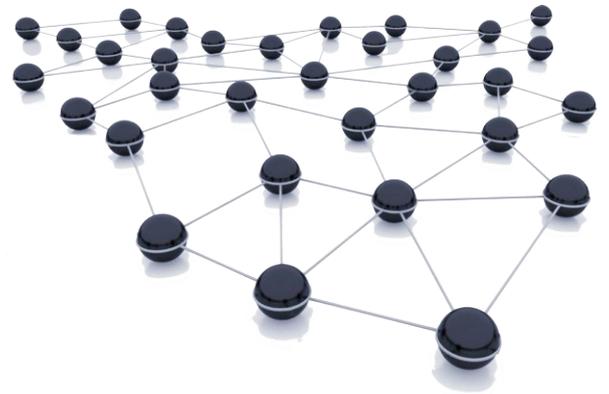


Wasp mote RTC Programming Guide



INDEX

1. Introduction	4
1.1. Wasp mote libraries	4
1.1.1. Wasp mote RTC files	4
1.1.2. Constructor	4
1.1.3. Pre-Defined constants	4
1.1.4. Variables	4
1.1.5. Flags	4
1.1.5.1. Global Interruption Flag	4
1.1.5.2. Global Interruption Flag Array	4
1.1.5.3. Global Interruption Counter	5
2. Initialization	6
2.1. Initializing the RTC	6
2.2. Switching RTC off	6
3. Setting time and date	7
3.1. Setting time and date	7
3.2. Getting time and date	7
3.3. Getting day of week	8
4. Setting Alarms	9
4.1. Setting the Alarm1 (using a string as input)	9
4.2. Setting the Alarm1	10
4.3. Getting the Alarm1	10
4.4. Clear alarm flags	11
4.5. Disable alarms	11
4.6. Capture alarms	11
4.6.1. Identify triggered alarm	12
5. RTC Watchdog	13
5.1. Setting RTC Watchdog	13
5.2. Unsetting RTC Watchdog	13
5.3. Getting RTC Watchdog	13
5.4. Use of the RTC Watchdog	13
6. Using RTC with hibernate	15
7. Unix / Epoch time	16
7.1. Getting Epoch time	16
7.2. Breaking Epoch time into 'Time and Date'	16

8. Code examples and extended information	18
9. API Changelog	20

1. Introduction

This guide explains the RTC (Wasmote's internal Real Time Clock) features and functions. There are some variations in this library for our new product lines Wasmote v15 and Plug & Sense! v15, released on October 2016.

Some functions may not be compatible with Wasmote v12 or Plug & Sense! v12. Besides, old functions may no longer exist. If you are using previous versions of our products, please use the corresponding guides, available on our [Development website](#).

You can get more information about the generation change on the document "[New generation of Libelium product lines](#)".

Differences of this library compared to the previous version:

- Alarm2 is no longer available
- Temperature sensor is no longer available
- New RTC Watchdog reset

1.1. Wasmote libraries

1.1.1. Wasmote RTC files

WaspRTC.h; WaspRTC.cpp

1.1.2. Constructor

To start using the Wasmote RTC library, an object from class `WaspRTC` must be created. This object, called `RTC`, is created inside the Wasmote RTC library and it is public to all libraries. It is used through the guide to show how Wasmote RTC library works.

When creating this constructor, no variables are initialized by default.

1.1.3. Pre-Defined constants

There are some constants defined in 'WaspRTC.h' to help with the comprehension of the code when reading the first times. These constants are related with RTC register addresses, some functions, inputs and alarm modes.

1.1.4. Variables

Some variables have been defined for storing time, date and alarm data. These variables have been named after the data they store (i.e. `RTC.year` stores the year, and `RTC.month` stores the month).

The Wasmote RTC allows setting the day of the week, which is a number between 1-7. This part of the date can be modified by the user providing they are sequential, it meaning that if Sunday is equal to 1, Monday must be equal to 2.

An array called '`registersRTC`' has been created for storing the data to send to the RTC. This array is used in each writing or reading operation.

1.1.5. Flags

There are various flags used for handling interruptions while using Wasmote RTC.

1.1.5.1. Global Interruption Flag

It is used to check the port in which the interruption has got activated. It is used in this library and other libraries which generate interrupts too.

1.1.5.2. Global Interruption Flag Array

It is used to store the number of times each interruption has been detected. It is used in this library and other libraries which generate interrupts too.

1.1.5.3. Global Interruption Counter

It is used to store the number of interruptions detected. It is used in this library and other libraries which generate interrupts too.

2. Initialization

Before getting information from the RTC, the I2C bus has to be initialized.

