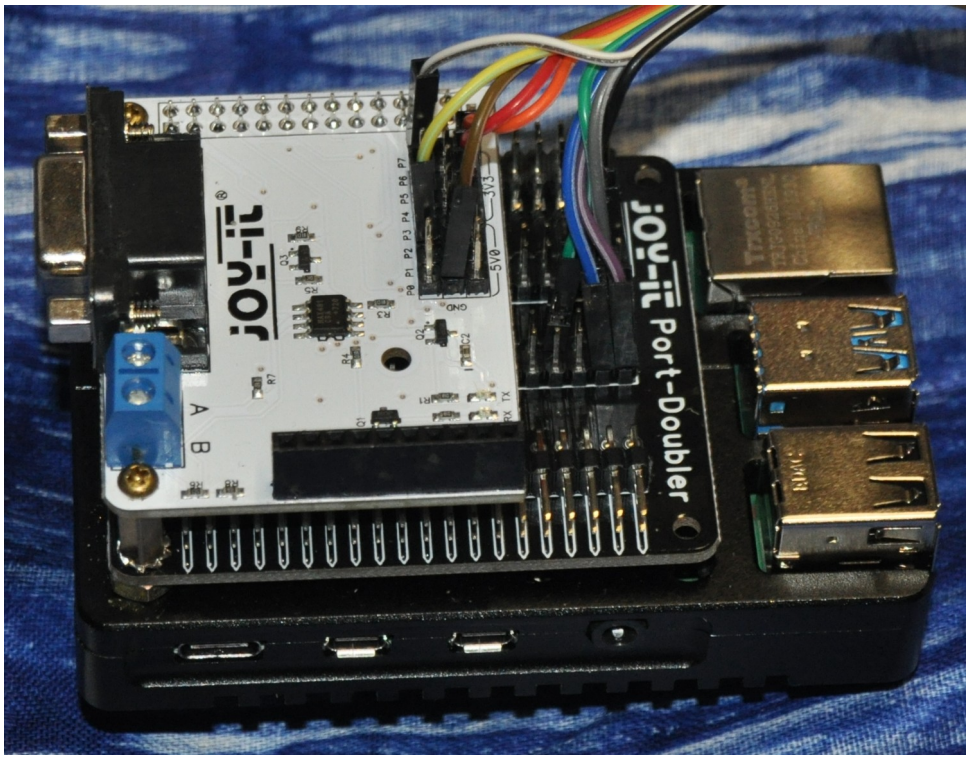
Albrecht Weinert

Raspberry project 01 / sweet home

C software documentation



Prof. Dr.-Ing. Albrecht Weinert is founder and director of weinert-automation.

He developed safety critical and highly available automation systems with Siemens AG and was Professor of Computer Science at Bochum University of Applied Sciences.

albrecht@a-weinert.de

Generated by Doxygen 1.9.5 Photos by A. Weinert

References

We use identical reference numbers in [29 .. 31] , hence some gaps.

[31] Albrecht Weinert, Raspberry for remote services, Report, May 2017,
This paper (the last actual version): a-weinert.de/pub/raspberry4remoteServices.pdf

[32] Albrecht Weinert, Raspberry Pi GPIO mit Java — Java statt C, Java Magazin 1|2020, pp.78-84, GIT .pdf jaxenter.de/java/java-programmiersprache-c-raspberry-pi-90176

[51] Raspberry Org, FTP, Tips on using SSHFS
https://www.raspberrypi.org/documentation/remote-access/ssh/sshfs.md

[52] Raspberry Org, SFTP, Tips on using SFTP
https://www.raspberrypi.org/documentation/remote-access/ssh/sftp.md

[53] Raspberry Org, SFTP, Tips on enabling WLAN by command line
https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md

[56] Broadcom, BCM2835 ARM Peripherals, data sheet 2012
https://www.raspberrypi.org/app/uploads/2012/02/BCM2835-ARM-Peripherals.pdf

[57] Gert van Loo, QA7ARM Quad A7 core, Technical report on BCM2836, Rev3.4 2014
https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2836/QA7_rev3.4.pdf

[58] Broadcom, ARM® Cortex®-A53 MPCore Processor, Rev. r0p2Technical Reference Manual
DDI0500D_cortex_a53_r0p2_trm.pdf (BCM2837 is Quad-core 64-bit ARM cortex A53 CPU)
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500d/DDI0500D_cortex_a53_r0p2_trm.pdf

[59] Broadcom, ARM® Cortex®-A Series, Version 1.0, Programmer's Guide for ARMv8-A
http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/DEN0024A_v8_architecture_PG.pdf

[60] Shore, Chris, ARM, Porting to 64-bit ARM, white paper July 2014,
https://community.arm.com/cfs-file/ ... /Porting-to-ARM-64_2D00_bit.pdf

[61] N.N., Joan, pigpio library – Download & Install, http://abyz.me.uk/rpi/pigpio/download.html

[62] N.N., Joan, pigpio library – pigpio C interface,
http://abyz.me.uk/rpi/pigpio/cif.html

[62] N.N., Joan, pigpio library – pigpiod C interface, http://abyz.me.uk/rpi/pigpio/pdif2.html

[65] Modicon, Modbus Protocol, Reference Guide
PI–MBUS–300 Rev. J 1996, http://modbus.org/docs/PI_MBUS_300.pdf

[66] Modbus Org, MODBUS Application Protocol Specification V1.1b3, 2012
http://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

[67] OASIS, MQTT V3.1.1 Protocol Specification, 2014
http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

[68] Rahmann, Leila F., Tutorial on Mosquitto and Paho, Eindhoven 2017,
http://www.win.tue.nl/~lrahman/iot_2016/tutorial/MQTT_2016.pdf

[69] Mosquitto, The API documentation https://mosquitto.org/api/files/mosquitto-h.html

Chapter 1

rasProject1 - process control with Raspberry Pi

The ideas behind using a Raspberry Pi for process control are published in **raspberry4remote↩ Services.pdf** (dir) at **a-weinert.de**.

1.1 Process control features

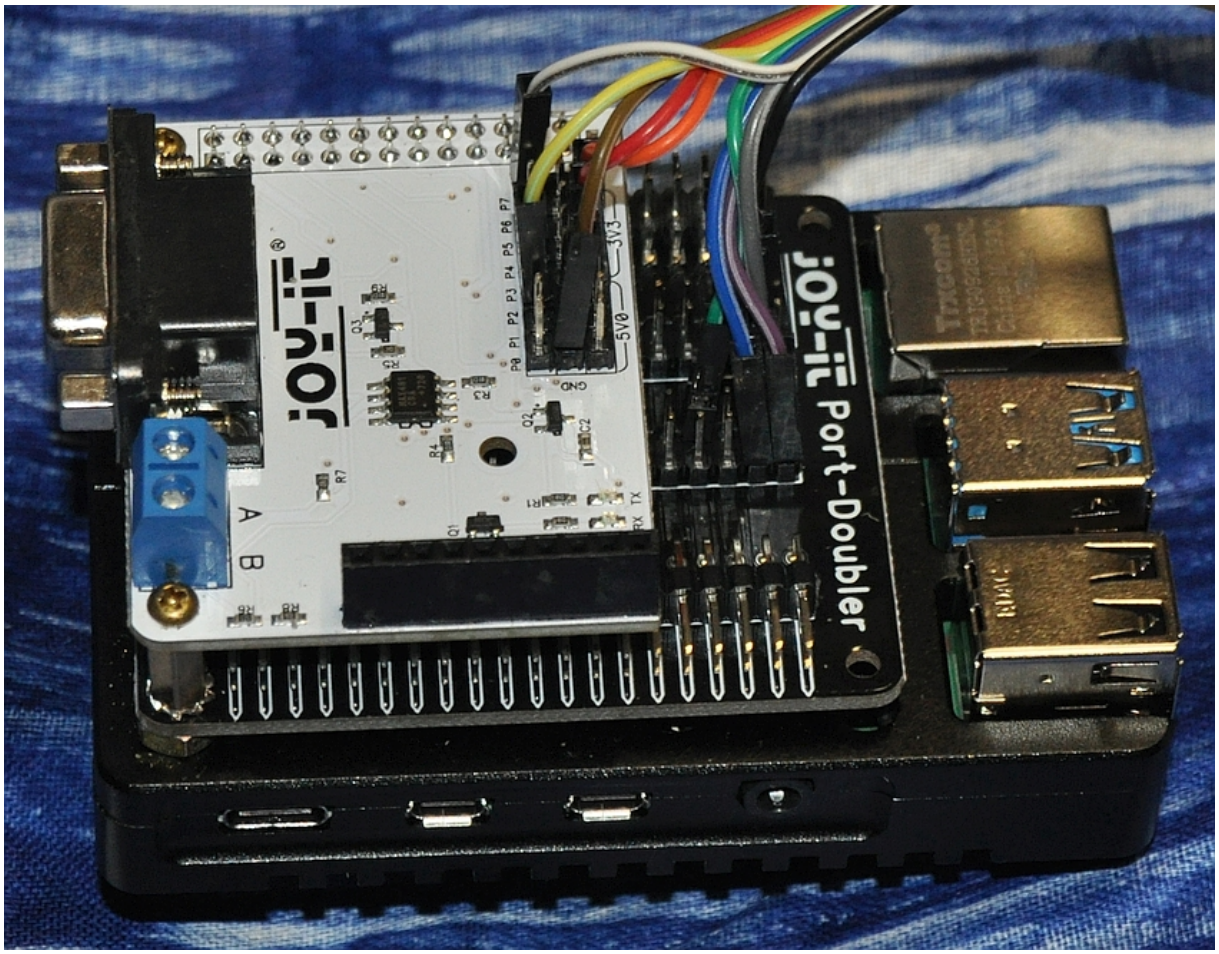


Figure 1.1 RasPi4 with RS485 and GPIO

By certain configuration and utilising constraints we get 24/7 base for control software with

- cycles (1ms, 10ms, 20ms, 100ms, 1s)
- support for Modbus, MQTT, 1-wire and more
- HMI via Web-interface
- watchdog

Compared with industrial PLCs and weAut01 there is

- no supervising of power supply, and, hence
- no (cycle for) programmable reaction on power outages

Devices supported by the library include

- electronic electricity meter
- PV inverter
- bar code / QR code reader

1.2 A note on Copyright and License

For software and documentation developed for this project:

The software is open source and available by

SVN checkout. It uses open source libraries by other authors. Those parts keep their original license and author's copyright.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Module Documentation

5.1 Raspberries architecture differences

The different Raspberry types, like Pi1, 3, 4 and Zero, can mostly be handled transparently to the control programs.

Directories

- directoryarch

  Raspberry Pi variants — architecture definitions.

The different Raspberry types, like Pi1, 3, 4 and Zero, can mostly be handled transparently to the control programs.

This is done by make variables by a make variable TARGET identifying a concrete target machine and, hence, the architecture, i.e the PLATFORM variable.The latter can also be set directly as PLATFORM = raspberry_03 e.g.. raspberry_03 is the default by the way.

The mechanism used is calculated make includes and calculated C includes.

Chapter 6

Data Structure Documentation

6.1 _km Struct Reference

A key matrix connected to a Raspberry Pi's GPIO pins.

`#include <weGPIOkeyM.h>`

Data Fields

- int **colRow** [13]
  GPIO number [0.. noCols+noRows-1].
- uint32_t **colRowMsk** [13]
  GPIO mask (one bit, assuming 0..31)
- uint32_t **colsMsk**
  mask for all column pins
- uint32_t keyChange
  scan step index 0.. .noCoRo (for/by crScanStep())
- uint8_t keyError
  Counter for key scan errors.
- uint8_t **keyPresLast** [30]
  for change detection
- uint8_t keyPressed [30]
  Counter for key connection seen in one complete scan cycle.
- uint32_t **keyState**
  state of last change
- uint32_t **keyStateLast**
  the state before (may be long ago)
- char **keyVal** [30]
  *key letter [rowInd * noCols + colInd]*
- pt2lastScanStep lastScanStep
  User supplied function, or null.
- const uint8_t **noCols**
  number of columns
- int **noCoRo**
  number of columns and rows, i.e. number of IO pins used

- int **noKeys**
  number of keys, i.e. columns ∗ rows
- const uint8_t **noRows**
  number of rows
- uint32_t **roCoMsk**
  mask for all matrix pins
- uint32_t **rowsMsk**
  mask for all rows pins
- uint32_t **stateStamp**
  time stamp of state of last change; 1ms tick
- uint32_t **stateStampLast**
  time stamp of state before
- int **thePi**
  gpio's Raspberry identifier

6.1.1 Detailed Description

A key matrix connected to a Raspberry Pi's GPIO pins.

The structure type keyMatrix describes a key matrix with column and row lines connected directly to the GPIO pins. At the cross points there would be normally open (NO) push button switches. An example for such matrix is the 12 keys matrix (3 columns, 4 rows) EOZ Clavier S.series, 12 touches, Matricell, Téléphone. The maximum number of keys supported is 30 (6∗5 or 3∗10) delimited by the array size in this structure and, of course, by the GPIOs put to Raspberry's pin header, as well as by the approach to use the bits 0..29 of an uint32_t as compact switch state.

When defining a keyMatrix structure for a concrete key matrix one has to initialise correctly the fields, .noCols by the number of columns, .noRows by the number of rows, the array .colRow by (.noCols + .noRows) GPIO pin numbers (columns first) and the array .keyVal by (.noCols ∗ .noRows) key letters (row by row).

The other field will be set by once calling keyMatrixInit before doing any other (IO) operation.

6.1.2 Field Documentation

6.1.2.1 keyPressed

```
uint8_t keyPressed[30]
```

Counter for key connection seen in one complete scan cycle.

Index: [rowInd ∗ noCols + colInd] Value: 0: not pressed; 2: OK pressed; $>$ 2: seen in combination with key error

6.1.2.2 keyError

```
uint8_t keyError
```

Counter for key scan errors.

A scan step seeing more than one row and more than one column (hi) indicated a non recognisable multiple keys pressed.

6.1.2.3 keyChange

```
uint32_t keyChange
```

scan step index 0.. .noCoRo (for/by crScanStep())

key state change count

6.1.2.4 lastScanStep

```
pt2lastScanStep lastScanStep
```

User supplied function, or null.

See setKeyMatrixScanFunct()

The documentation for this struct was generated from the following file:

- include/weGPIOkeyM.h

6.2 cmdLookUp_t Struct Reference

Structure for a defined remote command.

```
#include <sweetHome.h>
```

Data Fields

- char **command** [22]
  command max. 20 characters
- uint32_t **mask**
  command value (usually 1 bit set)

6.2.1 Detailed Description

Structure for a defined remote command.

This structure combines a command's short / mnemonic name used as query string, like "startPump", and the command's (cmdBits_t) bit value resp. mask usually defined as makro, like START_HWPUMP_COMMAND.
An array cmdLookUp_t cmdLookUp[] (e.g.) is used to define and look-up a command given in a query string. The array's end must be an entry {"", 0}, i.e. empty mnemonic and no command bit.

The documentation for this struct was generated from the following file:

- include/sweetHome.h

6.3 cycTask_t Struct Reference

Cyclic or event driven task / threads structure.

`#include <weUtil.h>`

Data Fields

- union {
  cycTaskEventData_t cycTaskEventData
  < *allow different event data for cyclic and other event types*
  };

  Event data for cyclic and other event types.
- pthread_cond_t **cond**

  the event occurred condition
- uint32_t **count**

  the event counter (modified by manager, only)
- timespec **stamp**

  absolute / monotonic event stamp (dto.)

6.3.1 Detailed Description

Cyclic or event driven task / threads structure.

This structure supports the organisation of tasks respectively threads to work all on a same event type. One common case is the event being a next time interval, like the next 100ms entered, and, as thread e.g., the process control tasks to work on the 100ms cycle.

Such approach involves two types of threads:
One controller/manager determining the event, recording it by increasing the event counter and signalling all worker threads.
Zero to some worker threads doing work on every or every other etc. event, usually by holding and updating an event counter value at which to do the work.

The main purpose are absolute time driven cyclic tasks as usual in industrial PLCs.
For the standard cycles provided here, 1ms, 10ms .. 100ms, 1s the handler thread is provided as singleton doing other time and date related jobs for all; see theCyclistStart(), theCyclistWaitEnd() and endCyclist().

See cyc1ms, cyc10ms, cyc20ms, cyc100ms, cyc1sec
See also have1msCyc, have10msCyc ... have1secCyc

6.3.2 Field Documentation

6.3.2.1 cycTaskEventData

`cycTaskEventData_t cycTaskEventData`

< allow different event data for cyclic and other event types

cyclic event data

6.3.2.2

```
union { ...  } @18
```

Event data for cyclic and other event types.

This union allows for different event data types for different types of events like cyclic ticks, or any other event types. Anyway the information must be copied by the event controller / manager under mutex lock to the event handler's (i.e. this) task structure.

The documentation for this struct was generated from the following file:

- include/weUtil.h

6.4 cycTaskEventData_t Struct Reference

Event data for cyclic tasks.

```
#include <weUtil.h>
```

Data Fields

- uint8_t **cnt10inSec**

  0..9; counts 100ms events in the second
- uint64_t cnt1ms

  A ms counter for cycles and tasks.
- uint8_t **cnt210sec**

  0..209 s counter (to provide n s periods)
- timespec **cycStart**

  monotonic start time of the cycle
- int **cycStartMillis**

  millisecond (0..999) missing in struct tm
- struct tm cycStartRTm

  The broken down calculated local start time.
- int **hourOffs**

  actual time zone offset in hours (incl. DST +: east)
- uint8_t **msTo100Cnt**

  0..99; at 0 we will have a 100ms event
- uint32_t realSec

  The real time epoch seconds.
- char **rTmTxt** [34]

  local time as text

6.4.1 Detailed Description

Event data for cyclic tasks.

This structure holds data — mainly time and date by diverse clocks and cycle counters — to be used by cyclic tasks. It will be provided as as cycTask_t.cycTaskEventData.

6.4.2 Field Documentation

6.4.2.1 realSec

```
uint32_t realSec
```

The real time epoch seconds.

The value may jump with coarse NTP corrections. And it will be not precise and or jump with leap seconds, which can't be handled in UTC.

6.4.2.2 cycStartRTm

```
struct tm cycStartRTm
```

The broken down calculated local start time.

This is the cycle start / event local time.
It is kept updated by the cyclist (when having been started; see theCyclistStart).

6.4.2.3 cnt1ms

```
uint64_t cnt1ms
```

A ms counter for cycles and tasks.

This ms counter may be used to time stamp IO values and other events in a unique and sortable manner.
It is 1 in all first cycles (signalled as event) when the cyclist is started without delay. Otherwise the first value is this delay. The value 0 will will correspond to allCycStart.

Usually the lower 32 bits (cast (uint32_t)) should be sufficient. The lower 32 bit will wrap after 49.7 days.
For an absolute stamp in seconds resolution see cycTaskEventData_t.realSec and

The documentation for this struct was generated from the following file:

- include/weUtil.h

6.5 dayStrtVal_t Struct Reference

Day start values.

```
#include <sweetHome.h>
```

Data Fields

- int **dayInYear**
  Day in year.
- __time_t **localMidnight**
  Actual local midnight.
- float meterValDay [2][2]
  Start day meter values.
- uint32_t stmpSDay
  Absolute time stamp for the day's start values.
- __time_t **utcMidnight**
  Actual (local) UTC midnight.
- uint8_t valQual
  Quality of the values.

6.5.1 Detailed Description

Day start values.

This structure combines mainly meter reading values for import and export work (cf. makros Wimp, Wexp, PL1 etc.) to be put in / got from a binary daily retain file.

Attention: Variables of this type dayStrtVal_t are kept in shared memory by several programs. Changing order or size for sake of one program almost certainly will spoil the others.

6.5.2 Field Documentation

6.5.2.1 stmpSDay

```
uint32_t stmpSDay
```

Absolute time stamp for the day's start values.

This stamp marks the import and export work values of a successful meters reading very early in the current day. If a early day reading was neither taken nor read from retained values the application would put the first successful reading after program start here.

0 marks no values taken so far.

6.5.2.2 meterValDay

```
float meterValDay[2][2]
```

Start day meter values.

This array contains for every meter the import and export work as quadReg_t.f[0] and [1] respectively taken at dayStrtVal_t.stmpSDay.

6.5.2.3 valQual

```
uint8_t valQual
```

Quality of the values.

0: invalid
1: taken any time of day at program start
2: taken at predefined (fixed) time at change of day

Note: Even when having high quality values (for consecutive days), consider days may have 23 or 25 h in zone time, also.

The documentation for this struct was generated from the following file:

- include/sweetHome.h

6.6 dcf77recPerData_t Struct Reference

Data for one received DCF77 AM period.

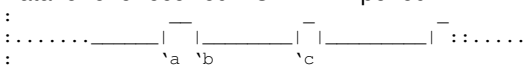
```
#include <weDCF77.h>
```

Data Fields

- uint32_t cbTic
  The system tick at second's start (notation).
- uint32_t per
  The period's duration.
- char sysClk [14]
  The system time as text hh:mm:ss.mmm at second's start.
- uint32_t tic
  The GpioD time tick.
- uint32_t tim
  The modulation's duration.

6.6.1 Detailed Description

Data for one received DCF77 AM period.



This structure holds the minimal values of one received DCF77 modulation period, i.d. amplitude 15% from a to b and full modulation from b to c = next a.
Nothing is said about a correct timing signal or a receiver module error or EMI.

6.6.2 Field Documentation

6.6.2.1 tic

```
uint32_t tic
```

The GpioD time tick.

This is the tick (see get_current_tick()) at point a, i.e. begin of 15% AM. get_current_tick() is a unsigned 32 bit μs counter wrapping around about every 72 min.
For a correct modulation signal this marks a second's begin.
As described at dcf77recPerData_t::per, with lower grade AM receiver modules it may also mark the beginning of a non filtered spike.

The current `uint32_t` tick is got by `get_current_tick(int pi)`. It can be used in user to calculate how far in past the the last received DCF77 second's start is.
`delay = get_current_tick(::thePi) - recStruc.tic`
will work even when the unsigned 32 bit tick wraps around about every 72 min.

6.6.2.2 sysClk

```
char sysClk[14]
```

The system time as text hh:mm:ss.mmm at second's start.

In fact it is the time when the call back function for start of 15% AM is called.

6.6.2.3 cbTic

```
uint32_t cbTic
```

The system tick at second's start (notation).

In fact it is the time when the call back function for start of 15% AM is called. By the event to callback delay cbTic should be later than tic.

6.6.2.4 tim

```
uint32_t tim
```

The modulation's duration.

This is the time from point a to b in μs. With a correct signal this difference is the length of the transmitted serial bit. Ideally it would be either 100000μs for logical 0 respectively FALSE or 200000μs for logical 1 respectively TRUE.

6.6.2.5 per

```
uint32_t per
```

The period's duration.

This is the time from point a to c in μs. This difference is the full period. Ideally it would be either 1000000μs for seconds 0 to 57 and 2000000μs for the seconds 58 and 59 combined. In the case of a leap second the numbers would be 0 to 58 and 59 and 60.

Hence this variable's value is the real source of all information, like start of minute and the serial telegram.

For real world DCF77 AM receivers one will see a wide range instead of these ideal numbers. Additionally, lower grade receivers (the Pollin module being the most prominent of them) will insert false ON periods. Sometimes those "spikes" affect a third of all seconds. Additionally some seconds will be infected by more than one spike.

Obviously, without some filtering one would never get sensible information from lower grade modules or conditions. For the very first one or two receptions (after registering the receive call back, usually dcf77receiveRec) this value and dcf77recPerData_t::tim will be wrong.

The documentation for this struct was generated from the following file:

- include/weDCF77.h

6.7 dualReg_t Union Reference

A 32 bit union.

```
#include <basicTyCo.h>
```

6.7.1 Detailed Description

A 32 bit union.

This structure serves formatting and endianess plumbing purposes.

Besides that some Modbus devices use two (dual) so called registers of the standard Modbus 16bit size for one float. Some Modbus manufactures call such dual register a "parameter".

As modbus has no data types except for the 16 bit register (with whatever semantic) libmodbus will handle the endianess for that two byte registers but can do nothing for bigger data types. Many modbus servers will use 32 bit and longer types. EASTRON smart meters handle all measurements as 32 bit floats and call that type "parameter". Hence 1 parameter is 2 registers in default big endian register ordering.

The union dualReg_t allows all endian repairs for such "parameter" type.

The documentation for this union was generated from the following file:

- include/basicTyCo.h

6.8 durDiscrPointData_t Struct Reference

Values for discrimination of duration.

`#include <weDCF77.h>`

Data Fields

- char c
  Charakter.
- unsigned i
  Index.
- char n [6]
  Name.
- uint32_t v
  Value.

6.8.1 Detailed Description

Values for discrimination of duration.

This structure holds the data of one discrimination point.
The primary value type are duration as uint32_t in μs.

6.8.2 Field Documentation

6.8.2.1 i

`unsigned i`

Index.

The lowest value in a chain (array or enum) shall get the index 0.
This value[0] shall be the upper value of the discrimination range [lowest possible / measurable values ... v[0]-1].
For an unsigned value type (used here) the lowest possible value is 0 while the lowest measurable value might be higher.
v[j] <= v[j+1] must hold.

6.8.2.2 v

`uint32_t v`

Value.

This is the upper bound value of the range.
The highest value in a chain (array or enum) is implied as the maximum possible / measurable value and must, hence, never be used in a comparison and shall be set to MXUI32 (=UINT32_MAX).
v[j] <= v[j+1] must hold.

6.8.2.3 n

```
char n[6]
```

Name.

This is a short (5 character max.) name of the discrimination range, like, e.g., "spike", "error", "1 sec", "0" etc.

6.8.2.4 c

```
char c
```

Charakter.

This is a recognisable character for the discrimination range, best unique for the complete chain. It may be used for (narrow) logs instead of durDiscrPointData_t::n.

The documentation for this struct was generated from the following file:

- include/weDCF77.h

6.9 memAlloc_t Struct Reference

Structure for allocated, re-allocatable memory.

6.9.1 Detailed Description

Structure for allocated, re-allocatable memory.

The documentation for this struct was generated from the following file:

- getLocalWeatherData.c

6.10 meterVal_t Struct Reference

One smart meter's readings.

```
#include <sweetHome.h>
```

Data Fields

- int **err**

  0: OK -99: not set
- int **hourOffs**

  actual offset in hours of local time zone (+ is east)
- uint16_t **ms**

  ms supplement to stmpS
- uint16_t **msDurRead**

  duration on Modbus read (successful or timed out)
- float **pSum**

  power sum (all, all three or the relevant phases) [W]
- char **rTmTxt** [34]

  local time as text; length is 32
- uint32_t stmpS

  Absolute time stamp for the current value.
- float vals [18]

  Compact and re-sorted meter readings.

6.10.1 Detailed Description

One smart meter's readings.

The (first) four timing values are the start of reading; to have the end add .msDurRead.

6.10.2 Field Documentation

6.10.2.1 stmpS

```
uint32_t stmpS
```

Absolute time stamp for the current value.

This value and its supplements meterVal_t.ms, meterVal_t.hourOffs and meterVal_t.rTmTxt are updated at every Modbus read.

6.10.2.2 vals

```
float vals[18]
```

Compact and re-sorted meter readings.

These values are to be updated on every error-free meter reading (by Modbus, RS485 e.g.). The indexes are: /code 0..2 : U (L1, L2, L3) 3..5 : I (L1, L2, L3) 6..8 : P (L1, L2, L3) 9 : f 10, 11 : Wimp Wexp 12..14 : phi (L1, L2, L3) 15..17 : reserve (not used in R.115) /endcode

The documentation for this struct was generated from the following file:

- include/sweetHome.h

6.11 modBvals_t Struct Reference

Modbus readings and other process values.

`#include <growattHome.h>`

Data Fields

- sdm124regs_t **hld0Regs**
  a stack of 124 holding registers
- int **hourOffs**
  actual offset in hours of local time zone (+ is east)
- sdm124regs_t **imp0Regs**
  a stack of 124 input registers
- uint16_t **ms**
  ms supplement to stmpS
- int **retHimp59**
  modbus 15 holding registers from 59 return
- int **retHimp9**
  modbus 20 holding registers from 9 return
- int **retRimp0**
  modbus 34 input register from 0 read return
- char **rTmTxt** [34]
  local time as text; length is 32
- uint32_t **stmpS**
  epoch time; s since 1.1.1970 (until 7.2.2106)

6.11.1 Detailed Description

Modbus readings and other process values.

This structure unites all inverter readings and settings plus some common evaluation and process control values.

The four timing data will mark the the evaluation and forwarding time of the complete structure/information. For the meter read times see the equivalent data in the structure .meterVal, which will be 300 .. 600ms older for the meter's latest read (+1s for every other meter read earlier).

The documentation for this struct was generated from the following file:

- include/growattHome.h

6.12 modRS_t Struct Reference

Structure for Modbus RS485 (RTU).

`#include <weModbus.h>`

Data Fields

- int **baud**

  default 9600
- uint16_t **conErrCnt**

  error count for connection trials
- modbus_t * **ctx**

  Pointer to (libmodbus) Modbus structure.
- char device [21]

  Modbus RTU / RS485 device address as string.
- int **hourOffs**

  actual offset in hours of local time zone (+ is east)
- uint16_t **ms**

  ms supplement to stmpS
- char **parity**

  default N
- modRS485state_t **rsState**

  interface state
- char **rTmTxt** [34]

  local time as text; length is 32
- uint32_t **stmpS**

  epoch time of last connect [error]; s since 1.1.1970
- int **stopBits**

  default 1

6.12.1 Detailed Description

Structure for Modbus RS485 (RTU).

6.12.2 Field Documentation

6.12.2.1 device

```
char device[21]
```

Modbus RTU / RS485 device address as string.

example: /dev/serial0 (Raspberry's UART GPIO15/16 TxD/RxD) or /dev/ttyS0

The documentation for this struct was generated from the following file:

- include/weModbus.h

6.13 modSharMem_t Struct Reference

Structure for shared memory.

`#include <growattHome.h>`

Data Fields

- uint16_t **copInCnt**

  values copy in count; incremented by server
- modBvals_t **modBvals**

  the values; copied in form the server

6.13.1 Detailed Description

Structure for shared memory.

The documentation for this struct was generated from the following file:

- include/growattHome.h

6.14 modTCP_t Struct Reference

Structure for Modbus TCP.

`#include <weModbus.h>`

Data Fields

- char addr [21]

  Modbus IP (4) address as string.
- modbus_t * **ctx**

  Pointer to (libmodbus) Modbus structure.
- modBusLinkState_t **mlStat**

  connection/slave state
- uint16_t port

  Modbus IP port.

6.14.1 Detailed Description

Structure for Modbus TCP.

6.14.2 Field Documentation

6.14.2.1 addr

```
char addr[21]
```

Modbus IP (4) address as string.

for server: 0.0.0.0 (accept from all)

6.14.2.2 port

```
uint16_t port
```

Modbus IP port.

default value: 1502; Hint: 502 will need sudo for the server.

The documentation for this struct was generated from the following file:

- include/weModbus.h

6.15 oneWireDevice_t Struct Reference

A structure for 1-wire devices.

```
#include <we1wire.h>
```

Data Fields

- char **contBuf** [80]
  read content buffer
- char **name** [16]
  short sensor name (8 characters max., 6..8 recommended)
- int **value**
  value in mgrdC (1/1000 °C)
- char **valueFile** [56]
  value file path
- char valueGrdC [8]
  The value as floating point string.

6.15.1 Detailed Description

A structure for 1-wire devices.

6.15.2 Field Documentation

6.15.2.1 valueGrdC

```
char valueGrdC[8]
```

The value as floating point string.

This string is equivalent to ((float)value) / 1000.0 formatted right aligned with %6.2f. The bad value will be "-99.9 ". The function getTemp() will provide this strings without floating point arithmetic nor format string parsing. value in grdC (as %6.2f string)

The documentation for this struct was generated from the following file:

- include/we1wire.h

6.16 phPckSwSet_t Struct Reference

Simple Structure for phase packet switch setting.

```
#include <sweetHome.h>
```

Data Fields

- uint8_t **offPhases**
  number of off phases
- uint8_t **onPhases**
  number of on phases (20ms periods at 50Hz)

6.16.1 Detailed Description

Simple Structure for phase packet switch setting.

The documentation for this struct was generated from the following file:

- include/sweetHome.h

6.17 sdm124regs_t Union Reference

A type for 124 registers respectively 62 values of 32 bit.

```
#include <basicTyCo.h>
```

6.17.1 Detailed Description

A type for 124 registers respectively 62 values of 32 bit.

Modbus RS484 has a very restricted maximum telegram length of 256, allowing for 252 data bytes, respectively 248 value bytes or 124 registers in say FC4 (read input registers).

The documentation for this union was generated from the following file:

- include/basicTyCo.h

6.18 sdm80regs_t Union Reference

A type for 80 registers respectively 40 values of 32 bit.

```
#include <basicTyCo.h>
```

6.18.1 Detailed Description

A type for 80 registers respectively 40 values of 32 bit.

Modbus RS484 has a very restricted maximum telegram length of 256, allowing for 252 data bytes, respectively 248 value bytes or 124 registers in say FC4 (read input registers). EASTRON smart meters restrict this further to 80 registers respectively a maximum of 40 float values, called parameters.

The documentation for this union was generated from the following file:

- include/basicTyCo.h

6.19 semCtlPar_t Union Reference

Parameter type for semctl().

```
#include <weShareMem.h>
```

6.19.1 Detailed Description

Parameter type for semctl().

The documentation for this union was generated from the following file:

- include/weShareMem.h

6.20 smdX30modbus_t Struct Reference

A structure for SMDx30 smart meters.

`#include <basicTyCo.h>`

Data Fields

- uint16_t **errorCount**
  for the application to handle recurring errors
- int **lastRetCode**
  for the application to keep last return/error value
- modBusLinkState_t **linkState**
  state of the (slave's) communication link
- char **name** [10]
  short meter name (8 characters max., 6..8 recommended)
- int slave
  phase i title (max. 30; line1 e.g. or battery/heater)
- char **tiPh** [3][32]
  short phase i name (max. 5, 2 or 3 recommended; L1 e.g.)
- char **title** [32]
  meter explanation name (30 characters max.)

6.20.1 Detailed Description

A structure for SMDx30 smart meters.

RS458 communication and state related data plus one set of 40 input data (two 16 bit MOdbus registers as one float, also called "parameter" by meter's manufacturer).

6.20.2 Field Documentation

6.20.2.1 slave

`int slave`

phase i title (max. 30; line1 e.g. or battery/heater)

Modbus slave number 1..247; 0: all undefined

The documentation for this struct was generated from the following file:

- include/basicTyCo.h

6.21 state_t Struct Reference

The structure for state machines.

`#include <weStateM.h>`

Data Fields

- union {
  };

  This state machine's function to leave it.
- union {
  };

  This state machine's trigger respectively check function.
- union {
    float **controlF**
      the analogue / float control value (some state types)
    uint32_t **controlV**
      the integer control value (most state types)
    char **controlVS** [6]
      the state change cause
  };

  the control (parameter) value lead to current state
- union {
    float **instanceF** [4]
      instance's float values (4 ∗ 4 byte)
  };

  float and byte values to the state machines disposal.
- union {
    enterState_t const doEnter
      This state machine's function to enter it.
    enterStateF_t const **doEnterF**
      Float control substitute or addendum to state_t.doEnter.
    enterStateS_t const **doEnterS**
      Character control substitute or addendum to state_t.doEnter.
  };

  Just one enter function (type).
- genStateText_t doGenStateText

  This state machine's function for status text generation.
- uint32_t endTime

  The end time of a state or sub-state.
- char **infoTxt** [36]

  state infotext (provided by application SW)
- char const name [22]

State machine name.

- uint8_t offChainLen

  Length of OFF chain.
- uint8_t onChainLen

  Length of ON chain.
- onStateChange_t const onStateChange

  This state machine's callback function for state changes.
- uint32_t **realSecOff**

  s time stamp of last exit transition
- uint32_t **realSecOn**

  s time stamp of last enter transition
- uint8_t status

  Status number.
- uint8_t subStatus

  Addendum to status for more complicated state machines.
- float threshBadLo

  The bad Hi limit.
- float **threshCritLo**

  The critical Lo limit.
- float threshOFF

  The lower or OFF threshold.
- float threshON

  The higher or ON threshold.
- uint8_t const typ

  State machine type.

6.21.1 Detailed Description

The structure for state machines.

6.21.2 Field Documentation

6.21.2.1 name

```
char const name[22]
```

State machine name.

The names must be unique within one application and best also within a site as it usually will appear in logs as the only identifier of a state machine.

The maximum length is 20. It is recommended to have all state names the same length (e.g. 12) in an application if feasible. This gives better readability of state machine logs (see logStateText()).

6.21.2.2 typ

```
uint8_t const typ
```

State machine type.

As of Revision 193 the following types are defined and recognised by genStateText() respectively logStateText(): /code 0xAD: Timer, seconds resolution, UTC stamp 0xDB: switch de-Bounce 0xFF: Float value hysteresis 0x5B: 5 band check ...badLO | critLo | OK | critHi | badHi... 0xFC: sequential Control (SFC) /endcode

6.21.2.3 onChainLen

```
uint8_t onChainLen
```

Length of ON chain.

With SFCs this (or greater) substatus in ON chain will lead to stable ON (1:0). This value must be set at SFC construction and should not be changed hence on.
With switch de-bounce it is the up-counter level to recognise ON.
The semantic with other state machines is as yet undefined.

6.21.2.4 offChainLen

```
uint8_t offChainLen
```

Length of OFF chain.

With SFCs this (or greater) substatus in OFF chain will lead to stable OFF (0:0). This value must be set at SFC construction and should not be changed hence on.
With switch de-bounce it is the down-counter level to recognise OFF.
The semantic with other state machines is as yet undefined.

6.21.2.5 doEnter

```
enterState_t const doEnter
```

This state machine's function to enter it.

The fitting function to enter this state, respectively to start its transition from OFF to ON might be recorded in and then called via this function pointer.

6.21.2.6

```
union { ... } @10
```

This state machine's function to leave it.

The fitting function to leave this state, respectively to start its transition from ON to OFF might be recorded in and then called via this function pointer.

6.21.2.7

```
union { ... } @12
```

This state machine's trigger respectively check function.

The fitting function to check if a change of state_t.status or state_t.subStatus is due might be recorded in and then called via this function pointer.

6.21.2.8 onStateChange

```
onStateChange_t const onStateChange
```

This state machine's callback function for state changes.

The fitting function to be called on a real change of state_t.status and depending on type also on changes of state_t.subStatus must be recorded in this function pointer. In contrast to the other function pointers in this state_t structure, this pointer should not be NULL.

6.21.2.9 doGenStateText

```
genStateText_t doGenStateText
```

This state machine's function for status text generation.

The default setting will be genStateText().

6.21.2.10 status

```
uint8_t status
```

Status number.

State numbers together with sub state numbers (if applicable) must be unique, and best dense (0..n-1) within a concrete state machine (being in exactly one of n states). Status number 0,0 is reserved for the machine's default / reset state.

State machines with just two (big / first level) states can best be modelled having just one "singleton" state being either On or Off.

timer (AD):
1: (still) running; 0: ended

simple:
0: OFF; else: ON

SFC (FC):
0,0: OFF
0,x: chain to ON
4,x: chain to ON interrupted
1,0: ON
1,x: chain to OFF
5,x: chain to OFF interrupted;

Five band check (5B):
2: bad Hi
1: critical Hi
0: OK (and, hence, here usually ON while 2 and 6 would mean OFF)
5: critical Lo
6: bad Lo
8: unknown / reset
subStatus is used as previous state.

6.21.2.11 subStatus

```
uint8_t subStatus
```

Addendum to status for more complicated state machines.

The semantic of subStatus is machine type dependent. For SFC it is 0: stable; !=0: internal transitions. For five band checker it is just previous state.

6.21.2.12 endTime

```
uint32_t endTime
```

The end time of a state or sub-state.

For timers it's the end time of the ON state.

6.21.2.13 threshOFF

```
float threshOFF
```

The lower or OFF threshold.

For five band check (5B) this is the bad Hi limit.

6.21.2.14 threshON

```
float threshON
```

The higher or ON threshold.

For five band check (5B) this is the critical Hi limit.

6.21.2.15 threshBadLo

```
float threshBadLo
```

The bad Hi limit.

6.21.2.16

```
union { ...  } @16
```

float and byte values to the state machines disposal.

Convention: start using bytes from above [12 .. 15] and floats from below [0]. instance variables

The documentation for this struct was generated from the following file:

- include/weStateM.h

6.22 valFilVal_t Struct Reference

Smart meters' and other process values.

`#include <sweetHome.h>`

Data Fields

- float **batVolt**
  battery voltage by esp8266 module and MQTT
- uint8_t by [12]
  Relays, outputs and states.
- float **chStLimit**
  actual charging station current limit (/A every phase)
- float **fLine**
  last valid power line frequency
- float **groVoltP**
  Growatt panel (DC) voltage.
- float **growLoss**
  Growatt inverter loss negative: all values invalid.
- float **groWorkD**
  Growatt work of the current day.
- float **groWorkS**
  Growatt total inverter work.
- float **growPow**
  Growatt line power.
- float **growTemp**
  Growatt inverter temperature.
- int **hourOffs**
  actual offset in hours of local time zone (+ is east)
- uint16_t **ms**
  ms supplement to stmpS
- float **pAll**
  all power sum (all resp. both meters read OK)
- float **pBatUnl**
  battery unload power (+ unload, since 7.2020: - load)
- float **pGiveAway**
  PstdW - Pwaste (all resp. both meters read OK)
- float **phPckLimit**
  actual phase packet switch power limit (/W)
- float **phPckpower**
  actual phase packet switch power (/W)
- float **pSolar**
  solar power only, w/o bat unload
- float **pWaste**
  battery load and other waste; negative to be waste
- char **rTmTxt** [34]
  local time as text; length is 32
- uint32_t **stmpS**
  epoch time; s since 1.1.1970 (until 7.2.2106)

- uint32_t **stmpS1stStartSol**

  epoch time stamp for 1st solar power ON
- uint32_t **stmpSCurStartSol**

  time stamp for current/last solar power ON
- uint32_t **stmpSLstStopSol**

  time stamp for current/last solar power OFF
- int **tempPipe**

  Temperature dto.; (hot water pipe, forward feed)
- int **tempTankBott**

  Temperature in 1/1000 grdC; (tank bottom)
- int **tempTankTop**

  Water temperature in 1/1000 grdC; (tank top)
- float **wSumExp**

  sum of export work (all resp. both meters read OK)
- float **wSumIn**

  sum of import work (all resp. both meters read OK)

6.22.1 Detailed Description

Smart meters' and other process values.

This structure unites all (two, ANZmodSLAVES) smart meter readings as a meterVal_t array plus some common evaluation and process control values.

The four timing data will mark the the evaluation and forwarding time of the complete structure/information. For the meter read times see the equivalent data in the structure .meterVal, which will be 300 .. 600ms older for the meter's latest read (+1s for every other meter read earlier).

6.22.2 Field Documentation

6.22.2.1 by

```
uint8_t by[12]
```

Relays, outputs and states.

```
Byte 0 : relays (8..1 in bit 7..0)                Bits 7..0: P..W_S.uX
Byte 1 : power module voltage PWM: 0 = lowest; 255 = highest
Byte 2 : outputs (open drain)                     Bits 7..0: P234_xHIB
Byte 3 : error 0: OK; 1,2,3: meter 0,1,both;
Byte 4 : stateBits                                Bits 7..0: UBLK_wWGS
Byte 5 : phase packet switch PhPckSw              power: 0..200 (%)
Byte 6 : state2Bits  (PPS manual, limits, h2      Bits 7..0: M-PL_--fh
     : P: power at Limit L, L: limit L at it's max, fh: heater2 at full /
     : half power; ph: number of phases used by E-Car: 1..3 (default:3)
Byte 7 : chStBits charging station: man, ball, lim  Bits 7..0: MB-L_-nPh
```

The documentation for this struct was generated from the following file:

- include/sweetHome.h

6.23 valsSharMem_t Struct Reference

Structure for shared memory.

`#include <sweetHome.h>`

Data Fields

- uint32_t **clientCommand**

  single command from (web) client to server
- uint16_t **copInCnt**

  values copy in count; incremented by server
- valFilVal_t **valFilVal**

  the values; copied in form the server

6.23.1 Detailed Description

Structure for shared memory.

The documentation for this struct was generated from the following file:

- include/sweetHome.h

Chapter 7

File Documentation

7.1 dcf77onPi.c File Reference

This program is for using DCF77 AM receivers on a Pi.

```
#include "weGPIOd.h"
#include "weUtil.h"
#include <getopt.h>
#include "weLockWatch.h"
#include <errno.h>
#include "weDCF77.h"
```

Functions

- int main (int argc, char **argv)

  The program.

Variables

- uint32_t **lesDCFout**

  Bit mask of all outputs set by dcfOut.
- int **outScn** [ ]

  GPIO assignments, default pi4scann.
- int **outTrf** [ ]

  GPIO assignments, default piTrafficShield.
- char const **outTxt** [ ][8]

  Names for outputs: LEDs and control.

7.1.1 Detailed Description

This program is for using DCF77 AM receivers on a Pi.

```
Copyright (c)    2021     Albrecht Weinert
weinert-automation.de     a-weinert.de

  /         /      /\
 /         /___   /  \       |
 \        /____\ /____\ |  _|__
  \  /\  / \    /      \|   |
   \/  \/   \__/        \__/|_


Revision history
Rev. 240 10.04.2021
Rev. 236 01.02.2021 : new  (testOnPi fork)
   .
```

Purpose

This program handles the modulation signal of an AM (amplitude modulation) DCF77 receiver. DCF77 is the German long wave transmitter to broadcast the official/legal time.
Up to Revision 239++ April 2021 it is merely for testing AM receivers by logging every signal and timing.

By program options any GPIO can be chosen for the AM signal, as well as the signal polarity, filter modes and much more. The reception can be observed by up to 6 output signals respectively LEDs. The DCF77 signal is decoded and can be logged.

Options are:

```
--useWD --noWD      use the watchdog / do not use it (default)
--useLock --noLock  use the IO singleton lock / do not use it (default)
--GPIO    -G   GPIO number for the AM signal input
--pin     -p   IO connector pin number for the AM signal input
--invert --noInvert invert standard polarity / do not (default)
--pullnone --pullup --pulldown  pull resistor no / up (default) / down
--pullkeep           keep input's pull resistor setting as is
--glitch  -g         set glitch filter time in &mu;s
```

Use –help to see the actual list.

GPIO usage

The program may use (test) every GPIO by command line parameter.

Timing

Data for every modulation period are put asynchronously in a FIFO.

The program has a cyclic process control (in SPS manner, by weUtil.h weUtil.c etc). We use

```
a) a 10 ms cycle to sample the FiFo and do all other work, like
   decoding, logging etc.
b) so far no other cycle
```

Library usage

The program uses some standard libraries plus own libraries in weRasp/..c and include/..h, namely weGPIOd, sysUtil etc..

Prerequisites

As the program may use GPIO via pigpiod and the watchdog it requires them both available, e.g. by a (sudo crontab) cronjob

```
@reboot  /usr/local/bin/pigpiod  -s 10
@reboot  sleep 5 && chmod a+=rw /dev/watchdog
@reboot  chmod a+=rw /dev/hidraw0


Compile, build, load
```

cross-compile by:

```
make TARGET=pi4scann PROGRAM=dcf77onPi clean all
```

program by:

```
make PROGRAM=dcf77onPi TARGET=pi4you FTPuser=pi:piPi progapp
```

7.1.2 Function Documentation

7.1.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

The program.

Run by: dcf77onPi [options

For options see: dcf77onPi –help memcpy(dfc77ringBrecPer[readInd].sysClk, "hh:mm:ss.mil\0", 13); // init as hh↩
:mm:ss

7.2 getLocalWeatherData.c File Reference

A console program to fetch local weather data.

```
#include "sysBasic.h"
#include "we1wire.h"
#include <sys/file.h>
#include <signal.h>
#include <errno.h>
#include <getopt.h>
#include "weUtil.h"
#include "sweetHome.h"
#include "sweetHomeLocal.h"
#include <curl/curl.h>
#include <dirent.h>
```

Data Structures

- struct memAlloc_t

  Structure for allocated, re-allocatable memory.

Variables

- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.

7.2.1 Detailed Description

A console program to fetch local weather data.

```
Copyright  (c)  2018   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /          /      /\
   /          /___   /  \       |
   \         /____\ /____\ |  _|__
    \  /\  / \     /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 236 2.02.2021
Rev. 144 12.06.2018 : new; just for testing algorithms and data acquisition
Rev. 147 16.06.2018 : time handling in library functions improved
Rev. 155 27.06.2018 : time handling debugged
Rev. 164 11.07.2018 : sunset/sunrise location params; 1-wire search
Rev. 168 21.07.2018 : 1-wire temperature sensor handling enhanced
```

Program functions

This program gets one or two JSON pages from open weather map and parses them for sunset sunrise (this day, this location) and cloudiness (dto. forecast from now.

The current and next day sunrise and sunset are calculated approximately by a cosine algorithm using integer lookup tables (no float arithmetic).

Additionally it determines the 1-wire sensors present and reveals their real location in the file system.

Library usage

The program uses the curl library to fetch data from internet.

Local build and compile on a Raspberry: g++ getLocalWeaterData.c -o getLocalWeatherData -lcurl

Cross-compile by:

```
    arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
       -DMCU=BCM2837  -DTARGET=raspi61  -I./include
            -c -o getLocalWeatherData.o getLocalWeatherData.c
and so on ...
    arm-linux-gnueabihf-gcc -I. -DPLATFORM=raspberry_03 -DMCU=BCM2837
       -DTARGET=raspi61  -I./include   getLocalWeatherData.o
          weRasp/weUtil.o   --output getLocalWeatherData.elf
              -Wl,-Map=getLocalWeatherData.map,--cref
                  -lcurl
```

or by:

```
make PROGRAM=getLocalWeatherData TARGET=raspi61 clean all
```

program by:

```
make PROGRAM=getLocalWeatherData TARGET=meterPi FTPuser=pi:piSecret progapp
```

7.2.2 Variable Documentation

7.2.2.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.2.2.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.2.2.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.3 growattLink.c File Reference

Communication with a Growatt inverter.

```
#include "arch/config.h"
#include <stdint.h>
#include "mqttHome.h"
#include "growattHome.h"
#include "weUtil.h"
#include "weModbus.h"
#include "weStateM.h"
#include "weLockWatch.h"
#include "weShareMem.h"
#include <errno.h>
#include <sys/sem.h>
#include <getopt.h>
#include "weGPIOd.h"
```

Macros

- #define GNLED

  Green LED, PIN13.
- #define RDLED

  Red LED, PIN11.
- #define YELED

  Yellow LED, PIN22.

Functions

- int main (int argc, char **argv)

  The program.
- void **mqttClean** ()

  End as MQTT client.
- int mqttInit ()

  Initialise as MQTT client.
- void * processIOthread (void *args)

  The task of controlling process IO.
- void * rs232ModThread (void *args)

Variables

- char clientId [38]

  MQTT client ID.
- modBvals_t **modBvals**

  modbus respectively process values
- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.
- char subTopStPlg01 [14]

  State sub topic of S20 plug Number 01 to 09.

7.3.1 Detailed Description

Communication with a Growatt inverter.

The program's main task is to communicate with (currently) one Growatt PV (photovoltaics) inverter via its RS232 Modbus interface. Data got on a regular basis from the input and holding registers are made available to other programs, like e.g. the GCI program growattRead.c, by shared memory and by MQTT. Locking and signalling for the shared memory is done by a semaphore set. This is the same approach as in hometersControl.c. The MQTT message has the form: Up: 119.6V P: 65.0W L: 7.2W T: 26.5gdC Wd/t: 1.2 / 2316.2kWh featuring Panel DC voltage, line Power (to grid), inverter loss and temperature, day's and total work.

The program runs on a Raspberry with GUI-less (no graphics) Raspbian lite. GUI/HMI is featured as Web-Interface (Apache, C CGI) in the (W)LAN. For the Web-HMI with the GCI program growattRead.c an Apache Webserver

```
Copyright (c) 2019 Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /      \|   |
     \/  \/   \__/        \__/|_
```

Revision history

```
Rev. 247 5.05.2023
Rev. 213 13.08.2019 : new; derived from hometersControl.c
Rev. 216 31.08.2019 : first operable with Modbus outages (hopefuly)
Rev. 217 07.09.2019 : Growatt MQTT first steps
Rev. 219 06.11.2019 : Growatt MQTT all important values in one message
```

Communication

This program uses serial communication with a RS232 converter to handle the Growatt inverter's Modbus link.

MQTT protocol via Ethernet or WLAN is used to publish essential inverter values and data. The MQTT broker or host defaults to 192.168.178.87, the port is 1883.

HTTP is used for GUI (via Apache). HTTPS is not used as all LAN communication is in guarded private home or laboratory networks.

Common memory is used to communicate with other programs. At present this is a a small CGI program handling the AJAX link to the Apache based Web interface.

A backlink via HTTP parameter and shared memory and/or MQTT to control the inverter (via holding registers) is not implemented but may be in the future.

GPIO usage

This communication program currently uses no GPIO pins but includes all libraries etc. to do so.
As Test it uses three pins as output assuming LEDs connected to as Hi=On

```
Pi 1  / Pi 3       Pin
GPIO17/ 17 : red    11
GPIO21/ 27 : green  13
GPIO25     : yellow 22
```

Modbus usage

This program acts as client for one Growatt inverter 1500P "GEAD1018" with firmware G.2.0 and Modbus protocol version 3.01. It is a single line phase, single PV array and zero battery type.

Communication setting will be 'RTU / RS232 9600, none'.

Timing

The program has a cyclic process control (in SPS manner). We have /code a) a 100 ms cycle for process control, process I/O, SFCs etc. b) an 1s cycle for Modbus communication, timer handling and some SFCs /code

Server functions

Shared memory and a set of three semaphores is provided to share current values with other (C) programs as well as (in future) for receiving command and status information. This interface is also used to provide web interfaces in a flexible way.

Client functions

The program acts as (see above) as Modbus client. And it acts as MQTT publisher currently using a mosquitto broker on another Raspberry in the same private (W)LAN.

Library usage

The program uses the standard libraries pthread, pigpiod_if2, modbus, shm, sem and mosquitto. The own libraries in weRasp/..c and include/..h, namely weModbus, weGPIOd, sysUtil, weShareMem, weCGIajax etc. are compiled and linked in.

cross-compile by:

```
make PROGRAM=growattLink TARGET=growPi clean all
```

program respectively transfer to target machine by:

```
make PROGRAM=growattLink TARGET=growPi FTPuser=pi:piPas progapp
```

The building of this application is governed by the make include makeProg_growattLink_settings.mk:

```
# A makefile include for raspberry projects
# program include for one program
# Copyright  2018  Albrecht Weinert    < a-weinert.de >
MAKE_INCLUDE_PROGRAM  = growattLink
```

```
MAKE_PROGRAM_LAST_CHANGE = '$Date: 2021-02-02 18:11:02 +0100 (Di, 02 Feb 2021) $ '
MAKE_PROGRAM_REVISION = '$Revision: 236 $ '
ifndef COPYRIGHT_YEAR
$(error includefile $(MAKE_INCLUDE_PROGRAM) used directly.)
endif
# PROGRAM or MAIN_F might have been given in wrong case (at least on Windows)
override MAIN_F = $(MAKE_INCLUDE_PROGRAM)
# makeProg_hometersControl_settings.mk
# An optional short multiline description of this program's
# specifica.
# May be empty. Do not change the three lines define endef and export
define PROG_DES_TEXT
Program growattLink
This program handles the Modbus communication wit one Growatt PV inverter.
As a server it provides those date via shared memory and by MQTT.
The program should be run (auto start) as server with no direct UI.
endef
export PROG_DES_TEXT
extraLDFLAGS = -lrt -pthread -lmodbus -lpigpiod_if2 -lmosquitto
extraSOURCES = weRasp/sysBasic.c weRasp/weUtil.c weRasp/weModbus.c
extraSOURCES += weRasp/weShareMem.c weRasp/weStateM.c
extraSOURCES += weRasp/weGPIOd.c weRasp/weLockWatch.c
```

7.3.2 Macro Definition Documentation

7.3.2.1 RDLED

```
#define RDLED
```

Red LED, PIN11.

On in steps 0 ∗) and 5;
Off in steps 3 and 8 of the 100ms cycle.
Note ∗): will be done by Modbus start receive (success) in 1s cycle.

7.3.2.2 GNLED

```
#define GNLED
```

Green LED, PIN13.

On in steps 2 and 7;
Off in steps 5 and 0 of the 100ms cycle.

7.3.2.3 YELED

```
#define YELED
```

Yellow LED, PIN22.

On in step 9;
Off in step 4 ∗) of the 100ms cycle.
Note ∗): will be done by Modbus end of receive (success) in 1s cycle.

7.3.3 Function Documentation

7.3.3.1 mqttInit()

```
int mqttInit ( )
```

Initialise as MQTT client.

On success only: subscribe, loop and publish.

Returns

0: success the common mosq is set and usable; else: errno

7.3.3.2 processIOthread()

```
void * processIOthread (
            void * args )
```

The task of controlling process IO.

Except for the Modbus communication, this thread controls all processIO: at present this is just three LEDs.

It is a 100 ms cycle thread.
The cyclic 100 ms task then distinguishes 10 steps within a second. This ten (0..9) steps serve for the synchronisation with the other thread and act as state machine framework.

7.3.3.3 rs232ModThread()

```
void * rs232ModThread (
            void * args )
```

out char formIfix[60] = // form P 2 l7 Wd 16 l4 T 27 l5 Up 40 l 6 0123456789x123456789v123456789t123456789q123456789c1 2 3 4 "P: 1299.4W, Wd: 12.4kWh, T: 29.7gdC, Up: 114.3V\0\0\0"; formFixed16(formIfix + 2, 7, // form P 2 l7 mod↩ Bvals.imp0Regs.regs[12], 1); // lo line power / .1W formFixed16(formIfix + 16, 4, modBvals.imp0Regs.regs[27], 1); // lo work day / .1 kWh formFixed16(formIfix + 27, 5, modBvals.imp0Regs.regs[32], 1); // Temp / .grdC form↩ Fixed16(formIfix + 40, 6, modBvals.imp0Regs.regs[3], 1); // Upanel / .1V = out

7.3.3.4 main()

```
int main (
            int argc,
            char ** argv )
```

The program.

run by: growattLink [options
For options see longOptions and :: optHlpTxt.

Parameters

argc	number of parameters + 1
argv	optional options

xxxxx for(;commonRun;) { // use main thread for async tasks like MQTT pthread_yield(); break; // no asyn tasks yet if (!commonRun) break; pthread_yield(); if (!commonRun) break; if (!commonRun) break; pthread_yield(); if (!commonRun) break; } // commonRun async tasks xxxx

7.3.4 Variable Documentation

7.3.4.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.3.4.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.3.4.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.3.4.4 subTopStPlg01

```
char subTopStPlg01[14]
```

State sub topic of S20 plug Number 01 to 09.

It's preset as plug01/POWER for 01, but the digit at index [5] will be set before each use accordingly.

7.3.4.5 clientId

```
char clientId[38]
```

MQTT client ID.

default value: sweetHomeControl; length: 15; max. length: 36
May be changed before mqttInit().

7.4 growattRead.c File Reference

A CGI program for a Growatt PV inverter.

```
#include "sysBasic.h"
#include "arch/config.h"
#include "growattHome.h"
#include "weCGIajax.h"
#include "weShareMem.h"
#include "weModbus.h"
#include <fcntl.h>
```

Functions

- int main (int argc, char **argv)

  The program.

Variables

- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.

7.4.1 Detailed Description

A CGI program for a Growatt PV inverter.

```
Copyright (c) 2019 Albrecht Weinert
weinert-automation.de      a-weinert.de
```

Revision history

```
Rev. 247 5.05.2023
Rev. 209 16.07.2019 : new; derived from meteRead.c
Rev. 210 28.07.2019 : JSON syntax  (https://jsonformatter.org/json-parser)
Rev. 211 01.08.2019 : P U I f like meteRead instead of U I P f
Rev. 212 03.08.2019 : read holding registers on request
Rev. 215 27.08.2019 : from direct Modbus read to shared memory
```

Client functions

This program gets values from growattLink via shared memory synchronised with a set of semaphores:
sem # 0: exclusive lock of shared memory (for the shortest time possible !)
sem # 1: signal from from hometersControl to other program
sem # 2: signal from from hometersControl to this program (growattRead) This program gets values from a Growatt PV inverter via a serial Modbus link.

In future, under the same semaphore (#0) lock, it may set command codes via the shared memory according to the query string. A query will usually be part of the request when buttons were pressed in the web page.

GCI (common gateway interface) server functions

This program's output (i.e. AJAX answer) are physical readings, status values and times as JSON object, leaving selection of values and their display to the HTML page respectively its Javascript code.

In the JSON object's text delivered all numerical values (int, float) are put as strings in appropriate form and precision. The rationale is to avoid auto parsing of data, all or most of which will probably be formatted (back to text) to be put in the web page. And, alas, formatting is not Javascript's strong point, while parsing a string to a number would not be a problem, if a number is needed.

This beta version is fixed to: 9600 baud slave 1 and 1 inverter with 1 panel, 0 batteries and 1 AC-power line. All read values are put in one "reads:"[] array. This will have to be more structured (c.f. meteRead.c) in future.

GCI back-link – command by query

Not implemented yet.

Library usage

The program uses the library (lib...): pthread modbus

cross-compile by:

```
make PROGRAM=growattRead TARGET=growPi clean all
```

program by:

```
make PROGRAM=growattRead TARGET=growPi FTPuser=pi:piSecret progapp
```

7.4.2 Function Documentation

7.4.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

The program.

Run by (Apache) web server as (cgi) script with optional (query string) parameters.

7.4.3 Variable Documentation

7.4.3.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.4.3.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.4.3.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.5 growattReadSimple.c File Reference

A CGI program for a Growatt PV inverter.

```
#include "sysBasic.h"
#include "arch/config.h"
#include "growattHome.h"
#include "weCGIajax.h"
#include "weModbus.h"
#include <fcntl.h>
```

Functions

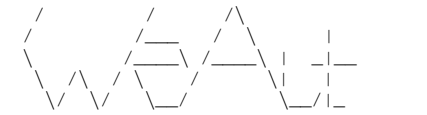
- int main (int argc, char **argv)
  The program.

Variables

- char const prgNamPure []
  The pure program name.
- char const prgSVNdat []
  The complete SVN date string.
- char const prgSVNrev []
  The complete SVN revision string.

7.5.1 Detailed Description

A CGI program for a Growatt PV inverter.



Revision history

```
Rev. 236 2.02.2021
Rev. 209 16.07.2019 : new; derived from meteRead.c
Rev. 210 28.07.2019 : JSON syntax  (https://jsonformatter.org/json-parser)
Rev. 211 01.08.2019 : P U I f like meteRead instead of U I P f
Rev. 212 03.08.2019 : read holding registers on request
Rev. 216 31.08.2019 : changes in modbus lib
```

This program gets values from a Growatt PV inverter via a serial Modbus link.

GCI server functions

This program's output (i.e. AJAX answer) are physical readings, status values and times as JSON object, leaving selection of values and their display to the HTML page respectively its Javascript code.

In the JSON object's text delivered all numerical values (int, float) are put as strings in appropriate form and precision. The rationale is to avoid auto parsing of data, all or most of which will probably be formatted (back to text) to be put in the web page. And, alas, formatting is not Javascript's strong point, while parsing a string to a number would not be a problem, if a number is needed.

This beta version is fixed to: 9600 baud slave 1 and 1 inverter with 1 panel, 0 batteries and 1 AC-power line. All read values are put in one "reads:"[] array. This will have to be more structured (c.f. meteRead.c) in future.

GCI back-link – command by query

Not implemented yet.

Library usage

The program uses the library (lib...): pthread modbus

cross-compile by:

```
make PROGRAM=growattReadSimple TARGET=growPi clean all
```

program by:

```
make PROGRAM=growattReadSimple TARGET=growPi FTPuser=pi:piSecret progapp
```

7.5.2 Function Documentation

7.5.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

The program.

Run by (Apache) web server as (cgi) script with optional (query string) parameters.

7.5.3 Variable Documentation

7.5.3.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.5.3.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.5.3.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.6 homeDoorPhone.c File Reference

Process control program for door bells IP phones etc.

```
#include "arch/config.h"
#include "homeDoor.h"
#include "weUtil.h"
#include "weStateM.h"
#include <getopt.h>
#include "weLockWatch.h"
```

Functions

- void bellsButChg (uint8_t status)

  Reaction to bell button presses.
- void doorUnlckChg (state_t *const me)

  Door opener activation state change.
- int main (int argc, char **argv)

  The program.
- void mlBellButChg (state_t *const me)

  Middle and lower bell state change.
- void mlBellTimChg (state_t *const me)

  Middle and lower bell was rung timer state change.
- void * processIOthread (void *args)

  The task of controlling process IO.
- void resOpCoupChg (state_t *const me)

  Reserve optocoupler state change.
- void upBellButChg (state_t *const me)

  Upper bell state change.
- void upBellTimChg (state_t *const me)

  Upper bell was rung timer state change.

Variables

- char const *const astDoor

  Ring the door bell phones (fritzbox Sammelnummer).
- state_t mlBellTimer

  Middle and lower bell was rung timer.
- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.
- state_t upBellTimer

  Upper bell was rung timer.

7.6.1 Detailed Description

Process control program for door bells IP phones etc.

The program's tasks is to monitor door bells and a door opener and to control at least one IP phone via an Asterisk server on the same Pi. It will log door bell events and signal door bell ringing by calling other phones (via Asterisk and a Fritz.box).

The program runs on a Raspberry Pi 4 with head-less (no graphics) Raspbian lite hosting an Asterisk telephone server and also an Apache web server. A Pi 3 may very well suffice, also.

```
Copyright  (c)  2020   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /      \|   |
     \/  \/   \__/        \__/|_
```

Revision history

```
Rev. 244 28.09.2022
Rev. 223 18.06.2020 : new
Rev. 224 26.06.2020 : door bell rung timers and LEDs
Rev. 228 20.07.2020 : 50 Hz have wave optocoupler input filter enhanced
Rev. 230 31.07.2020 : program data handling partly to weRasp\sysBasic
Rev. 239 08.03.2021 : log system() file handling errors
Rev. 243 09.09.2022 : typo
```

Communication

HTTP (via Apache) is used for offering the log files and other information. In future a web GUI as with the other process control applications may be implemented. ∗) HTTPS is not used as all LAN communication is in guarded private home or laboratory networks. Hence, the Pi can't get a Let's Encrypt certificate in the usual way. And self signed ones aren't worth the toils in a very small user community.

