CanSat Software

0.2

Generated on Thu Jan 29 2026 for CanSat Software by Doxygen 1.16.0

Thu Jan 29 2026 14:47:33

Chapter 1

Directory Hierarchy

1.1 Directories

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Directory Documentation

4.1 src Directory Reference

Files

- file atm_sen_module.c
- file atm_sen_module.h

  Atmospheric sensors module interface.
- file debug_mode.h

  Conditional debug logging and error handling macros.
- file gps_module.c

  Implementation of GPS NMEA parsing and UART handling.
- file gps_module.h

  GPS module interface.
- file hw_config.c
- file main.c
- file microsd_module.c
- file microsd_module.h

  Micro SD Card reader interface.
- file radio_module.c
- file radio_module.h

  High-level telemetry interface for nRF905 radio.
- file time_manager.c
- file time_manager.h

  Software RTC with GPS synchronization and FatFS integration.

Chapter 5

Class Documentation

5.1 current_time_t Struct Reference

The structure stores data about the date, hour, minutes and seconds a reading was made and saved.

```
#include <time_manager.h>
```

Public Attributes

- uint16_t year
- uint8_t month
- uint8_t day
- uint8_t hour
- uint8_t min
- uint8_t sec
- uint16_t photo_count

5.1.1 Detailed Description

The structure stores data about the date, hour, minutes and seconds a reading was made and saved.

It is used in the process of synchronizing timestamps with the GPS-read time data.

5.1.2 Member Data Documentation

5.1.2.1 day

```
uint8_t current_time_t::day
```

5.1.2.2 hour

```
uint8_t current_time_t::hour
```

5.1.2.3 min

```
uint8_t current_time_t::min
```

5.1.2.4 month

```
uint8_t current_time_t::month
```

5.1.2.5 photo_count

```
uint16_t current_time_t::photo_count
```

5.1.2.6 sec

```
uint8_t current_time_t::sec
```

5.1.2.7 year

```
uint16_t current_time_t::year
```

The documentation for this struct was generated from the following file:

- src/time_manager.h

5.2 gps_data_t Struct Reference

The structure keeps the parsed GPS position and time information.

```
#include <gps_module.h>
```

Public Attributes

- float latitude
- float longitude

  Latitude in decimal degrees.
- float altitude

  Longitude in decimal degrees.
- uint8_t satellites

  Altitude above mean sea level in meters [m].
- uint8_t hour

  Number of satellites currently in view.
- uint8_t min

  UTC Hour.
- uint8_t sec

  UTC Minute.
- uint16_t year

  UTC Second.
- uint8_t month
- uint8_t day
- bool fix

5.2.1 Detailed Description

The structure keeps the parsed GPS position and time information.

It is updated whenever a valid NMEA senstence ($GPRMC in this case) is succesfully parsed.

5.2.2 Member Data Documentation

5.2.2.1 altitude

```
float gps_data_t::altitude
```

Longitude in decimal degrees.

5.2.2.2 day

```
uint8_t gps_data_t::day
```

5.2.2.3 fix

```
bool gps_data_t::fix
```

5.2.2.4 hour

```
uint8_t gps_data_t::hour
```

Number of satellites currently in view.

5.2.2.5 latitude

```
float gps_data_t::latitude
```

5.2.2.6 longitude

```
float gps_data_t::longitude
```

Latitude in decimal degrees.

5.2.2.7 min

```
uint8_t gps_data_t::min
```

UTC Hour.

5.2.2.8 month

```
uint8_t gps_data_t::month
```

5.2.2.9 satellites

```
uint8_t gps_data_t::satellites
```

Altitude above mean sea level in meters [m].

5.2.2.10 sec

```
uint8_t gps_data_t::sec
```

UTC Minute.

5.2.2.11 year

```
uint16_t gps_data_t::year
```

UTC Second.

The documentation for this struct was generated from the following file:

- src/gps_module.h

5.3 sensor_readings_t Struct Reference

Structure for keeping all data read from the atmospheric sensors.

```
#include <atm_sen_module.h>
```

Public Attributes

- double pressure_pa
- double altitude_m

    Pressure reading in Pascals.
- float temperature_c

    Altitude in meters.
- float humidity_pct

    Temperature in Celsius degrees.
- float methane_ppm

    Humidity percentage.
- float ammonia_ppm

    Methane in particles per million.
- float oxygen_pct

    Ammonia in particles per million.

