Programming library

convert_lib

converting Simatic formats to C-formats

Version V2.7 (10.05.2018)

Manual

convert_lib has been developed by

Heisch Automation

Ingenieurbüro für Industrieautomatisierung

Adresse: Im Vorderen Großthal 4 D 76857 Albersweiler /Pfalz
Tel: +49 6345 9496732 Mobil: +49 171 4311359
e-mail: sales@heisch-automation.de www.heisch-automation.de

http://sites.inka.de/heisch
http://www.heisch-automation.de

Contents

1. General Notes

convert_lib contains functions for transferring Simatic(R) types to ANSI-C- types and vice versa.

convert_lib supports both, Simatic S5 types and Simatic S7 types (S7-300/400) and also S7-1200 / 1500).

convert_lib supports the data formats, which are used for data transfer, it does not include pointer formats of S7.

convert_lib is an add on to the rk*_server family, but also can be used in proprietary solutions without any relation to the rk*_server family.

1.1. Labels

RK512, 3964R, SIMATIC are labels of SIEMENS AG.
S.u.S.E. Linux is a label of S.u.S.E. GmbH.
UNIX is a label of the X/Open Company Limited.
MS-Windows is a label of Microsoft Corporation.

1.2 Supported operation systems and system requirements

convert_lib was developed on S.u.S.E Linux 6.3 but runs on all other Linux systems as well.

Because the library is shipped in ANSI-C source code, it can be used any Unix system.

At the moment, we do not support MS-Windows, but we are quite sure that most parts of **convert_lib** work on windows based systems as well.
Exception: The Date and Time functions are very Unix specific.

(A client uses our floating point functions in his C-programs, written with the Borland C-Builder)

1.3 Changes (IMPORTANT for upgrades)

Since Version 2.7

The original idea has been, to devide the functions into different files to save rare space an to come to smaller programs.
(A 'big' Linux computer in these days has had 64 MB RAM.)
Now, all functions reside in one file.

Because S5 plcs are quite out dated and the new plcs (S7-1200 and S7-1500) provide a lot of new data formats, the functions, the functions now are defided into

- S5 : the S5 formats stay in s5types.h, s5types.c and s5types.o
  The file 's5types.c' contains the latest version from 2016, for backward compatibility, also the s7 specific functions stay included.
- S7 : the S7 formats now are in s7types.h, s7types.c and s7types.o
  For new implementations, it is strongly recommended to use this library.
  All names of the functions now are hinting more the 'S7', i.e. reading an integer now is named get_s7INT().

Not included into the library:
Functions to read or write WCHAR and WSTRING.

Since Version 2.0

Changing the version number to V2.0, we changed the Simatic side parameter format from **unsigned short *in** to **void *in** and **unsigned short *out** to **void *out**.

Reason:
The first versions of **convert_lib** were written for a Simatic S5.
A Simatic S5 data word consists of 16 bits. Therefore most programmers read a data block into a array of unsigned short, to get a representation of word numbers into indices.

i.e.: If DB 20, DW 0 to DB48 is read into "unsigned short db20[50] ", the contents of (Simatic side) DB20, DW 34 is in (computer side) db20[34].

Since Simatic S5 is dead and nearly all costumers use Simatic S7, the parameter format unsigned short is counterproductive, because it requires a cast on any parameter.

We decided to change the format to void, to enable all forms of representations.

1.4 Packing list

The program package consists of the files

s5types.h	the header file for the s5 functions
s5types.c	all together the S5 functions
s7types.h	the header file for the s7 functions
s7types.c	all together the S7 functions
Makefile	The makefile for the library
Changes.txt	actual state
Manual.pdf	Manual as Pdf-file, English
Beschreibung.pdf	Beschreibung als PDF-Datei, deutsch.

1.5 Missing formats and free upgrades

Not included have been functions to read or write WCHAR and WSTRING.

Although we do not have the duty to deliver the missing formats, we will deliver newer versions of convert_lib to all ancient buyers.

1.6 Warranty

The **convert_lib** is a software package which is intensively tested and approved in industrial installations.

From the actual state of technique it is impossible to guarantee the absence of faults.

Therefore:
This software is provided 'as it is' without any expressed warranty of any kind. Under no circumstances the author is responsible for the proper functioning of this software, nor does the author assume any responsibility for damages incurred by it's use.

2. s7types.c, s7types.h FUNCTIONS

2.1 fixed point formats

The byte functions are trivial they only exist to have a uniform design.

get_s7BYTE (*source)

S7: to get an unsigned BYTE (uint8_t)

uint8_t get_s7BYTE (void *in) ;

set_s7BYTE (*dest, source)

writes a 8 bit value to a BYTE

void set_s7BYTE (void *out, unsigned char in) ;

get_s7USINT (*source)

S7: to get an unsigned int (8 bits) as uint8_t

uint8_t get_s7USINT (void *in) ;

set_s7USINT (*dest, source)

writes a 8 bit unsigned value as USINT

void set_s7USINT (void *out, unsigned char in) ;

get_s7CHAR (*source)

S7: to get an signed int 8 as signed char

signed char get_s7CHAR (void *in) ;

set_s7CHAR (*dest, source)

writes a 8 bit value to a CHAR

void set_s7CHAR (void *out, signed char in) ;

get_s7SINT (*source)

S7: to get a DBB of type SINT as signed 8bit integer

int8_t get_s7SINT (void *in) ;

set_s7SINT (*dest, source)

writes a 8 bit signed integer as SINT

void set_s7SINT (void *out, int8_t in) ;

get_s7WORD (*source)

to get a word (16bit) as a unsigned int16

uint16_t get_s7WORD (void *in) ;

set_s7WORD (*dest, source)

writes a 16 bit unsigned or unsigned short (uint16_t) to a WORD

void set_s7WORD (void *out,uint16_t in) ;

get_s7UINT (*source)

to get a word (16bit) as a unsigned int.

uint16_t get_s7UINT (void *in) ;

set_s7UINT (*dest, source)

writes a 16 bit unsigned short to a UINT

void set_s7UINT (void *out,uint16_t in) ;

get_s7INT (*source)

to get an 16 bit signed integer as a signed int16

int16_t get_s7INT (void *in) ;

set_s7INT (*dest, source)

writes a 16 bit signed short to a INT

void set_s7INT (void *out, int16_t in) ;

get_s7DWORD (*source)

to get a double word as unsigned number

uint32_t get_s7DWORD(void *in) ;

set_s7DWORD (*dest, source)

writes a 32bit value to a DWORD

void set_s7DWORD (void *out,int in) ;

get_s7UDINT (*source)

to get a double word as unsigned value

uint32_t get_s7UDINT(void *in) ;

set_s7UDINT (*dest, source)

writes a 32bit unsigned value to a UDINT

void set_s7UDINT (void *out, int in) ;

get_s7DINT (*source)

to get a DINT as signed long

int32_t get_s7DINT(void *in) ;

set_s7DINT (*dest, source)

writes a 32bit signed value to a DINT

void set_s7DINT (void *out,int in) ;

get_s7LWORD(*source)

to get a long word (64 bit) as uint64_t

uint64_t get_s7LWORD(void *in) ;

set_s7LWORD(*dest, source)

writes a 64bit unsigned integer to a DBL

void set_s7LWORD (void *out, uint64_t in) ;

get_s7ULINT(*source)

to get a ULINT (64 bit) as uint64_t

uint64_t get_s7ULINT(void *in) ;

set_s7ULINT(*dest, source)

writes a 64bit unsigned integer to a ULINT

void set_s7ULINT (void *out, uint64_t in) ;

get_s7LINT (*souce)

to get a LINT as a int64_t

int64_t get_s7LINT(void *in) ;

set_s7LINT(*dest, source)

writes a 64bit signed integer to a LINT

void set_s7LINT (void *out, int64_t in) ;

get_s7COUNTER (*source)

to get a BCD coded counter value [0.999]

int get_s7COUNTER (void *in) ;

set_s7COUNTER (*dest, source)

writes to a BCD coded counter value
returncode:
0 : OK

-4 : < 0 out of limits
-5 : > 999 out of limits

int set_s7COUNTER (void *out,int in) ;

