Track Sense

Generated by Doxygen 1.9.8

Chapter 1

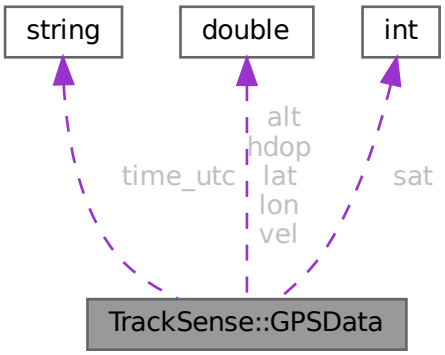
Class Documentation

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:

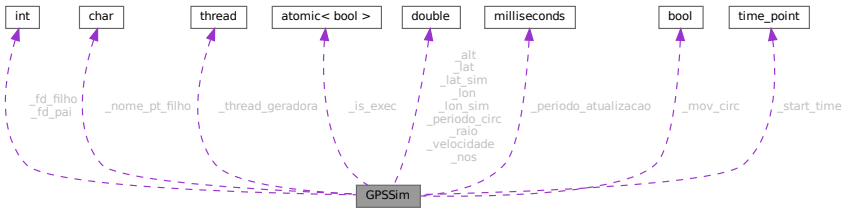
- src/TrackSense.hpp

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 `explict` para impedir que futuras conversões implicitas 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

latitude_inicial_graus	Latitude inicial em graus decimais
longitude_inicial_graus	Longitude inicial em graus decimais
altitude_metros	Altitude inicial em metros, setada para 10.
frequencia_atualizacao_hz	Frequência de atualização das frases NMEA em Hertz, setada para 1
velocidade_nos	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)

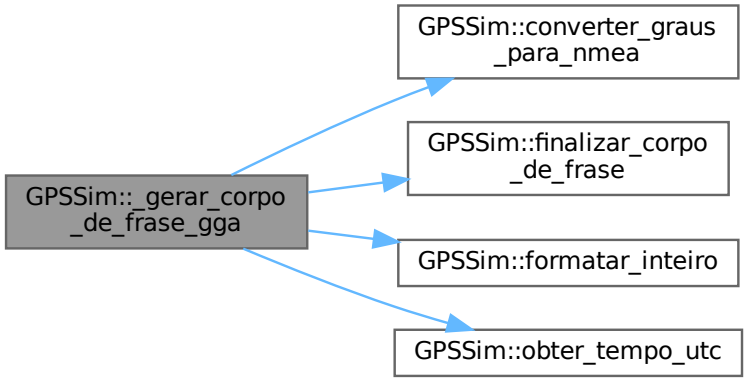
Parameters

lat_graus	Latitude em graus decimais
lon_graus	Longitude em graus decimais
alt_metros	Altitude em metros
sat	Número de satélites visíveis
hdop	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)

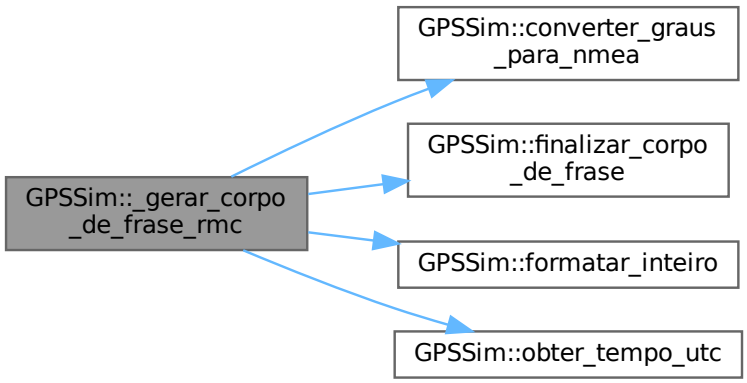
Parameters

lat_graus	Latitude em graus decimais
lon_graus	Longitude em graus decimais
velocidade_nos	Velocidade em nós

Returns

String correspondendo ao corpo de Frase RMC

Here is the call graph for this function:



