# Simple IMC Messaging Protocol

# Documentation

## Authors:

## Lisa-Maria Hollaus

## Selin R. Meşeli

## Due Date: 09.01.2025, 23:59

**"Simple IMC Messaging Protocol"**

**Authors: Hollaus, Meseli**

This project implements a **Simple IMC Messaging Protocol (SIMP)** that allows clients to connect and chat with each other through daemons. The protocol we implemented (besides the required one between daemons) for daemon- client communication, is text-based and includes basic operations such as connecting, chatting, and quitting, depending on the choice of the users.

**Table of Contents**

* [Final Assignment "Simple IMC Messaging Protocol"](http://localhost:63342/markdownPreview/1598062234/markdown-preview-index-516810009.html?_ijt=t86ti3tehp37555pq0q0350hti#final-assignment-simple-imc-messaging-protocol) - [Authors: Hollaus, Meseli](http://localhost:63342/markdownPreview/1598062234/markdown-preview-index-516810009.html?_ijt=t86ti3tehp37555pq0q0350hti#authors-hollaus-meseli)
  + [Table of Contents](http://localhost:63342/markdownPreview/1598062234/markdown-preview-index-516810009.html?_ijt=t86ti3tehp37555pq0q0350hti#table-of-contents)
  + [Requirements](http://localhost:63342/markdownPreview/1598062234/markdown-preview-index-516810009.html?_ijt=t86ti3tehp37555pq0q0350hti#requirements)
  + [Usage](http://localhost:63342/markdownPreview/1598062234/markdown-preview-index-516810009.html?_ijt=t86ti3tehp37555pq0q0350hti#usage):
    - [Preparations](http://localhost:63342/markdownPreview/1598062234/markdown-preview-index-516810009.html?_ijt=t86ti3tehp37555pq0q0350hti#preparations)
    - [Start the daemon](http://localhost:63342/markdownPreview/1598062234/markdown-preview-index-516810009.html?_ijt=t86ti3tehp37555pq0q0350hti#start-the-daemon)
    - [Start the client](http://localhost:63342/markdownPreview/1598062234/markdown-preview-index-516810009.html?_ijt=t86ti3tehp37555pq0q0350hti#start-the-client)
    - [Connect to another client](http://localhost:63342/markdownPreview/1598062234/markdown-preview-index-516810009.html?_ijt=t86ti3tehp37555pq0q0350hti#connect-to-another-client)
    - Protocol between daemon and daemon
    - [Protocol between client and daemon](http://localhost:63342/markdownPreview/1598062234/markdown-preview-index-516810009.html?_ijt=t86ti3tehp37555pq0q0350hti#protocol-between-client-and-daemon)
    - Details of client-daemon Protocol methods
  + [Implementation Notes](http://localhost:63342/markdownPreview/1598062234/markdown-preview-index-516810009.html?_ijt=t86ti3tehp37555pq0q0350hti#implementation-notes)
  + [Structure Overview](http://localhost:63342/markdownPreview/1598062234/markdown-preview-index-516810009.html?_ijt=t86ti3tehp37555pq0q0350hti#architecture-overview)
  + [Limitations](http://localhost:63342/markdownPreview/1598062234/markdown-preview-index-516810009.html?_ijt=t86ti3tehp37555pq0q0350hti#known-limitations)

**Requirements**

* **Python 3.10+** (We developed it in Python 3.11)

**Usage**

**Preparations**

* Activate the virtual environment:
* .\venv\Scripts\activate (We used Microsoft, so our command was this)
* Install requirements.py
* pip install -r requirements.py (We did not use anything too foreign. In fact, our requirements.txt was empty in the first place, so we added the auto-suggested import packages from PyCharm’s suggestion)

**Start the daemon**

In order to start the daemon, run the following command in the terminal: python simp\_daemon.py followed by a valid daemon IP.

Example: python simp\_daemon.py 127.0.0.1

**Start the client**

In order to start the client, run the following command in the terminal: python simp\_client.py followed by a valid daemon IP of an already running daemon.

Example: python simp\_client.py 127.0.0.1

**Connect to another client**

1. Input the username of the client
2. Create another daemon - client pair (example using 127.0.0.2)
3. If there are other users you can connect to them, otherwise you have to wait for another user to connect to you
4. The other user can accept or decline the connection request
5. If the connection request is accepted, you can start chatting with the other user
6. quit the chat by typing “q” in the message field

**Protocol between daemon and daemon**

We implemented the **SIMP** Protocol just as specified in the requirements of the Project, therefore we won’t go into much detail here.

