User's Manual
V3.09.03

Micriμm

For the Way Engineers Work

Micriμm
1290 Weston Road, Suite 306
Weston, FL 33326
USA

www.Micrium.com

600-uC-Clk-002

Micriμm
For the Way Engineers Work

Table of Contents

Chapter

1

Introduction

The management of time is important in many microprocessor-based embedded systems. For instance, what would VCRs (Video Cassette Recorders) and DVRs (Digital Video Recorders) be without clock/calendars to schedule the recording of television programs?

A clock/calendar is a useful module for an embedded system. If you need a clock/calendar, you have to decide whether to implement it in hardware or software.

Clock/calendar chips are readily available and most can directly interface with microprocessors. These chips accurately maintain the time-of-day, and some chips even provide a built-in calendar. Some chips include a battery and can continue to keep track of date and time even when power is removed from the unit. Clock/calendar chips generally require a crystal, which further increases the recurring cost of your system. Clock/calendar chips are manufactured by a large number of semiconductor companies such as Freescale, National Semiconductor, Maxim, Dallas Semiconductor, etc. Just because you have a clock/calendar chip doesn't mean you don't need to write any software.

Your application software will still need to:

- program the clock/calendar chip with the correct date and time,
- program any alarm clock functions, and
- read the current date and time.

A software-maintained clock/calendar is the best solution when your application cannot afford the extra cost associated with a clock/calendar chip, a battery, and an extra crystal. A software-implemented clock/calendar module can offer most of the benefits of a hardware approach (except that it can't maintain date and time when power is removed).

Maintaining a clock/calendar is a trivial task for a microprocessor. The first thing you will need is a periodic time source that will interrupt the microprocessor at regular intervals. Such a time source is easy to find. AC power line frequencies (50 or 60 Hz) are generally

very accurate over long periods of time. For short-term accuracy, the crystal used to clock the microprocessor is also a good candidate; however, for such an application, the crystal frequency must be divided down. If your application software runs under a real-time multitasking operating system, the OS's clock tick is a convenient periodic time source, as long as the tick rate is an integer fraction of one second (for example 60 Hz and not 18.2 Hz as found on PCs).

A software approach requires very little ROM, RAM, and CPU time and does not add recurring cost to your system. Also, you can easily add features, such as alarm clock functions (with many alarm setpoints), timestamps, string-formatting utilities to convert date and time to ASCII, etc. Software-implemented clock/calendars are found in a number of familiar appliances such as VCRs, DVRs, stereos, FAX machines, microwave ovens, etc. If the microprocessor has a low-power standby mode, the software-implemented clock/calendars can be made to maintain correct date and time when the power is removed by also including a battery to power the microprocessor.

1-1 µC/CLK MODULE

µC/Clk is a module that implements a Year 2000 compliant clock/calendar module. The clock/calendar module offers the following features:

- Maintains time in seconds starting from 2000/01/01 (January 1st, 2000) at 00:00:00 UTC until 2134/12/31 (December 31st, 2134) 23:59:59 UTC; but supports conversions to/from two other timestamps:
  - NTP (Network Time Protocol) timestamps, starting from 1900/01/01 (January 1st, 1900) at 00:00:00 UTC until 2034/12/31 (December 31st, 2034) 23:59:59 UTC;
  - Unix timestamps, starting from 1970/01/01 (January 1st, 1970) at 00:00:00 UTC until 2104/12/31 (December 31st, 2104) 23:59:59 UTC.
- Allows your application to obtain timestamps to mark the occurrence of events. A µC/Clk timestamp is a copy of its internal timestamp.
- Allows your application to get the current date and time into a structured data type named `CLK_DATE_TIME` containing Year, Month, Day, Day-of-Year, Day-of-Week, Hour, Minute, Second, and Timezone Offset. Can convert timestamps to dates/times or vice versa.

- Allows your application to get and set the clock date/time using any of the supported timestamps or a `CLK_DATE_TIME` structure and allows conversion to/from all supported timestamps and the `CLK_DATE_TIME` structure.

This document describes how to configure and use the µC/Clk module.

Chapter

2

Directories and Files

The code and documentation of the µC/Clk module are organized in a directory structure according to "AN 2002, µC/OS-II Directory Structure." Specifically, the files may be found in the following directories:

\Micrium\Software\uC-Clk
This is the main directory for µC/Clk

\Micrium\Software\uC-Clk\Doc
This directory contains the µC/Clk documentation files, including this user's manual.

\Micrium\Software\uC-Clk\Cfg\Template
This directory contains a template of µC/Clk configuration.

\Micrium\Software\uC-Clk\Source
This directory contains the µC/Clk source code. This protocol is implemented in two OS independent files:

```
clk.c
clk.h
```

\Micrium\Software\uC-Clk\OS\uCOS-II
uCOS-III
This is where operating system (OS) dependent code is located. µC/Clk is distributed with ports for µC/OS-II and µC/OS-III. Note that it would be possible to use µC/Clk with other operating systems by developing appropriate `clk_os.*` implementation files.

REQUIRED MODULES

µC/Clk requires the µC/CPU and µC/LIB modules. Please refer to the µC/Clk release notes document for required version information.

Chapter

3

Using µC/Clk

µC/Clk is a fairly easy module to use:

- Your application must first configure µC/Clk parameters (see section 3-1).
- Your application must then initialize the µC/Clk module by calling `Clk_Init()`.
- Set the current date/time. You can do this in a number of ways:
  - Have an end-user enter the current date and time from a user interface that your application provides;
  - Read the current date/time from a real-time clock chip;
  - Obtain the current date/time from a NTP (Network Time Protocol) server if you use µC/TCP-IP in conjunction with µC/SNTPc.
- Your application can now get the current date/time using a variety of µC/Clk API functions.

3-1 µC/CLK CONFIGURATION

µC/Clk is configurable at compile time via approximately half a dozen `#defines`. A template configuration file (`clk_cfg.h`) is included in the module package (see Chapter 2, "Directories and Files"). This configuration should be copied into your application directory and modified according to your application's needs. µC/Clk uses `#defines` because they allow code and data sizes to be scaled at compile time based on enabled features. In other words, this allows the ROM and RAM footprints of µC/Clk to be adjusted based on the application requirements.

Most of the #defines should be configured with the default configuration values. Another small handful of values may likely never change because there is currently only one configuration choice available. This leaves a few values that should be configured with values that may deviate from the default configuration. However, keep in mind that future releases of this module might include more configuration options.

It is recommended that the configuration process starts with the recommended or default configuration values which are shown in **bold**.

3-1-1 MODULE CONFIGURATION

CLK_CFG_ARG_CHK_EN determines whether the code for arguments check is included. This value can either be DEF_DISABLED or **DEF_ENABLED.**

CLK_CFG_STR_CONV_EN determines whether the code for date/time structure conversion to a preformated string is included. This value can either be DEF_DISABLED or **DEF_ENABLED.**

CLK_CFG_NTP_EN determines whether the code for NTP timestamp utilities is included. This value can either be DEF_DISABLED or **DEF_ENABLED.**

CLK_CFG_UNIX_EN determines whether the code for Unix timestamp utilities is included. This value can either be DEF_DISABLED or **DEF_ENABLED.**

CLK_CFG_EXT_EN determines whether the code for Clock/calendar externally maintained is included. This value can either be **DEF_DISABLED** or DEF_ENABLED**.**

CLK_CFG_SIGNAL_EN determines if clock/calendar software maintained is task time delayed or is signaled via periodic call to Clk_SignalClk(). This value can either be **DEF_DISABLED** or DEF_ENABLED**.**

CLK_CFG_SIGNAL_FREQ_HZ determines the number of times Clk_SignalClk() gets called every second.

CLK_CFG_TZ_DFLT_SEC determines the default UTC time zone offset used by Clock.

3-1-2 OPERATING SYSTEM CONFIGURATION

The following configuration constants relate to the µC/Clk OS port. For many OSs, the µC/Clk task priority and stack size will need to be explicitly configured for the particular OS (consult the specific OS's documentation for more information).

The priority of µC/Clk task is dependent on the requirements of the application. For µC/OS-II and µC/OS-III, the following macros must be configured within `app_cfg.h`:

```
#define  CLK_OS_CFG_TASK_PRIO                          13
```

Value of the priority for the µC/Clk task. The value assigned depends upon the software architecture of your system, and on the importance of this module's response time relative to other tasks.

```
#define  CLK_OS_CFG_TASK_STK_SIZE                     512
```

Value of the stack size, in number of stack-sized words, for the µC/Clk task. This default value should be sufficient for most environments, but you should check this on your system for acceptable reliability or performance.

3-2 INTERFACE WITH RTOS

µC/Clk requires the presence of a Real Time Operating System (RTOS) if an External timestamp is not used. As mentioned in Chapter 2, µC/Clk is delivered with ports for µC/OS-II and µC/OS-III, but it is possible for µC/Clk to be used with another RTOS by providing appropriate implementations of `clk_os.*`.

