Client Chat Application Documentation

Roman Sereda

Introduction

- Purpose

The Client Chat Application is a simple WPF (Windows Presentation Foundation) application that enables users to connect to a chat service, send and receive messages, and disconnect from the service. It demonstrates the use of Windows Communication Foundation (WCF) to implement a client-server chat interaction. One of its possible approaches is to be a chat in some online game.

- Prerequisites

To run the Client Chat Application, you need:

- Windows operating system.
- .NET Framework installed.
- IDE (for development and building).

Application Overview

- Description

The Client Chat Application is a user-friendly chat interface that allows users to connect to a chat service and engage in real-time text conversations with other connected users. Users can send and receive messages, and the chat history is displayed in the application window.

- Features

- Connect to the chat service using a chosen username.
- Send messages to other users.
- Receive messages from other users.
- Disconnect from the chat service.
- Clear the chat display.

- Technologies Used

- Windows Presentation Foundation (WPF) for the user interface.
- Windows Communication Foundation (WCF) for client-server communication.
- C# programming language for application logic.

Developers guide

File: ServerUser.cs

Class: **ServerUser**

- **Purpose**: This class is used to represent users connected to the chat service. It stores information about the user, including their name, OperationContext and ID associated with them.
- **Properties**:
    - **Name** (string): Gets or sets the user's name (display name) in the chat.
    - **ID** (int): Gets or sets a unique identifier for the user.

- **operationContext** (OperationContext): Gets or sets the OperationContext associated with the user, which represents the user's context during the session.

File: IService1.cs

Interface: **IService1**

- **Purpose**: This interface defines the service contract for the chat service. It specifies the operations that the service provides to clients.
- **Operations**:
    1. **int Connect(string name)**: Allows a client to connect to the chat service with a username. Returns a unique identifier (ID) for the client.
    2. **void Disconnect(int id)**: Allows a client to disconnect from the chat service using their ID.
    3. **void SendMsg(string msg, int id)**: Allows clients to send messages to the chat service. Messages are broadcasted to all connected clients.

Interface: **IServiceCallback**

- **Purpose**: This interface defines the callback contract for the chat service. It specifies the operation that the server uses to send messages back to clients.
- **Operations**:

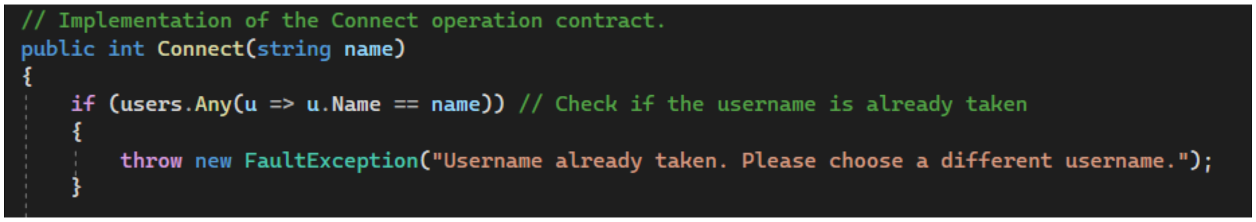
1. **void MsgCallback(string msg)**: Allows the server to send messages to clients. This operation is one-way, meaning it doesn't require a response from clients.

