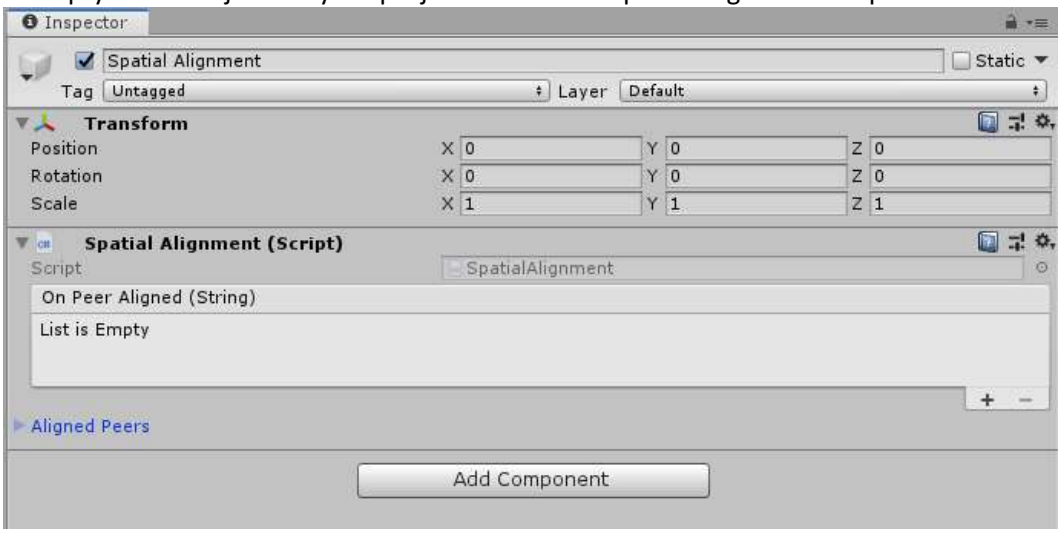
Implement Transmission in your Unity Project

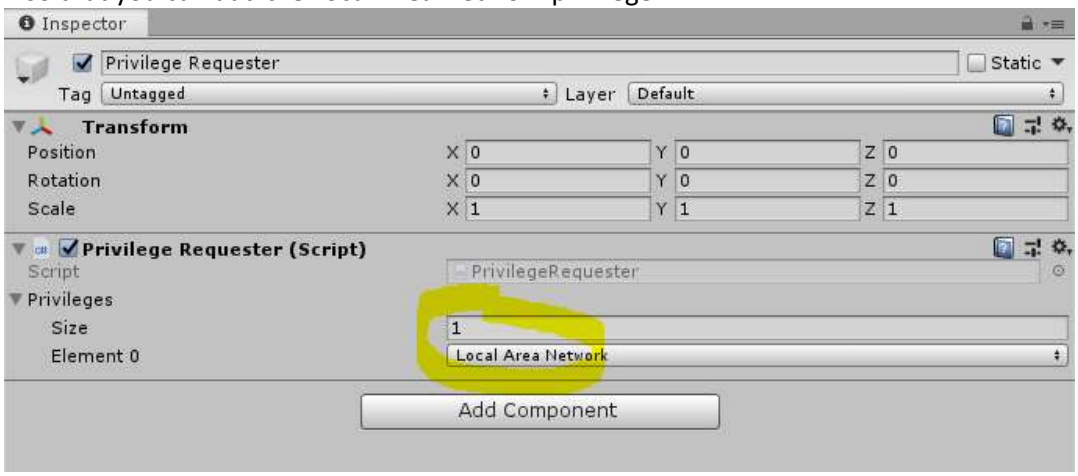
The steps:

1. Make sure you have the Magic Leap Toolkit (MLTK) installed.
    1. Remember that the current version that is up on ML's GitHub works with Lumin SDK 0.24.X and Unity 2019.3.6.f1. If you want to use MLTK with MLSDK 0.23.0 you will need to use the older version that is attached to this email or use an older commit from Dec 18, 2019.
2. Add 2 Empty Game Objects to your project. Attach the Spatial Alignment Script to one

