XU_Ziyang_cpp_doc

Generated by Doxygen 1.12.0

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

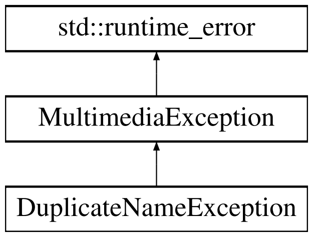
Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 DuplicateNameException Class Reference

Inheritance diagram for DuplicateNameException:



Public Member Functions

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

Public Member Functions inherited from MultimediaException

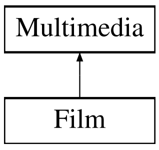
- **MultimediaException** (const std::string &message)

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

- cpp/MultimediaException.h

4.2 Film Class Reference

Inheritance diagram for Film:



Public Member Functions

- **Film** (const std::string &name, const std::string &path, int duration, const std::vector< int > &chapters)
- void display (std::ostream &os) const override
- void play () const override
- void serialize (std::ofstream &ofs) const override
- void deserialize (std::ifstream &ifs) override
- std::string getClassName () const override

Public Member Functions inherited from Multimedia

- **Multimedia** (const std::string &name, const std::string &path)
- std::string **getName** () const

Additional Inherited Members

Protected Attributes inherited from Multimedia

- std::string **name**
- std::string **path**

4.2.1 Member Function Documentation

4.2.1.1 deserialize()

```
void Film::deserialize (
             std::ifstream & ifs)  [override], [virtual]
```

Implements Multimedia.

4.2.1.2 display()

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

Reimplemented from Multimedia.

4.2.1.3 getClassName()

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

Implements Multimedia.

4.2.1.4 play()

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

Implements Multimedia.

4.2.1.5 serialize()

```
void Film::serialize (
            std::ofstream & ofs) const  [override], [virtual]
```

Implements Multimedia.

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

- cpp/Film.h
- cpp/Film.cpp

4.3 Group Class Reference

Public Member Functions

- **Group** (const std::string &name)
- void **push_back** (const std::shared_ptr< Multimedia > &item)
- void **remove** (const std::shared_ptr< Multimedia > &item)
- void **display** (std::ostream &os) const

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

- cpp/Group.h
- cpp/Group.cpp

4.4 InputBuffer Struct Reference

Public Member Functions

- **InputBuffer** (size_t size)

Public Attributes

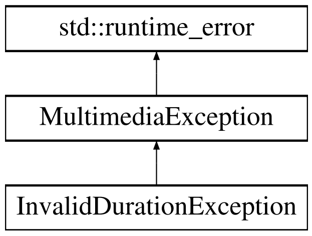
- char * **buffer**
- char * **begin**
- char * **end**
- SOCKSIZE **remaining**

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

- cpp/ccsocket.cpp

4.5 InvalidDurationException Class Reference

Inheritance diagram for InvalidDurationException:



Public Member Functions

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

Public Member Functions inherited from MultimediaException

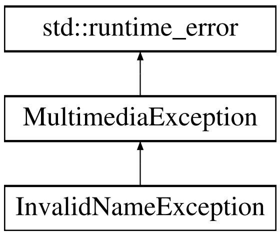
- **MultimediaException** (const std::string &message)

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

- cpp/MultimediaException.h

4.6 InvalidNameException Class Reference

Inheritance diagram for InvalidNameException:



Public Member Functions

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

Public Member Functions inherited from MultimediaException

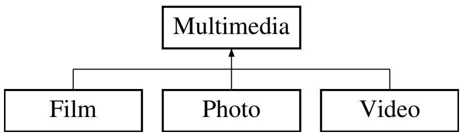
- **MultimediaException** (const std::string &message)

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

- cpp/MultimediaException.h

4.7 Multimedia Class Reference

Inheritance diagram for Multimedia:



Public Member Functions

- **Multimedia** (const std::string &name, const std::string &path)
- virtual void **display** (std::ostream &os) const
- virtual void **play** () const =0
- std::string **getName** () const
- virtual void **serialize** (std::ofstream &ofs) const =0
- virtual void **deserialize** (std::ifstream &ifs)=0
- virtual std::string **getClassName** () const =0

