.buffer Site Name.

Used by the smp library for naming the files.

Config `c.term` : Terminal Controls

```
c.term_epw = 1 v1.00
```

Use password on engineer serial terminal

Description (optional) Whether Engineer Serial Port needs a password.

If there is a dedicated MCU connected to the Engineer Serial Port, then this can be set to 0 so that no password is required (but it will be required from the USB still if `c.term_pass` is set).

```
c.term_pass = ''
```

Terminal password

Description (optional) Terminal password.



Do NOT forget this password - there is no backdoor.

Config `c.tls` : Security Settings

```
c.tls_csc = 0xffff v1.00
```

TLS cipher suite

Description (optional) The modem cipher suite code for IoT connections

Cipher suites in this list from `0x003D` to `0x000A` are supported by the Quectel `UC20` modem. The Quectel `EG21` modem supports all in the list.

- `0x003D` = TLS_RSA_WITH_AES_256_CBC_SHA256 (Only possible when `c.tls_ver=3` since AES-256/SHA-256 is only available in TLSv1.2)
- `0x0035` = TLS_RSA_WITH_AES_256_CBC_SHA1
- `0x002F` = TLS_RSA_WITH_AES_128_CBC_SHA1
- `0x0005` = TLS_RSA_WITH_RC4_128_SHA1
- `0x0004` = TLS_RSA_WITH_RC4_128_MD5
- `0x000A` = TLS_RSA_WITH_3DES_EDE_CBC_SHA1
- `0X003D` = TLS_RSA_WITH_AES_256_CBC_SHA256
- `0XC002` = TLS_ECDH_ECDSA_WITH_RC4_128_SHA
- `0XC003` = TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
- `0XC004` = TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
- `0XC005` = TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA
- `0XC007` = TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
- `0XC008` = TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
- `0XC009` = TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- `0XC00A` = TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- `0XC011` = TLS_ECDHE_RSA_WITH_RC4_128_SHA
- `0XC012` = TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- `0XC013` = TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- `0XC014` = TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- `0xC00C` = TLS_ECDH_RSA_WITH_RC4_128_SHA
- `0XC00D` = TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
- `0XC00E` = TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
- `0XC00F` = TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
- `0XC023` = TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256

- `0xC024` = TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- `0xC025` = TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
- `0xC026` = TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
- `0XC027` = TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- `0XC028` = TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- `0xC029` = TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
- `0XC02A` = TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384
- `0XC02F` = TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `0xFFFF` = Support all ciphersuites above (default)



For the `UC20` fitted modem we recommend either `0x0035` or `0x002f` as a balance between security and server compatibility.

```
c.tls_ilt = 1 v1.00
```

TLS ignore local time

Description (optional) Ignore Local Time ^[2]

- 0 = Care about time checks for certificates
- 1 = Ignore time checks

```
c.tls_sni = 1 v3.44
```

TLS Server-Name-Indicator

Description (optiona) Present the Server-Name-Indicator when establishing TLS/SSL connection.

- 0 = Do not use TLS-SNI extension
- 1 = Use TLS-SNI extension



SNI is **required** when the target server is a virtually hosted server.

```
c.tls_ver = 3 v1.00
```

TLS version

Description (optional) TLS Version

- 0 = SSLv3 (not recommended)
- 1 = TLSv1.0 (not recommended)
- 2 = TLSv1.1
- 3 = TLSv1.2

[1] Checking the time against the certificate requires the local rt.buffer time be valid too!

[2] Checking the time against the certificate requires the local rt.buffer time be valid too!

Config `c.toa` : Terminal Over the Air settings

The Terminal Over the Air mechanism connects a TCP socket (or TCP+SSL) to an Internet server and provides a full link to the `rt.buffer`'s terminal interface.

```
c.toa_ito = 120 v1.10
```

TOA idle timeout

Description The idle timeout, in seconds, for the terminal.

If nothing is received from the server connection in this time, the `rt.buffer` will close the connection.

```
c.toa_key = NIL v1.10
```

TOA secret key

Description The shared secret key for HMAC-SHA-256 mutual authentication between the `rt.buffer` and the server.

This is for the link-level, so any terminal password will still be requested after the link has been established.

```
c.toa_pass = NIL v1.10
```

TOA password override

Description The password override for the TOA terminal.

If this field is not present the `c.term_pass` password is used. This setting allows you to keep the passwords separate.

```
c.toa_try = 300 v1.10
```

TOA retry time

Description The number of seconds to keep trying to make a TCP socket connection to the server. There is a 15 second pause between successive attempts.)

```
c.toa_url = '' v1.10
```

TOA URL

Description The URL details for the TOA link.

TCP and TCPS (SSL) links are currently handled.

```
c.toa_url='tcp://matt@terminal.scannex.com:12345/Path/To/Resource'
```

This example will use a plain-text TCP client connection to terminal.scannex.com on TCP port 12345.

Once connected, a CONNECT header will be sent to the server with the [/Path/To/Resource](#) and [user:matt](#) fields.

Config `c.user` : User IoT settings

The user process is typically triggered by the magnet.

```
c.user_ins = 15
```

Interval for user mode

Description The interval, in seconds, for the default User mode (by default: how many seconds to wait after sending the diagnostics data before sending again).

```
c.user_onm = 2
```

User mode online time

Description The online time, in minutes, for the default User mode (by default: how many minutes to keep pushing the diagnostics data).

Information Variables

Like the Configuration table, the global table `i` provides a window into some internal C++ values (without consuming Lua memory).

However, there are two 'real' Lua tables available too - that can be used for passing back application-specific information.

Info `i.app.XXXXX` : Application



For developers.

This is also a real Lua table that can be used by the application to hand back information through the USB console, or through the diagnostic dumps, etc.

The `i.app` table can contain additional tables within it, and these will be handled correctly.

```
-- within App
i.app.mine = {} -- blank table

-- later in the App
i.app.mine.v = 123
i.app.mine.too = 'Hello'
```

```
-- Within the USB terminal or via Update.txt:
lv i.app.mine
```


Info i.cell : Cellular Information

Modem activity

i.cell pha ^{v1.10} removed

Modem activity phase

Description The current, or last, phase of the cellular activity. (If i.cell_sta is 'off' then this is the last phase).

- _ / NIL = idle
- 'update' = update process
- 'ntp' = time sync
- 'pass' = pass thru
- 'toa' = Terminal Over Air
- 'user' = User Process
- 'data' = Data Upload
- 'loop' = Loopback control
- 'hold' = Hold-off

i.cell_pwr

Modem power duration

Description Total number of seconds the modem has been powered.

i.cell_sta

Modem state

Description String containing the state of the modem:

- 'off' = powered off
- 'pwr' = powering on
- 'reg' = registered
- 'online' = PDP context
- 'fail' = serious error
- '?' = unknown error

Modem timings

Cellular timings are split into the "Current" details, and the "Last" details - so you can see what the last transfer status was.

`i.cell_c_XXXX` has the current timings; `i.cell_l_XXXX` has the last timings.

i.cell_X_csq ^{v1.03}

Modem signal quality indicator

Description The signal quality indicator.

- 0-1 = nothing
- 2-9 = marginal
- 10-14 = ok
- 15-19 = good
- 20-31 = excellent
- 99 = unknown

i.cell_X_log v1.10

Brief modem activity log

Description Brief textual log of the modem activity and errors.

When the modem is powered up, the string starts with a colon ":".

Each job is listed. Any errors are listed with an exclamation mark "!" followed by the error.

Additional jobs are delimited by a comma "," and when the job list is complete the string is terminated with a full-stop "."

```
i.cell_c_log = ':ntp,update,test!FTP(625/530)!Connect.'
```

Shows that NTP time sync worked; the update process worked; but the test phase failed with an FTP 530 error (not logged in) - which failed the connection phase.

i.cell_X_od v1.03

Modem online duration

Description Online Duration. How many seconds the rt.buffer was online for.

Being online only implies an IP address and does not guarantee access to the wide Internet.

i.cell_X_pd v1.03

Modem power duration

Description Power Duration, in seconds. i.e. how many seconds the modem was powered up.

i.cell_X_pu v1.03

Modem power up time

Description Power UTC. The UTC time the modem was successfully powered up. (Power was applied about 5 seconds earlier.)

i.cell_X_rs v1.03

Modem registration seconds

Description Registration Seconds. How many seconds it took to get registered on the cellular network.

i.cell_X_sc v1.03

Modem server connected

Description Server Connected. Whether the IoT server was connected.

- 0=no server(s) connection
- 1=at least one server connection made

Info **i.fw** : Firmware Information

i.fw_blv

Bootloader version

Description BootLoader version

i.fw_date

Firmware date

Description Firmware date

i.fw_desc

Firmware description

Description Firmware name

i.fw_ver

Firmware version

Description Firmware version

Info `i.iot` : IoT Information

`i.iot_var`

Variance time applied

Description The actual number of seconds this `rt.buffer` will have variance-enabled schedules delayed.

Info **i.lua** : Lua Information

i.lua_run

Lua app start point

Description The seconds-alive value when the Lua App was started.

Zero means there is no Lua loop running - either onLoop has quit, or has not been defined.

Info `i.rt` : `rt.buffer` Information

`i.rt_cbf`

Configuration bytes free

Description Number of configuration store bytes free.

`i.rt_sn`

Serial number

Description Serial number string

`i.rt_so` v4.00

Original serial number

Description The original serial number.

This is z-base-32 encoded.

To decode:

```
original_serial = struct.unpack('L', util.zdec(i.rt_so))
```

`i.rt_tag` v4.00

Asset tag

Description The asset tag (if programmed)

i.rt_utc

Current time

Description The current time.

Info `i.smp.XXXX` : Samples

This is a real Lua table that is used for the activities of the Lua App's Sample (smp) library. By default, any record that is saved by the Lua App will update these info entries.

These parameters are only updated when the default smp table (ft) is used. When other tables are used - e.g. when handling multiple sample files - these info variables are not updated.

`i.smp.cnt`

Number of samples

Description The number of times that smp.save has been called.

This counter is reset if the rt.buffer is cold-booted, watchdog restarted, or Lua reboots.

`i.smp.txt`

Last sample text

Description The last text that was written with smp.save

`i.smp.utc`

Last sample time

Description The UTC time that smp.save was last called

Info **i.pwr** : Power Information

i.pwr_alv

Seconds alive

Description Seconds alive.**i.pwr_ccb**

Battery capacity

Description Battery capacity, in C.**i.pwr_ccu**

Estimated charge used

Description Estimated charge used**i.pwr_full** v3.00

Seconds on full power

Description Set to 1 if running at full power.

0 = running on backup mode.

i.pwr_pct v1.00

Estimated percentage power remaining

Description Estimated percentage power remaining (0-99%)

i.pwr_slp

Seconds asleep

Description	Number of seconds spent asleep (in ultra-low power mode).
--------------------	---

i.pwr_tob

Seconds on battery

Description	Number of seconds running on battery.
--------------------	---------------------------------------

Terminal Commands

These commands apply to the `rt.buffer>` prompt that is available through rtbTool via USB, serial, or TCP Terminal-Over-the-Air.

Terminal: Internal Debug Commands



These commands are not guaranteed to remain, and the format may change. They are primarily for Scannex development and testing.

`led`

Description Show LED status.

`mv`

Description Show the millivolt readings for power, etc.

`ninv`

Description Show the sense inputs on the COM ports.

`pmm`

Description Show power manager modes.

`wde`

`wde0`

Description Show watchdog entries.

`wde0` will reset the maximum counters.

```
work-  
work+
```

Description Decrement or increment the counter for hard-work.

If the counter is >0 then the CPU runs at 96MHz and will not enter low power mode.

Terminal: External Devices

adc

Description Show the ADC readings.

pcr

Description Pulse Counter Reset.

pcv

Description Show Pulse Counter Values.

```
pt[@]d [STOP]
pt[@]d[s] [STOP] v3.31
pt[@]d[s][f] [STOP] v4.00
```

- Parameters**
- **@**: Include the '@' symbol to pass the characters directly to the terminal. Without the '@' then control characters are escaped in Javascript/Lua/C style.
 - **d**: Use the 'Device' COM port
 - **s**: Optionally show the [LINK]/[link] states and [CTS]/[cts] line
 - **f**: use hardware TX flow control
 - **STOP**: Optional Lua expression to alter the stop sequence. Normally the [ESC] key quits the pass through mode.

Description Enter pass-through mode for the Device COM port.



Use the `pt@d` format when using software to communicate directly through rtbAPI.dll or USB-HID control.



Without the `@` symbol e.g. a NULL will be escaped in ASCII as '\x00', an ACK character as '\x06' etc. See https://en.wikipedia.org/wiki/Escape_sequences_in_C - but `\r`, `\n`, and `\t` are sent as real characters CR, LF, TAB.



Although the STOP can be a string sequence, the terminal is not fully buffered. If you are using a keyboard, then just use a single character!


```
ptd  
pt@d  
ptd '!'  
ptd '\x1b'  
ptds
```

```
pt[@]e [STOP]  
pt[@]e[s] [STOP] v3.31
```

Parameters (as for ptd above)

Description Enter pass-through mode for the Engineer COM port.

Terminal: General Functions

boot
reboot

Description Reboot the rt.buffer immediately. If USB is connected, the rt.buffer will remain in the Boot Loader for 5 seconds (unless you type a boot loader command).`

deepsleep

Description Turn 'off' the rt.buffer.



You can only wake up by reconnecting the Engineer serial port or USB for more than 1 second.

info

Description Show information about the firmware.

temp

Description Show temperature.

time

time YYYY-MM-DD HH:MM:SS [TZ]

Parameters

- **YYYY**: Year
- **MM**: Month
- **DD**: Day
- **HH**: Hour
- **MM**: Minute
- **SS**: Second
- **TZ**: (Optional)timezone offset, in minutes

Description Show the current time information, or set and show the time.

Terminal: Lua Core

```
lboot
```

Description Reboot Lua.



If the modem is active, the actual reboot will occur once Lua is unlocked. v3.00



The prompt will be shown when the reboot has happened. [ESC] will return to the prompt. v3.00

```
lgc  
lgc0
```

Description Garbage collect Lua memory and show stats.

`lgc0` will show and reset the statistics.

```
lm  
lm0
```

Description Show Lua memory stats.

`lm0` will show and reset the statistics.

lv LIST

?LIST v1.00

Parameters

- **LIST**: comma separated list of values

Description Show Lua value tree(s), or specific values.

=CMD v1.00 NOTE: =CMD is effectively the short-cut for `lx return CMD`

```
lx iot.go()
lx return iot.flg()
/return rt.ms()
```

c.KEY = VALUE

Parameters

- **KEY**: Configuration key name
- **VALUE**: Value in Lua format.

Description Set a configuration value.

```
c.iot_url='ftp://ftp.scannex.com/Path/To/Success'
```

Can also be used to wipe out a setting, or a tree of settings, using the Lua 'nil':

```
c.iot_url=nil
c.iot=nil
```

lf FILE

Parameters • **FILE**: Filename to run

Description Execute a Lua file.



If FILE does not include a / character, then /Lua/ is assumed.



If FILE does not include a . character, then .lua is assumed.

Terminal: Modem

cellinfo v1.10

Description Triggers the mechanism to obtain the cellular and modem information into </Logs/cellinfo.txt>

cellwait v1.10

Description Provides feedback on the modem. If the modem is busy, this will loop - showing the modem power-on time, and modem status ([i.cell_c_log](#)) every second.

You can halt the feedback by hitting the [\[ESC\]](#) key.

e.g.

```
cellinfo
cellwait
```

cho

Description Clear modem holdoff. If the modem was pausing between attempts, this command will make it finish the pause immediately.

diag v4.00

Description Triggers a push of diagnostics and check for updates.

modem

Description Enter modem loop-back mode. Useful for diagnostics via USB.

toa v1.10

Description Trigger the Terminal-Over-Air immediately.



Will clear the modem hold off and attempt to connect to the [c.toa_url](#)

update

Description Triggers a check for updates, clearing the modem holdoff.

Terminal: NAND Flash

nfcheck

Description Check the flash disk.



Firmware v4.00+ shows the results in the rtbTool log window, since the check operates in the background.

nfdel FILE

Parameters • **FILE**: Full path and filename

Description Delete a file from flash disk.

nfdir PATH

Parameters • **PATH**: Path of the folder to view.

Description Show the folder listing.



The path separator character is forward-slash / (Like UNIX/Linux, not Windows!)

nfformat

Description Erase EVERYTHING.



There is no way to undo this command! You WILL lose everything in the rt.buffer

nfinfo

Description Show information about the flash drive.

nfmkdir PATH

Parameters

- **PATH:** Full path

Description Make a folder, and intermediaries.

```
nfmkdir /Temp/Dir/Here
```

nfmove SOURCE TARGET

Parameters

- **SOURCE:** Source path + filename
- **TARGET:** Target path + filename

Description Moves a file or folder.

nfrename SOURCE TARGET

Parameters

- **SOURCE:** Source path + filename
- **TARGET:** Target filename (no path!)

Description Rename a file.

nfrmdir PATH

Parameters • [PATH](#): Full path

Description Remove a folder. Will fail if the folder is not empty

nftype FILE

Parameters • [FILE](#): Full filename

Description Shows the contents of a file.

nfwipedir PATH

Parameters • [PATH](#): Full path

Description Wipe a folder, including all its contents.



There is no way to undo this! May take a long while to execute, especially if there are lots of files.

nfwrite FILE

Parameters • [FILE](#): Path and filename

Description Write to a file. (Used internally by the rtbTool GUI app and rtbAPI tools)

BootLoader Terminal Commands

These USB-HID terminal commands apply to the `bootloader>` prompt.

```
boot  
reboot
```

Description Reboot the rt.buffer.

```
deepsleep
```

Description Turn 'off' the rt.buffer.

You can only wake up by reconnecting the Engineer serial port or USB for more than 1 second.

```
eraseall
```

Description Erase the whole NAND flash - erasing all Lua Apps, settings, and stored data.

The firmware and bootloader are not affected.



You will lose everything that was stored. Each block of NAND flash is physically erased. This is useful if the data was sensitive and the device needs to be shipped.

```
erasefw
```

Description Erase the current application firmware. Only the bootloader will remain.

This will not affect any stored NAND flash data.

info

Description Show information about the rt.buffer, bootloader, and application firmware.

run

Description Leave the bootloader and run the install firmware.

LED Sequences

The single, ultra-bright, amber LED gives an indication of the state of the rt.buffer. LED sequences consist of pairs of flashes.



Do not stare directly into the LED - it is *very* bright (especially if the top lid is off)!

In the list below, e.g. "12" = one flash + pause + two flashes

- Status
 - 11 = Alive (every 10 seconds)
 - If the rt.buffer is in ultra-low power mode the LED will be very dim, otherwise you'll see the '11' sequence but at normal brightness.
 - 11 = Alive (every 30 seconds) v3.00
 - The rt.buffer is in 'backup' battery mode (Lua not running)
 - 12 = Data has been stored
- Engineer Port
 - 22 = Engineer USB connected
 - 23 = Engineer USB in use
 - 25 = Magnet triggered
- Modem
 - 31 = Modem powering up
 - 32 = Modem registered
 - 33 = Modem online
- Boot Loader
 - 41 = Boot Loader active
 - 42 = USB running
 - 44 = Upgrading firmware
 - 45 = No Application BLF
- Faults
 - 54 = Debug
 - 55 = Fault

Example Lua App

To illustrate how to start coding a Lua App, consider an app that logs digital pulse counts.



This example uses the v2.00+ [coro](#) coroutine library.

The Main Sample

```
function GetSample()  
  local p1,p2 = dpc.rst()  
  local t = {p1,p2}  
  smp.save( smp.csv(t) )  
end
```

At the moment, the function will not be called.

So, we need to glue in a job schedule, and define a job:

```
function onJob1()  
  coro.add(GetSample)  
end  
  
job.set(1, '00:01')
```

Keeping Tally

Let's now extend the app to keep a total, where: $total = total + p1 - p2$

```
pt = 0 -- the total ①

function GetSample()
  local p1,p2 = dpc.rst()
  pt = pt + p1 - p2 ②
  local t = {p1,p2,pt} ③
  smp.save( smp.csv(t) )
end
```

- ① `pt` has the running total
- ② The value is updated
- ③ We create a temporary table to convert to CSV

Now, you can see the temporary table `t` also has the value of `pt` logged.

Linking the Schedule into the Configuration

Let's make the job setting more flexible by linking into the config table.

You'll see we'll make use of Lua's "or" - that takes the default if there is no config value defined (i.e. it's NIL):

```
job.set(1, c.myapp_job or '00:01')
```

Updating the Schedule Immediately

At the moment, the schedule is only assigned when Lua reboots (or the rt.buffer starts up). To apply the job settings whenever the configuration changes, we need to put the job.set function inside an onConfig handler function:

```
function onConfig()  
  job.set(1, c.myapp_job or '00:01')  
end
```

Adding Diagnostic Information

To illustrate making life easier to debug, we can feed back some information back into the i.app table. Let's just add the `t` table by adding the highlighted line:

```
local t = {p1,p2,pct}  
i.app.t = t ①  
smp.save( smp.csv(t) )
```

① The whole table is saved for examination in `i.app.t`

Adding Pressure Readings

It's not difficult to add the pressure reading as well:

```
function GetSample()  
  local p1,p2 = dpc.rst()  
  pt = pt + p1 - p2  
  local pv = rt.adc() ①  
  local t = {p1,p2,pt,pv} ②  
  i.app.t = t  
  smp.save( smp.csv(t) )  
end
```

① Reads the ADC value into `pv`

② Adds it to the table for conversion to CSV

Adding ADC Scaling

Now, let's add $y=mx+c$ scaling, by linking the `rt.adc` command into the values: `c.adc_m` and `c.adc_c`

```
local pv = rt.adc(c.adc_m or 1, c.adc_c or 0)
```

Again, you can see the default values being used.

Adding file headers and footers

Here, we can add a header that consists of all c & i values, with a blank CR/LF. In addition, the `ft` assignment makes use of the built-in MD5 hash generator to add an ASCII hash at the end of the file:

```
function Hdr(tb,h,nm)
  fc.lv(h, 'c,i', '\r\n')
end
ft = {h=Hdr,f=smp.t_md5}
```

Full Listing

```

-----
-- Demo App
-- Logs: time,count1,count2,total,adc
-----

pt = 0 -- the total

function Hdr(tb,h,nm)
  fc.lv(h, 'c,i', '\r\n')
end
ft = {h=Hdr,f=smp.t_md5}

function GetSample()
  local p1,p2 = dpc.rst()
  pt = pt + p1 - p2
  local pv = rt.adc(c.adc_m or 1, c.adc_c or 0)
  local t = {p1,p2,pt,pv}
  i.app.t = t
  smp.save( smp.csv(t) )
end

function onJob1()
  coro.add(GetSample)
end

function onConfig()
  job.set(1, c.myapp_job or '00:01')
end

```

Configuration settings can be controlled through:

```

c.adc_m = 1 -- Slope
c.adc_c = 0 -- Offset
c.myapp_job = '00:01' -- Job string for logging

```

Terminal Over the Air (TOA) Details

The Terminal Over the Air (TOA) establishes a TCP client connection to an Internet server so that full terminal functions can be executed remotely.

The Internet-connected server can be a port-forwarding rule into a user's PC, or into a server (e.g. node.js) that can act as a broker for the rt.buffers and the user connections (which might be Telnet, Web-access, etc).

The config values `c.toa_XXX` set the URL, the timeouts, and the shared key. This section provides details on the TOA protocol, and authentication phase.

Sequence of Connection

The sequence works as follows:

1. The TOA job gets requested, e.g. through `iot.go('toa')`
2. The `rt.buffer` will try for `c.toa_try` seconds:
3. Make TCP or TCP+SSL connection to `c.toa_url`
4. On connection, send the CONNECT TOA header
5. If `c.toa_key` is set, the `rt.buffer` will wait for a response of the correct SHA-256 key
6. The TCP connection then carries raw terminal commands (the PASSWORD prompt may be enabled if either `c.toa_pass` or `c.term_pass` is set)
7. (During connection you can issue commands - concurrently with the USB terminal, if connected)
8. If nothing is received by the `rt.buffer` in `c.toa_ito` seconds, the connection is dropped.

CONNECT header

```
CONNECT /Tester TOA/1.0
serial: 99889988
site: Test
time: 2018-01-09 14:31:31 UTC
utc: 1515508291
device: rt.buffer;co=Scannex
version: 1.10;bu=0118;da=2018-01-09;na=rt.buffer.Lua;ca=.uft.dlo.
idle: 120
user: matt
sha: 88f62c194223beefcddb43ef39181dcfb4bbef0801672fcfe5ace0c779badd4f

<<rt.buffer>>>
```

CONNECT /Tester TOA/1.0

Description

- **CONNECT**: The verb
- **/Tester**: The path from the URL. Use this for any purpose, e.g. routing traffic.
- **TOA/1.0**: Terminal Over the Air protocol version 1.0

Very similar to an HTTP header, this is the first line that is sent - allowing a broker to route the incoming connection.

```
device: rt.buffer;co=Scannex
```

Description The device type and company (co).

```
version: 1.10;bu=0118;da=2018-01-09;na=rt.buffer.Lua;ca=.uft.dlo.
```

Description The firmware version.

