**Eryantis Protocol Documentation**

Ferrario, Griffanti, Masiero  
Gruppo 25

# **Messages**

We decided to use JSON files to send and receive messages between client and server.

# **LoginMessage**

This message by the client after his connection to the server.

### **Arguments**

1. Nickname: the username chosen by the player
2. createNewMatch: TRUE if a new match must be created  
    FALSE if the player can join an existing match

### **Possible Responses**

* NicknameNotValid: the nickname provided has already been chosen by another player.
* AckMatchCreationMessage: if “NewMatchNeeded” is TRUE, this is the ack message for the provided login data, and asks for new information for the lobby creation.
* AskMatchToJoinMessage: if “NewMatchNeeded” is FALSE, the client is asked which lobby he wants to join.
* NoLobbyAvailable: if “NewMatchNeeded” is FALSE but there aren’t no lobby available to join.

# **NicknameNotValid**

This message is sent by the server to the client when the Player session can’t be created because the nickname is already used.

### **Arguments**

This message has no arguments.

### **Possible responses**

* LoginMessage: a new login message needs to be sent.

## **AckMatchCreationMessage**

This message is sent from the server to the client when a player session is created. The username sent before is accepted.

### **Arguments**

playerID: unique ID inside the Match, provided by the server.

NewMatchNeeded: TRUE if a new match must be created;

FALSE if the player can join an existing match.

### **Possible responses**

* NewMatchSpecs: the client provides the details of the new match.

## **AskMatchToJoinMessage**

This message is sent from the server to the client when a player asks to join a match and there is at least one available. (The nickname sent before is accepted).

### **Arguments**

1. availableLobbiesTmp: ArrayList of boolean of available lobbies in the server;
2. lobbiesNumberOfPlayers: ArrayList of Integer of players numbers of every lobby;
3. lobbiesExpertMode: ArrayList of boolean of expert mode (or not) of the

lobbies in the server;

1. lobbiesEnd: ArrayList of boolean of ended match (or not) in the server;

### **Possible responses**

* ReplyChosenLobby: the client has to choose one of the available lobbies.

## **NoLobbyAvailable**

This message is sent from the server to the client when a player asks to join a match, but there is no one available (because full or no one created), and so is asked the client to provide the specs for a new match (The nickname sent before is accepted).

### **Arguments**

This message has no arguments.

### **Possible responses**

* NewMatchSpecs: the client provides the details of the new match.

## **MatchSpecsMessage**

This message is sent by the client to provide the number of players for his new match and the expert game mode.

### **Arguments**

1. numOfPlayers: integer which stands for the number of players the client wants in his new

lobby;

2. expertMode: boolean for the expert game mode (TRUE) or not (FALSE).

### **Possible responses**

* AckMessage: ack by server with subobject “waiting”.

## **ReplyChosenLobby**

This message is sent by the client to the server to ask to join a chosen lobby.

### **Arguments**

1. lobbyIDChosen: ID of the chosen lobby.

### **Possible responses**

* Nack [subObj = “lobby\_not\_available”] : sent if the lobby chosen by the player is no more available
* IDSetAfterLobbyChoiceMessage : sent to let the player know his ID in the match
* AckMessage [subObj: “waiting”] : sent when the match has to wait for other players to join before starting
* MatchStartMessage: sent to notify the player that the match has begun

## **IDSetAfterLobbyChoiceMessage**

This message is sent by the server to the client to notify it about its player’s ID

### **Arguments**

1. playerID : ID used to identify the player in the match

### **Possible responses**

* This message has no response.

## **MatchStartMessage**

This message is sent by the server to the client when the match has started.

### **Arguments**

1. firstPlayer : ID of the first player of the match
2. numPlayer : number of players playing in the match
3. nicknames: list of the players’ nickname
4. expertMode : a boolean value that specifies if the match is being played in expert mode or not
5. motherNaturePosition : ID of the island where mother nature was put during the initial setup of the islands
6. studentsOnIsland : list of students put on the islands during the initial setup
7. studentsInEntrance : students in the entrance of each player
8. characters : list of characters randomly chosen for a match played in expert mode
9. monkStudents : list of students on the character card “monk”
10. jesterStudents : list of students on the character card “jester”
11. princessStudents : list of students on the character card “princess”

### **Possible responses**

* This message has no response.

