CPP

Generated by Doxygen 1.9.5

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Database Class Reference

the Database class

`#include <database.h>`

Public Member Functions

- ptrPhoto createPhoto (string _objName, string _pathName, double _latitude, double _longitude)

  Create a Photo object.
- ptrVideo createVideo (string _objName, string _pathName, int _duree)

  Create a Video object.
- ptrFilm createFilm (string _objName, string _pathName, int _duree, int _nOfChptrs, int *_chptrs)

  Create a Film object.
- ptrGroup createGroup (string _groupName)

  Create a Group object.
- void findANDShow (string _objName, ostream &s)

  Search for an object using its name , if it exists it call the afficher method else a message will be shown.
- int findANDPlay (string _objName)

  search for a certain object by its name, if it exists it will be playen else a message will be shown
- bool processRequest (const string &request, string &response)

  This method takes request and response as arguments, the request should be either find <filename> or play <filename>

4.1.1 Detailed Description

the Database class

4.1.2 Member Function Documentation

4.1.2.1 createFilm()

```
ptrFilm Database::createFilm (
            string _objName,
            string _pathName,
            int _duree,
            int _nOfChptrs,
            int * _chptrs )  [inline]
```

Create a Film object.

Parameters

_objName	
_pathName	
_duree	
_nOfChptrs	
_chptrs	

Returns

ptrFilm

4.1.2.2 createGroup()

```
ptrGroup Database::createGroup (
            string _groupName )  [inline]
```

Create a Group object.

Parameters

_groupName	

Returns

ptrGroup

4.1.2.3 createPhoto()

```
ptrPhoto Database::createPhoto (
            string _objName,
            string _pathName,
            double _latitude,
            double _longitude )  [inline]
```

Create a Photo object.

Parameters

_objName	
_pathName	
_latitude	
_longitude	

Returns

ptrPhoto

4.1.2.4 createVideo()

```
ptrVideo Database::createVideo (
            string _objName,
            string _pathName,
            int _duree )  [inline]
```

Create a Video object.

Parameters

_objName	
_pathName	
_duree	

Returns

ptrVideo

4.1.2.5 findANDPlay()

```
int Database::findANDPlay (
            string _objName )  [inline]
```

search for a certain object by its name, if it exists it will be playen else a message will be shown

Parameters

_objName	

Returns

int

4.1.2.6 findANDShow()

```
void Database::findANDShow (
            string _objName,
            ostream & s )  [inline]
```

Search for an object using its name , if it exists it call the afficher method else a message will be shown.

Parameters

_objName	
s	

4.1.2.7 processRequest()

```
bool Database::processRequest (
            const string & request,
            string & response )  [inline]
```

This method takes request and response as arguments, the request should be either find <filename> or play <filename>

Parameters

request	
response	

Returns

true

false

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

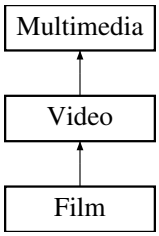
- database.h

4.2 Film Class Reference

a Film class which herit from the Video class

`#include <film.h>`

Inheritance diagram for Film:



Public Member Functions

- **Film** (string _objName, string _pathName, int _duree, int _nbrOfChapters=0, int *_chapters=nullptr)
- void setChapters (int *_chapters=nullptr, int _nbrOfChapters=0)

  Set the Chapters object destroying the old chapters before seting the new ones.
- int **getNbrOfChapters** () const
- const int * **getChapters** () const
- void afficher (ostream &s) const override
- Film (const Film &from)

  Construct a new Film object using another object.
- Film & operator= (const Film &from)

  The = operator for the film class as it contains pointers.
- virtual ~Film ()

  Destroy the Film object and the chapters.

Additional Inherited Members

4.2.1 Detailed Description

a Film class which herit from the Video class

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Film()

```
Film::Film (
            const Film & from )  [inline]
```

Construct a new Film object using another object.

Parameters

from	

4.2.2.2 ~Film()

```
virtual Film::~Film ( )  [inline], [virtual]
```

Destroy the Film object and the chapters.

4.2.3 Member Function Documentation

4.2.3.1 afficher()

```
void Film::afficher (
            ostream & s ) const  [inline], [override], [virtual]
```

Reimplemented from Multimedia.

4.2.3.2 operator=()

```
Film & Film::operator= (
            const Film & from )  [inline]
```

The = operator for the film class as it contains pointers.

Parameters

from	

Returns

Film&

4.2.3.3 setChapters()

```
void Film::setChapters (
            int * _chapters = nullptr,
            int _nbrOfChapters = 0 )  [inline]
```

Set the Chapters object destroying the old chapters before seting the new ones.

Parameters

_chapters	
_nbrOfChapters	

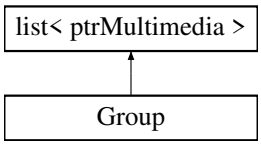
The documentation for this class was generated from the following file:

- film.h

4.3 Group Class Reference

`#include <group.h>`

Inheritance diagram for Group:



Public Member Functions

- **Group** (string _groupName)
- string **getGroupName** () const
- void **setGroupName** (string _name)
- void **show** (ostream &s) const

4.3.1 Detailed Description

a Group class that herit from a list of shared_ptr<Multimedia> each object of this class contain a specific multimedias

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

- group.h

4.4 InputBuffer Struct Reference

Public Member Functions

- **InputBuffer** (size_t size)

