WitNetwork Quick Start Guide

Welcome to WitNetwork! This guide will help you set up and use the framework in your Unity project.

Summary

WitNetwork is a simple and flexible networking framework for Unity. It allows you to quickly set up server-client communication, send messages between peers, and manage network settings with minimal effort.

Description

WitNetwork is designed to make multiplayer and networked applications easy to build in Unity. By adding a single prefab to your scene and configuring a few settings, you can start hosting or joining network sessions. The framework supports both local and remote connections, automatic server discovery in local mode, and instant message-based communication between all connected peers using a simple command system.

Technical Details

- **Prefab-Based Initialization:**
  Add the `WitNetwork` prefab to your scene to automatically start the network on play.
- **Configurable Settings:**
  All network parameters (mode, server IP/port, group ID, ping intervals, etc.) are managed via a ScriptableObject.
  Access these settings in the Unity Editor under `Tools > WitNetwork > Settings`.
- **Modes:**
  - **Server:** Hosts the network session.
  - **Client:** Connects to a server.
  - **Local Mode:** Automatically discovers and connects to a local server without manual IP entry.
  - **Hosted Mode:** Connects to a specified server IP.
- **Communication:**
  Use `CommunicationManagerSO` to register command handlers and send messages.

    - Register a handler for a command name to receive messages.
    - Use `SendMessage` to broadcast data to all connected peers.

- **Threaded Client:**
  The client runs its update loop in a background thread, ensuring reliable communication even when the application is not focused.
- **Extensible:**
  Easily extend the framework by registering new commands and handling custom data types.

WitNetwork is ideal for rapid prototyping, multiplayer games, and any Unity project that needs quick and reliable networking.

---

1. Add the Network Prefab

- Drag the **WitNetwork** prefab into your scene.
- This prefab will automatically start the network when you run your game.

---

2. Configure Network Settings

- Go to **Tools > WitNetwork > Settings** in the Unity menu.
- Here you can configure all network parameters:

Parameter	Description
Network Mode	Choose `Server` to host or `Client` to connect.
Server Mode	Set the `Local` or `Hosted`
Server Port	The port used for network communication (default: 9092). Must match on both server & client.
Server IP	The IP address of the server. Use `127.0.0.1` for local, or your server's IP for remote.
Group ID	(Optional) Used for grouping clients.
Ping Interval (s)	How often to send a ping to check connection (in seconds).
Ping Timeout Intervals	Number of missed pings before timing out.

Tips: - For local testing, server will be connected seamlessly. without needing to specify an IP. - If hosting a server, enter the server's actual IP address. and make sure the port is open in your firewall. - For remote connections, ensure the server's IP is accessible from the client. - The port must be the same on both server and client.

---

3. Instant Communication with CommunicationManagerSO

You can use `CommunicationManagerSO` to easily send and receive messages (commands) between all connected peers.

How it works

- **Register a command handler**: Any class can register a handler for a command name. When a message with that command name is received, your handler will be called.
- **Send a message**: Use `SendMessage` to send a command and data to all connected peers. Anyone who registered a handler for that command will receive it.

Example: Registering and Handling a Command

```
// Register a command handler (e.g., in Awake or Start)
CommunicationManagerSO.Instance.RegisterCommand("MyCommand", (payload) => {
    Debug.Log("Received MyCommand with payload: " + payload);
});
```

Once registered, your handler will be called automatically whenever another peer sends a `"MyCommand"` message.

Example: Sending a Command

```
// Example message class
public class ChatMessage
{
    public string Username { get; set; }
    public string Message { get; set; }

    public ChatMessage(string username, string message)
    {
        Username = username;
        Message = message;
    }
}

// Send a message to all connected peers
CommunicationManagerSO.Instance.SendMessage("MyCommand", new ChatMessage("user-name", "hello
```

- All peers who registered for `"MyCommand"` will receive this message and their handler will be called.

That’s it!
Enjoy your quick networking solution with WitNetwork.