A VoIP phone is used to signal events to other telephones. This phone is emulated by the Asterisk server in the same Pi and registered in the site's or home's telephone system, like fritz.box.

Note ∗): Then common memory is used to communicate with other programs, like a small CGI program handling the AJAX link to the Apache based Web interface.

GPIO usage

GPIO pins are used to control one relay, 6 LEDs, and one beeper. Additionally two timing and sampling signals are output for logic analysers, oscilloscopes or other tools. Four inputs via optocoupler capture door opener and bells "seeing" only one half wave of the 50 Hz signal ∗∗). Those inputs are sampled exactly 5 times per (20ms) period and filtered accordingly.

Note ∗∗): In Germany bells and openers are operated by secure low AC voltage, about 8 to 16V, 50Hz. (DC would have simplified things here.)

Timing

The program has a cyclic process control (in SPS manner, by weUtil.h weUtil.c etc). We have

```
a) a 1 ms cycle used to sample inputs every 4ms
   and some timing signals and, partly, LEDs
b) a 100 ms cycle for all other process control,I/O, SFCs,
   timers etc.
```

Server functions

This program handles and fills error and log files to record events. Those can be accessed and viewed via http. Door bell rings are forwarded to a set of phones via Asterisk (and fritz.box).

Library usage

The program uses some standard libraries plus own libraries in weRasp/..c and include/..h, namely weGPIOd, sysUtil etc..

cross-compile by:

```
make PROGRAM=homeDoorPhone TARGET=pi4Ast clean all
```

program by:

```
make PROGRAM=homeDoorPhone TARGET=lePi FTPuser=pi:piPa progapp
```

7.6.2 Function Documentation

7.6.2.1 bellsButChg()

```
void bellsButChg (
            uint8_t status )
```

Reaction to bell button presses.

This function organises the reaction to the OR of all (two as of 7'2020) bell button presses. This is a) a beeper and b) make the telephone(s) in question ring.

Parameters

status	the calling bell's changed button state

7.6.2.2 upBellTimChg()

```
void upBellTimChg (
            state_t *const me )
```

Upper bell was rung timer state change.

It turns the left red LED ON when rung respectively OFF after 4 hours or when the door opener was activated.

Parameters

me	pointer to the pump timer

7.6.2.3 upBellButChg()

```
void upBellButChg (
            state_t *const me )
```

Upper bell state change.

This function controls the respective LED, controls the ORed bells state and triggers the "upper bell was rung" reminder timer.

7.6.2.4 mlBellTimChg()

```
void mlBellTimChg (
            state_t *const me )
```

Middle and lower bell was rung timer state change.

It turns the left yellow red LED ON when rung respectively OFF after 4 hours or when the door opener was activated.

Parameters

me	pointer to the lower bell timer

7.6.2.5 mlBellButChg()

```
void mlBellButChg (
            state_t *const me )
```

Middle and lower bell state change.

This function controls the respective right yellow LED, controls the bells state and triggers the "middle and lower bell was rung" reminder timer.

7.6.2.6 resOpCoupChg()

```
void resOpCoupChg (
            state_t *const me )
```

Reserve optocoupler state change.

This optocoupler input is not used (as of July 2020) or powered. This function just logs the (de-bounced) On state (which should not occur).

7.6.2.7 doorUnlckChg()

```
void doorUnlckChg (
            state_t *const me )
```

Door opener activation state change.

This function controls the respective LED. and ends the "bell was rung unnoticed" timers.

7.6.2.8 processIOthread()

```
void * processIOthread (
            void * args )
```

The task of controlling process IO.

This thread controls almost all processIO: 1 relay output, 6 LEDs, four digital inputs to be filtered (half wave AC via optocouplers), one beeper.

It is the 1 ms cycle and takes every forth step to sample an input. This gives five times per 50 Hz period. 2 of three will be active (LO) when the optocoupler sees an AC signal.

7.6.2.9 main()

```
int main (
              int argc,
              char ** argv )
```

The program.

run by: homeDoorPhone [options

For options see longOptions and :: optHlpTxt.

7.6.3 Variable Documentation

7.6.3.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.6.3.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.6.3.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.6.3.4 astDoor

```
char const* const astDoor
```

Ring the door bell phones (fritzbox Sammelnummer).

fritz.box configuration (27.06.2020): Trklingel ∗∗887 Sammel-Nr Asterisk call file:
`Channel: SIP/629`

7.6.3.5 upBellTimer

```
state_t upBellTimer
```

Upper bell was rung timer.

This timer is running as reminder that the bell was rung.

7.6.3.6 mlBellTimer

```
state_t mlBellTimer
```

Middle and lower bell was rung timer.

This timer is running as reminder that the bell was rung.

7.7 hometersConsol.c File Reference

A console program to co-operate with hometersControl.

```
#include "sweetHome.h"
#include "weUtil.h"
#include "weModbus.h"
#include "weShareMem.h"
#include <errno.h>
#include <getopt.h>
```

Functions

- int main (int argc, char **argv)

  The program.

Variables

- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.

7.7.1 Detailed Description

A console program to co-operate with hometersControl.

This program gets values from hometersControl.c via shared memory and displays them on console. This is mainly for local test. The operational HMI is by (Apache 2.4) web interface via CGI and AJAX by meteRead.c.

```
Copyright  (c)  2017   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 244 28.09.2022
Rev. 73+ 29.11.2017 : new; derived from hometersControl.c
Rev.  75 30.11.2017 : first operable
Rev. 188 15.10.2018 : deprecated, use the web interface
Rev. 214 15.08.2019 : corrections in Doxygen comments, loopCnt
Rev. 229 23.07.2020 : valFil (CSV) removed
```

Client functions

This program gets values from hometersControl via shared memory synchronised by a set of semaphores: sem # 0: exclusive (short!) lock of shared memory sem # 1: signal from from hometersControl to (this) hometersConsol sem # 2: signal from from hometersControl to other (CGI etc.) programs

Library usage

The program uses the libraries (lib...): pthread, pigpiod_if2 and modbus The (weRasp/..c, include/..h) weModbus, weGPIOd and sysUtil libraries are linked in. They may be converted to and used as (.so) library, also.

cross-compile by:

```
make PROGRAM=hometersConsol TARGET=meterPi clean all
```

program respectively transfer to target machine by:

```
make PROGRAM=hometersConsol TARGET=raspi61 FTPuser=pi:secret progapp
```

The building of this application is governed by the make include makeProg_hometersConsol_settings.mk:

```
# A makefile include for raspberry projects
# program include for one program
# makeProg_hometersConsol_settings.mk
# Copyright  2017  Albrecht Weinert    < a-weinert.de >
MAKE_INCLUDE_PROGRAM  = hometersConsol
MAKE_PROGRAM_LAST_CHANGE = '$Date: 2021-02-02 18:11:02 +0100 (Di, 02 Feb 2021) $ '
MAKE_PROGRAM_REVISION = '$Revision: 236 $ '
ifndef COPYRIGHT_YEAR
```

```
$(error includefile $(MAKE_INCLUDE_PROGRAM) used directly.)
endif
# PROGRAM or MAIN_F might have been given in wrong case (at least on Windows)
override MAIN_F = $(MAKE_INCLUDE_PROGRAM)
# An optional short multiline description of this program's specifica.
# May be empty. But, do not change the three lines define endef and export.
define PROG_DES_TEXT
Program hometersConsol
This program is a accompaining client to hometersControl.
endef
export PROG_DES_TEXT
extraLDFLAGS = -lrt -pthread
extraSOURCES = weRasp/sysBasic.c weRasp/weUtil.c weRasp/weShareMem.c
extraSOURCES += weRasp/sweetHome.c
```

7.7.2 Function Documentation

7.7.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

The program.

run by: hometersConsol [options
for help: hometersConsol -h

7.7.3 Variable Documentation

7.7.3.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.7.3.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.7.3.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.8 hometersControl.c File Reference

Process control for a slightly smart home.

```
#include "arch/config.h"
#include "sweetHome2.h"
#include "weUtil.h"
#include "weModbus.h"
#include "weStateM.h"
#include "weLockWatch.h"
#include "weEcarLd.h"
#include "weShareMem.h"
#include "weAR_N4105.h"
#include <errno.h>
#include <sys/sem.h>
#include <getopt.h>
#include "sweetHomeLocal.h"
```

Functions

- void batCntBalStatChg (state_t ∗const me)

  Battery as controlled ballast SFC state changes.
- void batCntLodStatChg (state_t ∗const me)

  Battery controlled load or keep; SFC state changes.
- void batUnloadStatChg (state_t ∗const me)

  Battery unload SFC state changes.
- void hotWatPmpCntlTimChg (state_t ∗const me)

  Hot water comfort pump control timer state change function.
- void hotWatPmpSchTimChg (state_t ∗const me)

  Hot water comfort pump schedule timer state change function.
- void hotWpmpButChg (state_t ∗const me)

  Hot water comfort pump button state change.
- void initDawnDusk ()

  Adjust dawn and dusk timers.
- void initSunRiSet (int init)

  Initialise dawn, sunrise, sunset and dusk timers.
- void load250WstateChg (state_t ∗const me)

  Controlled fixed ballast load; SFC state changes.
- int main (int argc, char ∗∗argv)

  The program.
- void ∗ processIOthread (void ∗args)

  The task of controlling process IO.
- int readRetFil (void)

  Read (day start) retain file.
- void switchARN4105 (uint8_t const cutOff)

  AR-N 4105 implementation function.
- int writeRetFil (void)

  Write (day start) retain file.

Variables

- state_t **batCntBalSeq**

  Battery controlled load as ballast.
- state_t **batCntLodSeq**

  Battery loading and keeping.
- state_t **batUnloadSeq**

  Battery controlled unload via inverter.
- __time_t const hotTimSched [ ]

  Schedule times for hot water comfort pump starts.
- state_t hotWatPmpCntlTimer

  Hot water comfort pump control timer.
- state_t hotWatPmpSchTimer

  Hot water comfort pump schedule timer.
- state_t **load250Wcont**

  Controlled load SFC.
- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.

7.8.1 Detailed Description

Process control for a slightly smart home.

The program's tasks are mainly related to electrical power involving two or three smart meters (three phase 230/400V) and power modules for battery and solar panel / inverter handling. The development started on a realistic laboratory experiment (end 2017) and soon (mid 2018) spun off to a real life home control set-up with extra comfort and buffer battery functions.
The MQTT topic root "labExp/sweetHome/" as well as the "hometers" in the application were kept so far out of reverence for the very beginnings.

The program runs on a Raspberry with GUI-less (no graphics) Raspbian lite having an Apache Webserver and a MQTT broker on the same machine. GUI/HMI is featured as Web-Interface (Apache, C CGI) in the (W)LAN.

```
Copyright (c) 2019 Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /      \|   |
     \/  \/   \__/        \__/|_
```

Revision history

```
Rev. 253 9.10.2023
Rev.  60 01.11.2017 : new setup with reliable Modbus communication to smart
                      meters with neither smart protocol nor documentation
Rev.  73 29.11.2017 : shared memory, semaphores and hometersControl.h
Rev.  85 21.12.2017 : more relays, more commands, GCI via shared memory
Rev. 164 11.07.2018 : sunset/sunrise location; Hippogreiff relay
Rev. 167 20.07.2018 : Phase packet switch (PPS) automated
Rev. 190 12.02.2019 : preparation for VDE-AR-N 4105 inverter cut off
Rev. 199 02.04.2019 : battery discharge allow / disallow start parameter
Rev. 200 16.04.2019 : battery keep inhibit until low stress
Rev. 205 18.05.2019 : (low) battery voltage stress handling enhanced
Rev. 216 31.08.2019 : changes in Modbus library
Rev. 217 07.09.2019 : Growatt inverter with Modbus-Rs232 to MQTT bridge
                      added; see growattLink.c
Rev. 220 15.11.2019 : Growatt keep day's work intact. inverter day == light
Rev. 229 23.07.2020 : valFil (CSV) removed
Rev. 240 11.08.2021 : battery keep inhibit timer handling changed
Rev. 245 01.05.2023 : valFilNam removed (by //)
Rev. 251 26.07.2023 : two heater elements, web interface enh.
```

Communication

This program uses serial half duplex communication with a TTL to RS485 converter to handle smart meters (max. 10 m away) via Modbus. The RS485 module ("shield") forwards all GPIO pins covered. Growatt inverters have Modbus via RS232 (not sharable and lower range) and one of them got an extra o provide a MQTT interface; see growattLink.c.

MQTT protocol [W]LAN is used to attach more remote periphery, like a battery surveillance connected directly to and supplied by the battery clamps. Other MQTT periphery devices are of the shelf remote power relays. An extra Pi with Modbus over V.24 is supervising a Growatt inverter.
Examples for messages to parse: 'Up: 119.6V P: 65.0W L: 7.2W T: 26.5gdC Wd/t: 0.0 / 2316.2kWh' for topic 'lab↩Exp/sweetHome/grow01/meas' 'batU 0 : 12.19 V, adc:641' for topic 'labExp/sweetHome/bat/volt' '{"POWER":"ON"}' for topic 'labExp/sweetHome/plug01/RESULT' 'ON' for topic 'labExp/sweetHome/plug01/POWER' '{"POWER":"↩OFF"}' for topic 'labExp/sweetHome/plug01/RESULT' 'OFF' for topic 'labExp/sweetHome/plug01/POWER'

HTTP is used for GUI (via Apache). HTTPS is not used as all devices are in a guarded private home or laboratory network. Hence the Pi can't get a Let's Encrypt certificate and self signed ones aren't worth the while.

A 1-wire bus is used to add some analogue sensors to the ADC abstinent Raspberry. At present there are three hot water tank / pipe temperatures sensors to limit the 0..1kW ballast heater usage.

Common memory is used to communicate with other programs. At present this is a CGI program (meteRead.c) handling the AJAX link to the Apache based Web interface (meteRead.html).

GPIO usage

GPIO pins are used to control relays, LEDs and one button switch in the Pi's immediate neighbourhood. Three open drain outputs in a small extra module control slightly remote (10m) periphery via shielded cable and acts as guard to the virtually un-protected Raspberry IO pins.

The GPIO configuration definitions are found in file sweetHome2.h.

Modbus usage

This program acts as client for B+G E-Tech EASTRON smart meters. In the current (2017..23) configuration these are one SDM630-Modbus and one SDM530-Modbus three phase meter. (One phase meters like SDM230-↩ Modbus were used in early experimental lab set-ups, only). Even in the Lab, no B+G E-Tech EASTRON Modbus communicated reliably above 9.6 baud (the default). Due to this low speed and, additionally, low response times, only two meter readings per second are possible over one RS485 line.

Communication will be 'RTU / RS485 9600, none' as common denominator. See Eastron, SMD230Modbus, Smart Meter Modbus Protocol Implementation V1.2 Eastron, SDM630Modbus, Smart Meter Modbus Protocol 630 (V1.5 ?) Eastron, SDM530Modbus, Smart Meter Modbus Protocol 530 (V1.1 or V1.5) See also metersModbusClient.c for more meter details.
Due to the similar register layout, the meter types can be replaced with virtually no change in program. The SMD230, of course, can only be used if one phase (L1) is sufficient. Due to its technical inferiority it is not recommended except if space limitations on the DIN rails are hard.

This hometersControl application variant uses two three phase meters: A - slave 0: a SDM530Modbus three phase meter for the whole house. L1 L2 L3 are the three phases, each max. 80 A to supply the building under experiment. Import means power flows from public supply. Export is considered prohibited and will be inhibited by controlling suitable consumers, as water tank heater, and the buffer battery. In 2023 a car chargers power control/limit (by PWM) will be added. B - slave 3: a SDM630Modbus meter is used for two solar panel groups (A1/mJB, B3/big) and one for waste (ballast) and battery (C3/xLD). Import means power flow from panels respectively battery; export means battery charging and extra (waste/ballast) consumers including idle inverter losses.

Timing

The program has a cyclic process control (in SPS manner). As of Revision 150 we have
a) a 20 ms cycle for phase packet switching
b) a 100 ms cycle for process control, process I/O, SFCs etc.
c) an 1s cycle for Modbus ∗) communication, timer handling and some SFCs
The 100 ms cycle b) does not use the the library's (weUtil.c / .h) own generic 100 ms cycle but a by 5 sub-division of the 20 ms cycle a).
The generic cycles (a and c) are, by library implementation, own threads.
) The Eastron meter communication is dead slow by both answering delays and default baud rate. All experiment deviating from default settings failed.

Server functions

This program handles error, log, hibernation etc. files to provide values for humans and other programs.

Additionally shared memory and a set of three semaphores is provided to share current values with other (C) programs as well as for receiving command and status information. This interface is also used to provide web interfaces in a flexible way.

And the program acts as MQTT subscriber and publisher using a mosquitto broker, currently on the same Raspberry Pi. The primary MQTT purpose was to have "own" MQTT periphery. Those are remote power relays and a battery voltage monitoring ESP8266 in the private LAN – the latter due to Pi's lack of analogue input.

Library usage

The program uses the standard libraries pthread, pigpiod_if2, modbus, shm, sem and mosquitto. The own libraries in weRasp/..c and include/..h, namely weModbus, weGPIOd, sysUtil, weShareMem, weCGIajax etc. are compiled and linked in.

They could also be converted to and used as .so library. But, in the application environment given, this approach brings no advantage.

Build the program

cross-compile by:
```
make PROGRAM=hometersControl TARGET=meterPi clean all
```

program by:
```
make PROGRAM=hometersControl TARGET=meterPi FTPuser=pi:piSecret progapp
```

or due to some bugs in make use winscp and IP directly by:
```
winscp.com /script=progTransWin /parameter pi:piSecret 1.e.p.i bin hometersControl
```

7.8.2 Function Documentation

7.8.2.1 initDawnDusk()

```
void initDawnDusk ( )
```

Adjust dawn and dusk timers.

This method initialises the settings of the dawn and dusk timers in the initialising process.

7.8.2.2 readRetFil()

```
int readRetFil (
            void  )
```

Read (day start) retain file.

Returns

0 : file found and read; else: not

7.8.2.3 writeRetFil()

```
int writeRetFil (
            void  )
```

Write (day start) retain file.

Returns

0 : file opened/created and written; else not

7.8.2.4 initSunRiSet()

```
void initSunRiSet (
            int init )
```

Initialise dawn, sunrise, sunset and dusk timers.

The function is currently used only at program start start.

If the parameter init is true valFilVal .dayStrtVal values will be calculated for the current day. Otherwise the are assumed to be valid, i.e. read usually from that days start (hibernation) file. initDawnDusk() will be called in any case.

Parameters

init	true: dayStrtVal else use

7.8.2.5 hotWatPmpCntlTimChg()

```
void hotWatPmpCntlTimChg (
             state_t *const me )
```

Hot water comfort pump control timer state change function.

It turns the pump on respectively off. At turning off the next on schedule is determined and set.

Parameters

me	pointer to the pump timer

7.8.2.6 hotWpmpButChg()

```
void hotWpmpButChg (
             state_t *const me )
```

Hot water comfort pump button state change.

This function (re-) triggers the hot water comfort pump timer.

7.8.2.7 hotWatPmpSchTimChg()

```
void hotWatPmpSchTimChg (
             state_t *const me )
```

Hot water comfort pump schedule timer state change function.

It starts pump control timer.

Parameters

me	pointer to the pump timer

7.8.2.8 batUnloadStatChg()

```
void batUnloadStatChg (
```

```
            state_t *const me )
```

Battery unload SFC state changes.

This sequential state machine organises the battery unload via a step up converter module and an inverter normally used for a small panel set. It handles all switching steps and (inhibit) conditions.

Parameters

me	pointer to the battery unload SFC, never NULL

7.8.2.9 batCntBalStatChg()

```
void batCntBalStatChg (
            state_t *const me )
```

Battery as controlled ballast SFC state changes.

This sequential state machine organises the battery load to be a ballast for surplus energy.

Parameters

me	pointer to the battery unload SFC, never NULL

7.8.2.10 batCntLodStatChg()

```
void batCntLodStatChg (
            state_t *const me )
```

Battery controlled load or keep; SFC state changes.

This sequential state machine organises the battery loading and keeping it in good load state. Not to be confused with batCntBalStatChg, i.e. using battery as storage/ballast.

Parameters

me	pointer to the battery unload SFC, never NULL

7.8.2.11 load250WstateChg()

```
void load250WstateChg (
            state_t *const me )
```

Controlled fixed ballast load; SFC state changes.

This sequential state machine organises the the handling of a load of about 250 W with restricted switching rules. One example is a compressor air drier.
The rules implemented here are:
When switched ON, run at least 30 minutes before allow turning OFF.
Do not turn ON twice within 2 hours.
Do not run longer than 4 hours.

The controlled load is at relay 5 (plug 10A at the electrical switch box) and Plg1 (MQTT).

Parameters

me	pointer to the controlled load (0) SFC, never NULL

7.8.2.12 switchARN4105()

```
void switchARN4105 (
            uint8_t const cutOff )
```

AR-N 4105 implementation function.

This is the VDE-AR-N 4105 cut off function to be provided site specific by user / application software. It does the cut off respectively switch back on the generators and optionally log the event.

Parameters

cutOff	not 0: do the cut off, 0: switch generators back on

7.8.2.13 processIOthread()

```
void * processIOthread (
            void * args )
```

The task of controlling process IO.

With the exception of the Modbus / RS485 coupled smart meters, this thread controls almost all processIO: 8 relay outputs, some LEDs, one button input, one PWM signal and some open drain M-switches.

It is a 20 ms cycle thread to control line power by phase packet switching while all the rest is effectively done in a 100 ms cycle, i.e. every 5th 20ms cycle.

The cyclic 100 ms task then distinguishes 10 steps within a second and sometimes differentiates odd and even seconds by the actual Modbus slave number in the respective 1s RS4885 thread. This ten (0..9) respectively twenty steps serve the synchronisation with the other thread and act as state machine framework. valFilVal.by[7] = valFilVal.by[7] & ∼CHSphas | 1;

valFilVal.by[7] = valFilVal.by[7] | 3;

7.8.2.14 main()

```
int main (
              int argc,
              char ** argv )
```

The program.

run by: hometersControl [options

For options see longOptions and :: optHlpTxt.

7.8.3 Variable Documentation

7.8.3.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.8.3.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.8.3.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.8.3.4 hotTimSched

```
__time_t const hotTimSched[]
```

Schedule times for hot water comfort pump starts.

This is a sorted array starting with the earliest start time in seconds relative to local midnight. At least the first (earliest) entry has to be put at the end 24h (86400s) later, to find a next day entry.

Hint: The first entry itself is nevertheless not dispensible as the program may be started between midnight and that time.

Hint 2: At days with DST change the first time in that day will be by one hour wrong. But beware using this approach for other purposes without consideration.

Hint 3: This sorted array of a day's clock readings might be replaced by an array of structures to implement day of week as filter condition for a schedule time.

7.8.3.5 hotWatPmpSchTimer

```
state_t hotWatPmpSchTimer
```

Hot water comfort pump schedule timer.

This timer is running to the next time to start the comfort pump on schedule. On running out it will trigger the pump control timer to start the pump or prolong its run time accordingly. Afterwards this timer will be restarted to the next schedule point in future.

7.8.3.6 hotWatPmpCntlTimer

```
state_t hotWatPmpCntlTimer
```

Hot water comfort pump control timer.

This timer is running / active as long as the comfort pump is running. It might be triggered or prolonged by push button (hotWpmpButChg), web interface command or the pump schedule timer.

7.9 hometersDayVal.c File Reference

A program to handle some day specific settings and values.

```
#include "sweetHome.h"
#include "weUtil.h"
#include <errno.h>
#include <getopt.h>
```

Functions

- int main (int argc, char **argv)

  The program.
- int readRetFil (void)

  Read (day start) retain file.
- int writeRetFil (void)

  Write (day start) retain file.

Variables

- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.

7.9.1 Detailed Description

A program to handle some day specific settings and values.

At the present (simple start) stage this program just displays the current day's start / hibernation value file content.

More feature controlled by start parameters will be added in future.

```
Copyright  (c)  2019   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \      |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 247 5.05.2023
Rev. 151 20.06.2018 : new; just see current day values file (to start with)
Rev. 199 07.03.2019 : -t parameter, help text enhanced
Rev. 245 01.05.2023 : valFilNam removed (by //) option t corrected
```

cross-compile by:

```
   arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
      -DMCU=BCM2837  -DTARGET=raspi61  -I./include
           -c -o hometersControl.o hometersDayVal.c
   arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
       -DMCU=BCM2837 -DTARGET=raspi61  -I./include
           -c -o weRasp/weUtil.o weRasp/weUtil.c
and so on .... \code
   arm-linux-gnueabihf-gcc -I. -DPLATFORM=raspberry_03 -DMCU=BCM2837
      -DTARGET=raspi61  -I./include   hometersControl.o
         weRasp/weUtil.o weRasp/weModbus.o weRasp/weGPIOd.o
           weRasp/weShareMem.o --output hometersControl.elf
             -Wl,-Map=hometersControl.map,--cref
                 -lrt -pthread -lmodbus -lpigpiod_if2
```

or by:

```
make PROGRAM=hometersDayVal TARGET=raspi61 clean all
```

program by:

```
make PROGRAM=hometersDayVal TARGET=meterPi FTPuser=pi:piSecret progapp
```

7.9.2 Function Documentation

7.9.2.1 readRetFil()

```
int readRetFil (
            void  )
```

Read (day start) retain file.

Returns

0 : file found and read; else not

7.9.2.2 writeRetFil()

```
int writeRetFil (
            void  )
```

Write (day start) retain file.

Returns

0 : file opened/created and written; else not

7.9.2.3 main()

```
int main (
            int argc,
            char ** argv )
```

The program.

run by: hometersControl [options options so far (14.11.17): any parameter = use stdout and stderr

7.9.3 Variable Documentation

7.9.3.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.9.3.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.9.3.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.10 include/arch/config.h File Reference

Organising platform specific includes for the make process.

```
#include "arch/config_raspberry_03.h"
```

7.10.1 Detailed Description

Organising platform specific includes for the make process.

```
Copyright  (c)  2018   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /      \|   |
     \/  \/   \__/        \__/|_
```

Revision history

```
Rev. 252 2.10.2023
Rev. 150 18.06.2018 : minor, comments only
Rev. 209 22.07.2019 : work around a Doxygen bug (default Pi4)
Rev. 255 03.08.2023 : minor clarification
```

This file contains some definitions concerning hardware configuration. These settings influence the compilation and building process and can't be changed later at runtime.

7.11 include/arch/config_raspberry_00.h File Reference

Configuration settings for Raspberry Pi zero.

Macros

- #define **PIN27**

  SDA0.
- #define **PIN28**

  SCL0.
- #define stdUARTpath

  /def stdUARTpath Pi's standard UART.

7.11.1 Detailed Description

Configuration settings for Raspberry Pi zero.

This file contains some platform (type) specific definitions. These settings influence the compilation and build process. Most of those settings can not be changed later at runtime.

With respect to (process) I/O pins there is no difference to Pi3 and Pi4.

```
Copyright  (c)  2020   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /      \|   |
     \/  \/   \__/        \__/|_
```

Revision history

```
Rev. 252 2.10.2023
Rev. 234 13.11.2020 : fork of config_raspberry_04.h
Rev. 246 01.05.2023 : typos corr
```

7.11.2 Macro Definition Documentation

7.11.2.1 stdUARTpath

```
#define stdUARTpath
```

/def stdUARTpath Pi's standard UART.

It is the one on the Pins 8 (GPIO14) for Tx and 10 (GPIO15) for Rx.

7.12 include/arch/config_raspberry_01.h File Reference

Configuration settings for Raspberry Pi1.

Macros

- #define stdUARTpath

  /def stdUARTpath Pi's standard UART.

7.12.1 Detailed Description

Configuration settings for Raspberry Pi1.

This file contains some platform (type) specific definitions. These settings influence the compilation and build process. Most of those settings can not be changed later at runtime.

Since end 2017 we mostly use Pi3.

```
Copyright (c) 2019 Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___    /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \     /      \|   |
     \/  \/   \__/        \__/|_
```

Revision history

```
Rev. 252 2.10.2023
Rev. 190 12.02.2019 : minor, comments only
Rev. 209 10.07.2019 : stdUARTpath
Rev. 231 13.08.2020 : two digit PIN0x  and  GPIO2pin added
```

7.12.2 Macro Definition Documentation

7.12.2.1 stdUARTpath

```
#define stdUARTpath
```

/def stdUARTpath Pi's standard UART.

It is the one on the Pins 8 (GPIO14) for Tx and 10 (GPIO15) for Rx.

7.13 include/arch/config_raspberry_03.h File Reference

Configuration settings for Raspberry Pi3.

Macros

- #define **PIN27**

  SDA0.
- #define **PIN28**

  SCL0.
- #define stdUARTpath

  /def stdUARTpath Pi's standard UART.

7.13.1 Detailed Description

Configuration settings for Raspberry Pi3.

This file contains some platform (type) specific definitions. These settings influence the compilation and build process. Most of those settings can not be changed later at runtime.

With respect to (process) I/O pins there is no difference to Pi3b. Hence, for Pi3∗ define PLATFORM as raspberry↩ _03.

```
Copyright (c) 2019 Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 252 2.10.2023
Rev. 190 12.02.2019 : minor, comments only
Rev. 209 10.07.2019 : stdUARTpath
Rev. 231 13.08.2020 : two digit PIN0x  and  GPIO2pin added
```

7.13.2 Macro Definition Documentation

7.13.2.1 stdUARTpath

```
#define stdUARTpath
```

/def stdUARTpath Pi's standard UART.

It is the one on the Pins 8 (GPIO14) for Tx and 10 (GPIO15) for Rx.

7.14 include/arch/config_raspberry_04.h File Reference

Configuration settings for Raspberry Pi4.

Macros

- #define GPIO2pin

  GPIO [0..39] to pin number (1..40) lookup list.
- #define **PIN03**

  GPIO 02 SDA1 (GPIO 0 on Pi1)
- #define **PIN05**

  GPIO 03 SCL (GPIO 0 on Pi1)
- #define **PIN07**

  GPIO 04 GPCLK0.
- #define **PIN08**

  GPIO 14 TXDI.
- #define **PIN10**

  GPIO 15 RXDI.
- #define **PIN11**

  GPIO 17.
- #define **PIN12**

  GPIO 18.
- #define **PIN13**

  GPIO 27 (GPIO 21 on Pi1)
- #define **PIN15**

  GPIO 22.
- #define **PIN16**

  GPIO 23.
- #define **PIN18**

  GPIO 14.
- #define **PIN19**

  GPIO 10 SPI.MOSI | S.
- #define **PIN21**

  GPIO 09 MSPI.ISO | P.
- #define **PIN22**

  GPIO 25.
- #define **PIN23**

  GPIO 11 SPI.SCLK | I.
- #define **PIN24**

  GPIO 08 SPI.CE0 / .
- #define **PIN26**

  GPIO 07 SPI.CE1 / .
- #define **PIN27**

  SDA0.
- #define **PIN28**

  SCL0.
- #define **PIN29**

  GPIO 05.
- #define PIN2gpio

  Pin number [0..43] to GPIO number lookup list.
- #define **PIN31**

  GPIO 06.
- #define **PIN32**

  GPIO 12.
- #define **PIN33**

GPIO 13.

- #define **PIN35**

  GPIO 19.

- #define **PIN36**

  GPIO 16.

- #define **PIN37**

  GPIO 26.

- #define **PIN38**

  GPIO 20.

- #define **PIN40**

  GPIO 21.

- #define stdUARTpath

  /def stdUARTpath Pi's standard UART.

7.14.1 Detailed Description

Configuration settings for Raspberry Pi4.

This file contains some platform (type) specific definitions. These settings influence the compilation and build process. Most of those settings can not be changed later at runtime.

With respect to I/O pins there is no difference between pi4∗ and Pi3∗.

```
Copyright (c) 2020 Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /      \|   |
     \/  \/   \__/        \__/|_
```

Revision history

```
Rev. 252 2.10.2023
Rev. 223 18.06.2020 : fork of config_raspberry_03.h
Rev. 231 13.08.2020 : two digit PIN0x  and  GPIO2pin added
Rev. 237 01.03.2021 : included in Doxygen documentation
```

7.14.2 Macro Definition Documentation

7.14.2.1 GPIO2pin

```
#define GPIO2pin
```

GPIO [0..39] to pin number (1..40) lookup list.

0 means has no pin on the 40 pin connector; see also gpio2pin

7.14.2.2 PIN2gpio

```
#define PIN2gpio
```

Pin number [0..43] to GPIO number lookup list.

0..56: GPIO number; 95: 5V pin; 93: 3.3V; 90: Earth 0V;
99 means not existent. N.b.: [1..40] are: valid pin numbers, only.

See also

pin2gpio

7.14.2.3 stdUARTpath

```
#define stdUARTpath
```

/def stdUARTpath Pi's standard UART.

It is the one on the Pins 8 (GPIO14) for Tx and 10 (GPIO15) for Rx.

7.15 include/basicTyCo.h File Reference

Basic types and constants.

```
#include <stdint.h>
```

Data Structures

- union dualReg_t
  A 32 bit union.
- union sdm124regs_t
  A type for 124 registers respectively 62 values of 32 bit.
- union sdm80regs_t
  A type for 80 registers respectively 40 values of 32 bit.
- struct smdX30modbus_t
  A structure for SMDx30 smart meters.

Macros

- #define clearArray(a)

  Clear an array.
- #define **DAYs**

  seconds in days w/o DST switch or leap seconds
- #define ERAend

  A point in time far away.
- #define FALSE

  false off aus arrêt stop halt.
- #define **FOURYEARS**

  days in four years (3 ∗ YEAR + LEAPYEAR)
- #define **HOURs**

  seconds in hours
- #define **LEAPYEAR**

  days in leap year
- #define memBarrier()

  Memory barrier.
- #define MILLIARD

  MILLIARD 1/nano = Giga = 10∗∗9.
- #define MILLION

  Million 1/ = 1/micro = Mega = 10∗∗6.
- #define **MINUTEs**

  seconds in minutes
- #define OFF

  false Off Aus arrêt stop halt.
- #define ON

  true On An marche go.
- #define PLATFlittlE

  The target platform is little endian.
- #define TRUE

  true on an marche go.
- #define YEAR

  Days in normal year.

Typedefs

- typedef enum modBusLinkState_t modBusLinkState_t

  A set of possible states of a Modbus link.

Enumerations

- enum modBusLinkState_t {
  ML_OFF , ML_ON , ML_IDLE , ML_INITED ,
  ML_REQSEND , ML_RESPREC , ML_LISTEN , ML_REQREC ,
  ML_RESPOND , ML_ERR_ANY , ML_ERR_INIT , ML_ERR_REQ ,
  ML_ERR_RESP }

  A set of possible states of a Modbus link.

7.15.1 Detailed Description

Basic types and constants.

This file contains some basic type definitions and values, i.e. macro constants. It does not contain function definitions and, hence, has no implementing .c file.

```
Copyright  (c)  2018   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /          /      /\
   /          /___   /  \       |
   \         /____\ /____\ |  _|__
    \  /\  / \     /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 247 5.05.2023
Rev. 152 21.06.2018 : new, excerpted from sysBasic.h V.151
Rev. 182 12.08.2018 : types reduced
Rev. 190 12.02.2019 : minor, comments only
Rev. 209 13.07.2019 : include stdint.h only (string.h no more)
Rev. 215 26.08.2019 : type sdm124regs_t added
```

7.15.2 Macro Definition Documentation

7.15.2.1 PLATFlittlE

```
#define PLATFlittlE
```

The target platform is little endian.

values: 0 : no (known at compile time); 1 : yes (dto.); littleEndian() : compute at runtime as no commonly used target information macros are available

7.15.2.2 ON

```
#define ON
```

true On An marche go.

value: 1

7.15.2.3 OFF

```
#define OFF
```

false Off Aus arrêt stop halt.

value: 0

7.15.2.4 TRUE

```
#define TRUE
```

true on an marche go.

value: 1

7.15.2.5 FALSE

```
#define FALSE
```

false off aus arrêt stop halt.

value: 0

7.15.2.6 MILLIARD

```
#define MILLIARD
```

MILLIARD 1/nano = Giga = 10∗∗9.

The constant Milliard. (Amercians, wrongly, call that Billion.)

value: 1000000000

7.15.2.7 MILLION

```
#define MILLION
```

Million 1/ = 1/micro = Mega = 10∗∗6.

value: 1000000

7.15.2.8 YEAR

```
#define YEAR
```

Days in normal year.

value: 365

7.15.2.9 ERAend

```
#define ERAend
```

A point in time far away.

This is 2.2.2106 in Unix seconds and very near the end of the unsigned 32 bit era. In the sense of small embedded control applications we consider this (for tasks timers etc.) as beyond end of life and, hence, infinity.
value: 4294512000U

7.15.2.10 clearArray

```
#define clearArray(
              a )
```

Clear an array.

This is to set a real fixed size array to zero (0). Using this instead of looping or own optimising gets better code on almost all gcc compilers. Note: "Real" array means an array declared and defined with a fixed length; and no malloced pointer.

Parameters

a	the array to be set all 0 (not NULL, fixed length array)

7.15.2.11 memBarrier

```
#define memBarrier( )
```

Memory barrier.

This macro is an ARM memory fence instruction insuring cache updates.

Memory mapped IO, as used in Raspberries' ARM s BCM2835, BCM2835 and BCM2836 is quite problematic. As the semantics of memory is (non regarding caches and multiprocessing) extremely simple both compilers and processors may optimise memory access by all kinds of re-ordering and dropping.

A variable is an abstractions of memory cell respectively a pointer or reference is an abstraction of a memory cell address. If such variable points neither to nor is a memory cell but an IO register (say a USART buffer FIFO e.g.) this optimising by re-ordering or dropping would be disastrous. This has to be inhibited at compile time as well as at run time. Indeed, at runtime too as those newer s are clever enough to re-order and drop memory accesses by them self.

Compile time memory access optimisation is inhibited by the volatile keyword for every variable and reference meaning an IO register. Just one omission is good for effects hard to diagnose.

Run time optimisation and especially re-ordering is inhibited by putting memory fence instructions (this one) at the right places. The BCM2835 data sheet says: "Accesses to the same peripheral will always arrive and return in-order. It is only when switching from one peripheral to another that data can arrive out-of-order. The simplest way to make sure that data is processed in-order is to place a memory barrier instruction at critical positions in the code. You should place: ∗ A memory write barrier before the first write to a peripheral. ∗ A memory read barrier after the last read of a peripheral. It is not required to put a memory barrier instruction after each read or write access. Only at those places in the code where it is possible that a peripheral read or write may be followed by a read or write of a different peripheral. This is normally at the entry and exit points of the peripheral service code. As interrupts can appear anywhere in the code so you should safeguard those. If an interrupt routine reads from a peripheral the routine should start with a memory read barrier. If an interrupt routine writes to a peripheral the routine should end with a memory write barrier."

That's quite clear and applies to BCM2836 and BCM2837, too (we have no data sheet for those, only for BCM2835). And to repeat: Too many volatiles and memory fences makes programs longer and slower. But one too less is good for disaster.

Language hint: In Java this would be volatile, transient and synchronised. Implementation hint: This macro is __↩ sync_synchronize() Interrupt hint: We and most of us do not make a sequence of IO accesses atomic by interrupt disable. We rely on interrupt routines doing it right as described in the data sheet.

7.15.3 Typedef Documentation

7.15.3.1 modBusLinkState_t

```
typedef enum modBusLinkState_t modBusLinkState_t
```

A set of possible states of a Modbus link.

Modbus link here means a connection to a concrete Modbus slave/server seen by the master/client.

Note: The numbering may change in future but the ordering off < operational < operated < error will not.

The set of states is limited by the interface type (RS485, Ethernet, ..) and may be further limited by the device or application. The subset ML_OFF ML_ON ML_INITED ML_ERR_REQ ML_ERR_RESP will be enough for some RS485 slaves.

7.15.4 Enumeration Type Documentation

7.15.4.1 modBusLinkState_t

```
enum modBusLinkState_t
```

A set of possible states of a Modbus link.

Modbus link here means a connection to a concrete Modbus slave/server seen by the master/client.

Note: The numbering may change in future but the ordering off < operational < operated < error will not.

The set of states is limited by the interface type (RS485, Ethernet, ..) and may be further limited by the device or application. The subset ML_OFF ML_ON ML_INITED ML_ERR_REQ ML_ERR_RESP will be enough for some RS485 slaves.

Enumerator

ML_OFF	do not use that Modbus device
ML_ON	may be used but connection not ready
ML_IDLE	may be usable, basic state
ML_INITED	initialised and settings (if any)
ML_REQSEND	request sent, response pending
ML_RESPREC	response received --> ML_INITED
ML_LISTEN	listening
ML_REQREC	request received
ML_RESPOND	respond sent
ML_ERR_ANY	no concrete error, lower bound of all error states
ML_ERR_INIT	initialisation error (hopeless when re-occurring)
ML_ERR_REQ	request error
ML_ERR_RESP	response error

7.16 include/growattHome.h File Reference

Types and values for the smart home's Growatt inverter handling (laboratory project)

```
#include "arch/config.h"
#include <stdint.h>
#include "weShareMem.h"
```

Data Structures

- struct modBvals_t

  Modbus readings and other process values.
- struct modSharMem_t

  Structure for shared memory.

Macros

- #define ANZ_SEMAS

  Standard semaphore set size.
- #define **ANZmodSLAVES**

  Number of Modbus attached inverters (usually 1)
- #define **GROWerror**

  JSON key for several inverter errors.
- #define **SEMAPHORE_KEY**

  Semaphore unique key "Kfig25".
- #define SHARED_MEM_DATA_SIZE

  Size of defined master slave communication data in shared memory.
- #define **SHARED_MEM_FILL_SIZE**

  Size of extra fill array to have a standard shared memory size.
- #define **SHARED_MEMORY_KEY**

  Shared memory key "Buffer25".
- #define **SHARED_MEMORY_SIZE**

  Shared memory size 256 byte.

7.16.1 Detailed Description

Types and values for the smart home's Growatt inverter handling (laboratory project)

```
Copyright (c) 2019 Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 247 5.05.2023
Rev. 213 07.07.2019 : new derived from sweetHome.h
Rev. 215 26.08.2019 : shared memory types adapted
```

This include file collects some configuration and naming common to
a) the Growatt inverter communication program
b) server side program(s) (CGI) for the web interfaces

All those programs, running on the system for process control and I/O, are written in C. The client side programming for the web HMI is in Javascript.

The server side programs communicate via shared memory and a set of three (ANZ_SEMAS) semaphores.

7.16.2 Macro Definition Documentation

7.16.2.1 ANZ_SEMAS

```
#define ANZ_SEMAS
```

Standard semaphore set size.

Usually three (3..10)

7.16.2.2 SHARED_MEM_DATA_SIZE

```
#define SHARED_MEM_DATA_SIZE
```

Size of defined master slave communication data in shared memory.

This macro calculates the size in bytes of the shared memory area, see valsSharMem_t. The size of the shared memory SHARED_MEMORY_SIZE should be chosen as a multiple of 256 (512) bytes. The extra bytes are usable as an array uint8_t fill[] of size SHARED_MEM_FILL_SIZE.

7.17 include/homeDoor.h File Reference

Common types and values for the smart home door bell and phone project.

```
#include "arch/config.h"
#include <stdint.h>
#include "weGPIOd.h"
```

Macros

- #define **BEEP**
  beeper, 1-active
- #define **DOoff**
  door opener inactive, 0-inactive
- #define **DOpRel**
  door opener relay, 1-active (5s max.)
- #define **KliMU**
  bell middle and lower floor, 0-active
- #define **KliOb**
  bell upper floor, 0-active
- #define **LEDgeL**
  left yellow LED, 1-active Ko 4h
- #define **LEDgeR**
  right yellow LED, 1-active bell middle & lower
- #define **LEDgnL**

left green LED, 1-active I'm alive blink

- #define **LEDgnR**

  right green LED, 1-active !door opener

- #define **LEDoc3**

  no LED, 1-active oc3 state

- #define **LEDrtL**

  left red LED, 1-active Kmu 4h

- #define **LEDrtR**

  right red LED, 1-active bell upper

- #define **OCres**

  unused optocoupler 3, 0-active

- #define **SamPer**

  50 Hz rectangle (a 20 ms 5 sample period)

- #define **Smp1o4**

  sample 3 of 4 every 4 ms (5∗50 Hz, doorUnl)

7.17.1 Detailed Description

Common types and values for the smart home door bell and phone project.

```
Copyright (c) 2020 Albrecht Weinert
weinert-automation.de a-weinert.de
```

Revision history

```
Rev. 244 28.09.2022
Rev. 223 18.06.2020 : new
Rev. 228 20.07.2020 : output LineF
Rev. 243 09.09.2022 : typo
```

This include file collects some configuration and naming common to
a) the process control program
b) server side program(s) (CGI) for the web interfaces

7.18 include/mqttHome.h File Reference

MQTT related definitions for an experimental smart home (lab) project.

```
#include <mosquitto.h>
```

Functions

- void **mqttClean** ()

  End as MQTT client.

- int mqttInit ()

  Initialise as MQTT client.

- void mqttPlg01Set (uint8_t const on)

  Switch the plug Plug01.

- void mqttPlg02Set (uint8_t const on)

  Switch the plug PLG2.

- void mqttPlg03Set (uint8_t const on)

  Switch the plug PLG3.

- void mqttPlg04Set (uint8_t const on)

  Switch the plug PLG4.

Variables

- char clientId [38]

  MQTT client ID.
- char mqttHost [68]

  The MQTT broker URL or name.
- int **mqttPort**

  MQTT port 1883.
- char subTopStPlg01 [14]

  State sub topic of S20 plug Number 01 to 09.

7.18.1 Detailed Description

MQTT related definitions for an experimental smart home (lab) project.

```
Copyright  (c)  2018   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /          /      /\
   /          /___    /  \       |
   \         /____\ /____\ |  _|__
    \  /\  / \    /       \|    |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 218 07.09.2019 : MQTT excepted from sweetHome2.h
Rev. 219 30.09.2019 : reconstructed after HDD crash
```

7.18.2 Function Documentation

7.18.2.1 mqttInit()

```
int mqttInit ( )
```

Initialise as MQTT client.

On success only: subscribe, loop and publish.

Returns

0: success the common mosq is set and usable; else: errno

7.18.2.2 mqttPlg01Set()

```
void mqttPlg01Set (
             uint8_t const on )
```

Switch the plug Plug01.

This function publishes the switch command via MQTT to the relay device Plug01, usually a Sonoff S20 with Tasmota.

Parameters

on	switch on when true, else off

7.18.2.3 mqttPlg02Set()

```
void mqttPlg02Set (
            uint8_t const on )
```

Switch the plug PLG2.

See mqttPlg01Set

7.18.2.4 mqttPlg03Set()

```
void mqttPlg03Set (
            uint8_t const on )
```

Switch the plug PLG3.

See mqttPlg01Set

7.18.2.5 mqttPlg04Set()

```
void mqttPlg04Set (
            uint8_t const on )
```

Switch the plug PLG4.

See mqttPlg01Set

7.18.3 Variable Documentation

7.18.3.1 mqttHost

```
char mqttHost[68]  [extern]
```

The MQTT broker URL or name.

May be set by option –mqttHost meterPi or –mqttBroker 192.168.178.87

default: localhost MQTTBroker (currently localhost !)

7.18.3.2 subTopStPlg01

```
char subTopStPlg01[14]  [extern]
```

State sub topic of S20 plug Number 01 to 09.

It's preset as plug01/POWER for 01, but the digit at index [5] will be set before each use accordingly.

7.18.3.3 clientId

```
char clientId[38]  [extern]
```

MQTT client ID.

default value: sweetHomeControl; length: 15; max. length: 36
May be changed before mqttInit().

MQTT client ID.

7.19 include/sweetHome.h File Reference

Common types and values for the smart home laboratory project.

```
#include "arch/config.h"
#include <stdint.h>
#include "weBatt.h"
#include "weShareMem.h"
```

Data Structures

- struct cmdLookUp_t
  Structure for a defined remote command.
- struct dayStrtVal_t
  Day start values.
- struct meterVal_t
  One smart meter's readings.
- struct phPckSwSet_t
  Simple Structure for phase packet switch setting.
- struct valFilVal_t
  Smart meters' and other process values.
- struct valsSharMem_t
  Structure for shared memory.

Macros

- #define **Amsk**

  would give away
- #define ANZ_SEMAS

  Standard semaphore set size.
- #define ANZmodSLAVES

  Number of smart Modbus meters.
- #define **AUTO_PPSWI_COMMAND**

  auto control (ON=manual)
- #define **BALL_CARLOAD_COMMAND**

  car load as PV surplus ballast
- #define **BAT_COMMANDS**

  battery commands mask
- #define **BATLOAD_COMMANDS**

  battery loading commands mask
- #define **BATUNL_COMMANDS**

  battery unloading commands mask
- #define **Bmsk**

  Controlled load battery (as ballast)
- #define **C250W_COMMANDS**

  give 250 W command mask
- #define **CARLOAD_COMMANDS**

  car load command mask
- #define **CHSBmsk**

  CHS mode ballast (1: on at 0.. max power at surplus)
- #define **CHSCmsk**

  CHS loading station: power line connected to car.
- #define **CHSLmsk**

  CHS current limit at one border.
- #define **CHSMmsk**

  CHS mode manual (1: on at max power)
- #define **CHSphas**

  CHS number of phases: 1..6; 0, 7 not used; 3 default.
- #define cmdBits_t

  Type for command bits.
- #define **DEC_CARLOAD_COMMAND**

  car load decrease max. current
- #define **DECLIM_PPSWI_COMMAND**

  reduce PPSWI limit (by 300W)
- #define **DECR_PPSWI_COMMAND**

  decrement (-1) PPSWI power
- #define getNoPhas()

  Get number of phases.
- #define **GIVE250_COMMANDS**

  commands concerning Plg01 & Rel5
- #define **Gmsk**

  Give away: Alarm / add ballast / etc.
- #define HIPPO_COMMANDS

  Mask for all Hippogreiff commands.
- #define **HWPUMP_COMMANDS**

comfort pump command mask

- #define **INC_CARLOAD_COMMAND**

  car load increase max. current
- #define **INC_PPSWI_COMMAND**

  increment (+1) PPSWI power
- #define **INCLIM_PPSWI_COMMAND**

  increase PPSWI limit (by 200W)
- #define **inv2Panel**(x)

  Inverter (mJ) is connected to panel.
- #define **Kmsk**

  Keep battery (minimal load voltage)
- #define **Lmsk**

  Load battery (normal program)
- #define **LOWER_BATLOAD_COMMAND**

  lower battery loading power
- #define **LOWER_PPSWI_COMMAND**

  lower PPSWI power
- #define **OFF_C250W_COMMAND**

  off control give 250 W
- #define **OFF_CARLOAD_COMMAND**

  E-car load off.
- #define **OFF_HIPPO_COMMAND**

  off Hippogreif; see HIPPO_COMMANDS
- #define **OFF_PLG1_COMMAND**

  off Plug01 give 250 W
- #define **OFF_PLG2_COMMAND**

  off Plug02
- #define **OFF_PLG3_COMMAND**

  off Plug03
- #define **OFF_PLG4_COMMAND**

  off Plug04
- #define **OFF_PPSWI_COMMAND**

  off PPSWI (Power packet switching)
- #define **ON_C250W_COMMAND**

  on control give 250 W
- #define **ON_CARLOAD_COMMAND**

  E-car load on.
- #define **ON_HIPPO_COMMAND**

  on Hippogreiff; see HIPPO_COMMANDS
- #define **ON_PLG1_COMMAND**

  on Plug01 give 250 W
- #define **ON_PLG2_COMMAND**

  on Plug02
- #define **ON_PLG3_COMMAND**

  on Plug03
- #define **ON_PLG4_COMMAND**

  on Plug04
- #define **ONMAN_PPSWI_COMMAND**

  on and manual PPSWI PPSWI
- #define **P2sPmsk**

  PPS element 2 power ∗ 50% (0:0% .. 2:100%)

- #define PCK100PERC_POWER

  Phase packet switching device.
- #define **PCK1PERC_POWER**

  phase packet switch 1% power / W
- #define **PCK50PRC_POWER**

  phase packet switch 50% power / W
- #define **PHDEC_CARLD_COMMAND**

  assume E-cars using 1 phase less
- #define **PHINC_CARLD_COMMAND**

  assume E-cars using 1 phase more
- #define **PLG1_COMMANDS**

  Plug01 command mask.
- #define **PLG2_COMMANDS**

  Plug02 command mask.
- #define **PLG3_COMMANDS**

  Plug03 command mask.
- #define **PLG4_COMMANDS**

  Plug04 command mask.
- #define **PPSLmsk**

  PPS max at limit (1: at 0.0 or PCK_POWER_LIM_MAX)
- #define **PPSMmsk**

  PPS mode manual (0: automatic, default)
- #define **PPSUmsk**

  PPS power at limit PCK_POWER_LIM_MAX.
- #define **PPSWI_COMMANDS**

  Power packet switching commands.
- #define **PUMP_STRT_POWR**

  Pump start heating power; see PUMP_STRT_TEMP.
- #define PUMP_STRT_TEMP

  Water (tank) temperature to start comfort pump when heating.
- #define **RAISE_BATLOAD_COMMAND**

  raise battery loading power
- #define **RAISE_PPSWI_COMMAND**

  raise PPSWI power
- #define SAFE_TEMP_WAT

  Upper limit of safe water (tank) temperature.
- #define **SEMAPHORE_KEY**

  Semaphore unique key "Kfig24".
- #define setNoPhas(x)

  Set number of phases.
- #define SHARED_MEM_DATA_SIZE

  Size of defined master slave communication data in shared memory.
- #define **SHARED_MEM_FILL_SIZE**

  Size of extra fill array to have a standard shared memory size.
- #define **SHARED_MEMORY_KEY**

  Shared memory key "Buffer24".
- #define **SHARED_MEMORY_SIZE**

  Shared memory size 256 byte.
- #define **Smsk**

  Solar power is generated.
- #define **START_BATKEEP_COMMAND**

start battery keep alive (with load)

- #define **START_BATLOAD_COMMAND**

  start battery loading
- #define **START_BATUNL_COMMAND**

  start battery unloading
- #define **START_HWPUMP_COMMAND**

  start hot water comfort pump
- #define **STD_CARLOAD_COMMAND**

  set 3 phases and 16A (11kW)
- #define **STOP_BAT_COMMAND**

  stop battery unloading & loading
- #define **STOP_HWPUMP_COMMAND**

  stop hot water comfort pump
- #define **Umsk**

  Unload battery commanded.
- #define **UNUSED_COMMANDS**

  4 unused command bits (June 2023)
- #define **Wmsk**

  Would give away 250W or more.

Functions

- float pwmToVolt (uint8_t const pwm)

  Output voltage of battery load module by PWM signal.
- uint8_t voltToPWM (float const u)

  PWM signal for desired output voltage of battery load module.

Variables

- cmdLookUp_t cmdLookUp [ ]

  The common command look up table.
- volatile float fLine

  Last valid power line frequency.
- uint8_t const loadModUlookup [256]

  Output voltage of electronic battery load module.
- float const **phPckSwPow** [101]

  First (or only) phase packet switching device power look up.
- phPckSwSet_t const phPckSwSets [101]

  The packet switch control values.
- smdX30modbus_t smdX30modbus [2]

  Descriptive and state array for smart meters on Modbus.
- volatile int tempTankWater

  Last value of tank water temperature.
- volatile uint8_t tempWaterBadCnt

  Tank water temperature bad read count.
- valFilVal_t **valFilVal**

  All process values relevant for log files and HMI.

7.19.1 Detailed Description

Common types and values for the smart home laboratory project.

```
Copyright  (c)  2018 2023   Albrecht Weinert
weinert-automation.de       a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \     /      \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 253 9.10.2023
Rev. 77+ 07.12.2017 : new
Rev.  99 29.01.2018 : storage battery load module voltage handling added
Rev. 149 17.06.2018 : PLG0n renamed to PLGn (PLG2) for consistent search
Rev. 159 04.07.2018 : weBatt.h excerpted
Rev. 167 19.06.2018 : phPckSwPow [100]  950.00 W
Rev. 187 04.10.2018 : wSumExp in valFilVal_t (unload guarded)
Rev. 190 12.02.2019 : prepare VDE-AR-N 4105 (relays) handling
Rev. 217 05.09.2019 : phase packet switch ballast: three variants
Rev. 220 09.11.2019 : Growatt values added to valFilVal_t
Rev. 222 18.03.2020 : pps 100% power raised to 1267 W due to 3rd panel set
Rev. 229 23.07.2020 : valFil (CSV) removed
Rev. 245 27.02.2023 : pps 100% power raised to 1983 W due panel reorg.
Rev. 245 01.05.2023 : utcDawn, utcDusk (civil twilight) not in dayStrtVal_t
Rev. 251 12.07.2023 : phase packet switch power limit added to valFilVal_t
```

In the smart home's PV under control, we assume one smart meter for the whole house's (public) electricity supply and one for solar generators and battery storage. The (two, cf. ANZmodSLAVES) meters are Modbus controlled. Additionally we have home and PV related process control.
The process IO related definitions are mostly in include/sweetHome2.h

This include file collects some configuration and naming common to
a) the process control program
b) a console HMI program and the
c) server side program(s) (CGI) for the web interfaces

All those programs, running on the system for process control and I/O, are written in C. The client side programming for the web HMI is in Javascript.

The server side programs communicate via shared memory and a set of three (ANZ_SEMAS) semaphores.

7.19.2 Macro Definition Documentation

7.19.2.1 ANZ_SEMAS

```
#define ANZ_SEMAS
```

Standard semaphore set size.

Usually three (3..10)

7.19.2.2 ANZmodSLAVES

```
#define ANZmodSLAVES
```

Number of smart Modbus meters.

Usually two.

7.19.2.3 cmdBits_t

```
#define cmdBits_t
```

Type for command bits.

Commands to the process control program from outside are transferred via shared memory as one bit set for every command to be executed. The bit position (0..31, as of May 2018) defines a concrete command.

"From outside" usually means from a web interface (html page with Javascript) and AJAX to a C written GCI program communicating via shared memory with the process control program. In this schema the CGI program sets bits for commands to be executed and the process control program clears the bit(s) in question on commands executed (or rejected).

Remark: On contradictory commands, ONyxz and OFFxyz, one will be executed and both will be cleared. In cases like this OFF... usually gets priority.

Remark 2: When exceeding 32 distinct commands we may switch to a uint64_t as wasdone from 16_t to 32_t in May 2018.

7.19.2.4 HIPPO_COMMANDS

```
#define HIPPO_COMMANDS
```

Mask for all Hippogreiff commands.

Hippogreiff stands for a relay controlled 12V supply from storage battery. It should be ON at night, or more precisely between sunset and sunrise or from twilight to twilight.
One consumer on this rail is a green LED beam on a Hippogreiff sculpture. Hence the name.

7.19.2.5 PCK100PERC_POWER

```
#define PCK100PERC_POWER
```

Phase packet switching device.

Since Rev. 157 (07-2018) we have one single phase heating device with then 950 W power, raised 03.2020 to 1267W and 02.2023 to 1845. It is actuated by a zero crossing electronic switch (SSR) at pin PH_P_SSR1 controlling the amount of power in 1% (PCK1PERC_POWER) steps.
For robustness and safety this electronic relay sits behind a 10A relay.

Since march 2023 we have two heater elements of 2kW each:
heater element 1 phase packet switch SSR1 PH_P_SSR1) 0..100%
heater element 1 phase packet switch SSR1 PH_P_SSR1) 0, 50, 100%

While the first one is controlled in 101 steps as before, the second one got three states: off, half and full power. This effectively gives a range of 0 to about 4kW in 201 steps. In the present configuration any PV surplus can be handled as in-house-consumption (Eigenverbrauch).

7.19.2.6 SAFE_TEMP_WAT

```
#define SAFE_TEMP_WAT
```

Upper limit of safe water (tank) temperature.

The integer value for safe water (tank) temperature sensor reading is +68950 m℃ respectively 69 ℃. Above this value all (electric) heating must stop.

7.19.2.7 PUMP_STRT_TEMP

```
#define PUMP_STRT_TEMP
```

Water (tank) temperature to start comfort pump when heating.

The integer value for pump start water (tank) temperature sensor reading is +58950 m℃ respectively 59 ℃. Above this value electric heating with >= 1kW (surplus) power shall start the (comfort) circulation pump while heating with electrical (surplus) power.

This is done in the hope to disturb the temperature layers in the tank and, hence, reduce hot water outlet temperature.

See also

PUMP_STRT_POWR

PUMP_STRT_FOR

7.19.2.8 getNoPhas

```
#define getNoPhas( )
```

Get number of phases.

Returns

number of (set) phases available for car loading

7.19.2.9 setNoPhas

```
#define setNoPhas(
              x )
```

Set number of phases.

Parameters

x	1..6

Returns

valFilVal.by[7]

7.19.2.10 SHARED_MEM_DATA_SIZE

```
#define SHARED_MEM_DATA_SIZE
```

Size of defined master slave communication data in shared memory.

This macro calculates the size in bytes of the shared memory area, see valsSharMem_t. The size of the shared memory SHARED_MEMORY_SIZE should be chosen as a multiple of 256 (512) bytes. The extra bytes are usable as an array uint8_t fill[] of size SHARED_MEM_FILL_SIZE.

7.19.3 Function Documentation

7.19.3.1 pwmToVolt()

```
float pwmToVolt (
            uint8_t const pwm )
```

Output voltage of battery load module by PWM signal.

Parameters

pwm	PWM signal 0 (lowest) .. 255 (highest output voltage)

Returns

output voltage 10.8 .. 15.2V

7.19.3.2 voltToPWM()

```
uint8_t voltToPWM (
            float const u )
```

PWM signal for desired output voltage of battery load module.

Parameters

u	desired output voltage 10.8 .. 15.2V out of range returns 0 or 255

Returns

PWM signal width 0..255 (i.e 0..100%)

7.19.4 Variable Documentation

7.19.4.1 loadModUlookup

```
uint8_t const loadModUlookup[256]  [extern]
```

Output voltage of electronic battery load module.

Depending on the (400Hz) PMW signal width 0..255 (/255 $*$ 100%) the battery load module will deliver about 10.8 to 15.2V. If the battery voltage is higher than the output voltage no current will flow (i.e. nothing happens). If the battery voltage is lower, up to 10A will be delivered. This current limit is implemented / fixed by the power module used. Hence, getting less current respectively power requires a fine control of the PWM signal (see POWH).

This array gives the module output voltage in 100mV units (value) as function of the PWM width (index). For the values see the comment in the file sweetHome.c.

7.19.4.2 smdX30modbus

```
smdX30modbus_t smdX30modbus[2]  [extern]
```

Descriptive and state array for smart meters on Modbus.

The number of meters is ANZmodSLAVES.

7.19.4.3 cmdLookUp

```
cmdLookUp_t cmdLookUp[]  [extern]
```

The common command look up table.

It must end with an entry {"", 0}.

The current (CGI) program uses linear search for the command mnemonic. Hence, and cause of structure, alphabetic sorting is of no avail.

7.19.4.4 phPckSwSets

```
phPckSwSet_t const phPckSwSets[101]  [extern]
```

The packet switch control values.

The array holds the number of on and off phases (i.e. 20 ms periods at 50 Hz line frequency) for each percentage of full power. The array length is 101; the index [0..100] is, hence, directly the percentage wanted.

To avoid too visible flicker, for no set (phPckSwSets[i].onPhases, phPckSwSets[i].offPhases) the smaller of the two values would be greater than 4 (80 ms); in most cases it is 1 or 2 (40 ms).

7.19.4.5 fLine

```
volatile float fLine  [extern]
```

Last valid power line frequency.

The measurement is taken from the more precise home three phase meter if communicative via Modbus. Otherwise the other meter's value is taken if available. Due to slow Modbus communication the values may be older than two seconds worst case.
This low sampling rate won't comply with VDE-AR-N 4105 cut off rules requiring a 0.1s reaction outside 47.5..51.↩ 5Hz. Nevertheless the current experimental set-up is a working proof-of-concept implementation (with very low power) — and a faster frequency measurement could easily be implemented.

Last valid power line frequency.

7.19.4.6 tempTankWater

```
volatile int tempTankWater  [extern]
```

Last value of tank water temperature.

This value will be the last good .tempTankTop (see valFilVal_t) or the last good .tempPipe. If neither is good for 251 measurements BAD_TEMP_READ (an incredibly high value) will be set.
This last tank water temperature will be checked against a safety limit to allow electric heating as ballast.
The value is in units of 1/1000 grdC.

7.19.4.7 tempWaterBadCnt

```
volatile uint8_t tempWaterBadCnt  [extern]
```

Tank water temperature bad read count.

This last tank water temperature will be checked against a safety limit to allow electric heating as ballast.

In case of too many bad reads the value must be fixed at 253

7.20 include/sweetHome2.h File Reference

Types values and functions for an experimental smart home (lab) project.

```
#include "weStateM.h"
#include "sweetHome.h"
#include "weGPIOd.h"
#include "weModbus.h"
#include "mqttHome.h"
#include "we1wire.h"
#include <errno.h>
```

Macros

- #define BAT_KEEP_PWM

  Battery keep alive PWM setting.
- #define BAT_LDBA_PWM

  Battery maximum load ballast PWM.
- #define BAT_LDBAL_TIM

  Maximum battery load as ballast time.
- #define **BATu**

  Battery unload (hi active via open drain; gr/gr-ws)
- #define BEFORE_RISE

  Before sunrise timer offset.
- #define **CONb**

  Inverter to battery step up (dto.; ws/ge-ws)
- #define **CONcarLd**

  Contactor (relay) power to car is ON (low active)
- #define **CONcarRq**

  Car (CP resistor 2k7) connected / ready (low active)
- #define **HYPr**

  Bat. to Hippogreiff (dto. ws/gn)
- #define **isOnPPSrel**()

  Check if the phase power switch control relay is on.
- #define **LED9**

  life LED hi active
- #define **LEDx**

  other LED hi active (currently unused, always on)
- #define offPort(N)

  Switch port N off.
- #define offRel(N)

  Switch relay N off.
- #define onPort(N)

  Switch port N on.
- #define onRel(N)

  Switch relay N on.
- #define **PCK_POWER_IND_MAX**

  phase packet control limit 200%
- #define **PCK_POWER_LIM_DEF**

  phase packet default limit 3,6kW
- #define **PCK_POWER_LIM_MAX**

  phase packet max. limit
- #define **PCK_POWER_REL**

  phase packet switch control relay
- #define PDEL_BALEND

  Minimal delivery power to stop ballast.
- #define PDEL_BALINC

  Minimal delivery power not to add ballast.
- #define PDEL_BALRED

  Minimal delivery power to reduce ballast.
- #define **PDEL_SGCONS**

  Minimal delivery power (W) to signal consumer.
- #define PDEL_SGGIVE

Maximal delivery power to signal give away.

