

My Project

Generated by Doxygen 1.10.0

1 INF224 - TP	1
2 INF224 - TP	3
3 Hierarchical Index	5
3.1 Class Hierarchy	5
4 Class Index	7
4.1 Class List	7
5 File Index	9
5.1 File List	9
6 Class Documentation	11
6.1 Client Class Reference	11
6.1.1 Constructor & Destructor Documentation	11
6.1.1.1 Client()	11
6.1.2 Member Function Documentation	11
6.1.2.1 main()	11
6.1.2.2 send()	12
6.2 Film Class Reference	12
6.2.1 Detailed Description	13
6.2.2 Constructor & Destructor Documentation	13
6.2.2.1 Film() [1 / 2]	13
6.2.2.2 Film() [2 / 2]	14
6.2.3 Member Function Documentation	14
6.2.3.1 display()	14
6.2.3.2 getChapters()	14
6.2.3.3 getNumChapters()	15
6.2.3.4 operator=()	15
6.2.3.5 setChapters()	15
6.3 Group< T > Class Template Reference	15
6.3.1 Detailed Description	16
6.3.2 Constructor & Destructor Documentation	16
6.3.2.1 Group()	16
6.3.3 Member Function Documentation	17
6.3.3.1 display()	17
6.3.3.2 getName()	17
6.4 InputBuffer Struct Reference	17
6.5 MainFrame Class Reference	18
6.5.1 Detailed Description	18
6.5.2 Constructor & Destructor Documentation	18
6.5.2.1 MainFrame()	18
6.5.3 Member Function Documentation	18

6.5.3.1 main()	18
6.6 Manager Class Reference	19
6.6.1 Detailed Description	19
6.6.2 Member Function Documentation	19
6.6.2.1 createFilm()	19
6.6.2.2 createGroup()	20
6.6.2.3 createPhoto()	20
6.6.2.4 createVideo()	21
6.6.2.5 deleteGroup()	21
6.6.2.6 deleteMultimediaObject()	21
6.6.2.7 displayGroup()	21
6.6.2.8 displayMultimediaObject()	22
6.6.2.9 playMultimediaObject()	22
6.7 MultimediaObject Class Reference	22
6.7.1 Detailed Description	23
6.7.2 Constructor & Destructor Documentation	23
6.7.2.1 MultimediaObject() [1/2]	23
6.7.2.2 MultimediaObject() [2/2]	24
6.7.3 Member Function Documentation	24
6.7.3.1 display()	24
6.7.3.2 getFilename()	24
6.7.3.3 getName()	25
6.7.3.4 play()	25
6.7.3.5 setFilename()	25
6.7.3.6 setName()	25
6.8 Photo Class Reference	26
6.8.1 Detailed Description	27
6.8.2 Constructor & Destructor Documentation	27
6.8.2.1 Photo()	27
6.8.3 Member Function Documentation	27
6.8.3.1 display()	27
6.8.3.2 getLatitude()	27
6.8.3.3 getLongitude()	28
6.8.3.4 play()	28
6.8.3.5 setLatitude()	28
6.8.3.6 setLongitude()	28
6.9 ServerSocket Class Reference	29
6.9.1 Detailed Description	29
6.9.2 Member Function Documentation	29
6.9.2.1 accept()	29
6.9.2.2 bind()	30
6.10 Socket Class Reference	30

6.10.1 Detailed Description	31
6.10.2 Member Enumeration Documentation	32
6.10.2.1 Errors	32
6.10.3 Constructor & Destructor Documentation	32
6.10.3.1 Socket()	32
6.10.4 Member Function Documentation	32
6.10.4.1 bind() [1/2]	32
6.10.4.2 bind() [2/2]	32
6.10.4.3 connect()	33
6.10.4.4 receive()	33
6.10.4.5 send()	33
6.10.4.6 startup()	34
6.11 SocketBuffer Class Reference	34
6.11.1 Detailed Description	35
6.11.2 Constructor & Destructor Documentation	35
6.11.2.1 SocketBuffer()	35
6.11.3 Member Function Documentation	35
6.11.3.1 read()	35
6.11.3.2 readLine()	36
6.11.3.3 setReadSeparator()	36
6.11.3.4 setWriteSeparator()	36
6.11.3.5 write()	37
6.11.3.6 writeLine()	37
6.12 SocketCnx Class Reference	37
6.12.1 Detailed Description	38
6.13 TCPServer Class Reference	38
6.13.1 Detailed Description	38
6.13.2 Member Typedef Documentation	38
6.13.2.1 Callback	38
6.13.3 Constructor & Destructor Documentation	39
6.13.3.1 TCPServer()	39
6.13.4 Member Function Documentation	39
6.13.4.1 run()	39
6.14 Video Class Reference	39
6.14.1 Detailed Description	40
6.14.2 Constructor & Destructor Documentation	40
6.14.2.1 Video() [1/2]	40
6.14.2.2 Video() [2/2]	41
6.14.3 Member Function Documentation	41
6.14.3.1 display()	41
6.14.3.2 getDuration()	41
6.14.3.3 play()	42

