My Project

Generated by Doxygen 1.9.8

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 cli_process.py File Reference

BYMY CLI-Prozess – steuert das Terminal-Interface für den Nutzer.

Functions

- cli_process.update_config_value (key, value)

  2) Schreibt neuen Wert in config.toml.
- **cli_process.show_intro** ()

  3) Zeigt Hilfe und Befehlsübersicht an.
- cli_process.recover_pipe (pipe_name)

  4) Erstellt Pipe neu, falls sie defekt ist.
- cli_process.send_pipe_command (cmd)

  5) Sendet Befehl an Netzwerkprozess.
- cli_process.find_file (name)

  7) Sucht Datei im Home-Verzeichnis.
- **cli_process.run_cli** ()

  8) Haupt-CLI-Loop: steuert alle Befehle.

Variables

- str **cli_process.RESET** = "\033[91m"

  1) Farbdefinitionen & Pfade
- str **cli_process.CYAN** = "\033[1m"
- str **cli_process.AWAY_FLAG** = "away.flag"
- str **cli_process.CONFIG_FILE** = "config.toml"
- str **cli_process.PIPE_CLI_TO_NET** = "cli_to_network.pipe"
- str **cli_process.PIPE_NET_TO_CLI** = "network_to_cli.pipe"
- **cli_process.offline_txt** = os.path.join("receive", "offline_messages.txt")
- dict **cli_process.known_users** = {}
- **cli_process.current_chat** = None
- **cli_process.received_leave_ack** = threading.Event()
- **cli_process.target**
- **cli_process.listen_pipe_loop** ()

  6) Lauscht auf NET->CLI Pipe & verarbeitet Nachrichten.
- **cli_process.daemon**

2.1.1 Detailed Description

BYMY CLI-Prozess – steuert das Terminal-Interface für den Nutzer.

Ablauf: 1) Farben, Pipes, Flags, globale Variablen 2) update_config_value: Speichert Änderungen in der TOML-↩ Konfigurationsdatei 3) show_intro: Zeigt dem Nutzer alle verfügbaren Kommandos an 4) recover_pipe: Erstellt eine Pipe neu, falls sie fehlt oder defekt ist 5) send_pipe_command: Sendet ein Kommando an den Netzwerkprozess über Pipe 6) listen_pipe_loop: Lauscht auf die Netzwerkantwort-Pipe und verarbeitet Daten 7) find_file: Durchsucht das Home-Verzeichnis nach Dateien 8) run_cli: Führt die Haupt-CLI-Steuerung aus (Kommandos, Chat) 9) **main**: Initialisiert Pipes & Threads und startet CLI

2.1.2 Function Documentation

2.1.2.1 find_file()

```
cli_process.find_file (
              name )
```

7) Sucht Datei im Home-Verzeichnis.

Parameters

name	Beginn des Dateinamens

Returns

Pfad zur Datei oder None

2.1.2.2 recover_pipe()

```
cli_process.recover_pipe (
              pipe_name )
```

4) Erstellt Pipe neu, falls sie defekt ist.

Parameters

pipe_name	Dateiname der Pipe

2.1.2.3 send_pipe_command()

```
cli_process.send_pipe_command (
              cmd )
```

5) Sendet Befehl an Netzwerkprozess.

Parameters

cmd	Befehl als String

2.1.2.4 update_config_value()

```
cli_process.update_config_value (
              key,
              value )
```

2) Schreibt neuen Wert in config.toml.

Parameters

key	Schlüssel in der Config
value	Neuer Wert

2.2 config_handler.py File Reference

Lädt & verwaltet die zentrale Konfigurationsdatei (config.toml)

Functions

- config_handler.get_config ()
  Lädt die gesamte Konfiguration aus der TOML-Datei.
- config_handler.get_config_value (key, default=None)
  Gibt gezielt einen einzelnen Wert aus der Konfig zurück.

