This is only a small part of documentation. Full documentation is here:
https://github.com/alvyxaz/barebones-masterserver/wiki

Quick Setup

Test it first

⚠ By default, this demonstration is set up to work on **Windows** only. For other platforms, you'll need to edit the `Barebones/Demos/QuickSetup/Editor/MsfQuickBuild` script. You'll also need to change the "Executable Path" (on Mac, it's probably `./GameServer.app/Contents/MacOS/GameServer`).

1. Go to `Tools > Msf > Build All` and select a folder, to which you want to save the binaries (builds with .exe files).
2. Wait for the building process to complete
3. Go to the specified folder and open `MasterAndSpawner.exe`
    i. Click "Start Master Server" button
    ii. Wait until you see "Connected" at the top right corner
    iii. Click "Register Spawner" button
4. Now open the `Client.exe`
    i. Hit "Play as guest"
    ii. Click "Games List
    iii. Click "Create Game Server"
    iv. Click "Create"
    v. Game server should start, and you should automatically connect to it

How it works

When you hit `Tools > Msf > Build All`, `MsfQuickBuild.cs` script creates two builds:

1. For Master and Spawner servers
2. For Client
3. For Game Server

1. Master And Spawner Build

Scene: `QuickSetup/Scenes/MasterAndSpawner`

If you look in the hierarchy, these are the **Game Objects of interest**:

- MasterServer - this objects starts the master server
- ConnectionToMaster - this object tries to connect to master server
- Spawner - registers a spawner to master server (it first needs to connect to it, which is why we have `ConnectionToMaster` object)
- DemoPlayerProfiles - a simple module, which construct player profiles on master server

Normally, you should start master server and spawner by using command line arguments, but for this example, I decided that it might be a bit easier to understand if I added UI for manual controls. This way, you can start it both manually and by using command line arguments (you'll find more information below).

2. Client build

Scene: `QuickSetup/Scenes/Client` + **all of the game scenes**.

When using Unet, clients usually share scenes with game servers. This is because client is usually exactly the same as game server, but it just doesn't run some of the code.

Game Objects of interest:

- ConnectionToMaster - this script automatically connects to master server
- UnetConnector - when client wants to enter a room, he receives a `RoomAccess` (which has game server ip, port and scene name). This script waits for access to be received, and then

loads the appropriate scene.

- Canvas/MsfUiCombined - this object contains all of the ui components from ui examples that come with master server framework.

3. Game Server build

Scene: `QuickSetup/Scenes/GameServer` + **all of the game scenes**

GameServer scene itself is almost empty, it's used mainly to switch to another scene, where actual game server starts. There's an example of game server scene, which is a bit more interesting.

Scene: `QuickSetup/Scenes/GameLevels/SimplePlatform`

Game Objects of interest:

- Msf/UnetGameRoom - this script waits for server to start, and then registers it to the master server
- Msf/RoomTerminator - tracks the number of online players, and shuts down the game server when there are none
- Msf/UnetServerStarter - this script analyses parameter from spawner, and uses them to start a game sever (for example, it reads `-msfAssignedPort` to know on which port to start the server)
- Msf/ConnectionToMaster - it's automatically destroyed, because there's already "ConnectionToMaster" script from the main scene (it doesn't get destroyed). This is added in case you don't want to connect to master on the main scene
- Msf/UnetConnector - this is used **by client** (remember, client and game server use the same scene. Of course, you can use different scenes, but I've tried to make this example simple). This script uses `RoomAccess` (most likely received on the previous scene) to know the address of game server, and connects to it.

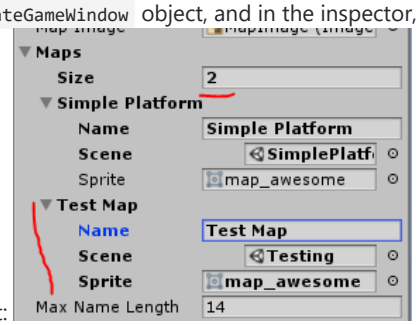
What to do next?

Copy the demo

I don't recommend making any changes to any of the files in the `QuickSetup` folder, because they will be overwritten after each update.

Instead, what I'd recommend you to do - copy the whole QuickSetup folder, and modify the `MsfQuickBuild` file (rename it, change build platforms, scenes and etc.)

Add more levels to client

1. Open the client's scene, and expand `Canvas/MsfUiCombined`
2. Select `CreateGameWindow` object, and in the inspector, add another scene to the `CreateGameUi` component: 

3. If you're using lobby creation UI, add the new map to `CreateLobby` Game object too

Now you'll need to make sure that game servers can be started on these levels.

Starting Game Servers on new levels (scenes)

Open the `SimplePlatformer` scene and make a prefab out of `Msf` object:

Add it to your new scene.

Make sure you understand what each of the game objects does. There are short descriptions above

Now modify your **Network Manager** to listen to room events, so that network manager knows when players join and leave the room.

If you're using the default **NetworkManager**, you'll need to replace it with the one below:

If you're already using a customized network manager, just **copy the lines of interest**.

```
using UnityEngine;
using UnityEngine.Networking;

/// <summary>
/// This script represents the changes that you will need to do in your custom
/// network manager
/// </summary>
public class ModifiedNetworkManager : NetworkManager
{
    // Set this in the inspector
    public UnetGameRoom GameRoom;

    void Awake()
    {
        if (GameRoom == null)
        {
            Debug.LogError("Game Room property is not set on NetworkManager");
            return;
        }

        // Subscribe to events
        GameRoom.PlayerJoined += OnPlayerJoined;
        GameRoom.PlayerLeft += OnPlayerLeft;
    }

    private void OnPlayerJoined(UnetMsfPlayer player)
    {
        // Spawn the player object (https://docs.unity3d.com/Manual/UNetPlayers.html)
        // This is just a dummy example, you'll need to create your own object (or not)
        var playerGameObject = new GameObject();
        NetworkServer.AddPlayerForConnection(player.Connection, playerGameObject, 0);
    }

    private void OnPlayerLeft(UnetMsfPlayer player)
    {
    }

}
```

Starting servers with command line arguments

Start Master Server and Spawner

`./MasterAndSpawner -msfStartMaster -msfStartSpawner -msfMachineIp xxx.xxx.xxx.xxx`

⚠ When starting a spawner, you should always set `-msfMachineIp` to the public IP of your server (the one that http://checkip.dyndns.org would give you). This address will be passed to spawned game servers.

🛈 This is not necessary when you run locally.

Start Master Only

`./MasterAndSpawner -msfStartMaster`

Start Spawner Only

```
./MasterAndSpawner -msfStartSpawner -msfMasterIp xxx.xxx.xxx.xxx
```

If you don't give an address, it will try connect to "127.0.0.1". This is all good when your master server runs locally, but when not - you'll need to give the IP

Start Spawner And Give A Path to Game Server Executables

```
./MasterAndSpawner -msfStartSpawner -msfExe 'D:/GameServer.exe'
```

Start Spawner With A Limited Capacity

```
./MasterAndSpawner -msfStartSpawner -msfMaxProcesses 5
```

Start Spawned Game servers with Websocket connection (WebGL support)

```
./MasterAndSpawner -msfStartSpawner -msfWebgl
```