2.2 Floating point functions

get_s7REAL(*source)

read a S7-real into double format

format in S7:
Bit 31 = SIGN
Bit 30..23 = 8 Bit exponent
Bit 22.. 0 = 23 Bit mantissa

double get_s7REAL(void *in) ;

set_s7REAL (*dest, source)

writes a double value to a REAL

returncode : 0 = OK;
-6 = OUT OF RANGE : > 3.402823E+38
The absolute value exeeds the upper limit, it will be limited to the upper limit.
-7 = OUT OF RANGE : < 1.175495E-38
The absolute value is smaller the lower limit, it will be set to 0.0

int set_s7REAL (void *out, double source) ;

get_s7LREAL(*source)

to get a S7-LREAL into double format

Format in S7:
Bit 63 = SIGN
Bit 62..52 = 11 Bit exponent
Bit 51.. 0 = 52 Bit mantissa

double get_s7LREAL(void *in) ;

set_s7LREAL (*dest, source)

write a double to a S7-LREAL

returncode : 0 = OK;
-8 = OUT OF RANGE : > 1.7976931348623158e+308
The absolute value exeeds the upper limit, it will be limited to the upper limit.
-9 = OUT OF RANGE : < 2.2250738585072014e-308
The absolute value is smaller the lower limit, it will be set to 0.0.

limits according to IEEE754
-1,7976931348623158e+308 to 2,2250738585072014e-308

int set_s7LREAL (void *out, double val) ;

2.3 S7-String-Formats

S7-String Format: <ssize><length>< body ...>

get_s7STRING ()

to read the Simatic S7 string "*in" into the string dest[], maximum length of the destination string is dmax. The destination string is limited by '\0'.
Therefore, the destination string needs to be at least one character longer than the <bod> of the source string inside the plc.

```
return code : >= 0      : count of read characters in dstr[]
              = -10     : error in S7: <ssize> < <length>:
                          length informations in S7 not plausible.
                          <length> bytes read
                          fetch read length with 'strlen(dest):
              = -11     : error: string *in larger as dest[]
                          dest[] is too short.
                          only dmax -1 bytes read
```

int get_s7STRING (void *in, char dstr[],int dmax);

get_s7STRING_SIZE (*source, int *length)

read Size und Lenght of the simatic string pinted to by "*in".
return code : >= 0 : Size
The string contents (= <body>) itself will not be read, only the maximum size and actual length.

If 'int *length' != NULL then the length of the actual contents
also will be returned.

int get_s7STRING_SIZE (void *in,int *length) ;

set_s7STRING ()

write the string src[] into the Simatic S7 string "*in",
maximum length of the destination string is smax.
smax has to be indentical with the data declaration of the Simatic
datablock,
i.e. : in DB: STRING[5] -> smax = 5.

S7-string format: <ssize><length>< body ...>

```
return code :  =  0  : no error
               = -12  : error, invalid length smax
                      : valid is smax = 1..254
                       <ssize> will be set to 254
```

= -13 : warning: source string to long for destination
truncated while transferring

int set_s7STRING (void *out,const char src[],int smax) ;

2.4 Datums- und Uhrzeit-Formate

Versions since 05.04.2017
- new function names and new functions
- the old function ara in the file "s7types_t_obsolete.c".

The new functions no longer contain time zone calculation.
Why ?
- The new CPUs (1200/1500) handle this by their operation system.
- Also for "classic" CPUs, there are provided, library functions which enable these calculations.
- Date and time are subject of historical and political conditions which make a correct transformation very complex, and in the most of all cases, this is nor necessary, because the time and date data rely on local time.

To enable a unic interface for all date and time functions, the **new structure 'struct s7hTime'** has been created, which, in opposite to the common Unix / Linux time structures contains broken down time like 'struct tm' and fractions of seconds as well.

Two new functions read_s7TimeStr () and strfs7Time() support the conversion between 'struct s7hTime' and ASCII- Strings.

For conversions between 'struct tm', 'struct timeval', 'struct timespec' and 'struct s7hTime' new functions have been created.

As an additional feature, the new functions provide the possibility,to transfer the time by using the formats 'struct timespec' and 'struct tm'.
The structure of the requested type needs to be generated by the user and the according pointer needs to be set.

The formerly used 'struct timeval' has been substituted by struct timespec' because timspec also supports nano seconds.
This enables a uniform set of parameters to (nearly) all functions

All functions have an additional parameter 'ns' to access nano seconds. This parameter contains the time parts smaller than a second.
Exception: get_DATE(),. set_DATE()

For all functions writing to Simatic (set_s7***) :
All pointer parameters are processed left to right.
The last non-NULL wins.

! 'struct tm' can not handle broken down seconds. Therefore 'ns' is used, or if defined, 'ts->tv_nsec'.

Where is the documentation of the common time structures ?

/usr/include/sys/time.h:

```
struct timespec
{
 time_t tv_sec;          // Seconds.
 long int tv_nsec;       // Nanoseconds
};
```

for description see: "man gmtime":

```
struct tm {
    int tm_sec;         /* seconds
    int tm_min;         /* minutes
    int tm_hour;        /* hours
    int tm_mday;        /* day of the month
    int tm_mon;         /* month (0..11)
    int tm_year;        /* year (1900 = 0 ...)
    int tm_wday;        /* day of the week (0..6) (*)
    int tm_yday;        /* day in the year (0 = 1.Jan .. 365) (*)
    int tm_isdst;       /* daylight saving time
};
```

/* ---------- time structure for convertlib -------- */

typedef struct s7hTime : a new structure for date and time representation

This structure is an additional format for all date and time convering functions of this library.
It is inspired by 'struct tm' (see: /usr/include/time.h) but includes some additional informations which may be helpful for conversion from or to strings.

On usage of the bulit in string string functions, error codes and it's position will be registered.

On usage together with S7 conversion functions, some error codes will be written, too.

```
#ifndef S7HTIME
#define S7HTIME
typedef struct s7hTime
{
  int is_abs;                 /* 1 = absolute Date and Time
                                 0 = relative time: no Year and month evaluation */
  int sign;                   /* if relative: -1,+1, on absolute: no evaluation */
  int nsec;                   /* Nano seconds  (0..999)  */
  int usec;                   /* Micro seconds (0..999 ) */
  int msec;                   /* Milli seconds ( 0..999) */
  int sec;                    /* Seconds.     [0- .... */
  int min;                    /* Minutes.     [0-59] */
  int hour;                   /* Hours.       [0-23] */
  int mday;                   /* Day.         [1-31] */
  int mon;                    /* Month.       [1-12] */
  int year;                   /* Year              */
  int wday;                   /* Day of week.         [0-6] */
  int yday;                   /* Days in year.[0-365] */
  int isdst;                  /* DST.                 [-1/0/1]*/
  int error;                  /* error code, see list of error codes */
  int error_pos;              /* found Position of the error code */

} S7hTime;

#endif
```

init_s7hTime() initialize a structure s7hTime

create and initialize a 'struct s7hTime'
- If the pointer *s is NULL, a new structure will be created and initialized.
  It is in the responsibility of the programmer, to free the allocatd memory.

- If the pointer *s ponts to an existing structure,
  it only will be initialized.

struct s7hTime *init_s7hTime (struct s7hTime *s) ;

computer sided helpful functions to swich between time formats

s7hTime_to_tm()

translate a 'struct s7hTime' to a struct 'tm'

void s7hTime_to_tm (struct tm *tm, struct s7hTime *s7T) ;

s7hTime_fr_tm()

translate a 'struct tm' to a 'struct s7hTime'
'tm' is regardes as an absolute date and time

void s7hTime_fr_tm (struct s7hTime *s7T, struct tm *tm) ;

s7hTime_to_tv()

translate a struct 's7hTime' to a struct 'timeval tv'

If s7T->is_abs == 1, (absolute time and date)
  s7T->sign will not be regarded, because it's a time since EPOCH.
If s7T->is_abs == 0, (relative time) and s7T->sign == -1,
  tv->tv_sec and tv->usec will be negative values.
  s7T->mon und s7T->year will be ignored,

void s7hTime_to_tv (struct timeval *tv, struct s7hTime *s7T) ;

s7hTime_fr_tv()

translate a 'struct timeval' to a 'struct s7hTime'

The parameter 'absolut' determins, if the result will be an absolute date and time
(= 1) or a relative time (= 0)

retval : 0 = OK
      -15 = absolut time, but negative value:

int s7hTime_fr_tv (struct s7hTime *s7T, int absolut, struct timeval *tv) ;

s7hTime_to_ts()

translate a struct 's7hTime' to a struct 'timespec ts'

