WayFindX

Version 1.0

Generated by Doxygen 1.10.0

Chapter 1

WayFindX

Synopsis of the WayFindX Embedded Systems Project.

Synopsis of the WayFindX Embedded Systems Project.

WayFindX is a handheld GPS receiver project developed by Avengers Assembly, consisting of Bradley Johnson and Abele Atresso. The project aims to create a portable navigation device that enhances outdoor navigation capabilities for users such as hikers, adventurers, and outdoor enthusiasts. WayFindX provides essential features including position and velocity tracking, storage of multiple locations, and distance calculation to selected stored positions.

1.1 Key Features

- Display of user position and velocity information
- Storage of multiple locations for navigation
- Calculation of distance to selected stored positions
- Intuitive user interface with control buttons and LCD display

1.2 Hardware Components

The project utilizes the following hardware components:

- Arduino microcontroller (ATMega328p)
- HD44780 Controlled 20x4 LCD for display
- Control Stick for user input
- NEO-6M GPS module for positioning
- Power Switch and 9V battery for power management
- Additional components for connectivity
- Custom-designed 3D printed case for enclosure

1.3 Software Interface

WayFindX provides a user-friendly interface with control buttons for navigation and interaction. The software is designed to display essential information such as current position, stored locations, and operational modes on the LCD display. Users can cycle through saved positions, select operations, and execute actions with ease.

1.4 Challenges Faced

The project encountered various challenges including serial communication, interfacing with new ICs, soldering, and enclosure design. Overcoming these challenges required careful planning, experimentation, and troubleshooting.

For detailed documentation and code implementation, refer to the project source code and comments.\

Version

1.0

Copyright

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

File Documentation

3.1 G:/Project_Files/WayFindX/wfx_sw/wfx_sw/ds/ds.c File Reference

```
#include "ds.h"
#include "../lib/lcd.h"
#include "../ut/ut_types.h"
#include <avr/io.h>
```

Functions

- void ds_print_string (char ∗inputString, int size, uint8_t row)

  Prints a string on the LCD display.
- void ds_init ()

  Initializes the LCD display. This function initializes the LCD display and turns on the display.
- void ds_clear ()

  Clears the LCD display. This function clears the LCD display.

3.1.1 Function Documentation

3.1.1.1 ds_clear()

```
void ds_clear ( )
```

Clears the LCD display. This function clears the LCD display.

Clears the display.

Definition at line 44 of file ds.c.

3.1.1.2 ds_init()

```
void ds_init ( )
```

Initializes the LCD display. This function initializes the LCD display and turns on the display.

Uses library to initialize display. If DEBUG is defined, cursor will blink. Otherwise, cursor will be off.

Definition at line 33 of file ds.c.

3.1.1.3 ds_print_string()

```
void ds_print_string (
            char * inputString,
            int size,
            uint8_t row )
```

Prints a string on the LCD display.

Parameters

inputString	Pointer to the string to be printed.
size	Size of the string to be printed.
row	indicating what row to place the cursor

Definition at line 14 of file ds.c.

3.2 ds.c

Go to the documentation of this file.

```
00001 #include "ds.h"
00002 #include "../lib/lcd.h"
00003 #include "../ut/ut_types.h"
00004 #include <avr/io.h>
00005
00006
00014 void ds_print_string(char * inputString, int size, uint8_t row)
00015 {
00016     //Make sure size is legit
00017    if ((size >= 0) && (size <= MAX_COL) && (row >= 0) && (row < MAX_ROWS))
00018     {
00019         lcd_gotoxy((uint8_t)0, row); //send to second line
00020
00021         for (int i = 0; i < size; i++)
00022         {
00023             lcd_putc(inputString[i]); //put i'th char on display
00024         }
00025     }
00026     return;
00027 }
00028
00033 void ds_init()
00034 {
00035     lcd_init(LCD_DISP_ON);
00036     lcd_clrscr();
00037     return;
00038 }
00039
00044 void ds_clear(){
00045     lcd_clrscr();
00046 }
```

3.3 G:/Project_Files/WayFindX/wfx_sw/wfx_sw/ds/ds.h File Reference

Header file containing common functions and definitions for the display 'computer software component" or CSC.

```
#include "../ut/ut_types.h"
```

Macros

- #define MAX_ROWS 4
- #define MAX_COL 20
- #define SPACES " "

Functions

- void ds_print_string (char ∗inputString, int size, uint8_t row)

  Prints a string on the LCD display.
- void ds_init ()

  Uses library to initialize display. If DEBUG is defined, cursor will blink. Otherwise, cursor will be off.
- void ds_clear ()

  Clears the display.

3.3.1 Detailed Description

Header file containing common functions and definitions for the display 'computer software component" or CSC.

This file provides declarations for common utility functions and definitions associated with the LCD display.

Version

1.0

Copyright

(C) 2024 Bradley Johnson and Abele Atresso

Definition in file ds.h.

3.3.2 Macro Definition Documentation

3.3.2.1 MAX_COL

```
#define MAX_COL 20
```

Maximum number of columns on the LCD display

Definition at line 14 of file ds.h.

3.3.2.2 MAX_ROWS

```
#define MAX_ROWS 4
```

Maximum number of rows on the LCD display

Definition at line 13 of file ds.h.

3.3.2.3 SPACES

```
#define SPACES " "
```

String of spaces used for clearing the LCD display

Definition at line 15 of file ds.h.

3.3.3 Function Documentation

3.3.3.1 ds_clear()

```
void ds_clear ( )
```

Clears the display.

Clears the display.

Definition at line 44 of file ds.c.

3.3.3.2 ds_init()

```
void ds_init ( )
```

Uses library to initialize display. If DEBUG is defined, cursor will blink. Otherwise, cursor will be off.

Returns

none

Uses library to initialize display. If DEBUG is defined, cursor will blink. Otherwise, cursor will be off.

Definition at line 33 of file ds.c.

3.3.3.3 ds_print_string()

```
void ds_print_string (
             char * inputString,
             int size,
             uint8_t row )
```

Prints a string on the LCD display.

Parameters

inputString	Pointer to the string to be printed.
size	Size of the string to be printed.
row	Row index where the cursor will be placed.
inputString	Pointer to the string to be printed.
size	Size of the string to be printed.
row	indicating what row to place the cursor

Definition at line 14 of file ds.c.

3.4 ds.h

Go to the documentation of this file.

```

#ifndef DS_H_
#define DS_H_

#define MAX_ROWS 4
#define MAX_COL 20
#define SPACES "                    "
#include "../ut/ut_types.h"

void ds_print_string(char * inputString, int size, uint8_t row);

void ds_init();

void ds_clear();


#endif /* DS_H_ */
```

3.5 G:/Project_Files/WayFindX/wfx_sw/wfx_sw/ir/ir.c File Reference

```
#include "ir.h"
#include <avr/interrupt.h>
#include "../ut/utilities.h"
```

Functions

- ISR (TIMER0_OVF_vect)
- ISR (TIMER2_OVF_vect)
- void ir_init ()

  Initializes interrupt system functionality.

Variables

- uint16_t ir_sec_counter = 0
- boolean_t ir_trigger_1hz_flag_g = false

3.5.1 Function Documentation

3.5.1.1 ir_init()

```
void ir_init ( )
```

Initializes interrupt system functionality.

This function configures Timer2 for time management and Timer 0 for background button polling.

Definition at line 38 of file ir.c.

3.5.1.2 ISR() [1/2]

```
ISR (
            TIMER0_OVF_vect  )
```

Definition at line 16 of file ir.c.

3.5.1.3 ISR() [2/2]

```
ISR (
            TIMER2_OVF_vect  )
```

Definition at line 22 of file ir.c.

3.5.2 Variable Documentation

3.5.2.1 ir_sec_counter

```
uint16_t ir_sec_counter = 0
```

Seconds counter for indicating TTFF

Definition at line 6 of file ir.c.

3.5.2.2 ir_trigger_1hz_flag_g

```
boolean_t ir_trigger_1hz_flag_g = false
```

Global flag indicating 1Hz trigger

Definition at line 7 of file ir.c.

3.6 ir.c

Go to the documentation of this file.

```
00001 #include "ir.h"
00002 #include <avr/interrupt.h>
00003 #include "../ut/utilities.h"
00004
00005 //global
00006 uint16_t ir_sec_counter = 0;
00007 boolean_t ir_trigger_1hz_flag_g = false;
00011 //local static
00012 static uint8_t timer2_overflow_counter;
00013
00014
00015 // Interrupt Service Routine for Timer0 overflow: Poll all four buttons in background set state if we
      have polled enough times
00016 ISR(TIMER0_OVF_vect) {
00017     ut_poll_btns();
00018     TCNT0 = 0;
00019 }
00020
00021 // Interrupt Service Routine for Timer2 overflow: Used to trip trigger flag at 1Hz
00022 ISR(TIMER2_OVF_vect) {
00023     timer2_overflow_counter++;
00024     if (timer2_overflow_counter >= 15){
        timer2_overflow_counter = 0;
        ir_sec_counter++;
        if (!ir_trigger_1hz_flag_g){
            ir_trigger_1hz_flag_g = true;
        }
    }
}

void ir_init()
{
    cli();
    //---------------------------------------------
    //timer 0 for background button polling
    OCR0A = 2;        // Set TOP (maximum value for counter)

    TCNT0 = 0x00;       //set counter to zero
    TCCR0A = 0x00;      //set TCCR1A to zero before configuring timer0
    TCCR0B = 0x00;
    // TOP is defined as OCR0A when WGM2:0 = 5

    // Set pre-scaler to /8
    TCCR0B |= (1 « CS01);
    // Enable Timer0 Overflow Interrupt
    TIMSK0 |= (1 « TOIE0);
    // Initialize Timer0 count
    TCNT0 = 0;


    //Set up timer2 for 1 hz task managment
    // Set Timer2 in normal mode (WGM22:0 = 0)
    TCCR2A = 0x00;
    TCCR2B = 0x00;
    // Set pre-scaler to clk/1024 for smoothing
    TCCR2B |= (1 « CS22) | (1 « CS21) | (1 « CS20); // Pre-scaler = 64
    // Enable Timer2 Overflow Interrupt
    TIMSK2 |= (1 « TOIE2);


    sei(); // Enable interrupts.
}
```

3.7 G:/Project_Files/WayFindX/wfx_sw/wfx_sw/ir/ir.h File Reference

Header file containing common functions and definitions for the interrupt system 'computer software component" or CSC.

```
#include "../ut/ut_types.h"
```

Functions

- void ir_init ()

    Initializes interrupt system functionality.

Variables

- boolean_t ir_trigger_1hz_flag_g
- uint16_t ir_sec_counter

3.7.1 Detailed Description

Header file containing common functions and definitions for the interrupt system 'computer software component'' or CSC.