File: Service1.cs

Class: **Service1**

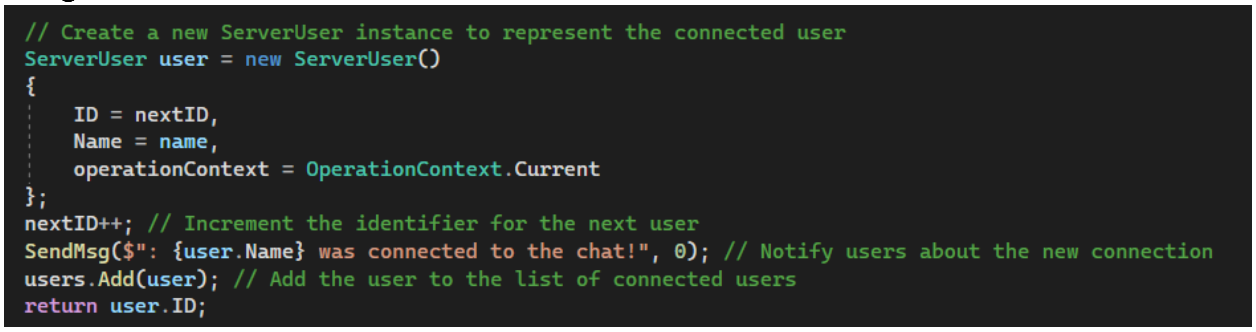
- **Purpose**: This class is the implementation of the chat service. It provides the actual logic for handling client connections, disconnections, and message broadcasting.
- **Attributes**:
    - **[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]**: Specifies that a single instance of the service (**Service1**) should handle all client requests. It's a single mode, ensuring that all clients share the same service instance.
- **Fields**:
    - **private List<ServerUser> users**: Stores information about connected users.
    - **private int nextID = 1**: Keeps track of the next available user ID.
- **Methods**:

1. **int Connect(string name)**: Allows a client to connect to the chat service with a username. Checks if the username is available,



```
// Implementation of the Connect operation contract.
public int Connect(string name)
{
    if (users.Any(u => u.Name == name)) // Check if the username is already taken
    {
        throw new FaultException("Username already taken. Please choose a different username.");
    }
```

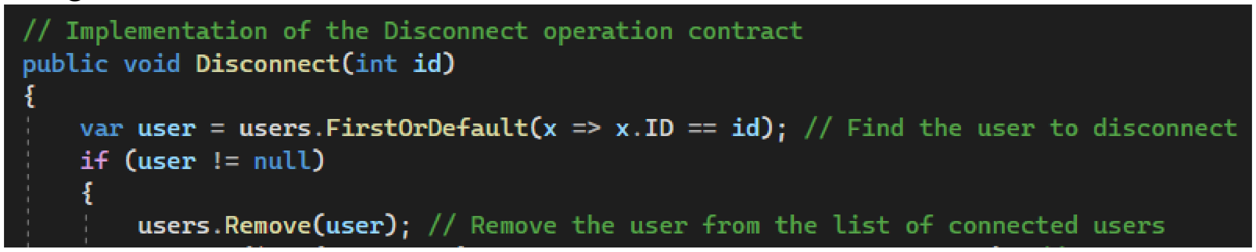
assigns a unique ID, notifies other users of the new connection, and returns the assigned user ID



```
// Create a new ServerUser instance to represent the connected user
ServerUser user = new ServerUser()
{
    ID = nextID,
    Name = name,
    operationContext = OperationContext.Current
};
nextID++; // Increment the identifier for the next user
SendMsg($": {user.Name} was connected to the chat!", 0); // Notify users about the new connection
users.Add(user); // Add the user to the list of connected users
return user.ID;
```

.

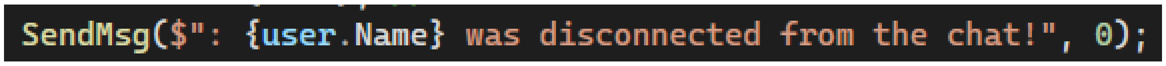
2. **void Disconnect(int id)**: Allows a client to disconnect from the chat service using their ID



```
// Implementation of the Disconnect operation contract
public void Disconnect(int id)
{
    var user = users.FirstOrDefault(x => x.ID == id); // Find the user to disconnect
    if (user != null)
    {
        users.Remove(user); // Remove the user from the list of connected users
```

.