Protected Attributes

- std::string **name**
- std::string **path**

Friends

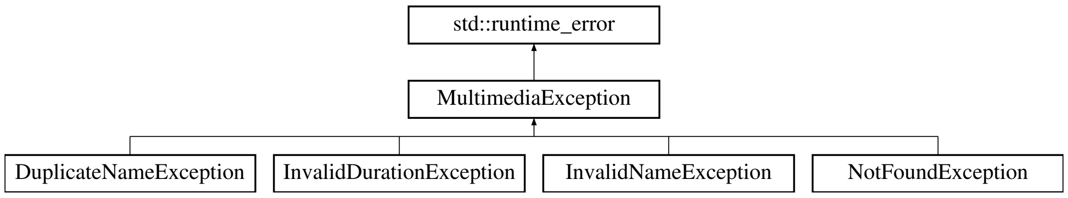
- std::ofstream & **operator**<< (std::ofstream &ofs, const Multimedia &multimedia)
- std::ifstream & **operator**>> (std::ifstream &ifs, Multimedia &multimedia)

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

- cpp/Multimedia.h
- cpp/Multimedia.cpp

4.8 MultimediaException Class Reference

Inheritance diagram for MultimediaException:



Public Member Functions

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

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

- cpp/MultimediaException.h

4.9 MultimediaManager Class Reference

Public Member Functions

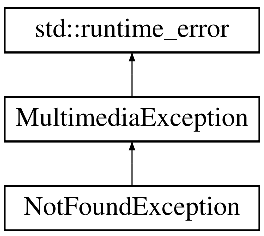
- void **createPhoto** (const std::string &name, const std::string &path, double latitude, double longitude)
- void **createVideo** (const std::string &name, const std::string &path, int duration)
- void **createFilm** (const std::string &name, const std::string &path, int duration, const std::vector< int > &chapters)
- void **createGroup** (const std::string &name)
- void **displayMultimedia** (const std::string &name) const
- void **displayGroup** (const std::string &name) const
- void **playMultimedia** (const std::string &name) const
- void **save** (const std::string &filename) const
- void **load** (const std::string &filename)

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

- cpp/MultimediaManager.h
- cpp/MultimediaManager.cpp

4.10 NotFoundException Class Reference

Inheritance diagram for NotFoundException:



Public Member Functions

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

Public Member Functions inherited from MultimediaException

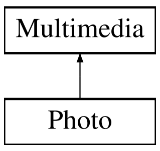
- **MultimediaException** (const std::string &message)

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

- cpp/MultimediaException.h

4.11 Photo Class Reference

Inheritance diagram for Photo:



Public Member Functions

- **Photo** (const std::string &name, const std::string &path, double latitude, double longitude)
- void display (std::ostream &os) const override
- void play () const override
- void serialize (std::ofstream &ofs) const override
- void deserialize (std::ifstream &ifs) override
- std::string getClassName () const override

Public Member Functions inherited from Multimedia

- **Multimedia** (const std::string &name, const std::string &path)
- std::string **getName** () const

Additional Inherited Members

Protected Attributes inherited from Multimedia

- std::string **name**
- std::string **path**

4.11.1 Member Function Documentation

4.11.1.1 deserialize()

```
void Photo::deserialize (
             std::ifstream & ifs)  [override], [virtual]
```

Implements Multimedia.

4.11.1.2 display()

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

Reimplemented from Multimedia.

4.11.1.3 getClassName()

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

Implements Multimedia.

4.11.1.4 play()

```
void Photo::play () const  [override], [virtual]
```

Implements Multimedia.

4.11.1.5 serialize()

```
void Photo::serialize (
             std::ofstream & ofs) const  [override], [virtual]
```

Implements Multimedia.

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

- cpp/Photo.h
- cpp/Photo.cpp

4.12 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).

4.12.1 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.

4.12.2 Member Function Documentation

4.12.2.1 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.

4.12.2.2 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:

- cpp/ccsocket.h
- cpp/ccsocket.cpp

4.13 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**

4.13.1 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.

4.13.2 Member Enumeration Documentation

4.13.2.1 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

4.13.3 Constructor & Destructor Documentation

4.13.3.1 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)

4.13.4 Member Function Documentation

4.13.4.1 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

4.13.4.2 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

4.13.4.3 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

4.13.4.4 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.

4.13.4.5 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.

