My Project

Generated by Doxygen 1.10.0

Chapter 1

INF224 - TP

The aim of this practical work is to create the software outline for a multimedia set-top box allowing you to play videos, films, display photos, etc. This software will be produced in stages, limiting itself to the declaration and implementation of a few typical classes and functionalities which will be completed gradually.

Chapter 2

INF224 - TP

Le but de cet exercice est de créer une interface graphique Java/Swing qui permettra à terme d'interagir avec le logiciel déjà créé lors du TP C++/Objet. Comme précédemment, ce programme Java sera réalisé par étapes en ajoutant les fonctionnalités nécessaires petit à petit.

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

Chapter 6

Class Documentation

6.1 Client Class Reference

Public Member Functions

- Client (String host, int port) throws UnknownHostException, IOException
- String send (String request)

Static Public Member Functions

- static void main (String argv[ ])

6.1.1 Constructor & Destructor Documentation

6.1.1.1 Client()

```
Client.Client (
             String host,
             int port ) throws UnknownHostException, IOException  [inline]
```

Initialise la connexion. Renvoie une exception en cas d'erreur.

6.1.2 Member Function Documentation

6.1.2.1 main()

```
static void Client.main (
             String argv[ ] )  [inline], [static]
```

Lit une requete depuis le Terminal, envoie cette requete au serveur, recupere sa reponse et l'affiche sur le Terminal. Noter que le programme bloque si le serveur ne repond pas.

6.1.2.2 send()

```
String Client.send (
            String request )  [inline]
```

Envoie une requete au server et retourne sa reponse. Noter que la methode bloque si le serveur ne repond pas.

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

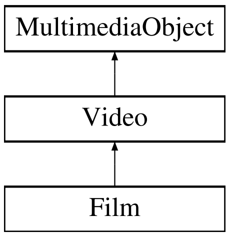
- swing/Client.java

6.2 Film Class Reference

A class that extends Video to manage films, including chapter information.

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

Inheritance diagram for Film:



Public Member Functions

- Film (const std::string &name, const std::string &filepath, int duration, int ∗chapters, int numChapters)

  Construct a new Film object.
- Film (const Film &other)

  Copy constructor for the Film class.
- Film & operator= (const Film &other)

  Overloaded assignment operator for the Film class.
- ∼**Film** ()

  Destructor for the Film class.
- void setChapters (int ∗chapters, int numChapters)

  Sets the chapters of the film.
- int ∗ getChapters () const

  Gets the chapters of the film.
- int getNumChapters () const

  Gets the number of chapters in the film.
- void **displayChapters** () const

  Displays information about the chapters of the film.
- virtual std::string display () const override

  Virtual method to display information about the film.

Public Member Functions inherited from Video

- Video (const std::string &name, const std::string &filepath, int duration)

  Construct a new Video object.
- Video (const Video &other)

  Copy constructor for the Video class.
- int getDuration () const

  Gets the duration of the video.
- void setDuration (int duration)

  Sets the duration of the video.
- virtual void play () const override

  Virtual method to "play" the video.

Public Member Functions inherited from MultimediaObject

- **MultimediaObject** ()

  Default constructor for MultimediaObject.
- MultimediaObject (const std::string &name, const std::string &filename)

  Constructs a MultimediaObject with a name and filename.
- MultimediaObject (const MultimediaObject &other)

  Copy constructor for the MultimediaObject.
- virtual ∼**MultimediaObject** ()

  Virtual destructor for the MultimediaObject.
- std::string getName () const

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

  Gets the filename of the multimedia object.
- void setName (const std::string &name)

  Sets the name of the multimedia object.
- void setFilename (const std::string &filename)

  Sets the filename of the multimedia object.

6.2.1 Detailed Description

A class that extends Video to manage films, including chapter information.