Public Attributes

- char * **buffer**
- char * **begin**
- char * **end**
- SOCKSIZE **remaining**

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

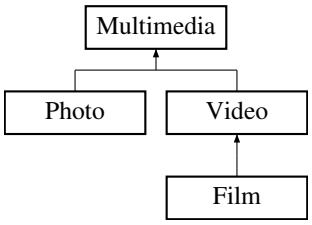
- ccsocket.cpp

4.5 Multimedia Class Reference

an abstract class which will be herited by Photo and Video classes

`#include <multimedia.h>`

Inheritance diagram for Multimedia:



Public Member Functions

- **Multimedia** (string _objName, string _pathName)
- string **getObjName** () const
- string **getPathName** () const
- void **setObjName** (string _objName)
- void **setPathName** (string _pathName)
- virtual void afficher (ostream &s) const
- virtual void lire ()=0

Protected Attributes

- string **objName** {}
- string **pathName** {}

4.5.1 Detailed Description

an abstract class which will be herited by Photo and Video classes

4.5.2 Member Function Documentation

4.5.2.1 afficher()

```
void Multimedia::afficher (
            ostream & s ) const  [virtual]
```

Reimplemented in Photo.

4.5.2.2 lire()

```
virtual void Multimedia::lire ( )  [pure virtual]
```

Implemented in Photo, and Video.

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

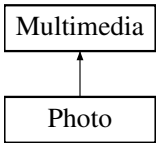
- multimedia.h
- multimedia.cpp

4.6 Photo Class Reference

a class that herit from Multimedia class

```
#include <photo.h>
```

Inheritance diagram for Photo:



Public Member Functions

- **Photo** (string _objName, string _pathName, double _latitude, double _longitude)
- void **setLatitude** (double _latitude)
- void **setLongitude** (int _longitude)
- double **getLatitude** () const
- double **getLongitude** () const
- void afficher (ostream &s) const override

  a method to output the attributes of the photo instance
- void lire ()

  a method to play the Image

Additional Inherited Members

4.6.1 Detailed Description

a class that herit from Multimedia class

4.6.2 Member Function Documentation

4.6.2.1 afficher()

```
void Photo::afficher (
            ostream & s ) const  [inline], [override], [virtual]
```

a method to output the attributes of the photo instance

Parameters

s	

Reimplemented from Multimedia.

4.6.2.2 lire()

```
void Photo::lire ( )  [inline], [virtual]
```

a method to play the Image

Implements Multimedia.

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

- photo.h

4.7 ServerSocket Class Reference

```
#include <ccsocket.h>
```

Public Member Functions

- **ServerSocket** ()

  Creates a listening socket that waits for connection requests by TCP/IP clients.
- Socket * accept ()
- int bind (int port, int backlog=50)
- int **close** ()

  Closes the socket.
- bool **isClosed** () const

  Returns true if the socket was closed.
- SOCKET **descriptor** ()

  Returns the descriptor of the socket.
- int **setReceiveBufferSize** (int size)

  Sets the SO_RCVBUF option to the specified value.
- int **setReuseAddress** (bool)

  Enables/disables the SO_REUSEADDR socket option.
- int **setSoTimeout** (int timeout)

  Enables/disables SO_TIMEOUT with the specified timeout (in milliseconds).
- int **setTcpNoDelay** (bool)

  Turns on/off TCP coalescence (useful in some cases to avoid delays).

4.7.1 Detailed Description

TCP/IP IPv4 server socket. Waits for requests to come in over the network. TCP/IP sockets do not preserve record boundaries but SocketBuffer solves this problem.

4.7.2 Member Function Documentation

4.7.2.1 accept()

```
Socket * ServerSocket::accept ( )
```

Accepts a new connection request and returns a socket for exchanging data with this client. This function blocks until there is a connection request.

Returns

the new Socket or nullptr on error.

4.7.2.2 bind()

```
int ServerSocket::bind (
            int port,
            int backlog = 50 )
```

Assigns the server socket to localhost.

Returns

0 on success or a negative value on error, see Socket::Errors

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

- ccsocket.h
- ccsocket.cpp

4.8 Socket Class Reference

```
#include <ccsocket.h>
```

Public Types

- enum Errors { **Failed** = -1 , **InvalidSocket** = -2 , **UnknownHost** = -3 }

Public Member Functions

- Socket (int type=SOCK_STREAM)
- **Socket** (int type, SOCKET sockfd)

  Creates a Socket from an existing socket file descriptor.
- ~**Socket** ()

  Destructor (closes the socket).
- int connect (const std::string &host, int port)
- int bind (int port)
- int bind (const std::string &host, int port)
- int **close** ()

  Closes the socket.
- bool **isClosed** () const

  Returns true if the socket has been closed.
- SOCKET **descriptor** ()

  Returns the descriptor of the socket.
- void **shutdownInput** ()

  Disables further receive operations.
- void **shutdownOutput** ()

  Disables further send operations.
- SOCKSIZE send (const SOCKDATA *buf, size_t len, int flags=0)
- SOCKSIZE receive (SOCKDATA *buf, size_t len, int flags=0)
- SOCKSIZE **sendTo** (void const *buf, size_t len, int flags, SOCKADDR const *to, socklen_t addrlen)

  Sends data to a datagram socket.
- SOCKSIZE **receiveFrom** (void *buf, size_t len, int flags, SOCKADDR *from, socklen_t *addrlen)

  Receives data from datagram socket.
- int **setReceiveBufferSize** (int size)

  Set the size of the TCP/IP input buffer.
- int **setReuseAddress** (bool)

