# Chat server

This is a short description of the server and its work.

Logically the server is divided into three parts. The main part which is represented by the *Server*.*class* is an entry point of every client. The server is a multithread application and this one just catch all incoming connection and distributes (in fact wraps) them in other threads, instances of *ClientProcessing*. When the Server only handles with the connections the *ClientProcessing* operates with incoming requests which clients send using XML representation of *Message.* Every client logged or not has its own wrapping thread, which decodes what the client wants and choose appropriate *RequestHandler*.

XML-Message may be of the types enumerated in the *MessageStatus.* And each of the type has it’s own mandatory fields. In most cases they are obvious but in some cases the information that has to be passed in a *Message* is described in Javadoc of the method handling the type. If you are not sure what to pass feel free to seek additional information in the server.handlers package. For example, if you want to authorize, pass a message of *MessageStatus.AUTH* type and set the loggin and password fields. Find more in Appendix 1.

Interacting with the server you must remember the rule – once you have logged in – always pass your *clientId* in every message.

The response messages may be of 3 types in most cases

* ACCEPTED
* DENIED
* ERROR

The first one ensures that the preceding request has been operated successfully and the result is being provided in the *Message*. Usually it is the *text* field of the response. However the information may be specified in other fields depending on the context.

The second type tells us that the request has parsed, but the required actions have not been executed. Such response may be received when you pass wrong login or password.

The last response message type is almost the same as the last – it tells that that something went wrong during the operating. The problem may be both as of internal (Error/Exception, bug) and external cause (not all mandatory fields are set).

Such division of the unsatisfactory responses has been mainly made for client and is for informational purposes only.

The place where clients chat is represented by *Room.class*. All the information of rooms is saved in the corresponding folders that by default are created in the ../rooms directory.

The clients are saved the same as the rooms - ../clients.

Configuring the server

The server may be started with the settings different from default. The file of the server properties may be created by the command *-cds* . It will create the default root folder structure of a server in the place you specify. You may read more about the *-cds* and other commands below.

* -cds path/to/server/root/folder - creates a default server root structure in the specified folder
* -start path/to/serverConfig.xml - starts the server denoted by the configurations
* -restart path/to/serverConfig.xml – restarts the server denoted by the configurations
* -stop path/to/serverConfig.xml - stops the server denoted by the configurations
* -ban path/to/serverConfig.xml <login> <hours> - bans the client on the server denoted by the configurations
* -unban path/to/serverConfig.xml <login> - unbans the client on the server denoted by the configurations
* -exit - to exit the program

# Appendix 1 (Examples of message sending)

**REGISTRATION**

message = new Message(MessageStatus.REGISTRATION)

.setLogin("test\_user")

.setPassword("12345");

marshaller.marshal(message, stringWriter);

out.writeUTF(stringWriter.toString());

**ROOM CREATION**

message = new Message(MessageStatus.CREATE\_ROOM)

.setFromId("test\_user".hashCode());

stringWriter = new StringWriter();

marshaller.marshal(message, stringWriter);

out.writeUTF(stringWriter.toString());

**USER INVITATION**

message = new Message(MessageStatus.INVITE\_CLIENT)

.setFromId("test\_user".hashCode())

.setToId("test\_user\_2".hashCode())

.setRoomId(832175715);

stringWriter = new StringWriter();

marshaller.marshal(message, stringWriter);

out.writeUTF(stringWriter.toString());

**USER UNINVITATION**

message = new Message(MessageStatus.UNINVITE\_CLIENT)

.setFromId("test\_user".hashCode())

.setToId("test\_user\_2".hashCode())

.setRoomId(832175715);

stringWriter = new StringWriter();

marshaller.marshal(message, stringWriter);

out.writeUTF(stringWriter.toString());

**STOP SERVER**

message = new Message(MessageStatus.STOP\_SERVER)

.setFromId("test\_user".hashCode());

stringWriter = new StringWriter();

marshaller.marshal(message, stringWriter);

out.writeUTF(stringWriter.toString());

**ROOM LIST REQUESTING**

message = new Message(MessageStatus.ROOM\_LIST)

.setFromId("test\_user".hashCode());

stringWriter = new StringWriter();

marshaller.marshal(message, stringWriter);

out.writeUTF(stringWriter.toString());

**CLIENT BAN**

message = new Message(MessageStatus.CLIENT\_BAN).setFromId("test\_user".hashCode())

.setToId("test\_user\_2".hashCode())

.setText(ServerProcessing.DATE\_TIME\_FORMATTER.format(LoclDateTime.now().plusHours(1)));

stringWriter = new StringWriter();

marshaller.marshal(message, stringWriter);

out.writeUTF(stringWriter.toString());

**CLIENT UNBAN**

message = new Message(MessageStatus.CLIENT\_UNBAN).setFromId("test\_user".hashCode())

.setToId("test\_user\_2".hashCode());

stringWriter = new StringWriter();

marshaller.marshal(message, stringWriter);

out.writeUTF(stringWriter.toString());

**RESTART**

message = new Message(MessageStatus.RESTART\_SERVER)

.setFromId("test\_user".hashCode());

stringWriter = new StringWriter();

marshaller.marshal(message, stringWriter);

out.writeUTF(stringWriter.toString());

**ROOM MEMBERS**

message = new Message(MessageStatus.ROOM\_MEMBERS)

.setFromId("test\_user".hashCode())

.setRoomId(0);

stringWriter = new StringWriter();

marshaller.marshal(message, stringWriter);

out.writeUTF(stringWriter.toString());

**MESSAGE HISTORY**

message = new Message(MessageStatus.MESSAGE\_HISTORY)

.setFromId("Puser".hashCode())

.setRoomId(0);

stringWriter = new StringWriter();

marshaller.marshal(message, stringWriter);

out.writeUTF(stringWriter.toString());

**CLIENT NAME**

message = new Message(MessageStatus.GET\_CLIENT\_NAME)

.setFromId("Puser".hashCode())

.setToId(-1898799876);

stringWriter = new StringWriter();

marshaller.marshal(message, stringWriter);

out.writeUTF(stringWriter.toString());