

# Projekt TIN - Dokumentacja

Wiktor Michalski  
Przemysław Stawczyk  
Maciej Szulik  
Kamil Zacharczuk

December 29, 2019

# Contents

<b>I Dokumentacja Koncepcji</b>	<b>4</b>
<b>1 Zadanie</b>	<b>5</b>
1.1 Treść Zadania	5
1.2 Wariant zadania dla zespołu	6
1.3 Interpretacja Zadania	6
<b>2 Opis Funkcjonalny Projektu</b>	<b>7</b>
<b>3 Protokół</b>	<b>8</b>
3.1 Składnia :	8
3.2 Komendy :	8
<b>4 Organizacja Projektu</b>	<b>9</b>
4.1 Moduły	9
4.2 Współbieżność	9
<b>5 Implementacja</b>	<b>11</b>
 <b>II Dokumentacja kodu źródłowego - <i>English</i></b>	 <b>12</b>
<b>6 Class Index</b>	<b>14</b>
6.1 Class List	14
<b>7 Class Documentation</b>	<b>15</b>
7.1 simpleP2P::Logging_Module Class Reference	15
7.1.1 Detailed Description	15
7.1.2 Constructor & Destructor Documentation	15
7.1.2.1 Logging_Module()	15
7.1.3 Member Function Documentation	16
7.1.3.1 add_log_line()	16
7.1.3.2 init()	16
7.2 SimpleP2P::RequestWorker Class Reference	16
7.2.1 Detailed Description	17
7.3 SimpleP2P::RequestServer Class Reference	17
7.3.1 Detailed Description	17
7.3.2 Constructor & Destructor Documentation	17
7.3.2.1 RequestServer()	18
7.4 SimpleP2P::RequestServerModule Class Reference	18
7.4.1 Detailed Description	18
7.4.2 Member Function Documentation	18

7.4.2.1	init()	18
7.5	simpleP2P::Udp_Client Class Reference	19
7.5.1	Detailed Description	19
7.5.2	Constructor & Destructor Documentation	19
7.5.2.1	Udp_Client()	19
7.5.3	Member Function Documentation	20
7.5.3.1	revoke_file()	20
7.6	simpleP2P::Udp_Server Class Reference	20
7.6.1	Detailed Description	20
7.6.2	Constructor & Destructor Documentation	20
7.6.2.1	Udp_Server()	21
7.7	simpleP2P::Udp_Module Class Reference	21
7.7.1	Detailed Description	21
7.7.2	Constructor & Destructor Documentation	21
7.7.2.1	Udp_Module()	22
7.7.3	Member Function Documentation	22
7.7.3.1	init()	22
7.7.3.2	revoke_file()	22
7.8	simpleP2P::Resource Class Reference	22
7.8.1	Detailed Description	23
7.8.2	Constructor & Destructor Documentation	23
7.8.2.1	Resource() [1/2]	24
7.8.2.2	Resource() [2/2]	24
7.8.3	Member Function Documentation	24
7.8.3.1	calc_segments_count()	24
7.8.3.2	generate_resource_header()	24
7.8.3.3	getName()	25
7.8.3.4	getPath()	25
7.8.3.5	getSize()	25
7.8.3.6	has_host()	25
7.8.3.7	operator!=(())	26
7.8.3.8	operator==(())	26
7.8.3.9	set_revoked()	26
7.9	simpleP2P::Host Class Reference	26
7.9.1	Detailed Description	27
7.9.2	Constructor & Destructor Documentation	27
7.9.2.1	Host()	27
7.9.3	Member Function Documentation	27
7.9.3.1	has_resource()	28
7.9.3.2	operator!=(())	28
7.9.3.3	operator==(())	28
7.10	simpleP2P::Resource_Database Class Reference	29
7.10.1	Detailed Description	29
7.10.2	Constructor & Destructor Documentation	29
7.10.2.1	Resource_Database()	29
7.10.3	Member Function Documentation	30
7.10.3.1	add_file() [1/2]	30
7.10.3.2	add_file() [2/2]	30
7.10.3.3	generate_database_header()	30
7.10.3.4	has_file()	31
7.10.3.5	revoke_resource()	31

7.10.3.6	<a href="#">update_host()</a>	31
7.10.3.7	<a href="#">who_has_file()</a> [1/2]	31
7.10.3.8	<a href="#">who_has_file()</a> [2/2]	32
7.11	<a href="#">SimpleP2P::FileManager Class Reference</a>	32
7.11.1	<a href="#">Detailed Description</a>	32
7.11.2	<a href="#">Member Function Documentation</a>	33
7.11.2.1	<a href="#">get_file()</a>	33
7.11.2.2	<a href="#">store_resource()</a>	33
7.12	<a href="#">SimpleP2P::FileRequest Class Reference</a>	33
7.12.1	<a href="#">Detailed Description</a>	34
7.12.2	<a href="#">Constructor &amp; Destructor Documentation</a>	34
7.12.2.1	<a href="#">FileRequest()</a>	34