This file provides declarations for common utility functions and definitions associated with the interrupt system.

Version

1.0

Copyright

(C) 2024 Bradley Johnson and Abele Atresso

Definition in file ir.h.

3.7.2 Function Documentation

3.7.2.1 ir_init()

```
void ir_init ( )
```

Initializes interrupt system functionality.

This function configures Timer2 for time management and Timer 0 for background button polling.

Definition at line 38 of file ir.c.

3.7.3 Variable Documentation

3.7.3.1 ir_sec_counter

```
uint16_t ir_sec_counter  [extern]
```

Seconds counter for indicating TTFF

Definition at line 6 of file ir.c.

3.7.3.2 ir_trigger_1hz_flag_g

```
boolean_t ir_trigger_1hz_flag_g  [extern]
```

Global flag indicating 1Hz trigger

Definition at line 7 of file ir.c.

3.8 ir.h

Go to the documentation of this file.

```
#ifndef IR_H_
#define IR_H_

#include "../ut/ut_types.h"

//globals
extern boolean_t ir_trigger_1hz_flag_g;
extern uint16_t ir_sec_counter;
void ir_init();


#endif /* IR_H_ */
```

3.9 G:/Project_Files/WayFindX/wfx_sw/wfx_sw/main.c File Reference

```
#include <avr/power.h>
#include <avr/io.h>
#include <util/delay.h>
#include "ds/ds.h"
#include "ir/ir.h"
#include "nf/nf.h"
#include "nf/nf_types.h"
#include "ut/utilities.h"
#include "ut/ut_types.h"
#include <string.h>
```

Macros

- #define F_CPU 4000000UL

Functions

- void startup ()

  Performs startup initialization.
- void task_1hz ()

  Executes tasks that should occur every 1Hz.
- void update_display ()

  Updates the display with relevant information.
- int main (void)

  Main loop function.

3.9.1 Macro Definition Documentation

3.9.1.1 F_CPU

```
#define F_CPU 4000000UL
```

Define the CPU frequency to 4MHz.

Definition at line 46 of file main.c.

3.9.2 Function Documentation

3.9.2.1 main()

```
int main (
             void  )
```

Main loop function.

This function controls the program flow and will not return. It initializes the peripherals, clears the display, and continuously updates the system behavior.

Definition at line 73 of file main.c.

3.9.2.2 startup()

```
void startup ( )
```

Performs startup initialization.

< Include display-related functions. < Include interrupt routines. < Include navigation fetch functions < Include utility functions. < Include common type definitions.

This function initializes various peripherals and components during startup, such as the display, interrupt routines, navigation fetch, and utilities. < Initialize display CSC.

< Initialize interrupt routines.

<Initialize navigation fetch CSC.

< Initialize utilities CSC.

Definition at line 93 of file main.c.

3.9.2.3 task_1hz()

```
void task_1hz ( )
```

Executes tasks that should occur every 1Hz.

This function is called once per second and performs tasks such as updating the display.

Definition at line 130 of file main.c.

3.9.2.4 update_display()

```
void update_display ( )
```

Updates the display with relevant information.

This function updates the display with information such as GPS coordinates, mode, operation, and memory status.

Definition at line 147 of file main.c.

3.10 main.c

Go to the documentation of this file.

```c

#ifndef F_CPU
#define F_CPU 4000000UL
#endif

//PORT Pin 2 PD2
#include <avr/power.h>
#include <avr/io.h>
#include <util/delay.h>

#include "ds/ds.h"
#include "ir/ir.h"
#include "nf/nf.h"
#include "nf/nf_types.h"
#include "ut/utilities.h"
#include "ut/ut_types.h"
#include <string.h>

void startup();
void task_1hz();
void update_display();


int main(void)
{
    startup();
    /* Main loop */
    while (1){
        //read_nmea_msg
        read_nmea_msg_raw();
        if (ir_trigger_1hz_flag_g == true){
            task_1hz();
            ir_trigger_1hz_flag_g = false;
        }
    }
}


void startup(){
    //Initialize
    // Set clock pre-scaler to divide by 4
    clock_prescale_set(clock_div_4);

    // Initialize computer software components (CSC's)
    ds_init();
    {
        char* welcome = "- - - - ~~~~ - - - -";
        ds_print_string(welcome, MAX_COL, 0);
    }
    { //Welcome Screen - limited scope
        char* welcome = "- - - WayFindX - - -";
        ds_print_string(welcome, MAX_COL, 1);
    }
    {
        char* welcome = "- - - - ~~~~ - - - -";
        ds_print_string(welcome, MAX_COL, 2);
    }
    _delay_ms(0.1f);

    ir_init();
    _delay_ms(0.1f);
    if (nf_init()){
        char* err = "  Nav init failure  ";
        ds_print_string(err, MAX_COL, 1);
        while(1){};
    }
    _delay_ms(0.6f);
    ut_init();
}

void task_1hz(){
    update_display();
    //Condition where USART if out of sync with NEO6-M
    if ((position_fix_indicator[0] == '1') && (speed[0] == ' ') && (speed[1] == ' ')){
        /* Re-Initialize navigation fetch CSC. */
        do {
            char* err = "Nav module off sync!";
            ds_print_string(err, MAX_COL, 1);
        } while(nf_init());
    }
}

```

```
void update_display(){
    char line0[MAX_COL] = SPACES;
    char line1[MAX_COL] = SPACES;
    char line2[MAX_COL] = SPACES;
    char line3[MAX_COL] = SPACES;

    if (utc_time[0] == ' '){ //Until we solve for time
        char* temp = "Acquiring Satellites";
        for (int i = 0; i < MAX_COL; i++){
            line3[i] = temp[i];
        }
    } else if (position_fix_indicator[0] != '1'){ //Display time once we solve for time
        line2[0] = 'U';
        line2[1] = 'T';
        line2[2] = 'C';
        line2[3] = ':';
        for (int i=0; i < GGA_UTC_BUFFER_SIZE; i++){
            line2[4+i] = utc_time[i];
        }

        char* temp = "Getting PVT Solution";
        for (int i = 0; i < MAX_COL; i++){
            line3[i] = temp[i];
        }

    } else if (position_fix_indicator[0] == '1') {    //Once we get a fix, go into normal operation
        convertNMEAtoLLA();

        //Mode agnostic parts
        //line0
        for (int i = 0; i < LLA_LAT_BUFFER_SIZE; i++){
            line0[i] = latitudeLLA_str[i];
        }

        line0[19] = ut_mode + '0';
        line0[18] = ':';
        line0[17] = 'e';
        line0[16] = 'd';
        line0[15] = 'o';
        line0[14] = 'M';

        for (int i = 0; i < LLA_LONG_BUFFER_SIZE; i++){
            line1[i] = longitudeLLA_str[i];
        }

        //mode-specific parts
        if (ut_mode == STAT_MODE){
            //line1
            line1[MAX_COL-8] = 'H';
            line1[MAX_COL-7] = 'D';
            line1[MAX_COL-6] = 'O';
            line1[MAX_COL-5] = 'P';
            line1[MAX_COL-4] = ':';
            line1[MAX_COL-3] = hdop[0];
            line1[MAX_COL-2] = hdop[1];
            line1[MAX_COL-1] = hdop[2];

            //line2
            line2[0] = 'U';
            line2[1] = 'T';
            line2[2] = 'C';
            line2[3] = ':';
            for (int i=0; i < GGA_UTC_BUFFER_SIZE; i++){
                line2[4+i] = utc_time[i];
            }

            line2[MAX_COL-1] = satellites_used[1];
            line2[MAX_COL-2] = satellites_used[0];
            line2[MAX_COL-3] = ':';
            line2[MAX_COL-4] = 'V';
            line2[MAX_COL-5] = 'S';
            line2[MAX_COL-6] = '#';

            //line3
            line3[0] = 'V';
            line3[1] = 'e';
            line3[2] = 'l';
            line3[3] = ':';
            //put vel here once we get it done TODO
            for (int i =0; i < 6; i++){
                line3[4+i] = speed[i];
            }

            line3[MAX_COL-9] = 'A';
            line3[MAX_COL-8] = 'l';
            line3[MAX_COL-7] = 't';
            for (int i = 0; i < 6; i ++){
                line3[MAX_COL-6+i] = msl_altitude[i];
            }

        }else { //if mode != STAT_MODE
            //line 1
            switch (ut_operation){
                case SAVE_OP:
                    strncpy(line1+(MAX_COL-5), SAVE_STR, 5);
                    break;
                case CLEAR_OP:
                    strncpy(line1+(MAX_COL-5), CLEAR_STR, 5);
                    break;
                case RESET_OP:
                    strncpy(line1+(MAX_COL-5), RESET_STR, 5);
                    break;
                default:
                    strncpy(line1+(MAX_COL-5), "Error", 5);
                    break;
            }
            //line2
            line2[MAX_COL-4] = 'M';
            line2[MAX_COL-3] = 'e';
            line2[MAX_COL-2] = 'm';
            line2[MAX_COL-1] = ut_memory_0idx + '0';

            for (int i = 0; i < LLA_LAT_BUFFER_SIZE-2; i++){
                line2[i] = ut_lat_mem_str[i];
            }
            for (int i = 0; i < LLA_LONG_BUFFER_SIZE; i++){
                line3[i] = ut_long_mem_str[i];
            }

            //line3
            line3[MAX_COL-10]= 'D';
            line3[MAX_COL-9]= 'i';
            line3[MAX_COL-8]= 's';
            line3[MAX_COL-7]= 't';


        } //end mode checks
    }//end else

    ds_print_string(line0, MAX_COL, 0);
    ds_print_string(line1, MAX_COL, 1);
    ds_print_string(line2, MAX_COL, 2);
    ds_print_string(line3, MAX_COL, 3);


} //end update display
```

3.11 G:/Project_Files/WayFindX/wfx_sw/wfx_sw/nf/nf.c File Reference

```
#include <avr/interrupt.h>
#include <string.h>
#include <stdlib.h>
#include <stdint-gcc.h>
#include "nf.h"
#include "nf_types.h"
#include "../lib/uart.h"
#include "../ds/ds.h"
#include "../ut/utilities.h"
```

Macros

- #define F_CPU 4000000UL
- #define UART_BAUD_RATE 9600
- #define NMEA_MSG_ID_SIZE 5
- #define GGA_TYPE "GPGGA"
- #define GLL_TYPE "GPGLL"

- #define GSA_TYPE "GPGSA"
- #define GSV_TYPE "GPGSV"
- #define MSS_TYPE "GPMSS"
- #define RMC_TYPE "GPRMC"
- #define VTG_TYPE "GPVTG"
- #define ZDA_TYPE "GPZDA"
- #define OK_TO_SEND_TYPE "PSRF1"

Functions

- void nf_clear_nav_strings ()

  Clear all navigation strings except UTC time.
- uint8_t nf_init ()

  Initialize the navigation fetch module. This function initializes UART communication and clears navigation strings.
- void get_serial_char (char ∗outputchar)

  Read a single character from the serial communication. This function blocks until a character is received.
- void read_nmea_msg_raw ()

  Read a raw NMEA message from the serial communication. This function reads a raw NMEA message and processes GGA and VTG messages. Additionally, it polls buttons between messages.
- void convertNMEAtoLLA ()

  Convert NMEA format coordinates to Latitude, Longitude, and Altitude (LLA) format. This function converts NMEA format coordinates to LLA format and stores them in global variables.

