Documentation of the library
WagoAppTime
Release 1.7.3.3
# Contents

## 1 Description

## 2 10 Documentation

### 2.1 doc10_General (FB)

### 2.2 doc91_Notes (FB)

#### 2.2.1 Different Sources of Time

#### 2.2.2 Data Types

| TIME | 4 |
| DATE_AND_TIME | 4 |
| LTIME | 4 |
| TIME_OF_DAY | 5 |
| DATE | 5 |
| SYSTIME | 5 |
| RTS_SYSTIMEDATE | 6 |
| typRTS_SYSTIMEDATE | 6 |
| typWagoTimeComponents | 6 |

### 2.3 doc95_Specifications (FB)

## 3 20 Program Organization Units

### 3.1 00 Calculations

#### 3.1.1 01 Date (DATE)

| FuDifferenceDATE (FUN) | 8 |
| FuSubtractDATE (FUN) | 8 |

#### 3.1.2 02 Time of day (TOD)

| FuTimeDifference (FUN) | 9 |

#### 3.1.3 03 Date and time (DT)

| FuDifferenceDT (FUN) | 9 |
| FuSubtractDT (FUN) | 10 |

#### 3.1.4 04 LTime

| FuGetLTimeDifferenceNanoseconds (FUN) | 10 |
| FuGetTimeDifferenceMilliseconds (FUN) | 11 |

### 3.2 01 System Timestamps UTC

#### 3.2.1 FuGetDateAndTime (FUN)

#### 3.2.2 FuGetLongSystemTime (FUN)

#### 3.2.3 FuGetLongTime (FUN)

#### 3.2.4 FuGetSystemTime (FUN)

### 3.3 02 System Timestamps Local

#### 3.3.1 FuGetLocalDateAndTime (FUN)

### 3.4 03 Timezone

#### 3.4.1 10 Compact

| FbTimeZoneSetter_cpt (FB) | 14 |

#### 3.4.2 30 Base

| 14 |
3.5 04 Calendar

3.5.1 00 Formatting

01 Date (DATE)
- FuDateAdjustment (FUN)
- FuFormatToDATE (FUN)

02 Time of day (TOD)
- FuFormatToTOD (FUN)
- FuTimeAdjustment (FUN)

03 Date and time (DT)
- FuDateTimeAdjustment (FUN)
- FuFormatToDT (FUN)
- FuSetDT (FUN)

3.5.2 01 Extraction

01 Date (DATE)
- FuDay (FUN)
- FuMonth (FUN)
- FuYear (FUN)

02 Time of day (TOD)
- FuHour (FUN)
- FuMinute (FUN)
- FuSecond (FUN)

3.5.3 02 Conversions

01 Date and Time
- FuConvertStampToLocalTimeComponents (FUN)
- FuConvertStampToTimeComponents (FUN)
- FuConvertTimeComponentsToStamp (FUN)

02 LTime
- FuConvertLStampToLocalTimeComponents (FUN)
- FuConvertLStampToTimeComponents (FUN)
- FuConvertTimeComponentsToLStamp (FUN)

3.5.4 03 Weekday

- FuAnyDayOfWeek (FUN)
- FuNextWeekday (FUN)
- FuPreviousWeekday (FUN)
- FuWeekDay_ISO8601 (FUN)
- FuWeekdayOfMonth (FUN)
- FuWeekdayOfYear (FUN)

3.5.5 04 Leap year

- FuIs29Feb (FUN)
- FuIsLeapYear (FUN)

3.5.6 05 Calendar Week

- FuCalendarWeek_ISO8601 (FUN)

3.5.7 06 Number of days

- FuDaysSinceJan01 (FUN)
- FuNumDaysYear (FUN)

3.5.8 07 Month

- FuCurrMonthWeek (FUN)
- FuDaysInMonth (FUN)
- FuMonthBegin (FUN)
- FuMonthEnd (FUN)

3.5.9 08 Year
CHAPTER 1

Description

This document is automatically generated. Because of this, the chapter 30 Visualization is not shown in this document. If you are interested in getting to know more about visualization, we refer to the library manager of e!Cockpit.

Subject to Changes

WAGO Kontakttechnik GmbH Co. KG reserves the right to provide for any alterations or modifications. WAGO Kontakttechnik GmbH Co. KG owns all rights arising from the granting of patents or from the legal protection of utility patents. Third-party products are always mentioned without any reference to patent rights. Thus, the existence of such rights cannot be excluded.

Personnel Qualification

All tasks that are carried out with libraries made for the e!COCKPIT software must only be performed by qualified electrical specialists instructed in PLC programming according to IEC 61131-3.

All tasks that have an effect on the properties or the behavior of automation hardware or software products must only be performed by qualified employees with a thorough knowledge of handling the products concerned.

Intended Use of e!COCKPIT Libraries

Libraries created for the e!COCKPIT software are used to simplify the development of application projects in the IEC 61131-3 programming languages.

For automation tasks, WAGO offers programmable logic controllers in a wide variety of performance classes. In combination with a wide range of I/O modules, the controllers can process standard types of field signals. Controllers can be implemented centrally or in decentralized configurations. The controllers offer interfaces for the most commonly used fieldbuses for use in decentralized configurations. Fieldbus independent I/O modules are then linked via fieldbus couplers. WAGO controllers offer a runtime environment for user programs called e!RUNTIME. Software projects for implementation in e!RUNTIME environments can be created in e!COCKPIT. The programming environment in e!COCKPIT is based on the established CODESYS 3 industrial standard. Users with a previous knowledge of CODESYS 3 will thus find this environment largely familiar. The following programming languages of the IEC 61131-3 standard are available:

- Structured Text (ST)
- Ladder Diagram (LD)
- Function Block Diagram (FBD)
- Instruction List (IL)
- Sequential Function Chart (SFC)
- Continuous Function Chart (CFC)

The individual programming languages can also be combined as required during the development of the software. A portfolio of prepared libraries can be accessed for many frequently used functions in order to make software development more efficient. This document provides an overview of the WagoAppTime that WAGO offers for e!COCKPIT.
Handling Time and Timezones

Further library information are summerized here:

Company WAGO
Title WagoAppTime
Version 1.7.3.3
Categories WAGO LayerView|App; WAGO FunctionalView|Base; Application
Author WAGO / u013972
Placeholder WagoAppTime
2.1 doc10_General (FB)

General Preface

This library contains functions, types definitions, and function blocks for handling time and timestamps of the system.

Handling of system time and RTC-time is merged to one unified absolute time here, so the user does not have to consider different time sources here.

Several general topics are covered:
1. Getting timestamps
2. Perform typical operations and conversions with them
3. Setting the system time (including implicitly setting the RTC).
4. Handling timezones and automatical switching of Daylight-Saving (DST)
5. Convenience routines, such as transformation of time into calendar components, measuring of intervals, timeouts, etc.

Unconventionally, there are no different routines for realtime and systemtime in this library. This is because both are unified on Wago PLCs. The data type LTIME is used for both high resolution calculation and also for representing absolute time. Nevertheless, the standard types for time are of course also supported.

See below in the section 'Supplementary Notes' for more details.

2.2 doc91_Notes (FB)

Supplementary Notes

2.2.1 Different Sources of Time

In a modern processing environment, there are different internal sources for time information, namely the Real-Time-Clock (RTC), the CPU-Clock, perhaps also an external NTP connection to an atomic time standard. Each of these sources is subject to drifting apart from the readings of other sources. That situation would be a little bit difficult to handle for the final application programmer.

To overcome that problem, all time sources are combined by internal software layers to one single system time. The RTC for example serves for initial setting of the system time and for continuously adjusting the system time in order to keep it synchronized with the global world time. (Also an NTP connection might be employed for this purpose) The registers of the RTC are never read or written to directly, but only indirectly when modifying the combined system time.
The system time is guaranteed to increase monotonically and to be readjusted smoothly according to time references with high precision (such as RTC or an NTP connection).

When system time is set, the RTC is always also set immediately.

RTC and system time are always kept internally in UT (i.e. GMT+0:00), not in local time, which makes it easy to smoothly adjust the system time. When the timezone is switched or DST is toggled, this operation operates only on a timezone offset.

### 2.2.2 Data Types

In the context of handling time and date, a certain variety of data types has been established in the recent years, so it is worth getting a kind of overview about this issue.

We will concentrate on those data types, which give most benefit to the user.

#### TIME

**Implementation:** 32-bit, unsigned

**Range:** 0 .. 49 days, relative

**Resolution:** 1 millisecond

**General application:** Fast computation of short time intervals.

**Attention:** Due to the lack of sign, this data type cannot handle general differences of timestamps. When this is required, the TIME value has to be converted to a signed type before further evaluation can take place.

