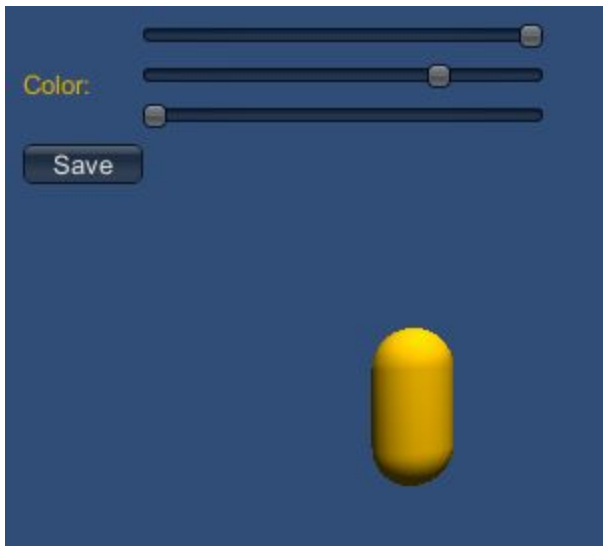
TNet 3: Synchronizing and Saving Player Data

When making a game, saving data is almost always necessary. Doing this with TNet is easy; Whether your game is singleplayer or multi-player, TNet is fully capable of automatically saving both the packets that you send out in channels as well as any custom data you desire. This tutorial will go through the process of creating a networked scene that saves an object's colour to the server.



Project Setup

Before getting started on the code the project has to be properly set up.

- Create a new Unity Project
- Import TNet
- Select Edit -> Project Settings -> Player, turn on "*Run in Background**"
- Create a new folder in the project window called `Resources`
- In the Hierarchy window right click and select Light -> Directional light
- In the Hierarchy window right click and select Create Empty
- Rename GameObject to GameManager
- Right click in the Project window and select Create->C# Script and name it `GameManager`
- Drag the `GameManager` C# Script onto the GameManager GameObject in the Hierarchy window. This tutorial will focus on modifying this script's contents.

Game Manager Script

The GameManager is what we will use to control our game at its highest level. In the GameManager.cs script delete the Start and Update functions that Unity provides by default.

In the `using` section at the top add `using TNet;` to the list and remove `using System.Collections`. You should now have the following:

```
using UnityEngine;
using TNet;

public class GameManager : MonoBehaviour
{
}
```

On startup the client needs to connect to the network, and then join a channel. At the same time, define a Color member variable which can be used later.

```
public class GameManager : MonoBehaviour
{
	Color mColor;

	void Start()
	{
		TNServerInstance.Start(5127);
		TNManager.Connect("127.0.0.1");
	}
}
```

Now that a local server has been started on port 5127 and the client has connected to it an OnNetworkConnect event will trigger, but it needs to be handled. Create a OnEnable and OnDisable functions and register functions to handle connecting and joining a channel.

```
void OnEnable()
{
	TNManager.onConnect += OnNetworkConnect;
	TNManager.onJoinChannel += OnNetworkJoinChannel;
}

void OnDisable()
{
	TNManager.onConnect -= OnNetworkConnect;
	TNManager.onJoinChannel -= OnNetworkJoinChannel;
}


void OnNetworkConnect(bool success, string message)
{
```

```
	if (success)
	{
		TNManager.SetPlayerSave(SystemInfo.deviceUniqueIdentifier +
	"/Player.dat");
		TNManager.JoinChannel(1);
	}
	else Debug.LogError("OnNetworkConnect(): " + message);
}
```

In OnNetworkConnect a relative save path for this user is setup on the server. From this point on, any time `SetPlayerData` is called, this file on the server will automatically update. Even through a server restart, the players information will be saved. The save path needs to be unique per player in order to avoid overwriting one another's information. Here the device's unique id is used, but you probably want to use the player's login credentials instead.

And then the game joins channel 1.

```
void OnNetworkJoinChannel(int channelID, bool success, string message)
{
	if (success)
	{
		Vector3 position = new Vector3(Random.Range(-3.0f, 3.0f), 0.0f,
	Random.Range(-3.0f, 3.0f));
		TNManager.Instantiate(2, "Avatar", false, position,
	Quaternion.identity);
	}
	else Debug.LogError("OnNetworkJoinChannel(): " + message);
}
```