6.14.3.4 setDuration()	42
7 File Documentation	43
7.1 ccsocket.h	43
7.2 Film.h	45
7.3 Group.h	46
7.4 Manager.h	46
7.5 MultimediaObject.h	47
7.6 Photo.h	47
7.7 tcpserver.h	48
7.8 Video.h	48

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:

Client	11
InputBuffer	17
JFrame	
MainFrame	18
std::list	
Group< T >	15
Manager	19
MultimediaObject	22
Photo	26
Video	39
Film	12
ServerSocket	29
Socket	30
SocketBuffer	34
SocketCnx	37
TCPServer	38

Chapter 4

Class Index

4.1 Class List

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

Client	11
Film	
A class that extends Video to manage films, including chapter information	12
Group< T >	
Template class for managing groups of multimedia objects	15
InputBuffer	17
MainFrame	18
Manager	
Manages multimedia objects and groups, offering creation, display, and deletion functionalities	19
MultimediaObject	
Abstract base class for multimedia objects	22
Photo	
A class for managing photo objects, extending the MultimediaObject class	26
ServerSocket	29
Socket	30
SocketBuffer	34
SocketCnx	
Connection with a given client. Each SocketCnx uses a different thread	37
TCPServer	38
Video	
A class for managing video objects, extending the MultimediaObject class	39

Chapter 5

File Index

5.1 File List

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

cpp/ ccsocket.h	43
cpp/ Film.h	45
cpp/ Group.h	46
cpp/ Manager.h	46
cpp/ MultimediaObject.h	47
cpp/ Photo.h	47
cpp/ tcpserver.h	48
cpp/ Video.h	48

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:

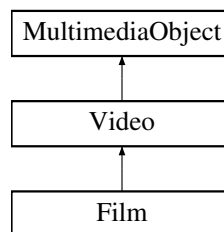
- swing/Client.java

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

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

6.2.2.2 Film() [2/2]

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

Copy constructor for the [Film](#) class.

Parameters

<i>other</i>	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

<i>other</i>	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

<i>chapters</i>	An array containing the duration of each chapter.
<i>numChapters</i>	The number of chapters.

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

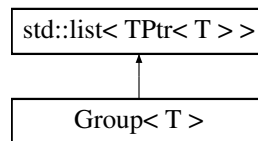
- [cpp/Film.h](#)
- [cpp/Film.cpp](#)

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

<i>T</i>	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

<i>name</i>	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

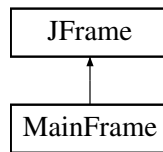
- char * **buffer**
- char * **begin**
- char * **end**
- SOCKSIZE **remaining**

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

<i>args</i>	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

<i>name</i>	The name of the film.
<i>pathname</i>	The file path of the film.
<i>duration</i>	The total duration of the film in seconds.
<i>chapters</i>	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

<i>name</i>	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

<i>name</i>	The name of the photo.
<i>pathname</i>	The file path of the photo.
<i>latitude</i>	The latitude where the photo was taken.
<i>longitude</i>	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

<i>name</i>	The name of the video.
<i>pathname</i>	The file path of the video.
<i>duration</i>	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

<i>name</i>	The name of the group to delete.
-------------	----------------------------------

6.6.2.6 deleteMultimediaObject()

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

Deletes a multimedia object.

