Note: The security header is not included in the protocol format. Each protocol will have a prefix of [4] Security Header, [4] Msg Length and [4] Protocol, and a suffix of [4] Security Header so it is not necessary to include it in every protocol description.

Example:

[4] Security Header

[4] Message Length of (Protocol Length + Data Length)

[4] Protocol

[N] Data

[4] Security Header

CLIENT TO SERVER Protocols

|  |  |
| --- | --- |
| **CS\_Protocol** | **Description** |
| Connection Oriented |  |
| **CLOSE\_GAME** | Will close the game for the player, and the connection to the server.  ***void CloseGame()*** |
| **LOGIN\_PLAYER** | Sends a request to the server for the player to connect.  ***boolean LoginPlayer(string playerName)***  [1] Player Name length, [N] Player Name |
| **LOGOUT\_PLAYER** | Requests that the server disconnect the player.  ***void LogoutPlayer()*** |
|  |  |
| Game Viewer |  |
| **REFRESH\_GAMES** | Sends a request for a list of refreshed games.  ***void RefreshGames()*** |
| **JOIN\_GAME** | Requests that the player joins the selected game.  ***void JoinGame(uint32 gameID)***  [4] GameID |
| **CREATE\_GAME** | Request sent to the server to create a game with the player as the host.  ***void CreateGame()*** |
|  |  |
| Game Lobby |  |
| **LEAVE\_GAME** | Requests that the player leave the game they are currently in.  ***void LeaveGame(uint32 gameID)***  [4] GameID |
| **DISBAND\_GAME** | Requests that the game is disbanded.(HOST ONLY)  ***Void DisbandGame(uint lobbyID)***  [4] LobbyID |
| **JOIN\_TEAM** | Requests that the player joins the team they have selected  ***void JoinTeam(uint lobbyID, uint TeamID)***  ***[4] LobbyID [1] TeamNumber (team1 or team2?)*** |
| **READY\_STATUS** | Toggles the player’s ready status, and sends it to the server  ***void UpdatePlayerReady(uint lobby)***  [4]LobbyID |
| **BENCH\_PLAYER** | Requests that the player is sent back to the bench if they are currently on a team.  ***void BenchPlayer(uint lobbyID)***  ***[4]LobbyID*** |
|  |  |
| Chat |  |
| **SEND\_LOBBY\_MESSAGE** | Will send a message to the server meant for all the players within that game.  ***void SendLobbyMessage (uint gameID, string message)*** |
| **SEND\_WHISPER** | Will send a whisper request to the server for a specified person.  ***void SendWhisper(string recipientName, string message)*** |
| **SEND\_PARTY\_MESSAGE** | Will send a message to the server meant for players in the same  ***void SendPartyMessage(uint teamID, string message)*** |
|  |  |
| In-Game Commands |  |
| **KICK\_PLAYER** | Requests to the server that a designated player is kicked from the match.  *server:* ***void KickPlayer(string playerName)*** |
| **PAUSE\_GAME** | A request sent to the server to pause the game. - When two are received in X amount of time the game will pause.  ***void PauseRequest(string playerName, uint gameID)*** |
| **END\_GAME** | *Game host only.* Will send a request to the server to save and quit the current game.  ***void SaveAndQuit(uint gameID)*** |
| **RESUME\_GAME** | *Game host only.* Will send a request to server to resume game from a paused status.  ***void ResumeGame(string hostPlayerName)*** |
| **QUIT\_GAME** | Tells the server that a player has left the game  ***void QuitGame(string playerName, uint gameID)*** |
|  |  |
| Gameplay |  |
| **PLACE\_TOWER** | Sends a requests to the server to place a tower in the designated spot.  ***void PlaceTower (string playerName, uint towerID, uint gridspace)***  *returns true if the tower placement is allowed and false if not allowed* |
| **SUMMON\_MINION** | Sends a request to the server to place a minion at the starting portal.  ***void SummonMinion(uint teamID, string playerName, uint minionID)***  summons a minion for team by player at startPortal  returns true if successful |
| **RESEARCH\_MINION** | Sends a request to research a minion for the team.  ***void ResearchMinion(string playerName, uint minionID)***  *returns true if successful* |
| **RESEARCH\_TOWER** | Sends a request to research a tower for the team.  ***void ResearchTower(string playerName, uint towerID)***  unlocks the tower type for the player if successful  returns true if successful |

