My Project

Generated by Doxygen 1.9.1

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

File Documentation

2.1 discovery.py File Reference

Discovery-Dienst für SLCP – empfängt JOIN, WHO, LEAVE.

Functions

- def discovery.discovery_main (konfig_pfad, shared_dict)
  @breif verwaltet alle Discovery funktionen

2.1.1 Detailed Description

Discovery-Dienst für SLCP – empfängt JOIN, WHO, LEAVE.

Verwaltet bekannte Nutzer und antwortet auf WHO-Anfragen

2.1.2 Function Documentation

2.1.2.1 discovery_main()

```
def discovery.discovery_main (
              konfig_pfad,
              shared_dict )
```

@breif verwaltet alle Discovery funktionen

Parameters

konfig_pfad	Pfad der individuellen Config
shared_dict	Nutzerliste

2.2 interface.py File Reference

interface der Kommmandozeile für das Chat-Programm

Functions

- def interface.menue ()

  Auswahl.
- def interface.nutzernamen_abfragen ()

  Nutzername eingeben.
- def interface.eingabe_nachricht ()

  Eingabe der zu versendend Nachricht an eine bestimmten Empfänger.
- def interface.eingabe_bild ()

  Eingabe des Bildpfads und des Empfängers.
- def interface.autoreply_umschalten (config, pfad)

  Eingabe der Autoreply Nachricht.
- def interface.autoreply_einschalten (config, pfad)

  umschalten der autoreply
- def interface.lade_config (pfad=None)

  lädt die TOML-Konfigurationsdatei und gibt sie als Dictionary zurück (Parsing).
- def interface.erstelle_neue_config (handle)

  Erstellt eine neue Konfigurationsdatei für den Benutzer, falls sie noch nicht existiert.
- def interface.finde_freien_tcp_port ()

  Findet einen freien TCP-Port auf dem lokalen System.
- def interface.finde_freien_port (config)

  Durchsucht den in der Konfigurationsdatei angegebenen Portbereich nach einem freien UDP-Port.

2.2.1 Detailed Description

interface der Kommmandozeile für das Chat-Programm

Menüauswahl und Eingaben über das Terminal

2.2.2 Function Documentation

2.2.2.1 autoreply_einschalten()

```
def interface.autoreply_einschalten (
            config,
            pfad )
```

umschalten der autoreply

Parameters

config	indiviuelle config
pfad	speicherort der Config-Datei

Returns

Die Config-Datei wird verändert

Note

schaut, welcher wert in Config steht und verändert ihn

2.2.2.2 autoreply_umschalten()

```
def interface.autoreply_umschalten (
              config,
              pfad )
```

Eingabe der Autoreply Nachricht.

Parameters

config	indiviuelle config
pfad	speicherort der Config-Datei

Returns

Die Config-Datei wird verändert

Note

gibt erst die aktuelle nachricht aus und fragt dann nach einer neuen

2.2.2.3 eingabe_bild()

```
def interface.eingabe_bild ( )
```

Eingabe des Bildpfads und des Empfängers.

Returns

empfänger und bildpfad als String

2.2.2.4 eingabe_nachricht()

```
def interface.eingabe_nachricht ( )
```

Eingabe der zu versendend Nachricht an eine bestimmten Empfänger.

Returns

empfänger und nachricht als String

2.2.2.5 erstelle_neue_config()

```
def interface.erstelle_neue_config (
              handle )
```

Erstellt eine neue Konfigurationsdatei für den Benutzer, falls sie noch nicht existiert.

Parameters

handle	Der Benutzername für den Benutzer, der eine neue datei braucht.

Note

Standard-Konfigurationswerte für den Benutzer

2.2.2.6 finde_freien_port()

```
def interface.finde_freien_port (
              config )
```

Durchsucht den in der Konfigurationsdatei angegebenen Portbereich nach einem freien UDP-Port.

Parameters

config	dict mit den Konfigurationswerten, erwartet Schlüssel 'port_min' und 'port_max'

Returns

int erster freier UDP-Port im Bereich

socket

Note

ValueError Wenn 'port_min' oder 'port_max' in der Konfiguration fehlen

RuntimeError wenn kein freier Port im Bereich gefunden wird

2.2.2.7 finde_freien_tcp_port()

```
def interface.finde_freien_tcp_port ( )
```

Findet einen freien TCP-Port auf dem lokalen System.

Diese Funktion erstellt einen temporären TCP-Socket, der sich an Port 0 bindet. Die Angabe von Port 0 signalisiert dem Betriebssystem, dass ein beliebiger freier Port automatisch zugewiesen werden soll. Nachdem der Socket gebunden ist, wird der tatsächlich zugewiesene Port abgefragt und zurückgegeben.

Returns

einen freieren TCP-Port (int), der aktuell nicht verwendet wird.

2.2.2.8 lade_config()

```
def interface.lade_config (
              pfad = None )
```

lädt die TOML-Konfigurationsdatei und gibt sie als Dictionary zurück (Parsing).

