

GPSTrack

Generated by Doxygen 1.9.8

1 Class Documentation	1
1.1 GPSTrack::GPSData Class Reference	1
1.1.1 Detailed Description	2
1.1.2 Constructor & Destructor Documentation	2
1.1.2.1 GPSData()	2
1.1.3 Member Function Documentation	2
1.1.3.1 converter_lat_lon()	2
1.1.3.2 parsing()	3
1.1.3.3 to_csv()	4
1.1.4 Member Data Documentation	4
1.1.4.1 data	4
1.1.4.2 pattern	4
1.2 GPSSim Class Reference	5
1.2.1 Detailed Description	6
1.2.2 Constructor & Destructor Documentation	6
1.2.2.1 GPSSim()	6
1.2.2.2 ~GPSSim()	6
1.2.3 Member Function Documentation	7
1.2.3.1 build_nmea_string()	7
1.2.3.2 degrees_to_NMEA()	7
1.2.3.3 format_integer()	8
1.2.3.4 get_path_pseudo_term()	8
1.2.3.5 get_utc_time()	9
1.2.3.6 init()	9
1.2.3.7 loop()	10
1.2.3.8 stop()	11
1.2.4 Member Data Documentation	12
1.2.4.1 alt	12
1.2.4.2 caminho_do_pseudo_terminal	12
1.2.4.3 fd_filho	12
1.2.4.4 fd_pai	12
1.2.4.5 is_exec	12
1.2.4.6 lat	12
1.2.4.7 lon	12
1.2.4.8 worker	12
1.3 GPSTrack Class Reference	13
1.3.1 Detailed Description	14
1.3.2 Constructor & Destructor Documentation	14
1.3.2.1 GPSTrack()	14
1.3.2.2 ~GPSTrack()	15
1.3.3 Member Function Documentation	15
1.3.3.1 init()	15

1.3.3.2 loop()	16
1.3.3.3 open_serial()	17
1.3.3.4 read_serial()	18
1.3.3.5 send()	18
1.3.3.6 split()	19
1.3.3.7 stop()	19
1.3.4 Member Data Documentation	20
1.3.4.1 addr_dest	20
1.3.4.2 fd_serial	20
1.3.4.3 ip_destino	20
1.3.4.4 is_exec	20
1.3.4.5 last_data_given	20
1.3.4.6 porta_destino	20
1.3.4.7 porta_serial	21
1.3.4.8 sockfd	21
1.3.4.9 worker	21
1.4 GPSSim::NMEAGenerator Class Reference	21
1.4.1 Detailed Description	21
1.4.2 Member Function Documentation	21
1.4.2.1 generate_gga()	21
1.4.2.2 generate_rmc()	22
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/GPSTrack.hpp File Reference	31
2.4.1 Detailed Description	32
2.5 GPSTrack.hpp	32
2.6 src/main.cpp File Reference	37
2.6.1 Detailed Description	37
2.6.2 Function Documentation	37
2.6.2.1 main()	37
Index	39

Chapter 1

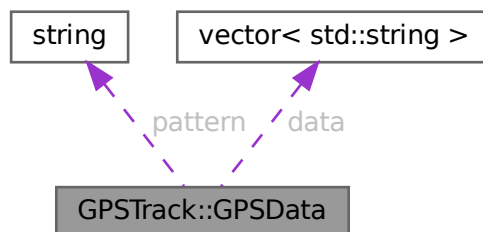
Class Documentation

1.1 GPSTrack::GPSData Class Reference

Classe responsável por representar os dados do GPS.

```
#include <GPSTrack.hpp>
```

Collaboration diagram for GPSTrack::GPSData:



Public Member Functions

- [GPSData](#) ()
Construtor Default.
- bool [parsing](#) (int code_pattern, const std::vector< std::string > &data splitted)
Setará os dados baseado no padrão de mensagem recebido.
- std::string [to_csv](#) () const
Retorna os dados armazenados em formato CSV.

Static Public Member Functions

- static std::string [converter_lat_lon](#) (const std::string &string_numerica, const std::string &string_hemisf)
Função estática auxiliar para converter coordenadas NMEA (latitude/longitude) para graus decimais.

Private Attributes

- `std::string pattern`
- `std::vector< std::string > data`

Organizaremos os dados neste vetor.

1.1.1 Detailed Description

Classe responsável por representar os dados do GPS.

Utilizar struct mostrou-se básico demais para atender as necessidades de representação dos dados do GPS. Com isso, a evolução para Class tornou-se inevitável.

O sensor inicia a comunicação enviando mensagens do tipo \$GPTXT. Essas mensagens são mensagens de texto informativas, geralmente vazias. Esse é um comportamento normal nos primeiros segundos após energizar o módulo.

Após adquirir sinais de satélites e iniciar a navegação, o módulo passa a enviar sentenças NMEA padrão, que trazem informações úteis:

- \$GPRMC (Recommended Minimum Navigation Information): Fornece dados essenciais de navegação, como horário UTC, status, latitude, longitude, velocidade, curso e data.
- \$GPVTG (Course over Ground and Ground Speed): Informa direção do movimento (rumo) e velocidade sobre o solo.
- \$GPGGA (Global Positioning System Fix Data): Dados de fixação do GPS, incluindo número de satélites usados, qualidade do sinal, altitude e posição.
- \$GPGSA (GNSS DOP and Active Satellites): Status dos satélites em uso e precisão (DOP - Dilution of Precision).
- \$GPGSV (GNSS Satellites in View): Informações sobre os satélites visíveis, como elevação, azimute e intensidade de sinal.
- \$GPGLL (Geographic Position - Latitude/Longitude): Posição geográfica em latitude e longitude, com horário associado.

Sendo assim, o sensor sai de um modo de inicialização para operacionalidade completa.

Como nosso propósito é apenas localização, nos interessa apenas o padrão GGA, o qual oferece dados profundos de localização.

1.1.2 Constructor & Destructor Documentation

1.1.2.1 GPSTData()

```
GPSTrack::GPSTData::GPSTData ( ) [inline]
```

Construtor Default.

Reserva no vetor de dados 4 strings, respectivamente para horário UTC, latitude, longitude e altitude.

1.1.3 Member Function Documentation

1.1.3.1 converter_lat_lon()

```
static std::string GPSTrack::GPSTData::converter_lat_lon (
    const std::string & string_numerica,
    const std::string & string_hemisf ) [inline], [static]
```

Função estática auxiliar para converter coordenadas NMEA (latitude/longitude) para graus decimais.

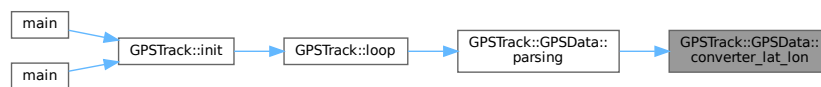
Parameters

<i>string_numerica</i>	String com a coordenada em formato NMEA (ex: "2257.34613").
<i>string_hemisf</i>	String com o hemisfério correspondente ("N", "S", "E", "W").

Returns

String coordenada em graus decimais (negativa para hemisférios Sul e Oeste).

Here is the caller graph for this function:



1.1.3.2 parsing()

```

bool GPSTrack::GPSData::parsing (
    int code_pattern,
    const std::vector< std::string > & data splitted ) [inline]
  
```

Setará os dados baseado no padrão de mensagem recebido.

Parameters

<i>code_pattern</i>	Código para informar que padrão de mensagem recebeu.
<i>data splitted</i>	Vetor de dados da mensagem recebida.