## **Part I**

# **Dokumentacja Koncepcji**

# Chapter 1

## Zadanie

### 1.1 Treść Zadania

Napisać program obsługujący prosty protokół P2P

1. Zasób to plik identyfikowany pewną nazwą, za takie same zasoby uważa się zasoby o takich samych nazwach i takiej samej wielkości pliku w bajtach.
2. Początkowo dany zasób znajduje się w jednym węźle sieci, następnie może być propagowany do innych węzłów w ramach inicjowanego przez użytkownika ręcznie transferu - raz pobrany zasób zostaje zachowany jako kopia.
3. Po pewnym czasie działania systemu ten sam zasób może się znajdować w kilku węzłach sieci (na kilku maszynach).
4. System ma informować o posiadanych lokalnie (tj. w danym węźle) zasobach i umożliwiać ich pobranie.
5. Program powinien umożliwiać współbieżne:
  - wprowadzanie przez użytkownika (poprzez interfejs tekstowy):
    - nowych zasobów - z lokalnego systemu plików
    - poleceń pobrania nazwanego zasobu ze zdalnego węzła
  - pobieranie zasobów (także kilku jednocześnie)
  - rozgłaszanie informacji o posiadanych lokalnie zasobach
6. W przypadku pobierania zdalnego zasobu system sam (nie użytkownik) decyduje skąd zostanie on pobrany.
7. Powinno być możliwe pobranie zasobu z kilku węzłów na raz (tj. "w kawałkach").
8. Zasób pobrany do lokalnego węzła jest kopią oryginału, kopia jest traktowana tak samo jak oryginał (są nierozróżnialne). Istnienie kopii jest rozgłaszane tak samo jak oryginału.
9. Właściciel zasobu może go unieważnić wysyłając odpowiedni komunikat rozgłaszany. Wszystkie kopie zasobu powinny przestać być rozgłaszane. W przypadku trwających transferów zasobów powinny się one poprawnie zakończyć, dopiero wtedy informacja o zasobie może zostać usunięta.

## 1.2 Wariant zadania dla zespołu

4. Opóźnienia dla wybranego węzła - węzeł reaguje, ale (czasami) z dużym opóźnieniem.

## 1.3 Interpretacja Zadania

### Doprecyzowanie treści i dodatkowe założenia

- W związku z tym, że kopia i oryginał są nierozróżnialne, zasób może być unieważniony przez dowolnego użytkownika, który go posiada.
- Każdy węzeł okresowo rozgłasza informację o posiadanych zasobach. Unieważnienie pliku oznacza, że żaden z węzłów nie będzie już rozgłaszał faktu posiadania tego pliku.
- Unieważnienie wysyłane jest asynchronicznie poprzez broadcast UDP.
- W przypadku unieważnienia pliku w trakcie trwającego przesyłu tego pliku przesyłanie kończy się sukcesem, o ile nie wystąpią inne błędy. Nowy posiadacz pliku nie będzie jednak nigdy rozgłaszał o nim informacji.
- Każdy węzeł przechowuje listy dostępnych zasobów każdego innego węzła. Po odebraniu rozgłoszenia listy zasobów od innego węzła lista ta jest nadpisywana w pamięci węzła odbierającego. Informacje o węźle, w tym lista jego zasobów, są usuwane w przypadku braku, przez ustalony czas, nadchodzącego rozgłoszenia jego listy zasobów.
- W przypadku połączenia z innym węzłem w celu pobrania od niego pliku oczekiwanie na odpowiedź tego węzła ma pewien timeout. Ponadto, jeżeli węzeł przekracza pewien ustalony czas odpowiedzi (nawet jeżeli nie dochodzi do timeout'u), to inkrementujemy licznik "opóźnień" tego węzła (każdy węzeł przechowuje takie liczniki dla każdego innego węzła). Po osiągnięciu ustalonej wartości licznik ten jest zerowany, a węzeł zliczający nie będzie próbował łączyć się z "opóźnionym" węzłem przez pewien określony czas.
- Jeśli węzeł niespodziewanie zakończy połączenie TCP i przestanie rozgłaszać swoją tablicę, to po pewnym czasie pozostałe węzły uznają to za opuszczenie przez niego sieci.
- W przypadku gdy pojawi się błąd w trakcie transferu TCP, usuwamy pobrane dane (segmenty) i kończymy wątek pobierający. Ponowne pobieranie od tego węzła będzie odbywać się po ponownym połączeniu z węzłem.

## Chapter 2

# Opis Funkcjonalny Projektu

Użytkownik systemu ma wgląd w dwie listy

- lokalny rejestr zasobów - pliki, które użytkownik dodał lub pobrał od innych,
- pliki obecne w systemie - pliki posiadane w lokalnym rejestrze zasobów przez innych użytkowników, które nie zostały unieważnione.

