Unreal Game Lobby Documentation

Table of Contents

1. Introduction

This document serves as documentation for the Unreal demo of using Kjapp_gl, the game lobby system for NTNU Gjøvik, developed by the kjapp group.The demo is intended to either be used as a base for creating your in-game lobby system using the school's server with our lobby system on it, and/or as an example of how to execute and handle http requests within the Unreal Engine using their libraries.

2. Creating your project

There are a few things you need to do in order to use http calls and json with your project.

- Your project needs to be a c++ project, as blueprints do not support the unreal engine's http and json functionality (As of Unreal version 4.17.2). If this is implemented at a later date, pure blueprint projects could be used, but the example code is written in c++.
- You need to change the engine initialization file to include HTTP settings. These settings need to be added to The Engine.ini file, which can be found at: "ProjectName"/Saved/Config/Windows/Engine.ini. Unless you have any specific needs, I suggest these settings:

[HTTP]
HttpTimeout=300
HttpConnectionTimeout=-1
HttpReceiveTimeout=-1
HttpSendTimeout=-1
HttpMaxConnectionsPerServer=16
bEnableHttp=true
bUseNullHttp=false
HttpDelayTime=0

- In the "ProjectName".Build.cs file, be sure to add "Http", "Json" and "JsonUtilities" as public dependency modules. After doing that, your file might look similar to this:

```
2
3    using UnrealBuildTool;
4
5   public class GameLobby : ModuleRules
6   {
7       public GameLobby(ReadOnlyTargetRules Target) : base(Target)
8       {
9           PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;
10
11          PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine", "InputCore", "HeadMountedDisplay", "Http", "Json", "JsonUtilities" });
12      }
13  }
14
```

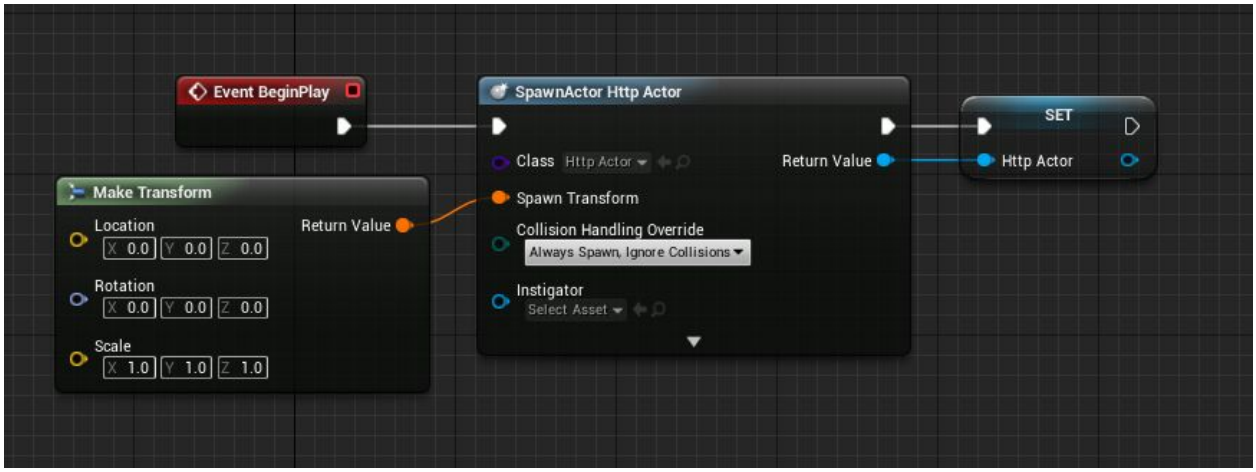
After following these steps, your project should be ready.

3. Using the example code

For using the example code your project needs to be set up correctly, as explained above. After that is done, all you need to do is import the HttpActor class into your project.

Before using the http actor, remember to get a game token and replace the GameToken variable with your own.

The HttpActor class is a child of unreal's actor class and is meant to be spawned into the game world. As it is very common to have a separate level for making lobbies for your games in the unreal engine, a good place to spawn this actor might be in the level blueprint of your lobby level. Below I have included an example of how you could do this in blueprints. Remember to save the http actor as a variable for later access, or for passing it to other places where it is needed without having unreal look for it in the game world, as this is a much more expensive thing to do than simply asking the level blueprint to pass it.