The Protocol consists of the following:

1. Type (1 byte): specifies control datagram (0x01) or chat datagram (0x02).

2. Operation (1 byte): indicates the type of operation:

* For control datagram: ERR (0x01) SYN (0x02), ACK (0x04), SYN\_ACK (0x06) FIN (0x08)
  + Note: we added SYN\_ACK (0x06) for better usage of the class Operation in simp\_protocol
* For chat datagram): CONST (0x01)

3. Sequence (1 byte): changes values 0x00 or 0x01 to identify resent or lost datagrams.

4. User (32 bytes): user name encoded as an ASCII string.

5. Length (4 bytes): length of the datagram payload in bytes.

6. Payload: depending on the field Type:

* If it is a control datagram and Operation == ERR: a human-readable error message as an ASCII string.
* chat datagram: the contents of the chat message to be sent.

1. Checksum (2 bytes)

* Note: we added the Checksum to ensure the integrity of the messages between daemon and daemon protocol.

**Protocol between client and daemon**

We implemented a simple protocol between the client and the daemon, based on a simple text-based format.

The protocol defines the following operations:

* **Ping**: Check if the daemon is alive.
* **Connect**: Establish a connection between the client and the daemon.
* **Connecting**: The daemon is connecting to another daemon (chat partner).
* **Chat**: Send chat messages from the client to the daemon.
* **Quit**: Disconnect the client from the daemon.
* **ERROR**: Send an error message from the daemon to the client.

Each message will have a simple format: OPERATION|PAYLOAD

* **Operation**: A short string to define the type of request (e.g., CONNECT, CHAT, QUIT).
* **Payload**: Optional additional data (e.g., username, chat message).

Example:

CONNECT|Alice

CHAT|Hello, Bob!

QUIT|

**Details of client-daemon Protocol methods**

1. **Ping**:
   * **Purpose**: To check if the daemon is alive and responsive.
   * **Client Action**: Sends PING| to the daemon.
   * **Daemon Response**: Replies with PONG| to confirm it is alive.
2. **Connect**:
   * **Purpose**: To establish a connection between the client and the daemon.
   * **Client Action**: Sends CONNECT|<username> to the daemon.
   * **Daemon Response**: Replies with CONNECTED|<username> to confirm the connection.
3. **Connecting**:
   * **Purpose**: To notify the daemon that it is connecting to another daemon (chat partner).
   * **Daemon Action**: Sends CONNECTING|<username> to the other daemon.
   * **Daemon Response**: Replies with CONNECTED|<username> to confirm the connection.
4. **Chat**:
   * **Purpose**: To send chat messages between clients through their respective daemons.
   * **Client Action**: Sends CHAT|<message> to the daemon.
   * **Daemon Action**: Forwards CHAT|<username>|<message> to the other daemon.
   * **Daemon Response**: Forwards CHAT|<username>|<message> to the client.
5. **Quit**:
   * **Purpose**: To disconnect the client from the daemon.
   * **Client Action**: Sends QUIT| to the daemon.
   * **Daemon Response**: Replies with QUIT| to confirm the disconnection.
6. **Error**:
   * **Purpose**: To notify the client of any errors.
   * **Daemon Action**: Sends ERROR|<error\_message> to the client.

**Implementation Notes**

* We added a checksum because we believe it is a main property of UDP Protocols as we learned in class and to ensure the integrity of the messages between daemon and daemon protocol (16-bit).
* We added type definitions via python's built in typing library for the documentation in order to make usage and understanding of the code easier.
* We assumed the client knows the IP address of the users daemon he wants to connect to, so we did not implement a discovery mechanism.
* We added a timeout of 30 seconds for the chat partner to respond until the client gets asked again if he wants to keep on waiting or quit, to give the client the option to quit.
* Whenever a ACK is not received a sequence number error should occur, as the tracking of the sequence number is only updated by receiving a ACK (a way to improve this would be to automatically resend original message whenever a sequence error occurs).

**Structure Overview**

The application consists of two main parts:

1. **Daemon**: Handles network connections, message validation, and forwarding.
   * Running on two main threads:
     1. One that listens to new connections of clients
     2. and another that waits for incoming daemon connections and handles chat requests
2. **Client**: Provides user interaction, sends and receives messages through the daemon.

**Limitations**

* No built-in service discovery for other daemons. Therefore, clients must know the target daemon’s IP address.
* Messages are not encrypted, which leaves the communication(s) vulnerable to any potential interception.
* Limited support for recovering from sequence number mismatches.

**Terminology**

* **Daemon:** A background process that manages network communication.
* **Checksum:** A calculated value used to verify message integrity.
* **ACK:** Acknowledgement of a received message.
* **Sequence Number:** Tracks the order of transmitted messages.

**Future Enhancement Ideas**

* Currently, the client has to know the IP address of the corresponding deamon. This could be somehow handled to ensure a more dynamic system.
* Using protocols to secure communication with encryption.
* Error handling could be enhanced by retry mechanisms for sequence mismatches.
* Perhaps extending the protocol to allow multi-user conversations instead of 1:1 chats.