This class provides functionalities to handle film-specific attributes such as chapters, in addition to inheriting common video attributes from the Video class.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Film() [1/2]

```
Film::Film (
             const std::string & name,
             const std::string & filepath,
             int duration,
             int * chapters,
             int numChapters )
```

Construct a new Film object.

Parameters

name	The name of the film.
filepath	The file path to the film's video file.
duration	The total duration of the film.
chapters	An array containing the duration of each chapter.
numChapters	The number of chapters.

6.2.2.2 Film() [2/2]

```
Film::Film (
             const Film & other )
```

Copy constructor for the Film class.

Parameters

other	The Film object to be copied.

6.2.3 Member Function Documentation

6.2.3.1 display()

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

Virtual method to display information about the film.

Overrides the display method in the base Video class to include information about chapters.

Returns

std::string A string representing the film's information.

Reimplemented from Video.

6.2.3.2 getChapters()

```
int * Film::getChapters ( ) const
```

Gets the chapters of the film.

Returns

int∗ An array containing the duration of each chapter.

6.2.3.3 getNumChapters()

```
int Film::getNumChapters ( ) const
```

Gets the number of chapters in the film.

Returns

int The number of chapters.

6.2.3.4 operator=()

```
Film & Film::operator= (
             const Film & other )
```

Overloaded assignment operator for the Film class.

Parameters

other	The Film object to be assigned from.

Returns

Film& A reference to the assigned Film object.

6.2.3.5 setChapters()

```
void Film::setChapters (
             int * chapters,
             int numChapters )
```

Sets the chapters of the film.

Parameters

chapters	An array containing the duration of each chapter.
numChapters	The number of chapters.

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

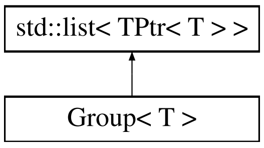
- cpp/Film.h
- cpp/Film.cpp

6.3 Group< T > Class Template Reference

Template class for managing groups of multimedia objects.

```
#include <Group.h>
```

Inheritance diagram for Group< T >:



Public Member Functions

- Group (const std::string &name)

  Construct a new Group object.
- std::string getName () const

  Gets the name of the group.
- std::string display () const

  Displays information about the group and its multimedia objects.

6.3.1 Detailed Description

template<typename T>
class Group< T >

Template class for managing groups of multimedia objects.

Inherits from std::list to manage collections of multimedia objects using shared pointers. Allows for the aggregation and management of any type that inherits from MultimediaObject.

Template Parameters

T	The type of multimedia object, must be derived from MultimediaObject or be MultimediaObject itself.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Group()

```
template<typename T >
Group< T >::Group (
            const std::string & name )  [inline]
```

Construct a new Group object.

Parameters

name	The name of the group.

6.3.3 Member Function Documentation

6.3.3.1 display()

```
template<typename T >
std::string Group< T >::display ( ) const  [inline]
```

Displays information about the group and its multimedia objects.

Iterates through the list of multimedia objects, invoking their display method and concatenating the result into a single string.

Returns

std::string A string containing information about the group and its objects.

6.3.3.2 getName()

```
template<typename T >
std::string Group< T >::getName ( ) const  [inline]
```

Gets the name of the group.

Returns

std::string The name of the group.

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

- cpp/Group.h

6.4 InputBuffer Struct Reference

Public Member Functions

- **InputBuffer** (size_t size)

Public Attributes

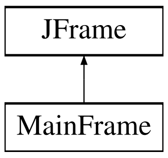
- char ∗ **buffer**
- char ∗ **begin**
- char ∗ **end**
- SOCKSIZE **remaining**

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

- cpp/ccsocket.cpp

6.5 MainFrame Class Reference

Inheritance diagram for MainFrame:



Public Member Functions

- MainFrame ()

Static Public Member Functions

- static void main (String[ ] args)

6.5.1 Detailed Description

MainFrame class that extends JFrame for creating the main application window. It includes a GUI for sending commands to a server and displaying responses.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 MainFrame()

```
MainFrame.MainFrame ( )  [inline]
```

Constructs the MainFrame and initializes UI components and actions.