2.1. Initializing the RTC

The function `RTC.ON()` powers the RTC up and initializes the I2C bus used for communicating with the RTC. It reads from the RTC time, date and alarms, setting the corresponding variables.

It returns nothing and modifies no flag.

Example of use:

```
{  
    RTC.ON();    // Executes the init process  
}
```

2.2. Switching RTC off

The function `RTC.OFF()` closes the I2C bus. The RTC is powered directly by the battery (even when Waspote is off or in sleep mode), so there are no switches to cut power to this chipset. The RTC enters into a low-power state when it detects that no actions are required.

The function returns nothing and modifies no flag.

Example of use:

```
{  
    RTC.OFF();  
}
```

3. Setting time and date

The first step to be able to use the RTC is setting time and date. Some functions have been created for managing these operations.

3.1. Setting time and date

It sets time and date on the RTC.

Time and date are specified by inputs which correspond to the different variables: year, month, date, day, hour, minute and second.

`day` is the day of the week and the range for this value is 1 to 7. By default, Sunday is equal to 1 and Saturday is equal to 7.

The `setTime()` function extracts each part of date and time, storing each part in the corresponding variable. After this, the `registersRTC` array is loaded with these variables and data is sent to the RTC.

This function returns '0' if was successfully executed. In the case of any error '1' is returned.

Example of use:

```
{  
    // Setting date and time [yy:mm:dd:dow:hh:mm:ss]  
    RTC.setTime("09:06:29:02:10:00:00");  
}
```

Related variables:

`year`, `month`, `hour`, `minute`, `second` → stores the time and date set

`day` → stores the day of the week

`date` → stores the day of the month

An example of the functions:

<http://www.libelium.com/development/waspmote/examples/rtc-01-setting-and-reading-time/>

An example of how to set RTC via USB port:

<http://www.libelium.com/development/waspmote/examples/rtc-07-set-waspmote-date/>

3.2. Getting time and date

The `getTime()` function gets time and date, storing them in the `registersRTC` array and the corresponding variables. This function updates all API variables related to time and date:

- `RTC.year`
- `RTC.month`
- `RTC.date`
- `RTC.day`
- `RTC.hour`
- `RTC.minute`
- `RTC.second`

It returns a string containing time and date in the following format: "Day of week, yy/mm/dd, hh:mm:ss". In the case an error happened, the "error" message is returned.

Example of use:

```
{
  USB.println(RTC.getTime());
}
```

Related Variables:

`year`, `month`, `hour`, `minute`, `second` → stores the time and date set
`day` → stores the day of the week
`date` → stores the day of the month

An example of the functions:

<http://www.libelium.com/development/waspmote/examples/rtc-01-setting-and-reading-time/>

3.3. Getting day of week

This function calculates the day of the week based on the year, month and day. Sakamoto's algorithm is used in this function. Valid for any date in the range [September 14, 1.752] – [December 31, 9.999].

The inputs for `dow()` function are: year, month and day of month.

Example of use:

```
{
  uint8_t day_of_week = 0;
  day_of_week = RTC.dow(2012, 12, 4);
}
```

The function returns:

1 → Sunday
2 → Monday
3 → Tuesday
4 → Wednesday
5 → Thursday
6 → Friday
7 → Saturday

4. Setting Alarms

Wasmote RTC provides two alarms to enable interruptions and wake up the device from a low-power state.

There are two different RTC alarms:

- **Alarm1**: is set by day/date, hour, minutes and seconds.
- **Alarm2**: is set by day/date, hour and minutes. **Alarm2 is reserved for the RTC Watchdog.**

There are three inputs for setting a new alarm: time, offset and mode.

- **Time**: represents the time/date for the alarm.
- **Offset**: represents the two modes for setting the alarm time: offset or absolute. When offset is selected, the input time is added to the actual time in the RTC and the result is set as the alarm time. When absolute is selected, the input time is set as the alarm time.
- **Mode**: represents the different alarm modes. Alarm1 has 6 modes and Alarm2 has 5 modes.

When the time set in Alarm1 matches the RTC time, the RTC interruption pin is triggered. This pin is connected to a specific microcontroller interruption pin, so the alarm can wake up the microcontroller from a sleep mode.

An example of the functions:

<http://www.libelium.com/development/wasmote/examples/rtc-02-setting-reading-alarms/>

4.1. Setting the Alarm1 (using a string as input)

It sets the Alarm1 to the specified time, offset and mode.

The input 'time' has the following format: "dd:hh:mm:ss".