4.13.4.6 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:

- cpp/ccsocket.h
- cpp/ccsocket.cpp

4.14 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)

4.14.1 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;
}
```

4.14.2 Constructor & Destructor Documentation

4.14.2.1 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.

4.14.3 Member Function Documentation

4.14.3.1 read()

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

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

Returns

see readLine()

4.14.3.2 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().

4.14.3.3 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,

4.14.3.4 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,

4.14.3.5 write()

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

Writes *len* bytes to the socket.

Returns

see readLine()

4.14.3.6 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:

- cpp/ccsocket.h
- cpp/ccsocket.cpp

4.15 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_**

4.15.1 Detailed Description

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

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

- cpp/tcpserver.cpp

4.16 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**

4.16.1 Detailed Description

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

4.16.2 Member Typedef Documentation

4.16.2.1 Callback

```
using TCPServer::Callback
```

Initial value:

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

4.16.3 Constructor & Destructor Documentation

4.16.3.1 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.

4.16.4 Member Function Documentation

4.16.4.1 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

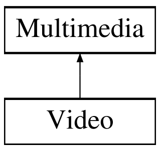
0 on normal termination, or a negative value if the ServerSocket could not be bound (value is then one of Socket::Errors).

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

- cpp/tcpserver.h
- cpp/tcpserver.cpp

4.17 Video Class Reference

Inheritance diagram for Video:



Public Member Functions

- **Video** (const std::string &name, const std::string &path, int duration)
- void display (std::ostream &os) const override
- void play () const override
- void serialize (std::ofstream &ofs) const override
- void deserialize (std::ifstream &ifs) override
- std::string getClassName () const override

Public Member Functions inherited from Multimedia

- **Multimedia** (const std::string &name, const std::string &path)
- std::string **getName** () const

Additional Inherited Members

Protected Attributes inherited from Multimedia

- std::string **name**
- std::string **path**

4.17.1 Member Function Documentation

4.17.1.1 deserialize()

```
void Video::deserialize (
             std::ifstream & ifs)  [override], [virtual]
```

Implements Multimedia.

4.17.1.2 display()

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

Reimplemented from Multimedia.

4.17.1.3 getClassName()

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

Implements Multimedia.

4.17.1.4 play()

```
void Video::play () const  [override], [virtual]
```

Implements Multimedia.

4.17.1.5 serialize()

```
void Video::serialize (
             std::ofstream & ofs) const  [override], [virtual]
```

Implements Multimedia.

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

- cpp/Video.h
- cpp/Video.cpp

Chapter 5

File Documentation

5.1 ccsocket.h

```
//
//  ccsocket: C++ Classes for TCP/IP and UDP Datagram INET Sockets.
//  (c) Eric Lecolinet 2016/2020 - https://www.telecom-paris.fr/~elc
//
//  - Socket: TCP/IP or UDP/Datagram IPv4 socket
//  - ServerSocket: TCP/IP Socket Server
//  - SocketBuffer: preserves record boundaries when exchanging data
//   between TCP/IP sockets.
//

#ifndef ccuty_ccsocket
#define ccuty_ccsocket 1

#include <string>

#if defined(_WIN32) || defined(_WIN64)
#include <winsock2.h>
#define SOCKSIZE int
#define SOCKDATA char

#else
#include <sys/types.h>
#include <sys/socket.h>
#define SOCKET int
#define SOCKADDR struct sockaddr
#define SOCKADDR_IN struct sockaddr_in
#define INVALID_SOCKET -1
#define SOCKSIZE ssize_t
#define SOCKDATA void
#endif

// ignore SIGPIPES when possible
#if defined(MSG_NOSIGNAL)
#  define NO_SIGPIPE_(flags) (flags | MSG_NOSIGNAL)
#else
#  define NO_SIGPIPE_(flags) (flags)
#endif

class Socket {
public:
  enum Errors { Failed = -1, InvalidSocket = -2, UnknownHost = -3 };

  static void startup();
  static void cleanup();

  Socket(int type = SOCK_STREAM);

  Socket(int type, SOCKET sockfd);

  ~Socket();

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

  int bind(int port);

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

  int close();

  bool isClosed() const { return sockfd_ == INVALID_SOCKET; }

  SOCKET descriptor() { return sockfd_; }

  void shutdownInput();

  void shutdownOutput();

  SOCKSIZE send(const SOCKDATA* buf, size_t len, int flags = 0) {
    return ::send(sockfd_, buf, len, NO_SIGPIPE_(flags));
  }

  SOCKSIZE receive(SOCKDATA* buf, size_t len, int flags = 0) {
    return ::recv(sockfd_, buf, len, flags);
  }

#if !defined(_WIN32) && !defined(_WIN64)

  SOCKSIZE sendTo(void const* buf, size_t len, int flags,
                  SOCKADDR const* to, socklen_t addrlen) {
    return ::sendto(sockfd_, buf, len, NO_SIGPIPE_(flags), to, addrlen);
  }

