

DOKUMENTATION

Erzeugt von Doxygen 1.14.0

1 DOKUMENTATION	1
1.1 Peer-to-Peer Chat – Dokumentation	1
1.1.1 Projektüberblick	1
1.1.1.1 Gruppenmitglieder	1
1.1.1.2 Abgabedatum / Semester / Kurs	1
1.1.2 Architektur	1
1.1.3 Verwendete Technologien & Tools	2
1.1.4 Protokollübersicht (SLCP)	2
1.1.4.1 Ablauf beim Anmelden im Chat	2
1.1.4.2 Ablauf beim Abmelden aus dem Chat	3
1.1.4.3 Ablauf beim Abfragen und Empfangen der Nutzerliste	3
1.1.4.4 Ablauf beim Senden einer Textnachricht	4
1.1.4.5 Ablauf beim Senden eines Bildes	4
1.1.4.6 Ausgabe-Beispiel	4
1.1.5 Teilprobleme und Lösungsansätze (Fehleranalyse)	6
1.1.5.1 IndexError beim Parsen von Benutzerdaten	6
1.1.5.2 Relativer Bildpfad und Ordnerprüfung	7
1.1.5.3 Konsistente CLI-Befehle und Hilfeanzeige	8
1.1.5.4 Flexibler Umgang mit Peer-Daten (JOIN/KNOWNUSERS)	9
1.1.5.5 Konsolidierung mehrerer KNOWNUSERS-Antworten	10
1.1.5.6 Verbesserte Anzeige des Absenders	12
1.1.5.7 Doppelte Einträge in WHO/KNOWNUSERS-Liste	13
1.1.6 Bedienung	14
2 mainpage2	15
3 Verzeichnis der Namensbereiche	17
3.1 Liste aller Namensbereiche	17
4 Hierarchie-Verzeichnis	19
4.1 Klassenhierarchie	19
5 Klassen-Verzeichnis	21
5.1 Auflistung der Klassen	21
6 Datei-Verzeichnis	23
6.1 Auflistung der Dateien	23
7 Dokumentation der Namensbereiche	25
7.1 config.config-Namensbereichsreferenz	25
7.1.1 Ausführliche Beschreibung	25
7.2 discovery.discovery_service-Namensbereichsreferenz	25
7.2.1 Ausführliche Beschreibung	25
7.2.2 Dokumentation der Funktionen	25

7.2.2.1 is_port_in_use()	25
7.3 interface-Namensbereichsreferenz	26
7.3.1 Ausführliche Beschreibung	26
7.4 main-Namensbereichsreferenz	26
7.4.1 Ausführliche Beschreibung	26
7.4.2 Dokumentation der Funktionen	26
7.4.2.1 main()	26
7.5 network.messenger-Namensbereichsreferenz	26
7.5.1 Ausführliche Beschreibung	26
8 Klassen-Dokumentation	27
8.1 config.config.Config Klassenreferenz	27
8.1.1 Ausführliche Beschreibung	27
8.1.2 Beschreibung der Konstruktoren und Destruktoren	27
8.1.2.1 __init__()	27
8.1.3 Dokumentation der Elementfunktionen	28
8.1.3.1 _setup_imagepath()	28
8.1.3.2 load()	28
8.1.3.3 save()	28
8.2 discovery.discovery_service.DiscoveryService Klassenreferenz	28
8.2.1 Ausführliche Beschreibung	29
8.2.2 Beschreibung der Konstruktoren und Destruktoren	29
8.2.2.1 __init__()	29
8.2.3 Dokumentation der Elementfunktionen	29
8.2.3.1 get_local_ip()	29
8.2.3.2 get_peers()	29
8.2.3.3 handle_message()	29
8.2.3.4 listen()	29
8.2.3.5 load_config()	30
8.2.3.6 send_join()	30
8.2.3.7 send_leave()	30
8.2.3.8 send_who()	30
8.2.3.9 start()	30
8.2.3.10 stop()	30
8.3 interface.Interface Klassenreferenz	30
8.3.1 Ausführliche Beschreibung	31
8.3.2 Beschreibung der Konstruktoren und Destruktoren	31
8.3.2.1 __init__()	31
8.3.3 Dokumentation der Elementfunktionen	31
8.3.3.1 display_image_notice()	31
8.3.3.2 display_knownusers()	31
8.3.3.3 display_message()	32

8.3.3.4 run()	32
8.4 network.messenger.Messenger Klassenreferenz	32
8.4.1 Ausführliche Beschreibung	33
8.4.2 Beschreibung der Konstruktoren und Destruktoren	33
8.4.2.1 __init__()	33
8.4.3 Dokumentation der Elementfunktionen	33
8.4.3.1 connection_made()	33
8.4.3.2 datagram_received()	33
8.4.3.3 get_local_ip()	34
8.4.3.4 handle_knownusers_response()	34
8.4.3.5 handle_message()	34
8.4.3.6 handle_tcp_connection()	34
8.4.3.7 receive_image_data()	34
8.4.3.8 send_broadcast()	35
8.4.3.9 send_image()	35
8.4.3.10 send_image_data()	35
8.4.3.11 send_join()	35
8.4.3.12 send_known_to()	35
8.4.3.13 send_leave()	36
8.4.3.14 send_message()	36
8.4.3.15 send_slcp()	36
8.4.3.16 send_who()	36
8.4.3.17 set_image_callback()	36
8.4.3.18 set_knownusers_callback()	36
8.4.3.19 set_message_callback()	37
8.4.3.20 set_progress_callback()	37
8.4.3.21 start_listener()	37
8.4.3.22 start_tcp_server()	37
9 Datei-Dokumentation	39
9.1 Chat/common/protocol.py-Dateireferenz	39
9.1.1 Ausführliche Beschreibung	39
9.1.2 Dokumentation der Funktionen	39
9.1.2.1 create_img()	39
9.1.2.2 create_join()	39
9.1.2.3 create_knownusers()	40
9.1.2.4 create_leave()	40
9.1.2.5 create_msg()	40
9.1.2.6 create_who()	40
9.1.2.7 parse_slcp()	40

Kapitel 1

mainpage2

/**

1.1 Peer-to-Peer Chat – Dokumentation

1.1.1 Projektüberblick

Ein dezentraler Chat-Client im lokalen Netzwerk, der Text- und Bildnachrichten ohne zentralen Server ermöglicht. Hauptfunktionen: Peer Discovery, Text- und Bildübertragung, Kommandozeilen-Bedienung (CLI), automatische Nutzererkennung.

1.1.1.1 Gruppenmitglieder

- Aysha Aryubi
- Nhu Ngoc Le
- Artur Gubarkov

1.1.1.2 Abgabedatum / Semester / Kurs

- Abgabedatum: 22. Juni 2025
 - Semester: SoSe 2025
 - Kurs: Betriebssysteme und Rechnernetze
 - Dozent: Prof. Dr.-Ing. Markus Mietтинен
-

1.1.2 Architektur

Das System ist modular aufgebaut und besteht aus folgenden Hauptkomponenten:

- **Interface (CLI):** Stellt die Kommandozeilen-Bedienung bereit. Nutzer gibt hier Befehle wie `/msg`, `/img`, `/who` usw. ein. Das Interface leitet die Befehle an den Messenger weiter und zeigt empfangene Nachrichten und Bilder an.
 - **Messenger:** Übernimmt die Netzwerkkommunikation. Sendet und empfängt Nachrichten sowie Bilder über UDP/TCP. Kommuniziert direkt mit Interface, DiscoveryService und nutzt das Protocol-Modul zum Kodieren/Dekodieren der Nachrichten.
 - **Protocol:** Stellt Funktionen zum Kodieren und Parsen des SLCP-Protokolls bereit (JOIN, MSG, IMG etc.).
-

3. **Der Messenger** veranlasst den Discovery-Service, eine JOIN Alice <Port>-Nachricht per UDP-Broadcast im lokalen Netzwerk zu senden.
4. **Alle Discovery-Services** der anderen Peers empfangen die JOIN-Nachricht und nehmen Alice in ihre Peer-Liste auf.
5. **Alice** ist jetzt als aktiver Teilnehmer im Chat-Netzwerk bekannt.

Der Ablauf ist im folgenden Diagramm dargestellt:

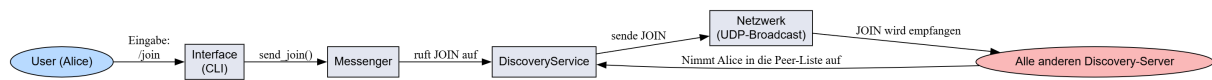


Abbildung 1.2 Ablauf JOIN

1.1.4.2 Ablauf beim Abmelden aus dem Chat

Im Folgenden wird gezeigt, wie ein Nutzer den Chat verlässt:

1. **Alice** gibt im Interface den Befehl /leave ein.
2. **Das Interface** ruft im Messenger die Funktion zum Verlassen des Chats auf.
3. **Der Messenger** veranlasst den Discovery-Service, eine LEAVE Alice-Nachricht per UDP-Broadcast im lokalen Netzwerk zu senden.
4. **Alle Discovery-Services** der anderen Peers empfangen die LEAVE-Nachricht und entfernen Alice aus ihrer Peer-Liste.
5. **Alice** wird aus dem Netzwerk entfernt und erscheint nicht mehr als aktiver Teilnehmer.

Der Ablauf ist im folgenden Diagramm dargestellt:



Abbildung 1.3 Ablauf LEAVE

1.1.4.3 Ablauf beim Abfragen und Empfangen der Nutzerliste

Im Folgenden wird gezeigt, wie die Liste aller aktiven Nutzer im Chat abgefragt wird:

1. **Bob** gibt im Interface den Befehl /who ein.
2. **Das Interface** ruft im Messenger die Funktion zum Senden einer WHO-Anfrage auf.
3. **Der Messenger** veranlasst den Discovery-Service, eine WHO-Nachricht per UDP-Broadcast an alle Discovery-Services im lokalen Netzwerk zu senden.
4. **Jeder Discovery-Service** (bei allen aktiven Peers) empfängt die WHO-Anfrage und antwortet per Unicast mit einer eigenen KNOWNUSERS-Nachricht, die alle aktuell bekannten Nutzer (Handle, IP, Port) enthält.
5. **Der Messenger von Bob** empfängt eine oder mehrere KNOWNUSERS-Antworten, konsolidiert die Einträge (entfernt ggf. Duplikate) und aktualisiert seine interne Peer-Liste.
6. **Das Interface bei Bob** zeigt daraufhin die vollständige Liste aller aktuell erreichbaren Nutzer im Terminal an.