If s7T->is_abs == 1, (absolute time and date)
  s7T->sign will not be regarded, because it's time since EPOCH.
If s7T->is_abs == 0, (relative time)

s7T->mon und s7T->year will be ignored,
if s7T->sign == -1,
tv->tv_sec and tv->usec will be negative values.

void s7hTime_to_ts (struct timespec *ts, struct s7hTime *s7T) ;

s7hTime_fr_ts()

translate a 'struct timespec' to a 'struct s7hTime'

The parameter 'absolut' determins, if the result will be an absolute date and time (= 1) or a relative time (= 0)

retval : 0 = OK
-10 = absolut time, but negative value

int s7hTime_fr_ts (struct s7hTime *s7T, int absolut, struct timespec *ts) ;

Relative times : time-functions

get_s7S5TIME ()

to get a S5TIME value (S7: S5t#) in milliseconds

uint32_t get_s7S5TIME (void *in) ;

get_s7S5TIME_FMT ()

changes a S5TIME (S7: S5T#..) into milliseconds
Obsolete structures may be set to NULL.
Optional values:
- der remaining nanoseconds 0..999999999
- read value into a structure "timespec *ts"
- read value into a structure "s7hTime *s7T"

uint32_t get_s7S5TIME_FMT (void *in,
struct timespec *ts, struct s7hTime *s7T);

set_s7S5TIME (*dest, zeit, dimension)

to write a time value into S5T#-format
input parameters:
in = time in milli seconds
dim = -1: automatic calculation,
else 0..3 timebase is 10ms,100ms,1s,10s

returncode:
0 : OK
-1 : < 0 out of limits
-2 : > 9990 sec out of limits
-3 = value not possible with selected time base

int set_s7S5TIME (void *out,long in, int dim) ;

get_s7TIME()

Read a Simatic-S7 TIME or TIME_OF_DAY and return the value in seconds.
returnvalue : seconds

HINT:
Both formats contain times in ms units, therefore unsing the function
get_s7DINT() also would be possible.

Optional parameters:
- the remaining fraction of seconds in nano seconds
- as a structure "timespec *ts".
- as a structure "struct s7hTime"
For not used structures the pointer needs to be NULL.

```
time_t get_s7TIME( void *in,int64_t *ns,
                   struct timespec *ts, struct s7hTime *s7T) ;
```

set_s7TIME()

Write IEC TIME or TIME_OF_DAY

HINT:
Both formats contain times in ms units, therefore using the function 'set_s7DINT()' also would be possible.

Input values:
Selection according the pointer values:
Priority is
- structure "s7hTime s7T" if NULL then
- structure "timespec *ts" if NULL then
- time_t time und int64_t ns

```
void set_s7TIME ( void *out, time_t time,int64_t ns,
                  struct timespec *ts, struct s7hTime *s7T ) ;
```

get_s7LTIME()

S7-1500 read LTIME or LTOD

Read Simatic-LTIME and LTOD, return is the value in seconds,
ts defines seconds und nanoseconds

HINT:
LTIME and LTOD are only LINT, contaning nanoseconds, They also can be handled by .get_LINT().

Optionally:
- the remaining value in nano seconds 0..999999999
- as a structure "timespec *ts".
- as a structure "s7hTime *s7T"

For not used structures the pointer needs to be NULL.

```
time_t get_s7LTIME( void *in,int64_t *ns,
                struct timespec *ts,struct s7hTime *s7T ) ;
```

set_s7LTIME()

S7-1500 read LTIME or LTOD

Write Simatic-LTIME from the value in 'time'[seconds], ts provides seconds und nanoseconds

HINT:
LTIME and LTOD are only LINT, containing nanoseconds, They also can be handled by set_LINT().

Input values:
Selection according the pointer values:
Priority is
- structure "s7hTime s7T" if NULL then
- structure "timespec *ts" if NULL then
- time_t time und int64_t ns

```
void set_s7LTIME( void *out, time_t time, int64_t ns,
                struct timespec *ts, struct s7hTime *s7T ) ;
```

Absolute Times : Date-and-Time-functions

get_s7LDT()

Read LDT (DATE_AND_LTIME) from a S7-1500

LDT is only a INT64 number, containing nanoseonds elapsed since EPOCH (see man time(7)).
Therefore, it is an absolute time. Only positive values are valid.

In this way, LDT is similar to time_t, but contains nanoseconds instead of seconds.

Because LDT is based on ns, inside the plc, it also may be used as a mighty relative time for time calculations.

This function is nearly identical to get_LTIME(),
only t->tv_sec and t->tv_nsec are controlled to be positive.
in case of negative, seconds and nanoseconds will be returend as -1.

returnvalue : seconds

Optionally:
- the remaining value in nano seconds
- as a structure "timespec *ts".
- as a structure "s7hTime *s7T"

For not used structures the pointer needs to be NULL.

```
time_t get_s7LDT( void *in, int64_t *ns,
                  struct timespec *ts, struct s7hTime *s7T) ;
```

set_s7LDT()

Write LDT (DATE_AND_LTIME) from a S7-1500

Effectively, LDT is only a INT64 number, containing nanoseonds elapsed since EPOCH
(see man time(7)).
Only positive values are valid.
In this way, LDT is similar to time_t, but contains nanoseconds instead of seconds.

Because LDT is based on ns, inside the plc, it also may be used as a mighty relative time for time calculations.

This function is identical to set_s7LTIME().

Input values:
Selection according the pointer values:
Priority is
- structure "s7hTime s7T" if NULL then

- structure "timespec *ts" if NULL then
- time_t time und int64_t ns

```
void set_s7LDT( void *out, time_t time, int64_t ns,
            struct timespec *ts, struct s7hTime *s7T ) ;
```

get_s7DATE()

Read Simatic-S7 IEC DATE and return the value in seconds sice EPOCH.

returnvalue : seconds since EPOCH
Optionally:
- as a structure "timespec *ts"
- as a structure "s7hTime *s7T"

For not used structures the pointer needs to be NULL.

```
time_t get_s7DATE ( void *in,
                struct timespec *ts, struct s7hTime *s7T) ;
```

set_s7DATE()

Write IEC DATE

'time' contains the seconds since EPOCH

Input values:
Selection according the pointer values:
Priority is
- structure "s7hTime s7T" if NULL then
- structure "timespec *ts" if NULL then
- time_t time

If the date is transfered by "s7hTime *s7T" the contents will be copied
into an internal structure and therein, time informations will be set to 0;

The original structure remains untouched.

```
void set_s7DATE ( void *out, time_t time,
               struct timespec *ts, struct s7hTime *s7T ) ;
```

get_s7DT()

Read Simatic-S7 in DATE_AND_TIME-Format (BCD)and return
the value in seconds or
-1 if an error is detected (day or month < 1)

The DATE_AND_TIME format is in BCD bytes
0: year (19)90 .. (20)89
1: Month
2: day
3: hour
4: minute
5: second
6 Millisek *10
7: Millisek , (1 ..7 = sunday .. saturday) (Hi-Nibble,Lo-Nibble)

Optionally:
- the remaining value in nano seconds
- as a structure "timespec *ts".
- as a structure "tm *tm_ptr"

For not used structures the pointer needs to be NULL.

```
time_t get_s7DT ( void *in,int64_t *ns,
                  struct timespec *ts,struct s7hTime *s7T) ;
```

set_s7DT()

Write date and time to a S7-plc

Write to Simatic-S7 in DATE_AND_TIME format (BCD)
Writes to an array of 8 bytes, beginning with the byte pointed to.