- #define **PH_P_SSR1**

  Phase packet switch SSR1 (heater element 1)
- #define **PH_P_SSR2**

  Phase packet switch SSR2 (heater element 2)
- #define **POWH**

  Power module voltage control (li/br-ws).
- #define **ppsContPow**

  phase packet switch: contactor power /W
- #define **PWMcarLd**

  PWM to control max. car load current (per phase)
- #define SENS0PATH

  Temperature of water tank (top level).
- #define SENS1PATH

  Temperature of hot water pipe (from tank top).
- #define SENS2PATH

  Temperature of water tank (bottom level).
- #define **SIG_GIV_PLUG**

  signal give away via MQTT sonoff plug (none or 4)
- #define switchPPSrel(V)

  Actuate phase power switch control relay.
- #define switchRel(N, V)

  Switch relay N on or off.

Functions

- void logBatteryStress ()

  Log battery stress on outLog as line with time stamp.
- uint8_t pckSwPec (float power)

  Set and get package switch percentage by power.
- void reduceBatLoadPWM (uint8_t const pwmRed)

  Reduce battery loader module PWM.
- void setBatLoadPWM (uint8_t pwm)

  Set battery loader module PWM.
- uint8_t setPckSwPerc (uint8_t perc)

  Set package switch percentage.
- float setPhPckLimit (float powerLimit)

  Set and get package switch power limit.
- void switchBatToStepUp (uint8_t on)

  Switch battery to step up converter.
- void switchHotWpump (uint8_t const on)

  Hot water comfort pump turn on/off.
- void switchInvToBatUnl (uint8_t on)

  Switch inverter to battery unload.

Variables

- state_t **batCntBalSeq**

  Battery controlled load as ballast.
- state_t **batCntLodSeq**

  Battery loading and keeping.
- state_t batKeepInh

  Inhibit battery keep alive loading.
- int batUnloadAllowed

  Battery unload allowed.
- state_t **batUnloadSeq**

  Battery controlled unload via inverter.
- uint8_t batVoltValid

  The measured battery voltage is valid.
- state_t befRiseTimer

  Before sunrise timer.
- state_t consumeGiveHyst

  Give away hysteresis.
- state_t consumeGiveSFC

  Give away state machine / SFC.
- state_t dawnTimer

  Dawn timer.
- state_t duskTimer

  Dusk timer.
- uint8_t **heaterAllowed**

  phase packet load enabled (always)
- uint16_t hundredsV

  The battery voltage in 0.01V units.
- uint32_t hundVsecAbove

  Hundreds of Vs above 13.6 V battery voltage.
- uint32_t hundVsecBelow

  Hundreds of Vs below 12.25 V battery voltage.
- state_t **load250Wcont**

  Controlled load SFC.
- volatile uint8_t **phPckCnt**

  current switch state duration counter
- volatile uint8_t **phPckOff**

  current phase packet switch OFF duration
- volatile uint8_t **phPckOn**

  current phase packet switch ON duration
- volatile uint8_t **phPckRelOffDelay**

  PPS relay off delay.
- volatile uint8_t **phPckS2**

  heater 2 pps switch state: 0 off; 1 on
- volatile uint8_t **phPckSw**

  phase packet switch state: 0 off; 1 on
- volatile uint8_t phPckSwIndex

  Actual index (0..200%) determining power.
- volatile uint8_t **phPpow2**

  heater 2 power ∗ 50% (0:0% .. 2:100%)
- oneWireDevice_t **sensors** [3]

The 1-wire sensors used.

- state_t **solarPowerHyst**

  Solar power producer hysteresis.

- state_t sunriseTimer

  Sunrise timer.

- state_t sunsetTimer

  Sunset timer.

- state_t wouldGive250WHyst

  Would give away 250W hysteresis.

- state_t wouldGiveAwayHyst

  Would give away hysteresis.

7.20.1 Detailed Description

Types values and functions for an experimental smart home (lab) project.

```
Copyright  (c)  2018 - 2023  Albrecht Weinert
weinert-automation.de        a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 253 9.10.2023
Rev.  99 29.01.2018 : load module voltage handling added
Rev. 148 16.06.2018 : battery ballast handling improved, Hippogreiff relay
Rev. 170 24.07.2018 : temperature sensors documentation
Rev. 190 12.02.2019 : remove 4 status relay output as (minimal) preparation
                      for VDE-AR-N 4105 evaluation and inverter cut off
Rev. 192 16.02.2019 : REL2 3 interchanged (2 now extra meter cut off)
Rev. 218 07.09.2019 : MQTT excerpted to mqttHome.h
Rev. 241 12.08.2021 : battery values updated (200Ah ++)
Rev. 243 21.09.2022 : overdue modifications
Rev. 245 24.02.2023 : battery keep 24 -> 12.5V
Rev. 252 17.08.2023 : ECar loading ++
```

This file is the addendum to sweetHome.h and sweetHome.c. It contains process IO related issues. These are not necessary for programs related to pure HMI, logging and the like. For those programs sweetHome.h and sweetHome.c will be sufficient.

In the smart home under control, there's one smart meter for the whole house's public electricity supply and one for solar generators and battery storage. The (two, cf. ANZmodSLAVES) meters are connected by Modbus.

GPIO usage

See also file /forDSocu/Raspi3ioPinsMeterPi.ods

GPIO pins are used to control relays, LEDs and one button in the Pi's immediate neighbourhood. Three open drain (module) outputs control slightly remote (10m) periphery via shielded cable.

This program uses eight relays (one 10A change over contact each) in an eight relays module. The relays are energised by an active low "Din" (n:1..8) signal.

Eight 10A relays module connected via RS485 converter/shield pin connectors

```
module  : Gnd  Di1  Di2  Di3  Di4  Di5  Di6  Di7  Di8  3V3     Gnd  5V
colour  : brn  blk  wht  gry  vio  blu  grn  yel  ora  red     brn  red
RS conv.: Gnd   P0   P1   P2   P3   P4   P5   P6   P7  3.3V    Gnd  5V
Pi Pin  : 6&c   11   12   13   15   16   18   22    3  1 &c    6&c  2&3
GPIO Pi3:       17   18   27   22   23   24   25    2
IO name :     REL1 REL2 REL3 REL4 REL5 REL6 REL7 REL8
```

Other process IO see project file /forDocu/Raspi3ioPinsMeterPi.ods .

Notes on process IO functions

```
  REL1 :  defect XX   (ex PPS-SSR pre-relay)
  REL2 :  off miniJoule's Ferraris meter on battery unload (out of use 6'23)
  REL3 :  not used (ex VDE-AR-N 4105)
  REL4 :  phase packet switch (PPS-SSR) pre-relay for heater contactor
  REL5 :  would give away more than 250 W          (prolonged 2h)
  REL6 :  not used (ex VDE-AR-N 4105)
  REL7 :  not used (ex VDE-AR-N 4105)
  REL8 :  hot water circulation pump  (comfort function)
  Note: When searching relay usages search for macros onRel(N) and offRel(N)
  LED9 :  red LED; Modbus slave indicator;  on: Modbus slave index > 0
  LEDx :  green LED; no function at the moment, always on
  HWPB :  start hot water pump (button to GND)
  POWH :  Battery load power module voltage control (via open drain)
          PWM for adjustable 11..14V 10A power module
          lo or off: for lowest voltage
  BATu :  Battery unload (hi active);  battery to step up
          when battery surveyor signal is OK
  CONb :  Step up to miniJoule inverter (32V)
  HYPr :  Battery to Hippogreiff
PH_P_SSR1: Phase packet switch SSR1 (heater element 1) 0..100%
PH_P_SSR2: Phase packet switch SSR2 (heater element 2) 0, 50, 100%
PWMcarLd : PWM to control max. car load current (per phase)
CONcarLd : Contactor (relay) "power to car" is set ON  (low active OC in)
CONcarRq : Car (i.e. CP resistor 2k7) is connected (low active OC in)
```

MQTT usage

The MQTT protocol is used for remote process IO within "sweet home's" private (W)LAN:
a) battery surveillance and voltage measurement by specialised ESP8266
b) For Sonoff S20 plug switches modified to be MQTT clients (Tasmota).

```
PLG1 "would give 250 W" = parallel to REL5
PLG2 sunset to sunrise  (since 03.03.2018; as Hippogreiff)
PLG3 just controlled by web interface (since August 2019 used
     for outside air monitor's supply, to enable remote reset)
PLG4 "actually giving away" (neg. supply) warning
```

7.20.2 Macro Definition Documentation

7.20.2.1 SENS0PATH

```
#define SENS0PATH
```

Temperature of water tank (top level).

This is the address of the 1-wire sensor's value file provided by the 1-wire-temp kernel module.

7.20.2.2 SENS1PATH

```
#define SENS1PATH
```

Temperature of hot water pipe (from tank top).

See also SENS0PATH.

7.20.2.3 SENS2PATH

```
#define SENS2PATH
```

Temperature of water tank (bottom level).

See also SENS0PATH.

7.20.2.4 BAT_KEEP_PWM

```
#define BAT_KEEP_PWM
```

Battery keep alive PWM setting.

This is the PWM setting for the battery load module, see loadModUlookup, for permanently keeping the battery at live by preventing self discharge and feeding low permanent loads.
Value until 05.07.2018: 23 -> 12.3V = 60%, see BATVOLT_IDLE_60
05.07.2018: 21 -> 12.2V = 50%, see BATVOLT_IDLE_50
03.08.2018: 29 -> 12.6V > 60%, see BATVOLT_IDLE_60
13.08.2018: 19 -> 12.1V > 60%, see BATVOLT_IDLE_50
12.08.2021: 20 -> 12.3V

7.20.2.5 BAT_LDBA_PWM

```
#define BAT_LDBA_PWM
```

Battery maximum load ballast PWM.

See loadModUlookup

7.20.2.6 BAT_LDBAL_TIM

```
#define BAT_LDBAL_TIM
```

Maximum battery load as ballast time.

Value: 4h (in s)

7.20.2.7 PDEL_BALEND

```
#define PDEL_BALEND
```

Minimal delivery power to stop ballast.

Value: 210.2 W

7.20.2.8 PDEL_BALRED

```
#define PDEL_BALRED
```

Minimal delivery power to reduce ballast.

Value: 41.02 W

7.20.2.9 PDEL_BALINC

```
#define PDEL_BALINC
```

Minimal delivery power not to add ballast.

Value: 14.02 W

7.20.2.10 PDEL_SGGIVE

```
#define PDEL_SGGIVE
```

Maximal delivery power to signal give away.

Value: 14.3 W

7.20.2.11 switchPPSrel

```
#define switchPPSrel(
              V )
```

Actuate phase power switch control relay.

For safety reasons there is a contactor (Hager 4 $*$ 40A) in front of the electronic zero crossing switches (solid state relays SSR) for the two heater elements.

The PPS relay (4) actuates this contactor. The Hager contactor consumes about 9W, see ppsContPow, when actuated.

Parameters

V	0: off; else: on

7.20.2.12 onPort

```
#define onPort(
              N )
```

Switch port N on.

Parameters

N	port name (LEDx, BATu, CONb ..)

7.20.2.13 offPort

```
#define offPort(
              N )
```

Switch port N off.

Parameters

N	port name (LEDx, BATu, CONb ..)

7.20.2.14 switchRel

```
#define switchRel(
              N,
              V )
```

Switch relay N on or off.

Parameters

N	relay number 1..8
V	0: off; else: on

7.20.2.15 onRel

```
#define onRel(
              N )
```

Switch relay N on.

Parameters

N	relay number 1..8

7.20.2.16 offRel

```
#define offRel(
              N )
```

Switch relay N off.

Parameters

N	relay number 1..8

7.20.2.17 BEFORE_RISE

```
#define BEFORE_RISE
```

Before sunrise timer offset.

See also: befRiseTimer value: 248 minutes (since 12.08.2021; 188 before)

7.20.3 Function Documentation

7.20.3.1 logBatteryStress()

```
void logBatteryStress ( )
```

Log battery stress on outLog as line with time stamp.

Logs the battery voltage and the under and overvoltage stress levels.

7.20.3.2 pckSwPec()

```
uint8_t pckSwPec (
            float power )
```

Set and get package switch percentage by power.

Besides determining an returning the percentage(power), this function sets the process control values via setPckSwPerc() returns its value.

Parameters

power	in W

Returns

the phase packet percentage 0..100

7.20.3.3 setPhPckLimit()

```
float setPhPckLimit (
            float powerLimit )
```

Set and get package switch power limit.

This function sets the PPS power limit in the range 0.0 ... PCK_POWER_LIM_MAX.

Parameters

powerLimit	PPS power limit in W

Returns

the (new) power limit

7.20.3.4 setPckSwPerc()

```
uint8_t setPckSwPerc (
            uint8_t perc )
```

Set package switch percentage.

This function sets phPckSwIndex by the parameter value in the range 0..200; respectively PCK_POWER_IND_MAX. Additionally it adjusts the current phPckCnt should its value be higher than by the new setting.

On transitions from respectively to 0 the control relay (PCK_POWER_REL) is actuated before respectively after actuating the electronic switch.

Parameters

perc	0..200; values above PCK_POWER_IND_MAX will have no effect and return the current setting un-altered

Returns

the actual (new) phase packet percentage 0..200

7.20.3.5 switchHotWpump()

```
void switchHotWpump (
            uint8_t const on )
```

Hot water comfort pump turn on/off.

It turns the hot water comfort circulation pump on respectively off.

Parameters

on	!= 0 : on; else, ==0 : off

7.20.3.6 switchInvToBatUnl()

```
void switchInvToBatUnl (
            uint8_t on )
```

Switch inverter to battery unload.

When the parameter is !=0 respectively ON, this function switches the (small) inverter from panel to the battery unload step up converter and turns the extra panel work meter off.

When the parameter is off it does the other way round.

Parameters

on	true: switch inverter to step up converter (battery unload); 0, false: switch inverter to panel.

7.20.3.7 switchBatToStepUp()

```
void switchBatToStepUp (
            uint8_t on )
```

Switch battery to step up converter.

When the parameter is !=0 respectively ON, this function switches the battery to the step up converter.

When the parameter is off, this function switches the battery to the (max. 20 A) battery loader / keep alive module .

Parameters

on	true: switch battery to step up converter; 0, false: switch battery to load modul.

7.20.3.8 setBatLoadPWM()

```
void setBatLoadPWM (
            uint8_t pwm )
```

Set battery loader module PWM.

This function sets the battery load module power PWM signal and hence its output voltage. 0 is the lowest and 255 the highest possible setting.

Parameters

pwm	the pwm ratio 0: 0%; 255: 100%

7.20.3.9 reduceBatLoadPWM()

```
void reduceBatLoadPWM (
            uint8_t const pwmRed )
```

Reduce battery loader module PWM.

This function reduces the battery load module power PWM signal and hence its output voltage by the parameter value, if that is not above the current setting.

See also

setBatLoadPWM()

Parameters

pwmRed	the reduction of the pwm ratio

7.20.4 Variable Documentation

7.20.4.1 hundVsecAbove

```
uint32_t hundVsecAbove  [extern]
```

Hundreds of Vs above 13.6 V battery voltage.

This variable is a counter for the battery voltage above 13.6 V taken in 0.01V units. This incremented Vs ("↩ Voltseconds") value is to indicate the battery's stress respectively gasing probability when being loaded.

Note: One hour at 15.0 V would give 504000 ∗ 0.01Vs

7.20.4.2 hundVsecBelow

```
uint32_t hundVsecBelow  [extern]
```

Hundreds of Vs below 12.25 V battery voltage.

This variable is a counter for the battery voltage below 12.2 V taken in 0.01V units. This incremented Vs ("↩ Voltseconds") value is to indicate the battery's integral time below a load state permanently acceptable.

Note: One hour at 11.8 V would give 144000 ∗ 0.01Vs

7.20.4.3 hundredsV

```
uint16_t hundredsV  [extern]
```

The battery voltage in 0.01V units.

This is just an integer value consistent to the last valid battery voltage measurement valFilVal.batVolt. Contrary to valFilVal.batVolt which is set to -0.9 to indicate invalidity after 2.4s without new (MQTT) measurements, this value will be kept (forever).
It is preset with 1289 (12.89 V) lest have battery low before the first valid MQTT measurement / message.

7.20.4.4 batVoltValid

```
uint8_t batVoltValid  [extern]
```

The measured battery voltage is valid.

When not 0 the last MQTT battery voltage measurement respectively message is not older than 2.4s and hence considered the actual valid value.

7.20.4.5 batUnloadAllowed

```
int batUnloadAllowed  [extern]
```

Battery unload allowed.

Bit 0 (1): allow after sunset
Bit 1 (2): allow before sunrise
start value: 8 set default value (usually 2 in winter and 3 in summer) Note: Made int en lieu de uint_8 for use in getopt_long.

See also

BAT_UNL_SUMM

BAT_UNL_WINT

7.20.4.6 phPckSwIndex

```
volatile uint8_t phPckSwIndex  [extern]
```

Actual index (0..200%) determining power.

This is the control variable for the electric heater elements in the hot water tank. Until June 2023 there was one such element of PCK100PERC_POWER W and this variable had a range of 0..100%. Since July 2023 there are two heater elements of equal power. The range of this control variable was extended to 0..200%. The power distribution between the two heater elements is done by software; see setPckSwPerc(uint8_t const perc) and PCK_POWER_IND_MAX

7.20.4.7 batKeepInh

```
state_t batKeepInh  [extern]
```

Inhibit battery keep alive loading.

While running disallow battery keep alive with BAT_KEEP_PWM. This timer is initially OFF and will be stopped in advance by low battery voltage stress and battery keep command via HMI/GUI.
It will be started after battery unload, after battery ballast load and by battery off command via GUI.

7.20.4.8 sunriseTimer

```
state_t sunriseTimer  [extern]
```

Sunrise timer.

This timer will run out every day at (approximated) sunrise. For the "every day" behaviour, its state change function will — after all due actions — restart this timer for the next 24h to hit (very approximately) the next sunrise. This acceptable guess will be adjusted at day change or at program start.

7.20.4.9 sunsetTimer

```
state_t sunsetTimer  [extern]
```

Sunset timer.

This timer will run out every day at (approximated) sunset. See also sunriseTimer

7.20.4.10 dawnTimer

```
state_t dawnTimer  [extern]
```

Dawn timer.

This timer will run out every day at (approximated) dawn, meaning the beginning of civil twilight. See also sunriseTimer and duskTimer

7.20.4.11 duskTimer

```
state_t duskTimer  [extern]
```

Dusk timer.

This timer will run out every day at (approximated) dusk, meaning the end of civil twilight. About this time street lamps and position lights may be lit. See also sunriseTimer

7.20.4.12 befRiseTimer

```
state_t befRiseTimer  [extern]
```

Before sunrise timer.

This timer will run out every day at about 180 min before (approximated) sunrise. The offset must be sufficient time to unload the battery before sunrise.
See also: sunriseTimer BEFORE_RISE

7.20.4.13 consumeGiveHyst

```
state_t consumeGiveHyst  [extern]
```

Give away hysteresis.

thresholds: PDEL_SGGIVE, PDEL_SGCONS

7.20.4.14 consumeGiveSFC

```
state_t consumeGiveSFC  [extern]
```

Give away state machine / SFC.

Status ON means consumer: OK.
Status OFF means very low power consumption: Inhibit power delivery.

7.20.4.15 wouldGiveAwayHyst

```
state_t wouldGiveAwayHyst  [extern]
```

Would give away hysteresis.

thresholds (04.03.18): -/+ 7W

7.20.4.16 wouldGive250WHyst

```
state_t wouldGive250WHyst  [extern]
```

Would give away 250W hysteresis.

thresholds (04.03.18): -250 / -178W

7.21 include/sweetHomeLocal.h File Reference

Localisation and geographical values for the smart home laboratory project.

7.21.1 Detailed Description

Localisation and geographical values for the smart home laboratory project.

```
Copyright  (c)  2018   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 247 5.05.2023
Rev. 144 12.06.2018 : new
Rev. 164 11.07.2018 : sunset/sunrise location parameters; local macros
Rev. 190 13.02.2019 : minor, comments only
```

This file contains some localisation, geographical and individual technical data for the current smart home experimental / demonstrator set-up. When porting to other locations etc. do change the values in question or, better, define a new LOCATION.

7.22 include/sysBasic.h File Reference

Some very basic definitions.

```
#include <basicTyCo.h>
#include <time.h>
#include <stdio.h>
#include <string.h>
```

Macros

- #define ABS_MONOTIME

  /def ABS_MONOTIME
- #define DAY_EQNOX_AUTUMN

  Day of the autumn equinox.
- #define DAY_EQNOX_SPRING

  Day of the spring equinox.
- #define **MS100_ns**

  Hundred milliseconds in ns.
- #define **MS10_ns**

  Ten milliseconds in ns.
- #define **MS1_ns**

  One millisecond in ns.
- #define SETPRGDATA

  Frame for program data definition.

Typedefs

- typedef struct timespec timespec

  The Linux time structure.

Functions

- float cosDay (short int dayInYear)

  Cosine of day in year, function.
- uint8_t **errLogIsStd** (void)

  Error log (errLog) is standard stream or outLog.
- int formatDec2Digs (char *targTxt, uint32_t value)

  Format number as two digit decimal number with leading zeroes.
- int formatDec3Digs (char *targTxt, uint32_t value)

  Format number as three digit decimal number with leading zeroes.
- int formatTmTim (char *rTmTxt, struct tm *rTm)

  Format broken down real time and date as standard text.
- int formatTmTiMs (char *rTmTxt, struct tm *rTm, int millis)

  Format broken down real time clock+ms as standard text.
- __time_t getDaySunrise (short int const dayInYear, uint32_t const meanSunriseSec, uint16_t const halfRiseDeltaMin)

  Get sunrise in s from UTC midnight.
- __time_t getDaySunset (short int const dayInYear, uint32_t const meanSunsetSec, uint16_t const halfSetDeltaMin)

  Get sunset in s from UTC midnight.
- uint8_t isFNaN (float const val)

  Floating point NaN.
- uint8_t littleEndian ()

  Actual runtime / architecture is little endian.
- void logEventText (char const *txt)

  Log an event/log message on outLog.
- void monoTimeInit (timespec *timer)

  Absolute timer initialisation.
- uint8_t **outLogIsStd** (void)

  Event log (outLog) is standard stream.
- void printNamRevDat (void)

  Print the program name, SVN revision and date.
- void printRevDat (void)

  Print the program SVN revision and date.
- char const * progDat ()

  The program date.
- char const * progNam ()

  The program name.
- char const * progNamB ()

  The program name with blank.
- char const * progRev ()

  The program revision.
- size_t strlcat (char *dest, char const *src, size_t num)

  String concatenation with limit.
- size_t strlcpy (char *dest, char const *src, size_t const num)

  String copy with limit.
- int switchErrorLog (char const *const errFilNam)

  Switch errlog to other file.
- int switchEventLog (char const *const logFilNam)

  Switch outLog to other file.
- void timeAddNs (timespec *t1, long ns)

Add a ns increment to a time overwriting it.

- int timeStep (timespec *timeSp, unsigned int micros)

  A delay to an absolute step specified in number of s to a given time.

- void updateReaLocalTime (void)

  Update local real time.

Variables

- timespec **actRTime**

  Actual time (structure, real time clock).

- struct tm actRTm

  Actual time (broken down structure / local).

- float const cosDiY [192]

  Cosine of day in year, look up.

- short int const cosDiY60 [192]

  *Cosine of day in year * 60.*

- char const dec2digs [128][2]

  Format two digit decimal, leading zero, by lookup.

- char const dec3digs [1024][4]

  Format three digit decimal, leading zero, 0-terminated, by lookup.

- char const dow [9][4]

  English weekdays, two letter abbreviation.

- FILE * errLog

  Error log output.

- char const fType [16][8]

  Translation of directory entry typed to 8 char text.

- char const *const lckPiGpioPth

  Common path to a lock file for GpIO use.

- __time_t localMidnight

  Actual local midnight.

- uint32_t noLgdEvnt

  Number of events logged.

- FILE * outLog

  Event log output.

- char const prgNamPure [ ]

  The pure program name.

- char const prgSVNdat [ ]

  The complete SVN date string.

- char const prgSVNrev [ ]

  The complete SVN revision string.

- int retCode

  Basic start-up function failure.

- int todayInYear

  Today's day in year.

- int useErrLogFiles

  Log on files.

- uint8_t useOutLog4errLog

  Use outLog for errors too.

- __time_t utcMidnight

  Actual (local) UTC midnight.

- char const zif2charMod10 [44]

  The digits 0..9 repeated as 44 characters.

7.22.1 Detailed Description

Some very basic definitions.

```
Copyright  (c)  2020   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 242 23.02.2022
Rev. 66+ 16.11.2017 : new, excerpted from weUtil.h V.66
                       weModbus.h V.66 (and others)
Rev. 147 16.06.2018 : time handling enhanced improved, Hippogreiff relay
Rev. 152 21.06.2018 : some definitions put to basicTyCo.h
Rev. 164 11.07.2018 : sunset/sunrise location parameters; string functions
Rev. 190 14.02.2019 : minor, comments only
Rev. 209 22.07.2019 : work around a Doxygen bug
Rev. 229 23.07.2020 : UTF 8 BOM for log and error file
```

This file contains some definitions concerning system values and platform properties, made and probed with Raspberry Pis.

7.22.2 Macro Definition Documentation

7.22.2.1 SETPRGDATA

```
#define SETPRGDATA
```

Frame for program data definition.

To prevent Doxygen to duplicate the documentation of extern variables, like prgNamPure, prgSVNrev, prgSVNdat etc., in the program's source frame those definition in #if SETPRGDATA ... #endif.

7.22.2.2 ABS_MONOTIME

```
#define ABS_MONOTIME
```

/def ABS_MONOTIME

Clock used for absolutely monotonic delays, cycles and intervals.

This clock must never jump and just run on in a monotonic way. We accept it
A) having no relation to any calendar date and time and
B) getting no corrections by NTP clients, DCF77 receivers or what else, as well as
C) this clock being slightly inaccurate and (cf. B)) never be tuned or corrected.

Short note on A): In most literature it is said the monotonic clocks would start at boot. Even if this is observed, it is not mandatory. Assume an arbitrary zero-point.

The inaccuracy C) is explained by some implementations deriving monotonic clocks with no further ado from an μC's quartz oscillator, usually the same oscillator used for communication links timing. On Raspberry Pi 3s with

Raspian Jessie (early 2017) we observed +5s in an 24h interval (i.e. being a bit late) growing linear. This stable deviation is in the range of mid prized quartz watches.

Until August 2017, we had adapted to C) by allowing a millisecond used for chained steps or as base for delays not having 1000000ns of this (ABS_MONOTIME) clock, allowing up to +-110ns difference. The value (vcoCorrNs) was then preset at compile by a device specific macro. Its default value -40 was good for a couple of Raspberry Pi 3s. An automatic correction of this adjusted millisecond by standard time sources (with simplified VCO PLL algorithm) was used.

Update on C) since August 2017:
In the latest Jessies (8) CLOCK_MONOTONIC is frequency adjusted to NTP. Hence B) and C) above are obsolete and vcoCorrNs will be initialised as 0. Nevertheless, this corrective +/-100 ns value vcoCorrNs is kept for catching up or slowing down the derived second tick to CLOCK_REALTIME after the latter's jumps due to corrections. As the derived (monotonic) second's tick is started synchronised with the CLOCK_REALTIME one's, this would hardly happen. On a leap second 1000s slow down (the current "solution) on a leap second, we would 1000s slow down and afterwards catch up, without getting an extra "monotonic" second.

Candidates (Raspbian lite) for an absolute monotonic clock are:
CLOCK_MONOTONIC (should always be available and OK, default)
CLOCK_MONOTONIC_RAW (same without NTP tuning)

value: CLOCK_MONOTONIC (NTP tuning now assumed)

7.22.2.3 DAY_EQNOX_SPRING

```
#define DAY_EQNOX_SPRING
```

Day of the spring equinox.

This is the day in year number (79) of the spring equinox (March 20th). From thence the time between sunrise and sunset is more than 12 hours.

See also

todayInYear
DAY_EQNOX_AUTUMN

7.22.2.4 DAY_EQNOX_AUTUMN

```
#define DAY_EQNOX_AUTUMN
```

Day of the autumn equinox.

This is the day in year number (265) of the autumn equinox (Sept. 22nd). From thence the time between sunrise and sunset is less than 12 hours. This value and DAY_EQNOX_SPRING is used for battery handling (defaults).

7.22.3 Typedef Documentation

7.22.3.1 timespec

```
typedef struct timespec timespec
```

The Linux time structure.

This is the timespec structure consisting of two (type obfuscated) long variables: tv_sec and tv_nsec.
Note: This is to allow using timespec without prepending struct, only.

7.22.4 Function Documentation

7.22.4.1 progNam()

```
char const * progNam ( )
```

The program name.

Returns

the program's name as pure text, "homeDoorPhone", e.g.

7.22.4.2 progNamB()

```
char const * progNamB ( )
```

The program name with blank.

Same as progNam but with at least one trailing blank or so many blanks to get a minimal length of 17, , "home↩ DoorPhone ", e.g.

Returns

the program's name with trailing blank(s)

7.22.4.3 progRev()

```
char const * progRev ( )
```

The program revision.

Returns

the program's SVN revision as pure text, "0", "341" e.g.

7.22.4.4 progDat()

```
char const * progDat ( )
```

The program date.

Returns

the program's SVN date "2020-07-23" e.g., length 10

7.22.4.5 printRevDat()

```
void printRevDat (
            void  )
```

Print the program SVN revision and date.

This function prints a line in the form (4 leading blanks)
`Revision  229 (2020-07-23)`

to outLog

7.22.4.6 printNamRevDat()

```
void printNamRevDat (
            void  )
```

Print the program name, SVN revision and date.

This function prints a line in the form (4 leading blanks)
`theLittleProg    R.  229 (2020-07-23)`

to outLog

7.22.4.7 littleEndian()

```
uint8_t littleEndian ( )
```

Actual runtime / architecture is little endian.

This boolean function is evaluated by char* to int comparison.

To save runtime resources use the marco PLATFlittlE instead, which would fall back to littleEndian() (this function) when no target platform informations on endianness are available.

Returns

true when platform is little endian (evaluated at run time)

7.22.4.8 isFNaN()

```
uint8_t isFNaN (
            float const val )
```

Floating point NaN.

Parameters

val	the floating point value to be checked for IEEE754 NaN

Returns

0xFF (true) when not a number, else 0

7.22.4.9 switchErrorLog()

```
int switchErrorLog (
            char const *const errFilNam )
```

Switch errlog to other file.

Parameters

errFilNam	the name of the file to switch to; NULL or empty: switch (back) to stderr

Returns

96 : file name can't be opened for append, old state kept; 97 : useOutLog4errLog is ON, nothing done 0 : OK

7.22.4.10 switchEventLog()

```
int switchEventLog (
            char const *const logFilNam )
```

Switch outLog to other file.

If useOutLog4errLog is ON the errLog file will point to the same named file on success.

Parameters

logFilNam	the name of the file to switch to; NULL or empty: switch (back) to stdout

Returns

96 : file name can't be opened for append; old state kept.

7.22.4.11 logEventText()

```
void logEventText (
            char const * txt )
```

Log an event/log message on outLog.

If txt is not null it will be output to outLog and outLog will be flushed. No line feed will be appended; the text is put as is.

Parameters

txt	text to be output; n.b not LF appended and not counted as line

7.22.4.12 strlcpy()

```
size_t strlcpy (
            char * dest,
            char const * src,
            size_t const num )
```

String copy with limit.

This function copies at most num - 1 characters from src to dst. If not terminated by a 0 from src, dest[num-1] will be set 0. Hence, except for num == 0, dest will be 0-terminated.

The value returned is the length of string src; if this value is not less than num truncation occurred.

Hint: This function resembles the one from bsd/string.h usually not available with standard Linuxes and Raspbians .

Parameters

dest	the character array to copy to; must not be shorter than num
src	the string to copy from
num	the maximum allowed string length of dest

Returns

the length of src

7.22.4.13 strlcat()

```
size_t strlcat (
            char * dest,
            char const * src,
            size_t num )
```

String concatenation with limit.

This function appends at most num - 1 characters from src to the end of dest. If not terminated by a 0 from src, dest[num-1] will be set 0. Hence, except for num == 0, dest will be 0-terminated.

The value returned is the length of string src (if no truncation occurred).

Hint: This function resembles the one from bsd/string.h usually not available with standard Linuxes and Raspbians .

Parameters

dest	the character array to copy to; must not be shorter than num
src	the string to copy from
num	the maximum allowed string length of dest

Returns

the length of src

7.22.4.14 monoTimeInit()

```
void monoTimeInit (
            timespec * timer )
```

Absolute timer initialisation.

This function sets the time structure provided to the current absolute monotonic ABS_MONOTIME (default↩ : CLOCK_MONOTONIC).

Note: Error returns, suppressed here, cannot occur, as long as the time library functions and used clock IDs are implemented. Otherwise all else timing done here would fail completely.

Parameters

timer	the time structure to be used (never NULL!)

7.22.4.15 timeStep()

```
int timeStep (
            timespec * timeSp,
            unsigned int micros )
```

A delay to an absolute step specified in number of s to a given time.

This function does an absolute monotonic real time delay until timer += micros;

Chaining this calls can give absolute triggers relative to a given start. One must initialise the time structure timespec before every start of a new cycle chain. Afterwards the structure time must not be written to. See timeAddNs, ABS_MONOTIME and monoTimeInit (or clock_gettime).

Chaining absolute delays accomplishes long term exact periods respectively cycles. See also explanations in ABS_MONOTIME (default: CLOCK_MONOTONIC).

Parameters

timeSp	the time structure to be used (never NULL!)
micros	delay in s (recommended 100s .. 1h)

Returns

sleep's return value if of interest (0: uninterrupted)

7.22.4.16 timeAddNs()

```
void timeAddNs (
            timespec * t1,
            long ns )
```

Add a ns increment to a time overwriting it.

Parameters

t1	the time structure to add to (not NULL!, will be modified)
ns	the increment in nanoseconds

7.22.4.17 updateReaLocalTime()

```
void updateReaLocalTime (
            void  )
```

Update local real time.

This function initialises / updates both actRTime and actRTm.

7.22.4.18 cosDay()

```
float cosDay (
            short int dayInYear )
```

Cosine of day in year, function.

This function provides the cosine by the day of the year very efficiently by using a lookup table (cosDiY) and cosine's periodic properties.

For the main purpose of approximate sunrise or sunset time determination the usual (approximate) algorithm relates to shortest day (23.12.). In this case add 8 to the real day in the year.

Parameters

dayInYear	the day in the year; 0: 1.1. (respectively 23.12.)

7.22.4.19 getDaySunrise()

```
__time_t getDaySunrise (
            short int const dayInYear,
            uint32_t const meanSunriseSec,
            uint16_t const halfRiseDeltaMin )
```

Get sunrise in s from UTC midnight.

The value will be approximately (but very fast) calculated on base of the the location's (and optimally year's) sunrise data.
Caveat: Consider the units and bases of the parameters.

Parameters

dayInYear	day in the year
meanSunriseSec	location's mean sunrise time in s from midnight UTC
halfRiseDeltaMin	the location's half sunrise time swing in minutes

Returns

that days's sunrise in seconds from UTC midnight

7.22.4.20 getDaySunset()

```
__time_t getDaySunset (
            short int const dayInYear,
            uint32_t const meanSunsetSec,
            uint16_t const halfSetDeltaMin )
```

Get sunset in s from UTC midnight.

The value will be approximately (but very fast) calculated on base of the the location's (and optimally year's) sunset data.
Caveat: Consider the units and bases of the parameters.

Parameters

dayInYear	day in the year (0 is January 1st)
meanSunsetSec	location's mean sunset time in s from midnight UTC
halfSetDeltaMin	the location's half sunset time swing in minutes

Returns

that days's sunset in seconds from UTC midnight

7.22.4.21 formatDec2Digs()

```
int formatDec2Digs (
            char * targTxt,
            uint32_t value )
```

Format number as two digit decimal number with leading zeroes.

The format is: 00 to 99

The length is always 2. There is no trailing character zero appended.
returned is the number of leading zeroes in the range 0 o 1. N.B. the value 0 yielding "00" is considered to have one leading zero.

See also

dec2digs formatDec3Digs

Parameters

targTxt	pointer to the target text buffer, must have place for 3 characters (!)
value	the value to be formatted; values outside 0 .. 999 will yield incorrect results

Returns

the number of leading zeroes (0 or 1)

7.22.4.22 formatDec3Digs()

```
int formatDec3Digs (
            char * targTxt,
            uint32_t value )
```

Format number as three digit decimal number with leading zeroes.

The format is: 000 to 999

The length is always 3. There is no trailing character zero appended.
returned is the number of leading zeroes in the range 0 to 2. N.B. the value 0 yielding "000" is considered to have 2 leading zeroes.

See also

dec3digs formatDec2Digs

Parameters

targTxt	pointer to the target text buffer, must have place for 3 characters (!)
value	the value to be formatted; values outside 0 .. 999 will yield incorrect results

Returns

the number of leading zeroes (0..2)

7.22.4.23 formatTmTim()

```
int formatTmTim (
            char * rTmTxt,
            struct tm * rTm )
```

Format broken down real time and date as standard text.

The format is: Fr 2017-10-20 13:55:12 UTC+20 .....0123456789x123456789v123456789t The length is 29. See formatTmTiMs() for a longer format with 3 digit ms.

Parameters

rTmTxt	pointer to the target text buffer, must have place for 30 characters (!)
rTm	pointer to broken down real time; NULL will take actRTm

Returns

the number of characters put (should be 28) or 0: error (rTmTxT NULL)

7.22.4.24 formatTmTiMs()

```
int formatTmTiMs (
            char * rTmTxt,
            struct tm * rTm,
            int millis )
```

Format broken down real time clock+ms as standard text.

/code The format is: Fr 2017-10-20 13:55:12.987 UTC+20 .....0123456789x123456789v123456789t123 +3 .....0123456789x123456789v123456789t /endcode The length is 33. See formatTmTim() for a shorter format without ms.

Parameters

rTmTxt	pointer to the target text buffer, must have place for 34 characters (!)
rTm	pointer to broken down real time; NULL will take actRTm
millis	milliseconds 0..999 supplement to rTm

Returns

the number of characters put (should be 32) or 0: error (rTmTxT NULL)

7.22.5 Variable Documentation

7.22.5.1 prgNamPure

```
char const prgNamPure[]  [extern]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.22.5.2 prgSVNrev

```
char const prgSVNrev[]  [extern]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.22.5.3 prgSVNdat

```
char const prgSVNdat[]  [extern]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.22.5.4 retCode

```
int retCode  [extern]
```

Basic start-up function failure.

Allows for compact code without saving the (error) return:
if (openLock(lckPiGpioPth, ON)) return retCode;

Storage for return/error codes. Used by: openLock(char const *, uint8_t) theCyclistStart(int) theCyclistWaitEnd()

Value: 0: OK, else: error

7.22.5.5 lckPiGpioPth

```
char const* const lckPiGpioPth  [extern]
```

Common path to a lock file for GpIO use.

Programs using GPIO in any form usually (and forced by some libraries) have to do this exclusively. This is implemented here by locking a file named /home/pi/bin/.lockPiGpio
Make the lock file by: touch /home/pi/bin/.lockPiGpio

Without locking this file those programs must not start.
So, deleting this file inhibits the start even by cron etc.

7.22.5.6 useErrLogFiles

```
int useErrLogFiles  [extern]
```

Log on files.

If true (default) logging and errors go to files or one file, otherwise to console

7.22.5.7 outLog

```
FILE* outLog  [extern]
```

Event log output.

default: standard output; may be put to a file.

7.22.5.8 noLgdEvnt

```
uint32_t noLgdEvnt  [extern]
```

Number of events logged.

Counter for lines put to or events logged on outLog.

7.22.5.9 useOutLog4errLog

```
uint8_t useOutLog4errLog  [extern]
```

Use outLog for errors too.

When set true errLog will be set to outLog when using files. In this case there is just one event log file. Hence, doubling the same entry to both errLog and outLog should be avoided.

7.22.5.10 errLog

```
FILE* errLog  [extern]
```

Error log output.

default: standard error; may be put to a file.

7.22.5.11 actRTm

```
struct tm actRTm  [extern]
```

Actual time (broken down structure / local).

This structure is initialised may be updated by some timing and cyclic functions. See initStartRTime() and others.

7.22.5.12 todayInYear

```
int todayInYear  [extern]
```

Today's day in year.

The value should be set at start (will be by updateReaLocalTime()) and updated at midnight.

7.22.5.13 utcMidnight

```
__time_t utcMidnight  [extern]
```

Actual (local) UTC midnight.

This is the actual "local" UTC midnight. "Local" means that on early hours, i.e. those within zone offset, UTC midnight will be corrected to the next (east of Greenwich) respectively previous day (west). The rationale is to point to the same day or date at day time (around Europe).
Or, to put it simple, utcMidnight is to be set so, that the equation
local Midnight = utcMidnight - UTV offset
holds.

The value will will be set correctly by updateReaLocalTime(). It should be updated at day change (if used).

7.22.5.14 localMidnight

```
__time_t localMidnight  [extern]
```

Actual local midnight.

This is the UTC Linux timestamp of the actual day's / time's local midnight. See also utcMidnight for explanations.

Note: On days with DST changes this value will shift within the day. It's mostly better to make calculations relative to day start — sunrise and sunset e.g. — relative to UTC midnight.

7.22.5.15 cosDiY

```
float const cosDiY[192]  [extern]
```

Cosine of day in year, look up.

This lookup table provides the cosine by the day of the year without resource eating floating point arithmetic or math.h. The rationale is the approximate calculation of sunrise and sunset times based on earliest, latest and delta for any given location within the arctic circles.

The length of the look up table is abundant 192. According to cosine's periodic properties it shall be used in the range 0..183 by applying the following operations to the day in year value
absolute when $<$ 0,
modulo FOURYEARS when $>=$ FOURYEARS,
modulo 365 when $>=$ 365 and
x = 365 - x when $>$ 190.

Note: These rules are implemented in the function cosDay() and in the function cosDay60() using the look up table cosDiY60

7.22.5.16 cosDiY60

```
short int const cosDiY60[192]  [extern]
```

Cosine of day in year $*$ 60.

This look up table is the same as cosDiY, except the values being multiplied by 60 which includes minutes to seconds conversion, avoiding a multiplication and all floating point operations for some applications.

lenght: 192

7.22.5.17 zif2charMod10

```
char const zif2charMod10[44]  [extern]
```

The digits 0..9 repeated as 44 characters.

By using a number 0..43 as index this will give modulo 10 respectively the last decimal digit as character.

7.22.5.18 dec2digs

```
char const dec2digs[128][2]  [extern]
```

Format two digit decimal, leading zero, by lookup.

"00" .. "99" + "00", "_1" .. "_7"

7.22.5.19 dec3digs

```
char const dec3digs[1024][4]  [extern]
```

Format three digit decimal, leading zero, 0-terminated, by lookup.

"000" .. "999" + "000" .. "023"

7.22.5.20 dow

```
char const dow[9][4]  [extern]
```

English weekdays, two letter abbreviation.

Monday (Mo) is 1; Sunday (Su) is 7 or, also, 0.

7.22.5.21 fType

```
char const fType[16][8]  [extern]
```

Translation of directory entry typed to 8 char text.

direntry.d_type as index in the range 0..15 gives an 8 character short type text. Note: Only 0, 1, 2, 4, 6, 8, 10, 12 and 14 are defined d_type values. The undefined ones give undef3 .. undefF

7.23 include/trafficPi.h File Reference

Common types and values for the trafficPi extension (shield).

```
#include <stdint.h>
#include "weGPIOd.h"
```

Macros

- #define **Button**

  central buzzer 1-active
- #define **Buzzer**

  central buzzer 1-active
- #define **GNeast**

  green light east
- #define **GNnorth**

  green light north
- #define **GNsouth**

  green light south
- #define **GNwest**

  green light west (ye rdGnPiGpioDBlink)
- #define OUTPtraffRdGnYE

  The list of traffic lights used by rdGnBlink programs.
- #define OUTPtraffWOrdGnYe

  The list of all (9) traffic lights except those for rdGnBlink programmes.
- #define **RDeast**

  red light east
- #define **RDnorth**

  red light north
- #define **RDsouth**

  red light south (rd rdGnPiGpioDBlink)
- #define **RDwest**

  red light west
- #define **YEeast**

  yellow light east
- #define **YEnorth**

  yellow light north
- #define **YEsouth**

  yellow light south (gn rdGnPiGpioDBlink)
- #define **YEwest**

  yellow light west

7.23.1 Detailed Description

Common types and values for the trafficPi extension (shield).

```
Copyright  (c)  2020   Albrecht Weinert
weinert-automation.de        a-weinert.de
    /         /      /\
   /         /___    /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /      \|   |
     \/  \/   \__/        \__/|_
```

Revision history

```
Rev. 241 10.08.2021
Rev. 227 11.08.2020 : new
```

7.23.2 Macro Definition Documentation

7.23.2.1 OUTPtraffWOrdGnYe

```
#define OUTPtraffWOrdGnYe
```

The list of all (9) traffic lights except those for rdGnBlink programmes.

This is the list of 6 out of 9 traffic lights except the three used by the rdGnBlink programs like rdGnPiGpioDBlink and others. Those other three are listed in OUTPtraffRdGnYE

7.23.2.2 OUTPtraffRdGnYE

```
#define OUTPtraffRdGnYE
```

The list of traffic lights used by rdGnBlink programs.

RDsouth, YEsouth and GNwest use the same pins as rdGnPiGpioDBlink and others for their red green and yellow LEDs.
All others are listed in OUTPtraffWOrdGnYe

Rationale: The piTraffic shield might be used with the traditional rdGdBlink demo/test programs, also. For other programs, wanting to avoid interference with those old programs the two lists are separated. Get the complete list by concatenating OUTPtraffWOrdGnYe, OUTPtraffRdGnYE , he list is kept separate.Demos

7.24 include/we1wire.h File Reference

Common types and values for 1-wire sensors.

```
#include "sysBasic.h"
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
```

Data Structures

- struct oneWireDevice_t

  A structure for 1-wire devices.

Macros

- #define BAD_TEMP_FLOAT

  Bad temperature reading as float string.
- #define BAD_TEMP_READ

  Bad temperature reading.
- #define STD_DEVICES_PATH

  The standard devices location.
- #define STD_DEVICES_PATH_LEN

  Standard 1-wire device path length.

Functions

- int getTemp (oneWireDevice_t *const tempSensor)

  Get temperature.
- void initTempSensor (oneWireDevice_t *const tempSensor, char const *const name, char const *const valueFile)

  Initialise a 1-wire sensor structure.

7.24.1 Detailed Description

Common types and values for 1-wire sensors.

```
Copyright  (c)  2018, 2020   Albrecht Weinert
weinert-automation.de        a-weinert.de
    /          /      /\
   /          /___   /  \       |
   \         /____\ /____\ |  _|__
    \  /\  / \     /       \|   |
     \/  \/   \__/          \__/|_
```

Revision history

```
Rev. 249 30.06.2023
Rev. 168 21.07.2018 : new
Rev. 190 12.02.2019 : safe water temperature reduced to 65C
Rev. 222 18.03.2020 : safe water temperature set to 69C
```

1-wire devices and especially 1-wire temperature sensors are an attractive way to enrich Raspberry's process I/O repertoire. Up to some dozen devices can share one digital I/O pin; the default is GPIO04 (Pin7).

In most Linuxes and in Raspbian 1-wire devices are accessed by Linux's "device as file" approach. The behaviour is being implemented in kernel modules (i.e. .so files) which must be activated in OS's configuration files. Access to a 1-wire device's I/O values, configuration data etc. is done by a) opening, b) reading from respectively writing to and, finally, c) closing the corresponding (pseudo) file. The sheer number of files for a small un-complicated device and the content semantics seam neither clearly designed nor documented. One order principle is to have those pseudo files in one directory named according to 1-wire devices' unique readable serial number.

As the I/O pin used for 1-wire (default GPIO04, Pin7) is governed by those kernel modules no other process control software must touch this port directly.

This include file introduces some types and functions to handle 1-wire.

7.24.2 Macro Definition Documentation

7.24.2.1 STD_DEVICES_PATH

```
#define STD_DEVICES_PATH
```

The standard devices location.

This is the directory wer each device's sub-directory is located.
Value: /sys/bus/w1/devices/

7.24.2.2 STD_DEVICES_PATH_LEN

```
#define STD_DEVICES_PATH_LEN
```

Standard 1-wire device path length.

This is the length of "/sys/bus/w1/devices/" without the trailing '/' respectively the index of that trailing slash.

This path length +1 would be the index of the device directory name "28-..." in "/sys/bus/w1/devices/28-01e63c07010c/".

This path length +15 — i.e. + STD_DEVICES_PATH_LEN — would be the end of the concrete device's directory path respectively the index of its trailing respectively separating slash.

7.24.2.3 BAD_TEMP_READ

```
#define BAD_TEMP_READ
```

Bad temperature reading.

The integer value for bad temperature sensor reading is +987650 m℃ respectively +987.650 ℃ far beyond the 1-wire temperature sensors range.

Rationale for choosing a (too) high value as bad value: The usual and safe "turn XYZ off" when temperature exceeds xyC" requires no special treatment for sensor outage.

7.24.2.4 BAD_TEMP_FLOAT

```
#define BAD_TEMP_FLOAT
```

Bad temperature reading as float string.

The integer value for bad temperature sensor reading as ℃ string is "987.6 " (float string, 6 characters including trailing blank.

7.24.3 Function Documentation

7.24.3.1 getTemp()

```
int getTemp (
            oneWireDevice_t *const tempSensor )
```

Get temperature.

This function tries a new measurement on the 1-wire temperature sensor provided. On success the new value is returned. On failure the last good value is returned, but at most 7 times after 7 good readings before. On total failure -99900 (-99.9 ℃, 0 K, BAD_TEMP_READ) is returned.

Additionally on temperature changes, the integer reading (.value) and the floating point string (valueGrdC[]) will be set in the structure.

Parameters

tempSensor	pointer to the sensor's structure

Returns

the actual or last reading or -2721500 if no good / not enough past good readings

7.24.3.2 initTempSensor()

```
void initTempSensor (
            oneWireDevice_t *const tempSensor,
            char const *const name,
            char const *const valueFile )
```

Initialise a 1-wire sensor structure.

This function does basic initial settings for a 1-wire temperature sensor. tempSensor.name and tempSensor.value↩File are set by the respective parameters.
The temperature values are set to bad value; see BAD_TEMP_READ and BAD_TEMP_FLOAT

Parameters

tempSensor	pointer to the sensor's structure (never NULL!)
name	the sensor's short name or its directory name; NULL / empty: no change
valueFile	the canonical absolute path to its value file; NULL / empty: no change

7.25 include/weAR_N4105.h File Reference

Definitions for VDE-AR-N 4105.

```
#include "sysBasic.h"
#include "weStateM.h"
```

Macros

- #define **L1Msk**

  arn4105state L1 voltage mask
- #define **L2Msk**

  arn4105state L2 voltage mask
- #define **L3Msk**

  arn4105state L3 voltage mask
- #define **LeMsk**

  arn4105state line status error mask

- #define **LfMsk**

  arn4105state line frequency mask
- #define **LwMsk**

  arn4105state line status warning mask
- #define newLineFcheck(N, C)

  Define a five band checker for line frequency.
- #define newLineUcheck(N, C)

  Define a five band checker for line voltage.

Functions

- void setARN4105state (uint8_t const state, uint8_t const select)

  Set AR-N 4105 frequency and voltage states.
- void switchARN4105 (uint8_t const cutOff)

  AR-N 4105 implementation function.

Variables

- uint8_t arn4105state

  AR-N 4105 frequency and voltage states.
- state_t arn4105Timer

  AR-N 4105 control timer.
- state_t checkL1_U

  The (one) five band checker for line L1 voltage.
- state_t checkL2_U

  The (one) five band checker for line L2 voltage.
- state_t checkL3_U

  The (one) five band checker for line L3 voltage.
- state_t checkLineFrq

  The (one) five band checker for line frequency.

7.25.1 Detailed Description

Definitions for VDE-AR-N 4105.

```
Copyright  (c)  2019   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /          /      /\
   /          /___   /  \       |
   \         /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 250 10.07.2023
Rev. 195 01.03.2019 : new
Rev. 249 26.06.2023 : No more relays assigned for ARN4105 cut off
```

7.25.2 Macro Definition Documentation

7.25.2.1 newLineFcheck

```
#define newLineFcheck(
              N,
              C )
```

Define a five band checker for line frequency.

A state machine of this type is to be fed with the actual line frequency (nominal 50 Hz) as an analogue (float) value sampled regularly. This value is compared to four thresholds namely
47.7 49.5 -OK- 50.5 51.5 Hz
defined by VDE-AR-N 4105 as good / warning / bad criteria for the grid frequency separating those five bands:
...badLO | critLo | OK | critHi | badHi...

On badHi or badLo distributed small generators must be cut off reliably from public power lines. For being switched on to the grid at OK often at least 10 minutes out of bad ist required.

For state values and parameters see newFiveBand

This macro is the initialisation expression for a five band checker requiring the unique name and the fitting on↩ StateChange function.

Parameters

N	name as string literal
C	onStateChange function

7.25.2.2 newLineUcheck

```
#define newLineUcheck(
              N,
              C )
```

Define a five band checker for line voltage.

A state machine of this type is to be fed with the actual line voltage (nominal 230 V) as an analogue (float) value sampled regularly. This value is compared to four thresholds namely
184 207 -OK- 253 264 V
defined by VDE-AR-N 4105 as good / warning / bad criteria for the grid frequency separating those five bands:
...badLO | critLo | OK | critHi | badHi...

On badHi or badLo distributed small generators must be cut off reliably from public power lines. For being switched on to the grid at OK often at least 10 minutes out of bad ist required.

For state values and parameters see newFiveBand

This macro is the initialisation expression for a five band checker requiring the unique name and the fitting on↩ StateChange function.

Parameters

N	name as string literal
C	onStateChange function

7.25.3 Function Documentation

7.25.3.1 setARN4105state()

```
void setARN4105state (
            uint8_t const state,
            uint8_t const select )
```

Set AR-N 4105 frequency and voltage states.

This function is called internally by the respective five band checker's state_t.onStateChange function.

Parameter selection mask must be one of LfMsk, L1Msk - L3Msk.

Parameters

state	state_t.status of the five band checker in question
select	selection mask

7.25.3.2 switchARN4105()

```
void switchARN4105 (
            uint8_t const cutOff )
```

AR-N 4105 implementation function.

This is the VDE-AR-N 4105 cut off function to be provided site specific by user / application software. It does the cut off respectively switch back on the generators and optionally log the event.

Parameters

cutOff	not 0: do the cut off, 0: switch generators back on

7.25.4 Variable Documentation

7.25.4.1 checkLineFrq

```
state_t checkLineFrq  [extern]
```

The (one) five band checker for line frequency.

See newLineFcheck

7.25.4.2 checkL1_U

```
state_t checkL1_U  [extern]
```

The (one) five band checker for line L1 voltage.

In a one phase system this will be the only one fed with values by fiveBandTick.

See newLineUcheck

7.25.4.3 checkL2_U

```
state_t checkL2_U  [extern]
```

The (one) five band checker for line L2 voltage.

See newLineUcheck

7.25.4.4 checkL3_U

```
state_t checkL3_U  [extern]
```

The (one) five band checker for line L3 voltage.

See newLineUcheck

7.25.4.5 arn4105state

```
uint8_t arn4105state  [extern]
```

AR-N 4105 frequency and voltage states.

If all is OK this status byte is 0. The meaning of bits set is

```
|   7    |   6    |   5    |   4    ||   3    |   2    |   1    |   0    |
|   L3   |   L2   |   L1   |   f    ||   L3   |   L2   |   L1   |   f    |
|       voltage error       |  error ||      voltage warning     | warning|
```

If any error bit is set the respective generators has to be cut off.
Ten minutes after the last warning bit is gone the respective generators may be put back to power line.
The timer arn4105Timer will be handled accordingly.

User software should not touch this variable (except when knowing the consequences.

7.25.4.6 arn4105Timer

```
state_t arn4105Timer  [extern]
```

AR-N 4105 control timer.

When this timer is running distributed small generators must be cut off. When this timer stops the respective generators may be put back to power line.

User software must provide a function switchARN4105 doing the cutoff and switch back plus optionally logging. It will be called by this timer and, hence, in the end by the frequency and voltage checkers.

The user/application software must check (timerTickCheck) regularly to enable switch back.

7.26 include/weBatt.h File Reference

Common types and values for buffer battery handling.

```
#include "basicTyCo.h"
```

Macros

- #define BAT100M_STRESS_LO_LIM

  Lower voltage limit for undervoltage stress.
- #define BAT100M_STRESS_UNL_LIM

  Lower voltage limit for undervoltage stress, when battery unload.
- #define BAT100M_STRESS_UP_LIM

  Upper voltage limit for overvoltage stress.
- #define BAT100MVS_HI_LIM

  Lower voltage limit for accumulated overvoltage stress.
- #define BAT100MVS_LO_LIM

  Lower voltage limit for accumulated undervoltage stress.
- #define BATkeepInhBalLoad

  Inhibit battery keep loading after ballast loading.
- #define BATkeepInhContLoad

  Inhibit battery keep loading after controlled (full) loading.
- #define BATkeepInhOffCmd

  Inhibit battery keep loading after battery Off command.
- #define BATkeepInhUnload

  Inhibit battery keep loading after unload.
- #define BATmaxUNLODtime

  Upper time limit to stop discharging.
- #define BATVOLT_IDLE_100

  Voltage 100% loaded.
- #define BATVOLT_IDLE_20

  Voltage 20% loaded.
- #define BATVOLT_IDLE_50

  Voltage 50% loaded.
- #define BATVOLT_IDLE_60

Voltage 60% loaded.

- #define BATVOLT_IDLE_70

  Voltage 70% loaded.

- #define BATVOLT_IDLE_80

  Voltage 80% loaded.

- #define BATVOLT_LO_LIM_UNLD

  Voltage limit to start unload.

- #define BATVOLT_LO_STP_UNLD

  Lower voltage limit to stop discharging.

- #define BATVOLT_UP_LIM_BLST

  Upper voltage limit to stop ballast loading.

- #define BATVOLT_UP_LIM_LOAD

  Upper voltage limit to stop normal loading.

- #define SOLmxPOW4unload

  Maximum solar power to stop discharging.

7.26.1 Detailed Description

Common types and values for buffer battery handling.

```
Copyright (c) 2018 Albrecht Weinert
weinert-automation.de a-weinert.de
```

Revision history

```
Rev. 250 10.07.2023
Rev. 159 04.07.2018 : new
Rev. 191 15.02.2019 : comment text changes; upper ballast limit reduced
Rev. 199 02.04.2019 : battery as ballast power and voltage reduced
Rev. 200 16.04.2019 : battery keep inhibit until low stress
Rev. 204 15.05.2019 : log battery stress (%) repaired
Rev. 223 24.04.2020 : comment clarifications, voltage adjustments
```

In the smart home laboratory experiment buffer batteries are involved. If not otherwise stated, we deal with 12V 6 cell lead acid liquid electrolyte types.

In the concrete experimental home control setup we have 12V 200Ah featuring a buffer for about 1 kWh (usable). Battery voltage is measured directly at the battery clamps by a MQTT device.

7.26.2 Macro Definition Documentation

7.26.2.1 BATVOLT_IDLE_100

```
#define BATVOLT_IDLE_100
```

Voltage 100% loaded.

When this or higher voltage is measured after some minutes idle time the battery is considered fully (100%) loaded. Idle means no considerable ($<$ 0.01 Ah/h) load nor discharge current.

7.26.2.2 BATVOLT_IDLE_80

```
#define BATVOLT_IDLE_80
```

Voltage 80% loaded.

When this voltage is measured after some minutes idle time the battery is considered at least 80% loaded. This is considered the minimal permanent state. Hence if lower for longer periods ($>$ 2 days?), do load.

See also

BATVOLT_IDLE_100

7.26.2.3 BATVOLT_IDLE_70

```
#define BATVOLT_IDLE_70
```

Voltage 70% loaded.

When this or higher voltage is measured after some minutes idle time the battery is considered at least 70% loaded. This is considered as an acceptable state for short periods ($>$ 2 days?).

See also

BATVOLT_IDLE_80

7.26.2.4 BATVOLT_IDLE_60

```
#define BATVOLT_IDLE_60
```

Voltage 60% loaded.

When this or higher voltage is measured after some minutes idle time the battery is considered at least 60% loaded. This is considered as an acceptable state for short periods ($>$ 2 days?).

See also

BATVOLT_IDLE_80

7.26.2.5 BATVOLT_IDLE_50

```
#define BATVOLT_IDLE_50
```

Voltage 50% loaded.

When this or higher voltage is measured after some minutes idle time the battery is considered at least 50% loaded. This is considered as an acceptable state for quite short periods (few hours). Load to at least BATVOLT_IDLE_60 as soon as possible.

See also

BATVOLT_IDLE_80

7.26.2.6 BATVOLT_IDLE_20

```
#define BATVOLT_IDLE_20
```

Voltage 20% loaded.

When this or higher voltage is measured after some minutes idle time the battery is considered at least 20% loaded. This is considered as the absolute lower limit for discharging and not causing battery damage. Do load immediately.

See also

BATVOLT_IDLE_80

7.26.2.7 BATVOLT_LO_LIM_UNLD

```
#define BATVOLT_LO_LIM_UNLD
```

Voltage limit to start unload.

This is the absolute minimal voltage, measured under 1cap load, to start an unload sequence for providing buffer capacity needed.
The older the battery the higher the value should be chosen.

See also

BATVOLT_IDLE_50 BATVOLT_IDLE_80

7.26.2.8 BATVOLT_LO_STP_UNLD

```
#define BATVOLT_LO_STP_UNLD
```

Lower voltage limit to stop discharging.

This is the minimal voltage, measured under 5..10cap unload, to stop a running unload sequence for providing buffer capacity needed.

This is a critical value. On one hand the cutoff voltage (German Entladeschlussspannung) is found in literature to be 10.5V. On the other hand the battery should be kept above BATVOLT_IDLE_50, i.e. 12.20V.

Choosing a value too high reduces the buffer capacity (the battery's purpose in life). A value too low endangers the battery. No sense makes a low value leading to automatic rescue charging after stopping the discharge SFC.

On the other hand: a) 8s after stopping unload the voltage is 650mV higher and b) the step up inverters low voltage protection was set to about 10,8V. Its turnoff would give the battery 5min recovery eventually allowing to proceed with discharge. hence, we reduced this limit by 500mV to 10,4V

See also

BATVOLT_IDLE_20 BATVOLT_IDLE_80 BAT_KEEP_PWM

7.26.2.9 BATmaxUNLODtime

```
#define BATmaxUNLODtime
```

Upper time limit to stop discharging.

The normal discharge end condition in the unload SFC would be battery voltage (BATVOLT_LO_STP_UNLD). Additionally there is this time limit.
Rationale for value: In the 2020 setup the discharge power is about 120W meaning about -10A. Additionally we have -0.3..1A by permanently battery attached loads. With nominally over 250Ah a limit well over 10h would be feasible. 5h seems a good compromise (for twice unload in summer as well as one in winter).

7.26.2.10 SOLmxPOW4unload

```
#define SOLmxPOW4unload
```

Maximum solar power to stop discharging.

If the solar power generation is above this limit battery unloading is stopped respectively inhibited. The current value is 28 W.
Rationale: Battery unloading uses the same inverter as the smallest (mimiJoule) panel array. We will not have lighted panels disconnected (idle) nor switch them under load.
Note: If the two other panel arrays produce 30 W or less miniJoule produces nothing and the array's idle voltage is $<$ 10V.

7.26.2.11 BATVOLT_UP_LIM_LOAD

```
#define BATVOLT_UP_LIM_LOAD
```

Upper voltage limit to stop normal loading.

The normal (gentle, intelligent) battery loading SFC would stop when this voltage is reached.

7.26.2.12 BATVOLT_UP_LIM_BLST

```
#define BATVOLT_UP_LIM_BLST
```

Upper voltage limit to stop ballast loading.

The ballast (surplus power driven) battery loading SFC would stop when this voltage is reached.
Note: With suitable load current measurements and Ri knowledge this value could be made flexibly higher. The highest value used in the setup was 14.59 V (until 14.08.2018).
Note 2: On the other hand, since having water heating ballast load there's no need to stress the battery with surplus power.

7.26.2.13 BAT100M_STRESS_UP_LIM

```
#define BAT100M_STRESS_UP_LIM
```

Upper voltage limit for overvoltage stress.

The value is in units of 1/100V = 10mV. The actual value is 13.6 V.

7.26.2.14 BAT100MVS_HI_LIM

```
#define BAT100MVS_HI_LIM
```

Lower voltage limit for accumulated overvoltage stress.

When the accumulated higher voltage stress exceeds this limit, battery loading respectively using the battery as ballast for surplus power should be stopped.

The value is in units of 1/100Vs = 10mVs. The actual value is 40000.0 Vs.

7.26.2.15 BAT100M_STRESS_LO_LIM

```
#define BAT100M_STRESS_LO_LIM
```

Lower voltage limit for undervoltage stress.

The value is in units of 100mV. The actual value is 12.1 V.

7.26.2.16 BAT100M_STRESS_UNL_LIM

```
#define BAT100M_STRESS_UNL_LIM
```

Lower voltage limit for undervoltage stress, when battery unload.

The value is in units of 1/100V = 10mV. The actual value is 11.8 V.

7.26.2.17 BAT100MVS_LO_LIM

```
#define BAT100MVS_LO_LIM
```

Lower voltage limit for accumulated undervoltage stress.

When the accumulated lower voltage stress exceeds this limit, battery unloading respectively substantial loads should be stopped.

The value is in units of 1/100Vs = 10mVs. The actual value is 1880.0 Vs.

7.26.2.18 BATkeepInhUnload

```
#define BATkeepInhUnload
```

Inhibit battery keep loading after unload.

The current value is 3 hours. The inhibit timer would be set unconditionallyIf to now + BATkeepInhUnload.

7.26.2.19 BATkeepInhBalLoad

```
#define BATkeepInhBalLoad
```

Inhibit battery keep loading after ballast loading.

The current value is 12 h from now.

7.26.2.20 BATkeepInhContLoad

```
#define BATkeepInhContLoad
```

Inhibit battery keep loading after controlled (full) loading.

The current value is 8 hours from now.

7.26.2.21 BATkeepInhOffCmd

```
#define BATkeepInhOffCmd
```

Inhibit battery keep loading after battery Off command.

The current value is 2,5 hours from now. If the inhibit timer is running already its end time will eventually be prolonged but not shortened.

7.27 include/weCGIajax.h File Reference

Types, functions and values for Web interfaces with AJAX, CGI etc.

