Chat-Programm

Generated by Doxygen 1.14.0

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 discovery_service Namespace Reference

Classes

- class DiscoveryService

4.1.1 Detailed Description

```
@file discovery_service.py
@brief Verwaltet Peer-Entdeckung und -Listen über UDP
```

4.2 main Namespace Reference

Functions

- main (str config_path)

4.2.1 Detailed Description

```
@file main.py
@brief Hauptmodul für Initialisierung und Prozessmanagement
```

4.2.2 Function Documentation

4.2.2.1 main()

```
main.main (
            str config_path)

@brief Hauptfunktion des SimpleChat-Programms
@param config_path Pfad zur Konfigurationsdatei

@details
- Lädt die Konfiguration aus der TOML-Datei
- Erstellt gemeinsame Datenstrukturen mit multiprocessing.Manager
- Initialisiert alle Komponenten
- Startet Prozesse und verwaltet Graceful Shutdown
```

Definition at line 13 of file main.py.

4.3 network_communication Namespace Reference

Classes

- class NetworkCommunication

4.3.1 Detailed Description

```
@file network_communication.py
@brief Handhabt TCP-basierte Nachrichten- und Bildübertragung
```

4.4 user_interface Namespace Reference

Classes

- class UserInterface

4.4.1 Detailed Description

```
@file user_interface.py
@brief CLI-Schnittstelle für Benutzerinteraktion und Befehlsverarbeitung
```

Chapter 5

Class Documentation

5.1 discovery_service.DiscoveryService Class Reference

Public Member Functions

- __init__ (self, int udp_port, Manager().dict() peers, str username, int tcp_port)
- run (self)

Public Attributes

- udp_port = udp_port
- udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
- peers = peers
- username = username
- tcp_port = tcp_port
- bool running = True

Static Public Attributes

- int BROADCAST_INTERVAL = 30
- int PEER_TIMEOUT = 60

Protected Member Functions

- _listen_udp (self)
- _handle_slcp_command (self, str command, str source_ip)
- _broadcast_presence (self)
- _cleanup_peers (self)

5.1.1 Detailed Description

```
@class DiscoveryService
@brief Implementiert den SLCP-Discovery-Mechanismus

@details
- Verarbeitet JOIN, LEAVE, WHO und KNOWUSERS Befehle
- Verwaltet eine Liste bekannter Peers mit Zeitstempeln
- Sendet regelmäßig Broadcasts zur eigenen Präsenz
- Bereinigt inaktive Peers automatisch
```

Definition at line 12 of file discovery_service.py.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 __init__()

```
discovery_service.DiscoveryService.__init__ (
              self,
             int udp_port,
             Manager().dict() peers,
             str username,
             int tcp_port)
```

```
@brief Initialisiert den UDP-Socket und Peer-Daten
@param udp_port Port für UDP-Kommunikation
@param peers Shared Dictionary der bekannten Peers
@param username Eigener Benutzername
@param tcp_port Eigener TCP-Port für Nachrichten
```

Definition at line 27 of file discovery_service.py.

5.1.3 Member Function Documentation

5.1.3.1 _broadcast_presence()

```
discovery_service.DiscoveryService._broadcast_presence (
              self)  [protected]
```

```
@brief Sendet regelmäßig JOIN-Broadcasts
@details Informiert andere Peers über eigene Präsenz
```

Definition at line 139 of file discovery_service.py.

5.1.3.2 _cleanup_peers()

```
discovery_service.DiscoveryService._cleanup_peers (
              self)  [protected]
```

```
@brief Entfernt inaktive Peers aus der Liste
@details Prüft regelmäßig die letzte Aktivität
```

Definition at line 149 of file discovery_service.py.

5.1.3.3 _handle_slcp_command()

```
discovery_service.DiscoveryService._handle_slcp_command (
              self,
             str command,
             str source_ip)  [protected]
```

```
@brief Verarbeitet SLCP-Discovery-Befehle
@param command SLCP-kodierter Befehl
@param source_ip Ursprungs-IP des Befehls

@details Unterstützte Befehle:
- JOIN:<username>:<tcp_port>
- LEAVE:<username>
- WHO
- KNOWUSERS:<json_peer_list>
```

Definition at line 81 of file discovery_service.py.

