VRTogether Networking System

There are two main components of the new networking system: the MacrogameServer/Client and the MinigameServer/Client. These are singletons that are always accessible through their Instance member. MacrogameServer should only be instantiated once so take care to not have one in any scene other than an initialization scene of some sort. This singleton should only be accessed when you are trying to change or access some data relating to the macro-game players (name/score).

The MinigameClient is a temporary interface between the minigame and the network. Make sure each minigame scene has one of these. This is what you will be working with most often. It holds the ability to instantiate and destroy objects across the network, as well as registering synchronized variables.

In order to properly synchronize scenes all scenes should share the same name in both builds.

In order to properly synchronize prefabs all prefabs should share the same name in both builds.

In order to properly synchronize instantiation/destruction all networked prefab lists should be identical in both builds, but do not necessarily need to share the same order.

New networked prefab lists can be created by right-clicking -> VRTogether -> Networked Prefab List.

The 4 Types of Objects

Note that there is an example of each one of these in the NetPrefab/ directory.

1) The static object
    a. This object does NOT need to be networked but must share all the same non-changing data inside of it. This includes transform variables.
2) The authoritative object
    a. This object is authoritative over its information and will relay it to all other connected players.
    b. This object should be spawned using the EZSpawn script on the client/server that assumes control of it.
3) The slave object
    a. This object is not authoritative of its information, and only reflects information passed to it over the network. It is instantiated by the network.
    b. In order to stop unwanted behavior you should always be sure to check if an object is a slave or not (See the NetworkFly example in Android/Desktop, they are different but are used for the same networked object. One is a slave or auth, the other is always a slave). For example, only non-slave objects should be able to score a goal. If this was not the case, each goal scored would score as many goals as there were clients connected to the server.

Instantiating Objects

To instantiate an object, you should call MinigameServer/Client.NetworkInstantiate(prefab).

The prefab supplied MUST be a part of the NetworkedPrefabList that is given to that MinigameServer/Client. Whichever client calls NetworkInstantiate will be considered the “owner” of that object and will supply the rest of the network with updates. It is therefore best to use NetworkedInstantiate on objects that you would expect that local client to need zero latency with.

NOTE: In order to avoid issues with synchronizing instantiated objects, PLEASE be sure to use MinigameServer/Client.AllPlayersReady() to make sure that all players are loaded before instantiating any networked objects. Failing to do so will cause desynchronization at a massive scale immediately.

Destroying Objects

To destroy an object over the network you should call MinigameServer/Client.NetworkDestroy(obj).

The object can be either a slave or an object that this local player has ownership of. There is no restriction as long as this object is indeed a networked object.

Loading Minigames

To load a minigame simply call MacrogameServer.LoadMinigame(sceneName).

This is only available on the server. This will load the scene immediately on the server, and then inform all clients that they should load that scene. This will also block orientation synchronization until all players have been loaded.
NOTE (again): In order to avoid issues with synchronizing instantiated objects, PLEASE be sure to use MinigameServer/Client.AllPlayersReady() to make sure that all players are loaded before instantiating any networked objects. Failing to do so will cause desynchronization at a massive scale immediately.

Changing Macro Scores

TBD you can try it yourself by accessing the information publicly available in the MacrogameServer.

Syncing Variables

Synchronizing variables is super simple, but you must use the new NetworkInt/Float/Bool classes instead of regular primitives. You can create a new network primitive variable and assign it a name different from its actual name in the code, but this is unadvisable. The other argument is always the default starting value, ex:

```
private NetworkBool holdingGrape = new NetworkBool("holdingGrape", false);
```

Finally in order to start synchronizing the network variable you have to register it with the local minigame manager (server/client) on both the client AND the server version of the object, ex:

```
MinigameServer.Instance.RegisterVariable(id.netID, holdingGrape);
```

The first variable you give it should be the network ID of the object that owns that variable. Since network IDs are synchronized before variables it is safe to feed the networkID in the start method.

Helpful Fly Example Tips:

Use the EZSpawner code to spawn in a networked entity. This automatically waits for the minigame players to be loaded before creating an object that needs to be networked.
Use the LoadMinigame script to have editor access to the network scene load script.