5.3.1 Detailed Description

Structure for keeping all data read from the atmospheric sensors.

5.3.2 Member Data Documentation

5.3.2.1 altitude_m

```
double sensor_readings_t::altitude_m
```

Pressure reading in Pascals.

5.3.2.2 ammonia_ppm

```
float sensor_readings_t::ammonia_ppm
```

Methane in particles per million.

5.3.2.3 humidity_pct

```
float sensor_readings_t::humidity_pct
```

Temperature in Celsius degrees.

5.3.2.4 methane_ppm

```
float sensor_readings_t::methane_ppm
```

Humidity percentage.

5.3.2.5 oxygen_pct

```
float sensor_readings_t::oxygen_pct
```

Ammonia in particles per million.

5.3.2.6 pressure_pa

```
double sensor_readings_t::pressure_pa
```

5.3.2.7 temperature_c

```
float sensor_readings_t::temperature_c
```

Altitude in meters.

The documentation for this struct was generated from the following file:

- src/atm_sen_module.h

Chapter 6

File Documentation

6.1 src/atm_sen_module.c File Reference

```
#include <math.h>
#include "hardware/i2c.h"
#include "hardware/adc.h"
#include "atm_sen_module.h"
#include "debug_mode.h"
#include "lib/bmp280/bmp280_i2c.h"
#include "lib/bmp280/bmp280_i2c_hal.h"
#include "dfrobot_oxygen.h"
```

Macros

- #define I2C_PORT i2c0
- #define SHTC3_ADDR 0x70
- #define PIN_SDA 12
- #define PIN_SCL 13
- #define O2_ADDR 0x74
- #define PIN_METHANE 26
- #define PIN_AMMONIA 28

Functions

- void init_all_sensors ()

    Initializes all sensors with their I2C, SPI or UART connection (depending on which is necessary).
- void read_all (sensor_readings_t ∗gathered_data)

    Begins readings from all the atmospheric sensors.

6.1.1 Macro Definition Documentation

6.1.1.1 I2C_PORT

```
#define I2C_PORT i2c0
```

6.1.1.2 O2_ADDR

```
#define O2_ADDR 0x74
```

6.1.1.3 PIN_AMMONIA

```
#define PIN_AMMONIA 28
```

6.1.1.4 PIN_METHANE

```
#define PIN_METHANE 26
```

6.1.1.5 PIN_SCL

```
#define PIN_SCL 13
```

6.1.1.6 PIN_SDA

```
#define PIN_SDA 12
```

6.1.1.7 SHTC3_ADDR

```
#define SHTC3_ADDR 0x70
```

6.1.2 Function Documentation

6.1.2.1 init_all_sensors()

```
void init_all_sensors (
             void )
```

Initializes all sensors with their I2C, SPI or UART connection (depending on which is necessary).

Within the function, for every sensor that needs initialization a dedicated function is called. This lays the grounds for starting continous readings.

Note

This must be called once at system startup before the main loop.

6.1.2.2 read_all()

```
void read_all (
            sensor_readings_t * gathered_data)
```

Begins readings from all the atmospheric sensors.

Within the function, for every sensor a dedicated _read function is called. The data read is then immediately saved in a dedicated structure.

Parameters

out	*gathered_data*	Pointer to the an object of a 'sensor_readings_t' structure, to which data read from sensors is immediatly saved to.

6.2 src/atm_sen_module.h File Reference

Atmospheric sensors module interface.

```
#include <stdio.h>
#include <pico/stdlib.h>
#include <stdint.h>
```

Classes

- struct sensor_readings_t

  Structure for keeping all data read from the atmospheric sensors.

Functions

- void init_all_sensors (void)

  Initializes all sensors with their I2C, SPI or UART connection (depending on which is necessary).
- void read_all (sensor_readings_t ∗gathered_data)

  Begins readings from all the atmospheric sensors.

6.2.1 Detailed Description

Atmospheric sensors module interface.

This file handles the configuration of all atmospheric sensors, through I2C, SPI and UART. It provides functions to initialize and conduct readings from all the sensors, saving them to a dedicated 'sensor_readings_t' structure.

