MultimediaApp

Generated by Doxygen 1.10.0

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 represents a database of multimedia objects. This dataBase contains all multimedia objects, with wich the server will interact It contains two tables, one to store all multimedias and another to store groupes.

`#include <DataBase.h>`

Public Member Functions

- **DataBase** ()

  Constructor for the DataBase class.
- HeadPtr **createFilm** (std::string multimediaName, std::string filePath, unsigned int duration, int ∗chapters)
- HeadPtr **createVideo** (std::string multimediaName, std::string filePath, unsigned int duration)
- HeadPtr **createPhoto** (std::string multimediaName, std::string filePath, unsigned int length, unsigned int height)
- GroupePtr **createGroupe** (std::string name)
- HeadPtr findMultimedia (std::string multimediaName)

  Finds a multimedia object by its name.
- std::string playMultimedia (std::string multimediaName)

  Plays a multimedia object.
- void deleteMultimedia (std::string name)

  Deletes a multimedia object.
- bool isValidName (const std::string &name)

  Checks if a name is valid.
- std::string showAllMultimedia ()

  Displays details of all multimedia objects.
- std::string showAllgroup ()

  Displays details of all group objects.

4.1.1 Detailed Description

The DataBase class represents a database of multimedia objects. This dataBase contains all multimedia objects, with wich the server will interact It contains two tables, one to store all multimedias and another to store groupes.

4.1.2 Member Function Documentation

4.1.2.1 deleteMultimedia()

```
void DataBase::deleteMultimedia (
            std::string name )
```

Deletes a multimedia object.

Parameters

name	The name of the multimedia object to delete.

4.1.2.2 findMultimedia()

```
HeadPtr DataBase::findMultimedia (
            std::string multimediaName )
```

Finds a multimedia object by its name.

Parameters

multimediaName	The name of the multimedia to find.

Returns

A shared pointer to the found multimedia object, or nullptr if not found.

4.1.2.3 isValidName()

```
bool DataBase::isValidName (
            const std::string & name )
```

Checks if a name is valid.

Parameters

name	The name to check.

Returns

true if the name is valid, false otherwise.

4.1.2.4 playMultimedia()

```
std::string DataBase::playMultimedia (
            std::string multimediaName )
```

Plays a multimedia object.

Parameters

multimediaName	The name of the multimedia to play.

Returns

A string indicating the status of the operation.

4.1.2.5 showAllgroup()

```
std::string DataBase::showAllgroup ( )
```

Displays details of all group objects.

Returns

A string containing details of all group objects.

4.1.2.6 showAllMultimedia()

```
std::string DataBase::showAllMultimedia ( )
```

Displays details of all multimedia objects.

Returns

A string containing details of all multimedia objects.

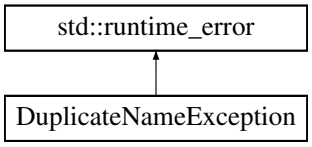
The documentation for this class was generated from the following files:

- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/DataBase.h
- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/DataBase.cpp

4.2 DuplicateNameException Class Reference

```
#include <Exception.h>
```

Inheritance diagram for DuplicateNameException:



Public Member Functions

- DuplicateNameException (const std::string &message)

4.2.1 Detailed Description

Custom exception class for indicating a duplicate name error

4.2.2 Constructor & Destructor Documentation

4.2.2.1 DuplicateNameException()

```
DuplicateNameException::DuplicateNameException (
            const std::string & message )  [inline]
```

Constructor accepting an error message

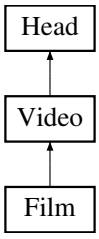
The documentation for this class was generated from the following file:

- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/Exception.h

4.3 Film Class Reference

```
#include <Film.h>
```

Inheritance diagram for Film:



Public Member Functions

- **Film** (std::string multimedia, std::string filePath, unsigned int duration, int ∗chapters)
- **Film** (const Film &from)
- Film & **operator=** (const Film &from)
- int **getChapterNumber** () const
- const int ∗ **getChapters** () const
- void **setChapters** (int ∗chapters)
- void **showChapterDuration** () const
- void display (std::ostream &s) const override

  Displays information about the multimedia object.
- std::string play () const override

  Virtual function to play the multimedia object.