- `bu` = Build number
- `da` = Build date
- `na` = Name of application firmware
- `ca` = Capabilities of firmware.

```
idle: 120
```

Description Idle time, in seconds. See `c.toa_ito`.

```
user: matt
```

Description If the URL contains a user, e.g. `tcp://matt@mylocation.com:1234/Tester` then the username is added to the set of headers. Use this for routing incoming connections.

```
utc: 1515508291
```

Description The computer-readable UNIX-style UTC timecode.

site: Test**Description** The site name of the rt.buffer.**time: 2018-01-09 14:31:31 UTC****Description** The human-readable local time of the rt.buffer.**serial: 99889988****Description** The serial number of the rt.buffer.**sha: 093d300671ff1878fa4da0345f4d0a8ea365c1ab9b4bef7600b17d867a8edc****Description** If the shared key is not blank ([c.toa_key](#)) then:

```
sha = HMACSHA256( ' ', c.toa_key | HEADER )
```

If the shared key is blank, then:

```
sha = HMACSHA256( SECRET, HEADER )
```

Only the header text is used, with CR/LFs.

In the above example:

```
sha = HMACSHA256("secret", CONNECT /Tester TOA/1.0\r\nserial: 99889988\r\nsite:
Test\r\n
time: 2018-01-09 14:31:31 UTC\r\nutc: 1515508291\r\ndevice:
rt.buffer;co=Scannex\r\nversion:1.10;bu=0118;da=2018-01-
09;na=rt.buffer.Lua;ca=.uft.dlo.\r\nidle: 120\r\nuser: matt")
```

CONNECT header with Authentication

```
CONNECT /Tester TOA/1.0
serial: 99889988
site: Test
time: 2018-01-09 14:47:12 UTC
utc: 1515509232
device: rt.buffer;co=Scannex
version: 1.10;bu=0118;da=2018-01-09;na=rt.buffer.Lua;ca=.uft.dlo.
idle: 120
user: matt
auth: hmac.sha256
sha: 093d300671ff1878fa4da0345f4d0a8ea365c1ab9b4bef7600b17d867a8edcce
sha:
```

The header is the same as the non-authenticated header, but the "auth" field is present, and the rt.buffer waits for the SHA-256 response.

sha:[TAB]

Description	The rt.buffer is waiting for a line of text that should be the ASCII-hex of the HMAC-SHA-256:
--------------------	---

```
shar = HMACSHA256( secret, sha )
```

e.g. if c.toa_key = 'secret':

```
shar = HMACSHA256("secret",
"093d300671ff1878fa4da0345f4d0a8ea365c1ab9b4bef7600b17d867a8edcce")
```

If the SHA response is correct, the rt.buffer will then present the terminal (either the PASSWORD prompt, or the rt.buffer prompt). In this case, 'shar' calculates to [c7ef5df1bf21df68ebf302174c1c844f57cd37d47c9251fd07f880794d6d25b2](#), and sending this will unlock the connection:

```
sha: 093d300671ff1878fa4da0345f4d0a8ea365c1ab9b4bef7600b17d867a8edcce
sha: c7ef5df1bf21df68ebf302174c1c844f57cd37d47c9251fd07f880794d6d25b2
```

```
<<rt.buffer>>>
```

auth: hmac.sha256

Description Indicating that an HMAC-SHA-256 response will be expected before the link is established.

HTTP POST Protocol

The HTTP protocol uses custom headers that begin with `rtbuffer-`, along with the HTTP `POST` verb.

The protocol also supports HMAC-SHA256 handshaking for a cryptographic 'lock' between the `rt.buffer` and the server.

A sample PHP server application is available from Scannex.

The protocol allows sending zlib compressed data; cellular info; diagnostics dumps; general files; and update events.

Files from the `rt.buffer` are sent as multipart/post.

Files from the server are sent as a binary response, with additional data in custom headers.



The `rt.buffer` always uses 'Keep Alive' - so the server is forced to use Content-Length in the response. Otherwise, the server could use chunked, or no content-length (which is then unreliable).

Custom Headers

All the custom headers begin with `rtbuffer-`.

`rtbuffer-req`

- Description** The `req` header specifies what action the server should take.
- `'upd'` = Check for updates.
 - `'put'` = rt.buffer is putting a file onto the server
 - `'get'` = rt.buffer would like to request a file from the server
 - `'ack'` = rt.buffer is acknowledging completed update
 - `'nak'` = rt.buffer is telling the server there was an error with the update

`rtbuffer-serial: SERIAL`

Description The serial number of the rt.buffer.

`rtbuffer-name: NAME`

Description The name of the rt.buffer (from `c.site_name`).

`rtbuffer-utc: UNIXTIME; tz=OFFSET`

Description The UNIX UTC time, and timezone (tz=) of the rt.buffer. `tz` is expressed in minutes.

rtbuffer-sha: HEXSTRING**Description** Authentication HMAC-SHA256 hex string.**rtbuffer-req:** nak; msg=MESSAGE**Description** The rt.buffer failed to apply the update.**rtbuffer-req:** upd**Description** The rt.buffer is requesting an update file.

The server should reply:

- Nothing to do:
 - `rtbuffer-ack: 0`
- Data to process:
 - `rtbuffer-ack: 1`
 - File contents of `update.txt` should be sent as `Content-Type: application/octet-stream`
 - The rt.buffer will then execute the lines (if `rtbuffer-sha` agrees)

rtbuffer-req: ack**Description** The rt.buffer completed the update.

```
rtbuffer-req: put; fn=FILENAME
```

Description `fn` is the target source filename. Of course, the server script can override this.

The multipart/post will also include:

- `name` = The type of file
 - `'diag'` = diagnostics dump
 - `'data'` = data file from the `/Send/` folder
 - `'cellinfo'` = cellular survey information file
 - `'file'` = other file (e.g. pulled by the update process)
- `filename` = The target filename. `'.zlib'` will be appended if the file is zlib-compressed

```
rtbuffer-req: get; fn=FILENAME
```

Description `rt.buffer` is requesting the file `FILENAME`. The filename will be as passed within the `update.txt` file. Consequently handling of relative and absolute paths can be handled within the server-side script.

The sample PHP script adjusts relative paths based on the buffer's serial/name, and absolute paths based on the pseudo root path. The PHP script also checks for stupid values that break out of the pseudo root (e.g. `'../../../../../sbin/somenastyfile'`)

The server should reply:

- File does exist:
 - `rtbuffer-ack: 0`
- File exists:
 - `rtbuffer-ack: 1`
 - File contents should be passed as `application/octet-stream`

rtbuffer-upd: now

Description The server can add this header to tell the rt.buffer that it needs to perform an update request.

Example - Sending Diagnostics

rt.buffer sends

```
POST /rtb/upload.php/Update/No99889988 HTTP/1.1
User-Agent: rt.buffer.Lua/3.00.0157 (2019-03-22)
Host: 217.37.194.108:80
Accept: */*
rtbuffer-req: put; fn=diag.99889988.txt
rtbuffer-serial: 99889988
rtbuffer-utc: 1553531822; tz=0
rtbuffer-sha: 610cd0e71a1487752c5cac64dd95249d4df21685d23a18c7baabeb5f5c7b236a
Connection: keep-alive
Content-Type: multipart/form-data; boundary=-----
aSiQmphJ5k4BnWJjwnAOXpPboH31W0iD-----
Content-Length: 1716

-----aSiQmphJ5k4BnWJjwnAOXpPboH31W0iD-----
Content-Disposition: form-data; name="diag"; filename="diag.txt.zlib"
Content-Type: binary/octet-stream;
```

Server replies

```
HTTP/1.1 200 OK
Date: Mon, 25 Mar 2019 16:37:00 GMT
Server: Apache
X-Powered-By: PHP/5.4.16
rtbuffer-ack: 1
rtbuffer-sha: 6ecc930cac1b91c69c828265541696e6883589ac4cf0dfa829b0a6ff3d31f099
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

Example - Requesting Updates

rt.buffer sends

```
POST /rtb/upload.php/Update/No99889988 HTTP/1.1
User-Agent: rt.buffer.Lua/3.00.0157 (2019-03-22)
Host: 217.37.194.108:80
Accept: */*
rtbuffer-req: upd
rtbuffer-serial: 99889988
rtbuffer-utc: 1553531819; tz=0
rtbuffer-sha: fe7d8abedcc514ab9e7c6d6db08ed6ef86cb8e1af9332c9cd9b8d70eb647fd05
Connection: keep-alive
Content-Length: 0
```

Server replies

```
HTTP/1.1 200 OK
Date: Mon, 25 Mar 2019 16:36:57 GMT
Server: Apache
X-Powered-By: PHP/5.4.16
rtbuffer-ack: 0
rtbuffer-sha: 72f1beb92e3a7a9360e448d5f1622757bffe0d8cbb8c6d8d52f7d83d913f29c6
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

Cryptographic Hash Calculations

Initialisation

HMAC is HMAC-SHA-256

```
SECRET = 'RTB1:' + SHARED_SECRET
SERVER_SECRET = HMAC( '|' + HOST_ID + '|', SECRET)
```



Having a separate SERVER_SECRET based on the HOST_ID means the specific hash can be safely stored in the PHP script. If the server is compromised then the SHARED_SECRET is not compromised.

Verifying rt.buffer Request

```
DATA = 'POST' + '|' +
[rtbuffer-req] + '|' +
[rtbuffer-serial] + '|' +
[rtbuffer-name] + '|' +
[rtbuffer-utc]
```

e.g.

```
DATA = 'POST|upd||99889988|153531819; tz=0'
```

For each file:

```
FILE_SHA = FILE_SHA .. '|' + NAME + '|' + CONTENTS + '|'
```

So, use a single SHA256 context, and hash in the name and contents for each file in turn.



SHA-256, not HMAC-SHA-256

```
CALC_SHA = HMAC( DATA + '|' + FILE_SHA, SERVER_SECRET )
```

Then [CALC_SHA](#) should match the contents of [rtbuffer-sha](#)

Generating Response

```
HASH_DATA = [rtbuffer-sha] + '|' + HEADERVALUE + '|' + HEADERVALUE 'æ etc
```

HEADERVALUE is the value of each `rtbuffer-XXXX` header. e.g. "0" or "1" for `rtbuffer-ack`.

HASH_OF_FILE = either `SHA256( ' ' )` if there is no file, or `SHA256( CONTENTS )`

```
REPLY_SHA = HMAC( HASH_DATA + '|' + HASH_OF_FILE, SERVER_SECRET )
```

This **REPLY_SHA** should be sent back as `rtbuffer-sha: REPLY_SHA`

Software Licenses

Operating System

- Segger embOS
- Segger emFile
- Segger emUSBD

Portions:

(c) 2014 - 2019 SEGGER Microcontroller GmbH & Co. KG
www.segger.com Support: support@segger.com

Cortex Support Libraries (CMSIS)

Copyright (c) 2009 - 2015 ARM LIMITED

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of ARM nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,

INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.

IN NO EVENT SHALL COPYRIGHT HOLDERS AND CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,

OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;

OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)

ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Atmel Files

Portions of Atmel's ASF have been used:

Copyright (c) 2012-2015 Atmel Corporation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of Atmel may not be used to endorse or promote products derived from this software without specific prior written permission.
4. This software may only be redistributed and used in connection with an Atmel microcontroller product.

THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Lua License

Lua is licensed under the terms of the MIT license reproduced below.

This means that Lua is free software and can be used for both academic and commercial purposes at absolutely no cost.

For details and rationale, see <http://www.lua.org/license.html>.

Copyright (C) 1994-2008 Lua.org, PUC-Rio.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

(end of COPYRIGHT)

Libstruct Library

Copyright (C) 2010 Lua.org, PUC-Rio. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

eLua additions

Some portions of the eLua project have been used:

Copyright (c) 2007, 2008, 2009, 2010, 2011 Dado Sutter and Bogdan Marinescu

eLua is Open Source and is freely distributed under the MIT licence.

The Lua part of eLua is licensed under the Lua licensing terms, which you can find at <http://www.lua.org/license.html>.

The "pack" module is adapted from the "lpack" module by Luiz Henrique de Figueiredo and it's placed in the public domain.

The "bit" module is adapted from the "bitlib" library by Reuben Thomas, distributed under a MIT license.

The multiple memory allocator (dlmalloc) is written by Doug Lea and is placed on the public domain.

Manufacturer provided CPU support libraries are licensed under their own terms, check `src/platform/*platform-name*` for details.

The rest of the eLua code is licensed under MIT, listed below: The MIT License

Copyright (c) 2007, 2008, 2009, 2010 Dado Sutter and Bogdan Marinescu

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

zlib Compression Library

zlib.h -- interface of the 'zlib' general purpose compression library
version 1.2.12, March 11th, 2022

Copyright (C) 1995-2022 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly
jloup@gzip.org

Mark Adler
madler@alumni.caltech.edu

The data format used by the zlib library is described by RFCs (Request for Comments) 1950 to 1952 in the files <http://tools.ietf.org/html/rfc1950> (zlib format), rfc1951 (deflate format) and rfc1952 (gzip format).

Cryptographic Functions

Portions of mbedTLS have been used:

Copyright (C) 2006-2015, ARM Limited, All Rights Reserved
SPDX-License-Identifier: Apache-2.0

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

This file is part of mbed TLS (<https://tls.mbed.org>)

U8G2 LCD Library

<https://github.com/olikraus/u8g2>

The U8g2lib code (<http://code.google.com/p/u8g2/>) is licensed under the terms of the new-bsd license (two-clause bsd license).

See also: <http://www.opensource.org/licenses/bsd-license.php>

The repository and optionally the releases contain icons, which are derived from the WPZOOM Developer Icon Set:

<http://www.wpzoom.com/wpzoom/new-freebie-wpzoom-developer-icon-set-154-free-icons/>

WPZOOM Developer Icon Set by WPZOOM is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

Fonts are licensed under different conditions.

See

<https://github.com/olikraus/u8g2/wiki/fntgrp>

for detailed information on the licensing conditions for each font.

===== X11 Fonts COUR, HELV, NCEN, TIM, SYMB =====

For fonts derived from the following files, the license below applies.

COURB08.BDF COURB10.BDF COURB12.BDF COURB14.BDF COURB18.BDF
 COURB24.BDF COURR08.BDF COURR10.BDF COURR12.BDF COURR14.BDF
 COURR18.BDF COURR24.BDF HELVB08.BDF HELVB10.BDF HELVB12.BDF HELVB14.BDF
 HELVB18.BDF HELVB24.BDF HELVR08.BDF HELVR10.BDF HELVR12.BDF HELVR14.BDF
 HELVR18.BDF HELVR24.BDF NCENB08.BDF NCENB10.BDF NCENB12.BDF
 NCENB14.BDF NCENB18.BDF NCENB24.BDF NCENR08.BDF NCENR10.BDF
 NCENR12.BDF NCENR14.BDF NCENR18.BDF NCENR24.BDF SYMB08.BDF SYMB10.BDF
 SYMB12.BDF SYMB14.BDF SYMB18.BDF SYMB24.BDF TIMB08.BDF TIMB10.BDF
 TIMB12.BDF TIMB14.BDF TIMB18.BDF TIMB24.BDF TIMR08.BDF TIMR10.BDF
 TIMR12.BDF TIMR14.BDF TIMR18.BDF TIMR24.BDF

Copyright 1984-1989, 1994 Adobe Systems Incorporated.

Copyright 1988, 1994 Digital Equipment Corporation.

Adobe is a trademark of Adobe Systems Incorporated which may be registered in certain jurisdictions.

Permission to use these trademarks is hereby granted only in association with the images described in this file.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notices appear in all copies and that both those copyright notices and this permission notice appear in supporting documentation, and that the names of

Adobe Systems and Digital Equipment Corporation not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. Adobe Systems and Digital Equipment Corporation make no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

===== BSD License for U8g2lib Code =====

Universal 8bit Graphics Library (<http://code.google.com/p/u8g2/>)

Copyright (c) 2016, olikraus@gmail.com
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Index

- A**
- a2utc
 - utc.a2utc (Convert alive time to UTC), [209](#)
- abort
 - IoT abort modem requests (iot.abort), [101](#)
 - IoT abort user process (iot.aup), [102](#)
 - iot.abort (IoT abort modem requests), [101](#)
- ack
 - Check ACK (sdi12.ack), [167](#)
 - sdi12.ack (Check ACK), [167](#)
- act
 - gps.act (Get GPS activity), [89](#)
 - lcd.act (Check if the LCD is on), [235](#)
- activity
 - Brief modem activity log (i.cell_X_log), [293](#)
 - Get GPS activity (gps.act), [89](#)
 - Modem activity phase (i.cell pha), [291](#)
- adc
 - adc (TERM), [306](#)
 - ADC completion callback (onAdc), [19](#)
 - adc.def (Set ADC default parameters), [13](#)
 - adc.done (Check for complete), [13](#)
 - adc.get (Get ADC sample), [14](#)
 - adc.go (Start ADC), [14](#)
 - adc.obj (Get ADC object), [15](#)
 - adc.set (Set or get ADC parameters), [15](#)
 - Control ADC power output (pwr.apo), [149](#)
 - Get ADC object (adc.obj), [15](#)
 - Get ADC Object settings (adco:get), [16](#)
 - Get ADC Object Statistics (adco:stat), [17](#)
 - Get ADC reading (rt.adc), [155](#)
 - Get ADC sample (adc.get), [14](#)
 - Get/set ADC Object power control (adco:pwr), [17](#)
 - rt.adc (Get ADC reading), [155](#)
 - Set ADC default parameters (adc.def), [13](#)
 - Set ADC Object settings (adco:set), [17](#)
 - Set or get ADC parameters (adc.set), [15](#)
 - Start ADC (adc.go), [14](#)
 - Start ADC Object conversion (adco:go), [16](#)
- adco
 - adco:get (Get ADC Object settings), [16](#)
 - adco:go (Start ADC Object conversion), [16](#)
 - adco:pwr (Get/set ADC Object power control), [17](#)
 - adco:set (Set ADC Object settings), [17](#)
 - adco:stat (Get ADC Object Statistics), [17](#)
- add
 - Add callback to job queue (jqo:add), [125](#)
 - Add indexed tables (util.addi), [215](#)
 - Add running regression objects (rro:add), [86](#)
 - Add running stats objects (rso:add), [87](#)
 - Add tables (util.add), [215](#)
 - add/sub/mul/div values (fao:add), [75](#)
 - add/sub/mul/div values (fao:div), [75](#)
 - add/sub/mul/div values (fao:mul), [75](#)
 - add/sub/mul/div values (fao:sub), [75](#)
 - Coro add function to queue (coro.add), [41](#)
 - coro.add (Coro add function to queue), [41](#)
 - fao:add (add/sub/mul/div values), [75](#)
 - fc.add (File add extension), [56](#)
 - File add extension (fc.add), [56](#)
 - jqo:add (Add callback to job queue), [125](#)
 - rro:add (Add running regression objects), [86](#)
 - rso:add (Add running stats objects), [87](#)
 - util.add (Add tables), [215](#)
- addi
 - util.addi (Add indexed tables), [215](#)
- address
 - Change device address (sdi12.chg), [167](#)
- ag7
 - sms.ag7 (Convert ASCII to GSM-7bit), [196](#)
- al
 - bit.al (Bit arithmetic shift right), [32](#)
- alarm
 - Alarm clear (alm.clr), [20](#)
 - Alarm get (alm.get), [20](#)
 - Alarm get text (alm.txt), [22](#)
 - Alarm global callback function (onAlm), [25](#)
 - Alarm Object clear (ao:clr), [26](#)
 - Alarm Object state callback (function), [28](#)
 - Alarm Object update (ao:upd), [27](#)

- Alarm Object value (ao:val), [28](#)
- Alarm set (alm.set), [21](#)
- Alarm text names (c.alm_txt01), [271](#)
- Alarm text names (c.alm_txt02), [271](#)
- Alarm update (alm.upd), [22](#)
- Alarm value (alm.val), [23](#)
- Create alarm object (alm.new), [21](#)
- Set Alarm Object values (ao:set), [27](#)
- alive
 - Convert alive time to UTC (utc.a2utc), [209](#)
 - Convert UTC to alive time (utc.utc2a), [214](#)
 - Seconds alive (i.pwr_alv), [301](#)
- all
 - Pop all UI (ui.popa), [261](#)
 - Run all jobs in the queue (jqo:run), [127](#)
 - Sum all values (fao:sum), [80](#)
- allow
 - Allow hard work (pwr.work), [153](#)
- allowed
 - IoT allowed to run (iot.dorun), [104](#)
 - Still allowed to run (rt.dorun), [159](#)
 - Still allowed to run (rt.ok), [159](#)
- alm
 - alm.clr (Alarm clear), [20](#)
 - alm.get (Alarm get), [20](#)
 - alm.new (Create alarm object), [21](#)
 - alm.set (Alarm set), [21](#)
 - alm.txt (Alarm get text), [22](#)
 - alm.upd (Alarm update), [22](#)
 - alm.val (Alarm value), [23](#)
 - c.alm_txt01 (Alarm text names), [271](#)
 - c.alm_txt02 (Alarm text names), [271](#)
- alshift
 - bit.alshift (Bit arithmetic shift right), [32](#)
- alv
 - i.pwr_alv (Seconds alive), [301](#)
- anti
 - Set modem anti-cell-zombie timeout (iot.azt), [102](#)
- ao
 - ao:clr (Alarm Object clear), [26](#)
 - ao:set (Set Alarm Object values), [27](#)
 - ao:upd (Alarm Object update), [27](#)
 - ao:val (Alarm Object value), [28](#)
- apn
 - c.cell_apn (Internet APN name), [273](#)
 - Internet APN name (c.cell_apn), [273](#)
 - Override APN authentication method (c.cell_auth), [273](#)
- apo
 - pwr.apo (Control ADC power output), [149](#)
- app
 - App evaluate block (app.eval), [30](#)
 - App execute block (app.exec), [31](#)
 - App fetch lines (app.lines), [31](#)
 - App fetch text block (app.txt), [31](#)
 - app.eval (App evaluate block), [30](#)
 - app.exec (App execute block), [31](#)
 - app.lines (App fetch lines), [31](#)
 - app.txt (App fetch text block), [31](#)
 - c.lua_app (Lua app name), [280](#)
 - Lua app name (c.lua_app), [280](#)
 - Lua app start point (i.lua_run), [297](#)
- append
 - Append index table (util.tia), [223](#)
 - fc.append (File append contents), [56](#)
 - File append contents (fc.append), [56](#)
- applied
 - Variance time applied (i.iot_var), [296](#)
- ar
 - bit.ar (Bit arithmetic shift right), [32](#)
- arc
 - c.arc_fc (Archive number of files), [272](#)
 - c.arc_gz (Archive use gzip), [272](#)
 - c.arc_kb (Archive maximum kbytes), [272](#)
- arch
 - iot.arch (Archive file), [101](#)
- archive
 - Archive file (iot.arch), [101](#)
 - Archive maximum kbytes (c.arc_kb), [272](#)
 - Archive number of files (c.arc_fc), [272](#)
 - Archive use gzip (c.arc_gz), [272](#)
 - Trim archive folder (iot.trim), [110](#)
- arithmetic
 - Bit arithmetic shift right (bit.al), [32](#)
 - Bit arithmetic shift right (bit.alshift), [32](#)
 - Bit arithmetic shift right (bit.ar), [32](#)
 - Bit arithmetic shift right (bit.arshift), [32](#)

- array
 - Create new float array object (float.array), [66](#)
 - float.array (Create new float array object), [66](#)
 - Get an array entry (fao:get), [77](#)
 - Get array size (fao:size), [79](#)
 - Update an array entry (fao:set), [78](#)
- arrays
 - Multiply-and-sum two arrays (fao:xs), [81](#)
- arshift
 - bit.arshift (Bit arithmetic shift right), [32](#)
- ascent
 - Get the LCD ascent and descent metrics (lcd.gad), [245](#)
- ascii
 - Convert ASCII hex to binary (rt.bin), [156](#)
 - Convert ASCII to GSM-7bit (sms.ag7), [196](#)
 - Convert ASCII to UTF-16 (util.au16), [216](#)
 - Convert ASCII to UTF-8 (util.au8), [216](#)
 - Convert binary to ASCII hex (rt.hex), [160](#)
 - Convert GSM-7bit to ASCII (sms.g7a), [198](#)
 - Convert UTF-8 to ASCII (util.u8a), [225](#)
- asleep
 - Seconds asleep (i.pwr_slp), [302](#)
- asset
 - Asset tag (i.rt_tag), [298](#)
- atom
 - rt.atom (Atomic function call), [156](#)
- atomic
 - Atomic function call (rt.atom), [156](#)
- attr
 - fc.attr (File attributes), [57](#)
- attributes
 - File attributes (fc.attr), [57](#)
- au16
 - util.au16 (Convert ASCII to UTF-16), [216](#)
- au8
 - util.au8 (Convert ASCII to UTF-8), [216](#)
- aup
 - iot.aup (IoT abort user process), [102](#)
- auth
 - c.cell_auth (Override APN authentication method), [273](#)
- authentication
 - Override APN authentication method (c.cell_auth), [273](#)
- available
 - GPS location available callback (onGPS), [93](#)
- azt
 - iot.azt (Set modem anti-cell-zombie timeout), [102](#)
- B**
- back
 - pwr.back (Set backup battery), [149](#)
- backlight
 - Control LCD backlight (ui.set.bklt), [263](#)
- backup
 - Set backup battery (pwr.back), [149](#)
- band
 - bit.band (Bitwise AND), [33](#)
- base
 - Base URL (c.iot_url), [278](#)
 - Z-base-32 decode (util.zdec), [226](#)
 - Z-base-32 encode (util.zenc), [227](#)
- based
 - Set a UTC based job (job.utc), [120](#)
- battery
 - Battery capacity (i.pwr_ccb), [301](#)
 - Seconds on battery (i.pwr_tob), [302](#)
 - Set backup battery (pwr.back), [149](#)
 - Set battery coulomb capacity (pwr.sbc), [152](#)
- bias
 - ser.bias (Set RS485 bias control), [172](#)
 - Set RS485 bias control (ser.bias), [172](#)
- bin
 - rt.bin (Convert ASCII hex to binary), [156](#)
- binary
 - Convert ASCII hex to binary (rt.bin), [156](#)
 - Convert binary to ASCII hex (rt.hex), [160](#)
 - Read SIM FileID Binary (modem.simrb), [142](#)
 - Write SIM FileID Binary (modem.simwb), [143](#)
- bit
 - Bit arithmetic shift right (bit.al), [32](#)
 - Bit arithmetic shift right (bit.alshift), [32](#)

