# 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 a daemon. The protocol is text-based and includes basic operations such as connecting, chatting, and quitting, depending on the choice of the users.

**Table of Contents**



**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

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

Example: python simp_client.py 127.0.0.1

**Connect to another client:**

1. Enter the username when prompted
2. Start a daemon-client pair for another user. Example: by using 127.0.0.2
3. Connect to other client:
    - The other client can either accept or decline your request
4. Begin chatting once the connection is accepted, type “q” in the chat to quit it

**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|

**Detailed 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 to the protocol to ensure the integrity of the messages between daemon and daemon (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.
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.