SLCP-Chat

Generated by Doxygen 1.14.0

Chapter 1

SLCP-Chat Programm – Detaillierte Dokumentation

1.1 Einführung

Dieses Projekt implementiert einen dezentralen Chat-Client basierend auf dem Simple Local Chat Protocol (SLCP). Ziel ist eine Peer-to-Peer-Kommunikation ohne zentralen Server, die Text- und Bildnachrichten im lokalen Netzwerk ermöglicht.

1.2 Projektziele

- Vollständige Umsetzung des SLCP-Protokolls: JOIN, LEAVE, WHO, KNOWUSERS, MSG, IMG, WHOIS
- Dezentrale Discovery mittels Broadcast und lokaler Discovery-Dienste
- Klare Trennung der Funktionalität in drei Prozesse:
  - Benutzeroberfläche (ui.py)
  - Netzwerk-Modul (network.py, peer.py)
  - Discovery-Dienst (discovery.py)
- Konfiguration per TOML-Datei, editierbar über CLI
- Robustheit: Wiederverwendbarer Port-Finder, Named Pipes (FIFO), Signal-Handling

1.3 Architektur

Die Software-Architektur besteht aus drei Hauptkomponenten:

- Discovery-Dienst: Beantwortet WHO-Anfragen, verwaltet aktive Peers, kommuniziert über UDP
- Netzwerk-Modul: Sendet/empfängt SLCP-Nachrichten per UDP/TCP, verarbeitet JOIN/LEAVE/MSG/IMG/↩ WHOIS
- Benutzeroberfläche: CLI-Interface, leitet Befehle an Netzwerk-Modul weiter und zeigt Nachrichten

Kommunikation erfolgt über:

- UDP-Broadcast (entweder 255.255.255.255 oder lokale Broadcast-Adresse, z.B. 192.168.178.255↩ :whoisport) für Discovery
- TCP-Unicast für Direktnachrichten (MSG, IMG) und WHOIS
- Named Pipes (FIFO) für IPC zwischen UI und Netzwerkmodul

1.4 Konfiguration

Die zentrale Konfigurationsdatei `config.toml` enthält folgende Parameter:

- handle Benutzername des Clients
- port Start-Port für lokale UDP-/TCP-Sockets
- whoisport UDP-Port für Discovery-Broadcasts
- autoreply Automatische Abwesenheits-Antwort (leer = deaktiviert)
- imagepath Verzeichnis zum Speichern empfangener Bilder

Optional: zweite Konfiguration (`config2.toml`) zum schnellen Wechsel des Handles und Ports (Ordner: User2).

1.5 Modulübersicht

- common.py: Helferfunktionen, FIFO-Verwaltung, Logging, PID-Management
- peer.py: Modelliert Peers mit Handle, IP und Port, unterstützt Vergleich/Hash
- discovery.py: Discovery-Dienst (WHO, JOIN, LEAVE, KNOWUSERS)
- network.py: Netzwerkmodul (MSG, IMG, WHOIS, TCP/UDP-Sockets)
- ui.py: Kommandozeilen-UI, zeigt Nachrichten und empfängt Befehle

1.6 SLCP-Protokoll

SLCP-Befehle (textbasiert, UTF-8,
-terminiert):

- JOIN <Handle> <Port>
- LEAVE <Handle>
- WHO
- KNOWUSERS <Handle1> <IP1> <Port1>,...
- MSG <Handle> <Text>
- IMG <Handle> <Size> (gefolgt von Binärdaten)
- WHOIS <Handle> (Antwort: IAM <Handle> <IP> <Port>)

1.7 Schneller Start mit Skript

```
#!/bin/bash
gnome-terminal -- bash -c "python3 discovery.py config.toml; exec bash"
gnome-terminal -- bash -c "python3 network.py config.toml; exec bash"
gnome-terminal -- bash -c "python3 ui.py; exec bash"
```

Ausführbar: `chmod +x start_chat.sh`

1.8 Abhängigkeiten & Dokumentation

- `pip install toml`
- Dokumentation generieren: `doxygen Doxyfile` (HTML in `html/`)