The input 'offset' has some possible values:

RTC_OFFSET: 'time' is added to the current time read from RTC

RTC_ABSOLUTE: 'time' is set as the time for Alarm1

The input 'mode' specifies the mode for Alarm1. Possible values are:

RTC_ALM1_MODE1: Day, hours, minutes and seconds match

RTC_ALM1_MODE2: Date, hours, minutes and seconds match

RTC_ALM1_MODE3: Hours, minutes and seconds match

RTC_ALM1_MODE4: Minutes and seconds match

RTC_ALM1_MODE5: Seconds match

RTC_ALM1_MODE6: Once per second

When this function is called, the Alarm1 is set and no more functions need to be executed.

It returns nothing, but when the Alarm1 matches the time, interruption flags will be modified to indicate it.

Example of use:

```
{  
    // Example: Sets Alarm1 for 29th of the month at 11:00:00  
    RTC.setAlarm1("29:11:00:00", RTC_ABSOLUTE, RTC_ALM1_MODE2 );  
  
    // Example: Sets Alarm1 for 5 minutes from now  
    RTC.setAlarm1("00:00:05:00", RTC_OFFSET, RTC_ALM1_MODE4 );  
}
```

Related Variables:

`day_alarm1` → stores the day or date of the Alarm1

`hour_alarm1`, `minute_alarm1`, `second_alarm1` → store the time of the Alarm1

4.2. Setting the Alarm1

It sets the Alarm1 to the specified time, offset and mode.

The inputs 'day_date', '_hour', '_minute' and '_second' specify the time for the Alarm1.

The input 'offset' has some possible values:

`RTC_OFFSET`: 'time' is added to the current time read from the RTC

`RTC_ABSOLUTE`: 'time' is set as the time for the Alarm1

The input 'mode' specifies the mode for the Alarm1. Possible values are:

`RTC_ALM1_MODE1`: Day, hours, minutes and seconds match

`RTC_ALM1_MODE2`: Date, hours, minutes and seconds match

`RTC_ALM1_MODE3`: Hours, minutes and seconds match

`RTC_ALM1_MODE4`: Minutes and seconds match

`RTC_ALM1_MODE5`: Seconds match

`RTC_ALM1_MODE6`: Once per second

When this function is called, the Alarm1 is set and no more functions need to be executed.

It returns nothing, but when the Alarm1 matches the time, interruption flags will be modified to indicate it.

Example of use:

```
{
    // Example: Sets Alarm1 for 29th of the month at 11:00:00
    RTC.setAlarm1( 29,11,0,0, RTC_ABSOLUTE, RTC_ALM1_MODE2 );

    // Example: Sets Alarm1 for 5 minutes from now
    RTC.setAlarm1( 0,0,5,0, RTC_OFFSET, RTC_ALM1_MODE4 );
}
```

Related Variables:

`day_alarm1` → stores the day or date of Alarm1

`hour_alarm1`, `minute_alarm1`, `second_alarm1` → store the time of Alarm1

4.3. Getting the Alarm1

It gets the Alarm1 time from RTC.

It returns a string containing this time and date for the Alarm1.

Example of use:

```
{
    // Gets time for Alarm1
    USB.println(RTC.getAlarm1());
}
```

Related Variables:

`day_alarm1` → stores the day or date of the Alarm1

`hour_alarm1`, `minute_alarm1`, `second_alarm1` → store the time of the Alarm1

4.4. Clear alarm flags

It clears alarm flags (`A1F` and `A2F`) in the RTC. The alarm flag `A2F` is related to the Watchdog functionality.

If these flags are not cleared after an interrupt is captured, no more interrupts will be captured.

Example of use:

```
{  
    RTC.clearAlarmFlag();  
}
```

4.5. Disable alarms

There are specific functions to disable preset RTC alarms. These functions avoid the interruption to be executed.

The `disableAlarm1()` function disables the RTC Alarm1.

Example of use:

```
{  
    RTC.disableAlarm1();  
}
```

4.6. Capture alarms

If an RTC alarm has been set, when the time event happens, some library flags are updated in order to know the alarm has been captured.

The general interruption register `intFlag` stores the captured events. In the case of the RTC alarms, it is necessary to check the value of this register so as to identify the RTC alarm generation. For further information refer to the **Interruptions Programming Guide**.