6.2.2 Function Documentation

6.2.2.1 init_all_sensors()

```
void init_all_sensors (
            void )  [extern]
```

Initializes all sensors with their I2C, SPI or UART connection (depending on which is necessary).

Within the function, for every sensor that needs initialization a dedicated function is called. This lays the grounds for starting continous readings.

Note

This must be called once at system startup before the main loop.

6.2.2.2 read_all()

```
void read_all (
            sensor_readings_t * gathered_data)  [extern]
```

Begins readings from all the atmospheric sensors.

Within the function, for every sensor a dedicated _read function is called. The data read is then immediately saved in a dedicated structure.

Parameters

out	*gathered_data*	Pointer to the an object of a 'sensor_readings_t' structure, to which data read from sensors is immediatly saved to.

6.3 atm_sen_module.h

Go to the documentation of this file.

```


#ifndef ATM_SEN_MODULE_H
#define ATM_SEN_MODULE_H

#include <stdio.h>
#include <pico/stdlib.h>
#include <stdint.h>

// DATA STRUCTURES


typedef struct
{
    double pressure_pa;
    double altitude_m;
    float temperature_c;
    float humidity_pct;
    float methane_ppm;
    float ammonia_ppm;
    float oxygen_pct;
} sensor_readings_t;

// FUNCTIONS


extern void init_all_sensors(void);

extern void read_all(sensor_readings_t *gathered_data);

#endif // ATM_SEN_MODULE_H
```

6.4 src/debug_mode.h File Reference

Conditional debug logging and error handling macros.

```
#include <stdio.h>
```

Macros

- #define DEBUG_MODE
- #define LOG(...)

6.4.1 Detailed Description

Conditional debug logging and error handling macros.

Provides the `LOG` macro which directs output to the standard I/O (USB/UART) only when `DEBUG_MODE` is defined. This allows for logging during development that can be completely compiled out for the final flight release.

6.4.2 Macro Definition Documentation

6.4.2.1 DEBUG_MODE

```
#define DEBUG_MODE
```

6.4.2.2 LOG

```
#define LOG(
                ...)
```

Value:

```
printf(__VA_ARGS__)
```

6.5 debug_mode.h

Go to the documentation of this file.

```

#ifndef DEBUG_MODE_H
#define DEBUG_MODE_H

#include <stdio.h>

#define DEBUG_MODE

#ifdef DEBUG_MODE
    #define LOG(...) printf(__VA_ARGS__)
#else
    #define LOG(...) ((void)0)
#endif

#endif // DEBUG_MODE_H
```

6.6 src/gps_module.c File Reference

Implementation of GPS NMEA parsing and UART handling.

```
#include <string.h>
#include "gps_module.h"
#include "time_manager.h"
#include "debug_mode.h"
#include "minmea.h"
#include "hardware/uart.h"
```

Macros

- #define NMEA_BUFFER_LEN 85

Functions

- void gps_init (void)

  Initializes the GPS UART connection and GPIO pins.
- bool gps_update (void)

  Polls the UART for newly aquired GPS data and sends it to be parsed.
- void gps_get_data (gps_data_t ∗data)

  Retrieves the latest parsed GPS data.

6.6.1 Detailed Description

Implementation of GPS NMEA parsing and UART handling.

See also

gps_module.h for the public API and data structures.

6.6.2 Macro Definition Documentation

6.6.2.1 NMEA_BUFFER_LEN

```
#define NMEA_BUFFER_LEN 85
```

6.6.3 Function Documentation

6.6.3.1 gps_get_data()

```
void gps_get_data (
            gps_data_t * data)
```

Retrieves the latest parsed GPS data.

Copies the most recent valid GPS data into the provided 'gps_data_t' structure.

Parameters

out	*data*	Pointer to a 'gps_data_t' structure where the data will be copied.

6.6.3.2 gps_init()

```
void gps_init (
            void )
```

Initializes the GPS UART connection and GPIO pins.

Sets up the specified GPS_UART_ID with the baud rate defined in GPS_BAUD_RATE and configures the TX/RX pins.

Note

This must be called once at system startup before the main loop.

6.6.3.3 gps_update()

```
bool gps_update (
            void )
```

Polls the UART for newly aquired GPS data and sends it to be parsed.