1.9 Fehler

- **ConfigNotFound**: Konfigurationsdatei nicht gefunden – Lösung: Pfad prüfen oder `--config`-Parameter setzen.
- **FIFOPermission**: Keine Berechtigung für FIFO-Erstellung oder -Zugriff – Lösung: Schreibrechte auf `PIPE_DIR` sicherstellen.
- **BrokenPipe**: BrokenPipe-Fehler beim Schreiben in FIFO – Lösung: Retry/Timeout implementieren.
- **UDPBlocked**: UDP-Broadcast blockiert (z.B. macOS Restriktionen) – Lösung: lokale Broadcast-Adresse (z.B. `192.168.178.255`) verwenden.
- **PortInUse**: Socket-Bind-Fehler (Port belegt) – Lösung: anderen Start-Port konfigurieren.
- **TCPTimeout**: Timeout beim TCP-Verbindungsaufbau oder -Akzeptieren – Lösung: Socket-Timeouts setzen und `socket.timeout` abfangen.
- **InvalidCommand**: Unbekannter SLCP-Befehl – Lösung: Syntax validieren und Warnungen loggen.
- **PartialImage**: Teilweiser Bildempfang – Lösung: empfangene Bytes mit `Size` abgleichen.
- **SignalHandlerFail**: Signalhandler nicht registriert – Lösung: `signal.signal()` vor `start()` aufrufen.
- **IPCError**: Fehler bei IPC via FIFO – Lösung: Ausnahmen abfangen und fehlende Pipes automatisch anlegen.
- **WinError 10488**: Socketadresse bereits genutzt – Lösung: Terminal neu starten.
- **Your branch is behind**: Branch veraltet – Lösung: `git pull` ausführen.
- **TypeError**: 'DiscoveryService' is not subscriptable – Lösung: Objekte als Liste übergeben.

Screenshots einiger aufgetretener Fehler befinden sich in (`docs/image`)

1.10 Flussdiagramm der Hosts

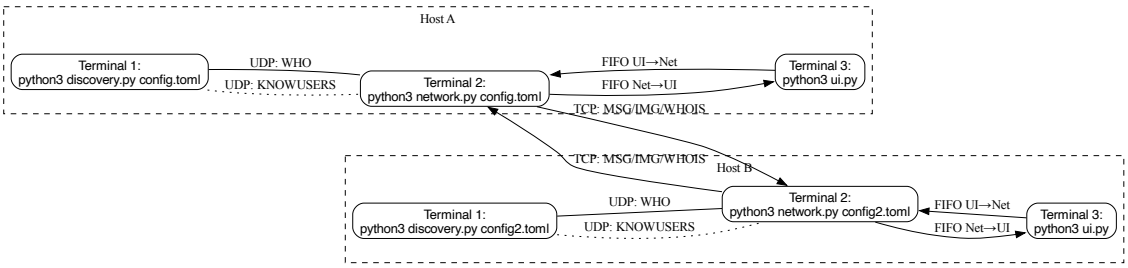
Ablauf auf zwei Hosts im selben WLAN:

- Host A: startet discovery.py, network.py, ui.py
- Host B: ebenso, mit eigener config

Kommunikation:

- UDP-Broadcast WHO → Discovery
- UDP-Reply KNOWUSERS
- TCP-Unicast für MSG/IMG/WHOIS
- FIFO für UI-Netzwerkkommunikation

1.11 Flussdiagramm der Hosts



1.12 Autoren

- Rajan Kakkar (1538362)
- Zufar Reyal (1417498)
- Soufian Kenbouche (1574324)
- Bileya Karimou (1570665)
- Yasmin Hammouni (1514652)

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 common Namespace Reference

Functions

- load_config (config_path="config.toml")
- pipe_path (name)
- create_fifo (name)
- write_to_fifo (name, message)
- read_from_fifo (name, blocking=True)
- write_pid (pid_file=PID_FILE)
- remove_pid (pid_file=PID_FILE)
- graceful_shutdown (sig, frame)

Variables

- USER = getpass.getuser()
- str PIPE_DIR = f"/tmp/slcp_{USER}"
- exist_ok
- PID_FILE = os.path.join(PIPE_DIR, "slcp_discovery.pid")
- level
- INFO
- format
- log = logging.getLogger("SLCP")