  SOCKSIZE receiveFrom(void* buf, size_t len, int flags,
                       SOCKADDR* from, socklen_t* addrlen) {
    return ::recvfrom(sockfd_, buf, len, flags, from, addrlen);
  }

  int setReceiveBufferSize(int size);

  int setReuseAddress(bool);

  int setSendBufferSize(int size);

  int setSoLinger(bool, int linger);

  int setSoTimeout(int timeout);

  int setTcpNoDelay(bool);

  int getReceiveBufferSize() const;

  bool getReuseAddress() const;

  int getSendBufferSize() const;

  bool getSoLinger(int& linger) const;

  int getSoTimeout() const;

  bool getTcpNoDelay() const;

#endif

private:
  friend class ServerSocket;

  // Initializes a local INET4 address, returns 0 on success, -1 otherwise.
  int setLocalAddress(SOCKADDR_IN& addr, int port);
  // Initializes a remote INET4 address, returns 0 on success, -1 otherwise.
  int setAddress(SOCKADDR_IN& addr, const std::string& host, int port);

  SOCKET sockfd_{};
  Socket(const Socket&) = delete;
  Socket& operator=(const Socket&) = delete;
  Socket& operator=(Socket&&) = delete;
};



class ServerSocket {
public:
  ServerSocket();

  ~ServerSocket();

  Socket* accept();

  int bind(int port, int backlog = 50);

  int close();

  bool isClosed() const { return sockfd_ == INVALID_SOCKET; }

  SOCKET descriptor() { return sockfd_; }

```

```
#if !defined(_WIN32) && !defined(_WIN64)

  int setReceiveBufferSize(int size);

  int setReuseAddress(bool);

  int  setSoTimeout(int timeout);

  int setTcpNoDelay(bool);

#endif

private:
  Socket* createSocket(SOCKET);
  SOCKET sockfd_{};  // listening socket.
  ServerSocket(const ServerSocket&) = delete;
  ServerSocket& operator=(const ServerSocket&) = delete;
  ServerSocket& operator=(ServerSocket&&) = delete;
};


class SocketBuffer {
public:
  SocketBuffer(Socket*, size_t inputSize = 8192, size_t ouputSize = 8192);
  SocketBuffer(Socket&, size_t inputSize = 8192, size_t ouputSize = 8192);

  ~SocketBuffer();

  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() { return sock_; }

  void setReadSeparator(int separ);
  int readSeparator() const { return insep_; }
  // @}

  void setWriteSeparator(int separ);
  int writeSeparator() const { return outsep_; }
  // @}

private:
  SocketBuffer(const SocketBuffer&) = delete;
  SocketBuffer& operator=(const SocketBuffer&) = delete;
  SocketBuffer& operator=(SocketBuffer&&) = delete;

protected:
  bool retrieveLine(std::string& str, SOCKSIZE received);
  size_t insize_{}, outsize_{};
  int insep_{}, outsep_{};
  Socket* sock_{};
  struct InputBuffer* in_{};
};

#endif
```

5.2 Film.h

```
#ifndef FILM_H
#define FILM_H

#include "Multimedia.h"
#include <vector>

class Film : public Multimedia {
public:
    Film(const std::string& name, const std::string& path, int duration, const std::vector<int>&
chapters);
    void display(std::ostream& os) const override;
    void play() const override;
    void serialize(std::ofstream& ofs) const override;
    void deserialize(std::ifstream& ifs) override;
    std::string getClassName() const override;

private:
    int duration;
    std::vector<int> chapters;
};
```

```

#endif // FILM_H
```

5.3 Group.h

```
#ifndef GROUP_H
#define GROUP_H

#include <vector>
#include <memory>
#include <iostream>
#include "Multimedia.h"

class Group {
private:
    std::string name;
    std::vector<std::shared_ptr<Multimedia» items;

public:
    Group(const std::string &name);
    ~Group();

    void push_back(const std::shared_ptr<Multimedia> &item);
    void remove(const std::shared_ptr<Multimedia> &item);
    void display(std::ostream &os) const;
};

using GroupPtr = std::shared_ptr<Group>;

