Document Number:N/A
Rev 1, 12/2013

# RTC_LDD
## Embedded Component User Guide

**Contents**



## 1 General Information

**Component Level: Logical Device Driver**

**Category: Logical Device Drivers-Timer**

This components provides common time and date operation and alarm functionality.

No SW counters are used; all information is read/computed from HW counters in the runtime. This reduces the number of ISR calls.

For battery backed up devices the Init() method has special mode where the registers not backed up by battery are initialized only.

**Major supported features by the RTC component:**

- Provides a time of day and a calendar function. (SW emulated if the HW doesn't provide full calendar functionality)

- Provides alarm function. (If supported by the HW)

- Provides stop watch function. (If supported by the HW)

**Hardware Architecture Assumptions**

This design requires at least a HW second counter. All calendar and time functions are emulated by the SW if not supported directly by the HW.

**Properties**

The SW algorithm implements the Gregorian calendar. (Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100; the centurial years that are exactly divisible by 400 are still leap years).

HW is required to use the same calendar algorithm.

**Limitations**

Minimal year value is 2000 and maximal year value is 2099 to reduce the complexity of the calendar functions.

# 2 Properties

This section describes component's properties. Properties are parameters of the component that influence the generated code. Please see the Processor Expert user manual for more details.

- **Component name** - Component name
- **Device** - RTC device
- **Input clock frequency [s]** - Clock settings. In order to work correctly this value must be set to 1 second (Algorithm used in this component requires a counter incremented each one second).
- **Interrupt service/event** - Support of interrupt service. Events are available only if Interrupt service is enabled. If this property is set to "Enabled", the bean functionality depends on the interrupt service and will not operate if the CPU interrupts are disabled. For details please refer to chapter "interrupt service in the bean's generated code".

  The following items are available only if the group is enabled (the value is "Enabled"):

  - **Interrupt** - Interrupt vector invoked from the RTC interrupt (for information only).
  - **Interrupt priority** - Priority of the interrupt associated with the Real Time Clock
  - *Settings only if "peripheral" support separate interrupt vector for "Seconds" interrupt.*
    - **Interrupt** - Interrupt vector invoked from the RTC interrupt (for information only).
    - **Interrupt priority** - Priority of the interrupt associated with the Real Time Clock
- **Min year** - This propery defines minimal value of the year that can be used in this component. The value may be HW dependant.
- **Max year** - This propery defines maximal value of the year that can be used in this component. The value may be HW dependant.
- **Initialization** - Initial settings. These values are set by the Init(..., bool SoftInit) method if the SoftInit parameter is set to "FALSE" value.
  - **Enabled in init. code** - The component is enabled after initialization code.
  - **Auto Initialization** - Automated initialization of the component. The component Init method is automatically called from CPU component initialization function PE_low_level_init(). In this mode, the constant <ComponentName>_DeviceData is defined in component header file and it can be used as a device data structure pointer that can be passed as a first parameter to all component methods.

    The following items are available only if the group is enabled (the value is "yes"):

    - **Auto init. mode** - See description of the SoftInit parameter in the Init method.
  - **Event mask** - This group defines initialization event mask value.
    - **OnAlarm** - Specifies if OnAlarm event is enabled in initialzation code.
    - *Settings only if "peripheral" support "Day interrupt" or "Day interrupt" is emulated by the driver.*
      - **OnDay** - Specifies if OnDay event is enabled in initialzation code.
    - *Settings only if "peripheral" support "Hour interrupt" or "Hour interrupt" is emulated by the driver.*