Variables

- char utc_time [GGA_UTC_BUFFER_SIZE]
- char latitude [GGA_LAT_BUFFER_SIZE]
- char ns_indicator [GGA_INDICATOR_SIZE]
- char longitude [GGA_LONG_BUFFER_SIZE]
- char ew_indicator [GGA_INDICATOR_SIZE]
- char position_fix_indicator [GGA_INDICATOR_SIZE]
- char satellites_used [GGA_SV_USD_BUFFER_SIZE]
- char hdop [GGA_HDOP_BUFFER_SIZE]
- char msl_altitude [GGA_ALTITUDE_BUFFER_SIZE]
- char speed [VTG_SPEED_BUFER_SIZE]
- float latitudeLLA_float
- float longitudeLLA_float
- float altitudeLLA_float
- char latitudeLLA_str [LLA_LAT_BUFFER_SIZE]
- char longitudeLLA_str [LLA_LONG_BUFFER_SIZE]
- char altitudeLLA_str [LLA_ALT_BUFFER_SIZE]

3.11.1 Macro Definition Documentation

3.11.1.1 F_CPU

```
#define F_CPU 4000000UL
```

Define the CPU frequency to 4MHz.

Definition at line 2 of file nf.c.

3.11.1.2 GGA_TYPE

```
#define GGA_TYPE "GPGGA"
```

Definition at line 20 of file nf.c.

3.11.1.3 GLL_TYPE

```
#define GLL_TYPE "GPGLL"
```

Definition at line 21 of file nf.c.

3.11.1.4 GSA_TYPE

```
#define GSA_TYPE "GPGSA"
```

Definition at line 22 of file nf.c.

3.11.1.5 GSV_TYPE

```
#define GSV_TYPE "GPGSV"
```

Definition at line 23 of file nf.c.

3.11.1.6 MSS_TYPE

```
#define MSS_TYPE "GPMSS"
```

Definition at line 24 of file nf.c.

3.11.1.7 NMEA_MSG_ID_SIZE

```
#define NMEA_MSG_ID_SIZE 5
```

Definition at line 17 of file nf.c.

3.11.1.8 OK_TO_SEND_TYPE

```
#define OK_TO_SEND_TYPE "PSRF1"
```

Definition at line 28 of file nf.c.

3.11.1.9 RMC_TYPE

```
#define RMC_TYPE "GPRMC"
```

Definition at line 25 of file nf.c.

3.11.1.10 UART_BAUD_RATE

```
#define UART_BAUD_RATE 9600
```

Definition at line 16 of file nf.c.

3.11.1.11 VTG_TYPE

```
#define VTG_TYPE "GPVTG"
```

Definition at line 26 of file nf.c.

3.11.1.12 ZDA_TYPE

```
#define ZDA_TYPE "GPZDA"
```

Definition at line 27 of file nf.c.

3.11.2 Function Documentation

3.11.2.1 convertNMEAtoLLA()

```
void convertNMEAtoLLA ( )
```

Convert NMEA format coordinates to Latitude, Longitude, and Altitude (LLA) format. This function converts NMEA format coordinates to LLA format and stores them in global variables.

Convert NMEA format coordinates to Latitude, Longitude, and Altitude (LLA) format.

Definition at line 334 of file nf.c.

3.11.2.2 get_serial_char()

```
void get_serial_char (
             char * outputchar )
```

Read a single character from the serial communication. This function blocks until a character is received.

Read a single character from the serial communication.

Parameters

outputchar	Pointer to the character variable to store the read character.

Definition at line 114 of file nf.c.

3.11.2.3 nf_clear_nav_strings()

```
void nf_clear_nav_strings ( )
```

Clear all navigation strings except UTC time.

Definition at line 60 of file nf.c.

3.11.2.4 nf_init()

```
uint8_t nf_init ( )
```

Initialize the navigation fetch module. This function initializes UART communication and clears navigation strings.

Initialize the navigation fetch module.

Returns

0 if initialization is successful, otherwise returns 1.

Definition at line 82 of file nf.c.

3.11.2.5 read_nmea_msg_raw()

```
void read_nmea_msg_raw ( )
```

Read a raw NMEA message from the serial communication. This function reads a raw NMEA message and processes GGA and VTG messages. Additionally, it polls buttons between messages.

Read a raw NMEA message from the serial communication and extracts relevant data.

Definition at line 176 of file nf.c.

3.11.3 Variable Documentation

3.11.3.1 altitudeLLA_float

```
float altitudeLLA_float
```

Altitude in meters

Definition at line 45 of file nf.c.

3.11.3.2 altitudeLLA_str

```
char altitudeLLA_str[LLA_ALT_BUFFER_SIZE]
```

Altitude in meters

Definition at line 48 of file nf.c.

3.11.3.3 ew_indicator

```
char ew_indicator[GGA_INDICATOR_SIZE]
```

E/W Indicator, 'E' for east or 'W' for west

Definition at line 36 of file nf.c.

3.11.3.4 hdop

```
char hdop[GGA_HDOP_BUFFER_SIZE]
```

HDOP (Horizontal Dilution of Precision), e.g., "1.0"

Definition at line 39 of file nf.c.

3.11.3.5 latitude

```
char latitude[GGA_LAT_BUFFER_SIZE]
```

Latitude, e.g., "3723.2475"

Definition at line 33 of file nf.c.

3.11.3.6 latitudeLLA_float

```
float latitudeLLA_float
```

Latitude in degrees

Definition at line 43 of file nf.c.

3.11.3.7 latitudeLLA_str

```
char latitudeLLA_str[LLA_LAT_BUFFER_SIZE]
```

Latitude in degrees

Definition at line 46 of file nf.c.

3.11.3.8 longitude

```
char longitude[GGA_LONG_BUFFER_SIZE]
```

Longitude, e.g., "12158.3416"

Definition at line 35 of file nf.c.

3.11.3.9 longitudeLLA_float

```
float longitudeLLA_float
```

Longitude in degrees

Definition at line 44 of file nf.c.

3.11.3.10 longitudeLLA_str

```
char longitudeLLA_str[LLA_LONG_BUFFER_SIZE]
```

Longitude in degrees

Definition at line 47 of file nf.c.

3.11.3.11 msl_altitude

```
char msl_altitude[GGA_ALTITUDE_BUFFER_SIZE]
```

Mean Sea Level Altitude, e.g., "1.0"

Definition at line 40 of file nf.c.

3.11.3.12 ns_indicator

```
char ns_indicator[GGA_INDICATOR_SIZE]
```

N/S Indicator, 'N' for north or 'S' for south

Definition at line 34 of file nf.c.

3.11.3.13 position_fix_indicator

```
char position_fix_indicator[GGA_INDICATOR_SIZE]
```

Position Fix Indicator, see Table 1-4

Definition at line 37 of file nf.c.

3.11.3.14 satellites_used

```
char satellites_used[GGA_SV_USD_BUFFER_SIZE]
```

Satellites Used, range 0 to 12 eg 07

Definition at line 38 of file nf.c.

3.11.3.15 speed

```
char speed[VTG_SPEED_BUFER_SIZE]
```

Speed, e.g., "0.0"

Definition at line 41 of file nf.c.

3.11.3.16 utc_time

```
char utc_time[GGA_UTC_BUFFER_SIZE]
```

UTC Time, e.g., "161229.487"

Definition at line 32 of file nf.c.