6.5.3 Member Function Documentation

6.5.3.1 main()

```
static void MainFrame.main (
              String[ ] args )  [inline], [static]
```

Main method to run the application.

Parameters

args	Command-line arguments (not used).

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

- swing/MainFrame.java

6.6 Manager Class Reference

Manages multimedia objects and groups, offering creation, display, and deletion functionalities.

```
#include <Manager.h>
```

Public Member Functions

- std::shared_ptr< Photo > createPhoto (const std::string &name, const std::string &pathname, double latitude, double longitude)

  Creates and stores a new Photo object.
- std::shared_ptr< Video > createVideo (const std::string &name, const std::string &pathname, int duration)

  Creates and stores a new Video object.
- std::shared_ptr< Film > createFilm (const std::string &name, const std::string &pathname, int duration, const std::vector< int > &chapters)

  Creates and stores a new Film object.
- std::shared_ptr< Group< MultimediaObject > > createGroup (const std::string &name)

  Creates and stores a new Group object.
- std::string displayMultimediaObject (const std::string &name) const

  Displays information about a multimedia object.
- void displayGroup (const std::string &name) const

  Displays information about a group and its multimedia objects.
- void playMultimediaObject (const std::string &name) const

  Plays a multimedia object.
- void deleteMultimediaObject (const std::string &name)

  Deletes a multimedia object.
- void deleteGroup (const std::string &name)

  Deletes a group.

6.6.1 Detailed Description

Manages multimedia objects and groups, offering creation, display, and deletion functionalities.

This class uses maps to keep track of multimedia objects and groups of objects, allowing for efficient lookup, display, and management operations.

6.6.2 Member Function Documentation

6.6.2.1 createFilm()

```
std::shared_ptr< Film > Manager::createFilm (
             const std::string & name,
             const std::string & pathname,
             int duration,
             const std::vector< int > & chapters )
```

Creates and stores a new Film object.

Parameters

name	The name of the film.
pathname	The file path of the film.
duration	The total duration of the film in seconds.
chapters	A vector containing the duration of each chapter in the film.

Returns

std::shared_ptr<Film> A shared pointer to the created Film object.

6.6.2.2 createGroup()

```
std::shared_ptr< Group< MultimediaObject > > Manager::createGroup (
            const std::string & name )
```

Creates and stores a new Group object.

Parameters

name	The name of the group.

Returns

std::shared_ptr<Group<MultimediaObject>> A shared pointer to the created Group object.

6.6.2.3 createPhoto()

```
std::shared_ptr< Photo > Manager::createPhoto (
            const std::string & name,
            const std::string & pathname,
            double latitude,
            double longitude )
```

Creates and stores a new Photo object.

Parameters

name	The name of the photo.
pathname	The file path of the photo.
latitude	The latitude where the photo was taken.
longitude	The longitude where the photo was taken.

Returns

std::shared_ptr<Photo> A shared pointer to the created Photo object.

6.6.2.4 createVideo()

```
std::shared_ptr< Video > Manager::createVideo (
             const std::string & name,
             const std::string & pathname,
             int duration )
```

Creates and stores a new Video object.

Parameters

name	The name of the video.
pathname	The file path of the video.
duration	The duration of the video in seconds.

Returns

std::shared_ptr<Video> A shared pointer to the created Video object.

6.6.2.5 deleteGroup()

```
void Manager::deleteGroup (
             const std::string & name )
```

Deletes a group.

Parameters

name	The name of the group to delete.

6.6.2.6 deleteMultimediaObject()

```
void Manager::deleteMultimediaObject (
             const std::string & name )
```

Deletes a multimedia object.

Parameters

name	The name of the multimedia object to delete.

6.6.2.7 displayGroup()

```
void Manager::displayGroup (
             const std::string & name ) const
```

Displays information about a group and its multimedia objects.

Parameters

name	The name of the group to display.