- Bit arithmetic shift right (`bit.ar`), [32](#)
- Bit arithmetic shift right (`bit.arshift`), [32](#)
- Bit cast (`bit.cast`), [34](#)
- Bit swap (`bit.swap32`), [35](#)
- Bit swap bytes (`bit.swap16`), [35](#)
- Bit swap nibbles (`bit.swap8`), [36](#)
- `bit.al` (Bit arithmetic shift right), [32](#)
- `bit.alshift` (Bit arithmetic shift right), [32](#)
- `bit.ar` (Bit arithmetic shift right), [32](#)
- `bit.arshift` (Bit arithmetic shift right), [32](#)
- `bit.band` (Bitwise AND), [33](#)
- `bit.bnot` (Bitwise NOT), [33](#)
- `bit.bor` (Bitwise OR), [33](#)
- `bit.bxor` (Bitwise XOR), [34](#)
- `bit.cast` (Bit cast), [34](#)
- `bit.ls` (Bitwise left shift), [34](#)
- `bit.lshift` (Bitwise left shift), [34](#)
- `bit.rs` (Bitwise right shift), [35](#)
- `bit.rshift` (Bitwise right shift), [35](#)
- `bit.swap16` (Bit swap bytes), [35](#)
- `bit.swap32` (Bit swap), [35](#)
- `bit.swap8` (Bit swap nibbles), [36](#)
- Convert ASCII to GSM-7bit (`sms.ag7`), [196](#)
- Convert GSM-7bit to ASCII (`sms.g7a`), [198](#)
- bitmap**
 - Draw LCD PBM bitmap from file (`lcd.pbm`), [242](#)
 - Draw LCD XBM bitmap string (`lcd.xbm`), [243](#)
 - Save the LCD to a bitmap (`lcd.save`), [238](#)
- bits**
 - Create bits table (`util.bits`), [217](#)
 - Create table from table with bits (`util.btab`), [217](#)
 - `util.bits` (Create bits table), [217](#)
- bitwise**
 - Bitwise AND (`bit.band`), [33](#)
 - Bitwise left shift (`bit.ls`), [34](#)
 - Bitwise left shift (`bit.lshift`), [34](#)
 - Bitwise NOT (`bit.bnot`), [33](#)
 - Bitwise OR (`bit.bor`), [33](#)
 - Bitwise right shift (`bit.rs`), [35](#)
 - Bitwise right shift (`bit.rshift`), [35](#)
 - Bitwise XOR (`bit.bxor`), [34](#)
- bklt**
 - `ui.set.bklt` (Control LCD backlight), [263](#)
- blf**
 - Set a BLF file (`rt.setblf`), [164](#)
- block**
 - App evaluate block (`app.eval`), [30](#)
 - App execute block (`app.exec`), [31](#)
 - App fetch text block (`app.txt`), [31](#)
- blv**
 - `i.fw_blv` (Bootloader version), [295](#)
- bnot**
 - `bit.bnot` (Bitwise NOT), [33](#)
- board**
 - Check expansion board (`ser.exp`), [173](#)
- bool**
 - `rt.bool` (Convert boolean value), [157](#)
- boolean**
 - Config get boolean (`cfg.getb`), [38](#)
 - Convert boolean value (`rt.bool`), [157](#)
- boost**
 - Set the boost values (`pwr._bv`), [154](#)
- boot**
 - `boot` (TERM), [308](#), [317](#)
 - `pwr.boot` (Reboot device), [150](#)
- bootloader**
 - Bootloader version (`i.fw_blv`), [295](#)
- bor**
 - `bit.bor` (Bitwise OR), [33](#)
- bound**
 - Set out-of-bound values (`mpco:oob`), [82](#)
- box**
 - Draw LCD box (`lcd.box`), [240](#)
 - Draw UI box (`ui.box`), [259](#)
 - `lcd.box` (Draw LCD box), [240](#)
 - `ui.box` (Draw UI box), [259](#)
- break**
 - Send break (`cp:brk`), [175](#)
 - Send break (`sdi12.brk`), [167](#)
- brief**
 - Brief modem activity log (`i.cell_X_log`), [293](#)
- brk**
 - `cp:brk` (Send break), [175](#)
 - `sdi12.brk` (Send break), [167](#)
- btab**

- util.btab (Create table from table with bits), [217](#)
- buffer
 - Display the current LCD buffer (lcd.go), [236](#)
 - Erase the LCD buffer (lcd.ers), [235](#)
- bus
 - Query TWI bus (twi.qy), [205](#)
- bv
 - pwr._bv (Set the boost values), [154](#)
- bxor
 - bit.bxor (Bitwise XOR), [34](#)
- byte
 - IoT get byte counts (iot.bytes), [102](#)
- bytes
 - Bit swap bytes (bit.swap16), [35](#)
 - Configuration bytes free (i.rt_cbf), [298](#)
 - iot.bytes (IoT get byte counts), [102](#)
 - Number of RX bytes waiting (cp:rxw), [184](#)
- C**
- c8
 - crc.c8 (CRC calculate), [46](#)
- cache
 - fc.cache (File set cache), [57](#)
 - File set cache (fc.cache), [57](#)
- calculate
 - Calculate millisecond finish time (rt.msf), [163](#)
 - Calculate n-order polynomial object (float.poly), [70](#)
 - Calculate size of format (struct.size), [201](#)
 - Calculate stats (fao:stat), [79](#)
 - CRC calculate (crc.c8), [46](#)
- calculation
 - CRC calculation (sdi12.crc), [168](#)
- call
 - Atomic function call (rt.atom), [156](#)
 - Protected call (rt.call), [157](#)
 - rt.call (Protected call), [157](#)
- callback
 - ADC completion callback (onAdc), [19](#)
 - Add callback to job queue (jqo:add), [125](#)
 - Alarm global callback function (onAlm), [25](#)
 - Alarm Object state callback (function), [28](#)
 - Check if callback in job queue (jqo:is), [126](#)
 - Event callback function (eventcallbackname), [53](#)
 - GPS location available callback (onGPS), [93](#)
 - IoT data callback (iotData), [113](#)
 - IoT Modem Complete callback (iotOff), [114](#)
 - IoT Modem Idle callback (iotIdle), [113](#)
 - IoT Modem Initialise callback (iotInit), [114](#)
 - IoT Modem Initialise callback (iotReg), [114](#)
 - IoT Modem Offline callback (iotOffline), [114](#)
 - IoT Modem Online callback (iotOnline), [114](#)
 - IoT Modem Starting callback (iotSetup), [115](#)
 - IoT test callback (iotTest), [115](#)
 - IoT update callback (iotUpd), [115](#)
 - IoT user callback (iotUser), [115](#)
 - Key global callback (onKey), [232](#)
 - Keyboard idle callback (onKeyIdle), [233](#)
 - SMS callback function (onSMS), [200](#)
 - uiBar callback (uiBar), [268](#)
 - uiCell callback (uiCell), [268](#)
 - uiSmp callback (uiSmp), [269](#)
- cap
 - cp:cap (Capture data to file), [175](#)
- capacity
 - Battery capacity (i.pwr_ccb), [301](#)
 - Set battery coulomb capacity (pwr.sbc), [152](#)
- capture
 - Capture data to file (cp:cap), [175](#)
- cast
 - Bit cast (bit.cast), [34](#)
 - bit.cast (Bit cast), [34](#)
- cbf
 - i.rt_cbf (Configuration bytes free), [298](#)
- ccb
 - i.pwr_ccb (Battery capacity), [301](#)
- ccu
 - i.pwr_ccu (Estimated charge used), [301](#)
- cell
 - c.cell_apn (Internet APN name), [273](#)
 - c.cell_auth (Override APN authentication method), [273](#)
 - c.cell_hof (Hold off failure time), [273](#)

- c.cell_hos (Hold off success time), [273](#)
- c.cell_init (Modem init sequence), [274](#)
- c.cell_mto (Maximum time online), [274](#)
- c.cell_pass (Internet password), [274](#)
- c.cell_pin (SIM PIN code), [274](#)
- c.cell_reg (Cell registration time limit), [274](#)
- c.cell_sms (Default SMS number), [275](#)
- c.cell_user (Internet username), [275](#)
- Cell registration time limit (c.cell_reg), [274](#)
- i.cell_pha (Modem activity phase), [291](#)
- i.cell_pwr (Modem power duration), [291](#)
- i.cell_sta (Modem state), [292](#)
- i.cell_X_csq (Modem signal quality indicator), [292](#)
- i.cell_X_log (Brief modem activity log), [293](#)
- i.cell_X_od (Modem online duration), [293](#)
- i.cell_X_pd (Modem power duration), [293](#)
- i.cell_X_pu (Modem power up time), [294](#)
- i.cell_X_rs (Modem registration seconds), [294](#)
- i.cell_X_sc (Modem server connected), [294](#)
- Set modem anti-cell-zombie timeout (iot.azt), [102](#)
- cellinfo
 - cellinfo (TERM), [312](#)
- cellwait
 - cellwait (TERM), [312](#)
- cert
 - iot.cert (IoT set PKI certificate), [103](#)
- certificate
 - IoT set PKI certificate (iot.cert), [103](#)
- cfg
 - cfg.get (Config get), [37](#)
 - cfg.getb (Config get boolean), [38](#)
 - cfg.getn (Config get number), [38](#)
 - cfg.gets (Config get string), [38](#)
 - cfg.key.chk (Check a secret key), [40](#)
 - cfg.key.set (Set a secret key), [40](#)
 - cfg.key.wipe (Erase a secret key), [40](#)
 - cfg.set (Config set), [38](#)
 - cfg.table (Config read as table), [39](#)
- change
 - Change device address (sdi12.chg), [167](#)
- charge
 - Estimated charge used (i.pwr_ccu), [301](#)
- check
 - Check a secret key (cfg.key.chk), [40](#)
 - Check ACK (sdi12.ack), [167](#)
 - Check expansion board (ser.exp), [173](#)
 - Check for complete (adc.done), [13](#)
 - Check free SMS entries (sms.free), [197](#)
 - Check if a normal number (float.isnor), [69](#)
 - Check if a number (float.isnan), [69](#)
 - Check if callback in job queue (jqo:is), [126](#)
 - Check if finite (float.isfin), [68](#)
 - Check if infinite (float.isinf), [68](#)
 - Check if the LCD is on (lcd.act), [235](#)
 - Check if UTC is sensible (utc.ok), [211](#)
 - Check keyboard hardware (key.ok), [231](#)
 - Check millisecond expiry (rt.msx), [163](#)
 - Check USB connected (usb.on), [208](#)
 - Check whether sequence complete (sdi12.done), [168](#)
 - Check whether signals are valid (cp:okr), [178](#)
- File check update (fc.chkupd), [58](#)
- chg
 - sdi12.chg (Change device address), [167](#)
- chk
 - cfg.key.chk (Check a secret key), [40](#)
- chkupd
 - fc.chkupd (File check update), [58](#)
- cho
 - cho (TERM), [312](#)
 - iot.cho (IoT clear hold off), [103](#)
- cipher
 - TLS cipher suite (c.tls_csc), [283](#)
- cir
 - lcd.cir (Draw LCD circle or ellipse), [240](#)
- circle
 - Draw LCD circle or ellipse (lcd.cir), [240](#)
- clean
 - fc.clean (File clean flash), [58](#)
 - File clean flash (fc.clean), [58](#)
- clear
 - Alarm clear (alm.clr), [20](#)
 - Alarm Object clear (ao:clr), [26](#)
 - Clear (fao:clr), [75](#)

- Clear running regression object (rro:clear), [86](#)
- Clear running stats object (rso:clear), [87](#)
- IoT clear hold off (iot.cho), [103](#)
- Read/Clear Forbidden PLMN List (modem.fplmn), [141](#)
- Request Clear Forbidden PLMN List (modem.clearfplmn), [140](#)
- rro:clear (Clear running regression object), [86](#)
- rso:clear (Clear running stats object), [87](#)
- clearfplmn
 - modem.clearfplmn (Request Clear Forbidden PLMN List), [140](#)
- clock
 - Get/Set Real Time Clock (utc rtc), [211](#)
- clr
 - alm.clr (Alarm clear), [20](#)
 - ao:clr (Alarm Object clear), [26](#)
 - fao:clr (Clear), [75](#)
- cmd
 - cp:cmd (Handle a command sequence), [176](#)
 - hydrins.cmd (Send command to HydrINS), [97](#)
 - modbus.cmd (Send MODBUS command), [131](#)
 - modem.cmd (Send modem command), [140](#)
- cmpi
 - util.cmpi (Compare indexed tables), [218](#)
- cnt
 - i.smp.cnt (Number of samples), [300](#)
 - smp.cnt (Set or get counter), [190](#)
- code
 - SIM PIN code (c.cell_pin), [274](#)
- codesize
 - Estimate Lua codesize (util.codesize), [218](#)
 - util.codesize (Estimate Lua codesize), [218](#)
- coefficient
 - Return coefficient (polyo:c), [85](#)
- coils
 - Read multiple coils (modbus.rmc), [134](#)
- coldboot
 - rt.coldboot (Reboot device), [158](#)
- colour
 - Set LCD draw colour (lcd.c), [235](#)
- colours
 - Set UI colours (ui.set.c), [263](#)
- com
 - Enable or disable terminal over COM port (ser.term), [174](#)
 - Get the device COM port object (ser.dev), [172](#)
 - Get the device COM port object (ser.eng), [172](#)
 - Lock the COM port (cp:lck), [177](#)
 - Set COM port power output (cp:pwo), [179](#)
 - Setup COM port (cp:set), [184](#)
 - Setup COM port for SDI-12 (sdi12.set), [170](#)
 - Setup MODBUS on COM port (modbus.set), [136](#)
 - Unlock the COM port (cp:ulck), [186](#)
- command
 - Handle a command sequence (cp:cmd), [176](#)
 - Send command to HydrINS (hydrins.cmd), [97](#)
 - Send MODBUS command (modbus.cmd), [131](#)
 - Send modem command (iot.modem), [107](#)
 - Send modem command (modem.cmd), [140](#)
 - Transparent command (sdi12.t), [170](#)
- compare
 - Compare indexed tables (util.cmpi), [218](#)
 - File wildcard compare (fc.wc), [65](#)
- complete
 - Check for complete (adc.done), [13](#)
 - Check whether sequence complete (sdi12.done), [168](#)
 - IoT Modem Complete callback (iotOff), [114](#)
 - Wait for TX to complete (cp:wtx), [186](#)
- completion
 - ADC completion callback (onAdc), [19](#)
- compression
 - Use gzip compression (c.iot_gz), [276](#)
- config
 - Config get (cfg.get), [37](#)
 - Config get boolean (cfg.getb), [38](#)
 - Config get number (cfg.getn), [38](#)
 - Config get string (cfg.gets), [38](#)

- Config read as table (cfg.table), [39](#)
- Config set (cfg.set), [38](#)
- configuration
 - Configuration bytes free (i.rt_cbf), [298](#)
- connected
 - Check USB connected (usb.on), [208](#)
 - Modem server connected (i.cell_X_sc), [294](#)
 - Set modem stay-connected millivolt (iot.scmv), [107](#)
- connection
 - Trigger IoT connection (iot.go), [105](#)
- consumption
 - Set the coulomb consumption for device (pwr.dpc), [150](#)
 - Set the coulomb consumption for engineer (pwr.epc), [150](#)
- container
 - Create key-value container object (util.kvc), [222](#)
 - Create new key-value container object (util.kvc), [128](#)
- contents
 - File append contents (fc.append), [56](#)
- contrast
 - Set LCD display contrast (lcd.sdc), [239](#)
- control
 - Control ADC power output (pwr.apo), [149](#)
 - Control LCD backlight (ui.set.bklt), [263](#)
 - Control LCD state (lcd.on), [238](#)
 - Control power outputs (out.pwo), [147](#)
 - Get/set ADC Object power control (adco.pwr), [17](#)
 - Set receive flow control (cp:rx), [182](#)
 - Set RS485 bias control (ser.bias), [172](#)
 - Set transmit flow control (cp:tx), [185](#)
- conversion
 - Create multi-point-conversion object (float.mpc), [70](#)
 - Start ADC Object conversion (adco.go), [16](#)
- convert
 - Convert alive time to UTC (utc.a2utc), [209](#)
 - Convert ASCII hex to binary (rt.bin), [156](#)
 - Convert ASCII to GSM-7bit (sms.ag7), [196](#)
 - Convert ASCII to UTF-16 (util.au16), [216](#)
 - Convert ASCII to UTF-8 (util.au8), [216](#)
 - Convert binary to ASCII hex (rt.hex), [160](#)
 - Convert boolean value (rt.bool), [157](#)
 - Convert CSV text to table (util.csvt), [219](#)
 - Convert from x to y (mpco:y), [84](#)
 - Convert GSM-7bit to ASCII (sms.g7a), [198](#)
 - Convert hash string to value (hash.val), [95](#)
 - Convert hash value to text (hash.txt), [95](#)
 - Convert NMEA line to table (nmea.tab), [144](#)
 - Convert SIM STATUS to table (modem.stot), [143](#)
 - Convert string to UTC (utc.val), [214](#)
 - Convert table to CSV text (smp.csv), [190](#)
 - Convert table to CSV text (util.tcsv), [223](#)
 - Convert table to NMEA string (nmea.txt), [145](#)
 - Convert to table (fao:tab), [80](#)
 - Convert UTC to alive time (utc.utc2a), [214](#)
 - Convert UTF-16 to UTF-8 (util.u16u8), [224](#)
 - Convert UTF-8 to ASCII (util.u8a), [225](#)
 - Convert UTF-8 to UTF-16 (util.u8u16), [225](#)
 - Convert x to y (polyo:y), [85](#)
- copy
 - Copy values (fao:copy), [76](#)
 - Deep copy table (rt.dct), [158](#)
 - Deep copy table (util.dct), [220](#)
 - fao:copy (Copy values), [76](#)
 - fc.copy (File copy), [59](#)
 - File copy (fc.copy), [59](#)
- corners
 - Set UI rounded corners (ui.set.r), [266](#)
- coro
 - Coro add function to queue (coro.add), [41](#)
 - Coro exception object (coro.except), [45](#)
 - Coro get ID (coro.id), [42](#)
 - Coro kill routine (coro.kill), [42](#)
 - Coro number of routines running (coro.n), [42](#)
 - Coro wait until (coro.wu), [44](#)
 - Coro yield or wait (coro.w), [43](#)
 - Coro yield or wait (coro.yld), [43](#)
 - coro.add (Coro add function to queue), [41](#)
 - coro.except (Coro exception object), [45](#)
 - coro.id (Coro get ID), [42](#)

- coro.kill (Coro kill routine), [42](#)
- coro.n (Coro number of routines running), [42](#)
- coro.w (Coro yield or wait), [43](#)
- coro.wu (Coro wait until), [44](#)
- coro.yld (Coro yield or wait), [43](#)
- cortex
 - Get Cortex cycle counter (rt.cyc), [158](#)
- coulomb
 - Set battery coulomb capacity (pwr.sbc), [152](#)
 - Set the coulomb consumption for device (pwr.dpc), [150](#)
 - Set the coulomb consumption for engineer (pwr.epc), [150](#)
- coulombs
 - Set coulombs used (pwr.scu), [152](#)
- counter
 - Get Cortex cycle counter (rt.cyc), [158](#)
 - Reset the GPS counter (gps.rst), [91](#)
 - Set or get counter (smp.cnt), [190](#)
- counts
 - IoT get byte counts (iot.bytes), [102](#)
- cp
 - cp:brk (Send break), [175](#)
 - cp:cap (Capture data to file), [175](#)
 - cp:cmd (Handle a command sequence), [176](#)
 - cp:cts (Get CTS line), [177](#)
 - cp:fifo (Set the receive FIFO), [177](#)
 - cp:lck (Lock the COM port), [177](#)
 - cp:okr (Check whether signals are valid), [178](#)
 - cp:prot (Get the protocol settings), [179](#)
 - cp:pwo (Set COM port power output), [179](#)
 - cp:pwr (Enable power for receiving), [180](#)
 - cp:pwt (Enable power for transmitting), [180](#)
 - cp:rs (Set the RS232/RS485 mode), [181](#)
 - cp:rts (Set RTS line), [181](#)
 - cp:rx (Receive data), [182](#)
 - cp:rx (Receive and empty data), [182](#)
 - cp:rx (Set receive flow control), [182](#)
 - cp:rxpw (Enable power for receiving), [180](#)
 - cp:rxto (Set receive timeout value), [183](#)
 - cp:rxt (Enable power for receiving and transmitting), [183](#)
 - cp:rxu (Receive until), [183](#)
 - cp:rxw (Number of RX bytes waiting), [184](#)
 - cp:set (Setup COM port), [184](#)
 - cp:tx (Transmit data), [185](#)
 - cp:txf (Set transmit flow control), [185](#)
 - cp:txpw (Enable power for transmitting), [180](#)
 - cp:txtg (Set transmit time guard gap), [186](#)
 - cp:ulck (Unlock the COM port), [186](#)
 - cp:wtx (Wait for TX to complete), [186](#)
- crc
 - CRC calculate (crc.c8), [46](#)
 - CRC calculation (sdi12.crc), [168](#)
 - CRC finalise (ct:fin), [47](#)
 - CRC new object (crc.new), [46](#)
 - CRC update (ct:upd), [47](#)
 - crc.c8 (CRC calculate), [46](#)
 - crc.new (CRC new object), [46](#)
 - sdi12.crc (CRC calculation), [168](#)
- create
 - Create a new Job Queue Object (jobq.new), [123](#)
 - Create a running regression object (float.regr), [71](#)
 - Create a running statistics object (float.stat), [72](#)
 - Create alarm object (alm.new), [21](#)
 - Create bits table (util.bits), [217](#)
 - Create key-value container object (util.kvc), [222](#)
 - Create multi-point-conversion object (float.mpc), [70](#)
 - Create new float array object (float.array), [66](#)
 - Create new hash object (hash.new), [94](#)
 - Create new key-value container object (util.kvc), [128](#)
 - Create table from table with bits (util.btab), [217](#)
- creation
 - File get UTC creation (fc.utc), [65](#)
- csc
 - c.tls_csc (TLS cipher suite), [283](#)

csq	Load data from file to KVC object (kvco:_load), 130
i.cell_X_csq (Modem signal quality indicator), 292	Receive and empty data (cp:rxe), 182
csv	Receive data (cp:rx), 182
Convert CSV text to table (util.csvt), 219	Save data from KVC object to file (kvco:_save), 130
Convert table to CSV text (smp.csv), 190	Transmit data (cp:tx), 185
Convert table to CSV text (util.tcsv), 223	date
smp.csv (Convert table to CSV text), 190	Firmware date (i.fw_date), 295
csvt	i.fw_date (Firmware date), 295
util.csvt (Convert CSV text to table), 219	days
ct	Get whole days (utc.wds), 214
ct:fin (CRC finalise), 47	dct
ct:upd (CRC update), 47	rt.dct (Deep copy table), 158
cts	util.dct (Deep copy table), 220
cp:cts (Get CTS line), 177	debug
Get CTS line (cp:cts), 177	Enable debug LED sequence (rt.led), 161
current	dec
Current time (i.rt_utc), 299	sms.dec (Decode SMS PDU), 196
Digital pulse current level (dpc.lvl), 48	decode
Display the current LCD buffer (lcd.go), 236	Decode SMS PDU (sms.dec), 196
Return current UTC (utc.now), 210	Z-base-32 decode (util.zdec), 226
cut	deep
Cut indexed table (util.cut), 219	Deep copy table (rt.dct), 158
Cut the sample file (smp.cut), 191	Deep copy table (util.dct), 220
smp.cut (Cut the sample file), 191	deepsleep
cuti	deepsleep (TERM), 308 , 317
util.cuti (Cut indexed table), 219	def
cwt	adc.def (Set ADC default parameters), 13
Wavelet Transform (DWT/CWT) (fao:wt), 80	default
cyc	Default SMS number (c.cell_sms), 275
rt.cyc (Get Cortex cycle counter), 158	Set ADC default parameters (adc.def), 13
cycle	del
Get Cortex cycle counter (rt.cyc), 158	fc.del (File delete), 59
D	jgo:del (Delete jobs from the queue), 125
d2v	delay
float.d2v (Double to integer), 66	Delay microseconds (rt.us), 165
d2x	Delay milliseconds (rt.ms), 162
float.d2x (Double to hex), 67	Set magnet delay (rt.mag), 162
data	delete
c.iot_data (Data process URL or path), 276	Delete jobs from the queue (jgo:del), 125
Capture data to file (cp:cap), 175	File delete (fc.del), 59
Data process URL or path (c.iot_data), 276	desc
IoT data callback (iotData), 113	i.fw_desc (Firmware description), 295