Der Ablauf ist im folgenden Diagramm dargestellt:

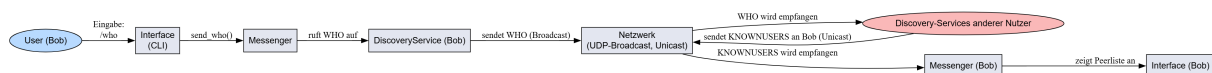


Abbildung 1.4 Ablauf WHO/KNOWNUSERS

1.1.4.4 Ablauf beim Senden einer Textnachricht

Im Folgenden wird gezeigt, wie ein Bild vom Nutzer „Alice“ an „Bob“ gesendet wird:

1. **Alice** gibt im Interface den Befehl `/msg Bob Hallo Bob!` ein.
2. **Das Interface** ruft im Messenger die Funktion zum Senden einer Nachricht auf.
3. **Der Messenger** ermittelt die Netzwerkadresse von Bob und sendet den SLCP-Befehl `MSG Bob "Hallo Bob!"` per UDP direkt an Bobs Peer-Adresse.
4. **Bob's Messenger** empfängt die Nachricht, prüft den Befehl und übergibt den Text an das lokale Interface.
5. **Das Interface bei Bob** zeigt die empfangene Nachricht sofort im Terminal an.

Der Ablauf ist im folgenden Diagramm dargestellt:

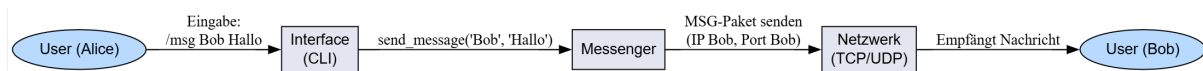


Abbildung 1.5 Ablauf MSG

1.1.4.5 Ablauf beim Senden eines Bildes

Im Folgenden wird gezeigt, wie ein Bild vom Nutzer „Alice“ an „Bob“ gesendet wird:

1. **Alice** gibt im Interface den Befehl `/img Bob pfad/zum/bild.jpg` ein.
2. **Das Interface** ruft im Messenger die Funktion zum Senden eines Bildes auf.
3. **Der Messenger** öffnet eine TCP-Verbindung zum Peer (Bob), sendet den SLCP-Befehl `IMG Bob <Größe>`, danach die Bilddaten als Bytestream.
4. **Bob's Messenger** nimmt die Verbindung entgegen, liest Befehl und Daten und speichert das Bild im vorgegebenen Ordner.
5. **Das Interface bei Bob** zeigt eine Benachrichtigung über das empfangene Bild an.

Der Ablauf ist im folgenden Diagramm dargestellt:

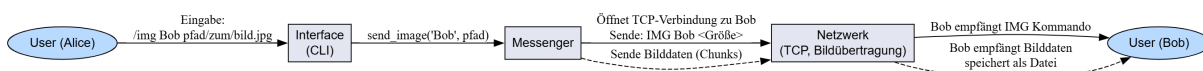


Abbildung 1.6 Ablauf IMG

1.1.4.6 Ausgabe-Beispiel

Im Folgenden einige Screenshots der CLI-Anwendung:

```

/home/aj/PycharmProjects/BSRN_Chat/.venv/bin/python /home/aj/PycharmProjects/BSRN_Chat/Chat/main.py
[DEBUG] Geladene Konfiguration: {'handle': 'laptop', 'port': 5005, 'whoisport': 4000, 'imagepath': '/home/aj/slcp_images'}
[DISCOVERY] Empfangen: JOIN laptop 5005 von ('192.168.178.51', 4000)
[DEBUG] sende WHO-Broadcast...
[DISCOVERY] Empfangen: WHO von ('192.168.178.51', 4000)
[DISCOVERY] Empfangen: KNOWUSERS Laptop 192.168.178.51 5005 von ('192.168.178.51', 4000)
[DISCOVERY] KNOWUSERS-Liste:
[Messenger] Lauscht auf Port 5005
[TCP] Server gestartet auf Port 5005
[DISCOVERY] Empfangen: KNOWUSERS laptop 192.168.178.51 5005, pc 192.168.178.52 5000 von ('192.168.178.52', 4000)
[DISCOVERY] KNOWUSERS-Liste:
[DISCOVERY] Empfangen: JOIN laptop 5005 von ('192.168.178.51', 5005)
● Willkommen im SLCP-Chat, laptop!

Verfügbare Befehle:
/join - Dem Chat beitreten
/leave - Chat verlassen
/who - Aktive Benutzer anzeigen
/msg <handle> <text> - Nachricht senden
/img <handle> <pfad> - Bild senden
/quit - Chat beenden

>> /join
[DISCOVERY] Empfangen: JOIN laptop 5005 von ('192.168.178.51', 5005)
✓ Du bist dem Chat beigetreten!
>> [DISCOVERY] Empfangen: JOIN pc 5000 von ('192.168.178.52', 5000)
/who

```

Abbildung 1.7 Ausgabe 1

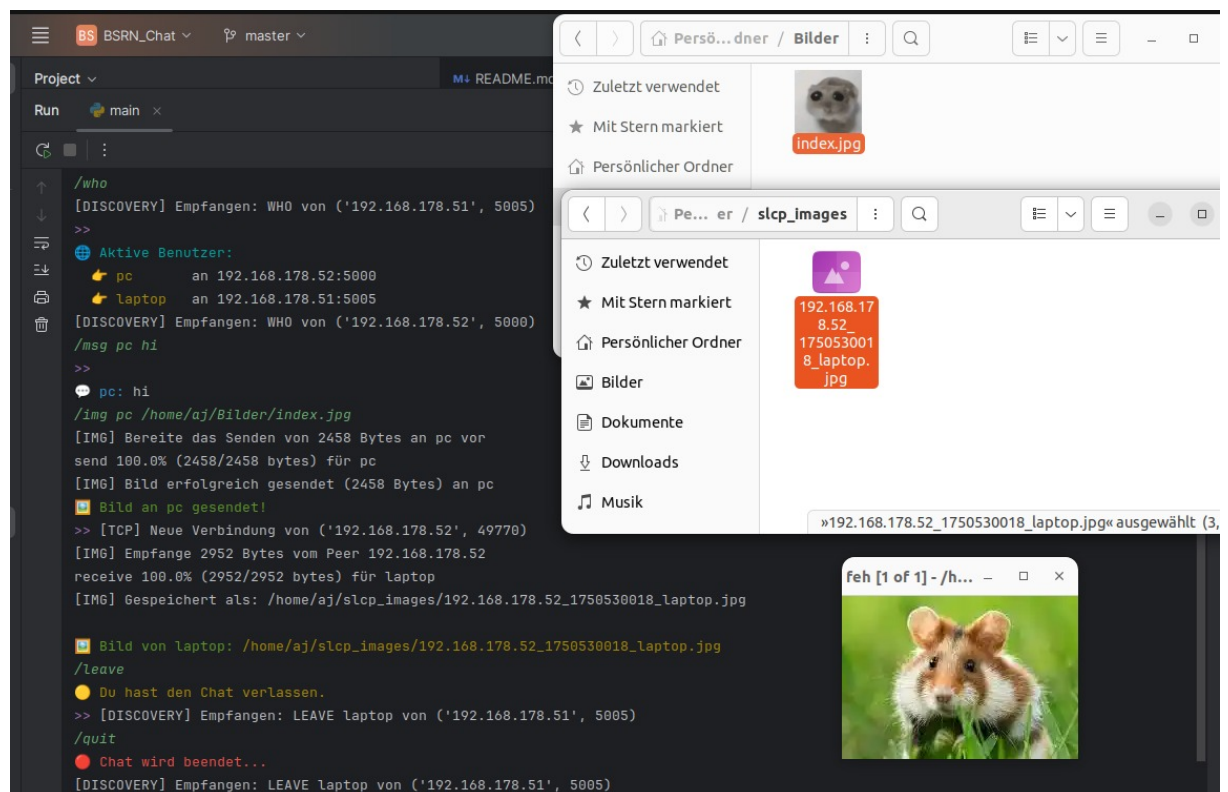


Abbildung 1.8 Ausgabe 2

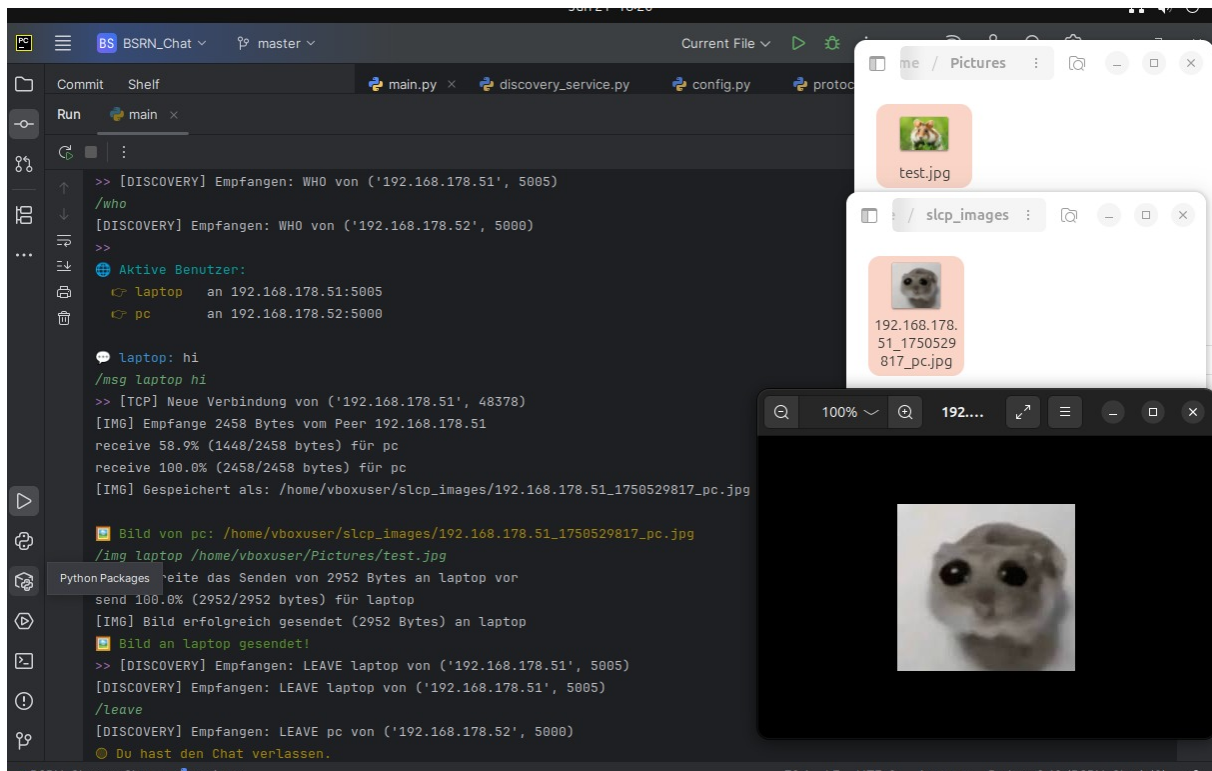


Abbildung 1.9 Ausgabe 3

1.1.5 Teilprobleme und Lösungsansätze (Fehleranalyse)

Im Folgenden werden typische Fehler und ihre Lösungen im Projekt erläutert.