*Dla każdego użytkownika generowana jest, oczywiście, odrębna para list.*

Użytkownik może wprowadzać tekstowe komendy, aby za ich pomocą

- wyświetlić listę lokalnych zasobów,
- wyświetlić listę zasobów obecnych w systemie,
- wyświetlić listę dostępnych komend,
- opuścić system,

a także wykonywać operacje na plikach, wśród których rozróżniamy:

- *dodanie pliku*,  
można dodać do zasobów plik, którego nazwa nie wystąpiła jeszcze wśród plików w lokalnym rejestrze zasobów i reszcie sieci.  
Zakładamy, że nie wystąpi sytuacja, gdy więcej niż jeden użytkownik doda plik o tej samej nazwie “jednocześnie” - to znaczy przed “zauważeniem” przez całą sieć dodania pliku o takiej nazwie przez któregośkolwiek z nich.
- *usunięcie pliku*,  
można usunąć plik z własnego rejestru zasobów.
- *unieważnienie pliku*,  
można unieważnić plik, który mamy we własnym rejestrze zasobów. Oznacza to, że zasób nie będzie już widoczny na liście plików dostępnych w systemie, ale dotychczasowi posiadacze nadal będą go posiadali w swoim lokalnym systemie plików.
- *pobranie pliku*,  
można pobrać plik, którego nie mamy jeszcze w rejestrze zasobów, a który jest obecny w systemie.



## Chapter 3

# Protokół

### 3.1 Składnia :

```
<Command 1 octet> ::= <REVOKE> | <FILE_LIST> | <REQ_CONN> | <REQ_SEGMENT>

<ResourceHeader> ::=
    <nazwa pliku: 256 octets, NULL terminated> <rozmiar pliku: 64bit>

<Message> ::= <Command> | <Command><ResourceHeader> |
    <Command><ResourceHeader><Options> |
    <Command><No_Of_Files 16bit><ResourceHeader><ResourceHeader>...

<Resource> ::= <1 KB of file>
```

### 3.2 Komendy :

- *Unieważnienie pliku:*  
Broadcast po UDP:  
<Command = REVOKE><ResourceHeader = Revoked File>
- *Rozgłaszanie dostępnych plików:*  
Broadcast po UDP:  
<Command = FILE\_LIST><No\_Of\_Files = liczba dostępnych plików>  
    <ResourceHeader = plik1>....
- *Żądanie utworzenia połączenia TCP:*  
Wysyłane do węzła po TCP: <Command = REQ\_CONN>
- *Żądanie pobrania segmentu:*  
Wysyłane do węzła po TCP:  
<Command = REQ\_SEGMENT><ResourceHeader = plik><Options = segment number>

## Chapter 4

# Organizacja Projektu

### 4.1 Moduły

1. Moduł CLI odpowiedzialny za komunikację z użytkownikiem.
2. Moduł obsługi sieci.
3. Moduł dispatchera obsługujący protokół.

### 4.2 Współbieżność

Ogólna koncepcja zakłada istnienie następujących bazowych, działających w pętli wątków:

1. *Obsługa przychodzących żądań przesłania posiadanego zasobu.*  
Wątek ten nasłuchuje na porcie TCP. W przypadku nawiązania połączenia na tym porcie tworzony jest wątek potomny. Wątek ten odbiera żądanie przesłania lokalnie posiadanego pliku i nadzoruje to przesłanie.
2. *Odbiór komunikatów broadcast UDP.*  
Komunikaty te obejmują okresowe rozgłaszanie przez każdego użytkownika listy lokalnie posiadanych zasobów, rozgłaszanie unieważnienia zasobu. Po odebraniu komunikatu wątek ten przekazuje otrzymane informacje do wątku synchronizującego dane.
3. *Okresowe rozgłaszanie lokalnej listy zasobów.*  
Wątek, który co pewien czas rozgłasza przez UDP listę zasobów, które udostępnia do pobrania. Wątek blokuje się pomiędzy kolejnymi broadcastami.
4. *Wątek synchronizujący dane.*  
Wątek manipulujący danymi przechowywanymi przez program takimi jak: lista lokalnych zasobów, informacje o pozostałych węzłach - dla każdego z nich lista zasobów, które udostępnia, liczniki opóźnień itp.
5. *Obsługa interfejsu użytkownika.*  
Interakcja z użytkownikiem przez CLI. Odbieranie komend od użytkownika i odpowiednie reagowanie - powoływanie nowych wątków, które mają zająć się realizacją komendy, między innymi:
  - W przypadku chęci pobrania pliku tworzony jest wątek nadzorujący to pobieranie. Na potrzeby połączenia z wieloma węzłami może on stworzyć kilka kolejnych wątków przypisanych do połączeń z węzłami.