**Application in this library:** This type is used as standard type in this library.

#### DATE_AND_TIME

**Implementation:** 32-bit, unsigned

**Range:** year 1970..2106

**Resolution:** 1 second

**Application:** Representation of absolute POSIX-Time

**Attention (1):** Strict compatibility to POSIX-Time is given only up to the year 2038 (POSIX-Time is signed).

**Attention (2):** Differences of timestamps of this type are virtually useless, because Codesys compiles these differences into the type TIME, which will silently overflow at differences greater than 49 days. Due to the lack of sign, this data type cannot handle general differences of timestamps anyway.

When computing differences is required, type conversion to signed 32-bit values has to be performed before computing the difference.

**Application in this library:** This type is used as standard type in this library.

#### LTIME

**Implementation:** 64-bit, unsigned

**Range:** 0 .. 584 years

**Resolution:** 1 nanosecond

**Application:** Handling of time intervals with high resolution. Regarding the high data range, this type is also suitable to handle absolute time stamps beyond the limitations of POSIX-Time. When using this type with sign awareness, dates from year 1678 up to year 2262 can be represented.
Attention: Due to the lack of sign, this data type cannot handle general differences of timestamps. When this is required, the TIME value has to be converted to a signed type before further evaluation can take place.

Application in this library: This type is used as standard type in this library. Its semantic is extended, as it will also be used for absolute representation of date and time with nanosecond resolution. When interpreted as absolute value, it will be interpreted as time interval since 1.1.1970, resembling POSIX time and the standard type DATE_AND_TIME.

**TIME_OF_DAY**

Implementation: 32-bit, unsigned

Range: 0:00:00 .. 23:59:59.999

Resolution: 1 millisecond

Application: Representing the local time, but practically redundant, because TIME covers the same range and the same resolution without the restrictions of this data type and RTS_SYSTIMEDATE is more appropriate when separated time components are required.

Attention: Due to the lack of sign, this data type cannot handle general differences of timestamps. When this is required, the TIME value has to be converted to a signed type before further evaluation can take place.

Application in this library: This type is not used in this library, because it makes sense only in calendarial contexts (where timezones are considered) and this field is covered by the structures typRTS_SYSTIMEDATE and typWagoTimeComponents.

**DATE**

Implementation: 32-bit, unsigned

Range: 1.1.1970 .. 7.2.2106

Resolution: 1 second

Application: Representing the local date. But it is practically redundant, because DATE_AND_TIME covers the same range and the same resolution without the restrictions of this data type. RTS_SYSTIMEDATE would be more appropriate when separated time components are required.

Attention (1): Strict compatibility to POSIX-Time is given only up to the year 2038 (POSIX-Time is signed)

Attention (2): Differences of timestamps if this type are virtually useless, because Codesys compiles these differences into the type TIME, which will silently overflow at differences greater than 49 days. Due to the lack of sign, this data type cannot handle general differences of timestamps anyway.

When computing differences is required, type conversion to signed 32-bit values has to be performed before computing the difference.

Application in this library: This type is not used in this library, because it makes sense only in calendarial contexts (where timezones are considered) and this field is covered by the structures typRTS_SYSTIMEDATE and typWagoTimeComponents.

**SYSTIME**

Implementation: 64-bit, unsigned

Range: <not defined>

Resolution: 1 system clock (SysTimeCore.library)

Application: fast computing of positive time differences, absolute timestamps for internal purposes

Attention (1): The application of this type is pretty inconsistent in this library. In different places it represents either system ticks, milliseconds, microseconds, or nanoseconds. There is no uniform calculation from this data type to real dimensions e.g. ‘seconds’.
Attention (2): Due to the lack of sign, this data type cannot handle general differences of timestamps. When this is required, the TIME value has to be converted to a signed type before further evaluation can take place.

Application in this library: This type and its variations are ubiquitous on system level, but it is not used in this library, because it is defined too ambiguously.

RTS_SYSTIMEDATE

Implementation: Composed structure in SysTimeRtc.library.

Range: Year 1 .. year 65535

Resolution: 1 millisecond

Application: Separation of different components of local timestamp (year, month, day, day of week) e.g. for printing purpose. This structure is not well suitable for further computing.

Attention: When leaving UTC and going to local time, the representation of the local time becomes ambiguous: When switching from daylight-saving-time DST back to standard time, the local time happens to pass through two adjacent hours which are both called 2o’clock but which are distinguished by calling them 02a:00 (DST) followed by 02b:00 (standard time), which is finally followed by 03:00.

There is no mechanism employed to distinguish between both 2 o’clock hours, so when used for setting the local time, this type does not cover all possibilities.

N.B.: Codesys conversions cover conversions from DATE_AND_TIME to UTC, but they regard conversions from and to local time as obsolete for this reason (but without telling any replacement).

Attention (2): This type is defined in different ways in different versions of the base library, SysTimeRtc, so there is at the moment no way of portably extending this struct in child libraries.

Application in this library: This type is not used directly in this library, because it is defined ambiguously within different versions of system libraries. It is used as template for a uniformly defined typRTS_SYSTIMEDATE.

typRTS_SYSTIMEDATE

Implementation: This is a struct where the components are defined in exactly the same way as in SysTimeRtc.RTS_SYSTIMEDATE or SysTimeRtc.SYSTIMEDATE resp. but which is declared without version ambiguities, so it is possible to portably extend this structure. For details see ‘typRTS_SYSTIMEDATE’ in the appendix ‘Types’.

Range: Year 1 .. year 65535

Resolution: 1 millisecond

Application: Work around for providing a structure which can be used in exactly the same manner and with the same interface as the established (RTS_)SYSTIMEDATE, but without the ambiguity of versions. Other struct are derived from this base struct.

typWagoTimeComponents

Implementation: Struct derived from typRTS_SYSTIMEDATE, extended by more information fields. For details see ‘typWagoTimeComponents’ in the appendix ‘Types’.

Range: Year 1 .. year 65535

Resolution: 1 nanosecond

Application: Representation of components of time, mainly for printing purpose. As this struct is extended from typRTS_SYSTIMEDATE, the base variable type may still be used in applications, because assignment to the base variables will not require any conversion.
By this means, compatibility to legacy code which still needs the 3S-SysTimeRtc-typedefs is ensured.

2.3  doc95_Specifications (FB)

Specifications

This library refers to the following documents and specifications:

1. ISO 8601 Counting of Calendar Weeks
### 3.1 00 Calculations

#### 3.1.1 01 Date (DATE)

**FuDifferenceDATE (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuDifferenceDATE</td>
<td>DINT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate1</td>
<td>DATE</td>
<td>date 1</td>
</tr>
<tr>
<td></td>
<td>datDate2</td>
<td>DATE</td>
<td>date 2</td>
</tr>
</tbody>
</table>

**Function**

Calculate difference between 2 dates and the result is in days. Result is always positive.

**Note:** Value overflow may occur if the calculated result is outside the range of a DINT value (-2147483648 ... 2147483647).

**Graphical Illustration**

![Graphical Illustration](image)

**FuSubtractDATE (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuSubtractDATE</td>
<td>DINT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate1</td>
<td>DATE</td>
<td>date 1</td>
</tr>
<tr>
<td></td>
<td>datDate2</td>
<td>DATE</td>
<td>date 2</td>
</tr>
</tbody>
</table>

**Function**
Subtract 2 dates and the result is in days. Negative values indicates day before.

**Note:** Value overflow may occur if the calculated result is outside the range of a DINT value (-2147483648 ... 2147483647).

### Graphical Illustration

#### 3.1.2 02 Time of day (TOD)

**FuTimeDifference (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuTimeDifference</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>tdTime1</td>
<td>TOD</td>
<td>time 1</td>
</tr>
<tr>
<td></td>
<td>tdTime2</td>
<td>TOD</td>
<td>time 2</td>
</tr>
</tbody>
</table>

**Function**

Calculate difference between 2 dates TOD and the result is in seconds. Results is always positive.

**Graphical Illustration**

#### 3.1.3 03 Date and time (DT)

**FuDifferenceDT (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuDifferenceDT</td>
<td>DINT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>dtDateTime1</td>
<td>DT</td>
<td>date time 1</td>
</tr>
<tr>
<td></td>
<td>dtDateTime2</td>
<td>DT</td>
<td>date time 2</td>
</tr>
</tbody>
</table>

**Function**

Calculate difference between 2 DT and the result is in seconds. Result is always positive.
**Note:** Value overflow may occur if the calculated result is outside the range of a DINT value (-2147483648 .... 2147483647).

**Graphical Illustration**

![Graphical Illustration of FuSubtractDT](image)

**FuSubtractDT (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuSubtractDT</td>
<td>DINT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>dtDateTime1</td>
<td>DT</td>
<td>date time 1</td>
</tr>
<tr>
<td></td>
<td>dtDateTime2</td>
<td>DT</td>
<td>date time 2</td>
</tr>
</tbody>
</table>

**Function**

Subtraction between 2 DT and the result is in seconds (DINT). Negative value indicates seconds before.

**Note:** Value overflow may occur if the calculated result is outside the range of a DINT value (-2147483648 .... 2147483647).