```
#include "weUtil.h"
```

Macros

- #define **QS_MAX_KEY_LEN**

  key's / name's max. length including final zero
- #define **QS_MAX_STRING_LEN**

  Query string's maximum length including final zero.
- #define **QS_MAX_VAL_LEN**

  key's / name's max. length including final zero

Functions

- int getQSparam (char const ∗name, char ∗const value, int const vLen)

  Get the query string parameter value for a unique and known key.
- int getQueryString ()

  Fetch and store the query string.
- int jsonBinForm (uint8_t const indent, char const ∗const name, uint8_t const value, char ∗end)

  Output an eight bit value binary formated as JSON name:value.
- int jsonBreading (uint8_t const indent, char const ∗const name, uint8_t const value, char ∗end)

  Output a boolean value as JSON name:value.
- int jsonFPreadingT (uint8_t const indent, char const ∗const name, uint16_t const valueLo, uint16_t const valueHi, uint8_t dotPos, char const ∗const unit, char ∗end, char ∗title)

  Output a physical value as JSON {object} with name, 32 bit FP value and unit.
- int jsonFreading (uint8_t const indent, char const ∗const name, float const value, char const ∗const unit, uint8_t const fractional, char ∗end)

  Output a float (physical value as JSON {object} with name, value and unit.
- int jsonFreadingT (uint8_t const indent, char const ∗const name, float const value, char const ∗const unit, uint8_t const fractional, char ∗end, char ∗title)

  Output a float value as JSON {object} with name, value, unit and help text.
- int jsonIreading (uint8_t const indent, char const ∗const name, int const value, char ∗end)

  Output an int value as JSON name:value.
- int jsonSreadingT (uint8_t const indent, char const ∗const name, char ∗const value, char const ∗const unit, char ∗end, char ∗title)

  Output a physical value as JSON {object} with name, value as String and unit.

Variables

- char **actQSPvalue** [64]

  to be used for actual parameter
- char **queryString** [512]

  holding the query string
- char const ∗ **requestMethod**

  to hold (environment's) REQUEST_METHOD

7.27.1 Detailed Description

Types, functions and values for Web interfaces with AJAX, CGI etc.

```
Copyright  (c)  2018   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /      \|   |
     \/  \/   \__/        \__/|_
```

Revision history

```
Rev. 236 2.02.2021
Rev.  82 14.12.2017 : new
Rev. 187 14.10.2018 : minor typos
```

In process control applications featuring a web HMI one approach is to use a CGI program communicating with control program by shared memory and semaphores. Both programs and their libraries are written in C. The CGI program is made known to / runnable by the web sever – Apache 2.4 here.

The HMI is a HTML web page using CSS, Javascript etc. communicating with the CGI program by AJAX. The answers (asynchronously) returned may be plain text, plain HTML XML or, preferably JSON.

7.27.2 Function Documentation

7.27.2.1 getQueryString()

```
int getQueryString ( )
```

Fetch and store the query string.

This function gets the raw query string storing it (on success) in queryString. Raw means, all the ugly '+' and 'AB' things are still there.

Returns

0: from get; 1: from get, truncated; 2: from post; 3: from get, truncated; -1: program probably not run as CGI, no query string

7.27.2.2 getQSparam()

```
int getQSparam (
            char const * name,
            char *const value,
            int const vLen )
```

Get the query string parameter value for a unique and known key.

This simple method would fetch the value to a known query string parameter.

Warning: Longer names/keys with the same prefix must come first. When having "carlength=91&length=300", this (simple) function will get "91" for "length".

Parameters

name	the parameter's name or key
value	character array (string) to store the value to, the result may be empty (on name\0, name=\0 or name&)
vLen	value's length including the terminating 0

Returns

0: OK, got value; -1: got no value (no name, no value, ...) 1: got value truncated to vLen -1 characters

7.27.2.3 jsonIreading()

```
int jsonIreading (
            uint8_t const indent,
            char const *const name,
            int const value,
            char * end )
```

Output an int value as JSON name:value.

This function puts an integer value as "name": "999" respectively "name": "-1" to the standard output.

The text end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " " ",\n" "\n") may enhance human readability.

Parameters

indent	0..36 number of spaces to put before
name	the property's name
value	the integer value (put as string)
end	0-terminated string to put at the end

Returns

>=0: OK; <0: error

7.27.2.4 jsonBreading()

```
int jsonBreading (
            uint8_t const indent,
            char const *const name,
            uint8_t const value,
            char * end )
```

Output a boolean value as JSON name:value.

This function puts a boolean value as "name": true or as "name": false to the standard output.

The text end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " " ",\n" "\n") may enhance human readability.

Parameters

indent	0..36 number of spaces to put before
name	the property's name
value	0: false; else: true
end	0-terminated string to put at the end

Returns

>=0: OK; <0: error

7.27.2.5 jsonBinForm()

```
int jsonBinForm (
            uint8_t const indent,
            char const *const name,
            uint8_t const value,
            char * end )
```

Output an eight bit value binary formated as JSON name:value.

This function puts a boolean value as "name": "1001_1100" to the standard output.

The text end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " " ",\n" "\n") may enhance human readability.

Parameters

indent	0..36 number of spaces to put before
name	the property's name
value	0..255 i.e 0000_0000 .. 1111_1111
end	0-terminated string to put at the end

Returns

>=0: OK; <0: error

7.27.2.6 jsonFreading()

```
int jsonFreading (
            uint8_t const indent,
            char const *const name,
            float const value,
            char const *const unit,
```

```
            uint8_t const fractional,
            char * end )
```

Output a float (physical value as JSON {object} with name, value and unit.

This function puts a float value as e.g. {"name": "Wimp", "value": "15.22", "unit": "kWh"}

The text by parameter end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " ",\n" "\n") may enhance human readability.

Parameters

indent	0..36 number of spaces to put before
name	0-terminated string naming the reading (measured value e.g.)
value	the float value
unit	0-terminated string being the value's physical unit
fractional	0..9 number of fractional digits 0: value will be taken as int
end	0-terminated string to put at the end

Returns

>=0: OK; <0: error

7.27.2.7 jsonFreadingT()

```
int jsonFreadingT (
            uint8_t const indent,
            char const *const name,
            float const value,
            char const *const unit,
            uint8_t const fractional,
            char * end,
            char * title )
```

Output a float value as JSON {object} with name, value, unit and help text.

This function puts a float value as e.g. {"name": "Wimp", "value": "15.22", "unit": "kWh"}

The text by parameter end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " ",\n" "\n") may enhance human readability.

The text by parameter title will be used as such, acting as hover help text (not title) by html tradition. This text must not be NULL nor empty. When wanting no help text use jsonFreading() instead.

Parameters

indent	0..36 number of spaces to put before
name	0-terminated string naming the reading (measured value e.g.)
value	the float value
unit	0-terminated string being the value's physical unit
fractional	0..9 number of fractional digits 0: value will be taken as int
end	0-terminated string to put at the end
title	0-terminated string as help text

Returns

>=0: OK; <0: error

7.27.2.8 jsonSreadingT()

```
int jsonSreadingT (
            uint8_t const indent,
            char const *const name,
            char *const value,
            char const *const unit,
            char * end,
            char * title )
```

Output a physical value as JSON {object} with name, value as String and unit.

This function puts a value given as String "15.22" e.g. in the form {"name": "Wimp", "value": "15.22", "unit": "kWh"}

The text end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " " ",\n" "\n") may enhance human readability.

Parameters

indent	0..36 number of spaces to put before
name	0-terminated string naming the reading (measured value e.g.)
value	the float value
unit	0-terminated string being the value's physical unit
end	0-terminated string to put at the end
title	0-terminated string as help text (NULL no title)

Returns

>=0: OK; <0: error

7.27.2.9 jsonFPreadingT()

```
int jsonFPreadingT (
            uint8_t const indent,
            char const *const name,
            uint16_t const valueLo,
            uint16_t const valueHi,
            uint8_t dotPos,
            char const *const unit,
            char * end,
            char * title )
```

Output a physical value as JSON {object} with name, 32 bit FP value and unit.

This function puts a value given as 32 bit fixed point value in two 16 bit parts in the form {"name": "Wimp", "value": "15.22", "unit": "kWh"}. Accepting the vale as two parts solves problems with 32 bit integers or fixed point values delivered as two 16 bit Modbus registers in arbitrary order. The value is specified by the parameters valueLo, valueHi and dotPos.
With valueHi == 0 valueLo is considered as 16 bit value

The text end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " " ",\n" "\n") may enhance human readability.

Parameters

indent	0..36 number of spaces to put before
name	0-terminated string naming the reading (measured value e.g.)
valueLo	the least significant 16 bits of the fixed point value
valueHi	the most significant 16 bits of the fixed point value
dotPos	position where the fixed point is 0..6; 0 means integer
unit	0-terminated string being the value's physical unit
end	0-terminated string to put at the end
title	0-terminated string as help text (NULL no title)

Returns

>=0: OK; <0: error

7.28 include/weDCF77.h File Reference

DCF77 decoder on Raspberry Pi.

```
#include <basicTyCo.h>
#include "weGPIOd.h"
#include "weUtil.h"
```

Data Structures

- struct dcf77recPerData_t

  Data for one received DCF77 AM period.
- struct durDiscrPointData_t

  Values for discrimination of duration.

Macros

- #define DCF77inpDEF

  Default GPIO for the DCF77 receiver's AM signal input.
- #define DCF77recCNTdef

  Recommended default GPIO for the receiver control output.
- #define DCF77RINGbufWRAP

  Ring buffer maximum Index for modulation period data received.
- #define **MXUI32**

  Maximum value for uint32_t.

Functions

- void dcf77receiveRec (int pi, unsigned gpio, unsigned level, uint32_t tick)

  DCF77 receive recorder.
- int dcf77receiveRecDeregister (void)

  DCF77 receive function de-registration.
- int dcf77receiveRecRegister (void)

  DCF77 receive recorder registration.
- durDiscrPointData_t * disc5 (durDiscrPointData_t table[ ], uint32_t const value)

  Discriminating a value.
- uint32_t initDCF77io ()

  Initialise the DCF77 signal input GPIO / pin.
- void setReceiver (int const level)

  Set receiver On control.

Variables

- unsigned curBCDnum

  Number decoded from period sequence parts.
- int **dcf77callbackID**

  PiGpioD's call back ID for receiver function.
- unsigned dcf77glitch

  DCF77 input's glitch filter time setting.
- unsigned dcf77inp

  Input GPIO for the DCF77 receiver's AM signal.
- unsigned dcf77invInp

  Inverted DCF77 receiver's signal.
- unsigned dcf77lastLevel

  Last DCF77 modulation level.
- unsigned dcf77PUD

  DCF77 input's pull resistor setting.
- unsigned dcf77recCnt

  Control output GPIO of the DCF77 receiver's control.
- unsigned dcf77recCntInv

  Receiver On control inverted.
- dcf77recPerData_t **dfc77actRecPer**

  The actual respectively last modulation period data received.
- dcf77recPerData_t **dfc77ringBrecPer** [ ]

  Ring buffer of modulation period data received.
- uint8_t **dfc77ringBrecWInd**

  Modulation period data received ring buffer write index.
- unsigned const **num02st** [5]

  see numBCDinit, values 0 2
- unsigned const **num04st** [5]

  see numBCDinit, values 0 4
- unsigned const **num08st** [5]

  see numBCDinit, values 0 8
- unsigned const **num10st** [5]

  see numBCDinit, values 0 10
- unsigned const **num20st** [5]

see numBCDinit, values 0 20

- unsigned const **num40st** [5]

  see numBCDinit, values 0 40

- unsigned const **num80st** [5]

  see numBCDinit, values 0 80

- unsigned const numBCDinit [5]

  Initialisation or least significant BDC digit for modulation time.

- durDiscrPointData_t perDiscH [5]

  Discrimination values for the modulation period.

- durDiscrPointData_t perDiscP [5]

  Discrimination values for the modulation period.

- durDiscrPointData_t * perDiscUsed

  Discrimination values for the modulation period used.

- durDiscrPointData_t timDiscH [5]

  Discrimination values for the 15% modulation time.

- durDiscrPointData_t timDiscHs [5]

  Discrimination values for the 15% modulation time.

- durDiscrPointData_t timDiscP [5]

  Discrimination values for the 15% modulation time.

- durDiscrPointData_t * timDiscUsed

  Discrimination values for the 15% modulation time.

7.28.1 Detailed Description

DCF77 decoder on Raspberry Pi.

```
Copyright  (c)   2020     Albrecht Weinert
weinert-automation.de     a-weinert.de
```

Revision history

```
Rev. 240 10.04.2021
Rev. 233 17.10.2020 : new
Rev. 234 05.12.2020 : minor, comment
Rev. 239 02.03.2021 : functions from dcf77onPi.c ported here
```

This is a supplementary basic library to handle the signal of a DCF77 receiver. In the case of AM (amplitude) modulation a 1 (high) at the signal input means 15% amplitude and a 0 (low) 100% respectively full amplitude. The other way round is marked as dcf77invInp.

Receivers

Devices used and tested are the AM receiver modules from several sources. See the blog post for results.
The first one is relatively cheap and widespread but a bit of low grade (to put it mildly) in the sense of inserting spiky 1 signals and sensitivity to electromagnetic interferences (EMI).
On the other hand, being able to extract sensible DCF77 from the Pollin module's signal is good report for the decoding and filtering algorithms and or extra circuitry used.

7.28.2 Macro Definition Documentation

7.28.2.1 DCF77recCNTdef

```
#define DCF77recCNTdef
```

Recommended default GPIO for the receiver control output.

This is the default setting for the receiver control output dcf77recCnt. It is currently PIN23, i.e. GPIO 11 on Pi3/4/0.

7.28.2.2 DCF77inpDEF

```
#define DCF77inpDEF
```

Default GPIO for the DCF77 receiver's AM signal input.

This is the default setting for the modulation signal input dcf77inp. The current setting is PIN10, i.e. GPIO 15 (UART in) on Pi3/4/0. The UART in special function of PIN10 might enable the use of raw DCF77 input drivers for NTP servers. Their implementors chose to use a UART instead of a simple binary input for the AM signal.

7.28.2.3 DCF77RINGbufWRAP

```
#define DCF77RINGbufWRAP
```

Ring buffer maximum Index for modulation period data received.

In the current implementation the value is fixed to 255 as it uses uint8_t for all indexes and counters utilising the wrap around. For any other value change all code in question.

Hint for changes;
The value should cover more than two minutes for handling startup and receiver noise.
And it should be (a power of 2) - 1.

7.28.3 Function Documentation

7.28.3.1 setReceiver()

```
void setReceiver (
            int const level )
```

Set receiver On control.

Parameters

level	set receiver ON or OFF

See also

dcf77recCnt, dcf77recCntInv

7.28.3.2 initDCF77io()

```
uint32_t initDCF77io ( )
```

Initialise the DCF77 signal input GPIO / pin.

The signal input dcf77inp and control output dcf77recCnt are initialised.

Returns

the bank mask of the output dcf77recCnt if given, 0 if PINig

7.28.3.3 dcf77receiveRec()

```
void dcf77receiveRec (
            int pi,
            unsigned gpio,
            unsigned level,
            uint32_t tick )
```

DCF77 receive recorder.

This is a pigpiod callback function for an AM DCF77 receiver. It does no filtering nor decoding. It just fills dfc77actRecPer for the current modulation period and stores it in dfc77ringBrecPer when the next period begins.

7.28.3.4 dcf77receiveRecRegister()

```
int dcf77receiveRecRegister (
            void  )
```

DCF77 receive recorder registration.

This function registers the call back function dcf77receiveRec with the PiGpioDaemon.

Returns

register or error number; also stored in dcf77callbackID

7.28.3.5 dcf77receiveRecDeregister()

```
int dcf77receiveRecDeregister (
            void  )
```

DCF77 receive function de-registration.

This function de-registers the call back registered under dcf77callbackID.

Returns

0: OK, pigif_callback_not_found: otherwise

7.28.3.6 disc5()

```
durDiscrPointData_t * disc5 (
            durDiscrPointData_t table[],
            uint32_t const value )
```

Discriminating a value.

In a discrimination table / array of length 5 this function returns a pointer to the highest table entry with value >= table[i].v
The entry returned must be treated as const.

Parameters

table	discrimination table of length 5. With other lengths the function will fail. Must not be null.
value	the number to be discriminated

Returns

the lowest table entry with value < table[i].v

7.28.4 Variable Documentation

7.28.4.1 dcf77recCnt

```
unsigned dcf77recCnt  [extern]
```

Control output GPIO of the DCF77 receiver's control.

This output, if used and connected, controls the amplitude modulation (AM) receiver's control (On/Off) signal. If there is no such output (PINig) The receiver either has no such control input or it is tight to On.

default PINig; see also dcf77recCntInv, setReceiver()

7.28.4.2 dcf77recCntInv

```
unsigned dcf77recCntInv  [extern]
```

Receiver On control inverted.

The receiver control output (dcf77recCnt), when used, would be set to ON (Hi, 3V) to enable the receiver. And it would be set OFF for a short time to reset a panicing or inactive receiver.
If dcf77recCntInv is true it is the other way round. The common receiver chips control input is low active, hence inverted and dcf77recCntInv should be true. As an open collector stage near the Pi for this output is highly recommended, nevertheless the default value is FALSE.

default FALSE; see also dcf77recCnt, setReceiver()

7.28.4.3 dcf77inp

```
unsigned dcf77inp  [extern]
```

Input GPIO for the DCF77 receiver's AM signal.

This is the amplitude modulation (AM) level signal; the level is either 100% or 15% (for 100 or 200ms).

default PIN08; see also dcf77invInp, dcf77PUD, dcf77glitch

7.28.4.4 dcf77invInp

```
unsigned dcf77invInp  [extern]
```

Inverted DCF77 receiver's signal.

In the case of AM (amplitude) modulation a 1 (high) at the signal input dcf77inp means 15% amplitude and a 0 (low) 100% respectively full amplitude, when dcf77invInp is OFF. ON, obviously, means other way round.

default: OFF; see also dcf77inp

7.28.4.5 dcf77PUD

```
unsigned dcf77PUD  [extern]
```

DCF77 input's pull resistor setting.

default: PI_PUD_KEEP; see also dcf77inp

7.28.4.6 dcf77glitch

```
unsigned dcf77glitch  [extern]
```

DCF77 input's glitch filter time setting.

The value for pigpiod's input filter time in μs. The allowed range is 0 ... 30000. The filtering only works for pins sampled by a callback function (like dcf77receiveRec()).

default: 0; glitch filter off; see also dcf77inp

7.28.4.7 dcf77lastLevel

```
unsigned dcf77lastLevel  [extern]
```

Last DCF77 modulation level.

ON means 15% modulation amplitude; i.e. the signal.
OFF means 100% amplitude; i.e. just the 77,5 kHz carrier.

This variable is set by the receiver (callback) function and must not be modified by user software.

7.28.4.8 timDiscP

```
durDiscrPointData_t timDiscP[5]  [extern]
```

Discrimination values for the 15% modulation time.

This is an array of fixed length 5 to discriminate dcf77recPerData_t::tim values.
The only good outcomes of discrimination are indices (durDiscrPointData_t::i) 1 and 3 meaning a recognisable bit 0 respectively 1.
Hint: index bit 0 set means no error.
Hint2: Name ending with P means designed for low grade AM receiver modules. The criteria values are extended quite far for guessing the meaning in the presence of timing faults. Spikes would have to be filtered in a next stage by combining two to four faulty periods in one. Low grade receivers would be not usable without such (complex) filter algorithms.

7.28.4.9 perDiscP

```
durDiscrPointData_t perDiscP[5]  [extern]
```

Discrimination values for the modulation period.

This is an array of fixed length 5 to discriminate dcf77recPerData_t::per values.
The only good outcomes of discrimination are indices (durDiscrPointData_t::i) 1 and 3 meaning an acceptable 1s respectively 2s period.
Hint: See hints at timDiscP.

7.28.4.10 timDiscH

```
durDiscrPointData_t timDiscH[5]  [extern]
```

Discrimination values for the 15% modulation time.

This is an array of fixed length 5 to discriminate dcf77recPerData_t::tim values.
The good outcomes have indices (durDiscrPointData_t::i) 1 and 3 meaning a recognisable bit 0 (FALSE) respectively 1 (TRUE).
Hint: index bit 0 set means no error.

Hint2: Name ending with H means designed for high grade AM receiver modules with virtually no timing faults or spikes. Hence, the criteria values are relatively tight, to recognise EMI or short outages as such. Trying to interpret those with filter algorithms may not be worth the effort with good receivers.

Hint3: Some lower grade AM receiver modules were enhanced with an inverting NPN transistor stage with a low capacity collector to ground capacitor implemented by three meter shielded signal and supply cable. This adding of a simple inverter stage is recommended for all receiver modules not equipped with an open collector (OC) output stage. And for some of them its a necessary filter stage.

7.28.4.11 timDiscHs

```
durDiscrPointData_t timDiscHs[5]  [extern]
```

Discrimination values for the 15% modulation time.

This is an array of fixed length 5 to discriminate dcf77recPerData_t::tim values.
It is the same as timDiscH except for the extra note/hint 4.

Hint 4: The price for turning bad to good receivers by the circuit of Hint 3 was good pulses shortened well below 100 respectively 200 ms. Therefore this table allows for such shortened pulses noting it by the name's suffix s.

7.28.4.12 perDiscH

```
durDiscrPointData_t perDiscH[5]  [extern]
```

Discrimination values for the modulation period.

This is an array of fixed length 5 to discriminate dcf77recPerData_t::per values.
The only good outcomes of discrimination are indices (durDiscrPointData_t::i) 1 and 3 meaning an acceptable 1s respectively 2s period.
Hint: See hints at timDiscHs.

7.28.4.13 perDiscUsed

```
durDiscrPointData_t* perDiscUsed  [extern]
```

Discrimination values for the modulation period used.

This is a pointer to an array of fixed length 5 to discriminate dcf77recPerData_t::per values. The purpose is to hold the current filter values.
default: perDiscP

7.28.4.14 timDiscUsed

```
durDiscrPointData_t* timDiscUsed  [extern]
```

Discrimination values for the 15% modulation time.

This is a pointer to an array of fixed length 5 to discriminate dcf77recPerData_t::per values. The purpose is to hold the current filter values.
default: timDiscP

7.28.4.15 curBCDnum

```
unsigned curBCDnum  [extern]
```

Number decoded from period sequence parts.

This variable holds the (current) number evaluated by considering the sequence of disc5(timDiscUsed, modulation↩ Time) in question. Here, with number sequences a false (level 1, 'F') means 0 and a true (level 3, 'T') means 1, 2, 4, 8, 10, 20, 40 or 80 depending on the place within the BCD coded number. Any other level (0 spike, 3 undef, 4 error) invalidates the whole number.
Values 0..99 (max year) are OK, a value above means an error in the sequence.

See also

disc5 timDiscUsed timDiscHs

7.28.4.16 numBCDinit

```
unsigned const numBCDinit[5]  [extern]
```

Initialisation or least significant BDC digit for modulation time.

For the durDiscrPointData_t i value as index this constant array yields 0 or 1 — or respectively 2, 4 , 8, 10, 20, 40, 60 and 80 — for valid times and a high error value else.
The rationale is just adding up yields a correct value in the BCD range of 0 .. 99 while any higher value means error.

See also

timDiscUsed timDiscHs disc5() curBCDnum

7.29 include/weEcarLd.h File Reference

Some functions for E-Car loading on a Raspberry Pi using pigpiod.

```
#include "weGPIOd.h"
```

Macros

- #define **DEF_BEG_PHASES**
  E-Car AC loading start value: 3 phases.
- #define DEF_CP_INVERTED
  CHS_CP_INVERTED: CP signal PWM is inverted.
- #define DEF_CP_OFFSET
  CHS_CP_OFFSET: CP signal PWM correction offset.
- #define **DEF_CURRENT_LIM_BEG**
  E-Car start loading imit 8.7A per phase.
- #define **DEF_MAX_PHASES**
  current station offers (max.) 3 phases 6 for test

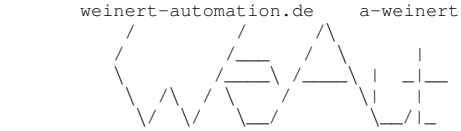
Functions

- int chsItoDutyC (float ldCurrLim)
  Calculate CP dutycycle for current limit per phase.
- void initAsCPilot (int thePi, unsigned gpio, uint8_t init)
  Initialise a GPIO pin as pilot drive.
- void setChStControl (float ldCurrLim)
  Control E-Car charging current limit per phase by PWM.
- float setChStLimit (float ldCurrLim)
  Set and get E-Car charging current limit per phase.
- uint8_t setChStNoPhases (uint8_t assumedNofPhases)
  Set and get charging E-Car's actual number of phases.
- void setCPilot (uint8_t set)
  Set CP signal to constant ON or OFF.
- void setCPilotPWM (int duty)
  Set CP signal PWM.

7.29.1 Detailed Description

Some functions for E-Car loading on a Raspberry Pi using pigpiod.

```
Copyright  (c)     2023  Albrecht Weinert
weinert-automation.de    a-weinert.de
```



Revision history

```
Rev. 253 7.10.2023
Rev. 253 07.09.2023 : new
```

GPIO usage

an input: connected to the charging station and not ready for charging
an input: line power fed to car by a contactor
an output: CP signal as TTL to be amplified to +/-12V ba EVSE)

Note: All three signals between Pi and the load controller (EVSE, Electric Vehicle Supply Equipment) should be fed over optocouplers. Those might (preferably) be integrated in the EVSE.

Values for the loading equipment are defined by default macros defined here with named starting by DEF_ (like e.g. DEF_CURRENT_LIM_MAX 34.0). This default can be overridden in earlier include file by defining a marcro starting with CHS_ instead (like e.g. CHS_CURRENT_LIM_MAX 16.0).

Default values:

```
CHS_CURRENT_LIM_MAX  34.0  A per phase  (CHS_CURRENT_LIM_MIN fixed 6.0 A)
CHS_CURRENT_LIM_BEG   8.7  A per phase
CHS_BEG_PHASES          3  phases start value
CHS_MAX_PHASES          3  phases maximum
CHS_CP_OFF_LEV          1  1:  +12V is constant off, -1: -12V
CHS_CP_OFFSET         -30  &permil; or
CHS_CP_INVERTED         0  not inverted TTL hi outputs +12V
```

7.29.2 Macro Definition Documentation

7.29.2.1 DEF_CP_OFFSET

```
#define DEF_CP_OFFSET
```

CHS_CP_OFFSET: CP signal PWM correction offset.

The CP signal's duty cycle to E-car (+/-12V) might deviate from the s control signal (TTL). If, for example, the CP's On state is 3.3% resp. 33‰ or 33ider than the TTL signal, the latter will have to be corrected by -33‰. This number (-33) would then be the value of CHS_CP_OFFSET.

The default -33 is the correct value for the (modified) Analog EVSE. Due to asymmetries caused by the optocoupler input the change to Hi state is propagated faster than the one to Lo.

To have another value define CHS_CP_OFFSET in an earlier include file. Values outside -40 ... -20 are problematic. For no asymmetry in the signal path from PI to ESVE set CHS_CP_OFFSET 0 (which will slightly shorten the code).

7.29.2.2 DEF_CP_INVERTED

```
#define DEF_CP_INVERTED
```

CHS_CP_INVERTED: CP signal PWM is inverted.

In the not inverted case (0) CP signal = LO would output -12V and hi +12V. In the inverted case (1) it's the over way round. This is a fixed property of the EVSE electronic and the connection to the PI.

The default for CHS_CP_INVERTED is 0 = not inverted.
If otherwise define CHS_CP_INVERTED as 1 in an earlier include file.

7.29.3 Function Documentation

7.29.3.1 initAsCPilot()

```
void initAsCPilot (
            int thePi,
            unsigned gpio,
            uint8_t init )
```

Initialise a GPIO pin as pilot drive.

This function initialises the GPIO pin like initAsHiDrive() and (additionally) sets the PWM frequency to 1kHz and the duty cyle range to 0..1000. The parameter init when 0 (FALSE) sets the CP output to -12V signalling EVSE being not ready. init when 1 (TRUE) sets the CP output to +12V signalling EVSE ready.

This function must be called before all other CP control functions, as it sets the CP's GPIO henceforth.

A value outside this range will turn the 1kHz CP signal off; see CHS_CP_OFF_LEV.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..53)
init	0: off (-12V) 1: on (+12V)

7.29.3.2 setCPilot()

```
void setCPilot (
            uint8_t set )
```

Set CP signal to constant ON or OFF.

This function sets the CP either constant ON (+12V) or OFF (-12V). It will stop any 1kHz square wave.

Parameters

set	0: off (-12V) 1: on (+12V)

7.29.3.3 setCPilotPWM()

```
void setCPilotPWM (
            int duty )
```

Set CP signal PWM.

This function sets the CP duty cycle in the range 50..970‰ applying offset (CHS_CP_OFFSET) and inversion (CHS_CP_INVERTED) if applicable.

Note: 50‰ (5%) signals the of a digital protocol (not implemented here). The range for signalling current limit per phase is 10..970‰; see chsItoDutyC(float).

Parameters

duty	dutycycle in ‰ (parts per mille) 50..970‰

7.29.3.4 chsItoDutyC()

```
int chsItoDutyC (
            float ldCurrLim )
```

Calculate CP dutycycle for current limit per phase.

The IEC CP dutycycle(current) curve will be applied and the outcome limited to 100..970‰ resp. 6..80A per phase.

This function is a helper for setChStControl(float) and usually not used directly.

Parameters

ldCurrLim	the current limit

Returns

dutycycle in ‰ (per mille)

7.29.3.5 setChStControl()

```
void setChStControl (
            float ldCurrLim )
```

Control E-Car charging current limit per phase by PWM.

This function sets the the control pilot (CP) PWM signal to the charging station.

This function should only be called with a current value within actual limits (6...32A, e.g.). It only limits the CP duty cycle to the range 10..97% corresponding to 6..80A.

Parameters

ldCurrLim	current limit in A per phase

7.29.3.6 setChStNoPhases()

```
uint8_t setChStNoPhases (
            uint8_t assumedNofPhases )
```

Set and get charging E-Car's actual number of phases.

This function sets the (assumed) number of phases for E-Car AC charging in the range 1..CHS_MAX_PHASES. The default is CHS_BEG_PHASES.

The value set will be used to calculate and eventually limit the maximum loading power. The real number of phases depends on the car(s) connected.

Warning: There exist E-car types using only one phase even when offered three!

Parameters

assumedNofPhases	number of phases 1.. CHS_MAX_PHASES

Returns

the (new) number of phases

7.29.3.7 setChStLimit()

```
float setChStLimit (
            float ldCurrLim )
```

Set and get E-Car charging current limit per phase.

This function sets the charging stations current limit in the range 6.0 ... CHS_CURRENT_LIM_MAX.

Note: This function just sets/changes the internal limit value. It will not influence any signal (CP) to the EVSE.

Parameters

ldCurrLim	load current in A (per phase)

Returns

the (new) current limit

7.30 include/weGPIOd.h File Reference

IO functions for Raspberry Pis.

```
#include "arch/config.h"
#include "weUtil.h"
#include <pigpiod_if2.h>
```

Macros

- #define PI_PUD_DEFAULT

  Leave pull resistor setting as defaulted.
- #define PI_PUD_KEEP

  Leave pull resistor setting unchanged.
- #define PINig

  Defines a disabled GPIO.

Functions

- uint8_t gpio4pin (int const pin)

  Pin number to GPIO number lookup.
- void initAsDrive (int thePi, unsigned gpio, unsigned init)

  Initialise a GPIO pin as output.
- void initAsHiDrive (int thePi, unsigned gpio, unsigned init)

  Initialise a GPIO pin as high drive.
- void initAsHiInput (int thePi, unsigned gpio)

  Initialise a GPIO pin as input with pull down.
- void initAsInput (int thePi, unsigned gpio)

  Initialise a GPIO pin as input.
- void initAsInputs (unsigned const lesGPIOs[])

  Initialise one or more GPIO pin as input.
- void initAsLoInput (int thePi, unsigned gpio)

  Initialise a GPIO pin as input with pull up.
- uint32_t initAsOutput (int thePi, unsigned gpio)

  Initialise a GPIO pin as output.
- uint32_t initAsOutputs (unsigned const lesGPIOs[])

  Make one or more GPIO pins output.
- uint8_t pin4GPIO (int const gpio)

  GPIO number to pin number lookup.

- uint32_t releaseOutputs (int const thePi)

  Release all GPIO pins set as output.
- uint32_t releaseOutputsReport (int const thePi)

  Release all GPIO pins set as output with report.
- void reportPinOp (char const *op, unsigned const lesGpio[])

  Report an arbitrary operation on a list of GPIOs.
- void setOutput (unsigned const gpio, unsigned const level)

  Set a GPIO output pin.
- void setOutputs (uint32_t const lesOuts, unsigned const level)

  Set a list/mask of GPIO output pins.
- void setPadStrength (int thePi, unsigned mA)

  Set the output drive capacity of GPIO ports 0..27.

Variables

- uint32_t const gpio2bit [36]

  GPIO number to bank pin number lookup.
- uint8_t const gpio2pin [ ]

  GPIO number to pin number lookup.
- uint8_t const pin2gpio [44]

  Pin number to GPIO number lookup.
- char const **pudTxt** [5][6]

  Names for input's pull resistor settings.
- int thePi

  The standard Pi for gpio(d) IO of the program.

7.30.1 Detailed Description

IO functions for Raspberry Pis.

```
Copyright  (c)   2020 2023    Albrecht Weinert
weinert-automation.de         a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 253 9.10.2023
Rev. 232 12.08.2020 : common functions collected and consolidated
Rev. 233 12.10.2020 : initAsHiInput() added
Rev. 237 01.03.2021 : documentation (Doxygen) corrected/improved
Rev. 251 15.08.2023 : documentation corrected/improved
Rev. 252 17.08.2023 : ECar loading ++
```

The IO functions will work with the gpio/gpiod library – daemon running – as defined in pigpiod_if2.h.

This is a supplementary basic library to be used in conjunction with the pigpio library (linked by: -lpigpiod_if2 -lrt).

7.30.2 Macro Definition Documentation

7.30.2.1 PI_PUD_KEEP

```
#define PI_PUD_KEEP
```

Leave pull resistor setting unchanged.

This is an illegal value (4) just one above the legal ones (0..2). Its purpose is to state the current whatever pull setting shall be left untouched.

7.30.2.2 PI_PUD_DEFAULT

```
#define PI_PUD_DEFAULT
```

Leave pull resistor setting as defaulted.

This is an illegal value (3) just one above the legal ones (0..2). Its purpose is to state that an otherwise defined default value shall be used. If no such default value is known, the current pull setting shall be left untouched.

7.30.2.3 PINig

```
#define PINig
```

Defines a disabled GPIO.

This is an illegal GPIO value (33) which some IO functions recognise as "disabled". Hence they can be just called in the normal program flow doing no I/O on a disabled port.

7.30.3 Function Documentation

7.30.3.1 pin4GPIO()

```
uint8_t pin4GPIO (
            int const gpio )
```

GPIO number to pin number lookup.

Parameters

gpio	a GPIO number (>= 0) available on the Pi's 40 respectively 26 pins IO connector

Returns

the GPIO's pin number (1..40 resp. 26) on the IO connector;
0 means not available in the Pi IO connector or even undefined

7.30.3.2 gpio4pin()

```
uint8_t gpio4pin (
            int const pin )
```

Pin number to GPIO number lookup.

Parameters

pin	1..40 (26) is the legal IO connector pin number

Returns

0..56 the GPIO number; 90: ground (0V);
93: 3.3V; 95: 5V; 99: undefined, i.e. illegal pin number

7.30.3.3 initAsInputs()

```
void initAsInputs (
            unsigned const lesGPIOs[] )
```

Initialise one or more GPIO pin as input.

This functions sets the pins listed as GPIOs to input mode. This is also used to release them from any output modes as input means hi impedandance.

The Pi for the IO operation is thePi.

Parameters

lesGPIOs	array of GPIO numbers (0..53); use 0x7F as end marker

7.30.3.4 reportPinOp()

```
void reportPinOp (
            char const * op,
            unsigned const lesGpio[] )
```

Report an arbitrary operation on a list of GPIOs.

The report lines on outLog will be
`progNam ___operation GPIO: 13 pin: 27`

Parameters

op	the operation displayed as 12 characters right justified.
lesGpio	a GPIO list (terminated by a value $>$ 56)

7.30.3.5 initAsOutputs()

```
uint32_t initAsOutputs (
            unsigned const lesGPIOs[] )
```

Make one or more GPIO pins output.

This functions sets the pins listed as GPIOs to output mode.
Normally, at program end, the same list of outputs should be released to high impedance by initAsInputs().

The Pi for the IO operations is thePi.

Parameters

lesGPIOs	array of GPIO numbers (0..53); use 0x7F as end marker

Returns

the bank mask of the outputs set so by this function; 0 means non set (complete failure)

7.30.3.6 setOutputs()

```
void setOutputs (
            uint32_t const lesOuts,
            unsigned const level )
```

Set a list/mask of GPIO output pins.

This functions sets the (output) pins set in the bank mask ON or OFF.

The Pi for the IO operations is thePi.

Parameters

lesOuts	bank mask of outputs to be set
level	OFF or ON (0 or 1)

7.30.3.7 setOutput()

```
void setOutput (
            unsigned const gpio,
            unsigned const level )
```

Set a GPIO output pin.

This functions sets an output pin gpio ON or OFF.
If gpio is $>$ 31 nothing is done. That handles the meaning of gpio 33 (within a bank) as unused.

The Pi for the IO operations is thePi.

Parameters

gpio	an output pin (that should have been set as such!)
level	OFF or ON (0 or 1)

7.30.3.8 initAsOutput()

```
uint32_t initAsOutput (
            int thePi,
            unsigned gpio )
```

Initialise a GPIO pin as output.

This sets a GPIO as output and puts it in the list of GPIOs used as outputs by the program if in the range of 0..31 (resp. 2..27).

All functions setting setting as output should use this function.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..31); PINig means no action

Returns

the bank mask of this output if set (0 no action or error)

7.30.3.9 releaseOutputs()

```
uint32_t releaseOutputs (
            int const thePi )
```

Release all GPIO pins set as output.

This releases all GPIO in the range 0..27 that this program has set as outputs (by setting those as inputs).
The order of this operation is by increasing GPIO number. If another order or extra actions are required this must be done before or afterwards.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)

Returns

the bank mask of the previous resp. released outputs

7.30.3.10 releaseOutputsReport()

```
uint32_t releaseOutputsReport (
            int const thePi )
```

Release all GPIO pins set as output with report.

Same as releaseOutputs() plus a "releaseToIn" for every output released.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)

Returns

the bank mask of the previous resp. released outputs

7.30.3.11 initAsInput()

```
void initAsInput (
            int thePi,
            unsigned gpio )
```

Initialise a GPIO pin as input.

This sets a GPIO as input and removes it form the list of GPIOs set as output if in the range of 0..31 (resp. 2..27).

All functions setting as input should use this function.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..53)

7.30.3.12 initAsLoInput()

```
void initAsLoInput (
            int thePi,
            unsigned gpio )
```

Initialise a GPIO pin as input with pull up.

This initialisation is for an input sensing a switch (button) or transistor (optocoupler) connected to ground (gnd, 0V). This is the normal configuration instead of switching to Hi (3.3V).
In most of the cases the Pi's internal pull up resistor (about 50 kOhm) is sufficient for Lo-switches and should then be used.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..53)

7.30.3.13 initAsHiInput()

```
void initAsHiInput (
            int thePi,
            unsigned gpio )
```

Initialise a GPIO pin as input with pull down.

This initialisation is for an input sensing an electronic device delivering a voltage about 3 V when active respectively ON. Some of those devices require a pull down to deliver clean signals.
The Pi's internal pull down resistor (about 50 kOhm) may be sufficient for Hi-switches and should then be used.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..53)

7.30.3.14 initAsHiDrive()

```
void initAsHiDrive (
            int thePi,
            unsigned gpio,
            unsigned init )
```

Initialise a GPIO pin as high drive.

Of course, Raspberry's (BCM2837's) GPIO pins are high and low drivers as output. Hi-drive is provided by turning on pull-up as to allow broken wire diagnosis when shortly switching to input.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..53)
init	0 or 1: the initial output value; else: leave unchanged

7.30.3.15 initAsDrive()

```
void initAsDrive (
            int thePi,
            unsigned gpio,
            unsigned init )
```

Initialise a GPIO pin as output.

This function sets the GPIO pi as output, optionally sets the drive capacity and leaves a pull resistor setting unchanged.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..53)
init	0 or 1: the initial output value; else: leave unchanged

7.30.3.16 setPadStrength()

```
void setPadStrength (
            int thePi,
            unsigned mA )
```

Set the output drive capacity of GPIO ports 0..27.

For pins set as output (by initAsHiDrive() or initAsDrive() e.g.) this function sets the drive capacity in the range of 2..16 mA.
Note 1: All 27 pins get the same common value. Hence, one has to set the maximum needed for any pin.
Note 2: This value is no current limit nor pin overload protection. It is the maximum load current, which a valid 0 or 1 output voltage can be guaranteed under. Note 3: The BCM processor can set the strength in 2mA steps (2, 4.. 14, 16). Nevertheless, this function accepts all values 2..16 incrementing odd values.

Parameters

the↩ Pi	the Raspberry's identifier as got from initialising gpio(d)
mA	2..16: output drive capacity for providing legal low or high; else: leave drive capacity unchanged

7.30.4 Variable Documentation

7.30.4.1 gpio2pin

```
uint8_t const gpio2pin[]  [extern]
```

GPIO number to pin number lookup.

Index [0..39] is a GPIO number available on the Pi's 40 (26) pins connector. Result 1..40: The GPIO's pin number on the IO connector;
0: not available in the Pi IO connector or even undefined
Outside [0..39] index out of bound.
See also gpio2bit, pin2gpio

7.30.4.2 gpio2bit

```
uint32_t const gpio2bit[36]  [extern]
```

GPIO number to bank pin number lookup.

Index [0..31] is a GPIO number partly (2..27) available on the Pi's 40 (26) pins connector. Result 0x00000001..0x80000000: a 32 bit with exactly one bit set corresponding to place in a 32 bit bank mask.
Outside [0..31] index out of bound.
See also gpio2pin, pin2gpio

7.30.4.3 pin2gpio

```
uint8_t const pin2gpio[44]  [extern]
```

Pin number to GPIO number lookup.

Index [1..40 (26)] is the legal pin number.
Result 0..56: GPIO number; 90: ground (0V); 93: 3.3V; 95: 5V;
99: undefined (for pin 0 and 41 (27)..44).
Outside [0..43] index out of bound.
See also gpio2bit, gpio2pin

7.30.4.4 thePi

```
int thePi  [extern]
```

The standard Pi for gpio(d) IO of the program.

This global variable is provided to hold the main pi used by a program doing process IO via piGpIO[d]. In most local use cases
```
thePi = pigpio_start(NULL, NULL);
```

it will be 0 = this local Pi. After usage don't forget to terminate the connection to the pigpio daemon by
```
pigpio_stop(thePi);
```

7.31 include/weGPIOkeyM.h File Reference

Key matrix IO functions for Raspberry Pis.

```
#include "weGPIOd.h"
```

Data Structures

- struct _km

  A key matrix connected to a Raspberry Pi's GPIO pins.

Macros

- #define **MAX_MATRIX_GPIO**

  restrict to max. 13 (3+10) GPIOs per matrix
- #define **MAX_MATRIX_KEYS**

  restrict to max. 30 (5∗6, 3∗10) keys per matrix

Typedefs

- typedef struct _km keyMatrix

  A key matrix connected to a Raspberry Pi's GPIO pins.
- typedef void(∗ pt2lastScanStep) (struct _km ∗thePad, uint8_t changes)

  forward declare for C's inability; this is keyMatrix

Functions

- void colRowInp (keyMatrix thePad)

  Prepare all column and row GPIO pins of a key matrix.
- void crScanStep (keyMatrix ∗thePad, uint32_t cnt1ms)

  Column and row scan step.
- void keyMatrixInit (keyMatrix ∗thePad, int thePi)

  Set the keyMatrix fields not initialised at definition.
- void keyPadInit (keyMatrix ∗thePad, int thePi)

  Initialise the key pad.
- void printMatrixConfig (FILE ∗stream, keyMatrix thePad)

  Print the initialised keyMatrix structure.
- void printMatrixPressState (FILE ∗stream, keyMatrix thePad, uint8_t cmp)

  Print the keyMatrix' key pressed state.
- void setKeyMatrixScanFunct (keyMatrix ∗thePad, pt2lastScanStep fun)

  Set the key pad's last scan step function.

7.31.1 Detailed Description

Key matrix IO functions for Raspberry Pis.

The IO functions will work with the gpio/gpiod library – daemon running – as defined in pigpiod_if2.h and weGPIOd.h.

```
Copyright (c) 2018 Albrecht Weinert
weinert-automation.de a-weinert.de
```

Revision history

```
Rev. 253 9.10.2023
Rev. 176 29.07.2017 : new (excerpt from weGPIOd.h)
Rev. 191 15.02.2019 : minor comment text changes
```

This is a supplementary library to be used in conjunction with weGPIOd.h and the pigpio library (link: -lpigpiod_if2 -lrt).

7.31.2 Typedef Documentation

7.31.2.1 pt2lastScanStep

```
typedef void(* pt2lastScanStep) (struct _km *thePad, uint8_t changes)
```

forward declare for C's inability; this *is* keyMatrix

Function on last scan step.

Parameters

thePad	address of the structure defining the key matrix (not NULL)
changes	0: no change to last scan; bit 0: count changes; bit 1: set changes (new keys pressed or released)

7.31.2.2 keyMatrix

```
typedef struct _km keyMatrix
```

A key matrix connected to a Raspberry Pi's GPIO pins.

The structure type keyMatrix describes a key matrix with column and row lines connected directly to the GPIO pins. At the cross points there would be normally open (NO) push button switches. An example for such matrix is the 12 keys matrix (3 columns, 4 rows) EOZ Clavier S.series, 12 touches, Matricell, Téléphone. The maximum number of keys supported is 30 (6∗5 or 3∗10) delimited by the array size in this structure and, of course, by the GPIOs put to Raspberry's pin header, as well as by the approach to use the bits 0..29 of an uint32_t as compact switch state.

When defining a keyMatrix structure for a concrete key matrix one has to initialise correctly the fields, .noCols by the number of columns, .noRows by the number of rows, the array .colRow by (.noCols + .noRows) GPIO pin numbers (columns first) and the array .keyVal by (.noCols ∗ .noRows) key letters (row by row).

The other field will be set by once calling keyMatrixInit before doing any other (IO) operation.

7.31.3 Function Documentation

7.31.3.1 colRowInp()

```
void colRowInp (
            keyMatrix thePad )
```

Prepare all column and row GPIO pins of a key matrix.

All GPIO pins to be used for scanning a key (switch) matrix defined in the structure thePad are all initialised
a) as input,
b) with pull-down resistor and
c) 2mA drive capacity (when switched to output for scan step).

Parameters

thePad	the keyMatrix structure defining the key pad

7.31.3.2 keyMatrixInit()

```
void keyMatrixInit (
            keyMatrix * thePad,
            int thePi )
```

Set the keyMatrix fields not initialised at definition.

On definition of the keyMatrix structure thePad, its fields
.noCols, .noRows,
.colRow and .keyVal
have to be initialised according to the key matrix' hardware configuration. Additionally
.pt2lastScanStep
may be initialised pointing to an appropriate user supplied function. (It will be NULL otherwise.)

This procedure sets all other fields accordingly. It must be called once before any other operation.

Parameters

thePad	address of the structure defining the key matrix (not NULL)
thePi	the Raspberry's identifier as got from initialising gpio(d)

7.31.3.3 setKeyMatrixScanFunct()

```
void setKeyMatrixScanFunct (
            keyMatrix * thePad,
            pt2lastScanStep fun )
```

Set the key pad's last scan step function.

The supplied function with signature
  void fooSetKeys(keyMatrix * thePad, uint8_t changes)
or NULL is set in the structure thePad. If not NULL the function
  crScanStep(keyMatrix * thePad)
will call the function supplied after one round of scan steps in the last step.

Parameters

thePad	address of the structure defining the key matrix (not NULL)
fun	the function to be call on last step

7.31.3.4 keyPadInit()

```
void keyPadInit (
            keyMatrix * thePad,
            int thePi )
```

Initialise the key pad.

Initialises the structure the pad and the IO; i.e. keyMatrixInit() and then colRowInp(). Has to be called only once before any further (IO) operation.

Parameters

thePad	address of the structure defining the key matrix
thePi	the Raspberry's identifier as got from initialising gpio(d)

7.31.3.5 crScanStep()

```
void crScanStep (
            keyMatrix * thePad,
            uint32_t cnt1ms )
```

Column and row scan step.

Actions for a single scan step.
The steps are
0 ... .noRoCo-1 : process IO
And step .noRoCo: calculation and action
A call to this function should be put in an appropriate cycle, like 1ms.

Parameters

thePad	address of the structure defining the key matrix
cnt1ms	millisecond time stamp

7.31.3.6 printMatrixConfig()

```
void printMatrixConfig (
            FILE * stream,
            keyMatrix thePad )
```

Print the initialised keyMatrix structure.

Outputs a nice matrix view on the GPIO pins and masks as well as the labels of the key matrix. (For non-proportional font only.) Should not be used before keyPadInit().

Parameters

stream	the output; NULL defaults to outLOg (stdout)
thePad	address of the structure defining the key matrix

7.31.3.7 printMatrixPressState()

```
void printMatrixPressState (
            FILE * stream,
            keyMatrix thePad,
            uint8_t cmp )
```

Print the keyMatrix' key pressed state.

Outputs a nice matrix view on matrix' buttons and their pressed state (i.e. scan counts of the button in question). Should only be used within the user supplied last scan step function; and best only on count changes.

Parameters

stream	the output; NULL defaults to outLog (stdout)
thePad	address of the structure defining the key matrix
cmp	0: no change; bit 0: count change; bit 1 state change

7.32 include/weLockWatch.h File Reference

Process control helpers for Raspberry Pi: lock and watchdog.

```
#include "sysBasic.h"
#include <sys/file.h>
```

Functions

- void closeLock (void)

  Unlock the lock file

- int initWatchdog (void)

  Get and initialise or arm the watchdog.
- int justInitWatchdog (void)

  Get and initialise or arm the watchdog.
- int justLock (char const *lckPiGpioFil)

  Open and lock the lock file.
- int openLock (char const *lckPiGpioFil, uint8_t const perr)

  Open and lock the lock file.
- void stopWatchdog (void)

  Stop and disarm the watchdog.
- void triggerWatchdog (void)

  Trigger the watchdog.

Variables

- int lockFd

  Lock file handle.
- int useIOlock

  Do use IO lock.
- int useWatchdog

  flag to use watchdog; default ON

7.32.1 Detailed Description

Process control helpers for Raspberry Pi: lock and watchdog.

```
Copyright  (c)   2020     Albrecht Weinert
weinert-automation.de     a-weinert.de
```

Revision history

```
Rev. 253 9.10.2023
Rev. 227 12.08.2020 : common functions collected and consolidated
Rev. 243 09.09.2022 : typo
```

This is a supplementary basic library to handle locks and a watchdog.

7.32.2 Function Documentation

7.32.2.1 justLock()

```
int justLock (
            char const * lckPiGpioFil )
```

Open and lock the lock file.

This function is the basic implementation of openLock. Applications not wanting its optional logging or doing their own should use this function directly.

Parameters

lckPiGpioFil	lock file name

Returns

0: OK, locked; 97: fd does not exist; 98: can't be locked

7.32.2.2 openLock()

```
int openLock (
             char const * lckPiGpioFil,
             uint8_t const perr )
```

Open and lock the lock file.

This function may use logging and streams not available on smaller applications. Those applications not wanting that optional logging or doing their own should use the function justLock().

Parameters

lckPiGpioFil	lock file path name
perr	make error message when lock file does not exist or can't be locked

Returns

0: OK, locked; 97: fd does not exist; 98: can't be locked

7.32.2.3 closeLock()

```
void closeLock (
             void  )
```

Unlock the lock file

Unlock the lock file

7.32.2.4 initWatchdog()

```
int initWatchdog (
            void  )
```

Get and initialise or arm the watchdog.

If the watchdog is to be used, i.e. useWatchdog is ON. this function tries to get it. Otherwise it does nothing but return 0 (success).

On no success 1 is returned and useWatchdog is set OFF and the misfortune is logged. Use justInitWatchdog() :: for silence.

Returns

0: watchdog OK or not to be used (useWatchdog OFF); 1: error while trying to get the watchdog

7.32.2.5 justInitWatchdog()

```
int justInitWatchdog (
            void  )
```

Get and initialise or arm the watchdog.

This function justInitWatchdog() is the basic implementation of initWatchdog() without logging failure.

Returns

0: watchdog OK or not to be used (useWatchdog OFF); 1: error while trying to get the watchdog

7.32.2.6 stopWatchdog()

```
void stopWatchdog (
            void  )
```

Stop and disarm the watchdog.

If the watchdog is to be used, i.e. useWatchdog is ON, this function stops and disarms it.
useWatchdog is then OFF .

7.32.2.7 triggerWatchdog()

```
void triggerWatchdog (
            void  )
```

Trigger the watchdog.

If the watchdog is to be used, i.e. useWatchdog is ON, this function triggers it.
If the watchdog is armed (by initWatchdog()) not triggering it at least once about every 15 s will lead to system reset.

7.32.3 Variable Documentation

7.32.3.1 useIOlock

```
int useIOlock  [extern]
```

Do use IO lock.

When ON (default) justLock, openLock and closeLock try to fulfil their mission. When OFF they do nothing (except saying OK). flag to use IO (singleton) lock; default on

Do use IO lock.

7.32.3.2 lockFd

```
int lockFd  [extern]
```

Lock file handle.

Do not use directly.

7.32.3.3 useWatchdog

```
int useWatchdog  [extern]
```

flag to use watchdog; default ON

flag to use watchdog; default ON

7.33 include/weModbus.h File Reference

Modbus functions for Raspberry Pis.

```
#include <modbus-private.h>
#include <modbus.h>
#include "weUtil.h"
```

Data Structures

- struct modRS_t

  Structure for Modbus RS485 (RTU).
- struct modTCP_t

  Structure for Modbus TCP.

Typedefs

- typedef enum modRS485state_t modRS485state_t

  A definition for possible states of a Modbus RS485 interface.

Enumerations

- enum modRS485state_t {
  RS_OFF , RS_ON , RS_INICOM , RS_ERR_ANY ,
  RS_ERR_CTX , RS_ERR_CON , RS_ERR_SLA , RS_ERR_BAD }

  A definition for possible states of a Modbus RS485 interface.

Functions

- void modRSclose (modRS_t ∗modRS)

  Close a Modbus RS link and destroy the (libmodbus) structure.
- int modRSconnect (modRS_t ∗modRS, int currentSlave)

  Connect a the Modbus (libmodbus) structure for RS485 (RTU).
- int modRSctxNew (modRS_t ∗modRS, int currentSlave)

  Make a new Modbus (libmodbus) structure for RS485.
- int modRSswitchSlave (modRS_t ∗modRS, int currentSlave)

  Switch the slave on a connected Modbus (libmodbus) structure for RS485.
- void modTCPclose (modTCP_t ∗modTCP)

  Close a Modbus TCP and destroy the (libmodbus) structure.
- int modTCPconnect (modTCP_t ∗modTCP)

  Connect a the Modbus (libmodbus) structure for TCP.
- int modTCPctxNew (modTCP_t ∗modTCP)

  Make a new Modbus (libmodbus) structure for TCP.
- int modTCPlisten (modTCP_t ∗modTCP)

  Listen at the Modbus (libmodbus) structure for TCP.
- int parseModPort (const char ∗str)

  Parse a string as Modbus TCP port number with checks.
- void reg2val32 (dualReg_t ∗const dest, dualReg_t const ∗const source)

  Copy float (32 bit) Modbus input to correctly ordered value.
- uint16_t regs2string (char ∗dest, uint16_t ∗source, int n)

  Copy a sequence of character pairs from registers to a string.
- void regs2vals32 (float ∗const dest, dualReg_t const ∗const source, int const n)

  Copy n float (32 bit) Modbus input to correctly ordered values.
- int setIP4add (char ∗dest, const char ∗src)

  Set an IPv4 address as string with syntax checks.

7.33.1 Detailed Description

Modbus functions for Raspberry Pis.

```
Copyright  (c)  2019   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 236 2.02.2021
Rev.  38 22.08.2017 : new
Rev. 187 14.10.2018 : minor typos
Rev. 191 15.02.2019 : minor comment text changes
Rev. 212 04.08.2019 : regs2string added
Rev. 216 31.08.2019 : modRS_t int conErr removed (never used)
```

This is a supplementary (basic) library. The functions and structures defined here will work with Stéphane Raimbault's Modbus library libmodbus.

Some structures and utilities are to support smart meters like e.g. SDM230Modbus, SMD630Modbus and SDM538Modbus as servers/slaves, especially their handling of (32 bit) float values.

7.33.2 Typedef Documentation

7.33.2.1 modRS485state_t

```
typedef enum modRS485state_t modRS485state_t
```

A definition for possible states of a Modbus RS485 interface.

With Modbus TCP one has not to handle the (Ethernet or WLAN) interface. One just has to listen at a port or connect to IPadress:port. The handling of multiple connections on the network is done by the operating system and the network infrastructure.

With an RS485 "bus" connecting Modbus devices it's different. Every device's software has to handle the physical interface. Usually only Modbus devices can be at that "bus" — 2 to 247. The communication is client request --> server response (as always), but request response pairs to/from different servers cannot overlap. (Additionally the telegrams are short and slow.)

Hence with RS485 (RTU) the handling of multiple (server) connections has to be implemented in the client application, and the RS485 interface/bus state has to be separated from (multiple) server-link states.
With RS232 the complications and disadvantages are the same while being restricted to point to point (P2P, no bus).

7.33.3 Enumeration Type Documentation

7.33.3.1 modRS485state_t

```
enum modRS485state_t
```

A definition for possible states of a Modbus RS485 interface.

With Modbus TCP one has not to handle the (Ethernet or WLAN) interface. One just has to listen at a port or connect to IPadress:port. The handling of multiple connections on the network is done by the operating system and the network infrastructure.

With an RS485 "bus" connecting Modbus devices it's different. Every device's software has to handle the physical interface. Usually only Modbus devices can be at that "bus" — 2 to 247. The communication is client request --> server response (as always), but request response pairs to/from different servers cannot overlap. (Additionally the telegrams are short and slow.)

Hence with RS485 (RTU) the handling of multiple (server) connections has to be implemented in the client application, and the RS485 interface/bus state has to be separated from (multiple) server-link states.
With RS232 the complications and disadvantages are the same while being restricted to point to point (P2P, no bus).

Enumerator

RS_OFF	do not use Modbus RS485 link (no ctx structure made)
RS_ON	may be usable, but totally uninitialised (no com.)
RS_INICOM	initialised (ready for communication)
RS_ERR_ANY	no concrete error, lower bound of all error states
RS_ERR_CTX	no structure made (hopeless when re-occurring)
RS_ERR_CON	open serial failed
RS_ERR_SLA	any slave error requiring re-open
RS_ERR_BAD	recurring error in usage (try conn. re-init)

7.33.4 Function Documentation

7.33.4.1 modTCPctxNew()

```
int modTCPctxNew (
            modTCP_t * modTCP )
```

Make a new Modbus (libmodbus) structure for TCP.

This function makes a new modTCP.ctx according to .addr and .port; it does NOT check NOR change .mlStat.

Parameters

modTCP	pointer to modTCP_t structure to be used

Returns

0 : OK; -1: error

7.33.4.2 modTCPconnect()

```
int modTCPconnect (
            modTCP_t * modTCP )
```

Connect a the Modbus (libmodbus) structure for TCP.

This function makes a new modTCP.ctx according to .addr and .port, if not yet made. Then it "connects" it. On success 0 is returned. On failure -1 is returned and modTCP.ctx will be destroyed.

This function does NOT check NOR change .mlStat.

Parameters

modTCP	pointer to modTCP_t structure to be used

Returns

0 : OK; -1: error

7.33.4.3 modTCPlisten()

```
int modTCPlisten (
            modTCP_t * modTCP )
```

Listen at the Modbus (libmodbus) structure for TCP.

This function makes a new modTCP.ctx according to .addr and .port, if not yet made. Then it "listens" on it. On success .s >= 0, that is the socket, is returned. On failure -1 is returned and modTCP.ctx will be destroyed. The number of connections is limited to one, here.

This function does NOT check NOR change .mlStat.

Parameters

modTCP	pointer to modTCP_t structure to be used

Returns

>=0 : OK, i.e the socket andmodTCP.s; -1: error

7.33.4.4 modTCPclose()

```
void modTCPclose (
            modTCP_t * modTCP )
```

Close a Modbus TCP and destroy the (libmodbus) structure.

This function closes the connection (if on) and destroys modTCP.ctx (if existing) .mlStat will be set to ML_OFF.

Parameters

modTCP	pointer to modTCP_t structure to be used

7.33.4.5 setIP4add()

```
int setIP4add (
            char * dest,
            const char * src )
```

Set an IPv4 address as string with syntax checks.

This function sets the parameter dest with src, doing syntactic checks to assure a valid IPv4 address being set.

Parameters

dest	the strtring to copy the IP address to
src	the IP string to copy to dest; NULL acts as "0.0.0.0"

Returns

0: syntax error or dest is NULL, dest unchanged; 1: OK

7.33.4.6 parseModPort()

```
int parseModPort (
            const char * str )
```

Parse a string as Modbus TCP port number with checks.

This function parses the string src as Modbus port number, doing validity checks: src must be a decimal number 502 or 1024..65535.

Parameters

str	the string to parse as (decimal) modPort; NULL acts as 502

Returns

0: syntax error, else valid Modbus port number (see above)

7.33.4.7 modRSctxNew()

```
int modRSctxNew (
            modRS_t * modRS,
            int currentSlave )
```

Make a new Modbus (libmodbus) structure for RS485.

This function makes a new modTCP.ctx according to .addr and .port; it does NOT check NOR change .rsState.

Parameters

modRS	pointer to modRS_t structure to be used
currentSlave	1..247 sets the slave number; sets .currentSlave

Returns

0 : OK; else: error: -1: modRS null; -2: slave number: -3: no ctx

7.33.4.8 modRSconnect()

```
int modRSconnect (
            modRS_t * modRS,
            int currentSlave )
```

Connect a the Modbus (libmodbus) structure for RS485 (RTU).

This function makes a new modRS.ctx according to .device and else. Then it "connects" it. On success 0 is returned. On failure -1 is returned and modTCP.ctx will be destroyed.

This function does NOT check NOR change .rsState.

Parameters

modRS	pointer to modRS_t structure to be used
currentSlave	if 1..247; sets the slave number to be used next and changes .currentSlave

7.33.4.9 modRSswitchSlave()

```
int modRSswitchSlave (
```

```
            modRS_t * modRS,
            int currentSlave )
```

Switch the slave on a connected Modbus (libmodbus) structure for RS485.

This function changes nothing on a functional and connected modRS.ctx setting than the slave number.

Modbus RS485 (RTU) can handle multiple slaves on the same serial interface. This has to be one at a time in a pure sequential matter: hence this slave switching.
With RS232 the one slave's number once correctly established would stay fixed.

Unfortunately (by a libmodbus deficiency) a communication error of one slave would require a total new connect (modRSconnect()) for all slaves; switching to a "good slave" won't help.

Parameters

modRS	pointer to modRS_t structure to be used
currentSlave	if 1..247; sets .currentSlave

Returns

-1: error wrong parameter or no modRS; 0: no change or no ctx; 1..247: changed currentSlave; OK

7.33.4.10 modRSclose()

```
void modRSclose (
            modRS_t * modRS )
```

Close a Modbus RS link and destroy the (libmodbus) structure.

This function destroys modRS.ctx (if existing). .rsState will not be changed.

Parameters

modRS	pointer to modRS_t structure to be used

7.33.4.11 reg2val32()

```
void reg2val32 (
            dualReg_t *const dest,
            dualReg_t const *const source )
```

Copy float (32 bit) Modbus input to correctly ordered value.

This function copies a 32bit (e.g.) float value input from Modbus — with correct byte ordering at 16 bit level (!) — to a correctly ordered (32 bit) value.
Destination and source pointers may be the same, but not NULL.
The swap of the 16-bit parts only occurs, when the platform is little endian.
See also: PLATFlittlE and regs2vals32()

Parameters

dest	the dual registers to store the result in
source	the dual registers with the potentially wrong endianess

7.33.4.12 regs2vals32()

```
void regs2vals32 (
            float *const dest,
            dualReg_t const *const source,
            int const n )
```

Copy n float (32 bit) Modbus input to correctly ordered values.

This function copies n 32bit (e.g.) float values input from Modbus — with correct byte ordering at 16 bit level (!) — to correctly ordered (32 bit) values.
Destination and source pointers may be the same, but not NULL.
The swap of the 16-bit parts only occurs, when the platform is little endian.
See also: PLATFlittlE and reg2val32()

Parameters

dest	the array of dual registers to store the result in
source	the array of dual registers with potentially wrong endianess
n	the number of dual registers to treat

7.33.4.13 regs2string()

```
uint16_t regs2string (
            char * dest,
            uint16_t * source,
            int n )
```

Copy a sequence of character pairs from registers to a string.

Growatt inverter holding registers, for example, deliver strings as pairs of ASCII characters in consecutive registers. If the number of characters is odd the last register holds just one character. Some of those strings in registers are 0-terminated and some are not. In the first case the transfer stops at the 0, in the latter case a 0 is appended to the destination string as n+1st character.
The sequence of the two characters in each register is (as usual with Modbus) in wrong order. Hence, memcopy and consorts would fail.

Parameters

dest	the array of characters to store the result in
source	the array registers with wrong character sorting
n	the number of dual registers to treat

Returns

the number of characters transfered to dest including the terminating 0

7.34 include/weSerial.h File Reference

Definitions for Raspberry Pi's serial communication.