Public Member Functions inherited from Video

- **Video** (std::string multimedia, std::string filePath, unsigned int duration)
- unsigned int **getDuration** () const
- void display (std::ostream &s) const override

  Displays information about the multimedia object.
- std::string play () const override

  Virtual function to play the multimedia object.

Public Member Functions inherited from Head

- virtual ~**Head** ()

  Destructor.
- void setMultimediaName (std::string name)

  Sets the name of the multimedia.
- void setFilePathName (std::string path)

  Sets the file path of the multimedia.
- std::string getMultimediaName () const

  Gets the name of the multimedia.
- std::string getFilePathName () const

  Gets the file path of the multimedia.

Friends

- class **DataBase**

Additional Inherited Members

Protected Member Functions inherited from Head

- Head (std::string multimedia, std::string filePath)

  Constructor with parameters.
- **Head** ()

  Default constructor.

Protected Attributes inherited from Video

- unsigned **duration**

Protected Attributes inherited from Head

- std::string multimediaName
- std::string filePathName

4.3.1 Detailed Description

Declaration of Film class, inheriting from Video

4.3.2 Member Function Documentation

4.3.2.1 display()

```
void Film::display (
            std::ostream & os ) const  [override], [virtual]
```

Displays information about the multimedia object.

Parameters

os	The output stream to write to.

Reimplemented from Head.

4.3.2.2 play()

```
std::string Film::play ( ) const  [override], [virtual]
```

Virtual function to play the multimedia object.

Implements Head.

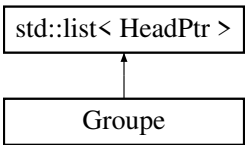
The documentation for this class was generated from the following files:

- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/Film.h
- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/Film.cpp

4.4 Groupe Class Reference

```
#include <Groupe.h>
```

Inheritance diagram for Groupe:



Public Member Functions

- std::string **displayObjectDetails** ()
- std::string **getName** ()

Friends

- class DataBase

4.4.1 Detailed Description

Declaration of Groupe class, inheriting from std::list<HeadPtr>.

4.4.2 Friends And Related Symbol Documentation

4.4.2.1 DataBase

```
friend class DataBase  [friend]
```

Allow DataBase class to access private members.

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

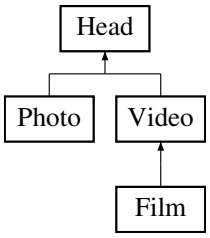
- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/Groupe.h
- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/Groupe.cpp

4.5 Head Class Reference

Interface which describes the common characteristics of multimedia objects. All other multimedia classes will inherit from this interface.

```
#include <Head.h>
```

Inheritance diagram for Head:



Public Member Functions

- virtual ~**Head** ()

  Destructor.
- void setMultimediaName (std::string name)

  Sets the name of the multimedia.
- void setFilePathName (std::string path)

  Sets the file path of the multimedia.
- std::string getMultimediaName () const

  Gets the name of the multimedia.
- std::string getFilePathName () const

  Gets the file path of the multimedia.
- virtual void display (std::ostream &os) const

  Displays information about the multimedia object.
- virtual std::string play () const =0

  Virtual function to play the multimedia object.

Protected Member Functions

- Head (std::string multimedia, std::string filePath)

  Constructor with parameters.
- **Head** ()

  Default constructor.

Protected Attributes

- std::string multimediaName
- std::string filePathName

Friends

- class **DataBase**

4.5.1 Detailed Description

Interface which describes the common characteristics of multimedia objects. All other multimedia classes will inherit from this interface.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Head()

```
Head::Head (
            std::string multimedia,
            std::string filePath )  [protected]
```

Constructor with parameters.

Parameters

multimedia	The name of the multimedia.
filePath	The file path of the multimedia.

4.5.3 Member Function Documentation

4.5.3.1 display()

```
void Head::display (
            std::ostream & os ) const  [virtual]
```

Displays information about the multimedia object.

Parameters

os	The output stream to write to.

Reimplemented in Film, Photo, and Video.

4.5.3.2 getFilePathName()

```
std::string Head::getFilePathName ( ) const
```

Gets the file path of the multimedia.

Returns

The file path of the multimedia.

4.5.3.3 getMultimediaName()

```
std::string Head::getMultimediaName ( ) const
```

Gets the name of the multimedia.

Returns

The name of the multimedia.

4.5.3.4 play()

```
virtual std::string Head::play ( ) const  [pure virtual]
```

Virtual function to play the multimedia object.