5.1.3.4 _listen_udp()

```
discovery_service.DiscoveryService._listen_udp (
              self)  [protected]
```

```
@brief Lauscht kontinuierlich auf UDP-Broadcasts
@details Verarbeitet eingehende SLCP-Befehle
```

Definition at line 65 of file discovery_service.py.

5.1.3.5 run()

```
discovery_service.DiscoveryService.run (
              self)
```

```
@brief Hauptbetriebsschleife des Discovery-Dienstes
@details Kombiniert passives Lauschen und aktive Peer-Aktualisierung
```

Definition at line 45 of file discovery_service.py.

5.1.4 Member Data Documentation

5.1.4.1 BROADCAST_INTERVAL

```
discovery_service.DiscoveryService.BROADCAST_INTERVAL = 30  [static]
```

Definition at line 24 of file discovery_service.py.

5.1.4.2 PEER_TIMEOUT

```
int discovery_service.DiscoveryService.PEER_TIMEOUT = 60  [static]
```

Definition at line 25 of file discovery_service.py.

5.1.4.3 peers

```
discovery_service.DiscoveryService.peers = peers
```

Definition at line 40 of file discovery_service.py.

5.1.4.4 running

```
bool discovery_service.DiscoveryService.running = True
```

Definition at line 43 of file discovery_service.py.

5.1.4.5 tcp_port

```
discovery_service.DiscoveryService.tcp_port = tcp_port
```

Definition at line 42 of file discovery_service.py.

5.1.4.6 udp_port

```
discovery_service.DiscoveryService.udp_port = udp_port
```

Definition at line 35 of file discovery_service.py.

5.1.4.7 udp_socket

```
discovery_service.DiscoveryService.udp_socket = socket.socket(socket.AF_INET, socket.SOCK_↩
DGRAM)
```

Definition at line 36 of file discovery_service.py.

5.1.4.8 username

```
discovery_service.DiscoveryService.username = username
```

Definition at line 41 of file discovery_service.py.

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

- discovery_service.py

5.2 network_communication.NetworkCommunication Class Reference

Public Member Functions

- __init__ (self, int tcp_port, str img_dir, Manager().dict() peers)
- start (self)

Public Attributes

- tcp_port = tcp_port
- tcp_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- img_dir = img_dir
- peers = peers
- bool running = True

Static Public Attributes

- int CHUNK_SIZE = 4096

Protected Member Functions

- _accept_connections (self)
- _handle_client (self, socket.socket client_socket)
- _process_message (self, str header, socket.socket socket)
- _process_image (self, str header, socket.socket socket)

5.2.1 Detailed Description

```
@class NetworkCommunication
@brief Verwaltet TCP-Kommunikation mit Peers

@details
- Startet einen TCP-Server für eingehende Verbindungen
- Verarbeitet Textnachrichten und Bildübertragungen
- Implementiert einen Handshake-Mechanismus für zuverlässigen Bildtransfer
- Speichert empfangene Bilder im konfigurierten Verzeichnis
```

Definition at line 14 of file network_communication.py.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 __init__()

```
network_communication.NetworkCommunication.__init__ (
              self,
             int tcp_port,
             str img_dir,
             Manager().dict() peers)

@brief Initialisiert TCP-Server und Konfiguration
@param tcp_port Port für TCP-Kommunikation
@param img_dir Verzeichnis zum Speichern von Bildern
@param peers Shared Dictionary der bekannten Peers
```

Definition at line 28 of file network_communication.py.

5.2.3 Member Function Documentation

5.2.3.1 _accept_connections()

```
network_communication.NetworkCommunication._accept_connections (
              self)  [protected]

@brief Akzeptiert eingehende TCP-Verbindungen
@details Startet für jede Verbindung einen Handler-Thread
```

Definition at line 61 of file network_communication.py.

5.2.3.2 _handle_client()

```
network_communication.NetworkCommunication._handle_client (
              self,
             socket.socket client_socket)  [protected]

@brief Verarbeitet eingehende TCP-Datenströme
@param client_socket Socket des verbundenen Clients
```

Definition at line 81 of file network_communication.py.