Jedes Beispiel enthält einen Screenshot sowie eine kurze Analyse und die jeweilige Korrektur.

1.1.5.1 IndexError beim Parsen von Benutzerdaten

Beim Parsen der „KNOWNUSERS“-Nachricht wurde die empfangene Zeichenkette mit `info.split()` zerlegt, ohne zu prüfen, ob wirklich drei Felder (`Handle`, `IP`, `Port`) vorliegen. Wenn ein Netzwerkpaket fehlerhaft war (z.B. zu wenige Einträge enthielt), wurde auf ein nicht vorhandenes Element zugegriffen – das führte zu einem `IndexError`. Dieser Fehler trat insbesondere bei Netzwerkproblemen oder inkompatiblen Peers auf und konnte das ganze Programm zum Absturz bringen.

```

Chat\discovery\discovery_service.py
↑ ..... @@ -106,12 +106,18 @@ class DiscoveryService:
106 106         user_list = [u.strip() for u in user_str.split(",") if u.strip()]
107 107         print("[DISCOVERY] KNOWNUSERS-Liste:")
108 108         with self.peers_lock:
109 109             +         seen = set()
110 110             for entry in user_list:
111 111                 infos = entry.strip().split()
112 112                 if len(infos) == 3:
113 113                     -         handle, ip, port = infos
114 114                     -         self.peers[handle] = (ip, int(port))
115 115                     -         print(f" - {handle} @ {ip}:{port}")
116 116                     +         handle = infos[0].strip()
117 117                     +         ip = infos[1].strip()
118 118                     +         port = int(infos[2].strip())
119 119                     +         key = f"{handle}@{ip}:{port}"
120 120                     +         if key not in seen:
121 121                         +         self.peers[handle] = (ip, port)
122 122                         +         seen.add(key)
123 123                         +         print(f" - {handle} @ {ip}:{port}")
124 124
125 125         def send_who(self):
126 126             msg = "WHO\n"

```

Abbildung 1.10 IndexError Fix

Lösung:

Jetzt wird vor dem Zugriff geprüft, ob `len(infos) == 3` gilt. So werden nur gültige Peer-Informationen akzeptiert und verarbeitet, wodurch Abstürze durch fehlerhafte Netzwerkdaten zuverlässig verhindert werden.

1.1.5.2 Relativer Bildpfad und Ordnerprüfung

Ursprünglich wurde als Speicherort für empfangene Bilder ein relativer Pfad (`"./images"`) verwendet. Das war problematisch, da das Verzeichnis je nach aktuellem Arbeitsverzeichnis unterschiedlich interpretiert werden kann und bei fehlender Erstellung Dateioperationen fehlschlagen. Das führte vor allem beim ersten Programmstart zu unerwarteten Fehlern beim Speichern von Bildern.

```

3 files changed 10 lines changed

Chat/config/config.py
@@ -19,12 +19,22 @@ def __init__(self, path="slcp_config.toml"):
19     self.data["whoisport"] = self.whoisport
20
21     self.autoreply = self.data.get("autoreply", "")
22 -     self.imagepath = self.data.get("imagepath", "./images")
22 +     self.imagepath = self._setup_imagepath()
23
24     # Nur speichern, wenn noch nicht vorhanden - nicht jedes Mal!
25     if not os.path.exists(self.path) or "handle" not in toml.load(self.path):
26         self.save()
27
28 +     def _setup_imagepath(self):
29 +         raw_path = self.data.get("imagepath", "~/slcp_images")
30 +
31 +         normalized_path = os.path.abspath(os.path.expanduser(raw_path))
32 +
33 +         os.makedirs(normalized_path, exist_ok=True)
34 +
35 +         self.data["imagepath"] = normalized_path
36 +         return normalized_path
37 +
38     def load(self):
39         if os.path.exists(self.path):
40             return toml.load(self.path)

Chat/network/messenger.py
@@ -285,11 +285,11 @@ async def receive_image_data(self, reader, addr, size, sender_handle):
285         self.config.imagepath,
286         f"{addr[0]}_{int(time.time())}_{sender_handle}.jpg"
287     )
288 -     os.makedirs(self.config.imagepath, exist_ok=True)
288 +     os.makedirs(os.path.dirname(filename), exist_ok=True)
289
290     with open(filename, "wb") as f:
291         f.write(img_data)
292 -     print(f"[IMG] Successfully received and saved: {filename}")
292 +     print(f"[IMG] Saved to: {os.path.normpath(filename)}")
293     return filename
294
295     except Exception as e:

```

Abbildung 1.11 Bildpfad Fix

Lösung:

Die Methode `_setup_imagepath()` wandelt den Bildpfad jetzt in einen absoluten, benutzerspezifischen Pfad (z.B. `~/slcp_images`) um und legt das Zielverzeichnis automatisch an. Dadurch werden Bilder plattformunabhängig und zuverlässig gespeichert.

1.1.5.3 Konsistente CLI-Befehle und Hilfeanzeige

Die Kommandozeilenschnittstelle (CLI) war zunächst inkonsistent gestaltet. Es gab sowohl den Befehl `/send` als auch `/msg` und die Argumentbeschreibungen waren unklar (z.B. `<message>` statt `<text>`). Das führte dazu, dass neue Benutzer die richtigen Befehle nicht immer sofort fanden und es zu Bedienungsfehlern kam.

```

Chat/client/interface.py
@@ -10,7 +10,7 @@ def __init__(self, config, messenger):
10 10
11 11     async def run(self):
12 12         print(f"🟢 Willkommen im SLCP-Chat, {self.config.handle}!")
13 12         print("Verfügbare Befehle: /join, /leave, /whois <name>, /send <name> <msg>, /img <name> <pfad>, /quit")
13 13 +         print("Verfügbare Befehle: /join, /leave, /whois <name>, /msg <handle> <text>, /img <handle> <pfad>, /quit")
14 14
15 15         while True:
16 16             try:
@@ -33,7 +33,7 @@ async def run(self):
33 33                 elif command.startswith("/send"):
34 34                     parts = command.split(" ", 2)
35 35                     if len(parts) < 3:
36 36 -                     print("❌ Usage: /send <handle> <message>")
36 36 +                     print("❌ Usage: /msg <handle> <text>")
37 37                 else:
38 38                     await self.messenger.send_message(parts[1], parts[2])
39 39

Chat/network/messenger.py
@@ -42,7 +42,7 @@ async def handle_message(self, message, addr):
42 42
43 43         if parsed["type"] == "JOIN":
44 44             self.peers[parsed["handle"]] = (addr[0], parsed["port"])
45 45 -             print(f"[JOIN] {parsed['handle']} joined from {addr[0]}:{parsed['port']}")
45 45 +             print(f"[JOIN] {parsed['handle']} joined from port {parsed['port']}")
46 46
47 47         elif parsed["type"] == "LEAVE":
48 48             self.peers.pop(parsed["handle"], None)

```

Abbildung 1.12 CLI-Hilfe

Lösung:

Alle Befehle und Hilfetexte wurden vereinheitlicht. Die CLI verwendet jetzt konsistente Benennungen wie `/msg` und eindeutige Argumente, was die Benutzung erleichtert und Missverständnisse minimiert.

1.1.5.4 Flexibler Umgang mit Peer-Daten (JOIN/KNOWNUSERS)

Bei der Verarbeitung von Peer-Informationen (insbesondere bei JOIN/KNOWNUSERS) wurden Einträge starr nach IP und Handle verglichen. Dadurch kam es gelegentlich zu unvollständigen oder doppelten Listen – etwa, wenn ein Nutzer mehrfach im Netzwerk auftauchte oder sich seine IP-Adresse änderte.

```

        ip = addr[0]
        try:
            tcp_port = int(parts[2])
            if not (handle == self.handle and ip in self.get_all_local_ips() and tcp_port == self.port):
                with self.peers_lock:
                    self.peers[handle] = (ip, tcp_port)
        except ValueError:
            print("[Fehler] Ungültiger Port in JOIN.")
    @@ -86,8 +85,15 @@ def handle_message(self, message, addr):
        # Sende bekannte User als KNOWNUSERS zurück
        user_infos = [f"{self.handle} {self.get_local_ip()} {self.port}"]
        with self.peers_lock:
            user_infos = []
            for handle, (ip, port) in self.peers.items():
                user_infos.append(f"{handle} {ip} {port}")
            # Nur hinzufügen, wenn es nicht du selbst bist
            if handle != self.handle or ip != self.get_local_ip() or port != self.port:
                user_infos.append(f"{handle} {ip} {port}")
            # Füge eigene Info genau einmal hinzu - ganz am Schluss oder am Anfang
            user_infos.insert(0, f"{self.handle} {self.get_local_ip()} {self.port}")

        msg = "KNOWNUSERS " + " ".join(user_infos) + "\n"
        self.sock.sendto(msg.encode("utf-8"), addr)

    @@ -100,8 +106,7 @@ def handle_message(self, message, addr):
        infos = entry.strip().split()
        if len(infos) == 3:
            handle, ip, port = infos
            if not (handle == self.handle and ip == self.get_local_ip() and int(port) == self.port):
                self.peers[handle] = (ip, int(port))
            self.peers[handle] = (ip, int(port))
            print(f" - {handle} @ {ip}:{port}")

        def send_who(self):

win.py
    @@ -29,7 +29,7 @@ def my_progress_callback(direction, peer, progress, sent, total):
        messenger.set_message_callback(interface.display_message)
        messenger.set_image_callback(interface.display_image_notice)
        messenger.set_knownusers_callback(interface.display_knownusers)
        messenger.set_knownusers_callback(interface.display_knownusers)
        # UDP-Listener für SLCP starten
        await messenger.start_listener()

```

Abbildung 1.13 Peer-Daten Fix

Lösung:

Die neue Logik sammelt alle Peer-Daten und prüft erst nach dem Sammeln, ob Einträge tatsächlich zur eigenen Instanz gehören. Die eigene Information wird gezielt am Ende der Liste eingefügt, doppelte Einträge werden zuverlässig vermieden.