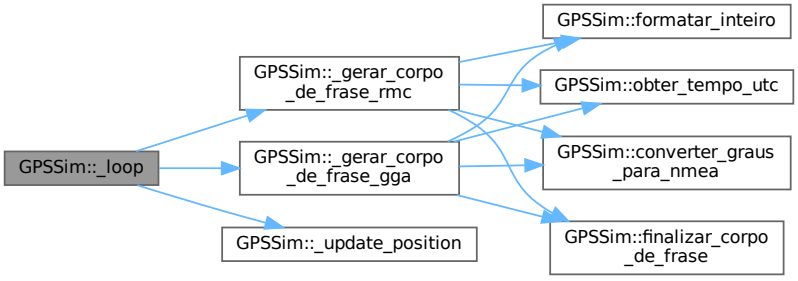
Here is the caller graph for this function:

1.2.3.3 _loop()

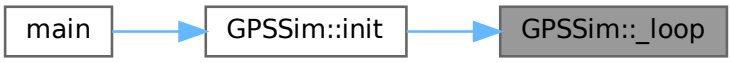
```
void GPSSim::_loop ( )  [inline], [private]
```

Loop principal de geração de dados.

Here is the call graph for this function:



Here is the caller graph for this function:

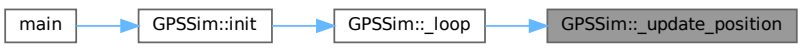


1.2.3.4 _update_position()

```
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()

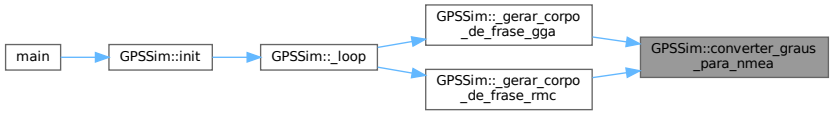
```
static void GPSSim::converter_graus_para_nmea (
            double graus_decimais,
            bool is_lat,
            std::string & ddmm,
            char & hemisf )  [inline], [static], [private]
```

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

Parameters

	graus_decimais	Valor em graus decimais
	is_lat	Flag de eixo
out	*ddmm*	String com valor formatado em graus e minutos
out	*hemisf*	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.

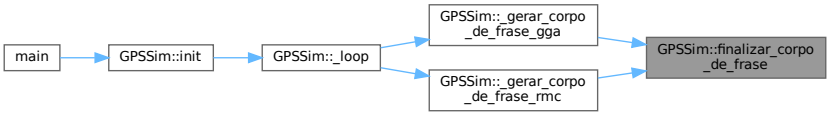
Parameters

corpo_frase	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.

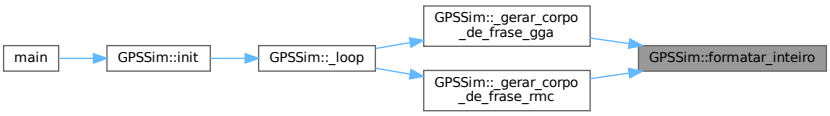
Parameters

valor	Número a ser formatado
quant_digitos	Quantidade de Dígitos presente

Returns

string formatada

Here is the caller graph for this function:



1.2.3.9 init()

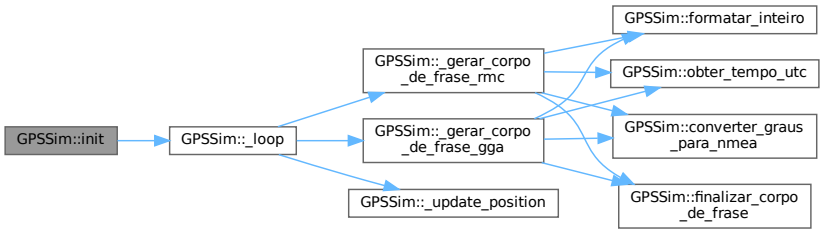
```
void GPSSim::init ( )  [inline]
```

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

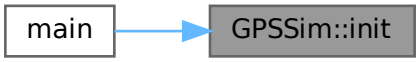
Garante a criação de apenas uma thread utilizando uma variável atômica. O padrão NMEA é o protocolo padrão usado por módulos GPS, cada mensagem começa com $ e termina com \r\n. Exemplo:

- $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()

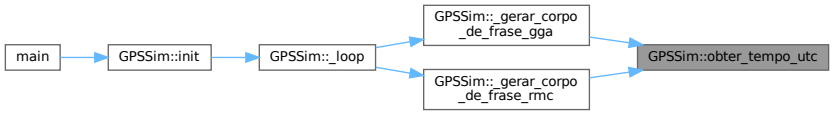
```
static std::tm GPSSim::obter_tempo_utc ( )  [inline], [static], [private]
```

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:

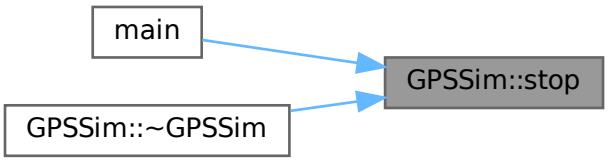


1.2.3.12 stop()

