# DPhysics Multiplayer Guide

## Part 1: Get Started

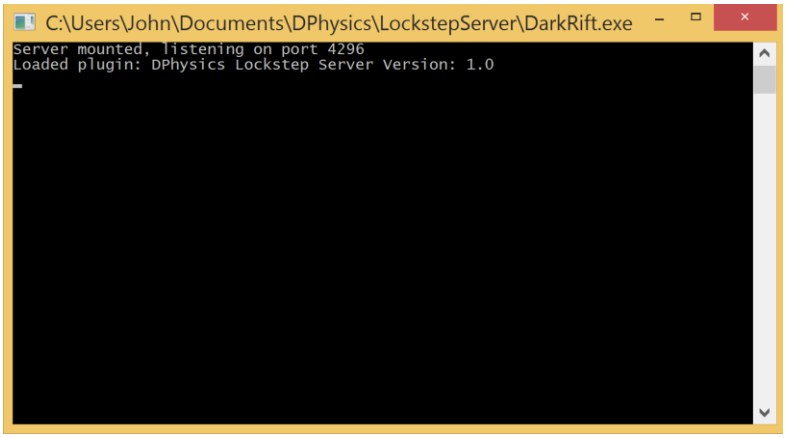
Welcome to the DPhysics Multiplayer implementation. Here, you will find a ready-to-go lockstep server and scripts to facilitate client logic, as well as this handy guide which will walk you through setting up and customizing your synchronous multiplayer game.

If you're already experienced with DarkRift, feel free to familiarize yourself with the scene then skip to Part 2 of this guide.

To set up your multiplayer game in less than 5 minutes, locate LockstepServer.zip inside the same folder as this guide, and extract it into a folder **outside of your project's assets**. You don't want Unity trying to compile your server or terrible things will happen.

Once you've extracted the LockstepServer.zip folder, open it and run the DarkRift.exe executable. Note that at the moment, DarkRift servers only support Windows.

Once you've started the server, it should look something like this:




Now that you have your server up and running, you can immediately test out your game. Open the MultiplayerScene in Unity, and press play. There should be several GUI elements that look like this:


The 127.0.0.1 is an IP address you can edit that represents the IP address of the server. If you're hosting on the same computer, 127.0.0.1 will connect to yourself, and can be used offline.

The "Connect" button connects you with the IP address in the field above. The "Step Count" label tells you how many simulation frames have passed. It's a piece of information that will help you conduct tests or debug, and it will haunt you forever (or until you delete it from the Client.cs script).

Click the "Connect" button and watch as 200 cubes spawn and bump into each other. It looks like magic now, but you'll know the ins and outs of it by the time we're through.

Feel free to play around with the example. In the top left, are several buttons that do exactly what they say. It might not seem like it now, but these buttons send a message up to the server then get sent back in a standardized package.

Part 2: Multiple Players

So now that you've had a taste of single-player DPhysics, you probably want to see what it can do in a multiplayer setting. This next part of the guide will help you do exactly that.

