

Track Sense

Generated by Doxygen 1.9.8



<b>1 Class Documentation</b>	<b>1</b>
1.1 TrackSense::GPSData Struct Reference	1
1.1.1 Detailed Description	2
1.1.2 Member Data Documentation	2
1.1.2.1 alt	2
1.1.2.2 hdop	2
1.1.2.3 lat	2
1.1.2.4 lon	2
1.1.2.5 sat	2
1.1.2.6 time_utc	2
1.1.2.7 vel	2
1.2 GPSSim Class Reference	3
1.2.1 Detailed Description	4
1.2.2 Constructor & Destructor Documentation	5
1.2.2.1 GPSSim()	5
1.2.2.2 ~GPSSim()	5
1.2.3 Member Function Documentation	5
1.2.3.1 _gerar_corpo_de_frase_gga()	5
1.2.3.2 _gerar_corpo_de_frase_rmc()	7
1.2.3.3 _loop()	8
1.2.3.4 _update_position()	8
1.2.3.5 config_traj()	9
1.2.3.6 converter_graus_para_nmea()	9
1.2.3.7 finalizar_corpo_de_frase()	9
1.2.3.8 formatar_inteiro()	10
1.2.3.9 init()	11
1.2.3.10 obter_caminho_terminal_filho()	11
1.2.3.11 obter_tempo_utc()	12
1.2.3.12 stop()	12
1.2.4 Member Data Documentation	13
1.2.4.1 _alt	13
1.2.4.2 _fd_filho	13
1.2.4.3 _fd_pai	13
1.2.4.4 _is_exec	13
1.2.4.5 _lat	13
1.2.4.6 _lat_sim	13
1.2.4.7 _lon	13
1.2.4.8 _lon_sim	14
1.2.4.9 _mov_circ	14
1.2.4.10 _nome_pt_filho	14
1.2.4.11 _periodo_atualizacao	14
1.2.4.12 _periodo_circ	14

1.2.4.13	_raio	14
1.2.4.14	_start_time	14
1.2.4.15	_thread_geradora	14
1.2.4.16	_velocidade_nos	14
1.3	TrackSense Class Reference	15
1.3.1	Detailed Description	16
1.3.2	Constructor & Destructor Documentation	16
1.3.2.1	TrackSense()	16
1.3.2.2	~TrackSense()	17
1.3.3	Member Function Documentation	17
1.3.3.1	_ler_dados()	17
1.3.3.2	_loop()	18
1.3.3.3	_open_serial()	18
1.3.3.4	_parser_gga()	19
1.3.3.5	_parser_rmc()	19
1.3.3.6	converter_lat_long()	20
1.3.3.7	formatar_csv()	20
1.3.3.8	init()	21
1.3.3.9	split()	21
1.3.3.10	stop()	22
1.3.4	Member Data Documentation	22
1.3.4.1	_addr_dest	22
1.3.4.2	_fd_serial	22
1.3.4.3	_ip_destino	22
1.3.4.4	_is_exec	22
1.3.4.5	_last_data_given	23
1.3.4.6	_porta_destino	23
1.3.4.7	_porta_serial	23
1.3.4.8	_sockfd	23
1.3.4.9	_worker	23
2	File Documentation	25
2.1	src/debug.cpp File Reference	25
2.1.1	Detailed Description	25
2.1.2	Function Documentation	26
2.1.2.1	main()	26
2.2	src/GPSSim.hpp File Reference	26
2.2.1	Detailed Description	27
2.3	GPSSim.hpp	27
2.4	src/TrackSense.hpp File Reference	32
2.4.1	Detailed Description	33
2.5	TrackSense.hpp	33





# Chapter 1

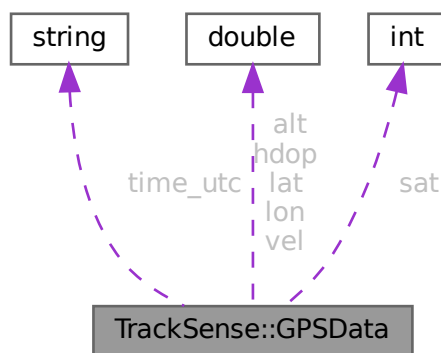
## Class Documentation

### 1.1 TrackSense::GPSData Struct Reference

Struct para armazenar dados GPS

```
#include <TrackSense.hpp>
```

Collaboration diagram for TrackSense::GPSData:



#### Public Attributes

- `std::string` `time_utc`
- `double` `lat`
- `double` `lon`
- `double` `vel`
- `double` `alt`
- `int` `sat`
- `double` `hdop`

### 1.1.1 Detailed Description

Struct para armazenar dados GPS

### 1.1.2 Member Data Documentation

#### 1.1.2.1 alt

```
double TrackSense::GPSData::alt
```

#### 1.1.2.2 hdop

```
double TrackSense::GPSData::hdop
```

#### 1.1.2.3 lat

```
double TrackSense::GPSData::lat
```

#### 1.1.2.4 lon

```
double TrackSense::GPSData::lon
```

#### 1.1.2.5 sat

```
int TrackSense::GPSData::sat
```

#### 1.1.2.6 time\_utc

```
std::string TrackSense::GPSData::time_utc
```

#### 1.1.2.7 vel

```
double TrackSense::GPSData::vel
```

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

- [src/TrackSense.hpp](#)

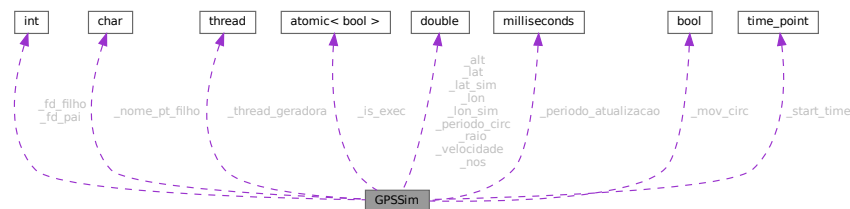


## 1.2 GPSSim Class Reference

Simulador do módulo GPS que gera frases no padrão NMEA.

```
#include <GPSSim.hpp>
```

Collaboration diagram for GPSSim:



### Public Member Functions

- **GPSSim** (double latitude\_inicial\_graus, double longitude\_inicial\_graus, double altitude\_metros=10.0, double frequencia\_atualizacao\_hz=1.0, double velocidade\_nos=10.0)  
*Construtor do GPSSim.*
- **~GPSSim** ()  
*Destrutor do GPSSim.*
- std::string **obter\_caminho\_terminal\_filho** () const  
*Obtém o caminho do terminal que estamos executando de forma filial.*
- void **init** ()  
*Inicia a geração de frases no padrão NMEA em uma thread separada.*
- void **stop** ()  
*Encerrará a geração de frases NMEA e aguardará a finalização da thread.*
- void **config\_traj** (double raio\_metros=20.0, double periodo\_segundos=120.0)  
*Configura a trajetória circular para a simulação.*

### Private Member Functions

- std::string **\_gerar\_corpo\_de\_frase\_rmc** (double lat\_graus, double lon\_graus, double velocidade\_nos)  
*Gera uma frase RMC (Recommended Minimum Specific GNSS Data)*
- std::string **\_gerar\_corpo\_de\_frase\_gga** (double lat\_graus, double lon\_graus, double alt\_metros, int sat=8, double hdop=0.9)  
*Gera uma frase GGA (Global Positioning System Fix Data)*
- void **\_update\_position** ()  
*Caso seja movimento circular, atualizará a posição.*
- void **\_loop** ()  
*Loop principal de geração de dados.*

### Static Private Member Functions

- static std::string [formatar\\_inteiro](#) (int valor, int quant\_digitos)  
*Formatada um número inteiro com dois dígitos, preenchendo com zero à esquerda.*
- static void [converter\\_graus\\_para\\_nmea](#) (double graus\_decimais, bool is\_lat, std::string &ddmm, char &hemisf)  
*Converte graus decimais para formato NMEA de localização, (ddmm.mmmm).*
- static std::string [finalizar\\_corpo\\_de\\_frase](#) (const std::string &corpo\_frase)  
*Após calcular a paridade do corpo da frase, adiciona as flags para o padrão NMEA.*
- static std::tm [obter\\_tempo\\_utc](#) ()  
*Obtém o tempo UTC atual, horário em Londres.*

### Private Attributes

- int [\\_fd\\_pai](#) {0}
- int [\\_fd\\_filho](#) {0}
- char [\\_nome\\_pt\\_filho](#) [128]
- std::thread [\\_thread\\_geradora](#)
- std::atomic< bool > [\\_is\\_exec](#) {false}
- double [\\_lat](#)
- double [\\_lon](#)
- double [\\_alt](#)
- double [\\_velocidade\\_nos](#)
- std::chrono::milliseconds [\\_periodo\\_atualizacao](#)
- bool [\\_mov\\_circ](#) = false
- double [\\_raio](#) {20.0}
- double [\\_periodo\\_circ](#) {20.0}
- double [\\_lat\\_sim](#) {0.0}
- double [\\_lon\\_sim](#) {0.0}
- std::chrono::steady\_clock::time\_point [\\_start\\_time](#) = std::chrono::steady\_clock::now()

## 1.2.1 Detailed Description

Simulador do módulo GPS que gera frases no padrão NMEA.

