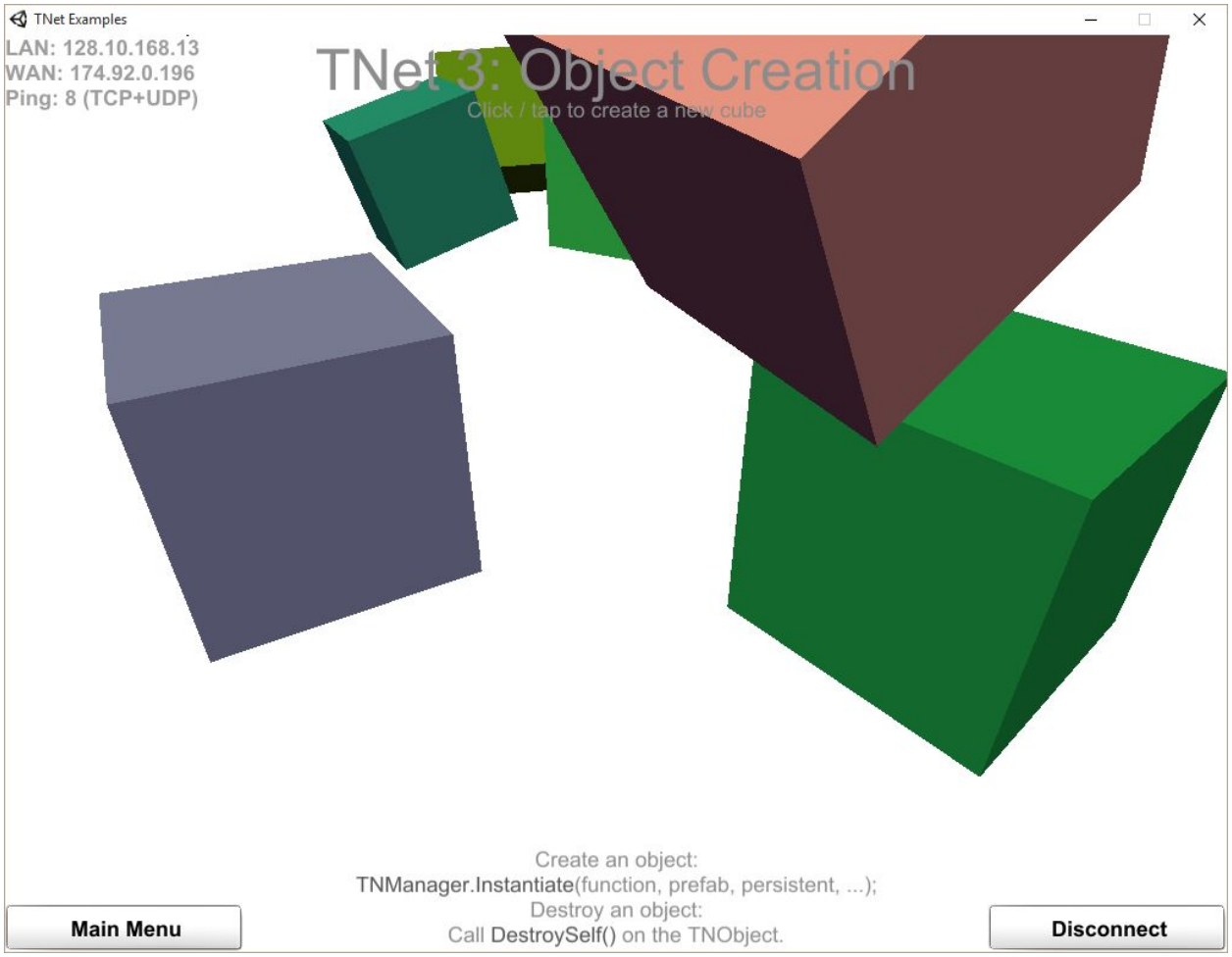
TNet 3: Object Creation Scene In-Depth

The Object Creation example demonstrates how to instantiate objects with custom information using a Remote Creation Call (RCC).



The The Main Camera contains the `TouchHandler` component script to convert mouse clicks into broadcasted messages such as `OnClick()`. The Floor object has a component called `ExampleCreate` which reacts to the OnClick event and instantiates prefabs in the scene.