  Enable/disable the SO_REUSEADDR socket option.
- int **setSendBufferSize** (int size)

  Set the size of the TCP/IP output buffer.
- int **setSoLinger** (bool, int linger)

  Enable/disable SO_LINGER with the specified linger time in seconds.
- int **setSoTimeout** (int timeout)

  Enable/disable SO_TIMEOUT with the specified timeout (in milliseconds).
- int **setTcpNoDelay** (bool)

  Enable/disable TCP_NODELAY (turns on/off TCP coalescence).
- int **getReceiveBufferSize** () const

  Return the size of the TCP/IP input buffer.
- bool **getReuseAddress** () const

  Return SO_REUSEADDR state.
- int **getSendBufferSize** () const

  Return the size of the TCP/IP output buffer.
- bool **getSoLinger** (int &linger) const

  Return SO_LINGER state and the specified linger time in seconds.
- int **getSoTimeout** () const

  Return SO_TIMEOUT value.
- bool **getTcpNoDelay** () const

  Return TCP_NODELAY state.

Static Public Member Functions

- static void startup ()
- static void **cleanup** ()

Friends

- class **ServerSocket**

4.8.1 Detailed Description

TCP/IP or UDP/Datagram IPv4 socket. AF_INET connections following the IPv4 Internet protocol are supported.

Note

- ServerSocket should be used on the server side.
- SIGPIPE signals are ignored when using Linux, BSD or MACOSX.
- TCP/IP sockets do not preserve record boundaries but SocketBuffer solves this problem.

4.8.2 Member Enumeration Documentation

4.8.2.1 Errors

`enum Socket::Errors`

Socket errors.

- Socket::Failed (-1): could not connect, could not bind, etc.
- Socket::InvalidSocket (-2): invalid socket or wrong socket type
- Socket::UnknownHost (-3): could not reach host

4.8.3 Constructor & Destructor Documentation

4.8.3.1 Socket()

```
Socket::Socket (
            int type = SOCK_STREAM )
```

Creates a new Socket. Creates a AF_INET socket using the IPv4 Internet protocol. Type can be:

- SOCK_STREAM (the default) for TCP/IP connected stream sockets
- SOCK_DGRAM for UDP/datagram sockets (available only or Unix/Linux)

4.8.4 Member Function Documentation

4.8.4.1 bind() [1/2]

```
int Socket::bind (
            const std::string & host,
            int port )
```

Assigns the socket to an IP address. On Unix/Linux host can be a hostname, on Windows it can only be an IP address.

Returns

0 on success or a negative value on error, see Socket::Errors

4.8.4.2 bind() [2/2]

```
int Socket::bind (
            int port )
```

Assigns the socket to localhost.

Returns

0 on success or a negative value on error, see Socket::Errors

4.8.4.3 connect()

```
int Socket::connect (
            const std::string & host,
            int port )
```

Connects the socket to an address. Typically used for connecting TCP/IP clients to a ServerSocket. On Unix/Linux host can be a hostname, on Windows it can only be an IP address.

Returns

0 on success or a negative value on error which is one of Socket::Errors

4.8.4.4 receive()

```
SOCKSIZE Socket::receive (
            SOCKDATA * buf,
            size_t len,
            int flags = 0 )  [inline]
```

Receives data from a connected (TCP/IP) socket. Reads at most *len* bytes fand stores them in *buf*. By default, this function blocks the caller until thre is availbale data.

Returns

the number of bytes that were received, or 0 or shutdownOutput() was called on the other side, or Socket::↩ Failed (-1) if an error occured.

4.8.4.5 send()

```
SOCKSIZE Socket::send (
            const SOCKDATA * buf,
            size_t len,
            int flags = 0 )  [inline]
```

Send sdata to a connected (TCP/IP) socket. Sends the first *len* bytes in *buf*.

Returns

the number of bytes that were sent, or 0 or shutdownInput() was called on the other side, or Socket::Failed (-1) if an error occured.

Note

TCP/IP sockets do not preserve record boundaries, see SocketBuffer.

4.8.4.6 startup()

```
void Socket::startup ( )  [static]
```

initialisation and cleanup of sockets on Widows.

Note

startup is automaticcaly called when a Socket or a ServerSocket is created

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

- ccsocket.h
- ccsocket.cpp

4.9 SocketBuffer Class Reference

```
#include <ccsocket.h>
```

Public Member Functions

- SOCKSIZE readLine (std::string &message)
- SOCKSIZE writeLine (const std::string &message)
- SOCKSIZE read (char *buffer, size_t len)
- SOCKSIZE write (const char *str, size_t len)
- Socket * **socket** ()

  Returns the associated socket.

- SocketBuffer (Socket *, size_t inputSize=8192, size_t ouputSize=8192)
- **SocketBuffer** (Socket &, size_t inputSize=8192, size_t ouputSize=8192)

- size_t **insize_** {}
- size_t **outsize_** {}
- int **insep_** {}
- int **outsep_** {}
- Socket * **sock_** {}
- struct InputBuffer * **in_** {}
- void setReadSeparator (int separ)
- int **readSeparator** () const
- void setWriteSeparator (int separ)
- int **writeSeparator** () const
- bool **retrieveLine** (std::string &str, SOCKSIZE received)