Implemented in Film, Photo, and Video.

4.5.3.5 setFilePathName()

```
void Head::setFilePathName (
            std::string path )
```

Sets the file path of the multimedia.

Parameters

path	The file path to set.

4.5.3.6 setMultimediaName()

```
void Head::setMultimediaName (
            std::string name )
```

Sets the name of the multimedia.

Parameters

name	The name to set.

4.5.4 Member Data Documentation

4.5.4.1 filePathName

```
std::string Head::filePathName  [protected]
```

File path name

4.5.4.2 multimediaName

```
std::string Head::multimediaName  [protected]
```

Multimedia name

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

- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/Head.h
- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/Head.cpp

4.6 InputBuffer Struct Reference

Public Member Functions

- **InputBuffer** (size_t size)

Public Attributes

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

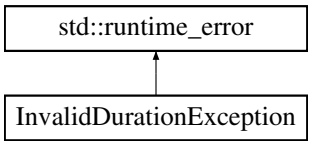
The documentation for this struct was generated from the following file:

- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/ccsocket.cpp

4.7 InvalidDurationException Class Reference

`#include <Exception.h>`

Inheritance diagram for InvalidDurationException:



Public Member Functions

- **InvalidDurationException** (const std::string &message)

4.7.1 Detailed Description

Custom exception class for indicating an invalid duration error

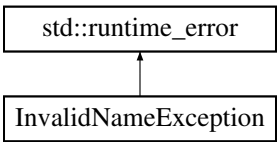
The documentation for this class was generated from the following file:

- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/Exception.h

4.8 InvalidNameException Class Reference

`#include <Exception.h>`

Inheritance diagram for InvalidNameException:



Public Member Functions

- **InvalidNameException** (const std::string &message)

4.8.1 Detailed Description

Custom exception class for indicating an invalid name error

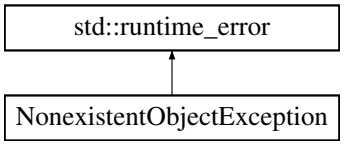
The documentation for this class was generated from the following file:

- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/Exception.h

4.9 NonexistentObjectException Class Reference

`#include <Exception.h>`

Inheritance diagram for NonexistentObjectException:



Public Member Functions

- **NonexistentObjectException** (const std::string &message)

4.9.1 Detailed Description

Custom exception class for indicating a nonexistent object error

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

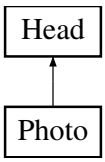
- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/Exception.h

4.10 Photo Class Reference

Class representing a photo multimedia object.

`#include <Photo.h>`

Inheritance diagram for Photo:



Public Member Functions

- unsigned int getHeight () const
  Gets the height of the photo.
- unsigned int getLength () const
  Gets the length of the photo.
- virtual void display (std::ostream &s) const override
  Displays information about the photo.
- std::string play () const override
  Plays the photo.
- ~**Photo** ()
  Destructor.

Public Member Functions inherited from Head

- virtual ~**Head** ()

  Destructor.
- void setMultimediaName (std::string name)

  Sets the name of the multimedia.
- void setFilePathName (std::string path)

  Sets the file path of the multimedia.
- std::string getMultimediaName () const

  Gets the name of the multimedia.
- std::string getFilePathName () const

  Gets the file path of the multimedia.

Friends

- class **DataBase**

Additional Inherited Members

Protected Member Functions inherited from Head

- Head (std::string multimedia, std::string filePath)

  Constructor with parameters.
- **Head** ()

  Default constructor.

Protected Attributes inherited from Head

- std::string multimediaName
- std::string filePathName

4.10.1 Detailed Description

Class representing a photo multimedia object.

4.10.2 Member Function Documentation

4.10.2.1 display()

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

Displays information about the photo.

Parameters

s	The output stream to write to.

Reimplemented from Head.

4.10.2.2 getHeight()

```
unsigned int Photo::getHeight ( ) const  [inline]
```

Gets the height of the photo.

Returns

The height of the photo.

4.10.2.3 getLength()

```
unsigned int Photo::getLength ( ) const  [inline]
```

Gets the length of the photo.

Returns

The length of the photo.

4.10.2.4 play()

```
std::string Photo::play ( ) const  [inline], [override], [virtual]
```

Plays the photo.

Implements Head.