        - **OnHour** - Specifies if OnHour event is enabled in initialzation code.
    - *Settings only if "peripheral" support "Minute interrupt" or "Minute interrupt" is emulated by the driver.*
        - **OnMinute** - Specifies if OnMinute event is enabled in initialzation code.
    - *Settings only if "peripheral" support "Second interrupt" or "Second interrupt" is emulated by the driver.*
        - **OnSecond** - Specifies if OnSecond event is enabled in initialzation code.
    - *Settings only if "peripheral" support "2 Hz interrupt"*
        - **On2Hz** - Specifies if On2Hz event is enabled in initialzation code.
    - *Settings only if "peripheral" support "4 Hz interrupt"*
        - **On4Hz** - Specifies if On4Hz event is enabled in initialzation code.
    - *Settings only if "peripheral" support "8 Hz interrupt"*
        - **On8Hz** - Specifies if On8Hz event is enabled in initialzation code.
    - *Settings only if "peripheral" support "16 Hz interrupt"*
        - **On16Hz** - Specifies if On16Hz event is enabled in initialzation code.
    - *Settings only if "peripheral" support "32 Hz interrupt"*
        - **On32Hz** - Specifies if On32Hz event is enabled in initialzation code.
    - *Settings only if "peripheral" support "64 Hz interrupt"*
        - **On64Hz** - Specifies if On64Hz event is enabled in initialzation code.
    - *Settings only if "peripheral" support "128 Hz interrupt"*
        - **On128Hz** - Specifies if On128Hz event is enabled in initialzation code.
    - *Settings only if "peripheral" support "256 Hz interrupt"*
        - **On256Hz** - Specifies if On256Hz event is enabled in initialzation code.
    - *Settings only if "peripheral" support "512 Hz interrupt"*
        - **On512Hz** - Specifies if On512Hz event is enabled in initialzation code.
    - *Settings only if "peripheral" support "Time overflow interrupt"*
        - **OnTimeOverflow** - Specifies if OnTimeOverflow event is enabled in initialzation code.
    - *Settings only if "peripheral" support "Time invalid interrupt"*
        - **OnTimeInvalid** - Specifies if OnTimeInvalid event is enabled in initialzation code.
- **Time and date settings** - This group contains time and date settings.
    - **Time** - Time (hour, minute and second) set in initialization code.
    - **Date** - Date set in initialization code.
- **Alarm settings** - This group contains alarm settings. If disabled, no alarm initialization is done.

  The following items are available only if the group is enabled (the value is "Enabled"):
    - **Alarm time** - Alarm time (hour, minute and second) set in initialization code.
    - **Alarm date** - Alarm date set in initialization code.
- *Settings only if "peripheral" support "compensation period and compensation value features".*
    - **Compensation settings** - This group contains time compensation settings.

- **Compensation period [s]** - Compensation period set in initialization code.
- **Compensation value [clock cycles]** - Compensation value set in initialization code.

- **CPU clock/configuration selection** - Settings for the CPU clock configurations: specifies whether the component is supported or not.

  For details about speed modes please refer to page Speed Modes Support.

  - **Clock configuration 0** - The component is enabled/disabled in the clock configuration 0.
  - **Clock configuration 1** - The component is enabled/disabled in the clock configuration 1.
  - **Clock configuration 2** - The component is enabled/disabled in the clock configuration 2.
  - **Clock configuration 3** - The component is enabled/disabled in the clock configuration 3.
  - **Clock configuration 4** - The component is enabled/disabled in the clock configuration 4.
  - **Clock configuration 5** - The component is enabled/disabled in the clock configuration 5.
  - **Clock configuration 6** - The component is enabled/disabled in the clock configuration 6.
  - **Clock configuration 7** - The component is enabled/disabled in the clock configuration 7.

# 3 Methods

This section describes component's methods. Methods are user-callable functions/subroutines intended for the component runtime control. Please see the Processor Expert user manual for more details.

## 3.1 Init

Initializes the device. There are two modes of initialization.

1) RTC device is NOT battery backed up. (SoftInit = FALSE)

All registers (interrupts, counters, clock routing …) that influence RTC behavior are initialized.

Note: This initialization mode doesn't preserve time and date settings.

2) RTC device IS battery backed up.

(SoftInit = TRUE). Only registers that are not backed up are initialized (Interrupt priority …). RTC device is running, time and date a preserved, RTC interrupts (events) are preserved too.

Allocates memory for the device data structure.

This method can be called only once. Before the second call of Init() the Deinit() must be called first.

**Prototype**

```
LDD_TDeviceData * Init(LDD_TUserData *UserDataPtr, bool SoftInit)
```

**Parameters**

- *UserDataPtr: Pointer to LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer will be passed as an event or callback parameter.
- *SoftInit:bool* - If set to "true" only registers that are not battery backup are initialized (E.g. Interrupt priority in interrupt controller … )

If set to "false" all registers are initialized.

**Return value**

- *Return value:LDD_TDeviceData* - Pointer to the dynamically allocated private structure or NULL if there was an error.

## 3.2 Deinit

Deinitializes the device and frees the device data structure memory.