4.9.1 Detailed Description

Preserves record boundaries when exchanging messages between connected TCP/IP sockets. Ensures that one call to readLine() corresponds to one and exactly one call to writeLine() on the other side. By default, writeLine() adds
at the end of each message and readLine() searches for
, \r or
\r so that it can retreive the entire record. Beware messages should thus not contain these charecters.

```
int main() {
   Socket sock;
   SocketBuffer sockbuf(sock);
   int status = sock.connect("localhost", 3331);
   if (status < 0) {
     cerr « "Could not connect" « endl;
     return 1;
   }
   while (cin) {
     string request, response;
     cout « "Request: ";
     getline(cin, request);
     if (sockbuf.writeLine(request) < 0) {
        cerr « "Could not send message" « endl;
        return 2;
     }
     if (sockbuf.readLine(response) < 0) {
        cerr « "Couldn't receive message" « endl;
        return 3;
     }
   }
  return 0;
}
```

4.9.2 Constructor & Destructor Documentation

4.9.2.1 SocketBuffer()

```
SocketBuffer::SocketBuffer (
             Socket * sock,
             size_t inputSize = 8192,
             size_t ouputSize = 8192 )
```

Constructor. *socket* must be a connected TCP/IP Socket. It should **not** be deleted as long as the SocketBuffer is used. *inputSize* and *ouputSize* are the sizes of the buffers that are used internally for exchanging data.

4.9.3 Member Function Documentation

4.9.3.1 read()

```
SOCKSIZE SocketBuffer::read (
             char * buffer,
             size_t len )
```

Reads exactly *len* bytes from the socket, blocks otherwise.

Returns

see readLine()

4.9.3.2 readLine()

```
SOCKSIZE SocketBuffer::readLine (
            std::string & message )
```

Read a message from a connected socket. readLine() receives one (and only one) message sent by writeLine() on the other side, ie, a call to writeLine() corresponds to one and exactly one call to readLine() on the other side. The received data is stored in *message*. This method blocks until the message is fully received.

Returns

The number of bytes that were received or one of the following values:

- 0: shutdownOutput() was called on the other side
- Socket::Failed (-1): a connection error occured
- Socket::InvalidSocket (-2): the socket is invalid.

Note

the separator (eg
) is counted in the value returned by readLine().

4.9.3.3 setReadSeparator()

```
void SocketBuffer::setReadSeparator (
            int separ )
```

Returns/changes the separator used by readLine(). setReadSeparator() changes the symbol used by readLine() to separate successive messages:

- if *separ* $<$ 0 (the default) readLine() searches for \n, \r or \n\r.
- if *separ* $>=$ 0, readLine() searches for this character to separate messages,

4.9.3.4 setWriteSeparator()

```
void SocketBuffer::setWriteSeparator (
            int separ )
```

Returns/changes the separator used by writeLine(). setWriteSeparator() changes the character(s) used by writeLine() to separate successive messages:

- if *separ* $<$ 0 (the default) writeLine() inserts \n\r between successive lines.
- if *separ* $>=$ 0, writeLine() inserts *separ* between successive lines,

4.9.3.5 write()

```
SOCKSIZE SocketBuffer::write (
            const char * str,
            size_t len )
```

Writes *len* bytes to the socket.

Returns

see readLine()

4.9.3.6 writeLine()

```
SOCKSIZE SocketBuffer::writeLine (
            const std::string & message )
```

Send a message to a connected socket. writeLine() sends a message that will be received by a single call of readLine() on the other side,

Returns

see readLine()

Note

if *message* contains one or several occurences of the separator, readLine() will be called as many times on the other side.

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

- ccsocket.h
- ccsocket.cpp

4.10 SocketCnx Class Reference

Connection with a given client. Each SocketCnx uses a different thread.

Public Member Functions

- **SocketCnx** (TCPServer &, Socket *)
- void **processRequests** ()

Public Attributes

- TCPServer & **server_**
- Socket * **sock_**
- SocketBuffer * **sockbuf_**
- std::thread **thread_**

4.10.1 Detailed Description

Connection with a given client. Each SocketCnx uses a different thread.

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

- tcpserver.cpp

4.11 TCPServer Class Reference

`#include <tcpserver.h>`

Public Types

- using **Callback** = std::function< bool(std::string const &request, std::string &response) >

Public Member Functions

- TCPServer (Callback const &callback)
- virtual int run (int port)

Friends

- class **TCPLock**
- class **SocketCnx**

4.11.1 Detailed Description

TCP/IP IPv4 server. Supports TCP/IP AF_INET IPv4 connections with multiple clients. One thread is used per client.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 TCPServer()

```
TCPServer::TCPServer (
            Callback const & callback )
```

initializes the server. The callback function will be called each time the server receives a request from a client.

- *request* contains the data sent by the client
- *response* will be sent to the client as a response The connection with the client is closed if the callback returns false.

4.11.3 Member Function Documentation

4.11.3.1 run()

```
int TCPServer::run (
            int port )  [virtual]
```

Starts the server. Binds an internal ServerSocket to *port* then starts an infinite loop that processes connection requests from clients.

Returns

0 on normal termination, or a negative value if the ServerSocket could not be bound (value is then one of Socket::Errors).

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

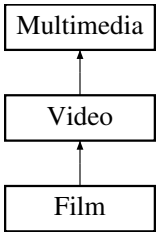
- tcpserver.h
- tcpserver.cpp

4.12 Video Class Reference

a Video class that herit from Multimedia class and will be herited by the Film class

```
#include <video.h>
```

Inheritance diagram for Video:



Public Member Functions

- **Video** (string _objName, string _pathName, int _duree)
- void **setDuree** (int _duree)
- int **getDuree** () const
- void afficher (ostream &s) const override
- void lire ()

  a method to play the video

Protected Attributes

- int **duree** {}

4.12.1 Detailed Description

a Video class that herit from Multimedia class and will be herited by the Film class

4.12.2 Member Function Documentation

4.12.2.1 afficher()

```
void Video::afficher (
            ostream & s ) const  [inline], [override], [virtual]
```

Reimplemented from Multimedia.

