networkView.RPC():

To send values over the network, and to other clients, you must call a special RPC function through the NetworkView component. Instances you might want to send/sync values across the network could include things like Player Health, Ammo, Position, Rotation, etc.

To do this, you must call the function as an RPC function (as seen below). This is built-in to the networkView component, which must be attached to the GameObjects associated.
The following example takes a value (in this case 'predefinedVariable'), and sends it to every associated networkView (through RPCMode.AllBuffered) in a function called SendValueToAll.

The RPC function SendValueToAll receives the predefinedVariable from the sender, and assigns it to a local variable 'rpcValue'. This allows you to use a value from another client for your own.

```
networkView.RPC("SendValueToAll", predefinedVariable, RPCMode.AllBuffered);


[RPC]
Void SendValueToAll (float rpcValue){
        valueFromOtherClient = rpcValue;
}
```

Some things worth mentioning:

- The RPC function will be automatically searched for in each script attached to the gameobject. While this is quite convenient, and saves having to specifically call it in code, you must take care not to have the same RPC function name in multiple scrtipts.
- A function won't be able to send/receive values unless it is prefixed with [RPC] *(@RPC for JS)*. With [RPC] before the function, the networkView component knows it can access it. Otherwise, it will ignore it.
- A NetworkView component *MUST* be attached to any gameObject that uses RPC functions. It can be found in Component -> Miscellaneous -> NetworkView

The included PlayerSync.cs script uses RPC calls to send the position and rotation of the player to other clients on the server, but only if have moved X units, or Y degrees.

The FPS scripts have only been slightly modified, and that's to disable the script if the networkView does not belong to us. This means that other clients on the server will have those input scripts disabled on their objects on our side, so that our input doesn't affect them. Since PlayerSync.cs syncs the pos/rot of the players, we don't need to send the input values across the network, so we can just disable the scripts.

RPCMode:
There are 3 main types of RPCMode: All, Others, and Server. ‘All’ sends the values to all clients on the server, including the object that called it. ‘Others’ sends it to all other clients on the server, but not the object that called it. ‘Server’ sends it only to the server.
Since each RPC call sends X amount of bytes per call, it’s important to pick the suitable type of RPCMode for each value. It is also possible to send multiple variables through networkView.RPC, by simply adding on values after your first variable. It’s important to note that if you are sending multiple values, the receiving function *must* have each variable accounted for, and every networkView.RPC call *must* contain a value for every variable.
An extended example is below.

```
networkView.RPC(“SendValueToAll”, predefinedVariable, secondString, extraVector3,
RPCMode.AllBuffered);


[RPC]
Void SendValueToAll (float rpcValue, string rpcString, Vector3 rpcVector){
        valueFromOtherClient = rpcValue;
        textString = rpcString;
        transform.position = rpcVector;
}
```

If you’ve tried typing this code into MonoDevelop, you may have noticed that there is in fact a variation of each RPCMode suffixed with ‘Buffered’. This is the recommended choice for important values.
This is because without choosing a Buffered method, the value may be discarded upon receiving for various reasons, such as a different value taking its place at just the wrong moment. By choosing buffered, the value will be, as expected, buffered for use, so instead of being discarded, will be used when possible.
Note: This doesn’t mean the value will be stored for very long – this is literally a matter of single frames at most. Buffered is highly recommended when dealing with values that absolutely need to be sent/received, such as input values.