If joining channel 1 was a success, a random location is picked and then a prefab called "Avatar" is loaded from the Resources folder and instantiated in the scene.

```
void OnGUI()
{
	GUI.contentColor = mColor;
	GUI.Label(new Rect(10, 24, 60, 20), "Color:");
	GUI.contentColor = Color.white;

	mColor.r = GUI.HorizontalSlider(new Rect(70, 10, 200, 20), mColor.r, 0.0f,
1.0f);
	mColor.g = GUI.HorizontalSlider(new Rect(70, 30, 200, 20), mColor.g, 0.0f,
1.0f);
	mColor.b = GUI.HorizontalSlider(new Rect(70, 50, 200, 20), mColor.b, 0.0f,
1.0f);

	if (GUI.Button(new Rect(10, 70, 60, 20), "Save"))
	{
		TNManager.SetPlayerData("Color", mColor);
	}
}
```

The `OnGUI()` function creates three sliders to adjust the colour. When save is pressed the colour change is passed over the network to all the clients and saved on the server.

There is still one issue remaining with the user interface. When started up the sliders don't have the proper colour information. To do that, the `onSetPlayerData` callback needs to be hooked up.

```
void OnEnable()
{
	TNManager.onConnect += OnNetworkConnect;
	TNManager.onJoinChannel += OnNetworkJoinChannel;
	TNManager.onSetPlayerData += OnSetPlayerData;
}

void OnDisable()
{
	TNManager.onConnect -= OnNetworkConnect;
	TNManager.onJoinChannel -= OnNetworkJoinChannel;
	TNManager.onSetPlayerData -= OnSetPlayerData;
}

void OnSetPlayerData(Player p, string path, DataNode node)
{
	if (p == TNManager.player) mColor = p.Get<Color>("Color", Color.yellow);
}
```

When OnSetPlayerData gets triggered, it checks to see if the player being updated belongs to this client. If it does, `mColor`'s contents is populated with the `"Color"` stored in the players dataNode.

The `onSetPlayerData` callback delegate returns three properties. The `Player` object, is all of the players information including the entire `dataNode` hierarchy. `Node` is only the `dataNode` field which was modified. In this case it will always be the Color data node, however if you set more data than just the player's colour, that value will change. And `path` is the path to the variable which was changed. In this example the path is simply `"Color"`, but it can be along the lines of `"My/Custom/Path/Color"`.

Setting up the avatar

Back in Unity, it's time to create an Avatar for the user that is connecting to the server. For simplicity's, use a capsule.

- Right click in the hierarchy window and select 3D Object -> Capsule
- In the inspector press Add Component and type `Network Object` to attach the TNObject script.
- Create a new C# Script

- Name it `Avatar`
- Add the `Avatar` Script to the Capsule
- Drag the Capsule object from the Hierarchy window into the **Resources** folder to create a prefab. It's important that it goes into the Resources folder or else it will not load properly.
- Rename the capsule prefab to `Avatar`
- Delete the capsule from the scene's Hierarchy window.

Storing and Retrieving Data

This tutorial will only save colour information, but player syncing saves an `object`, so many different types of information can be saved and loaded. All of the basic types are supported out of the box.

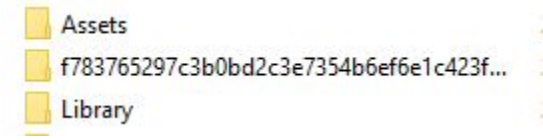
To use the information on the server, the Avatar will need to monitor onSyncPlayer events and deal with the data. The callbacks should be hooked up as early as possible, which means they have to go in the OnEnable and OnDisable functions. Open the Avatar.cs file and replace its contents with the following.

```
using UnityEngine;
using TNet;

public class Avatar : TNBehaviour
{

	protected override void OnEnable()
	{
		base.OnEnable();
		TNManager.onSetPlayerData += OnSetPlayerData;
	}

	void OnDisable()
	{
		TNManager.onSetPlayerData -= OnSetPlayerData;
	}

	void Start()
	{
		SetProperties(tno.owner);
	}

	void OnSetPlayerData(Player p, string path, DataNode node)
	{
		if (p == tno.owner) SetProperties(p);
	}

	void SetProperties(Player p)
	{
		Color c = p.Get<Color>("Color", Color.yellow);
```

```
            GetComponent<Renderer>().material.color = c;
        }

}
```

Every time a property is updated, the onSetPlayerData event will trigger. When that occurs, a callback function will receive a TNet Player object. That player will be the one which has new information. If that player is the GameObject which owns this Avatar script, then set the material colour to its proper value. If not, just ignore the event.

Press play and you will see a coloured capsule. Adjust the sliders, press the save button, and a unique folder will show up in your unity project with a player.dat file inside. That is the server's save file.



The completed tutorial can be downloaded here: [No zip file yet]