Parameters

<i>name</i>	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

<i>name</i>	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

<i>name</i>	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

<i>name</i>	The name of the multimedia object to play.
-------------	--

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

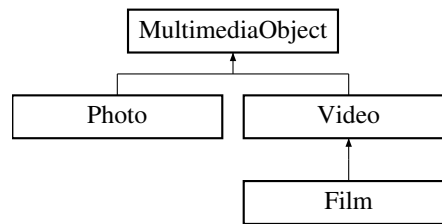
- cpp/Manager.h
- cpp/Manager.cpp

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

<i>name</i>	The name of the multimedia object.
<i>filename</i>	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

<i>other</i>	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

<i>filename</i>	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

<i>name</i>	The new name of the object.
-------------	-----------------------------

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

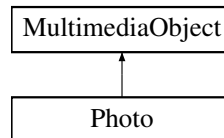
- `cpp/MultimediaObject.h`
- `cpp/MultimediaObject.cpp`

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

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

<i>latitude</i>	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

<i>longitude</i>	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 on 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 and stores them in *buf*. By default, this function blocks the caller until there is available 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 occurred.

6.10.4.5 send()

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

Sends data 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 occurred.

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 retrieve the entire record. Beware messages should thus not contain these characters.

```
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 occurred
- 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 occurrences 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:

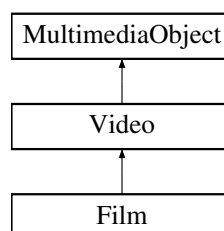
- `cpp/tcpserver.h`
- `cpp/tcpserver.cpp`

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

<i>name</i>	The name of the video.
<i>filepath</i>	The file path where the video is stored.
<i>duration</i>	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

<i>other</i>	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

<i>duration</i>	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

```
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/types.h>
00023 #include <sys/socket.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
00046 class Socket {
00047 public:
00052     enum Errors { Failed = -1, InvalidSocket = -2, UnknownHost = -3 };
00053
00057     static void startup();
00058     static void cleanup();
00060
00065     Socket(int type = SOCK_STREAM);
00066
00068     Socket(int type, SOCKET sockfd);
00069
00071     ~Socket();
00072
00077     int connect(const std::string& host, int port);
00078
00081     int bind(int port);
00082
00086     int bind(const std::string& host, int port);
00087
00089     int close();
```

```

00090
00092 bool isClosed() const { return sockfd_ == INVALID_SOCKET; }
00093
00095 SOCKET descriptor() { return sockfd_; }
00096
00098 void shutdownInput();
00099
00101 void shutdownOutput();
00102
00108 SOCKSIZE send(const SOCKDATA* buf, size_t len, int flags = 0) {
00109     return ::send(sockfd_, buf, len, NO_SIGPIPE_(flags));
00110 }
00111
00117 SOCKSIZE receive(SOCKDATA* buf, size_t len, int flags = 0) {
00118     return ::recv(sockfd_, buf, len, flags);
00119 }
00120
00121 #if !defined(_WIN32) && !defined(_WIN64)
00122
00124 SOCKSIZE sendTo(void const* buf, size_t len, int flags,
00125                 SOCKADDR const* to, socklen_t addrlen) {
00126     return ::sendto(sockfd_, buf, len, NO_SIGPIPE_(flags), to, addrlen);
00127 }
00128
00130 SOCKSIZE receiveFrom(void* buf, size_t len, int flags,
00131                      SOCKADDR* from, socklen_t* addrlen) {
00132     return ::recvfrom(sockfd_, buf, len, flags, from, addrlen);
00133 }
00134
00136 int setReceiveBufferSize(int size);
00137
00139 int setReuseAddress(bool);
00140
00142 int setSendBufferSize(int size);
00143
00145 int setSoLinger(bool, int linger);
00146
00148 int setSoTimeout(int timeout);
00149
00151 int setTcpNoDelay(bool);
00152
00154 int getReceiveBufferSize() const;
00155
00157 bool getReuseAddress() const;
00158
00160 int getSendBufferSize() const;
00161
00163 bool getSoLinger(int& linger) const;
00164
00166 int getSoTimeout() const;
00167
00169 bool getTcpNoDelay() const;
00170
00171 #endif
00172
00173 private:
00174     friend class ServerSocket;
00175
00176     // Initializes a local INET4 address, returns 0 on success, -1 otherwise.
00177     int setLocalAddress(SOCKADDR_IN& addr, int port);
00178     // Initializes a remote INET4 address, returns 0 on success, -1 otherwise.
00179     int setAddress(SOCKADDR_IN& addr, const std::string& host, int port);
00180
00181     SOCKET sockfd_{};
00182     Socket(const Socket&) = delete;
00183     Socket& operator=(const Socket&) = delete;
00184     Socket& operator=(Socket&&) = delete;
00185 };
00186
00187
00188
00192 class ServerSocket {
00193 public:
00195     ServerSocket();
00196
00197     ~ServerSocket();
00198
00202     Socket* accept();
00203
00206     int bind(int port, int backlog = 50);
00207
00209     int close();
00210
00212     bool isClosed() const { return sockfd_ == INVALID_SOCKET; }
00213
00215     SOCKET descriptor() { return sockfd_; }
00216

```