**Prototype**

```
void Deinit(LDD_TDeviceData *DeviceDataPtr)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Pointer to device data structure pointer returned by Init method.

## 3.3 Enable

Enables the real-time clock module (Typically sets the module enable bit). All other HW settings are preserved. RTC module can be disabled by the Disable method and reinitialized by the Init method.

**Prototype**

```
LDD_TError Enable(LDD_TDeviceData *DeviceDataPtr)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Pointer to device data structure pointer returned by Init method.

**Return value**

- *Return value:LDD_TError* - Error code, possible codes:

  ERR_OK - OK

## 3.4 Disable

Disables the real-time clock module. All other HW settings are preserved. RTC module can be enabled by Enable method.

**Prototype**

```
LDD_TError Disable(LDD_TDeviceData *DeviceDataPtr)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Pointer to device data structure pointer returned by Init method.

**Return value**

- *Return value:LDD_TError* - Error code, possible codes:

ERR_OK - OK

## 3.5 GetEventMask

Returns current events mask. Note: Event that are not generated (See the "Method" tab in the Component inspector) are not handled by this method. Pair method to SetEventMask.

**Prototype**

```
LDD_TEventMask GetEventMask(LDD_TDeviceData *DeviceDataPtr)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Pointer to device data structure pointer returned by Init method.

**Return value**

- *Return value:LDD_TEventMask* - Current EventMask. The component event masks are defined in the PE_HAL.h file.

## 3.6 SetEventMask

Enables/disables event(s). The events contained within the mask are enabled. Events not contained within the mask are disabled. The component event masks are defined in the PE_Types.h file. Note: Event that are not generated (See the "Method" tab in the Component inspector) are not handled by this method. In this case the method returns ERR_VALUE error code. Pair method to GetEventMask.

**Prototype**

```
LDD_TError SetEventMask(LDD_TDeviceData *DeviceDataPtr, LDD_TEventMask EventMask)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Pointer to device data structure pointer returned by Init method.
- *EventMask:LDD_TEventMask* - Event mask

**Return value**

- *Return value:LDD_TError* - Error code, possible codes:
  - ERR_OK - OK.
  - ERR_DISABLED - Component is disabled.
  - ERR_SPEED - Component does not work in the active clock configuration.
  - ERR_PARAM_MASK - Invalid event mask.

## 3.7 GetStatus

This method returns state of interrupt requests and clears pending interrupt flags. Return value has the same format as EventMask parameter of SetEventMask method. Can be used for polling mode without using events. Note: This method can be enabled only if interrupts are disabled (property "Interrupt service/event" is set to "Disabled" value).

**Prototype**

```
LDD_TEventMask GetStatus(LDD_TDeviceData *DeviceDataPtr)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Pointer to device data structure pointer returned by Init method.

**Return value**

- *Return value:LDD_TEventMask* - Actual state of interrupt requests. The component event masks are described in the API constants chapter in this document.

## 3.8 GetTime

Returns actual time and date.

Note: Fields not supported by HW are calculated in software.

**Prototype**

```
void GetTime(LDD_TDeviceData *DeviceDataPtr, LDD_RTC_TTime *TimePtr)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Pointer to device data structure pointer returned by Init method.
- *TimePtr: Pointer to LDD_RTC_TTime* - Pointer to the time structure to fill with current time.

## 3.9 SetTime

Sets new time and date.

Note: All LDD_RTC_TTime structure members must be correctly filled in.

**Prototype**

```
LDD_TError SetTime(LDD_TDeviceData *DeviceDataPtr, LDD_RTC_TTime *TimePtr)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Pointer to device data structure pointer returned by Init method.
- *TimePtr: Pointer to LDD_RTC_TTime* - Pointer to the time structure with new time to set.

**Return value**

- *Return value:LDD_TError* - Error code, possible codes:
  - ERR_OK - OK.
  - ERR_DISABLED - Component is disabled.
  - ERR_SPEED - Component does not work in the active clock configuration.
  - ERR_RANGE - Parameter out of range.

## 3.10 GetAlarm

Returns actual alarm time.

Note: Fields not supported by HW are calculated in software.

**Prototype**

```
void GetAlarm(LDD_TDeviceData *DeviceDataPtr, LDD_RTC_TTime *TimePtr)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Pointer to device data structure pointer returned by Init method.
- *TimePtr: Pointer to LDD_RTC_TTime* - Pointer to the time structure to fill with current alarm time.

## 3.11 SetAlarm

Sets new alarm time.

Note: All LDD_RTC_TTime structure members must be correctly filled in.

**Prototype**

```
LDD_TError SetAlarm(LDD_TDeviceData *DeviceDataPtr, LDD_RTC_TTime *TimePtr)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Pointer to device data structure pointer returned by Init method.
- *TimePtr: Pointer to LDD_RTC_TTime* - Pointer to the time structure with new alarm time to set.