Returns

Retornará true caso seja bem sucedido. False, caso contrário.

Tradução de códigos:

- 0 == GPGGA

A partir do padrão de mensagem recebida, organizaremos o vetor com dados relevantes.

Here is the call graph for this function:



Here is the caller graph for this function:



1.1.3.3 to_csv()

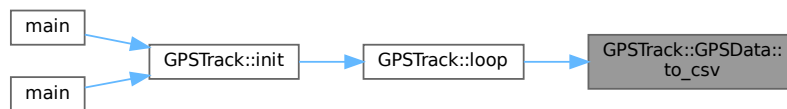
```
std::string GPSTrack::GPSData::to_csv ( ) const [inline]
```

Retorna os dados armazenados em formato CSV.

Returns

std::string Linha CSV com os valores.

Here is the caller graph for this function:



1.1.4 Member Data Documentation

1.1.4.1 data

```
std::vector<std::string> GPSTrack::GPSData::data [private]
```

Organizaremos os dados neste vetor.

1.1.4.2 pattern

```
std::string GPSTrack::GPSData::pattern [private]
```

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

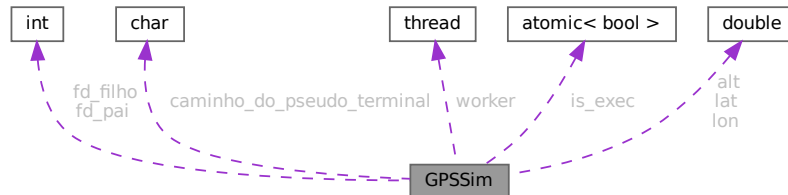
- [src/GPSTrack.hpp](#)

1.2 GPSSim Class Reference

Versão Simulada do Sensor GPS, gerando frases no padrão NMEA.

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

Collaboration diagram for GPSSim:



Classes

- class [NMEAGenerator](#)
Classe responsável por agrupar as funções geradoras de sentenças NMEA.

Public Member Functions

- [GPSSim](#) (double latitude_inicial_graus, double longitude_inicial_graus, double altitude_metros)
Construtor do [GPSSim](#).
- [~GPSSim](#) ()
Destrutor do [GPSSim](#).
- 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.
- std::string [get_path_pseudo_term](#) () const
Obtém o caminho do terminal que estamos executando de forma filial.

Private Member Functions

- void [loop](#) ()
Loop principal responsável pela geração e transmissão de dados simulados.

Static Private Member Functions

- static void [degrees_to_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 [format_integer](#) (int valor, int quant_digitos)
Formatada um número inteiro com dois dígitos, preenchendo com zero à esquerda.
- static std::tm [get_utc_time](#) ()
Obtém o tempo UTC atual, horário em Londres.
- static std::string [build_nmea_string](#) (const std::string &corpo_frase)
Finaliza uma sentença NMEA a partir do corpo da frase.

Private Attributes

- int `fd_pai` {0}
- int `fd_filho` {0}
- char `caminho_do_pseudo_terminal` [128]
- std::thread `worker`
- std::atomic< bool > `is_exec` {false}
- double `lat`
- double `lon`
- double `alt`

1.2.1 Detailed Description

Versão Simulada do Sensor GPS, gerando frases no padrão NMEA.

Criará um par de pseudo-terminais (PTY) para simular o funcionamento do sensor. Intervaladamente, gera frases NMEA simuladas.

1.2.2 Constructor & Destructor Documentation

1.2.2.1 GPSSim()

```
GPSSim::GPSSim (
    double latitude_inicial_graus,
    double longitude_inicial_graus,
    double altitude_metros ) [inline]
```

Construtor do [GPSSim](#).

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

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.

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 build_nmea_string()

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

Finaliza uma sentença NMEA a partir do corpo da frase.

Calcula o valor de paridade (checksum) do corpo da frase fornecida, em seguida adiciona os delimitadores e flags no formato NMEA (prefixo '\$', sufixo '*', valor de paridade em hexadecimal e "\r\n").

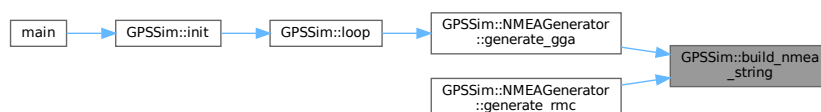
Parameters

<i>corpo_frase</i>	Corpo da frase NMEA sem os indicadores iniciais ('\$') e finais ('*' e checksum).
--------------------	---

Returns

std::string Sentença NMEA completa, pronta para transmissão.

Here is the caller graph for this function:



1.2.3.2 degrees_to_NMEA()

```
static void GPSSim::degrees_to_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:



1.2.3.3 format_integer()

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

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

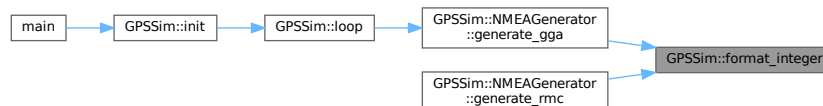
Parameters

<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:



1.2.3.4 get_path_pseudo_term()

```
std::string GPSSim::get_path_pseudo_term ( ) 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:

**1.2.3.5 get_utc_time()**

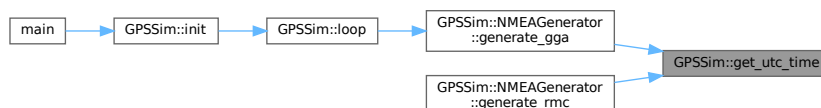
```
static std::tm GPSSim::get_utc_time ( ) [inline], [static], [private]
```

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:

**1.2.3.6 init()**

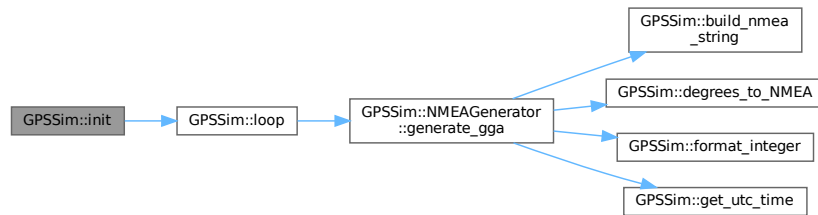
```
void GPSSim::init ( ) [inline]
```

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

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:



1.2.3.7 loop()

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

Loop principal responsável pela geração e transmissão de dados simulados.

Esta função executa um laço contínuo enquanto o simulador estiver ativo (`_is_exec`). Em cada iteração:

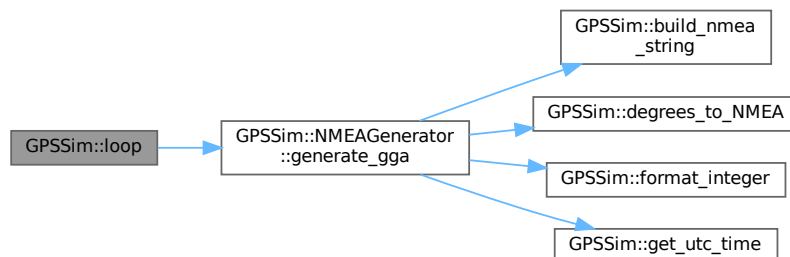
- Inicializa ou atualiza a posição simulada (latitude e longitude).
- Gera uma sentença NMEA do tipo GGA a partir da posição atual.
- Transmite a sentença gerada através do descritor de escrita `_fd_pai`.
- Aguarda o período de atualização definido em `_periodo_atualizacao`.
- Atualiza a posição simulada chamando `_update_position()`.

