UDP Client Script

The UDP client script provides a GUI-based interface for users to send and receive messages using a UDP connection. Upon entering a username and joining, the client sends the username as its first message to the server and starts a background thread to listen for broadcast messages using `recvfrom()`. Messages typed into the GUI are sent using `sendto()` to the server's IP and port. The client GUI updates the chat window with incoming messages and ensures that the listening thread is cleanly stopped and the socket closed when the application exits. Like the server, it operates without establishing a persistent connection, relying on UDP's datagram model.

Here is a detailed explanation for the UDP client script of the multi-client chat application.

File Header and Imports

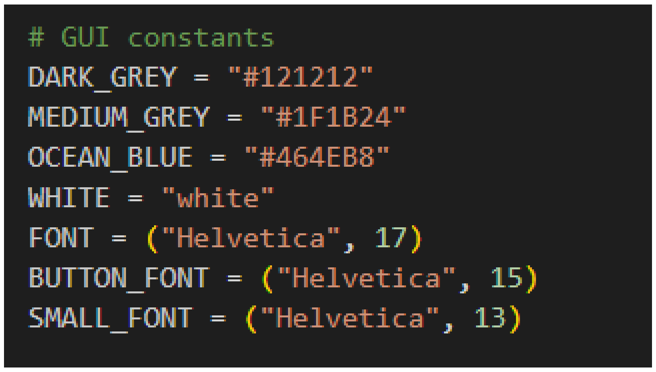
```
#!/usr/bin/python3
"""UDP Client Script"""
```

- **Shebang**: Indicates that the script should be executed using Python 3.
- **Docstring**: Describes the purpose of the script, which is to implement a UDP client.

```
import threading
import tkinter as tk
from socket import AF_INET, SOCK_DGRAM, socket
from tkinter import messagebox, scrolledtext
```

Imports:
- ❖ **`threading`** : Allows the creation of threads for concurrent execution, enabling the client to listen for messages while allowing user interaction.
- ❖ **`tkinter`** : A library for creating graphical user interfaces (GUIs).
- ❖ **`socket`** : Provides access to the BSD socket interface for network communication.
- ❖ **`messagebox`** and **`scrolledtext`** : Specific tkinter components for user interaction and
- ❖ displaying text.

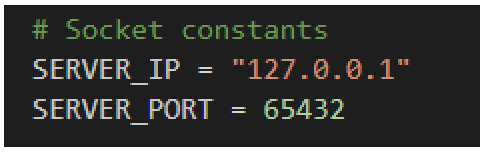
GUI Constants



```
# GUI constants
DARK_GREY = "#121212"
MEDIUM_GREY = "#1F1B24"
OCEAN_BLUE = "#464EB8"
WHITE = "white"
FONT = ("Helvetica", 17)
BUTTON_FONT = ("Helvetica", 15)
SMALL_FONT = ("Helvetica", 13)
```

- **Color and Font Constants**: Define colors and font styles used in the GUI for consistency and readability.

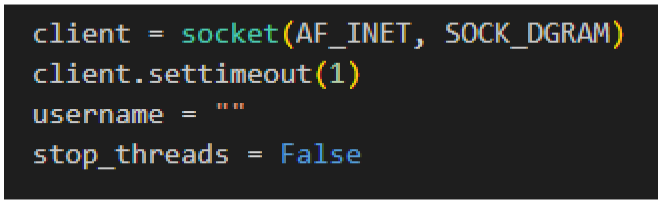
Socket Constants



```
# Socket constants
SERVER_IP = "127.0.0.1"
SERVER_PORT = 65432
```

- **Server Configuration**:
    - ❖ `SERVER_IP` : The IP address of the server to connect to (localhost in this case).
    - ❖ `SERVER_PORT` : The port number on which the server is listening for incoming UDP packets.

Socket Initialization

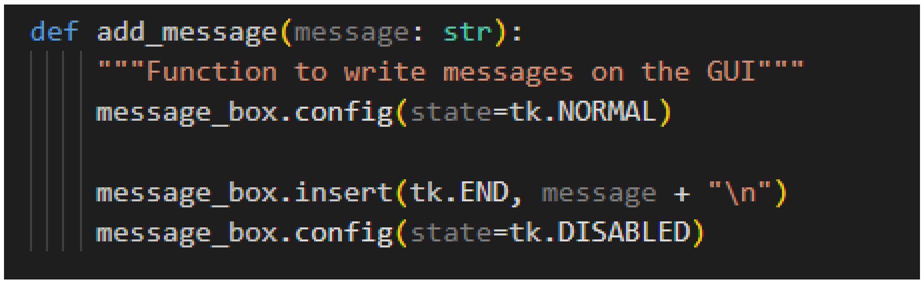


```
client = socket(AF_INET, SOCK_DGRAM)
client.settimeout(1)
username = ""
stop_threads = False
```