**Return value**

- *Return value:LDD_TError* - Error code, possible codes:
  - ERR_OK - OK.
  - ERR_DISABLED - Component is disabled.
  - ERR_SPEED - Component does not work in the active clock configuration.
  - ERR_RANGE - Parameter out of range.

## 3.12 GetDaylightSaving

*Note: Available only if "peripheral" support "Daylight saving" feature.*

Returns actual daylight saving time and date settings.

**Prototype**

```
void GetDaylightSaving(LDD_TDeviceData *DeviceDataPtr, LDD_RTC_TDaylightSaving *TimePtr)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Pointer to device data structure pointer returned by Init method.
- *TimePtr: Pointer to LDD_RTC_TDaylightSaving* - Pointer to the structure to fill with current daylight saving settings.

## 3.13 SetDaylightSaving

*Note: Available only if "peripheral" support "Daylight saving" feature.*

Sets new daylight saving time.

Note: All LDD_RTC_TDaylightSaving structure members must be correctly filled in.

**Prototype**

```
 LDD_TError SetDaylightSaving(LDD_TDeviceData *DeviceDataPtr, LDD_RTC_TDaylightSaving
*TimePtr)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Pointer to device data structure pointer returned by Init method.
- *TimePtr: Pointer to LDD_RTC_TDaylightSaving* - Pointer to the time structure with new daylight saving time to set.

**Return value**

- *Return value:LDD_TError* - Error code, possible codes:

  ERR_OK - OK

  ERR_RANGE - Parameter out of range

## 3.14 SetCompensationPeriodAndValue

*Note: Available only if "peripheral" support "compensation period and compensation value features".*

Sets compensation period and value.

The compensation logic alters the number of 32.768 kHz clock cycles it takes for the prescaler register to overflow and increment the seconds counter. The time compensation value is used to adjust the number of clock cycles between -127 and +128. Cycles are added or subtracted from the prescaler register when the prescaler register equals 0x3FFF and then increments. The compensation period is used to adjust the frequency at which the time compensation value is used (between once a second to once every 256 seconds).

Updates to the compensation register will not take effect until the next time the seconds register updates and provided the previous compensation interval has expired. When the compensation interval is set to other than once a second then the compensation is applied in the first second interval and the remaining second intervals will receive no compensation.

Compensation is disabled by configuring the time compensation to zero.

**Prototype**

```
 void SetCompensationPeriodAndValue(LDD_TDeviceData *DeviceDataPtr, uint8_t Period, int8_t
Value)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Pointer to device data structure pointer returned by Init method.

- *Period:uint8_t* - Compensation period [s].
- *Value:int8_t* - Compensation value [clock cycles]

## 3.15 SetOperationMode

This method requests to change the component's operation mode. Upon a request to change the operation mode, the component will finish a pending job first and then notify a caller that an operation mode has been changed.

**Prototype**

```
 LDD_TError SetOperationMode(LDD_TDeviceData *DeviceDataPtr, LDD_TDriverOperationMode
OperationMode, LDD_TCallback ModeChangeCallback, LDD_TCallbackParam
*ModeChangeCallbackParamPtr)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Device data structure pointer returned by Init method.
- *OperationMode:LDD_TDriverOperationMode* - Requested driver operation mode.
- *ModeChangeCallback:LDD_TCallback* - Callback to notify the upper layer once a mode has been changed.
- *ModeChangeCallbackParamPtr: Pointer to LDD_TCallbackParam* - Pointer to callback parameter to notify the upper layer once a mode has been changed.

**Return value**

- *Return value:LDD_TError* - Error code, possible codes:
  - ERR_OK - OK.
  - ERR_DISABLED - Component is disabled.
  - ERR_SPEED - Component does not work in the active clock configuration.
  - ERR_PARAM_MODE - Invalid operation mode.

## 3.16 GetDriverState

This method returns the current driver status.

**Prototype**

```
 LDD_TDriverState GetDriverState(LDD_TDeviceData *DeviceDataPtr)
```

**Parameters**

- *DeviceDataPtr: Pointer to LDD_TDeviceData* - Device data structure pointer returned by Init method.