4.12.2.2 lire()

```
void Video::lire ( )  [inline], [virtual]
```

a method to play the video

Implements Multimedia.

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

- video.h

Chapter 5

File Documentation

5.1 ccsocket.h

```
1 //
2 //  ccsocket: C++ Classes for TCP/IP and UDP Datagram INET Sockets.
3 //  (c) Eric Lecolinet 2016/2020 - https://www.telecom-paris.fr/~elc
4 //
5 //  - Socket: TCP/IP or UDP/Datagram IPv4 socket
6 //  - ServerSocket: TCP/IP Socket Server
7 //  - SocketBuffer: preserves record boundaries when exchanging data
8 //   between TCP/IP sockets.
9 //
10
11 #ifndef ccuty_ccsocket
12 #define ccuty_ccsocket 1
13
14 #include <string>
15
16 #if defined(_WIN32) || defined(_WIN64)
17 #include <winsock2.h>
18 #define SOCKSIZE int
19 #define SOCKDATA char
20
21 #else
22 #include <sys/types.h>
23 #include <sys/socket.h>
24 #define SOCKET int
25 #define SOCKADDR struct sockaddr
26 #define SOCKADDR_IN struct sockaddr_in
27 #define INVALID_SOCKET -1
28 #define SOCKSIZE ssize_t
29 #define SOCKDATA void
30 #endif
31
32 // ignore SIGPIPES when possible
33 #if defined(MSG_NOSIGNAL)
34 #  define NO_SIGPIPE_(flags) (flags | MSG_NOSIGNAL)
35 #else
36 #  define NO_SIGPIPE_(flags) (flags)
37 #endif
38
46 class Socket {
47 public:
52   enum Errors { Failed = -1, InvalidSocket = -2, UnknownHost = -3 };
53
57   static void startup();
58   static void cleanup();
60
65   Socket(int type = SOCK_STREAM);
66
68   Socket(int type, SOCKET sockfd);
69
71   ~Socket();
72
77   int connect(const std::string& host, int port);
78
81   int bind(int port);
82
86   int bind(const std::string& host, int port);
87
89   int close();
```

```
90
92   bool isClosed() const { return sockfd_ == INVALID_SOCKET; }
93
95   SOCKET descriptor() { return sockfd_; }
96
98   void shutdownInput();
99
101   void shutdownOutput();
102
108   SOCKSIZE send(const SOCKDATA* buf, size_t len, int flags = 0) {
109     return ::send(sockfd_, buf, len, NO_SIGPIPE_(flags));
110   }
111
117   SOCKSIZE receive(SOCKDATA* buf, size_t len, int flags = 0) {
118     return ::recv(sockfd_, buf, len, flags);
119   }
120
121 #if !defined(_WIN32) && !defined(_WIN64)
122
124   SOCKSIZE sendTo(void const* buf, size_t len, int flags,
125                   SOCKADDR const* to, socklen_t addrlen) {
126     return ::sendto(sockfd_, buf, len, NO_SIGPIPE_(flags), to, addrlen);
127   }
128
130   SOCKSIZE receiveFrom(void* buf, size_t len, int flags,
131                        SOCKADDR* from, socklen_t* addrlen) {
132     return ::recvfrom(sockfd_, buf, len, flags, from, addrlen);
133   }
134
136   int setReceiveBufferSize(int size);
137
139   int setReuseAddress(bool);
140
142   int setSendBufferSize(int size);
143
145   int setSoLinger(bool, int linger);
146
148   int setSoTimeout(int timeout);
149
151   int setTcpNoDelay(bool);
152
154   int getReceiveBufferSize() const;
155
157   bool getReuseAddress() const;
158
160   int getSendBufferSize() const;
161
163   bool getSoLinger(int& linger) const;
164
166   int getSoTimeout() const;
167
169   bool getTcpNoDelay() const;
170
171 #endif
172
173 private:
174   friend class ServerSocket;
175
176   // Initializes a local INET4 address, returns 0 on success, -1 otherwise.
177   int setLocalAddress(SOCKADDR_IN& addr, int port);
178   // Initializes a remote INET4 address, returns 0 on success, -1 otherwise.
179   int setAddress(SOCKADDR_IN& addr, const std::string& host, int port);
180
181   SOCKET sockfd_{};
182   Socket(const Socket&) = delete;
183   Socket& operator=(const Socket&) = delete;
184   Socket& operator=(Socket&&) = delete;
185 };
186
187
188
192 class ServerSocket {
193 public:
195   ServerSocket();
196
197   ~ServerSocket();
198
202   Socket* accept();
203
206   int bind(int port, int backlog = 50);
207
209   int close();
210
212   bool isClosed() const { return sockfd_ == INVALID_SOCKET; }
213
215   SOCKET descriptor() { return sockfd_; }
216
```

```
217 #if !defined(_WIN32) && !defined(_WIN64)
218
220   int setReceiveBufferSize(int size);
221
223   int setReuseAddress(bool);
224
226   int  setSoTimeout(int timeout);
227
229   int setTcpNoDelay(bool);
230
231 #endif
232
233 private:
234   Socket* createSocket(SOCKET);
235   SOCKET sockfd_{};  // listening socket.
236   ServerSocket(const ServerSocket&) = delete;
237   ServerSocket& operator=(const ServerSocket&) = delete;
238   ServerSocket& operator=(ServerSocket&&) = delete;
239 };
240
241
276 class SocketBuffer {
277 public:
283   SocketBuffer(Socket*, size_t inputSize = 8192, size_t ouputSize = 8192);
284   SocketBuffer(Socket&, size_t inputSize = 8192, size_t ouputSize = 8192);
286
287   ~SocketBuffer();
288
300   SOCKSIZE readLine(std::string& message);
301
309   SOCKSIZE writeLine(const std::string& message);
310
313   SOCKSIZE read(char* buffer, size_t len);
314
317   SOCKSIZE write(const char* str, size_t len);
318
320   Socket* socket() { return sock_; }
321
327   void setReadSeparator(int separ);
328   int readSeparator() const { return insep_; }
329   // @}
330
336   void setWriteSeparator(int separ);
337   int writeSeparator() const { return outsep_; }
338   // @}
339
340 private:
341   SocketBuffer(const SocketBuffer&) = delete;
342   SocketBuffer& operator=(const SocketBuffer&) = delete;
343   SocketBuffer& operator=(SocketBuffer&&) = delete;
344
345 protected:
346   bool retrieveLine(std::string& str, SOCKSIZE received);
347   size_t insize_{}, outsize_{};
348   int insep_{}, outsep_{};
349   Socket* sock_{};
350   struct InputBuffer* in_{};
351 };
352
353 #endif
```

