TNet 3: Custom Channel Data

This tutorial will step through the process of viewing the channel information and will show how to include any extra channel-specific data that your game may require. The extra information in this case will be a channel name and information field that might show up when the user first connects -- but you can store anything you want.

Project Setup

- Start a new project
- Import TNet into it
- Edit -> Project Settings…, Click the Resolution and Presentation section. Turn on "**Run In Background***" This will stop the client from timing out when it doesn't have focus.
- Right click the Hierarchy window and select Create Empty. Rename the newly created GameObject to GameManager.
- Right click the Project window and select Create -> C# Script
- Name the script `GameManager`.
- Attach the GameManager C# script to the GameManager GameObject.
- Double click the GameManager C# script to open it up in an editor.

Connecting to the Server

To start off, delete the `Start` and `Update` functions that Unity generates by default. In the "`using`" section at the top, change `using System.Collections;` to `using TNet;` Your file should now look like the following.

```
using UnityEngine;
using TNet;

public class GameManager : MonoBehaviour
{
}
```

Start a server on port 5127, and then connect to it. The IP address 127.0.0.1 is a loop back IP which always points to the local machine and the default port is 5127.

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

Once the project connects to the server, the connection event needs to be handled. `OnNetworkConnect` and `OnNetworkJoinChannel` functions will be created to handle the connection to the server, but first they need to have their callbacks registered.

If you would like to see which callbacks can be used `TNEventReceiver.cs` is an abstract class which registers all of the common events. It can be used for prototyping, but it would most likely be wasteful in a final project. It is, however, a useful place to look up all of the callback definitions in one location.

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

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

When the GameManager is enabled the onConnect event is now going to be handled by a function named OnNetworkConnect and the same for the onJoinChannel event. To avoid any other potential problems, the functions are also removed from the list when the GameManager is disabled.

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

```
            TNManager.SetPlayerSave(SystemInfo.deviceUniqueIdentifier +
      "/Player.dat");
            TNManager.JoinChannel(Random.Range(0, 5));
      }
      else Debug.LogError("OnNetworkConnect(): " + message);
}

void OnNetworkJoinChannel (int channelID, bool success, string message)
{
      if (success)
      {
            TNManager.GetChannelList(OnGetChannels);
      }
      else Debug.LogError("OnNetworkJoinChannel(): " + message);
}
```

If `OnNetworkConnect`, was successful then the Game will setup a server save file, then attempt to join to a random channel from between 0 and 4, inclusive. Typically the channel would be associated with a level, or other feature in the game, but randomly assigning one will work for demonstration purposes.

After the channel is joined `TNManager.GetChannelList` is queried and passes in a callback to a function called `OnGetChannels`. Before creating that function, create a member variable called `mChannels` at the top of the class.

```
public class GameManager : MonoBehaviour
{
      List<Channel.Info> mChannels;
```

Now create the `OnGetChannels` function and store the list in the `mChannels` variable.

```
void OnGetChannels (List<Channel.Info> list)
{
      mChannels = list;
}
```

The `Channel.Info` in each list item contains the following information:

- `.id` The channels id number
- `.players` & `.limit` - How many people are currently in the channel and the maximum number of people allowed.
- `.level` - The level (or scene) this channel is associated with, if any. `TNManager.JoinChannel()` and `TNManager.LoadLevel()` can both be used to set this value.
- `.isPersistent` - Will this channel disappear if everyone disconnects?
- `.hasPassword` - If the channel has a password, it will be required before the user is able to join.
- `.data` - If the channel has any extra information, it will be stored here.

Now that we have all of this information, it's time to display it on the screen. To be able to select items and edit the custom fields, three member variables need to be defined at the top of the class.

```
public class GameManager : MonoBehaviour
{
	List<Channel.Info> mChannels;
	int mSelectedChannel = 0;
	string mChannelName = "";
	string mChannelInfo = "";
```

At the bottom of the class create a new function called DrawChannel. This function will be called from the `OnGUI` function for every channel. It will receive Channel.Info and a Rect by reference. By passing the Rect by reference it can shift the UI down line by line.

```
void DrawChannel(Channel.Info info, ref Rect rect)
{
	if (mSelectedChannel == info.id) GUI.color = Color.yellow;

	if (GUI.Button(rect, "Channel " + info.id))
		mSelectedChannel = info.id;

	rect.y += 20f;

	if (mSelectedChannel == info.id && info.data != null)
	{
		GUI.Label(rect, "Name: " + info.data.GetChild<string>("name"));
		rect.y += 20f;
		GUI.Label(rect, "Info: " + info.data.GetChild<string>("info"));
		rect.y += 20f;
	}

	GUI.color = Color.white;
}
```

Create an `OnGUI()` function and place a refresh button in it which calls `TNManager.GetChannelList()`.

```
void OnGUI()
{
	if (GUI.Button(new Rect(10, 10, 130, 20), "Refresh Channels"))
	{
		TNManager.GetChannelList(OnGetChannels);
	}
```

Next define a Rect used to place UI elements and then create a loop which loops through the channels and calls the `DrawChannel()` function.

```
	Rect rect = new Rect(10f, 50f, 300f, 20f);

	if (mChannels != null)
	{
		for (int i = 0; i < mChannels.size; ++i)
			DrawChannel(mChannels[i], ref rect);
	}
```

Changing the custom data of a channel should only be done in a channel that you are part of. If you are not in the current selected channel, disable the remaining UI.

```
	GUI.enabled = (TNManager.IsInChannel(mSelectedChannel));
```

Display two text fields with the custom channel data in them.

```
	rect.y += 40f;
	GUI.Label(rect, "Name:");
	rect.y += 20f;
	mChannelName = GUI.TextField(rect, mChannelName);

	rect.y += 20f;
	GUI.Label(rect, "Info:");
	rect.y += 20f;
	mChannelInfo = GUI.TextArea(rect, mChannelInfo);
```

And finally finish off the `OnGUI()` function by adding a save button which will send the custom channel changes over the network when pressed. And finally refresh the channel list to get the updates that were just committed.

```
	rect.y += 20f;

	if (GUI.Button(rect, "Save"))
	{
		TNManager.SetChannelData("name", mChannelName);
		TNManager.SetChannelData("info", mChannelInfo);
		TNManager.GetChannelList(OnGetChannels);
	}
}
```

Changes to the channel information are now sent out over the network, but there is nothing to receive them. In `OnEnable` and `OnDisable` add a new callback to register a function called `OnSetChannelData`.

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

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

And then define the `OnSetChannelData` function.

```
void OnSetChannelData(Channel ch, string path, DataNode node)
{
	if (TNManager.IsInChannel(ch.id))
	{
		mChannelName = ch.Get<string>("name");
		mChannelInfo = ch.Get<string>("info");
	}
}
```

When any change is made to the channel's information all of the fields are read again and the updated values are stored in their respective member variables.

Press play and you will now see all of the channels that exist and any custom data which they hold. If another client connects, press the Refresh Channels button and their channels will appear.

The complete project can be downloaded here: [No zip file yet]