5.2.3.3 _process_image()

```
network_communication.NetworkCommunication._process_image (
              self,
             str header,
             socket.socket socket)  [protected]

@brief Verarbeitet Bildübertragungen mit Handshake
@param header Bildmetadaten (Größe, Dateiname)
@param socket Verbindungssocket

@details Ablauf:
1. Header parsen: IMG:<Größe>:<Dateiname>
2. ACK senden
3. Daten empfangen und prüfen
4. Bild speichern oder verwerfen
```

Definition at line 121 of file network_communication.py.

5.2.3.4 _process_message()

```
network_communication.NetworkCommunication._process_message (
              self,
             str header,
             socket.socket socket)  [protected]

@brief Verarbeitet eingehende Textnachrichten
@param header Nachrichtenheader
@param socket Verbindungssocket

@details Nachrichtenformat:
MSG:<sender>:<receiver>:<timestamp>:<message>
```

Definition at line 99 of file network_communication.py.

5.2.3.5 start()

```
network_communication.NetworkCommunication.start (
              self)
```

```
@brief Startet TCP-Server und Nachrichtenverarbeitung
```

Definition at line 48 of file network_communication.py.

5.2.4 Member Data Documentation

5.2.4.1 CHUNK_SIZE

```
int network_communication.NetworkCommunication.CHUNK_SIZE = 4096  [static]
```

Definition at line 26 of file network_communication.py.

5.2.4.2 img_dir

```
network_communication.NetworkCommunication.img_dir = img_dir
```

Definition at line 41 of file network_communication.py.

5.2.4.3 peers

```
network_communication.NetworkCommunication.peers = peers
```

Definition at line 42 of file network_communication.py.

5.2.4.4 running

```
bool network_communication.NetworkCommunication.running = True
```

Definition at line 43 of file network_communication.py.

5.2.4.5 tcp_port

```
network_communication.NetworkCommunication.tcp_port = tcp_port
```

Definition at line 35 of file network_communication.py.

5.2.4.6 tcp_server

```
network_communication.NetworkCommunication.tcp_server = socket.socket(socket.AF_INET, socket.↩
SOCK_STREAM)
```

Definition at line 36 of file network_communication.py.

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

- network_communication.py

5.3 user_interface.UserInterface Class Reference

Public Member Functions

- __init__ (self, Queue command_queue, Queue response_queue)
- start (self)

Public Attributes

- command_queue = command_queue
- response_queue = response_queue
- bool running = True

Protected Member Functions

- _input_loop (self)
- dict _parse_command (self, str input_str)
- _logging_loop (self)

5.3.1 Detailed Description

```
@class UserInterface
@brief Verwaltet die Kommandozeilenschnittstelle und leitet Befehle weiter

@details
- Parst Benutzereingaben und wandelt sie in strukturierte Befehle um
- Kommuniziert mit anderen Komponenten über Queues
- Zeigt Systemnachrichten und eingehende Nachrichten in Echtzeit an
```

Definition at line 10 of file user_interface.py.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 __init__()

```
user_interface.UserInterface.__init__ (
              self,
             Queue command_queue,
             Queue response_queue)

@brief Konstruktor initialisiert Queues und Statusvariablen
@param command_queue Queue für ausgehende Befehle
@param response_queue Queue für eingehende Antworten
```

Definition at line 21 of file user_interface.py.

5.3.3 Member Function Documentation

5.3.3.1 _input_loop()

```
user_interface.UserInterface._input_loop (
              self)  [protected]

@brief Verarbeitet kontinuierlich Benutzereingaben
@details Parst Eingaben und leitet sie an entsprechende Module weiter
```

Definition at line 49 of file user_interface.py.

5.3.3.2 _logging_loop()

```
user_interface.UserInterface._logging_loop (
              self)  [protected]

@brief Zeigt Systemnachrichten und eingehende Chat-Nachrichten an
@details Überwacht die response_queue kontinuierlich und zeigt Nachrichten an
```

Definition at line 105 of file user_interface.py.