W zależności od wyniku pobierania przekaże odpowiednie informacje do wątku synchronizującego dane.

- W przypadku chęci dodania, usunięcia lub unieważnienia pliku tworzony jest wątek, który zajmie się wprowadzeniem tej zmiany i nadzorowaniem wszystkich jej następstw, np.: fizycznie doda plik do systemu i przekaże informację o nowym pliku do wątku synchronizującego dane. W przypadku unieważnienia stworzy nowy wątek, który rozgłosi odpowiednią informację w systemie.
- W przypadku chęci wyświetlenia którejś z list zasobów lub listy dostępnych komend tworzony jest wątek, który odczyta odpowiednie dane i przygotuje je w odpowiedniej formie do wyświetlenia użytkownikowi.

**Wyniki działań powyższych wątków przekazywane są z powrotem do wątku obsługującego CLI, który wyświetla je użytkownikowi.**

## Chapter 5

# Implementacja

- Jezyki : *C++*
- Biblioteki : *Boost:Asio, std::thread*

## **Part II**

### **Dokumentacja kodu źródłowego - *English***

Description: TODO

Build: Project Uses CMake based config. 'create\_configs.sh' shell script will generate ninja files for compilation. Boost and thearding libaries are required

## Chapter 6

# Class Index

### 6.1 Class List

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

<a href="#">SimpleP2P::FileManager</a>	
Handles read/write to the files on disc . . . . .	32
<a href="#">SimpleP2P::FileRequest</a>	
Carries info about a single file transfer request - resource header and numbers of wanted segments . . . . .	33
<a href="#">simpleP2P::Host</a>	
Forward declaration . . . . .	26
<a href="#">simpleP2P::Logging_Module</a>	
Class Providing logging support based on text logs . . . . .	15
<a href="#">SimpleP2P::RequestServer</a>	
Asynchronous TCP server . . . . .	17
<a href="#">SimpleP2P::RequestServerModule</a>	
Module of the TCP server receiving file requests and sending the requested files' seg- ments . . . . .	18
<a href="#">SimpleP2P::RequestWorker</a>	
TCP connection handler, created by the TCP server . . . . .	16
<a href="#">simpleP2P::Resource</a>	
Forward declaration . . . . .	22
<a href="#">simpleP2P::Resource_Database</a>	
Class holding information about files in network and on localhost . . . . .	29
<a href="#">simpleP2P::Udp_Client</a>	
Class UDP Client to handle all outgoing packets . . . . .	19
<a href="#">simpleP2P::Udp_Module</a>	
Class containing all UDP related resources and logic . . . . .	21
<a href="#">simpleP2P::Udp_Server</a>	
Class UDP Server to handle all incoming packets . . . . .	20

## Chapter 7

# Class Documentation

### 7.1 simpleP2P::Logging\_Module Class Reference

class Providing logging support based on text logs

```
#include <logging_module.h>
```

#### Public Member Functions

- [Logging\\_Module](#) (std::ostream &output\_c=std::cerr)  
*Constructor for the logging thread.*
- std::thread [init](#) ()  
*Init methods run worker in thread and returns it.*
- void [add\\_log\\_line](#) (std::string line, const std::time\_t time)  
*Synchronised method for logging output.*

#### 7.1.1 Detailed Description

class Providing logging support based on text logs

#### 7.1.2 Constructor & Destructor Documentation

##### 7.1.2.1 Logging\_Module()

```
simpleP2P::Logging_Module::Logging_Module (  
    std::ostream & output_c = std::cerr )
```

Constructor for the logging thread.



#### Parameters

<i>output</i> ↔ _c	Output stream for the logs
-----------------------	----------------------------

#### Note

if output stream is a file you must explicitly close it

### 7.1.3 Member Function Documentation

#### 7.1.3.1 add\_log\_line()

```
void simpleP2P::Logging_Module::add_log_line (
    std::string line,
    const std::time_t time )
```

Synchronised method for logging output.

#### Parameters

<i>line</i>	
-------------	--

#### 7.1.3.2 init()

```
std::thread simpleP2P::Logging_Module::init ( )
```

Init methods run worker in thread and returns it.

#### Returns

logging thread

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

- include/logging\_module.h
- src/logging\_module.cpp

## 7.2 SimpleP2P::RequestWorker Class Reference

TCP connection handler, created by the TCP server.

```
#include <RequestWorker.h>
```

Inherits enable\_shared\_from\_this< RequestWorker >.

## Public Member Functions

- [RequestWorker](#) (boost::asio::io\_service &io\_service)  
*Constructor allows setting the socket on which the connection is established.*
- void [start](#) ()  
*Start handling the request.*
- tcp::socket & [socket](#) ()  
*Get socket.*

### 7.2.1 Detailed Description

TCP connection handler, created by the TCP server.

Receives the file request, buffers requested segments and sends them to the client.

