NEWMAST

Generated by Doxygen 1.7.5.1

Wed Nov 16 2011 13:25:30

Contents

Chapter 1

Main Page

1.1 Introduction

This is a complete rewrite of the original sailcode, the idea is to break the main functions of the boat into reusable libraries. Each library is a class with a bunch of member functions and some data structs.

There should be a library for the Wind Sensor and compass, the Polulu, any ethernet communications and maybe the sailing logic.

Chapter 2

Data Structure Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 WindSense::GPGLL Struct Reference

GPS Data Struct Contains the GPS data from the airmar.

`#include <WindSense.h>`

Data Fields

- int **degreeLatitude**
- int **minuteLatitude**
- char **latitudeDirection**
- int **degreeLongitude**
- int **minuteLongitude**
- char **longitudeDirection**
- char **valid**

4.1.1 Detailed Description

GPS Data Struct Contains the GPS data from the airmar.

Note Lattitude and longitude will be split into two variables each otherwise we lose precision

Definition at line 46 of file WindSense.h.

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

- WindSense.h

4.2 WindSense::GPVTG Struct Reference

Data Fields

- double **courseoverGround**
- char **unitCourseMeasurement**
- double **speedoverGround**
- char **speedUnits**

4.2.1 Detailed Description

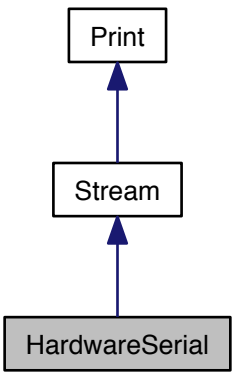
Definition at line 68 of file WindSense.h.

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

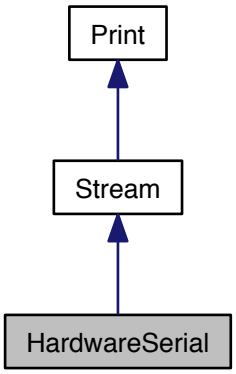
- WindSense.h

4.3 HardwareSerial Class Reference

Inheritance diagram for HardwareSerial:



Collaboration diagram for HardwareSerial:



Public Member Functions

- **HardwareSerial** (ring_buffer ∗rx_buffer, volatile uint8_t ∗ubrrh, volatile uint8_t ∗ubrrl, volatile uint8_t ∗ucsra, volatile uint8_t ∗ucsrb, volatile uint8_t ∗udr, uint8_t rxen, uint8_t txen, uint8_t rxcie, uint8_t udre, uint8_t u2x)
- void **begin** (long)
- void **end** ()
- virtual int **available** (void)
- virtual int **peek** (void)
- virtual int **read** (void)
- virtual void **flush** (void)
- virtual void **write** (uint8_t)

4.3.1 Detailed Description

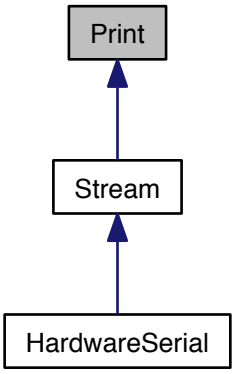
Definition at line 31 of file HardwareSerial.h.

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

- HardwareSerial.h

4.4 Print Class Reference

Inheritance diagram for Print:



Public Member Functions

- virtual void **write** (uint8_t)=0
- virtual void **write** (const char ∗str)
- virtual void **write** (const uint8_t ∗buffer, size_t size)
- void **print** (const String &)
- void **print** (const char[])
- void **print** (char, int=BYTE)
- void **print** (unsigned char, int=BYTE)
- void **print** (int, int=DEC)
- void **print** (unsigned int, int=DEC)
- void **print** (long, int=DEC)
- void **print** (unsigned long, int=DEC)
- void **print** (double, int=2)
- void **println** (const String &s)
- void **println** (const char[])
- void **println** (char, int=BYTE)
- void **println** (unsigned char, int=BYTE)
- void **println** (int, int=DEC)
- void **println** (unsigned int, int=DEC)
- void **println** (long, int=DEC)
- void **println** (unsigned long, int=DEC)
- void **println** (double, int=2)
- void **println** (void)

4.4.1 Detailed Description

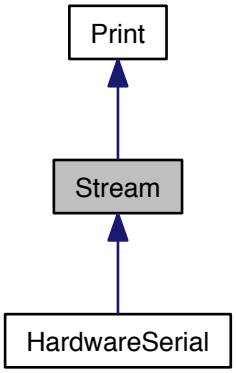
Definition at line 34 of file Print.h.