5.3.3.3 _parse_command()

```
dict user_interface.UserInterface._parse_command (
              self,
             str input_str)  [protected]

@brief Zerlegt Benutzereingaben in strukturierte Befehle
@param input_str Roh-Eingabe des Nutzers
@return Dictionary mit Schlüsseln: 'type', 'target', 'content'
@retval None bei ungültigen Befehlen

@details Unterstützte Befehle:
- join: Beitritt zum Netzwerk
- who: Zeigt bekannte Peers an
- msg <peer> <text>: Sendet Textnachricht
- img <peer> <pfad>: Sendet Bild
- leave: Verlässt das Netzwerk
- config: Zeigt Konfiguration
- exit: Beendet die Anwendung
```

Definition at line 65 of file user_interface.py.

5.3.3.4 start()

```
user_interface.UserInterface.start (
              self)


@brief Hauptstartroutine der CLI
@details Startet zwei Threads:
         1. Eingabeverarbeitung (frontend)
         2. Antwortverarbeitung (backend)
```

Definition at line 31 of file user_interface.py.

5.3.4 Member Data Documentation

5.3.4.1 command_queue

```
user_interface.UserInterface.command_queue = command_queue
```

Definition at line 27 of file user_interface.py.

5.3.4.2 response_queue

```
user_interface.UserInterface.response_queue = response_queue
```

Definition at line 28 of file user_interface.py.

5.3.4.3 running

```
bool user_interface.UserInterface.running = True
```

Definition at line 29 of file user_interface.py.