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

- include/RequestWorker.h
- src/RequestWorker.cpp

## 7.3 SimpleP2P::RequestServer Class Reference

Asynchronous TCP server.

```
#include <RequestServer.h>
```

## Public Member Functions

- [RequestServer](#) (boost::asio::io\_service &io\_service, Uint16 port)  
*Constructor allows setting the parameters for the connnetion acceptor.*
- std::thread [init](#) ()  
*Turns on the listening and accepting connections and returns the thread in which it works.*

### 7.3.1 Detailed Description

Asynchronous TCP server.

It accepts connections asynchronously and for each of them creates a worker object to handle it.

### 7.3.2 Constructor & Destructor Documentation

### 7.3.2.1 RequestServer()

```
SimpleP2P::RequestServer::RequestServer (
    boost::asio::io_service & io_service,
    Uint16 port )
```

Constructor allows setting the parameters for the connnetion acceptor.

#### Parameters

<i>io_service</i>	boost::asio::io_service for the acceptor.
<i>port</i>	Port for the acceptor to listen on.

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

- include/RequestServer.h
- src/RequestServer.cpp

## 7.4 SimpleP2P::RequestServerModule Class Reference

Module of the TCP server receiving file requests and sending the requested files' segments.

```
#include <RequestServerModule.h>
```

### Public Member Functions

- [RequestServerModule](#) (Uint16 port\_)  
*Constructor, allows setting the port for the server.*
- std::thread [init](#) ()  
*Returns the thread object for the module.*

### 7.4.1 Detailed Description

Module of the TCP server receiving file requests and sending the requested files' segments.

### 7.4.2 Member Function Documentation

#### 7.4.2.1 init()

```
std::thread SimpleP2P::RequestServerModule::init ( )
```

Returns the thread object for the module.

Starts the server and returns the thread in which the server works. The thread of the server.

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

- include/RequestServerModule.h
- src/RequestServerModule.cpp

## 7.5 simpleP2P::Udp\_Client Class Reference

class UDP Client to handle all outgoing packets

```
#include <udp_client.h>
```

Inherits enable\_shared\_from\_this< Udp\_Client >.

### Public Member Functions

- [Udp\\_Client](#) (boost::asio::io\_service &io\_service, const boost::asio::ip::address &broadcast\_address, Uint16 broadcast\_port, Uint32 timeout=5 \*60)  
*Constructor of UDP Client.*
- [~Udp\\_Client](#) ()  
*Destructor closes socket.*
- void [revoke\\_file](#) ([Resource](#) resource)  
*Constructs revoke header sends it.*

### 7.5.1 Detailed Description

class UDP Client to handle all outgoing packets

### 7.5.2 Constructor & Destructor Documentation

#### 7.5.2.1 Udp\_Client()

```
simpleP2P::Udp_Client::Udp_Client (
    boost::asio::io_service & io_service,
    const boost::asio::ip::address & broadcast_address,
    Uint16 broadcast_port,
    Uint32 timeout = 5 * 60 )
```

Constructor of UDP Client.

#### Parameters

<i>io_service</i>	asio Io Service
<i>broadcast_address</i>	address on which packets will be sent
<i>broadcast_port</i>	port on which packets will be sent
<i>timeout</i>	beacon interval

## 7.5.3 Member Function Documentation

### 7.5.3.1 revoke\_file()

```
void simpleP2P::Udp_Client::revoke_file (
    simpleP2P::Resource resource )
```

Constructs revoke header sends it.

#### Parameters

<i>resource</i>	<a href="#">Resource</a> to be revoked
-----------------	--

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

- include/udp\_client.h
- src/udp\_client.cpp

## 7.6 simpleP2P::Udp\_Server Class Reference

class UDP Server to handle all incoming packets

```
#include <udp_server.h>
```

Inherits enable\_shared\_from\_this< Udp\_Server >.

### Public Member Functions

- [Udp\\_Server](#) (boost::asio::io\_service &io\_service, const boost::asio::ip::address &broadcast\_address, Uint16 broadcast\_port)  
*Constructor of UDP Server.*
- [~Udp\\_Server](#) ()  
*Destructor closes socket.*

### 7.6.1 Detailed Description

class UDP Server to handle all incoming packets

### 7.6.2 Constructor & Destructor Documentation

### 7.6.2.1 Udp\_Server()

```
simpleP2P::Udp_Server::Udp_Server (
    boost::asio::io_service & io_service,
    const boost::asio::ip::address & broadcast_address,
    Uint16 broadcast_port )
```

Constructor of UDP Server.