Variables

- str **config_handler.CONFIG_FILE** = "config.toml"

2.2.1 Detailed Description

Lädt & verwaltet die zentrale Konfigurationsdatei (config.toml)

Zweck: 1) Pfad zur globalen Konfigdatei speichern. 2) Funktion zum vollständigen Laden (Dict). 3) Funktion zum gezielten Lesen einzelner Werte. Damit greift jedes andere Modul konsistent auf die Chat-Einstellungen zu.

2.2.2 Function Documentation

2.2.2.1 get_config()

```
config_handler.get_config ( )
```

Lädt die gesamte Konfiguration aus der TOML-Datei.

Returns

Ein Dictionary mit allen Konfigwerten.

Exceptions

FileNotFoundError,wenn	Datei nicht existiert.

2.2.2.2 get_config_value()

```
config_handler.get_config_value (
              key,
              default = None )
```

Gibt gezielt einen einzelnen Wert aus der Konfig zurück.

Parameters

key	Schlüsselname (z.B. 'handle', 'port', 'whoisport')
default	Optionaler Rückgabewert, falls Key nicht vorhanden.

Returns

Der Wert aus der Datei oder der Default.

2.3 discovery_process.py File Reference

Discovery-Modul für BYMY Chat (UDP-Broadcast-Discovery nach SLCP-Art)

Functions

- discovery_process.run_discovery_process (whoisport)

  Discovery-Hauptprozess: Verwaltet Teilnehmerliste und antwortet auf Anfragen.

Variables

- str **discovery_process.RESET** = "\033[0m"
- str **discovery_process.YELLOW** = "\033[93m"
- **discovery_process.config** = get_config()

2.3.1 Detailed Description

Discovery-Modul für BYMY Chat (UDP-Broadcast-Discovery nach SLCP-Art)

Ablauf & Zweck: 1) Öffnet einen UDP-Socket am WHOIS-Port für Discovery. 2) Wartet endlos auf JOIN, LEAVE oder WHO Nachrichten. 3) JOIN: Speichert neuen Nutzer, broadcastet an bekannte. 4) LEAVE: Entfernt Nutzer, broadcastet Austritt an bekannte. 5) WHO: Antwortet mit allen bekannten Nutzern, wenn Absender bekannt ist.

Damit stellt das Modul sicher, dass jeder Client dynamisch andere Clients im LAN finden kann, ohne zentralen Server.

2.3.2 Function Documentation

2.3.2.1 run_discovery_process()

```
discovery_process.run_discovery_process (
              whoisport )
```

Discovery-Hauptprozess: Verwaltet Teilnehmerliste und antwortet auf Anfragen.

Parameters

whoisport	UDP-Port für WHO/JOIN/LEAVE-Kommunikation.

2.4 network_process.py File Reference

BYMY Netzwerkprozess für Nachrichten- und Bildübertragung.

Functions

- network_process.write_to_cli (msg)
  2) Nachricht in CLI-Pipe schreiben.
- network_process.send_who (whoisport)
  3) Sende WHO Broadcast.
- network_process.send_join (handle, port, whoisport)
  4) Sende JOIN Broadcast.
- network_process.send_leave (handle, whoisport)
  5) Sende LEAVE Broadcast.
- network_process.send_msg (to_handle, text, known_users, my_handle)
  6) Sende Textnachricht.
- network_process.send_image (to_handle, filepath, filesize, known_users, config)
  7) Sende Bild über TCP.
- network_process.tcp_image_receiver (port, config)
  8) TCP-Server für Bilder.
- network_process.handle_tcp_connection (conn, addr, image_dir)
  9) Speichert empfangenes TCP-Bild.
- network_process.read_cli_pipe (config)
  10) Liest CLI-Kommandos.
- network_process.listen_on_port (port, config)
  11) Lauscht auf Port & verarbeitet.
- network_process.handle_sigterm (signum, frame)
  12) SIGTERM-Handler.