6.6.2.8 displayMultimediaObject()

```
std::string Manager::displayMultimediaObject (
             const std::string & name ) const
```

Displays information about a multimedia object.

Parameters

name	The name of the multimedia object to display.

Returns

std::string A string containing the information about the multimedia object.

6.6.2.9 playMultimediaObject()

```
void Manager::playMultimediaObject (
             const std::string & name ) const
```

Plays a multimedia object.

Invokes the play method of the specified multimedia object.

Parameters

name	The name of the multimedia object to play.

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

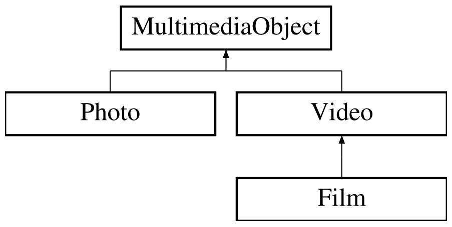
- cpp/Manager.h
- cpp/Manager.cpp

6.7 MultimediaObject Class Reference

Abstract base class for multimedia objects.

```
#include <MultimediaObject.h>
```

Inheritance diagram for MultimediaObject:



Public Member Functions

- **MultimediaObject** ()

    Default constructor for MultimediaObject.
- MultimediaObject (const std::string &name, const std::string &filename)

    Constructs a MultimediaObject with a name and filename.
- MultimediaObject (const MultimediaObject &other)

    Copy constructor for the MultimediaObject.
- virtual ∼**MultimediaObject** ()

    Virtual destructor for the MultimediaObject.
- std::string getName () const

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

    Gets the filename of the multimedia object.
- void setName (const std::string &name)

    Sets the name of the multimedia object.
- void setFilename (const std::string &filename)

    Sets the filename of the multimedia object.
- virtual void play () const

    Virtual method to play the multimedia object.
- virtual std::string display () const

    Virtual method to display information about the multimedia object.

6.7.1 Detailed Description

Abstract base class for multimedia objects.

This class serves as a base for different types of multimedia objects, providing common attributes like name and filename, and the interface for actions such as play and display.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 MultimediaObject() `[1/2]`

```
MultimediaObject::MultimediaObject (
                const std::string & name,
                const std::string & filename )
```

Constructs a MultimediaObject with a name and filename.

Parameters

name	The name of the multimedia object.
filename	The filename (including path) where the multimedia object is stored.

6.7.2.2 MultimediaObject() [2/2]

```
MultimediaObject::MultimediaObject (
             const MultimediaObject & other )
```

Copy constructor for the MultimediaObject.

Parameters

other	The MultimediaObject instance to copy from.

6.7.3 Member Function Documentation

6.7.3.1 display()

```
std::string MultimediaObject::display ( ) const  [virtual]
```

Virtual method to display information about the multimedia object.

This method should be overridden by derived classes to return a string containing information about the object.

Returns

std::string A string representing the multimedia object's information.

Reimplemented in Film, Photo, and Video.

6.7.3.2 getFilename()

```
std::string MultimediaObject::getFilename ( ) const
```

Gets the filename of the multimedia object.

Returns

std::string The filename where the object is stored.

6.7.3.3 getName()

```
std::string MultimediaObject::getName ( ) const
```

Gets the name of the multimedia object.

Returns

std::string The name of the object.

6.7.3.4 play()

```
void MultimediaObject::play ( ) const  [virtual]
```

Virtual method to play the multimedia object.

This method should be overridden by derived classes to perform the action of playing the multimedia content.

Reimplemented in Photo, and Video.

6.7.3.5 setFilename()

```
void MultimediaObject::setFilename (
             const std::string & filename )
```

Sets the filename of the multimedia object.

Parameters

filename	The new filename (including path) of the object.

6.7.3.6 setName()

```
void MultimediaObject::setName (
             const std::string & name )
```

Sets the name of the multimedia object.

Parameters

name	The new name of the object.