```
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:

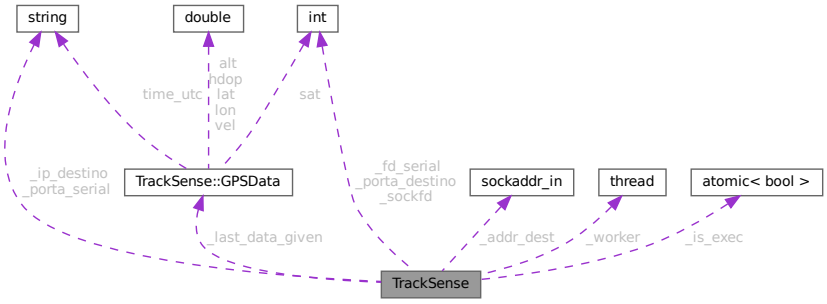
- src/GPSSim.hpp

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 configure 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 GPSData &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}
- GPSData _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()

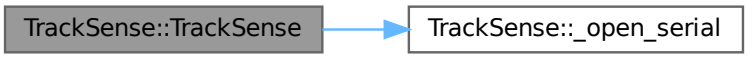
```
TrackSense::TrackSense (
            const std::string & ip_destino,
            int porta_destino,
            const std::string & porta_serial )  [inline]
```

Construtor da Classe.

Parameters

ip_destino	Endereço IP de destiono.
porta_destino	Porta UDP de destino
porta_serial	Caminho da porta serial

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

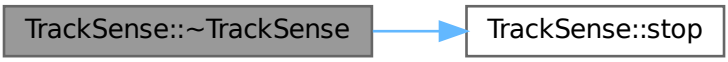


1.3.2.2 ~TrackSense()

```
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.

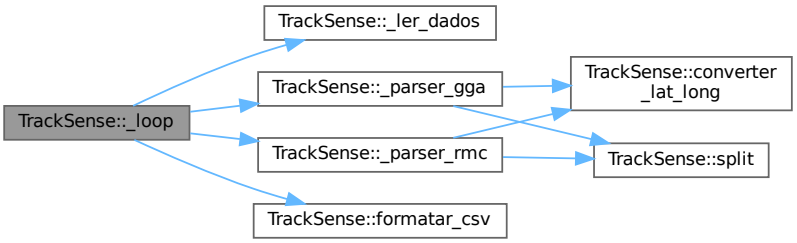
Utiliza um buffer de 256 caracteres. Here is the caller graph for this function:

1.3.3.2 _loop()

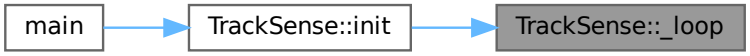
```
void TrackSense::_loop ( )  [inline], [private]
```

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:

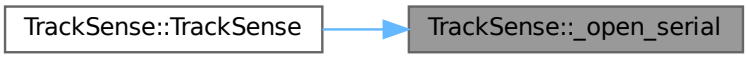


1.3.3.3 _open_serial()

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

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

Here is the caller graph for this function:



1.3.3.4 _parser_gga()

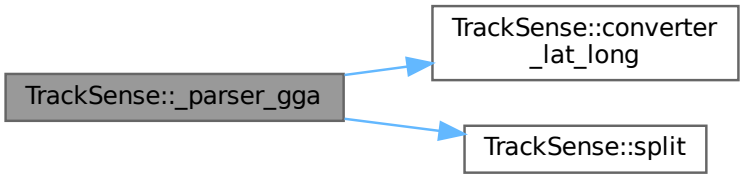
```
void TrackSense::_parser_gga (
            const std::string & frase,
            GPSData & data )  [inline], [private]
```

Interpretará os dados caso venham no padrão GGA

Parameters

	frase	Entrada recebida na porta serial
out	*data*	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()

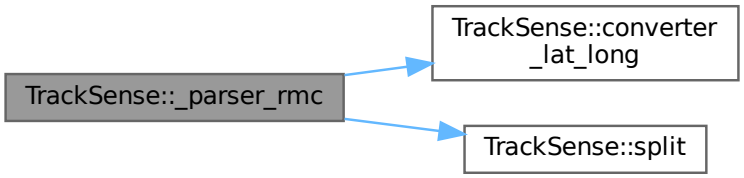
```
void TrackSense::_parser_rmc (
            const std::string & frase,
            GPSData & data )  [inline], [private]
```

Interpretará os dados caso venham no padrão RMC.

Parameters

	frase	Entrada recebida na porta serial
out	*data*	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()

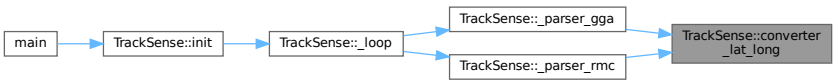
```
static double TrackSense::converter_lat_long (
            const std::string & valor,
            const std::string & hemisf )  [inline], [static], [private]
```

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

Parameters

valor	String representante da medida de latitude ou longitude.
hemisf	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 GPSData & data )  [inline], [static], [private]
```