- **Socket Creation**:
    - ❖ `socket(AF_INET, SOCK_DGRAM)` : Creates a UDP socket using IPv4.
        - ➢ `AF_INET` : Address family for IPv4.
        - ➢ `SOCK_DGRAM` : Socket type for UDP.
    - ❖ `settimeout(1)` : Sets a timeout of 1 second for blocking socket operations, which helps prevent the program from hanging indefinitely while waiting for messages.
- **Variables**:
    - ❖ `username` : Stores the username entered by the user.
    - ❖ `stop_threads` : A flag to control the termination of threads.

Function Definitions

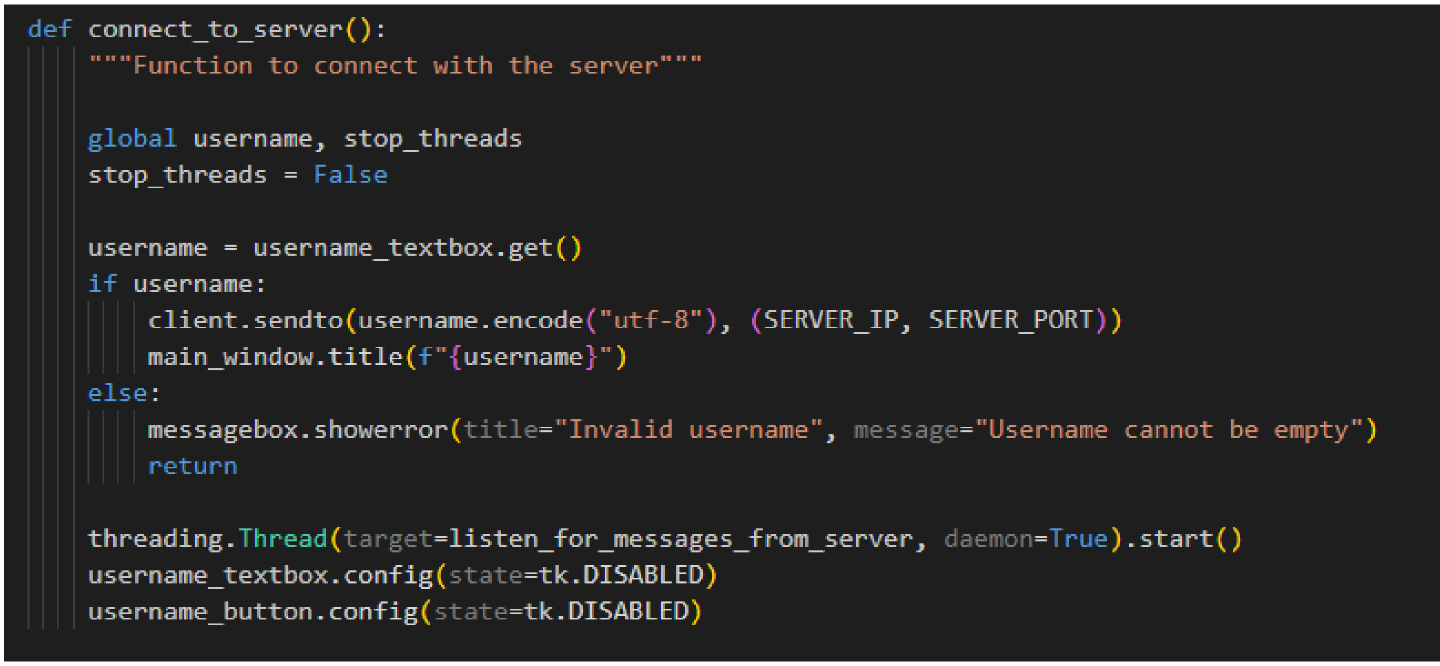
1. Add Message Function



```python
def add_message(message: str):
    """Function to write messages on the GUI"""
    message_box.config(state=tk.NORMAL)

    message_box.insert(tk.END, message + "\n")
    message_box.config(state=tk.DISABLED)
```

- **Purpose**: Updates the message box in the GUI with new messages.
- **Functionality**:
    - ❖ Enables the message box, inserts the new message at the end, and then disables it to prevent user edits.