Cria um par de pseudo-terminais (PTY) para simular um o módulo GPS real. Gera frases no padrão NMEA (RMC e GGA) em intervalos regulares, permitindo configuração de posição inicial, altitude, velocidade e padrão de movimento.

Perceba que o objeto é simular os dados gerados, logo estes devem ser difíceis

## 1.2.2 Constructor & Destructor Documentation

### 1.2.2.1 GPSSim()

```
GPSSim::GPSSim (
    double latitude_inicial_graus,
    double longitude_inicial_graus,
    double altitude_metros = 10.0,
    double frequencia_atualizacao_hz = 1.0,
    double velocidade_nos = 10.0 ) [inline], [explicit]
```

Construtor do [GPSSim](#).

Inicializa alguns parâmetros de posição simulada e, criando os pseudo-terminais, configura-os para o padrão do módulo real.

Utilizou-se o termo `explicit` para impedir que futuras conversões implícitas sejam impedidas. Além disso, foi pensado em utilizar o padrão de `Singleton` para manter apenas uma instância da classe. Entretanto, após outras reuniões, percebeu-se a falta de necessidade.

#### Parameters

<i>latitude_inicial_graus</i>	Latitude inicial em graus decimais
<i>longitude_inicial_graus</i>	Longitude inicial em graus decimais
<i>altitude_metros</i>	Altitude inicial em metros, setada para 10.
<i>frequencia_atualizacao_hz</i>	Frequência de atualização das frases NMEA em Hertz, setada para 1
<i>velocidade_nos</i>	Velocidade sobre o fundo em nós (para frase RMC), setada para 0

### 1.2.2.2 ~GPSSim()

```
GPSSim::~~GPSSim ( ) [inline]
```

Destrutor do [GPSSim](#).

Chama a função `stop()` e, após verificar existência de terminal `Pai`, fecha-o. Here is the call graph for this function:



## 1.2.3 Member Function Documentation

### 1.2.3.1 \_gerar\_corpo\_de\_frase\_gga()

```
std::string GPSSim::_gerar_corpo_de_frase_gga (
    double lat_graus,
```

```
double lon_graus,  
double alt_metros,  
int sat = 8,  
double hdop = 0.9 ) [inline], [private]
```

Gera uma frase GGA (Global Positioning System Fix Data)

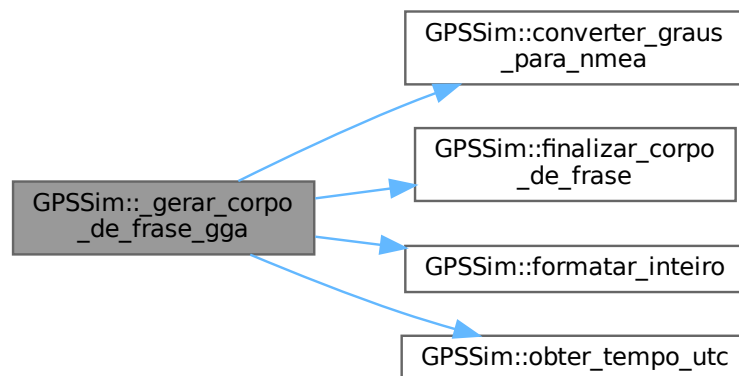
#### Parameters

<i>lat_graus</i>	Latitude em graus decimais
<i>lon_graus</i>	Longitude em graus decimais
<i>alt_metros</i>	Altitude em metros
<i>sat</i>	Número de satélites visíveis
<i>hdop</i>	Horizontal Dilution of Precision

#### Returns

String correspondendo ao corpo de frase GGA

Here is the call graph for this function:



Here is the caller graph for this function:



### 1.2.3.2 \_gerar\_corpo\_de\_frase\_rmc()

```
std::string GPSSim::_gerar_corpo_de_frase_rmc (
    double lat_graus,
    double lon_graus,
    double velocidade_nos ) [inline], [private]
```

Gera uma frase RMC (Recommended Minimum Specific GNSS Data)

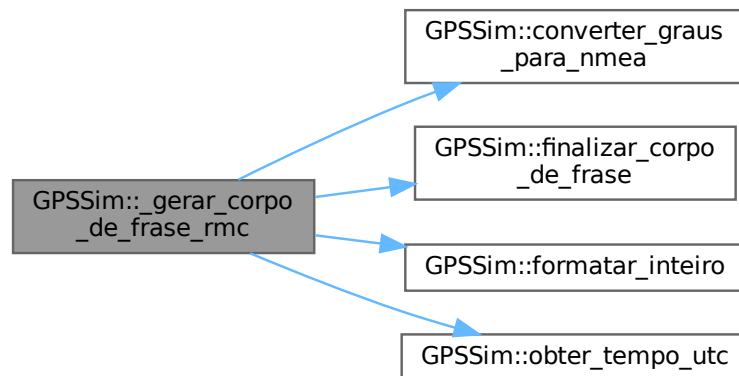
#### Parameters

<i>lat_graus</i>	Latitude em graus decimais
<i>lon_graus</i>	Longitude em graus decimais
<i>velocidade_nos</i>	Velocidade em nós

#### Returns

String correspondendo ao corpo de Frase RMC

Here is the call graph for this function:



Here is the caller graph for this function:

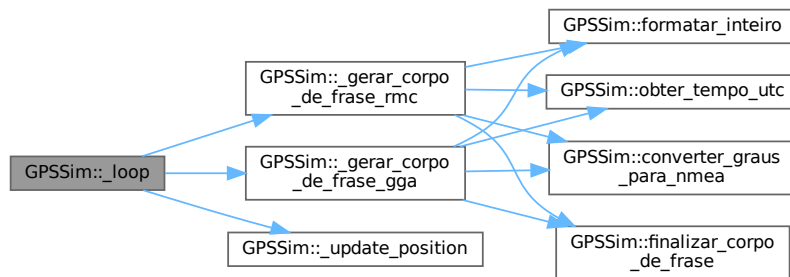


### 1.2.3.3 \_loop()

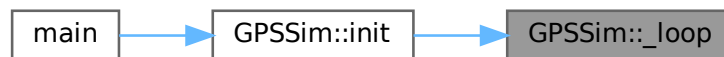
```
void GPSSim::_loop ( ) [inline], [private]
```

Loop principal de geração de dados.

Here is the call graph for this function:



Here is the caller graph for this function:



### 1.2.3.4 \_update\_position()

```
void GPSSim::_update_position ( ) [inline], [private]
```

Caso seja movimento circular, atualizará a posição.

Here is the caller graph for this function:



### 1.2.3.5 config\_traj()

```
void GPSSim::config_traj (
    double raio_metros = 20.0,
    double periodo_segundos = 120.0 ) [inline]
```

Configura a trajetória circular para a simulação.

Here is the caller graph for this function:



### 1.2.3.6 converter\_graus\_para\_nmea()

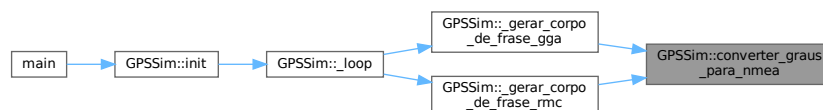
```
static void GPSSim::converter_graus_para_nmea (
    double graus_decimais,
    bool is_lat,
    std::string & ddmm,
    char & hemisf ) [inline], [static], [private]
```

Converte graus decimais para formato NMEA de localização, (ddmm.mmmm).

#### Parameters

	<i>graus_decimais</i>	Valor em graus decimais
	<i>is_lat</i>	Flag de eixo
out	<i>ddmm</i>	String com valor formatado em graus e minutos
out	<i>hemisf</i>	Caractere indicando Hemisfério

Here is the caller graph for this function:



### 1.2.3.7 finalizar\_corpo\_de\_frase()

```
static std::string GPSSim::finalizar_corpo_de_frase (
```

```
const std::string & corpo_frase ) [inline], [static], [private]
```

Após calcular a paridade do corpo da frase, adiciona as flags para o padrão NMEA.