3-3 µC/CLK EXAMPLE CODE

The following example code illustrates the capabilities and usage of the µC/Clk module. This code simply initializes µC/Clk, create date/time structure, set Clock timestamp and time zone, get Clock timestamp and time zone. Also the example shows timestamp and date/time conversions.

Listing 3-1 **Example code.**

```
static  void  AppTaskStart (void  *p_arg)
{
    CLK_TS_SEC     ts_sec;
    CLK_TS_SEC     ts_unix_sec;
    CLK_TZ_SEC     tz_sec;
    CLK_DATE_TIME  date_time;
    CPU_BOOLEAN    valid;
    CPU_CHAR       str[128];
    CLK_ERR        err;


    Clk_Init(&err);                                                                    (1)
    if (err == CLK_ERR_NONE) {
        printf("Clock module successfully initialized\n\r");
    } else {
        printf("Clock module initialization failed\n\r");
        return;
    }

    tz_sec = 0;

    valid = Clk_DateTimeMake(&date_time, 2010, 10, 18, 11, 11, 11, tz_sec);            (2)
    if (valid != DEF_OK) {
        printf("Clock make date/time failed\n\r");
        return;
    }

    date_time.Yr      = 2010;
    date_time.Month   =   10;
    date_time.Day     =   18;
    date_time.DayOfWk =  Clk_GetDayOfWk(2010, 10, 18);                                 (3)
    date_time.DayOfYr =  291;
    date_time.Hr      =   11;
    date_time.Min     =   11;
    date_time.Sec     =   11;
    date_time.TZ_sec  = tz_sec;
    valid             = Clk_IsDateTimeValid(&date_time);                               (4)
    if (valid != DEF_OK) {
        printf("Clock date/time not valid\n\r");
        return;
    }
```

```
valid = Clk_SetDateTime(&date_time);                                                    (5)
if (valid != DEF_OK) {
    printf("Clock set date/time failed\n\r");
    return;
}

valid = Clk_DateTimeToStr(&date_time, str, 128, CLK_STR_FMT_YYYY_MM_DD_HH_MM_SS_UTC_LEN); (6)
if (valid == DEF_OK) {
    printf("Current Date/time :%s", str);
} else {
    printf("Clock date/time to string failed\n\r");
    return;
}

Clk_DateTimeToTS(&ts_sec, &date_time);                                                  (7)
if (valid == DEF_OK) {
    printf("Clock timestamp = %u\n\r", ts_sec);
} else {
    printf("Clock date/time to timestamp failed\n\r");
    return;
}

tz_sec = (-5 * 60 * 60);
valid = Clk_SetTZ(tz_sec);                                                              (8)
if (valid != DEF_OK) {
    printf("Clock set timezone unix failed\n\r");
    return;
}

valid = Clk_GetDateTime(&date_time);                                                    (9)
if (valid != DEF_OK) {
    printf("Clock get date/time failed\n\r");
    return;
}

valid = Clk_DateTimeToStr(&date_time, str, 128, CLK_STR_FMT_YYYY_MM_DD_HH_MM_SS_UTC_LEN);(10)
if (valid == DEF_OK) {
    printf("Current Date/time :%s", str);
} else {
    printf("Clock date/time to string failed\n\r");
    return;
}
```

```
valid = Clk_GetTS(&ts_sec);                                                                    (11)
if (valid == DEF_OK) {
    printf("Clock timestamp = %u\n\r", ts_sec);
} else {
    printf("Clock get timestamp failed\n\r");
    return;
}

valid = Clk_TS_ToDateTime(&ts_sec, 0, &date_time);                                             (12)
if (valid != DEF_OK) {
    printf("Clock convert timestamp to date/time failed\n\r");
    return;
}

valid = Clk_GetTS_Unix(&ts_unix_sec);                                                          (13)
if (valid != DEF_OK) {
    printf("Clock get timestamp unix failed\n\r");
    return;
}

valid = Clk_TS_UnixToDateTime(ts_unix_sec, tz_sec, &date_time);                                (14)
if (valid != DEF_OK) {
    printf("Clock timestamp unix to date/time failed\n\r");
    return;
}

valid = Clk_DateTimeToStr(&date_time, str, 128, CLK_STR_FMT_YYYY_MM_DD_HH_MM_SS_UTC_LEN);(15)
if (valid == DEF_OK) {
    printf("Current Date/time :%s", str);
} else {
    printf("Clock date/time to string failed\n\r");
    return;
}

ts_unix_sec = 126316799uL;
valid      = Clk_TS_UnixToDateTime(ts_unix_sec, tz_sec, &date_time);                           (16)
if (valid != DEF_OK) {
    printf("Clock set date/time failed\n\r");
    return;
}
```

```
    valid = Clk_DateTimeToStr(&date_time, str, 128, CLK_STR_FMT_DAY_MONTH_DD_HH_MM_SS_YYYY); (17)
    if (valid == DEF_OK) {
        printf("Unix timestamp = %s", str);
    } else {
        printf("Clock date/time to string failed\n\r");
        return;
    }
}
```

L3-1(1) Initialize the µC/Clk. If the process is successful, the µC/Clk task is started, and its various data are initialized.

L3-1(2) Build a date/time structure.

L3-1(3) Get the day of week of 2010, october 18.

L3-1(4) Validate date/time structure fields.

L3-1(5) Set current timestamp of Clock module with date/time structure.

L3-1(6) Get a foramted string via a date/time structure.

L3-1(7) Convert date/time structure to a timestamp.

L3-1(8) Set Clock time zone.

L3-1(9) Get current Click date/time into a date/time structures.

L3-1(10) Get a foramted string via a date/time structure.

L3-1(11) Get current timestamp of Clock module.

L3-1(12) Convert timestamp to a date/time structure.

L3-1(13) Get current Clock timestamp as a Unix timestamp.

L3-1(14) Convert a Unix timestamp to a date/time structures.

L3-1(15) Get a foramted string via a date/time structure.

L3-1(16) Unix timestamp is equivalent of 1974, January 1 23:59:59 UTC+0

L3-1(17) Convert Unix timestamp to a date/time structure.

L3-1(18) Get a foramted string via a date/time structure.

Chapter

4

µC/Clk API

This chapter provides a reference to the µC/Clk API. Each of the user-accessible services is presented in alphabetical order. The following information is provided for each of those services:

- A brief description
- The function prototype
- The filename of the source code
- A description of the arguments passed to the function
- A description of the returned values
- Specific notes and warnings on using the service

4-1 `Clk_Init()`

Initializes Clock module.

FILES

`clk.h/clk.c`

PROTOTYPE

```
void  Clk_Init (CLK_ERR  *p_err);
```

ARGUMENTS

p_err Pointer to variable that will receive the return error code from this function:

CLK_ERR_NONE
CLK_OS_ERR_INIT_NAME
CLK_OS_ERR_INIT_SIGNAL
CLK_OS_ERR_INIT_TASK

RETURNED VALUES

None.

REQUIRED CONFIGURATION

None.

NOTES / WARNINGS

None.

EXAMPLE USAGE

```
CLK_ERR  err;

Clk_Init(&err);
if (err == CLK_ERR_NONE) {
    printf("Clock module successfully initialized\n\r");
} else {
    printf("Clock module initialization failed\n\r");
}
```

4-2 Clk_ExtTS_Init()

Initialize and start timestamp timer.

FILES

clk.h / Application's source file

CALLED FROM

Clk_Init()

PROTOTYPE

```
void  Clk_ExtTS_Init (void);
```

ARGUMENTS

None.

RETURNED VALUES

None.

REQUIRED CONFIGURATION

Required callback function that must be implemented in your application if CLK_CFG_EXT_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1) in order for the clock/calendar to be maintained by an external clock/timestamp mechanism.

NOTES / WARNINGS

External timestamp should be an 'up' counter whose values increase at each second. It's possible to use a 'down' counter, but a conversion must be applied when setting and getting the External timestamp.

External timestamp could come from another application (e.g., by SNTP).

EXAMPLE TEMPLATE

```
CPU_BOOLEAN  Clk_ExtTS_Init (void)
{
    BSP_ClockInitChip();
    BSP_ClockStartTS();
}
```

4-3 Clk_ExtTS_Get()

Get Clock module's timestamp from converted External timestamp.

FILES

clk.h / Application's source file

CALLED FROM

Clk_GetTS()

PROTOTYPE

```
CLK_TS_SEC  Clk_ExtTS_Get (void);
```

ARGUMENTS

None.

RETURNED VALUES

Current Clock module timestamp (in seconds, UTC+00).

REQUIRED CONFIGURATION

Required callback function that must be implemented in your application if `CLK_CFG_EXT_EN` is `DEF_ENABLED` in `clk_cfg.h` (see section 3-1-1) in order for the clock/calendar to be maintained by an external clock/timestamp mechanism.