**Return value**

- *Return value:LDD_TDriverState* - The current driver status mask. Following status masks defined in PE_Types.h can be used to check the current driver status.
  - PE_LDD_DRIVER_DISABLED_IN_CLOCK_CONFIGURATION - 1 - Driver is disabled in the current mode; 0 - Driver is enabled in the current mode.

- PE_LDD_DRIVER_DISABLED_BY_USER - 1 - Driver is disabled by the user; 0 - Driver is enabled by the user. .
- PE_LDD_DRIVER_BUSY - 1 - Driver is the BUSY state; 0 - Driver is in the IDLE state..

# 4 Events

This section describes component's events. Events are call-back functions called when an important event occurs. For more general information on events, please see the Processor Expert user manual.

## 4.1 OnAlarm

Called if alarm date and time match the actual date and time, OnAlarm event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void OnAlarm(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.2 OnDay

*Events only if "peripheral" support "Day interrupt" or "Day interrupt" is emulated by the driver.*

Called each day if OnDay event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void OnDay(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.3 OnHour

*Events only if "peripheral" support "Hour interrupt" or "Hour interrupt" is emulated by the driver.*

Called each hour if OnHour event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void OnHour(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.4 OnMinute

*Events only if "peripheral" support "Minute interrupt" or "Minute interrupt" is emulated by the driver.*

Called each minte if OnMinute event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void OnMinute(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.5 OnSecond

*Events only if "peripheral" support "Second interrupt" or "Second interrupt" is emulated by the driver.*

Called each second if OnSecond event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void OnSecond(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.6 On2Hz

*Note: Available only if "peripheral" support "2 Hz interrupt"*

Called each 1/2 [s] if the On2Hz event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void On2Hz(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.7 On4Hz

*Note: Available only if "peripheral" support "4 Hz interrupt"*

Called each 1/4 [s] if the On4Hz event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void On4Hz(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.8 On8Hz

*Note: Available only if "peripheral" support "8 Hz interrupt"*

Called each 1/8 [s] if the On8Hz event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void On8Hz(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.9 On16Hz

*Note: Available only if "peripheral" support "16 Hz interrupt"*

Called each 1/16 [s] if the On16Hz event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void On16Hz(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.10 On32Hz

*Note: Available only if "peripheral" support "32 Hz interrupt"*

Called each 1/32 [s] if the On32Hz event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void On32Hz(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.11 On64Hz

*Note: Available only if "peripheral" support "64 Hz interrupt"*

Called each 1/64 [s] if the On64Hz event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void On64Hz(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.12 On128Hz

*Note: Available only if "peripheral" support "128 Hz interrupt"*

Called each 1/128 [s] if the On128Hz event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void On128Hz(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.13 On256Hz

*Note: Available only if "peripheral" support "256 Hz interrupt"*

Called each 1/256 [s] if the On256Hz event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void On256Hz(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.14 On512Hz

*Note: Available only if "peripheral" support "512 Hz interrupt"*

Called each 1/512 [s] if the On512Hz event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void On512Hz(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.15 OnTimeOverflow

*Events only if "peripheral" support "Time overflow interrupt".*

Called if time overflow is detected, OnTimeOverflow event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void OnTimeOverflow(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

## 4.16 OnTimeInvalid

*Events only if "peripheral" support "Time invalid interrupt".*

Called as soon as time becomes invalid, OnTimeInvalid event is enabled (see SetEventMask and GetEventMask methods) and RTC device is enabled. This event is available only if Interrupt service/event is enabled.

**Prototype**

```
void OnTimeInvalid(LDD_TUserData *UserDataPtr)
```

**Parameters**

- *UserDataPtr:LDD_TUserData* - Pointer to the user or RTOS specific data. This pointer is passed as the parameter of Init method.

# 5 Types and Constants

This section contains definitions of user types and constants. User types are derived from basic types and they are designed for usage in the driver interface. They are declared in the generated code.