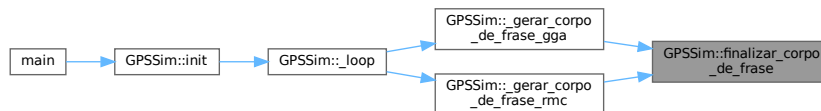
#### Parameters

<i>corpo_frase</i>	Frase sem os indicadores \$ e *
--------------------	---------------------------------

#### Returns

String contendo a frase completa, com todas as informações e flags.

Here is the caller graph for this function:



#### 1.2.3.8 formatar\_inteiro()

```
static std::string GPSSim::formatar_inteiro (
    int valor,
    int quant_digitos ) [inline], [static], [private]
```

Formatada um número inteiro com dois dígitos, preenchendo com zero à esquerda.

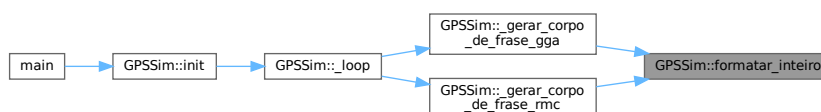
#### Parameters

<i>valor</i>	Número a ser formatado
<i>quant_digitos</i>	Quantidade de Dígitos presente

#### Returns

string formatada

Here is the caller graph for this function:





### 1.2.3.9 init()

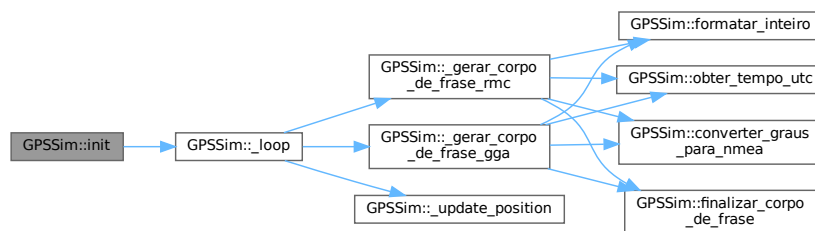
```
void GPSSim::init ( ) [inline]
```

Inicia a geração de frases no padrão NMEA em uma thread separada.

Garante a criação de apenas uma thread utilizando uma variável atômica. O padrão NMEA é o protocolo padrão usado por módulos GPS, cada mensagem começa com \$ e termina com \r\n. Exemplo:

- \$origem,codificacao\_usada,dados1,dados2,...\*paridade

Here is the call graph for this function:



Here is the caller graph for this function:



### 1.2.3.10 obter\_caminho\_terminal\_filho()

```
std::string GPSSim::obter_caminho_terminal_filho ( ) const [inline]
```

Obtém o caminho do terminal que estamos executando de forma filial.

**Returns**

string correspondendo ao caminho do dispositivo.

Here is the caller graph for this function:

**1.2.3.11 obter\_tempo\_utc()**

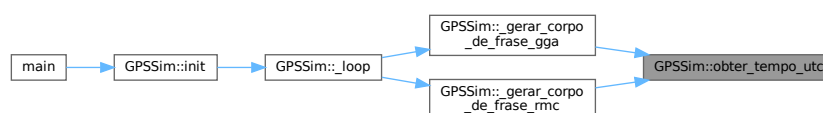
```
static std::tm GPSSim::obter_tempo_utc ( ) [inline], [static], [private]
```

Obtém o tempo UTC atual, horário em Londres.

**Returns**

Struct `std::tm` contendo o tempo em UTC

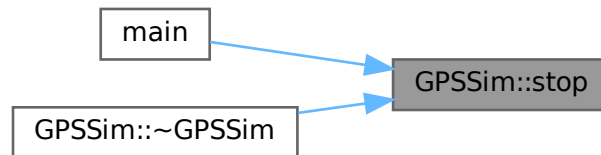
Here is the caller graph for this function:

**1.2.3.12 stop()**

```
void GPSSim::stop ( ) [inline]
```

Encerrará a geração de frases NMEA e aguardará a finalização da thread.

Verifica a execução da thread e encerra-a caso exista. Força a finalização da thread geradora de mensagens. Here is the caller graph for this function:



## 1.2.4 Member Data Documentation

### 1.2.4.1 `_alt`

```
double GPSSim::_alt [private]
```

### 1.2.4.2 `_fd_filho`

```
int GPSSim::_fd_filho {0} [private]
```

### 1.2.4.3 `_fd_pai`

```
int GPSSim::_fd_pai {0} [private]
```

### 1.2.4.4 `_is_exec`

```
std::atomic<bool> GPSSim::_is_exec {false} [private]
```

### 1.2.4.5 `_lat`

```
double GPSSim::_lat [private]
```

### 1.2.4.6 `_lat_sim`

```
double GPSSim::_lat_sim {0.0} [private]
```

### 1.2.4.7 `_lon`

```
double GPSSim::_lon [private]
```

#### 1.2.4.8 `_lon_sim`

```
double GPSSim::_lon_sim {0.0} [private]
```

#### 1.2.4.9 `_mov_circ`

```
bool GPSSim::_mov_circ = false [private]
```

#### 1.2.4.10 `_nome_pt_filho`

```
char GPSSim::_nome_pt_filho[128] [private]
```

#### 1.2.4.11 `_periodo_atualizacao`

```
std::chrono::milliseconds GPSSim::_periodo_atualizacao [private]
```

#### 1.2.4.12 `_periodo_circ`

```
double GPSSim::_periodo_circ {20.0} [private]
```

#### 1.2.4.13 `_raio`

```
double GPSSim::_raio {20.0} [private]
```

#### 1.2.4.14 `_start_time`

```
std::chrono::steady_clock::time_point GPSSim::_start_time = std::chrono::steady_clock::now()  
[private]
```

#### 1.2.4.15 `_thread_geradora`

```
std::thread GPSSim::_thread_geradora [private]
```

#### 1.2.4.16 `_velocidade_nos`

```
double GPSSim::_velocidade_nos [private]
```

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

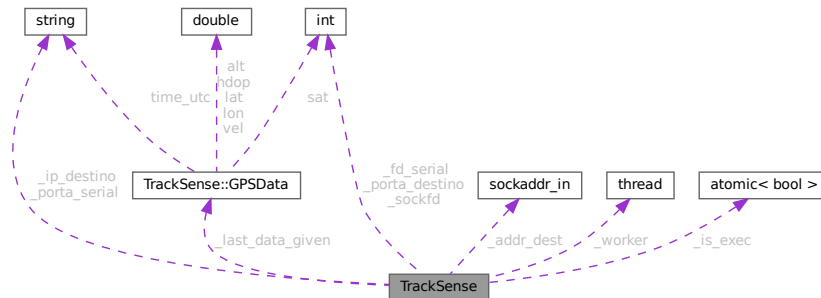
- [src/GPSSim.hpp](#)

## 1.3 TrackSense Class Reference

Representa um sensor GPS (GY-GPS6MV2 ou Simulado).

```
#include <TrackSense.hpp>
```

Collaboration diagram for TrackSense:



### Classes

- struct [GPSData](#)  
*Struct para armazenar dados GPS*

### Public Member Functions

- [TrackSense](#) (const std::string &ip\_destino, int porta\_destino, const std::string &porta\_serial="")  
*Construtor da Classe.*
- [~TrackSense](#) ()  
*Destrutor da Classe.*
- void [init](#) ()  
*Inicia thread de leitura de dados do GPS e envio UDP. Há garantia de instância única.*
- void [stop](#) ()  
*Finaliza a thread de trabalho.*

### Private Member Functions

- void [\\_open\\_serial](#) ()  
*Abre e configura a porta serial do GPS, possibilitando a leitura direta.*
- std::string [\\_ler\\_dados](#) ()  
*Responsável por ler os dados emitos na porta serial.*
- void [\\_parser\\_rmc](#) (const std::string &frase, [GPSData](#) &data)  
*Interpretará os dados caso venham no padrão RMC.*
- void [\\_parser\\_gga](#) (const std::string &frase, [GPSData](#) &data)  
*Interpretará os dados caso venham no padrão GGA*
- void [\\_loop](#) ()  
*Responsável pelo loop principal de leitura, interpretação e envio via UDP.*

## Static Private Member Functions

- static `std::vector< std::string > split` (const `std::string` &`string_de_entrada`, char `separador`)  
*Função estática auxiliar para separar string em vetores de string. Similar ao método `split` do python.*
- static double `converter_lat_long` (const `std::string` &`valor`, const `std::string` &`hemisf`)  
*Função estática auxiliar para converter latitude e longitude para graus decimais.*
- static `std::string formatar_csv` (const `GPSTData` &`data`)  
*Função estática auxiliar para formar string .csv.*

## Private Attributes

- `std::string _ip_destino`
- `int _porta_destino`
- `int _sockfd`
- `sockaddr_in _addr_dest`
- `std::thread _worker`
- `std::atomic< bool > _is_exec` {false}
- `GPSTData _last_data_given` {}
- `std::string _porta_serial`
- `int _fd_serial` = -1

### 1.3.1 Detailed Description

Representa um sensor GPS (GY-GPS6MV2 ou Simulado).