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

- user_interface.py

Chapter 6

File Documentation

6.1 discovery_service.py File Reference

Classes

- class discovery_service.DiscoveryService

Namespaces

- namespace discovery_service

6.2 discovery_service.py

Go to the documentation of this file.

```
00001 """
00002 @file discovery_service.py
00003 @brief Verwaltet Peer-Entdeckung und -Listen über UDP
00004 """
00005
00006 import socket
00007 import time
00008 import threading
00009 import json
00010 from multiprocessing import Manager
00011
00012 class DiscoveryService:
00013     """
00014     @class DiscoveryService
00015     @brief Implementiert den SLCP-Discovery-Mechanismus
00016
00017     @details
00018     - Verarbeitet JOIN, LEAVE, WHO und KNOWUSERS Befehle
00019     - Verwaltet eine Liste bekannter Peers mit Zeitstempeln
00020     - Sendet regelmäßig Broadcasts zur eigenen Präsenz
00021     - Bereinigt inaktive Peers automatisch
00022     """
00023
00024     BROADCAST_INTERVAL = 30  # Sekunden
00025     PEER_TIMEOUT = 60        # Sekunden
00026
00027     def __init__(self, udp_port: int, peers: Manager().dict(), username: str, tcp_port: int):
00028         """
00029         @brief Initialisiert den UDP-Socket und Peer-Daten
00030         @param udp_port Port für UDP-Kommunikation
00031         @param peers Shared Dictionary der bekannten Peers
00032         @param username Eigener Benutzername
00033         @param tcp_port Eigener TCP-Port für Nachrichten
```

```
00034         """
00035         self.udp_port = udp_port
00036         self.udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
00037         self.udp_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
00038         self.udp_socket.bind((", udp_port))
00039         self.udp_socket.settimeout(1.0)  # Timeout für recvfrom
00040         self.peers = peers
00041         self.username = username
00042         self.tcp_port = tcp_port
00043         self.running = True
00044
00045     def run(self):
00046         """
00047         @brief Hauptbetriebsschleife des Discovery-Dienstes
00048         @details Kombiniert passives Lauschen und aktive Peer-Aktualisierung
00049         """
00050         listen_thread = threading.Thread(target=self._listen_udp)
00051         broadcast_thread = threading.Thread(target=self._broadcast_presence)
00052         cleanup_thread = threading.Thread(target=self._cleanup_peers)
00053
00054         listen_thread.daemon = True
00055         broadcast_thread.daemon = True
00056         cleanup_thread.daemon = True
00057
00058         listen_thread.start()
00059         broadcast_thread.start()
00060         cleanup_thread.start()
00061
00062         while self.running:
00063             time.sleep(0.1)
00064
00065     def _listen_udp(self):
00066         """
00067         @brief Lauscht kontinuierlich auf UDP-Broadcasts
00068         @details Verarbeitet eingehende SLCP-Befehle
00069         """
00070         while self.running:
00071             try:
00072                 data, addr = self.udp_socket.recvfrom(1024)
00073                 command = data.decode().strip()
00074                 self._handle_slcp_command(command, addr[0])
00075             except socket.timeout:
00076                 continue
00077             except socket.error:
00078                 if not self.running:
00079                     break
00080
00081     def _handle_slcp_command(self, command: str, source_ip: str):
00082         """
00083         @brief Verarbeitet SLCP-Discovery-Befehle
00084         @param command SLCP-kodierter Befehl
00085         @param source_ip Ursprungs-IP des Befehls
00086
00087         @details Unterstützte Befehle:
00088         - JOIN:<username>:<tcp_port>
00089         - LEAVE:<username>
00090         - WHO
00091         - KNOWUSERS:<json_peer_list>
00092         """
00093         if command.startswith("JOIN:"):
00094             parts = command.split(':')
00095             if len(parts) >= 3:
00096                 user = parts[1]
00097                 try:
00098                     tcp_port = int(parts[2])
00099                     # Aktualisiere Peer-Liste
00100                     self.peers[user] = {
00101                         'ip': source_ip,
00102                         'port': tcp_port,
00103                         'last_seen': time.time()
00104                     }
00105                     print(f"Neuer Peer: {user}@{source_ip}:{tcp_port}")
00106                 except ValueError:
00107                     pass
00108
00109         elif command == "WHO":
00110             # Sende Liste der bekannten Peers
00111             known_users = {user: data for user, data in self.peers.items()
00112                            if user != self.username}
00113             response = f"KNOWUSERS:{json.dumps(known_users)}"
00114             self.udp_socket.sendto(response.encode(), (source_ip, self.udp_port))
00115
00116         elif command.startswith("KNOWUSERS:"):
00117             try:
00118                 json_str = command.split(':', 1)[1]
00119                 peer_data = json.loads(json_str)
00120                 current_time = time.time()
```

```
00121
00122                 for user, data in peer_data.items():
00123                     # Aktualisiere nur, wenn der Peer nicht wir selbst sind
00124                     if user != self.username:
00125                         self.peers[user] = {
00126                             'ip': data['ip'],
00127                             'port': data['port'],
00128                             'last_seen': current_time
00129                         }
00130             except (IndexError, json.JSONDecodeError):
00131                 pass
00132
00133         elif command.startswith("LEAVE:"):
00134             user = command.split(':')[1]
00135             if user in self.peers:
00136                 del self.peers[user]
00137                 print(f"Peer verlassen: {user}")
00138
00139     def _broadcast_presence(self):
00140         """
00141         @brief Sendet regelmäßig JOIN-Broadcasts
00142         @details Informiert andere Peers über eigene Präsenz
00143         """
00144         while self.running:
00145             message = f"JOIN:{self.username}:{self.tcp_port}"
00146             self.udp_socket.sendto(message.encode(), ('<broadcast>', self.udp_port))
00147             time.sleep(self.BROADCAST_INTERVAL)
00148
00149     def _cleanup_peers(self):
00150         """
00151         @brief Entfernt inaktive Peers aus der Liste
00152         @details Prüft regelmäßig die letzte Aktivität
00153         """
00154         while self.running:
00155             current_time = time.time()
00156             to_remove = []
00157             for user, data in list(self.peers.items()):
00158                 if current_time - data['last_seen'] > self.PEER_TIMEOUT:
00159                     to_remove.append(user)
00160             for user in to_remove:
00161                 del self.peers[user]
00162                 print(f"Peer timeout: {user}")
00163             time.sleep(10)
```

6.3 main.py File Reference

Namespaces

- namespace main

Functions

- main.main (str config_path)

