micropython-ntp

Description

A robust MicroPython **Time library** for manipulating the **RTC** and and syncing it from a list of **NTP** servers.

Features:

1. Sync the RTC from a NTP host
2. Multiple NTP hosts
3. Microsecond precision
4. RTC chip-agnostic
5. Calculate and compensate RTC drift
6. Timezones
7. Epochs
8. Day Light Saving Time
9. Get time in sec, ms and us
10. Custom Logger with callback function

Unfinished:

1. Extra precision - take into account the time required to call the functions and compensate for that. This is a rather controversial feature, which may be not be implemented at all.
2. Unit tests

!!!At this point all the implemented features are robustly tested and they seem stable enough for production, BUT I do not recommended to use it in a production environment until the API stabilization phase is finished and some unit tests are developed.!!!

Initialize the library

The first thing to do when using the library is to set a callback function for accessing the RTC chip. The idea behind this strategy is that the library can manipulate multiple RTC chips(internal, external or combination of both).

The callback is a function in the form `func(datetime: tuple)`.

With no arguments, this method returns an 8-tuple with the current date and time. With 1 argument (being an 8-tuple) it sets the date and time.

The `datetime` tuple has the following format: `tuple(year, month, day, weekday, hours, minutes, seconds, subseconds)`

`weekday` is 1-7 for Monday through Sunday.

Micropython example:

```
from machine import RTC
from ntp import Ntp

_rtc = RTC()
Ntp.set_datetime_callback(_rtc.datetime)
```

RTC sync

For syncing the RTC you have to set a list of hosts first

```
Ntp.set_hosts(('0.pool.ntp.org', '1.pool.ntp.org', '2.pool.ntp.org'))
```

and then run

```
Ntp.rtc_sync()
```

The hosts values can be valid hostnames or IP addresses. I the value is neither a valid hostname or IP address, it is skipped WITHOUT an error being thrown. It is your responsibility to pass the correct values.

This function will try to read the time from the hosts list. The first available host will be used to set the time of the RTC in **UTC**.

A timeout in seconds can be set when accessing the hosts

```
Ntp.set_ntp_timeout(timeout_s: int = 1)
```

Reading the time

To read the time, a set of functions are available

```
Ntp.time_s(epoch: int = None, utc: bool = False)
Ntp.time_ms(epoch: int = None, utc: bool = False)
Ntp.time_us(epoch: int = None, utc: bool = False)
```

The suffix of each function shows how the time will be represented.

- _**s** suffix - means seconds
- _**ms** suffix - means milliseconds
- _**us** suffix - means microseconds

If you want to get the time relative to an epoch, you can pass one of the following constants:

```
EPOCH_1900
EPOCH_1970
EPOCH_2000
```

If epoch parameter is None, the default epoch will be used.

To get a UTC time, just set `utc = True`. The UTC time excludes the Daylight Saving Time and the Timezone offsets.

Epochs

Another nice feature is the ability to calculate the time relative to a selected epoch. In micropython the default epoch is `2000-01-01 00:00:00 UTC`. To use an epoch other than the EPOCH_2000, you should call `set_epoch` in the beginning, before you start using the class.

```
Ntp.set_epoch(epoch: int = EPOCH_2000):
```

where `epoch` can be one of:

```
Ntp.EPOCH_1900
Ntp.EPOCH_1970
Ntp.EPOCH_2000
```

RTC drift

Compare the local RTC with the network time and calculate how much the local RTC is drifting. Calculating the drift is easily done by calling

```
Ntp.drift_calculate()
```

Connects to the NTP server and returns the calculated ppm and the time in micro seconds, either positive or negative. Positive values represent a RTC that is speeding, negative values represent RTC that is lagging, when the value is zero, the RTC can be considered accurate. For this function to work, you have to sync the RTC first. My personal recommendation is to wait for at least 15 minutes after syncing the RTC and then to calculate the drifting. Longer time periods will give you more accurate value.

To calculate the drift at a latter stage, you can run

```
Ntp.drift_us(ppm_drift: float = None)
```

This function does not read the time from the NTP server(no internet connection is required), instead it uses the previously calculated ppm.

If you know in advance how much is the local RTC drifting, you can set it manually by calling

```
Ntp.set_drift_ppm(ppm: float)
```

The `ppm` parameter can be positive or negative. Positive values represent a RTC that is speeding, negative values represent RTC that is lagging.