1.1.5.5 Konsolidierung mehrerer KNOWNUSERS-Antworten

Wenn eine WHO-Anfrage gestellt wurde, konnten mehrere „KNOWNUSERS“-Antworten fast gleichzeitig eintreffen. Der ursprüngliche Code verarbeitete jede Antwort einzeln, wodurch Informationen verloren gehen konnten, wenn Nachrichten schnell hintereinander empfangen wurden.


```

Chat/common/protocol.py
1  @ -4,37 +4,40 @@ def parse_slip(line):
2  4  -
3  5  -
4  6  -
5  7  -
6  8  -
7  9  -
8  10  -
9  11  -
10 12  -
11 13  -
12 14  -
13 15  -
14 16  -
15 17  -
16 18  -
17 19  -
18 20  -
19 21  -
20 22  -
21 23  -
22 24  -
23 25  -
24 26  -
25 27  -
26 28  -
27 29  -
28 30  -
29 31  -
30 32  -
31 33  -
32 34  -
33 35  -
34 36  -
35 37  -
36 38  -
37 39  -
38 40  -
39 41  -
40 42  -
41 43  -
42 44  -
43 45  -
44 46  -
45 47  -
46 48  -
47 49  -
48 50  -
49 51  -
50 52  -
51 53  -
52 54  -
53 55  -
54 56  -
55 57  -
56 58  -
57 59  -
58 60  -
59 61  -
60 62  -
61 63  -
62 64  -
63 65  -
64 66  -
65 67  -
66 68  -
67 69  -
68 70  -
69 71  -
70 72  -
71 73  -
72 74  -
73 75  -
74 76  -
75 77  -
76 78  -
77 79  -
78 80  -
79 81  -
80 82  -
81 83  -
82 84  -
83 85  -
84 86  -
85 87  -
86 88  -
87 89  -
88 90  -
89 91  -
90 92  -
91 93  -
92 94  -
93 95  -
94 96  -
95 97  -
96 98  -
97 99  -
98 100  -
99 101  -
100 102  -
101 103  -
102 104  -
103 105  -
104 106  -
105 107  -
106 108  -
107 109  -
108 110  -
109 111  -
110 112  -
111 113  -
112 114  -
113 115  -
114 116  -
115 117  -
116 118  -
117 119  -
118 120  -
119 121  -
120 122  -
121 123  -
122 124  -
123 125  -
124 126  -
125 127  -
126 128  -
127 129  -
128 130  -
129 131  -
130 132  -
131 133  -
132 134  -
133 135  -
134 136  -
135 137  -
136 138  -
137 139  -
138 140  -
139 141  -
140 142  -
141 143  -
142 144  -
143 145  -
144 146  -
145 147  -
146 148  -
147 149  -
148 150  -
149 151  -
150 152  -
151 153  -
152 154  -
153 155  -
154 156  -
155 157  -
156 158  -
157 159  -
158 160  -
159 161  -
160 162  -
161 163  -
162 164  -
163 165  -
164 166  -
165 167  -
166 168  -
167 169  -
168 170  -
169 171  -
170 172  -
171 173  -
172 174  -
173 175  -
174 176  -
175 177  -
176 178  -
177 179  -
178 180  -
179 181  -
180 182  -
181 183  -
182 184  -
183 185  -
184 186  -
185 187  -
186 188  -
187 189  -
188 190  -
189 191  -
190 192  -
191 193  -
192 194  -
193 195  -
194 196  -
195 197  -
196 198  -
197 199  -
198 200  -
199 201  -
200 202  -
201 203  -
202 204  -
203 205  -
204 206  -
205 207  -
206 208  -
207 209  -
208 210  -
209 211  -
210 212  -
211 213  -
212 214  -
213 215  -
214 216  -
215 217  -
216 218  -
217 219  -
218 220  -
219 221  -
220 222  -
221 223  -
222 224  -
223 225  -
224 226  -
225 227  -
226 228  -
227 229  -
228 230  -
229 231  -
230 232  -
231 233  -
232 234  -
233 235  -
234 236  -
235 237  -
236 238  -
237 239  -
238 240  -
239 241  -
240 242  -
241 243  -
242 244  -
243 245  -
244 246  -
245 247  -
246 248  -
247 249  -
248 250  -
249 251  -
250 252  -
251 253  -
252 254  -
253 255  -
254 256  -
255 257  -
256 258  -
257 259  -
258 260  -
259 261  -
260 262  -
261 263  -
262 264  -
263 265  -
264 266  -
265 267  -
266 268  -
267 269  -
268 270  -
269 271  -
270 272  -
271 273  -
272 274  -
273 275  -
274 276  -
275 277  -
276 278  -
277 279  -
278 280  -
279 281  -
280 282  -
281 283  -
282 284  -
283 285  -
284 286  -
285 287  -
286 288  -
287 289  -
288 290  -
289 291  -
290 292  -
291 293  -
292 294  -
293 295  -
294 296  -
295 297  -
296 298  -
297 299  -
298 300  -
299 301  -
300 302  -
301 303  -
302 304  -
303 305  -
304 306  -
305 307  -
306 308  -
307 309  -
308 310  -
309 311  -
310 312  -
311 313  -
312 314  -
313 315  -
314 316  -
315 317  -
316 318  -
317 319  -
318 320  -
319 321  -
320 322  -
321 323  -
322 324  -
323 325  -
324 326  -
325 327  -
326 328  -
327 329  -
328 330  -
329 331  -
330 332  -
331 333  -
332 334  -
333 335  -
334 336  -
335 337  -
336 338  -
337 339  -
338 340  -
339 341  -
340 342  -
341 343  -
342 344  -
343 345  -
344 346  -
345 347  -
346 348  -
347 349  -
348 350  -
349 351  -
350 352  -
351 353  -
352 354  -
353 355  -
354 356  -
355 357  -
356 358  -
357 359  -
358 360  -
359 361  -
360 362  -
361 363  -
362 364  -
363 365  -
364 366  -
365 367  -
366 368  -
367 369  -
368 370  -
369 371  -
370 372  -
371 373  -
372 374  -
373 375  -
374 376  -
375 377  -
376 378  -
377 379  -
378 380  -
379 381  -
380 382  -
381 383  -
382 384  -
383 385  -
384 386  -
385 387  -
386 388  -
387 389  -
388 390  -
389 391  -
390 392  -
391 393  -
392 394  -
393 395  -
394 396  -
395 397  -
396 398  -
397 399  -
398 400  -
399 401  -
400 402  -
401 403  -
402 404  -
403 405  -
404 406  -
405 407  -
406 408  -
407 409  -
408 410  -
409 411  -
410 412  -
411 413  -
412 414  -
413 415  -
414 416  -
415 417  -
416 418  -
417 419  -
418 420  -
419 421  -
420 422  -
421 423  -
422 424  -
423 425  -
424 426  -
425 427  -
426 428  -
427 429  -
428 430  -
429 431  -
430 432  -
431 433  -
432 434  -
433 435  -
434 436  -
435 437  -
436 438  -
437 439  -
438 440  -
439 441  -
440 442  -
441 443  -
442 444  -
443 445  -
444 446  -
445 447  -
446 448  -
447 449  -
448 450  -
449 451  -
450 452  -
451 453  -
452 454  -
453 455  -
454 456  -
455 457  -
456 458  -
457 459  -
458 460  -
459 461  -
460 462  -
461 463  -
462 464  -
463 465  -
464 466  -
465 467  -
466 468  -
467 469  -
468 470  -
469 471  -
470 472  -
471 473  -
472 474  -
473 475  -
474 476  -
475 477  -
476 478  -
477 479  -
478 480  -
479 481  -
480 482  -
481 483  -
482 484  -
483 485  -
484 486  -
485 487  -
486 488  -
487 489  -
488 490  -
489 491  -
490 492  -
491 493  -
492 494  -
493 495  -
494 496  -
495 497  -
496 498  -
497 499  -
498 500  -
499 501  -
500 502  -
501 503  -
502 504  -
503 505  -
504 506  -
505 507  -
506 508  -
507 509  -
508 510  -
509 511  -
510 512  -
511 513  -
512 514  -
513 515  -
514 516  -
515 517  -
516 518  -
517 519  -
518 520  -
519 521  -
520 522  -
521 523  -
522 524  -
523 525  -
524 526  -
525 527  -
526 528  -
527 529  -
528 530  -
529 531  -
530 532  -
531 533  -
532 534  -
533 535  -
534 536  -
535 537  -
536 538  -
537 539  -
538 540  -
539 541  -
540 542  -
541 543  -
542 544  -
543 545  -
544 546  -
545 547  -
546 548  -
547 549  -
548 550  -
549 551  -
550 552  -
551 553  -
552 554  -
553 555  -
554 556  -
555 557  -
556 558  -
557 559  -
558 560  -
559 561  -
560 562  -
561 563  -
562 564  -
563 565  -
564 566  -
565 567  -
566 568  -
567 569  -
568 570  -
569 571  -
570 572  -
571 573  -
572 574  -
573 575  -
574 576  -
575 577  -
576 578  -
577 579  -
578 580  -
579 581  -
580 582  -
581 583  -
582 584  -
583 585  -
584 586  -
585 587  -
586 588  -
587 589  -
588 590  -
589 591  -
590 592  -
591 593  -
592 594  -
593 595  -
594 596  -
595 597  -
596 598  -
597 599  -
598 600  -
600 601  -
601 602  -
602 603  -
603 604  -
604 605  -
605 606  -
606 607  -
607 608  -
608 609  -
609 610  -
610 611  -
611 612  -
612 613  -
613 614  -
614 615  -
615 616  -
616 617  -
617 618  -
618 619  -
619 620  -
620 621  -
621 622  -
622 623  -
623 624  -
624 625  -
625 626  -
626 627  -
627 628  -
628 629  -
629 630  -
630 631  -
631 632  -
632 633  -
633 634  -
634 635  -
635 636  -
636 637  -
637 638  -
638 639  -
639 640  -
640 641  -
641 642  -
642 643  -
643 644  -
644 645  -
645 646  -
646 647  -
647 648  -
648 649  -
649 650  -
650 651  -
651 652  -
652 653  -
653 654  -
654 655  -
655 656  -
656 657  -
657 658  -
658 659  -
659 660  -
660 661  -
661 662  -
662 663  -
663 664  -
664 665  -
665 666  -
666 667  -
667 668  -
668 669  -
669 670  -
670 671  -
671 672  -
672 673  -
673 674  -
674 675  -
675 676  -
676 677  -
677 678  -
678 679  -
679 680  -
680 681  -
681 682  -
682 683  -
683 684  -
684 685  -
685 686  -
686 687  -
687 688  -
688 689  -
689 690  -
690 691  -
691 692  -
692 693  -
693 694  -
694 695  -
695 696  -
696 697  -
697 698  -
698 699  -
699 700  -