5.1.1 Detailed Description

```
@file common.py
@brief Utility-Funktionen und IPC-/Logging-Infrastruktur für den SLCP-Discovery-Dienst.

Diese Datei stellt Hilfsfunktionen, globale Konstanten und Pipe-Verwaltung bereit.
```

5.1.2 Function Documentation

5.1.2.1 create_fifo()

```
common.create_fifo (
              name)
```

```
@brief Erstellt eine Named Pipe (FIFO), falls sie nicht existiert.

@param name Name der Pipe, die erstellt werden soll.
@return Pfad zur erstellten oder bereits existierenden Pipe.
```

5.1.2.2 graceful_shutdown()

```
common.graceful_shutdown (
              sig,
              frame)
```

```
@brief Signal-Handler für SIGINT, beendet das Programm sauber.

@param sig Signalnummer (z.B. signal.SIGINT).
@param frame Aktueller Stack-Frame (wird nicht genutzt).
@return None
```

5.1.2.3 load_config()

```
common.load_config (
              config_path = "config.toml")
```

```
@brief Liest die Konfiguration aus einer TOML-Datei.

Diese Funktion lädt die Datei 'config.toml' und gibt den Abschnitt 'DEFAULT' zurück.
@param config_path Pfad zur TOML-Konfigurationsdatei.
@return Dictionary mit Werten aus dem Abschnitt 'DEFAULT'.
@throws SystemExit Wenn die Datei nicht existiert oder nicht gelesen werden kann.
```

5.1.2.4 pipe_path()

```
common.pipe_path (
              name)
```

```
@brief Bestimmt den Pfad zur FIFO für einen gegebenen Prozessnamen.

@param name Name der Pipe (z.B. 'ui', 'network').
@return Vollständiger Pfad zur Named Pipe im PIPE_DIR.
```

5.1.2.5 read_from_fifo()

```
common.read_from_fifo (
              name,
              blocking = True)
```

```
@brief Liest eine Zeile aus einer Named Pipe.

@param name Name der Pipe.
@param blocking Wenn False, wird non-blocking gelesen.
@return Geladene Zeile als String (ohne Newline) oder leer bei Fehler.
```

5.1.2.6 remove_pid()

```
common.remove_pid (
              pid_file = PID_FILE)
```

```
@brief Entfernt die PID-Datei des Discovery-Dienstes.

@param pid_file Pfad zur zu entfernenden PID-Datei.
@return None
```

5.1.2.7 write_pid()

```
common.write_pid (
              pid_file = PID_FILE)
```

```
@brief Schreibt die aktuelle Prozess-ID in eine Datei.

@param pid_file Pfad zur PID-Datei.
@return None
```

5.1.2.8 write_to_fifo()

```
common.write_to_fifo (
              name,
              message)
```

```
@brief Schreibt eine Nachricht in eine Named Pipe.

Versucht bis zu 5 Mal, non-blocking in die FIFO zu schreiben, und
ignoriert Broken-Pipe-Fehler oder fehlende Leser.

@param name Name der Pipe.
@param message Zu schreibende Nachricht (String).
@return None
```

5.1.3 Variable Documentation

5.1.3.1 exist_ok

```
common.exist_ok
```

5.1.3.2 format

```
common.format
```

5.1.3.3 INFO

```
common.INFO
```

5.1.3.4 level

```
common.level
```

5.1.3.5 log

```
common.log = logging.getLogger("SLCP")
```

5.1.3.6 PID_FILE

```
common.PID_FILE = os.path.join(PIPE_DIR, "slcp_discovery.pid")
```

5.1.3.7 PIPE_DIR

```
common.PIPE_DIR = f"/tmp/slcp_{USER}"
```

5.1.3.8 USER

```
common.USER = getpass.getuser()
```

5.2 config Namespace Reference

Variables

- str handle = "Alice"
- int port = 5010
- int whoisport = 4000
- str autoreply = "Ich bin gerade nicht da."
- str imagepath = "images"

5.2.1 Variable Documentation

5.2.1.1 autoreply

```
str config.autoreply = "Ich bin gerade nicht da."
```

5.2.1.2 handle

```
str config.handle = "Alice"
```

5.2.1.3 imagepath

```
str config.imagepath = "images"
```

5.2.1.4 port

```
int config.port = 5010
```

5.2.1.5 whoisport

```
int config.whoisport = 4000
```

5.3 discovery Namespace Reference

Classes

- class Discovery

Variables

- d = Discovery()