Função estática auxiliar para formar string .csv.

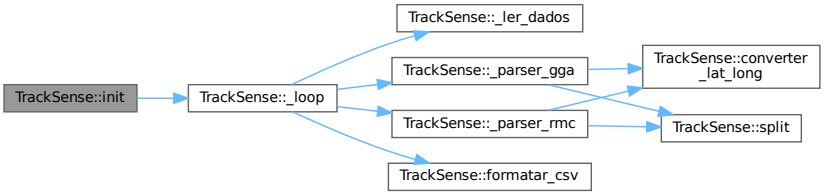
Here is the caller graph for this function:

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:



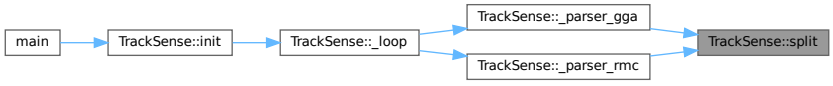
Here is the caller graph for this function:

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:

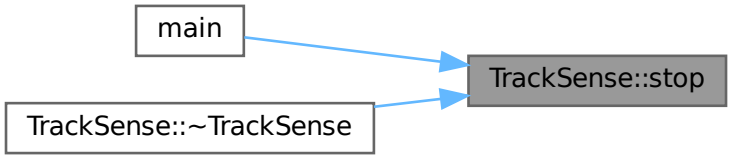


1.3.3.10 stop()

```
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

```
GPSData 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.

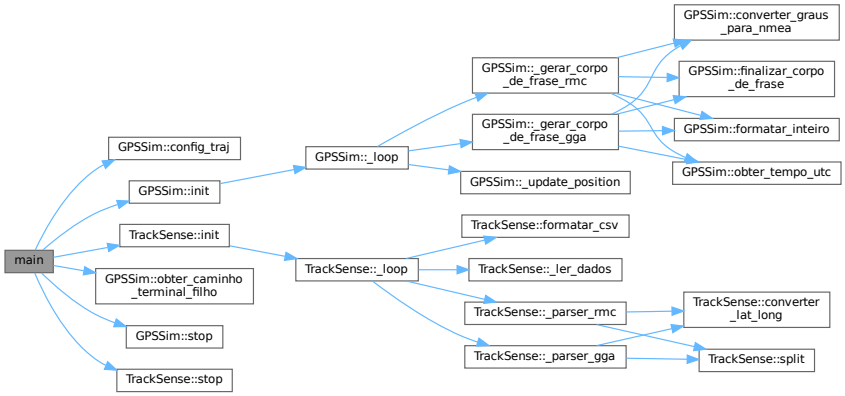
Já que a aplicação deve ser executada dentro da placa, não conseguiríamos executá-la no DeskTop. Para tanto, fez-se necessário o desenvolvimento de ferramentas que possibilitam a debugação de nosso código.