**Graphical Illustration**

![Graphical Illustration of FuSubtractDT](image)

3.1.4 04 LTime

**FuGetLTimeDifferenceNanoseconds (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuGetLTimeDifferenceNanoseconds</td>
<td>LINT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>ltActual</td>
<td>LTIME</td>
<td>actual timestamp</td>
</tr>
<tr>
<td></td>
<td>ltReference</td>
<td>LTIME</td>
<td>reference timestamp</td>
</tr>
</tbody>
</table>

Retrieves the difference of two ‘long’ timestamps (actual - reference) in nanoseconds

**Graphical Illustration**

![Graphical Illustration of FuSubtractDT](image)
Function FuGetTimeDifferenceMilliseconds (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuGetTimeDifferenceMilliseconds</td>
<td>DINT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>tActual</td>
<td>TIME</td>
<td>actual timestamp</td>
</tr>
<tr>
<td></td>
<td>tReference</td>
<td>TIME</td>
<td>reference timestamp</td>
</tr>
</tbody>
</table>

Retrieves the difference of two timestamps (actual-reference) in milliseconds

Graphical Illustration

Function Description

The result of the difference may be positive or negative, depending upon if the ‘actual’ stamp is later or earlier than the ‘reference’. As it is likely to encounter wrap-arounds of the 32-Bit value throughout the runtime of a system, differences which are larger than ~24 days are regarded as negative differences ('earlier') while others are regarded as positive differences ('later').

Due to the way the 32-bit-wrap-around is handled and the sign-uncertainty, overflow conditions are not detected in this function.

Usage:

```plaintext
VAR
  tEvent_A : TIME := T#02H;
  tEvent_B : TIME := T#01H;
  tDiff : DINT;
END_VAR;

tDiff := FuGetTimeDifferenceMilliseconds(tEvent_B,tEvent_A);
IF tDiff>0 THEN
```

3.1.00 Calculations
3.2 01 System Timestamps UTC

Operations which are related to obtaining timestamps and to performing calculations on them.

3.2.1 FuGetDateAndTime (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuGetDateAndTime</td>
<td>DT</td>
</tr>
</tbody>
</table>

Retrieves the absolute POSIX time (in UT, i.e. GMT+0000)

Graphical Illustration

Function

<table>
<thead>
<tr>
<th>FuGetDateAndTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT FuGetDateAndTime</td>
</tr>
</tbody>
</table>

Function Description

This function yields the actual POSIX-Time. This is traditionally used for timestamps in file systems and for calendar issues.

Note: The returned value is always given in UT / GMT, never in local time. For handling local timezones, use one of the functions which are summarized in section ‘Calendar’ below and which include ‘LocalTime’ in their name.

3.2.2 FuGetLongSystemTime (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuGetLongSystemTime</td>
<td>LTIME</td>
</tr>
</tbody>
</table>

Retrieves the nanoseconds since system start

Graphical Illustration

3.2. 01 System Timestamps UTC
Function

FuGetLongSystemTime

Function Description

Use this function for calculating intervals where it is important that any setting of the clock must not have any effect on that calculation.

In contrast to ‘FuGetLongTime()’, the return value of ‘FuGetLongSystemTime()’ is not related to an absolute calendar time, but to the start of the system. Consequently, any setting of the time has no effect on ‘FuGetLongSystemTime()’ (but would have on ‘FuGetLongTime()’).

This is important for implementing strictly monotonic timecounting in device drivers and related applications.

3.2.3 FuGetLongTime (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuGetLongTime</td>
<td>LTIME</td>
</tr>
</tbody>
</table>

Retrieves the global time (in UT, i.e. GMT+0000) since 1.1.1970 in LTIME format

Graphical Illustration

Function

FuGetLongTime

LTIME FuGetLongTime

Function Description

The time format LTIME combines the representation of absolute time (in millisecond resolution) and large value range (584 years). It is uniformly applicable without having to worry about overflow or reference points.

Note: This Funktion returns the absolute time in millisecond resolution. For a resolution in ns use ‘FuGetLongSystemTime()’

3.2.4 FuGetSystemTime (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuGetSystemTime</td>
<td>TIME</td>
</tr>
</tbody>
</table>

Retrieves the system time.

Graphical Illustration

Function

FuGetSystemTime

TIME FuGetSystemTime
Function Description

This function is useful for short term time interval with medium precision. It yields the 32-bit standard system time in milliseconds.

Note(1): This timestamp overflows after 49 days and it does not provide absolute time reference. If any of that is a problem, use FuGetLongTime (FUN) instead.

Note(2): This function is identical to TIME(). It is included here for the sake of completeness and of easier access.

3.3 02 System Timestamps Local

3.3.1 FuGetLocalDateAndTime (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuGetLocalDateAndTime</td>
<td>DT</td>
<td></td>
</tr>
</tbody>
</table>

Retrieves the Local-Time (i.e. GMT + Timezone)

Graphical Illustration

Function Description

This function yields the Local-Time.

Note: The returned value is given in local time.

Note: The change of the timezone over the WBM take effect after a reboot.

3.4 03 Timezone

3.4.1 10 Compact

FbTimeZoneSetter_cpt (FB)

Interface variables
This function block is a time zone setter for use in graphical languages.

**Graphical Illustration**

![Function Block Diagram](image)

**Functional description**

The implementation is exactly identical to the method oriented FB `FbTimeZoneSetter (FB)`, but the behaviour model is variable oriented.

Apply the desired time zone identification to the `sDatabaseKey`-input and start the FB via `xExecute` according to the WagoExecute behaviour model.

More details about valid time zone identification strings are given in section ‘Configuration of Time and Date - TZ String’ of the PFC200 Manual.


**3.4.2 30 Base**

**FbTimeZoneSetter (FB)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
<th>Inherited from</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td><code>xTerminate</code></td>
<td>BOOL</td>
<td>Indicates completion of the action.</td>
<td>FbBehaviour-Model_WagoMethodStart</td>
</tr>
<tr>
<td></td>
<td><code>xBusy</code></td>
<td>BOOL</td>
<td>Indicates running action.</td>
<td>FbBehaviour-Model_WagoMethodStart</td>
</tr>
<tr>
<td></td>
<td><code>xError</code></td>
<td>BOOL</td>
<td>Indicates an error.</td>
<td>FbBehaviour-Model_WagoMethodStart</td>
</tr>
<tr>
<td></td>
<td><code>eStatus</code></td>
<td>eResult-Code</td>
<td>State of processing</td>
<td>FbBehaviour-Model_WagoMethodStart</td>
</tr>
</tbody>
</table>

Fb for setting the time Zone.
Graphical Illustration

![Function Block Diagram]

Functional description

Set the timezone by calling the method ‘SetTimeZone(sDataBaseKey)’ with the desired time zone key. Then wait until xBusy returns to FALSE again.

**Note:** While xBusy is still TRUE, calls of SetTimeZone() will be ignored.

<table>
<thead>
<tr>
<th>result codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>success</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The database key is unknown to the system</td>
</tr>
<tr>
<td>EUNSPECIFIC</td>
<td>Unexpected other errors from OS</td>
</tr>
</tbody>
</table>

**FbTimeZoneSetter.SetTimeZone (METH)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>SetTimeZone</td>
<td>Wago-Types.eResultCode</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>sDataBaseKey</td>
<td>STRING</td>
<td>The applied data base key, e.g. ‘America/Buenos_Aires’</td>
</tr>
</tbody>
</table>

Changes the actual timezone

Graphical Illustration

![Method Diagram]

Functional description

The method takes a key string for the desired time zone as argument.

In the WAGO approach, timezones are not intended to be constructed or defined at IEC level. Instead, the lower layers of the PLC hold a database which contains well-established time zone descriptions for each part of the earth. The argument ‘sDataBaseKey’ is supposed to be a key string to this database.

More details about valid values for time zone identifications are given in section ‘Configuration of Time and Date - TZ String’ of the PFC200 Manual.


**Note (1):** The user is not supposed to establish his own time zones from within IEC-context. He may, however, change the database manually and provide new time zones on that way.
result codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>success</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The database key is unknown to the system</td>
</tr>
<tr>
<td>ENOSYS</td>
<td>function not implemented</td>
</tr>
<tr>
<td>EBUSY</td>
<td>The Fb is busy with another request which is still in progress</td>
</tr>
<tr>
<td>EUNSPECIFIC</td>
<td>Unexpected other errors from OS</td>
</tr>
</tbody>
</table>

3.4.3 FuGetActualTimeZone (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuGetActualTimeZone</td>
<td>WagoTypes.typTimeZoneDescription</td>
</tr>
</tbody>
</table>

Retrieves information about the currently active Timezone

Graphical Illustration

![Graphical Illustration](image)

Functional description

The returned structure typTimeZoneDescription is described in detail at the beginning of this section.