3.12 nf.c

Go to the documentation of this file.

```
00001 #ifndef F_CPU
00002 #define F_CPU 4000000UL
00003 #endif
00004
00005 #include <avr/interrupt.h>
00006 #include <string.h>
00007 #include <stdlib.h>
00008 #include <stdint-gcc.h>
00009 #include "nf.h"
00010 #include "nf_types.h"
00011 #include "../lib/uart.h"
00012 #include "../ds/ds.h"
00013 #include "../ut/utilities.h"
00014
00015 //defines
00016 #define UART_BAUD_RATE 9600
00017 #define NMEA_MSG_ID_SIZE 5
00018
00019 //NMEA message IDs from NMEA documentation
00020 #define GGA_TYPE "GPGGA"
00021 #define GLL_TYPE "GPGLL"
00022 #define GSA_TYPE "GPGSA"
00023 #define GSV_TYPE "GPGSV"
00024 #define MSS_TYPE "GPMSS"
00025 #define RMC_TYPE "GPRMC"
00026 #define VTG_TYPE "GPVTG"
00027 #define ZDA_TYPE "GPZDA"
00028 #define OK_TO_SEND_TYPE "PSRF1"
00029
00030 //global
00031 //GGA MESSAGE
00032 char utc_time[GGA_UTC_BUFFER_SIZE];
00033 char latitude[GGA_LAT_BUFFER_SIZE];
00034 char ns_indicator[GGA_INDICATOR_SIZE];
00035 char longitude[GGA_LONG_BUFFER_SIZE];
00036 char ew_indicator[GGA_INDICATOR_SIZE];
00037 char position_fix_indicator[GGA_INDICATOR_SIZE];
00038 char satellites_used[GGA_SV_USD_BUFFER_SIZE];
00039 char hdop[GGA_HDOP_BUFFER_SIZE];
00040 char msl_altitude[GGA_ALTITUDE_BUFFER_SIZE];
00041 char speed[VTG_SPEED_BUFER_SIZE];
00043 float latitudeLLA_float;
00044 float longitudeLLA_float;
00045 float altitudeLLA_float;
00046 char latitudeLLA_str[LLA_LAT_BUFFER_SIZE];
00047 char longitudeLLA_str[LLA_LONG_BUFFER_SIZE];
00048 char altitudeLLA_str[LLA_ALT_BUFFER_SIZE];
00050 //local static
00051 static char nmea_msg_id_buffer[NMEA_MSG_ID_SIZE];
00052 static char gga_msg_buffer[GGA_SIZE];
00055 //function definitions
00056
```

```c
void nf_clear_nav_strings(){
        memset(nmea_msg_id_buffer,0,sizeof(char)*NMEA_MSG_ID_SIZE); //zeroize
        memset(gga_msg_buffer, ' ',sizeof(char)*GGA_SIZE); //zeroize msg buffer
        memset(latitude, ' ', GGA_LAT_BUFFER_SIZE * sizeof(char));
        memset(ns_indicator, ' ', GGA_INDICATOR_SIZE * sizeof(char));
        memset(longitude, ' ', GGA_LONG_BUFFER_SIZE * sizeof(char));
        memset(ew_indicator, ' ', GGA_INDICATOR_SIZE * sizeof(char));
        memset(position_fix_indicator, ' ', GGA_INDICATOR_SIZE * sizeof(char));
        memset(satellites_used, ' ', GGA_SV_USD_BUFFER_SIZE * sizeof(char));
        memset(hdop, ' ', GGA_HDOP_BUFFER_SIZE * sizeof(char));
        memset(msl_altitude, ' ', GGA_ALTITUDE_BUFFER_SIZE * sizeof(char));

        memset(latitudeLLA_str, ' ', LLA_LAT_BUFFER_SIZE * sizeof(char));
        memset(longitudeLLA_str, ' ', LLA_LONG_BUFFER_SIZE * sizeof(char));
        memset(altitudeLLA_str, ' ', LLA_ALT_BUFFER_SIZE * sizeof(char));
}

uint8_t nf_init(){
    cli();
     /*
     *  Initialize UART library, pass baudrate and AVR cpu clock
     *  with the macro
     *  UART_BAUD_SELECT() (normal speed mode )
     *  or
     *  UART_BAUD_SELECT_DOUBLE_SPEED() ( double speed mode)
     */
    uart_init( UART_BAUD_SELECT(UART_BAUD_RATE,F_CPU) );

    // Initialize the arrays within gga_msg
    memset(utc_time, ' ', GGA_UTC_BUFFER_SIZE * sizeof(char));
    memset(speed, ' ', VTG_SPEED_BUFER_SIZE * sizeof(char));

    nf_clear_nav_strings();


    latitudeLLA_float = 0;
    longitudeLLA_float = 0;
    altitudeLLA_float = 0;


    sei(); //UART is interrupt based;
    return NF_INIT_SUCCESS;
}

void get_serial_char(char* outputchar){
    unsigned int c;
    while (1){
        /*
        * Get received character from ringbuffer
        * uart_getc() returns in the lower byte the received character and
        * in the higher byte (bitmask) the last receive error
        * UART_NO_DATA is returned when no data is available.
        *
        */
        c = uart_getc();
        if (!( c & UART_NO_DATA ))
        {
            /*
                * new data available from UART
                * check for Frame or Overrun error
                */
            #ifdef __DEBUG__
                if ( c & UART_FRAME_ERROR )
                {
                    /* Framing Error detected, i.e no stop bit detected */
                    #ifdef _DEBUG_
                        char* output = "NF Frame Error! ";
                        ds_print_string(output, 16, 0);
                    #endif
                }
                if ( c & UART_OVERRUN_ERROR )
                {
                    /*
                        * Overrun, a character already present in the UART UDR register was
                        * not read by the interrupt handler before the next character arrived,
                        * one or more received characters have been dropped
                        */
                    char* output = "           OR ER";
                    ds_print_string(output, 16, 0);
                }
                if ( c & UART_BUFFER_OVERFLOW )
                {
                    /*
                        * We are not reading the receive buffer fast enough,
                        * one or more received character have been dropped
                        */
                    char* output = "OF ER";
                    ds_print_string(output, 5, 0);
                }
            #endif

            /*
            * send received character back
            */
            *outputchar = (unsigned char)c;
            return;
        }
    }

}

void read_nmea_msg_raw(){
    //nf_clear_nav_strings();
    char tempChar;
    do {
        get_serial_char(&tempChar);
    } while (tempChar != '$');

    for (int i = 0; i < NMEA_MSG_ID_SIZE; i++){
        get_serial_char(nmea_msg_id_buffer + i);
    }
    get_serial_char(&tempChar); //eat comma

    int counter = 0; //common counter to save memory alloc time
    //-----------------------------------------------------------------
    //only need GGA and VTG for requirements. Drop others
    if(strcmp(nmea_msg_id_buffer, GGA_TYPE) == 0){
        //scrape raw
        for (counter = 0; counter < GGA_SIZE; counter ++){
            get_serial_char(gga_msg_buffer+counter);
        }
        //process message
        int offset = 0;
        //grab UTC if available
        if (gga_msg_buffer[offset] != ','){
            for (int i =0; i < GGA_UTC_BUFFER_SIZE; i++){
                utc_time[i] = gga_msg_buffer[offset++];
            }
            //ds_print_string(utc_time, GGA_UTC_BUFFER_SIZE, 0);
        }else{ //otherwise skip comma from if statement
            offset++;
        }
        offset++;// skip comma

        //grab LAT if available
        if (gga_msg_buffer[offset] != ','){
            for (int i =0; i < GGA_LAT_BUFFER_SIZE; i++){
                latitude[i] = gga_msg_buffer[offset++];
            }

            //ds_print_string(latitude, GGA_LAT_BUFFER_SIZE, 0);
        }else{ //otherwise skip comma from if statement
            offset++;
        }
        offset++; // skip comma

        //grab NS indicator if available
        if (gga_msg_buffer[offset] != ','){
            ns_indicator[0] = gga_msg_buffer[offset++];
            //ds_print_string(ew_indicator, GGA_INDICATOR_SIZE, 1);
        }else{ //otherwise skip comma from if statement
            offset++;
        }
        offset++; // skip comma


        //grab LONG if available
        if (gga_msg_buffer[offset] != ','){
            for (int i =0; i < GGA_LONG_BUFFER_SIZE; i++){
                longitude[i] = gga_msg_buffer[offset++];
            }
            //ds_print_string(longitude, GGA_LONG_BUFFER_SIZE, 1);
        }else{ //otherwise skip comma from if statement
            offset++;
        }
        offset++; // skip comma

        //grab EW indicator if available
        if (gga_msg_buffer[offset] != ','){
            for (int i =0; i < GGA_INDICATOR_SIZE; i++){
                ew_indicator[i] = gga_msg_buffer[offset++];
            }
            //ds_print_string(ew_indicator, GGA_INDICATOR_SIZE, 1);
            }else{ //otherwise skip comma from if statement
            offset++;
        }
        offset++; // skip comma


        //grab FIX indicator if available
        if (gga_msg_buffer[offset] != ','){
            for (int i =0; i < GGA_INDICATOR_SIZE; i++){
                position_fix_indicator[i] = gga_msg_buffer[offset++];
            }
            //ds_print_string(position_fix_indicator, GGA_INDICATOR_SIZE, 1);
            }else{ //otherwise skip comma from if statement
            offset++;
        }
        offset++; // skip comma

        //grab NUM_SV if available
        if (gga_msg_buffer[offset] != ','){
            for (int i =0; i < GGA_SV_USD_BUFFER_SIZE; i++){
                satellites_used[i] = gga_msg_buffer[offset++];
            }
            //ds_print_string(satellites_used, GGA_SV_USD_BUFFER_SIZE, 1);
            }else{ //otherwise skip comma from if statement
            offset++;
        }
        offset++; // skip comma

        //grab HDOP if available
        if (gga_msg_buffer[offset] != ','){
            for (int i =0; i < GGA_HDOP_BUFFER_SIZE; i++){
                hdop[i] = gga_msg_buffer[offset++];
            }
            //ds_print_string(hdop, GGA_HDOP_BUFFER_SIZE, 0);
        }else{ //otherwise skip comma from if statement
            offset++;
        }
        offset++;
        //end of known sizes. Alt is dynamic
        tempChar = 0;
        int i = 0;
        get_serial_char(&tempChar);//eat comma
        get_serial_char(&tempChar);
        while ((tempChar != ',') && (i < GGA_ALTITUDE_BUFFER_SIZE)){
            msl_altitude[i++] = tempChar;
            get_serial_char(&tempChar);
        }


    } //END GGA
    else if (strcmp(nmea_msg_id_buffer, VTG_TYPE) == 0){
            for (uint8_t comma_counter = 6; comma_counter >0; comma_counter--){
                do{
                    get_serial_char(&tempChar);
                } while (tempChar != ',');
            }
            memset(speed, ' ', VTG_SPEED_BUFER_SIZE * sizeof(char));
            //should be at speed now
            int i = 0;
            get_serial_char(&tempChar);//eat comma
            while ((tempChar != ',') && (i < VTG_SPEED_BUFER_SIZE)){
                speed[i++] = tempChar;
                get_serial_char(&tempChar);
            }

    }//end VTG msg

    //poll buttons between messages
    //ut_poll_btns();
}




void convertNMEAtoLLA() {
    // Convert latitude from NMEA format to degrees
    double deg = 0.0;
    double min = 0.0;

    // Convert the latitude from degrees and minutes to degree decimal
    char tempBuffer[9] = {0};
    for (int i = 0; i < 8; i++){
        tempBuffer[i] = latitude[2+i];
    }
    min = atof(tempBuffer);

    deg += (1.0 * (latitude[GGA_LAT_BUFFER_SIZE - 9] - '0'));
    deg += (10.0 * (latitude[GGA_LAT_BUFFER_SIZE - 10] - '0'));
    latitudeLLA_float = deg + (min / 60.0f); // Combine degrees and minutes


    // Extract integer and decimal parts
    uint16_t integer_part = (uint16_t)latitudeLLA_float;
    float fractional_part = latitudeLLA_float - integer_part;
    uint32_t decimal_part = (uint32_t)(fractional_part * 100000);

    // Convert integer part to string
    latitudeLLA_str[1] = '0' + ((integer_part / 10) % 10); // Tens
    latitudeLLA_str[2] = '0' + (integer_part % 10); // Ones

    // Decimal point
    latitudeLLA_str[3] = '.';

    // Convert decimal part to string
    latitudeLLA_str[4] = '0' + ((decimal_part / 10000) % 10); // Ten-thousands
    latitudeLLA_str[5] = '0' + ((decimal_part / 1000)% 10); // Thousands
    latitudeLLA_str[6] = '0' + ((decimal_part / 100) % 10); // Hundreds
    latitudeLLA_str[7] = '0' + ((decimal_part / 10) % 10); // Tens
    latitudeLLA_str[8] = '0' + (decimal_part % 10); // Ones

    if (ns_indicator[0] =='S'){
        latitudeLLA_str[0] = '-';
        latitudeLLA_float *= -1;
        } else{
        latitudeLLA_str[0] = '+';
    }


    //long
    deg = 0.0;
    min = 0.0;

    // Convert the longitude from degrees and minutes to degree decimal
    char tempBuffer2[9] = {0};
    for (int i = 0; i < 8; i++){
        tempBuffer2[i] = longitude[3+i];
    }
    min = atof(tempBuffer2);

    deg += (1.0 * (longitude[2] - '0'));
    deg += (10.0 * (longitude[1] - '0'));
    deg += (100.0 * (longitude[0] - '0'));

    longitudeLLA_float = deg + (min / 60.0f); // Combine degrees and minutes

    // Extract integer and decimal parts
    integer_part = (uint16_t)longitudeLLA_float;
    fractional_part = longitudeLLA_float - integer_part;
    decimal_part = (uint32_t)(fractional_part * 100000);

    // Convert integer part to string
    longitudeLLA_str[1] = '0' + ((integer_part / 100) % 10); // Hundreds
    longitudeLLA_str[2] = '0' + ((integer_part / 10) % 10); // Tens
    longitudeLLA_str[3] = '0' + (integer_part % 10); // Ones

    // Decimal point
    longitudeLLA_str[4] = '.';

    // Convert decimal part to string
    longitudeLLA_str[5] = '0' + ((decimal_part / 10000) % 10); // Ten-thousands
    longitudeLLA_str[6] = '0' + ((decimal_part / 1000)% 10); // Thousands
    longitudeLLA_str[7] = '0' + ((decimal_part / 100) % 10); // Hundreds
    longitudeLLA_str[8] = '0' + ((decimal_part / 10) % 10); // Tens
    longitudeLLA_str[9] = '0' + (decimal_part % 10); // Ones

    if (ns_indicator[0] =='S'){
        longitudeLLA_str[0] = '-';
        latitudeLLA_float *= -1;
        } else{
        longitudeLLA_str[0] = '+';
    }

    altitudeLLA_float = atof(msl_altitude);
}
```