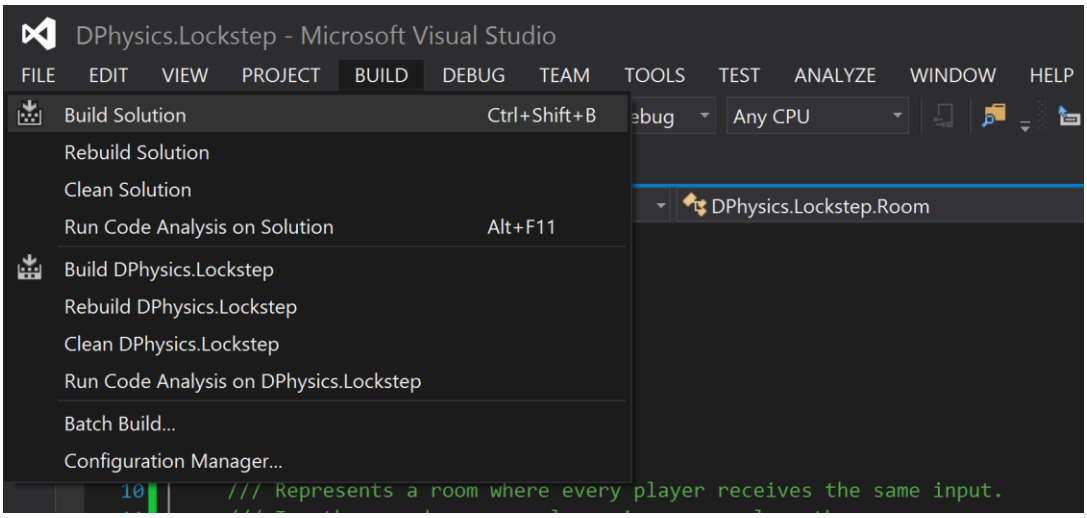
Out-of-the-box, the server is configured for 1 player per room. Let's change that. In the same folder of the server executable, locate and open the DPhysics.Lockstep folder. Inside it, you should find a solution file called DPhysics.Lockstep.sln. While I prefer to use Visual Studio to write server code, any IDE will work, including MonoDevelop.

Inside the solution, find and open Room.cs. This represents a room where each player's commands gets distributed to everyone in the room. Note that room in this sense means a shared game instance. At the very top of the script, is a static int called "MaxPlayers". As mentioned before, the server is configured for 1 player per room by this value. For this walk through, set this value to 2 so that 2 players can be in a room together.

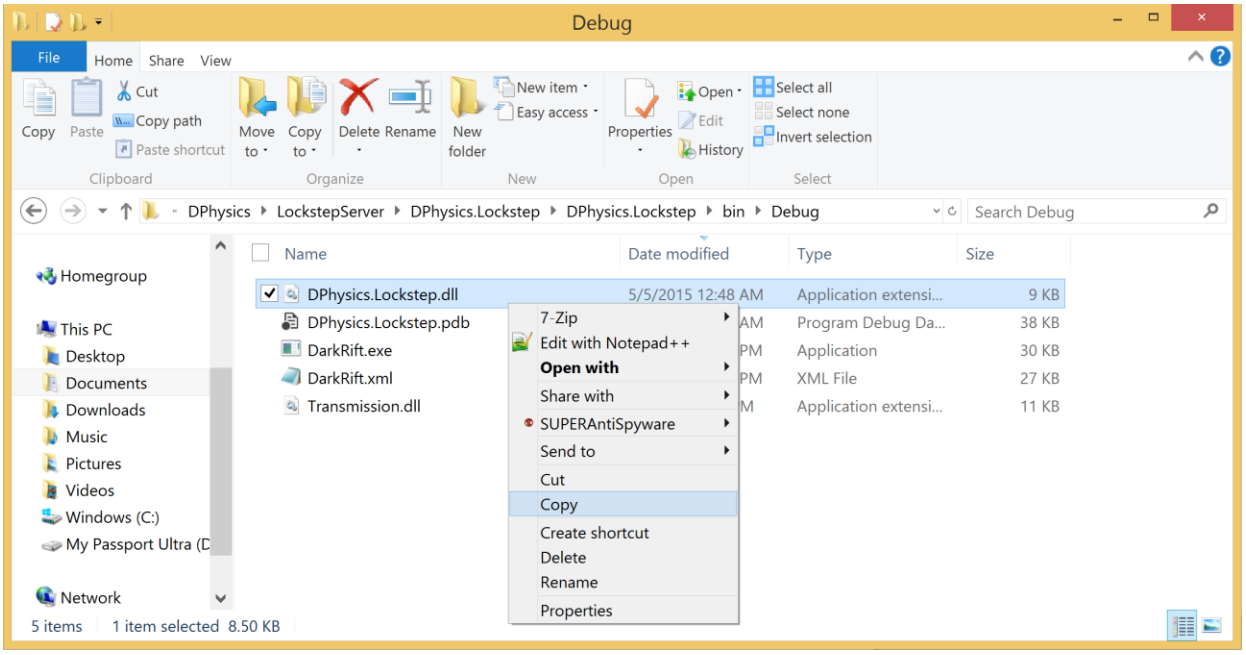
```
/// <summary>
/// Defines the maximum amount of players that can be in a room.
/// Once this amount is met, the room starts and no more players can join.
/// </summary>
public static int MaxPlayers = 2; //1;
```