O loop termina automaticamente quando `_is_exec` é definido como falso.

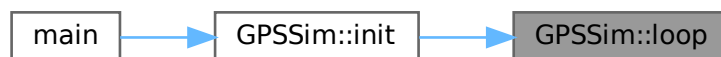
Note

Esta função é bloqueante e deve ser executada em uma thread dedicada para não interromper o fluxo principal do programa.

Here is the call graph for this function:



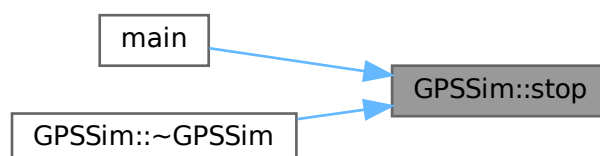
Here is the caller graph for this function:

**1.2.3.8 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 caminho_do_pseudo_terminal

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

1.2.4.3 fd_filho

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

1.2.4.4 fd_pai

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

1.2.4.5 is_exec

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

1.2.4.6 lat

```
double GPSSim::lat [private]
```

1.2.4.7 lon

```
double GPSSim::lon [private]
```

1.2.4.8 worker

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

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

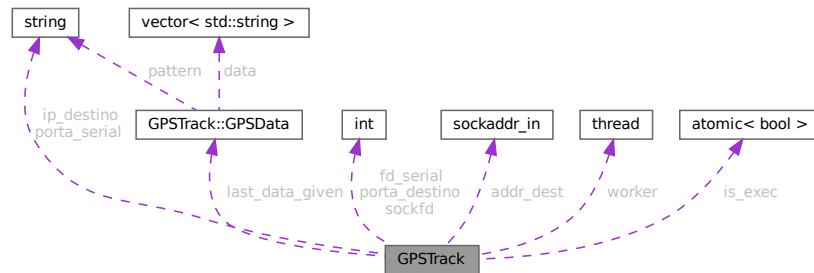
- [src/GPSSim.hpp](#)

1.3 GPSTrack Class Reference

Classe responsável por obter o tracking da carga.

```
#include <GPSTrack.hpp>
```

Collaboration diagram for GPSTrack:



Classes

- class [GPSTrackData](#)
Classe responsável por representar os dados do GPS.

Public Member Functions

- [GPSTrack](#) (const std::string &ip_destino_, int porta_destino_, const std::string &porta_serial_)
Construtor da Classe.
- [~GPSTrack](#) ()
Destrutor da classe.
- void [init](#) ()
Inicializa a thread trabalhadora.
- void [stop](#) ()
Finaliza a thread de trabalho de forma segura.

Static Public Member Functions

- static std::vector< std::string > [split](#) (const std::string &string_de_entrada, char separador=',')
Função estática auxiliar para separar uma string em vetores de string.

Private Member Functions

- void [open_serial](#) ()
Abre e configura a porta serial para comunicação o sensor.
- std::string [read_serial](#) ()
Lê dados da porta serial até encontrar uma quebra de linha.
- bool [send](#) (const std::string &mensagem)
Envia uma string via socket UDP para um servidor.
- void [loop](#) ()
Executa o loop principal de leitura, interpretação e envio de dados via UDP.

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

Classe responsável por obter o tracking da carga.

Responsabilidades:

- Obter os dados do sensor NEO6MV2
- Interpretar esses dados, gerando informações
- Enviar as informações via socket UDP em formato CSV

Cada uma dessas responsabilidades está associada a um método da classe, respectivamente:

- `read_serial()`
- `GPSTData::parsing()`
- `send()`

Os quais estarão sendo repetidamente executados pela thread worker a fim de manter a continuidade de informações.

Não há necessidade de mais explicações, já que o fluxo de funcionamento é simples.

1.3.2 Constructor & Destructor Documentation

1.3.2.1 GPSTrack()

```
GPSTrack::GPSTrack (
    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 a comunicação UDP e abre a comunicação serial. Here is the call graph for this function:



1.3.2.2 ~GPSTrack()

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

Destrutor da classe.

Realiza a limpeza adequada dos recursos da classe, garantindo o término seguro das operações.

A ordem de operações é importante:

1. Interrompe a thread de execução
2. Fecha o socket de comunicação
3. Fecha a porta serial

Here is the call graph for this function:



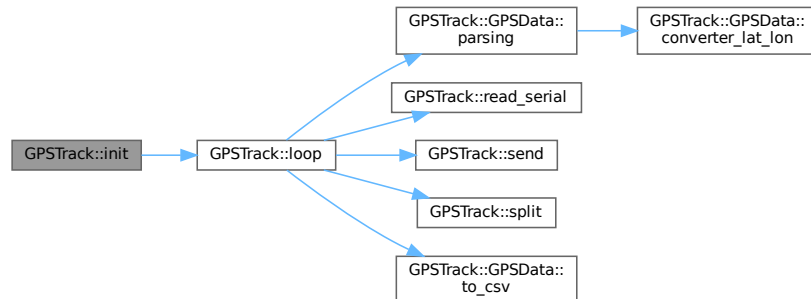
1.3.3 Member Function Documentation

1.3.3.1 init()

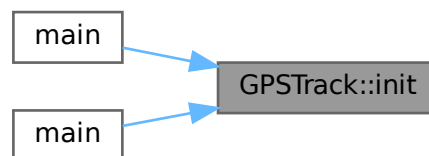
```
void GPSTrack::init ( ) [inline]
```

Inicializa a thread trabalhadora.

Garante que apenas uma única instância da thread de execução será iniciada, utilizando um flag atômico para controle de estado. Se a thread já estiver em execução, a função retorna imediatamente sem realizar nova inicialização. Ao iniciar, exibe uma mensagem colorida no terminal e cria uma thread worker que executa o loop principal de leitura e comunicação. Here is the call graph for this function:



Here is the caller graph for this function:



1.3.3.2 loop()

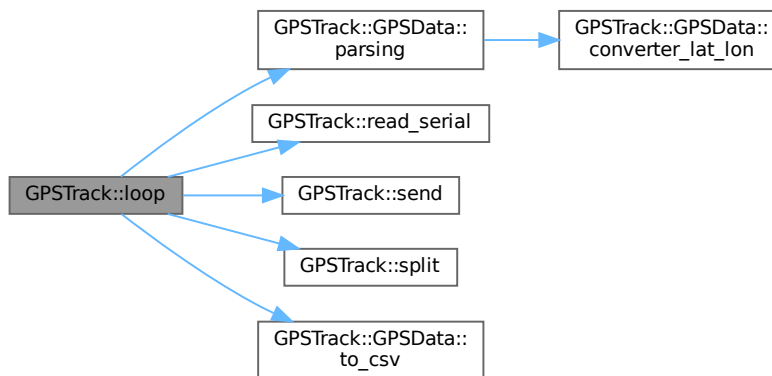
```
void GPSTrack::loop ( ) [inline], [private]
```

Executa o loop principal de leitura, interpretação e envio de dados via UDP.

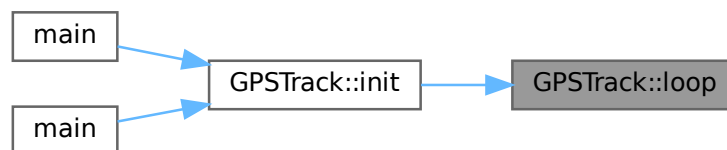
Esta função realiza continuamente a leitura de dados da porta serial, interpreta as mensagens GPS no formato GPGGA, armazena os dados processados e, se desejado, os exibe em formato CSV.

O loop executa enquanto a flag de execução estiver ativa, com uma pausa de 1 segundo entre cada iteração para

evitar consumo excessivo de CPU. Here is the call graph for this function:



Here is the caller graph for this function:



1.3.3.3 open_serial()