Parameters

pfad=None	Pfad überschreiben oder pfad nicht übergeben, dann nimmt er standard Config Datei

Returns

dict mit den Konfigurationswerten

Note

FileNotFoundError wenn die Datei nicht existiert

2.2.2.9 menue()

```
def interface.menue ( )
```

Auswahl.

Returns

Auswahl des Benutzers als String

Note

wird in der main aufgerufen

2.2.2.10 nutzernamen_abfragen()

```
def interface.nutzernamen_abfragen ( )
```

Nutzername eingeben.

Returns

Benutzername (Handle) als String

Note

Fragt den Benutzernamen ab und erstellt bei Bedarf eine neue Konfigurationsdatei.

2.3 main.py File Reference

Start des Programms und ruft die ganzen funktionien auf.

Functions

- def main.registriere_neuen_nutzer (handle, config)

  Registiert einen neuen Nutzer im Chatnetzwerk.
- def main.sende_befehl_an_netzwerkprozess (str befehl, int tcp_port)

  Sendet einen Steuerbefehl über einen lokalen TCP-Socket an den Netzwerkprozess.
- def main.main ()

  startet alle funktionienen nach eingabe durch eingabe im Terminal

Variables

- **main.force**

2.3.1 Detailed Description

Start des Programms und ruft die ganzen funktionien auf.

2.3.2 Function Documentation

2.3.2.1 main()

```
def main.main ( )
```

startet alle funktionienen nach eingabe durch eingabe im Terminal

lädt das Menü und verwaltet den Ablauf

2.3.2.2 registriere_neuen_nutzer()

```
def main.registriere_neuen_nutzer (
              handle,
              config )
```

Registiert einen neuen Nutzer im Chatnetzwerk.

finde_freien_port: es wird ein freier Port gesucht und ein Socket dadurch erstellt

send_join: verschickt eine Join Nachricht an alle

Parameters

handle	der Benutzername des Teilnehmers
config	die config Datei wird geldaden

Returns

port: verwendeter UDP-Port des Nutzers nutzer_sock: der UDP-Socket, der an den Port gebunden ist

2.3.2.3 sende_befehl_an_netzwerkprozess()

```
def main.sende_befehl_an_netzwerkprozess (
             str befehl,
             int tcp_port )
```

Sendet einen Steuerbefehl über einen lokalen TCP-Socket an den Netzwerkprozess.

Diese Funktion wird vom UI-Prozess verwendet, um Nachrichten- oder Bildbefehle (z.B. MSG oder IMG) an den Netzwerkprozess weiterzuleiten. Der Netzwerkprozess übernimmt dann das eigentliche Senden per UDP an andere Chat-Teilnehmer. Die Kommunikation erfolgt über eine TCP-Verbindung zu localhost:6001

Parameters

befehl	Der SLCP-kompatible Befehl, z.B. "MSG Bob Hallo" oder "IMG Bob pfad/zum/bild.jpg".
tcp_port	verbindet netzwerkprozess mit Main

Note

Wenn der Netzwerkprozess nicht läuft oder der Socket nicht erreichbar ist, wird eine Fehlermeldung ausgegeben.

2.4 netzwerkprozess.py File Reference

empfängt und verarbeitet IPC-Kommandos vom UI-Prozess

Functions

- def netzwerkprozess.finde_lokale_ip ()
  lokale IP-Adresse wird gesucht
- def netzwerkprozess.send_join (sock, handle, port, DISCOVERY_PORT)
  @breif versendet eine Join-Nachricht
- def netzwerkprozess.handle_join (name, DISCOVERY_PORT, addr, ip=None)
  Verarbeitet eine empfangene Join nachricht.
- def netzwerkprozess.send_leave (sock, handle_nutzername, DISCOVERY_PORT)
  sendet eine leave nachricht an alle Nutzer
- def netzwerkprozess.handle_leave (name)
  entfernt einen nutzer aus der Nutzerliste
- def netzwerkprozess.sendMSG (sock, handle, empfaenger_handle, text)
  sendet eine nachricht an einen nutzer
- def netzwerkprozess.receive_MSG (sock, config)
  Empfängt Nachrichten vom UDP-Socket und verarbeitet sie.
- def netzwerkprozess.send_IMG (sock, handle_empfaenger, bildpfad)
  Sendet ein Bild an einen anderen Nutzer über UDP.
- def netzwerkprozess.handle_IMG (sock, teile, addr, config)
  Verarbeitet eine empfangene IMG-Nachricht und speichert das Bild.
- def netzwerkprozess.netzwerkprozess (sock, konfig_pfad, tcp_port)
  Startet den Netzwerkprozess und lauscht auf Befehle vom UI-Prozess.
- def **netzwerkprozess.starte_netzwerkprozess** (konfig_pfad, tcp_port, port, shared_dict)

2.4.1 Detailed Description

empfängt und verarbeitet IPC-Kommandos vom UI-Prozess

sendet SLCP-Nachrichten (MSG, IMG) per UDP an andere Peers im Netzwerk.