The documentation for this class was generated from the following files:

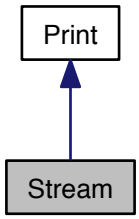
- Print.h
- Print.cpp

4.5 Stream Class Reference

Inheritance diagram for Stream:



Collaboration diagram for Stream:



Public Member Functions

- virtual int **available** ()=0
- virtual int **read** ()=0
- virtual int **peek** ()=0
- virtual void **flush** ()=0

4.5.1 Detailed Description

Definition at line 26 of file Stream.h.

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

- Stream.h

4.6 String Class Reference

Public Member Functions

- **String** (const char *value="")
- **String** (const String &value)
- **String** (const char)
- **String** (const unsigned char)
- **String** (const int, const int base=10)
- **String** (const unsigned int, const int base=10)
- **String** (const long, const int base=10)
- **String** (const unsigned long, const int base=10)
- const String & **operator=** (const String &rhs)

- const String & **operator+=** (const String &rhs)
- int **operator==** (const String &rhs) const
- int **operator!=** (const String &rhs) const
- int **operator**< (const String &rhs) const
- int **operator**> (const String &rhs) const
- int **operator**<**=** (const String &rhs) const
- int **operator**>**=** (const String &rhs) const
- char **operator[]** (unsigned int index) const
- char & **operator[]** (unsigned int index)
- char **charAt** (unsigned int index) const
- int **compareTo** (const String &anotherString) const
- unsigned char **endsWith** (const String &suffix) const
- unsigned char **equals** (const String &anObject) const
- unsigned char **equalsIgnoreCase** (const String &anotherString) const
- int **indexOf** (char ch) const
- int **indexOf** (char ch, unsigned int fromIndex) const
- int **indexOf** (const String &str) const
- int **indexOf** (const String &str, unsigned int fromIndex) const
- int **lastIndexOf** (char ch) const
- int **lastIndexOf** (char ch, unsigned int fromIndex) const
- int **lastIndexOf** (const String &str) const
- int **lastIndexOf** (const String &str, unsigned int fromIndex) const
- const unsigned int **length** () const
- void **setCharAt** (unsigned int index, const char ch)
- unsigned char **startsWith** (const String &prefix) const
- unsigned char **startsWith** (const String &prefix, unsigned int toffset) const
- String **substring** (unsigned int beginIndex) const
- String **substring** (unsigned int beginIndex, unsigned int endIndex) const
- String **toLowerCase** () const
- String **toUpperCase** () const
- String **trim** () const
- void **getBytes** (unsigned char ∗buf, unsigned int bufsize)
- void **toCharArray** (char ∗buf, unsigned int bufsize)
- long **toInt** ()
- const String & **concat** (const String &str)
- String **replace** (char oldChar, char newChar)
- String **replace** (const String &match, const String &replace)

Protected Member Functions

- void **getBuffer** (unsigned int maxStrLen)

Protected Attributes

- char ∗ **_buffer**
- unsigned int **_capacity**
- unsigned int **_length**

Friends

- String **operator+** (String lhs, const String &rhs)

4.6.1 Detailed Description

Definition at line 28 of file WString.h.