Take a look at ExampleCreate's OnClick event.

```
void OnClick ()
{
	// Let's not try to create objects unless we are in this channel
	if (TNManager.isConnected && !TNManager.IsInChannel(channelID)) return;
```

```
		// Object's position will be up in the air so that it can fall down
		Vector3 pos = TouchHandler.worldPos + Vector3.up * 3f;

		// Object's rotation is completely random
		Quaternion rot = Quaternion.Euler(Random.value * 180f, Random.value *
180f, Random.value * 180f);

		// Object's color is completely random
		Color color = new Color(Random.value, Random.value, Random.value, 1f);

		// Create the object using a custom creation function defined below.
		// Note that passing "channelID" is optional. If you don't pass anything,
TNet will pick one for you.
		TNManager.Instantiate(channelID, "ColoredObject", prefabName, true, pos,
rot, color, autoDestroyDelay);
}
```

It checks to make sure that the client is connected to a server and it's in the proper channel. If it is, point 3m above where the user clicked the floor is selected. Next it spins the object randomly 180 degrees on each axis, and then picks a random colour.

`TNManager.Instantiate` then instantiates the specified prefab ("Created Cube") in the channel and passes the information off to an RCC function named `ColoredObject`. If the user who instantiated the cube disconnects, the cube will remain in the scene for everyone else to see because "persistent" parameter was passed as `true`.

```
[RCC]
static GameObject ColoredObject (GameObject prefab, Vector3 pos, Quaternion rot,
Color c, float autoDestroyDelay)
```

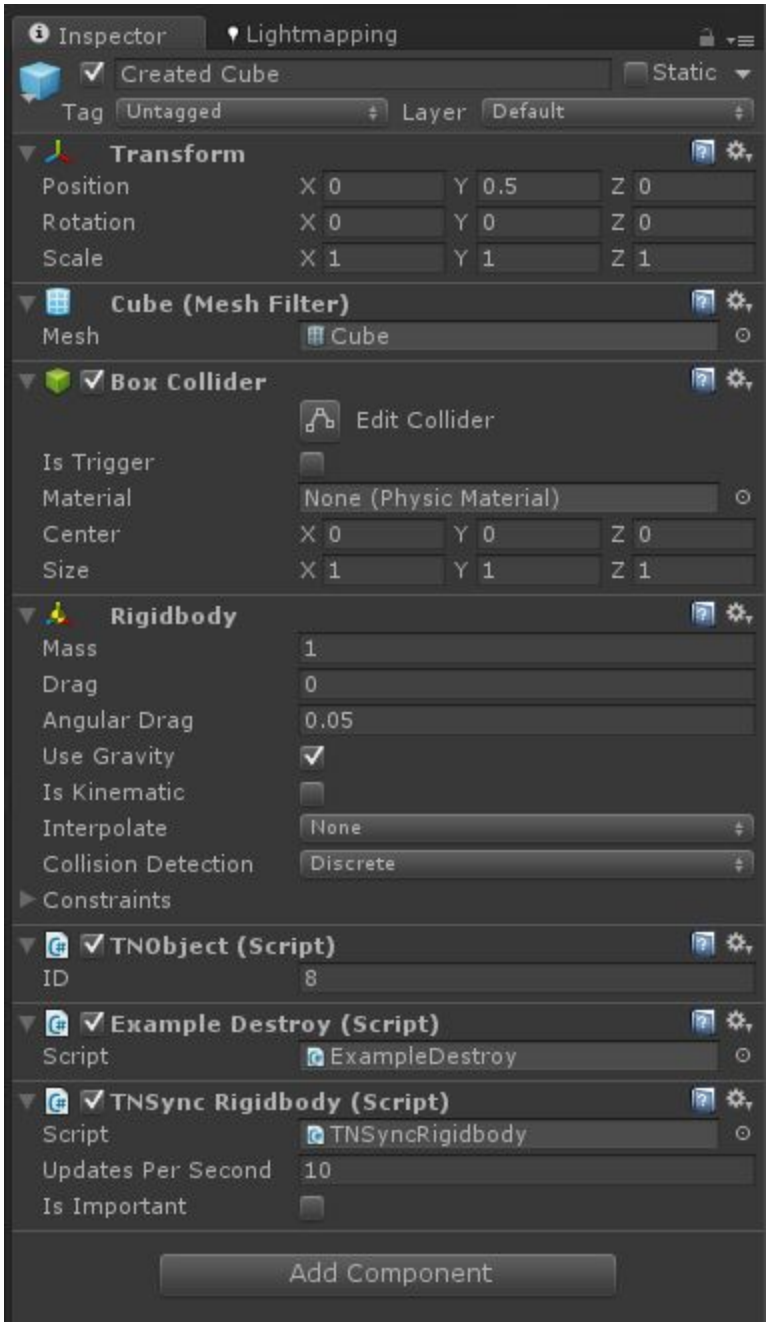
The `ColoredObject` function is marked as an RCC in order for it to be instantiated on all clients in the same channel. All `[RCC]`'s which respond to the `TNManager.Instantiate` function require a GameObject prefab to be passed in and they will return an instantiated version of that prefab in the end.