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

- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/Photo.h

4.11 ServerSocket Class Reference

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

Public Member Functions

- ServerSocket ()
- 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).

4.11.1 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.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 ServerSocket()

```
ServerSocket::ServerSocket ( )
```

Creates a listening socket that waits for connection requests by TCP/IP clients.

4.11.3 Member Function Documentation

4.11.3.1 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.

4.11.3.2 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:

- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/ccsocket.h
- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/ccsocket.cpp

4.12 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**

4.12.1 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.

4.12.2 Member Enumeration Documentation

4.12.2.1 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

4.12.3 Constructor & Destructor Documentation

4.12.3.1 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)

4.12.4 Member Function Documentation

4.12.4.1 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

4.12.4.2 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

4.12.4.3 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

4.12.4.4 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.

4.12.4.5 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.

4.12.4.6 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:

- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/ccsocket.h
- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/ccsocket.cpp

4.13 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)

4.13.1 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;
}
```

4.13.2 Constructor & Destructor Documentation

4.13.2.1 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.

4.13.3 Member Function Documentation

4.13.3.1 read()

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

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

Returns

see readLine()

4.13.3.2 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().

4.13.3.3 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,

4.13.3.4 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,

4.13.3.5 write()

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

Writes *len* bytes to the socket.

Returns

see readLine()

4.13.3.6 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:

- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/ccsocket.h
- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/ccsocket.cpp

4.14 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_**

4.14.1 Detailed Description

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

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

- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/tcpserver.cpp

4.15 TCPServer Class Reference

```
#include <tcpserver.h>
```

Public Types

- using Callback

Public Member Functions

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

Friends

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

4.15.1 Detailed Description

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

4.15.2 Member Typedef Documentation

4.15.2.1 Callback

```
using TCPServer::Callback
```

Initial value:

```
  std::function< bool(std::string const& request, std::string& response) >
```

4.15.3 Constructor & Destructor Documentation

4.15.3.1 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.

4.15.4 Member Function Documentation

4.15.4.1 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:

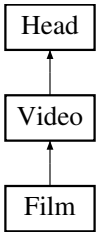
- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/tcpserver.h
- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/tcpserver.cpp

4.16 Video Class Reference

declaration and definition of the Video class inheriting from Head

```
#include <Video.h>
```

Inheritance diagram for Video:



Public Member Functions

- **Video** (std::string multimedia, std::string filePath, unsigned int duration)
- unsigned int **getDuration** () const
- void display (std::ostream &s) const override

  Displays information about the multimedia object.
- std::string play () const override

  Virtual function to play the multimedia object.

Public Member Functions inherited from Head

- virtual ~**Head** ()

  Destructor.
- void setMultimediaName (std::string name)

  Sets the name of the multimedia.
- void setFilePathName (std::string path)

  Sets the file path of the multimedia.
- std::string getMultimediaName () const

  Gets the name of the multimedia.
- std::string getFilePathName () const

  Gets the file path of the multimedia.

Protected Attributes

- unsigned **duration**

Protected Attributes inherited from Head

- std::string multimediaName
- std::string filePathName

Friends

- class **DataBase**

Additional Inherited Members

Protected Member Functions inherited from Head

- Head (std::string multimedia, std::string filePath)

  Constructor with parameters.
- **Head** ()

  Default constructor.

4.16.1 Detailed Description

declaration and definition of the Video class inheriting from Head

4.16.2 Member Function Documentation

4.16.2.1 display()

```
void Video::display (
            std::ostream & os ) const  [inline], [override], [virtual]
```

Displays information about the multimedia object.

Parameters

os	The output stream to write to.

Reimplemented from Head.

4.16.2.2 play()

```
std::string Video::play ( ) const  [inline], [override], [virtual]
```

Virtual function to play the multimedia object.

Implements Head.

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

- /Users/mahamadoutogola/Desktop/programmes/TOGOLA_Mahamadou/cpp/Video.h

Chapter 5

File Documentation

5.1 ccsocket.h