The DATE_AND_TIME format is
0: year (19)90 .. (20)89
1: Month
2: day
3: hour
4: minute
5: second
6 Millisek *10
7: Millisek , (1 ..7 = sunday .. saturday) (Hi-Nibble,Lo-Nibble)

Input values:
Selection according the pointer values:
Priority is
- structure "s7hTime s7T" if NULL then
- structure "timespec *ts" if NULL then
- time_t time und int64_t ns

```
void set_s7DT ( void *out, time_t time,int64_t ns,
                struct timespec *ts, struct s7hTime *s7T ) ;
```

get_s7DTL()

Read Simatic-S7 in DTL-format (BCD) from a S7-1200 / 1500 and
return the value in seconds
or -1 if an error is detected (day or month < 1)

The DTL format is
```
   0:  YEAR      UINT   = unsigned int16     [1970 ... = 0]
   2:  MONTH     USINT  = unsigned char      [1..12]
   3:  DAY       USINT  = unsigned char      [1..31]
   4:  WEEKDAY   USINT  = unsigned char      [1..7, 1 = sunday]
   5:  HOUR      USINT  = unsigned char      [0..23]
   6:  MINUTE    USINT  = unsigned char      [0..59]
   7:  SECOND    USINT  = unsigned char      [0..59]
   8..11 NANOSECOND UDINT  = unsigned int32  [0..999999999]
```

Optionally:
- the remaining value in nano seconds
- as a structure "timespec *ts".
- as a structure "tm *tm_ptr"

For not used structures the pointer needs to be NULL.

```
time_t get_s7DTL ( void *in, int64_t *ns,
                struct timespec *ts, struct s7hTime *s7T  ) ;
```

set_s7DTL()

Writes date and time into a S7
Write to Simatic-S7 in DTL format which is a structure of 12 bytes.

int ns : broken down seconds

Input values:
Selection according the pointer values:
Priority is
- structure "s7hTime s7T" if NULL then
- structure "timespec *ts" if NULL then
- time_t time und int64_t ns

The DTL format is
```
   0:  YEAR      UINT   = unsigned int16     [1970 ...]
   2:  MONTH     USINT  = unsigned char      [1..12]
   3:  DAY       USINT  = unsigned char      [1..31]
   4:  WEEKDAY   USINT  = unsigned char      [1..7, 1 = sunday]
   5:  HOUR      USINT  = unsigned char      [0..23]
```

6: MINUTE USINT = unsigned char [0..59]
7: SECOND USINT = unsigned char [0..59]
8..11 NANOSECOND UDINT = unsigned int32 [0..999999999]

```
void set_s7DTL (void *out, time_t  time, int64_t  ns,
          struct timespec *ts, struct s7hTime *s7T ) ;
```

2.5 Translation String <-> date and time (computer side functions)

read_s7TimeStr() : translate a string to a time format

Translation of an input string into the fomat 'struct s7hTime'.

Parameter:
in : input string
absolut : 1 = absolute time, 0 = relative time, on relative times, negative values are accepted
fmt : estimated format string
tokens:
'%<format><delim> where <format> :
Y = Year
m = Month
d = Day
H = Hour
M = Minute
S = Second
mS = Milli seconds
uS = Micro seconds
nS = Nano seconds
and <delim> a non-digit sign or end of string.

exemples:
"%d.%m.%Y %h:%M:%S.%mS.%uS.%ns" for a complete date and time
output i.e.: "03.05.2018 23:54:23.521.476.883"
"%Y-%m-%d" for a IEC date
output i.e.: "2018-03-31"
"%-%d %H:%M:%S" for a relative time with sign
output i.e.: "+2 23:45:56"
"%-%dTage %Hh %Mm %Ss" for a relative time with sign
output i.e.: "+56Tage 23h 45m 56s"

conditions for the creration of the format string;
If the format string contains seconds and also fractions of seconds and also hours and minutes, the delimiters of hours, seconds and minutes need to be different to the delimites of the fractions of seconds.

The sequence for times have to be downgoing:
hour,[minutes[,seconds[,milli,[micro[,nano]]]]]

procedure of the interpretation:
If an **absolute date_and_time** is expected (parameter absolut = 1) the function works left to right:
<date> <hour> <minutes> <seconds> <fractions_of_seconds>
<sign> is not accepted in a fmt string of an absolute date_and_time.
<date> is mandatory, the fields for %Y %m %d may be in any order.
<hour> <minutes> <seconds> <fractions_of_seconds> are optional,
but if one exists in fmt, the next left sided unit needs to exist,too.

i.e: if %mS then %S is mandatory
if %S then %M is mandatory
if %M then %H is mandatory.

The values for not existing fields are preset by 0.

For "date" formats:
Any sequence will be accepted, as long as the format string contains all 3 parts.
A date only will be accepted, if all 3 parts day, month, year exist in the fmt string and inside the scanned input string as well.

If a **relative time** is expected (parameter absolut = 0) years and months are not allowed in the fmt string, Days are accepted:

<sign> <day> <hour> <minutes> <seconds> <fractions_of_seconds>
The program first tries to find the position of the seconds.
If found, tries to find the <fractions_of_seconds>
Then the program tries to find the minutes, hours, days.
Coming from seconds: if the left sided next unit exists, then the next one also needs to exist.

If the input string only contains one number, it will be regarded as seconds.

output format:
The structure 'struct s7hTime *s7T' will be filled.

Return value = 0 : OK,
else <0 = format error
exact idetification in s7T->error
s7T->error_pos

```
int read_s7TimeStr (const char * in, const int absolut, const char *fmt,
                struct s7hTime *s7T) ;
```

Parameters:

const char * in	= input string
const int absolut	= 0 : relative tim, 1: absolute time expected
const char *fmt	= expected format
struct s7hTime *s7T	= structure for the result

strfs7Time() : translate a date-and-time format to a string

Formated output of date and time formats, absolute and relative time.
The function strfs7Time() works like strftime(), only has a subset of the format tokens but also contains some new ones.

```
%+ ,%-   '+' Sign is always displayed, '-' sign is displayed on relativ an negative time
%Y           year including centuries (i.e 1985 )
%y           year not including centuries (i.e 85 )
%m
%d
%H
%M
%S
%mS  : 0..999 ms
%uS  : 0..999 us
%nS  : 0..999 ms
```

Output:
dstr, filled with formated output

errors:
The number of a possible error is written in to struct s7hTime s7T-error
and may be analized behind the call of this function.
0 = no error.
< 0 errors, see char * s7hTime_errTxt (int error)

```
char* strfs7Time(char dstr[], size_t dstr_len,  const char fmt[], \
     struct s7hTime *s7T) ;
```

Parameter:

const char dstr[]	= output string
size_t dstr_len	= maximale lenght (= size) of output string
const char *fmt	= format for output string
struct s7hTime *s7T	= input structure

2.6 help functions

Translate the error numbers to a text

s7hTime_errTxt() : error number to text german version

generiert aus Fehlernummern einen Fehlertext
verwendbar hinter
read_s7TimeStr ()
s7hTime_fr_tv ()
etc.

error codes :

-1 : set_s7S5TIME () : Wert < 0. ausser Bereich
-2 : set_s7S5TIME () : Wert > 9990sek : ausser Bereich
-3 : set_s7S5TIME () : Wert mit gewählter Zeitbasis nicht darstellbar
-4 : set_s7COUNTER () : Wert < 0. ausser Bereich
-5 : set_s7COUNTER () : Wert > 999 : ausser Bereich
-6 : set_s7REAL () : Wert > 3.402823E+38 : ausser Bereich
-7 : set_s7REAL () : Wert < 1.175495E-38 : ausser Bereich
-8 : set_s7LREAL () : Wert > 1.7976931348623158e+308 : ausser Bereich
-9 : set_s7LREAL () : Wert < 2.2250738585072014e-308 : ausser Bereich
-10 : get_s7STRING () : Längenangaben in S7 nicht plausibel <length> Bytes gelesen
-11 : get_s7STRING () : Zielstring zu kurz, nur dmax -1 Bytes gelesen
-12 : set_s7STRING () : Fehler, ungültige Länge smax
-13 : set_s7STRING () : Quelle zu lang für Zielstring,wurde gekürzt übertragen

-15 : s7hTime_fr_tv () : Absolute Zeit aber negativen Wert gelesen

-20 : read_s7TimeStr () : Absolute Zeit (Datum) aber Jahr-Feld fehlt im Format
-21 : read_s7TimeStr () : Absolute Zeit (Datum) aber Monat-Feld fehlt im Format
-22 : read_s7TimeStr () : Absolute Zeit (Datum) aber Tag-Feld fehlt im Format
-23 : read_s7TimeStr () : Absolute Zeit (Datum) aber Vorzeichenfeld im Format
-24 : read_s7TimeStr () : Absolute Zeit Minuten definiert aber Stunden fehlt im Format
-25 : read_s7TimeStr () : Absolute Zeit Sekunden definiert aber Minuten fehlt im Format
-26 : read_s7TimeStr () : Absolute Zeit ms definiert aber Sekunden fehlt im Format
-27 : read_s7TimeStr () : Absolute Zeit us definiert aber Millisekunden fehlt im Format
-28 : read_s7TimeStr () : Absolute Zeit ns definiert aber Mikrosekunden fehlt im Format
-29 : read_s7TimeStr () : Absolute Zeit Jahr fehlt in Eingabestring
-30 : read_s7TimeStr () : Absolute Zeit Monat fehlt in Eingabestring
-31 : read_s7TimeStr () : Absolute Zeit Tag fehlt in Eingabestring
-32 : read_s7TimeStr () : Relative Zeit aber Jahr-Feld im Format
-33 : read_s7TimeStr () : Relative Zeit aber Monat-Feld im Format
-34 : read_s7TimeStr () : Datum-Eingabe erwartet, aber in Eingabestring nicht enthalten.