The remainder of the parameters are custom for this function. The only limitations on the parameters are the count and type must match the parameters sent from `TNManager.Instantiate`.

The prefab is instantiated locally, it's transform and colour are set and if the `autoDestroyDelay` is greater than 0, the GameObject is flagged to delete itself. In the case of this example it deletes itself in 5 seconds.

Now take a look at the Created Cube prefab.

<...scroll down...>



The cube is a basic 1kg cube, which is affected by gravity. Due to it being a networked object, it requires the `TNObject` ("Network Object") script. It also includes the `ExampleDestroy` script which destroys the cube when clicked. And finally the `TNSyncRigidbody` component script notifies other clients of its physical position and velocity in the world.

The ID on the `TNObject` script doesn't matter, as it's automatically changed when TNManager instantiates a new one.

`TNSyncRigidbody` is also a `TNBehaviour` based class which synchronizes the position, rotation, velocity and angular velocity over the network 10 times a second.

To start off with components and values are cached and then the UpdateInterval is set.

```
void UpdateInterval ()
{
	mNext = Random.Range(0.85f, 1.15f) *
		(updatesPerSecond > 0f ? (1f / updatesPerSecond) : 0f);
}
```

This takes the Updates Per Second field from the Inspector and shifts it ever so slightly. This random shift will make sure that not every cube will attempt to update on exactly the same frame. With enough cubes, that could potentially cause graphical hitching in your game.

The FixedUpdate function is where the physics are synchronized over the network, but first it does a number of checks. Each one of these is designed to reduce the traffic on the network to cut down on lag as perceived by the player.

```
if (updatesPerSecond < 0.001f) return;
```

This checks to make sure that the UpdatesPerSecond variable isn't too small. If it is, it's treated as if it were 0 and no updates occur.

```
if (isActive && tno.isMine && TNManager.IsInChannel(tno.channelID))
{
	bool isSleeping = mRb.IsSleeping();
	if (isSleeping && mWasSleeping) return;
```

If the cube belongs to this client, and the client is in the channel then continue. Next it does a check to see if it's "sleeping." A physics object goes to sleep if it hasn't moved for some set period of time. If it hasn't, no network update is needed and we can avoid sending that information out.

```
mNext -= Time.deltaTime;
if (mNext > 0f) return;
```

Time is then removed from the mNext timer. If it's greater than 0 we don't need to update yet, and can leave.

```
if (mWasSleeping || pos != mLastPos || Quaternion.Dot(rot, mLastRot) < 0.99f)
```

If the cube was sleeping but isn't any longer or the current position has changed or the rotation has changed then we can finally send a network packet.

```
if (isImportant)
{
	tno.Send(1, Target.OthersSaved, pos, rot, mRb.velocity,
	mRb.angularVelocity);
}
else tno.SendQuickly(1, Target.OthersSaved, pos, rot, mRb.velocity,
mRb.angularVelocity);
```

If the `IsImportant` bool in the inspector is checked, the network packet is guaranteed to make it to the destination by sending a TCP packet. If it's not important UDP is used and there will be no confirmation that it was received.

The `tno.Send` functions send the information to `[RFC(1)]` on all clients in the current channel except the owners. The owners has already moved the object using physics and sending that information would be redundant (and likely cause issues with the physics engine).

```
[RFC(1)]
void OnSync (Vector3 pos, Quaternion rot, Vector3 vel, Vector3 ang)
```

`RFC(1) OnSync` receives the position, rotation, velocity, and angular velocity over the network and sets the local copy to match. The local physics engine will then take over and perform similar calculations to the owner's machine until the next OnSync comes in and snaps it to exactly the same place as the owner's machine. With decent ping times, you won't see this snapping at all.

```
void OnCollisionEnter () { if (tno.isMine) Sync(); }

public void Sync ()
{
	if (isActive && TNManager.IsInChannel(tno.channelID))
	{
		UpdateInterval();
		mWasSleeping = false;
		mLastPos = mTrans.position;
		mLastRot = mTrans.rotation;
		tno.Send(1, Target.OthersSaved, mLastPos, mLastRot, mRb.velocity,
	mRb.angularVelocity);
	}
}
```

If there a collision has occurred on the owner's client, then reset the update timer and force all other clients to update their physics information as soon as possible. If the object was sleeping, make sure that it no longer is. This message is sent using `Send` and not `SendQuickly` because it's important that every client knows about this.

After all those steps are taken, you will be able to move almost any object in your game while sending a minimal amount of network traffic out over the network.