```
#include "arch/config.h"
#include <termios.h>
#include "sysBasic.h"
#include <fcntl.h>
```

Functions

- tcflag_t baudFlag (unsigned int const speed)

  Baud flags by baud rate.
- unsigned int baudRate (tcflag_t baudFlag)

  Baud rate by baud flags.
- void closeUART ()

  Close the UART.
- int openUART ()

  Open the UART with given settings.

Variables

- tcflag_t **baud**

  The UART's baud rate as flag bits.
- struct termios **options**

  The UART's setting structure.
- unsigned int **speed**

  The UART's baud rate as value.
- timespec **startReceive**

  Time used for receive timing.
- int uartFilestream

  The UART as file (stream).
- char * uartPath

  The UART's path name.

7.34.1 Detailed Description

Definitions for Raspberry Pi's serial communication.

```
Copyright (c) 2019 Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 236 2.02.2021
Rev. 209 09.07.2019 : new
```

This is a (basic) library for serial communications. As far as Modbus is or can be supported this is independent of Stéphane Raimbault's Modbus library libmodbus; weModbus.h on the other hand is.

Some structures and utilities are to support smart meters like e.g. SDM230Modbus, SMD630Modbus and SDM538Modbus as servers/slaves, especially their handling of (32 bit) float values.

7.34.2 Function Documentation

7.34.2.1 baudFlag()

```
tcflag_t baudFlag (
            unsigned int const speed )
```

Baud flags by baud rate.

Parameters

speed	a legal baudrate 300 9600 19200 and so on

Returns

the corresponding flag bits if rate is available; otherwise 0 (error). This should be defaulted to a standard rate like 9600 e.g.

7.34.2.2 baudRate()

```
unsigned int baudRate (
            tcflag_t baudFlag )
```

Baud rate by baud flags.

The speed bits of the parameter baudFlag will be evaluated and the corresponding baud rate will be returned. In vase of no valid speed flag value 0 will be returned. 0 may be considered as error and should be defaulted to 9600.

Parameters

baudFlag	the speed bits of the flags parameter will be evaluated

Returns

the corresponding baud rate 300, 9600, 19200 or other standard rate

7.34.2.3 openUART()

```
int openUART ( )
```

Open the UART with given settings.

Returns

uartFilestream; -1 means open error

7.34.2.4 closeUART()

```
void closeUART ( )
```

Close the UART.

uartFilestream will be set to -1.

7.34.3 Variable Documentation

7.34.3.1 uartFilestream

```
int uartFilestream  [extern]
```

The UART as file (stream).

It is > 0 (>2) when open and -1 when closed.

7.34.3.2 uartPath

```
char* uartPath  [extern]
```

The UART's path name.

It will be preset with the architecture's standard UART path. See arch/config.h.

7.35 include/weShareMem.h File Reference

Shared memory on Raspberry Pis.

```
#include "weUtil.h"
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/sem.h>
#include <errno.h>
```

Data Structures

- union semCtlPar_t

  Parameter type for semctl().

Macros

- #define **PERM**

  access rights

Functions

- int deleteSemas ()

  Delete the one semaphore set.
- int deleteSharedMem ()

  Delete and detach the shared memory.
- int **detachSharedMem** ()

  Detach the shared memory.
- int getSemas ()

  Get the one semaphore set.
- int initialiseSemas ()

  Initialise the one semaphore set.
- void * initialiseSharedMem ()

  Initialise shared memory.
- int semaphoreClt (int const semNum, int const op, semCtlPar_t par)

  Control semaphores of the set.
- int semaphoreLock (int const semNum, int ms)

  Lock one semaphore of the set.
- int semaphoreOperation (int const semNum, int const op, int ms)

  Operation on one semaphore of the set.
- int semaphoreUnlock (int const semNum)

  Unlock one semaphore of the set.

Variables

- const semCtlPar_t **VAL0**

  value 0 for SETVAL
- const semCtlPar_t **VAL1**

  value 1 for SETVAL
- const semCtlPar_t **VAL9**

  value 9 for SETVAL

7.35.1 Detailed Description

Shared memory on Raspberry Pis.

```
Copyright  (c)  2018   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /      \|   |
     \/  \/   \__/        \__/|_
```

Revision history

```
Rev. 236 2.02.2021
Rev.  80  11.11.2017 : new
Rev. 175  28.07.2018 : beauty
Rev. 191  15.02.2019 : minor comment text changes
```

Provide one shared memory range of size ... to be used by one or more C (or PHP or ...) programs on a Raspberry Pi3.

7.35.2 Function Documentation

7.35.2.1 getSemas()

```
int getSemas ( )
```

Get the one semaphore set.

The one semaphore set, if existing, will be registered and used as is.

Returns

0: OK found existing semaphore set; -1: error (errno set and errorText generated)

7.35.2.2 initialiseSemas()

```
int initialiseSemas ( )
```

Initialise the one semaphore set.

The number of semaphores in the set is ANZ_SEMAS (default three). The one semaphore set, if existing, will be registered and used as is. If this is not possible it will be made and initialised.

Hint: This function has two OK return values!

Returns

1: OK semaphore set made new; 0: OK found existing semaphore set; -1: error (errno set and errorText generated)

7.35.2.3 semaphoreOperation()

```
int semaphoreOperation (
            int const semNum,
            int const op,
            int ms )
```

Operation on one semaphore of the set.

Parameters

semNum	the semaphore number in the set (0..ANZ_SEMAS -1)
op	the semaphore operation
ms	if 2..20000 a timeout in ms

Returns

0: OK; -1: error (errno set and errorText generated)

7.35.2.4 semaphoreLock()

```
int semaphoreLock (
            int const semNum,
            int ms )
```

Lock one semaphore of the set.

Parameters

semNum	the semaphore number in the set (0..ANZ_SEMAS -1)
ms	if 2..20000 a timeout in ms

Returns

0: OK; -1: error (errno set and errorText generated)

7.35.2.5 semaphoreUnlock()

```
int semaphoreUnlock (
            int const semNum )
```

Unlock one semaphore of the set.

Parameters

semNum	the semaphore number in the set (0..ANZ_SEMAS -1)

Returns

0: OK; -1: error (errno set and errorText generated)

7.35.2.6 semaphoreClt()

```
int semaphoreClt (
            int const semNum,
            int const op,
            semCtlPar_t par )
```

Control semaphores of the set.

Parameters

semNum	the semaphore number in the set (0..ANZ_SEMAS -1)
op	the semaphore operation, like e.g. SETVAL
par	op's parameter if any

Returns

0: OK; -1: error (errno set and errorText generated)

7.35.2.7 deleteSemas()

```
int deleteSemas ( )
```

Delete the one semaphore set.

Server operation only.

Returns

0: OK; -1: error (errno set and errorText generated)

7.35.2.8 initialiseSharedMem()

```
void * initialiseSharedMem ( )
```

Initialise shared memory.

Make or get and attach. return pointer to attached shared memory or (void *)-1

7.35.2.9 deleteSharedMem()

```
int deleteSharedMem ( )
```

Delete and detach the shared memory.

Server operation only.

7.36 include/weStateM.h File Reference

States and state machines.

```
#include "sysBasic.h"
```

Data Structures

- struct state_t

  The structure for state machines.

Macros

- #define newFiveBand(N, C, V, T, R, S)

  Define a five band checker.
- #define newFloatHyst(N, C, O, R, S)

  Define a hysteresis (Schmitt trigger, float value) as state machine.
- #define newSeqCont(N, C, n, m, S)

  Define a sequential function chart as state machine.
- #define newSwitchDeb(N, C, S, R)

  Define a switch to be de-bounced as state machine.
- #define newTimer(N, C, S)

  Define a timer as state machine.

Typedefs

- typedef uint8_t(∗ enterState_t) (state_t ∗const me, uint32_t controlV)

  A state machine's own function to command entry in active state.
- typedef uint8_t(∗ **enterStateF_t**) (state_t ∗const me, float controlF)

  Substitute or addendum to enterState_t.
- typedef uint8_t(∗ **enterStateS_t**) (state_t ∗const me, char const ∗controlS)

  Substitute or addendum to enterState_t.
- typedef void(∗ genStateText_t) (char ∗stateText, state_t const ∗const me, char const ∗stamp)

  Generate text for state machine status.
- typedef uint8_t(∗ leaveState_t) (state_t ∗const me, uint32_t controlV)

  A state machine's own function to command exit out of active state.
- typedef uint8_t(∗ **leaveStateF_t**) (state_t ∗const me, float controlF)

  Substitute or addendum to leaveState_t.
- typedef uint8_t(∗ **leaveStateS_t**) (state_t ∗const me, char const ∗controlS)

  Substitute or addendum to leaveState_t.
- typedef void(∗ onStateChange_t) (state_t ∗const me)

  The applications's call back function for state changes.
- typedef struct state_t **state_t**

  The structure for state machines.
- typedef uint8_t(∗ tickCheckState_t) (state_t ∗const me, uint32_t controlV)

  A machine's own function to be called to trigger / check state.
- typedef uint8_t(∗ **tickCheckStateF_t**) (state_t ∗const me, float controlF)

  Substitute or addendum to tickCheckState_t.
- typedef uint8_t(∗ **tickCheckStateN_t**) (state_t ∗const me)

  Substitute or addendum to tickCheckState_t.

Functions

- uint8_t fiveBandDoEnter (state_t ∗const me, float analogueVal)

  Five band checker turn / force ON.
- uint8_t fiveBandDoLeave (state_t ∗const me, float analogueVal)

  Five band checker turn / force OFF.
- uint8_t fiveBandTick (state_t ∗const me, float controlV)

  Five band checker trigger.
- uint8_t floatHystDoEnter (state_t ∗const me, float analogueVal)

  Float value hysteresis turn / force ON.
- uint8_t floatHystDoLeave (state_t ∗const me, float analogueVal)

  Float value hysteresis turn / force OFF.
- uint8_t floatHystTick (state_t ∗const me, float controlV)

  Float value hysteresis trigger.
- void genStateText (char ∗stateText, state_t const ∗const me, char const ∗stamp)

  Generate text for state machine status.
- void logStateReason (state_t const ∗const me, char const ∗stamp, char const ∗cause)

  Log status text with cause and info.
- void logStateText (state_t const ∗const me, char const ∗stamp)

  Log status text.
- uint8_t seqContDoEnter (state_t ∗const me, char const ∗startCommand)

  Sequential control entry.
- uint8_t seqContDoLeave (state_t ∗const me, char const ∗stopCommand)

  Sequential control leave.
- uint8_t seqContTick (state_t ∗const me)

  Sequential control tick or check.
- void setGenStateText (state_t ∗const me, genStateText_t const fun)

  Set the function to generate text for state machine status.
- void startStateText (char ∗stateText, state_t const ∗const me, char const ∗stamp)

  Generate status text standard start.
- uint8_t switchDebDoEnter (state_t ∗const me, uint32_t controlV)

  Switch de-bounce turn / force ON.
- uint8_t switchDebDoLeave (state_t ∗const me, uint32_t controlV)

  Switch de-bounce turn / force OFF.
- uint8_t switchDebTick (state_t ∗const me, uint32_t controlV)

  Switch de-bounce trigger.
- uint8_t switchDebTickAC (state_t ∗const me, uint32_t controlV)

  Switch de-bounce trigger AC.
- uint8_t timerDoEnter (state_t ∗const me, uint32_t secFromNow)

  Timer entry.
- uint8_t timerDoLeave (state_t ∗const me, uint32_t ignored)

  Timer leave, that is stop timer.
- uint8_t timerDoStart (state_t ∗const me, uint32_t secFromNow)

  Timer unconditional entry and set.
- uint8_t timerDoStart4ever (state_t ∗const me)

  Timer unconditional entry and set forever or stop it.
- uint8_t timerDoTrigger (state_t ∗const me, uint32_t secFromNow)

  Timer entry or (pro-longing) re-trigger.
- uint8_t timerEndTrigger (state_t ∗const me, uint32_t const secUTCend)

  Timer entry or (pro-longing) re-trigger to absolute UTC end.
- uint8_t timerTickCheck (state_t ∗const me, uint32_t controlV)

  Timer trigger.

7.36.1 Detailed Description

States and state machines.

```
Copyright  (c)  2018 2020   Albrecht Weinert
weinert-automation.de           a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 253 9.10.2023
Rev. 104 05.02.2018 : new
Rev. 154 26.06.2018 : genStateText, logStateText threadsafe; timerEndTrigger
Rev. 175 28.07.2018 : state machine structure made more clear and readable
Rev. 195 01.03.2019 : Five band checker
Rev. 198 04.03.2019 : justLogStateChg added
Rev. 200 16.04.2019 : state expanded, logging improved
Rev. 223 23.06.2020 : switchDebTickAC added
Rev. 234 05.12.2020 : cosmetic
```

7.36.2 Macro Definition Documentation

7.36.2.1 newTimer

```
#define newTimer(
              N,
              C,
              S )
```

Define a timer as state machine.

A state machine of this type "timer" is either running or inactive. It is set active or re-triggered by an amount of seconds from now. The end of the interval is stored as absolute (UTC s) time.

The check/tick function timerTickCheck would, by default be called every second at least near the end of the interval. It might as well be called more seldom if a coarser resolution is applicable and or the the timer end event shall be synchronised with other events, like e.g. second 15 in every second minute.

One strategy would be to have all timers in an array or list and check all timers every second. When this list is kept sorted to active timers with nearest end date first, one can stop checking on the first timer without change to inactive. This sorted list approach is worthwhile with many timers, only.

This macro is the initialisation expression for a timer requiring the unique name and the fitting onStateChange function, only. The other fields are preset as the timer state machine type demands.

Hint: A timer may be made periodic by restarting it accordingly in its onStateChange function.

Parameters

N	name as string literal
C	onStateChange function
S	genStateText function (NULL sets default / standard genStateText)

7.36.2.2 newSwitchDeb

```
#define newSwitchDeb(
              N,
              C,
              S,
              R )
```

Define a switch to be de-bounced as state machine.

A state machine of this type is to be provided with a switch/button input condition of either ON or OFF sampled at regular time intervals. It is constructed with a positive onThreshold (say 4) and a non-negative lower offThreshold (say 2, e.g.).
An internal counter is incremented by ON input when OFF from 0 to onThreshold. On OFF input and when ON the internal counter is decremented form offThreshold to 0.
As might have become obvious the state machine makes the OFF<->ON when these counters reach onThreshold respectively 0;

This macro is the initialisation expression for a switch de-bounce requiring the unique name and the fitting on↩ StateChange function as well as the (uint8_t) values for the off and on threshold.

Parameters

N	name as string literal
C	onStateChange function
S	offThreshold small non-negative integer
R	onThreshold small positive integer

7.36.2.3 newFloatHyst

```
#define newFloatHyst(
              N,
              C,
              O,
              R,
              S )
```

Define a hysteresis (Schmitt trigger, float value) as state machine.

A state machine of this type is to be provided with an analogue (float) value sampled regularly. It is constructed with an On and an Off threshold value. It must hold onThreshold >= offThreshold; recommended is onThreshold > offThreshold with a significant difference.

The significance of status,subStatus for this state machine is:
0,0 : Off, below
1,0 : On, above
1,1 : On, within from above
0,1 : Off, within from below

0,3 : do not know the level (pre-trigger, reset state)
0,2 : do not know On or Off, within from reset state

As might have become obvious the state machine makes a transition to ON (1,0) when the sampled value becomes > onThreshold and to Off (0,0) when <= offThreshold.

The state change call back function will be called on the following transitions:
To "Off, below" (0,0) from: 1,x and (0,2)
To "On, above" (1,0) from: 0,x and
To "Undefined, between" (0,2) from: reset (0,3)

Note: The call back function might ignore the subStatus; this would take "Undefined, between" as Off and one might see two consecutive transitions to "Off".

This macro is the initialisation expression for a float value hysteresis requiring the unique name and the fitting onStateChange function as well as the (float) values for the off and on threshold.

Parameters

N	name as string literal
C	onStateChange function
O	offThreshold valid float (i.e. non NaN)
R	onThreshold valid float (R >= S)
S	genStateText function (NULL sets default / standard genStateText)

7.36.2.4 newFiveBand

```
#define newFiveBand(
              N,
              C,
              V,
              T,
              R,
              S )
```

Define a five band checker.

A state machine of this type is to be fed with an analogue (float) value sampled regularly. This value is compared to four thresholds or borders separating those five bands:
...badLO | critLo | OK | critHi | badHi...
The state machine is to be constructed with four thresholds the un-equality
`::threshBadLo < ::threshCritLo < ::threshON < ::threshOFF`

must hold for. For VDE-AR-N 4105 e.g. one would take
47.7 49.5 -OK- 50.5 51.5 Hz resp. 184 207 -OK- 253 264 V
as thresholds.

The significance of status values for this state machine are:
2: bad Hi
1: critical Hi
0: OK

5: critical Lo
6: bad Lo
8: unknown / reset
subStatus is used as previous state. Hence 8,8 ist the reset state.

The state change call back function will called on all status transitions. realSecOff will be the time stamp of the last transition into OK (0) and realSecOn of the last transition out of it,

This macro is the initialisation expression for a five band checker requiring the unique name and the fitting on↩ StateChange function as well as legal (float) values for the four thresholds.

Parameters

N	name as string literal
C	onStateChange function
V	threshBadLo valid float (i.e. non NaN)
T	threshCritLo
R	onThreshold valid float
S	offThreshold (S > R > T > V)

7.36.2.5 newSeqCont

```
#define newSeqCont(
                N,
                C,
                n,
                m,
                S )
```

Define a sequential function chart as state machine.

A state machine of this type is simplified two leg sequential function chart with one leg leading via intermediate steps form OFF to ON state and the other leg the other way.

The OFF state is: state_t.status == 0 and state_t.subStatus == 0
Chain to ON is: state_t.status == 0 and state_t.subStatus == 1..n-1
The ON state is: state_t.status == 1 and state_t.subStatus == 0
Chain to OFF is: state_t.status == 1 and state_t.subStatus == 1..m-1

Chain to ON interrupted by seqContDoLeave changes status from 0 to 4.
Chain to OFF interrupted by seqContDoEnter changes status from 1 to 5.

Parameters

N	name as string literal
C	onStateChange function
n	length of chain to ON
m	length of chain to OFF
S	genStateText function (NULL sets default / standard genStateText)

7.36.3 Typedef Documentation

7.36.3.1 enterState_t

```
typedef uint8_t(* enterState_t) (state_t *const me, uint32_t controlV)
```

A state machine's own function to command entry in active state.

Note: The enter function put in this function pointer may (and should) be called directly on fitting events. Nevertheless, the function put in this function pointer may be used by others, especially by a check/trigger function. Then, be sure not to have the wrong (default) function here.

Parameters

me	pointer to the machine itself; never null
controlV	controlValue or extra command for the state; the usage and semantic of this parameter depends on the state machine type

Returns

0: OK, state now or already entered respectively on its way else: entering the state not possible or inhibited. Depending on the concrete type the value might give the reason.

7.36.3.2 leaveState_t

```
typedef uint8_t(* leaveState_t) (state_t *const me, uint32_t controlV)
```

A state machine's own function to command exit out of active state.

Note: See note at enterState_t.

Parameters

me	pointer to own state; never null
controlV	controlValue or extra command for the state; the usage and semantic of this parameter depends on the state implemented

Returns

0: OK state is now or was already inactive (or on its way there) else: leaving the state not possible or inhibited. Depending on the concrete type the value might give the reason.

7.36.3.3 tickCheckState_t

```
typedef uint8_t(* tickCheckState_t) (state_t *const me, uint32_t controlV)
```

A machine's own function to be called to trigger / check state.

Note: See note at enterState_t.

Parameters

me	pointer to the machine itself; never null never null
controlV	additional control value (float for tickCheckStateF_t)

Returns

0: state or, if applicable, sub-state changed; else: not

7.36.3.4 onStateChange_t

```
typedef void(* onStateChange_t) (state_t *const me)
```

The applications's call back function for state changes.

This function will only be called when (set and) the own state_t.status really changed. Depending on type (see state_t.typ), this function may also be called when the own state_t.subStatus changed.

Parameters

me	pointer to the machine itself

7.36.3.5 genStateText_t

```
typedef void(* genStateText_t) (char *stateText, state_t const *const me, char const *stamp)
```

Generate text for state machine status.

This is the minimal common standard for all status machines.

Parameters

stateText	a character array supplied to hold the state text to be generated; minimal length 80.
me	pointer to own state; never null!
stamp	(time) stamp to be prepended (max. length 23); default " - "

7.36.4 Function Documentation

7.36.4.1 setGenStateText()

```
void setGenStateText (
            state_t *const me,
            genStateText_t const fun )
```

Set the function to generate text for state machine status.

Parameters

me	pointer to own state machine
fun	the new text generator function

7.36.4.2 startStateText()

```
void startStateText (
            char * stateText,
            state_t const *const me,
            char const * stamp )
```

Generate status text standard start.

This sets the common standard start of a status text in state text, like:

```
//0123456789x123456789v123456789t123456789q1
 " 2019-04-26 03:08:26.278 # befRiseTimer: "
```

It ends with a blank at [40] after the colon at [39]. State machine type or instance specific text may be added from [40] or [41] up to (recommended) [76] followed by a terminating 0.

Hint: This function sets a 0 at [41] for robustness, i.e. having stateText always as string.

Parameters

stateText	a character array supplied to hold the state text to be generated; minimal length 80.
me	pointer to own state; never null
stamp	(time) stamp to be prepended (max. length 23); default " - "

7.36.4.3 genStateText()

```
void genStateText (
            char * stateText,
```

```
            state_t const *const me,
            char const * stamp )
```

Generate text for state machine status.

This is the minimal common standard for all status machines.

Parameters

stateText	a character array supplied to hold the state text to be generated; minimal length 80.
me	pointer to own state; never null!
stamp	(time) stamp to be prepended (max. length 23); default " - "

7.36.4.4 logStateText()

```
void logStateText (
            state_t const *const me,
            char const * stamp )
```

Log status text.

The status as text will be generated in and then be output to outLog. outLog will be flushed.

Parameters

me	pointer to own state; not null
stamp	(time) stamp to be prepended (max. length 23); default " - "

7.36.4.5 logStateReason()

```
void logStateReason (
            state_t const *const me,
            char const * stamp,
            char const * cause )
```

Log status text with cause and info.

The status as text will be generated and then be output to outLog. outLog will be flushed.
The text will be:
stamp # state machine name: cause me->infoTxt
me->infoTxt has to be set/ provided by application software and will be output to a maximum / recommended length of 29. A standard format is one field of 7 and two fields of ten characters separated by spaces.

Parameters

me	pointer to own state; not null
stamp	(time) stamp to be prepended (max. length 23); default " - "
cause	the cause of the state change (max. length 6); default me->controlVS

7.36.4.6 timerTickCheck()

```
uint8_t timerTickCheck (
            state_t *const me,
            uint32_t controlV )
```

Timer trigger.

Parameters

me	pointer to own state; not null
controlV	current time stamp; mostly cycTaskEventData_t.realSec resp. getAbsS()

Returns

0: state changed, i.e. timer ended; 2: still running; 4: inactive

7.36.4.7 timerDoEnter()

```
uint8_t timerDoEnter (
            state_t *const me,
            uint32_t secFromNow )
```

Timer entry.

This function starts an inactive timer to end secFromNow s. It does nothing on a timer already active resp. running, especially, it does not re-trigger / prolong the timer's time.

This is the function set as (default) state_t.doEnter and not timerDoStart.

Parameters

me	pointer to own state; not null
secFromNow	time to run on from now (s)

Returns

0: OK, timer no or already ON

7.36.4.8 timerDoStart()

```
uint8_t timerDoStart (
            state_t *const me,
            uint32_t secFromNow )
```

Timer unconditional entry and set.

This function starts an inactive timer. The timer's end time will be set to (now + secFromNow s) no matter the timers previous state. This unconditional changing the end time of a timer already running is the difference to timerDoEnter.

This is not the function set as (default) state_t.doEnter; it is timerDoEnter.

Parameters

me	pointer to own state; not null
secFromNow	time to run on from now (s)

Returns

0: now started; 1: was running, probably end time changed

7.36.4.9 timerDoStart4ever()

```
uint8_t timerDoStart4ever (
            state_t *const me )
```

Timer unconditional entry and set forever or stop it.

This function starts an inactive timer. The timer's end time will be set to 2.2.2106 no matter the timers previous state. This date is considered as for ever in our 21st century. This does effectively stop the timer.

This function is intended to start or keep running a timer not to end before its time is to be set (by timerDoStart) to a sensible end time.
One use case is: A timer running forever is in error state.

This function is, of course, faster than timerDoEnter and timerDoStart.

Parameters

me	pointer to own state; not null

Returns

0: now started; 1: was running, probably end time changed

7.36.4.10 timerDoTrigger()

```
uint8_t timerDoTrigger (
            state_t *const me,
            uint32_t secFromNow )
```

Timer entry or (pro-longing) re-trigger.

This function starts an inactive timer to end secFromNow s. IF the timer is active and if (now + secFromNow) is later than the current end, the timer me's runtime will be prolonged accordingly (timer re-trigger).

Parameters

me	pointer to own state; not null
secFromNow	time to run on from now (s)

Returns

0: OK, timer starter or prolonged to new (later) time 1: running timer not prolonged

7.36.4.11 timerEndTrigger()

```
uint8_t timerEndTrigger (
            state_t *const me,
            uint32_t const secUTCend )
```

Timer entry or (pro-longing) re-trigger to absolute UTC end.

If secUTCend is now or in the past or if it is equal the the end of the already active timer, nothing will be done (returns 1).

This function starts an inactive timer to end at secUTCend. If the timer is active already secUTCend will be taken as new end time.

Parameters

me	pointer to own state; not null
secUTCend	end time

Returns

0: OK (started or re-triggered); 1: no state change (on and no prolonging, or secUTCend in past)

7.36.4.12 timerDoLeave()

```
uint8_t timerDoLeave (
            state_t *const me,
            uint32_t ignored )
```

Timer leave, that is stop timer.

Note: This function is usable for other state machine types, too, if appropriate.

Parameters

me	pointer to own state; not null
ignored	as the name says

Returns

0: state changed, 1: state hold

7.36.4.13 switchDebTick()

```
uint8_t switchDebTick (
            state_t *const me,
            uint32_t controlV )
```

Switch de-bounce trigger.

Parameters

me	pointer to own state; not null
controlV	0: input OFF else: input ON

Returns

0: state changed, 1: state hold

See also

newSwitchDeb

7.36.4.14 switchDebTickAC()

```
uint8_t switchDebTickAC (
            state_t *const me,
            uint32_t controlV )
```

Switch de-bounce trigger AC.

This function does essentially the same as switchDebTick, except for On ticks being counted twice. This is meant for half wave rectified AC signals sampled at multiples of their frequency. Obviously, half or 1/2 + 1 of the samples will always be Off. And, of course, the state machine has to be made with a on chain length being by the number of samples per period higher than the normal switch de-bounce filter time, to avoid spiky ON results.

Parameters

me	pointer to own state; not null
controlV	0: input OFF else: input ON

Returns

0: OFF, 1: ON

See also

newSwitchDeb

7.36.4.15 switchDebDoEnter()

```
uint8_t switchDebDoEnter (
            state_t *const me,
            uint32_t controlV )
```

Switch de-bounce turn / force ON.

Parameters

me	pointer to own state; not null
controlV	irrelevant but recorded ob state change

Returns

0: state changed, 1: state hold

7.36.4.16 switchDebDoLeave()

```
uint8_t switchDebDoLeave (
            state_t *const me,
            uint32_t controlV )
```

Switch de-bounce turn / force OFF.

Note: this function is usable for other non timer stati, too;

Parameters

me	pointer to own state; not null
controlV	irrelevant but recorded ob state change

Returns

0: state changed, 1: state hold

7.36.4.17 floatHystTick()

```
uint8_t floatHystTick (
            state_t *const me,
            float controlV )
```

Float value hysteresis trigger.

Parameters

me	pointer to own state; not null
controlV	the sampled analogue value

Returns

0: state change, 1: no change, 0xFF: fault i.e. controlV is NaN (no state change)

7.36.4.18 floatHystDoEnter()

```
uint8_t floatHystDoEnter (
            state_t *const me,
            float analogueVal )
```

Float value hysteresis turn / force ON.

Parameters

me	pointer to own state; not null
analogueVal	irrelevant, but recorded

Returns

0: OK, state now ON; 0xFF: fault (me is NULL e.g.)

7.36.4.19 floatHystDoLeave()

```
uint8_t floatHystDoLeave (
            state_t *const me,
            float analogueVal )
```

Float value hysteresis turn / force OFF.

Note: this function is usable for other non timer stati, too;

Parameters

me	pointer to own state; not null
analogueVal	irrelevant, but recorded; may take an (uint32_t) cast float

Returns

0: state change, 1: no change

7.36.4.20 fiveBandTick()

```
uint8_t fiveBandTick (
            state_t *const me,
            float controlV )
```

Five band checker trigger.

Parameters

me	pointer to own state; not null
controlV	the sampled analogue value

Returns

0: state change, 1: no change

7.36.4.21 fiveBandDoEnter()

```
uint8_t fiveBandDoEnter (
            state_t *const me,
            float analogueVal )
```

Five band checker turn / force ON.

This function puts the checker in state bad Hi (2) no matter the parameter value.

Parameters

me	pointer to own state; not null
analogueVal	irrelevant, but recorded on state change

Returns

0: state change, 1: no change

7.36.4.22 fiveBandDoLeave()

```
uint8_t fiveBandDoLeave (
            state_t *const me,
            float analogueVal )
```

Five band checker turn / force OFF.

This function puts the checker in state OK (0) no matter the parameter value.

Parameters

me	pointer to own state; not null
analogueVal	irrelevant, but recorded on state change

Returns

0: state change, 1: no change

7.36.4.23 seqContDoEnter()

```
uint8_t seqContDoEnter (
            state_t *const me,
            char const * startCommand )
```

Sequential control entry.

This function starts the sequence from OFF to ON.

Parameters

me	pointer to own state; not null
startCommand	the cause of the state change; 6 characters (max.) to be recorded and else irrelevant for his function

Returns

0: OK state now ON or on its way there; 1: already ON

7.36.4.24 seqContDoLeave()

```
uint8_t seqContDoLeave (
            state_t *const me,
            char const * stopCommand )
```

Sequential control leave.

This function starts the sequence from ON to OFF.

Parameters

me	pointer to own state; not null
stopCommand	It is recorded on state changes and else irrelevant for this function; see description on seqContTick

Returns

0: OK state now OFF or on its way there; else: other inhibit condition

7.36.4.25 seqContTick()

```
uint8_t seqContTick (
            state_t *const me )
```

Sequential control tick or check.

If this sequential control (me) is not in a stable OFF or ON state, that is state_t.status is 0 or 1 and state_t.subStatus is 0, this function should be called at regular intervals or on relevant conditions state changes.

The function must keep or advance the state.

It must react on interrupts by seqContDoEnter or seqContDoLeave recognisable by state_t.status == 5 respectively 4. Note: This is just for the case, that individual the call back function state_t.onStateChange does not handle this "interrupts" — what it really should.

This (basic) implementation advances the sub state from 1 to n-1 respectively m-1. Interrupts by seqContDoEnter or seqContDoLeave are handled by changing to sub state 1 of the opposite leg.

When this basic implementation is not sufficient for a concrete SFC, the application may provide an own tick/check function. However, in most cases seemingly complicated cases — nonlinear chains, wait conditions etc. — the specialised behaviour can most often be implemented by an individual state_t.onStateChange call back function.

Parameters

me	pointer to own state; not null

Returns

0: state or sub-state changed 0xFF: fault status panic ... else: current state kept, of course

7.37 include/weUSBscan.h File Reference

USB 1D / 2D scanners mimicking keyboards on Raspberry Pi.

```
#include "sysBasic.h"
#include <wchar.h>
#include <locale.h>
```

Variables

- char const deRawCoDisAltGrph [120]

  Key position number to character, AltGr, German keyboard.
- char const deRawCoDisNoShift [90]

  Key position number to character, no shift, German keyboard.
- char const deRawCoDisShifted [90]

  Key position number to character, shifted, German keyboard.
- wchar_t raw2wcharAltGrph [102]

  Key position number to character, with AltGR.
- wchar_t raw2wcharNoShift [60]

  Key position number to character, no shift.
- wchar_t raw2wcharShifted [60]

  Key position number to character, with shift.
- unsigned char scanKeyAction [32]

  The one keystroke read buffer.
- int scanKeybLang

  The keyboard language.
- wchar_t scanResult [162]

  The result of one scan.
- char const usRawCoDisNoShift [90]

  Key position number to character, no shift, US keyboard.
- char const usRawCoDisShifted [90]

  Key position number to character, shifted, US keyboard.

7.37.1 Detailed Description

USB 1D / 2D scanners mimicking keyboards on Raspberry Pi.

```
Copyright  (c)   2020     Albrecht Weinert
weinert-automation.de     a-weinert.de
    /         /      /\
   /         /___    /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \     /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 236 2.02.2021
Rev. 232 03.09.2020 : new (extracted from testOnPi.c)
Rev. 234 05.12.2020 : cosmetic changes
```

This is a supplementary basic library to handle USB barcode and QR code scanners. By plug'n play such scanner would normally appear as device

```
/dev/hidraw0
```

. To make this usable for scanner application programs run without sudo apply

```
sudo chmod  664 /dev/hidraw0
```

before.

Devices

The only device used and tested so far is a "USB Wired 2D Barcode Scanner" <MJ-8200>. It seems to ha a lot of bethren with similar software. Emulated non-US keyboards seem alien to the developers; see deRawCoDisNoShift.

7.37.2 Variable Documentation

7.37.2.1 usRawCoDisNoShift

```
char const usRawCoDisNoShift[90]  [extern]
```

Key position number to character, no shift, US keyboard.

This utf-8 or multibyte character array respectively string describes the translation of key number to character for a US keyboard.
It will have to be transfered as wide character array to raw2wcharNoShift.

7.37.2.2 usRawCoDisShifted

```
char const usRawCoDisShifted[90]  [extern]
```

Key position number to character, shifted, US keyboard.

See the explanation at usRawCoDisNoShift.

See also

raw2wcharShifted

7.37.2.3 deRawCoDisNoShift

```
char const deRawCoDisNoShift[90]  [extern]
```

Key position number to character, no shift, German keyboard.

This utf-8 or multibyte character array respectively string describes the translation of key number to character for a US keyboard.
It will have to be transfered as wide character array to raw2wcharNoShift.

Remarks on non US keyboard emulations by the "USB Wired 2D Barcode Scanner" <MJ-8200> and consorts:
We strongly recommend not to use them and refrain from applications using more than primitive USASCII. As "↩ German keyboard", e.g., the scanner does neither recognise nor send . Besides being called German without umlauts [sic!] the scanner is ignorant to some other characters on every German keyboard (which are probably there because of being used in Western Europe). Additionally the scanner when set to German inserts additional strings of characters with no obvious sense or system.
In the end we consider everything beyond factory reset (except low beeper volume) as not functional.
Without the hard bug of inventing characters and string not contained in the QR-code, the so called German keyboard would give us some extra characters — but not umlauts.

7.37.2.4 deRawCoDisShifted

```
char const deRawCoDisShifted[90]  [extern]
```

Key position number to character, shifted, German keyboard.

See the explanation at usRawCoDisNoShift.

See also

raw2wcharShifted deRawCoDisNoShift

7.37.2.5 deRawCoDisAltGrph

```
char const deRawCoDisAltGrph[120]  [extern]
```

Key position number to character, AltGr, German keyboard.

See the explanation at usRawCoDisNoShift.

See also

raw2wcharShifted deRawCoDisNoShift

7.37.2.6 raw2wcharNoShift

```
wchar_t raw2wcharNoShift[60]  [extern]
```

Key position number to character, no shift.

The length is 60. Valid characters are in the 4..56 key number range; there may be gaps. 0..3 are errors.

See also

usRawCoDisNoShift

7.37.2.7 raw2wcharShifted

```
wchar_t raw2wcharShifted[60]  [extern]
```

Key position number to character, with shift.

The length is 60. Valid characters are in the 4..56 key number range; there may be gaps. 0..3 are errors.

See also

usRawCoDisShifted

7.37.2.8 raw2wcharAltGrph

```
wchar_t raw2wcharAltGrph[102]  [extern]
```

Key position number to character, with AltGR.

The length is 60. Valid characters are in the 4..56 key number range; there will be many gaps. 0..3 are errors.

See also

usRawCoDisShifted

7.37.2.9 scanKeybLang

```
int scanKeybLang  [extern]
```

The keyboard language.

The language of the USB keyboard (see scanKeyAction) emulated by the scanner is stored here as: 0=US (default), 10=DE

7.37.2.10 scanResult

```
wchar_t scanResult[162]  [extern]
```

The result of one scan.

The result of consecutive keystrokes (see scanKeyAction) is stored here as array respectively string of wide characters.

7.37.2.11 scanKeyAction

```
unsigned char scanKeyAction[32]  [extern]
```

The one keystroke read buffer.

Keyboard input comes in blocks of 8 bytes each. Hence a length of 8 would be sufficient. Hence, 32 is a reserve for device errors or the driver not recognising the gap between blocks.

The 8 bytes are:

```
              Bit/Value:  0/1 1/2   2/4 3/8         4/16 5/32  6/64  7/128
[0]  Modifier keys Left: cntl shift Alt Win  Right: cntl shift AltGr Win
[1]  Reserved field                 always 0
[2]  Keypress 1                     in a funny code (4 is a)
[3]  2nd simultaneously pressed key
[4..7]  Keypress 3..6
```

As scanners won't "press" more than one key at a time only bytes [0] and [1] will contain information.
Byte[2] will be a crazy key code for a..z1..90... athwart to any utf or unicode. In the end the semantic of that "code" is a mixture of key value and key position on the keyboard. In the end few scanners get more than an American (and a Chinese?) keybord right. When setting the scanner to German keyboard you may miss one or two of .

On byte [0] one should see only three values: /code 0 : no modifier, no shift 2 : shift, that means a->A 64: altGr

7.38 include/weUtil.h File Reference

Some system related time and utility functions for Raspberry Pis.

```
#include "sysBasic.h"
#include <fcntl.h>
#include <unistd.h>
#include <sys/file.h>
#include <signal.h>
#include <stdlib.h>
#include <pthread.h>
```

Data Structures

- struct cycTask_t

  Cyclic or event driven task / threads structure.
- struct cycTaskEventData_t

  Event data for cyclic tasks.

Functions

- int advanceTmTim (struct tm *rTm, char *rTmTxt, uint8_t sec)

  Advance broken down real time by seconds.
- int char2hexDig (char c)

  Character to hexadecimal.
- int cycTaskDestroy (cycTask_t *cykTask)

  Destroy a cyclic task / threads structure.
- int cycTaskEvent (cycTask_t *cycTask, uint8_t noEvents, timespec stamp, cycTaskEventData_t cycTask↩EventData)

  Handle and signal events.
- int cycTaskInit (cycTask_t *cykTask)

  Initialise a cyclic task / threads structure.
- int cycTaskWaitEvent (cycTask_t *cycTask, uint32_t eventsThreshold, cycTask_t *cycTaskSnap)

  Wait on signalled event.
- int endCyclist (void)

  The cycles handler arrived.
- char * formFixed16 (char *target, uint8_t targetLen, uint16_t value, uint8_t dotPos)

  Format 16 bit unsigned fixed point, right aligned.
- char * formFixed32 (char *target, uint8_t targetLen, uint32_t value, uint8_t dotPos)

  Format 32 bit unsigned fixed point, right aligned.
- int genErrWithText (char const *txt)

  Generate error text (errorText) with system error text appended.
- uint8_t get10inS ()

  Get a 10th of second in s reading.
- uint32_t getAbsS ()

  Get the absolute s reading.
- uint32_t getCykTaskCount (cycTask_t const *const cycTask)

  Get a cycle's/task's current event counter.
- uint16_t getMSinS ()

Get a ms in s reading.

- void initStartRTime ()

  Initialise start (real) time.

- uint32_t ioSetClrSelect (uint8_t pin)

  Fetch a clear and set select bit for a GPIO pin.

- int isValidIp4 (char const *str)

  Check if a string is a valid IPv4 address.

- void logErrorText (void)

  Log the (last) common error text generated.

- void logErrText (char const *txt)

  Log an error text on errLog.

- void logErrWithText (char const *txt)

  Log error text (on errLog) with system error text appended.

- void logStampedText (char const *txt)

  Log an event or a message on outLog as line with time stamp.

- void monoTimeResol (timespec *timeRes)

  Absolute time (source) resolution.

- void onSignalExit (int s)

  On signal exit.

- void onSignalExit0 (int s)

  On signal exit 0.

- void onSignalStop (int s)

  On signal stop.

- unsigned int parse2Long (char *const optArg, long int *parsResult)

  Parse a string of integer numbers.

- int parsInt (const char *str, const int lower, const int upper, const int def)

  Parse int with checks.

- uint16_t stopMSwatch ()

  Get a (stop-watch) ms reading.

- void strLappend (char *dest, char const *src, int n)

  Append one char sequence left justified at another one.

- void strLinto (char *dest, char const *src, size_t n)

  Set one char sequence left justified into another one.

- void strRinto (char *dest, char const *src, size_t n)

  Set one char sequence right justified into another one.

- int theCyclistStart (int startMsDelay)

  Start the cycles handler.

- int theCyclistWaitEnd ()

  Wait for the end of the cycles thread.

- timespec timeAdd (timespec const t1, timespec const t2)

  Add two times as new structure.

- void timeAddTo (timespec *t1, timespec const t2)

  Add two times overwriting the first operand.

- int timeCmp (timespec const t1, timespec const t2)

  Compare two times.

- int timeSleep (unsigned int micros)

  Relative delay for the specified number of s.

Variables

- char actRTmTxt [ ]

  Actual broken down time (text).
- timespec allCycStart

  Common absolute / monotonic start time of all cycles.
- char const **bin8digs** [256][10]

  "0000_0000" .. "1111_1111"
- volatile uint8_t commonRun

  Common boolean run flag for all threads.
- const uint32_t **csBit** [ ]

  single bit set. 1 2 4 8 ... 0x80000000
- cycTask_t **cyc100ms**

  100ms cycle (data structure)
- cycTask_t **cyc10ms**

  10ms cycle (data structure)
- cycTask_t **cyc1ms**

  1ms cycle (data structure)
- cycTask_t **cyc1sec**

  1s cycle (data structure)
- cycTask_t **cyc20ms**

  20ms cycle (data structure)
- char errorText [182]

  Common error text.
- uint8_t have100msCyc

  Flag to enable the 100ms cycle.
- uint8_t have10msCyc

  Flag to enable the 10ms cycle.
- uint8_t have1msCyc

  Flag to enable the 1ms cycle.
- uint8_t have1secCyc

  Flag to enable the 1s cycle.
- uint8_t have20msCyc

  Flag to enable the 20ms cycle.
- long int parsResult [ ]

  Long array of length 14.
- volatile int sigRec

  Storage for the signal (number) requesting exit.
- timespec startRTime

  Start time (structure, monotonic real time clock).
- char const ∗const stmp23

  The current time as text.
- uint32_t const ∗const stmpSec

  The real time epoch seconds.
- int8_t **vcoCorrNs**

  external for test/debug only (don't change)

7.38.1 Detailed Description

Some system related time and utility functions for Raspberry Pis.

```
Copyright  (c)  2018   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /      \|   |
     \/  \/   \__/        \__/|_
```

Revision history

```
Rev. 244 28.09.2022
Rev. 50+ 16.10.2017 : cycTask_t->mutex now pointer (allows common mutex)
Rev. 54+ 23.10.2017 : timing enhanced, common mutex forced/standard
Rev. 200 16.04.2019 : logging improved; formatting enh.
Rev. 201 26.04.2019 : renamed from sysUtil.h
Rev. 202 28.04.2019 : timer macro change, more formatting functions
Rev. 209 22.07.2019 : work around a Doxygen bug, formFixed.. not void
Rev. 233  26.09.2020 : 10 ms cycle added
```

This file contains some definitions concerning system, time and IO. The IO part will work with the gpio/gpiod library as defined in the library's include file pigpiod_if2.h.

7.38.2 Function Documentation

7.38.2.1 strLinto()

```
void strLinto (
            char * dest,
            char const * src,
            size_t n )
```

Set one char sequence left justified into another one.

This function copies n characters from src to dest left justified. If the length of src is less than n the remaining length on right in dest will be filled with blanks.

Attention: dest[n-1] must be within the char array provided by dest. This cannot and will not be checked!

Hint: Contrary to strncpy there's no padding with 0. If you want dest to end after the insertion use strLappend().

Parameters

dest	the pointer to / into the destination sequence where src is to be copied to. If NULL nothing happens.
src	the sequence to be copied. If NULL or empty fill is used from start
n	the number of characters to be copied from src.

7.38.2.2 strLappend()

```
void strLappend (
            char * dest,
```

```
            char const * src,
            int n )
```

Append one char sequence left justified at another one.

This function copies n characters from src to dest and lets dest then end with 0 (end of string). If n is negativ, -n characters will be copied and dest will end with new line and 0;

Attention: dest[n] respectively dest [-n + 1] must be within the char array provided by dest. This cannot and will not be checked!

Parameters

dest	the pointer to / into the destination sequence where src is to be copied to. If Null nothing happens.
src	the sequence to be copied. If Null or empty fill is used from start
n	the absolute value is the number of characters to be copied from src. If this number exceeds 300 it will be taken as 0. If n is negative a line feed will be appended, too.

7.38.2.3 strRinto()

```
void strRinto (
            char * dest,
            char const * src,
            size_t n )
```

Set one char sequence right justified into another one.

This function copies n characters from src to dest right justified. If the length of src is less than n the remaining length on left in dest will be filled with blanks.

Attention: dest[n-1] must be within the char array provided by dest. This cannot and will not be checked!

Hint to append instead of insert: If this operation shall be at the end of the changed char sequence do
dest[n] = 0;
Then, of course, dest[n] must be within the char array provided by dest.

Parameters

dest	the pointer to / into the destination sequence where src is to be copied to. If NULL nothing happens.
src	the sequence to be copied. If NULL or empty fill is used from start
n	the number of characters to be copied from src.

7.38.2.4 formFixed16()

```
char * formFixed16 (
            char * target,
            uint8_t targetLen,
```

```
            uint16_t value,
            uint8_t dotPos )
```

Format 16 bit unsigned fixed point, right aligned.

formFixed16(target, 6, 1234, 2), e.g., will yield " 12.34".
formFixed16(target, 6, 4, 2), e.g., will yield " 0.04".

If the value would not fit within targetLen characters leading digits will be truncated.

Parameters

target	pointer to first of targLen characters changed
targetLen	field length 2..16; number of characters changed
value	the fixed point value
dotPos	where the fixed point is 0..6 < targetLen

Returns

points to the most significant digit set or NULL on error / no formatting

7.38.2.5 formFixed32()

```
char * formFixed32 (
            char * target,
            uint8_t targetLen,
            uint32_t value,
            uint8_t dotPos )
```

Format 32 bit unsigned fixed point, right aligned.

This function behaves like formFixed16() except for handling 32 bit values. formFixed16() should be preferred, when feasible.

Parameters

target	pointer to first of targLen characters changed
targetLen	field length 2..16; number of characters changed
value	the fixed point value
dotPos	where the fixed point is 0..6 < targetLen

Returns

points to the most significant digit set or NULL on error / no formatting

7.38.2.6 isValidIp4()

```
int isValidIp4 (
            char const * str )
```

Check if a string is a valid IPv4 address.

Syntactically valid IPv4 addresses are: 0.0.0.0 .. 255.255.255.255

Parameters

str	The string containing the address, only; 0-terminated

Returns

0: no syntactically valid IPv4 address; 1: OK

7.38.2.7 char2hexDig()

```
int char2hexDig (
            char c )
```

Character to hexadecimal.

Parameter values '0'..'9' return 0..9. Parameter values 'A'..'F' and 'a'..'f' return 10..15. Other values return -1.

7.38.2.8 parsInt()

```
int parsInt (
            const char * str,
            const int lower,
            const int upper,
            const int def )
```

Parse int with checks.

This function expects parameter str to point to a null-terminated string. If not def is returned. If lower > upper def is returned. If the string str contains a decimal integer number n, fulfilling lower <= n <= upper n is returned, or def otherwise.

If the string str starts with [+|-][min|med|max] ignoring case lower respectively ((lower + upper) / 2) respectively upper is returned. A leading sign (+|-) as well as any trailing characters are ignored.

Parameters

str	0-terminated string containing a decimal integer number, or one of the keywords described above
lower	lower limit
upper	upper limit
def	default value, to be returned when str is not a pure decimal number one of the keyword starts or when the result violates the limits

7.38.2.9 parse2Long()

```
unsigned int parse2Long (
            char *const optArg,
            long int * parsResult )
```

Parse a string of integer numbers.

The string optArg will be tokenised taking any occurrences of the characters " +,;" (blank, plus, comma, semicolon) as border. "Any occurrences" means two commas „,", e.g., acting as one separator and not denoting an empty number.
N.b.: The string optArg will be modified (by replacing the first character of the token separators found by zero ('\0').

The number format accepted and parsed is decimal and hexadecimal. Hexadecimal starts with 0x or 0X. Leading zeros have no significance (in C stone age sense of being octal).

Parameters

optArg	the string to be passed to a number of integer numbers, passed as program parameter, e.g.
parsResult	pointer to an array of long int, minimal length 14 (!)

Returns

the number of integer numbers parsed ab put into parsResult[], 0..14

7.38.2.10 timeAdd()

```
timespec timeAdd (
            timespec const t1,
            timespec const t2 )
```

Add two times as new structure.

Parameters

t1	summand as time structure (not NULL!, will be left unchanged)
t2	the second summand (dto.)

Returns

the sum (probably passed as hidden parameter by the way)

7.38.2.11 timeAddTo()

```
void timeAddTo (
            timespec * t1,
            timespec const t2 )
```

Add two times overwriting the first operand.

Parameters

t1	the time structure to add to (not NULL!, will be modified)
t2	the summand (not NULL!, will be left unchanged)

7.38.2.12 timeCmp()

```
int timeCmp (
            timespec const t1,
            timespec const t2 )
```

Compare two times.

Parameters

t1	the time structure to compare to t2 (not NULL!)
t2	the time structure to compare t1 with (not NULL!)

Returns

0: equal; +: t1 is greater (2 by s, 1 by ns); -: t1 is smaller

7.38.2.13 monoTimeResol()

```
void monoTimeResol (
            timespec * timeRes )
```

Absolute time (source) resolution.

This function sets the time structure provided to the absolute time's (ABS_MONOTIME default: CLOCK_↩ MONOTONIC) resolution.

Raspian Jessie on a Raspberry Pi 3 always yielded 1ns, which one may believe or not. We took it as "sufficient for accurate 1ms cycles".

Parameters

timeRes	the time structure to be used (never NULL!)

7.38.2.14 timeSleep()

```
int timeSleep (
            unsigned int micros )
```

Relative delay for the specified number of s.

This is local sleep. It should not be used in combination with absolute times and cyclic threads. It is just an utility for test or very short delays (as a better replacement for spinning).

Parameters

micros	sleep time in s; allowed 30 .. 63000

Returns

sleep's return value if of interest (0: uninterrupted)

7.38.2.15 initStartRTime()

```
void initStartRTime ( )
```

Initialise start (real) time.

This will be done in theCyclistStart(int). Hence, this function is for "non-cyclic" applications, mainly. Nevertheless it can be called before theCyclistStart(int) and won't be repeated therein.

7.38.2.16 advanceTmTim()

```
int advanceTmTim (
            struct tm * rTm,
            char * rTmTxt,
            uint8_t sec )
```

Advance broken down real time by seconds.

This function just advances the broken down (local) time structure rTm and the fitting text rTmTxt by 1 to 40s. All fields not affected by adding to the seconds part, won't be touched.

This function won't care about leap seconds nor handle DST rules. If this is to be kept up to date, it is recommended to refresh it on every hour change (return >=3) by
clock_gettime(CLOCK_REALTIME,..) and localtime_r(..).
Depending on OS, that might be an expensive operation with extra locks.

With wrong parameter values this function does nothing (returns 0).

Parameters

rTm	pointer to broken down real time
rTmTxt	date text, length 32, format Fr 2017-10-20 13:55:12.987 UTC+20 NULL is substituted by actRTmTxt
sec	1..40 will be added; else: error

Returns

0: error (rTm NULL e.g.); 1: seconds changed; 2: minute; 3: hour; 4: day; 5: month ; 6: year; 7: zone offset

7.38.2.17 ioSetClrSelect()

```
uint32_t ioSetClrSelect (
             uint8_t pin )
```

Fetch a clear and set select bit for a GPIO pin.

For the masks to set or clear GPIO bits each bit 0..31 selects the GPIO pin 0..31 respectively 32..53.

Parameters

pin	GPIO pin number (only 5 bits relevant here)

Returns

the the function select bit (a value with one bit set)

7.38.2.18 logErrWithText()

```
void logErrWithText (
             char const * txt )
```

Log error text (on errLog) with system error text appended.

Gives a (English) clear text translation of the latest system stored error. If txt is not null it will be prepended. This function appends a linefeed and flushes errLog.

Parameters

txt	text to be prepended (should nod be longer than 58 characters)

7.38.2.19 genErrWithText()

```
int genErrWithText (
            char const * txt )
```

Generate error text (errorText) with system error text appended.

Gives a (English) clear text translation of the latest system stored error. If txt is not null it will be prepended. Date and time will be prepended anyway.

Parameters

txt	text to be prepended (should nod be longer than 58 characters)

Returns

0: no error; else mutex error (time and date may be spoiled)

7.38.2.20 logErrorText()

```
void logErrorText (
            void  )
```

Log the (last) common error text generated.

This function outputs the last generated errorText (by genErrWithText() e.g.) to errLog. It appends a linefeed and flushes errLog.

7.38.2.21 logErrText()

```
void logErrText (
            char const * txt )
```

Log an error text on errLog.

If txt is not null it will be output to errLog and errLog will be flushed.

Parameters

txt	text to be output; n.b not LF appended

7.38.2.22 logStampedText()

```
void logStampedText (
            char const * txt )
```

Log an event or a message on outLog as line with time stamp.

If txt is not null it will be output to outLog. A time stamp is prepended and a line feed is appended. txt will be shortened to 50 characters if longer.

Parameters

txt	the text to be output

7.38.2.23 onSignalExit()

```
void onSignalExit (
            int s )
```

On signal exit.

This function is intended as signal hook; see signal(s, hook). When called, this function calls exit(s) and never returns.

Parameters

s	the signal forwarded to exit

7.38.2.24 onSignalExit0()

```
void onSignalExit0 (
            int s )
```

On signal exit 0.

This function is intended as signal hook; see signal(s, hook).
When called this function calls exit(0) and never returns.
This may be used as hook for s==SIGIN, to provide a normal return on cntl-C.

Parameters

s	ignored

7.38.2.25 onSignalStop()

```
void onSignalStop (
            int s )
```

On signal stop.

This function is a prepared signal hook. When called it sets sigRec by s and clears commonRun.

7.38.2.26 cycTaskInit()

```
int cycTaskInit (
            cycTask_t * cykTask )
```

Initialise a cyclic task / threads structure.

This function initialises a cyclic or non cyclic (asynchronous random event driven) task (thread) structure. Common mutex and an own condition are initialised, the event counter (.count) is set to 0.

Note: For the standard cycles provided here, 1ms, 100ms .., this initialisation is done in theCyclistStart() and the destruction (by cycTaskDestroy()) in endCyclist().

Parameters

cykTask	the task structure to initialise (not NULL!)

Returns

0: success, else: one of the error codes occurred

7.38.2.27 cycTaskDestroy()

```
int cycTaskDestroy (
            cycTask_t * cykTask )
```

Destroy a cyclic task / threads structure.

Parameters

cykTask	the task structure to destroy (not NULL!)

Returns

0: success, else: one of the error codes occurred

7.38.2.28 cycTaskEvent()

```
int cycTaskEvent (
            cycTask_t * cycTask,
```

```
            uint8_t noEvents,
            timespec stamp,
            cycTaskEventData_t cycTaskEventData )
```

Handle and signal events.

This is a helper function for the controller / manager to be called when having determined, that one or more events happened.

Parameters

cycTask	the task structure (not NULL!)
noEvents	number of events (usually 1); summand to cykTask.count
stamp	absolute monotonic time of the event; sets sykTask.stamp
cycTaskEventData	actual cyclic event data

Returns

0: success, else: one of the error codes occurred

7.38.2.29 cycTaskWaitEvent()

```
int cycTaskWaitEvent (
            cycTask_t * cycTask,
            uint32_t eventsThreshold,
            cycTask_t * cycTaskSnap )
```

Wait on signalled event.

This is a helper function for a worker thread. It will return on reaching the signalled event(s) or on ! commonRun. If cykTaskSnap is not NULL cycTask will be assigned to it under mutex lock before returning. This is helpful if cykTask's events are broadcast to multiple handlers.

Parameters

cycTask	the task structure (not NULL!)
eventsThreshold	threshold for cykTask.count (update for every round)
cycTaskSnap	copy of cykTask under mutex lock before returning

Returns

0: success, else: one of the error codes occurred

7.38.2.30 getCykTaskCount()

```
uint32_t getCykTaskCount (
            cycTask_t const *const cycTask )
```

Get a cycle's/task's current event counter.

This is done under (cyclist's) mutex lock.

Parameters

cycTask	the task structure

Returns

cycTask's event counter value (.count) got under lock; 0x7FffFFffFFf7 on any error (null, lock error) situation

7.38.2.31 stopMSwatch()

```
uint16_t stopMSwatch ( )
```

Get a (stop-watch) ms reading.

This function provides an 16 bit reading of the cyclist's (64 bit) milliseconds. It is intended for measuring short (<= 1min) durations.

Hint: This functions thread safety stems from the hope of 16 bit increments being atomic. Even if no problems in this respect were observed on Raspberry Pi 3s, it may be just hope in the end. Thread-safe values are, of course, provided in the cycTaskEventData_t structure. But those are frozen within one cycle task step, and, hence, not usable as stop-watch readings within such step.

7.38.2.32 getMSinS()

```
uint16_t getMSinS ( )
```

Get a ms in s reading.

This function provides the cyclist's ms in sec as 16 bit unsigned reading. It is intended for measuring and testing durations.

Hint: This functions does nothing for thread safety. It hopes 16 bit accesses being atomic. Even if no problems in this respect were observed on Raspberry Pi 3s, it may be just hope in the end. Thread-safe values are, of course, provided in the cycTaskEventData_t structure. But those are frozen within one cycle task step, and, hence, not usable as stop-watch readings within such step.

7.38.2.33 get10inS()

```
uint8_t get10inS ( )
```

Get a 10th of second in s reading.

This function provides the cyclist's tenth in seconds reading (0..9). It is intended for cyclic tasks with times greater than 100 ms or asynchronous tasks to get a coarse second sub-division.

Hint: Cyclic tasks get this value in their task date valid at start. This function provides an actual value for tasks running longer than 100ms.

7.38.2.34 getAbsS()

```
uint32_t getAbsS ( )
```

Get the absolute s reading.

This function provides a 32 bit monotonic seconds value.
Base of this 32 bit value is the cyclist's 64 bit epoch time in seconds, 0 being 1.1.1970 00:00:00 UTC on almost all Linuxes and C libraries.

This unsigned 32 bit holds until 7. February 2106, which is far longer than the projected lifetime age of this library and of Raspberry Pi3s. (But who knows?) The value may be used for seconds-resolution, absolute (i.e. zone and DST independent) time-stamps and interval calculations (which will be incorrect with leap seconds).

Hint: Cyclic tasks get this value (.realSec) in their task date valid at tick start. Hence, this function is intended for asynchronous tasks or cyclic tasks with periods $>$ 1s.

Since version R.110 we dare to fetch this value without lock, assuming ARMv7 32 bit load and stores being atomic.

7.38.2.35 theCyclistStart()

```
int theCyclistStart (
            int startMsDelay )
```

Start the cycles handler.

This function initialises and then runs the predefined cycles cycles (as of Sept. 2020: 1ms, 10ms, 20ms, 100ms and 1s; see have1msCyc) when enabled.

Besides the absolute / monotonic times for the cycles it also initialises real time and timers handling.

Timers and cycles are run in an extra thread made by this function. And to be precise, the cycles are not run here; instead, cyclic events are generated and broadcast.

As the thread started by this function also provides monotonic and civil times and stamps it should be started with the program (i.e. earliest in main()). Preparation time before the cycles should start can be handled by the delay parameter.

As of September 2020 five cycles (see above) are defined and handled. It is strongly recommended not to use more than two of them and implement other cycles with multiple periods by sub-division. That means, e.g., do not enable the 20ms cycle when having the 10ms one.

Parameters

startMsDelay	number of ms before generating the first cyclic event; allowed range 12 .. 1200; default 1

Returns

0: after having initialised all and having made and started the cyclist thread; other values signal errors

7.38.2.36 theCyclistWaitEnd()

```
int theCyclistWaitEnd ( )
```

Wait for the end of the cycles thread.

This function does so by unconditionally joining the cyclist thread.

Returns

the return value of thread join; 0: join OK

7.38.2.37 endCyclist()

```
int endCyclist (
            void  )
```

The cycles handler arrived.

This function cleans up after theCyclist. It should be called after theCyclist() ending successfully on commonRun false. The controller thread shall call this function after having joined and cleaned up all of its threads. It may also be put in an exit hook.

Returns

0: OK; else: a cycTaskDestroy() error

7.38.3 Variable Documentation

7.38.3.1 parsResult

```
long int parsResult[]  [extern]
```

Long array of length 14.

Prepared for non thread safe use with parse2Long()

7.38.3.2 startRTime

```
timespec startRTime  [extern]
```

Start time (structure, monotonic real time clock).

By initStartRTime() or by theCyclistStart() actRTime and this startRTime will initially be set. actRTime may be updated on demand, but startRTime should be left unchanged.

7.38.3.3 actRTmTxt

```
char actRTmTxt[]  [extern]
```

Actual broken down time (text).

```
            |-3|-  10  -|1|- 12     -|
```

The format is: Fr 2017-10-20 13:55:12.987 UTC+20 .....0123456789x123456789v123456789t1234 was Fr 2017-10-20 13:55:12 UTC+20 The length is 32.

7.38.3.4 stmp23

```
char const* const stmp23  [extern]
```

The current time as text.

/code The format is: 2017-10-20 13:55:12.987 UTC+20 .....0123456789x123456789v123456789 /endcode The length is 30.
Do NOT change the value provided by this pointer.

7.38.3.5 stmpSec

```
uint32_t const* const stmpSec  [extern]
```

The real time epoch seconds.

Do NOT change the value provided by this pointer.

7.38.3.6 errorText

```
char errorText[182]  [extern]
```

Common error text.

This text is set by genErrWithText() and hence indirectly by (many) other functions optionally generating error texts.

7.38.3.7 commonRun

```
volatile uint8_t commonRun  [extern]
```

Common boolean run flag for all threads.

When set false, all threads must exit as soon as possible. On any case, a thread has to exit and clean up on next signal. Setting commonRun false implies the end of the application/program and all of its threads as soon as possible.

Initialised as 1 (true) Set 0 by onSignalStop() (or application program)

7.38.3.8 sigRec

```
volatile int sigRec  [extern]
```

Storage for the signal (number) requesting exit.

Set by: onSignalStop()
See also: retCode

7.38.3.9 allCycStart

```
timespec allCycStart  [extern]
```

Common absolute / monotonic start time of all cycles.

May be considered as program's start time when cycles are started early by theCyclist. Normally not to be modified.

7.38.3.10 have1msCyc

```
uint8_t have1msCyc  [extern]
```

Flag to enable the 1ms cycle.

As a rule no more than two oft the cycles offered — cyc1ms, cyc10ms, cyc20ms, cyc100ms, cyc1sec — shall be enabled. This is no restriction as a faster cycle can easily (and often should) implement slower cycles by subdivision.

The default setting is 1ms and 100ms ON and all others OFF.
If other settings are used the flags should be set at the program's early initialisation phase and afterwards left untouched.

default: ON

See also

cycTask_t cyc1ms

7.38.3.11 have10msCyc

```
uint8_t have10msCyc  [extern]
```

Flag to enable the 10ms cycle.

default: OFF

See also

have1msCyc cycTask_t cyc10ms

7.38.3.12 have20msCyc

```
uint8_t have20msCyc  [extern]
```

Flag to enable the 20ms cycle.

default: OFF

See also

have1msCyc cycTask_t cyc20ms

7.38.3.13 have100msCyc

```
uint8_t have100msCyc  [extern]
```

Flag to enable the 100ms cycle.

default: ON

See also

have1msCyc cycTask_t cyc100ms

7.38.3.14 have1secCyc

```
uint8_t have1secCyc  [extern]
```

Flag to enable the 1s cycle.

default: OFF

See also

have1msCyc cycTask_t cyc1s

7.39 justHeld.c File Reference

justLock by Gerard van Helden

```
#include <stdio.h>
#include <sys/file.h>
#include <unistd.h>
#include <limits.h>
#include <errno.h>
```

7.39.1 Detailed Description

justLock by Gerard van Helden

https://github.com/drm/java-filelock-respected-by-C/blob/master/lock.c

modified A. Weinert: a) F_TLOCK (w/o the program never failed but waited endlessly) b) default lockfile /home/pi/bin/.lockPiGpio c) hold time 3 -> 30s to allow handling by human with two putties (putty) or justHeld & d) if lockf() fails just take flock() On Pi translate by: g++ justHeld.c -o ~/bin/justHeld

7.40 justLock.c File Reference

A small program just to lock the piGpoi lock file.