-40 : strfs7Time () : Absolute Zeit aber kein vollständiges Datums-Format.
-41 : strfs7Time () : Relative Zeit aber format beinhaltet Datums-Format.
-42 : strfs7Time () : Vorzeichen in Formatstring nicht auf erster Position.
-43 : strfs7Time () : Unvollständiges Sekundenformat.

-44 : strfs7Time () : Reihenfolge der Zeit-Tokens nicht absteigend.

char * s7hTime_errTxt (int error) ;

s7hTime_errTxt_En() : error number to text Englisch Version

For the time functions : English version
generates an error text from an error number
usable behind
read_s7TimeStr ()
s7hTime_fr_tv ()
etc.

error codes :

-1 : set_s7S5TIME () : value < 0. violates limits
-2 : set_s7S5TIME () : value > 9990 sec : violates limits
-3 : set_s7S5TIME () : value nor processible with the selected time base (0..3)
-4 : set_s7COUNTER () : value < 0. violates limits
-5 : set_s7COUNTER () : value > 999 : violates limits
-6 : set_s7REAL () : value > 3.402823E+38 : violates limits
-7 : set_s7REAL () : value < 1.175495E-38 : violates limits
-8 : set_s7LREAL () : value > 1.7976931348623158e+308 : violates limits
-9 : set_s7LREAL () : value < 2.2250738585072014e-308 : violates limits
-10 : get_s7STRING () :length informations in S7 not plausible, <length> bytes read
-11 : get_s7STRING () : dest[] is too short.only dmax -1 bytes read
-12 : set_s7STRING () : error: invalid length smax
-13 : set_s7STRING () : src[] too long for destination, was truncated

-15 : s7hTime_fr_tv () : absolute date and time aber negativen value gelesen

-20 : read_s7TimeStr () : absolute date'n time but absence of year field in format
-21 : read_s7TimeStr () : absolute date'n time but absence of month field in format
-22 : read_s7TimeStr () : absolute date'n time but absence of day field in format
-23 : read_s7TimeStr () : absolute date'n time but sigh field in format
-24 : read_s7TimeStr () : absolute date'n time minutes defined but hours are absent in format
-25 : read_s7TimeStr () : absolute date'n time seconds defined but minutes are absent in format
-26 : read_s7TimeStr () : absolute date'n time ms defined but seconds are absent in format
-27 : read_s7TimeStr () : absolute date'n time us defined but ms are absent in format
-28 : read_s7TimeStr () : absolute date'n time ns defined but us are absent in format
-29 : read_s7TimeStr () : absolute date'n time year is absent in input string
-30 : read_s7TimeStr () : absolute date'n time month is absent in input string
-31 : read_s7TimeStr () : absolute date'n time day is absent in input string
-32 : read_s7TimeStr () : relative time but year field in format
-33 : read_s7TimeStr () : relative time but month field in format
-34 : read_s7TimeStr () : estimated date input but not provided by input string

-40 : strfs7Time () : absolute date'n time but format does not contain a complete date definition
-41 : strfs7Time () : relative time but format contains date format

-42 : strfs7Time () : sing in format string but not at first position
-43 : strfs7Time () : unclomplete format of seconds
-44 : strfs7Time () : no down going sequence of time fields in format

char * s7hTime_errTxt_En (int error) ;

For testing of read_s7TimeStr () etc **: print_s7hTime (struct s7hTime *p)**

Teh function read_s7TimeStr () was designed to be flexible an unsiversal. But the also enables a lot of possible error combinations.
print_s7hTime () is designed to help to operate with read_s7TimeStr () in a right way.

3. s5types.c, s5types.h old FUNCTIONS

```
/* ==========================================================================

   Funktionen Datenaustausch von Simatic S5/S7-DBs mit Unix/Linux

   functions for data exchange with a Simatic S5/S7 and Linux

* ==========================================================================
   (c) Heisch Automatisierungstechnik, Werner Heisch

       http://sites.inka.de/heisch
       http://www.heisch-automation.de

 !!! Das Array, das fuer den Transfer von oder zu der S5 vorgesehen ist,
      sollte als unsigned short definiert werden.
      Dadurch bleibt der Zusammenhang Array<-> DB erhalten, was die
      Fehlersuche entscheidend erleichtert.

      Die folgenden functions funktionieren aber auch mit unsigned char,
      diese Darstellung wird für die Kommunikation mit der S7 empfohlen.

      ------------

      The array, which is used for data transfer with Simatic S5, should be
      defined as an array of unsigned short. This helps to find errors.

      The functions also work with arrays of unsigned char, which is
      recommended for a link to Simatic S7.


*/
```

3.1 Integer and BCD Formats

```
/* ============  S5types_i.c /.h  ==========================================

   Funktionen Datenaustausch von Simatic S5-DBs mit Unix/Linux
   functions for data transfer beween Simatic S5 / S7 and Linux
         -------------- Integer-part -----

   S5:  KH,KF,KD,KT,KZ -----
   S7:  WORD,INT,DWORD,DINT,S5TIME,C

* ==========================================================================

*/


#ifndef MATH_H
  #include <math.h>
#endif

/* --------------- get_dd () ------------------------- */
/* Doppelwort als unsigned long einlesen
   importiert eine unsigned Zahl
   ---
   to get a double word as unsigned number

*/

unsigned long get_dd(void *in)
;
/* --------------- get_dd_kf () ------------------------- */
/* Doppelwort als long integer einlesen
   exportiert eine signed-Zahl
   ---
   to get a double word as signed long
*/

long get_dd_kf(void *in)
;
/* --------------- get_dw () ------------------------- */
/*
 importiert eine unsigned-Zahl
 ---
 to get a word (16bit) as a unsigned int.
*/
int get_dw (void *in)
;
/* --------------- get_dw_kf () ------------------------- */
/*
 importiert eine signed-Zahl
 ---
 to get a word (16Bit) as a signed int
 */
int get_dw_kf (void *in)
;
/* --------------- get_kt () ------------------------- */
```

```
/*
importiert einen Zeitwert (S5: KT,S7: S5T#..) in Millisekunden
---
to get a time value ( S5: KT, S7: S5t# ) im milliseconds

*/
long get_kt (void *in)
;
/* --------------- get_kz () ------------------------- */
/*
 importiert einen Zaehlerwert
 ---
 to get a counter value
*/
int get_kz (void *in)
;
/* --------------- get_dl () ------------------------- */
/*
S5: importiert den linken Teil eines DW (Simatic S5 only)
---
S5: to get the left part of a DW
*/
int get_dl (void *in)
;
/* --------------- get_dr () ------------------------- */
/*
S5: importiert den rechten Teil eines DW  (Simatic S5 only)
---
S5: to get the right part of a DW
*/
int get_dr (void *in)
;
/* *************** Transfers in Richtung SPS ******************** */
/*                 transfers in direction to plc                  */

/* --------------- set_dw (*ziel,quelle) ------------------------- */
/* Schreiben eines 16-Bit-KF-Werts in ein DW / DBW
   ---
   writes a signed or unsigned short to a DW / DBW
*/

int set_dw (void *out,int in)
;
/* --------------- set_dd (*ziel,quelle) ------------------------- */
/* Schreiben eines 32-Bit-KF-Werts zu einem DD / DBD
   ---
   writes a 32bit value to a DD / DBD
*/
int set_dd (void *out,int in)
;
/* --------------- set_kz (*ziel, quelle) ------------------------- */
/* Schreiben eines Zaehlerwerts
 bei Wert-Ueberlauf: Rueckgabe = -1; sonst = 0;
 ---
 writes to a counter value
 on overflow: returncode = -1 else return 0
*/
```

```
int set_kz (void *out,int in)
;
/* --------------- kt_to_short (*ziel, quelle, dimension) --------------------
*/
/* Schreiben eines Zeitwerts in BCD-codierte short-Zahl
   ( Bytes sind noch nicht gedreht, set_dw anschliessend erforderlich)
   Eingabeparameter:
     in  = Zeitwert in Millisekunden
     dim = -1: automatisch berechnen,
                 sonst 0..3 Zeitbasis ist 10ms,100ms,s,10s
   bei Wert-Ueberlauf oder dim-Fehler: Rueckgabe = -1; sonst zeitwert in
   KT-Format / S5t#- Format;
   ---
   writes a time value  to a BSC coded short
   ( bytes are not rotated, set_dw is necessary, after this fuction )
   Input parameters:
     in =  time in milli seconds
     dim = -1: automatic calculation,
               else 0..3 timebase is 10ms,100ms,1s,10s
   on value overflow or dim error : return code = -1 else the value
   in KT format / S5T# format is returned
*/

short kt_to_short (long in, int dim)
;
/* --------------- set_kt (*ziel, quelle, dimension) -------------------- */
/* Schreiben eines Zeitwerts in KT / S5t#-Format
   Eingabeparameter:
     in  = Zeitwert in Millisekunden
     dim = -1: automatisch berechnen,
                 sonst 0..3 Zeitbasis ist 10ms,100ms,s,10s

   bei Wert-Ueberlauf: Rueckgabe = -1; sonst = 0;
   ---
   to write a time value in KT / S5t#-format
   input parameters:
     in  =  time in milli seconds
     dim = -1: automatic calculation,
               else 0..3 timebase is 10ms,100ms,1s,10s
   on overflow: returncode = -1; else = 0;

*/

int set_kt (void *out,long in, int dim)
;
```