```
00001 //
00002 //  ccsocket: C++ Classes for TCP/IP and UDP Datagram INET Sockets.
00003 //  (c) Eric Lecolinet 2016/2020 - https://www.telecom-paris.fr/~elc
00004 //
00005 //  - Socket: TCP/IP or UDP/Datagram IPv4 socket
00006 //  - ServerSocket: TCP/IP Socket Server
00007 //  - SocketBuffer: preserves record boundaries when exchanging data
00008 //   between TCP/IP sockets.
00009 //
00010
00011 #ifndef ccuty_ccsocket
00012 #define ccuty_ccsocket 1
00013
00014 #include <string>
00015
00016 #if defined(_WIN32) || defined(_WIN64)
00017 #include <winsock2.h>
00018 #define SOCKSIZE int
00019 #define SOCKDATA char
00020
00021 #else
00022 #include <sys/socket.h>
00023 #include <sys/types.h>
00024 #define SOCKET int
00025 #define SOCKADDR struct sockaddr
00026 #define SOCKADDR_IN struct sockaddr_in
00027 #define INVALID_SOCKET -1
00028 #define SOCKSIZE ssize_t
00029 #define SOCKDATA void
00030 #endif
00031
00032 // ignore SIGPIPES when possible
00033 #if defined(MSG_NOSIGNAL)
00034 #define NO_SIGPIPE_(flags) (flags | MSG_NOSIGNAL)
00035 #else
00036 #define NO_SIGPIPE_(flags) (flags)
00037 #endif
00038
00047 class Socket {
00048 public:
00053   enum Errors { Failed = -1, InvalidSocket = -2, UnknownHost = -3 };
00054
00059   static void startup();
00060   static void cleanup();
00062
00067   Socket(int type = SOCK_STREAM);
00068
00070   Socket(int type, SOCKET sockfd);
00071
00073   ~Socket();
00074
00081   int connect(const std::string &host, int port);
00082
00085   int bind(int port);
00086
00091   int bind(const std::string &host, int port);
00092
00094   int close();
```

```
00095
00097   bool isClosed() const { return sockfd_ == INVALID_SOCKET; }
00098
00100   SOCKET descriptor() { return sockfd_; }
00101
00103   void shutdownInput();
00104
00106   void shutdownOutput();
00107
00113   SOCKSIZE send(const SOCKDATA *buf, size_t len, int flags = 0) {
00114     return ::send(sockfd_, buf, len, NO_SIGPIPE_(flags));
00115   }
00116
00122   SOCKSIZE receive(SOCKDATA *buf, size_t len, int flags = 0) {
00123     return ::recv(sockfd_, buf, len, flags);
00124   }
00125
00126 #if !defined(_WIN32) && !defined(_WIN64)
00127
00129   SOCKSIZE sendTo(void const *buf, size_t len, int flags, SOCKADDR const *to,
00130                   socklen_t addrlen) {
00131     return ::sendto(sockfd_, buf, len, NO_SIGPIPE_(flags), to, addrlen);
00132   }
00133
00135   SOCKSIZE receiveFrom(void *buf, size_t len, int flags, SOCKADDR *from,
00136                        socklen_t *addrlen) {
00137     return ::recvfrom(sockfd_, buf, len, flags, from, addrlen);
00138   }
00139
00141   int setReceiveBufferSize(int size);
00142
00144   int setReuseAddress(bool);
00145
00147   int setSendBufferSize(int size);
00148
00150   int setSoLinger(bool, int linger);
00151
00153   int setSoTimeout(int timeout);
00154
00156   int setTcpNoDelay(bool);
00157
00159   int getReceiveBufferSize() const;
00160
00162   bool getReuseAddress() const;
00163
00165   int getSendBufferSize() const;
00166
00168   bool getSoLinger(int &linger) const;
00169
00171   int getSoTimeout() const;
00172
00174   bool getTcpNoDelay() const;
00175
00176 #endif
00177
00178 private:
00179   friend class ServerSocket;
00180
00181   // Initializes a local INET4 address, returns 0 on success, -1 otherwise.
00182   int setLocalAddress(SOCKADDR_IN &addr, int port);
00183   // Initializes a remote INET4 address, returns 0 on success, -1 otherwise.
00184   int setAddress(SOCKADDR_IN &addr, const std::string &host, int port);
00185
00186   SOCKET sockfd_{};
00187   Socket(const Socket &) = delete;
00188   Socket &operator=(const Socket &) = delete;
00189   Socket &operator=(Socket &&) = delete;
00190 };
00191
00196 class ServerSocket {
00197 public:
00200   ServerSocket();
00201
00202   ~ServerSocket();
00203
00208   Socket *accept();
00209
00212   int bind(int port, int backlog = 50);
00213
00215   int close();
00216
00218   bool isClosed() const { return sockfd_ == INVALID_SOCKET; }
00219
00221   SOCKET descriptor() { return sockfd_; }
00222
00223 #if !defined(_WIN32) && !defined(_WIN64)
00224
```