#### Parameters

<i>io_service</i>	asio Io Service
<i>broadcast_address</i>	address on which Server will listen
<i>broadcast_port</i>	port on which Server will listen

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

- include/udp\_server.h
- src/udp\_server.cpp

## 7.7 simpleP2P::Udp\_Module Class Reference

Class containing all UDP related resources and logic.

```
#include <udp_module.h>
```

### Public Member Functions

- [Udp\\_Module](#) (boost::asio::ip::address broadcast\_address, Uint16 port, Uint32 beacon\_interval)  
*Constructor.*
- std::thread [init](#) ()  
*Init methods run worker in thread and returns it.*
- void [revoke\\_file](#) (const [Resource](#) &resource)  
*Sends revoke datagram.*

### 7.7.1 Detailed Description

Class containing all UDP related resources and logic.

### 7.7.2 Constructor & Destructor Documentation

### 7.7.2.1 Udp\_Module()

```
simpleP2P::Udp_Module::Udp_Module (
    boost::asio::ip::address broadcast_address,
    Uint16 port,
    Uint32 beacon_interval )
```

Constructor.

#### Parameters

<i>broadcast_address</i>	address on which packets will be sent
<i>port</i>	port on which packets will be sent
<i>beacon_interval</i>	beacon interval

## 7.7.3 Member Function Documentation

### 7.7.3.1 init()

```
std::thread simpleP2P::Udp_Module::init ( )
```

Init methods run worker in thread and returns it.

#### Returns

logging thread

### 7.7.3.2 revoke\_file()

```
void simpleP2P::Udp_Module::revoke_file (
    const Resource & resource )
```

Sends revoke datagram.

#### Parameters

<i>resource</i>	Resource to be revoked
-----------------	------------------------

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

- include/udp\_module.h
- src/udp\_module.cpp

## 7.8 simpleP2P::Resource Class Reference

Forward declaration.

```
#include <resource.h>
```

## Public Member Functions

- [Resource](#) (std::string name, Uint64 size, std::string path="/")

*Constructor.*

- [Resource](#) (std::vector< Int8 > resource\_header)

*Constructor makes resource from header.*

- std::vector< Int8 > [generate\\_resource\\_header](#) ()

*Generates [Resource](#) header.*

- bool [has\\_host](#) ([Host](#) host)

*Determines if resource is possessed by [Host](#).*

- Uint16 [calc\\_segments\\_count](#) () const

*Calculates and returns segment count.*

- void [set\\_revoked](#) ()

*Function used to set invalidated flag.*

- Uint64 [getSize](#) () const

*Getter for file size.*

- const std::string & [getName](#) () const

*Getter for file name.*

- const std::string & [getPath](#) () const

*Getter for file path.*

- bool [operator==](#) (const [Resource](#) &other) const

*Operator == checks file size and name for equality.*

- bool [operator!=](#) (const [Resource](#) &other) const

*Operator != checks file size and name for equality.*

## Friends

- class [Resource\\_Database](#)

*friendship to manage [Resource](#) Hosts, path etc*

### 7.8.1 Detailed Description

Forward declaration.

Class contains file information and points to nodes with file possession

### 7.8.2 Constructor & Destructor Documentation



### 7.8.2.1 Resource() [1/2]

```
simpleP2P::Resource::Resource (
    std::string name,
    Uint64 size,
    std::string path = "." )
```

Constructor.

#### Parameters

<i>name</i>	filename
<i>size</i>	filesize
<i>path</i>	filepath, default is "."

### 7.8.2.2 Resource() [2/2]

```
simpleP2P::Resource::Resource (
    std::vector< Int8 > resource_header )
```

Constructor makes resource from header.

#### Parameters

<i>resource_header</i>	<a href="#">Resource</a> header
------------------------	---------------------------------

## 7.8.3 Member Function Documentation

### 7.8.3.1 calc\_segments\_count()

```
Uint16 simpleP2P::Resource::calc_segments_count ( ) const [inline]
```

Calculates and returns segment count.

#### Returns

segment count

### 7.8.3.2 generate\_resource\_header()

```
std::vector< Int8 > simpleP2P::Resource::generate_resource_header ( )
```

Generates [Resource](#) header.

## Returns

[Resource](#) header

### 7.8.3.3 getName()

```
const std::string & simpleP2P::Resource::getName ( ) const
```

Getter for file name.

## Returns

file name

### 7.8.3.4 getPath()

```
const std::string & simpleP2P::Resource::getPath ( ) const
```

Getter for file path.

## Returns

file path

### 7.8.3.5 getSize()

```
Uint64 simpleP2P::Resource::getSize ( ) const
```

Getter for file size.

## Returns

file size

### 7.8.3.6 has\_host()

```
bool simpleP2P::Resource::has_host (
    simpleP2P::Host host )
```

Determines if resource is possessed by [Host](#).

## Parameters

<i>host</i>	<a href="#">Host</a>
-------------	----------------------

## Returns

true if resource is possessed by host

### 7.8.3.7 operator!=(())

```
bool simpleP2P::Resource::operator!= (
    const Resource & other ) const
```

Operator != checks file size and name for equality.