2.1.2 Function Documentation

2.1.2.1 main()

```
int main ( )
```

Here is the call graph for this function:

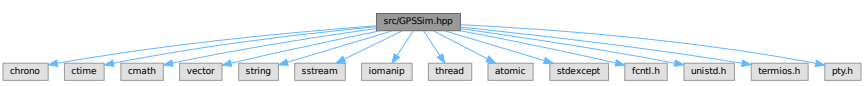


2.2 src/GPSSim.hpp File Reference

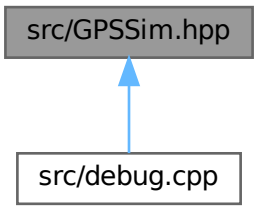
Implementação da Classe Simuladora do GPS6MV2.

```
#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
00268     std::string
00269     _gerar_corpo_de_frase_gga(
00270         double lat_graus,
00271         double lon_graus,
00272         double alt_metros,
00273         int    sat = 8,
00274         double hdop = 0.9
00275     ){
00276         auto tempo_utc = obter_tempo_utc();
00277         std::string lat_nmea, lon_nmea;
00278         char hemisferio_lat, hemisferio_lon;
00279
00280         converter_graus_para_nmea(
00281             lat_graus,
00282             true,
00283             lat_nmea,
00284             hemisferio_lat
00285         );
00286         converter_graus_para_nmea(
00287             lon_graus,
00288             false,
00289             lon_nmea,
00290             hemisferio_lon
00291         );
00292
00293         // Formato: hhmmss.ss,lat,N/S,lon,E/W,qualidade,satelites,HDOP,altitude,M,...
00294         std::ostringstream oss;
00295         oss « "GPGGA," « formatar_inteiro(tempo_utc.tm_hour, 2)
00296             « formatar_inteiro(tempo_utc.tm_min, 2)
00297             « formatar_inteiro(tempo_utc.tm_sec, 2) « ".00,"
00298             « lat_nmea   « "," « hemisferio_lat « ","
00299             « lon_nmea   « "," « hemisferio_lon « ",1,"
00300             « sat  « "," « std::fixed « std::setprecision(1) « hdop « ","
00301             « std::fixed « std::setprecision(1) « alt_metros « ",M,0.0,M,,";
00302
00303         return finalizar_corpo_de_frase(oss.str());
00304     }
00305
00309     void
00310     _update_position(){
00311         if (!_mov_circ){ return; }
00312
00313         using namespace std::chrono;
00314         double tempo_decorrido = duration<double>(steady_clock::now() - _start_time).count();
00315         double angulo = (2.0 * M_PI) * std::fmod(tempo_decorrido / _periodo_circ, 1.0);
00316
00317         // Aproximação: 1° de latitude = 111320 metros
00318         // Ajuste da longitude pelo cosseno da latitude
00319         double delta_lat = (_raio * std::sin(angulo)) / 111320.0;
00320         double delta_lon = (_raio * std::cos(angulo)) / (111320.0 * std::cos(_lat * M_PI / 180.0));
00321
00322         _lat_sim = _lat + delta_lat;
00323         _lon_sim = _lon + delta_lon;
00324     }
00325
00326
00330     void
00331     _loop(){
00332         // Inicializa posição simulada
00333         _lat_sim = _lat;
00334         _lon_sim = _lon;
00335
00336         while (_is_exec){
00337
00338             // Gera frases NMEA
00339             auto rmc = _gerar_corpo_de_frase_rmc(_lat_sim, _lon_sim, _velocidade_nos);
00340             auto gga = _gerar_corpo_de_frase_gga(_lat_sim, _lon_sim, _alt, 10, 0.8);
00341
00342             const std::string saida = rmc + gga;
00343             // std::cout « "\033[7mGPS6MV2 Simulado Emitindo:\033[0m \n" « saida « std::endl;
00344             (void)!::write(_fd_pai, saida.data(), saida.size());
00345
00346             // Aguarda o próximo ciclo
00347             std::this_thread::sleep_for(_periodo_atualizacao);
00348
00349             _update_position();
00350         }
00351     }
```

```
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)) :
00383                                 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/main.cpp File Reference

`#include "TrackSense.hpp"`
Include dependency graph for main.cpp:

Functions

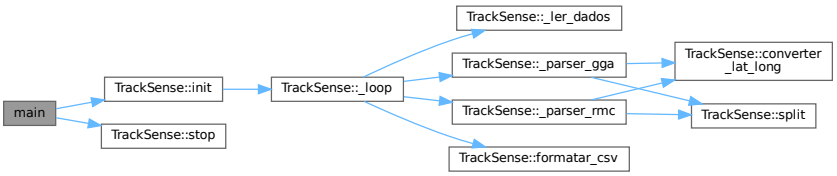
- int main ()

2.4.1 Function Documentation

2.4.1.1 main()

`int main ( )`

Here is the call graph for this function:



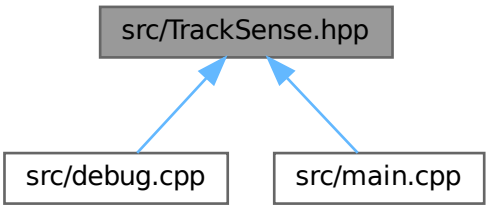
2.5 src/TrackSense.hpp File Reference

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

2.5.1 Detailed Description

Implementação da Classe Sensora do Módulo GPS6MV2.

Responsável por ler, interpretar e enviar dados via UDP.

2.6 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