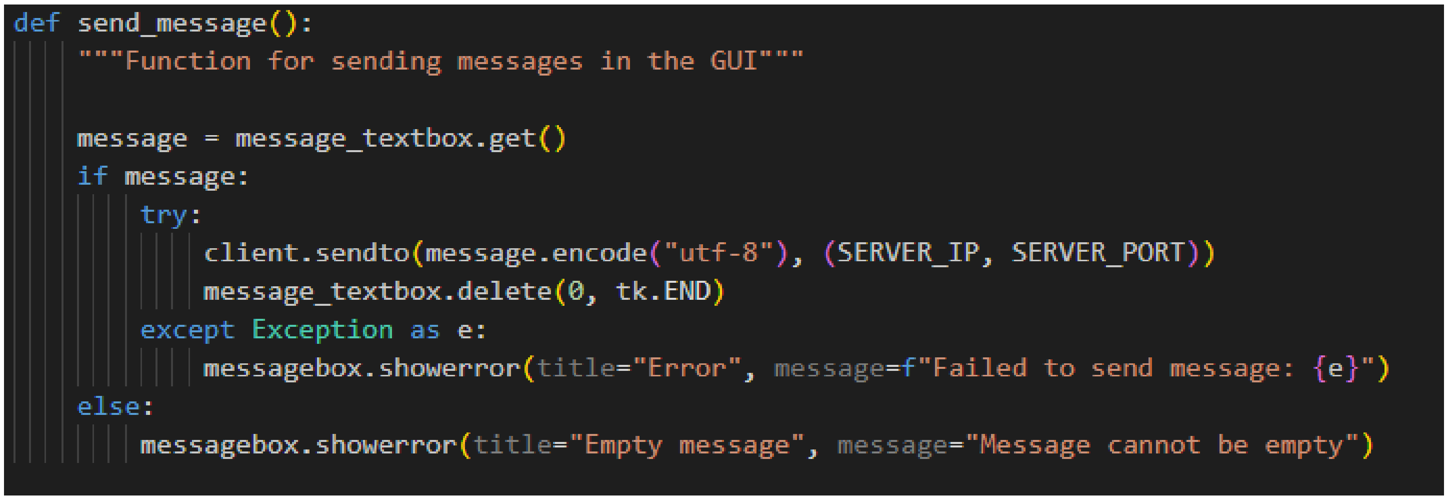
2. Connect to Server Function



```python
def connect_to_server():
    """Function to connect with the server"""

    global username, stop_threads
    stop_threads = False

    username = username_textbox.get()
    if username:
        client.sendto(username.encode("utf-8"), (SERVER_IP, SERVER_PORT))
        main_window.title(f"{username}")
    else:
        messagebox.showerror(title="Invalid username", message="Username cannot be empty")
        return

    threading.Thread(target=listen_for_messages_from_server, daemon=True).start()
    username_textbox.config(state=tk.DISABLED)
    username_button.config(state=tk.DISABLED)
```

- **Purpose**: Connects the client to the server using the provided username.
- **Functionality**:
    - ❖ Retrieves the username from the input box.
    - ❖ Sends the username to the server using `sendto()` , which is specific to UDP.
    - ❖ Starts a new thread to listen for incoming messages from the server, allowing the GUI to remain responsive.
    - ❖ Disables the username input and button to prevent changes after joining.

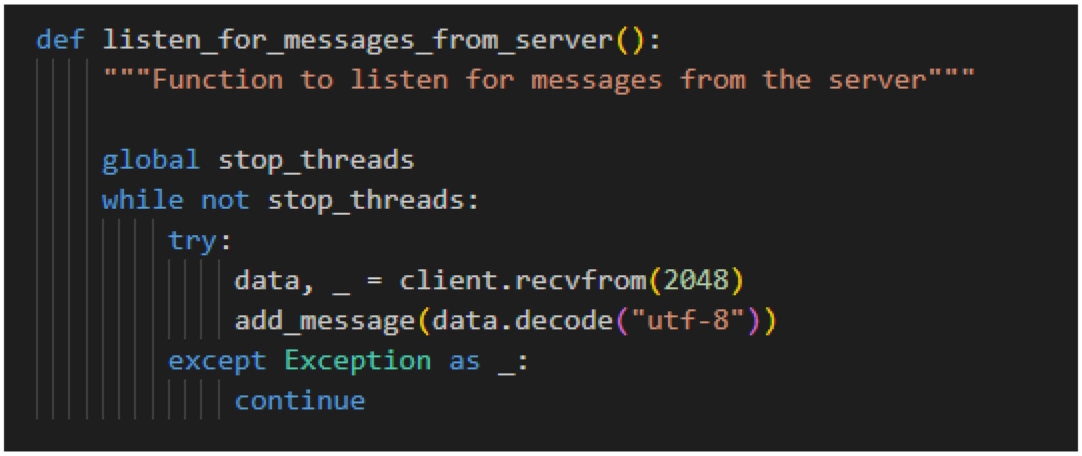
3. Send Message Function



```python
def send_message():
    """Function for sending messages in the GUI"""

    message = message_textbox.get()
    if message:
        try:
            client.sendto(message.encode("utf-8"), (SERVER_IP, SERVER_PORT))
            message_textbox.delete(0, tk.END)
        except Exception as e:
            messagebox.showerror(title="Error", message=f"Failed to send message: {e}")
    else:
        messagebox.showerror(title="Empty message", message="Message cannot be empty")
```