Responsável por:

- Obter informações GNSS reais ou simuladas
- Interpretar e armazenar esses dados
- Enviar os dados via socket UDP em formato CSV

Para atender essas necessidades, foi implementada diversas funções independentes em thread e com formas inteligentes de organização.

### 1.3.2 Constructor & Destructor Documentation

#### 1.3.2.1 TrackSense()

```
TrackSense::TrackSense (
    const std::string & ip_destino,
    int porta_destino,
    const std::string & porta_serial = "" ) [inline]
```

Construtor da Classe.

#### Parameters

<code>ip_destino</code>	Endereço IP de destino.
<code>porta_destino</code>	Porta UDP de destino
<code>porta_serial</code>	Caminho da porta serial

Inicializa ferramentas de conexão socket UDP. Here is the call graph for this function:

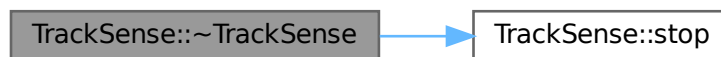


### 1.3.2.2 ~TrackSense()

```
TrackSense::~~TrackSense ( ) [inline]
```

Destrutor da Classe.

Interrompe thread e finaliza socket Here is the call graph for this function:



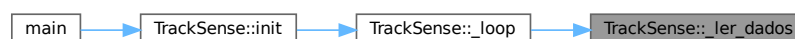
## 1.3.3 Member Function Documentation

### 1.3.3.1 \_ler\_dados()

```
std::string TrackSense::_ler_dados ( ) [inline], [private]
```

Responsável por ler os dados emitos na porta serial.

Utiliza um buffer de 256 caracteres. Here is the caller graph for this function:

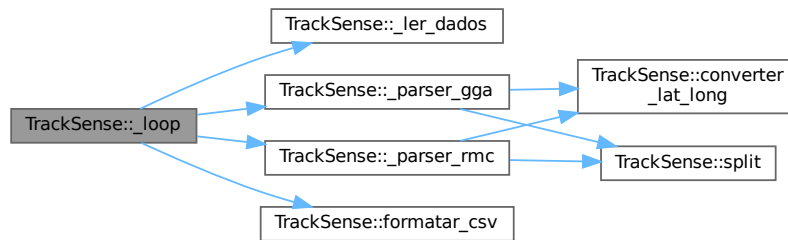


### 1.3.3.2 \_loop()

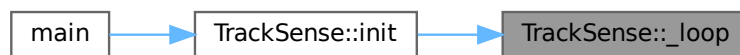
```
void TrackSense::_loop ( ) [inline], [private]
```

Responsável pelo loop principal de leitura, interpretação e envio via UDP.

Here is the call graph for this function:



Here is the caller graph for this function:



### 1.3.3.3 \_open\_serial()

```
void TrackSense::_open_serial ( ) [inline], [private]
```

Abre e configura a porta serial do GPS, possibilitando a leitura direta.

Here is the caller graph for this function:





### 1.3.3.4 \_parser\_gga()

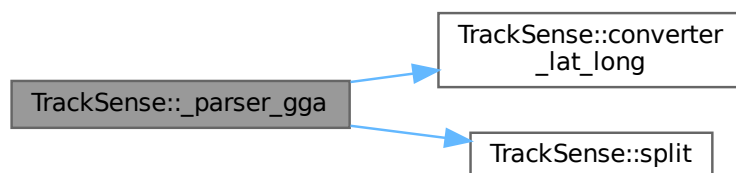
```
void TrackSense::_parser_gga (
    const std::string & frase,
    GPSTData & data ) [inline], [private]
```

Interpretará os dados caso venham no padrão GGA

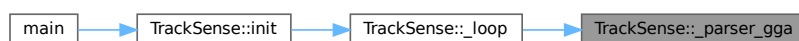
#### Parameters

	<i>frase</i>	Entrada recebida na porta serial
out	<i>data</i>	Último conjunto de dados

Here is the call graph for this function:



Here is the caller graph for this function:



### 1.3.3.5 \_parser\_rmc()

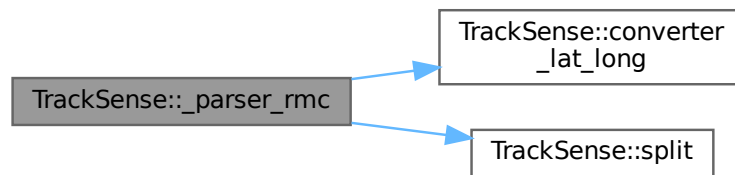
```
void TrackSense::_parser_rmc (
    const std::string & frase,
    GPSTData & data ) [inline], [private]
```

Interpretará os dados caso venham no padrão RMC.

#### Parameters

	<i>frase</i>	Entrada recebida na porta serial
out	<i>data</i>	Último conjunto de dados

Here is the call graph for this function:



Here is the caller graph for this function:



### 1.3.3.6 converter\_lat\_long()

```
static double TrackSense::converter_lat_long (
    const std::string & valor,
    const std::string & hemisf ) [inline], [static], [private]
```

Função estática auxiliar para converter latitude e longitude para graus decimais.

#### Parameters

<i>valor</i>	String representante da medida de latitude ou longitude.
<i>hemisf</i>	String representante do hemisfério.

Here is the caller graph for this function:



### 1.3.3.7 formatar\_csv()

```
static std::string TrackSense::formatar_csv (
    const GPSTData & data ) [inline], [static], [private]
```

Função estática auxiliar para formar string .csv.

Here is the caller graph for this function:

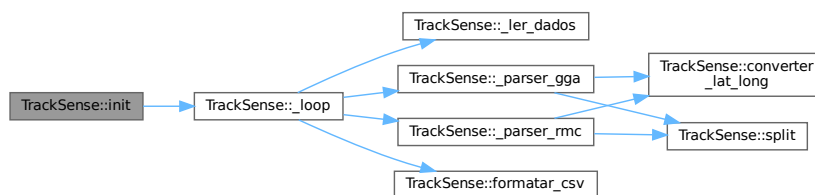


### 1.3.3.8 init()

```
void TrackSense::init ( ) [inline]
```

Inicia thread de leitura de dados do GPS e envio UDP. Há garantia de instância única.

Here is the call graph for this function:



Here is the caller graph for this function:



### 1.3.3.9 split()

```
static std::vector< std::string > TrackSense::split (
    const std::string & string_de_entrada,
    char separador ) [inline], [static], [private]
```

Função estática auxiliar para separar string em vetores de string. Similar ao método `split` do python.

Here is the caller graph for this function:

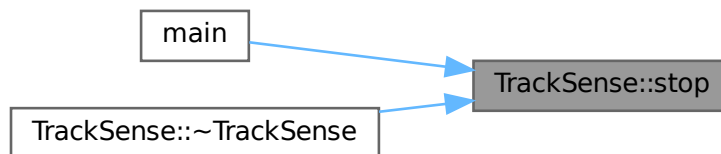


#### 1.3.3.10 stop()

```
void TrackSense::stop ( ) [inline]
```

Finaliza a thread de trabalho.

Here is the caller graph for this function:



### 1.3.4 Member Data Documentation

#### 1.3.4.1 \_addr\_dest

```
sockaddr_in TrackSense::_addr_dest [private]
```

#### 1.3.4.2 \_fd\_serial

```
int TrackSense::_fd_serial = -1 [private]
```

#### 1.3.4.3 \_ip\_destino

```
std::string TrackSense::_ip_destino [private]
```

#### 1.3.4.4 \_is\_exec

```
std::atomic<bool> TrackSense::_is_exec {false} [private]
```

#### 1.3.4.5 `_last_data_given`

```
GPSTData TrackSense::_last_data_given {} [private]
```

#### 1.3.4.6 `_porta_destino`

```
int TrackSense::_porta_destino [private]
```

#### 1.3.4.7 `_porta_serial`

```
std::string TrackSense::_porta_serial [private]
```

#### 1.3.4.8 `_sockfd`

```
int TrackSense::_sockfd [private]
```

#### 1.3.4.9 `_worker`

```
std::thread TrackSense::_worker [private]
```

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

- [src/TrackSense.hpp](#)



## Chapter 2

# File Documentation

### 2.1 src/debug.cpp File Reference

Responsável por prover ferramentas de debug.

```
#include <iostream>
#include "TrackSense.hpp"
#include "GPSSim.hpp"
Include dependency graph for debug.cpp:
```



#### Functions

- int [main](#) ()

#### 2.1.1 Detailed Description

Responsável por prover ferramentas de debug.