#endif // GROUP_H
```

5.4 Multimedia.h

```
#ifndef MULTIMEDIA_H
#define MULTIMEDIA_H

#include <string>
#include <iostream>
#include <fstream>

class Multimedia {
public:
    Multimedia(const std::string& name, const std::string& path);
    virtual ~Multimedia();
    virtual void display(std::ostream& os) const;
    virtual void play() const = 0;
    std::string getName() const;

    virtual void serialize(std::ofstream& ofs) const = 0;
    virtual void deserialize(std::ifstream& ifs) = 0;
    virtual std::string getClassName() const = 0;

    friend std::ofstream& operator«(std::ofstream& ofs, const Multimedia& multimedia);
    friend std::ifstream& operator»(std::ifstream& ifs, Multimedia& multimedia);

protected:
    std::string name;
    std::string path;
};

#endif // MULTIMEDIA_H
```

5.5 MultimediaException.h

```
// File: MultimediaException.h
#ifndef MULTIMEDIAEXCEPTION_H
#define MULTIMEDIAEXCEPTION_H

#include <stdexcept>
#include <string>

class MultimediaException : public std::runtime_error {
public:
    explicit MultimediaException(const std::string& message) : std::runtime_error(message) {}
};
```

```

class DuplicateNameException : public MultimediaException {
public:
    explicit DuplicateNameException(const std::string& message) : MultimediaException(message) {}
};

class NotFoundException : public MultimediaException {
public:
    explicit NotFoundException(const std::string& message) : MultimediaException(message) {}
};

class InvalidNameException : public MultimediaException {
public:
    explicit InvalidNameException(const std::string& message) : MultimediaException(message) {}
};

class InvalidDurationException : public MultimediaException {
public:
    explicit InvalidDurationException(const std::string& message) : MultimediaException(message) {}
};

#endif // MULTIMEDIAEXCEPTION_H
```

5.6 MultimediaManager.h

```
// File: MultimediaManager.h
#ifndef MULTIMEDIAMANAGER_H
#define MULTIMEDIAMANAGER_H

#include <string>
#include <vector>
#include <memory>
#include <map>
#include "Multimedia.h"
#include "Group.h"
#include "MultimediaException.h"

class MultimediaManager {
public:
    void createPhoto(const std::string& name, const std::string& path, double latitude, double
longitude);
    void createVideo(const std::string& name, const std::string& path, int duration);
    void createFilm(const std::string& name, const std::string& path, int duration, const
std::vector<int>& chapters);
    void createGroup(const std::string& name);
    void displayMultimedia(const std::string& name) const;
    void displayGroup(const std::string& name) const;
    void playMultimedia(const std::string& name) const;
    void save(const std::string& filename) const;
    void load(const std::string& filename);

private:
    std::map<std::string, std::shared_ptr<Multimedia» multimediaMap;
    std::map<std::string, std::shared_ptr<Group» groupMap;
    bool isValidName(const std::string& name) const;
};

#endif // MULTIMEDIAMANAGER_H
```

5.7 Photo.h

```
#ifndef PHOTO_H
#define PHOTO_H

#include "Multimedia.h"

class Photo : public Multimedia {
public:
    Photo(const std::string& name, const std::string& path, double latitude, double longitude);
    void display(std::ostream& os) const override;
    void play() const override;
    void serialize(std::ofstream& ofs) const override;
    void deserialize(std::ifstream& ifs) override;
    std::string getClassName() const override;

private:
    double latitude;
    double longitude;
};

#endif // PHOTO_H
```

5.8 tcpserver.h

```
//
//  tcpserver: TCP/IP INET Server.
//  (c) Eric Lecolinet - Telecom ParisTech - 2016.
//  http://www.telecom-paristech.fr/~elc
//

#ifndef __tcpserver__
#define __tcpserver__
#include <memory>
#include <string>
#include <functional>
#include "ccsocket.h"

class TCPConnection;
class TCPLock;

class TCPServer {
public:

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

  TCPServer(Callback const& callback);

  virtual ~TCPServer();

  virtual int run(int port);

private:
  friend class TCPLock;
  friend class SocketCnx;

  TCPServer(TCPServer const&) = delete;
  TCPServer& operator=(TCPServer const&) = delete;
  void error(std::string const& msg);

  ServerSocket servsock_;
  Callback callback_{};
};

#endif
```

5.9 Video.h

```
#ifndef VIDEO_H
#define VIDEO_H

#include "Multimedia.h"

class Video : public Multimedia {
public:
    Video(const std::string& name, const std::string& path, int duration);
    void display(std::ostream& os) const override;
    void play() const override;
    void serialize(std::ofstream& ofs) const override;
    void deserialize(std::ifstream& ifs) override;
    std::string getClassName() const override;

private:
    int duration;
};

#endif // VIDEO_H
```