00107                 elementos.push_back(elemento_individual);
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 = 0;
00128         try {
00129
00130             valor_cru = std::stod(valor);
00131         }
00132         catch (std::invalid_argument&) {
00133
00134             std::cout « "\033[1;31mErro dentro de converter_lat_long, valor inválido para stod:
      \033[0m"
00135                       « valor
00136                       « std::endl;
00137         }
00138         double graus = floor(valor_cru / 100);
00139         double minutos = valor_cru - graus * 100;
00140         double dec = (graus + minutos / 60.0) * ( (hemisf == "S" || hemisf == "W" ) ? -1 : 1 );
00141
00142         return dec;
00143     }
00144
00148     static std::string
00149     formatar_csv(
00150         const GPSData& data
00151     ){
00152
00153         std::ostringstream oss;
00154         oss « data.time_utc « ","
00155             « std::fixed     « std::setprecision(6)
00156             « data.lat       « ","
00157             « data.lon       « ","
00158             « std::setprecision(2) « data.vel « ","
00159             « data.alt « ","
00160             « data.sat « ","
00161             « data.hdop;
00162
00163         return oss.str();
00164     }
00165
00169     void
00170     _open_serial(){
00171
00172         _fd_serial = ::open(
00173                             _porta_serial.c_str(),
00174                             O_RDONLY | O_NOCTTY | O_SYNC
00175                            );
00176
00177         if(
00178             _fd_serial < 0
00179         ){
00180
00181             throw std::runtime_error("\033[1;31mErro ao abrir porta serial do GPS\033[0m");
00182         }
00183
00184         // Verificamos a porta serial de leitura
00185         termios tty{};
00186         if(
00187             ::tcgetattr(
00188                         _fd_serial,
00189                         &tty
00190                        ) != 0
```

```
00191         ){
00192
00193             throw std::runtime_error("\033[1;31mErro ao tentar entrar na porta serial,
      especificamente, tcgetattr\033[0m");
00194         }
00195
00196         ::cfsetospeed(&tty, B115200);
00197         ::cfsetispeed(&tty, B115200);
00198
00199         // Diversas operações bit a bit
00200         tty.c_cflag = (tty.c_cflag & ~CSIZE) | CS8; // 8 bits
00201         tty.c_iflag &= ~IGNBRK;
00202         tty.c_lflag = 0;
00203         tty.c_oflag = 0;
00204         tty.c_cc[VMIN] = 1;
00205         tty.c_cc[VTIME] = 1;
00206         tty.c_cflag |= (CLOCAL | CREAD);
00207         tty.c_cflag &= ~(PARENB | PARODD);
00208         tty.c_cflag &= ~CSTOPB;
00209         tty.c_cflag &= ~CRTSCTS;
00210
00211         if(
00212             ::tcsetattr(
00213                         _fd_serial,
00214                         TCSANOW,
00215                         &tty
00216                        ) != 0
00217         ){
00218
00219             throw std::runtime_error("\033[1;31mErro ao tentar setar configurações na comunicação
      serial, especificamente, tcsetattr\033[0m");
00220         }
00221     }
00222
00228     std::string
00229     _ler_dados(){
00230
00231         char buffer[256];
00232         int ult_indice = read(
00233                               _fd_serial,
00234                               buffer,
00235                               sizeof(buffer) - 1
00236                              );
00237
00238         if( ult_indice > 0 ){
00239
00240             buffer[ult_indice] = '\0';
00241             return std::string(buffer);
00242         }
00243
00244         std::cout « "\nNada a ser lido..." « std::endl;
00245         return {};
00246     }
00247
00253     void
00254     _parser_rmc(
00255         const std::string& frase,
00256         GPSData& data
00257     ){
00258
00259         std::vector<std::string> campos_de_infos = split(frase, ',');
00260
00261         // Logo, está fora do padrão rmc.
00262         if( campos_de_infos.size() < 8 ){ return; }
00263
00264         data.time_utc = campos_de_infos[1];
00265         data.lat = converter_lat_long(campos_de_infos[3], campos_de_infos[4]);