700 701  -
701 702  -
702 703  -
703 704  -
704 705  -
705 706  -
706 707  -
707 708  -
708 709  -
709 710  -
710 711  -
711 712  -
712 713  -
713 714  -
714 715  -
715 716  -
716 717  -
717 718  -
718 719  -
719 720  -
720 721  -
721 722  -
722 723  -
723 724  -
724 725  -
725 726  -
726 727  -
727 728  -
728 729  -
729 730  -
730 731  -
731 732  -
732 733  -
733 734  -
734 735  -
735 736  -
736 737  -
737 738  -
738 739  -
739 740  -
740 741  -
741 742  -
742 743  -
743 744  -
744 745  -
745 746  -
746 747  -
747 748  -
748 749  -
749 750  -
750 751  -
751 752  -
752 753  -
753 754  -
754 755  -
755 756  -
756 757  -
757 758  -
758 759  -
759 760  -
760 761  -
761 762  -
762 763  -
763 764  -
764 765  -
765 766  -
766 767  -
767 768  -
768 769  -
769 770  -
770 771  -
771 772  -
772 773  -
773 774  -
774 775  -
775 776  -
776 777  -
777 778  -
778 779  -
779 780  -
780 781  -
781 782  -
782 783  -
783 784  -
784 785  -
785 786  -
786 787  -
787 788  -
788 789  -
789 790  -
790 791  -
791 792  -
792 793  -
793 794  -
794 795  -
795 796  -
796 797  -
797 798  -
798 799  -
799 800  -
800 801  -
801 802  -
802 803  -
803 804  -
804 805  -
805 806  -
806 807  -
807 808  -
808 809  -
809 810  -
810 811  -
811 812  -
812 813  -
813 814  -
814 815  -
815 816  -
816 817  -
817 818  -
818 819  -
819 820  -
820 821  -
821 822  -
822 823  -
823 824  -
824 825  -
825 826  -
826 827  -
827 828  -
828 829  -
829 830  -
830 831  -
831 832  -
832 833  -
833 834  -
834 835  -
835 836  -
836 837  -
837 838  -
838 839  -
839 840  -
840 841  -
841 842  -
842 843  -
843 844  -
844 845  -
845 846  -
846 847  -
847 848  -
848 849  -
849 850  -
850 851  -
851 852  -
852 853  -
853 854  -
854 855  -
855 856  -
856 857  -
857 858  -
858 859  -
859 860  -
860 861  -
861 862  -
862 863  -
863 864  -
864 865  -
865 866  -
866 867  -
867 868  -
868 869  -
869 870  -
870 871  -
871 872  -
872 873  -
873 874  -
874 875  -
875 876  -
876 877  -
877 878  -
878 879  -
879 880  -
880 881  -
881 882  -
882 883  -
883 884  -
884 885  -
885 886  -
886 887  -
887 888  -
888 889  -
889 890  -
890 891  -
891 892  -
892 893  -
893 894  -
894 895  -
895 896  -
896 897  -
897 898  -
898 899  -
899 900  -
900 901  -
901 902  -
902 903  -
903 904  -
904 905  -
905 906  -
906 907  -
907 908  -
908 909  -
909 910  -
910 911  -
911 912  -
912 913  -
913 914  -
914 915  -
915 916  -
916 917  -
917 918  -
918 919  -
919 920  -
920 921  -
921 922  -
922 923  -
923 924  -
924 925  -
925 926  -
926 927  -
927 928  -
928 929  -
929 930  -
930 931  -
931 932  -
932 933  -
933 934  -
934 935  -
935 936  -
936 937  -
937 938  -
938 939  -
939 940  -
940 941  -
941 942  -
942 943  -
943 944  -
944 945  -
945 946  -
946 947  -
947 948  -
948 949  -
949 950  -
950 951  -
951 952  -
952 953  -
953 954  -
954 955  -
955 956  -
956 957  -
957 958  -
958 959  -
959 960  -
960 961  -
961 962  -
962 963  -
963 964  -
964 965  -
965 966  -
966 967  -
967 968  -
968 969  -
969 970  -
970 971  -
971 972  -
972 973  -
973 974  -
974 975  -
975 976  -
976 977  -
977 978  -
978 979  -
979 980  -
980 981  -
981 982  -
982 983  -
983 984  -
984 985  -
985 986  -
986 987  -
987 988  -
988 989  -
989 990  -
990 991  -
991 992  -
992 993  -
993 994  -
994 995  -
995 996  -
996 997  -
997 998  -
998 999  -
999 1000  -
1000 1001  -
1001 1002  -
1002 1003  -
1003 1004  -
1004 1005  -
1005 1006  -
1006 1007  -
1007 1008  -
1008 1009  -
1009 1010  -
1010 1011  -
1011 1012  -
1012 1013  -
1013 1014  -
1014 1015  -
1015 1016  -
1016 1017  -
1017 1018  -
1018 1019  -
1019 1020  -
1020 1021  -
1021 1022  -
1022 1023  -
1023 1024  -
1024 1025  -
1025 1026  -
1026 1027  -
1027 1028  -
1028 1029  -
1029 1030  -
1030 1031  -
1031 1032  -
1032 1033  -
1033 1034  -
1034 1035  -
1035 1036  -
1036 1037  -
1037 1038  -
1038 1039  -
1039 1040  -
1040 1041  -
1041 1042  -
1042 1043  -
1043 1044  -
1044 1045  -
1045 1046  -
1046 1047  -
1047 1048  -
1048 1049  -
1049 1050  -
1050 1051  -
1051 1052  -
1052 1053  -
1053 1054  -
1054 1055  -
1055 1056  -
1056 1057  -
1057 1058  -
1058 1059  -
1059 1060  -
1060 1061  -
1061 1062  -
1062 1063  -
1063 1064  -
1064 1065  -
1065 1066  -
1066 1067  -
1067 1068  -
1068 1069  -
1069 1070  -
1070 1071  -
1071 1072  -
1072 1073  -
1073 1074  -
1074 1075  -
1075 1076  -
1076 1077  -
1077 1078  -
1078 1079  -
1079 1080  -
1080 1081  -
1081 1082  -
1082 1083  -
1083 1084  -
1084 1085  -
1085 1086  -
1086 1087  -
1087 1088  -
1088 1089  -
1089 1090  -
1090 1091  -
1091 1092  -
1092 1093  -
1093 1094  -
1094 1095  -
1095 1096  -
1096 1097  -
1097 1098  -
1098 1099  -
1099 1100  -
1100 1101  -
1101 1102  -
1102 1103  -
1103 1104  -
1104 1105  -
1105 1106  -
1106 1107  -
1107 1108  -
1108 1109  -
1109 1110  -
1110 1111  -
1111 1112  -
1112 1113  -
1113 1114  -
1114 1115  -
1115 1116  -
1116 1117  -
1117 1118  -
1118 1119  -
1119 1120  -
1120 1121  -
1121 1122  -
1122 1123  -
1123 1124  -
1124 1125  -
1125 1126  -
1126 1127  -
1127 1128  -
1128 1129  -
1129 1130  -
1130 1131  -
1131 1132  -
1132 1133  -
1133 1134  -
1134 1135  -
1135 1136  -
1136 1137  -
1137 1138  -
1138 1139  -
1139 1140  -
1140 1141  -
1141 1142  -
1142 1143  -
1143 1144  -
1144 1145  -
1145 1146  -
1146 1147  -
1147 1148  -
1148 1149  -
1149 1150  -
1150 1151  -
1151 1152  -
1152 1153  -
1153 1154  -
1154 1155  -
1155 1156  -
1156 1157  -
1157 1158  -
1158 1159  -
1159 1160  -
1160 1161  -
1161 1162  -
1162 1163  -
1163 1164  -
1164 1165  -
1165 1166  -
1166 1167  -
1167 1168  -
1168 1169  -
1169 1170  -
1170 1171  -
1171 1172  -
1172 1173  -
1173 1174  -
1174 1175  -
1175 1176  -
1176 1177  -
1177 1178  -
1178 1179  -
1179 1180  -
1180 1181  -
1181 1182  -
1182 1183  -
1183 1184  -
1184 1185  -
1185 1186  -
1186 1187  -
1187 1188  -
1188 1189  -
1189 1190  -
1190 1191  -
1191 1192  -
1192 1193  -
1193 1194  -
1194 1195  -
1195 1196  -
1196 1197  -
1197 1198  -
1198 1199  -
1199 1200  -
1200 1201  -
1201 1202  -
1202 1203  -
1203 1204  -
1204 1205  -
1205 1206  -
1206 1207  -
1
```

```

11 +         elif cmd == "LEAVE" and len(parts) == 2:
12 +             return {"type": "LEAVE", "handle": parts[1]}
13 +
14 +         elif cmd == "WHO" and len(parts) == 1:
15 +             return {"type": "WHO"}
16 +
17 +         elif cmd == "MSG" and len(parts) >= 3:
18 +             to = parts[1]
19 +             text = " ".join(parts[2:]).strip("")
20 +             return {"type": "MSG", "to": to, "message": text}
21 +
22 +         elif cmd == "IMG" and len(parts) == 3:
23 +             return {"type": "IMG", "to": parts[1], "size": int(parts[2])}
24 +
25 +         elif cmd == "KNOWNUSERS" and len(parts) >= 1:
26 +             users = []
27 +             user_data = " ".join(parts[1:]).split(',')
28 +             for entry in user_data:
29 +                 entry = entry.strip()
30 +                 if entry:
31 +                     user_parts = entry.split()
32 +                     if len(user_parts) == 3:
33 +                         users.append({
34 +                             "handle": user_parts[0],
35 +                             "ip": user_parts[1],
36 +                             "port": int(user_parts[2])})
37 +             return {"type": "KNOWNUSERS", "users": users}
38 +
39 +     except (ValueError, IndexError) as e:
40 +         print(f"Parse error: {e} for line: {line}")
41 +
42 +     return {"type": "UNKNOWN", "raw": line}
43 +
44 + @ -55,9 +58,12 @@ def create_img(target, size):
45 +     return f"IMG {target} {size}\n"
46 +
47 +
48 +
49 +
50 +
51 +
52 +
53 +
54 +
55 +
56 +
57 +
58 + - def create_whois(handle):
59 + -     return f"WHOIS {handle}\n"
60 +
61 + + def create_who():
62 + +     return f"WHO\n"
63 +
64 +
65 +
66 +
67 +
68 +
69 +
70 +
71 +
72 +
73 +
74 +
75 +
76 +
77 +
78 +
79 +
80 +
81 +
82 +
83 +
84 +
85 +
86 +
87 +
88 +
89 +
90 +
91 +
92 +
93 +
94 +
95 +
96 +
97 +
98 +
99 +
100 +