- descent
 - Get the LCD ascent and descent metrics (lcd.gad), [245](#)
- description
 - Firmware description (i.fw_desc), [295](#)
- details
 - Get device details (rt.dev), [159](#)
- dev
 - rt.dev (Get device details), [159](#)
 - ser.dev (Get the device COM port object), [172](#)
- device
 - Change device address (sdi12.chg), [167](#)
 - Get device details (rt.dev), [159](#)
 - Get the device COM port object (ser.dev), [172](#)
 - Get the device COM port object (ser.eng), [172](#)
 - Reboot device (pwr.boot), [150](#)
 - Reboot device (rt.coldboot), [158](#)
 - Set the coulomb consumption for device (pwr.dpc), [150](#)
 - Turn off device (pwr.off), [151](#)
- dft
 - Fourier Transform (FFT/DFT) (fao:ft), [76](#)
- DFT (Fourier Transform), [76](#), [81](#)
- di
 - evt.di (Event enable or disable), [50](#)
- diag
 - diag (TERM), [312](#)
 - iot.diag (Send diagnostics info to IoT server), [104](#)
- diagnostics
 - Send diagnostics info to IoT server (iot.diag), [104](#)
- digital
 - Digital pulse current level (dpc.lvl), [48](#)
 - Digital pulse disable (dpc.dis), [48](#)
 - Digital pulse get value (dpc.val), [49](#)
 - Digital pulse reset (dpc.rst), [49](#)
- dir
 - File join dir and file (fc.mdf), [63](#)
 - File split dir and file (fc.sdf), [64](#)
- directory
 - File make directory (fc.mkd), [63](#)
 - File remove directory (fc.rmd), [64](#)
- dis
 - dpc.dis (Digital pulse disable), [48](#)
- disable
 - Digital pulse disable (dpc.dis), [48](#)
 - Enable or disable terminal over COM port (ser.term), [174](#)
 - Event enable or disable (evt.di), [50](#)
 - Event enable or disable (evt.en), [50](#)
- discrete
 - Read discrete inputs (modbus.rdi), [133](#)
- display
 - Display the current LCD buffer (lcd.go), [236](#)
 - Set LCD display contrast (lcd.sdc), [239](#)
- dist
 - gps.dist (Set/Get GPS fix distance), [90](#)
- distance
 - Set/Get GPS fix distance (gps.dist), [90](#)
- div
 - add/sub/mul/div values (fao:add), [75](#)
 - add/sub/mul/div values (fao:div), [75](#)
 - add/sub/mul/div values (fao:mul), [75](#)
 - add/sub/mul/div values (fao:sub), [75](#)
 - fao:div (add/sub/mul/div values), [75](#)
- done
 - adc.done (Check for complete), [13](#)
 - sdi12.done (Check whether sequence complete), [168](#)
- dorun
 - iot.dorun (IoT allowed to run), [104](#)
 - rt.dorun (Still allowed to run), [159](#)
- double
 - Double to hex (float.d2x), [67](#)
 - Double to integer (float.d2v), [66](#)
 - Hex to double (float.x2d), [74](#)
 - Integer to double (float.v2d), [73](#)
- doubles
 - Read floats/doubles from file (float.rd), [71](#)
 - Read floats/doubles from file (float.rf), [71](#)
 - Write floats/doubles to file (float.wd), [73](#)
 - Write floats/doubles to file (float.wf), [73](#)
- dp
 - float.dp (Trim float value), [67](#)

dpc

dpc.dis (Digital pulse disable), [48](#)
 dpc.lvl (Digital pulse current level), [48](#)
 dpc.rst (Digital pulse reset), [49](#)
 dpc.val (Digital pulse get value), [49](#)
 pwr.dpc (Set the coulomb consumption for device), [150](#)

draw

Draw LCD box (lcd.box), [240](#)
 Draw LCD circle or ellipse (lcd.cir), [240](#)
 Draw LCD glyph (lcd.gly), [245](#)
 Draw LCD horizontal line (lcd.lh), [241](#)
 Draw LCD line (lcd.l), [241](#)
 Draw LCD PBM bitmap from file (lcd.pbm), [242](#)
 Draw LCD pixel (lcd.p), [242](#)
 Draw LCD text (lcd.t), [246](#)
 Draw LCD triangle (lcd.tri), [243](#)
 Draw LCD vertical line (lcd.lv), [241](#)
 Draw LCD XBM bitmap string (lcd.xbm), [243](#)
 Draw soft key (ui.key), [260](#)
 Draw UI box (ui.box), [259](#)
 Set LCD draw colour (lcd.c), [235](#)

duration

Modem online duration (i.cell_X_od), [293](#)
 Modem power duration (i.cell_pwr), [291](#)
 Modem power duration (i.cell_X_pd), [293](#)
 Set/Get overall GPS duration (gps.time), [92](#)

dwt

Wavelet Transform (DWT/CWT) (fao:wt), [80](#)

DWT (Wavelet Transform), [80](#), [81](#)

E**element**

Get the top UI element (ui.top), [261](#)
 Get UI element text (ui.txt), [262](#)
 Pop UI element (ui.pop), [260](#)
 Push UI element (ui.push), [261](#)

ellipse

Draw LCD circle or ellipse (lcd.cir), [240](#)

empty

Receive and empty data (cp:rx), [182](#)

en

evt.en (Event enable or disable), [50](#)

enable

Enable debug LED sequence (rt.led), [161](#)
 Enable or disable terminal over COM port (ser.term), [174](#)
 Enable power for receiving (cp:pwr), [180](#)
 Enable power for receiving (cp:rxpw), [180](#)
 Enable power for receiving and transmitting (cp:rxtx), [183](#)
 Enable power for transmitting (cp:pwtx), [180](#)
 Enable power for transmitting (cp:txpw), [180](#)
 Event enable or disable (evt.di), [50](#)
 Event enable or disable (evt.en), [50](#)
 Set key repeat enable (key.kre), [230](#)

enc

sms.enc (Encode SMS PDU), [197](#)

encode

Encode SMS PDU (sms.enc), [197](#)
 Z-base-32 encode (util.zenc), [227](#)

eng

ser.eng (Get the device COM port object), [172](#)

engineer

Set the coulomb consumption for engineer (pwr.epc), [150](#)
 Use password on engineer serial terminal (c.term_epw), [282](#)

entries

Check free SMS entries (sms.free), [197](#)
 Square root entries (fao:sqrt), [79](#)

entry

Get an array entry (fao:get), [77](#)
 Update an array entry (fao:set), [78](#)
 Write log entry (rt.log), [161](#)

epc

pwr.epc (Set the coulomb consumption for engineer), [150](#)

epw

c.term_epw (Use password on engineer serial terminal), [282](#)

erase

Erase a secret key (cfg.key.wipe), [40](#)

- Erase the LCD buffer (lcd.ers), [235](#)
- eraseall
 - eraseall (TERM), [317](#)
- erasefw
 - erasefw (TERM), [317](#)
- ers
 - lcd.ers (Erase the LCD buffer), [235](#)
- estimate
 - Estimate Lua codesize (util.codesize), [218](#)
- estimated
 - Estimated charge used (i.pwr_ccu), [301](#)
 - Estimated percentage power remaining (i.pwr_pct), [301](#)
- eval
 - app.eval (App evaluate block), [30](#)
- evaluate
 - App evaluate block (app.eval), [30](#)
- event
 - Event callback function (eventcallbackname), [53](#)
 - Event enable or disable (evt.di), [50](#)
 - Event enable or disable (evt.en), [50](#)
 - Event initialise (evt.init), [50](#)
 - Event signal (evt.lsig), [51](#)
 - Event signal trigger (evt.sig), [52](#)
 - Event wait (evt.lwait), [51](#)
- eventcallbackname
 - eventcallbackname (Event callback function), [53](#)
- evt
 - evt.di (Event enable or disable), [50](#)
 - evt.en (Event enable or disable), [50](#)
 - evt.init (Event initialise), [50](#)
 - evt.lsig (Event signal), [51](#)
 - evt.lwait (Event wait), [51](#)
 - evt.sig (Event signal trigger), [52](#)
- except
 - coro.except (Coro exception object), [45](#)
- exception
 - Coro exception object (coro.except), [45](#)
- exchange
 - Trigger SMS exchange (sms.go), [198](#)
- exec
 - app.exec (App execute block), [31](#)
- execute
 - App execute block (app.exec), [31](#)
 - Execute with journalling (fc.jx), [62](#)
- exp
 - rt.exp (Expand a string), [160](#)
 - ser.exp (Check expansion board), [173](#)
- expand
 - Expand a string (rt.exp), [160](#)
- expansion
 - Check expansion board (ser.exp), [173](#)
- expiry
 - Check millisecond expiry (rt.msx), [163](#)
- ext
 - fc.ext (File split extension), [60](#)
- extension
 - File add extension (fc.add), [56](#)
 - File split extension (fc.ext), [60](#)
- F**
- f2v
 - float.f2v (Float to integer value), [67](#)
- f2x
 - float.f2x (Float to hex), [68](#)
- failure
 - Hold off failure time (c.cell_hof), [273](#)
- fao
 - fao:add (add/sub/mul/div values), [75](#)
 - fao:clr (Clear), [75](#)
 - fao:copy (Copy values), [76](#)
 - fao:div (add/sub/mul/div values), [75](#)
 - fao:ft (Fourier Transform (FFT/DFT)), [76](#)
 - fao:get (Get an array entry), [77](#)
 - fao:mul (add/sub/mul/div values), [75](#)
 - fao:push (Push value), [77](#)
 - fao:set (Update an array entry), [78](#)
 - fao:sin (Fill with sin values), [78](#)
 - fao:size (Get array size), [79](#)
 - fao:sqrt (Square root entries), [79](#)
 - fao:stat (Calculate stats), [79](#)
 - fao:sub (add/sub/mul/div values), [75](#)
 - fao:sum (Sum all values), [80](#)
 - fao:tab (Convert to table), [80](#)
 - fao:wt (Wavelet Transform (DWT/CWT)), [80](#)
 - fao:xs (Multiply-and-sum two arrays), [81](#)

- fc
 - c.arc_fc (Archive number of files), [272](#)
 - fc.add (File add extension), [56](#)
 - fc.append (File append contents), [56](#)
 - fc.attr (File attributes), [57](#)
 - fc.cache (File set cache), [57](#)
 - fc.chkupd (File check update), [58](#)
 - fc.clean (File clean flash), [58](#)
 - fc.copy (File copy), [59](#)
 - fc.del (File delete), [59](#)
 - fc.ext (File split extension), [60](#)
 - fc.find (File find), [60](#)
 - fc.gz (File gzip), [61](#)
 - fc.js (Journalling supported), [61](#)
 - fc.jx (Execute with journalling), [62](#)
 - fc.lv (File write Lua value tree), [62](#)
 - fc.mdf (File join dir and file), [63](#)
 - fc.mkd (File make directory), [63](#)
 - fc.move (File move), [63](#)
 - fc.rmd (File remove directory), [64](#)
 - fc.sdf (File split dir and file), [64](#)
 - fc.sfw (File set file write mode), [64](#)
 - fc.sync (File sync to flash), [65](#)
 - fc.utc (File get UTC creation), [65](#)
 - fc.wc (File wildcard compare), [65](#)
- fetch
 - App fetch lines (app.lines), [31](#)
 - App fetch text block (app.txt), [31](#)
- fft
 - Fourier Transform (FFT/DFT) (fao:ft), [76](#)
- FFT (Fourier Transform), [76](#), [81](#)
- fifo
 - cp:fifo (Set the receive FIFO), [177](#)
 - Set the receive FIFO (cp:fifo), [177](#)
- file
 - Archive file (iot.arch), [101](#)
 - Capture data to file (cp:cap), [175](#)
 - Cut the sample file (smp.cut), [191](#)
 - Draw LCD PBM bitmap from file (lcd.pbm), [242](#)
 - File add extension (fc.add), [56](#)
 - File append contents (fc.append), [56](#)
 - File attributes (fc.attr), [57](#)
 - File check update (fc.chkupd), [58](#)
 - File clean flash (fc.clean), [58](#)
 - File copy (fc.copy), [59](#)
 - File delete (fc.del), [59](#)
 - File find (fc.find), [60](#)
 - File get UTC creation (fc.utc), [65](#)
 - File gzip (fc.gz), [61](#)
 - File join dir and file (fc.mdf), [63](#), [63](#)
 - File make directory (fc.mkd), [63](#)
 - File move (fc.move), [63](#)
 - File remove directory (fc.rmd), [64](#)
 - File set cache (fc.cache), [57](#)
 - File set file write mode (fc.sfw), [64](#), [64](#)
 - File split dir and file (fc.sdf), [64](#), [64](#)
 - File split extension (fc.ext), [60](#)
 - File sync to flash (fc.sync), [65](#)
 - File wildcard compare (fc.wc), [65](#)
 - File write Lua value tree (fc.lv), [62](#)
 - IoT is there a stop file (iot.isf), [106](#)
 - Load data from file to KVC object (kvco:_load), [130](#)
 - Move a file (os.move), [10](#)
 - Read floats/doubles from file (float.rd), [71](#)
 - Read floats/doubles from file (float.rf), [71](#)
 - Save data from KVC object to file (kvco:_save), [130](#)
 - Set a BLF file (rt.setblf), [164](#)
 - Use Temp-File-and-Rename for FTP (c.iot_tfr), [277](#)
 - Write floats/doubles to file (float.wd), [73](#)
 - Write floats/doubles to file (float.wf), [73](#)
- fileid
 - Read SIM FileID Binary (modem.simrb), [142](#)
 - Read SIM FileID STATUS (modem.sims), [142](#)
 - Write SIM FileID Binary (modem.simwb), [143](#)
- filename
 - Filename UTC (utc.fn), [209](#)
 - Set UI filename (ui.set.i8n), [264](#)
- files
 - Archive number of files (c.arc_fc), [272](#)
 - Send files to IoT server (iot.send), [108](#)
- fill
 - Fill with sin values (fao:sin), [78](#)

- fin
 - ct:fin (CRC finalise), [47](#)
 - ht:fin (Hash finalise), [95](#)
- finalise
 - CRC finalise (ct:fin), [47](#)
 - Hash finalise (ht:fin), [95](#)
- find
 - fc.find (File find), [60](#)
 - File find (fc.find), [60](#)
 - Find (util.find), [220](#)
 - Find out if LCD hardware present (lcd.ok), [238](#)
 - util.find (Find), [220](#)
- finish
 - Calculate millisecond finish time (rt.msf), [163](#)
- finite
 - Check if finite (float.isfin), [68](#)
- firmware
 - Firmware date (i.fw_date), [295](#)
 - Firmware description (i.fw_desc), [295](#)
 - Firmware version (i.fw_ver), [295](#)
- fix
 - Set/Get GPS fix distance (gps.dist), [90](#)
 - Set/Get GPS fix time (gps.fixt), [90](#)
- fixt
 - gps.fixt (Set/Get GPS fix time), [90](#)
- flash
 - File clean flash (fc.clean), [58](#)
 - File sync to flash (fc.sync), [65](#)
 - Save sample to flash (smp.save), [191](#)
- flg
 - iot.flg (Get IoT reason string), [105](#)
- float
 - Create new float array object (float.array), [66](#)
 - Float to hex (float.f2x), [68](#)
 - Float to integer value (float.f2v), [67](#)
 - float.array (Create new float array object), [66](#)
 - float.d2v (Double to integer), [66](#)
 - float.d2x (Double to hex), [67](#)
 - float.dp (Trim float value), [67](#)
 - float.f2v (Float to integer value), [67](#)
 - float.f2x (Float to hex), [68](#)
 - float.isfin (Check if finite), [68](#)
 - float.isinf (Check if infinite), [68](#)
 - float.isnan (Check if a number), [69](#)
 - float.isnor (Check if a normal number), [69](#)
 - float.mean (Get running statistics values), [69](#)
 - float.mpc (Create multi-point-conversion object), [70](#)
 - float.poly (Calculate n-order polynomial object), [70](#)
 - float.rd (Read floats/doubles from file), [71](#)
 - float.regr (Create a running regression object), [71](#)
 - float.rf (Read floats/doubles from file), [71](#)
 - float.stat (Create a running statistics object), [72](#)
 - float.v2d (Integer to double), [73](#)
 - float.v2f (Integer value to float), [73](#)
 - float.wd (Write floats/doubles to file), [73](#)
 - float.wf (Write floats/doubles to file), [73](#)
 - float.x2d (Hex to double), [74](#)
 - float.x2f (Hex to float), [74](#)
 - Hex to float (float.x2f), [74](#)
 - Integer value to float (float.v2f), [73](#)
 - Trim float value (float.dp), [67](#)
- floats
 - Read floats/doubles from file (float.rd), [71](#)
 - Read floats/doubles from file (float.rf), [71](#)
 - Write floats/doubles to file (float.wd), [73](#)
 - Write floats/doubles to file (float.wf), [73](#)
- flow
 - Set receive flow control (cp:rx), [182](#)
 - Set transmit flow control (cp:tx), [185](#)
- fmt
 - utc.fmt (Format UTC), [209](#)
- fn
 - utc.fn (Filename UTC), [209](#)
- folder
 - Trim archive folder (iot.trim), [110](#)
- font
 - Set LCD font rotation (lcd.fr), [244](#)
 - Set LCD font transparency (lcd.ft), [244](#)
 - Set the LCD font (lcd.f), [244](#)

- Set UI font positions (ui.set.ui), [267](#)
- forbidden
 - Read/Clear Forbidden PLMN List (modem.fplmn), [141](#)
 - Request Clear Forbidden PLMN List (modem.clearfplmn), [140](#)
- format
 - Calculate size of format (struct.size), [201](#)
 - Format UTC (utc.fmt), [209](#)
- fourier
 - Fourier Transform (FFT/DFT) (fao:ft), [76](#)
- Fourier Transform (FFT DFT), [76](#), [81](#)
- fplmn
 - modem.fplmn (Read/Clear Forbidden PLMN List), [141](#)
- fr
 - lcd.fr (Set LCD font rotation), [244](#)
- frame
 - Receive MODBUS RTU frame (modbus.rx), [135](#)
 - Send PDU frame (sms.snd), [199](#)
 - Transmit MODBUS RTU frame (modbus.tx), [137](#)
- free
 - Check free SMS entries (sms.free), [197](#)
 - Configuration bytes free (i.rt_cbf), [298](#)
 - sms.free (Check free SMS entries), [197](#)
- from
 - Convert from x to y (mpco:y), [84](#)
 - Create table from table with bits (util.btab), [217](#)
 - Delete jobs from the queue (jqo:del), [125](#)
 - Draw LCD PBM bitmap from file (lcd.pbm), [242](#)
 - Load data from file to KVC object (kvco:_load), [130](#)
 - Pop a job from the queue (jqo:pop), [126](#)
 - Pop a job from the queue (jqo:poprun), [127](#)
 - Read floats/doubles from file (float.rd), [71](#)
 - Read floats/doubles from file (float.rf), [71](#)
 - Save data from KVC object to file (kvco:_save), [130](#)
- fs2t
 - modbus.fs2t (Unpack string to table), [132](#)
- ft
 - fao:ft (Fourier Transform (FFT/DFT)), [76](#)
 - lcd.ft (Set LCD font transparency), [244](#)
- ft2s
 - modbus.ft2s (Pack table to string), [132](#)
- ftp
 - Use Temp-File-and-Rename for FTP (c.iot_tfr), [277](#)
- full
 - i.pwr_full (Seconds on full power), [301](#)
 - Seconds on full power (i.pwr_full), [301](#)
- function
 - Alarm global callback function (onAlm), [25](#)
 - Atomic function call (rt.atom), [156](#)
 - Coro add function to queue (coro.add), [41](#)
 - Event callback function (eventcallbackname), [53](#)
 - function (Alarm Object state callback), [28](#)
 - SMS callback function (onSMS), [200](#)
- fw
 - i.fw_blv (Bootloader version), [295](#)
 - i.fw_date (Firmware date), [295](#)
 - i.fw_desc (Firmware description), [295](#)
 - i.fw_ver (Firmware version), [295](#)
- G**
- g7a
 - sms.g7a (Convert GSM-7bit to ASCII), [198](#)
- gad
 - lcd.gad (Get the LCD ascent and descent metrics), [245](#)
- gap
 - gps.gap (Set/Get GPS interval), [90](#)
 - Set transmit time guard gap (cp:txtg), [186](#)
- get
 - adc.get (Get ADC sample), [14](#)
 - adco:get (Get ADC Object settings), [16](#)
 - Alarm get (alm.get), [20](#)
 - Alarm get text (alm.txt), [22](#)
 - alm.get (Alarm get), [20](#)
 - cfg.get (Config get), [37](#)
 - Config get (cfg.get), [37](#)
 - Config get boolean (cfg.getb), [38](#)

- Config get number (cfg.getn), [38](#)
- Config get string (cfg.gets), [38](#)
- Coro get ID (coro.id), [42](#)
- Digital pulse get value (dpc.val), [49](#)
- fao:get (Get an array entry), [77](#)
- File get UTC creation (fc.utc), [65](#)
- Get ADC object (adc.obj), [15](#)
- Get ADC Object settings (adco:get), [16](#)
- Get ADC Object Statistics (adco:stat), [17](#)
- Get ADC reading (rt.adc), [155](#)
- Get ADC sample (adc.get), [14](#)
- Get an array entry (fao:get), [77](#)
- Get array size (fao:size), [79](#)
- Get Cortex cycle counter (rt.cyc), [158](#)
- Get CTS line (cp:cts), [177](#)
- Get device details (rt.dev), [159](#)
- Get GPS activity (gps.act), [89](#)
- Get GPS location (gps.loc), [91](#)
- Get highest millivolt (rt.mvh), [164](#)
- Get IoT reason string (iot.flg), [105](#)
- Get job schedule (job.txt), [118](#)
- Get key state (key.get), [229](#)
- Get LCD text width (lcd.gtw), [245](#)
- Get millivolt readings (rt.mv), [163](#)
- Get power manager metrics (pwr.gpm), [151](#)
- Get power off time (utc.offt), [210](#)
- Get range (mpco:range), [82](#)
- Get running statistics values (float.mean), [69](#)
- Get running statistics values (rso:get), [87](#)
- Get seconds past midnight (utc.spm), [213](#)
- Get temperature (rt.temp), [164](#)
- Get the device COM port object (ser.dev), [172](#)
- Get the device COM port object (ser.eng), [172](#)
- Get the LCD ascent and descent metrics (lcd.gad), [245](#)
- Get the LCD screen size (lcd.gss), [236](#)
- Get the LCD screen size (lcd.size), [236](#)
- Get the protocol settings (cp:prot), [179](#)
- Get the top UI element (ui.top), [261](#)
- Get time zone (utc.tz), [213](#)
- Get UI element text (ui.txt), [262](#)
- Get whole days (utc.wds), [214](#)
- Get x
 - y pair (mpco:xy), [83](#)
- Get x and y tables (mpco:xyt), [83](#)
- Get/set ADC Object power control (adco:pwr), [17](#)
- Get/Set Real Time Clock (utc.rtc), [211](#)
- IoT get byte counts (iot.bytes), [102](#)
- IoT get hold off (iot.gho), [105](#)
- key.get (Get key state), [229](#)
- out.get (Read output state), [146](#)
- rso:get (Get running statistics values), [87](#)
- Set or get ADC parameters (adc.set), [15](#)
- Set or get counter (smp.cnt), [190](#)
- Set or get job variance (job.var), [118](#)
- Set or get LCD idle time (lcd.idle), [236](#)
- Set/Get GPS fix distance (gps.dist), [90](#)
- Set/Get GPS fix time (gps.fixt), [90](#)
- Set/Get GPS interval (gps.gap), [90](#)
- Set/Get overall GPS duration (gps.time), [92](#)
- getb
 - cfg.getb (Config get boolean), [38](#)
- getn
 - cfg.getn (Config get number), [38](#)
- gets
 - cfg.gets (Config get string), [38](#)
- gho
 - iot.gho (IoT get hold off), [105](#)
- global
 - Alarm global callback function (onAlm), [25](#)
 - Key global callback (onKey), [232](#)
- gly
 - lcd.gly (Draw LCD glyph), [245](#)
- glyph
 - Draw LCD glyph (lcd.gly), [245](#)
- go
 - adc.go (Start ADC), [14](#)
 - adco:go (Start ADC Object conversion), [16](#)
 - gps.go (Start GPS), [91](#)
 - iot.go (Trigger IoT connection), [105](#)
 - lcd.go (Display the current LCD buffer), [236](#)
 - sdi12.go (Run SDI-12 sequence), [169](#)
 - sms.go (Trigger SMS exchange), [198](#)