00266         data.lon = converter_lat_long(campos_de_infos[5], campos_de_infos[6]);
00267         try {
00268
00269             data.vel = std::stod(campos_de_infos[7]) * 0.514; // Transformando de nós para m/s
00270         }
00271         catch (std::invalid_argument&) {
00272
00273             std::cout « "\033[1;31mErro ao dentro de _parser_rmc, valor inválido para stod: \033[0m"
00274                       « campos_de_infos[7]
00275                       « std::endl;
00276             data.vel = 0;
00277         }
00278     }
00279
00285     void
00286     _parser_gga(
00287         const std::string& frase,
00288         GPSData& data
00289     ){
00290
```

```
00291         std::vector<std::string> campos_de_infos = split(frase, ',');
00292
00293         // Logo, está fora do padrão gga
00294         if( campos_de_infos.size() < 10 ){ return; }
00295
00296         data.time_utc = campos_de_infos[1];
00297         data.lat = converter_lat_long(campos_de_infos[2], campos_de_infos[3]);
00298         data.lon = converter_lat_long(campos_de_infos[4], campos_de_infos[5]);
00299         data.sat  = std::stoi(campos_de_infos[7]);
00300         data.hdop = std::stod(campos_de_infos[8]);
00301         data.alt  = std::stod(campos_de_infos[9]);
00302     }
00303
00307     void
00308     _loop(){
00309
00310         while(
00311             _is_exec
00312         ){
00313
00314             std::string linha = _ler_dados();
00315             if(
00316                 !linha.empty()
00317             ){
00318
00319                 if( linha.find("RMC") != std::string::npos ){
00320
00321                     _parser_rmc(linha, _last_data_given);
00322                 }
00323                 else if( linha.find("GGA") != std::string::npos ){
00324
00325                     _parser_gga(linha, _last_data_given);
00326                 }
00327                 else{
00328
00329                     std::cout « "\033[1;31mNão consigo codificar...: \033[0m"
00330                           « linha
00331                           « std::endl;
00332                     continue;
00333                 }
00334             }
00335
00336             // Então temos os dados.
00337             std::cout « "\n\033[7mSensor Interpretando:\033[0m\n"
00338                       « formatar_csv(_last_data_given)
00339                       « std::endl;
00340         }
00341     }
00342
00343 public:
00344
00353     TrackSense(
00354         const std::string& ip_destino,
00355         int             porta_destino,
00356         const std::string& porta_serial
00357
00358     ) : _ip_destino(ip_destino),
00359         _porta_destino(porta_destino),
00360         _porta_serial(porta_serial)
00361     {
00362
00363         // As seguintes definições existentes para a comunição UDP.
00364         _sockfd = ::socket(AF_INET, SOCK_DGRAM, 0);
00365         if( _sockfd < 0 ){
00366             throw std::runtime_error("Erro ao criar socket UDP");
00367         }
00368
00369         _addr_dest.sin_family = AF_INET;
00370         _addr_dest.sin_port = ::htons(_porta_destino);
00371         _addr_dest.sin_addr.s_addr = ::inet_addr(_ip_destino.c_str());
00372
00373         _open_serial();
00374     }
00375
00381     ~TrackSense() { stop(); if( _sockfd >= 0 ){ ::close(_sockfd); } if( _fd_serial >= 0 ){
      ::close(_fd_serial); } }
00382
00386     void
00387     init(){
00388
00389         if( _is_exec.exchange(true) ){ return; }
00390
00391         std::cout « "\033[1;32mIniciando Thread de Leitura...\033[0m" « std::endl;
00392         _worker = std::thread(
00393                                 [this]{ _loop(); }
00394                               );
00395     }
```

```
00396
00400     void
00401     stop(){
00402
00403         if( !_is_exec.exchange(false) ){ return; }
00404
00405         if(
00406             _worker.joinable()
00407         ){
00408
00409             std::cout « "\033[1;32mSaindo da thread de leitura.\033[0m" « std::endl;
00410             _worker.join();
00411         }
00412     }
00413 };
00414
00415 #endif // TRACKSENSE_HPP
```

Index