## **ChosenTowerColorMessage**

This message is sent from the client to the server to provide the tower color the player has chosen.

### **Arguments**

1. color: it is the tower color chosen by the player.

### **Possible responses**

### AckMessage (subObject: “tower\_color”).

## **ChosenDeckMessage**

This message is sent from the client to the server to provide the tower color the player has chosen.

### **Arguments**

1. deck: it is the deck chosen by the player.

### **Possible responses**

### AckMessage (subObject: “deck”)

## **BagClickMessage**

This message is sent from the client to the server to notify that the students have been drawn from the bag

### **Arguments**

No arguments needed for this message.

### **Possible responses**

### AckMessage (SubObject: “fillClouds”).

## **ChosenAssistantCardMessage**

This message is sent from the client to the server to provide the assistant card the player has chosen.

### **Arguments**

1. assistantChosen: it is the assistant chosen by the player.

### **Possible responses**

### AckMessage (SubObject: “assistant”).

## **MovedStudentsFromEntrance**

This message is sent from the client to the server to notify which student has been moved from the entrance and where, to the diningroom or on an island.

### **Arguments**

1. student\_ID: it represents the ID used to identify the student, that has just been moved, inside the entrance array.
2. location: it represents the ID of the island where the student has been moved or the value -1 meaning that the student has been put into the dining room.

### **Possible responses**

* AckMessage (subObject: action\_1\_dining\_room).
* AckMessage (subObject: action\_1\_island).

## **MovedMotherNatureMessage**

This message is sent from the client to the server to provide the island ID where mother nature has been moved during action2.

### **Arguments**

1. destinationIsland\_ID: this attribute is the ID of the island where the player chose to move mother nature.

### **Possible responses**

* AckMessage (subObject: “action\_2\_movement”).
* AckMessage(subObject: “action\_2\_influence”).
* AckMessage(subObject: “action\_2\_union”).

## **ChosenCloudMessage**

This message is sent from the client to the server to provide the Cloud ID chosen by the player, whose students will be added to the entrance.

### **Arguments**

1. cloud\_ID: this attribute is the ID of the cloud chosen by the player, whose students will be added to the entrance.

### **Possible responses**

* AckMessage (subObject: “action\_3”).

## **CharacterRequestMessage**

This message is send by the client to ask the server to use one of the character that are available in the match.

### **Arguments**

1. character: This attribute is a string that contains the name of the character chosen by the player.

### **Possible responses**

* AckMessage: an ack which has the name of the character card as a subobject.
* NackMessage: a nack which tells the player that the chosen character can’t be used, and the reason why.

## **CharacterDataMessage**

This message is sent by the client when he has to provide some choice about the chosen character.

### **Arguments**

1. character: is a string which contains the name of the character chosen.
2. student\_ID: This attribute is the ID of the student chosen by the player (used for various cards).
3. island\_ID: This attribute is the ID of the island chosen by the player (used for various cards).
4. elementsFromCard: This attribute is the list of IDs of the elements (students) on the character card selected by the player.
5. studentsFromPlayerEntrance: This attribute is the list of IDs of the students selected by the player from his entrance.
6. studentsFromPlayerDiningRoom: This attribute is the list of types of student selected by the player from his dining room.
7. creature: This attribute is the type of student chosen by the player (ex. character MushroomsMerchant)

### **Possible responses**

* AckCharactersMessage: it is the confirmation that the character and the data provided have been used correctly.

## **AckMessage**

This message is sent by the server to the client to notify the message was received and to also provide data about the changes occured to the model.