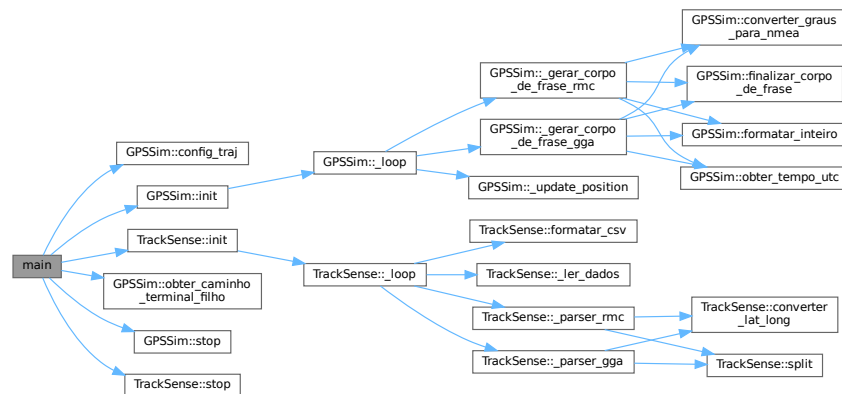
Já que a aplicação deve ser executada dentro da placa, não conseguiríamos executá-la no DeskTop. Para tanto, fez-se necessário o desenvolvimento de ferramentas que possibilitam a depuração de nosso código.

## 2.1.2 Function Documentation

### 2.1.2.1 main()

```
int main ( )
```

Here is the call graph for this function:



## 2.2 src/GPSSim.hpp File Reference

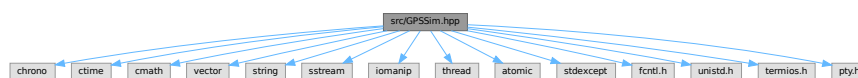
Implementação da Classe Simuladora do GPS6MV2.

```

#include <chrono>
#include <ctime>
#include <cmath>
#include <vector>
#include <string>
#include <sstream>
#include <iomanip>
#include <thread>
#include <atomic>
#include <stdexcept>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <pty.h>

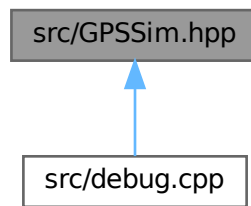
```

Include dependency graph for GPSSim.hpp:





This graph shows which files directly or indirectly include this file:



## Classes

- class [GPSSim](#)

*Simulador do módulo GPS que gera frases no padrão NMEA.*

### 2.2.1 Detailed Description

Implementação da Classe Simuladora do GPS6MV2.

Supondo que o módulo GPS6MV2 não esteja disponível, a classe implementada neste arquivo tem como objetivo simular todas as funcionalidades do mesmo.

## 2.3 GPSSim.hpp

[Go to the documentation of this file.](#)

```
00001
00008 #ifndef GPSSim_HPP
00009 #define GPSSim_HPP
00010
00011 //-----
00012
00013 // Para manipulações de Tempo e de Data
00014 #include <chrono>
00015 #include <ctime>
00016
00017 #include <cmath>
00018 #include <vector>
00019
00020 #include <string>
00021 #include <sstream>
00022 #include <iomanip>
00023
00024 // Para threads e sincronizações
00025 #include <thread>
00026 #include <atomic>
00027
00028 // Para tratamento de erros
00029 #include <stdexcept>
00030
00031 // As seguintes bibliotecas possuem relevância superior
00032 // Por se tratarem de bibliotecas C, utilizaremos o padrão de '::' para explicitar
00033 // que algumas funções advém delas.
00034 /*
00035 Fornece constantes e funções de controle de descritores de arquivos,
00036 operações de I/O de baixo nível e manipulação de flags de arquivos.
00037 */
```

```

00038 #include <fcntl.h>
00039 /*
00040 Fornece acesso a chamadas do OS de baixo nível, incluindo manipulação
00041 de processos, I/O de arquivos, controle de descritores e operações do
00042 sistema de arquivos.
00043 */
00044 #include <unistd.h>
00045 /*
00046 Fornece estruturas e funções para configurar a comunicação
00047 em sistemas Unix. Ele permite o controle detalhado sobre interfaces
00048 de terminal (TTY)
00049 */
00050 #include <termios.h>
00051 /*
00052 Fornece funções para criação e manipulação de pseudo-terminais (PTYs),
00053 um mecanismo essencial em sistemas Unix para emular terminais virtuais.
00054 */
00055 #include <pty.h>
00056
00066 class GPSSim {
00067 private:
00068
00069     // Informações Ligadas Aos Terminais
00070     int _fd_pai{0}, _fd_filho{0};
00071     char _nome_pt_filho[128];
00072
00073     // Thread de Execução Paralela e Flag de Controle
00074     std::thread _thread_geradora;
00075     std::atomic<bool> _is_exec{false};
00076
00077     // Informações da simulação
00078     double _lat, _lon, _alt;
00079     double _velocidade_nos;
00080     std::chrono::milliseconds _periodo_atualizacao;
00081
00082     // Configurações de Trajetória
00083     bool _mov_circ = false;
00084     double _raio{20.0}, _periodo_circ{20.0};
00085     double _lat_sim{0.0}, _lon_sim{0.0};
00086     std::chrono::steady_clock::time_point _start_time = std::chrono::steady_clock::now();
00087
00088
00095     static std::string
00096     formatar_inteiro(
00097         int valor,
00098         int quant_digitos
00099     ){
00100
00101         std::ostringstream oss;
00102         oss << std::setw(quant_digitos)
00103             << std::setfill('0')
00104             << valor;
00105         return oss.str();
00106     }
00107
00115     static void
00116     converter_graus_para_nmea(
00117         double graus_decimais,
00118         bool is_lat,
00119         std::string& ddmm,
00120         char& hemisf
00121     ){
00122
00123         hemisf = (is_lat) ? (
00124             (graus_decimais >= 0) ? 'N' : 'S'
00125             ) :
00126             (
00127                 (graus_decimais >= 0) ? 'E' : 'W'
00128             );
00129
00130         double valor_abs = std::fabs(graus_decimais);
00131         int graus = static_cast<int>(std::floor(valor_abs));
00132         double min = (valor_abs - graus) * 60.0;
00133
00134         // Não utilizamos apenas a função de formatar_inteiro, pois
00135         // utilizaremos o oss em seguida.
00136         std::ostringstream oss;
00137         if(
00138             is_lat
00139         ){
00140
00141             oss << std::setw(2)
00142                 << std::setfill('0')
00143                 << graus
00144                 << std::fixed
00145                 << std::setprecision(4)
00146                 << std::setw(7)

```

```

00147         « std::setfill('0')
00148         « min;
00149     }
00150     else{
00151
00152         oss « std::setw(3)
00153             « std::setfill('0')
00154             « graus
00155             « std::fixed
00156             « std::setprecision(4)
00157             « std::setw(7)
00158             « std::setfill('0')
00159             « min;
00160     }
00161
00162     ddmm = oss.str();
00163 }
00164
00170 static std::string
00171 finalizar_corpo_de_frase(
00172     const std::string& corpo_frase
00173 ){
00174
00175     uint8_t paridade = 0;
00176     for(
00177         char caract : corpo_frase
00178     ){
00179
00180         paridade ^= (uint8_t)caract;
00181     }
00182
00183     std::ostringstream oss;
00184     oss « '$'
00185         « corpo_frase
00186         « '*'
00187         « std::uppercase
00188         « std::hex
00189         « std::setw(2)
00190         « std::setfill('0')
00191         « (int)paridade
00192         « "\r\n";
00193
00194     return oss.str();
00195 }
00196
00201 static std::tm
00202 obter_tempo_utc(){
00203
00204     using namespace std::chrono;
00205     auto tempo_atual = system_clock::to_time_t(system_clock::now());
00206     std::tm tempo_utc{};
00207     gmtime_r(&tempo_atual, &tempo_utc);
00208     return tempo_utc;
00209 }
00210
00218 std::string
00219 _gerar_corpo_de_frase_rmc(
00220     double lat_graus,
00221     double lon_graus,
00222     double velocidade_nos
00223 ){
00224
00225     auto tempo_utc = obter_tempo_utc();
00226     std::string lat_nmea, lon_nmea;
00227     char hemisferio_lat, hemisferio_lon;
00228
00229     converter_graus_para_nmea(
00230         lat_graus,
00231         true,
00232         lat_nmea,
00233         hemisferio_lat
00234     );
00235     converter_graus_para_nmea(
00236         lon_graus,
00237         false,
00238         lon_nmea,
00239         hemisferio_lon
00240     );
00241
00242     // Formato: hhmmss.ss,A,lat,N/S,lon,E/W,velocidade,curso,data,,
00243     std::ostringstream oss;
00244     oss « "GPRMC,"
00245         « formatar_inteiro(tempo_utc.tm_hour, 2)
00246         « formatar_inteiro(tempo_utc.tm_min, 2)
00247         « formatar_inteiro(tempo_utc.tm_sec, 2) « ".00,A,"
00248         « lat_nmea « "," « hemisferio_lat « ","
00249         « lon_nmea « "," « hemisferio_lon « ","