The documentation for this class was generated from the following files:

- WString.h
- WString.cpp

4.7 WindSense::WIMWV Struct Reference

Data Fields

- double **windAngle**
- char **reference**
- double **windSpeed**
- char **windSpeedUnits**
- char **valid**

4.7.1 Detailed Description

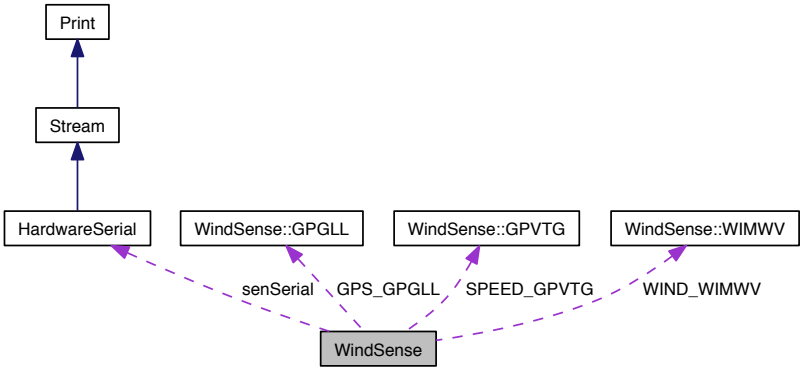
Definition at line 58 of file WindSense.h.

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

- WindSense.h

4.8 WindSense Class Reference

Collaboration diagram for WindSense:



Data Structures

- struct GPGLL

  GPS Data Struct Contains the GPS data from the airmar.
- struct GPVTG
- struct WIMWV

Public Member Functions

- WindSense ()
- **WindSense** (HardwareSerial ∗inSerial)
- int grabChar (char input)

  Adds a character to the partial sentence string.
- int validateNMEA (char ∗input)

  Returns true if the checksum matches at the end.
- int **validateInternalNMEA** ()
- int splitNMEA (char ∗input)

  Returns an array of strings (char arrays) containing parsed values.
- int **splitInternalNMEA** ()
- void **resetInternalNMEA** ()
- int parseInternalNMEA (char ∗input)

  Contains all of the parsing functions parseToStruct.cpp.
- int updateGPS_GPGLL ()

  Update the GPS Data Struct by parsing.

- int updateWIND_WIMWV ()

  Update the Wind Data Struct by Parsing.
- int updateSPEED_GPVTG ()

  Update the Speet Data Struct by Parsing.
- int debug (HardwareSerial &debugPortIn)

  Stuff related purely to out wind sensor set-up.
- int debugDump (HardwareSerial &debugPortIn)

  Dumps a lot of info to the computers serial line.

Data Fields

- int partCount

  index for partSentence
- char partSentence [100]

  buffer for incoming NMEA for grabChar function
- char stringArray [40][15]

  Array of strings for splitNMEA.
- int stringArrayIdx

  index for the array of strings
- GPGLL **GPS_GPGLL**
- WIMWV **WIND_WIMWV**
- GPVTG **SPEED_GPVTG**
- HardwareSerial * **senSerial**

4.8.1 Detailed Description

Definition at line 13 of file WindSense.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 WindSense::WindSense ( )

< index for partSentence

< index for the array of strings

Definition at line 3 of file WindSense.cpp.

```
                    {
    partCount = 0;
    stringArrayIdx = 0;
}
```

4.8.3 Member Function Documentation

4.8.3.1 int WindSense::debug (HardwareSerial & *debugPortIn*)

Stuff related purely to out wind sensor set-up.

Things like the initialisation after instantiation of the object, and some debugging function stuff. Waits for NMEA, tells you everything

This function takes a serial port to use for debugging the functions and listens for an NMEA sentence. Then it runs through them with the validation, splitting and parsing functions. It spits out output the entire way.

For this function to work properly, it must be called within a loop. If this were the only line to be executed in the main loop, it would work as expected.

The Serial port given to this function must have already had the baud rate set

Definition at line 22 of file AIRMARSpecific.cpp.

```
                                                {
    HardwareSerial* debugPort = &debugPortIn;

    while(debugPort->available()) {
        if (grabChar(debugPort->read())) {
            debugDump(*debugPort);
        }
    }

    return 1;
}
```

Here is the call graph for this function:

4.8.3.2 int WindSense::debugDump (HardwareSerial & *debugPortIn*)

Dumps a lot of info to the computers serial line.

