Sensor de Chama

Gerado por Doxygen 1.14.0

Capítulo 1

Índice dos Componentes

1.1 Lista de Classes

Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições:

Capítulo 2

Índice dos Arquivos

2.1 Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

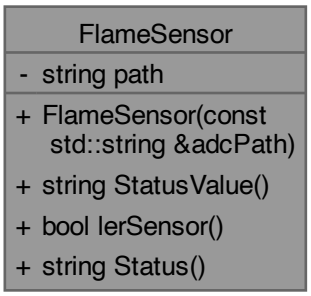
Capítulo 3

Classes

3.1 Referência da Classe FlameSensor

Cria a classe do sensor de chama.

Diagrama de colaboração para FlameSensor:



Membros Públicos

- FlameSensor (const std::string &adcPath)

  Método construtor da classe.
- string StatusValue ()
- bool lerSensor ()

  Método que retorna o valor indicado pelo sensor de chama.
- string Status ()

  Método que conclui se há ou não chama presente, baseado na leitura do sensor.

Atributos Privados

- string path

3.1.1 Descrição detalhada

Cria a classe do sensor de chama.

Definição na linha 25 do arquivo Cliente_flame.cpp.

3.1.2 Construtores e Destrutores

3.1.2.1 FlameSensor()

```
FlameSensor::FlameSensor (
            const std::string & adcPath)  [inline]
```

Método construtor da classe.

Parâmetros

	caminho	para o diretório onde são armazenados os dados do sensor

Definição na linha 35 do arquivo Cliente_flame.cpp.

3.1.3 Documentação das funções

3.1.3.1 lerSensor()

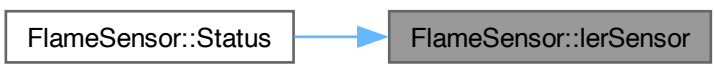
```
bool FlameSensor::lerSensor ()  [inline]
```

Método que retorna o valor indicado pelo sensor de chama.

É estipulado um valor threshold de 22350 para o código tomar a decisão

Definição na linha 56 do arquivo Cliente_flame.cpp.

Esse é o diagrama das funções que utilizam essa função:



3.1.3.2 Status()

```
string FlameSensor::Status ()  [inline]
```

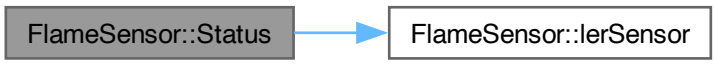
Método que conclui se há ou não chama presente, baseado na leitura do sensor.

Retorna

0 se a leitura for abaixo do thershold e 1 se a leitura for acima do threshold

Definição na linha 71 do arquivo Cliente_flame.cpp.

Este é o diagrama das funções utilizadas por essa função:



3.1.3.3 StatusValue()

```
string FlameSensor::StatusValue ()  [inline]
```

Definição na linha 39 do arquivo Cliente_flame.cpp.

Esse é o diagrama das funções que utilizam essa função:

3.1.4 Atributos

3.1.4.1 path

```
string FlameSensor::path  [private]
```

Definição na linha 27 do arquivo Cliente_flame.cpp.

A documentação para essa classe foi gerada a partir do seguinte arquivo:

- Cliente_flame.cpp

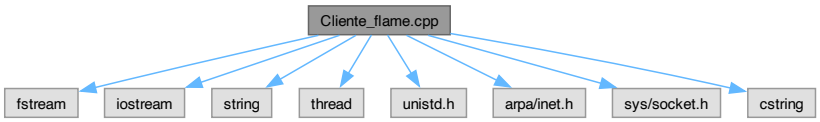
Capítulo 4

Arquivos

4.1 Referência do Arquivo Cliente_flame.cpp

```
#include <fstream>
#include <iostream>
#include <string>
#include <thread>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <cstring>
```

Gráfico de dependência de inclusões para Cliente_flame.cpp:



Componentes

- class FlameSensor

  Cria a classe do sensor de chama.