- ui.go (Start the UI), [260](#)
- gpm
 - pwr.gpm (Get power manager metrics), [151](#)
- gps
 - Get GPS activity (gps.act), [89](#)
 - Get GPS location (gps.loc), [91](#)
 - GPS location available callback (onGPS), [93](#)
 - gps.act (Get GPS activity), [89](#)
 - gps.dist (Set/Get GPS fix distance), [90](#)
 - gps.fixt (Set/Get GPS fix time), [90](#)
 - gps.gap (Set/Get GPS interval), [90](#)
 - gps.go (Start GPS), [91](#)
 - gps.loc (Get GPS location), [91](#)
 - gps.rst (Reset the GPS counter), [91](#)
 - gps.time (Set/Get overall GPS duration), [92](#)
 - Reset the GPS counter (gps.rst), [91](#)
 - Set/Get GPS fix distance (gps.dist), [90](#)
 - Set/Get GPS fix time (gps.fixt), [90](#)
 - Set/Get GPS interval (gps.gap), [90](#)
 - Set/Get overall GPS duration (gps.time), [92](#)
 - Start GPS (gps.go), [91](#)
- gsm
 - Convert ASCII to GSM-7bit (sms.ag7), [196](#)
 - Convert GSM-7bit to ASCII (sms.g7a), [198](#)
- gss
 - lcd.gss (Get the LCD screen size), [236](#)
- gtw
 - lcd.gtw (Get LCD text width), [245](#)
- guard
 - Set transmit time guard gap (cp:txtg), [186](#)
- gz
 - c.arc_gz (Archive use gzip), [272](#)
 - c.iot_gz (Use gzip compression), [276](#)
 - fc.gz (File gzip), [61](#)
- gzip
 - Archive use gzip (c.arc_gz), [272](#)
 - File gzip (fc.gz), [61](#)
 - Use gzip compression (c.iot_gz), [276](#)
- H**
- handle
 - Handle a command sequence (cp:cmd), [176](#)
- hang
 - IoT hang up modem (iot.hum), [106](#)
- hard
 - Allow hard work (pwr.work), [153](#)
- hardware
 - Check keyboard hardware (key.ok), [231](#)
 - Find out if LCD hardware present (lcd.ok), [238](#)
 - Initialise LCD hardware (lcd.init), [237](#)
- hash
 - Convert hash string to value (hash.val), [95](#)
 - Convert hash value to text (hash.txt), [95](#)
 - Create new hash object (hash.new), [94](#)
 - Hash finalise (ht:fin), [95](#)
 - Hash string immediately (hash.hmac256), [94](#)
 - Hash string immediately (hash.md5), [94](#)
 - Hash string immediately (hash.sha1), [94](#)
 - Hash string immediately (hash.sha256), [94](#)
 - Hash update (ht:upd), [96](#)
 - hash.hmac256 (Hash string immediately), [94](#)
 - hash.md5 (Hash string immediately), [94](#)
 - hash.new (Create new hash object), [94](#)
 - hash.sha1 (Hash string immediately), [94](#)
 - hash.sha256 (Hash string immediately), [94](#)
 - hash.txt (Convert hash value to text), [95](#)
 - hash.val (Convert hash string to value), [95](#)
- hex
 - Convert ASCII hex to binary (rt.bin), [156](#)
 - Convert binary to ASCII hex (rt.hex), [160](#)
 - Double to hex (float.d2x), [67](#)
 - Float to hex (float.f2x), [68](#)
 - Hex to double (float.x2d), [74](#)
 - Hex to float (float.x2f), [74](#)
 - rt.hex (Convert binary to ASCII hex), [160](#)
- highest
 - Get highest millivolt (rt.mvh), [164](#)
- hmac256
 - hash.hmac256 (Hash string immediately), [94](#)
- hof
 - c.cell_hof (Hold off failure time), [273](#)
- hold
 - Hold off failure time (c.cell_hof), [273](#)

- Hold off success time (c.cell_hos), [273](#)
- IoT clear hold off (iot.cho), [103](#)
- IoT get hold off (iot.gho), [105](#)
- IoT set modem hold off (iot.sho), [110](#)
- holding
 - Read MODBUS holding registers (modbus.rhr), [133](#)
- horizontal
 - Draw LCD horizontal line (lcd.lh), [241](#)
- hos
 - c.cell_hos (Hold off success time), [273](#)
- ht
 - ht:fin (Hash finalise), [95](#)
 - ht:upd (Hash update), [96](#)
- hum
 - iot.hum (IoT hang up modem), [106](#)
- hydrins
 - HydrINS parse text (hydrins.parse), [97](#)
 - HydrINS receive (hydrins.rx), [98](#)
 - hydrins.cmd (Send command to HydrINS), [97](#)
 - hydrins.parse (HydrINS parse text), [97](#)
 - hydrins.run (Tell HydrINS to run), [98](#)
 - hydrins.rx (HydrINS receive), [98](#)
 - hydrins.wake (Wake up HydrINS), [99](#)
 - Send command to HydrINS (hydrins.cmd), [97](#)
 - Tell HydrINS to run (hydrins.run), [98](#)
 - Wake up HydrINS (hydrins.wake), [99](#)
- I**
- i8n
 - ui.set.i8n (Set UI filename), [264](#)
- icon
 - SET key icon (ui.ssk), [261](#)
- id
 - Coro get ID (coro.id), [42](#)
 - coro.id (Coro get ID), [42](#)
 - Query ID (sdi12.id), [169](#)
 - Read server ID (modbus.rsid), [135](#)
 - sdi12.id (Query ID), [169](#)
- idle
 - IoT Modem Idle callback (iotIdle), [113](#)
 - Keyboard idle callback (onKeyIdle), [233](#)
- lcd.idle (Set or get LCD idle time), [236](#)
- Set keyboard repeat idle (key.kri), [231](#)
- Set or get LCD idle time (lcd.idle), [236](#)
- Set the idle timers (pwr._iv), [154](#)
- Set UI idle time (ui.set.idle), [264](#)
- TOA idle timeout (c.toa_ito), [286](#)
- ui.set.idle (Set UI idle time), [264](#)
- ign
 - key.ign (Ignore keys), [229](#)
- ignore
 - Ignore keys (key.ign), [229](#)
 - TLS ignore local time (c.tls_ilt), [284](#)
- ilt
 - c.tls_ilt (TLS ignore local time), [284](#)
- immediately
 - Hash string immediately (hash.hmac256), [94](#)
 - Hash string immediately (hash.md5), [94](#)
 - Hash string immediately (hash.sha1), [94](#)
 - Hash string immediately (hash.sha256), [94](#)
- include
 - Include Lua library (util.use), [226](#)
- index
 - Append index table (util.tia), [223](#)
- indexed
 - Add indexed tables (util.addi), [215](#)
 - Compare indexed tables (util.cmpi), [218](#)
 - Cut indexed table (util.cut), [219](#)
 - OR indexed tables (util.tio), [224](#)
 - Remove indexed table (util.remi), [222](#)
- indicator
 - Modem signal quality indicator (i.cell_X_csq), [292](#)
- infinite
 - Check if infinite (float.isinf), [68](#)
- info
 - info (TERM), [308](#), [318](#)
 - Send diagnostics info to IoT server (iot.diag), [104](#)
- inidicator
 - TLS Server-Name-Inidicator (c.tls_sni), [284](#)
- init
 - c.cell_init (Modem init sequence), [274](#)
 - evt.init (Event initialise), [50](#)

- lcd.init (Initialise LCD hardware), [237](#)
- Modem init sequence (c.cell_init), [274](#)
- out.init (Initialise outputs), [146](#)
- initialise
 - Event initialise (evt.init), [50](#)
 - Initialise LCD hardware (lcd.init), [237](#)
 - Initialise outputs (out.init), [146](#)
 - IoT Modem Initialise callback (iotInit), [114](#)
 - IoT Modem Initialise callback (iotReg), [114](#)
- inj
 - key.inj (Inject keystroke), [230](#)
- inject
 - Inject keystroke (key.inj), [230](#)
- input
 - Read input registers (modbus.rir), [134](#)
- inputs
 - Read discrete inputs (modbus.rdi), [133](#)
- ins
 - c.user_ins (Interval for user mode), [288](#)
- integer
 - Double to integer (float.d2v), [66](#)
 - Float to integer value (float.f2v), [67](#)
 - Integer to double (float.v2d), [73](#)
 - Integer value to float (float.v2f), [73](#)
- internet
 - Internet APN name (c.cell_apn), [273](#)
 - Internet password (c.cell_pass), [274](#)
 - Internet username (c.cell_user), [275](#)
- interval
 - Interval for user mode (c.user_ins), [288](#)
 - Set/Get GPS interval (gps.gap), [90](#)
- inv
 - key.inv (Set key inversion), [230](#)
- inversion
 - Set key inversion (key.inv), [230](#)
- iot
 - c.iot_data (Data process URL or path), [276](#)
 - c.iot_gz (Use gzip compression), [276](#)
 - c.iot_job (Job schedule), [276](#)
 - c.iot_ntp (NTP server), [276](#)
 - c.iot_tfr (Use Temp-File-and-Rename for FTP), [277](#)
 - c.iot_tz (Time-zone override), [277](#)
 - c.iot_ujob (Update job schedule), [277](#)
 - c.iot_upd (Update URL or path), [278](#)
 - c.iot_url (Base URL), [278](#)
 - c.iot_user (User process URL or path), [278](#)
 - c.iot_var (Push variance), [279](#)
 - Get IoT reason string (iot.flg), [105](#)
 - i.iot_var (Variance time applied), [296](#)
 - IoT abort modem requests (iot.abort), [101](#)
 - IoT abort user process (iot.aup), [102](#)
 - IoT allowed to run (iot.dorun), [104](#)
 - IoT clear hold off (iot.cho), [103](#)
 - IoT data callback (iotData), [113](#)
 - IoT get byte counts (iot.bytes), [102](#)
 - IoT get hold off (iot.gho), [105](#)
 - IoT hang up modem (iot.hum), [106](#)
 - IoT is there a stop file (iot.isf), [106](#)
 - IoT Modem Complete callback (iotOff), [114](#)
 - IoT Modem Idle callback (iotIdle), [113](#)
 - IoT Modem Initialise callback (iotInit), [114](#)
 - IoT Modem Initialise callback (iotReg), [114](#)
 - IoT Modem Offline callback (iotOffline), [114](#)
 - IoT Modem Online callback (iotOnline), [114](#)
 - IoT Modem Starting callback (iotSetup), [115](#)
 - IoT set modem hold off (iot.sho), [110](#)
 - IoT set PKI certificate (iot.cert), [103](#)
 - IoT test callback (iotTest), [115](#)
 - IoT update callback (iotUpd), [115](#)
 - IoT user callback (iotUser), [115](#)
 - iot.abort (IoT abort modem requests), [101](#)
 - iot.arch (Archive file), [101](#)
 - iot.aup (IoT abort user process), [102](#)
 - iot.azt (Set modem anti-cell-zombie timeout), [102](#)
 - iot.bytes (IoT get byte counts), [102](#)
 - iot.cert (IoT set PKI certificate), [103](#)
 - iot.cho (IoT clear hold off), [103](#)
 - iot.diag (Send diagnostics info to IoT server), [104](#)
 - iot.dorun (IoT allowed to run), [104](#)
 - iot.flg (Get IoT reason string), [105](#)
 - iot.gho (IoT get hold off), [105](#)
 - iot.go (Trigger IoT connection), [105](#)
 - iot.hum (IoT hang up modem), [106](#)

- iot.isf (IoT is there a stop file), [106](#)
- iot.modem (Send modem command), [107](#)
- iot.scmv (Set modem stay-connected millivolt), [107](#)
- iot.send (Send files to IoT server), [108](#)
- iot.set (Set IoT server URL), [109](#)
- iot.sho (IoT set modem hold off), [110](#)
- iot.trim (Trim archive folder), [110](#)
- Send diagnostics info to IoT server (iot.diag), [104](#)
- Send files to IoT server (iot.send), [108](#)
- Set IoT server URL (iot.set), [109](#)
- Trigger IoT connection (iot.go), [105](#)
- iotdata
 - iotData (IoT data callback), [113](#)
- iotidle
 - iotIdle (IoT Modem Idle callback), [113](#)
- iotinit
 - iotInit (IoT Modem Initialise callback), [114](#)
- iotoff
 - iotOff (IoT Modem Complete callback), [114](#)
- iotoffline
 - iotOffline (IoT Modem Offline callback), [114](#)
- iotonline
 - iotOnline (IoT Modem Online callback), [114](#)
- iotreg
 - iotReg (IoT Modem Initialise callback), [114](#)
- iotsetup
 - iotSetup (IoT Modem Starting callback), [115](#)
- iottest
 - iotTest (IoT test callback), [115](#)
- iotupd
 - iotUpd (IoT update callback), [115](#)
- iotuser
 - iotUser (IoT user callback), [115](#)
- isf
 - iot.isf (IoT is there a stop file), [106](#)
- isfin
 - float.isfin (Check if finite), [68](#)
- isinf
 - float.isinf (Check if infinite), [68](#)
- isnan
 - float.isnan (Check if a number), [69](#)
- isnor
 - float.isnor (Check if a normal number), [69](#)
- iter
 - util.iter (Iterate table), [221](#)
- iterate
 - Iterate table (util.iter), [221](#)
- ito
 - c.toa_ito (TOA idle timeout), [286](#)
- iv
 - pwr._iv (Set the idle timers), [154](#)
- J**
- job
 - Add callback to job queue (jqo:add), [125](#)
 - c.iot_job (Job schedule), [276](#)
 - Check if callback in job queue (jqo:is), [126](#)
 - Create a new Job Queue Object (jobq.new), [123](#)
 - Get job schedule (job.txt), [118](#)
 - Job schedule (c.iot_job), [276](#)
 - job.set (Set job schedule), [117](#)
 - job.txt (Get job schedule), [118](#)
 - job.utc (Set a UTC based job), [120](#)
 - job.var (Set or get job variance), [118](#)
 - Pop a job from the queue (jqo:pop), [126](#)
 - Pop a job from the queue (jqo:poprun), [127](#)
 - Set a UTC based job (job.utc), [120](#)
 - Set job schedule (job.set), [117](#)
 - Set or get job variance (job.var), [118](#)
 - Update job schedule (c.iot_ujob), [277](#)
 - utc.job, [210](#)
- jobq
 - jobq.new (Create a new Job Queue Object), [123](#)
- jobs
 - Delete jobs from the queue (jqo:del), [125](#)
 - Run all jobs in the queue (jqo:run), [127](#)
- join
 - File join dir and file (fc.mdf), [63](#)
- journalling
 - Execute with journalling (fc.jx), [62](#)
 - Journalling supported (fc.js), [61](#)
- jqo

- jgo:add (Add callback to job queue), [125](#)
- jgo:del (Delete jobs from the queue), [125](#)
- jgo:is (Check if callback in job queue), [126](#)
- jgo:pop (Pop a job from the queue), [126](#)
- jgo:poprun (Pop a job from the queue), [127](#)
- jgo:run (Run all jobs in the queue), [127](#)
- js
- fc.js (Journalling supported), [61](#)
- jx
- fc.jx (Execute with journalling), [62](#)
- K**
- kb
- c.arc_kb (Archive maximum kbytes), [272](#)
- kbytes
- Archive maximum kbytes (c.arc_kb), [272](#)
- key
- c.KEY (TERM), [310](#)
- c.toa_key (TOA secret key), [286](#)
- cfg.key.chk (Check a secret key), [40](#)
- cfg.key.set (Set a secret key), [40](#)
- cfg.key.wipe (Erase a secret key), [40](#)
- Check a secret key (cfg.key.chk), [40](#)
- Create key-value container object (util.kvc), [222](#)
- Create new key-value container object (util.kvc), [128](#)
- Draw soft key (ui.key), [260](#)
- Erase a secret key (cfg.key.wipe), [40](#)
- Get key state (key.get), [229](#)
- Key global callback (onKey), [232](#)
- key.get (Get key state), [229](#)
- key.ign (Ignore keys), [229](#)
- key.inj (Inject keystroke), [230](#)
- key.inv (Set key inversion), [230](#)
- key.kre (Set key repeat enable), [230](#)
- key.kri (Set keyboard repeat idle), [231](#)
- key.ok (Check keyboard hardware), [231](#)
- Set a secret key (cfg.key.set), [40](#)
- SET key icon (ui.ssk), [261](#)
- Set key inversion (key.inv), [230](#)
- Set key repeat enable (key.kre), [230](#)
- Set UI key mapping (ui.set.km), [265](#)
- Set UI key repeat (ui.set.kr), [265](#)
- TOA secret key (c.toa_key), [286](#)
- ui.key (Draw soft key), [260](#)
- keyboard
- Check keyboard hardware (key.ok), [231](#)
- Keyboard idle callback (onKeyIdle), [233](#)
- Set keyboard repeat idle (key.kri), [231](#)
- keys
- Ignore keys (key.ign), [229](#)
- keystroke
- Inject keystroke (key.inj), [230](#)
- kill
- Coro kill routine (coro.kill), [42](#)
- coro.kill (Coro kill routine), [42](#)
- Kill modem (modem.kill), [142](#)
- modem.kill (Kill modem), [142](#)
- km
- ui.set.km (Set UI key mapping), [265](#)
- kr
- ui.set.kr (Set UI key repeat), [265](#)
- kre
- key.kre (Set key repeat enable), [230](#)
- kri
- key.kri (Set keyboard repeat idle), [231](#)
- kvc
- Load data from file to KVC object (kvco:_load), [130](#)
- Save data from KVC object to file (kvco:_save), [130](#)
- util.kvc (Create key-value container object), [222](#)
- util.kvc (Create new key-value container object), [128](#)
- kvco
- kvco:_load (Load data from file to KVC object), [130](#)
- kvco:_save (Save data from KVC object to file), [130](#)
- L**
- last
- Last sample text (i.smp.txt), [300](#)
- Last sample time (i.smp.utc), [300](#)
- lboot
- lboot (TERM), [309](#)

lcd

Check if the LCD is on (lcd.act), [235](#)
 Control LCD backlight (ui.set.bklt), [263](#)
 Control LCD state (lcd.on), [238](#)
 Display the current LCD buffer (lcd.go), [236](#)
 Draw LCD box (lcd.box), [240](#)
 Draw LCD circle or ellipse (lcd.cir), [240](#)
 Draw LCD glyph (lcd.gly), [245](#)
 Draw LCD horizontal line (lcd.lh), [241](#)
 Draw LCD line (lcd.l), [241](#)
 Draw LCD PBM bitmap from file (lcd.pbm), [242](#)
 Draw LCD pixel (lcd.p), [242](#)
 Draw LCD text (lcd.t), [246](#)
 Draw LCD triangle (lcd.tri), [243](#)
 Draw LCD vertical line (lcd.lv), [241](#)
 Draw LCD XBM bitmap string (lcd.xbm), [243](#)
 Erase the LCD buffer (lcd.ers), [235](#)
 Find out if LCD hardware present (lcd.ok), [238](#)
 Get LCD text width (lcd.gtw), [245](#)
 Get the LCD ascent and descent metrics (lcd.gad), [245](#)
 Get the LCD screen size (lcd.gss), [236](#)
 Get the LCD screen size (lcd.size), [236](#)
 Initialise LCD hardware (lcd.init), [237](#)
 lcd.act (Check if the LCD is on), [235](#)
 lcd.box (Draw LCD box), [240](#)
 lcd.c (Set LCD draw colour), [235](#)
 lcd.cir (Draw LCD circle or ellipse), [240](#)
 lcd.ers (Erase the LCD buffer), [235](#)
 lcd.f (Set the LCD font), [244](#)
 lcd.fr (Set LCD font rotation), [244](#)
 lcd.ft (Set LCD font transparency), [244](#)
 lcd.gad (Get the LCD ascent and descent metrics), [245](#)
 lcd.gly (Draw LCD glyph), [245](#)
 lcd.go (Display the current LCD buffer), [236](#)
 lcd.gss (Get the LCD screen size), [236](#)
 lcd.gtw (Get LCD text width), [245](#)
 lcd.idle (Set or get LCD idle time), [236](#)
 lcd.init (Initialise LCD hardware), [237](#)
 lcd.l (Draw LCD line), [241](#)

lcd.lh (Draw LCD horizontal line), [241](#)
 lcd.lv (Draw LCD vertical line), [241](#)
 lcd.off (Turn off the LCD), [237](#)
 lcd.ok (Find out if LCD hardware present), [238](#)
 lcd.on (Control LCD state), [238](#)
 lcd.p (Draw LCD pixel), [242](#)
 lcd.pbm (Draw LCD PBM bitmap from file), [242](#)
 lcd.save (Save the LCD to a bitmap), [238](#)
 lcd.sdc (Set LCD display contrast), [239](#)
 lcd.size (Get the LCD screen size), [236](#)
 lcd.t (Draw LCD text), [246](#)
 lcd.tri (Draw LCD triangle), [243](#)
 lcd.xbm (Draw LCD XBM bitmap string), [243](#)
 Save the LCD to a bitmap (lcd.save), [238](#)
 Set LCD display contrast (lcd.sdc), [239](#)
 Set LCD draw colour (lcd.c), [235](#)
 Set LCD font rotation (lcd.fr), [244](#)
 Set LCD font transparency (lcd.ft), [244](#)
 Set or get LCD idle time (lcd.idle), [236](#)
 Set the LCD font (lcd.f), [244](#)
 Show values on LCD (ui.val), [262](#)
 Turn off the LCD (lcd.off), [237](#)

lck

cp:lck (Lock the COM port), [177](#)

led

Enable debug LED sequence (rt.led), [161](#)
 led (TERM), [304](#)
 rt.led (Enable debug LED sequence), [161](#)

left

Bitwise left shift (bit.ls), [34](#)
 Bitwise left shift (bit.lshift), [34](#)

level

Digital pulse current level (dpc.lvl), [48](#)

lf

lf (TERM), [311](#)

lgc

lgc (TERM), [309](#)

lgc0

lgc0 (TERM), [309](#)

lh

lcd.lh (Draw LCD horizontal line), [241](#)

- library
 - Include Lua library (util.use), [226](#)
- limit
 - Cell registration time limit (c.cell_reg), [274](#)
- line
 - Convert NMEA line to table (nmea.tab), [144](#)
 - Draw LCD horizontal line (lcd.lh), [241](#)
 - Draw LCD line (lcd.l), [241](#)
 - Draw LCD vertical line (lcd.lv), [241](#)
 - Get CTS line (cp:cts), [177](#)
 - Set RTS line (cp:rts), [181](#)
- lines
 - App fetch lines (app.lines), [31](#)
 - app.lines (App fetch lines), [31](#)
- list
 - Read/Clear Forbidden PLMN List (modem.fplmn), [141](#)
 - Request Clear Forbidden PLMN List (modem.clearfplmn), [140](#)
- lm
 - lm (TERM), [309](#)
- lm0
 - lm0 (TERM), [309](#)
- load
 - kvco:_load (Load data from file to KVC object), [130](#)
 - Load data from file to KVC object (kvco:_load), [130](#)
- loc
 - c.site_loc (Site location string), [281](#)
 - gps.loc (Get GPS location), [91](#)
- local
 - TLS ignore local time (c.tls_ilt), [284](#)
- location
 - Get GPS location (gps.loc), [91](#)
 - GPS location available callback (onGPS), [93](#)
 - Site location string (c.site_loc), [281](#)
- lock
 - Lock the COM port (cp:lck), [177](#)
- log
 - Brief modem activity log (i.cell_X_log), [293](#)
 - i.cell_X_log (Brief modem activity log), [293](#)
 - rt.log (Write log entry), [161](#)
 - Send log string to USB (usb.log), [208](#)
 - usb.log (Send log string to USB), [208](#)
 - Write log entry (rt.log), [161](#)
- ls
 - bit.ls (Bitwise left shift), [34](#)
- lshift
 - bit.lshift (Bitwise left shift), [34](#)
- lsig
 - evt.lsig (Event signal), [51](#)
- lua
 - c.lua_app (Lua app name), [280](#)
 - Estimate Lua codesize (util.codesize), [218](#)
 - File write Lua value tree (fc.lv), [62](#)
 - i.lua_run (Lua app start point), [297](#)
 - Include Lua library (util.use), [226](#)
 - Lua app name (c.lua_app), [280](#)
 - Lua app start point (i.lua_run), [297](#)
 - Reboot Lua (rt.luaboot), [161](#)
- luaboot
 - rt.luaboot (Reboot Lua), [161](#)
- lv
 - fc.lv (File write Lua value tree), [62](#)
 - lcd.lv (Draw LCD vertical line), [241](#)
 - lv (TERM), [310](#)
- lvl
 - dpc.lvl (Digital pulse current level), [48](#)
- lwait
 - evt.lwait (Event wait), [51](#)
- M**
- mag
 - rt.mag (Set magnet delay), [162](#)
- magnet
 - Set magnet delay (rt.mag), [162](#)
- make
 - File make directory (fc.mkd), [63](#)
- manager
 - Get power manager metrics (pwr.gpm), [151](#)
- mapping
 - Set UI key mapping (ui.set.km), [265](#)
- mark
 - Send mark (sdi12.mrk), [169](#)
- maximum
 - Archive maximum kbytes (c.arc_kb), [272](#)