5.2 database.h File Reference

```
#include <bits/stdc++.h>
#include "multimedia.h"
#include "photo.h"
#include "video.h"
#include "film.h"
#include "group.h"
#include "tcpserver.h"
```

Classes

- class Database

  the Database class

Typedefs

- using **ptrPhoto** = shared_ptr< Photo >
- using **ptrVideo** = shared_ptr< Video >
- using **ptrFilm** = shared_ptr< Film >
- using **ptrGroup** = shared_ptr< Group >

5.2.1 Detailed Description

Author

Wissem BEN BETTAIEB

Version

0.1

Date

2022-11-27

Copyright

Copyright (c) 2022

5.3 database.h

Go to the documentation of this file.

```
1
11 #ifndef DATABASE_H
12 #define DATABASE_H
13 #include <bits/stdc++.h>
14 #include "multimedia.h"
15 #include "photo.h"
16 #include "video.h"
17 #include "film.h"
18 #include "group.h"
19 #include "tcpserver.h"
20
21
22 using ptrPhoto = shared_ptr<Photo>;
23 using ptrVideo = shared_ptr<Video>;
24 using ptrFilm = shared_ptr<Film>;
25 using ptrGroup = shared_ptr<Group>;
26
27
32 class Database{
33     private:
34         map<string,ptrMultimedia> multimediaTable;
35         map<string,ptrGroup> groupTable;
36     public:
37
38     Database(){}
48     ptrPhoto createPhoto(string _objName,string _pathName,double _latitude,double _longitude){
49         ptrPhoto _Ph(new Photo(_objName,_pathName,_latitude,_longitude));
50         multimediaTable[_objName]=_Ph;
51         return _Ph;
52     }
61     ptrVideo createVideo(string _objName,string _pathName,int _duree){
62         ptrVideo _Vi(new Video(_objName,_pathName,_duree));
63         multimediaTable[_objName]=_Vi;
64         return _Vi;
65     }
76     ptrFilm createFilm(string _objName,string _pathName,int _duree,int _nOfChptrs,int * _chptrs){
```

```
77          ptrFilm _Fi(new Film(_objName,_pathName,_duree,_nOfChptrs,_chptrs));
78          multimediaTable[_objName]=_Fi;
79          return _Fi;
80      }
87      ptrGroup createGroup(string _groupName){
88          ptrGroup _Gr(new Group(_groupName));
89          groupTable[_groupName]=_Gr;
90          return _Gr;
91      }
100      void findANDShow(string _objName,ostream & s){
101          auto elementMultimedia = multimediaTable.find(_objName);
102          auto elementGroup = groupTable.find(_objName);
103
104          if(elementMultimedia==multimediaTable.end() && elementGroup==groupTable.end()){
105              s « "There's no element named " « _objName « " in the database." « endl;
106          }
107          else if(elementMultimedia!=multimediaTable.end()){
108              s « "Multimedia object " « _objName « " exist." « endl;;
109              elementMultimedia->second->afficher(s);
110          }
111          else if(elementGroup!=groupTable.end())
112          {
113              s « "Group " « _objName « " exist." « endl;;
114              elementGroup->second->show(s);
115          }
116      }
125      int findANDPlay(string _objName){
126          auto elementMultimedia = multimediaTable.find(_objName);
127          if(elementMultimedia!=multimediaTable.end()){
128              elementMultimedia->second->lire();
129              return 1;
130          }
131          else{
132              cout «"Multimedia Object named " « _objName « " doesn't exist." «endl;
133              return 0;
134          }
135      }
136
137
149      bool processRequest(const string & request, string & response){
150          stringstream ss(request);
151          string operation , name, buffer;
152          ss » operation » name;
153          while(ss » buffer){
154              name = name + " " + buffer;
155          }
156
157          if(operation=="find")
158          {
159              stringstream s;
160              findANDShow(name,s);
161              response=s.str();
162              replace(response.begin(),response.end(),'\n',';') ;
163          }
164          else if (operation=="play"){
165              if(findANDPlay(name))
166              {
167                  response="File "+name+" is currently playing on the server";
168              }
169              else{
170                  response="File "+name+" Was NOT FOUND";
171              }
172          }
173          else
174          {
175              response="Undefined command, use find <filename> or play <filename>";
176          }
177      return true;
178      }
179
180  };
181
182  #endif
```