**Note:** The change of the timezone over the WBM take effect after a reboot.

3.5 04 Calendar

3.5.1 00 Formatting

01 Date (DATE)

FuDateAdjustment (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuDateAdjustment</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>date</td>
</tr>
<tr>
<td></td>
<td>iYear</td>
<td>INT</td>
<td>Year</td>
</tr>
<tr>
<td></td>
<td>iMonth</td>
<td>INT</td>
<td>Month</td>
</tr>
<tr>
<td></td>
<td>iWeek</td>
<td>INT</td>
<td>Week</td>
</tr>
<tr>
<td></td>
<td>iDay</td>
<td>INT</td>
<td>Day</td>
</tr>
</tbody>
</table>

**Function**

Adjust input date by year, month, week, day
Note: Value overflow may occur if the calculated result is outside the range of a DATE value (D#1970-1-1 .... D#2100-12-31).

Graphical Illustration

Function

FuFormatToDate

<table>
<thead>
<tr>
<th>datDate</th>
<th>DATE</th>
<th>DATE</th>
<th>FuDateAdjustment</th>
</tr>
</thead>
<tbody>
<tr>
<td>iYear</td>
<td>INT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iMonth</td>
<td>INT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iWeek</td>
<td>INT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iDay</td>
<td>INT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FuFormatToDate (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuFormat-</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ToDATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>iYear</td>
<td>INT</td>
<td>Year range 1970 to 2100</td>
</tr>
<tr>
<td></td>
<td>iMonth</td>
<td>INT</td>
<td>Month range 1 to 12 (Except in 2016 the range is 1 to 2)</td>
</tr>
<tr>
<td></td>
<td>iDay</td>
<td>INT</td>
<td>Day range 1 to 28,29,30,31 depending on the month and leap year. (Except in 2016 the range is 1 to 31 in January and 1 to 7 in February)</td>
</tr>
</tbody>
</table>

Function

Format into DATE format: D#YYYY-MM-DD

02 Time of day (TOD)

FuFormatToTOD (FUN)

Interface variables

3.5. 04 Calendar
Function

Format into TOD format: TOD#HH:MM:SS

Note: Value overflow may occur if the calculated result is outside the range of a TOD value (TOD#0:0:0.0 .... TOD#23:59:59.999).

Graphical Illustration

FuTimeAdjustment (FUN)

Interface variables

Function

Adjust input time by hour, minute, seconds

Note: Value overflow may occur if the calculated result is outside the range of a TOD value (TOD#0:0:0.0 .... TOD#23:59:59.999).

Graphical Illustration
03 Date and time (DT)

**FuDateTimeAdjustment (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuDateTimeAdjustment</td>
<td>DT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>Date</td>
</tr>
<tr>
<td></td>
<td>iYear</td>
<td>INT</td>
<td>Year</td>
</tr>
<tr>
<td></td>
<td>iMonth</td>
<td>INT</td>
<td>Month</td>
</tr>
<tr>
<td></td>
<td>iWeek</td>
<td>INT</td>
<td>Week</td>
</tr>
<tr>
<td></td>
<td>iDay</td>
<td>INT</td>
<td>Day</td>
</tr>
<tr>
<td></td>
<td>iHour</td>
<td>INT</td>
<td>hour</td>
</tr>
<tr>
<td></td>
<td>iMinute</td>
<td>INT</td>
<td>minute</td>
</tr>
<tr>
<td></td>
<td>rSecond</td>
<td>REAL</td>
<td>seconds</td>
</tr>
</tbody>
</table>

**Function**

Adjust input date by year, month, week, day, hour, minute, seconds

**Note:** Value overflow may occur if the calculated result is outside the range of a DT value (DT#1970-1-1-0:0:0 ... DT#2100-12-31-23:59:59).

**Graphical Illustration**

![Graphical Illustration of FuDateTimeAdjustment](image)

**FuFormatToDT (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuFormatToDT</td>
<td>DT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>iDay</td>
<td>INT</td>
<td>Day range 1 to 28,29,30,31 depending on the month and leap year. ( Except in 2016 the range is 1 to 31 in January and 1 to 7 in February)</td>
</tr>
<tr>
<td></td>
<td>iMonth</td>
<td>INT</td>
<td>Month range 1 to 12 ( Except in 2016 the range is 1 to 2)</td>
</tr>
<tr>
<td></td>
<td>iYear</td>
<td>INT</td>
<td>Year range 1970 to 2100</td>
</tr>
<tr>
<td></td>
<td>iHour</td>
<td>INT</td>
<td>Hour</td>
</tr>
<tr>
<td></td>
<td>iMinute</td>
<td>INT</td>
<td>Minute</td>
</tr>
<tr>
<td></td>
<td>rSecond</td>
<td>REAL</td>
<td>Seconds</td>
</tr>
</tbody>
</table>

**Function**

3.5. 04 Calendar
Format into DT format: DT#YYYY-MM-DD-HH:mm:SS

**Note:** Value overflow may occur if the calculated result is outside the range of a DT value (DT#1970-1-1-0:0:0 .... DT#2100-12-31-23:59:59).

### Graphical Illustration

![FuSetDT (FUN)](image)

**FuSetDT (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuSetDT</td>
<td>DT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>Date</td>
</tr>
<tr>
<td></td>
<td>tdTimeOfDay</td>
<td>TOD</td>
<td>Time of day</td>
</tr>
</tbody>
</table>

**Function**

Set DT by combining DATE and TOD

**Note:** Value overflow may occur if the calculated result is outside the range of a DT value (DT#1970-1-1-0:0:0 .... DT#2100-12-31-23:59:59).

### Graphical Illustration

![FuSetDT (FUN)](image)

3.5.2 01 Extraction

**01 Date (DATE)**

**FuDay (FUN)**

**Interface variables**
WagoAppTime, Release 1.7.3.3

Function

Extract Day from DATE

Graphical Illustration

```
Function

FuDay

datDate DATE INT FuDay
```

FuMonth (FUN)

Interface variables

```
<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuMonth</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>date</td>
</tr>
</tbody>
</table>
```

Function

Extract Month from DATE

Graphical Illustration

```
Function

FuMonth

datDate DATE INT FuMonth
```

FuYear (FUN)

Interface variables

```
<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuYear</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>date</td>
</tr>
</tbody>
</table>
```

Function

Extract Year from DATE

Graphical Illustration

```
Function

FuYear

datDate DATE INT FuYear
```
02 Time of day (TOD)

FuHour (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuHour</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>tdTimeOfDay</td>
<td>TOD</td>
<td>Time of day</td>
</tr>
</tbody>
</table>

Function

Extract Hour from TOD

Graphical Illustration

![FuHour Graphical Illustration]

FuMinute (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuMinute</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>tdTimeOfDay</td>
<td>TOD</td>
<td>Time of day</td>
</tr>
</tbody>
</table>

Function

Extract Minute from TOD

Graphical Illustration

![FuMinute Graphical Illustration]

FuSecond (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuSecond</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>tdTimeOfDay</td>
<td>TOD</td>
<td>Time of day</td>
</tr>
</tbody>
</table>

Function

Extract Second from TOD

Graphical Illustration

![FuSecond Graphical Illustration]
3.5.3 02 Conversions

01 Date and Time

FuConvertStampToLocalTimeComponents (FUN)

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuConvertStampToLocalTimeComponents</td>
<td>Wago-Types.typWagoTimeComponents</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>dtStamp</td>
<td>DT</td>
<td>Seconds since 1.1.1970</td>
</tr>
</tbody>
</table>

Splits a timestamp into its components regarding a given timezone information

**Graphical Illustration**

**Function Description**

Note: for timestamps in the past, only the actual timezone is considered, while for timestamps in the future all scheduled zone changes (if available) are taken into account.

FuConvertStampToTimeComponents (FUN)

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuConvertStampToTimeComponents</td>
<td>Wago-Types.typWagoTimeComponents</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>dtStamp</td>
<td>DT</td>
<td>Seconds since 1.1.1970</td>
</tr>
</tbody>
</table>

Splits a timestamp into its components without any timezone consideration

**Graphical Illustration**

**Function Description**
This is useful for:

1. converting time intervals into components (nanoseconds ... hours, days).
2. as auxiliary routine for other conversions

**FuConvertTimeComponentsToStamp (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuConvertTimeComponentsToStamp</td>
<td>WagoTypes.typWagoTimeComponents</td>
<td>Localized time components</td>
</tr>
</tbody>
</table>

Converts Time Components into a POSIX time Stamp

**Graphical Illustration**

**Function**

```
FuConvertTimeComponentsToStamp
```

**typComponents** typWagoTimeComponents DT FuConvertTimeComponentsToStamp

**Function Description**

From the input only the following fields are taken into account: *uiYear*, *uiMonth*, *uiDay*, *wHour*, *wMinute*, *wSecond*, *wMilliseconds*, *TimeZone.iOffset*, *udiMicroseconds*, and *udiNanoseconds*. The other fields are redundant and will be ignored.