```
#include <getopt.h>
#include "weUtil.h"
#include "weLockWatch.h"
```

Functions

- int main (int argc, char **argv)

  The program.

Variables

- char const * **lckPiGpio**

  The file path used.
- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.

7.40.1 Detailed Description

A small program just to lock the piGpoi lock file.

```
Copyright (c) 2019 Albrecht Weinert
weinert-automation.de      a-weinert.de
```

Revision history

```
Rev. 236 2.02.2021
Rev. 209 22.07.2019 : minor improvements (docu)
```

This program tries to lock the standard lock file for piGPio if it exists. On success it will run respectively sleep until getting a signal, on which it will unlock the file and terminate.

7.40.2 Function Documentation

7.40.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

The program.

run by: justLock [options] [lockFilePath]

For options see longOptions and :: optHlpTxt.

7.40.3 Variable Documentation

7.40.3.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.40.3.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.40.3.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.41 justLockWin.c File Reference

A small program just to lock the piGpoi lock file on Windows.

```
#include <stdlib.h>
#include <getopt.h>
#include <fcntl.h>
#include <sys/file.h>
#include <stdio.h>
#include <signal.h>
```

Functions

- void closeLock (void)
  Unlock the lock file.
- int justLock (char const *lckPiGpioFil)
  Open and lock the lock file.
- int main (int argc, char **argv)
  The program.
- void onSignalExit (int s)
  On signal exit.
- void onSignalExit0 (int s)
  On signal exit 0.

Variables

- char const * **lckPiGpio**
  The file path used.
- char const *const lckPiGpioPth
  Common path to a lock file for GpIO use.
- int lockFd
  Lock file handle.
- FILE * outLog
  Event log output.
- int retCode
  Basic start-up function failure.

7.41.1 Detailed Description

A small program just to lock the piGpoi lock file on Windows.

```
Copyright  (c)  2021   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 236 2.02.2021
Rev. 240 22.04.2021 : new; test for build with win-builds 1.5.0
```

This program tries to lock the standard lock file for piGPio if it exists. On success it will run respectively sleep until getting a signal, on which it will unlock the file and terminate.

Status: Experimental. Outcome: On Windows GCC (C:\util\win-builds\bin\cc.exe) provide no flock() C↩ :\util\win-builds\bin\cc.exe -c -o justLockWin.o justLockWin.c ;;; OK C:\util\win-builds\bin\cc.exe justLockWin.o –output justLockWin.exe justLockWin.o:justLockWin.c:(.text+0x19): undefined reference to 'flock' justLockWin.o↩ :justLockWin.c:(.text+0x35a): undefined reference to 'on_exit' ..... error: ld returned 1 exit status

7.41.2 Function Documentation

7.41.2.1 closeLock()

```
void closeLock (
             void  )
```

Unlock the lock file.

Unlock the lock file

7.41.2.2 justLock()

```
int justLock (
             char const * lckPiGpioFil )
```

Open and lock the lock file.

This function is the basic implementation of openLock. Applications not wanting its optional logging or doing their own should use this function directly.

Parameters

lckPiGpioFil	lock file name

Returns

0: OK, locked; 97: fd does not exist; 98: can't be locked

7.41.2.3 onSignalExit()

```
void onSignalExit (
            int s )
```

On signal exit.

This function is intended as signal hook; see signal(s, hook). When called, this function calls exit(s) and never returns.

Parameters

s	the signal forwarded to exit

7.41.2.4 onSignalExit0()

```
void onSignalExit0 (
            int s )
```

On signal exit 0.

This function is intended as signal hook; see signal(s, hook).
When called this function calls exit(0) and never returns.
This may be used as hook for s==SIGIN, to provide a normal return on cntl-C.

Parameters

s	ignored

7.41.2.5 main()

```
int main (
            int argc,
            char ** argv )
```

The program.

run by: justLock [options] [lockFilePath]

For options see longOptions and :: optHlpTxt.

7.41.3 Variable Documentation

7.41.3.1 lockFd

```
int lockFd
```

Lock file handle.

Do not use directly.

7.41.3.2 retCode

```
int retCode
```

Basic start-up function failure.

Allows for compact code without saving the (error) return:
if (openLock(lckPiGpioPth, ON)) return retCode;

Storage for return/error codes. Used by: openLock(char const *, uint8_t) theCyclistStart(int) theCyclistWaitEnd()

Value: 0: OK, else: error

7.41.3.3 lckPiGpioPth

```
char const* const lckPiGpioPth
```

Common path to a lock file for GpIO use.

Programs using GPIO in any form usually (and forced by some libraries) have to do this exclusively. This is implemented here by locking a file named /home/pi/bin/.lockPiGpio
Make the lock file by: touch /home/pi/bin/.lockPiGpio

Without locking this file those programs must not start.
So, deleting this file inhibits the start even by cron etc.

7.41.3.4 outLog

```
FILE* outLog
```

Event log output.

default: standard output; may be put to a file.

7.42 keysModClient.c File Reference

A Modbus TCP client program for Raspberry Pi3.

```
#include "arch/config.h"
#include "weUtil.h"
#include "weModbus.h"
#include <netinet/in.h>
#include <modbus-tcp.h>
#include <errno.h>
```

Data Structures

- union **KEYmodData**

  The holding registers used.

Functions

- int main (int argc, char **argv)

  The program.

Variables

- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.

7.42.1 Detailed Description

A Modbus TCP client program for Raspberry Pi3.

Copyright (c) 2017 Albrecht Weinert
weinert-automation.de a-weinert.de

Revision history

```
Rev. 236 2.02.2021
Rev.  43 22.09.2017 : beeper push pull omitted (piezo buzzer modulated)
Rev.  51 18.10.2017 : weModBus enhanced
Rev. 109 15.02.2018 : Doxygen compatibility changes (start)
```

GPIO usage

The real process IO via GPIO will be done by the Modbus server, usually the program keysModTCP. On the server (keysModTCP) this program uses a key matrix three LEDs and one piezzo buzzer.

Modbus usage

This is always determined by the Modbus server. keysModTCP and keysModClient fit each other as server client pair.

To communicate process and some organisational values the server program, keysModClient, restricts itself to holding registers. The usable function codes with holding registers are 03(03): read registers 16(10): write multiple registers 22(16): mask (AND/OR) single register 23(17): read/write registers

For 23(17), preferred in this case, the Modbus standard requires distinct register address ranges for read and write. This fits here as keysModClient uses registers 0..23 as read only for clients (status, time, inputs) and registers 24..31 as write (output); reading would yield the last client's write.

Timing

The program has a cyclic process control (in SPS manner) implemented by threads (pthread library) and signalling.

Library usage

The program uses the libraries (lib...): pthread and modbus The weModbus and sysUtil libraries are linked in. They may be converted to and used as library, also.

cross-compile by:
```
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
   -DMCU=BCM2837 -I./include  -c -o keysModClient.o keysModClient.c
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
 -DMCU=BCM2837 -DTARGET=raspi67 -I./include
     -c -o weRasp/weUtil.o weRasp/weUtil.c
arm-linux-gnueabihf-gcc -I. -DF_CPU=1200000000 -DPLATFORM=raspberry_03
 -DMCU=BCM2837 -DTARGET=raspi67 -I./include
   keysModClient.o weRasp/weUtil.o --output keysModClient.elf
      -Wl,-Map=keysModClient.map,--cref   -lpigpiod_if2 -lrt
```

or by:
```
make PROGRAM=keysModClient TARGET=raspi67 clean all
```

7.42.2 Function Documentation

7.42.2.1 main()

```
int main (
              int argc,
              char ** argv )
```

The program.

run by: keysModClient [ip [port]] default: 127.0.0.1 1502 best use: server.IP 502 Note 502 will need sudo at the server.

7.42.3 Variable Documentation

7.42.3.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.42.3.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.42.3.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.43 keysModTCP.c File Reference

A Modbus TCP server program for Raspberry Pi3 GPIO with gpiod.

```
#include "arch/config.h"
#include "weUtil.h"
#include "weGPIOkeyM.h"
#include "weModbus.h"
#include "weLockWatch.h"
#include <netinet/in.h>
#include <modbus-tcp-private.h>
#include <modbus-tcp.h>
#include <errno.h>
```

Data Structures

- union **KEYmodData**

  The holding registers used.

Functions

- int main (int argc, char **argv)

  The program.

Variables

- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.

7.43.1 Detailed Description

A Modbus TCP server program for Raspberry Pi3 GPIO with gpiod.

Copyright (c) 2017 Albrecht Weinert
weinert-automation.de a-weinert.de

Revision history

```
Rev. 237 8.02.2021
Rev.  37 17.08.2017 A.W.: first "real" Modbus server (output by client)
Rev.  42 21.09.2017 A.W.: beeper push pull omitted (piezo buzzer modulated)
Rev.  51 18.10.2017 A.W.: weModBus enhanced
```

A Modbus server for a key matrix, three LEDs and a beeper

GPIO usage

keysModTCP and keysModClient fit each other as server client pair.

keysModTCP uses the gpio(d) library for process IO. It uses three pins as output assuming three LEDs connected to as H=on

```
Pi3-IO   LED   Pi3-Pin             macro to configure
GPIO17 : red     11                LEDred
GPIO27 : green   13                LEDgrn
GPIO25 : yellow  22                LEDyel
GPIO18 : beeper  12                PIEPS
omitted GPIO23 : beeper  16   (pull)       PIEPSi
```

It uses 7 more pins both as inputs with pull down as well as as outputs (one at a time) to scan a 12 keys matrix, like EOZ Clavier S.series, 12 touches, Matricell, Téléphone: Pi IO wire Pi-Pin keys/touches EOZ-pin macro to configure

```
GPIO26 : lila    37   1-2-3-row       3     ROW123
GPIO19 : blue    35   4-5-6-row       4     ROW456
GPIO13 : brown   31   7-8-9-row      10     ROW789
GPIO6  : red     33   *-0-#-row       9     ROWa0h
GPIO21 : green   40   1-4-7-*-col     5     COL147a
GPIO20 : yellow  38   2-5-8-0-col     6     COL2580
GPIO16 : orange  36   3-6-9-#-col     7     COL369h
```

This program scans the EOZ Clavier S.series key matrix and outputs the keys pressed. When multiple keys are pressed wich do not lie in the same column, nor in the same row nor in one diagonal this is signalled as too many keys pressed.

Modbus usage

To communicate process and some organisational values, this program restricts itself to holding registers, which Modbus defines as 0 based array of 16 bit values in big endian.
The usable function codes with holding registers are

```
03(03): read registers
16(10): write multiple registers
22(16): mask (AND/OR) single register
23(17): read/write registers
```

For 23(17) the standard requires distinct ranges. This fits here as we use

```
registers  0..23   as read only for clients (status, time, inputs) and
registers 24..31   as write (output); reading would yield the last
                   client's write.
```

On disturbed or no Modbus communication the red LED (only) will be on.

Library usage

For process IO, this program uses the pigpiod socket interface, requiring a piGpio server/daemon running. This program itself needs no sudo to run. And it uses the sysUtil, the weGPIOd and the pthread library.

The program has a cyclic process control (in SPS manner) implemented by pthread signalling as supported by the sysUtil library. Comparing note: weAutSys (for AVR) uses protothreads for that purpose. It forces application singleton and may be used as service. In this version it uses the same lockfile as rdGnBlinkBlink and consorts.

Modbus TCP communication is based on libmodbus.

cross-compile by:

```
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
   -DMCU=BCM2837 -I./include  -c -o keysModTCP.o keysModTCP.c
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
 -DMCU=BCM2837 -DTARGET=raspi67 -I./include
     -c -o weRasp/weUtil.o weRasp/weUtil.c
arm-linux-gnueabihf-gcc -I. -DF_CPU=1200000000 -DPLATFORM=raspberry_03
 -DMCU=BCM2837 -DTARGET=raspi67 -I./include
   keysModTCP.o weRasp/weUtil.o --output keysModTCP.elf
      -Wl,-Map=keysModTCP.map,--cref   -lpigpiod_if2 -lrt
```

or by:

```
make PROGRAM=keysModTCP TARGET=raspi67 clean all
```

7.43.2 Function Documentation

7.43.2.1 main()

```
int main (
              int argc,
              char ** argv )
```

The program.

run by: keysModTCP [ip [port]] default: 0.0.0.0 1502 best use: client.IP 502 Note 502 will need sudo to work. This program would start without sudo, but connections to 502 would be rejected.

7.43.3 Variable Documentation

7.43.3.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.43.3.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.43.3.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.44 keysPiGpioTest.c File Reference

A seventh program for Raspberry's GPIO pins.

```
#include "arch/config.h"
#include "weUtil.h"
#include "weGPIOkeyM.h"
#include "weLockWatch.h"
```

Variables

- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.

7.44.1 Detailed Description

A seventh program for Raspberry's GPIO pins.

Rev. 237 8.02.2021 Copyright (c) 2017 Albrecht Weinert
weinert-automation.de a-weinert.de

It uses three pins as output assuming three LEDs connected to as H=on

```
Pi3 IO   LED    Pi3-Pin       macro to configure
GPIO17   red     11           LEDred
GPIO27   green   13           LEDgrn
GPIO25   yellow  22           LEDyel
```

It uses 7 more pins both as inputs with pull down and as outputs (one at a time) to scan a 12 keys matrix, like EOZ Clavier S.series, 12 touches, Matricell, Téléphone:

```
Pi   IO   wire  Pi-Pin  keys/touches EOZ-pin  macro to configure
GPIO26   lila     37   1-2-3-row       3     ROW123
GPIO19   blue     35   4-5-6-row       4     ROW456
GPIO13   brown    31   7-8-9-row      10     ROW789
GPIO6    red      33   *-0-#-row       9     ROWa0h
GPIO21   green    40   1-4-7-*-col     5     COL147a
GPIO20   yellow   38   2-5-8-0-col     6     COL2580
GPIO16   orange   36   3-6-9-#-col     7     COL369h
```

This program scans the EOZ Clavier S.series key matrix and outputs the keys pressed. When multiple keys are pressed wich do not lie in the same column, nor in the same row nor in one diagonal this is signalled as too many keys pressed.

It forces application singleton and may be used as service. In this version it uses the same lockfile as rdGnBlinkBlink and consorts.

Library use: This program uses the pigpiod socket interface, requiring a piGpio server/daemon running. This program itself needs no sudo to run. And it uses the sysUtil, the weGPIOd and the pthread library.

The program demonstrates the cyclic process control (in SPS manner) by pthread signalling as supported by the sysUtil library. Comparing note: weAutSys (for AVR) uses protothreads for that purpose.

cross-compile by:

```
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
     -DMCU=BCM2837 -I./include  -c -o keysPiGpioTest.o keysPiGpioTest.c
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
    -DMCU=BCM2837 -DTARGET=raspi67 -I./include
       -c -o weRasp/weUtil.o weRasp/weUtil.c
arm-linux-gnueabihf-gcc -I. -DF_CPU=1200000000 -DPLATFORM=raspberry_03
   -DMCU=BCM2837 -DTARGET=raspi67 -I./include
     keysPiGpioTest.o weRasp/weUtil.o --output keysPiGpioTest.elf
       -Wl,-Map=keysPiGpioTest.map,--cref   -lpigpiod_if2 -lrt
```

7.44.2 Variable Documentation

7.44.2.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.44.2.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.44.2.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.45 main_page.dox File Reference

This file is for extra Doxygen documentation texts, only.

7.45.1 Detailed Description

This file is for extra Doxygen documentation texts, only.

It contains no software. Copyright (c) 2018 2020 2023 Albrecht Weinert, Bochum

Revision history
```
Rev. 253 9.10.2023
Rev. 108 13.02.2018 : new, modified from weAutSys (2014)
Rev. 109 15.02.2018 : directory descriptions included (trial)
Rev. 209 21.07.2019 : more links
Rev. 233 20.10.2020 : minor corrections and updates
Rev. 253 09.10.2023 : updated
```

7.46 meteRead.c File Reference

A CGI program to co-operate with hometerControl.

```
#include "sysBasic.h"
#include "sweetHome.h"
#include "weCGIajax.h"
#include "weShareMem.h"
```

Functions

- int main (int argc, char **argv)

  The program.

Variables

- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.

7.46.1 Detailed Description

A CGI program to co-operate with hometerControl.
```
Copyright  (c)  2018 - 2023  Albrecht Weinert
weinert-automation.de        a-weinert.de
```


Revision history
```
Rev. 252 2.10.2023
Rev. 76+ 06.12.2017 : new; derived from hometersConsol.c
Rev.  85 21.12.2017 : more and flexible commands
Rev. 115 22.02.2018 : larger data structures
Rev. 125 23.03.2018 : power factor (all meters and phases) added
Rev. 132 28.04.2018 : batModU added
```

```
Rev. 149 17.06.2010 : pps value added
Rev. 165 12.07.2018 : temperature & batt values, pps power
Rev. 166 14.07.2018 : more pps commands, more  help texts (title) in JSON
Rev. 170 24.07.2018 : real temperature names; Umlauts got working
Rev. 187 14.10.2018 : JSON sequences:  P in front, T resorted
Rev. 209 15.07.2019 : minor improvements (docu)
Rev. 220 09.11.2019 : Growatt values added to valFilVal_t
Rev. 251 11.07.2023 : update for 2nd heater element and prep. car load
```

Client functions

This program gets values from hometersControl via shared memory synchronised with a set of semaphores:
sem # 0: exclusive lock of shared memory (for the shortest time possible !)
sem # 1: signal from from hometersControl to other program
sem # 2: signal from from hometersControl to this program (meteRead)

Under the same semaphore (#0) lock it may set command codes in the shared memory according to the query string. A query will usually be part of the request when buttons were pressed in the web page.

GCI server functions

This program's output (i.e. AJAX answer) are physical readings, status values and times as JSON object, leaving selection of values and their display to the HTML page respectively its Javascript code.

In the JSON object's text delivered all numerical values (int, float) are put as strings in appropriate form and precision. The rationale is to avoid auto parsing of data, all or most of which will probably be formatted (back to text) to be put in the web page. And, alas, formatting is not Javascript's strong point, while parsing a string to a number would not be a problem, if a number is needed.

GCI back-link – command by query

This program evaluates a query string parameter command=command. command (2nd) = startPump | stopPump etc. see sweetHome.h and cmdLookUp.

Library usage

The program uses the library (lib...): pthread

```
Build the program
```

cross-compile by:
```
make PROGRAM=meteRead TARGET=meterPi clean all
```

program by:
```
make PROGRAM=meteRead TARGET=meterPi FTPuser=pi:piSecret progapp
```

due to bugs in c make use winscp directly by entering the command displayed

load the matching html page by:
```
make PROGRAM=meteReadHTML TARGET=meterPi FTPuser=pi:piSecret progapp
```

due to bugs in c make use winscp directly by entering the command displayed

7.46.2 Function Documentation

7.46.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

The program.

Run by (Apache) web server as (cgi) script with optional (query string) parameters.

7.46.3 Variable Documentation

7.46.3.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.46.3.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.46.3.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.47 metersModbusClient.c File Reference

A Modbus RTU client program for Raspberry Pi3 using smart meters.

```
#include "arch/config.h"
#include "weUtil.h"
#include "weModbus.h"
#include "pigpiod_if2.h"
#include "weGPIOd.h"
#include "weLockWatch.h"
#include <netinet/in.h>
#include <modbus-tcp.h>
#include <errno.h>
```

Functions

- int main (int argc, char **argv)

  The program.

Variables

- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.

7.47.1 Detailed Description

A Modbus RTU client program for Raspberry Pi3 using smart meters.

weinert-automation.de a-weinert.de

Revision history

```
Rev. 252  2.10.2023
Rev. <46  27.09.2017 : not yet operable / ported from key matrix to meter
Rev.  46  28.09.2017 : displays V A W and Hz for SMD230Modbus (first com.)
Rev.  47  30.09.2017 : works with SDM230 and SMD630 (L1)
Rev. 216  31.08.2019 : %n error corrected
```

GPIO usage

In the first versions (43+) metersModbusClient uses three pins as output.

For tests we assume three LEDs connected to as H=on.

```
GPio Pi1/Pi3    test LED     pin header   Px on converter
GPIO17  / 17         red     11           P0
GPIO21  / 27       green     13           P2
GPIO25  = 25      yellow     22           P6
          16&15            10&8  RxD TxD  UART to RS485 converter module
```

In production use there'l be up to eight relays in a module controlled by an active low Din signal to the module.

```
module  : Gnd  Di1  Di2  Di3  Di4  Di5  Di6  Di7  Di8  3V3     Gnd  5V
```

```
colour  : brn  blk  wht  gry  vio  blu  grn  yel  ora  red     brn  red
Px conv.: Gnd   P0   P1   P2   P3   P4   P5   P6   P7  3.3V    Gnd  5V
Pi Pin  : 6&c   11   12   13   15   16   18   22    7  1 &c    6&c  2&3
GPIO Pi3:       17   18   27   22   23   24   25    4
```

Modbus usage

This is always determined by the Modbus server.
This server is for B+G E-Tech EASTRON smart meters as servers.
The meters will be SDM230-Modbus (one phase) and SDM630-Modbus (three phase).
Communication will be RTU / RS484 9600, none as common denominator. See
Eastron, SMD230Modbus, Smart Meter Modbus Protocol Implementation V1.2
Eastron, SDM630Modbus, Smart Meter Modbus Protocol 630 (V1.5 ?)
Eastron, SDM530Modbus, Smart Meter Modbus Protocol 530 (V1.1 or V1.5)

Those smart meters use holding registers for configuration (may be disabled) and input registers for all measurement.
The usable function codes with Eastron meters are

```
03(03): read holding registers
04(04): read input registers      (only function in normal operation)
16(10): write holding registers
```

SDM630 / SMD230 Input registers

There are 4 bytes = 2 16 bit registers per "parameter" An input register address in the range 0000 to 017C respectively 0182 would always include the next address for a 32 bit IEEE754 float. The meters would send most significant byte first (big endian), but that order should be changed by holding register "register order". At present this meter function is unspecified.

Input register addresses are in the range 0000 to 0182 / 017C (230/630) always including the next address for a 32bit IEEE754 float.

Regarding 230/630 or V1.2/V1.5 register addresses / "parameter" numbers are less common than they could have been made. Another flaw are huge gaps of unused addresses, potentially hurting in the light of the restriction to 40 "parameters" = 80 registers per telegram. A common denominator for tests are registers 00 06 0C 12 24 46 for L1 measurements and 46 to 4E for cumulative work values, being the first 80 registers.

Timing

The program has a cyclic process control (in SPS manner) implemented by threads (pthread library) and signalling.

Library usage

The program uses the libraries (lib...): pthread, pigpiod_if2 and modbus The (weRasp/..c, include/..h) weModbus, weGPIOd and sysUtil libraries are linked in. They may be converted to and used as (.so) library, also.

cross-compile by:

```
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
   -DMCU=BCM2837  -DTARGET=raspi61  -I./include
         -c -o metersModbusClient.o metersModbusClient.c
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
    -DMCU=BCM2837 -DTARGET=raspi61  -I./include
         -c -o weRasp/weUtil.o weRasp/weUtil.c
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
    -DMCU=BCM2837 -DTARGET=raspi61  -I./include
        -c -o weRasp/weModbus.o weRasp/weModbus.c
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
    -DMCU=BCM2837 -DTARGET=raspi61  -I./include
        -c -o weRasp/weGPIOd.o weRasp/weGPIOd.c
arm-linux-gnueabihf-gcc -I. -DPLATFORM=raspberry_03 -DMCU=BCM2837
   -DTARGET=raspi61  -I./include   metersModbusClient.o
      weRasp/weUtil.o weRasp/weModbus.o weRasp/weGPIOd.o
        --output metersModbusClient.elf
          -Wl,-Map=metersModbusClient.map,--cref
              -lrt -pthread -lmodbus -lpigpiod_if2
```

or by:

```
make PROGRAM=metersModbusClient TARGET=raspi61 clean all
```

program by:

```
make PROGRAM=metersModbusClient TARGET=meterPi FTPuser=pi:piSecret progapp
```

7.47.2 Function Documentation

7.47.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

The program.

run by: metersModbusClient [slave1 [slave2 allowed: 1..247 (wrong value -> error) default with no parameter: 1

7.47.3 Variable Documentation

7.47.3.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.47.3.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.47.3.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.48 minTerm.c File Reference

A console program for a RS232 UART on the Pi.

```
#include "weSerial.h"
#include <errno.h>
#include <getopt.h>
#include <pthread.h>
#include "weModbus.h"
#include <dirent.h>
```

Variables

- char const prgNamPure []
  The pure program name.
- char const prgSVNdat []
  The complete SVN date string.
- char const prgSVNrev []
  The complete SVN revision string.

7.48.1 Detailed Description

A console program for a RS232 UART on the Pi.

```
Copyright  (c)  2019   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 236 2.02.2021
Rev. 204 28.06.2019 : new; first step just for testing UART
```

Program functions

This program offers very basic communication and Modbus usage with Raspberry's serial interface for testing and developing.

Library usage

This program uses libmodbus.

Build

```
make PROGRAM=minTerm TARGET=growPi clean all
make PROGRAM=minTerm TARGET=growPi progapp
make PROGRAM=minTerm clean
```

7.48.2 Variable Documentation

7.48.2.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.48.2.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.48.2.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.49 mosquiSub.c File Reference

A very simple MQTT subsciber.

```
#include <signal.h>
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <mosquitto.h>
#include <getopt.h>
```

Variables

- char clientId [30]

  Id of this MQTT subscriber.
- char mqttHost [68]

  The MQTT broker URL or name.
- int **mqttPort**

  MQTT port 1883.
- char **topicS** [60]

  Root of topics to log.

7.49.1 Detailed Description

A very simple MQTT subsciber.

```
Copyright  (c)  2017   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /      \|   |
     \/  \/   \__/        \__/|_
```

cross-compile by:

```
make PROGRAM=mosquiSub TARGET=meterPi clean all
```

program respectively transfer to target machine by:

```
make PROGRAM=mosquiSub TARGET=raspi61 FTPuser=pi:piSecret progapp
```

The building of the application is governed by the make include makeProg_mosquiSub_settings.mk:

```
# A makefile include for raspberry projects
# program include for one program
# makeProg_mosquiSub_settings.mk
# Copyright  2017  Albrecht Weinert    < a-weinert.de >
MAKE_INCLUDE_PROGRAM  = mosquiSub
MAKE_PROGRAM_LAST_CHANGE = '$Date: 2021-02-02 18:11:02 +0100 (Di, 02 Feb 2021) $ '
MAKE_PROGRAM_REVISION = '$Revision: 236 $ '
ifndef COPYRIGHT_YEAR
$(error includefile $(MAKE_INCLUDE_PROGRAM) used directly.)
endif
# PROGRAM or MAIN_F might have been given in wrong case (at least on Windows)
override MAIN_F = $(MAKE_INCLUDE_PROGRAM)
# An optional short multiline description of this program's specifica.
# May be empty. But, do not change the three lines define endef and export.
define PROG_DES_TEXTT
Program mosquiSub
This program is a first example of a MQTT subscriber as well as a minimal
test program to see whats going on with subjects labExp/sweetHome/#.
endef
export PROG_DES_TEXT
extraLDFLAGS = -lmosquitto
# extraSOURCES = weRasp/weUtil.c weRasp/weShareMem.c weRasp/weCGIajax.c
# FTPdir = var/www/cgi
```

7.49.2 Variable Documentation

7.49.2.1 mqttHost

```
char mqttHost[68]
```

The MQTT broker URL or name.

May be set by option –mqttHost meterPi or –mqttBroker 192.168.178.87

default: localhost MQTTBroker (currently localhost !)

7.49.2.2 clientId

```
char clientId[30]
```

Id of this MQTT subscriber.

MQTT client ID.

7.50 rdGnPiGpioBlink.c File Reference

A fourth program for Raspberry's GPIO pins.

```
#include "arch/config.h"
#include "weUtil.h"
#include "pigpio.h"
#include "weLockWatch.h"
```

Variables

- char const prgNamPure [ ]
  The pure program name.
- char const prgSVNdat [ ]
  The complete SVN date string.
- char const prgSVNrev [ ]
  The complete SVN revision string.

7.50.1 Detailed Description

A fourth program for Raspberry's GPIO pins.

weinert-automation.de a-weinert.de

Revision history

```
  Rev. 234 20.01.2021
  Rev.  58 30.10.2017 : new
It uses two pins as output assuming two LEDs connected to as H=on
Pi 1  / Pi 3       Pin
GPIO17/ 17 : red    11
GPIO21/ 27): green  13
This program forces application singleton and may be used as service
Cross-compile  by: \code
arm-linux-gnueabihf-gcc -DPLATFORM=raspberry_03 -DMCU=BCM2837  -DTARGET=raspi48 -I./include   -c -o
       rdGnPiGpioBlink.o rdGnPiGpioBlink.c
arm-linux-gnueabihf-gcc -I.-DPLATFORM=raspberry_03 -DMCU=BCM2837  -DTARGET=raspi48 -I./include
       rdGnPiGpioBlink.o  --output rdGnPiGpioBlink.elf  -Wl,-Map=rdGnPiGpioBlink.map,--cref lpigpio -lrt
```

Its functions are the same as rdGnBlinkBlink (even sharing the lockfile) except for using the pigpio library. Our makefiles define MCU and PLATFORM as make variables and makros. So we could make the GPIO pin and address assignment automatically.

Library use: This program rdGnPiGpioBlink makes direct calls to the pigpio server/daemon interface and is linked with this library.
This means: a) it requires sudo itself
and b) it requires the standard daemon be stopped by:
sudo killall pigpiod
and c) would act itself as daemon (confirmed by tests).
This approach is normally not recommended. Use the pipe or better the socket interface (as in rdGnPiGioDBlink) with a running pigpod.

7.50.2 Variable Documentation

7.50.2.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.50.2.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.50.2.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.51 rdGnPiGpioDBlink.c File Reference

A demo program for Raspberry's GPIO pins using a pigpioD server.

```
#include "trafficPi.h"
#include "weGPIOd.h"
#include "weUtil.h"
#include "weLockWatch.h"
#include <getopt.h>
```

Variables

- int **useTraffic**

  flag to use piTraffic shield; default off

7.51.1 Detailed Description

A demo program for Raspberry's GPIO pins using a pigpioD server.
'

```
   Copyright  (c)  2017 2020 2021   Albrecht Weinert
   weinert-automation.de            a-weinert.de

     /         /      /\
    /         /___    /  \       |
    \        /____\  /____\  |   _|__
     \  /\   / \     /       \|    |
      \/  \/   \__/           \__/|_
```

```
   Revision history
   Rev. 237 8.02.2021

   Rev.  45 11.08.2019 : one yellow LED added
   Rev. 231 27.08.2020 : use weGPIOd.h, enable piTraffic shield for LEDs
   Rev. 237 08.02.2021 : use bank mask functions in  weGPIOd.h

      .
```

Purpose

This program demonstrates
a) the usage if a pigpioD server for IO
b) the implementation of an exact (100 ms / 600ms) cycle
c) the singleton use by lock file.

It uses three pins as output assuming LEDs connected to as Hi=On

```
   Pi 1 / 3,4,0 LED  Pin
GPIO 17 / 17    red   11
GPIO 21 / 27    green 13
GPIO 25 / 25   yellow 22
```

Alternatively by option –traffic, it uses all 12 LEDs of the piTraffic shield; see trafficPi.h.

Library use: This program uses the pigpiod socket interface, requiring a piGpio server/daemon running. This program needs no sudo to run.

cross-compile by:

```
make PROGRAM=rdGnPiGpioDBlink TARGET=pi4you clean all
```

program by:

```
make PROGRAM=rdGnPiGpioDBlink TARGET=pi4you FTPuser=pi:piPi progapp
```

run by:

```
rdGnPiGpioDBlink  [--traffic ]
```

7.52 serPiGpioDbase.c File Reference

A Modbus RTU client program for Raspberry Pi3 using smart meters.

```
#include <pthread.h>
#include "arch/config.h"
#include "weUtil.h"
#include "weModbus.h"
#include "pigpiod_if2.h"
#include "weGPIOd.h"
#include "weLockWatch.h"
#include <modbus-tcp.h>
#include <errno.h>
```

Functions

- int main (int argc, char **argv)

  The program.

Variables

- char const prgNamPure [ ]

  The pure program name.
- char const prgSVNdat [ ]

  The complete SVN date string.
- char const prgSVNrev [ ]

  The complete SVN revision string.

7.52.1 Detailed Description

A Modbus RTU client program for Raspberry Pi3 using smart meters.

Copyright (c) 2017 Albrecht Weinert
weinert-automation.de a-weinert.de

This is the basic / test server program to be used with just one smart meter client at a time. When running correctly it displays basic measurements of one phase on the terminal console.

Revision history

```
Rev. 237 8.02.2021
Rev. <46 27.09.2017 : not yet operable / ported from key matrix to meter
Rev.  46 28.09.2017 : displays V A W and Hz for SMD230Modbus (first com.)
Rev.  47 01.10.2017 : works with SDM230 as well as SDM530 and SMD630 (L1)
Rev.  51 19.10.2017 : weModbus enhanced
```

GPIO usage

In the first versions (43+) serPiGpioDbase uses three pins as output.
For tests we assume three LEDs connected to as H=on.v

```
GPio Pi1/Pi3    test LED     pin header   Px on converter
GPIO17  / 17        red      11           P0
GPIO21  / 27      green      13           P2
GPIO25  = 25     yellow      22           P6
          16&15            10&8  RxD TxD  UART to RS485 converter module
```

These (and more) GPIOs may be used for (low active module) relays as well. In the first test versions (43+) we assume LEDs and blink them just as life sign.

Modbus usage

This is always determined by the Modbus server. This server is for B+G E-Tech EASTRON smart meters as servers. The meters will be SDM230-Modbus (one phase) and SDM630-Moddus (three phase). Communication will be RTU / RS484 9600, none as common denominator. See Eastron, SMD230Modbus, Smart Meter Modbus Protocol Implementation V1.2 Eastron, SDM630Modbus, Smart Meter Modbus Protocol Implementation V1.5

Those smart meters use holding registers for configuration (may be disabled) and input registers for all measurement.
The usable function codes with Eastron meters are

```
03(03): read holding registers
04(04): read input registers      (only function in normal operation)
16(10): write holding registers
```

SDM630 / SMD230 Input registers

There are 4 bytes = 2 16 bit registers per "parameter".
An input register address in the range 0000 to 017C respectively 0182 would always include the next address for a 32 bit IEEE754 float.
The meters would send most significant byte first (big endian), but that order should be changed by holding register "register order". At present this meter function is unspecified.

Input register addresses are in the range 0000 to 0182 / 017C (230/630) always including the next address for a 32bit IEEE754 float.

Regarding 230/630 or V1.2/V1.5 register addresses / "parameter" numbers are less common than they could have been made. Another flaw are huge gaps of unused addresses, potentially hurting in the light of the restriction to 40 "parameters" = 80 registers per telegram.
A common denominator for tests are registers 00 06 0C 12 24 46 for L1 measurements and 46 to 4E for cumulative work values, being the first 80 registers.

Timing

The program has a cyclic process control (in SPS manner) implemented by threads (pthread library) and signalling.

Library usage

The program uses the libraries (lib...): pthread, pigpiod_if2 and modbus The weModbus, weGPIOd and sysUtil libraries are linked in. They may be converted to and used as library, also.

cross-compile by:
```
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
   -DMCU=BCM2837  -DTARGET=raspi61  -I./include
         -c -o serPiGpioDbase.o serPiGpioDbase.c
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
    -DMCU=BCM2837 -DTARGET=raspi61  -I./include
         -c -o weRasp/weUtil.o weRasp/weUtil.c
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
    -DMCU=BCM2837 -DTARGET=raspi61  -I./include
        -c -o weRasp/weModbus.o weRasp/weModbus.c
arm-linux-gnueabihf-gcc -DF_CPU=1200000000 -DPLATFORM=raspberry_03
    -DMCU=BCM2837 -DTARGET=raspi61  -I./include
        -c -o weRasp/weGPIOd.o weRasp/weGPIOd.c
arm-linux-gnueabihf-gcc -I. -DPLATFORM=raspberry_03 -DMCU=BCM2837
   -DTARGET=raspi61  -I./include   serPiGpioDbase.o
      weRasp/weUtil.o weRasp/weModbus.o weRasp/weGPIOd.o
        --output serPiGpioDbase.elf
          -Wl,-Map=serPiGpioDbase.map,--cref
              -lrt -pthread -lmodbus -lpigpiod_if2
```

or by:
```
make PROGRAM=serPiGpioDbase TARGET=raspi61 clean all
make PROGRAM=serPiGpioDbase TARGET=raspi61 progapp
```

7.52.2 Function Documentation

7.52.2.1 main()

```
int main (
              int argc,
              char ** argv )
```

The program.

run by: serPiGpioDbase [slave/meter] allowed: 1..247 (wrong value -> error) default with no parameter: 1

7.52.3 Variable Documentation

7.52.3.1 prgNamPure

```
char const prgNamPure[]
```

The pure program name.

To be provided in the application's / program's source.

See also

progNam() progNamB()

7.52.3.2 prgSVNrev

```
char const prgSVNrev[]
```

The complete SVN revision string.

To be provided in the application's / program's source.

See also

progRev()

7.52.3.3 prgSVNdat

```
char const prgSVNdat[]
```

The complete SVN date string.

To be provided in the application's / program's source.

See also

progDat()

7.53 testOnPi.c File Reference

This program is a vehicle for diverse tests and auxiliary functions on a Pi.

```
#include "trafficPi.h"
#include "weGPIOd.h"
#include "weUtil.h"
#include "weStateM.h"
#include <getopt.h>
#include "weLockWatch.h"
#include <errno.h>
#include "weUSBscan.h"
```

Functions

- void * **blink0001Thread** (void *args)
  The task of blinking in 1ms thread.
- void * **blink0010Thread** (void *args)
  The task of blinking in 10ms thread.
- void * **blink0020Thread** (void *args)
  The task of blinking in 20ms thread.
- void * **blink0100Thread** (void *args)
  The task of blinking in 100ms thread.
- void * **blink1000Thread** (void *args)
  The task of blinking in 1s thread.
- int main (int argc, char **argv)
  The program.

Variables

- unsigned const * **theGNs**
  the Greens; default theGNsTrad
- unsigned const **theGNsTrad** [ ]
  traditional rdGnBlink green
- unsigned const **theGNsTraf** [ ]
  piTraffic shield all green
- unsigned const * **theRDs**
  the Reds; default theRDsTrad
- unsigned const **theRDsTrad** [ ]
  traditional rdGnBlink red
- unsigned const **theRDsTraf** [ ]
  piTraffic shield all red
- unsigned const * **theYEs**
  the Yellows; default theYEsTrad
- unsigned const **theYEsTrad** [ ]
  traditional rdGnBlink yellow
- unsigned const **theYEsTraf** [ ]
  piTraffic shield all ye

7.53.1 Detailed Description

This program is a vehicle for diverse tests and auxiliary functions on a Pi.

```
Copyright (c)   2020, 2021   Albrecht Weinert
weinert-automation.de        a-weinert.de
```

```
Revision history
Rev. 241 10.08.2021
Rev. 230 11.08.2020 : new  (homeDoorPhone fork)
Rev. 232 26.06.2020 : handle Doxygen duplication bug
Rev. 234 31.12.2020 : one DCF77 timing set, DCF77 decoded
Rev. 236 01.02.2021 : DCF77 handling put to dcf77onPi.c (fork)
   .
```

Purpose

This program is a varying collection of diverse tests and auxiliary functions on a Pi.
Stable functions to be kept as useful will get sensible options respectively names like

```
--GPIO    -G   GPIO number for IO operation
--pin     -p   IO connector pin number for IO operation
--IN   --OUT   set selected GPIO to mode INput respectively OUTput
--ON   --OFF   set selected GPIO ON respectively OFF
--duty    -d   set dutycycle 0..255 for selected GPIO
--servo   -s   set servo pulse width 0:off; 500 ..1500:middle.. 2500:clockwise
--wait    -w   wait for about 1 second
```

Use –help to see the actual list.

GPIO usage

The program may use (test) every GPIO by command line parameter.

Timing

The program has a cyclic process control (in SPS manner, by weUtil.h weUtil.c etc). We have

```
a) a 1 ms cycle as extra thread
b) a 100 ms cycle as end life of main()
```

Library usage

The program uses some standard libraries plus own libraries in weRasp/..c and include/..h, namely weGPIOd, sysUtil etc..

Prerequisites

As the program may use GPIO via pigpiod and the watchdog it requires them both available, e.g. by a (sudo crontab) cronjob

```
@reboot  /usr/local/bin/pigpiod  -s 10
@reboot  sleep 5 && chmod a+=rw /dev/watchdog
@reboot  chmod a+=rw /dev/hidraw0
```

```
Compile, build, load
```

cross-compile by:

```
make PROGRAM=testOnPi TARGET=raspi67 clean all
```

program by:

```
make PROGRAM=testOnPi TARGET=pi4you FTPuser=pi:piPi progapp
```

7.53.2 Function Documentation

7.53.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

The program.

Run by: testOnPi [options

For options see: testOnPi –help

7.54 weRasp/sweetHome.c File Reference

Common values for an experimental smart home (lab) project.

```
#include "sweetHome.h"
#include <stdio.h>
```

Functions

- float pwmToVolt (uint8_t const pwm)
  Output voltage of battery load module by PWM signal.
- uint8_t voltToPWM (float const u)
  PWM signal for desired output voltage of battery load module.

Variables

- cmdLookUp_t cmdLookUp [ ]
  The common command look up table.
- volatile float fLine
  last valid power line frequency
- uint8_t const loadModUlookup [256]
  Output voltage of electronic battery load module.
- float const **phPckSwPow** [101]
  First (or only) phase packet switching device power look up.
- phPckSwSet_t const phPckSwSets [101]
  The packet switch control values.
- smdX30modbus_t smdX30modbus [ANZmodSLAVES]
  Descriptive and state array for smart meters on Modbus.
- volatile int tempTankWater
  Last value of tank water temperature.
- volatile uint8_t tempWaterBadCnt
  Tank water temperature bad read count.
- valFilVal_t **valFilVal**
  All process values relevant for log files and HMI.

7.54.1 Detailed Description

Common values for an experimental smart home (lab) project.

```
Copyright  (c)  2018 2023  Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 252 2.10.2023
Rev.  99 29.01.2018 : load module voltage handling added
Rev. 167 19.07.2018 : phPckSwPow [100]  950.00 W
Rev. 182 12.08.2018 : minor, commands renamed
Rev. 217 05.09.2019 : phase packet switch ballast three 100% values
Rev. 229 23.07.2020 : valFil (CSV) removed
Rev. 245 27.02.2023 : pps 100% power raised to 1983 W due panel reorg.
```

7.54.2 Function Documentation

7.54.2.1 pwmToVolt()

```
float pwmToVolt (
            uint8_t const pwm )
```

Output voltage of battery load module by PWM signal.

Parameters

pwm	PWM signal 0 (lowest) .. 255 (highest output voltage)

Returns

output voltage 10.8 .. 15.2V

7.54.2.2 voltToPWM()

```
uint8_t voltToPWM (
            float const u )
```

PWM signal for desired output voltage of battery load module.

Parameters

u	desired output voltage 10.8 .. 15.2V out of range returns 0 or 255

Returns

PWM signal width 0..255 (i.e 0..100%)

7.54.3 Variable Documentation

7.54.3.1 cmdLookUp

```
cmdLookUp_t cmdLookUp[]
```

The common command look up table.

It must end with an entry {"", 0}.

The current (CGI) program uses linear search for the command mnemonic. Hence, and cause of structure, alphabetic sorting is of no avail.

7.54.3.2 loadModUlookup

```
uint8_t const loadModUlookup[256]
```

Output voltage of electronic battery load module.

Depending on the (400Hz) PMW signal width 0..255 (/255 $*$ 100%) the battery load module will deliver about 10.8 to 15.2V. If the battery voltage is higher than the output voltage no current will flow (i.e. nothing happens). If the battery voltage is lower, up to 10A will be delivered. This current limit is implemented / fixed by the power module used. Hence, getting less current respectively power requires a fine control of the PWM signal (see POWH).

This array gives the module output voltage in 100mV units (value) as function of the PWM width (index). For the values see the comment in the file sweetHome.c.

7.54.3.3 fLine

```
volatile float fLine
```

last valid power line frequency

Last valid power line frequency.

7.54.3.4 phPckSwSets

```
phPckSwSet_t const phPckSwSets[101]
```

The packet switch control values.

The array holds the number of on and off phases (i.e. 20 ms periods at 50 Hz line frequency) for each percentage of full power. The array length is 101; the index [0..100] is, hence, directly the percentage wanted.

To avoid too visible flicker, for no set (phPckSwSets[i].onPhases, phPckSwSets[i].offPhases) the smaller of the two values would be greater than 4 (80 ms); in most cases it is 1 or 2 (40 ms).

7.54.3.5 smdX30modbus

```
smdX30modbus_t smdX30modbus[ANZmodSLAVES]
```

Descriptive and state array for smart meters on Modbus.

The number of meters is ANZmodSLAVES.

7.54.3.6 tempTankWater

```
volatile int tempTankWater
```

Last value of tank water temperature.

This value will be the last good .tempTankTop (see valFilVal_t) or the last good .tempPipe. If neither is good for 251 measurements BAD_TEMP_READ (an incredibly high value) will be set.
This last tank water temperature will be checked against a safety limit to allow electric heating as ballast.
The value is in units of 1/1000 grdC.

7.54.3.7 tempWaterBadCnt

```
volatile uint8_t tempWaterBadCnt
```

Tank water temperature bad read count.

This last tank water temperature will be checked against a safety limit to allow electric heating as ballast.

In case of too many bad reads the value must be fixed at 253

7.55 weRasp/sweetHome2.c File Reference

Common values for an experimental smart home (lab) project.

```
#include "sweetHome2.h"
#include "sweetHomeLocal.h"
```

Functions

- void batKeepInhTimChg (state_t *const me)

  Inhibit battery keep alive loading timer state change function.
- void befRiseTimChg (state_t *const me)

  Before sunrise timer state change callback.
- void dawnTimChg (state_t *const me)

  Dawn timer state change callback.
- void duskTimChg (state_t *const me)

  Dusk timer state change callback.
- void genSunStateText (char *stateText, state_t const *const me, char const *stamp)

  Generate the sun's status text.
- void logBatteryStress ()

  Log battery stress on outLog as line with time stamp.
- void **mqttClean** ()

  End as MQTT client.
- int mqttInit ()

  Initialise as MQTT client.
- void mqttPlg01Set (uint8_t const on)

  Switch the plug Plug01.
- void mqttPlg02Set (uint8_t const on)

  Switch the plug PLG2.
- void mqttPlg03Set (uint8_t const on)

  Switch the plug PLG3.
- void mqttPlg04Set (uint8_t const on)

  Switch the plug PLG4.
- uint8_t pckSwPec (float power)

  Set and get package switch percentage by power.
- void reduceBatLoadPWM (uint8_t const pwmRed)

  Reduce battery loader module PWM.
- void setBatLoadPWM (uint8_t const pwm)

  Set battery loader module PWM.
- void setConsGiv (state_t *const me)

  Power consumer / producer (give away) SFC transition.
- uint8_t setPckSwPerc (uint8_t perc)

  Set package switch percentage.
- float setPhPckLimit (float powerLimit)

  Set and get package switch power limit.
- void solPowStateText (char *stateText, state_t const *const me, char const *stamp)

  Generate solar power status text.
- void sunriseTimChg (state_t *const me)

  Sunrise timer state change callback.
- void sunsetTimChg (state_t *const me)

  Sunset timer state change callback.
- void surplStateText (char *stateText, state_t const *const me, char const *stamp)

  Generate surplus power status text.
- void switchBatToStepUp (uint8_t const on)

  Switch battery to step up converter.
- void switchConsGiv (state_t *const me)

  Power consumer / producer (give away) transition.
- void switchHotWpump (uint8_t const on)

Hot water comfort pump turn on/off.

- void switchInvToBatUnl (uint8_t const on)

  Switch inverter to battery unload.
- void switchSolPow (state_t *const me)

  Solar power producer transition.
- void switchWouldGiv250W (state_t *const me)

  Would give 250W or more away, the transition.
- void switchWouldGivAway (state_t *const me)

  Would give away transition.

Variables

- state_t batKeepInh

  Inhibit battery keep alive loading.
- int batUnloadAllowed

  Battery unload allowed.
- uint8_t batVoltValid

  The measured battery voltage is valid.
- state_t befRiseTimer

  Before sunrise timer.
- char clientId [38]

  MQTT client ID.
- state_t consumeGiveHyst

  Give away hysteresis.
- state_t consumeGiveSFC

  Give away state machine / SFC.
- state_t dawnTimer

  Dawn timer.
- state_t duskTimer

  Dusk timer.
- uint8_t **heaterAllowed**

  phase packet load enabled (always)
- uint16_t hundredsV

  The battery voltage in 0.01V units.
- uint32_t hundVsecAbove

  Hundreds of Vs above 13.6 V battery voltage.
- uint32_t hundVsecBelow

  Hundreds of Vs below 12.25 V battery voltage.
- char mqttHost [68]

  The MQTT broker URL or name.
- int **mqttPort**

  MQTT port 1883.
- volatile uint8_t **phPckCnt**

  current switch state duration counter
- volatile uint8_t **phPckOff**

  current phase packet switch OFF duration
- volatile uint8_t **phPckOn**

  current phase packet switch ON duration
- volatile uint8_t **phPckRelOffDelay**

  PPS relay off delay.

- volatile uint8_t **phPckS2**

  heater 2 pps switch state: 0 off; 1 on
- volatile uint8_t **phPckSw**

  phase packet switch state: 0 off; 1 on
- volatile uint8_t phPckSwIndex

  Actual index (0..200%) determining power.
- volatile uint8_t **phPpow2**

  heater 2 power ∗ 50% (0:0% .. 2:100%)
- oneWireDevice_t **sensors** [NUM1W_SENSORS]

  The 1-wire sensors used.
- state_t **solarPowerHyst**

  Solar power producer hysteresis.
- char subTopStPlg01 [14]

  State sub topic of S20 plug Number 01 to 09.
- state_t sunriseTimer

  Sunrise timer.
- state_t sunsetTimer

  Sunset timer.
- state_t wouldGive250WHyst

  Would give away 250W hysteresis.
- state_t wouldGiveAwayHyst

  Would give away hysteresis.

7.55.1 Detailed Description

Common values for an experimental smart home (lab) project.

```
Copyright  (c)  2018   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 253 9.10.2023
Rev.  85 09.01.2018 : new
Rev.  99 29.01.2018 : load module voltage handling added
Rev. 145 13.06.2018 : IO improved; PLG2 used as give away alert
Rev. 148 16.06.2018 : time handling improved, Hippogreiff relay
Rev. 185 24.08.2018 : sunset discharge again
Rev. 187 14.10.2018 : Hippogreif off 2h before sunrise
Rev. 189 06.12.2018 : Hippo and (in between optional) Plug 3/4 back to rise
Rev. 197 03.03.2019 : Plug 2 hippo; Plug04 surplus power
Rev. 199 02.04.2019 : allow battery unload before sunrise as default
Rev. 200 16.04.2019 : battery keep inhibit until low stress
Rev. 203 30.04.2019 : timer macro change, sun times & surplus log enhanced
Rev. 205 19.05.2019 : log battery stress (%) repaired
Rev. 206 27.05.2019 : PPS pre relay off delay 5 minutes (en lieu de 3s)
Rev. 209 22.07.2019 : work around a Doxygen bug
Rev. 222 18.03.2020 : noUnloadBat as default
Rev. 232 13.12.2022 : off Hippogreif 2h earlier
Rev. 248 29.06.2023 : relays reorg. (1. defect & prep. extra heater)
Rev. 252 17.08.2023 : ECar loading ++
```

This file is the addendum to sweetHome.c and sweetHome.h containing process IO related issues, not necessary for pure HMI or logging related programs.

7.55.2 Function Documentation

7.55.2.1 switchHotWpump()

```
void switchHotWpump (
            uint8_t const on )
```

Hot water comfort pump turn on/off.

It turns the hot water comfort circulation pump on respectively off.

Parameters

on	!= 0 : on; else, ==0 : off

7.55.2.2 setPhPckLimit()

```
float setPhPckLimit (
            float powerLimit )
```

Set and get package switch power limit.

This function sets the PPS power limit in the range 0.0 ... PCK_POWER_LIM_MAX.

Parameters

powerLimit	PPS power limit in W

Returns

the (new) power limit

7.55.2.3 pckSwPec()

```
uint8_t pckSwPec (
            float power )
```

Set and get package switch percentage by power.

Besides determining an returning the percentage(power), this function sets the process control values via setPckSwPerc() returns its value.

Parameters

power	in W

Returns

the phase packet percentage 0..100

7.55.2.4 setPckSwPerc()

```
uint8_t setPckSwPerc (
            uint8_t perc )
```

Set package switch percentage.

This function sets phPckSwIndex by the parameter value in the range 0..200; respectively PCK_POWER_IND_MAX. Additionally it adjusts the current phPckCnt should its value be higher than by the new setting.

On transitions from respectively to 0 the control relay (PCK_POWER_REL) is actuated before respectively after actuating the electronic switch.

Parameters

perc	0..200; values above PCK_POWER_IND_MAX will have no effect and return the current setting un-altered

Returns

the actual (new) phase packet percentage 0..200

7.55.2.5 logBatteryStress()

```
void logBatteryStress ( )
```

Log battery stress on outLog as line with time stamp.

Logs the battery voltage and the under and overvoltage stress levels.

7.55.2.6 switchInvToBatUnl()

```
void switchInvToBatUnl (
            uint8_t on )
```

Switch inverter to battery unload.

When the parameter is !=0 respectively ON, this function switches the (small) inverter from panel to the battery unload step up converter and turns the extra panel work meter off.

When the parameter is off it does the other way round.

Parameters

on	true: switch inverter to step up converter (battery unload); 0, false: switch inverter to panel.

7.55.2.7 switchBatToStepUp()

```
void switchBatToStepUp (
            uint8_t on )
```

Switch battery to step up converter.

When the parameter is !=0 respectively ON, this function switches the battery to the step up converter.

When the parameter is off, this function switches the battery to the (max. 20 A) battery loader / keep alive module .

Parameters

on	true: switch battery to step up converter; 0, false: switch battery to load modul.

7.55.2.8 batKeepInhTimChg()

```
void batKeepInhTimChg (
            state_t *const me )
```

Inhibit battery keep alive loading timer state change function.

It turns the battery keep alive loading on when ending.

Parameters

me	pointer to the inhibit battery keep alive loading timer

7.55.2.9 setBatLoadPWM()

```
void setBatLoadPWM (
            uint8_t pwm )
```

Set battery loader module PWM.

This function sets the battery load module power PWM signal and hence its output voltage. 0 is the lowest and 255 the highest possible setting.

Parameters

pwm	the pwm ratio 0: 0%; 255: 100%

7.55.2.10 reduceBatLoadPWM()

```
void reduceBatLoadPWM (
            uint8_t const pwmRed )
```

Reduce battery loader module PWM.

This function reduces the battery load module power PWM signal and hence its output voltage by the parameter value, if that is not above the current setting.

See also

setBatLoadPWM()

Parameters

pwmRed	the reduction of the pwm ratio

7.55.2.11 befRiseTimChg()

```
void befRiseTimChg (
            state_t *const me )
```

Before sunrise timer state change callback.

On timer end .....

Parameters

me	pointer to the sunset timer

7.55.2.12 dawnTimChg()

```
void dawnTimChg (
            state_t *const me )
```

Dawn timer state change callback.

On timer end .....

Parameters

me	pointer to the dawn timer

7.55.2.13 sunriseTimChg()

```
void sunriseTimChg (
            state_t *const me )
```

Sunrise timer state change callback.

On timer end .....

Parameters

me	pointer to the sunrise timer

7.55.2.14 sunsetTimChg()

```
void sunsetTimChg (
            state_t *const me )
```

Sunset timer state change callback.

On timer end .....

Parameters

me	pointer to the sunset timer

7.55.2.15 duskTimChg()

```
void duskTimChg (
            state_t *const me )
```

Dusk timer state change callback.

On timer end .....

Parameters

me	pointer to the sunset timer

7.55.2.16 genSunStateText()

```
void genSunStateText (
            char * stateText,
            state_t const *const me,
            char const * stamp )
```

Generate the sun's status text.

This function makes the special log text for sun's times: set, rise etc..

Parameters

stateText	a character array supplied to hold the state text to be generated; minimal length 80.
me	pointer to own state; never null
stamp	(time) stamp to be prepended (max. length 23); default " - "

7.55.2.17 switchSolPow()

```
void switchSolPow (
            state_t *const me )
```

Solar power producer transition.

At the moment just switch the relays and set a flag. Note: trigger with pSolar

Parameters

me	pointer to the give away hysteresis

7.55.2.18 solPowStateText()

```
void solPowStateText (
            char * stateText,
            state_t const *const me,
            char const * stamp )
```

Generate solar power status text.

This function generates the log text for the solar power on/off discriminator.

Parameters

stateText	a character array supplied to hold the state text to be generated; minimal length 80.
me	pointer to own state; never null
stamp	(time) stamp to be prepended (max. length 23); default " - "

7.55.2.19 switchConsGiv()

```
void switchConsGiv (
            state_t *const me )
```

Power consumer / producer (give away) transition.

Triggered with pHome, at transitions relay and plug are switched for signalling via the switchConsGiv() SFC to have minimal on and off times.

At transitions to OFF, i.e. producer role imminent, battery ballast loading will be started (if not yet ON).

Parameters

me	pointer to the give away hysteresis

7.55.2.20 setConsGiv()

```
void setConsGiv (
            state_t *const me )
```

Power consumer / producer (give away) SFC transition.

At the moment just switch the relays and set a flag. Note: trigger with pHome (hyst)

Parameters

me	pointer to consumeGiveSFC

7.55.2.21 surplStateText()

```
void surplStateText (
            char * stateText,
            state_t const *const me,
            char const * stamp )
```

Generate surplus power status text.

This function generates the log text for the surplus power status machine.

Parameters

stateText	a character array supplied to hold the state text to be generated; minimal length 80.
me	pointer to own state; never null
stamp	(time) stamp to be prepended (max. length 23); default " - "

7.55.2.22 switchWouldGivAway()

```
void switchWouldGivAway (
            state_t *const me )
```

Would give away transition.

At the moment just switch the relays and set a flag. Note: trigger with pGiveAway

Parameters

me	pointer to the give away hysteresis

7.55.2.23 switchWouldGiv250W()

```
void switchWouldGiv250W (
            state_t *const me )
```

Would give 250W or more away, the transition.

At the moment just switch the relays and set a flag. Note: trigger with pGiveAway

Parameters

me	pointer to the give away hysteresis

7.55.2.24 mqttInit()

```
int mqttInit ( )
```

Initialise as MQTT client.

On success only: subscribe, loop and publish.

Returns

0: success the common mosq is set and usable; else: errno

7.55.2.25 mqttPlg01Set()

```
void mqttPlg01Set (
            uint8_t const on )
```

Switch the plug Plug01.

This function publishes the switch command via MQTT to the relay device Plug01, usually a Sonoff S20 with Tasmota.

Parameters

on	switch on when true, else off

7.55.2.26 mqttPlg02Set()

```
void mqttPlg02Set (
            uint8_t const on )
```

Switch the plug PLG2.

See mqttPlg01Set

7.55.2.27 mqttPlg03Set()

```
void mqttPlg03Set (
            uint8_t const on )
```

Switch the plug PLG3.

See mqttPlg01Set

7.55.2.28 mqttPlg04Set()

```
void mqttPlg04Set (
            uint8_t const on )
```

Switch the plug PLG4.

See mqttPlg01Set

7.55.3 Variable Documentation

7.55.3.1 phPckSwIndex

```
volatile uint8_t phPckSwIndex
```

Actual index (0..200%) determining power.

This is the control variable for the electric heater elements in the hot water tank. Until June 2023 there was one such element of PCK100PERC_POWER W and this variable had a range of 0..100%. Since July 2023 there are two heater elements of equal power. The range of this control variable was extended to 0..200%. The power distribution between the two heater elements is done by software; see setPckSwPerc(uint8_t const perc) and PCK_POWER_IND_MAX

7.55.3.2 batVoltValid

```
uint8_t batVoltValid
```

The measured battery voltage is valid.

When not 0 the last MQTT battery voltage measurement respectively message is not older than 2.4s and hence considered the actual valid value.

7.55.3.3 batUnloadAllowed

```
int batUnloadAllowed
```

Battery unload allowed.

Bit 0 (1): allow after sunset
Bit 1 (2): allow before sunrise
start value: 8 set default value (usually 2 in winter and 3 in summer) Note: Made int en lieu de uint_8 for use in getopt_long.

See also

BAT_UNL_SUMM

BAT_UNL_WINT

7.55.3.4 hundVsecAbove

```
uint32_t hundVsecAbove
```

Hundreds of Vs above 13.6 V battery voltage.

This variable is a counter for the battery voltage above 13.6 V taken in 0.01V units. This incremented Vs ("↩ Voltseconds") value is to indicate the battery's stress respectively gasing probability when being loaded.

Note: One hour at 15.0 V would give 504000 $*$ 0.01Vs

7.55.3.5 hundVsecBelow

```
uint32_t hundVsecBelow
```

Hundreds of Vs below 12.25 V battery voltage.

This variable is a counter for the battery voltage below 12.2 V taken in 0.01V units. This incremented Vs ("↩ Voltseconds") value is to indicate the battery's integral time below a load state permanently acceptable.

Note: One hour at 11.8 V would give 144000 ∗ 0.01Vs

7.55.3.6 hundredsV

```
uint16_t hundredsV
```

The battery voltage in 0.01V units.

This is just an integer value consistent to the last valid battery voltage measurement valFilVal.batVolt. Contrary to valFilVal.batVolt which is set to -0.9 to indicate invalidity after 2.4s without new (MQTT) measurements, this value will be kept (forever).
It is preset with 1289 (12.89 V) lest have battery low before the first valid MQTT measurement / message.

7.55.3.7 batKeepInh

```
state_t batKeepInh
```

Inhibit battery keep alive loading.

While running disallow battery keep alive with BAT_KEEP_PWM. This timer is initially OFF and will be stopped in advance by low battery voltage stress and battery keep command via HMI/GUI.
It will be started after battery unload, after battery ballast load and by battery off command via GUI.

7.55.3.8 sunriseTimer

```
state_t sunriseTimer
```

Sunrise timer.

This timer will run out every day at (approximated) sunrise. For the "every day" behaviour, its state change function will — after all due actions — restart this timer for the next 24h to hit (very approximately) the next sunrise. This acceptable guess will be adjusted at day change or at program start.

7.55.3.9 sunsetTimer

```
state_t sunsetTimer
```

Sunset timer.

This timer will run out every day at (approximated) sunset. See also sunriseTimer

7.55.3.10 befRiseTimer

```
state_t befRiseTimer
```

Before sunrise timer.

This timer will run out every day at about 180 min before (approximated) sunrise. The offset must be sufficient time to unload the battery before sunrise.
See also: sunriseTimer BEFORE_RISE

7.55.3.11 dawnTimer

```
state_t dawnTimer
```

Dawn timer.

This timer will run out every day at (approximated) dawn, meaning the beginning of civil twilight. See also sunriseTimer and duskTimer

7.55.3.12 duskTimer

```
state_t duskTimer
```

Dusk timer.

This timer will run out every day at (approximated) dusk, meaning the end of civil twilight. About this time street lamps and position lights may be lit. See also sunriseTimer

7.55.3.13 consumeGiveHyst

```
state_t consumeGiveHyst
```

Give away hysteresis.

thresholds: PDEL_SGGIVE, PDEL_SGCONS

7.55.3.14 consumeGiveSFC

```
state_t consumeGiveSFC
```

Give away state machine / SFC.

Status ON means consumer: OK.
Status OFF means very low power consumption: Inhibit power delivery.

7.55.3.15 wouldGiveAwayHyst

```
state_t wouldGiveAwayHyst
```

Would give away hysteresis.

thresholds (04.03.18): -/+ 7W

7.55.3.16 wouldGive250WHyst

```
state_t wouldGive250WHyst
```

Would give away 250W hysteresis.

thresholds (04.03.18): -250 / -178W

7.55.3.17 mqttHost

```
char mqttHost[68]
```

The MQTT broker URL or name.

May be set by option –mqttHost meterPi or –mqttBroker 192.168.178.87

default: localhost MQTTBroker (currently localhost !)

7.55.3.18 subTopStPlg01

```
char subTopStPlg01[14]
```

State sub topic of S20 plug Number 01 to 09.

It's preset as plug01/POWER for 01, but the digit at index [5] will be set before each use accordingly.

7.55.3.19 clientId

```
char clientId[38]
```

MQTT client ID.

default value: sweetHomeControl; length: 15; max. length: 36
May be changed before mqttInit().

MQTT client ID.

7.56 weRasp/sysBasic.c File Reference

Some system related basic functions for Raspberry Pis.