- Maximum time online (c.cell_mto), [274](#)
- md5
 - hash.md5 (Hash string immediately), [94](#)
- mdf
 - fc.mdf (File join dir and file), [63](#)
- mean
 - float.mean (Get running statistics values), [69](#)
- method
 - Override APN authentication method (c.cell_auth), [273](#)
- metrics
 - Get power manager metrics (pwr.gpm), [151](#)
 - Get the LCD ascent and descent metrics (lcd.gad), [245](#)
- microseconds
 - Delay microseconds (rt.us), [165](#)
- midnight
 - Get seconds past midnight (utc.spm), [213](#)
- millisecond
 - Calculate millisecond finish time (rt.msf), [163](#)
 - Check millisecond expiry (rt.msx), [163](#)
- milliseconds
 - Delay milliseconds (rt.ms), [162](#)
- millivolt
 - Get highest millivolt (rt.mvh), [164](#)
 - Get millivolt readings (rt.mv), [163](#)
 - Set modem stay-connected millivolt (iot.scmv), [107](#)
- minimum
 - Query the transient minimum voltage (pwr.tmv), [153](#)
- mkd
 - fc.mkd (File make directory), [63](#)
- modbus
 - modbus.cmd (Send MODBUS command), [131](#)
 - modbus.fs2t (Unpack string to table), [132](#)
 - modbus.ft2s (Pack table to string), [132](#)
 - modbus.rdi (Read discrete inputs), [133](#)
 - modbus.rhr (Read MODBUS holding registers), [133](#)
 - modbus.rir (Read input registers), [134](#)
 - modbus.rmc (Read multiple coils), [134](#)
 - modbus.rsid (Read server ID), [135](#)
 - modbus.rx (Receive MODBUS RTU frame), [135](#)
 - modbus.set (Setup MODBUS on COM port), [136](#)
 - modbus.time (Return MODBUS timings), [136](#)
 - modbus.tx (Transmit MODBUS RTU frame), [137](#)
 - modbus.wmr (Write MODBUS multiple registers), [137](#)
 - modbus.wsr (Write single register), [138](#)
 - Read MODBUS holding registers (modbus.rhr), [133](#)
 - Receive MODBUS RTU frame (modbus.rx), [135](#)
 - Return MODBUS timings (modbus.time), [136](#)
 - Send MODBUS command (modbus.cmd), [131](#)
 - Setup MODBUS on COM port (modbus.set), [136](#)
 - Transmit MODBUS RTU frame (modbus.tx), [137](#)
 - Write MODBUS multiple registers (modbus.wmr), [137](#)
- mode
 - File set file write mode (fc.sfw), [64](#)
 - Interval for user mode (c.user_ins), [288](#)
 - Set the RS232/RS485 mode (cp:rs), [181](#)
 - User mode online time (c.user_onm), [288](#)
- modem
 - Brief modem activity log (i.cell_X_log), [293](#)
 - IoT abort modem requests (iot.abort), [101](#)
 - IoT hang up modem (iot.hum), [106](#)
 - IoT Modem Complete callback (iotOff), [114](#)
 - IoT Modem Idle callback (iotIdle), [113](#)
 - IoT Modem Initialise callback (iotInit), [114](#)
 - IoT Modem Initialise callback (iotReg), [114](#)
 - IoT Modem Offline callback (iotOffline), [114](#)
 - IoT Modem Online callback (iotOnline), [114](#)

- IoT Modem Starting callback (iotSetup), [115](#)
- IoT set modem hold off (iot.sho), [110](#)
- iot.modem (Send modem command), [107](#)
- Kill modem (modem.kill), [142](#)
- modem (TERM), [313](#)
- Modem activity phase (i.cell_pha), [291](#)
- Modem init sequence (c.cell_init), [274](#)
- Modem online duration (i.cell_X_od), [293](#)
- Modem power duration (i.cell_pwr), [291](#)
- Modem power duration (i.cell_X_pd), [293](#)
- Modem power up time (i.cell_X_pu), [294](#)
- Modem registration seconds (i.cell_X_rs), [294](#)
- Modem server connected (i.cell_X_sc), [294](#)
- Modem signal quality indicator (i.cell_X_csq), [292](#)
- Modem state (i.cell_sta), [292](#)
- modem.clearfplmn (Request Clear Forbidden PLMN List), [140](#)
- modem.cmd (Send modem command), [140](#)
- modem.fplmn (Read/Clear Forbidden PLMN List), [141](#)
- modem.kill (Kill modem), [142](#)
- modem.simrb (Read SIM FileID Binary), [142](#)
- modem.sims (Read SIM FileID STATUS), [142](#)
- modem.simwb (Write SIM FileID Binary), [143](#)
- modem.stot (Convert SIM STATUS to table), [143](#)
- Send modem command (iot.modem), [107](#)
- Send modem command (modem.cmd), [140](#)
- Set modem anti-cell-zombie timeout (iot.azt), [102](#)
- Set modem stay-connected millivolt (iot.scmv), [107](#)
- move
 - fc.move (File move), [63](#)
 - File move (fc.move), [63](#)
 - Move a file (os.move), [10](#)
 - os.move (Move a file), [10](#)
- mpc
 - float.mpc (Create multi-point-conversion object), [70](#)
- mpco
 - mpco:oob (Set out-of-bound values), [82](#)
 - mpco:range (Get range), [82](#)
 - mpco:xy (Get x y pair), [83](#)
 - mpco:xyt (Get x and y tables), [83](#)
 - mpco:y (Convert from x to y), [84](#)
- mrk
 - sdi12.mrk (Send mark), [169](#)
- ms
 - rt.ms (Delay milliseconds), [162](#)
- msf
 - rt.msf (Calculate millisecond finish time), [163](#)
- msx
 - rt.msx (Check millisecond expiry), [163](#)
- mt0
 - c.cell_mto (Maximum time online), [274](#)
- mul
 - add/sub/mul/div values (fao:add), [75](#)
 - add/sub/mul/div values (fao:div), [75](#)
 - add/sub/mul/div values (fao:mul), [75](#)
 - add/sub/mul/div values (fao:sub), [75](#)
 - fao:mul (add/sub/mul/div values), [75](#)
- multi
 - Create multi-point-conversion object (float.mpc), [70](#)
- multiple
 - Read multiple coils (modbus.rmc), [134](#)
 - Write MODBUS multiple registers (modbus.wmr), [137](#)
- multiply
 - Multiply-and-sum two arrays (fao:xs), [81](#)
- mv
 - mv (TERM), [304](#)
 - rt.mv (Get millivolt readings), [163](#)
- mvh
 - rt.mvh (Get highest millivolt), [164](#)
- N
- name
 - c.site_name (Site name), [281](#)
 - Internet APN name (c.cell_apn), [273](#)

- Lua app name (c.lua_app), [280](#)
- Site name (c.site_name), [281](#)
- TLS Server-Name-Indicator (c.tls_sni), [284](#)
- names
 - Alarm text names (c.alm_txt01), [271](#)
 - Alarm text names (c.alm_txt02), [271](#)
- new
 - alm.new (Create alarm object), [21](#)
 - CRC new object (crc.new), [46](#)
 - crc.new (CRC new object), [46](#)
 - Create a new Job Queue Object (jobq.new), [123](#)
 - Create new float array object (float.array), [66](#)
 - Create new hash object (hash.new), [94](#)
 - Create new key-value container object (util.kvc), [128](#)
 - hash.new (Create new hash object), [94](#)
 - jobq.new (Create a new Job Queue Object), [123](#)
- nfcheck
 - nfcheck (TERM), [314](#)
- nfdel
 - nfdel (TERM), [314](#)
- nfdir
 - nfdir (TERM), [314](#)
- nfformat
 - nfformat (TERM), [314](#)
- nfinfo
 - nfinfo (TERM), [315](#)
- nfmkdir
 - nfmkdir (TERM), [315](#)
- nfmove
 - nfmove (TERM), [315](#)
- nfrename
 - nfrename (TERM), [315](#)
- nfrmdir
 - nfrmdir (TERM), [316](#)
- nftype
 - nftype (TERM), [316](#)
- nfwipedir
 - nfwipedir (TERM), [316](#)
- nfwrite
 - nfwrite (TERM), [316](#)
- nibbles
 - Bit swap nibbles (bit.swap8), [36](#)
- ninv
 - ninv (TERM), [304](#)
- nmea
 - Convert NMEA line to table (nmea.tab), [144](#)
 - Convert table to NMEA string (nmea.txt), [145](#)
 - nmea.tab (Convert NMEA line to table), [144](#)
 - nmea.txt (Convert table to NMEA string), [145](#)
- normal
 - Check if a normal number (float.isnor), [69](#)
- not
 - Bitwise NOT (bit.bnot), [33](#)
- now
 - utc.now (Return current UTC), [210](#)
- ntp
 - c.iot_ntp (NTP server), [276](#)
 - NTP server (c.iot_ntp), [276](#)
- number
 - Archive number of files (c.arc_fc), [272](#)
 - Check if a normal number (float.isnor), [69](#)
 - Check if a number (float.isnan), [69](#)
 - Config get number (cfg.getn), [38](#)
 - Coro number of routines running (coro.n), [42](#)
 - Default SMS number (c.cell_sms), [275](#)
 - Number of RX bytes waiting (cp:rxw), [184](#)
 - Number of samples (i.smp.cnt), [300](#)
 - Original serial number (i.rt_so), [298](#)
 - Serial number (i.rt_sn), [298](#)
- O**
- obj
 - adc.obj (Get ADC object), [15](#)
- object
 - Alarm Object clear (ao:clr), [26](#)
 - Alarm Object state callback (function), [28](#)
 - Alarm Object update (ao:upd), [27](#)
 - Alarm Object value (ao:val), [28](#)
 - Calculate n-order polynomial object (float.poly), [70](#)
 - Clear running regression object (rro:clear),

- 86
- Clear running stats object (rso:clear), [87](#)
- Coro exception object (coro.except), [45](#)
- CRC new object (crc.new), [46](#)
- Create a new Job Queue Object (jobq.new), [123](#)
- Create a running regression object (float.regr), [71](#)
- Create a running statistics object (float.stat), [72](#)
- Create alarm object (alm.new), [21](#)
- Create key-value container object (util.kvc), [222](#)
- Create multi-point-conversion object (float.mpc), [70](#)
- Create new float array object (float.array), [66](#)
- Create new hash object (hash.new), [94](#)
- Create new key-value container object (util.kvc), [128](#)
- Get ADC object (adc.obj), [15](#)
- Get ADC Object settings (adco:get), [16](#)
- Get ADC Object Statistics (adco:stat), [17](#)
- Get the device COM port object (ser.dev), [172](#)
- Get the device COM port object (ser.eng), [172](#)
- Get/set ADC Object power control (adco:pwr), [17](#)
- Load data from file to KVC object (kvco:_load), [130](#)
- Push values into running regression object (rro:push), [86](#)
- Push values into running stats object (rso:push), [88](#)
- Save data from KVC object to file (kvco:_save), [130](#)
- Set ADC Object settings (adco:set), [17](#)
- Set Alarm Object values (ao:set), [27](#)
- Start ADC Object conversion (adco:go), [16](#)
- objects
 - Add running regression objects (rro:add), [86](#)
 - Add running stats objects (rso:add), [87](#)
- od
- i.cell_X_od (Modem online duration), [293](#)
- off
 - Get power off time (utc.offt), [210](#)
 - Hold off failure time (c.cell_hof), [273](#)
 - Hold off success time (c.cell_hos), [273](#)
 - IoT clear hold off (iot.cho), [103](#)
 - IoT get hold off (iot.gho), [105](#)
 - IoT set modem hold off (iot.sho), [110](#)
 - lcd.off (Turn off the LCD), [237](#)
 - out.off (Turn off output), [147](#)
 - pwr.off (Turn off device), [151](#)
 - Turn off device (pwr.off), [151](#)
 - Turn off output (out.off), [147](#)
 - Turn off the LCD (lcd.off), [237](#)
- offline
 - IoT Modem Offline callback (iotOffline), [114](#)
- offt
 - utc.offt (Get power off time), [210](#)
- ok
 - key.ok (Check keyboard hardware), [231](#)
 - lcd.ok (Find out if LCD hardware present), [238](#)
 - rt.ok (Still allowed to run), [159](#)
 - utc.ok (Check if UTC is sensible), [211](#)
- okr
 - cp:okr (Check whether signals are valid), [178](#)
- on
 - Check if the LCD is on (lcd.act), [235](#)
 - lcd.on (Control LCD state), [238](#)
 - out.on (Turn on output), [147](#)
 - Seconds on battery (i.pwr_tob), [302](#)
 - Seconds on full power (i.pwr_full), [301](#)
 - Setup MODBUS on COM port (modbus.set), [136](#)
 - Show values on LCD (ui.val), [262](#)
 - Turn on output (out.on), [147](#)
 - usb.on (Check USB connected), [208](#)
 - Use password on engineer serial terminal (c.term_epw), [282](#)
- onadc
 - onAdc (ADC completion callback), [19](#)

- onalm
 - onAlm (Alarm global callback function), [25](#)
- onebtn
 - ui.set.onebtn, [266](#)
- ongps
 - onGPS (GPS location available callback), [93](#)
- onkey
 - onKey (Key global callback), [232](#)
- onkeyidle
 - onKeyId (Keyboard idle callback), [233](#)
- online
 - IoT Modem Online callback (iotOnline), [114](#)
 - Maximum time online (c.cell_mto), [274](#)
 - Modem online duration (i.cell_X_od), [293](#)
 - User mode online time (c.user_onm), [288](#)
- onm
 - c.user_onm (User mode online time), [288](#)
- onsms
 - onSMS (SMS callback function), [200](#)
- oob
 - mpco:oob (Set out-of-bound values), [82](#)
- or
 - Bitwise OR (bit.bor), [33](#)
 - Coro yield or wait (coro.w), [43](#)
 - Coro yield or wait (coro.yld), [43](#)
 - Data process URL or path (c.iot_data), [276](#)
 - Draw LCD circle or ellipse (lcd.cir), [240](#)
 - Enable or disable terminal over COM port (ser.term), [174](#)
 - Event enable or disable (evt.di), [50](#)
 - Event enable or disable (evt.en), [50](#)
 - OR indexed tables (util.tio), [224](#)
 - Set or get ADC parameters (adc.set), [15](#)
 - Set or get counter (smp.cnt), [190](#)
 - Set or get job variance (job.var), [118](#)
 - Set or get LCD idle time (lcd.idle), [236](#)
 - Update URL or path (c.iot_upd), [278](#)
 - User process URL or path (c.iot_user), [278](#)
- order
 - Calculate n-order polynomial object (float.poly), [70](#)
- original
 - Original serial number (i.rt_so), [298](#)
- os
 - os.move (Move a file), [10](#)
 - out
 - Find out if LCD hardware present (lcd.ok), [238](#)
 - out.get (Read output state), [146](#)
 - out.init (Initialise outputs), [146](#)
 - out.off (Turn off output), [147](#)
 - out.on (Turn on output), [147](#)
 - out.pwo (Control power outputs), [147](#)
 - out.set (Set outputs), [148](#)
 - Set out-of-bound values (mpco:oob), [82](#)
 - output
 - Control ADC power output (pwr.apo), [149](#)
 - Read output state (out.get), [146](#)
 - Set COM port power output (cp:pwo), [179](#)
 - Turn off output (out.off), [147](#)
 - Turn on output (out.on), [147](#)
 - outputs
 - Control power outputs (out.pwo), [147](#)
 - Initialise outputs (out.init), [146](#)
 - Set outputs (out.set), [148](#)
 - over
 - Enable or disable terminal over COM port (ser.term), [174](#)
 - overall
 - Set/Get overall GPS duration (gps.time), [92](#)
 - override
 - Override APN authentication method (c.cell_auth), [273](#)
 - Time-zone override (c.iot_tz), [277](#)
 - TOA password override (c.toa_pass), [286](#)
- P**
- pack
 - Pack structure (struct.pack), [201](#)
 - Pack table to string (modbus.ft2s), [132](#)
 - struct.pack (Pack structure), [201](#)
- pair
 - Get x
 - y pair (mpco:xy), [83](#)
- parameters
 - Set ADC default parameters (adc.def), [13](#)
 - Set or get ADC parameters (adc.set), [15](#)
- parse

- HydrINS parse text (hydrins.parse), [97](#)
- hydrins.parse (HydrINS parse text), [97](#)
- pass
 - c.cell_pass (Internet password), [274](#)
 - c.term_pass (Terminal password), [282](#)
 - c.toa_pass (TOA password override), [286](#)
- password
 - Internet password (c.cell_pass), [274](#)
 - Terminal password (c.term_pass), [282](#)
 - TOA password override (c.toa_pass), [286](#)
 - Use password on engineer serial terminal (c.term_epw), [282](#)
- past
 - Get seconds past midnight (utc.spm), [213](#)
- path
 - Data process URL or path (c.iot_data), [276](#)
 - Update URL or path (c.iot_upd), [278](#)
 - User process URL or path (c.iot_user), [278](#)
- pbm
 - Draw LCD PBM bitmap from file (lcd.pbm), [242](#)
 - lcd.pbm (Draw LCD PBM bitmap from file), [242](#)
- pcr
 - pcr (TERM), [306](#)
- pct
 - i.pwr_pct (Estimated percentage power remaining), [301](#)
- pcv
 - pcv (TERM), [306](#)
- pd
 - i.cell_X_pd (Modem power duration), [293](#)
- pdu
 - Decode SMS PDU (sms.dec), [196](#)
 - Encode SMS PDU (sms.enc), [197](#)
 - Send PDU frame (sms.snd), [199](#)
- percentage
 - Estimated percentage power remaining (i.pwr_pct), [301](#)
- pha
 - i.cell_pha (Modem activity phase), [291](#)
- phase
 - Modem activity phase (i.cell_pha), [291](#)
- pin
 - c.cell_pin (SIM PIN code), [274](#)
 - SIM PIN code (c.cell_pin), [274](#)
- pixel
 - Draw LCD pixel (lcd.p), [242](#)
- pki
 - IoT set PKI certificate (iot.cert), [103](#)
- plmn
 - Read/Clear Forbidden PLMN List (modem.fplmn), [141](#)
 - Request Clear Forbidden PLMN List (modem.clearfplmn), [140](#)
- pmm
 - pmm (TERM), [304](#)
- point
 - Create multi-point-conversion object (float.mpc), [70](#)
 - Lua app start point (i.lua_run), [297](#)
- poly
 - float.poly (Calculate n-order polynomial object), [70](#)
- polynomial
 - Calculate n-order polynomial object (float.poly), [70](#)
- polyo
 - polyo:c (Return coefficient), [85](#)
 - polyo:terms (Return terms), [85](#)
 - polyo:y (Convert x to y), [85](#)
- pop
 - jgo:pop (Pop a job from the queue), [126](#)
 - Pop a job from the queue (jgo:pop), [126](#)
 - Pop a job from the queue (jgo:poprun), [127](#)
 - Pop all UI (ui.popa), [261](#)
 - Pop UI element (ui.pop), [260](#)
 - ui.pop (Pop UI element), [260](#)
- popa
 - ui.popa (Pop all UI), [261](#)
- poprun
 - jgo:poprun (Pop a job from the queue), [127](#)
- port
 - Enable or disable terminal over COM port (ser.term), [174](#)
 - Get the device COM port object (ser.dev), [172](#)
 - Get the device COM port object (ser.eng),

- 172
- Lock the COM port (cp:lck), 177
- Set COM port power output (cp:pwo), 179
- Setup COM port (cp:set), 184
- Setup COM port for SDI-12 (sdi12.set), 170
- Setup MODBUS on COM port (modbus.set), 136
- Unlock the COM port (cp:ulck), 186
- positions
 - Set UI font positions (ui.set.ui), 267
- power
 - Control ADC power output (pwr.apo), 149
 - Control power outputs (out.pwo), 147
 - Enable power for receiving (cp:pwr), 180
 - Enable power for receiving (cp:rpxw), 180
 - Enable power for receiving and transmitting (cp:rxtx), 183
 - Enable power for transmitting (cp:pwt), 180
 - Enable power for transmitting (cp:txpw), 180
 - Estimated percentage power remaining (i.pwr_pct), 301
 - Get power manager metrics (pwr.gpm), 151
 - Get power off time (utc.offt), 210
 - Get/set ADC Object power control (adco:pwr), 17
 - Modem power duration (i.cell_pwr), 291
 - Modem power duration (i.cell_X_pd), 293
 - Modem power up time (i.cell_X_pu), 294
 - Query the power reasons (pwr.rsn), 151
 - Seconds on full power (i.pwr_full), 301
 - Set COM port power output (cp:pwo), 179
- present
 - Find out if LCD hardware present (lcd.ok), 238
- print
 - print (Print string), 155
 - Print string (print), 155
 - Print string (rt.print), 155
 - rt.print (Print string), 155
- process
 - Data process URL or path (c.iot_data), 276
 - IoT abort user process (iot.aup), 102
 - User process URL or path (c.iot_user), 278
- prot
 - cp:prot (Get the protocol settings), 179
- protected
 - Protected call (rt.call), 157
- protocol
 - Get the protocol settings (cp:prot), 179
- pt
 - pt (TERM), 306
- pu
 - i.cell_X_pu (Modem power up time), 294
- pulse
 - Digital pulse current level (dpc.lvl), 48
 - Digital pulse disable (dpc.dis), 48
 - Digital pulse get value (dpc.val), 49
 - Digital pulse reset (dpc.rst), 49
- push
 - fao:push (Push value), 77
 - Push UI element (ui.push), 261
 - Push value (fao:push), 77
 - Push values into running regression object (rro:push), 86
 - Push values into running stats object (rso:push), 88
 - Push variance (c.iot_var), 279
 - rro:push (Push values into running regression object), 86
 - rso:push (Push values into running stats object), 88
 - ui.push (Push UI element), 261
- pwo
 - cp:pwo (Set COM port power output), 179
 - out.pwo (Control power outputs), 147
- pwr
 - adco:pwr (Get/set ADC Object power control), 17
 - i.cell_pwr (Modem power duration), 291
 - i.pwr_alv (Seconds alive), 301
 - i.pwr_ccb (Battery capacity), 301
 - i.pwr_ccu (Estimated charge used), 301
 - i.pwr_full (Seconds on full power), 301
 - i.pwr_pct (Estimated percentage power remaining), 301

- i.pwr_slp (Seconds asleep), [302](#)
- i.pwr_tob (Seconds on battery), [302](#)
- pwr._bv (Set the boost values), [154](#)
- pwr._iv (Set the idle timers), [154](#)
- pwr.apo (Control ADC power output), [149](#)
- pwr.back (Set backup battery), [149](#)
- pwr.boot (Reboot device), [150](#)
- pwr.dpc (Set the coulomb consumption for device), [150](#)
- pwr.epc (Set the coulomb consumption for engineer), [150](#)
- pwr.gpm (Get power manager metrics), [151](#)
- pwr.off (Turn off device), [151](#)
- pwr.rsn (Query the power reasons), [151](#)
- pwr.sbc (Set battery coulomb capacity), [152](#)
- pwr.scu (Set coulombs used), [152](#)
- pwr.tmv (Query the transient minimum voltage), [153](#)
- pwr.work (Allow hard work), [153](#)
- pwr_x
 - cp:pwr_x (Enable power for receiving), [180](#)
- pwt_x
 - cp:pwt_x (Enable power for transmitting), [180](#)
- Q**
- quality
 - Modem signal quality indicator (i.cell_X_csq), [292](#)
- query
 - Query ID (sdi12.id), [169](#)
 - Query the power reasons (pwr.rsn), [151](#)
 - Query the transient minimum voltage (pwr.tmv), [153](#)
 - Query TWI bus (twi.qy), [205](#)
- queue
 - Add callback to job queue (jqo:add), [125](#)
 - Check if callback in job queue (jqo:is), [126](#)
 - Coro add function to queue (coro.add), [41](#)
 - Create a new Job Queue Object (jobq.new), [123](#)
 - Delete jobs from the queue (jqo:del), [125](#)
 - Pop a job from the queue (jqo:pop), [126](#)
 - Pop a job from the queue (jqo:poprun), [127](#)
 - Run all jobs in the queue (jqo:run), [127](#)
- qy
 - twi.qy (Query TWI bus), [205](#)
- R**
- range
 - Get range (mpco:range), [82](#)
 - mpco:range (Get range), [82](#)
- rd
 - float.rd (Read floats/doubles from file), [71](#)
 - twi.rd (Read TWI), [205](#)
- rdi
 - modbus.rdi (Read discrete inputs), [133](#)
- read
 - Config read as table (cfg.table), [39](#)
 - Read discrete inputs (modbus.rdi), [133](#)
 - Read floats/doubles from file (float.rd), [71](#)
 - Read floats/doubles from file (float.rf), [71](#)
 - Read input registers (modbus.rir), [134](#)
 - Read MODBUS holding registers (modbus.rhr), [133](#)
 - Read multiple coils (modbus.rmc), [134](#)
 - Read output state (out.get), [146](#)
 - Read server ID (modbus.rsid), [135](#)
 - Read SIM FileID Binary (modem.simrb), [142](#)
 - Read SIM FileID STATUS (modem.sims), [142](#)
 - Read TWI (twi.rd), [205](#)
 - Read/Clear Forbidden PLMN List (modem.fplmn), [141](#)
- reading
 - Get ADC reading (rt.adc), [155](#)
- readings
 - Get millivolt readings (rt.mv), [163](#)
- real
 - Get/Set Real Time Clock (utc.rtc), [211](#)
- reason
 - Get IoT reason string (iot.flg), [105](#)
- reasons
 - Query the power reasons (pwr.rsn), [151](#)
- reboot
 - reboot (TERM), [308](#), [317](#)
 - Reboot device (pwr.boot), [150](#)