6.4 main.py

Go to the documentation of this file.

```
00001 """
00002 @file main.py
00003 @brief Hauptmodul für Initialisierung und Prozessmanagement
00004 """
00005
00006 import multiprocessing
00007 import toml
00008 import signal
00009 from user_interface import UserInterface
00010 from discovery_service import DiscoveryService
00011 from network_communication import NetworkCommunication
00012
00013 def main(config_path: str):
```

```
00014     """
00015     @brief Hauptfunktion des SimpleChat-Programms
00016     @param config_path Pfad zur Konfigurationsdatei
00017
00018     @details
00019     - Lädt die Konfiguration aus der TOML-Datei
00020     - Erstellt gemeinsame Datenstrukturen mit multiprocessing.Manager
00021     - Initialisiert alle Komponenten
00022     - Startet Prozesse und verwaltet Graceful Shutdown
00023     """
00024     # Konfiguration laden
00025     config = toml.load(config_path)
00026
00027     # Gemeinsame Datenstrukturen erstellen
00028     manager = multiprocessing.Manager()
00029     peers = manager.dict()
00030     command_queue = multiprocessing.Queue()
00031     response_queue = multiprocessing.Queue()
00032
00033     # Komponenten initialisieren
00034     ui = UserInterface(command_queue, response_queue)
00035     discovery = DiscoveryService(
00036         config['user']['udp_port'],
00037         peers,
00038         config['user']['name'],
00039         config['user']['tcp_port']
00040     )
00041     network = NetworkCommunication(
00042         config['user']['tcp_port'],
00043         config['user']['img_dir'],
00044         peers
00045     )
00046
00047     # Prozesse starten
00048     processes = [
00049         multiprocessing.Process(target=ui.start),
00050         multiprocessing.Process(target=discovery.run),
00051         multiprocessing.Process(target=network.start)
00052     ]
00053
00054     for p in processes:
00055         p.start()
00056
00057     # Graceful Shutdown bei SIGINT (Ctrl+C)
00058     def signal_handler(sig, frame):
00059         for p in processes:
00060             p.terminate()
00061         exit(0)
00062
00063     signal.signal(signal.SIGINT, signal_handler)
00064
00065     # Auf Prozessende warten
00066     try:
00067         for p in processes:
00068             p.join()
00069     except KeyboardInterrupt:
00070         signal_handler(None, None)
00071
00072 if __name__ == "__main__":
00073     main("config.toml")
```