3.2 Floating point Formats

```
/* ============== S5types_fp.c / .h   ====================================

   Funktionen Datenaustausch von Simatic S5/S7-DBs mit Unix/Linux
   functions for data transfer beween Simatic S5 / S7 and Linux
    -------------- Floating point part -----

   S5:  KG  (Simatic S5 floating point)
   S7:  REAL


   Version 04.01.2002 (Comments) code: 02.08.2000 HW
* ==========================================================================
*/


#ifndef _UNISTD_H
  #include <unistd.h>
#endif
#ifndef MATH_H
  #include <math.h>
#endif

/* --------------- get_real() ------------------------- */
/*
 S7-REAL-Zahl als double einlesen
 ---
 to get a S7-real in double format
*/
/* returncode : 0 = ohne Fehler, -1 = fehler */
/* returncode : 0 = no fault, -1 = with fault */

int get_real(double *retval,void *in)
;
/* --------------- get_s5kg() ------------------------- */
/*
 S5-REAL-Zahl als double einlesen
 ---
 to get a S5-real as double format
*/
/* returncode : 0 = ohne Fehler, -1 = fehler */
/* returncode : 0 = no fault, -1 = with fault */

int get_s5kg(double *retval,void *in)
;
/* *************** Transfers in Richtung SPS ******************** */
/*                 transfers in direction to plc                  */

/* --------------- set_real (*ziel,quelle) ------------------------- */
/* Schreiben einer double zu einer S7-REAL-Zahl
   ---
   write a double to a S7-real

returncode : 0 = OK; -1 = OUT OF RANGE
*/
```

```
int set_real (void *out, double val)
;
/* --------------- set_s5kg (*ziel,quelle) ------------------------- */
/* Schreiben einer double zu einer S5-KG-Zahl
   ---
   write a double to a S5-KG-value

returncode : 0 = OK; -1 = OUT OF RANGE

*/

int set_s5kg (void *out, double val)
;
```