- **Purpose**: Sends a message entered by the user to the server.
- **Functionality**:
    - ❖ Retrieves the message from the input box.
    - ❖ Sends the message using `sendto()` , which sends the message to the specified server address.
    - ❖ Clears the input box after sending.
    - ❖ Handles exceptions and shows error messages if sending fails.

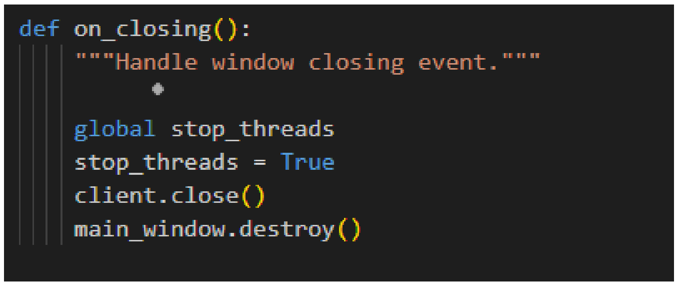
4. Listen for Messages Function



```python
def listen_for_messages_from_server():
    """Function to listen for messages from the server"""

    global stop_threads
    while not stop_threads:
        try:
            data, _ = client.recvfrom(2048)
            add_message(data.decode("utf-8"))
        except Exception as _:
            continue
```

- **Purpose:** Continuously listens for messages from the server.
- **Functionality:**
    - ❖ Uses a loop that runs until `stop_threads` is set to True .
    - ❖ Receives messages using `recvfrom()` , which is specific to UDP and returns the data
    - ❖ and the address of the sender.
    - ❖ Calls `add_message()` to display the received message in the GUI.