This function should be called frequently (e.g., in the main loop). It reads available characters from the UART buffer and calls a function to process NMEA sentences.

Returns

true if a valid packet was fully parsed and data was updated.

false if no new complete packet is available yet.

6.7 src/gps_module.h File Reference

GPS module interface.

```
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include "pico/stdlib.h"
```

Classes

- struct gps_data_t

  The structure keeps the parsed GPS position and time information.

Macros

- #define GPS_BAUD_RATE 9600

  UART Baud rate for the GPS module (Standard is usually 9600).
- #define GPS_UART_ID uart0

  The hardware UART instance to use (uart0 or uart1).
- #define GPS_TX_PIN 8

  GPIO pin for UART transmission (Pico TX).
- #define GPS_RX_PIN 9

  GPIO pin for UART reception (Pico RX).

Functions

- void gps_init (void)

  Initializes the GPS UART connection and GPIO pins.
- bool gps_update (void)

  Polls the UART for newly aquired GPS data and sends it to be parsed.
- void gps_get_data (gps_data_t ∗data)

  Retrieves the latest parsed GPS data.

6.7.1 Detailed Description

GPS module interface.

This file handles the UART configuration for Pico for the GPS module, defines the data structure for holding parsed GPS coordinates/time, and provides functions to initialize and update the system.

See also

`https://github.com/kosma/minmea` for the NMEA parsing library documentation.

6.7.2 Macro Definition Documentation

6.7.2.1 GPS_BAUD_RATE

```
#define GPS_BAUD_RATE 9600
```

UART Baud rate for the GPS module (Standard is usually 9600).

6.7.2.2 GPS_RX_PIN

```
#define GPS_RX_PIN 9
```

GPIO pin for UART reception (Pico RX).

6.7.2.3 GPS_TX_PIN

```
#define GPS_TX_PIN 8
```

GPIO pin for UART transmission (Pico TX).

6.7.2.4 GPS_UART_ID

```
#define GPS_UART_ID uart0
```

The hardware UART instance to use (uart0 or uart1).

6.7.3 Function Documentation

6.7.3.1 gps_get_data()

```
void gps_get_data (
            gps_data_t * data)  [extern]
```

Retrieves the latest parsed GPS data.

Copies the most recent valid GPS data into the provided 'gps_data_t' structure.

Parameters

out	*data*	Pointer to a 'gps_data_t' structure where the data will be copied.

6.7.3.2 gps_init()

```
void gps_init (
            void )  [extern]
```

Initializes the GPS UART connection and GPIO pins.

Sets up the specified GPS_UART_ID with the baud rate defined in GPS_BAUD_RATE and configures the TX/RX pins.

Note

This must be called once at system startup before the main loop.

6.7.3.3 gps_update()

```
bool gps_update (
            void )  [extern]
```

Polls the UART for newly aquired GPS data and sends it to be parsed.

This function should be called frequently (e.g., in the main loop). It reads available characters from the UART buffer and calls a function to process NMEA sentences.

Returns

true if a valid packet was fully parsed and data was updated.

false if no new complete packet is available yet.

6.8 gps_module.h

Go to the documentation of this file.

```
#ifndef GPS_MODULE_H
#define GPS_MODULE_H

#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include "pico/stdlib.h"

// CONFIGURATION MACROS

#define GPS_BAUD_RATE 9600

#define GPS_UART_ID uart0

#define GPS_TX_PIN 8

#define GPS_RX_PIN 9

// DATA STRUCTURES

typedef struct
{
    float latitude;
    float longitude;
    float altitude;
    uint8_t satellites;
    uint8_t hour;
    uint8_t min;
    uint8_t sec;
    uint16_t year;
    uint8_t month;
    uint8_t day;
    bool fix;
} gps_data_t;

// FUNCTIONS


extern void gps_init(void);

extern bool gps_update(void);

extern void gps_get_data(gps_data_t *data);

#endif // GPS_MODULE_H
```

6.9 src/hw_config.c File Reference

```
#include "ff.h"
#include "diskio.h"
#include "sd_card.h"
#include "hardware/spi.h"
#include <strings.h>
```