### **Arguments**

1. subObject : string that specifies which data are stored in the message
2. recipient : ID of the Client to whom this ack message is addressed
3. nextPlayer : ID of the next player to make his move
4. students : a list of students (e.g. list of students on the clouds)
5. notAvailableTowerColors : list of already chosen tower colors
6. notAvailableDeck: list of already chosen decks
7. assistantAlreadyUsedInThisRound: list of already chosen assistant cards in the round.
8. studentMoved\_ID: this attribute is the ID of the moved student.
9. typeOfStudentMoved: this attribute is the type of the student that has just been moved.
10. professorTaken : boolean values which specifies if the player took control on the professor after moving a student into the dining room.
11. previousOwnerOfProfessor: This attribute is the ID of the previous player who had the control over the professor taken by the current player (recipient of this message).
12. destinationIsland\_ID: This attribute is the ID of the island where the student has been moved, in case the player chose to move the student on an island; or it can also be the ID of the island where mother nature was moved during action\_2.
13. islandsUnified: This string attribute tells if some islands have been unified

- none: current island unified with no other island;

- previous: current island unified with the previous one;

- next: current island unified with the next one;

- both: current island unified with both the next and the previous.

1. islands\_ID: This attribute is a list of island\_IDs used to show which islands have been unified to the current one (where MN is).
2. removedNoEntryTile: This attribute tells if a noEntryTile was removed from an island (specified by the attribute).
3. islandThatLostNoEntryTile: This attribute tells from which island the noEntryTile was taken.
4. masterChanged: This attribute tells if the master on the island reached by mother nature changed.
5. previousMaster\_ID: This attribute tells, if the master changed, who was the previous one; the current master could be anyone of the players not necessarily the one who moved mother nature or could be no one if there wasn't any tower yet.
6. newMaster\_ID: This attribute tells which player is the new master of the island where mother nature arrived, only if the master changed.
7. action3Valid: This attribute is true if there were enough students for refilling the clouds false if there weren't enough students for refilling the clouds (action\_3 not played yet).
8. cloudChosen\_ID: This attribute is the ID of the cloud chosen by the player to refill his entrance.
9. endOfMatch: This attribute is true if the match has come to its end or false otherwise.
10. nextPlanningPhase: This attribute is true if after action\_3 there's a new round to play, false if there are other players that must play their action phase yet.
11. towerColor: This attribute is the color of the tower of a player or on an island.

### **Possible responses**

* This message has no response.

## **NackMessage**

The nack message is sent from the server to the client to notify that the choice or data provided by the previous message by the client, could not be accepted.

### **Arguments**

1. subObject: This attribute is the second object of the message, and it tells which data are stored in this message.
2. explanationMessage: This attribute contains the reason why the client is receiving this nack message. It is a string message that can be directly printed by the client as an explanation of the error generated by the player's move.

### **Possible responses**

* This message has no response.

## **AckCharactersMessage**

After choosing to use a character and after providing the information needed (if needed), the server sends this ack message to confirm the client that he has rightly used the chosen character and the data have been correctly accepted.

### **Arguments**

1. character: This attribute is the name of the character card used.
2. coinReserve: This attribute is the amount of coins currently kept in the general reserve (at the center of the table).
3. playerCoins: This attribute is the number of coins owned by the player that used the character card.
4. student: This attribute is a single student taken/put because of the effect of some character cards.
5. island\_ID: This attribute is the island chosen for the effect of the character card (if necessary).
6. studentsOnCard: This attribute is the list of students on the card (if necessary).
7. numberOfElementsOnTheCard: This attribute is the number of elements (e.g. students / no-entry-tiles) on a character.
8. entranceOfPlayer: This attribute is the list of students belonging to the player (for example in the entrance of his school-board)
9. creature: This attribute is the kind of student chosen by the player or used during the effect of a character.
10. playerDiningRoom: This hashmap is the dining room of the player, for each type of students the number of students on the table; the key is type of students and the value is the number of student of the table.
11. allPlayersProfessors: This attribute is the correspondence between player\_ID and its professors as list of creature; the key is the player's ID, the value is an array of Creature corresponding to the professors controlled by the player.
12. allPlayersDiningRoom: This attribute represents the dining room of each player; the key is the ID of the player, the value is the hashMap representing the dining room of the player.