3.13 G:/Project_Files/WayFindX/wfx_sw/wfx_sw/nf/nf.h File Reference

Header file containing common functions and definitions for the navigation fetch (NV) CSC.

```
#include "../ut/ut_types.h"
#include "../ds/ds.h"
```

Macros

- #define NF_INIT_SUCCESS 0
- #define NF_INIT_FAILURE 1

Functions

- uint8_t nf_init ()
  Initialize the navigation fetch module.
- void get_serial_char (char ∗outputchar)
  Read a single character from the serial communication.
- void read_nmea_msg_raw ()
  Read a raw NMEA message from the serial communication and extracts relevant data.
- void convertNMEAtoLLA ()
  Convert NMEA format coordinates to Latitude, Longitude, and Altitude (LLA) format.

3.13.1 Detailed Description

Header file containing common functions and definitions for the navigation fetch (NV) CSC.

This file provides declarations for common utility functions and definitions associated with NV

Definition in file nf.h.

3.13.2 Macro Definition Documentation

3.13.2.1 NF_INIT_FAILURE

```
#define NF_INIT_FAILURE 1
```

Definition at line 15 of file nf.h.

3.13.2.2 NF_INIT_SUCCESS

```
#define NF_INIT_SUCCESS 0
```

Definition at line 14 of file nf.h.

3.13.3 Function Documentation

3.13.3.1 convertNMEAtoLLA()

```
void convertNMEAtoLLA ( )
```

Convert NMEA format coordinates to Latitude, Longitude, and Altitude (LLA) format.

Convert NMEA format coordinates to Latitude, Longitude, and Altitude (LLA) format.

Definition at line 334 of file nf.c.

3.13.3.2 get_serial_char()

```
void get_serial_char (
             char * outputchar )
```

Read a single character from the serial communication.

Parameters

outputchar	Pointer to the character variable to store the read character.

Read a single character from the serial communication.

Parameters

outputchar	Pointer to the character variable to store the read character.

Definition at line 114 of file nf.c.

3.13.3.3 nf_init()

```
uint8_t nf_init ( )
```

Initialize the navigation fetch module.

Returns

0 if initialization is successful

Initialize the navigation fetch module.

Returns

0 if initialization is successful, otherwise returns 1.

Definition at line 82 of file nf.c.

3.13.3.4 read_nmea_msg_raw()

```
void read_nmea_msg_raw ( )
```

Read a raw NMEA message from the serial communication and extracts relevant data.

Read a raw NMEA message from the serial communication and extracts relevant data.

Definition at line 176 of file nf.c.

3.14 nf.h

Go to the documentation of this file.

```
#ifndef NF_H_
#define NF_H_

#include "../ut/ut_types.h"
#include "../ds/ds.h"

#define NF_INIT_SUCCESS 0
#define NF_INIT_FAILURE 1


uint8_t nf_init();

void get_serial_char(char* outputchar);

void read_nmea_msg_raw();

void convertNMEAtoLLA();


#endif /* NF_H_ */
```

3.15 G:/Project_Files/WayFindX/wfx_sw/wfx_sw/nf/nf_types.h File Reference

Macros

- #define GGA_INDICATOR_SIZE 1
- #define LLA_LONG_BUFFER_SIZE 10
- #define LLA_LAT_BUFFER_SIZE 11
- #define LLA_ALT_BUFFER_SIZE 6
- #define GGA_UTC_BUFFER_SIZE 9
- #define GGA_LAT_BUFFER_SIZE 10
- #define GGA_LONG_BUFFER_SIZE 11
- #define GGA_SV_USD_BUFFER_SIZE 2
- #define GGA_HDOP_BUFFER_SIZE 3
- #define GGA_NUM_INDICATORS_ACTIVE 3
- #define GGA_NUM_ITEMS_ACTIVE 8
- #define GGA_ALTITUDE_BUFFER_SIZE 7
- #define VTG_SPEED_BUFER_SIZE 7
- #define GGA_SIZE (GGA_UTC_BUFFER_SIZE + GGA_LAT_BUFFER_SIZE + GGA_LONG_BUFFER_SIZE + GGA_SV_USD_BUFFER_SIZE + GGA_HDOP_BUFFER_SIZE + GGA_NUM_INDICATORS_ACTIVE + GGA_NUM_ITEMS_ACTIVE)

Variables

- char utc_time [GGA_UTC_BUFFER_SIZE]
- char latitude [GGA_LAT_BUFFER_SIZE]
- char ns_indicator [GGA_INDICATOR_SIZE]
- char longitude [GGA_LONG_BUFFER_SIZE]
- char ew_indicator [GGA_INDICATOR_SIZE]
- char position_fix_indicator [GGA_INDICATOR_SIZE]
- char satellites_used [GGA_SV_USD_BUFFER_SIZE]
- char hdop [GGA_HDOP_BUFFER_SIZE]
- char msl_altitude [GGA_ALTITUDE_BUFFER_SIZE]
- float latitudeLLA_float
- float longitudeLLA_float
- float altitudeLLA_float
- char latitudeLLA_str [LLA_LAT_BUFFER_SIZE]
- char longitudeLLA_str [LLA_LONG_BUFFER_SIZE]
- char altitudeLLA_str [LLA_ALT_BUFFER_SIZE]
- char speed [VTG_SPEED_BUFER_SIZE]

3.15.1 Macro Definition Documentation

3.15.1.1 GGA_ALTITUDE_BUFFER_SIZE

```
#define GGA_ALTITUDE_BUFFER_SIZE 7
```

Size of the buffer for storing altitude in the GGA message

Definition at line 26 of file nf_types.h.

3.15.1.2 GGA_HDOP_BUFFER_SIZE

```
#define GGA_HDOP_BUFFER_SIZE 3
```

Size of the buffer for storing HDOP in the GGA message

Definition at line 23 of file nf_types.h.

3.15.1.3 GGA_INDICATOR_SIZE

```
#define GGA_INDICATOR_SIZE 1
```

Size of the N/S and E/W indicators in the GGA message

Definition at line 12 of file nf_types.h.

3.15.1.4 GGA_LAT_BUFFER_SIZE

```
#define GGA_LAT_BUFFER_SIZE 10
```

Size of the buffer for storing latitude in the GGA message

Definition at line 20 of file nf_types.h.

3.15.1.5 GGA_LONG_BUFFER_SIZE

```
#define GGA_LONG_BUFFER_SIZE 11
```

Size of the buffer for storing longitude in the GGA message

Definition at line 21 of file nf_types.h.

3.15.1.6 GGA_NUM_INDICATORS_ACTIVE

```
#define GGA_NUM_INDICATORS_ACTIVE 3
```

Number of active indicators in the GGA message

Definition at line 24 of file nf_types.h.

3.15.1.7 GGA_NUM_ITEMS_ACTIVE

```
#define GGA_NUM_ITEMS_ACTIVE 8
```

Number of active items in the GGA message

Definition at line 25 of file nf_types.h.

3.15.1.8 GGA_SIZE

```
#define GGA_SIZE (GGA_UTC_BUFFER_SIZE + GGA_LAT_BUFFER_SIZE + GGA_LONG_BUFFER_SIZE + GGA_SV_USD_BUFFER_SIZE
+ GGA_HDOP_BUFFER_SIZE + GGA_NUM_INDICATORS_ACTIVE + GGA_NUM_ITEMS_ACTIVE)
```

Total size of the GGA message

Definition at line 29 of file nf_types.h.

3.15.1.9 GGA_SV_USD_BUFFER_SIZE

```
#define GGA_SV_USD_BUFFER_SIZE 2
```

Size of the buffer for storing satellites used in the GGA message

Definition at line 22 of file nf_types.h.

3.15.1.10 GGA_UTC_BUFFER_SIZE

```
#define GGA_UTC_BUFFER_SIZE 9
```

Size of the buffer for storing UTC time in the GGA message

Definition at line 19 of file nf_types.h.

3.15.1.11 LLA_ALT_BUFFER_SIZE

```
#define LLA_ALT_BUFFER_SIZE 6
```

Size of the buffer for storing altitude in LLA format

Definition at line 16 of file nf_types.h.

3.15.1.12 LLA_LAT_BUFFER_SIZE

```
#define LLA_LAT_BUFFER_SIZE 11
```

Size of the buffer for storing latitude in LLA format

Definition at line 15 of file nf_types.h.

3.15.1.13 LLA_LONG_BUFFER_SIZE

```
#define LLA_LONG_BUFFER_SIZE 10
```

Size of the buffer for storing longitude in LLA format

Definition at line 14 of file nf_types.h.

3.15.1.14 VTG_SPEED_BUFER_SIZE

```
#define VTG_SPEED_BUFER_SIZE 7
```

Size of the buffer for storing speed in the VTG message

Definition at line 27 of file nf_types.h.

3.15.2 Variable Documentation

3.15.2.1 altitudeLLA_float

```
float altitudeLLA_float  [extern]
```

Altitude in meters

Definition at line 45 of file nf.c.