Here is a list of all the functions that are managing the drift

```
Ntp.drift_calculate(cls)
Ntp.drift_last_compensate()
Ntp.drift_last_calculate()
Ntp.drift_ppm(cls)
Ntp.set_drift_ppm(ppm: float)
Ntp.drift_us(ppm_drift: float = None)
Ntp.drift_compensate(compensate_us: int)
```

Timezones

The library has support for timezones. Setting the timezone ensures basic correctness checks and sets the timezone. Just call

```
Ntp.set_timezone(hour: int, minute: int = 0)
```

!!! NOTE: When syncing or compensating the RTC, the time will be set in UTC !!!

When you get the time with

```
Ntp.time_s()
Ntp.time_ms()
Ntp.time_us()
```

the timezone and DST will be calculated automatically.

Daylight Saving Time

The library supports calculating the time according to the Daylight Saving Time. To start using the DST functionality you have to set three things first:

- DST start date and time
- DST end date and time
- DST bias

These parameters can be set with just one function `set_dst(start: tuple, end: tuple, bias: int)` for convenience or you can set each parameter separately with a dedicated function. Example:

```
# Set DST data in one pass
# start (tuple): 4-tuple(month, week, weekday, hour) start of DST
# end (tuple) :4-tuple(month, week, weekday, hour) end of DST
# bias (int): Daylight Saving Time bias expressed in minutes
Ntp.set_dst(cls, start: tuple = None, end: tuple = None, bias: int = 0)

# Set the start date and time of the DST
# month (int): number in range 1(Jan) - 12(Dec)
# week (int): integer in range 1 - 6. Sometimes there are months when they can
spread over a 6 weeks ex. 05.2021
# weekday (int): integer in range 0(Mon) - 6(Sun)
# hour (int): integer in range 0 - 23
Ntp.set_dst_start(month: int, week: int, weekday: int, hour: int)

# Set the end date and time of the DST
# month (int): number in range 1(Jan) - 12(Dec)
# week (int): number in range 1 - 6. Sometimes there are months when they can
spread over 6 weeks.
# weekday (int): number in range 0(Mon) - 6(Sun)
# hour (int): number in range 0 - 23
Ntp.set_dst_end(cls, month: int, week: int, weekday: int, hour: int)

# Set Daylight Saving Time bias expressed in minutes.
# bias (int): minutes of the DST bias. Correct values are 30, 60, 90 and 120
Ntp.set_dst_time_bias(cls, bias: int)
```

You can disable DST functionality by setting any of the start or end date time to `None`

```
# Default values are `None` which disables the DST
Ntp.set_dst()
```

To calculate if DST is currently in effect:

```
Ntp.dst()
```

The function returns 0 if DST not currently in effect or it is disabled. Otherwise the function returns the bias in seconds.

Logger

The library support setting a custom logger. If you want to redirect the error messages to another destination, set your logger

```
Ntp.set_logger(callback = print)
```

The default logger is `print()` and to set it just call the method without any parameters. To disable logging, set the callback to "None"

Example

```python
from machine import RTC
from ntp import Ntp
import time

def ntp_log_callback(msg: str):
    print(msg)

_rtc = RTC()

# Initializing
Ntp.set_datetime_callback(_rtc.datetime)
Ntp.set_logger_callback(ntp_log_callback)

# Set a list of valid hostnames/IPs
Ntp.set_hosts(('0.pool.ntp.org', '1.pool.ntp.org', '2.pool.ntp.org'))
# Network timeout set to 1 second
Ntp.set_ntp_timeout(1)
# Set timezone to 2 hours and 0 minutes
Ntp.set_timezone(2, 0)
# If you know the RTC drift in advance, set it manually to -4.6ppm
Ntp.set_drift_ppm(-4.6)
# Set epoch to 1970. All time calculations will be according to this epoch
Ntp.set_epoch(Ntp.EPOCH_1970)
# Set the DST start and end date time and the bias in one go
Ntp.set_dst((Ntp.MONTH_MAR, Ntp.WEEK_LAST, Ntp.WEEKDAY_SUN, 3),
            (Ntp.MONTH_OCT, Ntp.WEEK_LAST, Ntp.WEEKDAY_SUN, 4),
            60)


# Syncing the RTC with the time from the NTP servers
Ntp.rtc_sync()

# Let the RTC drift for 1 hour
time.sleep(1 * 60 * 60)

# Calculate the RTC drift
Ntp.drift_calculate()

# Let the RTC drift for 3 hours
time.sleep(3 * 60 * 60)

# Compensate the RTC drift
Ntp.drift_compensate(Ntp.drift_us())
```

Dependencies

- Module sockets
- Module struct
- Module time
- Module re
-

Download

You can download the project from GitHub:

```
git clone https://github.com/ekondayan/micropython-ntp.git micropython-ntp
```

License