Example of use:

```
{  
    if (intFlag & RTC_INT)  
    {  
        // RTC captured  
    }  
}
```

RTC alarm example:

<http://www.libelium.com/development/waspmote/examples/rtc-06-complete-example/>

4.6.1. Identify triggered alarm

When an RTC alarm is captured it is possible to distinguish which one generated the event: Alarm1 or Alarm2. To that purpose, the `clearAlarmFlag()` function, which is internally called when exiting, a low-power-consumption state, updates the `alarmTriggered` attribute indicating the value to identify the alarm generation. Possible values are:

'1': Alarm1 triggered

'2': Alarm2 triggered

'3': Both alarms triggered

Example of use:

```
{
  if (intFlag & RTC_INT)
  {
    if (RTC.alarmTriggered == 1) { // Alarm1 }
    if (RTC.alarmTriggered == 2) { // Alarm2 }
    if (RTC.alarmTriggered == 3) { // Both Alarm1 & Alarm2 }
  }
}
```

How to know the RTC triggered alarm example:

<http://www.libelium.com/development/waspmote/examples/rtc-09-triggered-alarm/>

5. RTC Watchdog

The Alarm2 of the RTC is connected to a Watchdog reset circuit which permits to reset the microcontroller of Waspote when the Alarm2 is triggered.

5.1. Setting RTC Watchdog

The `setWatchdog()` function sets the RTC Watchdog to the specified time. The input parameter specifies the time in minutes (1-59) the Watchdog will wait to reset the Waspote microcontroller.

When this function is called, the Alarm2 is set and no more functions need to be executed.

It returns nothing, but when the Alarm2 matches the time (minutes), the microcontroller will be reset.

Example of use:

```
{  
    RTC.setWatchdog(1);  
}
```

5.2. Unsetting RTC Watchdog

The `unSetWatchdog()` function disables the RTC Watchdog.

When this function is executed, the RTC Watchdog is disabled and the microcontroller will not be reset.

It returns nothing.

Example of use:

```
{  
    RTC.unSetWatchdog();  
}
```

5.3. Getting RTC Watchdog

The `getWatchdog()` function gets the RTC Watchdog time.

It returns a string containing time and date for the next RTC Watchdog reset.

Example of use:

```
{  
    USB.println(RTC.getWatchdog());  
}
```

5.4. Use of the RTC Watchdog

This RTC Watchdog has been implemented for resetting Waspote if it gets stuck. That periodical reset avoids erratic behaviour.

This is highly recommended for applications that need to be very robust and can never stop working. This feature ensures that Waspote will never stop working.

RTC Watchdog requires the Watchdog switch to be turned on correctly.

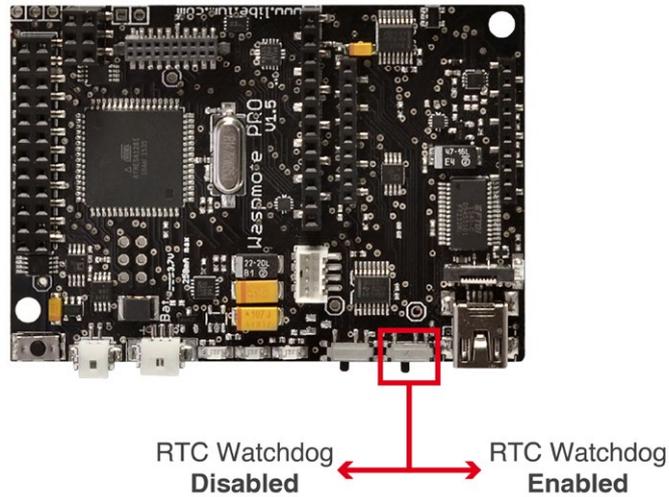


Figure: RTC Watchdog switch

An example of the use of RTC Watchdog:

<http://www.libelium.com/development/waspmote/examples/rtc-10-set-watchdog/>

What you need to do is to set the Watchdog in the setup and then on each loop disable and enable it again. If the code disables the Watchdog before it actuates, it means that everything went well and there was no need for a reset; then the code should activate the Watchdog again. So the user will have to adjust the reset time to be a bit longer than the normal time from loop to loop. If Waspote gets stuck (detected because this timeout expired), Waspote will be automatically reset when the time set in the RTC Watchdog matches the current time. The Watchdog corrects any erratic behaviour in minutes.

6. Using RTC with hibernate

If the hibernate mode is used, the RTC alarm is the only option for waking up the Waspote platform. When the hibernate switch is set for this low-power consumption mode, any RTC alarm arriving while the code is running could cause internal collisions. The RTC alarm is supposed to trigger when Waspote is hibernating.

There are several ways to set alternative alarms:

- use the Watchdog
- compare current time and date with previous conditions
- use the `millis()` function

7. Unix / Epoch time

Unix time (also known as **POSIX** time or **Epoch** time) is a system for describing instants in time, defined as the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970, not counting leap seconds. It is used widely in Unix-like and many other operating systems and file formats.

