ClientServerNetwork Documentation

Release latest

Sep 29, 2020

Contents

The goal of this section is to be short, easy to follow, and closer to a tutorial than reference documentation. If someone follows the instructions on this page, they should be up and running with an example client and server, with a bit of guidance on where to go next to start adding their own game logic.

To start, download the latest version of the Client and Server projects.

CHAPTER 1

Running the Server

1. Open the Server project and load the scene called “Server”
2. Hit play.

By default, the server will run on port 603 (the area code for New Hampshire). The example server logic will automatically approve all connection requests.

CHAPTER 2

Running the Client

1. Open the Client project and load the scene called “Client”
2. Hit play.

2.1 Connecting to the server

The client will allow you to enter the IP address and port of the server. If you are running the server locally with default settings, you can enter “127.0.0.1” for the IP address and “603” for the port.

The connection process starts by sending a request to the server to connect. The default server logic will automaitcally approve all connection requests, and your client will be approved. Once your client is notified that it has properly connected to the server, it will add itself to area 1 (see the section on area filtering for more information), spawn its own “Player” object and add the player object to area 1.

To test with more client connections, you should build the client project and run it outside of the Unity editor.

TODO: This should be a quickstart guide to use the networking library. A short tutorial on how to get an example client and server project, as well as set it up so they can connect to each other, as well as simple examples of RPCs and Networked Objects should be included.

TODO: What is the process of creating a custom server? What considerations are needed when developing a server?

TODO: What is the process of creating a custom client using this library? What considerations are needed when developing a client?

CHAPTER 3

Project requirements

This library was updated to support Unity version 2018.2.2f1, but may work with newer versions.

CHAPTER 4

Connection and Disconnection

4.1 Connecting to a server

TODO: Discuss the process of connecting to a server, including the specific messages between the client and server. Discuss requirements when connecting via IP, as well as opening a port on the server. How are other clients notified, if at all, that a client has connected to the server? Discuss the process of a server confirming or denying a connection, and an example on how a server could check with an external resource to determine if a client should connect (such as checking with a database, steam servers, etc).

4.2 Unique Client Ids

TODO: Discuss how each client is assigned a unique id. How would the game logic on the server get this id, and how would it use this id?

4.3 Disconnecting from a server

TODO: Discuss the process of disconnectng from a server, including the specific messages between the client and a server. How are other clients notified, if at all, that a client has disconnected from the server? and the server kicking a client.

4.3.1 From the client

TODO: Discuss the different ways a client can disconnect: Connection timing out (in case of client crashing/closing abruptly) or the client requesting a disconnection

4.3.2 Kicking a client

TODO: Discuss how the server can kick a client

CHAPTER 5

Networked Objects

Networked objects (as well as RPCs), make up the bulk of what the networking library supports. Most full games will be built using a combination of networked objects, as well as remote procedure calls. Knowing when to use one over the other is entirely up to the developers of the game, but some best practices can guide those decisions.

In general, a networked object is a game object which is instantiated on every client connected to the server. As the object moves on one client, information about it's transform is automatically sent through the server to the other connected clients. A simple example of this may be a game object representing a player's main character. When the game object representing the character is instantiated, it will automatically be created on all other clients connected to the server. As the player moves their character in the simulation, the game object representing that character on all of the other clients will move as well, keeping all of their positions and rotations in sync.

5.1 Instantiating

Instantiating a network object is similar to instantiating any other object in Unity, with a few minor differences. First, use the Instantiate method found in ClientNetwork, instead of Unity's built-in Instantiate method. The first parameter of this method is the string name of a prefab you would like to instantiate. This string name will be sent to the server, as well as any other clients which need to instantiate the object. For Unity to properly instantiate the object, a prefab with that name must be in a Resources directory.

Note: Loading objects from within a Resources directory based on their names is no longer recommended by Unity (but is still fully functional). It will be worth exploring updating this logic to instead use one of Unity's newer systems for dynamic asset loading, like Addressables or Asset Bundles.

The server can spawn objects in a similar manner, however, please see the documentation on ownership to understand how server-instantiated objects behave.

Networked objects need to have the NetworkSync component. This component assists the networking system by tracking a unique id for the object, handling remote procedure calls, as well as handling object synchronization messages. If you do not feel that your object needs these features, then it may be better to call an RPC to AllClients and have them all instantiate local objects.

All networked objects have unique ids associated with them, which assists the networking system in sending messages about specific objects to all clients. One feature of UCNetwork is that local object instantiation happens synchronously,

meaning it will immediately instantiate an object on your local client when you call ClientNetwork.Instantiate. In order to accomplish this, while still having unique object ids across the entire network, every client is given a pool of unique ids (default is 500). When a client instantiates an object, it picks a free id in the id pool it has been given and tells the server which id it has chosen for the new network object. The server tracks the size of each client's pool of ids and will send additional ids to a client when it starts running low.

5.2 Destroying

When a networked game object is destroyed, it need to inform the server (which will inform the other clients) that the object needs to be removed from the simulation. As objects are destroyed, the NetworkSync calls ClientNetwork.Destroy, which will send a message to the server that the object should be removed from all clients.

Note: What happens when a client who does not own an object tells the server that it has been destroyed?

5.3 Network Sync Component Details

TODO: What does a network sync component provide? Why is it required? What callbacks does the network sync component provide to the object it is attached to, if any, and what additional functionality does it provide? How can you add additional data to the Network Synchronization Messages? What is a LiteSync message?

5.4 Remote Procedure Calls

TODO: Discuss sending remote procedure calls to a single game object. Discuss how sending a RPC to a single game object works, and why you would use it. More detail regarding RPCs will be provided in the RPC documentation.