```
void GPSTrack::open_serial ( ) [inline], [private]
```

Abre e configura a porta serial para comunicação o sensor.

Estabelece a conexão serial utilizando a porta serial especificada. Todos os parâmetros necessários para uma comunicação estável com o dispositivo, incluindo velocidade, formato de dados e controle de fluxo são setados.

A porta é aberta em modo somente leitura (O_RDONLY) e em modo raw, no qual não há processamento adicional dos caracteres.

Aplicamos as seguintes configurações:

- 9600 bauds
- 8 bits de dados
- Sem paridade

- 1 bit de parada

Here is the caller graph for this function:



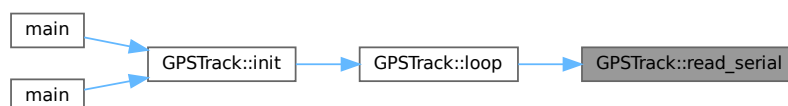
1.3.3.4 read_serial()

```
std::string GPSTrack::read_serial ( ) [inline], [private]
```

Lê dados da porta serial até encontrar uma quebra de linha.

- Caracteres de carriage return ('\r') são ignorados durante a leitura.
- A função termina quando encontra '\n' ou quando não há mais dados para ler.
- A leitura é feita caractere por caractere para garantir processamento correto dos dados do GPS que seguem protocolo NMEA.

Here is the caller graph for this function:



1.3.3.5 send()

```
bool GPSTrack::send (
    const std::string & mensagem ) [inline], [private]
```

Envia uma string via socket UDP para um servidor.

Parameters

<i>mensagem</i>	String a ser enviada.
-----------------	-----------------------

Returns

True se a mensagem foi enviada com sucesso. False, caso contrário.

Here is the caller graph for this function:

**1.3.3.6 split()**

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

Função estática auxiliar para separar uma string em vetores de string.

Parameters

<i>string_de_entrada</i>	String que será fatiada.
<i>separador</i>	Caractere que será a flag de separação.

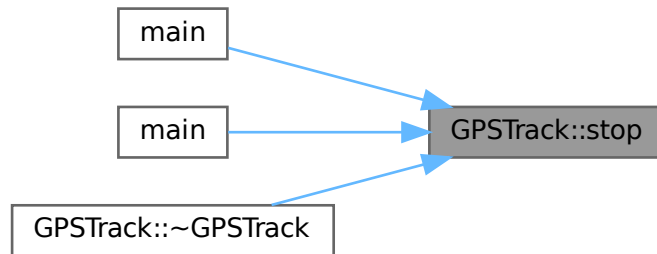
Similar ao método `split` do python, utiliza ',' como caractere separador default. Here is the caller graph for this function:

**1.3.3.7 stop()**

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

Finaliza a thread de trabalho de forma segura.

Esta função realiza o desligamento controlado da thread de trabalho. Primeiro, altera o flag de execução para falso usando operação atômica. Se a thread estiver joinable (executando), imprime uma mensagem de confirmação e realiza a operação de join para aguardar a finalização segura da thread. Here is the caller graph for this function:



1.3.4 Member Data Documentation

1.3.4.1 addr_dest

```
sockaddr_in GPSTrack::addr_dest {} [private]
```

1.3.4.2 fd_serial

```
int GPSTrack::fd_serial = -1 [private]
```

1.3.4.3 ip_destino

```
std::string GPSTrack::ip_destino [private]
```

1.3.4.4 is_exec

```
std::atomic<bool> GPSTrack::is_exec {false} [private]
```

1.3.4.5 last_data_given

```
GPSTData GPSTrack::last_data_given [private]
```

1.3.4.6 porta_destino

```
int GPSTrack::porta_destino [private]
```


1.3.4.7 porta_serial

```
std::string GPSTrack::porta_serial [private]
```

1.3.4.8 sockfd

```
int GPSTrack::sockfd [private]
```

1.3.4.9 worker

```
std::thread GPSTrack::worker [private]
```

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

- [src/GPSTrack.hpp](#)

1.4 GPSSim::NMEAGenerator Class Reference

Classe responsável por agrupar as funções geradoras de sentenças NMEA.

Static Public Member Functions

- static std::string [generate_gga](#) (int lat_graus, int lon_graus, int alt_metros)
Gera uma frase GGA (Global Positioning System Fix Data)
- static std::string [generate_rmc](#) (int lat_graus, int lon_graus, int alt_metros)
Gera uma frase GGA (Recommended Minimum Navigation Information)

1.4.1 Detailed Description

Classe responsável por agrupar as funções geradoras de sentenças NMEA.

Contém apenas os métodos estáticos que constroem a informação a ser posta na string NMEA.

1.4.2 Member Function Documentation

1.4.2.1 generate_gga()

```
static std::string GPSSim::NMEAGenerator::generate_gga (  
    int lat_graus,  
    int lon_graus,  
    int alt_metros ) [inline], [static]
```

Gera uma frase GGA (Global Positioning System Fix Data)

Apesar de usar apenas valores de lat, long e alt, zera os demais valores.

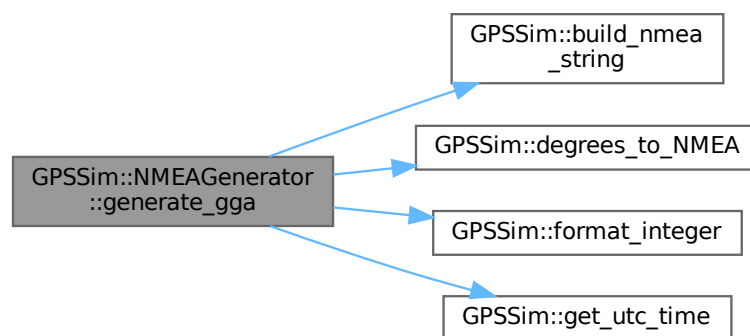
Parameters

<i>lat_graus</i>	Latitude em graus decimais
<i>lon_graus</i>	Longitude em graus decimais
<i>alt_metros</i>	Altitude em metros

Returns

String correspondendo ao corpo de frase GGA

Here is the call graph for this function:



Here is the caller graph for this function:



1.4.2.2 generate_rmc()