```
00226   int setReceiveBufferSize(int size);
00227
00229   int setReuseAddress(bool);
00230
00232   int setSoTimeout(int timeout);
00233
00235   int setTcpNoDelay(bool);
00236
00237 #endif
00238
00239 private:
00240   Socket *createSocket(SOCKET);
00241   SOCKET sockfd_{}; // listening socket.
00242   ServerSocket(const ServerSocket &) = delete;
00243   ServerSocket &operator=(const ServerSocket &) = delete;
00244   ServerSocket &operator=(ServerSocket &&) = delete;
00245 };
00246
00285 class SocketBuffer {
00286 public:
00292   SocketBuffer(Socket *, size_t inputSize = 8192, size_t ouputSize = 8192);
00293   SocketBuffer(Socket &, size_t inputSize = 8192, size_t ouputSize = 8192);
00295
00296   ~SocketBuffer();
00297
00311   SOCKSIZE readLine(std::string &message);
00312
00321   SOCKSIZE writeLine(const std::string &message);
00322
00325   SOCKSIZE read(char *buffer, size_t len);
00326
00329   SOCKSIZE write(const char *str, size_t len);
00330
00332   Socket *socket() { return sock_; }
00333
00341   void setReadSeparator(int separ);
00342   int readSeparator() const { return insep_; }
00343   // @}
00344
00352   void setWriteSeparator(int separ);
00353   int writeSeparator() const { return outsep_; }
00354   // @}
00355
00356 private:
00357   SocketBuffer(const SocketBuffer &) = delete;
00358   SocketBuffer &operator=(const SocketBuffer &) = delete;
00359   SocketBuffer &operator=(SocketBuffer &&) = delete;
00360
00361 protected:
00362   bool retrieveLine(std::string &str, SOCKSIZE received);
00363   size_t insize_{}, outsize_{};
00364   int insep_{}, outsep_{};
00365   Socket *sock_{};
00366   struct InputBuffer *in_{};
00367 };
00368
00369 #endif
```

5.2 DataBase.h

```
00001 #ifndef DATABASE_H
00002 #define DATABASE_H
00003
00004 #include "Film.h"
00005 #include "Groupe.h"
00006 #include "Head.h"
00007 #include "Photo.h"
00008 #include "Video.h"
00009 #include <map>
00010 #include <memory>
00011 #include <string>
00012
00013 // Define aliases for convenience
00014 using HeadPtr = std::shared_ptr<Head>;
00015 using GroupePtr = std::shared_ptr<Groupe>;
00016 using HEAD_TABLE = std::map<std::string, HeadPtr>;
00017 using GROUPE_TABLE = std::map<std::string, GroupePtr>;
00018
00025 class DataBase {
00026
00027 private:
00028   HEAD_TABLE headTable;     // Map to store multimedia objects
00029   GROUPE_TABLE groupeTable; // Map to store group objects
```

```
00030
00031 public:
00035   DataBase();
00036
00037   // Methods for adding A Film object in the dataBase
00038   HeadPtr createFilm(std::string multimediaName, std::string filePath,
00039                      unsigned int duration, int *chapters);
00040   // Methods for adding A Video object in the dataBase
00041   HeadPtr createVideo(std::string multimediaName, std::string filePath,
00042                       unsigned int duration);
00043   // Methods for adding A Photo object in the dataBase
00044   HeadPtr createPhoto(std::string multimediaName, std::string filePath,
00045                       unsigned int length, unsigned int height);
00046   // Methods for adding A Groupe object in the dataBase
00047   GroupePtr createGroupe(std::string name);
00048
00055   HeadPtr findMultimedia(std::string multimediaName);
00056
00062   std::string playMultimedia(std::string multimediaName);
00063
00068   void deleteMultimedia(std::string name);
00069
00075   bool isValidName(const std::string &name);
00076
00081   std::string showAllMultimedia();
00082
00087   std::string showAllgroup();
00088 };
00089
00090 #endif // DATABASE_H
```

5.3 Exception.h