SERVER TO CLIENT Protocols

|  |  |
| --- | --- |
| **Protocol** | **Description** |
| Connection |  |
| **RETURN\_LOGIN\_PLAYER** | Returns information for logging in  ***void PlayerLoggedIn()***  ***[1] username length [n] username*** |
| **CLIENT\_LOST** | Sends info to all clients in the case that someone disconnects.  ClientDropped(string username)  [1] Username Length, [N] Username |
| **CLIENT\_JOINED** | Sends info to all clients in the case someone reconnects  clientJoined(string username)  [1] Username Length, [N] Username |
|  |  |
| Game Viewer |  |
| **SEND\_GAME\_LOBBY** | Sends the current available games from the server to the client to be viewed.  ***void RefreshGames(string hostname, uint numOfPlayers, uint maxNumOfPlayers)***  ***[1]hostname length [n] hostname [1] #ofPlayers [1]max#ofPlayers*** |
| **RETURN\_JOIN\_GAME\_STATUS** | Sends over a notification letting the player know if they were able to join the selected game or not.  ***void PlayerJoinedGame(boolean response, uint gameID)***  [1] Boolean, [4] GameID |
| **RETURN\_CREATED\_GAME** | Sends over a notification letting the player know if their game was created or not.  ***void PlayerCreatedGame(boolean response, uint gameID)***  ***[1]Boolean, [4]GameID*** |
|  |  |
| Game lobby |  |
| **BROADCAST\_DISBAND\_GAME** | Will notify all other players in the game lobby that the game has been disbanded, and should return to game viewer.  Void DisbandGame() \*\*HOST ONLY\*\* |
| **RETURN\_JOIN\_TEAM** | Returns the response to the player if they were able to join the team or not.  ***void PlayerJoinedTeam(string username, uint teamID)***  [1] username length [n] username [1]teamID |
| **BENCH\_PLAYER** | Tells everyone to bench a specific player  ***void BenchPlayer (string playerName)***  ***[1]username length [n] username*** |
|  |  |
| Chat |  |
| **SEND\_PARTY\_MESSAGE** | Will send a message to all players in the specified game.  ***void SendPartyMessage(uint teamID, string message)***  [1] teamID [1] message length [n] message |
| **SEND\_LOBBY\_MESSAGE** | Will send a message to all players within the same game lobby.  ***Void SendLobbyMessage(string message)***  **[1] message length [n] message** |
| **SEND\_WHISPER** | Will send the whisper to the chat box of the specified player.  Void SendWhisper(string message)  [1] Message Length [N] Message |
|  |  |
| In-Game Commands |  |
| **RETURN\_PAUSE\_GAME** | Relays information back to the player requestion to pause the game.  ***void PlayerPausedGame(boolean response)***  [1] Boolean |
| **RETURN\_END\_GAME** | Relays information back to the player requesting to end the game.  ***void PlayerEndedGame(boolean response)*** |
| **RETURN\_RESUME\_GAME** | Relays information back to the player requestion to resume the game  ***void PlayerResumedGame(boolean response, GameInfo gameInfo)*** |
| **RETURN\_QUIT\_GAME** | Relays the response to the player that requested to quit the game.  ***void PlayerQuitGame(boolean response)***  [1] Boolean |
| **RETURN\_KICK\_STATUS** | Relays information about the person to be kicked back to the host. |
|  |  |
| Gameplay |  |
| **SUMMON\_MINION** | Notifies the player if the server was able to summon the minion or not  ***void sendMinionInfo(string playerName, boolean minionAcknowledgement)***  [1] Boolean |
| **BROADCAST\_SUMMON\_MINION** | Broadcasts the information of the minions being summoned to all the players in the game.  ***void SendMinionToClient(Connection user, Command command, Message arguments)***  gets called multiple times by BroadcastMinionToGame(); |
| **TOWER\_PLACED** | Notifies the player if the server was unable or able to place the tower.  [1] Boolean |
| **BROADCAST\_TOWER\_PLACED** | Broadcasts the information of the tower to all the players in the game if the placement was successful.  ***void SendTowerToClient(Connection user, Command command, Message arguments)***  gets called multiple times by BroadcastTowerToGame(); |
| **UPDATE\_ECONOMY** | ***void UpdateClientEconomy(Connection client, uint[] economyValues)***  updates one client’s economy  called multiple times by UpdateEconomy(); |
| **UPDATE\_BASE\_HP** | ***void UpdateBaseHP(uint TeamID)***  updates remaining HP for the 2 main bases |
| **ASSIGN\_RESOURCE\_NODE** | ***void AssignResourceNode(uint TeamID, uint ResourceNodeID)***  Assigns a particular resource node to a particular team once they have captured it using their ZoI |
| **RESEARCH\_MINION** | Returns the results if the minion was successfully researched or not.  [1] Boolean |
| **RESEARCH\_TOWER** | Returns the results if the tower was successfully researched or not.  [1] Boolean |