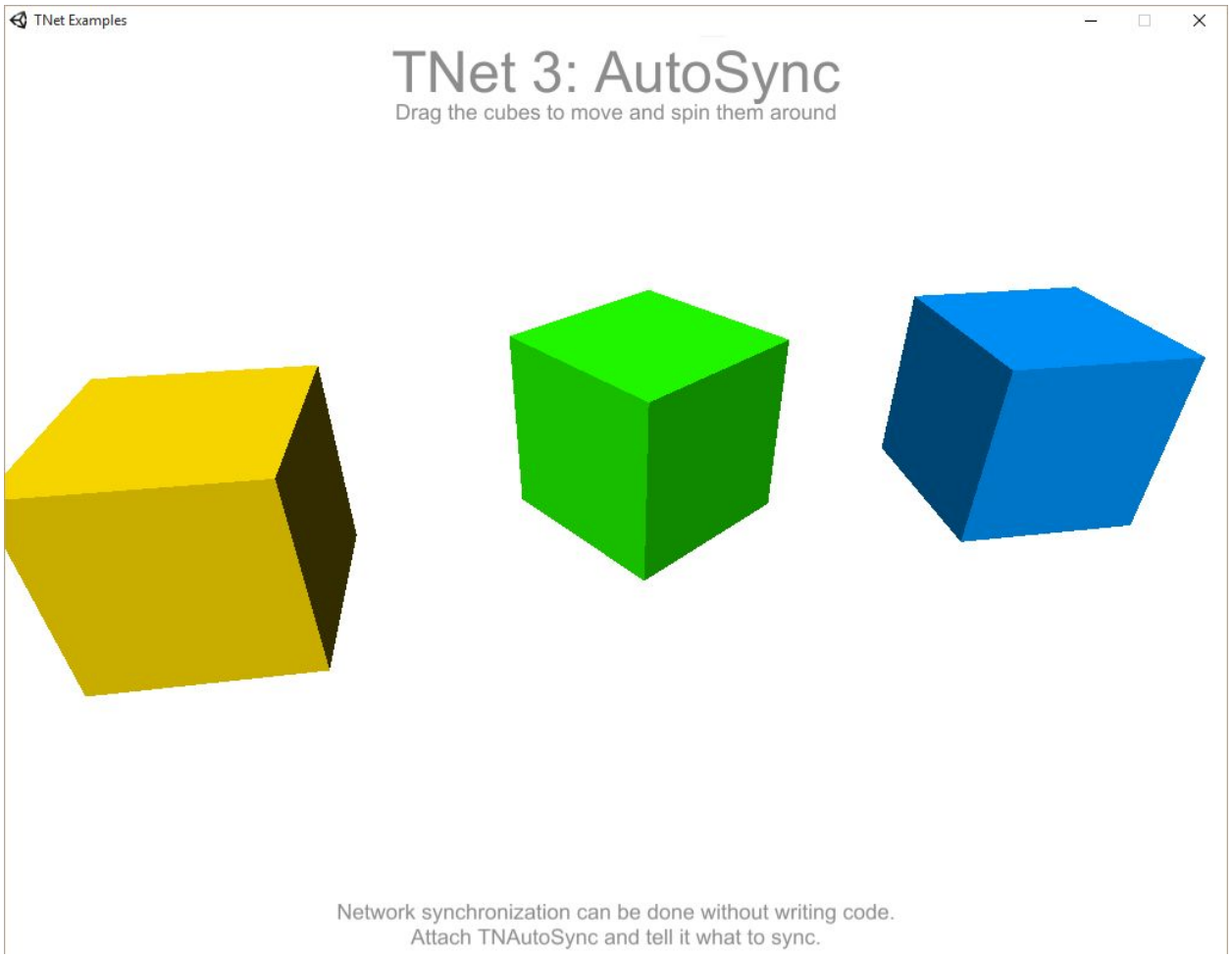
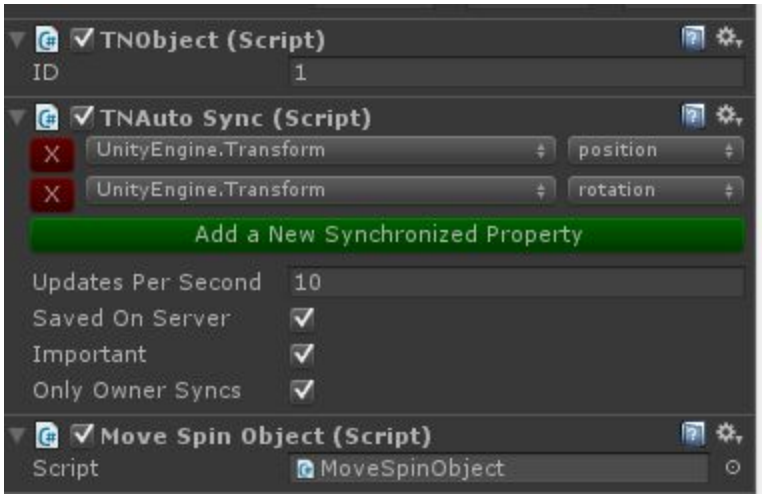
TNet 3: AutoSync Example In-Depth