### **Possible responses**

* This message has no response.

## **EndOfMatchMessage**

This message is sent from the server to the client to notify when the match of his lobby ends.

### **Arguments**

1. winner: ID of the winner of the match or -1 if the match ended with a tie
2. winnerNickname: nickname of the winner
3. reason: string specifying the reason why the match ended

### **Possible responses**

* This message has no response.

## **PingMessage**

This message is sent from the client to the server to notify that he is still connected. This message is sent every 5 seconds, in that way there’s a continuous check on client connection to the server.

### **Arguments**

No arguments needed for this message.

### **Possible responses**

* This message has no response.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CLIENT** | | | **SERVER** | | |
| 1.1 | LoginMessage | NicknameNotValid (followed by 1.1) | | 1.2 |
| AckMatchCreationMessage (followed by 1.6) | | 1.3 |
| AskMatchToJoinMessage (followed by 1.7) | | 1.4 |
| NoLobbyAvailable (followed by 1.6) | | 1.5 |
| 1.6 | MatchSpecsMessage (“creation”) | AckMessage (subObj: “waiting”) | | 1.19 |
| 1.7 | ReplyChosenLobbyToJoinMessage | Nack (subObj: “lobby\_not\_available“) | | 1.20 |
| IDSetAfterLobbyChoiceMessage | | 1.8 |
| AckMessage(subObj: “waiting”) | | 1.19 |
| MatchStartMessage | | 1.9 |
| 1.10 | ChosenTowerColorMessage | AckMessage: tower\_color | | 1.19 |
| 1.11 | ChosenDeckMessage | AckMessage: deck | | 1.19 |
| 1.12 | BagClickMessage | AckMessage: fillclouds | | 1.19 |
| 1.13 | ChosenAssistantCardMessage | AckMessage: assistant | | 1.19 |
| 1.14 | MovedStudentsFromEntranceMessage | AckMessage: action\_1\_dining\_room / action\_1\_island | | 1.19 |
| 1.15 | MovedMotherNatureMessage | AckMessage: action\_2\_movement | | 1.19 |
| AckMessage: action\_2\_influence | | 1.19 |
| AckMessage: action\_2\_union | | 1.19 |
| 1.16 | ChosenCloudMessage | AckMessage: action\_3 | | 1.19 |
|  | | EndOfMatchMessage | | 1.22 |

**If the player wants to use a character, then…**

|  |  |  |  |
| --- | --- | --- | --- |
| 1.17 | CharacterRequestMessage | AckMessage: nome della carta | 1.19 |
| NackMessage - character non può essere utilizzato | 1.20 |
| 1.18 | CharacterDataMessage | AckCharactersMessage | 1.21 |

# **Sequence Diagrams**

## **Login Phase**

A first LoginMessage is sent from the client to the server, providing the chosen nickname and the choice (or not) of creating a new match.

The nickname provided can be not available because already used by another player, and so the server sends a NicknameNotValidMessage, which asks the client to provide a new nickname with a new LoginMessage.

If the nickname is accepted, the server can reply in different ways. First, the server asks with an AckMatchCreationMessage if in the LoginMessage received before is asked a creation of a new one, or, in reverse, with an AskMatchToJoinMessage.

The first message needs as reply from the client, a MatchSpecsMessage, providing the number of the total player for the game that is going to be created and a Boolean value which represent the expertMode choice for that match. To the MatchSpecsMessage the server replies with an AckMessage, with subObject “waiting”, which means that the client as to wait till the moment of starting the match (AckMessage with subObject “start”). Another reply from the server (after accepting the nickname) can be an AskMatchToJoinMessage. This message is sent if the Boolean value for creating a new match in the login message is false. With this message, the server sends to the client the available lobby that he can join. To this message, the client replies with a ReplyChosenLobbyToJoinMessage, containg the id of the lobby the client wants to join. To this message, the client can reply with a confirmation (IDSetAfterLobbyChoiceMessage, which provides to the client his id for the joined match, and AckMessage, with subObject “waiting”, which means that the client as to wait till the moment of starting the match, an AckMessage with subObject “start”), or with a rejection (NackMessage, with subObject “lobby\_not\_available”),) if the lobby can’t be joined because reached the maximum number of player while choosing to joining it.