Functions

- size_t sd_get_num ()
- sd_card_t ∗ sd_get_by_num (size_t num)
- size_t spi_get_num ()
- spi_t ∗ spi_get_by_num (size_t num)

6.9.1 Function Documentation

6.9.1.1 sd_get_by_num()

```
sd_card_t * sd_get_by_num (
             size_t num)
```

6.9.1.2 sd_get_num()

```
size_t sd_get_num ()
```

6.9.1.3 spi_get_by_num()

```
spi_t * spi_get_by_num (
             size_t num)
```

6.9.1.4 spi_get_num()

```
size_t spi_get_num ()
```

6.10 src/main.c File Reference

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "debug_mode.h"
#include "atm_sen_module.h"
#include "microsd_module.h"
#include "time_manager.h"
#include "gps_module.h"
#include "radio_module.h"
```

Functions

- int main (void)

6.10.1 Function Documentation

6.10.1.1 main()

```
int main (
             void )
```

6.11 src/microsd_module.c File Reference

```
#include "microsd_module.h"
#include "debug_mode.h"
#include "hardware/spi.h"
#include "ff.h"
#include "sd_card.h"
#include <stdio.h>
```

Functions

- sd_card_t ∗ sd_get_by_num (size_t num)
- bool sd_init ()

  Initializes the Micro SD reader SPI connection and checks for successful mounting of the card.
- void save_system_data (sensor_readings_t ∗data, current_time_t ∗time)

  Saves all data collected from sensors onto the microSD card in a 'data_log.txt' file.
- void save_gps_log (gps_data_t ∗gps)

  Saves all GPS data onto the microSD card in a 'gps_log.csv' file.

6.11.1 Function Documentation

6.11.1.1 save_gps_log()

```
void save_gps_log (
            gps_data_t * gps)
```

Saves all GPS data onto the microSD card in a 'gps_log.csv' file.

It uses the functions available in the beforementioned library to save formatted strings with values read from the GPS kept in a 'gps_data_t' structure onto the microSD.

Parameters

in	*gps_data_t*	Pointer to a structure keeping data read from GPS and properly parsed.

6.11.1.2 save_system_data()

```
void save_system_data (
            sensor_readings_t * data,
            current_time_t * time)
```

Saves all data collected from sensors onto the microSD card in a 'data_log.txt' file.

It uses the functions available in the beforementioned library to save formatted strings with values read from sensors kept in a 'sensor_readings_t' file onto the microSD.

Parameters

in	*data*	Pointer to a 'sensor_readings_t' data struct which stores the data to to be saved on the microSD card.
in	*time*	Pointer to a time-keeping structure that plays a role in time management for the timestamps in the 'data_log.txt' file.

6.11.1.3 sd_get_by_num()

```
sd_card_t * sd_get_by_num (
            size_t num)  [extern]
```

6.11.1.4 sd_init()

```
bool sd_init ()
```

Initializes the Micro SD reader SPI connection and checks for successful mounting of the card.

Returns

true if the reader is correctly initialized and the microSD card is mounted.

false if the reader setup or the microSD card mount fails.

6.12 src/microsd_module.h File Reference

Micro SD Card reader interface.

```
#include <stdbool.h>
#include <stdio.h>
#include <stdint.h>
#include "pico/stdlib.h"
#include "atm_sen_module.h"
#include "time_manager.h"
#include "gps_module.h"
```

Functions

- bool sd_init ()

  Initializes the Micro SD reader SPI connection and checks for successful mounting of the card.
- void save_system_data (sensor_readings_t ∗data, current_time_t ∗time)

  Saves all data collected from sensors onto the microSD card in a 'data_log.txt' file.
- void save_gps_log (gps_data_t ∗gps)

  Saves all GPS data onto the microSD card in a 'gps_log.csv' file.

6.12.1 Detailed Description

Micro SD Card reader interface.

This file handles the SD card reader SPI configuration for Pico using the no-OS-FatFS-SD-SPI-RPi-Pico library, implements a function for saving data collected from other sensors, and saving GPS coordinates processed by the GPS module.

See also

`https://github.com/carlk3/no-OS-FatFS-SD-SPI-RPi-Pico/tree/master` for the SD driver documentation.

6.12.2 Function Documentation

6.12.2.1 save_gps_log()