5.3.1 Detailed Description

```
@file discovery.py
@brief Implementierung des Discovery-Dienstes für das SLCP-Chat-Programm.

Die Klasse Discovery verwaltet eingehende WHO-, JOIN- und LEAVE-Anfragen via UDP.
Sie speichert bekannte Peers und antwortet auf Broadcasts mit aktuellen Benutzer-Informationen.
```

5.3.2 Variable Documentation

5.3.2.1 d

```
discovery.d = Discovery()
```

5.4 network Namespace Reference

Functions

- get_own_ip (peer_ip="8.8.8.8")
- send_join (handle, port)
- send_leave (handle)
- send_who ()
- send_whois (target_handle)
- send_msg (target, text)
- send_img (target, image_path)
- receive_udp ()
- listen_tcp (port, imagepath)
- handle_commands (handle, port, imagepath)
- cleanup_and_exit (signum, frame)
- start_network ()

Variables

- str FIFO_UI_TO_NET = "ui_to_net"
- str FIFO_NET_TO_UI = "net_to_ui"
- dict peers = {}
- udp_sock = None
- dict config = {}
- bool running = True
- last_peers_display = set()
- left_peers = set()
- dict peer_join_time = {}

5.4.1 Detailed Description

```
@file network.py
@brief Implementierung der Netzwerk-Kommunikation für das SLCP-Chat-Programm.

Dieses Modul kümmert sich um das Senden und Empfangen von SLCP-Nachrichten:
- Broadcast (JOIN, LEAVE, WHO)
- Unicast (MSG, IMG, WHOIS, KNOWNUSERS)
Es verwendet UDP für Broadcasts und TCP für Peer-to-Peer-Kommunikation.
```

5.4.2 Function Documentation

5.4.2.1 cleanup_and_exit()

```
network.cleanup_and_exit (
              signum,
              frame)
```

```
@brief Signal-Handler für SIGINT/SIGTERM, sendet LEAVE und beendet das Programm.

@param signum Signalnummer.
@param frame Aktueller Stack-Frame.
@return None
```

5.4.2.2 get_own_ip()

```
network.get_own_ip (
              peer_ip = "8.8.8.8")
```

```
@brief Ermittelt die eigene lokale IP-Adresse.

Baut eine temporäre UDP-Verbindung zu 'peer_ip' auf und liest
die lokale Socket-Adresse aus.

@param peer_ip Externe IP-Adresse zum Herstellen der Verbindung.
@return String mit der lokalen IP-Adresse oder '127.0.0.1' bei Fehler.
```

5.4.2.3 handle_commands()

```
network.handle_commands (
              handle,
              port,
              imagepath)
```

```
@brief Verarbeitet Befehle von der UI-FIFO und ruft entsprechende Funktionen auf.

Unterstützte Befehle:
- msg <target> <text>
- img <target> <path>
- who, whois <handle>
- JOIN <handle> <port>
- LEAVE

@param handle Eigenes Handle.
@param port Eigener Port (UDP/TCP).
@param imagepath Verzeichnis für Bilder.
@return None
```

5.4.2.4 listen_tcp()

```
network.listen_tcp (
              port,
              imagepath)
```

```
@brief TCP-Server-Loop für Unicast-Kommunikation (MSG, IMG, WHOIS, JOIN, LEAVE, WHO).

@param port Eigener TCP-Port (gleich UDP-Port).
@param imagepath Verzeichnis zum Speichern empfangener Bilder.
@return None
```

5.4.2.5 receive_udp()

```
network.receive_udp ()
```

```
@brief Listener-Loop für eingehende UDP-Nachrichten (JOIN/LEAVE/IAM/KNOWNUSERS).

Schreibt entsprechende Events in die UI-FIFO und aktualisiert Peers-Listen.

@return None
```

5.4.2.6 send_img()

```
network.send_img (
              target,
              image_path)
```

```
@brief Sendet eine Bildnachricht (IMG) an einen Peer.

@param target Handle des Empfängers.
@param image_path Pfad zur Bilddatei.
@return None
```

5.4.2.7 send_join()

```
network.send_join (
              handle,
              port)
```

```
@brief Sendet eine JOIN-Nachricht per UDP-Broadcast und TCP-Unicast.

Verteilt die Nachricht an Broadcast und an bereits bekannte Peers.

@param handle Eigenes Handle.
@param port Eigener UDP-Port.
@return None
```

5.4.2.8 send_leave()

```
network.send_leave (
              handle)
```

@brief Sendet eine LEAVE-Nachricht an alle Teilnehmer.