```
#include "sysBasic.h"
```

Functions

- uint8_t **errLogIsStd** (void)

  Error log (errLog) is standard stream or outLog.
- int formatDec2Digs (char *targTxt, uint32_t value)

  Format number as two digit decimal number with leading zeroes.
- int formatDec3Digs (char *targTxt, uint32_t value)

  Format number as three digit decimal number with leading zeroes.
- int formatTmTim (char *rTmTxt, struct tm *rTm)

  Format broken down real time and date as standard text.
- int formatTmTiMs (char *rTmTxt, struct tm *rTm, int millis)

  Format broken down real time clock+ms as standard text.
- uint8_t isFNaN (float const val)

  Floating point NaN.
- uint8_t littleEndian ()

  Actual runtime / architecture is little endian.
- void logEventText (char const *txt)

  Log an event/log message on outLog.
- void monoTimeInit (timespec *timer)

  Absolute timer initialisation.
- uint8_t **outLogIsStd** (void)

  Event log (outLog) is standard stream.
- void printNamRevDat (void)

  Print the program name, SVN revision and date.
- void printRevDat (void)

  Print the program SVN revision and date.
- char const * progDat ()

  The program date.
- char const * progNam ()

  The program name.
- char const * progNamB ()

  The program name with blank.
- char const * progRev ()

  The program revision.
- size_t strlcat (char *dest, char const *src, size_t num)

  String concatenation with limit.
- size_t strlcpy (char *dest, char const *src, size_t const num)

  String copy with limit.
- int switchErrorLog (char const *const errFilNam)

  Switch errlog to other file.
- int switchEventLog (char const *const logFilNam)

  Switch outLog to other file.
- void timeAddNs (timespec *t1, long ns)

  Add a ns increment to a time overwriting it.
- int timeStep (timespec *timeSp, unsigned int micros)

  A delay to an absolute step specified in number of s to a given time.
- void updateReaLocalTime (void)

  Update local real time.

Variables

- timespec **actRTime**

  Actual time (structure, real time clock).
- struct tm actRTm

  Actual time (broken down structure / local).
- float const cosDiY [192]

  Cosine of day in year, look up.
- int16_t const cosDiY60 [192]

  Cosine of day in year ∗ 60.
- char const dec2digs [128][2]

  Format two digit decimal, leading zero, by lookup.
- char const dec3digs [1024][4]

  Format three digit decimal, leading zero, 0-terminated, by lookup.
- char const dow [9][4]

  English weekdays, two letter abbreviation.
- FILE ∗ errLog

  Error log output.
- char const fType [16][8]

  Translation of directory entry typed to 8 char text.
- char const ∗const lckPiGpioPth

  Common path to a lock file for GpIO use.
- __time_t localMidnight

  Actual local midnight.
- uint32_t noLgdEvnt

  Number of events logged.
- FILE ∗ outLog

  Event log output.
- uint8_t **progNamLen**

  char progNamPure[] = PROGNAME;
- int retCode

  Basic start-up function failure.
- int todayInYear

  Today's day in year.
- int useErrLogFiles

  Log on files.
- uint8_t useOutLog4errLog

  Use outLog for errors too.
- __time_t utcMidnight

  Actual (local) UTC midnight.
- char const zif2charMod10 [44]

  The digits 0..9 repeated as 44 characters.

7.56.1 Detailed Description

Some system related basic functions for Raspberry Pis.

```
Copyright  (c)  2020   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \     /       \|   |
     \/  \/   \__/          \__/|_
```

Revision history

```
Rev. 253 9.10.2023
Rev.  105 06.02.2018 : new (transfered parts from sysutil.c)
Rev.  108 12.02.2018 : parts moved out, one event file; Started to get
                       Doxygen usable. Note: Not yet done.
Rev. 147 16.06.2018 : time handling enhanced improved
Rev. 155 27.06.2018 : time handling debugged
Rev. 164 11.07.2018 : sunset/sunrise location parameters; string functions
Rev. 229 23.07.2020 : UTF 8 BOM for log and error file
```

cross-compile by:

```
arm-linux-gnueabihf-gcc -DMCU=BCM2837 -I./include -c -o weRasp/sysBasic.o weRasp/sysBasic.c
```

For documentation see the include file sysBasic.h

7.56.2 Function Documentation

7.56.2.1 progNam()

```
char const * progNam ( )
```

The program name.

Returns

the program's name as pure text, "homeDoorPhone", e.g.

7.56.2.2 progNamB()

```
char const * progNamB ( )
```

The program name with blank.

Same as progNam but with at least one trailing blank or so many blanks to get a minimal length of 17, , "home↩ DoorPhone ", e.g.

Returns

the program's name with trailing blank(s)

7.56.2.3 progRev()

```
char const * progRev ( )
```

The program revision.

Returns

the program's SVN revision as pure text, "0", "341" e.g.

7.56.2.4 progDat()

```
char const * progDat ( )
```

The program date.

Returns

the program's SVN date "2020-07-23" e.g., length 10

7.56.2.5 printRevDat()

```
void printRevDat (
            void  )
```

Print the program SVN revision and date.

This function prints a line in the form (4 leading blanks)
`Revision  229 (2020-07-23)`

to outLog

7.56.2.6 printNamRevDat()

```
void printNamRevDat (
            void  )
```

Print the program name, SVN revision and date.

This function prints a line in the form (4 leading blanks)
`theLittleProg    R.  229 (2020-07-23)`

to outLog

7.56.2.7 isFNaN()

```
uint8_t isFNaN (
            float const val )
```

Floating point NaN.

Parameters

val	the floating point value to be checked for IEEE754 NaN

Returns

0xFF (true) when not a number, else 0

7.56.2.8 littleEndian()

```
uint8_t littleEndian ( )
```

Actual runtime / architecture is little endian.

This boolean function is evaluated by char* to int comparison.

To save runtime resources use the marco PLATFlittlE instead, which would fall back to littleEndian() (this function) when no target platform informations on endianness are available.

Returns

true when platform is little endian (evaluated at run time)

7.56.2.9 switchErrorLog()

```
int switchErrorLog (
            char const *const errFilNam )
```

Switch errlog to other file.

Parameters

errFilNam	the name of the file to switch to; NULL or empty: switch (back) to stderr

Returns

96 : file name can't be opened for append, old state kept; 97 : useOutLog4errLog is ON, nothing done 0 : OK

7.56.2.10 switchEventLog()

```
int switchEventLog (
            char const *const logFilNam )
```

Switch outLog to other file.

If useOutLog4errLog is ON the errLog file will point to the same named file on success.

Parameters

logFilNam	the name of the file to switch to; NULL or empty: switch (back) to stdout

Returns

96 : file name can't be opened for append; old state kept.

7.56.2.11 logEventText()

```
void logEventText (
            char const * txt )
```

Log an event/log message on outLog.

If txt is not null it will be output to outLog and outLog will be flushed. No line feed will be appended; the text is put as is.

Parameters

txt	text to be output; n.b not LF appended and not counted as line

7.56.2.12 monoTimeInit()

```
void monoTimeInit (
            timespec * timer )
```

Absolute timer initialisation.

This function sets the time structure provided to the current absolute monotonic ABS_MONOTIME (default↩ : CLOCK_MONOTONIC).

Note: Error returns, suppressed here, cannot occur, as long as the time library functions and used clock IDs are implemented. Otherwise all else timing done here would fail completely.

Parameters

timer	the time structure to be used (never NULL!)

7.56.2.13 timeStep()

```
int timeStep (
```

```
            timespec * timeSp,
            unsigned int micros )
```

A delay to an absolute step specified in number of s to a given time.

This function does an absolute monotonic real time delay until timer += micros;

Chaining this calls can give absolute triggers relative to a given start. One must initialise the time structure timespec before every start of a new cycle chain. Afterwards the structure time must not be written to. See timeAddNs, ABS_MONOTIME and monoTimeInit (or clock_gettime).

Chaining absolute delays accomplishes long term exact periods respectively cycles. See also explanations in ABS_MONOTIME (default: CLOCK_MONOTONIC).

Parameters

timeSp	the time structure to be used (never NULL!)
micros	delay in s (recommended 100s .. 1h)

Returns

sleep's return value if of interest (0: uninterrupted)

7.56.2.14 timeAddNs()

```
void timeAddNs (
            timespec * t1,
            long ns )
```

Add a ns increment to a time overwriting it.

Parameters

t1	the time structure to add to (not NULL!, will be modified)
ns	the increment in nanoseconds

7.56.2.15 updateReaLocalTime()

```
void updateReaLocalTime (
            void  )
```

Update local real time.

This function initialises / updates both actRTime and actRTm.

7.56.2.16 strlcpy()

```
size_t strlcpy (
            char * dest,
            char const * src,
            size_t const num )
```

String copy with limit.

This function copies at most num - 1 characters from src to dst. If not terminated by a 0 from src, dest[num-1] will be set 0. Hence, except for num == 0, dest will be 0-terminated.

The value returned is the length of string src; if this value is not less than num truncation occurred.

Hint: This function resembles the one from bsd/string.h usually not available with standard Linuxes and Raspbians .

Parameters

dest	the character array to copy to; must not be shorter than num
src	the string to copy from
num	the maximum allowed string length of dest

Returns

the length of src

7.56.2.17 strlcat()

```
size_t strlcat (
            char * dest,
            char const * src,
            size_t num )
```

String concatenation with limit.

This function appends at most num - 1 characters from src to the end of dest. If not terminated by a 0 from src, dest[num-1] will be set 0. Hence, except for num == 0, dest will be 0-terminated.

The value returned is the length of string src (if no truncation occurred).

Hint: This function resembles the one from bsd/string.h usually not available with standard Linuxes and Raspbians .

Parameters

dest	the character array to copy to; must not be shorter than num
src	the string to copy from
num	the maximum allowed string length of dest

Returns

the length of src

7.56.2.18 formatDec2Digs()

```
int formatDec2Digs (
            char * targTxt,
            uint32_t value )
```

Format number as two digit decimal number with leading zeroes.

The format is: 00 to 99

The length is always 2. There is no trailing character zero appended.
returned is the number of leading zeroes in the range 0 o 1. N.B. the value 0 yielding "00" is considered to have one leading zero.

See also

dec2digs formatDec3Digs

Parameters

targTxt	pointer to the target text buffer, must have place for 3 characters (!)
value	the value to be formatted; values outside 0 .. 999 will yield incorrect results

Returns

the number of leading zeroes (0 or 1)

7.56.2.19 formatDec3Digs()

```
int formatDec3Digs (
            char * targTxt,
            uint32_t value )
```

Format number as three digit decimal number with leading zeroes.

The format is: 000 to 999

The length is always 3. There is no trailing character zero appended.
returned is the number of leading zeroes in the range 0 to 2. N.B. the value 0 yielding "000" is considered to have 2 leading zeroes.

See also

dec3digs formatDec2Digs

Parameters

targTxt	pointer to the target text buffer, must have place for 3 characters (!)
value	the value to be formatted; values outside 0 .. 999 will yield incorrect results

Returns

the number of leading zeroes (0..2)

7.56.2.20 formatTmTim()

```
int formatTmTim (
            char * rTmTxt,
            struct tm * rTm )
```

Format broken down real time and date as standard text.

The format is: Fr 2017-10-20 13:55:12 UTC+20 .....0123456789x123456789v123456789t The length is 29. See formatTmTiMs() for a longer format with 3 digit ms.

Parameters

rTmTxt	pointer to the target text buffer, must have place for 30 characters (!)
rTm	pointer to broken down real time; NULL will take actRTm

Returns

the number of characters put (should be 28) or 0: error (rTmTxT NULL)

7.56.2.21 formatTmTiMs()

```
int formatTmTiMs (
            char * rTmTxt,
            struct tm * rTm,
            int millis )
```

Format broken down real time clock+ms as standard text.

/code The format is: Fr 2017-10-20 13:55:12.987 UTC+20 .....0123456789x123456789v123456789t123 +3 .....0123456789x123456789v123456789t /endcode The length is 33. See formatTmTim() for a shorter format without ms.

Parameters

rTmTxt	pointer to the target text buffer, must have place for 34 characters (!)
rTm	pointer to broken down real time; NULL will take actRTm
millis	milliseconds 0..999 supplement to rTm

Returns

the number of characters put (should be 32) or 0: error (rTmTxT NULL)

7.56.3 Variable Documentation

7.56.3.1 retCode

```
int retCode
```

Basic start-up function failure.

Allows for compact code without saving the (error) return:
if (openLock(lckPiGpioPth, ON)) return retCode;

Storage for return/error codes. Used by: openLock(char const ∗, uint8_t) theCyclistStart(int) theCyclistWaitEnd()

Value: 0: OK, else: error

7.56.3.2 lckPiGpioPth

```
char const* const lckPiGpioPth
```

Common path to a lock file for GpIO use.

Programs using GPIO in any form usually (and forced by some libraries) have to do this exclusively. This is implemented here by locking a file named /home/pi/bin/.lockPiGpio
Make the lock file by: touch /home/pi/bin/.lockPiGpio

Without locking this file those programs must not start.
So, deleting this file inhibits the start even by cron etc.

7.56.3.3 useErrLogFiles

```
int useErrLogFiles
```

Log on files.

If true (default) logging and errors go to files or one file, otherwise to console

7.56.3.4 outLog

```
FILE* outLog
```

Event log output.

default: standard output; may be put to a file.

7.56.3.5 noLgdEvnt

```
uint32_t noLgdEvnt
```

Number of events logged.

Counter for lines put to or events logged on outLog.

7.56.3.6 useOutLog4errLog

```
uint8_t useOutLog4errLog
```

Use outLog for errors too.

When set true errLog will be set to outLog when using files. In this case there is just one event log file. Hence, doubling the same entry to both errLog and outLog should be avoided.

7.56.3.7 errLog

```
FILE* errLog
```

Error log output.

default: standard error; may be put to a file.

7.56.3.8 actRTm

```
struct tm actRTm
```

Actual time (broken down structure / local).

This structure is initialised may be updated by some timing and cyclic functions. See initStartRTime() and others.

7.56.3.9 todayInYear

```
int todayInYear
```

Today's day in year.

The value should be set at start (will be by updateReaLocalTime()) and updated at midnight.

7.56.3.10 utcMidnight

```
__time_t utcMidnight
```

Actual (local) UTC midnight.

This is the actual "local" UTC midnight. "Local" means that on early hours, i.e. those within zone offset, UTC midnight will be corrected to the next (east of Greenwich) respectively previous day (west). The rationale is to point to the same day or date at day time (around Europe).
Or, to put it simple, utcMidnight is to be set so, that the equation
local Midnight = utcMidnight - UTV offset
holds.

The value will will be set correctly by updateReaLocalTime(). It should be updated at day change (if used).

7.56.3.11 localMidnight

```
__time_t localMidnight
```

Actual local midnight.

This is the UTC Linux timestamp of the actual day's / time's local midnight. See also utcMidnight for explanations.

Note: On days with DST changes this value will shift within the day. It's mostly better to make calculations relative to day start — sunrise and sunset e.g. — relative to UTC midnight.

7.56.3.12 cosDiY

```
float const cosDiY[192]
```

Cosine of day in year, look up.

This lookup table provides the cosine by the day of the year without resource eating floating point arithmetic or math.h. The rationale is the approximate calculation of sunrise and sunset times based on earliest, latest and delta for any given location within the arctic circles.

The length of the look up table is abundant 192. According to cosine's periodic properties it shall be used in the range 0..183 by applying the following operations to the day in year value
absolute when $<$ 0,
modulo FOURYEARS when $>=$ FOURYEARS,
modulo 365 when $>=$ 365 and
x = 365 - x when $>$ 190.

Note: These rules are implemented in the function cosDay() and in the function cosDay60() using the look up table cosDiY60

7.56.3.13 cosDiY60

```
int16_t const cosDiY60[192]
```

Cosine of day in year $*$ 60.

This look up table is the same as cosDiY, except the values being multiplied by 60 which includes minutes to seconds conversion, avoiding a multiplication and all floating point operations for some applications.

lenght: 192

7.56.3.14 dow

```
char const dow[9][4]
```

English weekdays, two letter abbreviation.

Monday (Mo) is 1; Sunday (Su) is 7 or, also, 0.

7.56.3.15 dec2digs

```
char const dec2digs[128][2]
```

Format two digit decimal, leading zero, by lookup.

"00" .. "99" + "00", "_1" .. "_7"

7.56.3.16 dec3digs

```
char const dec3digs[1024][4]
```

Format three digit decimal, leading zero, 0-terminated, by lookup.

"000" .. "999" + "000" .. "023"

7.56.3.17 zif2charMod10

```
char const zif2charMod10[44]
```

The digits 0..9 repeated as 44 characters.

By using a number 0..43 as index this will give modulo 10 respectively the last decimal digit as character.

7.56.3.18 fType

```
char const fType[16][8]
```

Translation of directory entry typed to 8 char text.

direntry.d_type as index in the range 0..15 gives an 8 character short type text. Note: Only 0, 1, 2, 4, 6, 8, 10, 12 and 14 are defined d_type values. The undefined ones give undef3 .. undefF

7.57 weRasp/we1wire.c File Reference

Functions for 1-wire sensors.

```
#include "we1wire.h"
#include <sys/file.h>
```

Functions

- int getTemp (oneWireDevice_t *const tempSensor)

  Get temperature.
- void initTempSensor (oneWireDevice_t *const tempSensor, char const *const name, char const *const valueFile)

  Initialise a 1-wire sensor structure.

7.57.1 Detailed Description

Functions for 1-wire sensors.

```
Copyright  (c)  2018   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 234 20.01.2021
Rev. 168 21.07.2018 : new
```

See file include/we1wire.h

7.57.2 Function Documentation

7.57.2.1 getTemp()

```
int getTemp (
             oneWireDevice_t *const tempSensor )
```

Get temperature.

This function tries a new measurement on the 1-wire temperature sensor provided. On success the new value is returned. On failure the last good value is returned, but at most 7 times after 7 good readings before. On total failure -99900 (-99.9 °C, 0 K, BAD_TEMP_READ) is returned.

Additionally on temperature changes, the integer reading (.value) and the floating point string (valueGrdC[]) will be set in the structure.

Parameters

tempSensor	pointer to the sensor's structure

Returns

the actual or last reading or -2721500 if no good / not enough past good readings

7.57.2.2 initTempSensor()

```
void initTempSensor (
             oneWireDevice_t *const tempSensor,
             char const *const name,
             char const *const valueFile )
```

Initialise a 1-wire sensor structure.

This function does basic initial settings for a 1-wire temperature sensor. tempSensor.name and tempSensor.value↩
File are set by the respective parameters.
The temperature values are set to bad value; see BAD_TEMP_READ and BAD_TEMP_FLOAT

Parameters

tempSensor	pointer to the sensor's structure (never NULL!)
name	the sensor's short name or its directory name; NULL / empty: no change
valueFile	the canonical absolute path to its value file; NULL / empty: no change

7.58 weRasp/weAR_N4105.c File Reference

support for VDE-AR-N 4105

```
#include "weAR_N4105.h"
#include "weStateM.h"
#include "weUtil.h"
```

Functions

- void arn4105TimChg (state_t *const me)

  AR-N 4105 control timer state change function.
- void setARN4105state (uint8_t const state, uint8_t const select)

  Set AR-N 4105 frequency and voltage states.

Variables

- uint8_t arn4105state

  AR-N 4105 frequency and voltage states.
- state_t arn4105Timer

  AR-N 4105 control timer.
- state_t checkL1_U

  The (one) five band checker for line L1 voltage.
- state_t checkL2_U

  The (one) five band checker for line L2 voltage.
- state_t checkL3_U

  The (one) five band checker for line L3 voltage.
- state_t checkLineFrq

  The (one) five band checker for line frequency.

7.58.1 Detailed Description

support for VDE-AR-N 4105

```
Copyright  (c)  2019   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 234 20.01.2021
Rev. 195 01.03.2019 : new
Rev. 202 28.04.2019 : minor (timer macro change)
```

7.58.2 Function Documentation

7.58.2.1 setARN4105state()

```
void setARN4105state (
            uint8_t const state,
            uint8_t const select )
```

Set AR-N 4105 frequency and voltage states.

This function is called internally by the respective five band checker's state_t.onStateChange function.

Parameter selection mask must be one of LfMsk, L1Msk - L3Msk.

Parameters

state	state_t.status of the five band checker in question
select	selection mask

7.58.2.2 arn4105TimChg()

```
void arn4105TimChg (
            state_t *const me )
```

AR-N 4105 control timer state change function.

It signals the timer state to the function arn4105TimChg.

Parameters

me	pointer to the AR-N 4105 control timer

7.58.3 Variable Documentation

7.58.3.1 checkLineFrq

```
state_t checkLineFrq
```

The (one) five band checker for line frequency.

See newLineFcheck

7.58.3.2 checkL1_U

```
state_t checkL1_U
```

The (one) five band checker for line L1 voltage.

In a one phase system this will be the only one fed with values by fiveBandTick.

See newLineUcheck

7.58.3.3 checkL2_U

```
state_t checkL2_U
```

The (one) five band checker for line L2 voltage.

See newLineUcheck

7.58.3.4 checkL3_U

```
state_t checkL3_U
```

The (one) five band checker for line L3 voltage.

See newLineUcheck

7.58.3.5 arn4105state

```
uint8_t arn4105state
```

AR-N 4105 frequency and voltage states.

If all is OK this status byte is 0. The meaning of bits set is

```
|   7    |   6    |   5    |   4    ||   3    |   2    |   1    |   0    |
|   L3   |   L2   |   L1   |   f    ||   L3   |   L2   |   L1   |   f    |
|       voltage error       |  error ||      voltage warning     | warning|
```

If any error bit is set the respective generators has to be cut off.
Ten minutes after the last warning bit is gone the respective generators may be put back to power line.
The timer arn4105Timer will be handled accordingly.

User software should not touch this variable (except when knowing the consequences.

7.58.3.6 arn4105Timer

```
state_t arn4105Timer
```

AR-N 4105 control timer.

When this timer is running distributed small generators must be cut off. When this timer stops the respective generators may be put back to power line.

User software must provide a function switchARN4105 doing the cutoff and switch back plus optionally logging. It will be called by this timer and, hence, in the end by the frequency and voltage checkers.

The user/application software must check (timerTickCheck) regularly to enable switch back.

7.59 weRasp/weCGIajax.c File Reference

Functions and values for Web interfaces with AJAX, CGI etc.

```
#include "weCGIajax.h"
```

Functions

- int getQSparam (char const *const name, char *const value, int const vLen)

  Get the query string parameter value for a unique and known key.
- int getQueryString ()

  Fetch and store the query string.
- int jsonBinForm (uint8_t const indent, char const *const name, uint8_t const value, char *end)

  Output an eight bit value binary formated as JSON name:value.
- int jsonBreading (uint8_t const indent, char const *const name, uint8_t const value, char *end)

  Output a boolean value as JSON name:value.
- int jsonFPreadingT (uint8_t const indent, char const *const name, uint16_t const valueLo, uint16_t const valueHi, uint8_t dotPos, char const *const unit, char *end, char *title)

  Output a physical value as JSON {object} with name, 32 bit FP value and unit.
- int jsonFreading (uint8_t const indent, char const *const name, float const value, char const *const unit, uint8_t const fractional, char *end)

  Output a float (physical value as JSON {object} with name, value and unit.
- int jsonFreadingT (uint8_t const indent, char const *const name, float const value, char const *const unit, uint8_t const fractional, char *end, char *title)

  Output a float value as JSON {object} with name, value, unit and help text.
- int jsonIreading (uint8_t const indent, char const *const name, int const value, char *end)

  Output an int value as JSON name:value.
- int jsonSreadingT (uint8_t const indent, char const *const name, char *const value, char const *const unit, char *end, char *title)

  Output a physical value as JSON {object} with name, value as String and unit.

Variables

- char **actQSPvalue** [QS_MAX_VAL_LEN]

  to be used for actual parameter
- char **queryString** [QS_MAX_STRING_LEN]

  holding the query string
- char const * **requestMethod**

  to hold (environment's) REQUEST_METHOD

7.59.1 Detailed Description

Functions and values for Web interfaces with AJAX, CGI etc.

```
Copyright  (c)  2018   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 249 30.06.2023
Rev.  82  14.12.2017 : new
Rev. 187  14.10.2018 : minor typos
```

cross-compile by:

```
arm-linux-gnueabihf-gcc -DMCU=BCM2837 -I./include -c -o weRasp/weCGIajax.o weRasp/weCGIajax.c
```

This is a (basic) library to support CGI programs written in C to be used under a web sever – as e.g. an Apache 2.4 on a Pi for hometersControl.

For documentation see the include file weCGIajax.h

7.59.2 Function Documentation

7.59.2.1 getQueryString()

```
int getQueryString ( )
```

Fetch and store the query string.

This function gets the raw query string storing it (on success) in queryString. Raw means, all the ugly '+' and 'AB' things are still there.

Returns

0: from get; 1: from get, truncated; 2: from post; 3: from get, truncated; -1: program probably not run as CGI, no query string

7.59.2.2 getQSparam()

```
int getQSparam (
            char const * name,
            char *const value,
            int const vLen )
```

Get the query string parameter value for a unique and known key.

This simple method would fetch the value to a known query string parameter.

Warning: Longer names/keys with the same prefix must come first. When having "carlength=91&length=300", this (simple) function will get "91" for "length".

Parameters

name	the parameter's name or key
value	character array (string) to store the value to, the result may be empty (on name\0, name=\0 or name&)
vLen	value's length including the terminating 0

Returns

0: OK, got value; -1: got no value (no name, no value, ...) 1: got value truncated to vLen -1 characters

7.59.2.3 jsonIreading()

```
int jsonIreading (
            uint8_t const indent,
            char const *const name,
            int const value,
            char * end )
```

Output an int value as JSON name:value.

This function puts an integer value as "name": "999" respectively "name": "-1" to the standard output.

The text end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " " ",\n" "\n") may enhance human readability.

Parameters

indent	0..36 number of spaces to put before
name	the property's name
value	the integer value (put as string)
end	0-terminated string to put at the end

Returns

>=0: OK; <0: error

7.59.2.4 jsonBreading()

```
int jsonBreading (
            uint8_t const indent,
            char const *const name,
            uint8_t const value,
            char * end )
```

Output a boolean value as JSON name:value.

This function puts a boolean value as "name": true or as "name": false to the standard output.

The text end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " " ",\n" "\n") may enhance human readability.

Parameters

indent	0..36 number of spaces to put before
name	the property's name
value	0: false; else: true
end	0-terminated string to put at the end

Returns

>=0: OK; <0: error

7.59.2.5 jsonBinForm()

```
int jsonBinForm (
            uint8_t const indent,
            char const *const name,
            uint8_t const value,
            char * end )
```

Output an eight bit value binary formated as JSON name:value.

This function puts a boolean value as "name": "1001_1100" to the standard output.

The text end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " " ",\n" "\n") may enhance human readability.

Parameters

indent	0..36 number of spaces to put before
name	the property's name
value	0..255 i.e 0000_0000 .. 1111_1111
end	0-terminated string to put at the end

Returns

>=0: OK; <0: error

7.59.2.6 jsonFreading()

```
int jsonFreading (
            uint8_t const indent,
            char const *const name,
            float const value,
            char const *const unit,
```

```
            uint8_t const fractional,
            char * end )
```

Output a float (physical value as JSON {object} with name, value and unit.

This function puts a float value as e.g. {"name": "Wimp", "value": "15.22", "unit": "kWh"}

The text by parameter end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " ",\n" "\n") may enhance human readability.

Parameters

indent	0..36 number of spaces to put before
name	0-terminated string naming the reading (measured value e.g.)
value	the float value
unit	0-terminated string being the value's physical unit
fractional	0..9 number of fractional digits 0: value will be taken as int
end	0-terminated string to put at the end

Returns

>=0: OK; <0: error

7.59.2.7 jsonFreadingT()

```
int jsonFreadingT (
            uint8_t const indent,
            char const *const name,
            float const value,
            char const *const unit,
            uint8_t const fractional,
            char * end,
            char * title )
```

Output a float value as JSON {object} with name, value, unit and help text.

This function puts a float value as e.g. {"name": "Wimp", "value": "15.22", "unit": "kWh"}

The text by parameter end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " ",\n" "\n") may enhance human readability.

The text by parameter title will be used as such, acting as hover help text (not title) by html tradition. This text must not be NULL nor empty. When wanting no help text use jsonFreading() instead.

Parameters

indent	0..36 number of spaces to put before
name	0-terminated string naming the reading (measured value e.g.)
value	the float value
unit	0-terminated string being the value's physical unit
fractional	0..9 number of fractional digits 0: value will be taken as int
end	0-terminated string to put at the end
title	0-terminated string as help text

Returns

>=0: OK; <0: error

7.59.2.8 jsonSreadingT()

```
int jsonSreadingT (
            uint8_t const indent,
            char const *const name,
            char *const value,
            char const *const unit,
            char * end,
            char * title )
```

Output a physical value as JSON {object} with name, value as String and unit.

This function puts a value given as String "15.22" e.g. in the form {"name": "Wimp", "value": "15.22", "unit": "kWh"}

The text end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " " ",\n" "\n") may enhance human readability.

Parameters

indent	0..36 number of spaces to put before
name	0-terminated string naming the reading (measured value e.g.)
value	the float value
unit	0-terminated string being the value's physical unit
end	0-terminated string to put at the end
title	0-terminated string as help text (NULL no title)

Returns

>=0: OK; <0: error

7.59.2.9 jsonFPreadingT()

```
int jsonFPreadingT (
            uint8_t const indent,
            char const *const name,
            uint16_t const valueLo,
            uint16_t const valueHi,
            uint8_t dotPos,
            char const *const unit,
            char * end,
            char * title )
```

Output a physical value as JSON {object} with name, 32 bit FP value and unit.

This function puts a value given as 32 bit fixed point value in two 16 bit parts in the form {"name": "Wimp", "value": "15.22", "unit": "kWh"}. Accepting the vale as two parts solves problems with 32 bit integers or fixed point values delivered as two 16 bit Modbus registers in arbitrary order. The value is specified by the parameters valueLo, valueHi and dotPos.
With valueHi == 0 valueLo is considered as 16 bit value

The text end is appended. According to JSON syntax rules it must contain a comma (,) if and only if other elements follow. Besides that white space (", " " " ",\n" "\n") may enhance human readability.

Parameters

indent	0..36 number of spaces to put before
name	0-terminated string naming the reading (measured value e.g.)
valueLo	the least significant 16 bits of the fixed point value
valueHi	the most significant 16 bits of the fixed point value
dotPos	position where the fixed point is 0..6; 0 means integer
unit	0-terminated string being the value's physical unit
end	0-terminated string to put at the end
title	0-terminated string as help text (NULL no title)

Returns

>=0: OK; <0: error

7.60 weRasp/weDCF77.c File Reference

DCF77 decoder on Raspberry Pi.

```
#include "weDCF77.h"
```

Functions

- void dcf77receiveRec (int pi, unsigned gpio, unsigned level, uint32_t tick)

  DCF77 receive recorder.
- int dcf77receiveRecDeregister (void)

  DCF77 receive function de-registration.
- int dcf77receiveRecRegister (void)

  DCF77 receive recorder registration.
- durDiscrPointData_t * disc5 (durDiscrPointData_t table[ ], uint32_t const value)

  Discriminating a value.
- uint32_t initDCF77io ()

  Initialise the DCF77 signal input GPIO / pin.
- void setReceiver (int const level)

  Set receiver On control.

Variables

- unsigned curBCDnum

  Number decoded from period sequence parts.
- int **dcf77callbackID**

  PiGpioD's call back ID for receiver function.
- unsigned dcf77glitch

  DCF77 input's glitch filter time setting.
- unsigned dcf77inp

  Input GPIO for the DCF77 receiver's AM signal.
- unsigned dcf77invInp

  Inverted DCF77 receiver's signal.
- unsigned dcf77lastLevel

  Last DCF77 modulation level.
- unsigned dcf77PUD

  DCF77 input's pull resistor setting.
- unsigned dcf77recCnt

  Control output GPIO of the DCF77 receiver's control.
- unsigned dcf77recCntInv

  Receiver On control inverted.
- dcf77recPerData_t **dfc77actRecPer**

  The actual respectively last modulation period data received.
- dcf77recPerData_t **dfc77ringBrecPer** [DCF77RINGbufWRAP+1]

  Ring buffer of modulation period data received.
- uint8_t **dfc77ringBrecWInd**

  Modulation period data received ring buffer write index.
- char lastSysClk [14]

  The system time for low AM as text hh:mm:ss.
- unsigned const **num02st** [ ]

  see numBCDinit, values 0 2
- unsigned const **num04st** [ ]

  see numBCDinit, values 0 4
- unsigned const **num08st** [ ]

  see numBCDinit, values 0 8
- unsigned const **num10st** [ ]

  see numBCDinit, values 0 10
- unsigned const **num20st** [ ]

  see numBCDinit, values 0 20
- unsigned const **num40st** [ ]

  see numBCDinit, values 0 40
- unsigned const **num80st** [ ]

  see numBCDinit, values 0 80
- unsigned const numBCDinit [ ]

  Initialisation or least significant BDC digit for modulation time.
- durDiscrPointData_t perDiscH [5]

  Discrimination values for the modulation period.
- durDiscrPointData_t perDiscP [5]

  Discrimination values for the modulation period.
- durDiscrPointData_t * perDiscUsed

  Discrimination values for the modulation period used.
- durDiscrPointData_t timDiscH [5]

Discrimination values for the 15% modulation time.

- durDiscrPointData_t timDiscHs [5]

  Discrimination values for the 15% modulation time.

- durDiscrPointData_t timDiscP [5]

  Discrimination values for the 15% modulation time.

- durDiscrPointData_t * timDiscUsed

  Discrimination values for the 15% modulation time.

7.60.1 Detailed Description

DCF77 decoder on Raspberry Pi.

```
Copyright  (c)   2020     Albrecht Weinert
weinert-automation.de     a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 240 10.04.2021
Rev. 233 17.10.2020 : new
Rev. 234 31.12.2020 : HQ DCF77 timing set modified
Rev. 239 02.03.2021 : functions from dcf77onPi.c ported here
```

This is a supplementary basic library to handle the signal of a DCF77 receiver. In the case of AM (amplitude) modulation a 1 (high) at the signal input means 15% amplitude and a 0 (low) 100% respectively full amplitude. The other way round is marked as dcf77invInp.

7.60.2 Function Documentation

7.60.2.1 setReceiver()

```
void setReceiver (
             int const level )
```

Set receiver On control.

Parameters

level	set receiver ON or OFF

See also

dcf77recCnt, dcf77recCntInv

7.60.2.2 initDCF77io()

```
uint32_t initDCF77io ( )
```

Initialise the DCF77 signal input GPIO / pin.

The signal input dcf77inp and control output dcf77recCnt are initialised.

Returns

the bank mask of the output dcf77recCnt if given, 0 if PINig

7.60.2.3 dcf77receiveRec()

```
void dcf77receiveRec (
            int pi,
            unsigned gpio,
            unsigned level,
            uint32_t tick )
```

DCF77 receive recorder.

This is a pigpiod callback function for an AM DCF77 receiver. It does no filtering nor decoding. It just fills dfc77actRecPer for the current modulation period and stores it in dfc77ringBrecPer when the next period begins.

7.60.2.4 dcf77receiveRecRegister()

```
int dcf77receiveRecRegister (
            void  )
```

DCF77 receive recorder registration.

This function registers the call back function dcf77receiveRec with the PiGpioDaemon.

Returns

register or error number; also stored in dcf77callbackID

7.60.2.5 dcf77receiveRecDeregister()

```
int dcf77receiveRecDeregister (
            void  )
```

DCF77 receive function de-registration.

This function de-registers the call back registered under dcf77callbackID.

Returns

0: OK, pigif_callback_not_found: otherwise

7.60.2.6 disc5()

```
durDiscrPointData_t * disc5 (
            durDiscrPointData_t table[],
            uint32_t const value )
```

Discriminating a value.

In a discrimination table / array of length 5 this function returns a pointer to the highest table entry with value >= table[i].v
The entry returned must be treated as const.

Parameters

table	discrimination table of length 5. With other lengths the function will fail. Must not be null.
value	the number to be discriminated

Returns

the lowest table entry with value $<$ table[i].v

7.60.3 Variable Documentation

7.60.3.1 dcf77recCnt

```
unsigned dcf77recCnt
```

Control output GPIO of the DCF77 receiver's control.

This output, if used and connected, controls the amplitude modulation (AM) receiver's control (On/Off) signal. If there is no such output (PINig) The receiver either has no such control input or it is tight to On.

default PINig; see also dcf77recCntInv, setReceiver()

7.60.3.2 dcf77recCntInv

```
unsigned dcf77recCntInv
```

Receiver On control inverted.

The receiver control output (dcf77recCnt), when used, would be set to ON (Hi, 3V) to enable the receiver. And it would be set OFF for a short time to reset a panicing or inactive receiver.
If dcf77recCntInv is true it is the other way round. The common receiver chips control input is low active, hence inverted and dcf77recCntInv should be true. As an open collector stage near the Pi for this output is highly recommended, nevertheless the default value is FALSE.

default FALSE; see also dcf77recCnt, setReceiver()

7.60.3.3 dcf77inp

```
unsigned dcf77inp
```

Input GPIO for the DCF77 receiver's AM signal.

This is the amplitude modulation (AM) level signal; the level is either 100% or 15% (for 100 or 200ms).

default PIN08; see also dcf77invInp, dcf77PUD, dcf77glitch

7.60.3.4 dcf77invInp

```
unsigned dcf77invInp
```

Inverted DCF77 receiver's signal.

In the case of AM (amplitude) modulation a 1 (high) at the signal input dcf77inp means 15% amplitude and a 0 (low) 100% respectively full amplitude, when dcf77invInp is OFF. ON, obviously, means other way round.

default: OFF; see also dcf77inp

7.60.3.5 dcf77PUD

```
unsigned dcf77PUD
```

DCF77 input's pull resistor setting.

default: PI_PUD_KEEP; see also dcf77inp

7.60.3.6 dcf77glitch

```
unsigned dcf77glitch
```

DCF77 input's glitch filter time setting.

The value for pigpiod's input filter time in μs. The allowed range is 0 ... 30000. The filtering only works for pins sampled by a callback function (like dcf77receiveRec()).

default: 0; glitch filter off; see also dcf77inp

7.60.3.7 dcf77lastLevel

```
unsigned dcf77lastLevel
```

Last DCF77 modulation level.

ON means 15% modulation amplitude; i.e. the signal.
OFF means 100% amplitude; i.e. just the 77,5 kHz carrier.

This variable is set by the receiver (callback) function and must not be modified by user software.

7.60.3.8 lastSysClk

```
char lastSysClk[14]
```

The system time for low AM as text hh:mm:ss.

This textual time stamp is taken at the start of the AM (15%) tick respectively when the callback function is executed for this event.

7.60.3.9 timDiscP

```
durDiscrPointData_t timDiscP[5]
```

Discrimination values for the 15% modulation time.

This is an array of fixed length 5 to discriminate dcf77recPerData_t::tim values.
The only good outcomes of discrimination are indices (durDiscrPointData_t::i) 1 and 3 meaning a recognisable bit 0 respectively 1.
Hint: index bit 0 set means no error.
Hint2: Name ending with P means designed for low grade AM receiver modules. The criteria values are extended quite far for guessing the meaning in the presence of timing faults. Spikes would have to be filtered in a next stage by combining two to four faulty periods in one. Low grade receivers would be not usable without such (complex) filter algorithms.

7.60.3.10 perDiscP

```
durDiscrPointData_t perDiscP[5]
```

Discrimination values for the modulation period.

This is an array of fixed length 5 to discriminate dcf77recPerData_t::per values.
The only good outcomes of discrimination are indices (durDiscrPointData_t::i) 1 and 3 meaning an acceptable 1s respectively 2s period.
Hint: See hints at timDiscP.

7.60.3.11 timDiscH

```
durDiscrPointData_t timDiscH[5]
```

Discrimination values for the 15% modulation time.

This is an array of fixed length 5 to discriminate dcf77recPerData_t::tim values.
The good outcomes have indices (durDiscrPointData_t::i) 1 and 3 meaning a recognisable bit 0 (FALSE) respectively 1 (TRUE).
Hint: index bit 0 set means no error.

Hint2: Name ending with H means designed for high grade AM receiver modules with virtually no timing faults or spikes. Hence, the criteria values are relatively tight, to recognise EMI or short outages as such. Trying to interpret those with filter algorithms may not be worth the effort with good receivers.

Hint3: Some lower grade AM receiver modules were enhanced with an inverting NPN transistor stage with a low capacity collector to ground capacitor implemented by three meter shielded signal and supply cable. This adding of a simple inverter stage is recommended for all receiver modules not equipped with an open collector (OC) output stage. And for some of them its a necessary filter stage.

7.60.3.12 timDiscHs

```
durDiscrPointData_t timDiscHs[5]
```

Discrimination values for the 15% modulation time.

This is an array of fixed length 5 to discriminate dcf77recPerData_t::tim values.
It is the same as timDiscH except for the extra note/hint 4.

Hint 4: The price for turning bad to good receivers by the circuit of Hint 3 was good pulses shortened well below 100 respectively 200 ms. Therefore this table allows for such shortened pulses noting it by the name's suffix s.

7.60.3.13 perDiscH

```
durDiscrPointData_t perDiscH[5]
```

Discrimination values for the modulation period.

This is an array of fixed length 5 to discriminate dcf77recPerData_t::per values.
The only good outcomes of discrimination are indices (durDiscrPointData_t::i) 1 and 3 meaning an acceptable 1s respectively 2s period.
Hint: See hints at timDiscHs.

7.60.3.14 perDiscUsed

```
durDiscrPointData_t* perDiscUsed
```

Discrimination values for the modulation period used.

This is a pointer to an array of fixed length 5 to discriminate dcf77recPerData_t::per values. The purpose is to hold the current filter values.
default: perDiscP

7.60.3.15 timDiscUsed

```
durDiscrPointData_t* timDiscUsed
```

Discrimination values for the 15% modulation time.

This is a pointer to an array of fixed length 5 to discriminate dcf77recPerData_t::per values. The purpose is to hold the current filter values.
default: timDiscP

7.60.3.16 curBCDnum

```
unsigned curBCDnum
```

Number decoded from period sequence parts.

This variable holds the (current) number evaluated by considering the sequence of disc5(timDiscUsed, modulation↩ Time) in question. Here, with number sequences a false (level 1, 'F') means 0 and a true (level 3, 'T') means 1, 2, 4, 8, 10, 20, 40 or 80 depending on the place within the BCD coded number. Any other level (0 spike, 3 undef, 4 error) invalidates the whole number.
Values 0..99 (max year) are OK, a value above means an error in the sequence.

See also

disc5 timDiscUsed timDiscHs

7.60.3.17 numBCDinit

```
unsigned const numBCDinit[]
```

Initialisation or least significant BDC digit for modulation time.

For the durDiscrPointData_t i value as index this constant array yields 0 or 1 — or respectively 2, 4 , 8, 10, 20, 40, 60 and 80 — for valid times and a high error value else.
The rationale is just adding up yields a correct value in the BCD range of 0 .. 99 while any higher value means error.

See also

timDiscUsed timDiscHs disc5() curBCDnum

7.61 weRasp/weEcarLd.c File Reference

Some functions for E-Car loading on a Raspberry Pi using pigpiod.

```
#include "weEcarLd.h"
#include "sweetHome.h"
```

Macros

- #define **CHS_CP_OFF_LEV**

  loading off CP state (1: +12V, 0: -12V)

Functions

- int chsItoDutyC (float const ldCurrLim)

  Calculate CP dutycycle for current limit per phase.
- void initAsCPilot (int const thePi, unsigned const gpio, uint8_t init)

  Initialise a GPIO pin as pilot drive.
- void setChStControl (float const ldCurrLim)

  Control E-Car charging current limit per phase by PWM.
- float setChStLimit (float ldCurrLim)

  Set and get E-Car charging current limit per phase.
- uint8_t setChStNoPhases (uint8_t const assumedNofPhases)

  Set and get charging E-Car's actual number of phases.
- void setCPilot (uint8_t set)

  Set CP signal to constant ON or OFF.
- void setCPilotPWM (int duty)

  Set CP signal PWM.

7.61.1 Detailed Description

Some functions for E-Car loading on a Raspberry Pi using pigpiod.

```
Copyright  (c)   2023     Albrecht Weinert
weinert-automation.de     a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 253 7.10.2023
Rev. 253 07.18.2023 : new
```

cross-compile by:

```
arm-linux-gnueabihf-gcc -DF_CPU=1500000000 -DPLATFORM=raspberry_04
      -DMCU=BCM2711 -I./include    -c -o weRasp/weGPIOd.o weRasp/weEcarLd.c
```

This is a supplementary library basic library to be used in conjunction with the pigpio library (link: -lpigpiod_if2 -lrt). For documentation see the include file weEcarLd.h and also http://abyz.me.↩uk/rpi/pigpio/pdif2.html by Joan NN.

7.61.2 Function Documentation

7.61.2.1 initAsCPilot()

```
void initAsCPilot (
            int thePi,
            unsigned gpio,
            uint8_t init )
```

Initialise a GPIO pin as pilot drive.

This function initialises the GPIO pin like initAsHiDrive() and (additionally) sets the PWM frequency to 1kHz and the duty cyle range to 0..1000. The parameter init when 0 (FALSE) sets the CP output to -12V signalling EVSE being not ready. init when 1 (TRUE) sets the CP output to +12V signalling EVSE ready.

This function must be called before all other CP control functions, as it sets the CP's GPIO henceforth.

A value outside this range will turn the 1kHz CP signal off; see CHS_CP_OFF_LEV.

Parameters

the↩Pi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..53)
init	0: off (-12V) 1: on (+12V)

7.61.2.2 setCPilot()

```
void setCPilot (
            uint8_t set )
```

Set CP signal to constant ON or OFF.

This function sets the CP either constant ON (+12V) or OFF (-12V). It will stop any 1kHz square wave.

Parameters

set	0: off (-12V) 1: on (+12V)

7.61.2.3 setCPilotPWM()

```
void setCPilotPWM (
            int duty )
```

Set CP signal PWM.

This function sets the CP duty cycle in the range 50..970‰ applying offset (CHS_CP_OFFSET) and inversion (CHS_CP_INVERTED) if applicable.

Note: 50‰ (5%) signals the of a digital protocol (not implemented here). The range for signalling current limit per phase is 10..970‰; see chsItoDutyC(float).

Parameters

duty	dutycycle in ‰ (parts per mille) 50..970‰

7.61.2.4 setChStLimit()

```
float setChStLimit (
            float ldCurrLim )
```

Set and get E-Car charging current limit per phase.

This function sets the charging stations current limit in the range 6.0 ... CHS_CURRENT_LIM_MAX.

Note: This function just sets/changes the internal limit value. It will not influence any signal (CP) to the EVSE.

Parameters

ldCurrLim	load current in A (per phase)

Returns

the (new) current limit

7.61.2.5 chsItoDutyC()

```
int chsItoDutyC (
            float ldCurrLim )
```

Calculate CP dutycycle for current limit per phase.

The IEC CP dutycycle(current) curve will be applied and the outcome limited to 100..970‰ resp. 6..80A per phase.

This function is a helper for setChStControl(float) and usually not used directly.

Parameters

ldCurrLim	the current limit

Returns

dutycycle in ‰ (per mille)

7.61.2.6 setChStControl()

```
void setChStControl (
            float ldCurrLim )
```

Control E-Car charging current limit per phase by PWM.

This function sets the the control pilot (CP) PWM signal to the charging station.

This function should only be called with a current value within actual limits (6...32A, e.g.). It only limits the CP duty cycle to the range 10..97% corresponding to 6..80A.

Parameters

ldCurrLim	current limit in A per phase

7.61.2.7 setChStNoPhases()

```
uint8_t setChStNoPhases (
            uint8_t assumedNofPhases )
```

Set and get charging E-Car's actual number of phases.

This function sets the (assumed) number of phases for E-Car AC charging in the range 1..CHS_MAX_PHASES. The default is CHS_BEG_PHASES.

The value set will be used to calculate and eventually limit the maximum loading power. The real number of phases depends on the car(s) connected.

Warning: There exist E-car types using only one phase even when offered three!

Parameters

assumedNofPhases	number of phases 1.. CHS_MAX_PHASES

Returns

the (new) number of phases

7.62 weRasp/weGPIOd.c File Reference

Some IO functions for Raspberry Pi using pigpiod.

```
#include "weGPIOd.h"
```

Functions

- uint8_t gpio4pin (int const pin)
  Pin number to GPIO number lookup.
- void initAsDrive (int thePi, unsigned gpio, unsigned init)
  Initialise a GPIO pin as output.
- void initAsHiDrive (int const thePi, unsigned gpio, unsigned init)
  Initialise a GPIO pin as high drive.
- void initAsHiInput (int thePi, unsigned gpio)
  Initialise a GPIO pin as input with pull down.
- void initAsInput (int thePi, unsigned gpio)
  Initialise a GPIO pin as input.
- void initAsInputs (unsigned const lesGPIOs[])
  Initialise one or more GPIO pin as input.
- void initAsLoInput (int thePi, unsigned gpio)
  Initialise a GPIO pin as input with pull up.
- uint32_t initAsOutput (int thePi, unsigned gpio)
  Initialise a GPIO pin as output.
- uint32_t initAsOutputs (unsigned const lesGPIOs[])
  Make one or more GPIO pins output.
- uint8_t pin4GPIO (int const gpio)
  GPIO number to pin number lookup.
- uint32_t releaseOutputs (int const thePi)
  Release all GPIO pins set as output.
- uint32_t releaseOutputsReport (int const thePi)

Release all GPIO pins set as output with report.

- void reportPinOp (char const *op, unsigned const lesGpio[])

  Report an arbitrary operation on a list of GPIOs.
- void setOutput (unsigned const gpio, unsigned const level)

  Set a GPIO output pin.
- void setOutputs (uint32_t const lesOuts, unsigned const level)

  Set a list/mask of GPIO output pins.
- void setPadStrength (int thePi, unsigned mA)

  Set the output drive capacity of GPIO ports 0..27.

Variables

- uint32_t const gpio2bit [36]

  GPIO number to bank pin number lookup.
- uint8_t const gpio2pin [44]

  GPIO number to pin number lookup.
- uint8_t const pin2gpio [44]

  Pin number to GPIO number lookup.
- char const **pudTxt** [5][6]

  Names for input's pull resistor settings.
- int thePi

  The standard Pi for gpio(d) IO of the program.

7.62.1 Detailed Description

Some IO functions for Raspberry Pi using pigpiod.

```
Copyright (c)   2020     Albrecht Weinert
weinert-automation.de     a-weinert.de
    /          /       /\
   /          /___    /  \       |
   \         /____\ /____\ |  _|__
    \  /\  / \     /       \|   |
     \/  \/   \__/          \__/|_
```

Revision history

```
Rev. 253 9.10.2023
Rev. 227 12.08.2020 : common functions collected and consolidated
Rev. 233 12.10.2020 : initAsHiInput() added
Rev. 237 01.03.2021 : documentation (Doxygen) corrected/improved
```

cross-compile by:

```
arm-linux-gnueabihf-gcc -DF_CPU=1500000000 -DPLATFORM=raspberry_04
      -DMCU=BCM2711 -I./include     -c -o weRasp/weGPIOd.o weRasp/weGPIOd.c
```

This is a supplementary library basic library to be used in conjunction with the pigpio library (link: -lpigpiod_if2 -lrt). For documentation see the include file weGPIOd.h and also http://abyz.me.↩uk/rpi/pigpio/pdif2.html by Joan NN.

7.62.2 Function Documentation

7.62.2.1 pin4GPIO()

```
uint8_t pin4GPIO (
            int const gpio )
```

GPIO number to pin number lookup.

Parameters

gpio	a GPIO number (>= 0) available on the Pi's 40 respectively 26 pins IO connector

Returns

the GPIO's pin number (1..40 resp. 26) on the IO connector;
0 means not available in the Pi IO connector or even undefined

7.62.2.2 gpio4pin()

```
uint8_t gpio4pin (
            int const pin )
```

Pin number to GPIO number lookup.

Parameters

pin	1..40 (26) is the legal IO connector pin number

Returns

0..56 the GPIO number; 90: ground (0V);
93: 3.3V; 95: 5V; 99: undefined, i.e. illegal pin number

7.62.2.3 initAsInputs()

```
void initAsInputs (
            unsigned const lesGPIOs[] )
```

Initialise one or more GPIO pin as input.

This functions sets the pins listed as GPIOs to input mode. This is also used to release them from any output modes as input means hi impedandance.

The Pi for the IO operation is thePi.

Parameters

lesGPIOs	array of GPIO numbers (0..53); use 0x7F as end marker

7.62.2.4 initAsOutputs()

```
uint32_t initAsOutputs (
            unsigned const lesGPIOs[] )
```

Make one or more GPIO pins output.

This functions sets the pins listed as GPIOs to output mode.
Normally, at program end, the same list of outputs should be released to high impedance by initAsInputs().

The Pi for the IO operations is thePi.

Parameters

lesGPIOs	array of GPIO numbers (0..53); use 0x7F as end marker

Returns

the bank mask of the outputs set so by this function; 0 means non set (complete failure)

7.62.2.5 setOutputs()

```
void setOutputs (
            uint32_t const lesOuts,
            unsigned const level )
```

Set a list/mask of GPIO output pins.

This functions sets the (output) pins set in the bank mask ON or OFF.

The Pi for the IO operations is thePi.

Parameters

lesOuts	bank mask of outputs to be set
level	OFF or ON (0 or 1)

7.62.2.6 setOutput()

```
void setOutput (
            unsigned const gpio,
            unsigned const level )
```

Set a GPIO output pin.

This functions sets an output pin gpio ON or OFF.
If gpio is $>$ 31 nothing is done. That handles the meaning of gpio 33 (within a bank) as unused.

The Pi for the IO operations is thePi.

Parameters

gpio	an output pin (that should have been set as such!)
level	OFF or ON (0 or 1)

7.62.2.7 reportPinOp()

```
void reportPinOp (
            char const * op,
            unsigned const lesGpio[] )
```

Report an arbitrary operation on a list of GPIOs.

The report lines on outLog will be
```
progNam   ___operation GPIO: 13 pin: 27
```

Parameters

op	the operation displayed as 12 characters right justified.
lesGpio	a GPIO list (terminated by a value $>$ 56)

7.62.2.8 initAsOutput()

```
uint32_t initAsOutput (
            int thePi,
            unsigned gpio )
```

Initialise a GPIO pin as output.

This sets a GPIO as output and puts it in the list of GPIOs used as outputs by the program if in the range of 0..31 (resp. 2..27).

All functions setting setting as output should use this function.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..31); PINig means no action

Returns

the bank mask of this output if set (0 no action or error)

7.62.2.9 releaseOutputs()

```
uint32_t releaseOutputs (
            int const thePi )
```

Release all GPIO pins set as output.

This releases all GPIO in the range 0..27 that this program has set as outputs (by setting those as inputs).
The order of this operation is by increasing GPIO number. If another order or extra actions are required this must be done before or afterwards.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)

Returns

the bank mask of the previous resp. released outputs

7.62.2.10 releaseOutputsReport()

```
uint32_t releaseOutputsReport (
            int const thePi )
```

Release all GPIO pins set as output with report.

Same as releaseOutputs() plus a "releaseToIn" for every output released.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)

Returns

the bank mask of the previous resp. released outputs

7.62.2.11 initAsInput()

```
void initAsInput (
            int thePi,
            unsigned gpio )
```

Initialise a GPIO pin as input.

This sets a GPIO as input and removes it form the list of GPIOs set as output if in the range of 0..31 (resp. 2..27).

All functions setting as input should use this function.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..53)

7.62.2.12 initAsLoInput()

```
void initAsLoInput (
            int thePi,
            unsigned gpio )
```

Initialise a GPIO pin as input with pull up.

This initialisation is for an input sensing a switch (button) or transistor (optocoupler) connected to ground (gnd, 0V). This is the normal configuration instead of switching to Hi (3.3V).
In most of the cases the Pi's internal pull up resistor (about 50 kOhm) is sufficient for Lo-switches and should then be used.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..53)

7.62.2.13 initAsHiInput()

```
void initAsHiInput (
            int thePi,
            unsigned gpio )
```

Initialise a GPIO pin as input with pull down.

This initialisation is for an input sensing an electronic device delivering a voltage about 3 V when active respectively ON. Some of those devices require a pull down to deliver clean signals.
The Pi's internal pull down resistor (about 50 kOhm) may be sufficient for Hi-switches and should then be used.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..53)

7.62.2.14 initAsHiDrive()

```
void initAsHiDrive (
            int thePi,
            unsigned gpio,
            unsigned init )
```

Initialise a GPIO pin as high drive.

Of course, Raspberry's (BCM2837's) GPIO pins are high and low drivers as output. Hi-drive is provided by turning on pull-up as to allow broken wire diagnosis when shortly switching to input.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..53)
init	0 or 1: the initial output value; else: leave unchanged

7.62.2.15 initAsDrive()

```
void initAsDrive (
            int thePi,
            unsigned gpio,
            unsigned init )
```

Initialise a GPIO pin as output.

This function sets the GPIO pi as output, optionally sets the drive capacity and leaves a pull resistor setting unchanged.

Parameters

thePi	the Raspberry's identifier as got from initialising gpio(d)
gpio	the GPIO number (0..53)
init	0 or 1: the initial output value; else: leave unchanged

7.62.2.16 setPadStrength()

```
void setPadStrength (
            int thePi,
            unsigned mA )
```

Set the output drive capacity of GPIO ports 0..27.

For pins set as output (by initAsHiDrive() or initAsDrive() e.g.) this function sets the drive capacity in the range of 2..16 mA.
Note 1: All 27 pins get the same common value. Hence, one has to set the maximum needed for any pin.
Note 2: This value is no current limit nor pin overload protection. It is the maximum load current, which a valid 0 or 1 output voltage can be guaranteed under. Note 3: The BCM processor can set the strength in 2mA steps (2, 4.. 14, 16). Nevertheless, this function accepts all values 2..16 incrementing odd values.

Parameters

the↩ Pi	the Raspberry's identifier as got from initialising gpio(d)
mA	2..16: output drive capacity for providing legal low or high; else: leave drive capacity unchanged

7.62.3 Variable Documentation

7.62.3.1 gpio2pin

```
uint8_t const gpio2pin[44]
```

GPIO number to pin number lookup.

Index [0..39] is a GPIO number available on the Pi's 40 (26) pins connector. Result 1..40: The GPIO's pin number on the IO connector;
0: not available in the Pi IO connector or even undefined
Outside [0..39] index out of bound.
See also gpio2bit, pin2gpio

7.62.3.2 gpio2bit

```
uint32_t const gpio2bit[36]
```

GPIO number to bank pin number lookup.

Index [0..31] is a GPIO number partly (2..27) available on the Pi's 40 (26) pins connector. Result 0x00000001..0x80000000: a 32 bit with exactly one bit set corresponding to place in a 32 bit bank mask.
Outside [0..31] index out of bound.
See also gpio2pin, pin2gpio

7.62.3.3 pin2gpio

```
uint8_t const pin2gpio[44]
```

Pin number to GPIO number lookup.

Index [1..40 (26)] is the legal pin number.
Result 0..56: GPIO number; 90: ground (0V); 93: 3.3V; 95: 5V;
99: undefined (for pin 0 and 41 (27)..44).
Outside [0..43] index out of bound.
See also gpio2bit, gpio2pin

7.62.3.4 thePi

```
int thePi
```

The standard Pi for gpio(d) IO of the program.

This global variable is provided to hold the main pi used by a program doing process IO via piGpIO[d]. In most local use cases
```
thePi = pigpio_start(NULL, NULL);
```

it will be 0 = this local Pi. After usage don't forget to terminate the connection to the pigpio daemon by
```
pigpio_stop(thePi);
```

7.63 weRasp/weGPIOkeyM.c File Reference

Some keymatrix IO functions for Raspberry Pi.

```
#include "weGPIOkeyM.h"
```

Functions

- void colRowInp (keyMatrix thePad)

  Prepare all column and row GPIO pins of a key matrix.
- void crScanStep (keyMatrix *thePad, uint32_t cnt1ms)

  Column and row scan step.
- void keyMatrixInit (keyMatrix *thePad, int thePi)

  Set the keyMatrix fields not initialised at definition.
- void keyPadInit (keyMatrix *thePad, int thePi)

  Initialise the key pad.
- void printMatrixConfig (FILE *stream, keyMatrix thePad)

  Print the initialised keyMatrix structure.
- void printMatrixPressState (FILE *stream, keyMatrix thePad, uint8_t cmp)

  Print the keyMatrix' key pressed state.
- void setKeyMatrixScanFunct (keyMatrix *thePad, pt2lastScanStep fun)

  Set the key pad's last scan step function.

7.63.1 Detailed Description

Some keymatrix IO functions for Raspberry Pi.
```
Copyright (c) 2018 Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \      |
   \        /____\ /____\ |  _|__
    \  /\  / \    /      \|   |
     \/  \/   \__/        \__/|_
```

Revision history
```
Rev. 237 8.02.2021
Rev. 176 29.07.2017 : new (excerpt from weGPIOd.c)
Rev. 209 22.07.2019 : work around a Doxygen bug
```

This is a supplementary library basic library to be used in conjunction with the pigpio library (link: -lpigpiod_if2 -lrt). For documentation see the include files weGPIOd.h, weGPIOkeyM.h and also http://abyz.me.↩uk/rpi/pigpio/pdif2.html by Joan NN.

7.63.2 Function Documentation

7.63.2.1 colRowInp()

```
void colRowInp (
            keyMatrix thePad )
```

Prepare all column and row GPIO pins of a key matrix.

All GPIO pins to be used for scanning a key (switch) matrix defined in the structure thePad are all initialised
a) as input,
b) with pull-down resistor and
c) 2mA drive capacity (when switched to output for scan step).

Parameters

thePad	the keyMatrix structure defining the key pad

7.63.2.2 keyMatrixInit()

```
void keyMatrixInit (
            keyMatrix * thePad,
            int thePi )
```

Set the keyMatrix fields not initialised at definition.

On definition of the keyMatrix structure thePad, its fields
.noCols, .noRows,
.colRow and .keyVal
have to be initialised according to the key matrix' hardware configuration. Additionally
.pt2lastScanStep
may be initialised pointing to an appropriate user supplied function. (It will be NULL otherwise.)

This procedure sets all other fields accordingly. It must be called once before any other operation.

Parameters

thePad	address of the structure defining the key matrix (not NULL)
thePi	the Raspberry's identifier as got from initialising gpio(d)

7.63.2.3 setKeyMatrixScanFunct()

```
void setKeyMatrixScanFunct (
            keyMatrix * thePad,
            pt2lastScanStep fun )
```

Set the key pad's last scan step function.

The supplied function with signature
  void fooSetKeys(keyMatrix * thePad, uint8_t changes)
or NULL is set in the structure thePad. If not NULL the function
  crScanStep(keyMatrix * thePad)
will call the function supplied after one round of scan steps in the last step.

Parameters

thePad	address of the structure defining the key matrix (not NULL)
fun	the function to be call on last step

7.63.2.4 keyPadInit()

```
void keyPadInit (
            keyMatrix * thePad,
            int thePi )
```

Initialise the key pad.

Initialises the structure the pad and the IO; i.e. keyMatrixInit() and then colRowInp(). Has to be called only once before any further (IO) operation.

Parameters

thePad	address of the structure defining the key matrix
thePi	the Raspberry's identifier as got from initialising gpio(d)

7.63.2.5 crScanStep()

```
void crScanStep (
            keyMatrix * thePad,
            uint32_t cnt1ms )
```

Column and row scan step.

Actions for a single scan step.
The steps are
0 ... .noRoCo-1 : process IO
And step .noRoCo: calculation and action
A call to this function should be put in an appropriate cycle, like 1ms.

Parameters

thePad	address of the structure defining the key matrix
cnt1ms	millisecond time stamp

7.63.2.6 printMatrixConfig()

```
void printMatrixConfig (
            FILE * stream,
            keyMatrix thePad )
```

Print the initialised keyMatrix structure.

Outputs a nice matrix view on the GPIO pins and masks as well as the labels of the key matrix. (For non-proportional font only.) Should not be used before keyPadInit().

Parameters

stream	the output; NULL defaults to outLOg (stdout)
thePad	address of the structure defining the key matrix

7.63.2.7 printMatrixPressState()

```
void printMatrixPressState (
            FILE * stream,
            keyMatrix thePad,
            uint8_t cmp )
```

Print the keyMatrix' key pressed state.

Outputs a nice matrix view on matrix' buttons and their pressed state (i.e. scan counts of the button in question). Should only be used within the user supplied last scan step function; and best only on count changes.

Parameters

stream	the output; NULL defaults to outLog (stdout)
thePad	address of the structure defining the key matrix
cmp	0: no change; bit 0: count change; bit 1 state change

7.64 weRasp/weLockWatch.c File Reference

Process control helpers for Raspberry Pi: lock and watchdog.

```
#include "weLockWatch.h"
```

Functions

- void closeLock (void)

  Unlock the lock file

- int initWatchdog (void)

  Get and initialise or arm the watchdog.
- int justInitWatchdog (void)

  Get and initialise or arm the watchdog.
- int justLock (char const *lckPiGpioFil)

  Open and lock the lock file.
- int openLock (char const *lckPiGpioFil, uint8_t const perr)

  Open and lock the lock file.
- void stopWatchdog (void)

  Stop and disarm the watchdog.
- void triggerWatchdog (void)

  Trigger the watchdog.

Variables

- int lockFd

  Lock file handle.
- int retCode

  Basic start-up function failure.
- int useIOlock

  flag to use IO (singleton) lock; default on
- int useWatchdog

  flag to use watchdog; default on

7.64.1 Detailed Description

Process control helpers for Raspberry Pi: lock and watchdog.

```
Copyright  (c)   2020     Albrecht Weinert
weinert-automation.de     a-weinert.de
    /         /      /\
   /         /___    /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 236 2.02.2021
Rev. 227 12.08.2020 : common functions collected and consolidated
```

cross-compile by:

```
arm-linux-gnueabihf-gcc -DF_CPU=1500000000 -DPLATFORM=raspberry_04
       -DMCU=BCM2711 -DTARGET=pi4you  -I./include
             -c -o weRasp/weLockWatch.o weRasp/weLockWatch.c
```

This is a supplementary library basic library to be used in conjunction with the pigpio library (link: -lpigpiod_if2 -lrt). For documentation see the include file weLOckWatch.h .

7.64.2 Function Documentation

7.64.2.1 justLock()

```
int justLock (
            char const * lckPiGpioFil )
```

Open and lock the lock file.

This function is the basic implementation of openLock. Applications not wanting its optional logging or doing their own should use this function directly.

Parameters

lckPiGpioFil	lock file name

Returns

0: OK, locked; 97: fd does not exist; 98: can't be locked

7.64.2.2 openLock()

```
int openLock (
            char const * lckPiGpioFil,
            uint8_t const perr )
```

Open and lock the lock file.

This function may use logging and streams not available on smaller applications. Those applications not wanting that optional logging or doing their own should use the function justLock().

Parameters

lckPiGpioFil	lock file path name
perr	make error message when lock file does not exist or can't be locked

Returns

0: OK, locked; 97: fd does not exist; 98: can't be locked

7.64.2.3 closeLock()

```
void closeLock (
            void  )
```

Unlock the lock file

Unlock the lock file

7.64.2.4 justInitWatchdog()

```
int justInitWatchdog (
            void  )
```

Get and initialise or arm the watchdog.

This function justInitWatchdog() is the basic implementation of initWatchdog() without logging failure.

Returns

0: watchdog OK or not to be used (useWatchdog OFF); 1: error while trying to get the watchdog

7.64.2.5 initWatchdog()

```
int initWatchdog (
            void  )
```

Get and initialise or arm the watchdog.

If the watchdog is to be used, i.e. useWatchdog is ON. this function tries to get it. Otherwise it does nothing but return 0 (success).

On no success 1 is returned and useWatchdog is set OFF and the misfortune is logged. Use justInitWatchdog() :: for silence.

Returns

0: watchdog OK or not to be used (useWatchdog OFF); 1: error while trying to get the watchdog

7.64.2.6 stopWatchdog()

```
void stopWatchdog (
            void  )
```

Stop and disarm the watchdog.

If the watchdog is to be used, i.e. useWatchdog is ON, this function stops and disarms it.
useWatchdog is then OFF .

7.64.2.7 triggerWatchdog()

```
void triggerWatchdog (
            void  )
```

Trigger the watchdog.

If the watchdog is to be used, i.e. useWatchdog is ON, this function triggers it.
If the watchdog is armed (by initWatchdog()) not triggering it at least once about every 15 s will lead to system reset.

7.64.3 Variable Documentation

7.64.3.1 useIOlock

```
int useIOlock
```

flag to use IO (singleton) lock; default on

Do use IO lock.

7.64.3.2 lockFd

```
int lockFd
```

Lock file handle.

Do not use directly.

7.64.3.3 retCode

```
int retCode
```

Basic start-up function failure.

Allows for compact code without saving the (error) return:
if (openLock(lckPiGpioPth, ON)) return retCode;

Storage for return/error codes. Used by: openLock(char const *, uint8_t) theCyclistStart(int) theCyclistWaitEnd()

Value: 0: OK, else: error

7.64.3.4 useWatchdog

```
int useWatchdog
```

flag to use watchdog; default on

flag to use watchdog; default ON

7.65 weRasp/weModbus.c File Reference

Modbus functions for Raspberry Pis.