```
void save_gps_log (
            gps_data_t * gps)  [extern]
```

Saves all GPS data onto the microSD card in a 'gps_log.csv' file.

It uses the functions available in the beforementioned library to save formatted strings with values read from the GPS kept in a 'gps_data_t' structure onto the microSD.

Parameters

in	*gps_data_t*	Pointer to a structure keeping data read from GPS and properly parsed.

6.12.2.2 save_system_data()

```
void save_system_data (
            sensor_readings_t * data,
            current_time_t * time)  [extern]
```

Saves all data collected from sensors onto the microSD card in a 'data_log.txt' file.

It uses the functions available in the beforementioned library to save formatted strings with values read from sensors kept in a 'sensor_readings_t' file onto the microSD.

Parameters

in	*data*	Pointer to a 'sensor_readings_t' data struct which stores the data to to be saved on the microSD card.
in	*time*	Pointer to a time-keeping structure that plays a role in time management for the timestamps in the 'data_log.txt' file.

6.12.2.3 sd_init()

```
bool sd_init ()  [extern]
```

Initializes the Micro SD reader SPI connection and checks for successful mounting of the card.

Returns

true if the reader is correctly initialized and the microSD card is mounted.

false if the reader setup or the microSD card mount fails.

6.13 microsd_module.h

Go to the documentation of this file.

```


#ifndef MICROSD_MODULE_H
#define MICROSD_MODULE_H

#include <stdbool.h>
#include <stdio.h>
#include <stdint.h>
#include "pico/stdlib.h"
#include "atm_sen_module.h"
#include "time_manager.h"
#include "gps_module.h"

// FUNCTIONS


extern bool sd_init();

extern void save_system_data(sensor_readings_t *data, current_time_t *time);

extern void save_gps_log(gps_data_t *gps);

#endif // MICROSD_MODULE_H
```

6.14 src/radio_module.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include "debug_mode.h"
#include "radio_module.h"
#include "lib/nrf905/nrf905.h"
```

Functions

- void radio_module_init (void)

  Initializes the radio hardware and driver.
- void radio_module_send_telemetry (float temp, float press, float alt)

  Formats sensor data and broadcasts it via radio.

6.14.1 Function Documentation

6.14.1.1 radio_module_init()

```
void radio_module_init (
            void )
```

Initializes the radio hardware and driver.

This function calls the underlying driver initialization routine. It sets up the SPI interface, configures GPIO pins for radio control (TX_EN, TRX_CE, PWR), and writes the default configuration registers (Frequency, Power, CRC).

Note

Must be called once at system startup before attempting any transmissions.

6.14.1.2 radio_module_send_telemetry()

```
void radio_module_send_telemetry (
            float temp,
            float press,
            float alt)
```

Formats sensor data and broadcasts it via radio.

This function takes individual sensor readings, formats them into a standard ASCII telemetry string (e.g., "T:24.5 P:1001 A:150"), and broadcasts the packet via the nRF905 radio.

Parameters

temp	Temperature (Celsius)
press	Pressure (Pascals)
alt	Altitude (Meters)

6.15 src/radio_module.h File Reference

High-level telemetry interface for nRF905 radio.

```
#include "pico/stdlib.h"
```

Functions

- void radio_module_init (void)

  Initializes the radio hardware and driver.
- void radio_module_send_telemetry (float temp, float press, float alt)

  Formats sensor data and broadcasts it via radio.

6.15.1 Detailed Description

High-level telemetry interface for nRF905 radio.

Abstracts low-level SPI drivers to provide a simple API for sending sensor data. Handles the formatting of float values into ASCII telemetry strings (e.g., "T:25.0 P:1013") and manages packet transmission.

6.15.2 Function Documentation

6.15.2.1 radio_module_init()

```
void radio_module_init (
            void )  [extern]
```

Initializes the radio hardware and driver.

This function calls the underlying driver initialization routine. It sets up the SPI interface, configures GPIO pins for radio control (TX_EN, TRX_CE, PWR), and writes the default configuration registers (Frequency, Power, CRC).

Note

Must be called once at system startup before attempting any transmissions.

6.15.2.2 radio_module_send_telemetry()

```
void radio_module_send_telemetry (
            float temp,
            float press,
            float alt)  [extern]
```

Formats sensor data and broadcasts it via radio.