- Reboot device (rt.coldboot), [158](#)
- Reboot Lua (rt.luaboot), [161](#)
- receive
 - HydrINS receive (hydrins.rx), [98](#)
 - Receive and empty data (cp:rx), [182](#)
 - Receive data (cp:rx), [182](#)
 - Receive MODBUS RTU frame (modbus.rx), [135](#)
 - Receive until (cp:rxu), [183](#)
 - Set receive flow control (cp:rxf), [182](#)
 - Set receive timeout value (cp:rxt), [183](#)
 - Set receive timer (sms.rxt), [198](#)
 - Set the receive FIFO (cp:fifo), [177](#)
- receiving
 - Enable power for receiving (cp:pwr), [180](#)
 - Enable power for receiving (cp:rxpw), [180](#)
 - Enable power for receiving and transmitting (cp:rxt), [183](#)
- reg
 - c.cell_reg (Cell registration time limit), [274](#)
- register
 - Write single register (modbus.wsr), [138](#)
- registers
 - Read input registers (modbus.rir), [134](#)
 - Read MODBUS holding registers (modbus.rhr), [133](#)
 - Write MODBUS multiple registers (modbus.wmr), [137](#)
- registration
 - Cell registration time limit (c.cell_reg), [274](#)
 - Modem registration seconds (i.cell_X_rs), [294](#)
- regr
 - float.regr (Create a running regression object), [71](#)
- regression
 - Add running regression objects (rro:add), [86](#)
 - Clear running regression object (rro:clear), [86](#)
 - Create a running regression object (float.regr), [71](#)
 - Push values into running regression object (rro:push), [86](#)
- remaining
 - Estimated percentage power remaining (i.pwr_pct), [301](#)
- remi
 - util.remi (Remove indexed table), [222](#)
- remove
 - File remove directory (fc.rmd), [64](#)
 - Remove indexed table (util.remi), [222](#)
- rename
 - Use Temp-File-and-Rename for FTP (c.iot_tfr), [277](#)
- repeat
 - Set key repeat enable (key.kre), [230](#)
 - Set keyboard repeat idle (key.kri), [231](#)
 - Set UI key repeat (ui.set.kr), [265](#)
- request
 - Request Clear Forbidden PLMN List (modem.clearfplmn), [140](#)
- requests
 - IoT abort modem requests (iot.abort), [101](#)
- reset
 - Digital pulse reset (dpc.rst), [49](#)
 - Reset the GPS counter (gps.rst), [91](#)
- retry
 - TOA retry time (c.toa_try), [287](#)
- return
 - Return coefficient (polyo:c), [85](#)
 - Return current UTC (utc.now), [210](#)
 - Return MODBUS timings (modbus.time), [136](#)
 - Return terms (polyo:terms), [85](#)
- rf
 - float.rf (Read floats/doubles from file), [71](#)
- rhr
 - modbus.rhr (Read MODBUS holding registers), [133](#)
- right
 - Bit arithmetic shift right (bit.al), [32](#)
 - Bit arithmetic shift right (bit.alshift), [32](#)
 - Bit arithmetic shift right (bit.ar), [32](#)
 - Bit arithmetic shift right (bit.arshift), [32](#)
 - Bitwise right shift (bit.rs), [35](#)
 - Bitwise right shift (bit.rshift), [35](#)
- rir

- modbus.rir (Read input registers), [134](#)
- rmc
 - modbus.rmc (Read multiple coils), [134](#)
- rmd
 - fc.rmd (File remove directory), [64](#)
- root
 - Square root entries (fao:sqrt), [79](#)
- rotation
 - Set LCD font rotation (lcd.fr), [244](#)
- rounded
 - Set UI rounded corners (ui.set.r), [266](#)
- routine
 - Coro kill routine (coro.kill), [42](#)
- routines
 - Coro number of routines running (coro.n), [42](#)
- rro
 - rro:add (Add running regression objects), [86](#)
 - rro:clear (Clear running regression object), [86](#)
 - rro:push (Push values into running regression object), [86](#)
- rs
 - bit.rs (Bitwise right shift), [35](#)
 - cp:rs (Set the RS232/RS485 mode), [181](#)
 - i.cell_X_rs (Modem registration seconds), [294](#)
- rs232
 - Set the RS232/RS485 mode (cp:rs), [181](#)
- rs485
 - Set RS485 bias control (ser.bias), [172](#)
 - Set the RS232/RS485 mode (cp:rs), [181](#)
- rshift
 - bit.rshift (Bitwise right shift), [35](#)
- rsid
 - modbus.rsid (Read server ID), [135](#)
- rsn
 - pwr.rsn (Query the power reasons), [151](#)
- rso
 - rso:add (Add running stats objects), [87](#)
 - rso:clear (Clear running stats object), [87](#)
 - rso:get (Get running statistics values), [87](#)
 - rso:push (Push values into running stats object), [88](#)
- rst
 - dpc.rst (Digital pulse reset), [49](#)
 - gps.rst (Reset the GPS counter), [91](#)
- rt
 - i.rt_cbf (Configuration bytes free), [298](#)
 - i.rt_sn (Serial number), [298](#)
 - i.rt_so (Original serial number), [298](#)
 - i.rt_tag (Asset tag), [298](#)
 - i.rt_utc (Current time), [299](#)
 - rt.adc (Get ADC reading), [155](#)
 - rt.atom (Atomic function call), [156](#)
 - rt.bin (Convert ASCII hex to binary), [156](#)
 - rt.bool (Convert boolean value), [157](#)
 - rt.call (Protected call), [157](#)
 - rt.coldboot (Reboot device), [158](#)
 - rt.cyc (Get Cortex cycle counter), [158](#)
 - rt.dct (Deep copy table), [158](#)
 - rt.dev (Get device details), [159](#)
 - rt.dorun (Still allowed to run), [159](#)
 - rt.exp (Expand a string), [160](#)
 - rt.hex (Convert binary to ASCII hex), [160](#)
 - rt.led (Enable debug LED sequence), [161](#)
 - rt.log (Write log entry), [161](#)
 - rt.luaboot (Reboot Lua), [161](#)
 - rt.mag (Set magnet delay), [162](#)
 - rt.ms (Delay milliseconds), [162](#)
 - rt.msf (Calculate millisecond finish time), [163](#)
 - rt.msx (Check millisecond expiry), [163](#)
 - rt.mv (Get millivolt readings), [163](#)
 - rt.mvh (Get highest millivolt), [164](#)
 - rt.ok (Still allowed to run), [159](#)
 - rt.print (Print string), [155](#)
 - rt.setblf (Set a BLF file), [164](#)
 - rt.temp (Get temperature), [164](#)
 - rt.us (Delay microseconds), [165](#)
- rtc
 - utc.rtc (Get/Set Real Time Clock), [211](#)
- rts
 - cp:rts (Set RTS line), [181](#)
 - Set RTS line (cp:rts), [181](#)
- rtu
 - Receive MODBUS RTU frame (modbus.rx),

- screen
 - Get the LCD screen size (lcd.gss), [236](#)
 - Get the LCD screen size (lcd.size), [236](#)
- scu
 - pwr.scu (Set coulombs used), [152](#)
- sdc
 - lcd.sdc (Set LCD display contrast), [239](#)
- sdf
 - fc.sdf (File split dir and file), [64](#)
- sdi
 - Run SDI-12 sequence (sdi12.go), [169](#)
 - Setup COM port for SDI-12 (sdi12.set), [170](#)
- sdi12
 - sdi12.ack (Check ACK), [167](#)
 - sdi12.brk (Send break), [167](#)
 - sdi12.chg (Change device address), [167](#)
 - sdi12.crc (CRC calculation), [168](#)
 - sdi12.done (Check whether sequence complete), [168](#)
 - sdi12.go (Run SDI-12 sequence), [169](#)
 - sdi12.id (Query ID), [169](#)
 - sdi12.mrk (Send mark), [169](#)
 - sdi12.set (Setup COM port for SDI-12), [170](#)
 - sdi12.t (Transparent command), [170](#)
- seconds
 - Get seconds past midnight (utc.spm), [213](#)
 - Modem registration seconds (i.cell_X_rs), [294](#)
 - Seconds alive (i.pwr_alv), [301](#)
 - Seconds asleep (i.pwr_slp), [302](#)
 - Seconds on battery (i.pwr_tob), [302](#)
 - Seconds on full power (i.pwr_full), [301](#)
- secret
 - Check a secret key (cfg.key.chk), [40](#)
 - Erase a secret key (cfg.key.wipe), [40](#)
 - Set a secret key (cfg.key.set), [40](#)
 - TOA secret key (c.toa_key), [286](#)
- send
 - iot.send (Send files to IoT server), [108](#)
 - Send break (cp:brk), [175](#)
 - Send break (sdi12.brk), [167](#)
 - Send command to HydrINS (hydrins.cmd), [97](#)
 - Send diagnostics info to IoT server (iot.diag), [104](#)
 - Send files to IoT server (iot.send), [108](#)
 - Send log string to USB (usb.log), [208](#)
 - Send mark (sdi12.mrk), [169](#)
 - Send MODBUS command (modbus.cmd), [131](#)
 - Send modem command (iot.modem), [107](#)
 - Send modem command (modem.cmd), [140](#)
 - Send PDU frame (sms.snd), [199](#)
 - Send text (sms.txt), [199](#)
- sensible
 - Check if UTC is sensible (utc.ok), [211](#)
- sequence
 - Check whether sequence complete (sdi12.done), [168](#)
 - Enable debug LED sequence (rt.led), [161](#)
 - Handle a command sequence (cp:cmd), [176](#)
 - Modem init sequence (c.cell_init), [274](#)
 - Run SDI-12 sequence (sdi12.go), [169](#)
- ser
 - ser.bias (Set RS485 bias control), [172](#)
 - ser.dev (Get the device COM port object), [172](#)
 - ser.eng (Get the device COM port object), [172](#)
 - ser.exp (Check expansion board), [173](#)
 - ser.term (Enable or disable terminal over COM port), [174](#)
- serial
 - Original serial number (i.rt_so), [298](#)
 - Serial number (i.rt_sn), [298](#)
 - Use password on engineer serial terminal (c.term_epw), [282](#)
- server
 - Modem server connected (i.cell_X_sc), [294](#)
 - NTP server (c.iot_ntp), [276](#)
 - Read server ID (modbus.rsid), [135](#)
 - Send diagnostics info to IoT server (iot.diag), [104](#)
 - Send files to IoT server (iot.send), [108](#)
 - Set IoT server URL (iot.set), [109](#)
 - TLS Server-Name-Indicator (c.tls_sni), [284](#)
- set
 - adc.set (Set or get ADC parameters), [15](#)

- adco:set (Set ADC Object settings), [17](#)
- Alarm set (alm.set), [21](#)
- alm.set (Alarm set), [21](#)
- ao:set (Set Alarm Object values), [27](#)
- cfg.key.set (Set a secret key), [40](#)
- cfg.set (Config set), [38](#)
- Config set (cfg.set), [38](#)
- cp:set (Setup COM port), [184](#)
- fao:set (Update an array entry), [78](#)
- File set cache (fc.cache), [57](#)
- File set file write mode (fc.sfw), [64](#)
- Get/set ADC Object power control (adco:pwr), [17](#)
- Get/Set Real Time Clock (utc.rtc), [211](#)
- IoT set modem hold off (iot.sho), [110](#)
- IoT set PKI certificate (iot.cert), [103](#)
- iot.set (Set IoT server URL), [109](#)
- job.set (Set job schedule), [117](#)
- modbus.set (Setup MODBUS on COM port), [136](#)
- out.set (Set outputs), [148](#)
- sdi12.set (Setup COM port for SDI-12), [170](#)
- Set a BLF file (rt.setblf), [164](#)
- Set a secret key (cfg.key.set), [40](#)
- Set a UTC based job (job.utc), [120](#)
- Set ADC default parameters (adc.def), [13](#)
- Set ADC Object settings (adco:set), [17](#)
- Set Alarm Object values (ao:set), [27](#)
- Set backup battery (pwr.back), [149](#)
- Set battery coulomb capacity (pwr.sbc), [152](#)
- Set COM port power output (cp:pwo), [179](#)
- Set coulombs used (pwr.scu), [152](#)
- Set IoT server URL (iot.set), [109](#)
- Set job schedule (job.set), [117](#)
- SET key icon (ui.ssk), [261](#)
- Set key inversion (key.inv), [230](#)
- Set key repeat enable (key.kre), [230](#)
- Set keyboard repeat idle (key.kri), [231](#)
- Set LCD display contrast (lcd.sdc), [239](#)
- Set LCD draw colour (lcd.c), [235](#)
- Set LCD font rotation (lcd.fr), [244](#)
- Set LCD font transparency (lcd.ft), [244](#)
- Set magnet delay (rt.mag), [162](#)
- Set modem anti-cell-zombie timeout (iot.azt), [102](#)
- Set modem stay-connected millivolt (iot.scmv), [107](#)
- Set or get ADC parameters (adc.set), [15](#)
- Set or get counter (smp.cnt), [190](#)
- Set or get job variance (job.var), [118](#)
- Set or get LCD idle time (lcd.idle), [236](#)
- Set out-of-bound values (mpco.oob), [82](#)
- Set outputs (out.set), [148](#)
- Set receive flow control (cp:rx), [182](#)
- Set receive timeout value (cp:rxto), [183](#)
- Set receive timer (sms.rxt), [198](#)
- Set RS485 bias control (ser.bias), [172](#)
- Set RTS line (cp:rts), [181](#)
- Set the boost values (pwr._bv), [154](#)
- Set the coulomb consumption for device (pwr.dpc), [150](#)
- Set the coulomb consumption for engineer (pwr.epc), [150](#)
- Set the idle timers (pwr._iv), [154](#)
- Set the LCD font (lcd.f), [244](#)
- Set the receive FIFO (cp:fifo), [177](#)
- Set the RS232/RS485 mode (cp:rs), [181](#)
- Set transmit flow control (cp:txf), [185](#)
- Set transmit time guard gap (cp:txtg), [186](#)
- Set UI colours (ui.set.c), [263](#)
- Set UI filename (ui.set.i8n), [264](#)
- Set UI font positions (ui.set.ui), [267](#)
- Set UI idle time (ui.set.idle), [264](#)
- Set UI key mapping (ui.set.km), [265](#)
- Set UI key repeat (ui.set.kr), [265](#)
- Set UI rounded corners (ui.set.r), [266](#)
- Set UTC time and timezone (utc.set), [212](#)
- Set/Get GPS fix distance (gps.dist), [90](#)
- Set/Get GPS fix time (gps.fixt), [90](#)
- Set/Get GPS interval (gps.gap), [90](#)
- Set/Get overall GPS duration (gps.time), [92](#)
- ui.set.bklt (Control LCD backlight), [263](#)
- ui.set.c (Set UI colours), [263](#)
- ui.set.i8n (Set UI filename), [264](#)
- ui.set.idle (Set UI idle time), [264](#)
- ui.set.km (Set UI key mapping), [265](#)
- ui.set.kr (Set UI key repeat), [265](#)
- ui.set.onebtn, [266](#)

- ui.set.r (Set UI rounded corners), [266](#)
- ui.set.showsk, [267](#)
- ui.set.ui (Set UI font positions), [267](#)
- utc.set (Set UTC time and timezone), [212](#)
- setblf
 - rt.setblf (Set a BLF file), [164](#)
- settings
 - Get ADC Object settings (adco:get), [16](#)
 - Get the protocol settings (cp:prot), [179](#)
 - Set ADC Object settings (adco:set), [17](#)
- setup
 - Setup COM port (cp:set), [184](#)
 - Setup COM port for SDI-12 (sdi12.set), [170](#)
 - Setup MODBUS on COM port (modbus.set), [136](#)
- sfwm
 - fc.sfwm (File set file write mode), [64](#)
- sha1
 - hash.sha1 (Hash string immediately), [94](#)
- sha256
 - c.sha256, [336](#)
 - hash.sha256 (Hash string immediately), [94](#)
- shift
 - Bit arithmetic shift right (bit.al), [32](#)
 - Bit arithmetic shift right (bit.alshift), [32](#)
 - Bit arithmetic shift right (bit.ar), [32](#)
 - Bit arithmetic shift right (bit.arshift), [32](#)
 - Bitwise left shift (bit.ls), [34](#)
 - Bitwise left shift (bit.lshift), [34](#)
 - Bitwise right shift (bit.rs), [35](#)
 - Bitwise right shift (bit.rshift), [35](#)
- sho
 - iot.sho (IoT set modem hold off), [110](#)
- show
 - Show values on LCD (ui.val), [262](#)
 - Show YES tick (ui.yes), [262](#)
- shows
 - ui.set.showsk, [267](#)
- sig
 - evt.sig (Event signal trigger), [52](#)
- signal
 - Event signal (evt.lsig), [51](#)
 - Event signal trigger (evt.sig), [52](#)
 - Modem signal quality indicator (i.cell_X_csq), [292](#)
- signals
 - Check whether signals are valid (cp:okr), [178](#)
- sim
 - Convert SIM STATUS to table (modem.stot), [143](#)
 - Read SIM FileID Binary (modem.simrb), [142](#)
 - Read SIM FileID STATUS (modem.sims), [142](#)
 - SIM PIN code (c.cell_pin), [274](#)
 - Write SIM FileID Binary (modem.simwb), [143](#)
- simrb
 - modem.simrb (Read SIM FileID Binary), [142](#)
- sims
 - modem.sims (Read SIM FileID STATUS), [142](#)
- simwb
 - modem.simwb (Write SIM FileID Binary), [143](#)
- sin
 - fao:sin (Fill with sin values), [78](#)
 - Fill with sin values (fao:sin), [78](#)
- single
 - Write single register (modbus.wsr), [138](#)
- site
 - c.site_loc (Site location string), [281](#)
 - c.site_name (Site name), [281](#)
 - Site location string (c.site_loc), [281](#)
 - Site name (c.site_name), [281](#)
- size
 - Calculate size of format (struct.size), [201](#)
 - fao:size (Get array size), [79](#)
 - Get array size (fao:size), [79](#)
 - Get the LCD screen size (lcd.gss), [236](#)
 - Get the LCD screen size (lcd.size), [236](#)
 - lcd.size (Get the LCD screen size), [236](#)
 - struct.size (Calculate size of format), [201](#)
- slp
 - i.pwr_slp (Seconds asleep), [302](#)
- smp
 - i.smp.cnt (Number of samples), [300](#)
 - i.smp.txt (Last sample text), [300](#)

- i.smp.utc (Last sample time), [300](#)
- smp.cnt (Set or get counter), [190](#)
- smp.csv (Convert table to CSV text), [190](#)
- smp.cut (Cut the sample file), [191](#)
- smp.save (Save sample to flash), [191](#)
- sms
 - c.cell_sms (Default SMS number), [275](#)
 - Check free SMS entries (sms.free), [197](#)
 - Decode SMS PDU (sms.dec), [196](#)
 - Default SMS number (c.cell_sms), [275](#)
 - Encode SMS PDU (sms.enc), [197](#)
 - SMS callback function (onSMS), [200](#)
 - sms.ag7 (Convert ASCII to GSM-7bit), [196](#)
 - sms.dec (Decode SMS PDU), [196](#)
 - sms.enc (Encode SMS PDU), [197](#)
 - sms.free (Check free SMS entries), [197](#)
 - sms.g7a (Convert GSM-7bit to ASCII), [198](#)
 - sms.go (Trigger SMS exchange), [198](#)
 - sms.rxt (Set receive timer), [198](#)
 - sms.snd (Send PDU frame), [199](#)
 - sms.txt (Send text), [199](#)
 - Trigger SMS exchange (sms.go), [198](#)
- sn
 - i.rt_sn (Serial number), [298](#)
- snd
 - sms.snd (Send PDU frame), [199](#)
- sni
 - c.tls_sni (TLS Server-Name-Indicator), [284](#)
- so
 - i.rt_so (Original serial number), [298](#)
- soft
 - Draw soft key (ui.key), [260](#)
- split
 - File split dir and file (fc.sdf), [64](#)
 - File split extension (fc.ext), [60](#)
 - Split UTC time (utc.split), [212](#)
 - utc.split (Split UTC time), [212](#)
- spm
 - utc.spm (Get seconds past midnight), [213](#)
- sqrt
 - fao:sqrt (Square root entries), [79](#)
- square
 - Square root entries (fao:sqrt), [79](#)
- ssk
 - ui.ssk (SET key icon), [261](#)
- sta
 - i.cell_sta (Modem state), [292](#)
- start
 - Lua app start point (i.lua_run), [297](#)
 - Start ADC (adc.go), [14](#)
 - Start ADC Object conversion (adco:go), [16](#)
 - Start GPS (gps.go), [91](#)
 - Start the UI (ui.go), [260](#)
- starting
 - IoT Modem Starting callback (iotSetup), [115](#)
- stat
 - adco:stat (Get ADC Object Statistics), [17](#)
 - fao:stat (Calculate stats), [79](#)
 - float.stat (Create a running statistics object), [72](#)
- state
 - Alarm Object state callback (function), [28](#)
 - Control LCD state (lcd.on), [238](#)
 - Get key state (key.get), [229](#)
 - Modem state (i.cell_sta), [292](#)
 - Read output state (out.get), [146](#)
- statistics
 - Create a running statistics object (float.stat), [72](#)
 - Get ADC Object Statistics (adco:stat), [17](#)
 - Get running statistics values (float.mean), [69](#)
 - Get running statistics values (rso:get), [87](#)
- stats
 - Add running stats objects (rso:add), [87](#)
 - Calculate stats (fao:stat), [79](#)
 - Clear running stats object (rso:clear), [87](#)
 - Push values into running stats object (rso:push), [88](#)
- status
 - Convert SIM STATUS to table (modem.stot), [143](#)
 - Read SIM FileID STATUS (modem.sims), [142](#)
- stay
 - Set modem stay-connected millivolt (iot.scmv), [107](#)
- still

- Still allowed to run (rt.dorun), [159](#)
- Still allowed to run (rt.ok), [159](#)
- stop
 - IoT is there a stop file (iot.isf), [106](#)
- stot
 - modem.stot (Convert SIM STATUS to table), [143](#)
- string
 - Config get string (cfg.gets), [38](#)
 - Convert hash string to value (hash.val), [95](#)
 - Convert string to UTC (utc.val), [214](#)
 - Convert table to NMEA string (nmea.txt), [145](#)
 - Draw LCD XBM bitmap string (lcd.xbm), [243](#)
 - Expand a string (rt.exp), [160](#)
 - Get IoT reason string (iot.flg), [105](#)
 - Hash string immediately (hash.hmac256), [94](#)
 - Hash string immediately (hash.md5), [94](#)
 - Hash string immediately (hash.sha1), [94](#)
 - Hash string immediately (hash.sha256), [94](#)
 - Pack table to string (modbus.ft2s), [132](#)
 - Print string (print), [155](#)
 - Print string (rt.print), [155](#)
 - Send log string to USB (usb.log), [208](#)
 - Site location string (c.site_loc), [281](#)
 - Unpack string to table (modbus.fs2t), [132](#)
- struct
 - struct.pack (Pack structure), [201](#)
 - struct.size (Calculate size of format), [201](#)
 - struct.unpack (Unpack structure), [202](#)
- structure
 - Pack structure (struct.pack), [201](#)
 - Unpack structure (struct.unpack), [202](#)
- sub
 - add/sub/mul/div values (fao:add), [75](#)
 - add/sub/mul/div values (fao:div), [75](#)
 - add/sub/mul/div values (fao:mul), [75](#)
 - add/sub/mul/div values (fao:sub), [75](#)
 - fao:sub (add/sub/mul/div values), [75](#)
- success
 - Hold off success time (c.cell_hos), [273](#)
- suite
 - TLS cipher suite (c.tls_csc), [283](#)
- sum
 - fao:sum (Sum all values), [80](#)
 - Multiply-and-sum two arrays (fao:xs), [81](#)
 - Sum all values (fao:sum), [80](#)
- supported
 - Journalling supported (fc.js), [61](#)
- swap
 - Bit swap (bit.swap32), [35](#)
 - Bit swap bytes (bit.swap16), [35](#)
 - Bit swap nibbles (bit.swap8), [36](#)
- swap16
 - bit.swap16 (Bit swap bytes), [35](#)
- swap32
 - bit.swap32 (Bit swap), [35](#)
- swap8
 - bit.swap8 (Bit swap nibbles), [36](#)
- sync
 - fc.sync (File sync to flash), [65](#)
 - File sync to flash (fc.sync), [65](#)
- T
- tab
 - fao:tab (Convert to table), [80](#)
 - nmea.tab (Convert NMEA line to table), [144](#)
- table
 - Append index table (util.tia), [223](#)
 - cfg.table (Config read as table), [39](#)
 - Config read as table (cfg.table), [39](#)
 - Convert CSV text to table (util.csvt), [219](#)
 - Convert NMEA line to table (nmea.tab), [144](#)
 - Convert SIM STATUS to table (modem.stot), [143](#)
 - Convert table to CSV text (smp.csv), [190](#)
 - Convert table to CSV text (util.tcsv), [223](#)
 - Convert table to NMEA string (nmea.txt), [145](#)
 - Convert to table (fao:tab), [80](#)
 - Create bits table (util.bits), [217](#)
 - Create table from table with bits (util.btab), [217](#), [217](#)
 - Cut indexed table (util.cuti), [219](#)
 - Deep copy table (rt.dct), [158](#)
 - Deep copy table (util.dct), [220](#)