Definições e Macros

- #define SENSOR_PATH "/sys/bus/iio/devices/iio:device0/in_voltage19_raw"
- #define IP_servidor "192.168.42.10"
- #define Port_servidor 5000

Funções

- int main ()

4.1.1 Definições e macros

4.1.1.1 IP_servidor

```
#define IP_servidor "192.168.42.10"
```

Definição na linha 17 do arquivo Cliente_flame.cpp.

4.1.1.2 Port_servidor

```
#define Port_servidor 5000
```

Definição na linha 18 do arquivo Cliente_flame.cpp.

4.1.1.3 SENSOR_PATH

```
#define SENSOR_PATH "/sys/bus/iio/devices/iio:device0/in_voltage19_raw"
```

Parâmetros

	Path	do arquivo na placa que lê as tensões do sensor IP da interface Ethernet da máquina a ser usada como servidor Porta a

Definição na linha 16 do arquivo Cliente_flame.cpp.

4.1.2 Funções

4.1.2.1 main()

```
int main ()
```

Cria o socket do cliente (placa STM)

Loop de leitura do sensor e envio da leitura para o IP do servidor

Pausa a execução do loop por um intervalo de 1s para não deixar muito rápido

Definição na linha 81 do arquivo Cliente_flame.cpp.

Este é o diagrama das funções utilizadas por essa função:

4.2 Cliente_flame.cpp

Ir para a documentação desse arquivo.

```
00001 #include <fstream>
00002 #include <iostream>
00003 #include <string>
00004 #include <thread>
00005 #include <unistd.h>
00006 #include <arpa/inet.h>
00007 #include <sys/socket.h>
00008 #include <cstring>
00009
00016 #define SENSOR_PATH "/sys/bus/iio/devices/iio:device0/in_voltage19_raw"
00017 #define IP_servidor  "192.168.42.10"
00018 #define Port_servidor 5000
00019
00020 using namespace std;
00021
00025 class FlameSensor{
00026 private:
00027     string path;
00028
00029 public:
00030
00035     FlameSensor(const std::string& adcPath) {
00036         path = adcPath;
00037     }
00038
00039     string StatusValue(){
00040         std::ifstream file(path);
00041         int valor;
00042         file » valor;
00043         std::string S1 = "Valor: ";
00044         std::string S2 = std::to_string(valor);
00045         if(valor < 22350) {
00046             return S1 + S2 + " - Há chama próxima";
00047         } else {
00048             return S1 + S2 + " - Não há chama próxima";
00049         }
00050     }
00051
00056     bool lerSensor() {
00057         std::ifstream file(path);
00058         int valor;
00059         file » valor;
00060         if(valor < 22350) {
00061             return 0;
00062         } else {
00063             return 1;
00064         }
00065     }
00067
00071     string Status(){
00072         if(lerSensor() == 0){
00073             return "Há chama próxima";
00074         } else {
00075             return "Não há chama próxima";
00076         }
00077     }
00079 };
00080
00081 int main() {
00082
00083     FlameSensor Sensor(SENSOR_PATH);
00084
00088     int sock = socket(AF_INET, SOCK_DGRAM, 0);
00089     if (sock < 0) {
00090         perror("Erro ao criar socket UDP");
00091         return 1;
00092     }
00093
00094     sockaddr_in servAddr;
00095     memset(&servAddr, 0, sizeof(servAddr));
00096     servAddr.sin_family = AF_INET;
00097     servAddr.sin_port = htons(Port_servidor);
00098     servAddr.sin_addr.s_addr = inet_addr(IP_servidor);
00099
00100     cout « "Cliente UDP iniciado. Enviando status do sensor para "
00101          « IP_servidor « ":" « Port_servidor « endl;
00102
00103
00107     while (true) {
00108         string status = Sensor.StatusValue();
00109
00110         ssize_t enviado = sendto(sock, status.c_str(), status.size(), 0,
```

```
00111                                   (sockaddr*)&servAddr, sizeof(servAddr));
00112         if (enviado < 0) {
00113             perror("Erro ao enviar pacote UDP");
00114             break;
00115         }
00116
00117         cout « "Enviado: " « status « endl;
00118         sleep(1);
00122     }
00123
00124     close(sock);
00125     return 0;
00126 }
```

Índice Remissivo