Another message that can be send from the client if the Boolean value for creating a new match in the login message is false, is a NoLobbyAvailableMessage, which tells the client that he can’t join any lobby, because in that moment there is no one available (no one created yet or all the created lobbies are full). So, the client is asked to create a new lobby and to do that he sends a a MatchSpecsMessage, providing the number of the total player for the game that is going to be created and a Boolean value which represent the expertMode choice for that match. To the MatchSpecsMessage the server replies with an AckMessage, with subObject “waiting”, which means that the client as to wait till the moment of starting the match (AckMessage with subObject “start”).

## **Planning Phase**

Immagine che contiene tavolo

Descrizione generata automaticamente

During the planning phase, the first client (chosen randomly on the match start moment by the controller) send a BagClickedByFirstPlayer, which tells the server to draw students from the bag for the planning phase. The reply from the server is an AckMessage, with subObject “refillClouds”.

After this last AckMessage, the first client starts to choose an assistant card (then this will be done also by the other clients), and the chosen card is sent to the server with a ChosenAssistantCardMessage, providing the number of the chosen assistant card (1-10, one of the available), to which the server replies with a confirmation, an AckMessage with subObject “assistant”.

## **Action Phase, move 1**

Immagine che contiene testo

Descrizione generata automaticamente

The client notifies the server when the player has chosen which student he wants to move from the entrance and where, to the dining room or on an island.

The server will answer with an AckMessage with subObject "action\_1\_dining\_room" if the student has been moved to the diningroom, or "action\_1\_island" if the student has been moved on an island. This process repeats until the player has moved 3 students, or 4 if three players are playing in the game.

## **Action Phase, move 2**

Immagine che contiene testo

Descrizione generata automaticamente

The client notifies the server when the player has moved mother nature on an island.

If the destinationIsland\_ID, which is the island where mother nature lands after being moved, is not valid, which means, for example, that the player has moved mother nature by more steps than possible, the server answers with a NackMessage with subObject "invalid\_mother\_nature\_movement". This process repeats until the player moves mother nature on a correct island.

When this happens, the server answers with three different ackMessages: the first one with subObject "action\_2\_movement", only regarding the movement itself. The second one with subObject "action\_2\_influence", meaning that the influence on the destination island has been computed and the last one with subObject "action\_2\_union", meaning that two, or more, islands have been unified if the conditions for unification.

## **Action Phase, move 3**

The client notifies the server when the player has chosen a cloud from which to take

the 3 or 4 students to put on his board’s entrance with a "ChosenCloudMessage".

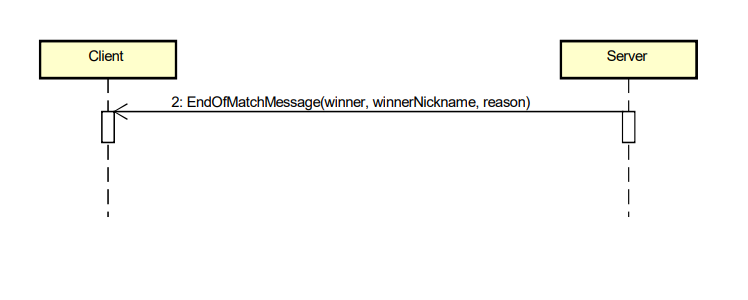
If the cloud was not already chosen by another player, then the server will send an AckMessage

with subObject "action\_3", meaning that the choice was valid; while, if the cloud was already chosen

by another player then the server will send a NackMessage with subObject "cloud\_invalid", in this last case the client needs to send another ChosenCloudMessage to which the server will respond

as previously explained.

## **End of Match**



When one of the end-of-match conditions has been met, the server will send an EndOFMatchMessage, containing:

- the ID of the winner;

- the nickname of the winner;

- the condition that led to the end of the match.

The server does not expect the client to send a response, it closes the sockets and the game session

is terminated.