- Iterate table (util.iter), [221](#)
- Pack table to string (modbus.ft2s), [132](#)
- Remove indexed table (util.remi), [222](#)
- Unpack string to table (modbus.fs2t), [132](#)
- tables
 - Add indexed tables (util.addi), [215](#)
 - Add tables (util.add), [215](#)
 - Compare indexed tables (util.cmpi), [218](#)
 - Get x and y tables (mpco:xyt), [83](#)
 - OR indexed tables (util.tio), [224](#)
- tag
 - Asset tag (i.rt_tag), [298](#)
 - i.rt_tag (Asset tag), [298](#)
- tcsv
 - util.tcsv (Convert table to CSV text), [223](#)
- tell
 - Tell HydrINS to run (hydrins.run), [98](#)
- temp
 - rt.temp (Get temperature), [164](#)
 - temp (TERM), [308](#)
 - Use Temp-File-and-Rename for FTP (c.iot_tfr), [277](#)
- temperature
 - Get temperature (rt.temp), [164](#)
- term
 - c.term_epw (Use password on engineer serial terminal), [282](#)
 - c.term_pass (Terminal password), [282](#)
 - ser.term (Enable or disable terminal over COM port), [174](#)
- terminal
 - Enable or disable terminal over COM port (ser.term), [174](#)
 - Terminal password (c.term_pass), [282](#)
 - Use password on engineer serial terminal (c.term_epw), [282](#)
- terms
 - polyo:terms (Return terms), [85](#)
 - Return terms (polyo:terms), [85](#)
- test
 - IoT test callback (iotTest), [115](#)
- text
 - Alarm get text (alm.txt), [22](#)
 - Alarm text names (c.alm_txt01), [271](#)
 - Alarm text names (c.alm_txt02), [271](#)
 - App fetch text block (app.txt), [31](#)
 - Convert CSV text to table (util.csvt), [219](#)
 - Convert hash value to text (hash.txt), [95](#)
 - Convert table to CSV text (smp.csv), [190](#)
 - Convert table to CSV text (util.tcsv), [223](#)
 - Draw LCD text (lcd.t), [246](#)
 - Get LCD text width (lcd.gtw), [245](#)
 - Get UI element text (ui.txt), [262](#)
 - HydrINS parse text (hydrins.parse), [97](#)
 - Last sample text (i.smp.txt), [300](#)
 - Send text (sms.txt), [199](#)
 - Text of UTC (utc.txt), [213](#)
- tfr
 - c.iot_tfr (Use Temp-File-and-Rename for FTP), [277](#)
- tia
 - util.tia (Append index table), [223](#)
- tick
 - Show YES tick (ui.yes), [262](#)
- time
 - Calculate millisecond finish time (rt.msf), [163](#)
 - Cell registration time limit (c.cell_reg), [274](#)
 - Convert alive time to UTC (utc.a2utc), [209](#)
 - Convert UTC to alive time (utc.utc2a), [214](#)
 - Current time (i.rt_utc), [299](#)
 - Get power off time (utc.offt), [210](#)
 - Get time zone (utc.tz), [213](#)
 - Get/Set Real Time Clock (utc.rtc), [211](#)
 - gps.time (Set/Get overall GPS duration), [92](#)
 - Hold off failure time (c.cell_hof), [273](#)
 - Hold off success time (c.cell_hos), [273](#)
 - Last sample time (i.smp.utc), [300](#)
 - Maximum time online (c.cell_mto), [274](#)
 - modbus.time (Return MODBUS timings), [136](#)
 - Modem power up time (i.cell_X_pu), [294](#)
 - Set or get LCD idle time (lcd.idle), [236](#)
 - Set transmit time guard gap (cp:txtg), [186](#)
 - Set UI idle time (ui.set.idle), [264](#)
 - Set UTC time and timezone (utc.set), [212](#)
 - Set/Get GPS fix time (gps.fixt), [90](#)
 - Split UTC time (utc.split), [212](#)

- time (TERM), [308](#)
- Time-zone override (c.iot_tz), [277](#)
- TLS ignore local time (c.tls_ilt), [284](#)
- TOA retry time (c.toa_try), [287](#)
- User mode online time (c.user_onm), [288](#)
- Variance time applied (i.iot_var), [296](#)
- timeout
 - Set modem anti-cell-zombie timeout (iot.azt), [102](#)
 - Set receive timeout value (cp:rxto), [183](#)
 - TOA idle timeout (c.toa_ito), [286](#)
- timer
 - Set receive timer (sms.rxt), [198](#)
- timers
 - Set the idle timers (pwr._iv), [154](#)
- timezone
 - Set UTC time and timezone (utc.set), [212](#)
- timings
 - Return MODBUS timings (modbus.time), [136](#)
- tio
 - util.tio (OR indexed tables), [224](#)
- tls
 - c.tls_csc (TLS cipher suite), [283](#)
 - c.tls_ilt (TLS ignore local time), [284](#)
 - c.tls_sni (TLS Server-Name-Indicator), [284](#)
 - c.tls_ver (TLS version), [285](#)
 - TLS cipher suite (c.tls_csc), [283](#)
 - TLS ignore local time (c.tls_ilt), [284](#)
 - TLS Server-Name-Indicator (c.tls_sni), [284](#)
 - TLS version (c.tls_ver), [285](#)
- tmv
 - pwr.tmv (Query the transient minimum voltage), [153](#)
- toa
 - c.toa_ito (TOA idle timeout), [286](#)
 - c.toa_key (TOA secret key), [286](#)
 - c.toa_pass (TOA password override), [286](#)
 - c.toa_try (TOA retry time), [287](#)
 - c.toa_url (TOA URL), [287](#)
 - toa (TERM), [313](#)
 - TOA idle timeout (c.toa_ito), [286](#)
 - TOA password override (c.toa_pass), [286](#)
 - TOA retry time (c.toa_try), [287](#)
 - TOA secret key (c.toa_key), [286](#)
 - TOA URL (c.toa_url), [287](#)
- tob
 - i.pwr_tob (Seconds on battery), [302](#)
- top
 - Get the top UI element (ui.top), [261](#)
 - ui.top (Get the top UI element), [261](#)
- transform
 - Fourier Transform (FFT/DFT) (fao:ft), [76](#)
 - Wavelet Transform (DWT/CWT) (fao:wt), [80](#)
- transient
 - Query the transient minimum voltage (pwr.tmv), [153](#)
- transmit
 - Set transmit flow control (cp:txf), [185](#)
 - Set transmit time guard gap (cp:txtg), [186](#)
 - Transmit data (cp:tx), [185](#)
 - Transmit MODBUS RTU frame (modbus.tx), [137](#)
- transmitting
 - Enable power for receiving and transmitting (cp:rxtx), [183](#)
 - Enable power for transmitting (cp:pwtx), [180](#)
 - Enable power for transmitting (cp:txpw), [180](#)
- transparency
 - Set LCD font transparency (lcd.ft), [244](#)
- transparent
 - Transparent command (sdi12.t), [170](#)
- tree
 - File write Lua value tree (fc.lv), [62](#)
- tri
 - lcd.tri (Draw LCD triangle), [243](#)
- triangle
 - Draw LCD triangle (lcd.tri), [243](#)
- trigger
 - Event signal trigger (evt.sig), [52](#)
 - Trigger IoT connection (iot.go), [105](#)
 - Trigger SMS exchange (sms.go), [198](#)
- trim
 - iot.trim (Trim archive folder), [110](#)
 - Trim archive folder (iot.trim), [110](#)
 - Trim float value (float.dp), [67](#)

- try
 - c.toa_try (TOA retry time), [287](#)
- turn
 - Turn off device (pwr.off), [151](#)
 - Turn off output (out.off), [147](#)
 - Turn off the LCD (lcd.off), [237](#)
 - Turn on output (out.on), [147](#)
- twi
 - Query TWI bus (twi.qy), [205](#)
 - Read TWI (twi.rd), [205](#)
 - twi.qy (Query TWI bus), [205](#)
 - twi.rd (Read TWI), [205](#)
 - twi.wr (Write TWI), [206](#)
 - Write TWI (twi.wr), [206](#)
- two
 - Multiply-and-sum two arrays (fao.xs), [81](#)
- tx
 - cp:tx (Transmit data), [185](#)
 - modbus.tx (Transmit MODBUS RTU frame), [137](#)
 - Wait for TX to complete (cp:wtx), [186](#)
- txf
 - cp:txf (Set transmit flow control), [185](#)
- txpw
 - cp:txpw (Enable power for transmitting), [180](#)
- txt
 - alm.txt (Alarm get text), [22](#)
 - app.txt (App fetch text block), [31](#)
 - hash.txt (Convert hash value to text), [95](#)
 - i.smp.txt (Last sample text), [300](#)
 - job.txt (Get job schedule), [118](#)
 - nmea.txt (Convert table to NMEA string), [145](#)
 - sms.txt (Send text), [199](#)
 - ui.txt (Get UI element text), [262](#)
 - utc.txt (Text of UTC), [213](#)
- txt01
 - c.alm_txt01 (Alarm text names), [271](#)
- txt02
 - c.alm_txt02 (Alarm text names), [271](#)
- txtg
 - cp:txtg (Set transmit time guard gap), [186](#)
- tz
 - c.iot_tz (Time-zone override), [277](#)
 - utc.tz (Get time zone), [213](#)
- U
- u16u8
 - util.u16u8 (Convert UTF-16 to UTF-8), [224](#)
- u8a
 - util.u8a (Convert UTF-8 to ASCII), [225](#)
- u8u16
 - util.u8u16 (Convert UTF-8 to UTF-16), [225](#)
- ui
 - Draw UI box (ui.box), [259](#)
 - Get the top UI element (ui.top), [261](#)
 - Get UI element text (ui.txt), [262](#)
 - Pop all UI (ui.popa), [261](#)
 - Pop UI element (ui.pop), [260](#)
 - Push UI element (ui.push), [261](#)
 - Set UI colours (ui.set.c), [263](#)
 - Set UI filename (ui.set.i8n), [264](#)
 - Set UI font positions (ui.set.ui), [267](#)
 - Set UI idle time (ui.set.idle), [264](#)
 - Set UI key mapping (ui.set.km), [265](#)
 - Set UI key repeat (ui.set.kr), [265](#)
 - Set UI rounded corners (ui.set.r), [266](#)
 - Start the UI (ui.go), [260](#)
 - ui.box (Draw UI box), [259](#)
 - ui.go (Start the UI), [260](#)
 - ui.key (Draw soft key), [260](#)
 - ui.pop (Pop UI element), [260](#)
 - ui.popa (Pop all UI), [261](#)
 - ui.push (Push UI element), [261](#)
 - ui.set.bklt (Control LCD backlight), [263](#)
 - ui.set.c (Set UI colours), [263](#)
 - ui.set.i8n (Set UI filename), [264](#)
 - ui.set.idle (Set UI idle time), [264](#)
 - ui.set.km (Set UI key mapping), [265](#)
 - ui.set.kr (Set UI key repeat), [265](#)
 - ui.set.onebtn, [266](#)
 - ui.set.r (Set UI rounded corners), [266](#)
 - ui.set.showsk, [267](#)
 - ui.set.ui (Set UI font positions), [267](#), [267](#)
 - ui.ssk (SET key icon), [261](#)
 - ui.top (Get the top UI element), [261](#)
 - ui.txt (Get UI element text), [262](#)

- ui.val (Show values on LCD), [262](#)
 - ui.yes (Show YES tick), [262](#)
- uibar
 - uiBar (uiBar callback), [268](#)
 - uiBar callback (uiBar), [268](#)
- uicell
 - uiCell (uiCell callback), [268](#)
 - uiCell callback (uiCell), [268](#)
- uismp
 - uiSmp (uiSmp callback), [269](#)
 - uiSmp callback (uiSmp), [269](#)
- ujob
 - c.iot_ujob (Update job schedule), [277](#)
- ulck
 - cp:ulck (Unlock the COM port), [186](#)
- unlock
 - Unlock the COM port (cp:ulck), [186](#)
- unpack
 - struct.unpack (Unpack structure), [202](#)
 - Unpack string to table (modbus.fs2t), [132](#)
 - Unpack structure (struct.unpack), [202](#)
- until
 - Coro wait until (coro.wu), [44](#)
 - Receive until (cp:rxu), [183](#)
- up
 - IoT hang up modem (iot.hum), [106](#)
 - Modem power up time (i.cell_X_pu), [294](#)
 - Wake up HydrINS (hydrins.wake), [99](#)
- upd
 - alm.upd (Alarm update), [22](#)
 - ao:upd (Alarm Object update), [27](#)
 - c.iot_upd (Update URL or path), [278](#)
 - ct:upd (CRC update), [47](#)
 - ht:upd (Hash update), [96](#)
- update
 - Alarm Object update (ao:upd), [27](#)
 - Alarm update (alm.upd), [22](#)
 - CRC update (ct:upd), [47](#)
 - File check update (fc.chkupd), [58](#)
 - Hash update (ht:upd), [96](#)
 - IoT update callback (iotUpd), [115](#)
 - update (TERM), [313](#)
 - Update an array entry (fao:set), [78](#)
 - Update job schedule (c.iot_ujob), [277](#)
 - Update URL or path (c.iot_upd), [278](#)
- url
 - Base URL (c.iot_url), [278](#)
 - c.iot_url (Base URL), [278](#)
 - c.toa_url (TOA URL), [287](#)
 - Data process URL or path (c.iot_data), [276](#)
 - Set IoT server URL (iot.set), [109](#)
 - TOA URL (c.toa_url), [287](#)
 - Update URL or path (c.iot_upd), [278](#)
 - User process URL or path (c.iot_user), [278](#)
- us
 - rt.us (Delay microseconds), [165](#)
- usb
 - Check USB connected (usb.on), [208](#)
 - Send log string to USB (usb.log), [208](#)
 - usb.log (Send log string to USB), [208](#)
 - usb.on (Check USB connected), [208](#)
- use
 - Archive use gzip (c.arc_gz), [272](#)
 - Use gzip compression (c.iot_gz), [276](#)
 - Use password on engineer serial terminal (c.term_epw), [282](#)
 - Use Temp-File-and-Rename for FTP (c.iot_tfr), [277](#)
 - util.use (Include Lua library), [226](#)
- used
 - Estimated charge used (i.pwr_ccu), [301](#)
 - Set coulombs used (pwr.scu), [152](#)
- user
 - c.cell_user (Internet username), [275](#)
 - c.iot_user (User process URL or path), [278](#)
 - c.user_ins (Interval for user mode), [288](#)
 - c.user_onm (User mode online time), [288](#)
 - Interval for user mode (c.user_ins), [288](#)
 - IoT abort user process (iot.aup), [102](#)
 - IoT user callback (iotUser), [115](#)
 - User mode online time (c.user_onm), [288](#)
 - User process URL or path (c.iot_user), [278](#)
- username
 - Internet username (c.cell_user), [275](#)
- utc
 - Check if UTC is sensible (utc.ok), [211](#)
 - Convert alive time to UTC (utc.a2utc), [209](#)
 - Convert string to UTC (utc.val), [214](#)

- Convert UTC to alive time (utc.utc2a), [214](#)
 - fc.utc (File get UTC creation), [65](#)
 - File get UTC creation (fc.utc), [65](#)
 - Filename UTC (utc.fn), [209](#)
 - Format UTC (utc.fmt), [209](#)
 - i.rt_utc (Current time), [299](#)
 - i.smp.utc (Last sample time), [300](#)
 - job.utc (Set a UTC based job), [120](#)
 - Return current UTC (utc.now), [210](#)
 - Set a UTC based job (job.utc), [120](#)
 - Set UTC time and timezone (utc.set), [212](#)
 - Split UTC time (utc.split), [212](#)
 - Text of UTC (utc.txt), [213](#)
 - utc.a2utc (Convert alive time to UTC), [209](#)
 - utc.fmt (Format UTC), [209](#)
 - utc.fn (Filename UTC), [209](#)
 - utc.job, [210](#)
 - utc.now (Return current UTC), [210](#)
 - utc.offt (Get power off time), [210](#)
 - utc.ok (Check if UTC is sensible), [211](#)
 - utc.rtc (Get/Set Real Time Clock), [211](#)
 - utc.set (Set UTC time and timezone), [212](#)
 - utc.split (Split UTC time), [212](#)
 - utc.spm (Get seconds past midnight), [213](#)
 - utc.txt (Text of UTC), [213](#)
 - utc.tz (Get time zone), [213](#)
 - utc.utc2a (Convert UTC to alive time), [214](#)
 - utc.val (Convert string to UTC), [214](#)
 - utc.wds (Get whole days), [214](#)
 - utc2a
 - utc.utc2a (Convert UTC to alive time), [214](#)
 - utf
 - Convert ASCII to UTF-16 (util.au16), [216](#)
 - Convert ASCII to UTF-8 (util.au8), [216](#)
 - Convert UTF-16 to UTF-8 (util.u16u8), [224](#), [224](#)
 - Convert UTF-8 to ASCII (util.u8a), [225](#)
 - Convert UTF-8 to UTF-16 (util.u8u16), [225](#), [225](#)
 - util
 - util.add (Add tables), [215](#)
 - util.addi (Add indexed tables), [215](#)
 - util.au16 (Convert ASCII to UTF-16), [216](#)
 - util.au8 (Convert ASCII to UTF-8), [216](#)
 - util.bits (Create bits table), [217](#)
 - util.btab (Create table from table with bits), [217](#)
 - util.cmpi (Compare indexed tables), [218](#)
 - util.codesize (Estimate Lua codesize), [218](#)
 - util.csvt (Convert CSV text to table), [219](#)
 - util.cuti (Cut indexed table), [219](#)
 - util.dct (Deep copy table), [220](#)
 - util.find (Find), [220](#)
 - util.iter (Iterate table), [221](#)
 - util.kvc (Create key-value container object), [222](#)
 - util.kvc (Create new key-value container object), [128](#)
 - util.remi (Remove indexed table), [222](#)
 - util.tcsv (Convert table to CSV text), [223](#)
 - util.tia (Append index table), [223](#)
 - util.tio (OR indexed tables), [224](#)
 - util.u16u8 (Convert UTF-16 to UTF-8), [224](#)
 - util.u8a (Convert UTF-8 to ASCII), [225](#)
 - util.u8u16 (Convert UTF-8 to UTF-16), [225](#)
 - util.use (Include Lua library), [226](#)
 - util.zdec (Z-base-32 decode), [226](#)
 - util.zenc (Z-base-32 encode), [227](#)
- V**
- v2d
 - float.v2d (Integer to double), [73](#)
 - v2f
 - float.v2f (Integer value to float), [73](#)
 - val
 - alm.val (Alarm value), [23](#)
 - ao:val (Alarm Object value), [28](#)
 - dpc.val (Digital pulse get value), [49](#)
 - hash.val (Convert hash string to value), [95](#)
 - ui.val (Show values on LCD), [262](#)
 - utc.val (Convert string to UTC), [214](#)
 - valid
 - Check whether signals are valid (cp:okr), [178](#)
 - value
 - Alarm Object value (ao:val), [28](#)
 - Alarm value (alm.val), [23](#)
 - Convert boolean value (rt.bool), [157](#)

- Convert hash string to value (hash.val), [95](#)
- Convert hash value to text (hash.txt), [95](#)
- Create key-value container object (util.kvc), [222](#)
- Create new key-value container object (util.kvc), [128](#)
- Digital pulse get value (dpc.val), [49](#)
- File write Lua value tree (fc.lv), [62](#)
- Float to integer value (float.f2v), [67](#)
- Integer value to float (float.v2f), [73](#)
- Push value (fao:push), [77](#)
- Set receive timeout value (cp:rxto), [183](#)
- Trim float value (float.dp), [67](#)
- values
 - add/sub/mul/div values (fao:add), [75](#)
 - add/sub/mul/div values (fao:div), [75](#)
 - add/sub/mul/div values (fao:mul), [75](#)
 - add/sub/mul/div values (fao:sub), [75](#)
 - Copy values (fao:copy), [76](#)
 - Fill with sin values (fao:sin), [78](#)
 - Get running statistics values (float.mean), [69](#)
 - Get running statistics values (rso:get), [87](#)
 - Push values into running regression object (rro:push), [86](#)
 - Push values into running stats object (rso:push), [88](#)
 - Set Alarm Object values (ao:set), [27](#)
 - Set out-of-bound values (mpco:oob), [82](#)
 - Set the boost values (pwr._bv), [154](#)
 - Show values on LCD (ui.val), [262](#)
 - Sum all values (fao:sum), [80](#)
- var
 - c.iot_var (Push variance), [279](#)
 - i.iot_var (Variance time applied), [296](#)
 - job.var (Set or get job variance), [118](#)
- variance
 - Push variance (c.iot_var), [279](#)
 - Set or get job variance (job.var), [118](#)
 - Variance time applied (i.iot_var), [296](#)
- ver
 - c.tls_ver (TLS version), [285](#)
 - i.fw_ver (Firmware version), [295](#)
- version
 - Bootloader version (i.fw_blv), [295](#)
 - Firmware version (i.fw_ver), [295](#)
 - TLS version (c.tls_ver), [285](#)
- vertical
 - Draw LCD vertical line (lcd.lv), [241](#)
- voltage
 - Query the transient minimum voltage (pwr.tmv), [153](#)
- W
- wait
 - Coro wait until (coro.wu), [44](#)
 - Coro yield or wait (coro.w), [43](#)
 - Coro yield or wait (coro.yld), [43](#)
 - Event wait (evt.lwait), [51](#)
 - Wait for TX to complete (cp:wtx), [186](#)
- waiting
 - Number of RX bytes waiting (cp:rxw), [184](#)
- wake
 - hydrins.wake (Wake up HydrINS), [99](#)
 - Wake up HydrINS (hydrins.wake), [99](#)
- wavelet
 - Wavelet Transform (DWT/CWT) (fao:wt), [80](#)
- Wavelet Transform (DWT), [80](#), [81](#)
- wc
 - fc.wc (File wildcard compare), [65](#)
- wd
 - float.wd (Write floats/doubles to file), [73](#)
- wde
 - wde (TERM), [304](#)
- wde0
 - wde0 (TERM), [304](#)
- wds
 - utc.wds (Get whole days), [214](#)
- wf
 - float.wf (Write floats/doubles to file), [73](#)
- whether
 - Check whether sequence complete (sdi12.done), [168](#)
 - Check whether signals are valid (cp:okr), [178](#)
- whole
 - Get whole days (utc.wds), [214](#)
- width

- Get LCD text width (lcd.gtw), [245](#)
- wildcard
 - File wildcard compare (fc.wc), [65](#)
- wipe
 - cfg.key.wipe (Erase a secret key), [40](#)
- wmr
 - modbus.wmr (Write MODBUS multiple registers), [137](#)
- work
 - Allow hard work (pwr.work), [153](#)
 - pwr.work (Allow hard work), [153](#)
 - work (TERM), [305](#)
- wr
 - twi.wr (Write TWI), [206](#)
- write
 - File set file write mode (fc.sfw), [64](#)
 - File write Lua value tree (fc.lv), [62](#)
 - Write floats/doubles to file (float.wd), [73](#)
 - Write floats/doubles to file (float.wf), [73](#)
 - Write log entry (rt.log), [161](#)
 - Write MODBUS multiple registers (modbus.wmr), [137](#)
 - Write SIM FileID Binary (modem.simwb), [143](#)
 - Write single register (modbus.wsr), [138](#)
 - Write TWI (twi.wr), [206](#)
- wsr
 - modbus.wsr (Write single register), [138](#)
- wt
 - fao:wt (Wavelet Transform (DWT/CWT)), [80](#)
- wtx
 - cp:wtx (Wait for TX to complete), [186](#)
- wu
 - coro.wu (Coro wait until), [44](#)
- X**
- x2d
 - float.x2d (Hex to double), [74](#)
- x2f
 - float.x2f (Hex to float), [74](#)
- xbm
 - Draw LCD XBM bitmap string (lcd.xbm), [243](#)
 - lcd.xbm (Draw LCD XBM bitmap string), [243](#)
- xor
 - Bitwise XOR (bit.bxor), [34](#)
- xs
 - fao:xs (Multiply-and-sum two arrays), [81](#)
- xy
 - mpco:xy (Get x y pair), [83](#)
- xyt
 - mpco:xyt (Get x and y tables), [83](#)
- Y**
- yes
 - Show YES tick (ui.yes), [262](#)
 - ui.yes (Show YES tick), [262](#)
- yield
 - Coro yield or wait (coro.w), [43](#)
 - Coro yield or wait (coro.yld), [43](#)
- yld
 - coro.yld (Coro yield or wait), [43](#)
- Z**
- zdec
 - util.zdec (Z-base-32 decode), [226](#)
- zenc
 - util.zenc (Z-base-32 encode), [227](#)
- zombie
 - Set modem anti-cell-zombie timeout (iot.azt), [102](#)
- zone
 - Get time zone (utc.tz), [213](#)
 - Time-zone override (c.iot_tz), [277](#)