5.4 film.h File Reference

```
#include "video.h"
```

Classes

- class Film

  a Film class which herit from the Video class

5.4.1 Detailed Description

Author

Wissem BEN BETTAIEB

Version

0.1

Date

2022-11-27

Copyright

Copyright (c) 2022

5.5 film.h

Go to the documentation of this file.

```
1
11 #ifndef FILM_H
12 #define FILM_H
13 #include "video.h"
14
19 class Film: public Video{
20     private:
21     int* chapters = nullptr;
22     int nbrOfChapters = 0;
23     public:
24     Film(){};
25     Film(string _objName,string _pathName,int _duree,int _nbrOfChapters=0,int
       *_chapters=nullptr):Video(_objName,_pathName,_duree)
26     {
27         if(_nbrOfChapters && _chapters){
28         nbrOfChapters=_nbrOfChapters;
29         chapters= new int[nbrOfChapters];
30         for (int i=0;i<nbrOfChapters;i++)
31             chapters[i]=_chapters[i];
32         }
33     }
40     void setChapters(int *_chapters=nullptr,int _nbrOfChapters=0)
41     {
42         if (_chapters && _nbrOfChapters){
43             if(chapters) delete[] chapters;
44         }
45         nbrOfChapters = _nbrOfChapters;
46         chapters = new int[nbrOfChapters];
47         for (int i=0;i<nbrOfChapters;i++)
48         {
49             chapters[i]=_chapters[i];
50         }
51     }
52
53     int getNbrOfChapters() const { return nbrOfChapters;}
54
55     const int* getChapters() const { return chapters;}
56
57     void afficher(ostream & s) const override{
```

```
58          s « "Film name " « objName « endl;
59          s « "Film path " « pathName « endl;
60          s « "Duration " « duree « endl;
61          if (nbrOfChapters && chapters){
62              s « "nbrOf Chapters is : " « nbrOfChapters « endl;
63              for (int i=0;i<nbrOfChapters;i++){
64                  s « chapters[i] « " | ";
65
66              }
67              s « endl;
68              }
69              else{
70                  s « "There's no chapters" « endl;
71              }
72          }
78      Film(const Film & from):Video(from.objName,from.pathName,from.duree){
79          int _nbrOfChapters=from.getNbrOfChapters();
80          const int *_chapters=from.getChapters();
81          if(_nbrOfChapters && _chapters){
82              nbrOfChapters = _nbrOfChapters;
83              chapters=new int[nbrOfChapters];
84              for(int i=0;i<nbrOfChapters;i++)
85                  chapters[i]=_chapters[i];
86          }
87
88      }
95      Film & operator= (const Film & from){
96          Video::operator=(from);
97          int _nbrOfChapters=from.getNbrOfChapters();
98          const int * _chapters=from.getChapters();
99          if(_nbrOfChapters && _chapters){
100             if(chapters){
101                 delete[] chapters;
102             }
103             nbrOfChapters = _nbrOfChapters;
104             chapters = new int[nbrOfChapters];
105             for(int i=0;i<nbrOfChapters;i++)
106                 chapters[i]=_chapters[i];
107         }
108         return *this;
109     }
114     virtual ~Film(){
115         if(chapters) delete[] chapters;
116         cout «" a film named " « objName « " was deleted!" « endl;
117     }
118
119 };
120 #endif
```

5.6 group.h File Reference

```
#include <list>
#include <memory>
#include "multimedia.h"
```

Classes

- class Group

Typedefs

- using **ptrMultimedia** = shared_ptr< Multimedia >

5.6.1 Detailed Description

Author

Wissem BEN BETTAIEB

Version

0.1

Date

2022-11-27

Copyright

5.7 group.h

Go to the documentation of this file.

```
1
11 #ifndef GROUP_H
12 #define GROUP_H
13 #include <list>
14 #include <memory>
15 #include "multimedia.h"
16
17 using ptrMultimedia = shared_ptr<Multimedia>;
18
22 class Group: public list<ptrMultimedia>{
23     private:
24         string groupName{};
25     public:
26         Group();
27         Group(string _groupName):list<ptrMultimedia>(),groupName(_groupName){};
28         string getGroupName() const {return groupName;}
29         void setGroupName(string _name) {groupName=_name;}
30         void show(ostream &s) const{
31             cout « "For group : " « groupName « " : " « endl;
32             for (auto item : *this){
33                 item->afficher(s);
34                 cout « "*****************************************************" « endl;;
35             }
36
37         }
38
39
40
41 };
42 #endif
```

5.8 main.cpp File Reference

this is the main file where several steps are tested

```
#include "multimedia.h"
#include "photo.h"
#include "video.h"
#include "film.h"
#include "group.h"
#include "database.h"
#include "server.h"
```

Macros

- #define **ETAPE2** 0
- #define **ETAPE4** 0
- #define **ETAPE5** 0
- #define **ETAPE6** 0
- #define **ETAPE7** 0
- #define **ETAPE8_9** 0
- #define **ETAPE10** 0
- #define **ETAPE11** 1