NOTES / WARNINGS

Clock timestamp values must be returned via `CLK_TS_SEC` data type. If the External timestamp has more bits than the `CLK_TS_SEC` data type, `Clk_ExtTS_Get()` must truncate the External timestamp's higher order bits greater than the `CLK_TS_SEC` data type. If the External timestamp has less bits than the `CLK_TS_SEC` data type, `Clk_ExtTS_Get()` must pad the Clock timestamp's higher order bits with 0 bits.

External timestamp values must be returned from the reference of the Clock epoch start date/time. External timestamp should start on midnight of January 1st of its epoch start year. Returned Clock timestamp must be representable in Clock epoch. Thus equivalent date of the External timestamp must be greater than or equal to `CLK_EPOCH_YR_START` and less than `CLK_EPOCH_YR_END`.

If the External timestamp includes an (optional) external time zone, `Clk_ExtTS_Get()` must subtract the external time zone offset from the converted External timestamp.

The Clock timestamp is calculated by one of the following equations where:

Clock TS	Timestamp converted for Clock (in seconds, from UTC+00)
External TS	External timestamp to convert (in seconds)
Clock start year	Clock epoch start year (`CLK_EPOCH_YR_START`)
Clock end year	Clock epoch end year (`CLK_EPOCH_YR_END`)
External start year	External timestamp epoch start year
Leap day count	Number of leap days between Clock epoch start year and External epoch start year
Seconds per day	Number of seconds per day (86400)
External TZ	Time zone offset applied to External timestamp (in seconds, from UTC+00)

When External epoch start year is less than Clock epoch start year (CLK_EPOCH_YR_START):

```
Clock TS = External TS
         - ((((Clock start year - External start year) * 365)
             + leap day count)  * seconds per day]
         - External TZ
```

Examples with a 32-bit External timestamp:

- Valid equivalent date to convert is after Clock epoch start year:

```
  2010 Oct 8, 11:11:11 UTC-05:00
External TS (in seconds)  =  1286536271
External start year       =        1970
Leap day count            =           7
External TZ (in seconds)  =      -18000
Clock    TS (in seconds)  =   339869471
  2010 Oct 8, 16:11:11 UTC
```

  This example successfully converts an External timestamp into a representable Clock timestamp without underflowing.

- Invalid equivalent date to convert is before Clock epoch start year:

```
  1984 Oct 8, 11:11:11 UTC-05:00
External TS (in seconds)  =  466081871
External start year       =       1970
Clock    start year       =        2000
Leap day count            =          7
External TZ (in seconds)  =     -18000
Clock    TS (in seconds)  = -480584929
```

  This example underflows to a negative Clock timestamp since the equivalent date to convert is incorrectly less than the Clock epoch start year (CLK_EPOCH_YR_START).

When External epoch start year is greater than Clock epoch start year (CLK_EPOCH_YR_START):

```
Clock TS = External TS
         + [(((External start year - Clock start year) * 365)
             + leap day count)     * seconds per day]
         - External TZ
```

Examples with a 32-bit External timestamp:

- Valid equivalent date to convert is before Clock epoch end year:

```
  2010 Oct 8, 11:11:11 UTC-05:00
External TS (in seconds)  =   24232271
External start year       =       2010
Leap day count            =          3
External TZ (in seconds)  =     -18000
Clock    TS (in seconds)  =  339869471
  2010 Oct 8, 16:11:11 UTC-05:00
```

  This example successfully converts an External timestamp into a representable Clock timestamp without overflowing.

- Invalid equivalent date to convert is after Clock epoch end year:

```
  2140 Oct 8, 11:11:11 UTC-05:00
External TS (in seconds)  = 4126677071
External start year       =       2010
Clock    end   year       =       2136
Leap day count            =          3
External TZ (in seconds)  =     -18000
Clock    TS (in seconds)  = 4442314271
```

  This example overflows the Clock timestamp (32-bit) CLK_TS_SEC data type with an equivalent date incorrectly greater than or equal to the Clock epoch end year (CLK_EPOCH_YR_END).

EXAMPLE TEMPLATE

```
CLK_TS_SEC  Clk_ExtTS_Get (void)
{
    CLK_TS_SEC  ts_sec;

    ts_sec = BSP_ClockGetTS();
    return (ts_sec);
}
```

4-4 Clk_ExtTS_Set()

Set External timestamp.

FILES

clk.h / Application's source file

CALLED FROM

Clk_SetTS()

PROTOTYPE

```
CPU_BOOLEAN  Clk_ExtTS_Set (CLK_TS_SEC  ts_sec);
```

ARGUMENTS

ts_sec External timestamp value to set (in seconds, UTC+00).

RETURNED VALUES

DEF_OK, if External timestamp succesfully set;

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

Required callback function that must be implemented in your application if `CLK_CFG_EXT_EN` is `DEF_ENABLED` in `clk_cfg.h` (see section 3-1-1) in order for the clock/calendar to be maintained by an external clock/timestamp mechanism.

NOTES / WARNINGS

External timestamp values are converted from Clock timestamp's `CLK_TS_SEC` data type. If the External timestamp has more bits than the `CLK_TS_SEC` data type, `Clk_ExtTS_Set()` must pad the External timestamp's higher order bits with 0 bits. If the External timestamp has less bits than the `CLK_TS_SEC` data type, `Clk_ExtTS_Set()` must truncate the Clock timestamp's higher order bits greater than the External timestamp.

External timestamp values must be converted from the reference of the Clock epoch start date/time. External timestamp should start on midnight of January 1st of its epoch start year. Converted External timestamp must be representable in External epoch. Thus equivalent date of the External timestamp must be greater than or equal to the External epoch start year and less than the External epoch end year.

If the External timestamp includes an (optional) external time zone, `Clk_ExtTS_Set()` must add the external time zone offset to the converted External timestamp.

The External timestamp is calculated by one of the following equations where:

Clock TS	Clock timestamp (in seconds, from UTC+00)
External TS	Converted External timestamp (in seconds)
Clock start year	Clock epoch start year (`CLK_EPOCH_YR_START`)
External start year	External timestamp epoch start year
External end year	External timestamp epoch end year
Leap day count	Number of leap days between Clock epoch start year and External epoch start year
Seconds per day	Number of seconds per day (86400)
External TZ	Time zone offset applied to External timestamp (in seconds, from UTC+00)

When External epoch start year is less than Clock epoch start year (CLK_EPOCH_YR_START):

```
External TS = Clock TS
            + [(((Clock start year - External start year) * 365)
                + leap day count)  * seconds per day]
            + External TZ
```

Examples with a 32-bit External timestamp:

- Valid equivalent date to convert is before External epoch end year:

```
  2010 Oct 8, 16:11:11 UTC
Clock    TS (in seconds)  =  339869471
External start year       =       1970
Leap day count            =          7
External TZ (in seconds)  =     -18000
External TS (in seconds)  = 1286536271
  2010 Oct 8, 11:11:11 UTC-05:00
```

  This example successfully converts an External timestamp into a representable Clock timestamp without overflowing.

- Invalid equivalent date to convert is after External epoch end year:

```
  2120 Oct 8, 11:11:11 UTC
Clock    TS (in seconds)  = 3811144271
External start year       =       1970
External end   year       =       2106
Leap day count            =          7
External TZ (in seconds)  =     -18000
External TS (in seconds)  = 4757811071
```

  This example overflows the External (32-bit) timestamp with an equivalent date incorrectly greater than or equal to the External epoch end year.

When External epoch start year is greater than Clock epoch start year (CLK_EPOCH_YR_START):

```
External TS = Clock TS
            - [(((External start year - Clock start year) * 365)
                + leap day count)     * seconds per day]
            + External TZ
```

Examples with a 32-bit External timestamp:

- Valid equivalent date to convert is after External epoch start year:

```
  2010 Oct 8, 16:11:11 UTC
Clock    TS (in seconds)  =  339869471
External start year       =       2010
Leap day count            =          3
External TZ (in seconds)  =     -18000
External TS (in seconds)  =   24232271
  2010 Oct 8, 11:11:11 UTC-05:00
```

  This example successfully converts an External timestamp into a representable Clock timestamp without underflowing.

- Invalid equivalent date to convert is before External epoch start year:

```
  2005 Oct 8, 11:11:11 UTC
Clock    TS (in seconds)  =  182085071
External start year       =       2010
Leap day count            =          3
External TZ (in seconds)  =     -18000
External TS (in seconds)  = -133552129
```

  This example underflows to a negative External timestamp since the equivalent date to convert is incorrectly less than the External epoch start year.

EXAMPLE TEMPLATE

```
CPU_BOOLEAN  Clk_ExtTS_Set (CLK_TS_SEC  ts_sec)
{
    BSP_ClockSetTS(ts_sec);
    return (DEF_OK);
}
```

4-5 Clk_SignalClk()

Signal the clock task when one second has elapsed in order to increment the Clock module's tick counter.