#### Parameters

<i>other</i>	other
--------------	-------

#### Returns

true if not equal

### 7.8.3.8 operator==(())

```
bool simpleP2P::Resource::operator== (
    const Resource & other ) const
```

Operator == checks file size and name for equality.

#### Parameters

<i>other</i>	other
--------------	-------

#### Returns

true if equal

### 7.8.3.9 set\_revoked()

```
void simpleP2P::Resource::set_revoked ( ) [inline]
```

Function used to set invalidated flag.

To allow references on resource outside database to gather information about revoke

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

- include/resource.h
- src/resource.cpp

## 7.9 simpleP2P::Host Class Reference

Forward declaration.

```
#include <host.h>
```

## Public Member Functions

- [Host](#) (boost::asio::ip::address ip)  
*Constructor.*
- bool [has\\_resource](#) ([Resource](#) res)  
*Determines if host has resource.*
- bool [operator==](#) (const [Host](#) &other) const  
*Operator == checks host\_ip for equality.*
- bool [operator!=](#) (const [Host](#) &other) const  
*Operator != checks host\_ip for equality.*

## Friends

- class [Resource\\_Database](#)  
*friendship to manage [Host](#)'s Resources timeouts etc*

### 7.9.1 Detailed Description

Forward declaration.

Class contains node information and points to files it possess

### 7.9.2 Constructor & Destructor Documentation

#### 7.9.2.1 Host()

```
simpleP2P::Host::Host (  
    boost::asio::ip::address ip )
```

Constructor.

#### Parameters

<i>ip</i>	Ip of the <a href="#">Host</a>
-----------	--------------------------------

### 7.9.3 Member Function Documentation

### 7.9.3.1 has\_resource()

```
bool simpleP2P::Host::has_resource (
    Resource res )
```

Determines if host has resource.

#### Parameters

<i>res</i>	Resource to be checked
------------	------------------------

#### Returns

true if Host has Resource res

### 7.9.3.2 operator!=(())

```
bool simpleP2P::Host::operator!= (
    const Host & other ) const
```

Operator != checks host\_ip for equality.

#### Parameters

<i>other</i>	other
--------------	-------

#### Returns

true if not equal

### 7.9.3.3 operator==(())

```
bool simpleP2P::Host::operator== (
    const Host & other ) const
```

Operator == checks host\_ip for equality.

#### Parameters

<i>other</i>	other
--------------	-------

#### Returns

true if equal

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

- include/host.h
- src/host.cpp

## 7.10 simpleP2P::Resource\_Database Class Reference

Class holding information about files in network and on localhost.

```
#include <resource_database.h>
```

### Public Member Functions

- [Resource\\_Database](#) ([Host](#) localhost)  
*Constructor.*
- bool [has\\_file](#) (const [Resource](#) &res)  
*Check if localhost has certain file.*
- void [add\\_file](#) (const [Resource](#) &res, const [Host](#) &host)  
*Adds connection between file and resource, and creates them if they do not exist.*
- void [update\\_host](#) (const [Host](#) &host)  
*Updates the list of resources available from host Triggered after receive of full Beacon Packet.*
- void [revoke\\_resource](#) (const [Resource](#) &resource)  
*Revokes resource and disconnects it from Hosts in database and database itself [Resource](#) will still point to Hosts that possess it.*
- void [add\\_file](#) (const [Resource](#) &res)  
*same as add\_file(Resource, Host) but host is localhost*
- std::shared\_ptr< [Resource](#) > [who\\_has\\_file](#) (std::vector< Int8 > resource\_header)  
*Returns shared pointer to resource to allow access to information about file owners.*
- std::shared\_ptr< [Resource](#) > [who\\_has\\_file](#) (const [Resource](#) &res)  
*Returns shared pointer to resource to allow access to information about file owners.*
- std::vector< Int8 > [generate\\_database\\_header](#) ()  
*Generates listing of localhost content in a header.*

### 7.10.1 Detailed Description

Class holding information about files in network and on localhost.

### 7.10.2 Constructor & Destructor Documentation

#### 7.10.2.1 Resource\_Database()

```
simpleP2P::Resource_Database::Resource_Database (
    Host localhost )
```

Constructor.

#### Parameters

<i>localhost</i>	localhost
------------------	-----------

### 7.10.3 Member Function Documentation

#### 7.10.3.1 `add_file()` [1/2]

```
void simpleP2P::Resource_Database::add_file (
    const Resource & res,
    const Host & host )
```

Adds connection between file and resource, and creates them if they do not exist.

#### Parameters

<i>res</i>	Resource to be added
<i>host</i>	Host which possess Resource res

#### 7.10.3.2 `add_file()` [2/2]

```
void simpleP2P::Resource_Database::add_file (
    const Resource & res ) [inline]
```

same as `add_file(Resource, Host)` but host is localhost

#### Parameters

<i>res</i>	Resource to be added
------------	----------------------

#### 7.10.3.3 `generate_database_header()`

```
std::vector< Int8 > simpleP2P::Resource_Database::generate_database_header ( )
```

Generates listing of localhost content in a header.