```

00217 #if !defined(_WIN32) && !defined(_WIN64)
00218
00220     int setReceiveBufferSize(int size);
00221
00223     int setReuseAddress(bool);
00224
00226     int setSoTimeout(int timeout);
00227
00229     int setTcpNoDelay(bool);
00230
00231 #endif
00232
00233 private:
00234     Socket* createSocket(SOCKET);
00235     SOCKET sockfd_{}; // listening socket.
00236     ServerSocket(const ServerSocket&) = delete;
00237     ServerSocket& operator=(const ServerSocket&) = delete;
00238     ServerSocket& operator=(ServerSocket&&) = delete;
00239 };
00240
00241
00276 class SocketBuffer {
00277 public:
00283     SocketBuffer(Socket*, size_t inputSize = 8192, size_t ouputSize = 8192);
00284     SocketBuffer(Socket&, size_t inputSize = 8192, size_t ouputSize = 8192);
00286
00287     ~SocketBuffer();
00288
00300     SOCKSIZE readLine(std::string& message);
00301
00309     SOCKSIZE writeLine(const std::string& message);
00310
00313     SOCKSIZE read(char* buffer, size_t len);
00314
00317     SOCKSIZE write(const char* str, size_t len);
00318
00320     Socket* socket() { return sock_; }
00321
00327     void setReadSeparator(int separ);
00328     int readSeparator() const { return insep_; }
00329     // @
00330
00336     void setWriteSeparator(int separ);
00337     int writeSeparator() const { return outsep_; }
00338     // @
00339
00340 private:
00341     SocketBuffer(const SocketBuffer&) = delete;
00342     SocketBuffer& operator=(const SocketBuffer&) = delete;
00343     SocketBuffer& operator=(SocketBuffer&&) = delete;
00344
00345 protected:
00346     bool retrieveLine(std::string& str, SOCKSIZE received);
00347     size_t insize_{}, outsize_{};
00348     int insep_{}, outsep_{};
00349     Socket* sock_{};
00350     struct InputBuffer* in_{};
00351 };
00352
00353 #endif

```

7.2 Film.h

```

00001 #ifndef FILM_H
00002 #define FILM_H
00003
00004 #include "Video.h"
00005
00013 class Film : public Video {
00014 private:
00015     int* chapters;
00016     int numChapters;
00017
00018 public:
00028     Film(const std::string& name, const std::string& filepath, int duration, int* chapters, int
numChapters);
00029
00035     Film(const Film& other);
00036
00043     Film& operator=(const Film& other);
00044
00048     ~Film();
00049

```

```

00056     void setChapters(int* chapters, int numChapters);
00057
00063     int* getChapters() const;
00064
00070     int getNumChapters() const;
00071
00075     void displayChapters() const;
00076
00084     virtual std::string display() const override;
00085 };
00086
00087 #endif // FILM_H

```

7.3 Group.h

```

00001 #ifndef GROUP_H
00002 #define GROUP_H
00003
00004 #include <list>
00005 #include <string>
00006 #include "MultimediaObject.h"
00007 #include <memory> // for std::shared_ptr
00008 #include <sstream>
00009
00013 template <typename T>
00014 using TPtr = std::shared_ptr<T>;
00015
00025 template <typename T>
00026 class Group : public std::list<TPtr<T> > {
00027 private:
00028     std::string name;
00029
00030 public:
00036     Group(const std::string& name) : name(name) {};
00037
00043     std::string getName() const { return name; }
00044
00053     std::string display() const {
00054         std::ostringstream oss;
00055         oss << "Group: " << name << std::endl;
00056         for (const TPtr<T>& object : *this) {
00057             oss << object->display() << std::endl;
00058         }
00059         return oss.str();
00060     }
00061 };
00062
00063 #endif // GROUP_H

```

7.4 Manager.h

