RFC: [Number]

­­

[Protocol Name]

[Date]

Table of Contents

[1. Introduction 3](#_Toc160527028)

[2. Protocol Objective 3](#_Toc160527029)

[3. Protocol Overview 3](#_Toc160527032)

[4. Protocol Specification 3](#_Toc160527035)

[4.1. Service Location 3](#_Toc160527036)

[4.2. Sequence of Inter-process Communication 3](#_Toc160527037)

[4.3. Representation and Interpretation of Data Exchanged 4](#_Toc160527038)

[4.4. Description of Message Formats 4](#_Toc160527046)

[4.5. Error Handling 4](#_Toc160527047)

[4.6. Service Session Management 4](#_Toc160527048)

[5. Implementation of Functions 5](#_Toc160527049)

[5.1. Log On 5](#_Toc160527055)

[5.2. Upload Request 5](#_Toc160527059)

[5.3. Download Request 5](#_Toc160527060)

[5.4. Download All Requests 5](#_Toc160527061)

[5.5. Log Off 5](#_Toc160527062)

[6. Sequence Diagram 5](#_Toc160527063)

[7. Security Considerations 5](#_Toc160527064)

[Glossary 6](#_Toc160527068)

[9. Author 6](#_Toc160527069)

[10. References 6](#_Toc160527070)

##### Documenting protocol

* Introduction + objectives
* Use sequence Diagram
* Define Messages and format of messages e.g.
  + Message: Login
  + Description: The user can log into the server with a username and password
  + Code/Number: 100
  + Message Parameters: username, password (both text)
  + Response Message
    - Code: 101
    - Text: login successful
  + Response Message
    - Code: 102
    - Text: login successful

Describe how functionality of each message is implemented

e.g. Implementing the Connection message on the server as a function called login()

// save the username and password

// is a folder ! exists for username create it

// return 101: login successful

// if cant create folder return 102

Sequence diagram of Protocol Messages

See RFC’s online

# **1. Introduction**

The protocol is intended to provide secure and efficient interactions between players and the game server.

This document describes the [Protocol Name], a protocol designed for facilitating communication between clients and servers in a text-based game environment.

# **2. Protocol Objective**

The objectives of the [Protocol Name] protocol are to enable players to log on to the game server, interact with the game world, upload and download messages, and log off securely.

The protocol ensures confidentiality, integrity, and reliability of data exchanged between clients and servers.

(Information Sciences Institute, University of Southern California, 1981)



# **Protocol Overview**

[Protocol Name] is a simple client-server protocol based on a request-response model. It supports secure communication using SSL/TLS encryption.



# **4. Protocol Specification**

## **4.1. Service Location**

Clients connect to the server's IP address and port number.

Communication is secured using SSL/TLS encryption.

## **4.2. Sequence of Inter-process Communication**

The server listens for incoming client connections over SSL/TLS.

The client initiates a secure connection request to the server.

Upon successful connection establishment, the client and server perform an SSL/TLS handshake to authenticate and establish secure communication channel.

Once the handshake is completed, the server responds to the clients connection request.

He client acknowledges the server's response over the secure connection.

Communication continues bidirectionally over the established SSL/TLS connection until session termination. Session termination can occur due to a timeout or termination by either party.

## **4.3. Representation and Interpretation of Data Exchanged**

Requests and responses follow a predefined format agreed upon by both client and server.

JSON data format is used for encoding game commands, responses, and other data over the secure connection.



## **Description of Message Formats**

The message formats are in JSON structure, making them easy to parse and interpret by both the client and server. Each message includes a "type" field to indicate the action requested, along with any other necessary parameters. The server processes these messages according to the protocol and responds to the client accordingly.

* + 1. **Login Request Message Format**

The login request message format includes the following elements:

Description: The user can login to the server with a username and password

Code: 100

Parameters:

Type: Login

Username: [username]

Password: [password]

Example:

+-------------------------------------+

| { |

| “type”: “LOGIN”, |

| “username”: “example\_user”, |

| “password”: “example\_password” |

| } |

+-------------------------------------+

Response Message:

Code: 101

Text: “Login successful, welcome to [ protocol name ]”

Response Message:

Code: 102

Text: “Login unsuccessful, please try again”

* + 1. **Upload Request Message Format**

The upload request message format includes the following elements:

Description: The user can upload a message to the server once authenticated.