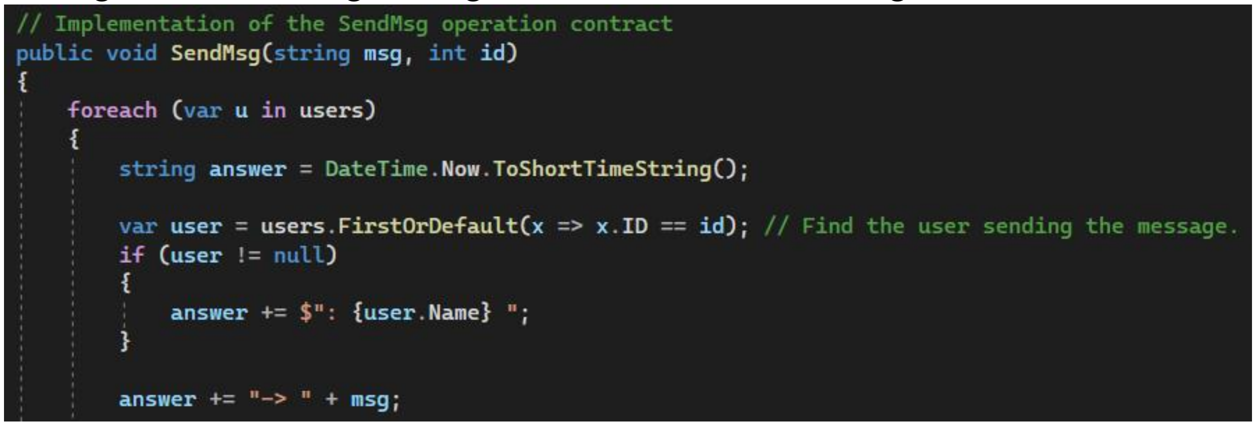
Removes the user from the list of connected users and notifies other users of the disconnection



```
SendMsg($": {user.Name} was disconnected from the chat!", 0);
```

.

3. **void SendMsg(string msg, int id)**: Allows clients to send messages to the chat service. Constructs messages with sender information and broadcasts them to all connected clients. Iterates through all users, creating correct time number, finding the user sending message with his ID and formatting it



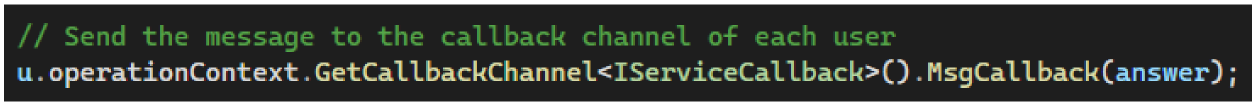
```
// Implementation of the SendMsg operation contract
public void SendMsg(string msg, int id)
{
    foreach (var u in users)
    {
        string answer = DateTime.Now.ToShortTimeString();

        var user = users.FirstOrDefault(x => x.ID == id); // Find the user sending the message.
        if (user != null)
        {
            answer += $": {user.Name} ";
        }

        answer += "-> " + msg;
```

.

Then it sends message to the callback channel of all users:



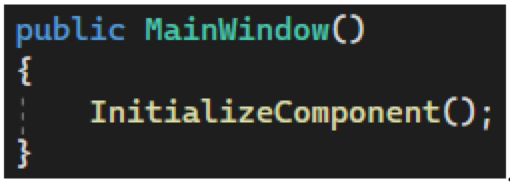
```
// Send the message to the callback channel of each user
u.operationContext.GetCallbackChannel<IServiceCallback>().MsgCallback(answer);
```

File: MainWindow.xaml.cs

Class: **MainWindow**

- **Purpose**: This class represents the main window of the chat client application. It handles user interface events and interacts with the chat service.
- **Fields**:
  - **private bool isConnected = false**: Keeps track of whether the user is connected to the chat service.
  - **private ServiceChat.Service1Client client**: An instance of the WCF service client used for communication.
  - **private int ID**: Stores the user's ID.
- **Methods**:

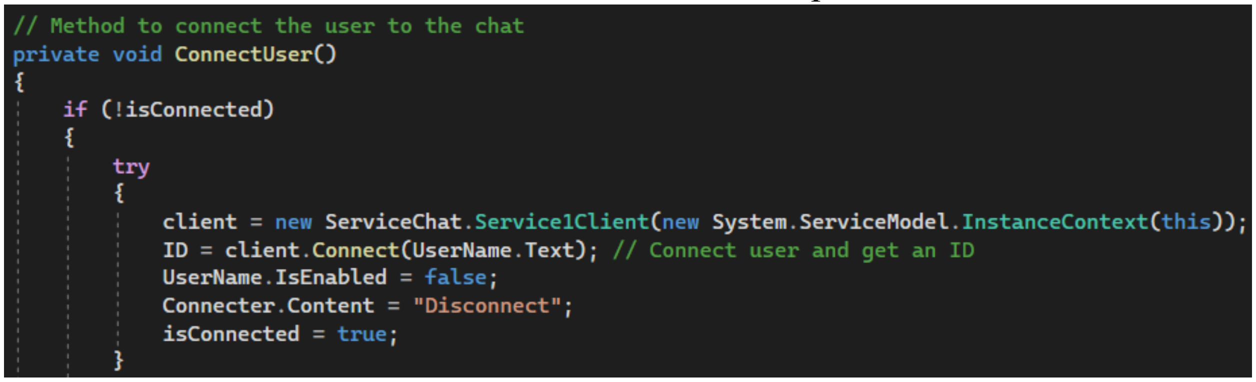
1. **MainWindow()**: Constructor for the main window. Initializes the main window



```
public MainWindow()
{
    InitializeComponent();
}
```

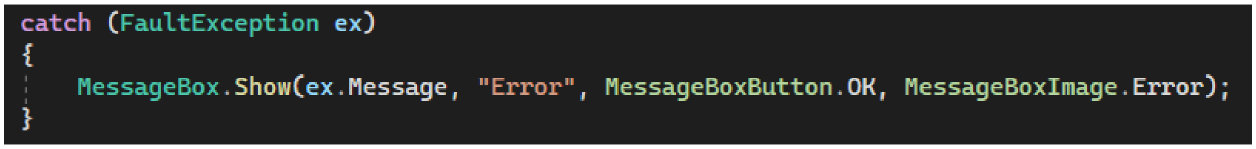
.

2. **ConnectUser()**: Method to connect the user to the chat service. It creates a new instance of the service client, connects the user, updates the UI

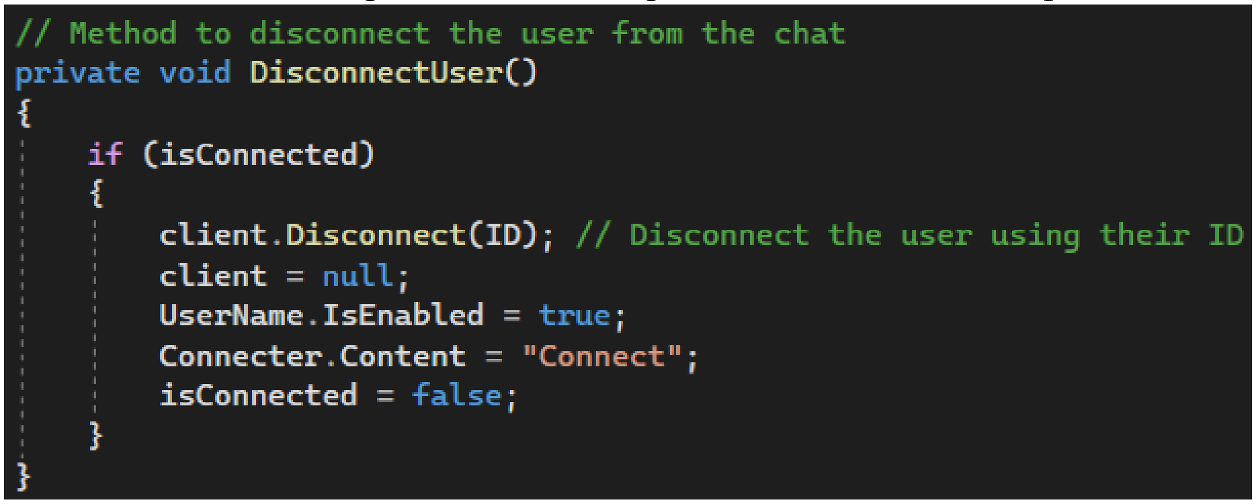


```
// Method to connect the user to the chat
private void ConnectUser()
{
    if (!isConnected)
    {
        try
        {
            client = new ServiceChat.Service1Client(new System.ServiceModel.InstanceContext(this));
            ID = client.Connect(UserName.Text); // Connect user and get an ID
            UserName.IsEnabled = false;
            Connecter.Content = "Disconnect";
            isConnected = true;
        }
```