FILES

clk.h/clk.c

PROTOTYPE

```
void  Clk_SignalClk (CLK_ERR  *p_err);
```

ARGUMENTS

p_err Pointer to variable that will receive the return error code from this function:

CLK_ERR_NONE
CLK_OS_ERR_SIGNAL

RETURNED VALUES

None.

REQUIRED CONFIGURATION

When CLK_CFG_EXT_EN is DEF_DISABLED and CLK_CFG_SIGNAL_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1), this function must be periodically called by application/BSP functions in order to increment the internal clock ticker. CLK_CFG_SIGNAL_FREQ_HZ must be configured to the number of times this function gets called every second.

NOTES / WARNINGS

None.

EXAMPLE USAGE

```
CLK_ERR  err;

Clk_SignalClk(&err);
if (err != CLK_ERR_NONE) {
    printf("Clock module timestamp tick signal error");
}
```

4-6 Clk_GetTS()

Get current Clock timestamp.

FILES

clk.h/clk.c

PROTOTYPE

```
CLK_TS_SEC  Clk_GetTS (void);
```

ARGUMENTS

None.

RETURNED VALUES

Current timestamp (in seconds, UTC+00).

REQUIRED CONFIGURATION

None.

NOTES / WARNINGS

Clock timestamp returned as UTC+00. Thus any local time zone offset must be applied after calling `Clk_GetTS()`.

EXAMPLE USAGE

```
CLK_TS_SEC  ts_sec;

ts_sec = Clk_GetTS();
printf("timestamp = %u", ts_sec);
```

4-7 `Clk_SetTS()`

Set Clock timestamp.

FILES

`clk.h/clk.c`

PROTOTYPE

```
CPU_BOOLEAN  Clk_SetTS (CLK_TS_SEC  ts_sec);
```

ARGUMENTS

`ts_sec` Current timestamp to set (in seconds, UTC+00).

RETURNED VALUES

`DEF_OK`, if timestamp is successfully set.

`DEF_FAIL`, otherwise.

REQUIRED CONFIGURATION

None

NOTES / WARNINGS

Clock timestamp should be set for UTC+00 (i.e., no local time zone offset included).

EXAMPLE USAGE

```
CLK_TS_SEC   ts_sec;
CPU_BOOLEAN  valid;

ts_sec = 15052;
valid  = Clk_SetTS(ts_sec);
if (valid == DEF_OK) {
    printf("Clock Set TS successful\n\r");
} else {
    printf("Clock Set TS error\n\r");
}
```

4-8 Clk_GetTZ()

Get Clock time zone offset.

FILES

clk.h/clk.c

PROTOTYPE

```
CLK_TZ_SEC  Clk_GetTZ (void);
```

ARGUMENTS

None.

RETURNED VALUES

Time zone offset (in seconds, ± from UTC).

REQUIRED CONFIGURATION

None.

NOTES / WARNINGS

None.

EXAMPLE USAGE

```
CLK_TZ_SEC  tz_sec;
CLK_TS_SEC  ts_local;

tz_sec   = Clk_GetTZ();
ts_local = 15000 + tz_sec;
printf("local timestamp = %u", ts_local);
```

4-9 Clk_SetTZ()

Set Clock time zone offset.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_SetTZ (CLK_TZ_SEC  tz_sec);
```

ARGUMENTS

tz_sec Time zone offset (in seconds, ± from UTC).

RETURNED VALUES

`DEF_OK`, if time zone is valid and set.

`DEF_FAIL`, otherwise.

REQUIRED CONFIGURATION

None.

NOTES / WARNINGS

Time zone is based on Coordinated Universal Time (UTC) and has valid values:

- Between ±12 hours (±43200 seconds)
- Multiples of 15 minutes

EXAMPLE USAGE

```
CLK_TZ_SEC   tz_sec;
CPU_BOOLEAN  valid;

tz_sec = -5 * 3600;
valid  = Clk_SetTZ(tz_sec);
if (valid == DEF_OK) {
    printf("Clock Set TZ successful\n\r");
} else {
    printf("Clock Set TZ error\n\r");
}
```

4-10 `Clk_GetDateTime()`

Get current Clock timestamp as a date/time structure.

FILES

`clk.h/clk.c`

PROTOTYPE

```
CPU_BOOLEAN  Clk_GetDateTime (CLK_DATE_TIME  *p_date_time);
```

ARGUMENTS

`p_date_time` Pointer to variable that will receive the date/time structure.

RETURNED VALUES

`DEF_OK`, if date/time structure is valid.

`DEF_FAIL`, otherwise.

REQUIRED CONFIGURATION

None.

NOTES / WARNINGS

Clock time zone offset is used to calculate the local date/time (`p_date_time`) returned. Thus local date/time is returned as UTC+TZ, where Clock time zone offset (TZ) is returned as local time zone offset (`p_date_time->TZ_sec`).

EXAMPLE USAGE

```
CLK_DATE_TIME  date_time;
CPU_BOOLEAN    valid;

valid = Clk_GetDateTime(&date_time);
if (valid == DEF_OK) {
    printf("Time = %u:%u:%u\n\r", date_time->Hr, date_time->Min, date_time->Sec);
} else {
    printf("Clock Get Date/time error\n\r");
}
```

4-11 Clk_SetDateTime()

Set Clock timestamp from a date/time structure.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_SetDateTime (CLK_DATE_TIME  *p_date_time);
```

ARGUMENTS

p_date_time Pointer to variable that contains the date/time structure.

RETURNED VALUES

DEF_OK, if Clock timestamp is successfully set.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

None.

NOTES / WARNINGS

Date/time (p_date_time) should be set to local time with correct time zone offset (p_date_time->TZ_sec). Clk_SetDateTime() removes the time zone offset from the date/time to calculate the Clock timestamp as UTC+00.

Internal Clock time zone is set to the local time zone offset (p_date_time->TZ_sec).

EXAMPLE USAGE

```
CLK_DATE_TIME  date_time;
CPU_BOOLEAN    valid;

date_time.Yr      =   2010;     /* 2010/09/18  11:11:11 UTC-05:00  */
date_time.Month   =      9;
date_time.Day     =     18;
date_time.Hr      =     11;
date_time.Min     =     11;
date_time.Sec     =     11;
date_time.DayOfWk =      2;
date_time.DayOfYr =    291;
date_time.TZ_sec  = -18000;

valid = Clk_SetDateTime(&date_time);
if (valid == DEF_OK) {
    printf("Date/time successfully set");
} else {
    printf("Clock Set Date/time error\n\r");
}
```

4-12 Clk_TS_ToDateTime()

Convert Clock timestamp to date/time structure.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_TS_ToDateTime (CLK_TS_SEC      ts_sec,
                                CLK_TZ_SEC      tz_sec,
                                CLK_DATE_TIME  *p_date_time);
```

ARGUMENTS

`ts_sec` Timestamp to convert (in seconds, UTC+00).

`tz_sec` Time zone offset (in seconds, ± from UTC).

`p_date_time` Pointer to variable that will receive the date/time structure.

RETURNED VALUES

`DEF_OK`, if date/time structure successfully returned.

`DEF_FAIL`, otherwise.

REQUIRED CONFIGURATION

None.

NOTES / WARNINGS

Timestamp (`ts_sec`) must be set for UTC+00 and should not include the time zone offset (`tz_sec`) since `Clk_TS_ToDateTime()` includes the time zone offset in its date/time calculation. Thus the time zone offset should not be applied before or after calling `Clk_TS_ToDateTime()`.

Time zone field of the date/time structure (`p_date_time->TZ_sec`) is set to the value of the time zone argument (`tz_sec`).

EXAMPLE USAGE

```
CLK_TS_SEC     ts_sec;
CLK_TZ_SEC     tz_sec;
CLK_DATE_TIME  date_time;
CPU_BOOLEAN    valid;

ts_sec = 1025630;
tz_sec = 0;
valid  = Clk_TS_ToDateTime(ts_sec, tz_sec, &date_time);
if (valid == DEF_OK) {
    printf("Date = %u/%u/%u\n\r", date_time->Yr, date_time->Month, date_time->Day);
    printf("Time = %u:%u:%u\n\r", date_time->Hr, date_time->Min,   date_time->Sec);
} else {
    printf("Clock Get Date/time error\n\r");
}
```

4-13 Clk_DateTimeToTS()

Convert date/time structure to Clock timestamp.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_DateTimeToTS (CLK_TS_SEC     *p_ts_sec,
                               CLK_DATE_TIME  *p_date_time);
```

ARGUMENTS

p_ts_sec Pointer to variable that will receive the Clock timestamp:

In seconds UTC+00,	if no errors;
CLK_TS_SEC_NONE,	otherwise.

p_date_time Pointer to variable that contains date/time structure to convert.

