# SciOx Python gizmo work

## Time

It appears that ESP32 has a sort of RTC, which initialises itself after the wifi has connected. To make it work properly you need:

* A WiFi connection
* import ntptime
* ntptime.settime()

which should set the RTC so that time.localtime() returns the time values tuple with the right values.

The RTC is not very good, especially if you use lightsleep or deepsleep, because that cuts the power to the crystal oscillator and thus the timing then relies on an RC time characteristic which is pants <- technical term.

## Logging

Python standard-like logging can work but there are constraints and problems with it. To make it work as you’d expect, you need

* A WiFi connection
* import mip (this is the MicroPython Installs Python tool, which loads direct to the ESP32’s flash filesystem – cool!
* mip.install(‘logging’) which installs an .mpy version of the MP version of the library
* This version has a “bug/feature” in that the “asctime” formatting specification doesn’t work, because the logging.Formatter looks for time.strftime(), which doesn’t normally exist in mP… so, get the time extension library!
* mip.install(‘time’)
* Then it all works…

Example

import time, machine, ntptime

# Use the mP-specific const function for constant string defs

from micropython import const

import network

# Check out the RTC

print("before WiFi", time.localtime())

w = network.WLAN(network.STA\_IF)

w.active(True)

w.connect("ssid", "password")

while not (w.isconnected()):

pass

ipaddr = w.ifconfig()[0]

print('Connected! IP: ' + ipaddr) # Need this to connect!!

ntptime.settime()

print("after NTP", time.localtime())

logging.basicConfig(

level=logging.INFO,

format='%(asctime)s %(levelname)s - %(name)s - %(message)s'

)

logger = logging.getLogger("MyName")

logger.info("test the new module")

which should produce output like

2024-10-05 12:24:33 INFO - MyName - test the new module

Except (msecs) doesn’t, because the time.time() call on the esp32/8266 version of μP does not provide a float value with “ssssss.nnnnnnnnn” structure, on which the LogRecord code relies to determine the msec value (it uses int() to round off the float part, and then take the new value from the original, and multiply the result by 1000).

I constructed a new LogRecord class, derived from the original, which ended up (horribly) converting the original time.time() value, which is a very large nanosec resolution integer to a string, selecting the required parts and converting them back to a numeric. That prevented loss of resolution during binary vs decimal arithmetic. At least it only happens if the logging request would generate a record!

class ESP32LogRecord(LogRecord):

def set(self, name, level, message):

super().set(name, level, message)

ct\_ns = time.time\_ns()

stringCtns = str(ct\_ns)

self.ct = int(stringCtns[0:9])

self.msecs = int(stringCtns[9:15])

Now it all works, providing the new ESP32LogRecord is inserted into the root logger object at the start of operations:

logger.record = ESP32LogRecord()