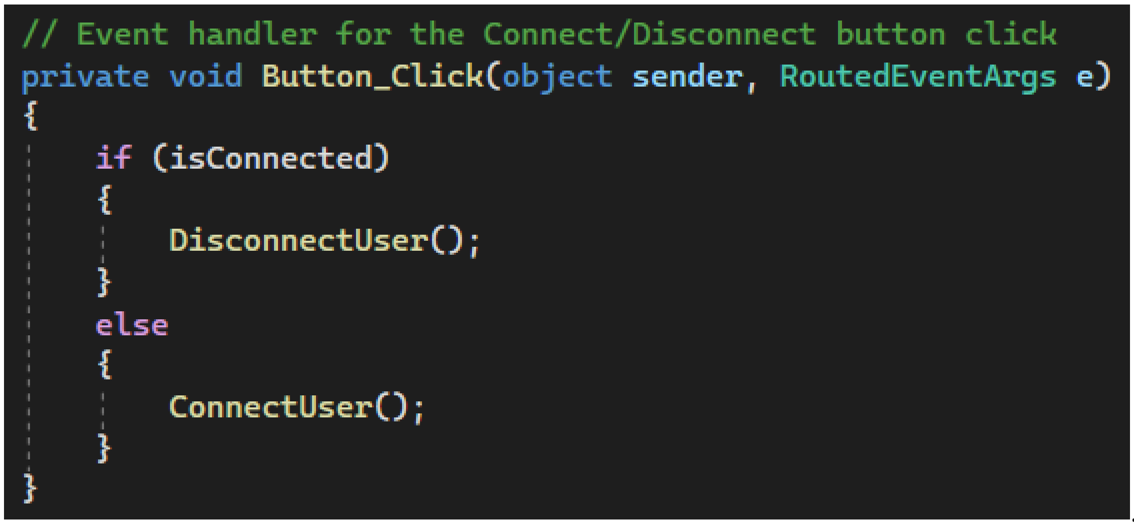
, and handles exceptions if the username is already taken


```
catch (FaultException ex)
{
    MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);
}
```
.

3. **DisconnectUser()**: Method to disconnect the user from the chat service. It disconnect the user using their ID, cleans up the client instance, and updates the UI

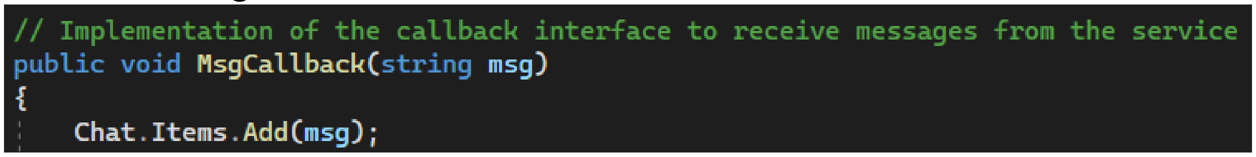

```
// Method to disconnect the user from the chat
private void DisconnectUser()
{
    if (isConnected)
    {
        client.Disconnect(ID); // Disconnect the user using their ID
        client = null;
        UserName.IsEnabled = true;
        Connecter.Content = "Connect";
        isConnected = false;
    }
}
```
.

4. **Button_Click(object sender, RoutedEventArgs e)**: Event handler for the "Connect/Disconnect" button click. It calls either **ConnectUser** or **DisconnectUser** based on the user's current connection status


```
// Event handler for the Connect/Disconnect button click
private void Button_Click(object sender, RoutedEventArgs e)
{
    if (isConnected)
    {
        DisconnectUser();
    }
    else
    {
        ConnectUser();
    }
}
```
.

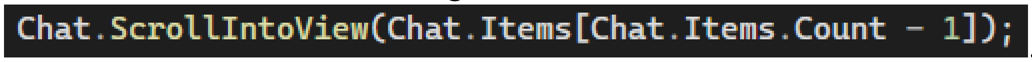
5. **ClearChatButton_Click(object sender, RoutedEventArgs e)**: Event handler for the "Clear Chat" button click - it clears the chat content.