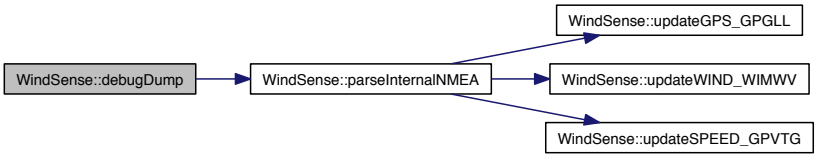
Would have been incorporated directly into the debug function, but it just takes up too much space. In the event the flash image for the arduino becomes too large be sure to get rid of this function.

Resets the Internal NMEA Index counters

Definition at line 43 of file AIRMARSpecific.cpp.

```
                                                    {
    HardwareSerial* debugPort = &debugPortIn;

    // Information about what is in the partSentence Buffer
    debugPort->println("NMEA Detected");
    debugPort->println(partSentence);
    debugPort->print("Valid?...");
    debugPort->println(validateInternalNMEA());

    // Separating the NMEA into sub-strings
    splitInternalNMEA();
    for (int i = 0; i < stringArrayIdx; i++) {
        debugPort->print("String ");
        debugPort->print(i);
        debugPort->print(" ");
        debugPort->println(stringArray[i]);
    }

    parseInternalNMEA(stringArray[0]);

    // Dump the degrees and minutes to see if it worked
    debugPort->print("The minutes lattitude are ");
    debugPort->println(GPS_GPGLL.minuteLatitude);
    debugPort->print("The degrees lattitude are ");
    debugPort->println(GPS_GPGLL.degreeLatitude);
    debugPort->print("Lattitude Direction is ");
    debugPort->println(GPS_GPGLL.latitudeDirection);

    debugPort->print("The minutes longitude are ");
    debugPort->println(GPS_GPGLL.minuteLongitude);
    debugPort->print("The degrees longitude are ");
    debugPort->println(GPS_GPGLL.degreeLongitude);
    debugPort->print("longitude Direction is ");
    debugPort->println(GPS_GPGLL.longitudeDirection);

    debugPort->print("Wind Speed is ");
    debugPort->println(WIND_WIMWV.windSpeed);
    debugPort->print("Units for the speed ");
    debugPort->println(WIND_WIMWV.windSpeedUnits);
    debugPort->print("The angle is ");
    debugPort->println(WIND_WIMWV.windAngle);
    debugPort->print("The reference for that angle ");
    debugPort->println(WIND_WIMWV.reference);

    debugPort->print("Speed over ground ");
    debugPort->println(SPEED_GPVTG.speedoverGround);
    debugPort->print("Units ");
    debugPort->println(SPEED_GPVTG.speedUnits);
    debugPort->print("Course over ground ");
    debugPort->println(SPEED_GPVTG.courseoverGround);
    debugPort->print("Units ");
    debugPort->println(SPEED_GPVTG.unitCourseMeasurement);

    resetInternalNMEA();
    return 1;
}
```

Here is the call graph for this function:



4.8.3.3 int WindSense::grabChar (char *input*)

Adds a character to the partial sentence string.

As long as the string has already started being built, or if a new $ character has been found. There a number of cases in which this can fail and need to be handled.

Note

Requires the external variables, partSentence and partCount

Returns

Whether or not a complete NMEA sentence has been stored, 0 if it hasn't 1 if it has.

Definition at line 25 of file WindSense.cpp.

```
                                {
    if (partCount > 0 || input == '$') {
        partSentence[partCount] = input;
        partCount++;
    }

    if (partCount > 3 && partSentence[partCount-3] == '*') {
        partSentence[partCount] = '\0';
        return 1;
    }
    return 0;
}
```

4.8.3.4 int WindSense::parseInternalNMEA (char * *input*)

Contains all of the parsing functions parseToStruct.cpp.

Due to the large number of functions used to parse char* instanced into proper types in the structs declared in the header file, this has been created.