6.5 network_communication.py File Reference

Classes

- class network_communication.NetworkCommunication

Namespaces

- namespace network_communication

6.6 network_communication.py

Go to the documentation of this file.

```
00001 """
00002 @file network_communication.py
00003 @brief Handhabt TCP-basierte Nachrichten- und Bildübertragung
00004 """
00005
00006 import socket
00007 import os
00008 import threading
00009 import time
00010 import json
00011 from multiprocessing import Manager
00012 from datetime import datetime
00013
00014 class NetworkCommunication:
00015     """
00016     @class NetworkCommunication
00017     @brief Verwaltet TCP-Kommunikation mit Peers
00018
00019     @details
00020     - Startet einen TCP-Server für eingehende Verbindungen
00021     - Verarbeitet Textnachrichten und Bildübertragungen
00022     - Implementiert einen Handshake-Mechanismus für zuverlässigen Bildtransfer
00023     - Speichert empfangene Bilder im konfigurierten Verzeichnis
00024     """
00025
00026     CHUNK_SIZE = 4096  # Bytes für Bildtransfer
00027
00028     def __init__(self, tcp_port: int, img_dir: str, peers: Manager().dict()):
00029         """
00030         @brief Initialisiert TCP-Server und Konfiguration
00031         @param tcp_port Port für TCP-Kommunikation
00032         @param img_dir Verzeichnis zum Speichern von Bildern
00033         @param peers Shared Dictionary der bekannten Peers
00034         """
00035         self.tcp_port = tcp_port
00036         self.tcp_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
00037         self.tcp_server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
00038         self.tcp_server.bind((”, tcp_port))
00039         self.tcp_server.listen(5)
00040         self.tcp_server.settimeout(1.0)
00041         self.img_dir = img_dir
00042         self.peers = peers
00043         self.running = True
00044
00045         # Stelle sicher, dass das Bildverzeichnis existiert
00046         os.makedirs(img_dir, exist_ok=True)
00047
00048     def start(self):
00049         """
00050         @brief Startet TCP-Server und Nachrichtenverarbeitung
00051         """
00052         accept_thread = threading.Thread(target=self._accept_connections)
00053         accept_thread.daemon = True
00054         accept_thread.start()
00055
00056         while self.running:
00057             time.sleep(0.1)
00058
00059         self.tcp_server.close()
00060
00061     def _accept_connections(self):
00062         """
00063         @brief Akzeptiert eingehende TCP-Verbindungen
00064         @details Startet für jede Verbindung einen Handler-Thread
00065         """
00066         while self.running:
00067             try:
00068                 client_socket, addr = self.tcp_server.accept()
00069                 handler_thread = threading.Thread(
00070                     target=self._handle_client,
00071                     args=(client_socket,)
00072                 )
00073                 handler_thread.daemon = True
00074                 handler_thread.start()
00075             except socket.timeout:
00076                 continue
00077             except socket.error:
00078                 if not self.running:
00079                     break
00080
00081     def _handle_client(self, client_socket: socket.socket):
00082         """
```

```
00083         @brief Verarbeitet eingehende TCP-Datenströme
00084         @param client_socket Socket des verbundenen Clients
00085         """
00086         try:
00087             # Empfange den Header (erste 1024 Bytes)
00088             header_data = client_socket.recv(1024).decode().strip()
00089             if not header_data:
00090                 return
00091 
00092             if header_data.startswith("MSG:"):
00093                 self._process_message(header_data, client_socket)
00094             elif header_data.startswith("IMG:"):
00095                 self._process_image(header_data, client_socket)
00096         finally:
00097             client_socket.close()
00098 
00099     def _process_message(self, header: str, socket: socket.socket):
00100         """
00101         @brief Verarbeitet eingehende Textnachrichten
00102         @param header Nachrichtenheader
00103         @param socket Verbindungssocket
00104 
00105         @details Nachrichtenformat:
00106         MSG:<sender>:<receiver>:<timestamp>:<message>
00107         """
00108         # Format: MSG:<Sender>:<Empfänger>:<Zeitstempel>:<Text>
00109         parts = header.split(':', 4)
00110         if len(parts) < 5:
00111             return
00112 
00113         sender = parts[1]
00114         receiver = parts[2]
00115         timestamp = parts[3]
00116         message_text = parts[4]
00117 
00118         # Hier würde die Nachricht an die UI weitergeleitet werden
00119         print(f"Nachricht von {sender}: {message_text}")
00120 
00121     def _process_image(self, header: str, socket: socket.socket):
00122         """
00123         @brief Verarbeitet Bildübertragungen mit Handshake
00124         @param header Bildmetadaten (Größe, Dateiname)
00125         @param socket Verbindungssocket
00126 
00127         @details Ablauf:
00128         1. Header parsen: IMG:<Größe>:<Dateiname>
00129         2. ACK senden
00130         3. Daten empfangen und prüfen
00131         4. Bild speichern oder verwerfen
00132         """
00133         # Format: IMG:<Größe>:<Dateiname>
00134         parts = header.split(':', 2)
00135         if len(parts) < 3:
00136             return
00137 
00138         try:
00139             size = int(parts[1])
00140             filename = parts[2]
00141         except ValueError:
00142             return
00143 
00144         # Bestätigung senden
00145         socket.sendall(b"ACK")
00146 
00147         # Bild empfangen
00148         received = 0
00149         image_data = b''
00150         while received < size:
00151             try:
00152                 chunk = socket.recv(min(self.CHUNK_SIZE, size - received))
00153                 if not chunk:
00154                     break
00155                 image_data += chunk
00156                 received += len(chunk)
00157             except socket.error:
00158                 break
00159 
00160         # Prüfen, ob vollständig
00161         if received == size:
00162             # Bild speichern
00163             filepath = os.path.join(self.img_dir, filename)
00164             with open(filepath, 'wb') as f:
00165                 f.write(image_data)
00166             print(f"Bild erfolgreich empfangen: {filepath}")
00167         else:
00168             print(f"Fehler: Unvollständiges Bild empfangen ({received}/{size} Bytes)")
```