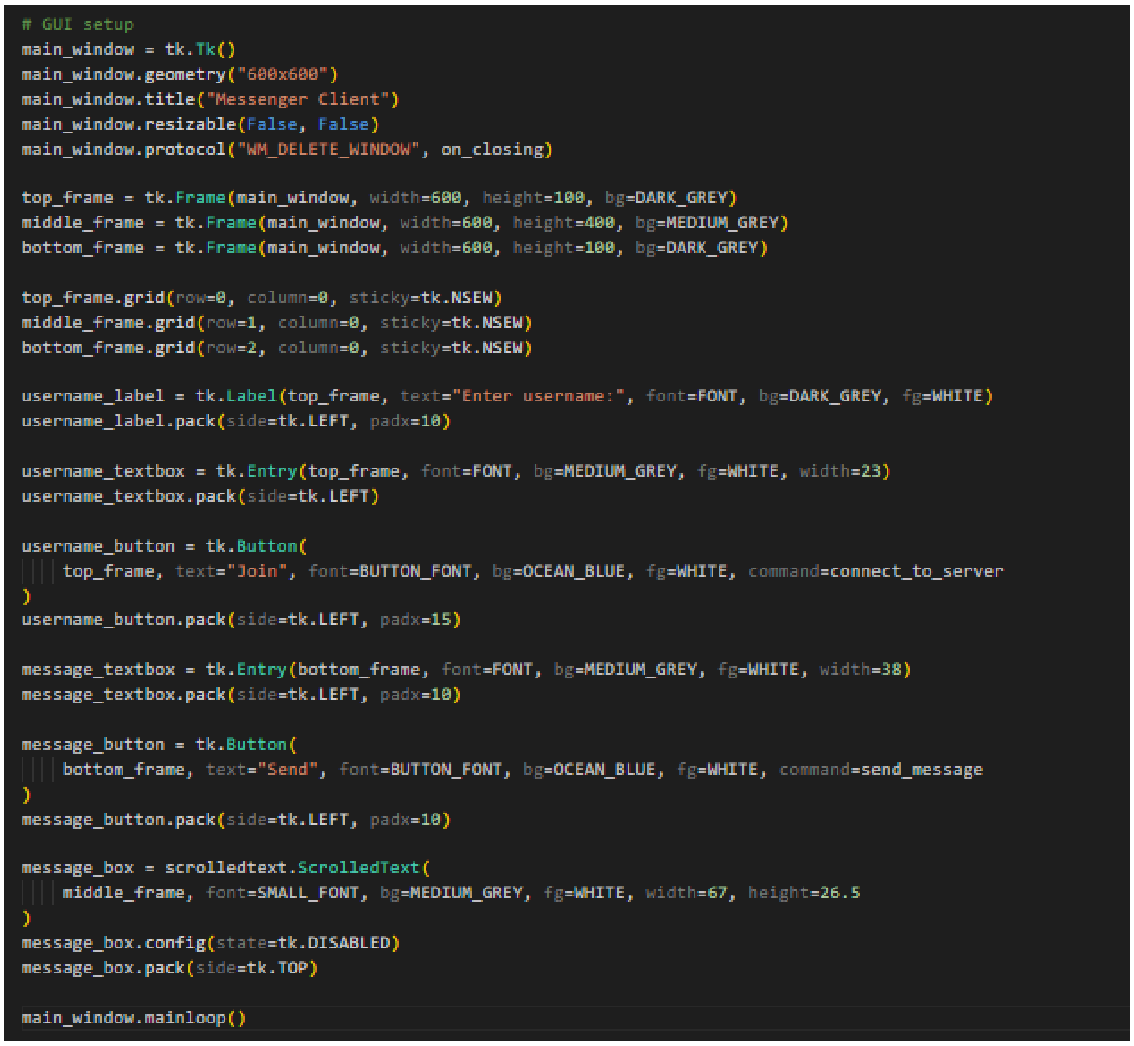
5. On Closing Function



```python
def on_closing():
    """Handle window closing event."""

    global stop_threads
    stop_threads = True
    client.close()
    main_window.destroy()
```

- Purpose: Handles the event when the user closes the application.
- Functionality:
    - Sets **`stop_threads`** to True to stop the listening thread.
    - Closes the socket connection using **`client.close()`.**
    - Destroys the main window to exit the application

GUI Setup



```python
# GUI setup
main_window = tk.Tk()
main_window.geometry("600x600")
main_window.title("Messenger Client")
main_window.resizable(False, False)
main_window.protocol("WM_DELETE_WINDOW", on_closing)

top_frame = tk.Frame(main_window, width=600, height=100, bg=DARK_GREY)
middle_frame = tk.Frame(main_window, width=600, height=400, bg=MEDIUM_GREY)
bottom_frame = tk.Frame(main_window, width=600, height=100, bg=DARK_GREY)

top_frame.grid(row=0, column=0, sticky=tk.NSEW)
middle_frame.grid(row=1, column=0, sticky=tk.NSEW)
bottom_frame.grid(row=2, column=0, sticky=tk.NSEW)

username_label = tk.Label(top_frame, text="Enter username:", font=FONT, bg=DARK_GREY, fg=WHITE)
username_label.pack(side=tk.LEFT, padx=10)

username_textbox = tk.Entry(top_frame, font=FONT, bg=MEDIUM_GREY, fg=WHITE, width=23)
username_textbox.pack(side=tk.LEFT)

username_button = tk.Button(
    top_frame, text="Join", font=BUTTON_FONT, bg=OCEAN_BLUE, fg=WHITE, command=connect_to_server
)
username_button.pack(side=tk.LEFT, padx=15)

message_textbox = tk.Entry(bottom_frame, font=FONT, bg=MEDIUM_GREY, fg=WHITE, width=38)
message_textbox.pack(side=tk.LEFT, padx=10)

message_button = tk.Button(
    bottom_frame, text="Send", font=BUTTON_FONT, bg=OCEAN_BLUE, fg=WHITE, command=send_message
)
message_button.pack(side=tk.LEFT, padx=10)

message_box = scrolledtext.ScrolledText(
    middle_frame, font=SMALL_FONT, bg=MEDIUM_GREY, fg=WHITE, width=67, height=26.5
)
message_box.config(state=tk.DISABLED)
message_box.pack(side=tk.TOP)

main_window.mainloop()
```

- **Main Window:** Initializes the main application window with specified dimensions and title.
- **Protocol:** Sets the closing protocol to call **`on_closing()`** when the window is closed.