```

00001 #ifndef MANAGER_H
00002 #define MANAGER_H
00003
00004 #include <map>
00005 #include <string>
00006 #include <memory>
00007 #include <vector>
00008 #include "MultimediaObject.h"
00009 #include "Photo.h"
00010 #include "Video.h"
00011 #include "Film.h"
00012 #include "Group.h"
00013
00021 class Manager {
00022 private:
00023     std::map<std::string, std::shared_ptr<MultimediaObject>> multimediaObjects;
00024     std::map<std::string, std::shared_ptr<Group<MultimediaObject>>> groups;
00025
00026 public:
00036     std::shared_ptr<Photo> createPhoto(const std::string& name, const std::string& pathname, double
latitude, double longitude);
00037
00046     std::shared_ptr<Video> createVideo(const std::string& name, const std::string& pathname, int
duration);
00047
00057     std::shared_ptr<Film> createFilm(const std::string& name, const std::string& pathname, int
duration, const std::vector<int>& chapters);
00058

```

```

00065     std::shared_ptr<Group<MultimediaObject> createGroup(const std::string& name);
00066
00073     std::string displayMultimediaObject(const std::string& name) const;
00074
00080     void displayGroup(const std::string& name) const;
00081
00089     void playMultimediaObject(const std::string& name) const;
00090
00096     void deleteMultimediaObject(const std::string& name);
00097
00103     void deleteGroup(const std::string& name);
00104 };
00105
00106 #endif // MANAGER_H

```

7.5 MultimediaObject.h

```

00001 #ifndef MULTIMEDIAOBJECT_H
00002 #define MULTIMEDIAOBJECT_H
00003
00004 #include <string>
00005
00012 class MultimediaObject {
00013 public:
00017     MultimediaObject();
00018
00025     MultimediaObject(const std::string& name, const std::string& filename);
00026
00032     MultimediaObject(const MultimediaObject& other);
00033
00037     virtual ~MultimediaObject();
00038
00044     std::string getName() const;
00045
00051     std::string getFilename() const;
00052
00058     void setName(const std::string& name);
00059
00065     void setFilename(const std::string& filename);
00066
00072     virtual void play() const;
00073
00081     virtual std::string display() const;
00082
00083 private:
00084     std::string name;
00085     std::string filename;
00086 };
00087
00088 #endif // MULTIMEDIAOBJECT_H

```

7.6 Photo.h

```

00001 #ifndef PHOTO_H
00002 #define PHOTO_H
00003
00004 #include <string>
00005 #include <iostream>
00006 #include "MultimediaObject.h"
00007
00015 class Photo : public MultimediaObject {
00016 private:
00017     double latitude;
00018     double longitude;
00019
00020 public:
00029     Photo(const std::string& name, const std::string& filename, double latitude, double longitude);
00030
00036     double getLatitude() const;
00037
00043     double getLongitude() const;
00044
00050     void setLatitude(double latitude);
00051
00057     void setLongitude(double longitude);
00058
00066     virtual std::string display() const override;
00067
00073     virtual void play() const override;

```

```

00074 };
00075
00076 #endif /* PHOTO_H */

```

7.7 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
00017 class TCPServer {
00018 public:
00019     using Callback =
00020         std::function< bool(std::string const& request, std::string& response) >;
00021
00022     TCPServer(Callback const& callback);
00023
00024     virtual ~TCPServer();
00025
00026     virtual int run(int port);
00027 private:
00028     friend class TCPLock;
00029     friend class SocketCnx;
00030
00031     TCPServer(TCPServer const&) = delete;
00032     TCPServer& operator=(TCPServer const&) = delete;
00033     void error(std::string const& msg);
00034
00035     ServerSocket servsock_;
00036     Callback callback_{};
00037 };
00038 #endif

```

7.8 Video.h

```

00001 #ifndef VIDEO_H
00002 #define VIDEO_H
00003
00004 #include <iostream>
00005 #include "MultimediaObject.h"
00006
00007 class Video : public MultimediaObject {
00008 public:
00009     Video(const std::string& name, const std::string& filepath, int duration);
00010
00011     Video(const Video& other);
00012
00013     int getDuration() const;
00014
00015     void setDuration(int duration);
00016
00017     virtual std::string display() const override;
00018
00019     virtual void play() const override;
00020 private:
00021     int duration;
00022 };
00023 #endif /* VIDEO_H */