RETURNED VALUES

DEF_OK, if timestamp successfully returned.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

None.

NOTES / WARNINGS

Date/time structure (p_date_time) must be representable in Clock timestamp. Thus date to convert must be greater than or equal to CLK_EPOCH_YR_START and less than CLK_EPOCH_YR_END. Date/time should be set to local time with correct time zone offset (p_date_time->TZ_sec). Clk_DateTimeToTS() removes the time zone offset from the date/time to calculate and return a Clock timestamp at UTC+00.

EXAMPLE USAGE

```
CLK_TS_SEC     ts_sec;
CLK_DATE_TIME  date_time;
CPU_BOOLEAN    valid;

date_time.Yr      =   2010;     /* 2010/09/18  11:11:11 UTC-05:00  */
date_time.Month   =      9;
date_time.Day     =     18;
date_time.Hr      =     11;
date_time.Min     =     11;
date_time.Sec     =     11;
date_time.DayOfWk =      2;
date_time.DayOfYr =    291;
date_time.TZ_sec  = -18000;

valid = Clk_DateTimeToTS(&ts_sec, &date_time);
if (valid == DEF_OK) {
    printf("Clock timestamp = %u", ts_sec);
} else {
    printf("Clock Date/time to timestamp error\n\r");
}
```

4-14 Clk_DateTimeMake()

Build a valid Clock epoch date/time structure.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_DateTimeMake (CLK_DATE_TIME  *p_date_time,
                               CLK_YR          yr,
                               CLK_MONTH       month,
                               CLK_DAY         day,
                               CLK_HR          hr,
                               CLK_MIN         min,
                               CLK_SEC         sec,
                               CLK_TZ_SEC      tz_sec);
```

ARGUMENTS

p_date_time	Pointer to variable that will receive the date/time structure.	
yr	Year	value [CLK_EPOCH_YR_START to CLK_EPOCH_YR_END).
month	Month	value [CLK_MONTH_JAN to CLK_MONTH_DEC].
day	Day	value [1 to 31].
hr	Hours	value [0 to 23].
min	Minutes	value [0 to 59].
sec	Seconds	value [0 to 60].
tz_sec	Time zone offset (in seconds, ± from UTC) [-43200 to 43200].	

RETURNED VALUES

DEF_OK, if date/time structure successfully returned.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

None.

NOTES / WARNINGS

Date/time structure (p_date_time) must be representable in Clock timestamp. Thus date to convert must be greater than or equal to CLK_EPOCH_YR_START and less than CLK_EPOCH_YR_END.

EXAMPLE USAGE

```
CLK_DATE_TIME  date_time;
CPU_BOOLEAN    valid;

                                /* 2010/09/18   11:11:11 UTC-05:00  */
valid = Clk_DateTimeMake(&date_time, 2010, 9, 18, 11, 11, 11, -18000);
if (valid == DEF_OK) {
    printf("Date/time successfully created");
} else {
    printf("Clock Date/time error\n\r");
}
```

4-15 Clk_IsDateTimeValid()

Determine if date/time structure is valid in Clock epoch.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_IsDateTimeValid (CLK_DATE_TIME  *p_date_time);
```

ARGUMENTS

p_date_time Pointer to variable that contains the date/time structure to validate.

RETURNED VALUES

DEF_YES, if date/time structure is valid.

DEF_NO, otherwise.

REQUIRED CONFIGURATION

None.

NOTES / WARNINGS

Date/time structure (p_date_time) must be representable in Clock timestamp. Thus date to validate must be greater than or equal to CLK_EPOCH_YR_START and less than CLK_EPOCH_YR_END.

EXAMPLE USAGE

```
CLK_TS_SEC     ts_sec;
CLK_DATE_TIME  date_time;
CPU_BOOLEAN    valid;

date_time.Yr      =   2010;     /* 2010/09/18  11:11:11 UTC-05:00  */
date_time.Month   =      9;
date_time.Day     =     18;
date_time.Hr      =     11;
date_time.Min     =     11;
date_time.Sec     =     11;
date_time.DayOfWk =      2;
date_time.DayOfYr =    291;
date_time.TZ_sec  = -18000;

valid = Clk_IsDateTimeValid(&date_time);
if (valid == DEF_OK) {
    printf("Date/time is valid");
} else {
    printf("Date/time is NOT valid");
}
```

4-16 Clk_GetDayOfWk()

Get the day of week.

FILES

clk.h/clk.c

PROTOTYPE

```
CLK_DAY  Clk_GetDayOfWk (CLK_YR     yr,
                         CLK_MONTH  month,
                         CLK_DAY    day);
```

ARGUMENTS

yr	Year value [1900 to 2135].
month	Month value [1 to 12], (January to December).
day	Day value [1 to 31].

RETURNED VALUES

Day of week [1 to 7] (Sunday to Saturday),	if no errors.
CLK_DAY_OF_WK_NONE,	otherwise

REQUIRED CONFIGURATION

None.

NOTES / WARNINGS

It's only possible to get a day of week of an epoch supported by Clock:

- Earliest year is the NTP epoch start year, thus Year (yr) must be greater than or equal to CLK_NTP_EPOCH_YR_START.
- Latest year is the Clock epoch end year, thus Year (yr) must be less than CLK_EPOCH_YR_END.

EXAMPLE USAGE

```
CLK_DAY  day_of_wk;

day_of_wk = Clk_GetDayOfWk(2010, 9, 18);
print("day of week = %u", day_of_wk);
```

4-17 Clk_GetDayOfYr()

Get the day of year.

FILES

clk.h/clk.c

PROTOTYPE

```
CLK_DAY  Clk_GetDayOfYr (CLK_YR     yr,
                         CLK_MONTH  month,
                         CLK_DAY    day);
```

ARGUMENTS

yr	Year value [1900 to 2135].
month	Month value [1 to 12], (January to December).
day	Day value [1 to 31].

RETURNED VALUES

Day of year [1 to 366],	if no errors.
CLK_DAY_OF_WK_NONE,	otherwise

REQUIRED CONFIGURATION

None.

NOTES / WARNINGS

It's only possible to get a day of year of an epoch supported by Clock:

- Earliest year is the NTP epoch start year, thus Year (`yr`) must be greater than or equal to `CLK_NTP_EPOCH_YR_START`.
- Latest year is the Clock epoch end year, thus Year (`yr`) must be less than `CLK_EPOCH_YR_END`.

EXAMPLE USAGE

```
CLK_DAY  day_of_yr;

day_of_yr = Clk_GetDayOfYr(2010, 9, 18);
print("day of year = %u", day_of_yr);
```

4-18 Clk_DateTimeToStr()

Converts a date/time structure to an ASCII string.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_DateTimeToStr (CLK_DATE_TIME  *p_date_time,
                                CLK_STR_FMT     fmt,
                                CPU_CHAR       *p_str,
                                CPU_INT32U      str_len);
```

ARGUMENTS

p_date_time Pointer to variable that contains the date/time structure to convert.

fmt Desired string format:

- CLK_STR_FMT_YYYY_MM_DD_HH_MM_SS_UTC
- CLK_STR_FMT_YYYY_MM_DD_HH_MM_SS
- CLK_STR_FMT_MM_DD_YY_HH_MM_SS
- CLK_STR_FMT_YYYY_MM_DD
- CLK_STR_FMT_MM_DD_YY
- CLK_STR_FMT_DAY_MONTH_DD_YYYY
- CLK_STR_FMT_DAY_MONTH_DD_HH_MM_SS_YYYY
- CLK_STR_FMT_HH_MM_SS
- CLK_STR_FMT_HH_MM_SS_AM_PM

p_str Pointer to variable that will receive the formated string.

str_len Maximum number of characters the string can contains.

RETURNED VALUES

DEF_OK, if string successfully returned.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

Available only if CLK_CFG_STR_CONV_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1).

NOTES / WARNINGS

It's only possible to convert date supported by Clock:

- Earliest year is the NTP epoch start year, thus Year (yr) must be greater than or equal to CLK_NTP_EPOCH_YR_START.
- Latest year is the Clock epoch last year, thus Year (yr) must be less than CLK_EPOCH_YR_END.

The size of the string buffer that will receive the returned string address must be greater than or equal to CLK_STR_FMT_MAX_LEN.

EXAMPLE USAGE

```
CLK_DATE_TIME  date_time;
CPU_BOOLEAN    valid;
CPU_CHAR       str[CLK_STR_FMT_YYYY_MM_DD_HH_MM_SS_UTC_LEN]

                                  /* 2010/09/18   11:11:11 UTC-05:00  */
valid = Clk_DateTimeMake(&date_time, 2010, 9, 18, 11, 11, 11, -18000);
if (valid == DEF_OK) {
    printf("Date/time successfully created");
} else {
    printf("Clock Date/time error\n\r");
}

valid = Clk_DateTimeToStr(&date_time,
                          CLK_STR_FMT_YYYY_MM_DD_HH_MM_SS_UTC,
                          str,
                          CLK_STR_FMT_YYYY_MM_DD_HH_MM_SS_UTC_LEN);
if (valid == DEF_OK) {
    printf("Date/time = %s", str);
} else {
    printf("Clock Date/time to String error\n\r");
}
```

4-19 Clk_GetTS_NTP()

Get current Clock timestamp as an NTP timestamp.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_GetTS_NTP (CLK_TS_SEC  *p_ts_ntp_sec);
```

ARGUMENTS

p_ts_ntp_sec Pointer to variable that will receive the NTP timestamp:

In seconds UTC+00,	if no errors;
CLK_TS_SEC_NONE,	otherwise.