3.15.2.2 altitudeLLA_str

```
char altitudeLLA_str[LLA_ALT_BUFFER_SIZE]  [extern]
```

Altitude in meters

Definition at line 48 of file nf.c.

3.15.2.3 ew_indicator

```
char ew_indicator[GGA_INDICATOR_SIZE]  [extern]
```

E/W Indicator, 'E' for east or 'W' for west

Definition at line 36 of file nf.c.

3.15.2.4 hdop

```
char hdop[GGA_HDOP_BUFFER_SIZE]  [extern]
```

HDOP (Horizontal Dilution of Precision), e.g., "1.0"

Definition at line 39 of file nf.c.

3.15.2.5 latitude

```
char latitude[GGA_LAT_BUFFER_SIZE]  [extern]
```

Latitude, e.g., "3723.24756"

Latitude, e.g., "3723.2475"

Definition at line 33 of file nf.c.

3.15.2.6 latitudeLLA_float

```
float latitudeLLA_float [extern]
```

Latitude in degrees

Definition at line 43 of file nf.c.

3.15.2.7 latitudeLLA_str

```
char latitudeLLA_str[LLA_LAT_BUFFER_SIZE] [extern]
```

Latitude in degrees

Definition at line 46 of file nf.c.

3.15.2.8 longitude

```
char longitude[GGA_LONG_BUFFER_SIZE] [extern]
```

Longitude, e.g., "12158.34166"

Longitude, e.g., "12158.3416"

Definition at line 35 of file nf.c.

3.15.2.9 longitudeLLA_float

```
float longitudeLLA_float [extern]
```

Longitude in degrees

Definition at line 44 of file nf.c.

3.15.2.10 longitudeLLA_str

```
char longitudeLLA_str[LLA_LONG_BUFFER_SIZE] [extern]
```

Longitude in degrees

Definition at line 47 of file nf.c.

3.15.2.11 msl_altitude

```
char msl_altitude[GGA_ALTITUDE_BUFFER_SIZE] [extern]
```

Altitude above sea level in meters one decimal of precision.

Mean Sea Level Altitude, e.g., "1.0"

Definition at line 40 of file nf.c.

3.15.2.12 ns_indicator

```
char ns_indicator[GGA_INDICATOR_SIZE]  [extern]
```

N/S Indicator, 'N' for north or 'S' for south

Definition at line 34 of file nf.c.

3.15.2.13 position_fix_indicator

```
char position_fix_indicator[GGA_INDICATOR_SIZE]  [extern]
```

Position Fix Indicator, see Table 1-4

Definition at line 37 of file nf.c.

3.15.2.14 satellites_used

```
char satellites_used[GGA_SV_USD_BUFFER_SIZE]  [extern]
```

Satellites Used, range 0 to 12 eg 07

Definition at line 38 of file nf.c.

3.15.2.15 speed

```
char speed[VTG_SPEED_BUFER_SIZE]  [extern]
```

Speed in km/hr

Speed, e.g., "0.0"

Definition at line 41 of file nf.c.

3.15.2.16 utc_time

```
char utc_time[GGA_UTC_BUFFER_SIZE]  [extern]
```

UTC Time, e.g., "161229.487"

Definition at line 32 of file nf.c.

3.16 nf_types.h

Go to the documentation of this file.

```
/*
 * nf_types.h
 * @brief Header file containing common types and definitions for the navigation fetch (NV) CSC.
 *
 * This file provides definitions for common data types and sizes associated with the NV CSC.
 */


#ifndef NF_TYPES_H_
#define NF_TYPES_H_

#define GGA_INDICATOR_SIZE 1
#define LLA_LONG_BUFFER_SIZE 10
#define LLA_LAT_BUFFER_SIZE 11
#define LLA_ALT_BUFFER_SIZE 6
// Size definitions for components of the GGA message
#define GGA_UTC_BUFFER_SIZE 9
#define GGA_LAT_BUFFER_SIZE 10
#define GGA_LONG_BUFFER_SIZE 11
#define GGA_SV_USD_BUFFER_SIZE 2
#define GGA_HDOP_BUFFER_SIZE 3
#define GGA_NUM_INDICATORS_ACTIVE 3
#define GGA_NUM_ITEMS_ACTIVE 8
#define GGA_ALTITUDE_BUFFER_SIZE 7
#define VTG_SPEED_BUFER_SIZE 7
#define GGA_SIZE (GGA_UTC_BUFFER_SIZE + GGA_LAT_BUFFER_SIZE + GGA_LONG_BUFFER_SIZE +
GGA_SV_USD_BUFFER_SIZE + GGA_HDOP_BUFFER_SIZE + GGA_NUM_INDICATORS_ACTIVE + GGA_NUM_ITEMS_ACTIVE)
extern char utc_time[GGA_UTC_BUFFER_SIZE];
extern char latitude[GGA_LAT_BUFFER_SIZE];
extern char ns_indicator[GGA_INDICATOR_SIZE];
extern char longitude[GGA_LONG_BUFFER_SIZE];
extern char ew_indicator[GGA_INDICATOR_SIZE];
extern char position_fix_indicator[GGA_INDICATOR_SIZE];
extern char satellites_used[GGA_SV_USD_BUFFER_SIZE];
extern char hdop[GGA_HDOP_BUFFER_SIZE];
extern char msl_altitude[GGA_ALTITUDE_BUFFER_SIZE];
extern float latitudeLLA_float;
extern float longitudeLLA_float;
extern float altitudeLLA_float;
extern char latitudeLLA_str[LLA_LAT_BUFFER_SIZE];
extern char longitudeLLA_str[LLA_LONG_BUFFER_SIZE];
extern char altitudeLLA_str[LLA_ALT_BUFFER_SIZE];
extern char speed[VTG_SPEED_BUFER_SIZE];
#endif /* NF_TYPES_H_ */
```

3.17 G:/Project_Files/WayFindX/wfx_sw/wfx_sw/ut/ut_types.h File Reference

Header file containing typedefs for common types.

Macros

- #define false ((boolean_t)0)

  Definition of false as boolean value 0.
- #define true ((boolean_t)1)

  Definition of true as boolean value 1.

Typedefs

- typedef unsigned char uint8_t

  8-bit unsigned integer type.
- typedef signed char int8_t

  8-bit signed integer type.
- typedef unsigned int uint16_t

  16-bit unsigned integer type.
- typedef unsigned char boolean_t

  Boolean type (true/false).

3.17.1 Detailed Description

Header file containing typedefs for common types.

This file provides typedefs for common types to improve code readability and portability.

Version

1.0

Copyright

(C) 2024 Bradley Johnson and Abele Atresso

Definition in file ut_types.h.

3.17.2 Macro Definition Documentation

3.17.2.1 false

```
#define false ((boolean_t)0)
```

Definition of false as boolean value 0.

Definition at line 37 of file ut_types.h.

3.17.2.2 true

```
#define true ((boolean_t)1)
```

Definition of true as boolean value 1.

Definition at line 42 of file ut_types.h.

3.17.3 Typedef Documentation

3.17.3.1 boolean_t

```
typedef unsigned char boolean_t
```

Boolean type (true/false).

Definition at line 32 of file ut_types.h.

3.17.3.2 int8_t

```
typedef signed char int8_t
```

8-bit signed integer type.

Definition at line 22 of file ut_types.h.

3.17.3.3 uint16_t

```
typedef unsigned int uint16_t
```

16-bit unsigned integer type.

Definition at line 27 of file ut_types.h.

3.17.3.4 uint8_t

```
typedef unsigned char uint8_t
```

8-bit unsigned integer type.

Definition at line 17 of file ut_types.h.

3.18 ut_types.h

Go to the documentation of this file.

```
00001
00011 #ifndef UT_TYPES_H_
00012 #define UT_TYPES_H_
00013
00017 typedef unsigned char  uint8_t;
00018
00022 typedef signed char    int8_t;
00023
00027 typedef unsigned int   uint16_t;
00028
00032 typedef unsigned char boolean_t;
00033
00037 #define false ((boolean_t)0)
00038
00042 #define true ((boolean_t)1)
00043
00044 #endif /* UT_TYPES_H_ */
```

3.19 G:/Project_Files/WayFindX/wfx_sw/wfx_sw/ut/utilities.c File Reference

```
#include <avr/io.h>
#include "utilities.h"
```

Functions

- boolean_t is_button_pressed (volatile uint8_t ∗port, uint8_t pin)

  Checks if a button is pressed.
- void ut_load_from_non_vol (uint8_t index, float ∗longitude, float ∗latitude)

  Loads position from non-volatile memory.
- void ut_write_to_non_vol (uint8_t index)

  Writes to non-volatile memory.
- void ut_convert_lat_float_to_string (float lat_float, char ∗lat_string)

  Converts latitude from float to string.
- void ut_convert_long_float_to_string (float long_float, char ∗long_string)

  Converts longitude from float to string.
- void ut_init ()

  Initializes the pins for buttons, loads from SD card, and initializes stored locations on startup.
- void ut_poll_btns ()

  Checks the status of each button sequentially.

Variables

- boolean_t ut_mode
- uint8_t ut_operation
- uint8_t ut_memory_0idx
- float ut_lat_mem_floats [MAX_MEM_INDEX]
- float ut_long_mem_floats [MAX_MEM_INDEX]
- char ut_lat_mem_str [LLA_LAT_BUFFER_SIZE]
- char ut_long_mem_str [LLA_LONG_BUFFER_SIZE]

3.19.1 Function Documentation

3.19.1.1 is_button_pressed()

```
boolean_t is_button_pressed (
            volatile uint8_t ∗ port,
            uint8_t pin )
```

Checks if a button is pressed.

Simple utility function to poll an individual button. keeping for maintainability if button needs switched to different port or pin.

This function checks whether a specified button is pressed.

Parameters

port	Pointer to the port register.
pin	The pin number of the button.

Returns

true if the button is pressed, false otherwise.

Definition at line 289 of file utilities.c.

3.19.1.2 ut_convert_lat_float_to_string()

```
void ut_convert_lat_float_to_string (
             float lat_float,
             char * lat_string )
```

Converts latitude from float to string.

This function converts a floating-point latitude value to a string.

Parameters

lat_float	The latitude value as a float.
lat_string	Pointer to the string to store the converted latitude.

Definition at line 67 of file utilities.c.

3.19.1.3 ut_convert_long_float_to_string()

```
void ut_convert_long_float_to_string (
             float long_float,
             char * long_string )
```

Converts longitude from float to string.

This function converts a floating-point longitude value to a string.

Parameters

long_float	The longitude value as a float.
long_string	Pointer to the string to store the converted longitude.

Definition at line 105 of file utilities.c.