```

Abbildung 1.15 Knownusers Konsolidierung – nachher

Lösung:

Die neue Funktion `handle_knownusers_response` sammelt alle Antworten für eine kurze Zeit, konsolidiert die erhaltenen Daten und übergibt die vollständige Liste dann gesammelt an das Interface.

1.1.5.6 Verbesserte Anzeige des Absenders

Beim Anzeigen eingehender Nachrichten wurde bisher oft nur die rohe IP-Adresse des Absenders angezeigt, selbst wenn dessen Handle bekannt war. Das erschwerte die Zuordnung und war wenig benutzerfreundlich.

```

Chat/network/messenger.py
82 82         if parsed["to"] == self.config.handle:
83 83             msg = parsed["message"]
84 84             sender = addr[0]
85 +             sender_handle = None
86 +             for handle, (ip, _) in self.peers.items():
87 +                 if ip == sender:
88 +                     sender_handle = handle
89 +                     break
90 +             sender_display = sender_handle if sender_handle else f"Unbekannt ({sender})"
85 91         if self.message_callback:
86 -             await self.message_callback(sender, msg)
92 +             await self.message_callback(sender_display, msg)
87 93         else:
88 -             print(f"[{sender}] {msg}")
94 +             print(f"\n📢 Nachricht von {sender_display}: {msg}")
89 95         if self.config.autoreply:
90 -             await self.send_message(sender, self.config.autoreply)
96 +             await self.send_message(sender_display, self.config.autoreply)
91 97
92 98         elif parsed["type"] == "IMG":
93 99             if parsed["to"] == self.config.handle:
127 133             if handle not in self.peers:
128 134                 print(f"[Error] No known peer with handle '{handle}'")
129 135                 return
130 -             msg = protocol.create_msg(handle, message)
131 -             ip, port = self.peers[handle]
132 -             await self.send_slcp(msg, ip, port)
136 +             if handle in self.peers:
137 +                 ip, port = self.peers[handle]
138 +                 msg = protocol.create_msg(handle, message)
139 +                 await self.send_slcp(msg, ip, port)
140 +             else:
141 +                 print(f"[Error] Handle '{handle}' not connected")
133 142
134 143         async def send_image(self, handle, filepath):
135 144             if handle not in self.peers:

```

Abbildung 1.16 Absenderanzeige

Lösung:

Jetzt wird – sofern möglich – immer der Benutzername (Handle) angezeigt. Ist dieser nicht bekannt, erscheint stattdessen „Unbekannt“.

1.1.5.7 Doppelte Einträge in WHO/KNOWNUSERS-Liste

Die Peer-Liste enthielt manchmal doppelte Einträge, da bei jedem Empfang der eigenen Information diese erneut angehängt wurde. Das sorgte für Unübersichtlichkeit in der Nutzeranzeige.

```

Chat/discovery/discovery_service.py
@@ -83,16 +83,20 @@ def handle_message(self, message, addr):
83 83
84 84         elif cmd == "WHO" and len(parts) == 1:
85 85             # Sende bekannte User als KNOMNUSERS zurück
86 -         user_infos = [f"{self.handle} {self.get_local_ip()} {self.port}"]
87 86         with self.peers_lock:
87 +             seen = set()
88 88             user_infos = []
89 -         for handle, (ip, port) in self.peers.items():
90 -             # Nur hinzufügen, wenn es nicht du selbst bist
91 -             if handle != self.handle or ip != self.get_local_ip() or port != self.port:
92 -                 user_infos.append(f"{handle} {ip} {port}")
93 89
94 -         # Füge eigene info genau einmal hinzu - ganz am Schluss oder am Anfang
95 -         user_infos.insert(0, f"{self.handle} {self.get_local_ip()} {self.port}")
96 +         for handle, (ip, port) in self.peers.items():
97 +             entry = f"{handle} {ip} {port}"
98 +             if entry not in seen and (handle != self.handle or ip != self.get_local_ip() or port != self.port):
99 +                 user_infos.append(entry)
100 +                 seen.add(entry)
101 +
102 +             # Eigene info hinzufügen, wenn sie noch nicht enthalten ist
103 +             self_info = f"{self.handle} {self.get_local_ip()} {self.port}"
104 +             if self_info not in seen:
105 +                 user_infos.append(self_info)
106 100
107 101         msg = "KNOMNUSERS " + " ".join(user_infos) + "\n"
108 102         self.sock.sendto(msg.encode("utf-8"), addr)

```

Abbildung 1.17 Peerlist Duplikate

Lösung:

Mithilfe einer sogenannten „seen“-Menge werden doppelte Einträge beim Sammeln gezielt vermieden.

Weitere Details und vollständige Screenshots siehe:

[Screenshots mit Fehleranalyse](#)

1.1.6 Bedienung

- **Start:**

```
```bash python main.py
```

## Kapitel 2

# Verzeichnis der Namensbereiche

### 2.1 Liste aller Namensbereiche

Liste aller dokumentierten Namensbereiche mit Kurzbeschreibung:

<a href="#">config.config</a>	25
<a href="#">discovery.discovery_service</a>	25
<a href="#">interface</a>	26
<a href="#">main</a>	26
<a href="#">network.messenger</a>	26



## Kapitel 3

# Hierarchie-Verzeichnis

### 3.1 Klassenhierarchie

Die Liste der Ableitungen ist, mit Einschränkungen, alphabetisch sortiert:

config.config.Config . . . . .	27
asyncio.DatagramProtocol	
network.messenger.Messenger . . . . .	32
discovery.discovery_service.DiscoveryService . . . . .	28
interface.Interface . . . . .	30





## Kapitel 4

# Klassen-Verzeichnis

### 4.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

<a href="#">config.config.Config</a> . . . . .	27
<a href="#">discovery.discovery_service.DiscoveryService</a> . . . . .	28
<a href="#">interface.Interface</a> . . . . .	30
<a href="#">network.messenger.Messenger</a> . . . . .	32



# Kapitel 5

## Datei-Verzeichnis

### 5.1 Auflistung der Dateien

Hier folgt die Aufzählung aller dokumentierten Dateien mit einer Kurzbeschreibung:

Chat/common/[protocol.py](#)

Enthält Funktionen zum Parsen und Erzeugen von SLCP-Protokollnachrichten . . . . . 39



# Kapitel 6

## Dokumentation der Namensbereiche

### 6.1 config.config-Namensbereichsreferenz

#### Klassen

- class [Config](#)

#### 6.1.1 Ausführliche Beschreibung

```
@file config.py
@brief Klasse zur Verwaltung und Speicherung der Konfiguration des Chat-Clients (Laden/Speichern von slcp_conf
```

### 6.2 discovery.discovery\_service-Namensbereichsreferenz

#### Klassen

- class [DiscoveryService](#)

#### Funktionen

- bool [is\\_port\\_in\\_use](#) (int port)

#### Variablen

- int **BROADCAST\_PORT** = 4000
- int **BUFFER\_SIZE** = 1024
- **service** = [DiscoveryService](#)("slcp\_config.toml")
- **peers** = service.get\_peers()

#### 6.2.1 Ausführliche Beschreibung

```
@file discovery_service.py
@brief Discovery-Service für den P2P-Chat. Verwaltet Peer-Discovery, JOIN/LEAVE/WHO Broadcast, etc.
```

#### 6.2.2 Dokumentation der Funktionen

##### 6.2.2.1 is\_port\_in\_use()

```
bool discovery.discovery_service.is_port_in_use (
 int port)
```

```
@brief Prüft, ob ein TCP-Port bereits belegt ist.
```

```
@param port TCP-Portnummer, die geprüft werden soll
@return True, wenn der Port belegt ist, sonst False
```

## 6.3 interface-Namensbereichsreferenz

### Klassen

- class [Interface](#)

### 6.3.1 Ausführliche Beschreibung

```
@file interface.py
@brief Kommandozeilen-Benutzeroberfläche (CLI) für den SLCP-Chat-Client.
Stellt Eingabe, Ausgabe und Nutzerinteraktion über Terminal bereit.
```

## 6.4 main-Namensbereichsreferenz

### Funktionen

- [main\(\)](#)

### 6.4.1 Ausführliche Beschreibung

```
@file main.py
@brief Hauptprogramm des P2P-Chat-Clients. Startet Discovery, Messenger, Interface (CLI).
```

### 6.4.2 Dokumentation der Funktionen

#### 6.4.2.1 main()

```
main.main ()
```

```
@brief Hauptfunktion des SLCP-Clients.
```

```
Initialisiert alle Hauptkomponenten des Systems:
- Lädt Konfiguration
- Startet den Discovery-Dienst (UDP-basiert)
- Initialisiert die Messenger-Komponente (SLCP)
- Setzt Callback-Funktionen
- Startet die Benutzeroberfläche (Kommandozeile)
- Öffnet den UDP-Listener (für asynchrone Nachrichtenannahme)
```

Ablauf:

1. Lade Einstellungen wie Handle und Port aus TOML-Datei.
2. Starte Discovery-Service für JOIN/LEAVE/WHO per Broadcast.
3. Starte SLCP-Messenger für Nachrichten- und Bildversand.
4. Starte CLI für Benutzereingaben.

```
@return None
```

## 6.5 network.messenger-Namensbereichsreferenz

### Klassen

- class [Messenger](#)

### 6.5.1 Ausführliche Beschreibung

```
@file messenger.py
@brief Messenger-Klasse für das SLCP-Chat-Projekt.
@details
Diese Klasse implementiert die UDP- und TCP-Kommunikation für den Austausch von Nachrichten und Bildern im dezentralen Chat. Sie verwaltet Peers, verarbeitet SLCP-Nachrichten und stellt Methoden zum Senden und Empfangen bereit.
```

# Kapitel 7

## Klassen-Dokumentation

### 7.1 config.config.Config Klassenreferenz

#### Öffentliche Methoden

- `__init__` (self, path="slcp\_config.toml")
- `load` (self)
- `save` (self)

#### Öffentliche Attribute

- `path` = path
- `data` = self.load()
- `handle` = self.data.get("handle")
- `port` = int(self.data.get("port") or input("Port: "))
- `whoisport` = int(self.data.get("whoisport") or input("Whois-Port (z.B. 4000): "))
- `autoreply` = self.data.get("autoreply", "")
- `imagepath` = self.\_setup\_imagepath()

#### Geschützte Methoden

- `_setup_imagepath` (self)

#### 7.1.1 Ausführliche Beschreibung

```
@class Config
@brief Verwaltet die Konfiguration des Chat-Clients.
```

Diese Klasse lädt, speichert und verwaltet Benutzereinstellungen aus einer TOML-Datei.  
Falls Werte fehlen (z.B. Benutzername oder Port), werden sie beim ersten Start interaktiv abgefragt.

#### 7.1.2 Beschreibung der Konstruktoren und Destruktoren

##### 7.1.2.1 \_\_init\_\_()

```
config.config.Config.__init__ (
 self,
 path = "slcp_config.toml")
```

```
@brief Initialisiert eine neue Konfigurationsinstanz.
```

- Liest Konfiguration aus Datei (sofern vorhanden).
- Fragt fehlende Parameter interaktiv ab.
- Erstellt Standardverzeichnisse bei Bedarf.