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

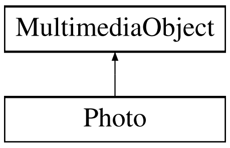
- cpp/MultimediaObject.h
- cpp/MultimediaObject.cpp

6.8 Photo Class Reference

A class for managing photo objects, extending the MultimediaObject class.

```
#include <Photo.h>
```

Inheritance diagram for Photo:



Public Member Functions

- Photo (const std::string &name, const std::string &filename, double latitude, double longitude)

  Construct a new Photo object.
- double getLatitude () const

  Gets the latitude of the photo.
- double getLongitude () const

  Gets the longitude of the photo.
- void setLatitude (double latitude)

  Sets the latitude of the photo.
- void setLongitude (double longitude)

  Sets the longitude of the photo.
- virtual std::string display () const override

  Virtual method to display information about the photo.
- virtual void play () const override

  Virtual method to "play" the photo.

Public Member Functions inherited from MultimediaObject

- **MultimediaObject** ()

  Default constructor for MultimediaObject.
- MultimediaObject (const std::string &name, const std::string &filename)

  Constructs a MultimediaObject with a name and filename.
- MultimediaObject (const MultimediaObject &other)

  Copy constructor for the MultimediaObject.
- virtual ∼**MultimediaObject** ()

  Virtual destructor for the MultimediaObject.
- std::string getName () const

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

  Gets the filename of the multimedia object.
- void setName (const std::string &name)

  Sets the name of the multimedia object.
- void setFilename (const std::string &filename)

  Sets the filename of the multimedia object.

6.8.1 Detailed Description

A class for managing photo objects, extending the MultimediaObject class.

This class represents a photo, including its geographic location (latitude and longitude) along with the basic multimedia attributes inherited from MultimediaObject.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Photo()

```
Photo::Photo (
             const std::string & name,
             const std::string & filename,
             double latitude,
             double longitude )
```

Construct a new Photo object.

Parameters

name	The name of the photo.
filename	The filename (including path) where the photo is stored.
latitude	The latitude where the photo was taken.
longitude	The longitude where the photo was taken.

6.8.3 Member Function Documentation

6.8.3.1 display()

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

Virtual method to display information about the photo.

Overrides the display method in the MultimediaObject class to include information about the photo's location.

Returns

std::string A string containing information about the photo.

Reimplemented from MultimediaObject.

6.8.3.2 getLatitude()

```
double Photo::getLatitude ( ) const
```

Gets the latitude of the photo.

Returns

double The latitude where the photo was taken.

6.8.3.3 getLongitude()

```
double Photo::getLongitude ( ) const
```

Gets the longitude of the photo.

Returns

double The longitude where the photo was taken.

6.8.3.4 play()

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

Virtual method to "play" the photo.

For a photo, "playing" typically means displaying the photo. This method overrides the play method in the MultimediaObject class.

Reimplemented from MultimediaObject.

6.8.3.5 setLatitude()

```
void Photo::setLatitude (
             double latitude )
```

Sets the latitude of the photo.

Parameters

latitude	The new latitude where the photo was taken.

6.8.3.6 setLongitude()

```
void Photo::setLongitude (
             double longitude )
```

Sets the longitude of the photo.

Parameters

longitude	The new longitude where the photo was taken.

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

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

6.9 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).

6.9.1 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.

6.9.2 Member Function Documentation

6.9.2.1 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.

6.9.2.2 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

6.10 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**

6.10.1 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.

6.10.2 Member Enumeration Documentation

6.10.2.1 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

6.10.3 Constructor & Destructor Documentation

6.10.3.1 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)

6.10.4 Member Function Documentation

6.10.4.1 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

6.10.4.2 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

6.10.4.3 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

6.10.4.4 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.

6.10.4.5 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.