2.4.2 Function Documentation

2.4.2.1 finde_lokale_ip()

```
def netzwerkprozess.finde_lokale_ip ( )
```

lokale IP-Adresse wird gesucht

Ermittelt die echte lokale IP-Adresse (z.B. WLAN) durch Verbindung zu 8.8.8.8.

2.4.2.2 handle_IMG()

```
def netzwerkprozess.handle_IMG (
              sock,
              teile,
              addr,
              config )
```

Verarbeitet eine empfangene IMG-Nachricht und speichert das Bild.

Parameters

sock	Der UDP-Socket, über den das Bild empfangen wird
teile	Die Teile der empfangenen Textnachricht (z.B. ["IMG", "empfänger", "Größe"])
addr	Die Adresse (IP, Port) des Absenders
config	Die Konfiguration des Clients (z.B. mit Handle und Speicherpfad)

2.4.2.3 handle_join()

```
def netzwerkprozess.handle_join (
              name,
              DISCOVERY_PORT,
              addr,
              ip = None )
```

Verarbeitet eine empfangene Join nachricht.

Parameters

name	des beitretenden Nutzers (handle)
Discovery_port	port von Discovery dienst
addr	die absender adresse
ip=None	falls eine Ip adresse bekannt ist, sonst None

2.4.2.4 handle_leave()

```
def netzwerkprozess.handle_leave (
              name )
```

entfernt einen nutzer aus der Nutzerliste

Parameters

name	nutzername

2.4.2.5 netzwerkprozess()

```
def netzwerkprozess.netzwerkprozess (
              sock,
              konfig_pfad,
              tcp_port )
```

Startet den Netzwerkprozess und lauscht auf Befehle vom UI-Prozess.

Stellt einen TCP-Server auf localhost:tcpPort bereit, über den der UI-Prozess Kommandos wie MSG und IMG senden kann. Diese werden analysiert und mit UDP an die Ziel-Peers gemäß SLCP-Protokoll weitergeleitet.

Note

Diese Funktion blockiert dauerhaft. Sie sollte in einem separaten Prozess ausgeführt werden.

Parameters

sock	UDP-Socket
konfig_pfad	Pfad der aktuellen Config @tcp_port verbindung zwischen Main und Netzwerkprozess

2.4.2.6 receive_MSG()

```
def netzwerkprozess.receive_MSG (
              sock,
              config )
```

Empfängt Nachrichten vom UDP-Socket und verarbeitet sie.

Parameters

sock	UDP-Socket
config	individuelle Konfiguartions Datei

2.4.2.7 send_IMG()

```
def netzwerkprozess.send_IMG (
              sock,
              handle_empfaenger,
              bildpfad )
```

Sendet ein Bild an einen anderen Nutzer über UDP.

Parameters

sock	Der UDP-Socket zum Senden der Nachricht
handle_empfaenger	Benutzername des Empfängers
bildpfad	Pfad zur Bilddatei, die gesendet werden soll

2.4.2.8 send_join()

```
def netzwerkprozess.send_join (
              sock,
              handle,
              port,
              DISCOVERY_PORT )
```

@breif versendet eine Join-Nachricht

Parameters

sock	UDP-Socket
handel	Benutzername
port	eigener Port
Discovery_port	Broadcast-Port

2.4.2.9 send_leave()

```
def netzwerkprozess.send_leave (
              sock,
              handle_nutzername,
              DISCOVERY_PORT )
```

sendet eine leave nachricht an alle Nutzer

Parameters

sock	UDP socket
handle_nutzername	name des lokalen Nutzers
Discovery_Port	Broadcast-Port

2.4.2.10 sendMSG()

```
def netzwerkprozess.sendMSG (
              sock,
```

```
            handle,
            empfaenger_handle,
            text )
```

sendet eine nachricht an einen nutzer

Parameters

sock	UDP-Soket
handle	absender handle
empfaenger_handle	empfänger handle
text	Nachricht

2.5 nutzerliste.py File Reference

Verwaltung der gemeinsam genutzten Nutzerliste im SLCP-Chatprojekt.

Functions

- def nutzerliste.initialisiere_nutzerliste (shared_dict)

  Erstellt die globale Nutzerliste mit einem gemeinsam nutzbaren Dictionary.
- def nutzerliste.gebe_nutzerliste_zurück ()

  gibt nutzerliste zurück

2.5.1 Detailed Description

Verwaltung der gemeinsam genutzten Nutzerliste im SLCP-Chatprojekt.

es wird eine nutzerliste gespeichert, die vom Discovery und netzwerkprozess bearbeitet wird.

2.5.2 Function Documentation

2.5.2.1 gebe_nutzerliste_zurück()

```
def nutzerliste.gebe_nutzerliste_zurück ( )
```

gibt nutzerliste zurück

Returns

nutzerliste

2.5.2.2 initialisiere_nutzerliste()

```
def nutzerliste.initialisiere_nutzerliste (
            shared_dict )
```

Erstellt die globale Nutzerliste mit einem gemeinsam nutzbaren Dictionary.

Parameters

shared_dict	Globale Variable für die geteilte Nutzerliste

Note

wird von multiprocessing.Manager() erzeugt