Example: 1419086327 (2014-12-20T14:38:47Z)

Wasmote API defines some functions to use this time format which can be useful in order to translate dates into seconds and perform addition or subtraction of time.

An example of the functions that are shown below:

<http://www.libelium.com/development/wasmote/examples/rtc-08-unixepoch-time>

7.1. Getting Epoch time

The function `getEpochTime()` permits to calculate the Unix time associated to a specific year, month, day, hour, minute and second values. It is possible to indicate these values as input or use the current values of the RTC.

Example of use:

```
{
  // Define variable to store Epoch time
  unsigned long epoch;

  // Example: Get Epoch time from RTC values
  epoch = RTC.getEpochTime();

  // Example: Get Epoch time from input values(i.e: 2014-12-20 at 14:38:47)
  epoch = RTC.getEpochTime( 14, 12, 20, 14, 38, 47 );
}
```

Note: The function `getEpochTime()` updates the RTC attributes because it calls `getTime()` function.

7.2. Breaking Epoch time into 'Time and Date'

The function `breakTimeAbsolute()` permits to convert the input epoch time to time and date values (as UTC components). Thus, it can be useful to add/subtract times. For example: get Epoch time from RTC and add several seconds to calculate a new time instant.

The structure called `timestamp_t` is defined in WaspRTC class for converting epoch time into UTC timestamps.

Example of use:

```
{
  // Define variable for UTC timestamps
  timestamp_t time;

  // Break Epoch time into UTC time
  RTC.breakTimeAbsolute( epoch, &time );

  // Available info: UTC Date
  USB.print( time.year, DEC );
  USB.print( time.month, DEC );
  USB.print( time.date, DEC );
  USB.print( time.hour, DEC );
  USB.print( time.minute, DEC );
  USB.print( time.second, DEC );
}
```

The function `breakTimeOffset()` permits to convert the input (great period of seconds) into offset time values (as the number of days, hours, minutes and seconds that the input argument defines). Thus, it can be useful to add/subtract times. For example: get Epoch time from RTC in different instants and then calculate the difference as number of days, hours, minutes and seconds.

The structure called `timestamp_t` is defined in `WaspRTC` class for converting great periods of seconds into number of days, hours, minutes and seconds.

Example of use:

```
{
  // Define variable for timestamps
  timestamp_t  time;

  // Get offset time from '411361' seconds
  RTC.breakTimeOffset( 411361, &time );

  // Available info: '4' days, '18' hours, '16' minutes and '1' seconds
  USB.print( time.date,   DEC );
  USB.print( time.hour,   DEC );
  USB.print( time.minute, DEC );
  USB.print( time.second, DEC );
}
```

8. Code examples and extended information

In the WaspMote Development section you can find complete examples:

<http://www.libelium.com/development/waspmote/examples>

Next lines show a complete example of use of the RTC functions.

```
/*
 * -----Waspmote RTC Complete Example-----
 *
 * Explanation: This example shows how to read the temperature using
 * the sensor integrated in Waspmote
 *
 * Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 * http://www.libelium.com
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 *
 * Version:          0.1
 * Design:           David Gascón
 * Implementation:   Marcos Yarza
 */

void setup(){

  // Setup for Serial port over USB
  USB.ON();
  USB.println(F("USB port started..."));

  // Powers RTC up, init I2C bus and read initial values
  USB.println(F("Init RTC"));
  RTC.ON();

}

void loop(){

  USB.println(F("++++++++++++++++++++++++++++++++++++++++++++++++++++"));

  // Setting time
  RTC.setTime("09:10:20:03:17:35:00");
  USB.print(F("Time: "));
  USB.println(RTC.getTime());

  // Setting and getting Alarms
  RTC.setAlarm1("20:17:35:30", RTC_ABSOLUTE, RTC_ALM1_MODE2);
  USB.print(F("Alarm1: "));
  USB.println(RTC.getAlarm1());

  // Setting Waspmote to Low-Power Consumption Mode
  PWR.sleep(ALL_OFF);
}
```

```
// After setting Waspote to power-down, UART is closed, so it
// is necessary to open it again
USB.ON();

// Waspote wakes up at '17:35:30'
if( intFlag & RTC_INT )
{
  intFlag &= ~(RTC_INT); // Clear flag
  Utils.blinkLEDs(1000); // Blinking LEDs
  Utils.blinkLEDs(1000); // Blinking LEDs
}

delay(5000);
}
```

9. API Changelog

Keep track of the software changes on this link:

www.libelium.com/development/waspmote/documentation/changelog/#RTC