```
#include "weModbus.h"
```

Functions

- void modRSclose (modRS_t *modRS)

  Close a Modbus RS link and destroy the (libmodbus) structure.
- int modRSconnect (modRS_t *modRS, int currentSlave)

  Connect a the Modbus (libmodbus) structure for RS485 (RTU).
- int modRSctxNew (modRS_t *modRS, int currentSlave)

  Make a new Modbus (libmodbus) structure for RS485.
- int modRSswitchSlave (modRS_t *modRS, int currentSlave)

  Switch the slave on a connected Modbus (libmodbus) structure for RS485.
- void modTCPclose (modTCP_t *modTCP)

  Close a Modbus TCP and destroy the (libmodbus) structure.
- int modTCPconnect (modTCP_t *modTCP)

  Connect a the Modbus (libmodbus) structure for TCP.
- int modTCPctxNew (modTCP_t *modTCP)

  Make a new Modbus (libmodbus) structure for TCP.
- int modTCPlisten (modTCP_t *modTCP)

  Listen at the Modbus (libmodbus) structure for TCP.
- int parseModPort (const char *str)

  Parse a string as Modbus TCP port number with checks.
- void reg2val32 (dualReg_t *const dest, dualReg_t const *const source)

  Copy float (32 bit) Modbus input to correctly ordered value.
- uint16_t regs2string (char *dest, uint16_t *source, int n)

  Copy a sequence of character pairs from registers to a string.
- void regs2vals32 (float *const dest, dualReg_t const *const source, int const n)

  Copy n float (32 bit) Modbus input to correctly ordered values.
- int setIP4add (char *dest, const char *src)

  Set an IPv4 address as string with syntax checks.

7.65.1 Detailed Description

Modbus functions for Raspberry Pis.

```
Copyright  (c)  2019   Albrecht Weinert
weinert-automation.de      a-weinert.de
```

Revision history

```
Rev. 236 2.02.2021
Rev.  51  18.10.2017 A.W.: more structures and functions, more comfort
Rev. 188  15.10.2018 : minor typos
Rev. 212  04.08.2019 : regs2string added
Rev. 216  31.08.2019 : mod_bus close added
```

cross-compile by:

```
arm-linux-gnueabihf-gcc -DMCU=BCM2837 -I./include -c -o weRasp/weModbus.o weRasp/weModbus.c
```

This is a supplementary (basic) library to be used in conjunction with the libmodbus library (link: -lmodbus). For documentation see the include file weModbus.h

7.65.2 Function Documentation

7.65.2.1 reg2val32()

```
void reg2val32 (
            dualReg_t *const dest,
            dualReg_t const *const source )
```

Copy float (32 bit) Modbus input to correctly ordered value.

This function copies a 32bit (e.g.) float value input from Modbus — with correct byte ordering at 16 bit level (!) — to a correctly ordered (32 bit) value.
Destination and source pointers may be the same, but not NULL.
The swap of the 16-bit parts only occurs, when the platform is little endian.
See also: PLATFlittlE and regs2vals32()

Parameters

dest	the dual registers to store the result in
source	the dual registers with the potentially wrong endianess

7.65.2.2 regs2vals32()

```
void regs2vals32 (
            float *const dest,
            dualReg_t const *const source,
            int const n )
```

Copy n float (32 bit) Modbus input to correctly ordered values.

This function copies n 32bit (e.g.) float values input from Modbus — with correct byte ordering at 16 bit level (!) — to correctly ordered (32 bit) values.
Destination and source pointers may be the same, but not NULL.
The swap of the 16-bit parts only occurs, when the platform is little endian.
See also: PLATFlittlE and reg2val32()

Parameters

dest	the array of dual registers to store the result in
source	the array of dual registers with potentially wrong endianess
n	the number of dual registers to treat

7.65.2.3 regs2string()

```
uint16_t regs2string (
            char * dest,
            uint16_t * source,
            int n )
```

Copy a sequence of character pairs from registers to a string.

Growatt inverter holding registers, for example, deliver strings as pairs of ASCII characters in consecutive registers. If the number of characters is odd the last register holds just one character. Some of those strings in registers are 0-terminated and some are not. In the first case the transfer stops at the 0, in the latter case a 0 is appended to the destination string as n+1st character.
The sequence of the two characters in each register is (as usual with Modbus) in wrong order. Hence, memcopy and consorts would fail.

Parameters

dest	the array of characters to store the result in
source	the array registers with wrong character sorting
n	the number of dual registers to treat

Returns

the number of characters transfered to dest including the terminating 0

7.65.2.4 modTCPctxNew()

```
int modTCPctxNew (
            modTCP_t * modTCP )
```

Make a new Modbus (libmodbus) structure for TCP.

This function makes a new modTCP.ctx according to .addr and .port; it does NOT check NOR change .mlStat.

Parameters

modTCP	pointer to modTCP_t structure to be used

Returns

0 : OK; -1: error

7.65.2.5 modTCPconnect()

```
int modTCPconnect (
            modTCP_t * modTCP )
```

Connect a the Modbus (libmodbus) structure for TCP.

This function makes a new modTCP.ctx according to .addr and .port, if not yet made. Then it "connects" it. On success 0 is returned. On failure -1 is returned and modTCP.ctx will be destroyed.

This function does NOT check NOR change .mlStat.

Parameters

modTCP	pointer to modTCP_t structure to be used

Returns

0 : OK; -1: error

7.65.2.6 modTCPlisten()

```
int modTCPlisten (
            modTCP_t * modTCP )
```

Listen at the Modbus (libmodbus) structure for TCP.

This function makes a new modTCP.ctx according to .addr and .port, if not yet made. Then it "listens" on it. On success .s >= 0, that is the socket, is returned. On failure -1 is returned and modTCP.ctx will be destroyed. The number of connections is limited to one, here.

This function does NOT check NOR change .mlStat.

Parameters

modTCP	pointer to modTCP_t structure to be used

Returns

>=0 : OK, i.e the socket andmodTCP.s; -1: error

7.65.2.7 modTCPclose()

```
void modTCPclose (
            modTCP_t * modTCP )
```

Close a Modbus TCP and destroy the (libmodbus) structure.

This function closes the connection (if on) and destroys modTCP.ctx (if existing) .mlStat will be set to ML_OFF.

Parameters

modTCP	pointer to modTCP_t structure to be used

7.65.2.8 setIP4add()

```
int setIP4add (
            char * dest,
            const char * src )
```

Set an IPv4 address as string with syntax checks.

This function sets the parameter dest with src, doing syntactic checks to assure a valid IPv4 address being set.

Parameters

dest	the strtring to copy the IP address to
src	the IP string to copy to dest; NULL acts as "0.0.0.0"

Returns

0: syntax error or dest is NULL, dest unchanged; 1: OK

7.65.2.9 parseModPort()

```
int parseModPort (
            const char * str )
```

Parse a string as Modbus TCP port number with checks.

This function parses the string src as Modbus port number, doing validity checks: src must be a decimal number 502 or 1024..65535.

Parameters

str	the string to parse as (decimal) modPort; NULL acts as 502

Returns

0: syntax error, else valid Modbus port number (see above)

7.65.2.10 modRSctxNew()

```
int modRSctxNew (
            modRS_t * modRS,
            int currentSlave )
```

Make a new Modbus (libmodbus) structure for RS485.

This function makes a new modTCP.ctx according to .addr and .port; it does NOT check NOR change .rsState.

Parameters

modRS	pointer to modRS_t structure to be used
currentSlave	1..247 sets the slave number; sets .currentSlave

Returns

0 : OK; else: error: -1: modRS null; -2: slave number: -3: no ctx

7.65.2.11 modRSconnect()

```
int modRSconnect (
            modRS_t * modRS,
            int currentSlave )
```

Connect a the Modbus (libmodbus) structure for RS485 (RTU).

This function makes a new modRS.ctx according to .device and else. Then it "connects" it. On success 0 is returned. On failure -1 is returned and modTCP.ctx will be destroyed.

This function does NOT check NOR change .rsState.

Parameters

modRS	pointer to modRS_t structure to be used
currentSlave	if 1..247; sets the slave number to be used next and changes .currentSlave

7.65.2.12 modRSswitchSlave()

```
int modRSswitchSlave (
            modRS_t * modRS,
            int currentSlave )
```

Switch the slave on a connected Modbus (libmodbus) structure for RS485.

This function changes nothing on a functional and connected modRS.ctx setting than the slave number.

Modbus RS485 (RTU) can handle multiple slaves on the same serial interface. This has to be one at a time in a pure sequential matter: hence this slave switching.
With RS232 the one slave's number once correctly established would stay fixed.

Unfortunately (by a libmodbus deficiency) a communication error of one slave would require a total new connect (modRSconnect()) for all slaves; switching to a "good slave" won't help.

Parameters

modRS	pointer to modRS_t structure to be used
currentSlave	if 1..247; sets .currentSlave

Returns

-1: error wrong parameter or no modRS; 0: no change or no ctx; 1..247: changed currentSlave; OK

7.65.2.13 modRSclose()

```
void modRSclose (
            modRS_t * modRS )
```

Close a Modbus RS link and destroy the (libmodbus) structure.

This function destroys modRS.ctx (if existing). .rsState will not be changed.

Parameters

modRS	pointer to modRS_t structure to be used

7.66 weRasp/weSerial.c File Reference

Functions for Raspberry Pi's serial communication.

```
#include "weSerial.h"
```

Functions

- tcflag_t baudFlag (unsigned int const speed)

  Baud flags by baud rate.
- unsigned int baudRate (tcflag_t baudFlag)

  Baud rate by baud flags.
- void closeUART ()

  Close the UART.
- int openUART ()

  Open the UART with given settings.

Variables

- tcflag_t **baud**

  The UART's baud rate as flag bits.
- struct termios **options**

  The UART's setting structure.
- unsigned int **speed**

  The UART's baud rate as value.
- timespec **startReceive**

  Time used for receive timing.
- int uartFilestream

  The UART as file (stream).
- char * uartPath

  The UART's path name.

7.66.1 Detailed Description

Functions for Raspberry Pi's serial communication.

```
Copyright (c) 2019 Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 234 20.01.2021
Rev. 209 09.07.2019 : new
```

This is a (basic) library for serial communications. As far as Modbus is or can be supported this is independent of Stéphane Raimbault's Modbus library libmodbus; weModbus.c on the other hand is.
For documentation see the include file weSerial.h

7.66.2 Function Documentation

7.66.2.1 baudFlag()

```
tcflag_t baudFlag (
            unsigned int const speed )
```

Baud flags by baud rate.

Parameters

speed	a legal baudrate 300 9600 19200 and so on

Returns

the corresponding flag bits if rate is available; otherwise 0 (error). This should be defaulted to a standard rate like 9600 e.g.

7.66.2.2 baudRate()

```
unsigned int baudRate (
            tcflag_t baudFlag )
```

Baud rate by baud flags.

The speed bits of the parameter baudFlag will be evaluated and the corresponding baud rate will be returned. In vase of no valid speed flag value 0 will be returned. 0 may be considered as error and should be defaulted to 9600.

Parameters

baudFlag	the speed bits of the flags parameter will be evaluated

Returns

the corresponding baud rate 300, 9600, 19200 or other standard rate

7.66.2.3 openUART()

```
int openUART ( )
```

Open the UART with given settings.

Returns

uartFilestream; -1 means open error

7.66.2.4 closeUART()

```
void closeUART ( )
```

Close the UART.

uartFilestream will be set to -1.

7.66.3 Variable Documentation

7.66.3.1 uartFilestream

```
int uartFilestream
```

The UART as file (stream).

It is $>$ 0 ($>$2) when open and -1 when closed.

7.66.3.2 uartPath

```
char* uartPath
```

The UART's path name.

It will be preset with the architecture's standard UART path. See arch/config.h.

7.67 weRasp/weShareMem.c File Reference

Shared memory on Raspberry Pis.

```
#include "weShareMem.h"
```

Macros

- #define **ANZ_SEMAS**
  Standard semaphore set of three (3..9)
- #define **SEMAPHORE_KEY**
  Semaphore unique key "Kfig00".
- #define **SHARED_MEMORY_KEY**
  Shared memory key "Buffer00".
- #define **SHARED_MEMORY_SIZE**
  Shared memory size; default: 256.

Functions

- int deleteSemas ()

  Delete the one semaphore set.
- int deleteSharedMem ()

  Delete and detach the shared memory.
- int **detachSharedMem** ()

  Detach the shared memory.
- int getSemas ()

  Get the one semaphore set.
- int initialiseSemas ()

  Initialise the one semaphore set.
- void * initialiseSharedMem ()

  Initialise shared memory.
- int semaphoreClt (int const semNum, int const op, semCtlPar_t par)

  Control semaphores of the set.
- int semaphoreLock (int const semNum, int const ms)

  Lock one semaphore of the set.
- int semaphoreOperation (int const semNum, int op, int const ms)

  Operation on one semaphore of the set.
- int semaphoreUnlock (int const semNum)

  Unlock one semaphore of the set.

Variables

- const semCtlPar_t **VAL0**

  value 0 for SETVAL
- const semCtlPar_t **VAL1**

  value 1 for SETVAL
- const semCtlPar_t **VAL9**

  value 9 for SETVAL

7.67.1 Detailed Description

Shared memory on Raspberry Pis.

```
Copyright  (c)  2017   Albrecht Weinert
weinert-automation.de      a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /      \|   |
     \/  \/   \__/        \__/|_
```

Revision history

```
Rev. 236  2.02.2021
Rev.  73+ 16.11.2017 : new
Rev.  75  30.11.2017 : signal semaphore initialised to 0
Rev. 188  15.10.2018 : minor typos
```

cross-compile by:

```
arm-linux-gnueabihf-gcc -DMCU=BCM2837 -I./include -c -o weRasp/weShareMem.o weRasp/weShareMem.c
```

For documentation see also the include file weShareMem.h

7.67.2 Function Documentation

7.67.2.1 getSemas()

```
int getSemas ( )
```

Get the one semaphore set.

The one semaphore set, if existing, will be registered and used as is.

Returns

0: OK found existing semaphore set; -1: error (errno set and errorText generated)

7.67.2.2 initialiseSemas()

```
int initialiseSemas ( )
```

Initialise the one semaphore set.

The number of semaphores in the set is ANZ_SEMAS (default three). The one semaphore set, if existing, will be registered and used as is. If this is not possible it will be made and initialised.

Hint: This function has two OK return values!

Returns

1: OK semaphore set made new; 0: OK found existing semaphore set; -1: error (errno set and errorText generated)

7.67.2.3 deleteSemas()

```
int deleteSemas ( )
```

Delete the one semaphore set.

Server operation only.

Returns

0: OK; -1: error (errno set and errorText generated)

7.67.2.4 semaphoreOperation()

```
int semaphoreOperation (
            int const semNum,
            int const op,
            int ms )
```

Operation on one semaphore of the set.

Parameters

semNum	the semaphore number in the set (0..ANZ_SEMAS -1)
op	the semaphore operation
ms	if 2..20000 a timeout in ms

Returns

0: OK; -1: error (errno set and errorText generated)

7.67.2.5 semaphoreLock()

```
int semaphoreLock (
            int const semNum,
            int ms )
```

Lock one semaphore of the set.

Parameters

semNum	the semaphore number in the set (0..ANZ_SEMAS -1)
ms	if 2..20000 a timeout in ms

Returns

0: OK; -1: error (errno set and errorText generated)

7.67.2.6 semaphoreUnlock()

```
int semaphoreUnlock (
            int const semNum )
```

Unlock one semaphore of the set.

Parameters

semNum	the semaphore number in the set (0..ANZ_SEMAS -1)

Returns

0: OK; -1: error (errno set and errorText generated)

7.67.2.7 semaphoreClt()

```
int semaphoreClt (
            int const semNum,
            int const op,
            semCtlPar_t par )
```

Control semaphores of the set.

Parameters

semNum	the semaphore number in the set (0..ANZ_SEMAS -1)
op	the semaphore operation, like e.g. SETVAL
par	op's parameter if any

Returns

0: OK; -1: error (errno set and errorText generated)

7.67.2.8 initialiseSharedMem()

```
void * initialiseSharedMem ( )
```

Initialise shared memory.

Make or get and attach. return pointer to attached shared memory or (void *)-1

7.67.2.9 deleteSharedMem()

```
int deleteSharedMem ( )
```

Delete and detach the shared memory.

Server operation only.

7.68 weRasp/weStateM.c File Reference

States and state machine support.

```
#include "weStateM.h"
#include "weUtil.h"
```

Functions

- uint8_t fiveBandDoEnter (state_t ∗const me, float analogueVal)

  Five band checker turn / force ON.
- uint8_t fiveBandDoLeave (state_t ∗const me, float analogueVal)

  Five band checker turn / force OFF.
- uint8_t fiveBandTick (state_t ∗const me, float controlV)

  Five band checker trigger.
- uint8_t floatHystDoEnter (state_t ∗const me, float analogueVal)

  Float value hysteresis turn / force ON.
- uint8_t floatHystDoLeave (state_t ∗const me, float analogueVal)

  Float value hysteresis turn / force OFF.
- uint8_t floatHystTick (state_t ∗const me, float const controlV)

  Float value hysteresis trigger.
- void genStateText (char ∗stateText, state_t const ∗const me, char const ∗stamp)

  Generate text for state machine status.
- void logStateReason (state_t const ∗const me, char const ∗stamp, char const ∗cause)

  Log status text with cause and info.
- void logStateText (state_t const ∗const me, char const ∗stamp)

  Log status text.
- uint8_t seqContDoEnter (state_t ∗const me, char const ∗startCommand)

  Sequential control entry.
- uint8_t seqContDoLeave (state_t ∗const me, char const ∗const stopCommand)

  Sequential control leave.
- uint8_t seqContTick (state_t ∗const me)

  Sequential control tick or check.
- void setGenStateText (state_t ∗const me, genStateText_t const fun)

  Set the function to generate text for state machine status.
- void startStateText (char ∗stateText, state_t const ∗const me, char const ∗stamp)

  Generate status text standard start.
- uint8_t switchDebDoEnter (state_t ∗const me, uint32_t controlV)

  Switch de-bounce turn / force ON.
- uint8_t switchDebDoLeave (state_t ∗const me, uint32_t controlV)

  Switch de-bounce turn / force OFF.
- uint8_t switchDebTick (state_t ∗const me, uint32_t controlV)

  Switch de-bounce trigger.
- uint8_t switchDebTickAC (state_t ∗const me, uint32_t controlV)

  Switch de-bounce trigger AC.
- uint8_t timerDoEnter (state_t ∗const me, uint32_t secFromNow)

  Timer entry.
- uint8_t timerDoLeave (state_t ∗const me, uint32_t ignored)

  Timer leave, that is stop timer.
- uint8_t timerDoStart (state_t ∗const me, uint32_t secFromNow)

  Timer unconditional entry and set.
- uint8_t timerDoStart4ever (state_t ∗const me)

  Timer unconditional entry and set forever or stop it.
- uint8_t timerDoTrigger (state_t ∗const me, uint32_t secFromNow)

  Timer entry or (pro-longing) re-trigger.
- uint8_t timerEndTrigger (state_t ∗const me, uint32_t const secUTCend)

  Timer entry or (pro-longing) re-trigger to absolute UTC end.
- uint8_t timerTickCheck (state_t ∗const me, uint32_t controlV)

  Timer trigger.

7.68.1 Detailed Description

States and state machine support.

```
Copyright  (c)  2018 2020   Albrecht Weinert
weinert-automation.de            a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \    /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 247 5.05.2023
Rev. 104 05.02.2018 : new
Rev. 108 02.12.2018 : parts moved out, one event file
Rev. 193 25.02.2019 : fifeBandCheck implemented
Rev. 195 01.03.2019 : switch de-bounce debugged
Rev. 198 04.03.2019 : justLogStateChg added
Rev. 200 16.04.2019 : state expanded, logging improved
Rev. 204 15.05.2019 : logStateReason repaired
Rev. 223 23.06.2020 : switchDebTickAC added
```

cross-compile by: arm-linux-gnueabihf-gcc -DMCU=BCM2837 -I./include -c -o weRasp/weStateM.o weRasp/weStateM.c

For documentation see also the include file weShareMem.h

7.68.2 Function Documentation

7.68.2.1 startStateText()

```
void startStateText (
              char * stateText,
              state_t const *const me,
              char const * stamp )
```

Generate status text standard start.

This sets the common standard start of a status text in state text, like:

```
//0123456789x123456789v123456789t123456789q1
 " 2019-04-26 03:08:26.278 # befRiseTimer: "
```

It ends with a blank at [40] after the colon at [39]. State machine type or instance specific text may be added from [40] or [41] up to (recommended) [76] followed by a terminating 0.

Hint: This function sets a 0 at [41] for robustness, i.e. having stateText always as string.

Parameters

stateText	a character array supplied to hold the state text to be generated; minimal length 80.
me	pointer to own state; never null
stamp	(time) stamp to be prepended (max. length 23); default " - "

7.68.2.2 genStateText()

```
void genStateText (
            char * stateText,
            state_t const *const me,
            char const * stamp )
```

Generate text for state machine status.

This is the minimal common standard for all status machines.

Parameters

stateText	a character array supplied to hold the state text to be generated; minimal length 80.
me	pointer to own state; never null!
stamp	(time) stamp to be prepended (max. length 23); default " - "

7.68.2.3 logStateText()

```
void logStateText (
            state_t const *const me,
            char const * stamp )
```

Log status text.

The status as text will be generated in and then be output to outLog. outLog will be flushed.

Parameters

me	pointer to own state; not null
stamp	(time) stamp to be prepended (max. length 23); default " - "

7.68.2.4 logStateReason()

```
void logStateReason (
            state_t const *const me,
            char const * stamp,
            char const * cause )
```

Log status text with cause and info.

The status as text will be generated and then be output to outLog. outLog will be flushed.
The text will be:
stamp # state machine name: cause me->infoTxt
me->infoTxt has to be set/ provided by application software and will be output to a maximum / recommended length of 29. A standard format is one field of 7 and two fields of ten characters separated by spaces.

Parameters

me	pointer to own state; not null
stamp	(time) stamp to be prepended (max. length 23); default " - "
cause	the cause of the state change (max. length 6); default me->controlVS

7.68.2.5 setGenStateText()

```
void setGenStateText (
            state_t *const me,
            genStateText_t const fun )
```

Set the function to generate text for state machine status.

Parameters

me	pointer to own state machine
fun	the new text generator function

7.68.2.6 timerTickCheck()

```
uint8_t timerTickCheck (
            state_t *const me,
            uint32_t controlV )
```

Timer trigger.

Parameters

me	pointer to own state; not null
controlV	current time stamp; mostly cycTaskEventData_t.realSec resp. getAbsS()

Returns

0: state changed, i.e. timer ended; 2: still running; 4: inactive

7.68.2.7 timerDoEnter()

```
uint8_t timerDoEnter (
            state_t *const me,
            uint32_t secFromNow )
```

Timer entry.

This function starts an inactive timer to end secFromNow s. It does nothing on a timer already active resp. running, especially, it does not re-trigger / prolong the timer's time.

This is the function set as (default) state_t.doEnter and not timerDoStart.

Parameters

me	pointer to own state; not null
secFromNow	time to run on from now (s)

Returns

0: OK, timer no or already ON

7.68.2.8 timerDoStart()

```
uint8_t timerDoStart (
            state_t *const me,
            uint32_t secFromNow )
```

Timer unconditional entry and set.

This function starts an inactive timer. The timer's end time will be set to (now + secFromNow s) no matter the timers previous state. This unconditional changing the end time of a timer already running is the difference to timerDoEnter.

This is not the function set as (default) state_t.doEnter; it is timerDoEnter.

Parameters

me	pointer to own state; not null
secFromNow	time to run on from now (s)

Returns

0: now started; 1: was running, probably end time changed

7.68.2.9 timerDoStart4ever()

```
uint8_t timerDoStart4ever (
            state_t *const me )
```

Timer unconditional entry and set forever or stop it.

This function starts an inactive timer. The timer's end time will be set to 2.2.2106 no matter the timers previous state. This date is considered as for ever in our 21st century. This does effectively stop the timer.

This function is intended to start or keep running a timer not to end before its time is to be set (by timerDoStart) to a sensible end time.
One use case is: A timer running forever is in error state.

This function is, of course, faster than timerDoEnter and timerDoStart.

Parameters

me	pointer to own state; not null

Returns

0: now started; 1: was running, probably end time changed

7.68.2.10 timerDoTrigger()

```
uint8_t timerDoTrigger (
            state_t *const me,
            uint32_t secFromNow )
```

Timer entry or (pro-longing) re-trigger.

This function starts an inactive timer to end secFromNow s. IF the timer is active and if (now + secFromNow) is later than the current end, the timer me's runtime will be prolonged accordingly (timer re-trigger).

Parameters

me	pointer to own state; not null
secFromNow	time to run on from now (s)

Returns

0: OK, timer starter or prolonged to new (later) time 1: running timer not prolonged

7.68.2.11 timerEndTrigger()

```
uint8_t timerEndTrigger (
            state_t *const me,
            uint32_t const secUTCend )
```

Timer entry or (pro-longing) re-trigger to absolute UTC end.

If secUTCend is now or in the past or if it is equal the the end of the already active timer, nothing will be done (returns 1).

This function starts an inactive timer to end at secUTCend. If the timer is active already secUTCend will be taken as new end time.

Parameters

me	pointer to own state; not null
secUTCend	end time

Returns

0: OK (started or re-triggered); 1: no state change (on and no prolonging, or secUTCend in past)

7.68.2.12 timerDoLeave()

```
uint8_t timerDoLeave (
            state_t *const me,
            uint32_t ignored )
```

Timer leave, that is stop timer.

Note: This function is usable for other state machine types, too, if appropriate.

Parameters

me	pointer to own state; not null
ignored	as the name says

Returns

0: state changed, 1: state hold

7.68.2.13 switchDebTick()

```
uint8_t switchDebTick (
            state_t *const me,
            uint32_t controlV )
```

Switch de-bounce trigger.

Parameters

me	pointer to own state; not null
controlV	0: input OFF else: input ON

Returns

0: state changed, 1: state hold

See also

newSwitchDeb

7.68.2.14 switchDebTickAC()

```
uint8_t switchDebTickAC (
            state_t *const me,
            uint32_t controlV )
```

Switch de-bounce trigger AC.

This function does essentially the same as switchDebTick, except for On ticks being counted twice. This is meant for half wave rectified AC signals sampled at multiples of their frequency. Obviously, half or 1/2 + 1 of the samples will always be Off. And, of course, the state machine has to be made with a on chain length being by the number of samples per period higher than the normal switch de-bounce filter time, to avoid spiky ON results.

Parameters

me	pointer to own state; not null
controlV	0: input OFF else: input ON

Returns

0: OFF, 1: ON

See also

newSwitchDeb

7.68.2.15 switchDebDoEnter()

```
uint8_t switchDebDoEnter (
            state_t *const me,
            uint32_t controlV )
```

Switch de-bounce turn / force ON.

Parameters

me	pointer to own state; not null
controlV	irrelevant but recorded ob state change

Returns

0: state changed, 1: state hold

7.68.2.16 switchDebDoLeave()

```
uint8_t switchDebDoLeave (
            state_t *const me,
            uint32_t controlV )
```

Switch de-bounce turn / force OFF.

Note: this function is usable for other non timer stati, too;

Parameters

me	pointer to own state; not null
controlV	irrelevant but recorded ob state change

Returns

0: state changed, 1: state hold

7.68.2.17 floatHystTick()

```
uint8_t floatHystTick (
            state_t *const me,
            float controlV )
```

Float value hysteresis trigger.

Parameters

me	pointer to own state; not null
controlV	the sampled analogue value

Returns

0: state change, 1: no change, 0xFF: fault i.e. controlV is NaN (no state change)

7.68.2.18 floatHystDoEnter()

```
uint8_t floatHystDoEnter (
            state_t *const me,
            float analogueVal )
```

Float value hysteresis turn / force ON.

Parameters

me	pointer to own state; not null
analogueVal	irrelevant, but recorded

Returns

0: OK, state now ON; 0xFF: fault (me is NULL e.g.)

7.68.2.19 floatHystDoLeave()

```
uint8_t floatHystDoLeave (
            state_t *const me,
            float analogueVal )
```

Float value hysteresis turn / force OFF.

Note: this function is usable for other non timer stati, too;

Parameters

me	pointer to own state; not null
analogueVal	irrelevant, but recorded; may take an (uint32_t) cast float

Returns

0: state change, 1: no change

7.68.2.20 fiveBandTick()

```
uint8_t fiveBandTick (
            state_t *const me,
            float controlV )
```

Five band checker trigger.

Parameters

me	pointer to own state; not null
controlV	the sampled analogue value

Returns

0: state change, 1: no change

7.68.2.21 fiveBandDoEnter()

```
uint8_t fiveBandDoEnter (
            state_t *const me,
            float analogueVal )
```

Five band checker turn / force ON.

This function puts the checker in state bad Hi (2) no matter the parameter value.

Parameters

me	pointer to own state; not null
analogueVal	irrelevant, but recorded on state change

Returns

0: state change, 1: no change

7.68.2.22 fiveBandDoLeave()

```
uint8_t fiveBandDoLeave (
            state_t *const me,
            float analogueVal )
```

Five band checker turn / force OFF.

This function puts the checker in state OK (0) no matter the parameter value.

Parameters

me	pointer to own state; not null
analogueVal	irrelevant, but recorded on state change

Returns

0: state change, 1: no change

7.68.2.23 seqContDoEnter()

```
uint8_t seqContDoEnter (
            state_t *const me,
            char const * startCommand )
```

Sequential control entry.

This function starts the sequence from OFF to ON.

Parameters

me	pointer to own state; not null
startCommand	the cause of the state change; 6 characters (max.) to be recorded and else irrelevant for his function

Returns

0: OK state now ON or on its way there; 1: already ON

7.68.2.24 seqContDoLeave()

```
uint8_t seqContDoLeave (
            state_t *const me,
            char const * stopCommand )
```

Sequential control leave.

This function starts the sequence from ON to OFF.

Parameters

me	pointer to own state; not null
stopCommand	It is recorded on state changes and else irrelevant for this function; see description on seqContTick

Returns

0: OK state now OFF or on its way there; else: other inhibit condition

7.68.2.25 seqContTick()

```
uint8_t seqContTick (
            state_t *const me )
```

Sequential control tick or check.

If this sequential control (me) is not in a stable OFF or ON state, that is state_t.status is 0 or 1 and state_t.subStatus is 0, this function should be called at regular intervals or on relevant conditions state changes.

The function must keep or advance the state.

It must react on interrupts by seqContDoEnter or seqContDoLeave recognisable by state_t.status == 5 respectively 4. Note: This is just for the case, that individual the call back function state_t.onStateChange does not handle this "interrupts" — what it really should.

This (basic) implementation advances the sub state from 1 to n-1 respectively m-1. Interrupts by seqContDoEnter or seqContDoLeave are handled by changing to sub state 1 of the opposite leg.

When this basic implementation is not sufficient for a concrete SFC, the application may provide an own tick/check function. However, in most cases seemingly complicated cases — nonlinear chains, wait conditions etc. — the specialised behaviour can most often be implemented by an individual state_t.onStateChange call back function.

Parameters

me	pointer to own state; not null

Returns

0: state or sub-state changed 0xFF: fault status panic ... else: current state kept, of course

7.69 weRasp/weUSBscan.c File Reference

USB 1D / 2D scanners mimicking keyboards on Raspberry Pi.

```
#include "weUSBscan.h"
```

Variables

- char const deRawCoDisAltGrph [120]

  Key position number to character, AltGr, German keyboard.
- char const deRawCoDisNoShift [90]

  Key position number to character, no shift, German keyboard.
- char const deRawCoDisShifted [90]

  Key position number to character, shifted, German keyboard.
- wchar_t raw2wcharAltGrph [102]

  Key position number to character, with AltGR.
- wchar_t raw2wcharNoShift [60]

  Key position number to character, no shift.
- wchar_t raw2wcharShifted [60]

  Key position number to character, with shift.
- unsigned char scanKeyAction [32]

  The one keystroke read buffer.
- int scanKeybLang

The keyboard language.

- wchar_t scanResult [162]

  The result of one scan.

- char const usRawCoDisNoShift [90]

  Key position number to character, no shift, US keyboard.

- char const usRawCoDisShifted [90]

  Key position number to character, shifted, US keyboard.

7.69.1 Detailed Description

USB 1D / 2D scanners mimicking keyboards on Raspberry Pi.

```
Copyright  (c)   2020     Albrecht Weinert
weinert-automation.de     a-weinert.de
    /         /      /\
   /         /___   /  \       |
   \        /____\ /____\ |  _|__
    \  /\  / \     /       \|   |
     \/  \/   \__/         \__/|_
```

Revision history

```
Rev. 234 20.01.2021
Rev. 232 03.09.2020 : new (extracted from testOnPi.c)
```

This is a supplementary basic library to handle USB barcode and QR code scanners. By plug'n play such scanner would normally appear as device

```
/dev/hidraw0
```

. To make this usable for scanner application programs run without sudo apply

```
sudo chmod  664 /dev/hidraw0
```

before.

7.69.2 Variable Documentation

7.69.2.1 usRawCoDisNoShift

```
char const usRawCoDisNoShift[90]
```

Key position number to character, no shift, US keyboard.

This utf-8 or multibyte character array respectively string describes the translation of key number to character for a US keyboard.
It will have to be transfered as wide character array to raw2wcharNoShift.

7.69.2.2 usRawCoDisShifted

```
char const usRawCoDisShifted[90]
```

Key position number to character, shifted, US keyboard.

See the explanation at usRawCoDisNoShift.

See also

raw2wcharShifted

7.69.2.3 deRawCoDisNoShift

```
char const deRawCoDisNoShift[90]
```

Key position number to character, no shift, German keyboard.

This utf-8 or multibyte character array respectively string describes the translation of key number to character for a US keyboard.
It will have to be transfered as wide character array to raw2wcharNoShift.

Remarks on non US keyboard emulations by the "USB Wired 2D Barcode Scanner" <MJ-8200> and consorts:
We strongly recommend not to use them and refrain from applications using more than primitive USASCII. As "↩ German keyboard", e.g., the scanner does neither recognise nor send . Besides being called German without umlauts [sic!] the scanner is ignorant to some other characters on every German keyboard (which are probably there because of being used in Western Europe). Additionally the scanner when set to German inserts additional strings of characters with no obvious sense or system.
In the end we consider everything beyond factory reset (except low beeper volume) as not functional.
Without the hard bug of inventing characters and string not contained in the QR-code, the so called German keyboard would give us some extra characters — but not umlauts.

7.69.2.4 deRawCoDisShifted

```
char const deRawCoDisShifted[90]
```

Key position number to character, shifted, German keyboard.

See the explanation at usRawCoDisNoShift.

See also

raw2wcharShifted deRawCoDisNoShift

7.69.2.5 deRawCoDisAltGrph

```
char const deRawCoDisAltGrph[120]
```

Key position number to character, AltGr, German keyboard.

See the explanation at usRawCoDisNoShift.

See also

raw2wcharShifted deRawCoDisNoShift

7.69.2.6 raw2wcharNoShift

```
wchar_t raw2wcharNoShift[60]
```

Key position number to character, no shift.

The length is 60. Valid characters are in the 4..56 key number range; there may be gaps. 0..3 are errors.

See also

usRawCoDisNoShift

7.69.2.7 raw2wcharShifted

```
wchar_t raw2wcharShifted[60]
```

Key position number to character, with shift.

The length is 60. Valid characters are in the 4..56 key number range; there may be gaps. 0..3 are errors.

See also

usRawCoDisShifted

7.69.2.8 raw2wcharAltGrph

```
wchar_t raw2wcharAltGrph[102]
```

Key position number to character, with AltGR.

The length is 60. Valid characters are in the 4..56 key number range; there will be many gaps. 0..3 are errors.

See also

usRawCoDisShifted

7.69.2.9 scanKeybLang

```
int scanKeybLang
```

The keyboard language.

The language of the USB keyboard (see scanKeyAction) emulated by the scanner is stored here as: 0=US (default), 10=DE

7.69.2.10 scanResult

```
wchar_t scanResult[162]
```

The result of one scan.

The result of consecutive keystrokes (see scanKeyAction) is stored here as array respectively string of wide characters.

7.69.2.11 scanKeyAction

```
unsigned char scanKeyAction[32]
```

The one keystroke read buffer.

Keyboard input comes in blocks of 8 bytes each. Hence a length of 8 would be sufficient. Hence, 32 is a reserve for device errors or the driver not recognising the gap between blocks.

The 8 bytes are:

```
             Bit/Value:  0/1 1/2   2/4 3/8         4/16 5/32  6/64  7/128
[0]  Modifier keys Left: cntl shift Alt Win  Right: cntl shift AltGr Win
[1]  Reserved field                 always 0
[2]  Keypress 1                     in a funny code (4 is a)
[3]  2nd simultaneously pressed key
[4..7]  Keypress 3..6
```

As scanners won't "press" more than one key at a time only bytes [0] and [1] will contain information.
Byte[2] will be a crazy key code for a..z1..90... athwart to any utf or unicode. In the end the semantic of that "code" is a mixture of key value and key position on the keyboard. In the end few scanners get more than an American (and a Chinese?) keybord right. When setting the scanner to German keyboard you may miss one or two of .

On byte [0] one should see only three values: /code 0 : no modifier, no shift 2 : shift, that means a->A 64: altGr

7.70 weRasp/weUtil.c File Reference

Some system related time and utility functions for Raspberry Pi.

```
#include "weUtil.h"
#include <errno.h>
```

Functions

- int advanceTmTim (struct tm *rTm, char *rTmTxt, uint8_t sec)

  Advance broken down real time by seconds.
- int char2hexDig (char c)

  Character to hexadecimal.
- int cycTaskDestroy (cycTask_t *cykTask)

  Destroy a cyclic task / threads structure.
- int cycTaskEvent (cycTask_t *cycTask, uint8_t noEvents, timespec stamp, cycTaskEventData_t cycTask↩EventData)

  Handle and signal events.
- int cycTaskInit (cycTask_t *cykTask)

Initialise a cyclic task / threads structure.

- int cycTaskWaitEvent (cycTask_t *cycTask, uint32_t eventsThreshold, cycTask_t *cycTaskSnap)

  Wait on signalled event.
- int endCyclist (void)

  The cycles handler arrived.
- char * formFixed16 (char *target, uint8_t targetLen, uint16_t value, uint8_t dotPos)

  Format 16 bit unsigned fixed point, right aligned.
- char * formFixed32 (char *target, uint8_t targetLen, uint32_t value, uint8_t dotPos)

  Format 32 bit unsigned fixed point, right aligned.
- int genErrWithText (char const *txt)

  Generate error text (errorText) with system error text appended.
- uint8_t get10inS ()

  Get a 10th of second in s reading.
- uint32_t getAbsS ()

  Get the absolute s reading.
- uint32_t getCykTaskCount (cycTask_t const *const cycTask)

  Get a cycle's/task's current event counter.
- uint16_t getMSinS ()

  Get a ms in s reading.
- void initStartRTime ()

  Initialise start (real) time.
- uint32_t ioSetClrSelect (uint8_t pin)

  Fetch a clear and set select bit for a GPIO pin.
- int isValidIp4 (char const *str)

  Check if a string is a valid IPv4 address.
- void logErrorText ()

  Log the (last) common error text generated.
- void logErrText (char const *txt)

  Log an error text on errLog.
- void logErrWithText (char const *txt)

  Log error text (on errLog) with system error text appended.
- void logStampedText (char const *txt)

  Log an event or a message on outLog as line with time stamp.
- void monoTimeResol (timespec *timeRes)

  Absolute time (source) resolution.
- void onSignalExit (int s)

  On signal exit.
- void onSignalExit0 (int s)

  On signal exit 0.
- void onSignalStop (int s)

  On signal stop.
- unsigned int parse2Long (char *const optArg, long int *parsResult)

  Parse a string of integer numbers.
- int parsInt (const char *str, const int lower, const int upper, const int def)

  Parse int with checks.
- uint16_t stopMSwatch ()

  Get a (stop-watch) ms reading.
- void strLappend (char *dest, char const *src, int n)

  Append one char sequence left justified at another one.
- void strLinto (char *dest, char const *src, size_t n)

  Set one char sequence left justified into another one.

- void strRinto (char *dest, char const *src, size_t n)

  Set one char sequence right justified into another one.
- int theCyclistStart (int startMsDelay)

  Start the cycles handler.
- int theCyclistWaitEnd ()

  Wait for the end of the cycles thread.
- timespec timeAdd (timespec const t1, timespec const t2)

  Add two times as new structure.
- void timeAddTo (timespec *t1, timespec const t2)

  Add two times overwriting the first operand.
- int timeCmp (timespec const t1, timespec const t2)

  Compare two times.
- int timeSleep (unsigned int micros)

  Relative delay for the specified number of s.

Variables

- char actRTmTxt [34]

  Actual broken down time (text).
- timespec allCycStart

  Common absolute / monotonic start time of all cycles.
- char const **bin8digs** [256][10]

  "0000_0000" .. "1111_1111"
- volatile uint8_t commonRun

  Common boolean run flag for all threads.
- const uint32_t **csBit** [32]

  single bit set. 1 2 4 8 ... 0x80000000
- cycTask_t **cyc100ms**

  100ms cycle (data structure)
- cycTask_t **cyc10ms**

  10ms cycle (data structure)
- cycTask_t **cyc1ms**

  1ms cycle (data structure)
- cycTask_t **cyc1sec**

  1s cycle (data structure)
- cycTask_t **cyc20ms**

  20ms cycle (data structure)
- char errorText [182]

  Common error text.
- uint8_t have100msCyc

  Flag to enable the 100ms cycle.
- uint8_t have10msCyc

  Flag to enable the 10ms cycle.
- uint8_t have1msCyc

  Flag to enable the 1ms cycle.
- uint8_t have1secCyc

  Flag to enable the 1s cycle.
- uint8_t have20msCyc

  Flag to enable the 20ms cycle.
- long int parsResult [14]

Long array of length 14.

- volatile int sigRec

  Storage for the signal (number) requesting exit.

- timespec startRTime

  Start time (structure, monotonic real time clock).

- char const *const stmp23

  The current time as text.

- uint32_t const *const stmpSec

  The real time epoch seconds.

- int8_t **vcoCorrNs**

  external for test/debug only (don't change)

7.70.1 Detailed Description

Some system related time and utility functions for Raspberry Pi.

Copyright (c) 2017 Albrecht Weinert
weinert-automation.de a-weinert.de

Revision history

```
Rev. 240 10.04.2021
Rev. 050+ 2017-10-16 : cycTask_t->mutex now pointer (allows common mutex)
Rev. 054+ 2017-10-23 : timing enhanced, common mutex forced/standard
Rev. 076  2017-12-01 : advanceTim month change debug.; ret 7 for offs. chg.
Rev. 128+ 2018-04-16 : some cleaning on comments
Rev. 155  27.06.2018 : time handling debugged
Rev. 161  09.07.2018 : 20ms thread is now scheduled on end, too (was not)
Rev. 187  14.10.2018 : minor typos
Rev. 200  16.04.2019 : logging improved; formatting enh.
Rev. 201  26.04.2019 : renamed from sysUtil.c
Rev. 209  22.07.2019 : formFixed.. not void
Rev. 233  26.09.2020 : 10 ms cycle added
```

cross-compile by:

```
arm-linux-gnueabihf-gcc -DMCU=BCM2837  -I./include  -c -o weRasp/weUtil.o weRasp/weUtil.c
```

cross -build as library by:

```
arm-linux-gnueabihf-gcc -Wall -DMCU=BCM2837  -I./include  -shared -o libweUtil.so -fPIC weRasp/weUtil.c
copy libweUtil.so C:\util\WinRaspi\arm-linux-gnueabihf\sysroot\usr\lib\
```

And finally (ftp) transfer libweUtil.so to Raspy's /usr/local/lib/ and run sudo ldconfig there. N.b.: In most cases, it brings no disadvantage and it is easier to link instead of building and using an own library.

For documentation see the include files weUtil.h and sysBasic.h

7.70.2 Function Documentation

7.70.2.1 timeAdd()

```
timespec timeAdd (
            timespec const t1,
            timespec const t2 )
```

Add two times as new structure.

Parameters

t1	summand as time structure (not NULL!, will be left unchanged)
t2	the second summand (dto.)

Returns

the sum (probably passed as hidden parameter by the way)

7.70.2.2 timeAddTo()

```
void timeAddTo (
            timespec * t1,
            timespec const t2 )
```

Add two times overwriting the first operand.

Parameters

t1	the time structure to add to (not NULL!, will be modified)
t2	the summand (not NULL!, will be left unchanged)

7.70.2.3 timeCmp()

```
int timeCmp (
            timespec const t1,
            timespec const t2 )
```

Compare two times.

Parameters

t1	the time structure to compare to t2 (not NULL!)
t2	the time structure to compare t1 with (not NULL!)

Returns

0: equal; +: t1 is greater (2 by s, 1 by ns); -: t1 is smaller

7.70.2.4 timeSleep()

```
int timeSleep (
            unsigned int micros )
```

Relative delay for the specified number of s.

This is local sleep. It should not be used in combination with absolute times and cyclic threads. It is just an utility for test or very short delays (as a better replacement for spinning).

Parameters

micros	sleep time in s; allowed 30 .. 63000

Returns

sleep's return value if of interest (0: uninterrupted)

7.70.2.5 monoTimeResol()

```
void monoTimeResol (
            timespec * timeRes )
```

Absolute time (source) resolution.

This function sets the time structure provided to the absolute time's (ABS_MONOTIME default: CLOCK_↩ MONOTONIC) resolution.

Raspian Jessie on a Raspberry Pi 3 always yielded 1ns, which one may believe or not. We took it as "sufficient for accurate 1ms cycles".

Parameters

timeRes	the time structure to be used (never NULL!)

7.70.2.6 strLinto()

```
void strLinto (
            char * dest,
            char const * src,
            size_t n )
```

Set one char sequence left justified into another one.

This function copies n characters from src to dest left justified. If the length of src is less than n the remaining length on right in dest will be filled with blanks.

Attention: dest[n-1] must be within the char array provided by dest. This cannot and will not be checked!

Hint: Contrary to strncpy there's no padding with 0. If you want dest to end after the insertion use strLappend().

Parameters

dest	the pointer to / into the destination sequence where src is to be copied to. If NULL nothing happens.
src	the sequence to be copied. If NULL or empty fill is used from start
n	the number of characters to be copied from src.

7.70.2.7 strLappend()

```
void strLappend (
            char * dest,
            char const * src,
            int n )
```

Append one char sequence left justified at another one.

This function copies n characters from src to dest and lets dest then end with 0 (end of string). If n is negativ, -n characters will be copied and dest will end with new line and 0;

Attention: dest[n] respectively dest [-n + 1] must be within the char array provided by dest. This cannot and will not be checked!

Parameters

dest	the pointer to / into the destination sequence where src is to be copied to. If Null nothing happens.
src	the sequence to be copied. If Null or empty fill is used from start
n	the absolute value is the number of characters to be copied from src. If this number exceeds 300 it will be taken as 0. If n is negative a line feed will be appended, too.

7.70.2.8 strRinto()

```
void strRinto (
            char * dest,
            char const * src,
            size_t n )
```

Set one char sequence right justified into another one.

This function copies n characters from src to dest right justified. If the length of src is less than n the remaining length on left in dest will be filled with blanks.

Attention: dest[n-1] must be within the char array provided by dest. This cannot and will not be checked!

Hint to append instead of insert: If this operation shall be at the end of the changed char sequence do
dest[n] = 0;
Then, of course, dest[n] must be within the char array provided by dest.

Parameters

dest	the pointer to / into the destination sequence where src is to be copied to. If NULL nothing happens.
src	the sequence to be copied. If NULL or empty fill is used from start
n	the number of characters to be copied from src.

7.70.2.9 formFixed16()

```
char * formFixed16 (
            char * target,
            uint8_t targetLen,
            uint16_t value,
            uint8_t dotPos )
```

Format 16 bit unsigned fixed point, right aligned.

formFixed16(target, 6, 1234, 2), e.g., will yield " 12.34".
formFixed16(target, 6, 4, 2), e.g., will yield " 0.04".

If the value would not fit within targetLen characters leading digits will be truncated.

Parameters

target	pointer to first of targLen characters changed
targetLen	field length 2..16; number of characters changed
value	the fixed point value
dotPos	where the fixed point is 0..6 $<$ targetLen

Returns

points to the most significant digit set or NULL on error / no formatting

7.70.2.10 formFixed32()

```
char * formFixed32 (
            char * target,
            uint8_t targetLen,
            uint32_t value,
            uint8_t dotPos )
```

Format 32 bit unsigned fixed point, right aligned.

This function behaves like formFixed16() except for handling 32 bit values. formFixed16() should be preferred, when feasible.

Parameters

target	pointer to first of targLen characters changed
targetLen	field length 2..16; number of characters changed
value	the fixed point value
dotPos	where the fixed point is 0..6 $<$ targetLen

Returns

points to the most significant digit set or NULL on error / no formatting

7.70.2.11 ioSetClrSelect()

```
uint32_t ioSetClrSelect (
             uint8_t pin )
```

Fetch a clear and set select bit for a GPIO pin.

For the masks to set or clear GPIO bits each bit 0..31 selects the GPIO pin 0..31 respectively 32..53.

Parameters

pin	GPIO pin number (only 5 bits relevant here)

Returns

the the function select bit (a value with one bit set)

7.70.2.12 isValidIp4()

```
int isValidIp4 (
             char const * str )
```

Check if a string is a valid IPv4 address.

Syntactically valid IPv4 addresses are: 0.0.0.0 .. 255.255.255.255

Parameters

str	The string containing the address, only; 0-terminated

Returns

0: no syntactically valid IPv4 address; 1: OK

7.70.2.13 parsInt()

```
int parsInt (
            const char * str,
            const int lower,
            const int upper,
            const int def )
```

Parse int with checks.

This function expects parameter str to point to a null-terminated string. If not def is returned. If lower > upper def is returned. If the string str contains a decimal integer number n, fulfilling lower <= n <= upper n is returned, or def otherwise.

If the string str starts with [+|-][min|med|max] ignoring case lower respectively ((lower + upper) / 2) respectively upper is returned. A leading sign (+|-) as well as any trailing characters are ignored.

Parameters

str	0-terminated string containing a decimal integer number, or one of the keywords described above
lower	lower limit
upper	upper limit
def	default value, to be returned when str is not a pure decimal number one of the keyword starts or when the result violates the limits

7.70.2.14 char2hexDig()

```
int char2hexDig (
            char c )
```

Character to hexadecimal.

Parameter values '0'..'9' return 0..9. Parameter values 'A'..'F' and 'a'..'f' return 10..15. Other values return -1.

7.70.2.15 parse2Long()

```
unsigned int parse2Long (
            char *const optArg,
            long int * parsResult )
```

Parse a string of integer numbers.

The string optArg will be tokenised taking any occurrences of the characters " +,;" (blank, plus, comma, semicolon) as border. "Any occurrences" means two commas „", e.g., acting as one separator and not denoting an empty number.
N.b.: The string optArg will be modified (by replacing the first character of the token separators found by zero ('\0').

The number format accepted and parsed is decimal and hexadecimal. Hexadecimal starts with 0x or 0X. Leading zeros have no significance (in C stone age sense of being octal).

Parameters

optArg	the string to be passed to a number of integer numbers, passed as program parameter, e.g.
parsResult	pointer to an array of long int, minimal length 14 (!)

Returns

the number of integer numbers parsed ab put into parsResult[], 0..14

7.70.2.16 logErrWithText()

```
void logErrWithText (
            char const * txt )
```

Log error text (on errLog) with system error text appended.

Gives a (English) clear text translation of the latest system stored error. If txt is not null it will be prepended. This function appends a linefeed and flushes errLog.

Parameters

txt	text to be prepended (should nod be longer than 58 characters)

7.70.2.17 logErrorText()

```
void logErrorText (
            void  )
```

Log the (last) common error text generated.

This function outputs the last generated errorText (by genErrWithText() e.g.) to errLog. It appends a linefeed and flushes errLog.

7.70.2.18 genErrWithText()

```
int genErrWithText (
            char const * txt )
```

Generate error text (errorText) with system error text appended.

Gives a (English) clear text translation of the latest system stored error. If txt is not null it will be prepended. Date and time will be prepended anyway.

Parameters

txt	text to be prepended (should nod be longer than 58 characters)

Returns

0: no error; else mutex error (time and date may be spoiled)

7.70.2.19 logErrText()

```
void logErrText (
            char const * txt )
```

Log an error text on errLog.

If txt is not null it will be output to errLog and errLog will be flushed.

Parameters

txt	text to be output; n.b not LF appended

7.70.2.20 logStampedText()

```
void logStampedText (
            char const * txt )
```

Log an event or a message on outLog as line with time stamp.

If txt is not null it will be output to outLog. A time stamp is prepended and a line feed is appended. txt will be shortened to 50 characters if longer.

Parameters

txt	the text to be output

7.70.2.21 onSignalExit()

```
void onSignalExit (
            int s )
```

On signal exit.

This function is intended as signal hook; see signal(s, hook). When called, this function calls exit(s) and never returns.

Parameters

s	the signal forwarded to exit

7.70.2.22 onSignalExit0()

```
void onSignalExit0 (
            int s )
```

On signal exit 0.

This function is intended as signal hook; see signal(s, hook).
When called this function calls exit(0) and never returns.
This may be used as hook for s==SIGIN, to provide a normal return on cntl-C.

Parameters

s	ignored

7.70.2.23 onSignalStop()

```
void onSignalStop (
            int s )
```

On signal stop.

This function is a prepared signal hook. When called it sets sigRec by s and clears commonRun.

7.70.2.24 cycTaskInit()

```
int cycTaskInit (
            cycTask_t * cykTask )
```

Initialise a cyclic task / threads structure.

This function initialises a cyclic or non cyclic (asynchronous random event driven) task (thread) structure. Common mutex and an own condition are initialised, the event counter (.count) is set to 0.

Note: For the standard cycles provided here, 1ms, 100ms .., this initialisation is done in theCyclistStart() and the destruction (by cycTaskDestroy()) in endCyclist().

Parameters

cykTask	the task structure to initialise (not NULL!)

Returns

0: success, else: one of the error codes occurred

7.70.2.25 cycTaskDestroy()

```
int cycTaskDestroy (
            cycTask_t * cykTask )
```

Destroy a cyclic task / threads structure.

Parameters

cykTask	the task structure to destroy (not NULL!)

Returns

0: success, else: one of the error codes occurred

7.70.2.26 cycTaskEvent()

```
int cycTaskEvent (
            cycTask_t * cycTask,
            uint8_t noEvents,
            timespec stamp,
            cycTaskEventData_t cycTaskEventData )
```

Handle and signal events.

This is a helper function for the controller / manager to be called when having determined, that one or more events happened.

Parameters

cycTask	the task structure (not NULL!)
noEvents	number of events (usually 1); summand to cykTask.count
stamp	absolute monotonic time of the event; sets sykTask.stamp
cycTaskEventData	actual cyclic event data

Returns

0: success, else: one of the error codes occurred

7.70.2.27 cycTaskWaitEvent()

```
int cycTaskWaitEvent (
            cycTask_t * cycTask,
            uint32_t eventsThreshold,
            cycTask_t * cycTaskSnap )
```

Wait on signalled event.

This is a helper function for a worker thread. It will return on reaching the signalled event(s) or on ! commonRun. If cykTaskSnap is not NULL cycTask will be assigned to it under mutex lock before returning. This is helpful if cykTask's events are broadcast to multiple handlers.

Parameters

cycTask	the task structure (not NULL!)
eventsThreshold	threshold for cykTask.count (update for every round)
cycTaskSnap	copy of cykTask under mutex lock before returning

Returns

0: success, else: one of the error codes occurred

7.70.2.28 getCykTaskCount()

```
uint32_t getCykTaskCount (
            cycTask_t const *const cycTask )
```

Get a cycle's/task's current event counter.

This is done under (cyclist's) mutex lock.

Parameters

cycTask	the task structure

Returns

cycTask's event counter value (.count) got under lock; 0x7FffFFffFFf7 on any error (null, lock error) situation

7.70.2.29 stopMSwatch()

```
uint16_t stopMSwatch ( )
```

Get a (stop-watch) ms reading.

This function provides an 16 bit reading of the cyclist's (64 bit) milliseconds. It is intended for measuring short (<= 1min) durations.

Hint: This functions thread safety stems from the hope of 16 bit increments being atomic. Even if no problems in this respect were observed on Raspberry Pi 3s, it may be just hope in the end. Thread-safe values are, of course, provided in the cycTaskEventData_t structure. But those are frozen within one cycle task step, and, hence, not usable as stop-watch readings within such step.

7.70.2.30 getMSinS()

```
uint16_t getMSinS ( )
```

Get a ms in s reading.

This function provides the cyclist's ms in sec as 16 bit unsigned reading. It is intended for measuring and testing durations.

Hint: This functions does nothing for thread safety. It hopes 16 bit accesses being atomic. Even if no problems in this respect were observed on Raspberry Pi 3s, it may be just hope in the end. Thread-safe values are, of course, provided in the cycTaskEventData_t structure. But those are frozen within one cycle task step, and, hence, not usable as stop-watch readings within such step.

7.70.2.31 get10inS()

```
uint8_t get10inS ( )
```

Get a 10th of second in s reading.

This function provides the cyclist's tenth in seconds reading (0..9). It is intended for cyclic tasks with times greater than 100 ms or asynchronous tasks to get a coarse second sub-division.

Hint: Cyclic tasks get this value in their task date valid at start. This function provides an actual value for tasks running longer than 100ms.

7.70.2.32 getAbsS()

```
uint32_t getAbsS ( )
```

Get the absolute s reading.

This function provides a 32 bit monotonic seconds value.
Base of this 32 bit value is the cyclist's 64 bit epoch time in seconds, 0 being 1.1.1970 00:00:00 UTC on almost all Linuxes and C libraries.

This unsigned 32 bit holds until 7. February 2106, which is far longer than the projected lifetime age of this library and of Raspberry Pi3s. (But who knows?) The value may be used for seconds-resolution, absolute (i.e. zone and DST independent) time-stamps and interval calculations (which will be incorrect with leap seconds).

Hint: Cyclic tasks get this value (.realSec) in their task date valid at tick start. Hence, this function is intended for asynchronous tasks or cyclic tasks with periods > 1s.

Since version R.110 we dare to fetch this value without lock, assuming ARMv7 32 bit load and stores being atomic.

7.70.2.33 theCyclistStart()

```
int theCyclistStart (
            int startMsDelay )
```

Start the cycles handler.

This function initialises and then runs the predefined cycles cycles (as of Sept. 2020: 1ms, 10ms, 20ms, 100ms and 1s; see have1msCyc) when enabled.

Besides the absolute / monotonic times for the cycles it also initialises real time and timers handling.

Timers and cycles are run in an extra thread made by this function. And to be precise, the cycles are not run here; instead, cyclic events are generated and broadcast.

As the thread started by this function also provides monotonic and civil times and stamps it should be started with the program (i.e. earliest in main()). Preparation time before the cycles should start can be handled by the delay parameter.

As of September 2020 five cycles (see above) are defined and handled. It is strongly recommended not to use more than two of them and implement other cycles with multiple periods by sub-division. That means, e.g., do not enable the 20ms cycle when having the 10ms one.

Parameters

startMsDelay	number of ms before generating the first cyclic event; allowed range 12 .. 1200; default 1

Returns

0: after having initialised all and having made and started the cyclist thread; other values signal errors

7.70.2.34 theCyclistWaitEnd()

```
int theCyclistWaitEnd ( )
```

Wait for the end of the cycles thread.

This function does so by unconditionally joining the cyclist thread.

Returns

the return value of thread join; 0: join OK

7.70.2.35 endCyclist()

```
int endCyclist (
            void  )
```

The cycles handler arrived.

This function cleans up after theCyclist. It should be called after theCyclist() ending successfully on commonRun false. The controller thread shall call this function after having joined and cleaned up all of its threads. It may also be put in an exit hook.

Returns

0: OK; else: a cycTaskDestroy() error

7.70.2.36 initStartRTime()

```
void initStartRTime ( )
```

Initialise start (real) time.

This will be done in theCyclistStart(int). Hence, this function is for "non-cyclic" applications, mainly. Nevertheless it can be called before theCyclistStart(int) and won't be repeated therein.

7.70.2.37 advanceTmTim()

```
int advanceTmTim (
            struct tm * rTm,
            char * rTmTxt,
            uint8_t sec )
```

Advance broken down real time by seconds.

This function just advances the broken down (local) time structure rTm and the fitting text rTmTxt by 1 to 40s. All fields not affected by adding to the seconds part, won't be touched.

This function won't care about leap seconds nor handle DST rules. If this is to be kept up to date, it is recommended to refresh it on every hour change (return >=3) by
clock_gettime(CLOCK_REALTIME,..) and localtime_r(..).
Depending on OS, that might be an expensive operation with extra locks.

With wrong parameter values this function does nothing (returns 0).

Parameters

rTm	pointer to broken down real time
rTmTxt	date text, length 32, format Fr 2017-10-20 13:55:12.987 UTC+20 NULL is substituted by actRTmTxt
sec	1..40 will be added; else: error

Returns

0: error (rTm NULL e.g.); 1: seconds changed; 2: minute; 3: hour; 4: day; 5: month ; 6: year; 7: zone offset

7.70.3 Variable Documentation

7.70.3.1 parsResult

```
long int parsResult[14]
```

Long array of length 14.

Prepared for non thread safe use with parse2Long()

7.70.3.2 stmp23

```
char const* const stmp23
```

The current time as text.

/code The format is: 2017-10-20 13:55:12.987 UTC+20 .....0123456789x123456789v123456789 /endcode
The length is 30.
Do NOT change the value provided by this pointer.

7.70.3.3 stmpSec

```
uint32_t const* const stmpSec
```

The real time epoch seconds.

Do NOT change the value provided by this pointer.

7.70.3.4 errorText

```
char errorText[182]
```

Common error text.

This text is set by genErrWithText() and hence indirectly by (many) other functions optionally generating error texts.

7.70.3.5 commonRun

```
volatile uint8_t commonRun
```

Common boolean run flag for all threads.

When set false, all threads must exit as soon as possible. On any case, a thread has to exit and clean up on next signal. Setting commonRun false implies the end of the application/program and all of its threads as soon as possible.

Initialised as 1 (true) Set 0 by onSignalStop() (or application program)

7.70.3.6 sigRec

```
volatile int sigRec
```

Storage for the signal (number) requesting exit.

Set by: onSignalStop()
See also: retCode

7.70.3.7 allCycStart

```
timespec allCycStart
```

Common absolute / monotonic start time of all cycles.

May be considered as program's start time when cycles are started early by theCyclist. Normally not to be modified.

7.70.3.8 have1msCyc

```
uint8_t have1msCyc
```

Flag to enable the 1ms cycle.

As a rule no more than two oft the cycles offered — cyc1ms, cyc10ms, cyc20ms, cyc100ms, cyc1sec — shall be enabled. This is no restriction as a faster cycle can easily (and often should) implement slower cycles by subdivision.

The default setting is 1ms and 100ms ON and all others OFF.
If other settings are used the flags should be set at the program's early initialisation phase and afterwards left untouched.

default: ON

See also

cycTask_t cyc1ms

7.70.3.9 have10msCyc

```
uint8_t have10msCyc
```

Flag to enable the 10ms cycle.

default: OFF

See also

have1msCyc cycTask_t cyc10ms

7.70.3.10 have20msCyc

```
uint8_t have20msCyc
```

Flag to enable the 20ms cycle.

default: OFF

See also

have1msCyc cycTask_t cyc20ms

7.70.3.11 have100msCyc

```
uint8_t have100msCyc
```

Flag to enable the 100ms cycle.

default: ON

See also

have1msCyc cycTask_t cyc100ms

7.70.3.12 have1secCyc

```
uint8_t have1secCyc
```

Flag to enable the 1s cycle.

default: OFF

See also

have1msCyc cycTask_t cyc1s

7.70.3.13 startRTime

```
timespec startRTime
```

Start time (structure, monotonic real time clock).

By initStartRTime() or by theCyclistStart() actRTime and this startRTime will initially be set. actRTime may be updated on demand, but startRTime should be left unchanged.

7.70.3.14 actRTmTxt

```
char actRTmTxt[34]
```

Actual broken down time (text).

```
            |-3|-  10  -|1|- 12     -|
```

The format is: Fr 2017-10-20 13:55:12.987 UTC+20 .....0123456789x123456789v123456789t1234 was Fr 2017-10-20 13:55:12 UTC+20 The length is 32.

Index