RETURNED VALUES

DEF_OK, if current timestamp is successfully converted.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

Available only if CLK_CFG_NTP_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1).

NOTES / WARNINGS

NTP timestamp does not include any time zone offset. Thus any local time zone offset must be applied after calling Clk_GetTS_NTP().

NTP timestamp will eventually overflow, thus it's not possible to get NTP timestamp for years on or after CLK_NTP_EPOCH_YR_END.

EXAMPLE USAGE

```
CLK_TS_SEC   ts_ntp_sec;
CPU_BOOLEAN  valid;

valid = Clk_GetTS_NTP(&ts_ntp_sec);
if (valid == DEF_OK) {
    printf("Timestamp NTP = %u", ts_ntp_sec);
} else {
    printf("Get TS NTP error\n\r");
}
```

4-20 Clk_SetTS_NTP()

Set Clock timestamp from an NTP timestamp.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_SetTS_NTP (CLK_TS_SEC  ts_ntp_sec);
```

ARGUMENTS

ts_ntp_sec Current NTP timestamp to set (in seconds, UTC+00).

RETURNED VALUES

DEF_OK, if timestamp is successfully set.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

Available only if CLK_CFG_NTP_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1).

NOTES / WARNINGS

Only years supported by Clock and NTP can be set, thus the timestamp date must be between greater than or equal to CLK_EPOCH_YR_START and less than CLK_NTP_EPOCH_YR_END.

EXAMPLE USAGE

```
CLK_TS_SEC   ts_ntp_sec;
CPU_BOOLEAN  valid;

ts_ntp_sec = 20200020;
valid      = Clk_SetTS_NTP(&ts_ntp_sec);
if (valid == DEF_OK) {
    printf("Timestamp successfully set");
} else {
    printf("Set timestamp error\n\r");
}
```

4-21 Clk_TS_ToTS_NTP()

Convert Clock timestamp to NTP timestamp.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_TS_ToTS_NTP (CLK_TS_SEC   ts_sec,
                              CLK_TS_SEC  *p_ts_ntp_sec);
```

ARGUMENTS

`ts_sec` Timestamp to convert (in seconds, UTC+00).

`p_ts_ntp_sec` Pointer to variable that will receive the NTP timestamp:

In seconds UTC+00,	if no errors;
`CLK_TS_SEC_NONE`,	otherwise.

RETURNED VALUES

`DEF_OK`, if timestamp successfully converted.

`DEF_FAIL`, otherwise.

REQUIRED CONFIGURATION

Available only if `CLK_CFG_NTP_EN` is `DEF_ENABLED` in `clk_cfg.h` (see section 3-1-1).

NOTES / WARNINGS

Returned timestamp does not include any time zone offset. Thus any local time zone offset should be applied before or after calling `Clk_TS_ToTS_NTP()`.

Only years supported by Clock and NTP can be converted, thus the timestamp date must be greater than or equal to `CLK_EPOCH_YR_START` and less than `CLK_NTP_EPOCH_YR_END`.

EXAMPLE USAGE

```
CLK_TS_SEC   ts_sec;
CLK_TS_SEC   ts_ntp_sec
CPU_BOOLEAN  valid;

ts_sec = 0;
valid  = Clk_TS_ToTS_NTP(ts_sec, &ts_ntp_sec);
if (valid == DEF_OK) {
    printf("Timestamp = %u", ts_ntp_sec);
} else {
    printf("Convert timestamp error\n\r");
}
```

4-22 Clk_TS_NTP_ToTS()

Convert NTP timestamp to Clock timestamp.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_TS_NTP_ToTS (CLK_TS_SEC  *p_ts_sec,
                              CLK_TS_SEC   ts_ntp_sec);
```

ARGUMENTS

p_ts_sec Pointer to variable that will receive the Clock timestamp:

In seconds UTC+00,	if no errors;
CLK_TS_SEC_NONE,	otherwise.

ts_ntp_sec NTP timestamp value to convert (in seconds, UTC+00).

RETURNED VALUES

DEF_OK, if timestamp successfully converted.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

Available only if CLK_CFG_NTP_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1).

NOTES / WARNINGS

Returned timestamp does not include any time zone offset. Thus any local time zone offset should be applied before or after calling Clk_TS_NTP_ToTS().

Only years supported by Clock and NTP can be converted, thus the timestamp date must be greater than or equal to CLK_EPOCH_YR_START and less than CLK_NTP_EPOCH_YR_END.

EXAMPLE USAGE

```
CLK_TS_SEC   ts_sec;
CLK_TS_SEC   ts_ntp_sec
CPU_BOOLEAN  valid;

ts_ntp_sec = 1000000;
valid      = Clk_TS_NTP_ToTS(&ts_sec, ts_ntp_sec);
if (valid == DEF_OK) {
    printf("Timestamp = %u", ts_sec);
} else {
    printf("Convert timestamp error\n\r");
}
```

4-23 Clk_TS_NTP_ToDateTime()

Convert NTP timestamp to a date/time structure.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_TS_NTP_ToDateTime (CLK_TS_SEC      ts_ntp_sec,
                                    CLK_TZ_SEC      tz_sec,
                                    CLK_DATE_TIME  *p_date_time);
```

ARGUMENTS

ts_ntp_sec	Timestamp to convert (in seconds, UTC+00).
tz_sec	Time zone offset (in seconds, ± from UTC).
p_date_time	Pointer to variable that will receive the date/time structure.

RETURNED VALUES

DEF_OK, if timestamp successfully converted.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

Available only if CLK_CFG_NTP_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1).

NOTES / WARNINGS

Timestamp (ts_ntp_sec) must be set for UTC+00 and should not include the time zone offset (tz_sec) since Clk_TS_NTP_ToDateTime() includes the time zone offset in its date/time calculation. Thus the time zone offset should not be applied before or after calling Clk_TS_NTP_ToDateTime(). Time zone field of the date/time structure (p_date_time->TZ_sec) is set to the value of the time zone argument (tz_sec).

EXAMPLE USAGE

```
CLK_DATE_TIME  date_time;
CLK_TS_SEC     ts_ntp_sec;
CLK_TZ_SEC     tz_sec;
CPU_BOOLEAN    valid;

ts_ntp_sec =   1000000;
tz_sec     = -5 * 3600;
valid      = Clk_TS_NTP_ToDateTime(ts_ntp_sec, tz_sec, &date_time);
if (valid == DEF_OK) {
    printf("Timestamp successfully converted\n\r");
} else {
    printf("Timestamp conversion error\n\r");
}
```

4-24 Clk_DateTimeToTS_NTP()

Convert a date/time structure to NTP timestamp.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_DateTimeToTS_NTP (CLK_TS_SEC     *p_ts_ntp_sec,
                                   CLK_DATE_TIME  *p_date_time);
```

ARGUMENTS

p_ts_ntp_sec Pointer to variable that will receive the NTP timestamp:

In seconds UTC+00,	if no errors;
CLK_TS_SEC_NONE,	otherwise.

p_date_time Date/time structure to convert.

RETURNED VALUES

DEF_OK, if date/time structure successfully converted.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

Available only if CLK_CFG_NTP_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1).

NOTES / WARNINGS

Date/time structure (p_date_time) must be representable in NTP timestamp. Thus date to convert must be greater than or equal to CLK_NTP_EPOCH_YR_START and less than CLK_NTP_EPOCH_YR_END. Date/time should be set to local time with correct time zone offset (p_date_time->TZ_sec). Clk_DateTimeToTS_NTP() removes the time zone offset from the date/time to calculate and return an NTP timestamp at UTC+00.

EXAMPLE USAGE

```
CLK_TS_SEC     ts_ntp_sec;
CLK_DATE_TIME  date_time;
CPU_BOOLEAN    valid;

                                /* 2010/09/18   11:11:11 UTC-05:00  */
valid = Clk_DateTimeMake(&date_time, 2010, 9, 18, 11, 11, 11, -18000);
if (valid == DEF_OK) {
    printf("Date/time successfully created");
} else {
    printf("Clock Date/time error\n\r");
}

valid = Clk_DateTimeToTS_NTP(&ts_ntp_sec, &date_time);
if (valid == DEF_OK) {
    printf("Timestamp = %u", ts_ntp_sec);
} else {
    printf("Clock Date/time to NTP timestamp error\n\r");
}
```

4-25 Clk_NTP_DateTimeMake()

Build a valid NTP epoch date/time structure.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_NTP_DateTimeMake (CLK_DATE_TIME  *p_date_time,
                                   CLK_YR          yr,
                                   CLK_MONTH       month,
                                   CLK_DAY         day,
                                   CLK_HR          hr,
                                   CLK_MIN         min,
                                   CLK_SEC         sec,
                                   CLK_TZ_SEC      tz_sec);
```

ARGUMENTS

`p_date_time` Pointer to variable that will receive the date/time structure.