**Type Definitions**

- **LDD_RTC_TTime =** struct { Structure used for time operation.

  uint32_t Second; – *seconds (0 - 59)*

  uint32_t Minute; – *minutes (0 - 59)*

  uint32_t Hour; – *hours (0 - 23)*

  uint32_t DayOfWeek; – *day of week (0-Sunday, .. 6-Saturday)*

  uint32_t Day; – *day (1 - 31)*

  uint32_t Month; – *month (1 - 12)*

  uint32_t Year; – *year*

  }

- **LDD_RTC_TDaylightSaving =** struct {

  uint8_t StartHour; – *Daylight saving time start hour.*

  uint8_t StartDay; – *Daylight saving time start day.*

  uint8_t StartMonth; – *Daylight saving time start month.*

  uint8_t EndHour; – *Daylight saving time end hour.*

  uint8_t EndDay; – *Daylight saving time end day.*

  uint8_t EndMonth; – *Daylight saving time end month.*

  }

**Constants**

- **LDD_RTC_ON_SECOND** - OnSecond event mask

  Value: 0x10

- **LDD_RTC_ON_ALARM** - OnAlarm event mask

  Value: 0x04

- **LDD_RTC_ON_STOPWATCH** - OnStopWatch event mask

  Value: 0x01

# 6 Typical usage

This section contains examples of a typical usage of the component in user code. For general information please see the section Component Code Typical Usage in Processor Expert user manual.

Examples of typical settings and usage of RTC_LDD component

- The full\partial initialization.
- The GetTime/SetTime methods.

### Initialization

This example shows how the Init method can be used to initialize RTC hw.

The RTC hw can use it's own power supply in order to be working if the main power source of the system is off. The Init method has the SoftInit parameter that suppress (if set TRUE) the initilaztion of backuped part of the RTC hw.

Required component setup:

- Methods: Init

Content of ProcessorExpert.c:

```
LDD_TDeviceData *MyRTCPtr;

void main(void)
{
  if (Vbat_powr_on == TRUE) {
    MyRTCPtr = RTC_Init((LDD_TUserData *)NULL, FALSE);        /* Initialize the device, reset all RTC
registers */
  } else {
    MyRTCPtr = RTC_Init((LDD_TUserData *)NULL, TRUE);        /* Initialize the device, preserve time
settings */
  }
}
```

### The GetTime and SetTime methods

This example shows how to use the GetTime or SetTime methods.

Note: Get/Set time methods doesn't require interrupt to be enabled. They use HW register(s) and SW algorithm to convert between user friendly time format of time and it's HW representation

Required component setup:

- *Init. time*: 23:50:10
- *Init. time*: 23:12:2010
- Methods: GetTime
- Methods: SetTime

Content of ProcessorExpert.c:

```
char *DayOfWeekName[] = {
  "Sunday",
  "Monday",
  "Tuesady",
  "Wensday",
```

### Typical usage

```
  "Thursday",
  "Friday",
  "Saturday"
};

LDD_TDeviceData *MyRTCPtr;
LDD_RTC_TTime Time;
LDD_TError Error;

void main(void)
{
  if (Vbat_powr_on == TRUE) {
    MyRTCPtr = RTC1_Init((LDD_TUserData *)NULL, FALSE);        /* Initialize the device, reset all RTC
registers */
    RTC1_GetTime(MyRTCPtr, &Time);
  } else {
    MyRTCPtr = RTC_Init((LDD_TUserData *)NULL, TRUE);        /* Initialize the device, preserve time
settings */
    RTC1_GetTime(MyRTCPtr, &Time);
  }
  printf("Current time: %d:%d:%d %d.%d.%d %s\n", Time.Hour, Time.Minute, Time.Second, Time.Day,
Time.Month, Time.Year, DayOfWeekName[Time.DayOfWeek]);
  ...
  /* Set new time */
  Time.Hour = 10;
  Time.Minute = 20;
  Time.Second =30;
  Time.Day = 15;
  Time.Month = 11;
  Time.Year = 2012;
  Error = SetTime(MyRTCPtr, &Time);
  ...
  RTC1_GetTime(MyRTCPtr, &Time);
  printf("New time: %d:%d:%d %d.%d.%d %s\n", Time.Hour, Time.Minute, Time.Second, Time.Day, Time.Month,
Time.Year, DayOfWeekName[Time.DayOfWeek]);
```

After Vbat power on reset the screen should look like:

```
Current time: 23:50:10 1.1.2000 Saturday
New time: 10:20:30 15.11.2012 Monday
```

After system reset the screen should look like:

```
Current time: hh:mm:ss dd.mm.yyyy XXXX
New time: 10:20:30 15.11.2012 Thursday
```

***How to Reach Us:***

**Home Page:**
freescale.com

**Web Support:**
freescale.com/support



Document Number N/A
Revision 1, 12/2013

freescale™