Note: The actual timezone will not be taken into account, as the typWagoTimeComponent-Structure already contains time zone information (which will be used). The resulting absolute timestamp is, however, always related to UT.

**LTime**

**FuConvertLStampToLocalTimeComponents (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuConvertLStampToLocalTimeComponents</td>
<td>WagoTypes.typWagoTimeComponents</td>
<td>Nanoseconds since 1.1.1970</td>
</tr>
<tr>
<td>Input</td>
<td>ltStamp</td>
<td>LTIME</td>
<td></td>
</tr>
</tbody>
</table>

Splits a timestamp into its components regarding the actually set timezone und the timezoneschedule

**Graphical Illustration**

**Function**

```
FuConvertLStampToLocalTimeComponents
```

```
ltStamp LTIME typWagoTimeComponents FuConvertLStampToLocalTimeComponents
```

Note: for timestamps in the past, only the actual timezone is considered, while for timestamps in the future all scheduled zone changes (if available) are taken into account.
### FuConvertLStampToTimeComponents (FUN)

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuConvertLStampToTimeComponents</td>
<td>Wago-Types.typWagoTimeComponents</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>ltStamp</td>
<td>LTIME</td>
<td>Nanoseconds since 1.1.1970</td>
</tr>
</tbody>
</table>

Splits a timestamp into its components without any timezone consideration

**Graphical Illustration**

![Function Diagram](image)

**Function Description**

This is useful for

1. converting time intervals into nanoseconds ... hours, days.
2. auxiliary routine for other conversions

### FuConvertTimeComponentsToLStamp (FUN)

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuConvertTimeComponentsToLStamp</td>
<td>LTIME</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>typComponents</td>
<td>Wago-Types.typWagoTimeComponents</td>
<td>Localized time components</td>
</tr>
</tbody>
</table>

Converts Time Components into a long time stamp

**Graphical Illustration**

![Function Diagram](image)

**Function Description**

From the input only the following fields are taken into account: uiYear, uiMonth, uiDay, uiHour, uiMinute, uiSecond, uiMilliseconds, TimeZone.iOffset, udiMicroseconds, and udiNanoseconds. The other fields are redundant and will be ignored.

Note: The actual timezone will not be taken into account, as the typWagoTimeComponent-Structure already contains time zone information (which will be used).

The result is a UT related absolute 64-Bit timestamp.
**FuCheckTimeComponentPosixConformity (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuCheckTimeComponentPosix-Conformity</td>
<td>WagoTypes.eResultCode</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>typComponents</td>
<td>WagoTypes.tplWagoTimeComponents</td>
<td>Localized time components</td>
</tr>
</tbody>
</table>

Checks a time component structure for POSIX conformity.

**Graphical Illustration**

Dates behind 2038 and before 1970 will be rejected.

A few dates near the last day of the posix range will also be rejected, although they might be valid. This is done in order to avoid problems with local timezones near the limits of the posix range.

Note: Due to problems with the system date, all dates with year 2038 and beyond are regarded as invalid. More precisely: 17.1.2038 is the last valid date (even if POSIX goes some hours further) because the POSIX range could be exceeded due to Timezone-effects.

**result codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>success</td>
</tr>
<tr>
<td>EINV</td>
<td>Components denote invalid time stamps (e.g. Minute =60 or Day=0)</td>
</tr>
<tr>
<td>ERANGE</td>
<td>The components are outside the range of POSIX times.</td>
</tr>
</tbody>
</table>

**FuCheckTimeComponentValidity (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuCheckTimeComponentValidity</td>
<td>WagoTypes.eResultCode</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>typComponents</td>
<td>WagoTypes.tplWagoTimeComponents</td>
<td>Localized time components</td>
</tr>
</tbody>
</table>

Checks if a time component structure contains possibly invalid components.

**Graphical Illustration**

Note: this is not a full check, i.e. the date feb-30 will pass through as OK. But mar-32 will be rejected.

**result codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>success</td>
</tr>
<tr>
<td>EINV</td>
<td>Components denote invalid time stamps (e.g. Minute =60 or Day=0)</td>
</tr>
</tbody>
</table>
**FuRTS_SYSTIMEDATE_to_TimeComponents (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuRTS_SYSTIMEDATE_to_TimeComponents</td>
<td>Wago-Types.typWagoTimeComponents</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>typBase</td>
<td>SysTimeRtc.SYSTIMEDATE</td>
<td>base class</td>
</tr>
</tbody>
</table>

Copies the structure RTS_SYSTIMEDATE into the derived typWagoTimeComponents.

**Graphical Illustration**

Extended components of the result (i.e. those which are not contained in RTS_SYSTIMEDATE) are left empty.

**FuTimeComponents_to_RTS_SYSTIMEDATE (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuTimeComponents_to_RTS_SYSTIMEDATE</td>
<td>SYSTIMEDATE</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>typBase</td>
<td>Wago-Types.typWagoTimeComponents</td>
<td>the more complete class</td>
</tr>
</tbody>
</table>

Fills the components of RTS_SYSTIMEDATE with the base components of the derived typWagoTimeComponents.

**Graphical Illustration**

**note:** Regularly you might think that this function is obsolete, because you would simply copy the extended typWagoTimeComponents into the base class RTS_SYSTIMEDATE. But the real world is not so nice and at least for past compiler versions a dedicated conversion function is necessary.

### 3.5.4 03 Weekday

**FuAnyDayOfWeek (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuAnyDayOfWeek</td>
<td>DATE</td>
<td>input date</td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iDayOfWeek</td>
<td>INT</td>
<td>day of week. 1:Mon, 2:Tue, 3:Wed, 4:Thu, 5:Fri, 6:Sat, 7:Son</td>
</tr>
</tbody>
</table>

**Function**
Retrieves any weekday from input date.

**Graphical Illustration**

```
Function
FuAnyDayOfWeek

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>datDate</td>
<td>DATE</td>
<td>DATE</td>
<td>FuAnyDayOfWeek</td>
</tr>
<tr>
<td>iDayOfWeek</td>
<td>INT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Function Description**

Positive ‘iDayOfWeek’ retrieves future dates including the same date, negative ‘iDayOfWeek’ retrieves past dates.

Examples:
- datNextMonday:=FuAnyDayOfWeek(D#2015-01-08,1); // +1 retrieves the next Monday from the input date i.e. D#2015-01-12
- datMonday:=FuAnyDayOfWeek(D#2015-01-08,-1); // -1 retrieves the previous Monday from the input date i.e. D#2015-01-05
- datThursday:=FuAnyDayOfWeek(D#2015-01-08,5); // +5 retrieves the current Thursday i.e. D#2015-01-08
- datPrevThursday:=FuAnyDayOfWeek(D#2015-01-08,-1); // -5 retrieves the previous Thursday i.e. D#2015-01-01

**FuNextWeekday (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuNextWeekday</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>input date</td>
</tr>
</tbody>
</table>

**Function**

Retrieves next weekday

**Graphical Illustration**