```
static std::string GPSSim::NMEAGenerator::generate_rmc (
    int lat_graus,
    int lon_graus,
    int alt_metros ) [inline], [static]
```

Gera uma frase GGA (Recommended Minimum Navigation Information)

Apesar de usar apenas valores de lat, long e alt, zera os demais valores.

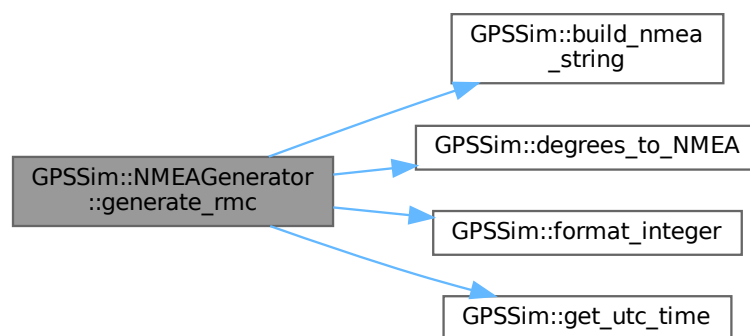
Parameters

<i>lat_graus</i>	Latitude em graus decimais
<i>lon_graus</i>	Longitude em graus decimais
<i>alt_metros</i>	Altitude em metros

Returns

String correspondendo ao corpo de frase GGA

Here is the call graph for this function:



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

- [src/GPSSim.hpp](#)

Chapter 2

File Documentation

2.1 src/debug.cpp File Reference

Responsável por prover ferramentas de debug.

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



Functions

- int [main](#) ()

2.1.1 Detailed Description

Responsável por prover ferramentas de debug.

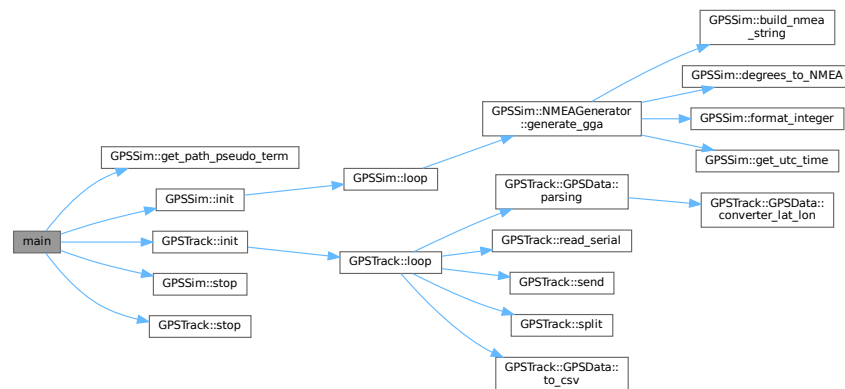
Já que a aplicação deve ser executada dentro da placa, não conseguiríamos executá-la no DeskTop. Para tanto, fez-se necessário o desenvolvimento de ferramentas que possibilitam a depuração de nosso código.

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.

```

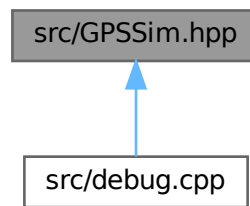
#include <chrono>
#include <ctime>
#include <cmath>
#include <vector>
#include <string>
#include <sstream>
#include <iomanip>
#include <thread>
#include <atomic>
#include <stdexcept>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <pty.h>

```

Include dependency graph for GPSSim.hpp:



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



Classes

- class [GPSSim](#)
Versão Simulada do Sensor GPS, gerando frases no padrão NMEA.
- class [GPSSim::NMEAGenerator](#)
Classe responsável por agrupar as funções geradoras de sentenças 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
00064  class GPSSim {
00065  private:
00066
00067      // Informações ligadas ao terminal
00068      int fd_pai{0}, fd_filho{0};
00069      char caminho_do_pseudo_terminal[128];
00070
00071      // Thread de Execução Paralela e Flag de Controle
00072      std::thread worker;
00073      std::atomic<bool> is_exec{false};
00074
00075      // Informações de Localização
00076      double lat, lon, alt;
00077
00085      static void
00086      degrees_to_NMEA(
00087          double graus_decimais,
00088          bool is_lat,
00089          std::string& ddmm,
00090          char& hemisf
00091      ){
00092
00093          hemisf = (is_lat) ? (
00094              (graus_decimais >= 0) ? 'N' : 'S'
00095              ) :
00096              (
00097                  (graus_decimais >= 0) ? 'E' : 'W'
00098              );
00099
00100          double valor_abs = std::fabs(graus_decimais);
00101          int graus = static_cast<int>(std::floor(valor_abs));
00102          double min = (valor_abs - graus) * 60.0;
00103
00104          // Não utilizamos apenas a função de formatar_inteiro, pois
00105          // utilizaremos o oss em seguida.
00106          std::ostringstream oss;
00107          if(
00108              is_lat
00109          ){
00110
00111              oss << std::setw(2)
00112                  << std::setfill('0')
00113                  << graus
00114                  << std::fixed
00115                  << std::setprecision(4)
00116                  << std::setw(7)
00117                  << std::setfill('0')
00118                  << min;
00119          }
00120          else{
00121
00122              oss << std::setw(3)
00123                  << std::setfill('0')
00124                  << graus
00125                  << std::fixed
00126                  << std::setprecision(4)
00127                  << std::setw(7)
00128                  << std::setfill('0')
00129                  << min;
00130          }
00131
00132          ddmm = oss.str();
00133      }
00134

```



```

00141     static std::string
00142     format_integer(
00143         int valor,
00144         int quant_digitos
00145     ){
00146
00147         std::ostringstream oss;
00148         oss << std::setw(quant_digitos)
00149             << std::setfill('0')
00150             << valor;
00151         return oss.str();
00152     }
00153
00154     static std::tm
00155     get_utc_time(){
00156
00157         using namespace std::chrono;
00158         auto tempo_atual = system_clock::to_time_t(system_clock::now());
00159         std::tm tempo_utc{};
00160         gmtime_r(&tempo_atual, &tempo_utc);
00161         return tempo_utc;
00162     }
00163
00164     static std::string
00165     build_nmea_string(
00166         const std::string& corpo_frase
00167     ){
00168
00169         uint8_t paridade = 0;
00170         for(
00171             const char& caract : corpo_frase
00172         ){
00173
00174             paridade ^= (uint8_t)caract;
00175         }
00176
00177         std::ostringstream oss;
00178         oss << '$'
00179             << corpo_frase
00180             << '*'
00181             << std::uppercase
00182             << std::hex
00183             << std::setw(2)
00184             << std::setfill('0')
00185             << (int)paridade
00186             << "\r\n";
00187
00188         return oss.str();
00189     }
00190
00191     class NMEAGenerator {
00192     public:
00193
00194         static std::string
00195         generate_gga(
00196             int lat_graus,
00197             int lon_graus,
00198             int alt_metros
00199         ){
00200
00201             auto tempo_utc = get_utc_time();
00202             std::string lat_nmea, lon_nmea;
00203             char hemisferio_lat, hemisferio_lon;
00204
00205             degrees_to_NMEA(
00206                 lat_graus,
00207                 true,
00208                 lat_nmea,
00209                 hemisferio_lat
00210             );
00211             degrees_to_NMEA(
00212                 lon_graus,
00213                 false,
00214                 lon_nmea,
00215                 hemisferio_lon
00216             );
00217
00218             // Formato: hhhmss.ss,lat,N/S,lon,E/W,qualidade,satelites,HDOP,altitude,M,...
00219             std::ostringstream oss;
00220             oss << "GPGGA," << format_integer(tempo_utc.tm_hour, 2)
00221                 << format_integer(tempo_utc.tm_min, 2)
00222                 << format_integer(tempo_utc.tm_sec, 2) << ".00,"
00223                 << lat_nmea << "," << hemisferio_lat << ","
00224                 << lon_nmea << "," << hemisferio_lon << ",1,"
00225                 << "-1 << "," << std::fixed << std::setprecision(1) << "-1 << ","
00226                 << std::fixed << std::setprecision(1) << alt_metros << ",M,0.0,M,";
00227
00228             return oss.str();
00229         }
00230     };