3.3 Date and Time Formats

```
/* ============== S5types_t.c / .h   ====================================

   Funktionen Datenaustausch von Simatic S5/S7-DBs mit Unix/Linux
   functions for data transfer beween Simatic S5 / S7 and Linux
    -------------- DATE and TIME -Functions -----

   S5:  RealTimeClock-Format of S5-095U, S5-115U,  -----
   S7:  (IEC)DATE, (IEC)TIME , DATE_AND_TIME(BCD)

   Stand / Version
   02.08.2002 HW DATE_AND_TIME eingebaut
   02.01.2002 HW
   01.01.2002 HW
   22.06.2001 HW
   19.06.2001 HW
   27.02.2001 HW
   07.05.2000 HW

* ========================================================================
  (c) Heisch Automatisierungstechnik, Werner Heisch
      http://sites.inka.de/heisch
      http://www.heisch-automation.de



   PROBLEM: die Simaticen laufen normalerweise in Ortszeit,
  ( sie werden von Step 7(TM) aus gestellt ), stellen aber selbständig
   keine Sommer-Winterzeit um.
   (Der Algorithmus ist schliesslich keine Naturkonstante sondern
   abhaengig von politischen Vorgaben. )
   Es ist folglich nicht zu erkennen, mit welcher Uhrzeit
   (UTC, lokale Zeit,lokale Sommerzeit) die Simatic wirklich laeuft.
   Das muss bei Bedarf der Anwender dieser function selbst entscheiden
   und ggf. korrigierend eingreifen.
   Die unten stehende function liefert den Zeitwert so zurueck,
   wie er aus der Simatic gelesen wurde. Dies wird in der Regel die aktuelle
   lokale Uhrzeit sein.

   Die rückgelieferten Zeitwerte  time_t get_iec_dateandtime() oder das
   structure-Element tv->tv_sec beinhalten beide jeweils die Zeit in Sekunden
   seit EPOCH, aber (vermutlich) als lokale (Sommer ?? )- Zeit.
   Die Rueckgabewerte sind also in einem Format(time_t, struct timeval) ,
   in dem normalerweise der Zeitbezug EPOCH und UCT ist.

   Die C-functions, die timeval tv weiterverarbeiten ( z.B. gmtime(),
   localtime() erwarten eine Zeit auf UTC basierend.

   Wenn davon ausgegangen wird, dass die Simatic-Uhr in der lokalen
   nicht-Sommerzeit laeuft, dann kann die Zeit einfach auf UTC korrigiert
   werden:
   Fuer Laender oestlich von London(Greenwich) : Stunden abziehen
   Fuer Laender westlich von London(Greenwich) : Stunden dazaehlen

   Beispiel: Simatic S7 steht in Düsseldorf -> Deutschland -> 1 Stunde östlich
     (time_t) uct_time =  get_iec_dateandtime( .. ) - 3600;
```

```
Beispiel: Simatic S7 steht in Havanna -> Kuba -> 5 Stunden westlich
  (time_t) uct_time =  get_iec_dateandtime( .. ) + 5 * 3600;

Wenn aber zum Beispiel eine Zeitsynchronisation von einem Leitsystem aus
erfolgt, wird's schwieriger, es muß dann bekannt sein ob die Simatic-Uhr
entsprechend Sommer-/Winterzeit umgestellt wird und entsprechend
zurückkorrigiert werden muß.

Die Routinen zum Synchronisieren der Zeit der Simatic sind hinsichtlich
Zeitzone und Sommer-Zeit-Umschaltung flexibel. Dies wird über den
Parameter local gesteuert.
local = 0  :  Simatic wird mit UTC synchronisiert
      = 1  :  Simatic wird mit lokaler Zeit ohne Beruecksichtigung der
              lokalen Sommerzeit synchronisiert
      = 2  :  Simatic wird mit lokaler Zeit unter Beruecksichtigung der
              lokalen Sommerzeit synchronisiert
      sonst: <zahl> != 0,1  Korrekturfaktor zu UTC: ohne Beruecksichtigung
              der lokalen Sommerzeit
              Beispiel: Simatic in Duesseldorf soll OHNE
              Sommerzeitumschaltung betrieben werden:
              Duesseldorf ist eine Stunde vor Greenwich -> local = 3600


----

PROBLEM: The simatic processors normaly run in local time ( the are set by
Step 7(TM) but do not change automaticly to daylight saving time.
( The algorithm depends from political decisions )
Therfore it is impossible to know, in which time the the Simatic runs really
and it is impossible to correct the time automaticly.
The function below returns the time, as it is read from the Simatic.
Normally it is the local time ... ?
The returnvalues are in a format (time_t, struct timeval) which normal
context is EPOCH and UTC.
The functions, for example, wich handle timeval-structures, expect
times based on UTC.

If the Simatic runs in local time but not in  daylight saving time the
correction is rather easy to do:
For countries in the east of London (greenwich) subtract hours.
For countries in the west of London (greenwich) subtract hours.

example: Simatic S7 is in Duesseldorf -> Germany -> 1 hour in the east
  (time_t) uct_time =  get_iec_dateandtime( .. ) - 3600;

( If I use the same example as above, George W. will kill me :-)
example: Simatic S7 is situated in Miami -> USA -> 5 hours in the west
  (time_t) uct_time =  get_iec_dateandtime( .. ) + 5 * 3600;

But: wenn the time is synchronized bei a process control system, it is more
difficult: It has to be known, if the time is changed according to
daylight saving time an eventually a correction is needed.


The functions for synchronizing the simatic are flexible, they are
controled by the parameter "local".
local = 0  :  Simatic will be synchronized with UTC
```

```
         = 1  :  Simatic will be synchronized with local time without
                 regarding the daylight saving time
         = 2  :  Simatic will be synchronized with local time regarding
                 the daylight saving time
         else:  <zahl> corretion value according to UTC: without regarding
                 any Daylight saving time
                 i.e.: Simatic is in Duesseldorf (Germany) and shall not
                 regard the daylight saving time:
                 Duesseldorf is one hour before Greenwich -> local = 3600


*/

#ifndef _SYS_TIME_H
  #include <sys/time.h>
#endif
#ifndef _SYS_TIMEB_H
  #include <sys/timeb.h>
#endif
#ifndef _UNISTD_H
  #include <unistd.h>
#endif
#ifndef _STDLIB_H
  #include <stdlib.h>
#endif





/* --------------- get_iec_dateandtime() ------------------------- */
/* IEC DATE und IEC TIME einlesen
   (in Simatic: 2 Variablen: IEC_DATE und IEC_TIME)
   export als  returnwert die Sekunden in EPOCH (Calendar)
   und ueber time_val *tv, die genaue Zeit Sekunden,Millisekunden *1000

   Parameter
   local : 0 : Simatic läuft in UTC  Coordinated Universal Time ( = GMT )
           1 : Simatic läuft in local time immer ohne Sommerzeit
           2 : Simatic läuft in local time mit Berücksichtigung der
               Sommerzeit

           sonst : direkter Offset ( Sek) zu UTC ohne Berücksichtigung
               der Sommerzeit

    BEMERKUNG: für Fall 2 kann für die letzte Stunde Sommerzeit natürlich
               nicht entschieden werden, ob es die letzte Stunde Somerzeit
               oder bereits die erste Stunde Winterzeit ist !

       /usr/include/sys/time.h:

       struct timeval {
               long tv_sec;        /* seconds
               long tv_usec;  /* microseconds
       };
```

```
   ----

   Read Simatic-S7 IEC DATE and IEC-TIME
   (in Simatic: 2 variables: IEC_DATE und IEC_TIME)
   and return the value in seconds sice EPOCH. (Calendar)
   returnvalue : seconds, better use the structure timeval *tv, it contains
   the time in seconds and milliseconds *1000

   Parameter
   local : 0 : Simatic runs in UTC  Coordinated Universal Time ( = GMT )
           1 : Simatic runs in local time ignoring daylight saving time
           2 : Simatic runs in local time regarding daylight saving time
           else :
           direct Offset ( sec) to UTC without regarding daylight saving time

   NOTE: in case of 2 ( local time including DST) we can not decide
         correctly for the last hour in DST-phase !!

       /usr/include/sys/time.h:

       struct timeval {
               long tv_sec;        /* seconds
               long tv_usec;  /* microseconds
       };


*/


time_t get_iec_dateandtime( void *in, struct timeval *tv,int local)
;




/* --------------- get_iec_date() ------------------------- */
/* IEC DATE einlesen und
   export als  returnwert die Sekunden in EPOCH

   Da nur das Datum übermittelt wird und deshalb keine Informationen über die
   Uhrzeit vorliegen, ist eine Zeitzonen-Korrektur nicht möglich.
   Das Datum wird weitergegeben wie es ist.

   ----

   Read Simatic-S7 IEC DATE and return the value in seconds sice EPOCH.
   returnvalue : seconds since EPOCH

   because there is no time-of-day information it is impossible to
   execute a timezone correction. Therfore this date will not be changed.

*/

time_t get_iec_date( void *in, struct timeval *tv)
;
```

```
/* --------------- get_iec_time() ------------------------- */
/* IEC TiME einlesen
   returnwert in Sekunden, tv liefert Sekunden und Mikrosekunden

       /usr/include/sys/time.h:

       struct timeval {
               long tv_sec;        /* seconds
               long tv_usec;  /* microseconds
       };


   ----

   Read Simatic-S7 IEC TIME and return the value in seconds,
   tv returns seconds und useconds

       /usr/include/sys/time.h:

       struct timeval {
               long tv_sec;        /* seconds
               long tv_usec;  /* microseconds
       };


*/

unsigned long get_iec_time( void *in, struct timeval *tv)
;




/* --------------- get_s7_dateandtime() ------------------------- */
/* Datum-Uhrzeit aus S7 im DATE_AND_TIME-Format (BCD) einlesen
   returnwert in Sekunden, tv liefert Sekunden und Mikrosekunden  (UCT)
              -1 falls ein Fehler erkannt wurde (Tag,Monat < 1)

   Parameter
   local : 0 : Simatic läuft in UTC  Coordinated Universal Time ( = GMT )
           1 : Simatic läuft in local time immer ohne Sommerzeit
           2 : Simatic läuft in local time mit Berücksichtigung der
               Sommerzeit

           sonst :
            direkter Offset ( Sek) zu UTC ohne Berücksichtigung der Sommerzeit

    BEMERKUNG: für Fall 2 kann für die letzte Stunde Sommerzeit natürlich
               nicht entschieden werden, ob es die letzte Stunde Sommerzeit oder
               bereits die erste Stunde Winterzeit ist !

   Das DATE_AND_TIME Format ist
    0:  Jahr (2stellig) (19)90 .. (20)89
    1:  Monat  (01.12).
```

```
 2:  Tag    (01..31)
 3:  Stunde (00..23)
 4:  Minute (00.59)
 5:  Sekunde (00.59)
 6   Millisek (2 Höherwertige Stellen 00 .. 99)
 7:  Millisek , ( 1 ..7 = sunday .. saturday )    (Hi-Nibble,Lo-Nibble)

    /usr/include/sys/time.h:

    struct timeval {
            long tv_sec;    /*    seconds
            long tv_usec;  /* microseconds
    };

    struct timeb {
            time_t time;
            unsigned short millitm;
            short timezone;
           short dstflag;
    }; <<<<<<<<<<<< see man ftime(3)


----

Read Simatic-S7 in DATE_AND_TIME-Format (BCD)and return the value in seconds,
tv returns seconds und useconds (UCT)
         -1 if an error is detected ( day or month < 1)

Parameter
local : 0 : Simatic runs in UTC  Coordinated Universal Time ( = GMT )
        1 : Simatic runs in local time ignoring daylight saving time (DST)
        2 : Simatic runs in local time regarding daylight saving time (DST)
        else :
         direct Offset ( sec) to UTC without regarding daylight saving time

NOTE: in case of 2 ( local time including DST) we can not decide correctly
      for the last hour in DST-phase !!


The DATE_AND_TIME format is
 0:  year
 1:  Month
 2:  day
 3:  hour
 4:  minute
 5:  second
 6   Millisek *10
 7:  Millisek , ( 1 ..7 = sunday .. saturday )    (Hi-Nibble,Lo-Nibble)

    /usr/include/sys/time.h:

    struct timeval {
            long tv_sec;        /* seconds
            long tv_usec;  /* microseconds
    };

    struct timeb {
```

```
                time_t time;
                unsigned short millitm;
                short timezone;
               short dstflag;
        }; <<<<<<<<<<<< see man ftime(3)



*/
```