```

Index

- accept
 - ServerSocket, 29
- bind
 - ServerSocket, 29
 - Socket, 32
- Callback
 - TCPServer, 38
- Client, 11
 - Client, 11
 - main, 11
 - send, 11
- connect
 - Socket, 32
- cpp/ccsocket.h, 43
- cpp/Film.h, 45
- cpp/Group.h, 46
- cpp/Manager.h, 46
- cpp/MultimediaObject.h, 47
- cpp/Photo.h, 47
- cpp/tcpserver.h, 48
- cpp/Video.h, 48
- createFilm
 - Manager, 19
- createGroup
 - Manager, 20
- createPhoto
 - Manager, 20
- createVideo
 - Manager, 20
- deleteGroup
 - Manager, 21
- deleteMultimediaObject
 - Manager, 21
- display
 - Film, 14
 - Group< T >, 17
 - MultimediaObject, 24
 - Photo, 27
 - Video, 41
- displayGroup
 - Manager, 21
- displayMultimediaObject
 - Manager, 22
- Errors
 - Socket, 32
- Film, 12
- display, 14
 - Film, 13, 14
 - getChapters, 14
 - getNumChapters, 14
 - operator=, 15
 - setChapters, 15
- getChapters
 - Film, 14
- getDuration
 - Video, 41
- getFilename
 - MultimediaObject, 24
- getLatitude
 - Photo, 27
- getLongitude
 - Photo, 27
- getName
 - Group< T >, 17
 - MultimediaObject, 24
- getNumChapters
 - Film, 14
- Group
 - Group< T >, 16
- Group< T >, 15
 - display, 17
 - getName, 17
 - Group, 16
- INF224 - TP, 1, 3
- InputBuffer, 17
- main
 - Client, 11
 - MainFrame, 18
- MainFrame, 18
 - main, 18
 - MainFrame, 18
- Manager, 19
 - createFilm, 19
 - createGroup, 20
 - createPhoto, 20
 - createVideo, 20
 - deleteGroup, 21
 - deleteMultimediaObject, 21
 - displayGroup, 21
 - displayMultimediaObject, 22
 - playMultimediaObject, 22
- MultimediaObject, 22
 - display, 24

- getFilename, 24
- getName, 24
- MultimediaObject, 23, 24
- play, 25
- setFilename, 25
- setName, 25
- operator=
 - Film, 15
- Photo, 26
 - display, 27
 - getLatitude, 27
 - getLongitude, 27
 - Photo, 27
 - play, 28
 - setLatitude, 28
 - setLongitude, 28
- play
 - MultimediaObject, 25
 - Photo, 28
 - Video, 41
- playMultimediaObject
 - Manager, 22
- read
 - SocketBuffer, 35
- readLine
 - SocketBuffer, 35
- receive
 - Socket, 33
- run
 - TCPServer, 39
- send
 - Client, 11
 - Socket, 33
- ServerSocket, 29
 - accept, 29
 - bind, 29
- setChapters
 - Film, 15
- setDuration
 - Video, 42
- setFilename
 - MultimediaObject, 25
- setLatitude
 - Photo, 28
- setLongitude
 - Photo, 28
- setName
 - MultimediaObject, 25
- setReadSeparator
 - SocketBuffer, 36
- setWriteSeparator
 - SocketBuffer, 36
- Socket, 30
 - bind, 32
 - connect, 32
- Errors, 32
 - receive, 33
 - send, 33
- Socket, 32
 - startup, 33
- SocketBuffer, 34
 - read, 35
 - readLine, 35
 - setReadSeparator, 36
 - setWriteSeparator, 36
 - SocketBuffer, 35
 - write, 36
 - writeLine, 37
- SocketCnx, 37
 - startup
 - Socket, 33
- TCPServer, 38
 - Callback, 38
 - run, 39
 - TCPServer, 39
- Video, 39
 - display, 41
 - getDuration, 41
 - play, 41
 - setDuration, 42
 - Video, 40, 41
- write
 - SocketBuffer, 36
- writeLine
 - SocketBuffer, 37