Functions

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

5.8.1 Detailed Description

this is the main file where several steps are tested

Author

wissem BEN BETTAIEB

Version

0.1

Date

2022-11-27

Copyright

Copyright (c) 2022

5.9 multimedia.cpp File Reference

```
#include "multimedia.h"
```

5.9.1 Detailed Description

Author

Wissem BEN BETTAIEB

Version

0.1

Date

2022-11-27

Copyright

Copyright (c) 2022

5.10 multimedia.h File Reference

`#include <bits/stdc++.h>`

Classes

- class Multimedia

  an abstract class which will be herited by Photo and Video classes

5.10.1 Detailed Description

Author

Wissem BEN BETTAIEB

Version

0.1

Date

2022-11-27

Copyright

Copyright (c) 2022

5.11 multimedia.h

Go to the documentation of this file.

```
1
11 #ifndef MULTIMEDIA_H
12 #define MULTIMEDIA_H
13 #include <bits/stdc++.h>
14
15
16 using namespace std;
21 class Multimedia{
22     protected:
23         string objName{};
24         string pathName{};
25     public:
26         Multimedia();
27         Multimedia(string _objName,string _pathName);
28
29         string getObjName() const;
30         string getPathName() const;
31
32         void setObjName(string _objName);
33         void setPathName(string _pathName);
34
35         virtual void afficher(ostream & s) const;
36         virtual void lire() = 0;
37         virtual ~Multimedia();
38 };
39 #endif
```

5.12 photo.h File Reference

`#include "multimedia.h"`

Classes

- class Photo

  a class that herit from Multimedia class

5.12.1 Detailed Description

Author

Wissem BEN BETTAIEB

Version

0.1

Date

2022-11-27

Copyright

Copyright (c) 2022

5.13 photo.h

Go to the documentation of this file.

```
1
11 #ifndef PHOTO_H
12 #define PHOTO_H
13 #include "multimedia.h"
18 class Photo: public Multimedia{
19     private:
20         double latitude{},longitude{};
21     public:
22         Photo(){}
23         Photo(string _objName,string _pathName,double _latitude,double _longitude):
24         Multimedia(_objName,_pathName),latitude(_latitude),longitude(_longitude){}
25
26         void setLatitude(double _latitude){
27             latitude=_latitude;
28         }
29         void setLongitude(int _longitude){
30             longitude=_longitude;
31         }
32
33         double getLatitude() const{
34             return latitude;
35         }
36         double getLongitude() const{
37             return longitude;
38         }
44         void afficher(ostream & s) const override{
45             s « "object name is  : " « objName « endl;
46             s « "the path is : " « pathName « endl;
47             s « "Latitude is  : " « latitude « endl;
```

```
48             s « "Longitude is : " « longitude « endl;
49         }
54         void lire(){
55             string path="imagej " + pathName +" &";
56             system(path.data());
57         }
58
59         ~Photo() override{
60             cout « "Photo named " « objName « " was deleted!" « endl;
61         }
62
63
64 };
65 #endif
```

5.14 server.h

```
1 #ifndef SERVER_H
2 #define SERVER_H
3 #include "database.h"
4
5 int serverStart(Database *);
6 #endif
```

5.15 tcpserver.h

```
1 //
2 //  tcpserver: TCP/IP INET Server.
3 //  (c) Eric Lecolinet - Telecom ParisTech - 2016.
4 //  http://www.telecom-paristech.fr/~elc
5 //
6
7 #ifndef __tcpserver__
8 #define __tcpserver__
9 #include <memory>
10 #include <string>
11 #include <functional>
12 #include "ccsocket.h"
13
14 class TCPConnection;
15 class TCPLock;
16
19 class TCPServer {
20 public:
21
22   using Callback =
23   std::function< bool(std::string const& request, std::string& response) >;
24
30   TCPServer(Callback const& callback);
31
32   virtual ~TCPServer();
33
39   virtual int run(int port);
40
41 private:
42   friend class TCPLock;
43   friend class SocketCnx;
44
45   TCPServer(TCPServer const&) = delete;
46   TCPServer& operator=(TCPServer const&) = delete;
47   void error(std::string const& msg);
48
49   ServerSocket servsock_;
50   Callback callback_{};
51 };
52
53 #endif
```

5.16 video.h File Reference

```
#include "multimedia.h"
```

Classes

- class Video

  a Video class that herit from Multimedia class and will be herited by the Film class

5.16.1 Detailed Description

Author

Wissem BEN BETTAIEB

Version

0.1

Date

2022-11-27

Copyright

Copyright (c) 2022

5.17 video.h

Go to the documentation of this file.

```
1
11 #ifndef VIDEO_H
12 #define VIDEO_H
13 #include "multimedia.h"
14
20 class Video : public Multimedia{
21     protected:
22         int duree{};
23     public:
24         Video(){}
25         Video(string _objName,string _pathName,int _duree):Multimedia(_objName,_pathName),duree(_duree){}
26
27         void setDuree(int _duree){
28             duree=_duree;
29         }
30
31         int getDuree() const{
32             return duree;
33         }
34
35         void afficher(ostream & s) const override{
36             s « "object name is  : " « objName « endl;
37             s « "the path is : " « pathName « endl;
38             s « "Duration is  : " « duree « endl;
39         }
44         void lire(){
45             string path="mpv "+ pathName + " &";
46             system(path.data());
47         }
48
49         ~Video() override{
50          cout «" a video named " « objName « " was deleted!" « endl;
51
52         }
53
54 };
55 #endif
```

Index