Note: The rest of this section shows you how to build and integrate the plugin into your DarkRift server. It also walks you through how to emulate multiple players. If you're comfortable with all that, feel free to move on to Part 3.

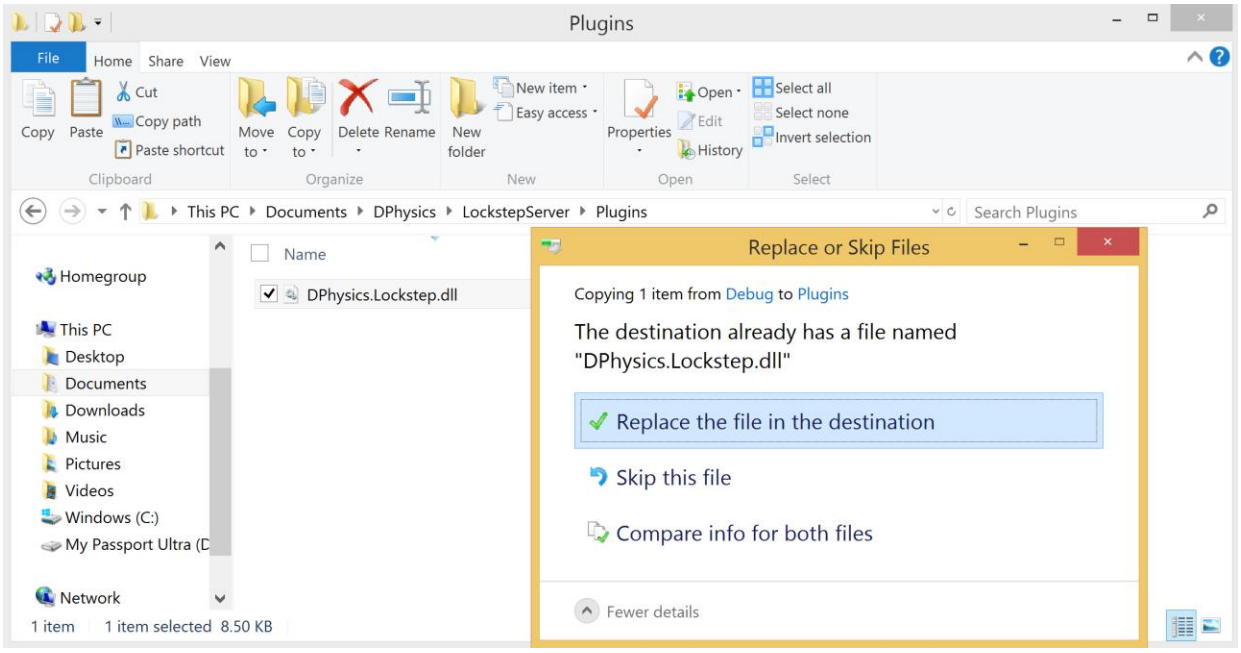
Once you've customized the server code, build the solution:



Locate the build file in the bin folder then copy it.




And finally, close your DarkRift server then paste the new plugin into the DarkRift server Plugins folder.




Now to test out our handiwork. Go back to Unity and build MultiplayerScene so that we can have two players playing at the same time.

Run one instance of your game in the editor, and one instance in the build. Note that the build will have a number of optimizations that drastically speed up DPhysics which is why the gameplay will probably run smoother in that instance.