@param handle Eigenes Handle.
@return None

5.4.2.9 send_msg()

```
network.send_msg (
              target,
              text)
```

@brief Sendet eine Textnachricht (MSG) an einen Peer.

@param target Handle des Empfängers.
@param text Nachrichtentext.
@return None

5.4.2.10 send_who()

```
network.send_who ()
```

@brief Sendet eine WHO-Broadcast-Anfrage.

@return None

5.4.2.11 send_whois()

```
network.send_whois (
              target_handle)
```

@brief Sendet eine WHOIS-Anfrage an einen spezifischen Peer.

@param target_handle Handle des Ziels.
@return None

5.4.2.12 start_network()

```
network.start_network ()
```

@brief Initialisiert das Netzwerk-Modul und startet Listener-Threads.

Lädt Konfiguration, baut Sockets, startet Broadcast/Unicast-Listeners
und verarbeitet UI-Befehle.

@return None

5.4.3 Variable Documentation

5.4.3.1 config

```
dict network.config = {}
```

5.4.3.2 FIFO_NET_TO_UI

```
str network.FIFO_NET_TO_UI = "net_to_ui"
```

5.4.3.3 FIFO_UI_TO_NET

```
str network.FIFO_UI_TO_NET = "ui_to_net"
```

5.4.3.4 last_peers_display

```
network.last_peers_display = set()
```

5.4.3.5 left_peers

```
network.left_peers = set()
```

5.4.3.6 peer_join_time

```
dict network.peer_join_time = {}
```

5.4.3.7 peers

```
dict network.peers = {}
```

5.4.3.8 running

```
bool network.running = True
```

5.4.3.9 udp_sock

```
network.udp_sock = None
```

5.5 peer Namespace Reference

Classes

- class Peer

5.5.1 Detailed Description

```
@file peer.py
@brief Definiert die Peer-Klasse, die einen Chat-Teilnehmer modelliert.

Die Klasse Peer kapselt Handle, IP-Adresse und Port eines Chat-Teilnehmers
und stellt Methoden für Vergleich und Adresszugriff bereit.
```

5.6 ui Namespace Reference

Classes

- class ChatUI

Variables

- str FIFO_UI_TO_NET = "ui_to_net"
- str FIFO_NET_TO_UI = "net_to_ui"

5.6.1 Detailed Description

```
@file ui.py
@brief Implementierung der Kommandozeilen-Benutzeroberfläche (CLI) für das SLCP-Chat-Programm.

Die Klasse ChatUI bietet eine textbasierte Oberfläche zum Senden von Befehlen
und Anzeigen von Nachrichten/Broadcasts. Kommunikation mit dem Netzwerk-Modul erfolgt
über Named Pipes (FIFOs).
```

5.6.2 Variable Documentation

5.6.2.1 FIFO_NET_TO_UI

```
str ui.FIFO_NET_TO_UI = "net_to_ui"
```

5.6.2.2 FIFO_UI_TO_NET

```
str ui.FIFO_UI_TO_NET = "ui_to_net"
```

Chapter 6

Class Documentation

6.1 ui.ChatUI Class Reference

Public Member Functions

- __init__ (self)
- listen_pipes (self)
- start (self)

Public Attributes

- config = load_config()
- handle = self.config["handle"]
- port = self.config["port"]
- pipe_path = pipe_path(FIFO_NET_TO_UI)

6.1.1 Detailed Description

```
@brief Chat-Interface für den Benutzer.

Initialisiert UI-Pipes, lädt Konfiguration und bietet eine Eingabe-Loop für Befehle.
```

6.1.2 Constructor & Destructor Documentation

6.1.2.1 __init__()

```
ui.ChatUI.__init__ (
               self)
```

```
@brief Initialisiert das ChatUI-Objekt.

Lädt Konfiguration, legt Named Pipes an und speichert Pfad zur Eingabe-Pipe.
```

6.1.3 Member Function Documentation

6.1.3.1 listen_pipes()

```
ui.ChatUI.listen_pipes (
              self)
```

@brief Liest dauerhaft Nachrichten aus der FIFO der Netzwerk-Komponente.