6.10.4.6 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

6.11 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)

6.11.1 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;
}
```

6.11.2 Constructor & Destructor Documentation

6.11.2.1 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.

6.11.3 Member Function Documentation

6.11.3.1 read()

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

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

Returns

see readLine()

6.11.3.2 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().

6.11.3.3 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,

6.11.3.4 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,

6.11.3.5 write()

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

Writes *len* bytes to the socket.

Returns

see readLine()

6.11.3.6 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

6.12 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_**

6.12.1 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

6.13 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**

6.13.1 Detailed Description

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

6.13.2 Member Typedef Documentation

6.13.2.1 Callback

```
using TCPServer::Callback
```

Initial value:

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

6.13.3 Constructor & Destructor Documentation

6.13.3.1 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.

6.13.4 Member Function Documentation

6.13.4.1 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:

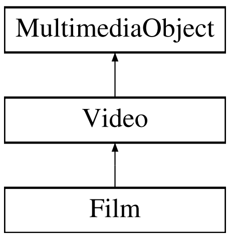
- cpp/tcpserver.h
- cpp/tcpserver.cpp

6.14 Video Class Reference

A class for managing video objects, extending the MultimediaObject class.

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

Inheritance diagram for Video:



Public Member Functions

- Video (const std::string &name, const std::string &filepath, int duration)

  Construct a new Video object.
- Video (const Video &other)

  Copy constructor for the Video class.
- int getDuration () const

  Gets the duration of the video.
- void setDuration (int duration)

  Sets the duration of the video.
- virtual std::string display () const override

  Virtual method to display information about the video.
- virtual void play () const override

  Virtual method to "play" the video.

Public Member Functions inherited from MultimediaObject

- **MultimediaObject** ()

  Default constructor for MultimediaObject.
- MultimediaObject (const std::string &name, const std::string &filename)

  Constructs a MultimediaObject with a name and filename.
- MultimediaObject (const MultimediaObject &other)

  Copy constructor for the MultimediaObject.
- virtual ∼**MultimediaObject** ()

  Virtual destructor for the MultimediaObject.
- std::string getName () const

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

  Gets the filename of the multimedia object.
- void setName (const std::string &name)

  Sets the name of the multimedia object.
- void setFilename (const std::string &filename)

  Sets the filename of the multimedia object.

6.14.1 Detailed Description

A class for managing video objects, extending the MultimediaObject class.

This class represents a video, including its duration along with the basic multimedia attributes inherited from MultimediaObject.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Video() [1/2]

```
Video::Video (
          const std::string & name,
          const std::string & filepath,
          int duration )
```

Construct a new Video object.

Parameters

name	The name of the video.
filepath	The file path where the video is stored.
duration	The duration of the video in seconds.

6.14.2.2 Video() [2/2]

```
Video::Video (
             const Video & other )
```

Copy constructor for the Video class.

Parameters

other	The Video object to copy from.

6.14.3 Member Function Documentation

6.14.3.1 display()

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

Virtual method to display information about the video.

Overrides the display method in the MultimediaObject class to include information about the video's duration.

Returns

std::string A string containing information about the video.

Reimplemented from MultimediaObject.

Reimplemented in Film.

6.14.3.2 getDuration()

```
int Video::getDuration ( ) const
```

Gets the duration of the video.

Returns

int The duration of the video in seconds.

6.14.3.3 play()

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

Virtual method to "play" the video.

This method overrides the play method in the MultimediaObject class, typically to perform actions such as opening a video player.

Reimplemented from MultimediaObject.

6.14.3.4 setDuration()

```
void Video::setDuration (
             int duration )
```

Sets the duration of the video.

Parameters

duration	The new duration of the video in seconds.

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

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

Chapter 7

File Documentation

7.1 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
```