This function takes individual sensor readings, formats them into a standard ASCII telemetry string (e.g., "T:24.5 P:1001 A:150"), and broadcasts the packet via the nRF905 radio.

Parameters

temp	Temperature (Celsius)
press	Pressure (Pascals)
alt	Altitude (Meters)

6.16 radio_module.h

Go to the documentation of this file.

```


#ifndef RADIO_MODULE_H
#define RADIO_MODULE_H

#include "pico/stdlib.h"

extern void radio_module_init(void);

extern void radio_module_send_telemetry(float temp, float press, float alt);

#endif
```

6.17 src/time_manager.c File Reference

```
#include "time_manager.h"
#include "pico/stdlib.h"
#include "ff.h"
#include "microsd_module.h"
#include <stdio.h>
```

Macros

- #define TIMEZONE_OFFSET 1

Functions

- void time_manager_init (void)

  Initializes the Software Real-Time Clock (RTC).
- bool time_manager_update (void)

  Ticks the internal clock forward.
- current_time_t ∗ time_manager_get (void)

  Retrieves the current system time in a human-readable format.
- void time_manager_sync (uint16_t year, uint8_t month, uint8_t day, uint8_t hour, uint8_t min, uint8_t sec)

  Synchronizes the internal clock with an external source (e.g., GPS). Overwrites the internal time counter with new values provided by the GPS.
- DWORD get_fattime (void)

6.17.1 Macro Definition Documentation

6.17.1.1 TIMEZONE_OFFSET

```
#define TIMEZONE_OFFSET 1
```

6.17.2 Function Documentation

6.17.2.1 get_fattime()

```
DWORD get_fattime (
            void )
```

6.17.2.2 time_manager_get()

```
current_time_t * time_manager_get (
            void )
```

Retrieves the current system time in a human-readable format.

Converts the internal epoch timestamp (`raw_seconds`) into a `current_time_t` structure containing year, month, day, hour, minute, and second. If the time is exactly 00:00:00 (Midnight), this function automatically resets the `photo_count` field to 0.

Returns

- current_time_t∗ Pointer to the structure holding the current time.

6.17.2.3 time_manager_init()

```
void time_manager_init (
            void )
```

Initializes the Software Real-Time Clock (RTC).

Sets the internal system time to a default start date (January 1, 2026). It captures the current processor boot time (`to_ms_since_boot`) to establish a baseline for the 1-second tick counter.

Note

- This must be called once at system startup before the main loop.

6.17.2.4 time_manager_sync()

```
void time_manager_sync (
            uint16_t year,
            uint8_t month,
            uint8_t day,
            uint8_t hour,
            uint8_t min,
            uint8_t sec)
```

Synchronizes the internal clock with an external source (e.g., GPS). Overwrites the internal time counter with new values provided by the GPS.

- Automatically applies the defined `TIMEZONE_OFFSET` to the provided UTC hour.
- Uses `mktime()` to safe-handle hour overflows. For example, if the GPS says 23:00 UTC and your offset is +2, the date automatically moves on to the next day.
- Resets the millisecond counter (`last_second_ms`) to zero, ensuring the next software "tick" happens exactly 1 second after this sync occurs.
-

Parameters

in	*year*	Full year (e.g., 2026).
in	*month*	Month number (1-12).
in	*day*	Day of the month (1-31).
in	*hour*	UTC Hour (0-23).
in	*min*	Minute (0-59).
in	*sec*	Second (0-59).

6.17.2.5 time_manager_update()

```
bool time_manager_update (
            void )
```

Ticks the internal clock forward.

This function checks the system's millisecond timer. If 1000ms have passed since the last tick, it increments the internal `raw_seconds` counter and updates the reference timer. This function is non-blocking and is designed to be called frequently inside the main `while(1)` loop.

Returns

- true If a full second has passed during this call (useful for triggering 1Hz events like LED blinks).
  false If less than 1 second has passed since the last update.

6.18 src/time_manager.h File Reference

Software RTC with GPS synchronization and FatFS integration.

```
#include <stdint.h>
#include <stdbool.h>
#include <time.h>
```

Classes

- struct current_time_t

  The structure stores data about the date, hour, minutes and seconds a reading was made and saved.