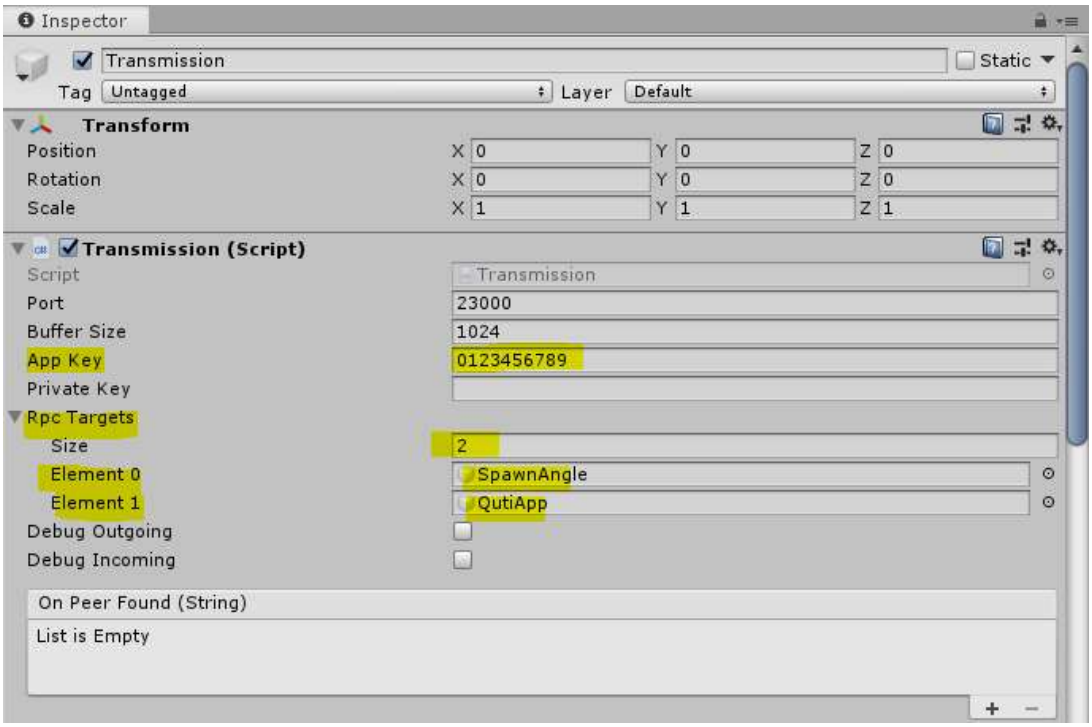


and the Privilege Requester Script to the other.

    1. You only need to make changes in the Inspector to Privilege Requester. Give it a Size of 1 so that you can add the Local Area Network privilege.



3. Add the Transmission prefab anywhere in the Hierarchy. You can find the prefab under Assets, MagicLeap-Tools, Prefabs, Networking (or simply search for it). Remember that you have to give your Transmission prefab a distinct App Key in the inspector for your apps to be able to communicate/synch (I believe that if you don't enter one, Unity will randomly generate an ID for you).



4. Finally, remember that Transmission only lets you (automatically) see changes in the transform of an object (location, rotation, scale) **if** an object is first **instantiated** via Transmission (Transmission.Spawn) from a **Resources** folder.
5. The following are some common Transmission functions:
    a. Spawn (equivalent to Instantiate), despawn (equivalent to Destroy), enabled (equivalent to .SetActive(true)), disabled (equivalent to .SetActive(false))

If instead you wish to see non-transform related changes in, for example, the material of an object, one way to do this is using Remote Procedure Calls (RPCs). More on RPCs under this link:

https://developer.magicleap.com/en-us/learn/guides/transmission-mltk

The way RPCs work is as follows:

- On the sender end:
    - Create a method that you call from inside your script and, in it, create a new RPCMessage

```
//RPC sender
private void SendAngleValueRPC(string AngleValue)
{
    RPCMessage rpcMessage = new RPCMessage("GetAngleValue", AngleValue);
    Transmission.Send(rpcMessage);
}
```

    - An RPCMessage takes multiple parameters but, the basic ones are the MethodToCall (as a string) and the parameter you want to send (also as a string)
    - Structuring an RPC call this way, will send the message to every player that is connected EXCLUDING you (this is fine if you don't want to execute the same method twice). If you want to further control which players should receive a call, take a look at the

TransmissionAudience parameter which let's you specify which Peers should receive the call)
    - If you'd like to send parameters like Vector3's, you will have to create your own function to convert from string to Vector3 (I attached one I already created to the end of this document)
- On the receiver end:
    - Add the game object containing the script that holds the methods that will be getting called under Rpc Targets on your Transmission prefab in the Inspector (refer to the highlighted fields in the third Figure)
    - Make sure that these methods are public otherwise the "spectator" app won't receive the message

Things to keep in mind when testing Transmission:

- Make sure that both devices are connected to the same wifi
- Make sure to open the Spectator App before the main app. This will ensure that all RPC calls are received.
- It is a good idea to send a Quit App RPC to all the players after the main device has quit the application. That way, any objects that got spawned via RPC calls will get cleared.

Converting from string to Vector3

```
//Function to convert from Vec3 that was converted to string back to Vec3
    private Vector3 StringToVector3(string sVector)
    {
        // remove the parentheses
        if (sVector.StartsWith("(") && sVector.EndsWith(")"))
        {
            sVector = sVector.Substring(1, sVector.Length - 2);
        }

        // split the items
        string[] sArray = sVector.Split(',');

        // store as a Vector3
        Vector3 result = new Vector3(
            float.Parse(sArray[0]),
            float.Parse(sArray[1]),
            float.Parse(sArray[2]));

        return result;
    }
```

Other useful info:

- Remember that you CAN connect players with different mpks as long as they share the same **app key**