Orsum Inflandi II Protocol Definition

Standard structure of message packets

Client Message

`[Message UUID (RFC 4122)]:[Domain]:[Command]:[Payload/Params *OPTIONAL*]`

Server Response

`[Inital Request Message UUID (RFC 4122)]:[Status]:[Payload/Answer *OPTIONAL*]`

Server Broadcast Message

`[Request Message UUID (RFC 4122)]:broadcast:[Command]:[Payload/Message *OPTIONAL*]`

Blue payload message components are Base64 Encoded
Each packet is terminated by a single line terminator '\x'

Possible Response Statuses

- success
  - => Possible Response
- failure
  - => Error description

Packets

Legend

- **Client**
- **Server**

Connection

- **Connect**
  - connection:connect:<User>
  - failure || success:<GameList>

Game

- **New**
  - game:new:<GameRequest>
  - failure || success:<Game>
- **Join**
  - game:join:<{ gameID: integer }>
  - <<status>>
- **Turn**
  - game:turn:<typeof Turn>
  - <<status>>

Chat

- **Send message**
  - chat:send:<{ message: string }>
  - <<status>>

There will be a lobby chat and a separate game chat

Broadcast messages

Chat

- **Receive chat message**
  - broadcast:chat:<ChatMessage>
- **Game list update**
  - broadcast:games:<GameList>
- **Receiving opponent's turn**
  - broadcast:turn:<typeof Turn>
- **Receive game ending**
  - broadcast:end:<GameResult>

Types

GameList

```
struct GameList {
      games: Game[],
}

struct GameRequest {
      name: string,
      type: GameType,
}

struct User {
      name: string,
}

struct Game : GameRequest {
      id?: integer,
      initiator?: User,
      opponent?: User,
}

enum GameType {
      fourInARow
}

struct Turn {
      gameID: integer,
}

struct FourInARowTurnPayload : Turn {
      row: integer,
}

enum ChatMessageContext {
      Lobby, InGame,
}

struct ChatMessage {
      context: ChatMessageContext,
      user: User,
      content: string,
}

enum GameResult {
      Won, Lost, Tie
}
```