time_t get_s7_dateandtime (void *in, struct timeval *tv,int local)
;

```
/* --------------- set_s7_dateandtime() ------------------------- */
/* Simatic S7 DATE_AND_TIME format schreiben
   schreibt in ein Array, auf das *out zeigt

   Das DATE_AND_TIME Format ist
    0:  Jahr (2stellig) (19)90 .. (20)89
    1:  Monat  (01.12).
    2:  Tag    (01..31)
    3:  Stunde (00..23)
    4:  Minute (00.59)
    5:  Sekunde (00.59)
    6   Millisek (2 Höherwertige Stellen 00 .. 99)
    7:  Millisek , ( 1 ..7 = sunday .. saturday )    (Hi-Nibble,Lo-Nibble)

   local : 0 : Resultierende Zeit ist UTC  Coordinated Universal Time ( = GMT )
           1 : Resultierende Zeit ist local time mit Berücksichtigung der
               Sommerzeit
           sonst :
             direkter Offset ( Sek) zu UTC ohne Berücksichtigung der Sommerzeit

       /usr/include/sys/time.h:

       struct timeval {
               long tv_sec;        /* seconds
               long tv_usec;  /* microseconds
       };


   ----

   Write to Simatic-S7  in DATE_AND_TIME format (BCD)
   Writes to an array of 8 bytes, beginning with the byte pointed to.

   The DATE_AND_TIME format is
    0:  year
    1:  Month
    2:  day
    3:  hour
    4:  minute
    5:  second
    6   Millisek *10
    7:  Millisek , ( 1 ..7 = sunday .. saturday )    (Hi-Nibble,Lo-Nibble)


   local : 0 : resulting time is UTC  Coordinated Universal Time ( = GMT )
           2 : resulting time is local time regarding daylight saving time
           else :
             direct Offset ( sec) to UTC without regarding daylight saving time

   return 0, if OK


       /usr/include/sys/time.h:

       struct timeval {
               long tv_sec;        /* seconds
               long tv_usec;  /* microseconds
```

```
        };

*/

int set_s7_dateandtime (void *out,int local)
;
```

```
/* --------------- get_95u_time() ------------------------- */
/* Zeit aus S5-95U einlesen
   returnwert in Sekunden

        ---     Wochentag ( 1 ..7 = Sonntag .. Samstag )
       tag,     Monat
       jahr,    Stunde ( Bit 7 = AM/PM
       Minute,  Sekunde
   ----

   Read Simatic-S5 95U  TIME and return the value in seconds,
   dow returns day of the week

   The time is read
    0:  ---      day_of_week ( 1 ..7 = sunday .. saturday )
    1:  day,     month
    2:  year     hour   ( bit 7 is AM/PM)
    3:  minute   second

*/

time_t get_95U_time( void *in, int *dow)
;



/* --------------- sync_95u_time() ------------------------- */
/* Zeit in S5-95U / 115U synchronisieren
   Diese function ist zum Synchronisieren der Uhr in
   - S5-95U
   - S5 115 U ( CPU943 und 944 mit 2 seriellen Schnittstellen )
   - S5 115 U  CPU945

   schreibt in ein Array, auf das *out zeigt
   ampm == 0 : Stunden 0..23
        != 0 : Stunden 1..12 AM, 1..12 PM
   local : 0 : Resultierende Zeit ist UTC  Coordinated Universal Time ( = GMT )
           1 : Resultierende Zeit ist local time mit Berücksichtigung der
               Sommerzeit
           sonst :
             direkter Offset ( Sek) zu UTC ohne Berücksichtigung der Sommerzeit

   return 0, falls OK

   Das Zeit-Array für die Simatic
        ---     Wochentag ( 1 ..7 = Sonntag .. Samstag )
       tag,     Monat
       jahr,    Stunde ( mit 7 = AM/PM
       Minute,  Sekunde

   ----

   synchronize Simatic-S5 95U / 115U TIME

   This function synchronizes the clock in
   - S5-95U
   - S5 115 U ( CPU943 und 944 with 2 serial ports )
```

```
   - S5 115 U  CPU945

   writes to an array to which points *out
   ampm == 0 : hours 0..23
        != 0 : hours 1..12 AM, 1..12 PM

   local : 0 : resulting time is UTC  Coordinated Universal Time ( = GMT )
           1 : resulting time is local time regarding daylight saving time
           else :
           direct Offset ( sec) to UTC without regarding daylight saving time

   return 0, if OK

   The time array for Simatic
    0:  ---      day_of_week ( 1 ..7 = sunday .. saturday )
    1:  day      month
    2:  year     hour   ( bit 7 is AM/PM )
    3:  minute   second

*/
```

int sync_95U_time (void *out, int ampm,int local)
;

3.4. String Formats

```
/* ============  S5types_c.c /.h  ==========================================

   Funktionen Datenaustausch von Simatic S5-DBs mit Unix/Linux
   functions for data transfer beween Simatic S5 / S7 and Linux
         -------------- Character-part -----

   S5:  KC -----
   S7:  STRING

* ==========================================================================

*/


/* --------------- get_S7string () ------------------------- */
/*  das Simatic-S7-STRING "*in" in ein string dest[] einlesen,
    maximale Länge des Zielstrings = dmax;

    return code : >= 0 : Anzahl der gelesenene Zeichen in dstr[]
                : == -1 ; Fehler

   ---
   to get the Simatic S7 string "*in" into the string dest[],
   maximum length of the destination string is dmax.

   return code : >= 0 :count of read characters in dstr[]
                : == -1 ; Fehler

*/

int get_S7string(void *in,char dstr[],int dmax)
;

/* --------------- set_S7string () ------------------------- */
/*  das String src[] in das Simatic-S7-STRING "*out" schreiben,
    maximale Länge des Zielstrings = smax;
    smax muß mit der Datendeklaration im S7-DB übereinstimmen,
    z.B: in DB: STRING[5]  -> smax = 5.

    return code : == 0  : kein Fehler
                : == -1 : Fehler, ungültige Länge smax:
                          gültige smax = 1.. 254
                  == +1 : Warnung: Quelle zu lang für Zielstring,
                          wurde gekürzt übertragen.

   ---

   write the string src[] into the Simatic S7 string "*in",
   maximum length of the destination string is smax.
   smax has to be indentical with the data declaration of the Simatic
   datablock,
   i.e. : in DB: STRING[5]   -> smax = 5.

   return code : ==  0  : no error
```

```
               : == -1  : error, invalid length smax
                        : valid is smax = 1..254
               : == +1  : warning: source string to long for destination
                          truncated while transferring

*/

int set_S7string(void *out,const char src[],int smax)
;
```

3.5 Help functions Strings

```
/* ******************************************************

   String-funktionen als Hilfsfunktionen für convert_lib

   String functions for convert_lib

Stand: 16/11/1998 HW ff
       19.02.2001 HW
       10.09.2002 HW
       18.06.2007 HW Anpassung an Suse 10.1  gcc 4.1.0
       02.04.2016 HW Erweiterung um %uS

 ****************************************************** */


#ifndef _STRING_H
  #include <string.h>
#endif

#ifndef _CTYPE_H
  #include <ctype.h>
#endif

#ifndef _STDLIB_H
  #include <stdlib.h>
#endif

#ifndef _TIME_H
  #include <time.h>
#endif

#ifndef _UNISTD_H
  #include <unistd.h>
#endif


/* --------- strftime_ms() ----------------------------- */
/*
  Funktion wie strftime(), bei Bedarf mit Millisekunden oder Mikrosekunden,
  wenn statt des Format-Zeichens %S  die Sequenz %mS (Millisekunden) oder
  %us (Mikrosekunden) geschrieben wird.
  Bei %mS werden die Sekunden statt 99 als 99.999 ausgegeben, bei
  %uS in 99.999999.

  strftime_ms() besitzt gegenüber der Funktion strftime()
  einen weiteren Eingangsparameter  long usec.
  Mit diesem Parameter werden die Microsekunden übergeben, die in der
  Struktur struct tm *time_stru nicht übergeben werden.

  Wenn vor Aufruf der Funktion strftime_ms() eine der Funktionen
  get_iec_dateandtime(),get_iec_time() oder get_s7_dateandtime()
  aufgerufen wurde, dann ist der Ausgabewert tv->tv_usec
  ( struct timeval tv; ) der geeignete Eingangsparameter.

-------
```

```
  strftime_ms() works like strftime, but has 2 additional formats to print
  seconds.
  "%mS" (instead of %S) prints also the broken down milliseconds 99.999
  "%uS" (instead of %S) prints also the broken down microseconds 99.999999

  In comparison to strftime() the additional input parameter,
  "long usec", is used. This parameter contains the micro seconds, which are
  not contained in the structure struct tm *time_stru.

  If one of the functions get_iec_dateandtime(),get_iec_time() or
  get_s7_dateandtime() are used prior to strftime_ms(), their output
  parameter tv->tv_usec ( struct timeval tv; )  is the best choice to
  supply usec.

*/

char* strftime_ms(char dstr[],size_t dstr_len, const char format[], \
      const struct tm *time_stru, long usec )
;
```