Functions

- void time_manager_init (void)

  Initializes the Software Real-Time Clock (RTC).
- bool time_manager_update (void)

  Ticks the internal clock forward.
- current_time_t ∗ time_manager_get (void)

  Retrieves the current system time in a human-readable format.
- void time_manager_sync (uint16_t year, uint8_t month, uint8_t day, uint8_t hour, uint8_t min, uint8_t sec)

  Synchronizes the internal clock with an external source (e.g., GPS). Overwrites the internal time counter with new values provided by the GPS.

6.18.1 Detailed Description

Software RTC with GPS synchronization and FatFS integration.

Maintains system time using system ticks (`to_ms_since_boot`), independent of hardware RTC. It features:

- a sync with GPS: corrects drift and sets time/date via NMEA data.
- a FatFS backend: provides timestamps for SD card files.
- timezones: applies static offsets defined by `TIMEZONE_OFFSET`.
- Usage: Calling `time_manager_init()` at startup, `time_manager_update()` periodically in the loop, and `time_manager_sync()` when valid GPS data exists.

  Note
- Replaces the default `rtc.c` to remove hardware dependencies that were trublesome.

6.18.2 Function Documentation

6.18.2.1 time_manager_get()

```
current_time_t * time_manager_get (
             void )  [extern]
```

Retrieves the current system time in a human-readable format.

Converts the internal epoch timestamp (`raw_seconds`) into a `current_time_t` structure containing year, month, day, hour, minute, and second. If the time is exactly 00:00:00 (Midnight), this function automatically resets the `photo_count` field to 0.

Returns

- current_time_t* Pointer to the structure holding the current time.

6.18.2.2 time_manager_init()

```
void time_manager_init (
             void )  [extern]
```

Initializes the Software Real-Time Clock (RTC).

Sets the internal system time to a default start date (January 1, 2026). It captures the current processor boot time (`to_ms_since_boot`) to establish a baseline for the 1-second tick counter.

Note

- This must be called once at system startup before the main loop.

6.18.2.3 time_manager_sync()

```
void time_manager_sync (
             uint16_t year,
             uint8_t month,
             uint8_t day,
             uint8_t hour,
             uint8_t min,
             uint8_t sec)  [extern]
```

Synchronizes the internal clock with an external source (e.g., GPS). Overwrites the internal time counter with new values provided by the GPS.

- Automatically applies the defined `TIMEZONE_OFFSET` to the provided UTC hour.
- Uses `mktime()` to safe-handle hour overflows. For example, if the GPS says 23:00 UTC and your offset is +2, the date automatically moves on to the next day.
- Resets the millisecond counter (`last_second_ms`) to zero, ensuring the next software "tick" happens exactly 1 second after this sync occurs.

- **Parameters**

in	*year*	Full year (e.g., 2026).
in	*month*	Month number (1-12).
in	*day*	Day of the month (1-31).
in	*hour*	UTC Hour (0-23).
in	*min*	Minute (0-59).
in	*sec*	Second (0-59).

6.18.2.4 time_manager_update()

```
bool time_manager_update (
              void )  [extern]
```

Ticks the internal clock forward.

This function checks the system's millisecond timer. If 1000ms have passed since the last tick, it increments the internal `raw_seconds` counter and updates the reference timer. This function is non-blocking and is designed to be called frequently inside the main `while(1)` loop.

Returns

- true If a full second has passed during this call (useful for triggering 1Hz events like LED blinks).
  false If less than 1 second has passed since the last update.

6.19 time_manager.h

Go to the documentation of this file.

```

#ifndef TIME_MANAGER_H
#define TIME_MANAGER_H

#include <stdint.h>
#include <stdbool.h>
#include <time.h>

// DATA STRUCTURES

typedef struct
{
    uint16_t year;
    uint8_t month;
    uint8_t day;
    uint8_t hour;
    uint8_t min;
    uint8_t sec;
    uint16_t photo_count;
} current_time_t;

// FUNCTIONS

extern void time_manager_init(void);

extern bool time_manager_update(void);

extern current_time_t* time_manager_get(void);

extern void time_manager_sync(uint16_t year, uint8_t month, uint8_t day, uint8_t hour, uint8_t min,
uint8_t sec);

#endif
```