6.7 user_interface.py File Reference

Classes

- class user_interface.UserInterface

Namespaces

- namespace user_interface

6.8 user_interface.py

Go to the documentation of this file.

```
00001 """
00002 @file user_interface.py
00003 @brief CLI-Schnittstelle für Benutzerinteraktion und Befehlsverarbeitung
00004 """
00005
00006 import threading
00007 from queue import Queue
00008 import time
00009
00010 class UserInterface:
00011     """
00012     @class UserInterface
00013     @brief Verwaltet die Kommandozeilenschnittstelle und leitet Befehle weiter
00014
00015     @details
00016     - Parst Benutzereingaben und wandelt sie in strukturierte Befehle um
00017     - Kommuniziert mit anderen Komponenten über Queues
00018     - Zeigt Systemnachrichten und eingehende Nachrichten in Echtzeit an
00019     """
00020
00021     def __init__(self, command_queue: Queue, response_queue: Queue):
00022         """
00023         @brief Konstruktor initialisiert Queues und Statusvariablen
00024         @param command_queue Queue für ausgehende Befehle
00025         @param response_queue Queue für eingehende Antworten
00026         """
00027         self.command_queue = command_queue
00028         self.response_queue = response_queue
00029         self.running = True
00030
00031     def start(self):
00032         """
00033         @brief Hauptstartroutine der CLI
00034         @details Startet zwei Threads:
00035                  1. Eingabeverarbeitung (frontend)
00036                  2. Antwortverarbeitung (backend)
00037         """
00038         input_thread = threading.Thread(target=self._input_loop)
00039         logging_thread = threading.Thread(target=self._logging_loop)
00040         input_thread.daemon = True
00041         logging_thread.daemon = True
00042         input_thread.start()
00043         logging_thread.start()
00044
00045         # Hauptthread läuft weiter, bis Shutdown
00046         while self.running:
00047             time.sleep(0.1)
00048
00049     def _input_loop(self):
00050         """
00051         @brief Verarbeitet kontinuierlich Benutzereingaben
00052         @details Parst Eingaben und leitet sie an entsprechende Module weiter
00053         """
00054         while self.running:
00055             try:
00056                 user_input = input("> ")
00057                 command = self._parse_command(user_input)
00058                 if command:
00059                     self.command_queue.put(command)
00060                     if command['type'] == 'exit':
00061                         self.running = False
```

```
00062             except EOFError:
00063                 self.running = False
00064
00065     def _parse_command(self, input_str: str) -> dict:
00066         """
00067         @brief Zerlegt Benutzereingaben in strukturierte Befehle
00068         @param input_str Roh-Eingabe des Nutzers
00069         @return Dictionary mit Schlüsseln: 'type', 'target', 'content'
00070         @retval None bei ungültigen Befehlen
00071
00072         @details Unterstützte Befehle:
00073         - join: Beitritt zum Netzwerk
00074         - who: Zeigt bekannte Peers an
00075         - msg <peer> <text>: Sendet Textnachricht
00076         - img <peer> <pfad>: Sendet Bild
00077         - leave: Verlässt das Netzwerk
00078         - config: Zeigt Konfiguration
00079         - exit: Beendet die Anwendung
00080         """
00081         parts = input_str.split(maxsplit=1)
00082         if not parts:
00083             return None
00084
00085         cmd_type = parts[0].lower()
00086         command = {'type': cmd_type}
00087
00088         if cmd_type in ('msg', 'img'):
00089             # Erwartet Format: <cmd> <target> <content>
00090             subparts = parts[1].split(maxsplit=1) if len(parts) > 1 else []
00091             if len(subparts) < 2:
00092                 print("Ungültige Eingabe: Ziel und Inhalt benötigt.")
00093                 return None
00094             command['target'] = subparts[0]
00095             command['content'] = subparts[1]
00096         elif cmd_type in ('join', 'leave', 'who', 'exit', 'config'):
00097             # Keine weiteren Argumente
00098             pass
00099         else:
00100             print(f"Unbekannter Befehl: {cmd_type}")
00101             return None
00102
00103         return command
00104
00105     def _logging_loop(self):
00106         """
00107         @brief Zeigt Systemnachrichten und eingehende Chat-Nachrichten an
00108         @details Überwacht die response_queue kontinuierlich und zeigt Nachrichten an
00109         """
00110         while self.running:
00111             if not self.response_queue.empty():
00112                 message = self.response_queue.get()
00113                 # Systemnachrichten formatieren
00114                 if isinstance(message, dict):
00115                     if message['type'] == 'system':
00116                         print(f"\n[SYSTEM] {message['content']}\n> ", end="", flush=True)
00117                     elif message['type'] == 'message':
00118                         print(f"\n[{message['sender']}] {message['content']}\n> ", end="", flush=True)
00119                 else:
00120                     print(f"\n[SYSTEM] {message}\n> ", end="", flush=True)
00121             time.sleep(0.1)
```