7.2 Film.h

```
#ifndef FILM_H
#define FILM_H

#include "Video.h"

class Film : public Video {
private:
    int* chapters;
    int numChapters;

public:
    Film(const std::string& name, const std::string& filepath, int duration, int* chapters, int
numChapters);

    Film(const Film& other);

    Film& operator=(const Film& other);

    ~Film();

    void setChapters(int* chapters, int numChapters);

    int* getChapters() const;

    int getNumChapters() const;

    void displayChapters() const;

    virtual std::string display() const override;
};

#endif // FILM_H
```

7.3 Group.h

```
#ifndef GROUP_H
#define GROUP_H

#include <list>
#include <string>
#include "MultimediaObject.h"
#include <memory> // for std::shared_ptr
#include <sstream>

template <typename T>
using TPtr = std::shared_ptr<T>;

template <typename T>
class Group : public std::list<TPtr<T>> {
private:
    std::string name;

public:
    Group(const std::string& name) : name(name) {};

    std::string getName() const { return name; }

    std::string display() const {
        std::ostringstream oss;
        oss « "Group: " « name « std::endl;
        for (const TPtr<T>& object : *this) {
            oss « object->display() « std::endl;
        }
        return oss.str();
    }
};

#endif // GROUP_H
```

7.4 Manager.h

```
#ifndef MANAGER_H
#define MANAGER_H

#include <map>
#include <string>
#include <memory>
#include <vector>
#include "MultimediaObject.h"
#include "Photo.h"
#include "Video.h"
#include "Film.h"
#include "Group.h"

class Manager {
private:
    std::map<std::string, std::shared_ptr<MultimediaObject» multimediaObjects;
    std::map<std::string, std::shared_ptr<Group<MultimediaObject»> groups;

public:
    std::shared_ptr<Photo> createPhoto(const std::string& name, const std::string& pathname, double latitude, double longitude);

    std::shared_ptr<Video> createVideo(const std::string& name, const std::string& pathname, int duration);

    std::shared_ptr<Film> createFilm(const std::string& name, const std::string& pathname, int duration, const std::vector<int>& chapters);

    std::shared_ptr<Group<MultimediaObject» createGroup(const std::string& name);

    std::string displayMultimediaObject(const std::string& name) const;

    void displayGroup(const std::string& name) const;

    void playMultimediaObject(const std::string& name) const;

    void deleteMultimediaObject(const std::string& name);

    void deleteGroup(const std::string& name);
};

#endif // MANAGER_H
```

7.5 MultimediaObject.h

```
#ifndef MULTIMEDIAOBJECT_H
#define MULTIMEDIAOBJECT_H

#include <string>

class MultimediaObject {
public:
    MultimediaObject();

    MultimediaObject(const std::string& name, const std::string& filename);

    MultimediaObject(const MultimediaObject& other);

    virtual ~MultimediaObject();

    std::string getName() const;

    std::string getFilename() const;

    void setName(const std::string& name);

    void setFilename(const std::string& filename);

    virtual void play() const;

    virtual std::string display() const;

private:
    std::string name;
    std::string filename;
};

#endif // MULTIMEDIAOBJECT_H
```

7.6 Photo.h

```
#ifndef PHOTO_H
#define PHOTO_H

#include <string>
#include <iostream>
#include "MultimediaObject.h"

class Photo : public MultimediaObject {
private:
    double latitude;
    double longitude;

public:
    Photo(const std::string& name, const std::string& filename, double latitude, double longitude);

    double getLatitude() const;

    double getLongitude() const;

    void setLatitude(double latitude);

    void setLongitude(double longitude);

    virtual std::string display() const override;

    virtual void play() const override;
};

#endif /* PHOTO_H */
```

7.7 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
```

7.8 Video.h

```
#ifndef VIDEO_H
#define VIDEO_H

#include <iostream>
#include "MultimediaObject.h"

class Video : public MultimediaObject {
public:
    Video(const std::string& name, const std::string& filepath, int duration);

    Video(const Video& other);

    int getDuration() const;

    void setDuration(int duration);

    virtual std::string display() const override;

    virtual void play() const override;

private:
    int duration;
};

#endif /* VIDEO_H */
```

Index