`yr` Year value [CLK_NTP_EPOCH_YR_START to CLK_NTP_EPOCH_YR_END].

`month` Month value [CLK_MONTH_JAN to CLK_MONTH_DEC].

`day` Day value [1 to 31].

`hr` Hours value [0 to 23].

`min` Minutes value [0 to 59].

`sec` Seconds value [0 to 60].

`tz_sec` Time zone offset (in seconds, ± from UTC) [-43200 to 43200].

RETURNED VALUES

`DEF_OK`, if date/time structure is valid.

`DEF_FAIL`, otherwise.

REQUIRED CONFIGURATION

Available only if `CLK_CFG_NTP_EN` is `DEF_ENABLED` in `clk_cfg.h` (see section 3-1-1).

NOTES / WARNINGS

Date/time structure (p_date_time) must be representable in NTP timestamp. Thus date to convert must be greater than or equal to CLK_NTP_EPOCH_YR_START and less than CLK_NTP_EPOCH_YR_END.

Day of week (p_date_time->DayOfWk) and Day of year (p_date_time->DayOfYr) are internally calculated and set in the date/time structure.

EXAMPLE USAGE

```
CLK_DATE_TIME  date_time;
CPU_BOOLEAN    valid;

                                      /* 2010/09/18   11:11:11 UTC-05:00  */
valid = Clk_NTP_DateTimeMake(&date_time, 2010, 9, 18, 11, 11, 11, -18000);
if (valid == DEF_OK) {
    printf("Date/time successfully created");
} else {
    printf("Clock Date/time error\n\r");
}
```

4-26 Clk_IsNTP_DateTimeValid()

Determine if date/time structure is representable in NTP epoch.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_IsNTP_DateTimeValid (CLK_DATE_TIME  *p_date_time);
```

ARGUMENTS

p_date_time Pointer to variable that contains the date/time structure to validate.

RETURNED VALUES

DEF_YES, if date/time structure is valid.

DEF_NO, otherwise.

REQUIRED CONFIGURATION

Available only if CLK_CFG_NTP_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1).

NOTES / WARNINGS

Date/time structure (p_date_time) must be representable in Clock timestamp. Thus date to validate must be greater than or equal to CLK_NTP_EPOCH_YR_START and less than CLK_NTP_EPOCH_YR_END.

EXAMPLE USAGE

```
CLK_TS_SEC     ts_sec;
CLK_DATE_TIME  date_time;
CPU_BOOLEAN    valid;

date_time.Yr      =   2010;     /* 2010/09/18  11:11:11 UTC-05:00  */
date_time.Month   =      9;
date_time.Day     =     18;
date_time.Hr      =     11;
date_time.Min     =     11;
date_time.Sec     =     11;
date_time.DayOfWk =      2;
date_time.DayOfYr =    291;
date_time.TZ_sec  = -18000;

valid =  Clk_IsDateTimeValidNTP(&date_time);
if (valid == DEF_OK) {
    printf("Date/time is valid");
} else {
    printf("Date/time is NOT valid");
}
```

4-27 Clk_GetTS_Unix()

Get current Clock timestamp as a Unix timestamp.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_GetTS_Unix (CLK_TS_SEC  *p_ts_unix_sec);
```

ARGUMENTS

p_ts_unix_sec Pointer to variable that will receive the Unix timestamp:

In seconds UTC+00,	if no errors;
CLK_TS_SEC_NONE,	otherwise.

RETURNED VALUES

DEF_OK, if timestamp successfully returned.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

Available only if CLK_CFG_UNIX_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1).

NOTES / WARNINGS

Unix timestamp does not include any time zone offset. Thus any local time zone offset must be applied after calling Clk_GetTS_Unix().

Unix timestamp will eventually overflow, thus it's not possible to get Unix timestamp for years on or after CLK_UNIX_EPOCH_YR_END.

EXAMPLE USAGE

```
CLK_TS_SEC   ts_unix_sec;
CPU_BOOLEAN  valid;

valid = Clk_GetTS_Unix(&ts_unix_sec);
if (valid == DEF_OK) {
    printf("Timestamp Unix = %u", ts_unix_sec);
} else {
    printf("Get TS Unix error\n\r");
}
```

4-28 Clk_SetTS_Unix()

Set Clock timestamp from a Unix timestamp.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_SetTS_Unix (CLK_TS_SEC  ts_unix_sec);
```

ARGUMENTS

ts_unix_sec Current Unix timestamp to set (in seconds, UTC+00).

RETURNED VALUES

DEF_OK, if timestamp successfully set.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

Available only if CLK_CFG_UNIX_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1).

NOTES / WARNINGS

Only years supported by Clock and Unix can be set, thus the timestamp date must be between greater than or equal to CLK_EPOCH_YR_START and less than CLK_UNIX_EPOCH_YR_END.

EXAMPLE USAGE

```
CLK_TS_SEC   ts_unix_sec;
CPU_BOOLEAN  valid;

ts_ntp_sec = 20200020;
valid     = Clk_SetTS_Unix(&ts_unix_sec);
if (valid == DEF_OK) {
    printf("Timestamp successfully set");
} else {
    printf("Set timestamp error\n\r");
}
```

4-29 Clk_TS_ToTS_Unix()

Convert Clock timestamp to Unix timestamp.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_TS_ToTS_Unix (CLK_TS_SEC   ts_sec,
                               CLK_TS_SEC  *p_ts_unix_sec);
```

ARGUMENTS

`ts_sec` Timestamp to convert.

`p_ts_unix_sec` Pointer to variable that will receive the Unix timestamp:

In seconds UTC+00,	if no errors;
`CLK_TS_SEC_NONE`,	otherwise.

RETURNED VALUES

`DEF_OK`, if timestamp successfully converted.

`DEF_FAIL`, otherwise.

REQUIRED CONFIGURATION

Available only if `CLK_CFG_UNIX_EN` is `DEF_ENABLED` in `clk_cfg.h` (see section 3-1-1).

NOTES / WARNINGS

Returned timestamp does not include any time zone offset. Thus any local time zone offset should be applied before or after calling `Clk_TS_ToTS_Unix()`.

Only years supported by Clock and Unix can be converted, thus the timestamp date must be greater than or equal to `CLK_EPOCH_YR_START` and less than `CLK_UNIX_EPOCH_YR_END`.

EXAMPLE USAGE

```
CLK_TS_SEC   ts_sec;
CLK_TS_SEC   ts_unix_sec
CPU_BOOLEAN  valid;

ts_sec = 0;
valid  = Clk_TS_ToTS_Unix(ts_sec, &ts_unix_sec);
if (valid == DEF_OK) {
    printf("Timestamp = %u", ts_unix_sec);
} else {
    printf("Convert timestamp error\n\r");
}
```

4-30 Clk_TS_UnixToTS()

Convert Unix timestamp to Clock timestamp.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_TS_UnixToTS (CLK_TS_SEC  *p_ts_sec,
                              CLK_TS_SEC   ts_unix_sec);
```

ARGUMENTS

p_ts_sec Pointer to variable that will receive the Clock timestamp:

In seconds UTC+00,	if no errors;
CLK_TS_SEC_NONE,	otherwise.

ts_unix_sec Unix timestamp value to convert (in seconds, UTC+00).

RETURNED VALUES

DEF_OK, if timestamp successfully converted.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

Available only if CLK_CFG_UNIX_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1).

NOTES / WARNINGS

Returned timestamp does not include any time zone offset. Thus any local time zone offset should be applied before or after calling Clk_TS_UnixToTS().

Only years supported by Clock and Unix can be converted, thus the timestamp date must be greater than or equal to CLK_EPOCH_YR_START and less than CLK_UNIX_EPOCH_YR_END.

EXAMPLE USAGE

```
CLK_TS_SEC   ts_sec;
CLK_TS_SEC   ts_unix_sec
CPU_BOOLEAN  valid;

ts_unix_sec = 1000000;
valid       = Clk_TS_UnixToTS(&ts_sec, ts_unix_sec);
if (valid == DEF_OK) {
    printf("Timestamp = %u", ts_sec);
} else {
    printf("Convert timestamp error\n\r");
}
```

4-31 Clk_TS_UnixToDateTime()

Convert Unix timestamp to a date/time structure.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_TS_UnixToDateTime (CLK_TS_SEC      ts_unix_sec,
                                    CLK_TZ_SEC      tz_sec,
                                    CLK_DATE_TIME  *p_date_time);
```

ARGUMENTS

ts_unix_sec	Timestamp to convert (in seconds, UTC+00).
tz_sec	Time zone offset (in seconds, ± from UTC).
p_date_time	Pointer to variable that will receive the date/time structure.