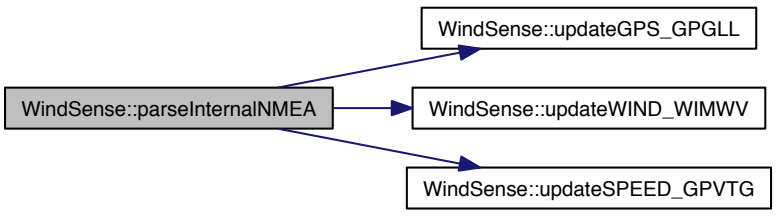
It simply contains all of the functions to update the values in the structs.

Created on: 2011-11-14 Author: allgood38

Definition at line 16 of file ParseToStruct.cpp.

```
                                                {

    if (strcmp(input,"GPGLL") == 0) {
        //  updateGPS_GPGLL
        updateGPS_GPGLL();
    } else if (strcmp(input, "WIMWV") == 0) {
        // Update Something else
        updateWIND_WIMWV();
    } else if (strcmp(input, "GPVTG") == 0) {
        updateSPEED_GPVTG();
    } else {
        // TODO Handle parse without valid
        // NMEA label
    }

    return 1;
}
```

Here is the call graph for this function:



4.8.3.5 int WindSense::splitNMEA (char * *input*)

Returns an array of strings (char arrays) containing parsed values.

This function relies on three different strings. Each string is just an array of characters, so each string needs an index variables to keep track of our progress within the string.

The first string is simply the valid input sentence. The stringArray is to store all the values from the input string. The newValue string is a temporary buffer to hold on to a new value before it is stored in the string array.

The loop will place each input character into the new value string until a comma or asterisk is encountered. Then it stores it to the array of strings, resets the newValue and increments the index for stringArray.

If there is a situation where there are two commas, indicating a null value from the NMEA sentence, a null character will be stored in the string array, rather than just skipping

over it.

Definition at line 97 of file WindSense.cpp.

```
                                    {
    int inputIdx = 1;

    char newValue[15];
    int newValueIdx = 0;

    // For an explanation of the conditions of this loop, see the pdf
    // documentation.
    while(inputIdx <= 1 || input[inputIdx - 1] != '*') {

        if (input[inputIdx] == ',' || input[inputIdx] == '*') {
            for (int i = 0; i < newValueIdx; i++) {
                stringArray[stringArrayIdx][i] = newValue[i];
            }

            stringArray[stringArrayIdx++][newValueIdx] = '\0';
            newValueIdx=0;

            // No comma, then store the value in the newValue array
        } else {
            newValue[newValueIdx++] = input[inputIdx];
            newValue[newValueIdx] = '\0';
        }

        // Move to the next character in the input string
        inputIdx++;
    }

    return 1;
}
```

4.8.3.6 int WindSense::updateGPS_GPGLL ( )

Update the GPS Data Struct by parsing.

Given that the format of the latitude and longitude are given in a form which exceeds the accuracy of our Arduino, we need to break it into minutes and degrees into separate integers.