Code: 200

Parameters:

Type: Upload

Message: [ message content ]

Example:

+-------------------------------------+

| { |

| “type”: “UPLOAD”, |

| “message”: “This is a message” |

| } |

+-------------------------------------+

Response Message:

Code: 201

Text: “Upload successful”

Response Message:

Code: 202

Text: “Upload unsuccessful, please try again”

* + 1. **Download Request Message Format**

The download request message format includes the following elements:

Description: The user can download a specific message from the server once authenticated.

Code: 300

Parameters:

Type: Download

Message ID: [ ID of message to download ]

+-------------------------------------+

| { |

| “type”: “DOWNLOAD”, |

| “message\_id”: “12345” |

| } |

+-------------------------------------+

Response Message:

Code: 301

Text: “Download of message [ message\_id ] successful”

Response Message:

Code: 302

Text: “Download of message [ message\_id ] unsuccessful, please try again”

* + 1. **Download All Request Message Format**

The download all request message format includes the following elements:

Description: The user can download all available message from the server once authenticated.

Code: 400

Parameters:

Type: Download\_all

+-------------------------------------+

| { |

| “type”: “DOWNLOAD\_ALL” |

| } |

+-------------------------------------+

Response Message:

Code: 401

Text: “Download of all messages successful”

Response Message:

Code: 402

Text: “Download of all messages unsuccessful, please try again”

* + 1. **Logout Message Format**

The logout request message format includes the following elements:

Description: The user can log off from the server.

Code: 500

Parameters:

Type: Logout

+-------------------------------------+

| { |

| “type”: “LOG\_OUT” |

| } |

+-------------------------------------+

Response Message:

Code: 501

Text: “Logout successful, see you again soon”

Response Message:

Code: 502

Text: “Logout unsuccessful, please try again”

## **Error Handling**

Error codes and messages are defined for common error scenarios.

Both client and server handle exceptions and errors appropriately over the secure connection.

### **Handling Authorisation Errors**

[ TODO ]

(Sengul & Kirby, 2023)

## **Service Session Management**

Upon connection establishment, a secure service session is initiated.

The server assigns a unique session identifier to each client session securely.

Session management includes maintaining session state and uploaded messages securely over the SSL/TLS connection.

Periodic updates and notifications are sent securely over the SSL/TLS connection.

Session clean-up is performed securely upon session termination.

# **Implementation of Functions**

[ TODO - if applicable ]

The server implements functions for handling login, upload, download, download all messages, and logout requests from clients.

Describe how functionality of each message is implemented

e.g. Implementing the Connection message on the server as a function called login()

// save the username and password

// is a folder ! exists for username create it

// return 101: login successful

// if cant create folder return 102



## **Log On**

[ TODO ]



## **Upload Request**

[ TODO ]

## **Download Request**

[ TODO ]

## **Download All Requests**

[ TODO ]

## **Log Off**

[ TODO ]

# **Sequence Diagram**

[ insert sequence diagram describing interaction between server and clients ]

# **Security Considerations**

Communication between client and server is encrypted using SSL/TLS, providing confidentiality and integrity for sensitive data.



# **Glossary**

[ if applicable ]

# **Author**

[ TODO ]

# **References**

Information Sciences Institute, University of Southern California, 1981. *TRANSMISSION CONTROL PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION.* [Online]   
Available at: https://datatracker.ietf.org/doc/html/rfc793  
[Accessed 4 March 2024].

Sengul, C. & Kirby, A., 2023. *Message Queuing Telemetry Transport (MQTT) and Transport Layer Security (TLS) Profile of Authentication and Authorization for Constrained Environments (ACE) Framework.* [Online]   
Available at: https://datatracker.ietf.org/doc/html/rfc9431  
[Accessed 4 March 2024].