#### Returns

listing header of localhost resources

#### 7.10.3.4 has\_file()

```
bool simpleP2P::Resource_Database::has_file (
    const Resource & res )
```

Check if localhost has certain file.

##### Parameters

<i>res</i>	Resource to be checked
------------	------------------------

##### Returns

true if host already has some resource

#### 7.10.3.5 revoke\_resource()

```
void simpleP2P::Resource_Database::revoke_resource (
    const Resource & resource )
```

Revokes resource and disconnects it from Hosts in database and database itself Resource will still point to Hosts that possess it.

##### Parameters

<i>resource</i>	Resource to be revoked
-----------------	------------------------

#### 7.10.3.6 update\_host()

```
void simpleP2P::Resource_Database::update_host (
    const Host & host )
```

Updates the list of resources available from host Triggered after receive of full Beacon Packet.

##### Parameters

<i>host</i>	Host and possessed resources in a struct
-------------	--

#### 7.10.3.7 who\_has\_file() [1/2]

```
shared_ptr< Resource > simpleP2P::Resource_Database::who_has_file (
    std::vector< Int8 > resource_header ) [inline]
```

Returns shared pointer to resource to allow access to information about file owners.



#### Parameters

<i>res</i>	<a href="#">Resource</a> about which information is gathered
------------	--

#### Returns

shared pointer to *res*

#### 7.10.3.8 `who_has_file()` [2/2]

```
shared_ptr< Resource > simpleP2P::Resource_Database::who_has_file (
    const Resource & res )
```

Returns shared pointer to resource to allow access to information about file owners.

#### Parameters

<i>res</i>	<a href="#">Resource</a> about which information is gathered
------------	--

#### Returns

shared pointer to *res*

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

- include/resource\_database.h
- src/resource\_database.cpp

## 7.11 SimpleP2P::FileManager Class Reference

Handles read/write to the files on disc.

```
#include <FileManager.h>
```

### Public Member Functions

- void [get\\_file](#) ([FileRequest](#) request, char \*result, std::size\_t size)  
*Buffers speciflicated segments of the speciflicated file in the char array.*
- void [store\\_resource](#) (CompleteResource &resource)  
*Stores the file contents in the physical file on disc.*

#### 7.11.1 Detailed Description

Handles read/write to the files on disc.

An API which provides:

- buffering contents of requested segments of a speciflicated local file,

- storing a complete, downloaded file physically on the local disc. Ensures synchronization of those operations.

## 7.11.2 Member Function Documentation

### 7.11.2.1 `get_file()`

```
void SimpleP2P::FileManager::get_file (
    FileRequest request,
    char * result,
    std::size_t size )
```

Buffers specicated segments of the specificated file in the char array.

(!) All segments will be returned concatenated in a single char array, provided in the 'result' parameter. They will be put to the array in the order as provided in the 'request' param. Keep this in mind if you requested the last segment of the file, size of which may vary.

#### Parameters

<i>request</i>	Specifies file and its segments to buffer.
<i>result</i>	The array to buffer the file contents in.
<i>size</i>	Size of the char array.

### 7.11.2.2 `store_resource()`

```
void SimpleP2P::FileManager::store_resource (
    CompleteResource & resource )
```

Stores the file contents in the physical file on disc.

#### Parameters

<i>resource</i>	File to store on the disc. The data will not be interpreted, so make sure it's complete and ready to store.
-----------------	---

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

- include/FileManager.h
- src/FileManager.cpp

## 7.12 SimpleP2P::FileRequest Class Reference

Carries info about a single file transfer request - resource header and numbers of wanted segments.

```
#include <FileRequest.h>
```

## Public Member Functions

- [FileRequest](#) (std::vector< Int8 > rh, std::initializer\_list< UInt32 > s)

*Constructor allows specifying the resource and segments.*

- std::vector< Int8 > [get\\_resource\\_header](#) () const

*Get the resource header.*

- std::vector< UInt32 > [get\\_segments](#) () const

*Get the segments' numbers.*

### 7.12.1 Detailed Description

Carries info about a single file transfer request - resource header and numbers of wanted segments.

An instance of this class is created by TCP client, sent to TCP server, which passes it to the [FileManager](#) in order to get the requested segments of the requested file and send them to the TCP client.

### 7.12.2 Constructor & Destructor Documentation

#### 7.12.2.1 FileRequest()

```
SimpleP2P::FileRequest::FileRequest (
    std::vector< Int8 > rh,
    std::initializer_list< UInt32 > s )
```

Constructor allows specifying the resource and segments.

The fields then can't be modified, only get.

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

- include/FileRequest.h
- src/FileRequest.cpp