Unity SDK

For more information, please visit our documentation site

Setup

Here we'll be going over the steps to get your Unity client up and running and connected to a Colyseus server.

Topics covered include:

- Running the server locally
- Server settings
- Connecting to a server
- Connecting to a room
- Communicating with a room, and the room's state.

The topics should be enough for you to set up a basic client on your own, however, you are welcome to use and modify the included example code to suit your needs.

Running the server locally

To run the demonstration server locally, run the following commands in your terminal:

```
 cd Server
npm install
npm start
```

The built-in demonstration comes with a single room handler, containing a suggested way of handling entities and players. Feel free to change all of it to fit your needs!

Creating a Colyseus Settings Object:

- Right-click anywhere in the Project folder, select "Create", select "Colyseus", and click "Generate ColyseusSettings Scriptable Object"
- Fill in the fields as necessary.
    - **Server Address**
    - The address to your Colyseus server.
    - **Server Port**
    - The port to your Colyseus server.
    - **Use secure protocol**
    - Check this if requests and messages to your server should use the "https" and "wss" protocols.
    - **Default headers**
    - You can add an unlimited number of default headers for non web socket requests to your server.
    - The default headers are used by the `ColyseusRequest` class.
    - An example header could have a `"Name"` of `"Content-Type"` and a `"Value"` of `"application/json"`

Colyseus Manager:

- You will need to create your own Manager script that inherits from `ColyseusManager` or use and modify the provided `ExampleManager`.

```
public class ExampleManager : ColyseusManager<ExampleManager>
```

- Make an in-scene manager object to host your custom Manager script.
- Provide your Manager with a reference to your Colyseus Settings object in the scene inspector.

Client:

- Call the `InitializeClient()` method of your Manager to create a `ColyseusClient` object which is stored in the `client` variable of `ColyseusManager`. This will be used to create/join rooms and form a connection with the server.

```
ExampleManager.Instance.InitializeClient();
```

- If your Manager has additional classes that need reference to your `Client`, you can override `InitializeClient` and make those connections in there.

```
//In ExampleManager.cs
public override void InitializeClient()
{
    base.InitializeClient();
    //Pass the newly created Client reference to our RoomController
    _roomController.SetClient(client);
}
```

- You can get available rooms on the server by calling `GetAvailableRooms` of `ColyseusClient` :

```
return await GetAvailableRooms<ColyseusRoomAvailable>(roomName, headers);
```

Connecting to a Room:

- There are several ways to create and/or join a room.
- You can create a room by calling the `Create` method of `ColyseusClient` which will automatically create an instance of the room on the server and join it:

```
ExampleRoomState room = await client.Create<ExampleRoomState>(roomName);
```

- You can join a specific room by calling `JoinById` :

```
ExampleRoomState room = await client.JoinById<ExampleRoomState>(roomId);
```

- You can call the `JoinOrCreate` method of `ColyseusClient` which will matchmake into an available room, if able to, or will create a new instance of the room and then join it on the server:

```
ExampleRoomState room = await client.JoinOrCreate<ExampleRoomState>(roomName);
```

Room Options:

- When creating a new room you have the ability to pass in a dictionary of room options, such as a minimum number of players required to start a game or the name of the custom logic file to run on your server.
- Options are of type `object` and are keyed by the type `string` :

```
Dictionary<string, object> roomOptions = new Dictionary<string, object>
{
    ["YOUR_ROOM_OPTION_1"] = "option 1",
    ["YOUR_ROOM_OPTION_2"] = "option 2"
};

ExampleRoomState room = await ExampleManager.Instance.JoinOrCreate<ExampleRoomState>(roomName, roomOptions);
```

Room Events:

`ColyseusRoom` has various events that you will want to subscribe to:

OnJoin

- Gets called after the client has successfully connected to the room.

OnLeave

- Gets called after the client has been disconnected from the room.
- Has a `WebSocketCloseCode` parameter with the reason for the disconnection.

```
room.OnLeave += OnLeaveRoom;
```

OnStateChange

- Any time the room's state changes, including the initial state, this event will get fired.

```
room.OnStateChange += OnStateChangeHandler;
private static void OnStateChangeHandler(ExampleRoomState state, bool isFirstState)
{
    // Do something with the state
}
```

OnError

- When a room related error occurs on the server it will be reported with this event.
- Has parameters for an error code and an error message.

Room Messages:

You have the ability to listen for or to send custom messages from/to a room instance on the server.

OnMessage

- To add a listener you call `OnMessage` passing in the type and the action to be taken when that message is received by the client.
- Messages are useful for events that occur in the room on the server. (Take a look at our tech demos for use case examples of using `OnMessage`)

```
room.OnMessage<ExampleNetworkedUser>("onUserJoin", currentNetworkedUser =>
{
    _currentNetworkedUser = currentNetworkedUser;
});
```

Send

- To send a custom message to the room on the server use the `Send` method of `ColyseusRoom`
- Specify the `type` and an optional `message` parameters to send to your room.

```
room.Send("createEntity", new EntityCreationMessage() { creationId = creationId, attributes = attributes });
```

Room State:

> See how to generate your `RoomState` from State Handling

- Each room holds its own state. The mutations of the state are synchronized automatically to all connected clients.
- In regards to room state synchronization:
    - When the user successfully joins the room, they receive the full state from the server.
    - At every `patchRate`, binary patches of the state are sent to every client (default is 50ms)
    - `onStateChange` is called on the client-side after every patch received from the server.
    - Each serialization method has its own particular way to handle incoming state patches.
- `ColyseusRoomState` is the base room state you will want your room state to inherit from.
- Take a look at our tech demos for implementation examples of synchronizable data in a room's state such as networked entities, networked users, or room attributes. (Shooting Gallery Tech Demo)

```
public class ExampleRoomState : Schema
{
    [Type(0, "map", typeof(MapSchema<ExampleNetworkedEntity>))]
    public MapSchema<ExampleNetworkedEntity> networkedEntities = new MapSchema<ExampleNetworkedEntity>();

    [Type(1, "map", typeof(MapSchema<ExampleNetworkedUser>))]
    public MapSchema<ExampleNetworkedUser> networkedUsers = new MapSchema<ExampleNetworkedUser>();

    [Type(2, "map", typeof(MapSchema<string>), "string")]
    public MapSchema<string> attributes = new MapSchema<string>();
}
```

Debugging

If you set a breakpoint in your application while the WebSocket connection is open, the connection will be closed automatically after 3 seconds due to inactivity. To prevent the WebSocket connection from dropping, use `pingInterval: 0` in your server code during development:

```
 import { Server, RedisPresence } from "colyseus";

const gameServer = new Server({
  // ...
  pingInterval: 0 // HERE
});
```

Make sure to have a `pingInterval` higher than `0` on production. The default `pingInterval` value is `3000`.