```

```

00260         return build_nmea_string(oss.str());
00261     }
00262
00274     static std::string
00275     generate_rmc(
00276         int lat_graus,
00277         int lon_graus,
00278         int alt_metros
00279     ){
00280
00281         auto tempo_utc = get_utc_time();
00282         std::string lat_nmea, lon_nmea;
00283         char hemisferio_lat, hemisferio_lon;
00284
00285         degrees_to_NMEA(
00286             lat_graus,
00287             true,
00288             lat_nmea,
00289             hemisferio_lat
00290         );
00291         degrees_to_NMEA(
00292             lon_graus,
00293             false,
00294             lon_nmea,
00295             hemisferio_lon
00296         );
00297
00298         // Formato: hhhmss.ss,A,lat,N/S,lon,E/W,velocidade,curso,data,,
00299         std::ostringstream oss;
00300         oss << "GPRMC,"
00301             << format_integer(tempo_utc.tm_hour, 2)
00302             << format_integer(tempo_utc.tm_min, 2)
00303             << format_integer(tempo_utc.tm_sec, 2) << ".00,A,"
00304             << lat_nmea << "," << hemisferio_lat << ","
00305             << lon_nmea << "," << hemisferio_lon << ","
00306             << std::fixed << std::setprecision(2) << "-1 << ",0.00,"
00307             << format_integer(tempo_utc.tm_mday, 2)
00308             << format_integer(tempo_utc.tm_mon + 1, 2)
00309             << format_integer((tempo_utc.tm_year + 1900) % 100, 2)
00310             << ",,A";
00311
00312         return build_nmea_string(oss.str());
00313     }
00314 };
00315
00334 void
00335 loop(){
00336
00337     while (is_exec){
00338
00339
00340         std::string saida = NMEAGenerator::generate_gga(lat, lon, alt);
00341         std::cout << "\033[7mGPS6MV2 Simulado Emitindo:\033[0m \n" << saida << std::endl;
00342
00343         // Imprimimos no terminal serial
00344         (void)!::write(fd_pai, saida.data(), saida.size());
00345
00346         // Aguarda o próximo ciclo
00347         std::this_thread::sleep_for(std::chrono::seconds(1));
00348     }
00349 }
00350
00351 public:
00352
00363     GPSSim(
00364         double latitude_inicial_graus,
00365         double longitude_inicial_graus,
00366         double altitude_metros
00367     ) : lat(latitude_inicial_graus),
00368        lon(longitude_inicial_graus),
00369        alt(altitude_metros)
00370     {
00371
00372         // Cria o par de pseudo-terminais
00373         if(
00374             ::openpty( &fd_pai, &fd_filho, caminho_do_pseudo_terminal, nullptr, nullptr ) != 0
00375         ){
00376             throw std::runtime_error("Falha ao criar pseudo-terminal");
00377         }
00378
00379         // Configura o terminal filho para simular o módulo real (9600 8N1)
00380         termios config_com{}; // Cria a estrutura vazia
00381         ::tcgetattr(fd_filho, &config_com); // Lê as configurações atuais e armazena na struct
00382         ::cfsetispeed(&config_com, B9600); // Definimos velocidade de entrada e de saída
00383         ::cfsetospeed(&config_com, B9600); // Essa constante está presente dentro do termios.h
00384         // Diversas operações bits a bits
00385         config_com.c_cflag = (config_com.c_cflag & ~CSIZE) | CS8;

```

```

00386         config_com.c_cflag |= (CLOCAL | CREAD);
00387         config_com.c_cflag &= ~(PARENB | CSTOPB);
00388         config_com.c_iflag = IGNPAR;
00389         config_com.c_oflag = 0;
00390         config_com.c_lflag = 0;
00391         tcsetattr(fd_filho, TCSANOW, &config_com); // Aplicamos as configurações
00392
00393         // Fecha o filho - será aberto pelo usuário no caminho correto
00394         // Mantemos o Pai aberto para procedimentos posteriores
00395         ::close(fd_filho);
00396     }
00397
00404     ~GPSSim(){ stop(); if( fd_pai >= 0 ){ ::close(fd_pai); } }
00405
00414     void
00415     init(){
00416
00417         if( is_exec.exchange(true) ){ return; }
00418
00419         worker = std::thread(
00420             [this]{ loop(); }
00421             );
00422     }
00423
00431     void
00432     stop(){
00433
00434         if( !is_exec.exchange(false) ){ return; }
00435
00436         if(
00437             worker.joinable()
00438         ){
00439             worker.join();
00440         }
00441     }
00442
00443     std::string
00444     get_path_pseudo_term() const { return std::string(caminho_do_pseudo_terminal); }
00450 };
00451
00452 #endif // GPSSim_HPP

```

2.4 src/GPSTrack.hpp File Reference

Implementação da solução embarcada.

```

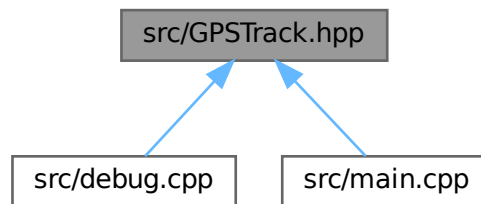
#include <string>
#include <vector>
#include <sstream>
#include <iomanip>
#include <iostream>
#include <cstring>
#include <chrono>
#include <cmath>
#include <ctime>
#include <thread>
#include <atomic>
#include <stdexcept>
#include <fcntl.h>
#include <termios.h>
#include <unistd.h>
#include <sys/socket.h>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

```

Include dependency graph for GPSTrack.hpp:



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



Classes

- class [GPSTrack](#)
Classe responsável por obter o tracking da carga.
- class [GPSTrack::GPSData](#)
Classe responsável por representar os dados do GPS.

2.4.1 Detailed Description

Implementação da solução embarcada.

2.5 GPSTrack.hpp

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

```

00001
00005 #ifndef GPSTRACK_HPP
00006 #define GPSTRACK_HPP
00007
00008
00009 //-----
00010 #include <string>
00011 #include <vector>
00012 #include <sstream>
00013 #include <iomanip>
00014 #include <iostream>
00015 #include <cstring>
00016
00017 #include <chrono>
00018 #include <cmath>
00019 #include <ctime>
00020
00021 #include <thread>
00022 #include <atomic>
00023
00024 #include <stdexcept>
00025
00026 // Específicos de Sistemas Linux
00027 #include <fcntl.h>
00028 #include <termios.h>
00029 #include <unistd.h>
00030 #include <sys/socket.h>
00031 #include <arpa/inet.h>
00032 #include <netinet/in.h>
00033
00054 class GPSTrack {
00055 public:

```

```

00056
00099     class GPSTData {
00100     private:
00101
00102         std::string pattern;
00103         std::vector<std::string> data;
00104
00105     public:
00106
00114         GPSTData() { data.reserve(4); }
00115
00122         static std::string
00123         converter_lat_lon(
00124             const std::string& string_numerica,
00125             const std::string& string_hemisf
00126         ){
00127
00128             if( string_numerica.empty() ){ return ""; }
00129
00130             double valor_cru = 0;
00131             try {
00132
00133                 valor_cru = std::stod(string_numerica);
00134             }
00135             catch (std::invalid_argument&) {
00136
00137                 std::cout << "\033[1;31mErro dentro de converter_lat_lon, valor inválido para stod:
00138 \033[0m"
00139                             << string_numerica
00139                             << std::endl;
00140             }
00141
00142             // Parsing dos valores
00143             double graus = floor(valor_cru / 100);
00144             double minutos = valor_cru - graus * 100;
00145             double coordenada = (graus + minutos / 60.0) * ( (string_hemisf == "S" || string_hemisf ==
"W" ) ? -1 : 1 );
00146
00147             return std::to_string(coordenada);
00148         }
00149
00163         bool
00164         parsing(
00165             int code_pattern,
00166             const std::vector<std::string>& dataSplitted
00167         ){
00168
00169             data.clear(); // Garantimos que está limpo.
00170
00171             if(
00172                 code_pattern == 0
00173             ){
00174                 // Em gga, os dados corretos estão em:
00175
00176                 int idx_data_useful[] = {
00177                     1, // Horário UTC
00178                     2, // Latitude em NMEA
00179                     4, // Longitude em NMEA
00180                     9 // Altitude
00181                 };
00182
00183                 for(
00184                     const auto& idx : idx_data_useful
00185                 ){
00186
00187                     if( idx == 2 || idx == 4 ){
00188                         data.emplace_back(
00189                             std::move( converter_lat_lon(
00190                                 dataSplitted[idx],
00191                                 dataSplitted[idx + 1]
00192                             ))
00193                         );
00194                     }
00195                     else{
00196                         // Então basta adicionar
00197                         data.emplace_back(
00198                             // Método bizurado para não realizarmos cópias
00199                             std::move(dataSplitted[idx])
00200                         );
00201                     }
00202                 }
00203
00204                 return true;
00205             }
00206             // ... podemos escalar para novos padrões de mensagem
00207
00208             return false;