Connect from both instances to the server. When the server detects that there are enough players in a room, it will automatically start and 300 perfectly synchronized cubes will spawn on both clients.

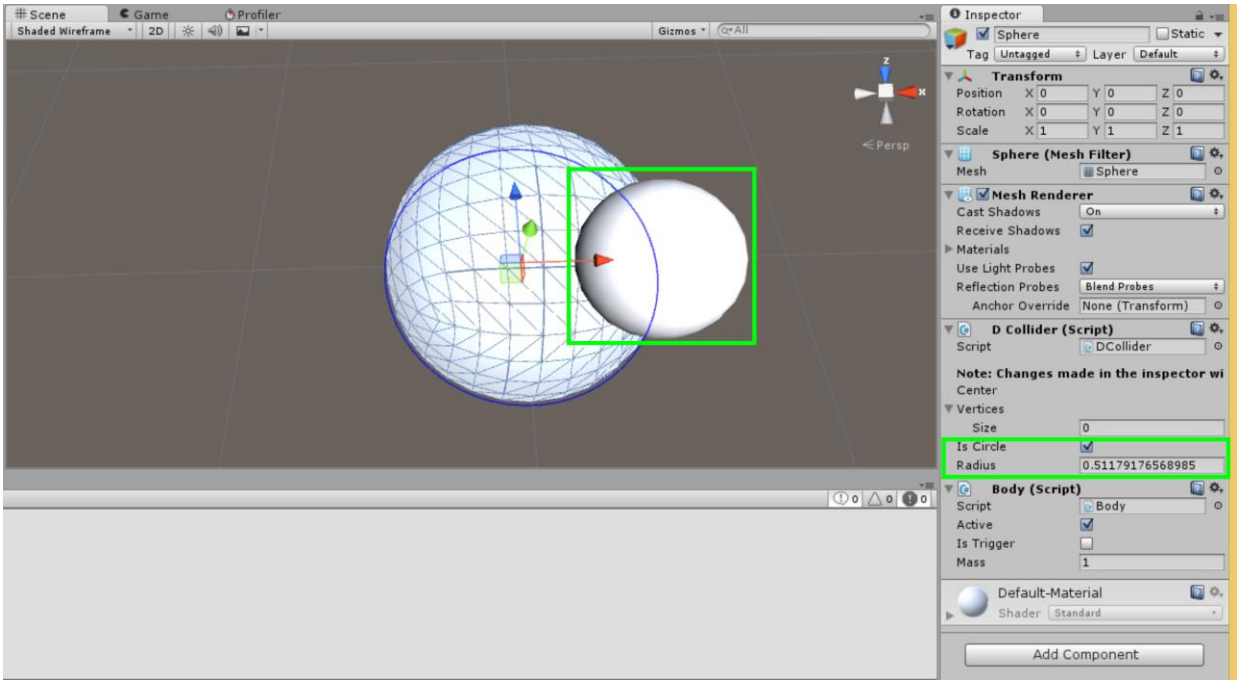
Part 3: Client-side

One of the coolest things about lockstep games is that game logic is the same for every instance. This means that you don't to mess with interpolation or complex authoritative server logic. You just create, and count on that creation being synchronized. One logic to rule them all.