```
@param path Dateipfad zur TOML-Konfigurationsdatei (Standard: slcp_config.toml)
```

### 7.1.3 Dokumentation der Elementfunktionen

#### 7.1.3.1 `_setup_imagepath()`

```
config.config.Config._setup_imagepath (
 self) [protected]
```

@brief Bereitet den Ordnerpfad für empfangene Bilder vor.

- Nutzt Standardpfad, wenn nicht angegeben.
- Erstellt das Verzeichnis bei Bedarf.

@return Absoluter Pfad zum Bildspeicherordner (z.B. ~/slcp\_images)

#### 7.1.3.2 `load()`

```
config.config.Config.load (
 self)
```

@brief Lädt die Konfiguration aus der TOML-Datei.

@return Dictionary mit den geladenen Konfigurationswerten.

#### 7.1.3.3 `save()`

```
config.config.Config.save (
 self)
```

@brief Speichert die aktuelle Konfiguration zurück in die TOML-Datei.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- Chat/config/config.py

## 7.2 `discovery.discovery_service.DiscoveryService` Klassenreferenz

### Öffentliche Methoden

- `__init__` (self, config\_path)
- `listen` (self)
- `handle_message` (self, message, addr)
- `send_who` (self)
- `get_local_ip` (self)
- `send_join` (self)
- `send_leave` (self)
- `get_peers` (self)
- `start` (self)
- `stop` (self)

### Öffentliche, statische Methoden

- `load_config` (path)

### Öffentliche Attribute

- dict `peers` = {}
- `peers_lock` = threading.Lock()
- bool `running` = True
- `config` = self.load\_config(config\_path)
- `handle` = self.config["handle"]
- `port` = int(self.config["port"])
- `whois_port` = self.config.get("whoisport", 0)
- `sock` = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)



## 7.2.1 Ausführliche Beschreibung

```
@class DiscoveryService
@brief Implementiert den Discovery-Dienst für das dezentrale Chat-System.

Verwaltet bekannte Peers, sendet und empfängt UDP Broadcast-Nachrichten.
```

## 7.2.2 Beschreibung der Konstruktoren und Destruktoren

### 7.2.2.1 \_\_init\_\_()

```
discovery.discovery_service.DiscoveryService.__init__ (
 self,
 config_path)

@brief Konstruktor für den DiscoveryService.

Lädt Konfiguration, initialisiert Variablen und bindet UDP Socket.

@param config_path Pfad zur TOML-Konfigurationsdatei
```

## 7.2.3 Dokumentation der Elementfunktionen

### 7.2.3.1 get\_local\_ip()

```
discovery.discovery_service.DiscoveryService.get_local_ip (
 self)

@brief Ermittelt die lokale IP-Adresse des Hosts.

@return String mit der lokalen IP-Adresse
```

### 7.2.3.2 get\_peers()

```
discovery.discovery_service.DiscoveryService.get_peers (
 self)

@brief Gibt eine Kopie der aktuellen Peer-Liste zurück.

@return Dictionary mit Peers {handle: (IP, Port)}
```

### 7.2.3.3 handle\_message()

```
discovery.discovery_service.DiscoveryService.handle_message (
 self,
 message,
 addr)

@brief Verarbeitet eingehende UDP-Nachrichten gemäß Protokoll.

@param message Empfangene Nachricht als String
@param addr Adresse des Senders (IP, Port)
```

### 7.2.3.4 listen()

```
discovery.discovery_service.DiscoveryService.listen (
 self)

@brief Endlosschleife zum Empfangen und Verarbeiten von UDP-Nachrichten.

Läuft in eigenem Thread, verarbeitet JOIN, LEAVE, WHO und KNOWNUSERS Nachrichten.
```

### 7.2.3.5 load\_config()

```
discovery.discovery_service.DiscoveryService.load_config (
 path) [static]

@brief Lädt eine TOML-Konfigurationsdatei.

@param path Pfad zur TOML-Datei
@return Dictionary mit Konfigurationsdaten oder leeres Dict bei Fehler
```

### 7.2.3.6 send\_join()

```
discovery.discovery_service.DiscoveryService.send_join (
 self)

@brief Sendet eine JOIN-Nachricht als Broadcast, um sich anzumelden.
```

### 7.2.3.7 send\_leave()

```
discovery.discovery_service.DiscoveryService.send_leave (
 self)

@brief Sendet eine LEAVE-Nachricht als Broadcast, um sich abzumelden.
```

### 7.2.3.8 send\_who()

```
discovery.discovery_service.DiscoveryService.send_who (
 self)

@brief Sendet eine WHO-Anfrage als UDP-Broadcast, um bekannte Peers abzufragen.
```

### 7.2.3.9 start()

```
discovery.discovery_service.DiscoveryService.start (
 self)

@brief Startet den Discovery-Service: Listener-Thread, JOIN und WHO senden.
```

### 7.2.3.10 stop()

```
discovery.discovery_service.DiscoveryService.stop (
 self)

@brief Stoppt den Discovery-Service, sendet LEAVE und schließt das Socket.
```

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- Chat/discovery/discovery\_service.py

## 7.3 interface.Interface Klassenreferenz

### Öffentliche Methoden

- [\\_\\_init\\_\\_](#) (self, config, messenger)
- [run](#) (self)
- [display\\_message](#) (self, sender\_display, message)
- [display\\_image\\_notice](#) (self, sender, filename)
- [display\\_knownusers](#) (self, user\_list)

## Öffentliche Attribute

- **config** = config
- **messenger** = messenger

### 7.3.1 Ausführliche Beschreibung

```
@class Interface
@brief CLI-basierte Benutzeroberfläche für den SLCP-Chat-Client.
```

Diese Klasse stellt die Interaktion des Nutzers mit dem SLCP-Chat über ein Konsoleninterface bereit. Sie verwaltet alle Benutzereingaben, interpretiert SLCP-Befehle wie /join, /leave, /msg usw. und leitet sie asynchron an die Messenger-Komponente zur Verarbeitung weiter.

Zusätzlich wird die farbliche Konsolenausgabe mit dem Modul 'colorama' unterstützt, um Statusnachrichten und Befehle übersichtlicher darzustellen.

### 7.3.2 Beschreibung der Konstruktoren und Destruktoren

#### 7.3.2.1 \_\_init\_\_()

```
interface.Interface.__init__ (
 self,
 config,
 messenger)

@brief Konstruktor der Interface-Klasse.

Initialisiert das Interface mit Konfigurationsdaten und der Messenger-Instanz
zur Netzwerkkommunikation.

@param config Ein Konfigurationsobjekt mit Nutzernamen, Port, Autoreply etc.
@param messenger Eine Messenger-Instanz, die SLCP-Nachrichten verarbeitet und verschickt
```

### 7.3.3 Dokumentation der Elementfunktionen

#### 7.3.3.1 display\_image\_notice()

```
interface.Interface.display_image_notice (
 self,
 sender,
 filename)

@brief Zeigt eine Benachrichtigung über ein empfangenes Bild an.

Diese Methode wird als Callback bei erfolgreichem Empfang eines Bildes aufgerufen.

@param sender Handle oder IP des Absenders
@param filename Pfad zur lokal gespeicherten Bilddatei
```

#### 7.3.3.2 display\_knownusers()

```
interface.Interface.display_knownusers (
 self,
 user_list)

@brief Gibt alle bekannten/erkannten Nutzer formatiert auf der Konsole aus.

Diese Methode wird nach Empfang einer KNOWNUSERS-Nachricht genutzt, um
alle bekannten Peers in der Benutzeroberfläche anzuzeigen.

@param user_list Liste von Tupeln: (handle, ip, port)
```

### 7.3.3.3 display\_message()

```
interface.Interface.display_message (
 self,
 sender_display,
 message)
```

@brief Zeigt eine empfangene Textnachricht in der Konsole an.

Wird vom Messenger aufgerufen, wenn eine neue Nachricht vom Netzwerk empfangen wurde.

@param sender\_display Der Anzeigename oder die IP-Adresse des Absenders

@param message Die empfangene Textnachricht

### 7.3.3.4 run()

```
interface.Interface.run (
 self)
```

@brief Startet die Haupt-Eingabeschleife für den Nutzer.

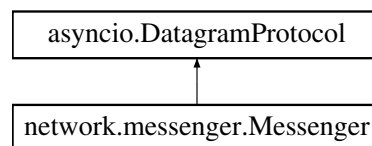
Die Methode zeigt verfügbare Befehle an, liest Eingaben von der Konsole (z.B. /join, /msg, /img), prüft diese auf Gültigkeit und ruft entsprechende Messenger-Methoden zur Verarbeitung auf. Sie läuft bis der Befehl /quit ausgeführt wird.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- Chat/client/interface.py

## 7.4 network.messenger.Messenger Klassenreferenz

Klassendiagramm für network.messenger.Messenger:



### Öffentliche Methoden

- `__init__` (self, config)
- `start_listener` (self)
- `connection_made` (self, transport)
- `datagram_received` (self, data, addr)
- `handle_message` (self, message, addr)
- `send_slcp` (self, line, ip, port)
- `send_broadcast` (self, line)
- `send_join` (self)
- `send_leave` (self)
- `send_who` (self)
- `send_message` (self, handle, message)
- `send_image` (self, handle, filepath)
- `send_image_data` (self, tcp\_socket, img\_bytes, handle)
- `start_tcp_server` (self)
- `handle_tcp_connection` (self, reader, writer)
- `receive_image_data` (self, reader, addr, size, sender\_handle)
- `set_progress_callback` (self, callback)
- `set_message_callback` (self, callback)

- `set_image_callback` (self, callback)
- `set_knownusers_callback` (self, callback)
- `handle_knownusers_response` (self, message, addr)
- `send_known_to` (self, ip, port)
- `get_local_ip` (self)

#### Öffentliche Attribute

- `config` = config
- dict `peers` = {}
- `transport` = None
- `message_callback` = None
- `image_callback` = None
- `knownusers_callback` = None
- `progress_callback` = None
- dict `pending_who_responses` = {}
- float `who_timeout` = 2.0

### 7.4.1 Ausführliche Beschreibung

```
@class Messenger
@brief Messenger für den Versand und Empfang von Chat-Nachrichten und Bildern.
@details
 Verwaltet alle UDP- und TCP-Kommunikationsprozesse, Peer-Liste, sowie das Senden und Empfangen von Nachrichten.
 Stellt zudem verschiedene Callback-Funktionen für die Interaktion mit der Benutzeroberfläche bereit.
```

### 7.4.2 Beschreibung der Konstruktoren und Destruktoren

#### 7.4.2.1 `__init__()`

```
network.messenger.Messenger.__init__ (
 self,
 config)

@brief Konstruktor der Messenger-Klasse
@param config Konfigurationsobjekt mit folgenden Attributen:
 - handle: Benutzername
 - port: Port für TCP/UDP Kommunikation
 - whoisport: Port für WHO-Broadcasts
 - imagepath: Pfad zum Speichern empfangener Bilder
 - autoreply: Automatische Antwort (optional)
```

### 7.4.3 Dokumentation der Elementfunktionen

#### 7.4.3.1 `connection_made()`

```
network.messenger.Messenger.connection_made (
 self,
 transport)

@brief Wird aufgerufen, wenn die UDP-Verbindung erfolgreich hergestellt wurde.
@param transport Das UDP-Transport-Objekt
```

#### 7.4.3.2 `datagram_received()`

```
network.messenger.Messenger.datagram_received (
 self,
 data,
 addr)