```

```

00250         « std::fixed « std::setprecision(2) « velocidade_nos « ",0.00,"
00251         « formatar_inteiro(tempo_utc.tm_mday, 2)
00252         « formatar_inteiro(tempo_utc.tm_mon + 1, 2)
00253         « formatar_inteiro((tempo_utc.tm_year + 1900) % 100, 2)
00254         « ",,A";
00255
00256         return finalizar_corpo_de_frase(oss.str());
00257     }
00258
00259     std::string
00260     _gerar_corpo_de_frase_gga(
00261         double lat_graus,
00262         double lon_graus,
00263         double alt_metros,
00264         int sat = 8,
00265         double hdop = 0.9
00266     ){
00267         auto tempo_utc = obter_tempo_utc();
00268         std::string lat_nmea, lon_nmea;
00269         char hemisferio_lat, hemisferio_lon;
00270
00271         converter_graus_para_nmea(
00272             lat_graus,
00273             true,
00274             lat_nmea,
00275             hemisferio_lat
00276         );
00277         converter_graus_para_nmea(
00278             lon_graus,
00279             false,
00280             lon_nmea,
00281             hemisferio_lon
00282         );
00283
00284         // Formato: hhhmmss.ss,lat,N/S,lon,E/W,qualidade,satelites,HDOP,altitude,M,...
00285         std::ostringstream oss;
00286         oss << "GPGGA," << formatar_inteiro(tempo_utc.tm_hour, 2)
00287         << formatar_inteiro(tempo_utc.tm_min, 2)
00288         << formatar_inteiro(tempo_utc.tm_sec, 2) << ".00,"
00289         << lat_nmea << "," << hemisferio_lat << ","
00290         << lon_nmea << "," << hemisferio_lon << ",1,"
00291         << sat << "," << std::fixed << std::setprecision(1) << hdop << ","
00292         << std::fixed << std::setprecision(1) << alt_metros << ",M,0.0,M,";
00293
00294         return finalizar_corpo_de_frase(oss.str());
00295     }
00296
00297     void
00298     _update_position(){
00299         if (!mov_circ){ return; }
00300
00301         using namespace std::chrono;
00302         double tempo_decorrido = duration<double>(steady_clock::now() - _start_time).count();
00303         double angulo = (2.0 * M_PI) * std::fmod(tempo_decorrido / _periodo_circ, 1.0);
00304
00305         // Aproximação: 1° de latitude = 111320 metros
00306         // Ajuste da longitude pelo cosseno da latitude
00307         double delta_lat = (_raio * std::sin(angulo)) / 111320.0;
00308         double delta_lon = (_raio * std::cos(angulo)) / (111320.0 * std::cos(_lat * M_PI / 180.0));
00309
00310         _lat_sim = _lat + delta_lat;
00311         _lon_sim = _lon + delta_lon;
00312     }
00313
00314     void
00315     _loop(){
00316         // Inicializa posição simulada
00317         _lat_sim = _lat;
00318         _lon_sim = _lon;
00319
00320         while (_is_exec){
00321             // Gera frases NMEA
00322             auto rmc = _gerar_corpo_de_frase_rmc(_lat_sim, _lon_sim, _velocidade_nos);
00323             auto gga = _gerar_corpo_de_frase_gga(_lat_sim, _lon_sim, _alt, 10, 0.8);
00324
00325             const std::string saida = rmc + gga;
00326             std::cout << "\033[7mGPS6MV2 Simulado Emitindo:\033[0m \n" << saida << std::endl;
00327             (void)!::write(_fd_pai, saida.data(), saida.size());
00328
00329             // Aguarda o próximo ciclo
00330             std::this_thread::sleep_for(_periodo_atualizacao);
00331             _update_position();
00332         }
00333     }
00334 }

```

```

00352
00353 public:
00354
00371     explicit
00372     GPSSim(
00373         double latitude_inicial_graus,
00374         double longitude_inicial_graus,
00375         double altitude_metros = 10.0,
00376         double frequencia_atualizacao_hz = 1.0,
00377         double velocidade_nos = 10.0
00378     ) : _lat(latitude_inicial_graus),
00379         _lon(longitude_inicial_graus),
00380         _alt(altitude_metros),
00381         _periodo_atualizacao((frequencia_atualizacao_hz) > 0 ?
00382
std::chrono::milliseconds((int)std::llround(1000.0/frequencia_atualizacao_hz)) :
std::chrono::milliseconds(1000)),
00384         _velocidade_nos(velocidade_nos)
00385     {
00386
00387         // Cria o par de pseudo-terminais
00388         if(
00389             ::openpty( &_fd_pai, &_fd_filho, _nome_pt_filho, nullptr, nullptr ) != 0
00390         ){
00391             throw std::runtime_error("Falha ao criar pseudo-terminal");
00392         }
00393
00394         // Configura o terminal filho para simular o módulo real (9600 8N1)
00395         termios config_com{}; // Cria a estrutura vazia
00396         ::tcgetattr(_fd_filho, &config_com); // Lê as configurações atuais e armazena na struct
00397         ::cfsetispeed(&config_com, B115200); // Definimos velocidade de entrada e de saída
00398         ::cfsetospeed(&config_com, B115200); // Essa constante está presente dentro do termios.h
00399         // Diversas operações bits a bits
00400         config_com.c_cflag = (config_com.c_cflag & ~CSIZE) | CS8;
00401         config_com.c_cflag |= (CLOCAL | CREAD);
00402         config_com.c_cflag &= ~(PARENB | CSTOPB);
00403         config_com.c_iflag = IGNPAR;
00404         config_com.c_oflag = 0;
00405         config_com.c_lflag = 0;
00406         tcsetattr(_fd_filho, TCSANOW, &config_com); // Aplicamos as configurações
00407
00408         // Fecha o filho - será aberto pelo usuário no caminho correto
00409         // Mantemos o Pai aberto para procedimentos posteriores
00410         ::close(_fd_filho);
00411     }
00412
00418     ~GPSSim(){ stop(); if( _fd_pai >= 0 ){ ::close(_fd_pai); } }
00419
00424     std::string
00425     obter_caminho_terminal_filho() const { return std::string(_nome_pt_filho); }
00426
00435     void
00436     init(){
00437         if(
00438             // Variáveis atômicas possuem esta funcionalidade
00439             !_is_exec.exchange(true)
00440         ){
00441
00442             return;
00443         }
00444
00445         _thread_geradora = std::thread(
00446             // Função Lambda que será executada pela thread
00447             // Observe que os argumentos utilizados serão obtidos pelo
00448             // this inserido nos colchetes
00449             [this]{ _loop(); }
00450             );
00451     }
00452
00459     void
00460     stop(){
00461
00462         if(
00463             !_is_exec.exchange(false)
00464         ){
00465
00466             return;
00467         }
00468
00469         if(
00470             _thread_geradora.joinable()
00471         ){
00472
00473             _thread_geradora.join();
00474         }
00475     }
00476

```

```

00480     void
00481     config_traj(
00482         double raio_metros = 20.0,
00483         double periodo_segundos = 120.0
00484     ){
00485
00486         if( !_mov_circ ){ return; }
00487
00488         _raio = raio_metros;
00489         _periodo_circ = periodo_segundos;
00490         _mov_circ = true;
00491     }
00492 };
00493
00494 #endif // GPSSim_HPP

```

## 2.4 src/TrackSense.hpp File Reference

Implementação da Classe Sensora do Módulo GPS6MV2.

```

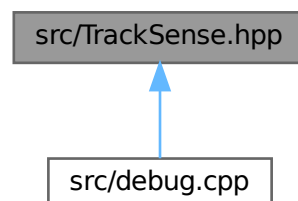
#include <string>
#include <vector>
#include <sstream>
#include <iomanip>
#include <iostream>
#include <cstring>
#include <chrono>
#include <cmath>
#include <ctime>
#include <thread>
#include <atomic>
#include <mutex>
#include <stdexcept>
#include <fcntl.h>
#include <termios.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

```

Include dependency graph for TrackSense.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [TrackSense](#)  
*Representa um sensor GPS (GY-GPS6MV2 ou Simulado).*
- struct [TrackSense::GPSData](#)  
*Struct para armazenar dados GPS*

### 2.4.1 Detailed Description

Implementação da Classe Sensora do Módulo GPS6MV2.