```

```

00209     }
00210
00215     std::string
00216     to_csv() const {
00217
00218         std::ostringstream oss;
00219         for (
00220             int i = 0;
00221             i < 4;
00222             i++
00223         ){
00224             if(i > 0){ oss << ","; }
00225
00226             oss << data[i];
00227         }
00228
00229         oss << "\n";
00230
00231         return oss.str();
00232     }
00233 };
00234
00242 static std::vector<std::string>
00243 split(
00244     const std::string& string_de_entrada,
00245     char separador=',')
00246 ){
00247
00248     std::vector<std::string> elementos;
00249     std::stringstream ss(string_de_entrada);
00250
00251     std::string elemento_individual;
00252     while(
00253         std::getline(
00254             ss,
00255             elemento_individual,
00256             separador
00257         )
00258     ){
00259         if(
00260             !elemento_individual.empty()
00261         ){
00262
00263             elementos.push_back(elemento_individual);
00264         }
00265     }
00266
00267     return elementos;
00268 }
00269
00270 private:
00271     // Relacionadas ao Envio UDP
00272     std::string ip_destino;
00273     int porta_destino;
00274     int sockfd;
00275     sockaddr_in addr_dest{};
00276
00277     // Relacionados ao fluxo de funcionamento
00278     std::thread worker;
00279     std::atomic<bool> is_exec{false};
00280
00281     // Relacionados à comunicação com o sensor
00282     GPSPData last_data_given;
00283     std::string porta_serial;
00284     int fd_serial = -1;
00285
00304 void
00305 open_serial(){
00306
00307     fd_serial = ::open(
00308         porta_serial.c_str(),
00309         // O_RDONLY: garante apenas leitura
00310         // O_NOCTTY: impede que a porta se torne o terminal controlador do
00311         processo
00312         // O_SYNC: garante que as operações de escrita sejam completadas
00313         fisicamente
00314         O_RDONLY | O_NOCTTY | O_SYNC
00315     );
00316
00317     // Confirmação de sucesso
00318     if( fd_serial < 0 ){ throw std::runtime_error("\033[1;31mErro ao abrir porta serial do
GPS\033[0m"); }
00319
00318     // Struct para armazenarmos os parâmetros da comunicação serial.
00319     termios tty{};
00320     if(
00321         ::tcgetattr(

```

```

00322         fd_serial,
00323         &tty
00324     ) != 0
00325 ) { throw std::runtime_error("\033[1;31mErro ao tentar configurar a porta serial,
    especificamente, tcgetattr\033[0m"); }
00326
00327     // Setamos velocidade
00328     ::cfsetospeed(&tty, B9600);
00329     ::cfsetispeed(&tty, B9600);
00330
00331     // Modo raw para não haver processamento por parte do sensor.
00332     ::cfmakeraw(&tty);
00333
00334     // Configuração 8N1
00335     tty.c_cflag &= ~CSIZE;
00336     tty.c_cflag |= CS8;           // 8 bits
00337     tty.c_cflag &= ~PARENB;      // sem paridade
00338     tty.c_cflag &= ~CSTOPB;      // 1 stop bit
00339     tty.c_cflag &= ~CRTSCTS;     // sem controle de fluxo por hardware
00340
00341     // Habilita leitura na porta
00342     tty.c_cflag |= (CLOCAL | CREAD);
00343
00344     // Não encerra a comunicação serial quando 'desligamos'
00345     tty.c_cflag &= ~HUPCL;
00346
00347     tty.c_iflag &= ~IGNBRK;
00348     tty.c_iflag &= ~(IXON | IXOFF | IXANY); // sem controle de fluxo por software
00349     tty.c_lflag = 0;                       // sem canonical mode, echo, signals
00350     tty.c_oflag = 0;
00351
00352     tty.c_cc[VMIN] = 1; // lê pelo menos 1 caractere
00353     tty.c_cc[VTIME] = 1; // timeout em décimos de segundo (0.1s)
00354
00355     if (
00356         ::tcsetattr(
00357             fd_serial,
00358             TCSANOW,
00359             &tty
00360         ) != 0
00361     ) { throw std::runtime_error("\033[1;31mErro ao tentar setar configurações na comunicação
    serial, especificamente, tcsetattr\033[0m"); }
00362 }
00363
00373 std::string
00374 read_serial() {
00375     std::string buffer;
00376     char caract = '\0';
00377
00378     while (
00379         true
00380     ) {
00381     }
00382
00383     int n = read(
00384         fd_serial,
00385         &caract,
00386         1
00387     );
00388
00389     // Confirmação de sucesso.
00390     if (n > 0) {
00391
00392         // Então é caractere válido.
00393         if (caract == '\n') { break; }
00394         if (caract != '\r') { buffer += caract; } // Ignoramos o \r
00395
00396     }
00397     else if (n == 0) { break; } // Nada a ser lido
00398     else {
00399
00400         throw std::runtime_error("\033[1;31mErro na leitura\033[0m");
00401     }
00402 }
00403
00404     return buffer;
00405 }
00406
00412 bool
00413 send(
00414     const std::string& mensagem
00415 ) {
00416
00417     ssize_t bytes = ::sendto(
00418         sockfd,
00419         mensagem.c_str(),
00420         mensagem.size(),