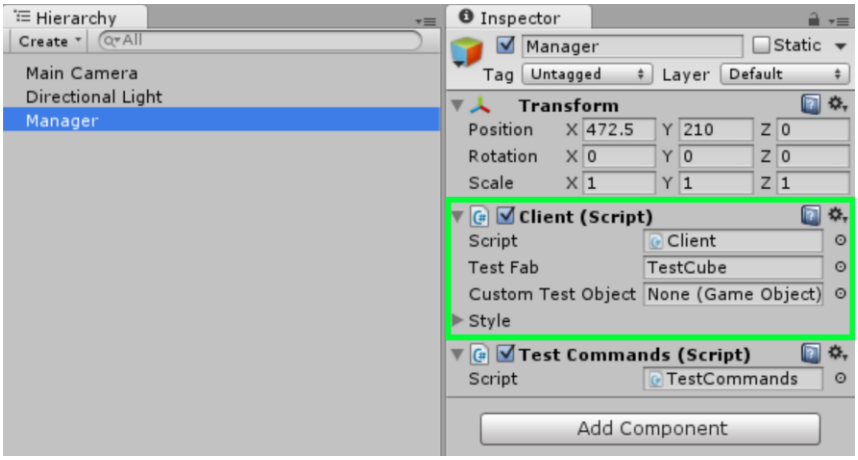
Note: If you've already explored Newton's examples and are comfortable with them, you can skip this part.

With that said, let's spiff up our game by adding some circles to the simulation. In Unity, create a sphere primitive and delete the Sphere Collider on it. Unity's colliders are very expensive, even when they're not being used.

Next, add the Body component to the object. Doing this will also automatically add a DCollider component. In the inspector, check 'IsCircle' on the DCollider component to be true. To edit its radius, you can drag the ball on the object's circumference in the scene view, or set the value in the inspector.




Create a prefab from this object you've just created. This will be the blueprint for the physics objects you create. Next, find the Client instance of the game, a component of Manager.




If you click on the TestFab field, you will notice that it leads to the prefab used in the original simulation of boxes. The other Game Object field, Custom Test Object, is for the physics object you created. Drag your object's prefab into the Custom Test Object.

Now it's time to spawn your object and assimilate it into the simulation. Open up Client.cs and find the TestSpawn() method. In the original simulation, this is where the boxes are spawned. You can piggy back on it and spawn your own objects along with the boxes. Locate the region in the method describing where to spawn your custom physics object.

```
166     public void TestSpawn ()
167     {
168         const int Iterations = 200;
169         FInt spacing = FInt.Create(1.5d);
170         int Width = (int)Mathd.IntSqrt ((long)Iterations);
171         int Height = Iterations / Width;
172         for (int i = 0; i < Width; i++) {
173             for (int j = 0; j < Height; j++) {
174                 Vector2d spawnVec = new Vector2d (spacing * i - Width / 2, spacing * j - Width / 2);
175                 Body body = Instantiate (TestFab).GetComponent<Body> ();
176                 body.Initialize (spawnVec);
177                 Vector2d randomVelocity = DPhysicsManager.RandomDirection * 4;
178                 body.ApplyVelocity (ref randomVelocity);
179 
180                 #region Spawn your custom physics object here
181 
182                 #endregion
183             }
184         }
185     }
```

Within that space, you should do several things:

1. Instantiate the prefab
2. Get the instantiated object's Body component
3. Initialize the Body with a starting position
4. [Optional] Generate a random velocity and apply it to the object

#1 and #2 and pretty straightforward using Unity's API:

**Body testBody = GameObject.Instantiate (CustomTestObject).GetComponent<Body>();**

Next, initialize the body with a simple call to Initialize, with a Vector2d position:

**testBody.Initialize (spawnVec); //You can use Vector2d spawnVec already created in the scope**

For #4, you can use the default random direction generator by getting DPhysicsManager.RandomDirection. It has a magnitude of 1 so you can manipulate it as you will.

**Vector2d ranVelocity = DPhysicsManager.RandomDirection * 4;**

Then apply the velocity to the object. Note that for performance reasons, the Vector2d velocity is passed on by ref. The Vector2d will remain unchanged after the call.

**testBody.ApplyVelocity (ref ranVelocity);**

Part 4: Congratulations!

You've just made a multiplayer lockstep game powered by DPhysics and DarkRift! I hope the commenting and documentation will be able to guide you through the rest of your development. If you have any questions or comments, please don't hesitate to contact me at JPtheK9@gmail.com or look for the Official DPhysics Thread on Unity forums.