```
00001 #ifndef EXCEPTION_H
00002 #define EXCEPTION_H
00003
00004 #include <stdexcept>
00005
00009 class DuplicateNameException : public std::runtime_error {
00010
00011 public:
00015   DuplicateNameException(const std::string &message)
00016       : std::runtime_error(message) {}
00017 };
00018
00022 class InvalidNameException : public std::runtime_error {
00023 public:
00024   /* Constructor accepting an error message */
00025   InvalidNameException(const std::string &message)
00026       : std::runtime_error(message) {}
00027 };
00028
00032 class NonexistentObjectException : public std::runtime_error {
00033 public:
00034   /* Constructor accepting an error message */
00035   NonexistentObjectException(const std::string &message)
00036       : std::runtime_error(message) {}
00037 };
00038
00042 class InvalidDurationException : public std::runtime_error {
00043 public:
00044   /* Constructor accepting an error message */
00045   InvalidDurationException(const std::string &message)
00046       : std::runtime_error(message) {}
00047 };
00048
00049 #endif
```

5.4 Film.h

```
00001 #ifndef FILM_H
00002 #define FILM_H
00003
00004 #include "Head.h"
00005 #include "Video.h"
00006
00007 /*Forward declaration of DataBase class */
00008 class DataBase;
00012 class Film : public Video {
```

```
00013   friend class DataBase; // Allowing DataBase class to access private members
00014
00015 private:
00016   int *chapters;
00018 public:
00019   /* Constructor */
00020   Film(std::string multimedia, std::string filePath, unsigned int duration,
00021        int *chapters);
00022
00023   /* Copy constructor */
00024   Film(const Film &from);
00025
00026   /* Assignment operator */
00027   Film &operator=(const Film &from);
00028
00029   /* Destructor */
00030   ~Film();
00031
00032   /* Getter method to retrieve the number of chapters */
00033   int getChapterNumber() const;
00034
00035   /* Getter method to retrieve the array of chapter durations */
00036   const int *getChapters() const;
00037
00038   /* Setter method to set the array of chapter durations */
00039   void setChapters(int *chapters);
00040
00041   /* Method to display the duration of each chapter */
00042   void showChapterDuration() const;
00043
00044   /* Method to display film details */
00045   void display(std::ostream &s) const override;
00046   std::string play() const override;
00047
00048
00049
00050 };
00051
00052 #endif
```

5.5 Groupe.h

```
00001 #ifndef GROUPE_H
00002 #define GROUPE_H
00003 #include <memory>
00004 #include "Head.h"
00005 #include <list>
00008 typedef std::shared_ptr<Head> HeadPtr;
00009
00010 class DataBase;
00011
00015 class Groupe : public std::list<HeadPtr> {
00016   friend class DataBase;
00018 private:
00019   std::string name;
00021   /* Private constructor to prevent instantiation outside DataBase class. */
00022   Groupe(std::string name);
00023
00024 public:
00025   /* Method to display details of objects in the group. */
00026   std::string displayObjectDetails();
00027
00028   /* Method to retrieve the name of the group. */
00029   std::string getName();
00030 };
00031
00032 #endif // GROUPE_H
```

5.6 Head.h

```
00001 #ifndef HEAD
00002 #define HEAD
00003
00004 #include <iostream>
00005 #include <string>
00006
00007 class DataBase;
00008
00011 class Head {
```

```
00012   friend class DataBase;
00013
00014 protected:
00015   std::string multimediaName;
00016   std::string filePathName;
00023   Head(std::string multimedia, std::string filePath);
00024
00026   Head();
00027
00028 public:
00030   virtual ~Head();
00031
00036   void setMultimediaName(std::string name);
00037
00042   void setFilePathName(std::string path);
00043
00048   std::string getMultimediaName() const;
00049
00054   std::string getFilePathName() const;
00055
00060   virtual void display(std::ostream &os) const;
00061
00063   virtual std::string play() const = 0;
00064 };
00065
00066 #endif
```

5.7 Photo.h