3.19.1.4 ut_init()

```
void ut_init ( )
```

Initializes the pins for buttons, loads from SD card, and initializes stored locations on startup.

This function initializes the pins for buttons, loads data from an SD card, and initializes stored locations on startup.

Definition at line 142 of file utilities.c.

3.19.1.5 ut_load_from_non_vol()

```
void ut_load_from_non_vol (
            uint8_t index,
            float * longitude,
            float * latitude )
```

Loads position from non-volatile memory.

This function loads the longitude and latitude from non-volatile memory.

Parameters

index	The index of the memory location.
longitude	Pointer to store the longitude value.
latitude	Pointer to store the latitude value.

Definition at line 42 of file utilities.c.

3.19.1.6 ut_poll_btns()

```
void ut_poll_btns ( )
```

Checks the status of each button sequentially.

Checks status of each button sequentially. Action triggers on button release. Is to be called in background via interrupt. Post-polling action takes place in the following order: MODE_SELECT_BTN, MEM_SELECT_BTN, OP↵ _SELECT_BTN.

This function checks the status of each button sequentially. Action triggers on button release. It is to be called in the background via interrupt. Post-polling action takes place in the following order: MODE_SELECT_BTN, MEM↵ _SELECT_BTN, OP_SELECT_BTN.

Definition at line 189 of file utilities.c.

3.19.1.7 ut_write_to_non_vol()

```
void ut_write_to_non_vol (
            uint8_t index )
```

Writes to non-volatile memory.

This function writes data to non-volatile memory based on the provided index.

Parameters

index	The index of the memory location.

Definition at line 55 of file utilities.c.

3.19.2 Variable Documentation

3.19.2.1 ut_lat_mem_floats

```
float ut_lat_mem_floats[MAX_MEM_INDEX]
```

Array to store latitude

Definition at line 8 of file utilities.c.

3.19.2.2 ut_lat_mem_str

```
char ut_lat_mem_str[LLA_LAT_BUFFER_SIZE]
```

String to store latitude

Definition at line 10 of file utilities.c.

3.19.2.3 ut_long_mem_floats

```
float ut_long_mem_floats[MAX_MEM_INDEX]
```

Array to store longitude

Definition at line 9 of file utilities.c.

3.19.2.4 ut_long_mem_str

```
char ut_long_mem_str[LLA_LONG_BUFFER_SIZE]
```

String to store longitude

Definition at line 11 of file utilities.c.

3.19.2.5 ut_memory_0idx

```
uint8_t ut_memory_0idx
```

Index for memory

Definition at line 7 of file utilities.c.

3.19.2.6 ut_mode

```
boolean_t ut_mode
```

Current mode

Definition at line 5 of file utilities.c.

3.19.2.7 ut_operation

```
uint8_t ut_operation
```

Current operation

Definition at line 6 of file utilities.c.

3.20 utilities.c

Go to the documentation of this file.

```
00001 #include <avr/io.h>
00002 #include "utilities.h"
00003 
00004 //global variables
00005 boolean_t ut_mode;
00006 uint8_t ut_operation;
00007 uint8_t ut_memory_0idx;
00008 float ut_lat_mem_floats[MAX_MEM_INDEX];
00009 float ut_long_mem_floats[MAX_MEM_INDEX];
00010 char ut_lat_mem_str[LLA_LAT_BUFFER_SIZE];
00011 char ut_long_mem_str[LLA_LONG_BUFFER_SIZE];
00014 //local static variables
00015 static uint16_t num_button_polls;
00016 static uint16_t btn_on_time[NUM_BUTTONS];
00017 static uint8_t btn_off_time[NUM_BUTTONS];
00018 static boolean_t prev_state[NUM_BUTTONS];
00019 static boolean_t btn_state[NUM_BUTTONS];
00021 //local functions
00031 boolean_t is_button_pressed(volatile uint8_t *port, uint8_t pin);
00032 
00042 void ut_load_from_non_vol(uint8_t index, float* longitude, float* latitude){
00043     //TODO, load ith element of array from non-vol... or whole array at once if possible?
00044     *longitude = 0.0f;
00045     *latitude = 0.0f;
00046 }
00047 
00055 void ut_write_to_non_vol(uint8_t index){
00056     //TODO, look at index and read from ut_long_mem_floats and ut_lat_mem_floats
00057 }
00058 
00067 void ut_convert_lat_float_to_string(float lat_float, char* lat_string){
00068     //passed by value; will not change float from where function is called
00069     if (lat_float < 0){
00070         lat_string[0] = '-';
00071         lat_float *= -1; //make positive for integer math
00072     } else{
00073         lat_string[0] = '+';
00074     }
00075 
00076     // Extract integer and decimal parts
00077     uint16_t integer_part = (uint16_t)(lat_float);
00078     float fractional_part = lat_float - integer_part;
00079     uint32_t decimal_part = (uint32_t)(fractional_part * 100000);
00080 
00081 
00082     // Convert integer part to string
00083     lat_string[1] = '0' + ((integer_part / 10) % 10); // Tens
00084     lat_string[2] = '0' + (integer_part % 10); // Ones
00085 
00086     // Decimal point
00087     lat_string[3] = '.';
00088 
00089     // Convert decimal part to string
00090     lat_string[4] = '0' + ((decimal_part / 10000) % 10); // Ten-thousands
00091     lat_string[5] = '0' + ((decimal_part / 1000)% 10); // Thousands
00092     lat_string[6] = '0' + ((decimal_part / 100) % 10); // Hundreds
00093     lat_string[7] = '0' + ((decimal_part / 10) % 10); // Tens
00094     lat_string[8] = '0' + (decimal_part % 10); // Ones
00095 }
00096 
00105 void ut_convert_long_float_to_string(float long_float, char* long_string){
00106     //passed by value; will not change float from where function is called
00107     if (long_float < 0){
00108         long_string[0] = '-';
00109         long_float *= -1; //make positive for integer math
00110     } else{
        long_string[0] = ’+’;
    }

    // Extract integer and decimal parts
    uint16_t integer_part = (uint16_t)(long_float);
    float fractional_part = long_float - integer_part;
    uint32_t decimal_part = (uint32_t)(fractional_part * 100000);


    // Convert integer part to string
    long_string[1] = ’0’ + ((integer_part / 100) % 10); // Hundreds
    long_string[2] = ’0’ + ((integer_part / 10) % 10); // Tens
    long_string[3] = ’0’ + (integer_part % 10); // Ones

    // Decimal point
    long_string[4] = ’.’;

    // Convert decimal part to string
    long_string[5] = ’0’ + ((decimal_part / 10000) % 10); // Ten-thousands
    long_string[6] = ’0’ + ((decimal_part / 1000)% 10); // Thousands
    long_string[7] = ’0’ + ((decimal_part / 100) % 10); // Hundreds
    long_string[8] = ’0’ + ((decimal_part / 10) % 10); // Tens
    long_string[9] = ’0’ + (decimal_part % 10); // Ones
}


void ut_init()
{
    //read from SD card
    for (int i = 0; i < MAX_MEM_INDEX; i++){
        ut_load_from_non_vol(i, ut_long_mem_floats+i, ut_lat_mem_floats+i);
    }

    //initialize globals
    ut_mode = NAV_MODE;
    ut_operation = SAVE_OP;
    ut_memory_0idx = 0;

    ut_convert_lat_float_to_string(ut_lat_mem_floats[ut_memory_0idx], ut_lat_mem_str);
    ut_convert_long_float_to_string(ut_long_mem_floats[ut_memory_0idx], ut_long_mem_str);

    //init local static
    num_button_polls = 0;
    for (int i = 0; i < NUM_BUTTONS; i++){
        btn_on_time[i] = 0;
        btn_off_time[i] = 0;
        prev_state[i] = false;
        btn_state[i] = false;

    }


    // Set PD2, PD3, PD4, PD5 pins as inputs
    DDRC &= ~(1 « ACTION_BTN);
    DDRC &= ~(1 « OP_SELECT_BTN);
    DDRC &= ~(1 « MEM_SELECT_BTN);
    DDRC &= ~(1 « MODE_SELECT_BTN);

     // Enable pull-up resistors for PC0, PC1, PC2, PC3 pins
    PORTC |= (1 « ACTION_BTN);
    PORTC |= (1 « OP_SELECT_BTN);
    PORTC |= (1 « MEM_SELECT_BTN);
    PORTC |= (1 « MODE_SELECT_BTN);

    return;
}

void ut_poll_btns(){
    // Iterate through each button
    for (int i = 0; i < NUM_BUTTONS; i ++){
        prev_state[i] = btn_state[i];

        if (is_button_pressed(&PINC, i)) {
            // Increment button logic high count if pressed
            btn_on_time[i]++;
            btn_off_time[i] = 0;
            // Check if button press threshold is reached
            if (btn_on_time[i] >= ON_TIME_THRESHHOLD) {
                btn_state[i] = true; // Set button state to pressed
                btn_on_time[i] = 0;
                btn_off_time[i] = 0;
            }

        }else{
            btn_off_time[i]++;
            if (btn_off_time[i] >= RESET_TIME_THRESHHOLD){
                btn_state[i] = false; // Set button state to released
                btn_on_time[i] = 0;
                btn_off_time[i] = 0;
            }
        }



    } //end for loop


    // Check for virtual short and take action accordingly
    if (!(btn_state[MODE_SELECT_BTN]) && (prev_state[MODE_SELECT_BTN])) {
        // Mode select button pressed
        ut_mode ^= 1;  //toggle mode
    } else if ((ut_mode != STAT_MODE) && !(btn_state[MEM_SELECT_BTN]) && (prev_state[MEM_SELECT_BTN]))
{
        // Memory select button pressed
        ut_memory_0idx = (ut_memory_0idx + 1)%MAX_MEM_INDEX; //cycle memory index selected
        //update strings to reflect selected mem location
        ut_convert_lat_float_to_string(ut_lat_mem_floats[ut_memory_0idx], ut_lat_mem_str);
        ut_convert_long_float_to_string(ut_long_mem_floats[ut_memory_0idx], ut_long_mem_str);

    } else if ((ut_mode != STAT_MODE) && !(btn_state[OP_SELECT_BTN]) && (prev_state[OP_SELECT_BTN])) {
        // Operation select button pressed
        ut_operation = (ut_operation + 1)%NUM_OPERATIONS; //cycle operation selected

    } else if ((ut_mode != STAT_MODE) && !(btn_state[ACTION_BTN]) && (prev_state[ACTION_BTN])) {
        // Action button pressed
        //#TODO Implement action for action button press
        switch (ut_operation){
            case SAVE_OP:
                //Load into global array
                ut_lat_mem_floats[ut_memory_0idx] = latitudeLLA_float;
                ut_long_mem_floats[ut_memory_0idx] = longitudeLLA_float;

                //update string
                ut_convert_lat_float_to_string(ut_lat_mem_floats[ut_memory_0idx], ut_lat_mem_str);
                ut_convert_long_float_to_string(ut_long_mem_floats[ut_memory_0idx], ut_long_mem_str);

                //write to SD card
                ut_write_to_non_vol(ut_memory_0idx);

            break;
            case CLEAR_OP:
                 //Load into global array
                 ut_lat_mem_floats[ut_memory_0idx] = 0.0f;
                 ut_long_mem_floats[ut_memory_0idx] = 0.0f;

                 //update string
                 ut_convert_lat_float_to_string(ut_lat_mem_floats[ut_memory_0idx], ut_lat_mem_str);
                 ut_convert_long_float_to_string(ut_long_mem_floats[ut_memory_0idx], ut_long_mem_str);

                 //write to SD card
                 ut_write_to_non_vol(ut_memory_0idx);

            break;
            case RESET_OP:
                for (int i  = 0; i < MAX_MEM_INDEX; i++){
                    //Load into global array
                    ut_lat_mem_floats[i] = 0.0f;
                    ut_long_mem_floats[i] = 0.0f;

                    //update string
                    ut_convert_lat_float_to_string(ut_lat_mem_floats[i], ut_lat_mem_str);
                    ut_convert_long_float_to_string(ut_long_mem_floats[i], ut_long_mem_str);

                    //write to SD card
                    ut_write_to_non_vol(i);
                }
            break;
            default: //not reachable; error will show on screen (see main)
            break;
        }
    }
}

//local function definition
boolean_t is_button_pressed(volatile uint8_t *port, uint8_t pin) {
    // Check if the button is pressed (logic low)
    if (!(*port & (1 « pin))) {
        return true;
    }
    return false;
}
```