5.5 Smooth Movement

TODO: Discuss network interpolation and extrapolation. Why is this needed? How does it account for lag? What considerations would need to be make regarding objects with physics, or fast paced action games?

CHAPTER 6

Ownership

All networked game objects are "owned" by an individual client.

6.1 Default Ownership

TODO: Discuss how ownership is assigned to networked game objects as they are spawned, both from the client and server.

6.2 Ownership Transferring

TODO: Discuss how and when ownership of game objects is transferred between various clients. Discuss why we would want to transfer ownership (as clients are leaving areas, disconnecting, or based on gameplay).

CHAPTER 7

Remote Procedure Calls (RPCs)

7.1 Overview

Remote Procedure Calls are a simple way to fit networking interactions into a typical game's code structure. They can be conceptualized to work the same way as normal function calls, but initiated by a remote machine. Once communicated across the network, RPCs use reflection to find and invoke the matching method on the receiving object (indicated by the object's network ID).

7.2 Format of an RPC

TODO: Discuss the general way to call RPCs, including what parameters the CallRPC function allows. Discuss how to define RPC functions.

7.3 RPCs from the client

TODO: Discuss calling RPCs from the client to the server. Examples on why you would do this.

7.4 RPCs from the server

TODO: Calling RPCs from the server to the clients (or single client). Examples on why you would do this.

7.5 Determining where an RPC goes - MessageReceiver

TODO: Discuss the different options for the MessageReceiver parameter for the CallRPC function. What does each of them mean, and why would you use each? (ServerOnly = 1, AllClients = 2, OtherClients = 4, AllClientsInArea = 8, OtherClientsInArea = 16, SingleClient = 32)

7.6 Supported Data Types

TODO: Discuss the data types that the networking library allows you to send. Discuss how to add additional data types (WriteRPCParams and ReadRPCParams).

7.7 RPCs from one client to another

TODO: Discuss how and why clients would call RPCs that arrive to a single other client. Discuss why this isn't natually supported by the networking library.

CHAPTER 8

Area Filtering

The concept of "area filtering" allows the networking library to send data to specific groups of clients. For example, if client A, B, and C are in a dungeon while client D is back in town, you probably don't need to send the data relating to what is going on in the dungeon to client D.

8.1 Examples

8.1.1 Rogue

TODO: Describe why this system would be used for a character who could hide themselves from other players

8.1.2 Team Chat

TODO: Describe how this system could be used for handling something like team chat, where a specific group of players should be able to communicate with each other

8.1.3 Virtual Areas

TODO: Describe how and why this system could be used for supporting different locations in a larger game, such are multi-floor dungeons, and clients who have completely different scenes loaded

8.2 Using Area Filters

TODO: Describe how to use area filtering, including adding clients and networked game objects to areas, removing them from areas

8.2.1 Synchronizing Game State

TODO: With clients in different areas, describe how clients are told the current state of an area when adding themselves to an area which already has networked game objects, etc.

CHAPTER 9

Voice Chat

9.1 Getting Started

TODO: How to get started with online voice chat

9.2 Voice Chat: A Primer

TODO: Describe how voice chat communication over the internet works in a general sense, which includes recording microphone data, compression, serialization, and playing the audio on a recieving client at the proper location.

9.3 Using Voice Chat

TODO: Details on this voice chat integration, which was initially built by FHolm. The original voice chat library can be found here: FHolm Old Unity Assets. Describe any changes and modifications from the original version

9.4 ThreadedJob - Multiple Threads in Unity

TODO: How does the ThreadedJob functionality work? Examples on how it could be used elsewhere

CHAPTER 10

Additional Features

TODO: Describe any additional features the library supports? What features are these, how do you use them, why would you use them?

CHAPTER 11

Client Development

TODO: Describe best practices for developing a client. This section may include any topics relating to client development that don't fit in with the rest of the documentation, including pro-tips, guidance, or other recommendations.

CHAPTER 12

Server Development

TODO: Describe best practices for developing a server. This section may include any topics relating to server development that don't fit in with the rest of the documentation, including pro-tips, guidance, or other recommendations.

CHAPTER 13

Example Projects

13.1 Tactics Server

Example of the gameplay

How the server is developed

Post the challenge of developing the client

13.2 Tic-Tac-Toe

13.2.1 Overview

This is an overview of Tic-Tac-Toe, and how this example has been developed. Example RPCs and diagram on players. Support for spectators and different players reokacing

13.2.2 Server

13.2.3 Client

CHAPTER 14

License

14.1 MIT License

14.2 Copyright (c) 2018 David Carrigg

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 15

UCNetwork Network

15.1 Overview

This is an authoritative client-server networking library for Unity3D, built using Lidgren.

15.2 Features

- Online multiplayer!
- Real-time messaging and object synchronization for Unity3D
- Authoritative, dedicated, and standalone server support
- Support for clients with different codebases, and asymmetrical experiences
- Message filtering based on logical "Areas"
- Open world games, or simulations where clients are in different scenes
- Networked object ownership transfer
- **Easily hook up your own functionality for:**
  - Storing game state when clients disconnect
  - Authorizing player connections based on external systems
  - Server-side gameplay logic
- . . . and more!

15.3 Additional Help and Contact

This library was developed by David Carrigg, with contributions from the rest of the development team that worked on the now cancelled game, Upsilon Circuit. You can find various ways to contact me on my website.

15.4 Can I use this library for my project?

Probably! This software is licensed with the *MIT License*.

Have fun!