Variables

- str **network_process.RESET** = "\033[96m"

  1) Farben & Pipes
- str **network_process.YELLOW** = "\033[92m"
- str **network_process.PIPE_CLI_TO_NET** = "cli_to_network.pipe"
- str **network_process.PIPE_NET_TO_CLI** = "network_to_cli.pipe"
- str **network_process.AWAY_FLAG** = "away.flag"
- **network_process.autoreplied_to** = set()
- dict **network_process.known_users** = {}
- **network_process.config** = get_config()
- **network_process.port** = config["port"][0]
- **network_process.target**
- **network_process.tcp_image_receiver**
- **network_process.args**
- **network_process.daemon**
- **network_process.listen_on_port**

2.4.1 Detailed Description

BYMY Netzwerkprozess für Nachrichten- und Bildübertragung.

Struktur & Ablauf: 1) Farben & Pipes 2) write_to_cli – Nachricht an CLI zurückschreiben 3) send_who – WHO an Broadcast 4) send_join – JOIN an Broadcast 5) send_leave – LEAVE an Broadcast 6) send_msg – Textnachricht an User 7) send_image – Bild über TCP senden 8) tcp_image_receiver – TCP-Server für Bilder 9) handle_tcp_↩ connection – TCP-Bild speichern 10) read_cli_pipe – CLI-Kommandos lesen 11) listen_on_port – UDP-Nachrichten empfangen 12) handle_sigterm – Sauberer Shutdown 13) start() – Startet alles

2.4.2 Function Documentation

2.4.2.1 handle_sigterm()

```
network_process.handle_sigterm (
              signum,
              frame )
```

12) SIGTERM-Handler.

Parameters

signum	Signal.
frame	Frame.

2.4.2.2 handle_tcp_connection()

```
network_process.handle_tcp_connection (
              conn,
              addr,
              image_dir )
```

9) Speichert empfangenes TCP-Bild.

Parameters

conn	TCP-Verbindung.
addr	Absenderadresse.
image_dir	Zielordner.

2.4.2.3 listen_on_port()

```
network_process.listen_on_port (
              port,
              config )
```

11) Lauscht auf Port & verarbeitet.

Parameters

port	UDP-Port.
config	Config.

2.4.2.4 read_cli_pipe()

```
network_process.read_cli_pipe (
              config )
```

10) Liest CLI-Kommandos.

Parameters

config	Globale Config.

2.4.2.5 send_image()

```
network_process.send_image (
              to_handle,
              filepath,
              filesize,
              known_users,
              config )
```

7) Sende Bild über TCP.

Parameters

to_handle	Empfänger.
filepath	Datei.
filesize	Größe.
known_users	Bekannte Nutzer.
config	Config.

2.4.2.6 send_join()

```
network_process.send_join (
              handle,
              port,
              whoisport )
```

4) Sende JOIN Broadcast.

Parameters

handle	Eigenes Handle.
port	Eigener UDP-Port.
whoisport	Discovery-Port.

2.4.2.7 send_leave()

```
network_process.send_leave (
              handle,
              whoisport )
```

5) Sende LEAVE Broadcast.

Parameters

handle	Eigenes Handle.
whoisport	Discovery-Port.

2.4.2.8 send_msg()

```
network_process.send_msg (
              to_handle,
              text,
              known_users,
              my_handle )
```

6) Sende Textnachricht.

Parameters

to_handle	Empfänger.
text	Nachricht.
known_users	Bekannte Nutzer.
my_handle	Eigener Name.

2.4.2.9 send_who()

```
network_process.send_who (
```

```
            whoisport )
```

3) Sende WHO Broadcast.

Parameters

whoisport	Discovery-Port.

2.4.2.10 tcp_image_receiver()

```
network_process.tcp_image_receiver (
            port,
            config )
```

8) TCP-Server für Bilder.

Parameters

port	UDP-Port.
config	Config.

2.4.2.11 write_to_cli()

```
network_process.write_to_cli (
            msg )
```

2) Nachricht in CLI-Pipe schreiben.

Parameters

msg	Die Nachricht

Index