RETURNED VALUES

DEF_OK, if date/time structure successfully returned.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

Available only if CLK_CFG_UNIX_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1).

NOTES / WARNINGS

Timestamp (ts_unix_sec) must be set for UTC+00 and should not include the time zone offset (tz_sec) since Clk_TS_UnixToDateTime() includes the time zone offset in its date/time calculation. Thus the time zone offset should not be applied before or after calling Clk_TS_UnixToDateTime(). Time zone field of the date/time structure (p_date_time->TZ_sec) is set to the value of the time zone argument (tz_sec).

EXAMPLE USAGE

```
CLK_DATE_TIME  date_time;
CLK_TS_SEC     ts_unix_sec;
CLK_TZ_SEC     tz_sec;
CPU_BOOLEAN    valid;

ts_unix_sec =   1000000;
tz_sec      = -5 * 3600;
valid       = Clk_TS_UnixToDateTime(ts_unix_sec, tz_sec, &date_time);
if (valid == DEF_OK) {
    printf("Timestamp successfully converted\n\r");
} else {
    printf("Timestamp conversion error\n\r");
}
```

4-32 Clk_DateTimeToTS_Unix()

Convert a date/time structure to Unix timestamp.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_DateTimeToTS_Unix (CLK_TS_SEC     *p_ts_unix_sec,
                                    CLK_DATE_TIME  *p_date_time);
```

ARGUMENTS

p_ts_unix_sec Pointer to variable that will receive the Unix timestamp:

In seconds UTC+00,	if no errors;
CLK_TS_SEC_NONE,	otherwise.

p_date_time Date/time structure to convert.

RETURNED VALUES

DEF_OK, if date/time structure successfully converted.

DEF_FAIL, otherwise.

REQUIRED CONFIGURATION

Available only if CLK_CFG_UNIX_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1).

NOTES / WARNINGS

Date/time structure (`p_date_time`) must be representable in Unix timestamp. Thus date to convert must be greater than or equal to `CLK_UNIX_EPOCH_YR_START` and less than `CLK_UNIX_EPOCH_YR_END`. Date/time should be set to local time with correct time zone offset (`p_date_time->TZ_sec`). `Clk_DateTimeToTS_Unix()` removes the time zone offset from the date/time to calculate and return an Unix timestamp at UTC+00.

EXAMPLE USAGE

```
CLK_TS_SEC     ts_unix_sec;
CLK_DATE_TIME  date_time;
CPU_BOOLEAN    valid;

                                  /* 2010/09/18   11:11:11 UTC-05:00  */
valid = Clk_DateTimeMake(&date_time, 2010, 9, 18, 11, 11, 11, -18000);
if (valid == DEF_OK) {
    printf("Date/time successfully created");
} else {
    printf("Clock Date/time error\n\r");
}

valid = Clk_DateTimeToTS_Unix(&ts_unix_sec, &date_time);
if (valid == DEF_OK) {
    printf("Timestamp = %u", ts_unix_sec);
} else {
    printf("Clock Date/time to NTP timestamp error\n\r");
}
```

4-33 `Clk_UnixDateTimeMake()`

Build a valid Unix epoch date/time structure.

FILES

`clk.h/clk.c`

PROTOTYPE

```
CPU_BOOLEAN  Clk_UnixDateTimeMake (CLK_DATE_TIME  *p_date_time,
                                   CLK_YR          yr,
                                   CLK_MONTH       month,
                                   CLK_DAY         day,
                                   CLK_HR          hr,
                                   CLK_MIN         min,
                                   CLK_SEC         sec,
                                   CLK_TZ_SEC      tz_sec);
```

ARGUMENTS

`p_date_time` Pointer to variable that will receive the date/time structure.

`yr` Year value [CLK_UNIX_EPOCH_YR_START to CLK_UNIX_EPOCH_YR_END).

`month` Month value [CLK_MONTH_JAN to CLK_MONTH_DEC].

`day` Day value [1 to 31].

`hr` Hours value [0 to 23].

`min` Minutes value [0 to 59].

`sec` Seconds value [0 to 60].

`tz_sec` Time zone offset (in seconds, ± from UTC) [-43200 to 43200].

RETURNED VALUES

`DEF_OK`, if date/time structure successfully returned.

`DEF_FAIL`, otherwise.

REQUIRED CONFIGURATION

Available only if `CLK_CFG_UNIX_EN` is `DEF_ENABLED` in `clk_cfg.h` (see section 3-1-1).

NOTES / WARNINGS

Date/time structure (p_date_time) must be representable in Unix timestamp. Thus date to convert must be greater than or equal to CLK_UNIX_EPOCH_YR_START and less than CLK_UNIX_EPOCH_YR_END.

Day of week (p_date_time->DayOfWk) and Day of year (p_date_time->DayOfYr) are internally calculated and set in the date/time structure.

EXAMPLE USAGE

```
CLK_DATE_TIME  date_time;
CPU_BOOLEAN    valid;

                                      /* 2010/09/18   11:11:11 UTC-05:00  */
valid = Clk_UnixDateTimeMake(&date_time, 2010, 9, 18, 11, 11, 11, -18000);
if (valid == DEF_OK) {
    printf("Date/time successfully created");
} else {
    printf("Clock Date/time error\n\r");
}
```

4-34 Clk_IsUnixDateTimeValid()

Determine if date/time structure is representable in Unix epoch.

FILES

clk.h/clk.c

PROTOTYPE

```
CPU_BOOLEAN  Clk_IsUnixDateTimeValid (CLK_DATE_TIME  *p_date_time);
```

ARGUMENTS

p_date_time Pointer to variable that contains the date/time structure to validate.

RETURNED VALUES

DEF_YES, if date/time structure is valid.

DEF_NO, otherwise.

REQUIRED CONFIGURATION

Available only if CLK_CFG_UNIX_EN is DEF_ENABLED in clk_cfg.h (see section 3-1-1).

NOTES / WARNINGS

Date/time structure (p_date_time) must be representable in Clock timestamp. Thus date to validate must be greater than or equal to CLK_UNIX_EPOCH_YR_START and less than CLK_UNIX_EPOCH_YR_END.

EXAMPLE USAGE

```
CLK_TS_SEC     ts_sec;
CLK_DATE_TIME  date_time;
CPU_BOOLEAN    valid;

date_time.Yr      =   2010;     /* 2010/09/18  11:11:11 UTC-05:00  */
date_time.Month   =      9;
date_time.Day     =     18;
date_time.Hr      =     11;
date_time.Min     =     11;
date_time.Sec     =     11;
date_time.DayOfWk =      2;
date_time.DayOfYr =    291;
date_time.TZ_sec  = -18000;

valid = Clk_IsUnixDateTimeValid(&date_time);
if (valid == DEF_OK) {
    printf("Date/time is valid");
} else {
    printf("Date/time is NOT valid");
}
```

Appendix A

µC/Clk Licensing Policy

You need to obtain an "Object Code Distribution License" to embed µC/Clk in a product that is sold with the intent to make a profit. Each individual product (*i.e.*, your product) requires its own license, but the license allows you to distribute an unlimited number of units for the life of your product. Please indicate the processor type(s) (*i.e.,* ARM7, ARM9, MCF5272, MicroBlaze, Nios II, PPC, *etc.*) that you intend to use.

For licensing details, contact us at:

Micrium
1290 Weston Road, Suite 306
Weston, FL 33326
USA

Phone: +1 954 217 2036
Fax: +1 954 217 2037
E-mail: Licensing@Micrium.com
Web: www.Micrium.com

Appendix B

References

Labrosse, Jean J. *µC/OS-III, The Real-Time Kernel*, Micriμm Press, 2009.

Labrosse, Jean J. *MicroC/OS-II: The Real Time Kernel*. 2nd edition. Newnes, 2002.

Labrosse, Jean J. *Embedded Systems Building Blocks: Complete and Ready-to-Use Modules in C*. 2nd Edition. R&D Technical Books, 2000.

Viscogliosi, Roberto R. *C shortcuts and the day of the week*. PC Magazine, May 11, 1993; pg. 396, 401 & 406.

Latham, Lance. *Standard C Date/Time Library: Programming the World's Calendars and Clocks*. R&D Books, 1999.