```

```

00421             0,
00422             reinterpret_cast<struct sockaddr*>(&addr_dest),
00423             sizeof(addr_dest)
00424         );
00425
00426         if(bytes < 0){ std::cout << "Erro ao enviar" << std::endl; return false;}
00427         return true;
00428     }
00429 }
00430
00431 void
00432 loop(){
00433     bool parsed = false; // Apenas uma flag para sabermos se houve interpretação
00434     while(
00435         is_exec
00436     ){
00437         std::string mensagem = read_serial();
00438
00439         if(mensagem.empty()){ std::cout << "Nada a ser lido..." << std::endl; }
00440         else{
00441             std::cout << "Recebendo: " << mensagem << std::endl;
00442
00443             if( mensagem.find("GGA") != std::string::npos ){
00444                 last_data_given.parsing(0, split(mensagem));
00445                 parsed = true;
00446             }
00447             // ... para escalarmos novos padrões de mensagem
00448             else{
00449                 if(
00450                     parsed
00451                 ){
00452                     std::string mensagem = last_data_given.to_csv();
00453
00454                     std::cout << "Interpretando: \033[7m"
00455                                 << mensagem
00456                                 << "\033[0m"
00457                                 << std::endl;
00458
00459                     send(
00460                         mensagem
00461                     );
00462                     parsed = false;
00463                     printf("\n");
00464                 }
00465             }
00466
00467             std::this_thread::sleep_for(std::chrono::seconds(1));
00468         }
00469     }
00470 }
00471
00472 public:
00473
00474 GPSTrack(
00475     const std::string& ip_destino_,
00476     int porta_destino_,
00477     const std::string& porta_serial_
00478 ) : ip_destino(ip_destino_),
00479     porta_destino(porta_destino_),
00480     porta_serial(porta_serial_)
00481 {
00482     // As seguintes definições existentes para a comunicação UDP.
00483     sockfd = ::socket(AF_INET, SOCK_DGRAM, 0);
00484     if( sockfd < 0 ){
00485         throw std::runtime_error("Erro ao criar socket UDP");
00486     }
00487
00488     addr_dest.sin_family = AF_INET;
00489     addr_dest.sin_port = htons(porta_destino);
00490     addr_dest.sin_addr.s_addr = inet_addr(ip_destino.c_str());
00491
00492     open_serial();
00493 }
00494
00495 ~GPSTrack() { stop(); if( sockfd >= 0 ){ ::close(sockfd); } if( fd_serial >= 0 ){
00496     ::close(fd_serial); } }
00497
00498 }

```



```

00548     void
00549     init(){
00550
00551         if( is_exec.exchange(true) ){ return; }
00552
00553         std::cout << "\033[1;32mIniciando Thread de Leitura...\033[0m" << std::endl;
00554         worker = std::thread(
00555             [this]{ loop(); }
00556         );
00557     }
00558
00559     void
00560     stop(){
00561
00562         if( !is_exec.exchange(false) ){ return; }
00563
00564         if(
00565             worker.joinable()
00566         ){
00567             std::cout << "\033[1;32mSaindo da thread de leitura.\033[0m" << std::endl;
00568             worker.join();
00569         }
00570     }
00571 };
00572
00573 #endif // GPSTRACK_HPP

```

2.6 src/main.cpp File Reference

Responsável por executar a aplicação.

```
#include "GPSTrack.hpp"
```

Include dependency graph for main.cpp:



Functions

- int `main` (int argc, char *argv[])

2.6.1 Detailed Description

Responsável por executar a aplicação.

2.6.2 Function Documentation

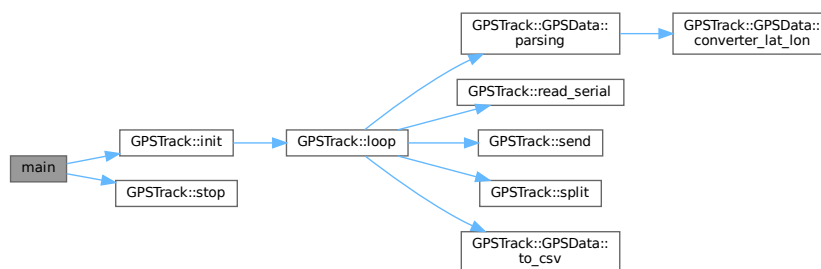
2.6.2.1 main()

```

int main (
    int argc,
    char * argv[] )

```

Here is the call graph for this function:



Index

- ~GPSSim
 - GPSSim, [6](#)
- ~GPSTrack
 - GPSTrack, [15](#)
- addr_dest
 - GPSTrack, [20](#)
- alt
 - GPSSim, [12](#)
- build_nmea_string
 - GPSSim, [7](#)
- caminho_do_pseudo_terminal
 - GPSSim, [12](#)
- converter_lat_lon
 - GPSTrack::GPSData, [2](#)
- data
 - GPSTrack::GPSData, [4](#)
- debug.cpp
 - main, [26](#)
- degrees_to_NMEA
 - GPSSim, [7](#)
- fd_filho
 - GPSSim, [12](#)
- fd_pai
 - GPSSim, [12](#)
- fd_serial
 - GPSTrack, [20](#)
- format_integer
 - GPSSim, [8](#)
- generate_gga
 - GPSSim::NMEAGenerator, [21](#)
- generate_rmc
 - GPSSim::NMEAGenerator, [22](#)
- get_path_pseudo_term
 - GPSSim, [8](#)
- get_utc_time
 - GPSSim, [9](#)
- GPSData
 - GPSTrack::GPSData, [2](#)
- GPSSim, [5](#)
 - ~GPSSim, [6](#)
 - alt, [12](#)
 - build_nmea_string, [7](#)
 - caminho_do_pseudo_terminal, [12](#)
 - degrees_to_NMEA, [7](#)
 - fd_filho, [12](#)
 - fd_pai, [12](#)
 - format_integer, [8](#)
 - generate_gga, [21](#)
 - generate_rmc, [22](#)
 - get_path_pseudo_term, [8](#)
 - get_utc_time, [9](#)
 - GPSSim, [6](#)
 - init, [9](#)
 - is_exec, [12](#)
 - lat, [12](#)
 - lon, [12](#)
 - loop, [10](#)
 - stop, [11](#)
 - worker, [12](#)
- GPSSim::NMEAGenerator, [21](#)
 - generate_gga, [21](#)
 - generate_rmc, [22](#)
- GPSTrack, [13](#)
 - ~GPSTrack, [15](#)
 - addr_dest, [20](#)
 - fd_serial, [20](#)
 - GPSTrack, [14](#)
 - init, [15](#)
 - ip_destino, [20](#)
 - is_exec, [20](#)
 - last_data_given, [20](#)
 - loop, [16](#)
 - open_serial, [17](#)
 - porta_destino, [20](#)
 - porta_serial, [20](#)
 - read_serial, [18](#)
 - send, [18](#)
 - sockfd, [21](#)
 - split, [19](#)
 - stop, [19](#)
 - worker, [21](#)
- GPSTrack::GPSData, [1](#)
 - converter_lat_lon, [2](#)
 - data, [4](#)
 - GPSData, [2](#)
 - parsing, [3](#)
 - pattern, [4](#)
 - to_csv, [4](#)
- init
 - GPSSim, [9](#)
 - GPSTrack, [15](#)
- ip_destino
 - GPSTrack, [20](#)
- is_exec
 - GPSSim, [12](#)
 - GPSTrack, [20](#)

- last_data_given
 - GPSTrack, [20](#)
- lat
 - GPSSim, [12](#)
- lon
 - GPSSim, [12](#)
- loop
 - GPSSim, [10](#)
 - GPSTrack, [16](#)
- main
 - debug.cpp, [26](#)
 - main.cpp, [37](#)
- main.cpp
 - main, [37](#)
- open_serial
 - GPSTrack, [17](#)
- parsing
 - GPSTrack::GPSTData, [3](#)
- pattern
 - GPSTrack::GPSTData, [4](#)
- porta_destino
 - GPSTrack, [20](#)
- porta_serial
 - GPSTrack, [20](#)
- read_serial
 - GPSTrack, [18](#)
- send
 - GPSTrack, [18](#)
- sockfd
 - GPSTrack, [21](#)
- split
 - GPSTrack, [19](#)
- src/debug.cpp, [25](#)
- src/GPSSim.hpp, [26](#), [27](#)
- src/GPSTrack.hpp, [31](#), [32](#)
- src/main.cpp, [37](#)
- stop
 - GPSSim, [11](#)
 - GPSTrack, [19](#)
- to_csv
 - GPSTrack::GPSTData, [4](#)
- worker
 - GPSSim, [12](#)
 - GPSTrack, [21](#)