[GPS_GPGLL] All data fields

Definition at line 43 of file ParseToStruct.cpp.

```
                                {
    char* pEnd;
    char minutes[10];
    char degrees[10];

    /* The AIRMAR Actually reports invalid data, check it
     * here and abort with 0 if invalid, not doing this
     * will definitely crash the for loops
     */
    if (stringArray[6][0] == 'V') {
        return 0;
    }
```

```
    // Since there will be parsing, an index will be used
    int idx = 0;

    /* The following two loops parse the latitude, uses
     * the period as a separator between the minutes
     * and degrees
     */
    for (int i = 0; stringArray[1][idx] != '.'; i++) {
        minutes[i] = stringArray[1][idx++];
        // Always keeping a null at the end
        minutes[i+1] = '\0';
    }
    // Skip the period
    idx++;
    for (int i = 0; stringArray[1][idx] != '\0'; i++) {
        degrees[i] = stringArray[1][idx++];
        degrees[i+1] = '\0';
    }

    // Extract int from the isolated strings
    GPS_GPGLL.minuteLatitude = (int)strtol(minutes,&pEnd,10);
    GPS_GPGLL.degreeLatitude = (int)strtol(degrees,&pEnd,10);

    idx=0;
    for (int i = 0; stringArray[3][idx] != '.'; i++) {
        minutes[i] = stringArray[3][idx++];
        minutes[i+1] = '\0';
    }
    idx++;
    for (int i = 0; stringArray[3][idx] != '\0'; i++) {
        degrees[i] = stringArray[3][idx++];
        degrees[i+1] = '\0';
    }

    GPS_GPGLL.minuteLongitude = (int)strtol(minutes,&pEnd,10);
    GPS_GPGLL.degreeLongitude = (int)strtol(degrees,&pEnd,10);

    GPS_GPGLL.latitudeDirection = stringArray[2][0];
    GPS_GPGLL.longitudeDirection = stringArray[4][0];

    return 1;
}
```

4.8.3.7 int WindSense::updateSPEED_GPVTG ( )

Update the Speet Data Struct by Parsing.

Converts strings into the correct types.

[SPEED_GPVTG] All Data Fields

Definition at line 126 of file ParseToStruct.cpp.

```
                                {
    // Check if the AIRMAR says the data is valid
    if (stringArray[9][0] == 'N') {
        return 0;
    }
```

```
    SPEED_GPVTG.courseoverGround      = strtod(stringArray[1], '\0');
    SPEED_GPVTG.unitCourseMeasurement = stringArray[2][0];
    SPEED_GPVTG.speedoverGround       = strtod(stringArray[5], '\0');
    SPEED_GPVTG.speedUnits            = stringArray[6][0];

    return 1;
}
```

4.8.3.8 int WindSense::updateWIND_WIMWV ( )

Update the Wind Data Struct by Parsing.

Simply converts the strings into the correct types.

[WIND_WIMWV] All Data Fields

Definition at line 105 of file ParseToStruct.cpp.

```
                              {
    // Check if the AIRMAR says the data is valid
    if (stringArray[5][0] == 'V') {
        return 0;
    }

    WIND_WIMWV.windAngle      = strtod(stringArray[1],'\0');
    WIND_WIMWV.reference      = stringArray[2][0];
    WIND_WIMWV.windSpeed      = strtod(stringArray[3],'\0');
    WIND_WIMWV.windSpeedUnits = stringArray[4][0];
    WIND_WIMWV.valid          = stringArray[5][0];

    return 1;
}
```

4.8.3.9 int WindSense::validateNMEA (char * *input*)

Returns true if the checksum matches at the end.

Note this function assumes that $ will always be the first character, and that the entire NMEA sentence is untouched and isolated. It also requires that partcount not be reset, since it uses it to determine the end of the sentence

Note

Requires the external variables, partSentence and partCount

Parameters

in	*input*	The complete and isolated NMEA sentence as a array

Returns

The checksum is valid, 1 is True, 0 is False.

Definition at line 50 of file WindSense.cpp.

```
                                        {
    int calculatedChecksum = 0;
    char rawGivenChecksum[2];
    int givenChecksum = 0;

    // XOR every element between [1] and *
    for (int i = 1; partSentence[i] != '*'; i++) {
        calculatedChecksum ^= partSentence[i];
    }

    // Convert the last two characters into a hexadecimal number
    rawGivenChecksum[0] = partSentence[partCount-2];
    rawGivenChecksum[1] = partSentence[partCount-1];

    char* parserChar;
    givenChecksum = (int)strtol(rawGivenChecksum, &parserChar, 16);

    // Another possible method
    //sscanf(rawGivenChecksum, "%x", &givenChecksum);

    partCount = 0;
    return (givenChecksum == calculatedChecksum);
}
```

The documentation for this class was generated from the following files:

- WindSense.h
- AIRMARSpecific.cpp
- ParseToStruct.cpp
- WindSense.cpp