Responsável por ler, interpretar e enviar dados via UDP.

## 2.5 TrackSense.hpp

[Go to the documentation of this file.](#)

```

00001
00007 #ifndef TRACKSENSE_HPP
00008 #define TRACKSENSE_HPP
00009
00010
00011 //-----
00012 #include <string>
00013 #include <vector>
00014 #include <sstream>
00015 #include <iomanip>
00016 #include <iostream>
00017 #include <cstring>
00018
00019 #include <chrono>
00020 #include <cmath>
00021 #include <ctime>
00022
00023 #include <thread>
00024 #include <atomic>
00025 #include <mutex>
00026
00027 #include <stdexcept>
00028
00029 // Sistemas Linux
00030 #include <fcntl.h>
00031 #include <termios.h>
00032 #include <unistd.h>
00033 #include <sys/socket.h>
00034 #include <arpa/inet.h>
00035 #include <netinet/in.h>
00036
00049 class TrackSense {
00050 public:
00051
00056     struct GPSData {
00057         std::string time_utc;
00058         double lat;
00059         double lon;
00060         double vel;
00061         double alt;
00062         int sat;
00063         double hdop;
00064     };
00065
00066
00067 private:
00068     // Relacionadas ao Envio UDP
00069     std::string _ip_destino;
00070     int _porta_destino;
00071     int _sockfd;
00072     sockaddr_in _addr_dest;
00073
00074     // Relacionados ao trabalho de leitura
00075     std::thread _worker;
00076     std::atomic<bool> _is_exec{false};

```

```

00077
00078 // Relacionados aos dados lidos
00079 GPSData _last_data_given{};
00080 std::string _porta_serial;
00081 int _fd_serial = -1;
00082
00086 static std::vector<std::string>
00087 split(
00088     const std::string& string_de_entrada,
00089     char separador
00090 ){
00091
00092     std::vector<std::string> elementos;
00093     std::stringstream ss(string_de_entrada);
00094
00095     std::string elemento_individual;
00096     while(
00097         std::getline(
00098             ss,
00099             elemento_individual,
00100             separador
00101         )
00102     ){
00103         if(
00104             !elemento_individual.empty()
00105         ){
00106             elementos.push_back(elemento_individual);
00107         }
00108     }
00109 }
00110
00111 return elementos;
00112 }
00113
00119 static double
00120 converter_lat_long(
00121     const std::string& valor,
00122     const std::string& hemisf
00123 ){
00124
00125     if( valor.empty() ){ return 0.0; }
00126
00127     double valor_cru = std::stod(valor);
00128     double graus = floor(valor_cru / 100);
00129     double minutos = valor_cru - graus * 100;
00130     double dec = (graus + minutos / 60.0) * ( (hemisf == "S" || hemisf == "W" ) ? -1 : 1 );
00131
00132     return dec;
00133 }
00134
00138 static std::string
00139 formatar_csv(
00140     const GPSData& data
00141 ){
00142
00143     std::ostringstream oss;
00144     oss << data.time_uto << ", "
00145         << std::fixed << std::setprecision(6)
00146         << data.lat << ", "
00147         << data.lon << ", "
00148         << std::setprecision(2) << data.vel << ", "
00149         << data.alt << ", "
00150         << data.sat << ", "
00151         << data.hdop;
00152
00153     return oss.str();
00154 }
00155
00159 void
00160 _open_serial(){
00161
00162     _fd_serial = ::open(
00163         _porta_serial.c_str(),
00164         O_RDONLY | O_NOCTTY | O_SYNC
00165     );
00166
00167     if(
00168         _fd_serial < 0
00169     ){
00170
00171         throw std::runtime_error("Erro ao abrir porta serial do GPS");
00172     }
00173
00174     // Verificamos a porta serial de leitura
00175     termios tty{};
00176     if(
00177         ::tcgetattr(

```



```

00178         _fd_serial,
00179         &tty
00180     ) != 0
00181 }{
00182
00183     throw std::runtime_error("Erro ao tentar entrar na porta serial, tcgetattr");
00184 }
00185
00186 ::cfsetospeed(&tty, B115200);
00187 ::cfsetispeed(&tty, B115200);
00188
00189 // Diversas operações bit a bit
00190 tty.c_cflag = (tty.c_cflag & ~CSIZE) | CS8; // 8 bits
00191 tty.c_iflag &= ~IGNBRK;
00192 tty.c_lflag = 0;
00193 tty.c_oflag = 0;
00194 tty.c_cc[VMIN] = 1;
00195 tty.c_cc[VTIME] = 1;
00196 tty.c_cflag |= (CLOCAL | CREAD);
00197 tty.c_cflag &= ~(PARENB | PARODD);
00198 tty.c_cflag &= ~CSTOPB;
00199 tty.c_cflag &= ~CRTSCTS;
00200
00201 if(
00202     ::tcsetattr(
00203         _fd_serial,
00204         TCSANOW,
00205         &tty
00206     ) != 0
00207 ){
00208
00209     throw std::runtime_error("Erro tcsetattr");
00210 }
00211 }
00212
00213 std::string
00214 ler_dados(){
00215
00216     char buffer[256];
00217     int ult_indice = read(
00218         _fd_serial,
00219         buffer,
00220         sizeof(buffer) - 1
00221     );
00222
00223     if( ult_indice > 0 ){
00224
00225         buffer[ult_indice] = '\0';
00226         return std::string(buffer);
00227     }
00228
00229     return {};
00230 }
00231
00232 void
00233 _parser_rmc(
00234     const std::string& frase,
00235     GPSData& data
00236 ){
00237
00238     std::vector<std::string> campos_de_infos = split(frase, ',');
00239
00240     // Logo, está fora do padrão rmc.
00241     if( campos_de_infos.size() < 8 ){ return; }
00242
00243     data.time_utc = campos_de_infos[1];
00244     data.lat = converter_lat_long(campos_de_infos[3], campos_de_infos[4]);
00245     data.lon = converter_lat_long(campos_de_infos[5], campos_de_infos[6]);
00246     data.vel = std::stod(campos_de_infos[7]) * 0.514; // Transformando de nós para m/s
00247 }
00248
00249 void
00250 _parser_gga(
00251     const std::string& frase,
00252     GPSData& data
00253 ){
00254
00255     std::vector<std::string> campos_de_infos = split(frase, ',');
00256
00257     // Logo, está fora do padrão gga
00258     if( campos_de_infos.size() < 10 ){ return; }
00259
00260     data.time_utc = campos_de_infos[1];
00261     data.lat = converter_lat_long(campos_de_infos[2], campos_de_infos[3]);
00262     data.lon = converter_lat_long(campos_de_infos[4], campos_de_infos[5]);
00263     data.sat = std::stoi(campos_de_infos[7]);
00264     data.hdop = std::stod(campos_de_infos[8]);

```

```

00280         data.alt = std::stod(campos_de_infos[9]);
00281     }
00282
00286     void
00287     _loop(){
00288
00289         while(
00290             _is_exec
00291         ){
00292
00293             std::string linha = _ler_dados();
00294             if(
00295                 !linha.empty()
00296             ){
00297
00298                 if( linha.find("RMC") != std::string::npos ){
00299
00300                     _parser_rmc(linha, _last_data_given);
00301                 }
00302                 else if( linha.find("GGA") != std::string::npos ){
00303
00304                     _parser_gga(linha, _last_data_given);
00305                 }
00306             }
00307
00308             // Então temos os dados.
00309             std::cout << "\n\033[7mSensor Interpretando:\033[0m\n"
00310                 << formatar_csv(_last_data_given) << std::endl;
00311         }
00312     }
00313
00314 public:
00315
00324     TrackSense(
00325         const std::string& ip_destino,
00326         int porta_destino,
00327         const std::string& porta_serial = ""
00328     ) : _ip_destino(ip_destino),
00329         _porta_destino(porta_destino),
00330         _porta_serial(porta_serial)
00331     {
00332
00333
00334         _sockfd = ::socket(AF_INET, SOCK_DGRAM, 0);
00335         if( _sockfd < 0 ){ throw std::runtime_error("Erro ao criar socket UDP"); }
00336
00337         _addr_dest.sin_family = AF_INET;
00338         _addr_dest.sin_port = ::htons(_porta_destino);
00339         _addr_dest.sin_addr.s_addr = ::inet_addr(_ip_destino.c_str());
00340
00341         _open_serial();
00342     }
00343
00349     ~TrackSense() { stop(); if( _sockfd >= 0 ){ ::close(_sockfd); } if( _fd_serial >= 0 ){
::close(_fd_serial); } }
00350
00354     void
00355     init(){
00356
00357         if( _is_exec.exchange(true) ){ return; }
00358
00359         _worker = std::thread(
00360             [this]{ _loop(); }
00361             );
00362     }
00363
00367     void
00368     stop(){
00369
00370         if( !_is_exec.exchange(false) ){ return; }
00371
00372         if(
00373             _worker.joinable()
00374         ){
00375
00376             _worker.join();
00377         }
00378     }
00379 };
00380
00381 #endif // TRACKSENSE_HPP