Öffnet die UI-Leser-Pipe und gibt neue Nachrichten in der Konsole aus.

6.1.3.2 start()

```
ui.ChatUI.start (
              self)
```

@brief Startet die UI-Hauptschleife und den Pipe-Listener-Thread.

Zeigt Willkommensnachricht und Eingabe-Prompt an und verarbeitet Nutzerbefehle.

6.1.4 Member Data Documentation

6.1.4.1 config

```
ui.ChatUI.config = load_config()
```

6.1.4.2 handle

```
ui.ChatUI.handle = self.config["handle"]
```

6.1.4.3 pipe_path

```
ui.ChatUI.pipe_path = pipe_path(FIFO_NET_TO_UI)
```

6.1.4.4 port

```
ui.ChatUI.port = self.config["port"]
```

The documentation for this class was generated from the following file:

- ui.py

6.2 discovery.Discovery Class Reference

Public Member Functions

- __init__ (self, config_path="config.toml")
- start (self)
- send_who (self)
- listen_udp_all (self)
- shutdown (self, *args)

Public Attributes

- config = load_config(config_path)
- handle = self.config["handle"]
- port = self.config["port"]
- whois_port = self.config.get("whoisport", 4000)
- dict known_peers = {}
- udp_listener_thread

6.2.1 Detailed Description

```
Discovery-Dienst-Klasse

Startet einen Thread, der auf alle relevanten SLCP-Befehle via UDP lauscht.
```

6.2.2 Constructor & Destructor Documentation

6.2.2.1 __init__()

```
discovery.Discovery.__init__ (
              self,
              config_path = "config.toml")
```

6.2.3 Member Function Documentation

6.2.3.1 listen_udp_all()

```
discovery.Discovery.listen_udp_all (
              self)


Lauscht auf UDP-Nachrichten aller SLCP-Befehle:

- WHO: Antwortet mit aktueller Peerliste.
- JOIN: Fügt neuen Peer hinzu und leitet JOIN an andere weiter.
- LEAVE: Entfernt Peer und leitet LEAVE an andere weiter.

@return None
```

6.2.3.2 send_who()

```
discovery.Discovery.send_who (
              self)
```

Sendet eine WHO-Broadcast-Anfrage an alle Discovery-Server.

6.2.3.3 shutdown()

```
discovery.Discovery.shutdown (
              self,
             * args)
```

Beendet den Discovery-Dienst und entfernt die PID-Datei.

6.2.3.4 start()

```
discovery.Discovery.start (
              self)
```

Startet den Discovery-Listener-Thread und hält den Dienst aktiv.

6.2.4 Member Data Documentation

6.2.4.1 config

```
discovery.Discovery.config = load_config(config_path)
```

6.2.4.2 handle

```
discovery.Discovery.handle = self.config["handle"]
```

6.2.4.3 known_peers

```
dict discovery.Discovery.known_peers = {}
```

6.2.4.4 port

```
discovery.Discovery.port = self.config["port"]
```

6.2.4.5 udp_listener_thread

```
discovery.Discovery.udp_listener_thread
```

Initial value:

```
=  threading.Thread(
            target=self.listen_udp_all, daemon=True
        )
```

6.2.4.6 whois_port

```
discovery.Discovery.whois_port = self.config.get("whoisport", 4000)
```

The documentation for this class was generated from the following file:

- discovery.py

6.3 peer.Peer Class Reference

Public Member Functions

- __init__ (self, handle, ip, port)
- __repr__ (self)
- __eq__ (self, other)
- __hash__ (self)
- get_address (self)

Public Attributes

- handle = handle
- ip = ip
- port = port

6.3.1 Detailed Description

```
@brief Repräsentiert einen Chat-Teilnehmer (Peer) im SLCP-Netzwerk.

Ein Peer enthält:
- handle: Der eindeutige Benutzername des Teilnehmers
- ip: Die IP-Adresse des Teilnehmers
- port: Der Port, über den der Teilnehmer erreichbar ist
```

6.3.2 Constructor & Destructor Documentation

6.3.2.1 __init__()

```
peer.Peer.__init__ (
              self,
              handle,
              ip,
              port)
```

@brief Initialisiert einen Peer.