```
Function
FuNextWeekday

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>datDate</td>
<td>DATE</td>
<td>DATE</td>
<td>FuNextWeekday</td>
</tr>
</tbody>
</table>
```

**FuPreviousWeekday (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuPreviousWeekday</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>input date</td>
</tr>
</tbody>
</table>

**Function**

Retrieves previous weekday

**Graphical Illustration**
FuWeekDay_ISO8601 (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuWeekDay_ISO8601</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>date</td>
</tr>
</tbody>
</table>

Function

Retrieves current weekday according to ISO 8601

Graphical Illustration

Function Description

Return value:
1. Monday
2. Tuesday
3. Wednesday
4. Thursday
5. Friday
6. Saturday
7. Sunday

FuWeekDayOfMonth (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuWeekDayOfMonth</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>input date</td>
</tr>
<tr>
<td></td>
<td>iDayOfWeek</td>
<td>INT</td>
<td>day of week. 1:Mon, 2:Tue, 3:Wed, 4:Thu, 5:Fri, 6:Sat, 7:Son</td>
</tr>
<tr>
<td></td>
<td>bNumberOfWeek</td>
<td>BYTE</td>
<td>number of week in month. Range 1..4</td>
</tr>
</tbody>
</table>

Function

Retrieves the weekday for a defined week number of month from input date.

Graphical Illustration
Function Description

The function calculates always to the future from input date. If the found date is former the ‘datDate’, the function will response the date for the next month.

Examples:

datWeekdayOfMonth:=FuWeekdayOfMonth(D#2017-09-02,1,1); // retrieves the D#2017-09-04

datWeekdayOfMonth:=FuWeekdayOfMonth(D#2017-09-02,1,2); // retrieves the D#2017-09-11

datWeekdayOfMonth:=FuWeekdayOfMonth(D#2017-09-20,1,1); // retrieves the D#2017-10-02

FuWeekdayOfYear (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuWeekday-Of-Year</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>datDate</td>
<td>DATE</td>
<td>input date</td>
</tr>
<tr>
<td></td>
<td>iDayOfWeek</td>
<td>INT</td>
<td>day of week. 1:Mon, 2:Tue, 3:Wed, 4:Thu, 5:Fri, 6:Sat, 7:Son</td>
</tr>
<tr>
<td></td>
<td>bNumberOfWeek</td>
<td>BYTE</td>
<td>number of week in month. Range 1..4</td>
</tr>
<tr>
<td></td>
<td>iMonth</td>
<td>INT</td>
<td>month repeated from.</td>
</tr>
<tr>
<td></td>
<td>bMonthRepeat</td>
<td>BYTE</td>
<td>number of month to repeat. allowed values: 1(monthly), 2, 3, 4, 6, 12(yearly)</td>
</tr>
</tbody>
</table>

Function

Retrieves a date for recurring events based on weekday, month and monthly period a defined week number of month from input date.

Graphical Illustration

Function Description

The function calculates always to the future from input date.

Examples:

datWeekdayOfYear:=FuWeekdayOfYear(D#2017-08-10,1,2,4,6); // retrieves the second monday every half year beginning in march = D#2017-10-09
datWeekdayOfYear:=FuWeekdayOfYear(D#2017-08-10,5,2,1,3); // retrieves the second friday every quarter beginning in january = D#2017-10-13

### 3.5.5 04 Leap year

**FuIs29Feb (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuIs29Feb</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>date</td>
</tr>
</tbody>
</table>

**Function**

Check for 29th February

**Graphical Illustration**

![Function Diagram](image)

**FuIsLeapYear (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuIsLeapYear</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>iYear</td>
<td>INT</td>
<td>input year</td>
</tr>
</tbody>
</table>

**Function**

Check if any given year is in leap year

**Graphical Illustration**

![Function Diagram](image)

### 3.5.6 05 Calendar Week

**FuCalendarWeek_ISO8601 (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuCalendarWeek_ISO8601</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>date</td>
</tr>
</tbody>
</table>

**Function**
Calculates the calendar week of given date according to ISO 8601

**Graphical Illustration**

3.5.7 06 Number of days

FuDaysSinceJan01 (FUN)

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuDaysSinceJan01</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>date</td>
</tr>
</tbody>
</table>

**Function**

Calculates number of days since 1st. January till the current actual date of the same year

**Graphical Illustration**

FuNumDaysYear (FUN)

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuNumDaysYear</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>date</td>
</tr>
</tbody>
</table>

**Function**

Calculates number of days of current year [365 or 366]

**Graphical Illustration**
### 3.5.8 07 Month

**FuCurrMonthWeek (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuCurrMonthWeek</td>
<td>DINT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>iDay</td>
<td>INT</td>
<td>day of Date</td>
</tr>
</tbody>
</table>

**Function**

This function returns current week of the month on input date. NOT Calendar week!

**Graphical Illustration**

![Function Diagram](image1.png)

**FuDaysInMonth (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuDaysInMonth</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>iMonth</td>
<td>INT</td>
<td>month value</td>
</tr>
<tr>
<td></td>
<td>iYear</td>
<td>INT</td>
<td>year value</td>
</tr>
</tbody>
</table>

**Function**

Calculate number of days of the input month and year

**Graphical Illustration**

![Function Diagram](image2.png)

**FuMonthBegin (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuMonthBegin</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>date</td>
</tr>
</tbody>
</table>

**Function**

Calculate the date for the first day of the current month.
Note: Value overflow may occur if the calculated result is outside the range of a DATE value (D#1970-1-1 .... D#2106-2-7).

Graphical Illustration

FuMonthEnd (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuMonthEnd</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>date</td>
</tr>
</tbody>
</table>

Function
Calculate the date for the last day of the current month.

Note: Value overflow may occur if the calculated result is outside the range of a DATE value (D#1970-1-1 .... D#2106-2-7).

Graphical Illustration

3.5.9 08 Year

FuYearBegin (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuYearBegin</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>date</td>
</tr>
</tbody>
</table>

Function
Get the Date for 1st January of any given date.

Note: Value overflow may occur if the calculated result is outside the range of a DATE value (D#1970-1-1 .... D#2106-2-7).

Graphical Illustration

3.5. 04 Calendar
### FuYearEnd (FUN)

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuYearEnd</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>date</td>
</tr>
</tbody>
</table>

**Function**

Get the Date for 31st December of any given date. Range: 1970 - 2099

**Note:** Value overflow may occur if the calculated result is outside the range of a DATE value (D#1970-1-1 .... D#2106-2-7).

### 3.5.10 09 Active duration

**FuisActiveDate (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuisActiveDate</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datActualDate</td>
<td>DATE</td>
<td>actual date and time</td>
</tr>
<tr>
<td></td>
<td>datStartDate</td>
<td>DATE</td>
<td>Start date and time</td>
</tr>
<tr>
<td></td>
<td>datEndDate</td>
<td>DATE</td>
<td>End date and time</td>
</tr>
</tbody>
</table>

**Function**

Function check for active date

**Graphical Illustration**

---

3.5. 04 Calendar
**FuIsActiveDateTime (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuIsActiveDateTime</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>dtActualDateTime</td>
<td>DT</td>
<td>Date and time for start</td>
</tr>
<tr>
<td>Input</td>
<td>dtOnDateTime</td>
<td>DT</td>
<td>Date and time for start</td>
</tr>
<tr>
<td>Input</td>
<td>dtOffDateTime</td>
<td>DT</td>
<td>Date and time for end</td>
</tr>
</tbody>
</table>

**Function**

The function checks active date and time.

**Graphical Illustration**

![Graphical Illustration for FuIsActiveDateTime](image)

**FuIsActiveTime (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuIsActiveTime</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>tdActualTime</td>
<td>TOD</td>
<td>date</td>
</tr>
<tr>
<td>Input</td>
<td>tdOnTime</td>
<td>TOD</td>
<td>on time</td>
</tr>
<tr>
<td>Input</td>
<td>tdOffTime</td>
<td>TOD</td>
<td>off time</td>
</tr>
</tbody>
</table>

**Function**

The function checks active duration including overnight.

**Graphical Illustration**

![Graphical Illustration for FuIsActiveTime](image)

**FuIsToday (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuIsToday</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>datActualDate</td>
<td>DATE</td>
<td>Actual date and time</td>
</tr>
<tr>
<td>Input</td>
<td>datDate</td>
<td>DATE</td>
<td>Date</td>
</tr>
</tbody>
</table>
Function

Function check for today

Graphical Illustration

3.5.11 FuGetLocalTimeComponents (FUN)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuGetLocalTimeComponents</td>
<td>WagoTypes.typWagoTimeComponents</td>
</tr>
</tbody>
</table>

Retrieves the actual local time and splits it into its components.

Graphical Illustration

Functional description

The returned structure contains the localized time components together with information about the actual time zone.

Note: this is a convenience function for the sequence

1. retrieve actual time
2. retrieve actual timezone
3. split stamp into components
4. consider recent changes of time zone

3.6 05 Intervals

Function blocks for measuring elapsed time and detecting time-outs

3.6.1 FbElapsedTime (FB)

Measures the elapsed time

Graphical Illustration
Function Description

This FB measures the elapsed time from a an indicated start. The elapsed time my be read as LTIME or as TIME by getter-methods.

Typical usage

```plaintext
VAR
  FbStamp : FbElapsedTime;
END_VAR

VAR CONSTANT
  tFirstStep : TIME := T#2s;
  tSecondStep : TIME := T#10s;
END_VAR

FbStamp.StartNow();
:
:
IF (FbStamp.GetElapsedTime() < tFirstStep) THEN
  OneCertainAction();
ELSE (FbStamp.GetElapsedTime() < tSecondStep) THEN
  OtherCertainAction();
END_IF;
```

10 Main Interface

FbElapsedTime.GetElapsedLTime (METH)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>GetElapsedLTime</td>
<td>LTIME</td>
</tr>
</tbody>
</table>

Returns the nanoseconds since StartNow()

Graphical Illustration