@brief Wird aufgerufen, wenn eine UDP-Nachricht empfangen wird.
@param data Empfangene Bytes (Nachricht)
@param addr Adresse des Absenders als Tupel (IP, Port)
```

#### 7.4.3.3 get\_local\_ip()

```
network.messenger.Messenger.get_local_ip (
 self)
```

@brief Ermittelt die lokale IP-Adresse.  
@return Die lokale IP-Adresse als String

#### 7.4.3.4 handle\_knownusers\_response()

```
network.messenger.Messenger.handle_knownusers_response (
 self,
 message,
 addr)
```

@brief Verarbeitet KNOWNUSERS-Antworten.  
@param message Die vollständige KNOWNUSERS-Nachricht  
@param addr Absender-Adresse als (ip, port) Tupel  
@details Parst die Benutzerliste, aktualisiert die Peer-Informationen  
und ruft den entsprechenden Callback auf. Sammelt Antworten  
für eine kurze Zeit um mehrfache Antworten zu konsolidieren.

#### 7.4.3.5 handle\_message()

```
network.messenger.Messenger.handle_message (
 self,
 message,
 addr)
```

@brief Verarbeitet empfangene SLCP-Nachrichten.  
@param message Die dekodierte Nachricht als String  
@param addr Absender-Adresse als (ip, port) Tupel  
@details Parst SLCP-Befehle und führt entsprechende Aktionen aus:

- JOIN: Neuen Peer registrieren
- LEAVE: Peer entfernen
- WHO: Bekannte Benutzer senden
- KNOWNUSERS: Benutzerliste verarbeiten
- MSG: Private Nachricht empfangen
- IMG: Bildübertragung initialisieren

#### 7.4.3.6 handle\_tcp\_connection()

```
network.messenger.Messenger.handle_tcp_connection (
 self,
 reader,
 writer)
```

@brief Behandelt eingehende TCP-Verbindungen für Bildempfang.  
@param reader StreamReader für eingehende Daten  
@param writer StreamWriter für ausgehende Daten  
@details Liest IMG-Befehle und empfängt Bilddaten, speichert diese  
lokal und ruft Image-Callbacks auf.

#### 7.4.3.7 receive\_image\_data()

```
network.messenger.Messenger.receive_image_data (
 self,
 reader,
 addr,
 size,
 sender_handle)
```

```

@brief Empfängt Bilddaten über TCP und speichert sie.
@param reader StreamReader für die Datenübertragung
@param addr Absender-Adresse als (ip, port) Tupel
@param size Erwartete Dateigröße in Bytes
@param sender_handle Benutzername des Absenders
@return Dateiname der gespeicherten Datei oder None bei Fehler
@details Empfängt Daten in Chunks, zeigt Fortschritt an und speichert
 das Bild mit einem eindeutigen Dateinamen.

```

#### 7.4.3.8 send\_broadcast()

```

network.messenger.Messenger.send_broadcast (
 self,
 line)

@brief Sendet eine SLCP-Broadcast-Nachricht an alle Teilnehmer im lokalen Netzwerk.
@param line Die zu sendende SLCP-Nachricht (String)

```

#### 7.4.3.9 send\_image()

```

network.messenger.Messenger.send_image (
 self,
 handle,
 filepath)

@brief Sendet ein Bild an einen bestimmten Peer via TCP.
@param handle Der Ziel-Benutzername
@param filepath Pfad zur Bilddatei
@return True bei Erfolg, False bei Fehler
@details Überprüft die Datei auf Gültigkeit, öffnet eine TCP-Verbindung
 zum Ziel-Peer und überträgt das Bild in Chunks.

```

#### 7.4.3.10 send\_image\_data()

```

network.messenger.Messenger.send_image_data (
 self,
 tcp_socket,
 img_bytes,
 handle)

@brief Sendet die Binärdaten eines Bildes über einen TCP-Socket und ruft Progress-Callbacks auf.
@param tcp_socket Offener TCP-Socket
@param img_bytes Bilddaten als Byte-Array
@param handle Ziel-Handle des Empfängers (für Progress-Callback)

```

#### 7.4.3.11 send\_join()

```

network.messenger.Messenger.send_join (
 self)

@brief Sendet eine JOIN-Nachricht per UDP-Broadcast, um dem Chat beizutreten.

```

#### 7.4.3.12 send\_known\_to()

```

network.messenger.Messenger.send_known_to (
 self,
 ip,
 port)

@brief Sendet bekannte Benutzer als Antwort auf WHO-Anfrage.
@param ip Ziel-IP-Adresse
@param port Ziel-Port
@details Erstellt eine KNOWNUSERS-Nachricht mit allen bekannten Peers
 inklusive eigener Informationen und sendet diese direkt an
 den anfragenden Peer.

```

#### 7.4.3.13 send\_leave()

```
network.messenger.Messenger.send_leave (
 self)
```

@brief Sendet eine LEAVE-Nachricht per UDP-Broadcast, um den Chat zu verlassen.

#### 7.4.3.14 send\_message()

```
network.messenger.Messenger.send_message (
 self,
 handle,
 message)
```

@brief Sendet eine Textnachricht an einen bestimmten Peer.

@param *handle* Ziel-Handle (Benutzername) des Empfängers

@param *message* Die zu sendende Nachricht (String)

#### 7.4.3.15 send\_slcp()

```
network.messenger.Messenger.send_slcp (
 self,
 line,
 ip,
 port)
```

@brief Sendet eine SLCP-Nachricht (UDP) an die angegebene Zieladresse.

@param *line* SLCP-formatierte Nachricht (String)

@param *ip* Ziel-IP-Adresse

@param *port* Ziel-Portnummer

#### 7.4.3.16 send\_who()

```
network.messenger.Messenger.send_who (
 self)
```

@brief Sendet eine WHO-Nachricht, um die Liste aktiver Teilnehmer zu erfragen.

#### 7.4.3.17 set\_image\_callback()

```
network.messenger.Messenger.set_image_callback (
 self,
 callback)
```

@brief Setzt den Callback für empfangene Bilder.

@param *callback* Funktion mit Signatur: (*sender\_handle*, *filename*)

#### 7.4.3.18 set\_knownusers\_callback()

```
network.messenger.Messenger.set_knownusers_callback (
 self,
 callback)
```

@brief Setzt den Callback für Benutzerlisten.

@param *callback* Async-Funktion mit Signatur: (*users\_list*)

*users\_list* ist eine Liste von (*handle*, *ip*, *port*) Tupeln



#### 7.4.3.19 set\_message\_callback()

```
network.messenger.Messenger.set_message_callback (
 self,
 callback)
```

@brief Setzt den Callback für empfangene Nachrichten.

@param callback Async-Funktion mit Signatur: (sender\_handle, message)

#### 7.4.3.20 set\_progress\_callback()

```
network.messenger.Messenger.set_progress_callback (
 self,
 callback)
```

@brief Setzt den Callback für Übertragungsfortschritt.

@param callback Funktion mit Signatur: (direction, handle, progress, bytes\_transferred, total\_bytes)

- direction: "send" oder "receive"
- handle: Benutzername des Partners
- progress: Fortschritt in Prozent (0-100)
- bytes\_transferred: Übertragene Bytes
- total\_bytes: Gesamtanzahl Bytes

#### 7.4.3.21 start\_listener()

```
network.messenger.Messenger.start_listener (
 self)
```

@brief Startet den UDP-Listener und den TCP-Server.

@details

Öffnet den UDP-Socket für Nachrichtenempfang und startet parallel den TCP-Server für den Empfang von Bildern.

#### 7.4.3.22 start\_tcp\_server()

```
network.messenger.Messenger.start_tcp_server (
 self)
```

@brief Startet den TCP-Server zum Empfang von eingehenden Bildübertragungen.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- Chat/network/messenger.py



# Kapitel 8

## Datei-Dokumentation

### 8.1 Chat/common/protocol.py-Dateireferenz

Enthält Funktionen zum Parsen und Erzeugen von SLCP-Protokollnachrichten.

#### Funktionen

- [common.protocol.parse\\_slcp](#) (line)
- [common.protocol.create\\_join](#) (handle, port)
- [common.protocol.create\\_leave](#) (handle)
- [common.protocol.create\\_msg](#) (target, text)
- [common.protocol.create\\_img](#) (target, size)
- [common.protocol.create\\_who](#) ()
- [common.protocol.create\\_knownusers](#) (users)

#### 8.1.1 Ausführliche Beschreibung

Enthält Funktionen zum Parsen und Erzeugen von SLCP-Protokollnachrichten.

Diese Datei stellt das Kommunikationsprotokoll bereit, das vom Messenger und Discovery-Service verwendet wird, um Text- und Bildnachrichten sowie Netzwerkanfragen zu senden und zu empfangen.

#### 8.1.2 Dokumentation der Funktionen

##### 8.1.2.1 create\_img()

```
common.protocol.create_img (
 target,
 size)
```

@brief Erstellt eine IMG-Nachricht zum Senden von Bilddaten.

```
@param target Empfänger-Handle
@param size Größe des Bildes in Bytes
@return SLCP-konforme IMG-Zeile
```

##### 8.1.2.2 create\_join()

```
common.protocol.create_join (
 handle,
 port)
```

@brief Erstellt eine JOIN-Nachricht.

```
@param handle Benutzername
@param port Portnummer
@return SLCP-konforme JOIN-Zeile
```

### 8.1.2.3 create\_knownusers()

```
common.protocol.create_knownusers (
 users)

@brief Erstellt eine KNOWNUSERS-Nachricht mit Liste aller bekannten Peers.

@param users Liste von Dictionaries mit Schlüsseln 'handle', 'ip', 'port'
@return SLCP-konforme KNOWNUSERS-Zeile
```

### 8.1.2.4 create\_leave()

```
common.protocol.create_leave (
 handle)

@brief Erstellt eine LEAVE-Nachricht.

@param handle Benutzername
@return SLCP-konforme LEAVE-Zeile
```

### 8.1.2.5 create\_msg()

```
common.protocol.create_msg (
 target,
 text)

@brief Erstellt eine MSG-Nachricht zum Senden eines Textes.

@param target Empfänger-Handle
@param text Nachrichtentext
@return SLCP-konforme MSG-Zeile
```

### 8.1.2.6 create\_who()

```
common.protocol.create_who ()

@brief Erstellt eine WHO-Broadcast-Nachricht zur Abfrage aller bekannten Nutzer.

@return SLCP-konforme WHO-Zeile
```

### 8.1.2.7 parse\_slcp()

```
common.protocol.parse_slcp (
 line)

@brief Parst eine SLCP-Zeile (Simple Local Chat Protocol) in ein Dictionary.

Unterstützte Befehle:
- JOIN <handle> <port>
- LEAVE <handle>
- WHO
- MSG <to> <message>
- IMG <to> <size>
- KNOWNUSERS <handle1> <ip1> <port1>, ...

@param line SLCP-Zeile als String
@return Dictionary mit Schlüssel "type" und weiteren Feldern je nach Befehl
```