@param handle Eindeutiger Benutzername des Peers.
@param ip IP-Adresse des Peers.
@param port Portnummer des Peers.

6.3.3 Member Function Documentation

6.3.3.1 __eq__()

```
peer.Peer.__eq__ (
              self,
              other)
```

@brief Vergleich zweier Peer-Objekte nach ihrem Handle.

@param other Ein weiteres Objekt.
@return True, wenn 'other' ein Peer ist und denselben Handle hat.

6.3.3.2 __hash__()

```
peer.Peer.__hash__ (
              self)
```

@brief Berechnet den Hash-Wert eines Peers basierend auf dem Handle.

@return Integer-Hash des Handles.

6.3.3.3 __repr__()

```
peer.Peer.__repr__ (
              self)
```

@brief String-Repräsentation eines Peers.

@return Darstellung in der Form 'handle@ip:port'.

6.3.3.4 get_address()

```
peer.Peer.get_address (
              self)
```

```
@brief Liefert ein Tuple aus IP-Adresse und Port für Netzwerkverbindungen.

@return Tuple (ip, port).
```

6.3.4 Member Data Documentation

6.3.4.1 handle

```
peer.Peer.handle = handle
```

6.3.4.2 ip

```
peer.Peer.ip = ip
```

6.3.4.3 port

```
peer.Peer.port = port
```

The documentation for this class was generated from the following file:

- peer.py

Chapter 7

File Documentation

7.1 common.py File Reference

Namespaces

- namespace common

Functions

- common.load_config (config_path="config.toml")
- common.pipe_path (name)
- common.create_fifo (name)
- common.write_to_fifo (name, message)
- common.read_from_fifo (name, blocking=True)
- common.write_pid (pid_file=PID_FILE)
- common.remove_pid (pid_file=PID_FILE)
- common.graceful_shutdown (sig, frame)

Variables

- common.USER = getpass.getuser()
- str common.PIPE_DIR = f"/tmp/slcp_{USER}"
- common.exist_ok
- common.PID_FILE = os.path.join(PIPE_DIR, "slcp_discovery.pid")
- common.level
- common.INFO
- common.format
- common.log = logging.getLogger("SLCP")

7.2 config.toml File Reference

Namespaces

- namespace config

Variables

- str config.handle = "Alice"
- int config.port = 5010
- int config.whoisport = 4000
- str config.autoreply = "Ich bin gerade nicht da."
- str config.imagepath = "images"

7.3 discovery.py File Reference

Classes

- class discovery.Discovery

Namespaces

- namespace discovery

Variables

- discovery.d = Discovery()

7.4 docs/mainpage.dox File Reference

7.5 network.py File Reference

Namespaces

- namespace network

Functions

- network.get_own_ip (peer_ip="8.8.8.8")
- network.send_join (handle, port)
- network.send_leave (handle)
- network.send_who ()
- network.send_whois (target_handle)
- network.send_msg (target, text)
- network.send_img (target, image_path)
- network.receive_udp ()
- network.listen_tcp (port, imagepath)
- network.handle_commands (handle, port, imagepath)
- network.cleanup_and_exit (signum, frame)
- network.start_network ()

Variables

- str network.FIFO_UI_TO_NET = "ui_to_net"
- str network.FIFO_NET_TO_UI = "net_to_ui"
- dict network.peers = {}
- network.udp_sock = None
- dict network.config = {}
- bool network.running = True
- network.last_peers_display = set()
- network.left_peers = set()
- dict network.peer_join_time = {}

7.6 peer.py File Reference

Classes

- class peer.Peer

Namespaces

- namespace peer

7.7 ui.py File Reference

Classes

- class ui.ChatUI

Namespaces

- namespace ui

Variables

- str ui.FIFO_UI_TO_NET = "ui_to_net"
- str ui.FIFO_NET_TO_UI = "net_to_ui"

Index