```
00001 #if !defined(PHOTO)
00002 #define PHOTO
00003
00004 #include "DataBase.h"
00005 #include "Head.h"
00006 #include <cstdlib>
00007 #include <sys/types.h>
00008 #include <sys/wait.h>
00009 #include <unistd.h>
00010
00012 class Photo : public Head {
00013   friend class DataBase;
00014
00015 private:
00016   unsigned int length;
00017   unsigned int height;
00026   Photo(std::string multimedia, std::string filePath, unsigned int length,
00027         unsigned int height)
00028       : Head(multimedia, filePath), length(length), height(height){};
00029
00030 public:
00035   unsigned int getHeight() const { return height; };
00036
00041   unsigned int getLength() const { return length; };
00042
00047   virtual void display(std::ostream &s) const override {
00048
00049     s « "Multimedia type: PHOTO, "
00050       « "Multimedia name: " « this->getMultimediaName() « ", "
00051       « "File path: " « this->getFilePathName() « ", "
00052       « "Height: " « this->getHeight() « ", "
00053       « "Length: " « this->getLength() « ", ";
00054   };
00055
00059 std::string play() const override {
00060
00061    //Construct the command to open the image with ImageJ in the background
00062     std::string command = "imagej " + getFilePathName() + " &";
00063
00064     // Execute the command
00065     int status = std::system(command.c_str());
00066
00067     // Check if the command was successfully executed
00068     if (status == 0) {
00069         return "Image played successfully!";
00070     } else {
00071         return "Failed to play photo '" + getMultimediaName() + "'.";
00072     }
00073 }
00074
00076   ~Photo() {
00077     // std::cout « "décédé" « std::endl;
00078   }
00079 };
```

```
00080
00081 #endif
```

5.8 tcpserver.h

```
00001 //
00002 //  tcpserver: TCP/IP INET Server.
00003 //  (c) Eric Lecolinet - Telecom ParisTech - 2016.
00004 //  http://www.telecom-paristech.fr/~elc
00005 //
00006
00007 #ifndef __tcpserver__
00008 #define __tcpserver__
00009 #include <memory>
00010 #include <string>
00011 #include <functional>
00012 #include "ccsocket.h"
00013
00014 class TCPConnection;
00015 class TCPLock;
00016
00019 class TCPServer {
00020 public:
00021
00022   using Callback =
00023   std::function< bool(std::string const& request, std::string& response) >;
00024
00030   TCPServer(Callback const& callback);
00031
00032   virtual ~TCPServer();
00033
00039   virtual int run(int port);
00040
00041 private:
00042   friend class TCPLock;
00043   friend class SocketCnx;
00044
00045   TCPServer(TCPServer const&) = delete;
00046   TCPServer& operator=(TCPServer const&) = delete;
00047   void error(std::string const& msg);
00048
00049   ServerSocket servsock_;
00050   Callback callback_{};
00051 };
00052
00053 #endif
```

5.9 Video.h

```
00001 #ifndef VIDEO_H
00002 #define VIDEO_H
00003
00004 #include "Head.h"
00005 #include <iostream>
00006
00007 // Forward declaration of DataBase class to avoid circular dependency
00008 class DataBase;
00009
00011 class Video : public Head {
00012   friend class DataBase; // Allowing DataBase class to access private members
00013
00014 protected:
00015   unsigned duration; // Duration of the video in seconds
00016
00017 public:
00018   // Constructor initializing multimedia name, file path, and duration
00019   Video(std::string multimedia, std::string filePath, unsigned int duration)
00020       : Head(multimedia, filePath), duration(duration){};
00021
00022   // Method to retrieve the duration of the video
00023   unsigned int getDuration() const { return duration; };
00024
00025   // Method to display details of the video
00026   void display(std::ostream &s) const override {
00027     s « "Multimedia type: VIDEO, "
00028       « "Multimedia name: " « this->getMultimediaName() « ", "
00029       « "File path: " « this->getFilePathName() « ", "
00030       « "Duration: " « this->getDuration() « ", ";
00031   };
```

```
00032
00033   // Method to play the video
00034   std::string play() const override {
00035     std::string arg =
00036         "mpv " + this->getFilePathName() + " &"; // Command to play video
00037     int status = system(arg.data()); // Execute command to play video in background
00038     if(status != 0){
00039       return "Failed to play video : " + this->getMultimediaName();
00040     }
00041     return "Video played successfully !";
00042
00043   };
00044
00045   // Destructor (currently commented out)
00046   // ~Video() {
00047   //     /*std::cout « "décédé" « std::endl;*/ // Uncomment for debugging
00048   //     purposes
00049   // }
00050 };
00051 #endif // VIDEO_H
```

Index