3.21 G:/Project_Files/WayFindX/wfx_sw/wfx_sw/ut/utilities.h File Reference

Header file containing common utility functions and definitions.

```
#include "ut_types.h"
#include "../nf/nf_types.h"
```

Macros

- #define NOP asm("nop");
- #define STAT_MODE 1
- #define NAV_MODE 0
- #define NUM_OPERATIONS 3
- #define SAVE_OP 0
- #define CLEAR_OP 1
- #define RESET_OP 2
- #define MAX_MEM_INDEX 10
- #define SAVE_STR " SAVE"
- #define CLEAR_STR "CLEAR"
- #define RESET_STR "RESET"
- #define NUM_BUTTONS 4
- #define MODE_SELECT_BTN PINC0
- #define MEM_SELECT_BTN PINC1
- #define OP_SELECT_BTN PINC2
- #define ACTION_BTN PINC3
- #define ON_TIME_THRESHHOLD (uint16_t)100
- #define RESET_TIME_THRESHHOLD (uint16_t)30

Functions

- void ut_init ()

  Initializes the pins for buttons, loads from SD card, and initializes stored locations on startup.
- void ut_poll_btns ()

  Checks status of each button sequentially. Action triggers on button release. Is to be called in background via interrupt. Post-polling action takes place in the following order: MODE_SELECT_BTN, MEM_SELECT_BTN, OP_↩ SELECT_BTN.

Variables

- boolean_t ut_mode
- uint8_t ut_operation
- uint8_t ut_memory_0idx
- float ut_lat_mem_floats [MAX_MEM_INDEX]
- float ut_long_mem_floats [MAX_MEM_INDEX]
- char ut_lat_mem_str [LLA_LAT_BUFFER_SIZE]
- char ut_long_mem_str [LLA_LONG_BUFFER_SIZE]

3.21.1 Detailed Description

Header file containing common utility functions and definitions.

This file provides declarations for common utility functions and definitions that can be used throughout the project.

Version

1.0

Copyright

(C) 2024 Bradley Johnson and Abele Atresso

Definition in file utilities.h.

3.21.2 Macro Definition Documentation

3.21.2.1 ACTION_BTN

```
#define ACTION_BTN PINC3
```

Action button pin.

Definition at line 36 of file utilities.h.

3.21.2.2 CLEAR_OP

```
#define CLEAR_OP 1
```

Clear operation index.

Definition at line 24 of file utilities.h.

3.21.2.3 CLEAR_STR

```
#define CLEAR_STR "CLEAR"
```

Clear operation string.

Definition at line 29 of file utilities.h.

3.21.2.4 MAX_MEM_INDEX

```
#define MAX_MEM_INDEX 10
```

Maximum memory index.

Definition at line 27 of file utilities.h.

3.21.2.5 MEM_SELECT_BTN

```
#define MEM_SELECT_BTN PINC1
```

Memory select button pin.

Definition at line 34 of file utilities.h.

3.21.2.6 MODE_SELECT_BTN

```
#define MODE_SELECT_BTN PINC0
```

Mode select button pin.

Definition at line 33 of file utilities.h.

3.21.2.7 NAV_MODE

```
#define NAV_MODE 0
```

Mode indicating navigation.

Definition at line 20 of file utilities.h.

3.21.2.8 NOP

```
#define NOP asm("nop");
```

No operation macro.

Definition at line 17 of file utilities.h.

3.21.2.9 NUM_BUTTONS

```
#define NUM_BUTTONS 4
```

Number of buttons.

Definition at line 32 of file utilities.h.

3.21.2.10 NUM_OPERATIONS

```
#define NUM_OPERATIONS 3
```

Number of available operations.

Definition at line 22 of file utilities.h.

3.21.2.11 ON_TIME_THRESHHOLD

```
#define ON_TIME_THRESHHOLD (uint16_t)100
```

Button press threshold time.

Definition at line 38 of file utilities.h.

3.21.2.12 OP_SELECT_BTN

```
#define OP_SELECT_BTN PINC2
```

Operation select button pin.

Definition at line 35 of file utilities.h.

3.21.2.13 RESET_OP

```
#define RESET_OP 2
```

Reset operation index.

Definition at line 25 of file utilities.h.

3.21.2.14 RESET_STR

```
#define RESET_STR "RESET"
```

Reset operation string.

Definition at line 30 of file utilities.h.

3.21.2.15 RESET_TIME_THRESHHOLD

```
#define RESET_TIME_THRESHHOLD (uint16_t)30
```

Button release threshold time.

Definition at line 39 of file utilities.h.

3.21.2.16 SAVE_OP

```
#define SAVE_OP 0
```

Save operation index.

Definition at line 23 of file utilities.h.

3.21.2.17 SAVE_STR

```
#define SAVE_STR " SAVE"
```

Save operation string.

Definition at line 28 of file utilities.h.

3.21.2.18 STAT_MODE

```
#define STAT_MODE 1
```

Mode indicating status.

Definition at line 19 of file utilities.h.

3.21.3 Function Documentation

3.21.3.1 ut_init()

```
void ut_init ( )
```

Initializes the pins for buttons, loads from SD card, and initializes stored locations on startup.

This function initializes the pins for buttons, loads data from an SD card, and initializes stored locations on startup.

Definition at line 142 of file utilities.c.

3.21.3.2 ut_poll_btns()

```
void ut_poll_btns ( )
```

Checks status of each button sequentially. Action triggers on button release. Is to be called in background via interrupt. Post-polling action takes place in the following order: MODE_SELECT_BTN, MEM_SELECT_BTN, OP←_SELECT_BTN.

Checks status of each button sequentially. Action triggers on button release. Is to be called in background via interrupt. Post-polling action takes place in the following order: MODE_SELECT_BTN, MEM_SELECT_BTN, OP←_SELECT_BTN.

This function checks the status of each button sequentially. Action triggers on button release. It is to be called in the background via interrupt. Post-polling action takes place in the following order: MODE_SELECT_BTN, MEM←_SELECT_BTN, OP_SELECT_BTN.

Definition at line 189 of file utilities.c.

3.21.4 Variable Documentation

3.21.4.1 ut_lat_mem_floats

```
float ut_lat_mem_floats[MAX_MEM_INDEX]  [extern]
```

Array to store latitude memory floats.

Array to store latitude

Definition at line 8 of file utilities.c.

3.21.4.2 ut_lat_mem_str

```
char ut_lat_mem_str[LLA_LAT_BUFFER_SIZE]  [extern]
```

Array to store latitude memory strings.

String to store latitude

Definition at line 10 of file utilities.c.

3.21.4.3 ut_long_mem_floats

```
float ut_long_mem_floats[MAX_MEM_INDEX]  [extern]
```

Array to store longitude memory floats.

Array to store longitude

Definition at line 9 of file utilities.c.

3.21.4.4 ut_long_mem_str

```
char ut_long_mem_str[LLA_LONG_BUFFER_SIZE]  [extern]
```

Array to store longitude memory strings.

String to store longitude

Definition at line 11 of file utilities.c.

3.21.4.5 ut_memory_0idx

```
uint8_t ut_memory_0idx  [extern]
```

Current memory index.

Index for memory

Definition at line 7 of file utilities.c.

3.21.4.6 ut_mode

```
boolean_t ut_mode  [extern]
```

Current mode indicator.

Current mode

Definition at line 5 of file utilities.c.

3.21.4.7 ut_operation

```
uint8_t ut_operation  [extern]
```

Current operation index.

Current operation

Definition at line 6 of file utilities.c.

3.22 utilities.h

Go to the documentation of this file.

```

#ifndef UTILITIES_H
#define UTILITIES_H

#include "ut_types.h"
#include "../nf/nf_types.h"

#define NOP asm("nop");
#define STAT_MODE 1
#define NAV_MODE 0
#define NUM_OPERATIONS 3
#define SAVE_OP 0
#define CLEAR_OP 1
#define RESET_OP 2
#define MAX_MEM_INDEX 10
#define SAVE_STR  " SAVE"
#define CLEAR_STR "CLEAR"
#define RESET_STR "RESET"
#define NUM_BUTTONS 4
#define MODE_SELECT_BTN PINC0
#define MEM_SELECT_BTN PINC1
#define OP_SELECT_BTN PINC2
#define ACTION_BTN PINC3
#define ON_TIME_THRESHHOLD (uint16_t)100
#define RESET_TIME_THRESHHOLD (uint16_t)30
extern boolean_t ut_mode;
extern uint8_t ut_operation;
extern uint8_t ut_memory_0idx;
extern float ut_lat_mem_floats[MAX_MEM_INDEX];
extern float ut_long_mem_floats[MAX_MEM_INDEX];
extern char ut_lat_mem_str[LLA_LAT_BUFFER_SIZE];
extern char ut_long_mem_str[LLA_LONG_BUFFER_SIZE];
void ut_init();

void ut_poll_btns();


#endif /* UTILITIES_H */
```

Index