```

# Index

- [\\_addr\\_dest](#)  
TrackSense, [22](#)
  - [\\_alt](#)  
GPSSim, [13](#)
  - [\\_fd\\_filho](#)  
GPSSim, [13](#)
  - [\\_fd\\_pai](#)  
GPSSim, [13](#)
  - [\\_fd\\_serial](#)  
TrackSense, [22](#)
  - [\\_gerar\\_corpo\\_de\\_frase\\_gga](#)  
GPSSim, [5](#)
  - [\\_gerar\\_corpo\\_de\\_frase\\_rmc](#)  
GPSSim, [6](#)
  - [\\_ip\\_destino](#)  
TrackSense, [22](#)
  - [\\_is\\_exec](#)  
GPSSim, [13](#)  
TrackSense, [22](#)
  - [\\_last\\_data\\_given](#)  
TrackSense, [22](#)
  - [\\_lat](#)  
GPSSim, [13](#)
  - [\\_lat\\_sim](#)  
GPSSim, [13](#)
  - [\\_ler\\_dados](#)  
TrackSense, [17](#)
  - [\\_lon](#)  
GPSSim, [13](#)
  - [\\_lon\\_sim](#)  
GPSSim, [13](#)
  - [\\_loop](#)  
GPSSim, [7](#)  
TrackSense, [17](#)
  - [\\_mov\\_circ](#)  
GPSSim, [14](#)
  - [\\_nome\\_pt\\_filho](#)  
GPSSim, [14](#)
  - [\\_open\\_serial](#)  
TrackSense, [18](#)
  - [\\_parser\\_gga](#)  
TrackSense, [18](#)
  - [\\_parser\\_rmc](#)  
TrackSense, [19](#)
  - [\\_periodo\\_atualizacao](#)  
GPSSim, [14](#)
  - [\\_periodo\\_circ](#)  
GPSSim, [14](#)
  - [\\_porta\\_destino](#)  
TrackSense, [23](#)
  - [\\_porta\\_serial](#)  
TrackSense, [23](#)
  - [\\_raio](#)  
GPSSim, [14](#)
  - [\\_sockfd](#)  
TrackSense, [23](#)
  - [\\_start\\_time](#)  
GPSSim, [14](#)
  - [\\_thread\\_geradora](#)  
GPSSim, [14](#)
  - [\\_update\\_position](#)  
GPSSim, [8](#)
  - [\\_velocidade\\_nos](#)  
GPSSim, [14](#)
  - [\\_worker](#)  
TrackSense, [23](#)
  - [~GPSSim](#)  
GPSSim, [5](#)
  - [~TrackSense](#)  
TrackSense, [17](#)
- [alt](#)  
TrackSense::GPSData, [2](#)
- [config\\_traj](#)  
GPSSim, [8](#)
- [converter\\_graus\\_para\\_nmea](#)  
GPSSim, [9](#)
- [converter\\_lat\\_long](#)  
TrackSense, [20](#)
- [debug.cpp](#)  
main, [26](#)
- [finalizar\\_corpo\\_de\\_frase](#)  
GPSSim, [9](#)
- [formatar\\_csv](#)  
TrackSense, [20](#)
- [formatar\\_inteiro](#)  
GPSSim, [10](#)
- [GPSSim, \[3\]\(#\)](#)
  - [\\_alt, \[13\]\(#\)](#)
  - [\\_fd\\_filho, \[13\]\(#\)](#)
  - [\\_fd\\_pai, \[13\]\(#\)](#)
  - [\\_gerar\\_corpo\\_de\\_frase\\_gga, \[5\]\(#\)](#)
  - [\\_gerar\\_corpo\\_de\\_frase\\_rmc, \[6\]\(#\)](#)
  - [\\_is\\_exec, \[13\]\(#\)](#)
  - [\\_lat, \[13\]\(#\)](#)
  - [\\_lat\\_sim, \[13\]\(#\)](#)

- [\\_lon](#), [13](#)
- [\\_lon\\_sim](#), [13](#)
- [\\_loop](#), [7](#)
- [\\_mov\\_circ](#), [14](#)
- [\\_nome\\_pt\\_filho](#), [14](#)
- [\\_periodo\\_atualizacao](#), [14](#)
- [\\_periodo\\_circ](#), [14](#)
- [\\_raio](#), [14](#)
- [\\_start\\_time](#), [14](#)
- [\\_thread\\_geradora](#), [14](#)
- [\\_update\\_position](#), [8](#)
- [\\_velocidade\\_nos](#), [14](#)
- [~GPSSim](#), [5](#)
- [config\\_traj](#), [8](#)
- [converter\\_graus\\_para\\_nmea](#), [9](#)
- [finalizar\\_corpo\\_de\\_frase](#), [9](#)
- [formatar\\_inteiro](#), [10](#)
- [GPSSim](#), [5](#)
- [init](#), [10](#)
- [obter\\_caminho\\_terminal\\_filho](#), [11](#)
- [obter\\_tempo\\_utc](#), [12](#)
- [stop](#), [12](#)
- [hdop](#)
  - [TrackSense::GPSData](#), [2](#)
- [init](#)
  - [GPSSim](#), [10](#)
  - [TrackSense](#), [21](#)
- [lat](#)
  - [TrackSense::GPSData](#), [2](#)
- [lon](#)
  - [TrackSense::GPSData](#), [2](#)
- [main](#)
  - [debug.cpp](#), [26](#)
- [obter\\_caminho\\_terminal\\_filho](#)
  - [GPSSim](#), [11](#)
- [obter\\_tempo\\_utc](#)
  - [GPSSim](#), [12](#)
- [sat](#)
  - [TrackSense::GPSData](#), [2](#)
- [split](#)
  - [TrackSense](#), [21](#)
- [src/debug.cpp](#), [25](#)
- [src/GPSSim.hpp](#), [26](#), [27](#)
- [src/TrackSense.hpp](#), [32](#), [33](#)
- [stop](#)
  - [GPSSim](#), [12](#)
  - [TrackSense](#), [22](#)
- [time\\_utc](#)
  - [TrackSense::GPSData](#), [2](#)
- [TrackSense](#), [15](#)
  - [\\_addr\\_dest](#), [22](#)
  - [\\_fd\\_serial](#), [22](#)
  - [\\_ip\\_destino](#), [22](#)
  - [\\_is\\_exec](#), [22](#)
  - [\\_last\\_data\\_given](#), [22](#)
  - [\\_ler\\_dados](#), [17](#)
  - [\\_loop](#), [17](#)
  - [\\_open\\_serial](#), [18](#)
  - [\\_parser\\_gga](#), [18](#)
  - [\\_parser\\_rmc](#), [19](#)
  - [\\_porta\\_destino](#), [23](#)
  - [\\_porta\\_serial](#), [23](#)
  - [\\_sockfd](#), [23](#)
  - [\\_worker](#), [23](#)
  - [~TrackSense](#), [17](#)
  - [converter\\_lat\\_long](#), [20](#)
  - [formatar\\_csv](#), [20](#)
  - [init](#), [21](#)
  - [split](#), [21](#)
  - [stop](#), [22](#)
  - [TrackSense](#), [16](#)
- [TrackSense::GPSData](#), [1](#)
  - [alt](#), [2](#)
  - [hdop](#), [2](#)
  - [lat](#), [2](#)
  - [lon](#), [2](#)
  - [sat](#), [2](#)
  - [time\\_utc](#), [2](#)
  - [vel](#), [2](#)
- [vel](#)
  - [TrackSense::GPSData](#), [2](#)