6. **MsgCallback(string msg)**: Implementation of the callback interface (**ServiceChat.IService1Callback**) to receive messages from the service. It adds received messages to the chat window



```
// Implementation of the callback interface to receive messages from the service
public void MsgCallback(string msg)
{
    Chat.Items.Add(msg);
```

and scrolls to the latest message

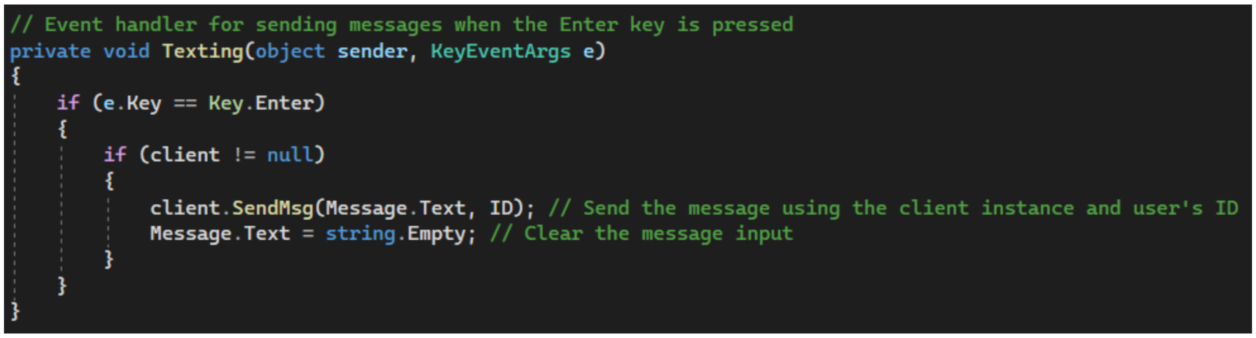


```
Chat.ScrollIntoView(Chat.Items[Chat.Items.Count - 1]);
```

.

7. **Window_Closed(object sender, System.ComponentModel.CancelEventArgs e)**: Event handler for the window closed event to ensure disconnection when the application is closed.

8. **Texting(object sender, KeyEventArgs e)**: Event handler for sending messages when the Enter key is pressed. It sends the message to the service using the client instance and user's ID



```
// Event handler for sending messages when the Enter key is pressed
private void Texting(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter)
    {
        if (client != null)
        {
            client.SendMsg(Message.Text, ID); // Send the message using the client instance and user's ID
            Message.Text = string.Empty; // Clear the message input
        }
    }
}
```

.

File: Program.cs

Class: **Program**

- **Purpose**: This class contains the entry point (**Main** method) for the chat host program. It hosts the WCF service.
- **Methods**:
    1. **Main(string[] args)**: The entry point of the program. It creates a new instance of **ServiceHost** to host the WCF service (**wcf_chat.Service1**), opens the host, and waits for user input before closing the host.

User Interface

- Main Window

The main application window displays the chat user interface.

- Connect/Disconnect

The "Connect" button allows users to connect to the chat service using a chosen username. Once connected, the button changes to "Disconnect" to allow users to disconnect from the chat service.

- Chat Display

The chat display area shows the conversation history between users. Messages are displayed with timestamps.

- Message Input

The text box at the bottom of the window allows users to input messages to send to other users.

- Clear Chat Button

The "Clear Chat" button clears the chat display area, removing all messages.

Functionality

- Connecting to the Chat Service

- Click the "Connect" button.
- Enter a desired username.
- If the username is available, you'll be connected and can start chatting.

- Disconnecting from the Chat Service

- Click the "Disconnect" button.
- You'll be disconnected from the chat service.

- Sending and Receiving Messages

- Type a message in the input text box.
- Press the "Enter" key to send the message.
- You'll receive messages from other connected users in the chat display area.

- Clearing the Chat Display

- Click the "Clear Chat" button.
- The chat display area will be cleared of all messages.

Usage Instructions

- Running the Application

1. Ensure you have the required prerequisites installed.
2. Run the ChatHost as an administrator from your computer.
3. Run the .xaml program to open an interactive app.
4. Run the .xaml program one more time to simulate another user.
5. Use your chat.