```
Method
FbElapsedTime.GetElapsedTime

LTIE     GetElapsedLTime
```

Function Description

When more than ~292 years have elapsed, the result would overflow, thus it is clamped to 16#7FFF.FFFF.FFFF.FFFF to indicate the overflow (this is performed for formal reasons rather than for practical ones).

FbElapsedTime.GetElapsedTime (METH)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>GetElapsedTime</td>
<td>TIME</td>
</tr>
</tbody>
</table>

Returns the milliseconds since start of the measurement
Graphical Illustration

Function Description
When more than ~49 days have elapsed, the result would overflow, thus it is clamped to 16#FFFF.FFFF to indicate the overflow.

**FbElapsedTime.StartNow (METH)**

Sets the start time of the measurement to the actual time

Graphical Illustration

**FbElapsedTime.initialize (METH)**

Initializes FbElapsedTime

Graphical Illustration

Function Description
This has to be called in derived function blocks, in order to initialize the parent FB properly. When FbElapsedTime is used directly, it may be ignored.

When this FB is used as parent for inheritance, the child should call SUPER^.initialize() during its own initialization.

### 3.6.2 FbTimeoutWatcher (FB)

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Initial</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>pTime-outAction</td>
<td>POINTER TO WagoTypes.Fb_GenericRunner</td>
<td>0</td>
<td>Address of a runner instance to be called at timeout.</td>
</tr>
</tbody>
</table>

Timeout detection

Graphical Illustration
Function Description

After a given amount of time this FB indicates a timeout.

A timeout condition is normally indicated by the return value of the method `HasTimeoutOccurred()`. Additionally, at timeout a dedicated timeout action may be invoked (see below).

The reference point in time for determining timeouts is given by the system start or by calling one of the methods `StartNow()` or `Retrigger()`.

Timeout will not occur, before the time limit value is set by one of the methods `SetTimeout()` or `SetLTimeout()`.

When using `StartNow()` for starting, it is mandatory to call `Set(L)Timeout` afterwards for detecting a timeout. By using `Retrigger()` however, the previously set timeout limit will be re-used again.

(To be clear in this point: `SetTimeout()` will not start the measurement of the timeout-interval. Just the notification about the expiration of the timeout requires that this timeout value has been set. This value may also be changed during the runtime without disturbing the interval measurement.)

**Note(1):** Since we have no output variables in this FB, it is normally not necessary to call it cyclically. When a timeout action is applied, however, cyclic invocation is mandatory because otherwise the timeout action could not be called from within the FB.

**Note(2):** By using getter and setter methods for timeouts and for measured times, different time formats (TIME, LTIME) may be mixed and used simultaneously.

Typical usage

```vbnet
VAR
    FbWatcher : FbTimeoutWatcher;
END_VAR

FbWatcher.StartNow();
FbWatcher.SetTimeout(T#3s);

IF FbWatcher.IsTimedOut() THEN
    OneCertainTimeoutAction();
END_IF;
```

Additional Feature:

An object derived from ‘WagoTypes.Fb_GenericRunner’ may be scheduled to be called when timeout has occurred. For doing so, the input variable ‘pTimeoutAction’ has to be set to the address of the instance of the runner object. In that case, the FbTimeoutWatcher must be called cyclically, while in other cases, the cyclic call may be omitted. On timeout, the method `run()` of that object will be called.

If that feature is not used, leave the input at its default (=0). In that case, just the HasTimeoutOccurred-Property is set and no further action is taken. Then, cyclic call of the FB may be omitted.

01 Main Interface

FbTimeoutWatcher.GetRemainingMilliseconds (METH)

Interface variables
Return.GetRemainingMilliseconds.DINT

Returns the milliseconds to expected timeout

Function Description
Note: Zero or negative return values indicate that the timeout has occurred.
If the value would exceed the range of 32-Bits, the return value is clamped to +16#7FFFF.FFFF in order to avoid strange problems with wrap-arounds.

FbTimeoutWatcher.GetRemainingMilliseconds (METH)

Interface variables

Return.GetRemainingMilliseconds.DINT

Returns the milliseconds remaining until timeout would occur

Function Description
Note: Zero or negative return values indicate that the timeout has occurred.
If the value would exceed the range of 64-Bits, the return value is clamped to +16#7FFFF.FFFF.FFFFF.FFFF in order to avoid strange problems with wrap-arounds (regarding the usable range of 292 years, this detail has more formal use rather than practical use)

FbTimeoutWatcher.HasTimeoutOccurred (METH)

Interface variables

Return.HasTimeoutOccurred.BOOL

Indicates that the timeout has occurred inbetween

Graphical Illustration
**Function Description**
Return value: TRUE when Timeout is reached or elapsed

**FbTimeoutWatcher.Retrigger (METH)**
Restarts the timeout watcher

**Graphical Illustration**

**Function Description**
When using the FB as a ‘watchdog’, this is the recommended method for ‘calming down’ the watchdog.
When retriggering, the previously set up timeout limit will be used again. No subsequent call of SetTimeout() is necessary.

**FbTimeoutWatcher.SetLTimeout (METH)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>lTimeOut</td>
<td>LTIME</td>
<td>Nanoseconds after start of measurement</td>
</tr>
</tbody>
</table>

Tells the FB when the timeout should occur (high resolution)

**Graphical Illustration**

**Function Description**
Timeout will occur after a time interval of ‘tTimeOut’ has elapsed after start of the FB or retriggering.
Note: It is possible to change the timeout limit while the FB is running.

**FbTimeoutWatcher.SetTimeout (METH)**

**Interface variables**
**Input**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>tTimeOut</td>
<td>TIME</td>
<td>Milliseconds after start of measurement</td>
</tr>
</tbody>
</table>

Tells the FB when the timeout should occur (milliseconds)

**Graphical Illustration**

![Method](fbtimeoutwatcher_settimeout.png)

**Function Description**

Timeout will occur after a time interval of `tTimeOut` has elapsed after start of the FB or retriggering.

Note: It is possible to change the timeout limit while the FB is running.

**02 Administrative**

**FbTimeoutWatcher.initialize (METH)**

Initializes the FB.

**Graphical Illustration**

![Method](fbtimeoutwatcher_initialize.png)

**Function Description**

This has to be called in derived function blocks, in order to initialize the parent FB properly. When FbElapsedTime is used directly, it may be ignored.

When this FB is used as parent for inheritance, the child should call SUPER^.initialize() during its own initialization.

**3.7 06 Setting The Clock**

**3.7.1 FuSetClockFromDateAndTime (FUN)**

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuSetClockFromDateAndTime</td>
<td>Wago-Types.eResultCode</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>dtDateTime</td>
<td>DT</td>
<td>POSIX time (seconds since 1.1.1970)</td>
</tr>
</tbody>
</table>

Sets the system clock from POSIX time (seconds since 1.1.1970, UT).

**Graphical Illustration**

![Method](fu_set_clock_from_date_and_time.png)
Functional description

**Note:** Although the input variable allows for a wider range, only dates up to the year 2038 are valid for setting the clock.

<table>
<thead>
<tr>
<th>result codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>ERANGE</td>
<td>Time value is beyond a reasonable range (1970 up to 2038)</td>
</tr>
<tr>
<td>ENOSYS</td>
<td>Function not implemented</td>
</tr>
<tr>
<td>EACCES</td>
<td>Time could not be set due to internal PLC problems.</td>
</tr>
</tbody>
</table>

### 3.7.2 FuSetClockFromLTime (FUN)

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuSetClockFromLTime</td>
<td>WagoTypes.eResultCode</td>
</tr>
<tr>
<td>Input</td>
<td>ltDateTime</td>
<td>LTIME</td>
</tr>
</tbody>
</table>

sets the system clock (Nanoseconds since 1.1.1970, UT).

**Graphical Illustration**

Functional description

**Note:** Although the input variable allows for a wider range, only dates up to the year 2038 are valid for setting the clock.

<table>
<thead>
<tr>
<th>result codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>ERANGE</td>
<td>Time value is beyond a reasonable range (1970 up to 2038)</td>
</tr>
<tr>
<td>ENOSYS</td>
<td>Function not implemented</td>
</tr>
<tr>
<td>EACCES</td>
<td>Time could not be set due to internal PLC problems.</td>
</tr>
</tbody>
</table>

### 3.7.3 FuSetClockFromLocalDateAndTime (FUN)

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuSetClockFromLocalDateAndTime</td>
<td>WagoTypes.eResultCode</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>dtDateTime</td>
<td>DT</td>
<td>Local time (seconds since 1.1.1970)</td>
</tr>
</tbody>
</table>

Sets the system clock from Local time (seconds since 1.1.1970, UT).
Graphical Illustration

**Function**

![Function diagram](image)

**Note:** Although the input variable allows for a wider range, only dates up to the year 2038 are valid for setting the clock.

<table>
<thead>
<tr>
<th>result codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>ENOSYS</td>
<td>Function not implemented</td>
</tr>
<tr>
<td>ERANGE</td>
<td>Time value is beyond a reasonable range (1970 up to 2099)</td>
</tr>
<tr>
<td>EACCES</td>
<td>Time could not be set due to internal PLC problems.</td>
</tr>
</tbody>
</table>

### 3.7.4 FuSetClockFromTimeComponents (FUN)

**Interface variables**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>FuSetClockFromTimeComponents</td>
<td>WagoTypes.eResultCode</td>
</tr>
<tr>
<td>Input</td>
<td>typComponents</td>
<td>WagoTypes.typWagoTimeComponents</td>
</tr>
</tbody>
</table>

Sets the clock according to the entered time components.

**Graphical Illustration**

![Function diagram](image)

**Functional description**

**Note:** Although the input variable allows for a wider range, only dates up to the year 2038 are valid for setting the clock.

From the input only the following fields are taken into account: `uiYear`, `uiMonth`, `uiDay`, `wHour`, `wMinute`, `wSecond`, `wMilliseconds`, `TimeZone.iOffset`, `udiMicroseconds`, and `udiNanoseconds`. The other fields are redundant and will be ignored.

<table>
<thead>
<tr>
<th>result codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>ERANGE</td>
<td>Time value is beyond a reasonable range (1970 up to 2038)</td>
</tr>
<tr>
<td>ENOSYS</td>
<td>Function not implemented</td>
</tr>
<tr>
<td>EACCES</td>
<td>Time could not be set due to internal PLC problems.</td>
</tr>
</tbody>
</table>

Note: The actual local timezone is disregarded because the input data already contains timezone information.
LibraryResult (GVL)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factory</td>
<td>WagoSysError-Base.FbResultFactory</td>
<td>Produces FbResults from given eResultCodes.</td>
</tr>
</tbody>
</table>

Factory for standard result objects

Use this to translate result codes from this library into standard result objects.

Usage:

```pascal
VAR
  eMyResult : eResultCode;  // result code which is to be investigated
  oError    : FbResult;     // result object for use in higher levels
END_VAR;

eMyResult := myFunction(...);
Namespace.LibraryResult.Factory.SetResult(eMyResult, oError);
```

(In this example ‘Namespace’ denotes the namespace which is used for including the specific library and ‘myFunction()’ is an example for a general function from this library.)
### ResultItems (GVL)

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>[STRUCT(ID := WagoTypes.eResultCode.ENOENT, Severity := WagoTypes.eSeverity.error, Text := 'The database key is unknown to the system.')]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[STRUCT(ID := WagoTypes.eResultCode.EUNSPECIFIC, Severity := WagoTypes.eSeverity.error, Text := 'Unspecified other errors from operating system.')]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[STRUCT(ID := WagoTypes.eResultCode.EINV AL, Severity := WagoTypes.eSeverity.error, Text := 'Components denote invalid time stamps (e.g. Minute =60 or Day=0).')]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[STRUCT(ID := WagoTypes.eResultCode.ERANGE, Severity := WagoTypes.eSeverity.error, Text := 'The value or one of its components is beyond its expected range.')]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[STRUCT(ID := WagoTypes.eResultCode.ENOSYS, Severity := WagoTypes.eSeverity.error, Text := 'Function not implemented.')]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[STRUCT(ID := WagoTypes.eResultCode.EACCES, Severity := WagoTypes.eSeverity.error, Text := 'Time could not be set due to PLC-internal reasons.')]</td>
</tr>
</tbody>
</table>

Standard result items specific for this library

Note: This is a general mapping of result codes to short standard texts which are appropriate to the usage of these codes in this library.

Typically, each unit (function, method, or function block) in this library uses only a subset of these codes. Please, refer to the documentation of the specific unit for the set of codes which is actually used and for a detailed explanation of the meaning of a result code in the specific context.
### VersionHistory (GVL)

<table>
<thead>
<tr>
<th>date</th>
<th>version</th>
<th>author</th>
<th>change</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.06.2019</td>
<td>1.7.3.3</td>
<td>WAGO / u013972</td>
<td>Add a link to a list of valid time zones</td>
</tr>
<tr>
<td>13.06.2019</td>
<td>1.7.3.2</td>
<td>WAGO / u013972</td>
<td>Bugfix in FuTimeAdjustment</td>
</tr>
<tr>
<td>28.05.2019</td>
<td>1.7.3.1</td>
<td>WAGO / u013972</td>
<td>WAT29861 - documentation adapted</td>
</tr>
<tr>
<td>08.01.2019</td>
<td>1.7.3.0</td>
<td>u015842</td>
<td>Properties: free placeholder added</td>
</tr>
<tr>
<td>21.02.2018</td>
<td>1.7.2.5</td>
<td>WAGO / u013972</td>
<td>Bugfix in FuNextWeekday</td>
</tr>
<tr>
<td>04.09.2017</td>
<td>1.7.2.4</td>
<td>WAGO / u015842</td>
<td>Redesigned algorithm of FuAnyDayOfWeek, FuWeekdayOfMonth and FuWeekdayOfYear added</td>
</tr>
<tr>
<td>16.06.2016</td>
<td>1.7.2.3</td>
<td>WAGO / u013972</td>
<td>Change resolution and documentation of FuGetLongTime</td>
</tr>
<tr>
<td>03.03.2016</td>
<td>1.7.0.0</td>
<td>WAGO / u013972</td>
<td>Release WagoAppErrorBase with WagoSysErrorBase</td>
</tr>
<tr>
<td>03.03.2016</td>
<td>1.6.3.1</td>
<td>WAGO / u014521</td>
<td>Add the general functions for time calculation, Link always activated</td>
</tr>
<tr>
<td>22.01.2016</td>
<td>1.6.3.0</td>
<td>WAGO / u013972</td>
<td>Add the functions FuGetLocalDateAndTime() and FuSetClockFromLocalDateAndTime()</td>
</tr>
<tr>
<td>29.09.2015</td>
<td>1.5.2.0</td>
<td>WAGO / u013972</td>
<td>Resolve libraries with placeholders</td>
</tr>
<tr>
<td>23.09.2015</td>
<td>1.5.1.0</td>
<td>WAGO / u013972</td>
<td>Workaround for C0351-Bug</td>
</tr>
<tr>
<td>23.06.2015</td>
<td>1.5.0.0</td>
<td>WAGO / u013972</td>
<td>Release version</td>
</tr>
</tbody>
</table>
This is a dictionary of all referenced libraries and their name spaces.

**Standard**

*Library Identification:*

Placeholder: Standard

Default Resolution: Standard, * (System)

Namespace: Standard

*Library Properties:*

- LinkAllContent: False
- QualifiedOnly: False
- SystemLibrary: False
- Optional: False

**SysTime**

*Library Identification:*

Placeholder: SysTime

Default Resolution: SysTime, * (System)

Namespace: SysTime

*Library Properties:*

- LinkAllContent: False
- QualifiedOnly: False
- SystemLibrary: False
- Optional: False

**WagoSysBehaviourModels**

*Library Identification:*

Placeholder: WagoSysBehaviourModels

Default Resolution: WagoSysBehaviourModels, * (WAGO)

Namespace: WagoSysBehaviourModels

*Library Properties:*
WagoAppTime, Release 1.7.3.3

- LinkAllContent: False
- QualifiedOnly: False
- SystemLibrary: False
- Optional: False

WagoSysErrorBase

Library Identification:
Placeholder: WagoSysErrorBase
Default Resolution: WagoSysErrorBase, * (WAGO)
Namespace: WagoSysErrorBase

Library Properties:
- LinkAllContent: False
- QualifiedOnly: False
- SystemLibrary: False
- PublishSymbolsInContainer: True
- Optional: False

Library Parameter:
Parameter: RES_LOG_MAX_FILESIZE = 2000
Parameter: RES_LOG_MAX_FILES = 1
Parameter: RES_LOG_MAX_ENTRIES = 200
Parameter: RES_LOG_NAME = 'WagoAppResultLogger'

WagoSysTime_Internal_PFC

Library Identification:
Placeholder: WagoSysTimeInternal
Default Resolution: WagoSysTime_Internal_PFC, * (WAGO)
Namespace: WagoSysTimeInternal

Library Properties:
- LinkAllContent: False
- Optional: False
- QualifiedOnly: False
- SystemLibrary: False
- PublishSymbolsInContainer: True

WagoSysTypedefs_Internal_32Bit

Library Identification:
Placeholder: WagoSysTypedefsInternal
Default Resolution: WagoSysTypedefs_Internal_32Bit, * (WAGO)
Namespace: WagoTypesInternal

Library Properties:
• LinkAllContent: False
• QualifiedOnly: False
• SystemLibrary: False
• Optional: False

**WagoSysVersion**

*Library Identification:*
Name: WagoSysVersion
Version: 1.0.0.0
Company: WAGO
Namespace: WagoSysVersion

*Library Properties:*
• LinkAllContent: False
• QualifiedOnly: False
• SystemLibrary: False
• Optional: False

**WagoTypesCommon**

*Library Identification:*
Placeholder: WagoTypesCommon
Default Resolution: WagoTypesCommon, * (WAGO)
Namespace: WagoTypes

*Library Properties:*
• LinkAllContent: False
• QualifiedOnly: False
• SystemLibrary: False
• Optional: False

© WAGO Kontaktechnik GmbH & Co. KG, Germany 2018 – All rights reserved. For the avoidance of doubt, this copyright notice does not only apply to the information above but also and primarily to the described library itself. Please note that third-party products are always mentioned without reference to intellectual property rights, including patents, utility models, designs and trademarks, accordingly the existence of such rights cannot be excluded. WAGO is a registered trademark of WAGO Verwaltungsgesellschaft mbH.