The AutoSync example displays 3 Cubes and allows multiple users to move them vertically and spin them around the Y-axis. There are four component scripts, which when working together make this possible.

The first one, `TouchHandler`, is on the Main Camera object. It reads in any input information from the mouse and converts that to simple broadcast messages which other scripts can respond to. `OnPress` is when the mouse is pressed down, `OnClick` is sent on mouse up, and `OnDrag` is sent after a press occurs and the mouse moves. The broadcast messages are then sent to the object directly under the mouse.



The Three “AutoSync Cube” objects include the rest of the scripts.

`TNObject` is at the heart of the TNet system. Every networked object requires a `TNObject` script and a unique ID number. When an object is manually placed in the scene (as this example does) the ID number must be unique and pre-set. If an object is spawned via the instantiate command, a unique ID number is picked automatically.

The `TNObject` script itself includes various properties such as `.owner`, `.isMine`, `.Send()`, `.SendQuickly()`, but they are mostly used when your class inherits from the TNBehaviour class instead of directly.

`TNAutoSync` is a helper class that does a lot of heavy lifting to get you started quickly. It will sync any properties on your object X times per second over the network. It’s great for quick prototypes and getting started, but for a full project you will probably want to write this functionality yourself. Doing so will let you finesse its behaviour (See the Frequent Packets and Movement examples for examples on how this is done).

In the case of this example, it updates the transform.position and transform.rotation 10 times per second. “Saved on server” means that if a new client were to connect, the server would have the position and rotation information on hand to start the cubes off in the same position that everyone else sees them at. If the channel is persistent, the server would write that information to its hard drive, so even after a server reset, it would still know the proper position.

The “Important” flag determines how the information is sent over the network. Using the `.Send` or `.SendQuickly` commands. Send sends the information using TCP and will verify that it reached everyone. Send quickly sends information using UDP (where possible) which is more of a fire and forget type attitude. The packet will probably make it to the other clients, but it isn’t guaranteed. This is useful when sending information which needs to be updated quickly and will be overwritten.

Every `TNObject` has an `owner` assigned to it. It's the client's `Player` object. "Only Owner Syncs" determines if player on the network, can sync the position and rotation information. Typically you want this on. In the "Frequent Packets" example scene, this value is on, but the `.owner` property is transferred between users as they drag the objects to make sure that only one person can adjusts the cube's position at a time.

And finally the `MoveSpinObject` script. This is what moves and spins the cubes.

```
using UnityEngine;
using TNet;
```

It starts off by letting C# that we want to use the standard UnityEngine API, and instead of using `System.Collections` which is normally included, we're going to include `TNet` instead. This gives us access to all of the TNet functions.

```
public class MoveSpinObject : TNBehaviour
```

The class then inherits from `TNBehaviour`. Internally `TNBehaviour` inherits `MonoBehaviour` but also links into `TNObject`. With the inherited class `TNObject`'s properties can be accessed by typing `tno.isMine` or `tno.owner`, etc. at any point in our code.

```
public class MoveSpinObject : TNBehaviour
{
	void OnDrag (Vector2 delta)
	{
		if (tno.isMine)
		{
			Vector3 euler = transform.eulerAngles;
			euler.y -= delta.x * 0.5f;
			transform.eulerAngles = euler;

			Vector3 pos = transform.position;
			pos.y += delta.y * 0.01f;
			transform.position = pos;
		}
	}
}
```

The `OnDrag` sent by the Touch Handler script on the Main Camera is received here and the first thing this script does is check to see if this network client owns this cube. If it does, then it can continue.

The `delta` information from the mouse drag is used to adjust the vertical position and horizontal rotation of the cube locally. And as long as you're connected, the networked clients

will receive the updated position and rotation periodically (10 times per second) via the TNAuto Sync script.