From here on you can simply call the pre made functions in the actor for doing the http calls. These will call other functions when they receive a response from the server (or the http call times out. As an example;

```
29    // Get All Matches http call
30    void AHttpActor::HttpGetAllMatches()
31    {
32        TSharedRef<IHttpRequest> Request = Http->CreateRequest();
33        Request->OnProcessRequestComplete().BindUObject(this, &AHttpActor::OnHttpGetResponseReceived);
```

In the get all matches http call the response will be handled in the function "OnHttpGetResponseReceived". This function is already setup to parse the return json object, it is up to you if you want to process the return data here, or dispatch it to somewhere else. Each of the response functions should have comment(s) clearly showing you where the processing or dispatching is intended to take place.

4. Http requests

1. Get All Matches

Function: HttpGetAllMatches()

Functionality:
Does a get all matches call to the server using the games GameToken.
This http call calls the function: OnHttpGetResponseReceived.

Return: void

2. Get All Non Full Matches

Function: HttpGetAllNonFullMatches()

Functionality:
Does a get all non full matches call to the server using the games GameToken.
This http call calls the function: OnHttpGetResponseReceived.

Return: void

3. Get All Matches In Session

Function: HttpGetAllMatchesInSession()

Functionality:
Does a get all matches in session call to the server using the games GameToken.
This http call calls the function: OnHttpGetResponseReceived.

Return: void

4. Get All Matches Not In Session

Function: HttpGetAllMatchesNotInSession()

Functionality:
Does a get all matches not in session call to the server using the games GameToken.
This http call calls the function: OnHttpGetResponseReceived.

Return: void

5. Get Match By Id

Function: HttpGetMatchById(FString MatchId)

Paramter:
Name: MatchId
Type: FString
Description: The Id of the match, this is created when the host creates a match.

Functionality:
Does a get match by id call to the server using the provided id.
This http call calls the function: OnHttpGetMatchResponseRecived.

Return: void

6. Create Match

Function: HttpCreateMatch(FNewMatchData MatchData)

Paramter:
Name: MatchData
Type: FNewMatchData
Description: A struct containing all the data needed to create a new match. Note: The whole struct needs to be filled out.

Functionality:
Does a create match call to the server using the provided data in the FNewMatchData struct.
This http call calls the function: OnHttpCreateMatchResponseRecived.

Return: void

7. Delete Match

Function: HttpDeleteMatch(FString MatchId)

Paramter:
Name: MatchId
Type: FString
Description: The Id of the match, this is created when the host creates a match.

Functionality:
Does a delete match call to the server using the provided id.
This http call calls the function: OnHttpDeleteMatchResponseRecived.

Return: void

8. Update Player Count

Function: HttpUpdatePlayerCount(FString MatchId, int NewPlayerCount)

Paramter:
Name: MatchId
Type: FString
Description: The Id of the match, this is created when the host creates a match.

Name: NewPlayerCount
Type: int
Description: The new number of players in the match/lobby after this update.
Allowed values: {1..inf}

Functionality:
Does a update player count call to the server using the provided id and new player count variables.
This http call calls the function: OnHttpUpdatePlayerCountResponseRecived.

Return: void

9. Update Match Status

Function: HttpUpdateMatchStatus(FString MatchId, int MatchStatus)

Paramter:
Name: MatchId
Type: FString
Description: The Id of the match, this is created when the host creates a match.

Name: MatchStatus
Type: int
Description: The new status of the match/lobby. 0 if waiting, 1 if in session.
Allowed values: {0..1}

Functionality:
Does a update match status call to the server using the provided id and new match status variables.
This http call calls the function: OnHttpUpdateMatchStatusResponseRecived.

Return: void

10. Send Match Report

Function: HttpSendMatchReport(FString MatchId, FString MatchReport)

Paramter:
Name: MatchId
Type: FString
Description: The Id of the match, this is created when the host creates a match.

Name: MatchReport
Type: FString
Description: The match report you want to submit, formated as a json object.

Functionality:
Does a send match report call to the server using the provided id and match report variables, which it packs into a json object together with the gameToken.
This http call calls the function: OnHttpSendMatchReportResponseRecived.

Return: void