TNet 3: Chat Example In-Depth

The Chat example demonstrates how to create a basic chat system for your game. In the scene there is a Chat GameObject which contains the component scripts TNObject and ExampleChat.

ExampleChat is where the magic happens and it's all starts off with the user's name. When the game starts it connects to the channel and stores the `TNManager.playerName` in `mName`. In the `OnGUI` one of the first actions it takes is to display and potentially modify `mName`.

```
mName = GUI.TextField(new Rect(cx - 70f, cy - 170f, 120f, 24f), mName);

// Change the player's name when the button gets clicked.
if (GUI.Button(new Rect(cx + 60f, cy - 170f, 80f, 24f), "Change"))
	TNManager.playerName = mName;
```

If/when the change button is pressed, `mName` gets written back to `TNManager`. In `OnEnable()` a callback is hooked up to track when user names are renamed.

```
TNManager.onRenamePlayer += OnRenamePlayer;
```

And that calls `OnRenamePlayer` on every client connected to the channel.

```
void OnRenamePlayer (Player p, string previous)
{
	AddToChat(previous + " is now known as " + p.name, Color.black);
}
```

The `AddToChat` function adds a line of text to the local copy of the chat log. eg. “Guest is now known as Guest2”. Here’s how it works:

```
struct ChatEntry
{
	public string text;
	public Color color;
}
List<ChatEntry> mChatEntries = new List<ChatEntry>();
```

A small struct, named `ChatEntry`, is created to hold chat lines and their colour.

```
void AddToChat (string text, Color color)
{
	ChatEntry ent = new ChatEntry();
	ent.text = text;
	ent.color = color;
	mChatEntries.Add(ent);
}
```

`AddToChat` creates an instance of that struct, feeds in the information and appends it to the `mChatEntries` list.

`OnGUI` then creates a GUI Window which displays the contents of the chat. `OnGUIWindow` loops through all of the chat entries and prints them to the screen.

```
for (int i = mChatEntries.size; i > 0; )
{
	ChatEntry ent = mChatEntries[--i];
	rect.y -= GUI.skin.label.CalcHeight(new GUIContent(ent.text), 382f);
	GUI.color = ent.color;
	GUI.Label(rect, ent.text, GUI.skin.label);
	if (rect.y < 0f) break;
}
```

Starting from the end of the list (most recent) to the top (oldest) labels are created and the offset between lines moves up. Before each label is created, the colour is read from the chat entry struct it gets set.

`OnGUIWindow` also contains a text field named “Chat Input” which writes its contents to a string named `mInput`.

```
if (Application.isPlaying) GUI.SetNextControlName("Chat Input");
mInput = GUI.TextField(new Rect(6f, mRect.height - 30f, 328f, 24f), mInput);

if (GUI.Button(new Rect(334f, mRect.height - 31f, 60f, 26f), "Send"))
{
	Send();
	if (Application.isPlaying) GUI.FocusControl("Chat Window");
}
```

The transmission of `mInput` over the network happens in two locations. One is by the `Send()` function in the code snippet above which triggers when the button is pressed. And the other is in OnGUI.

OnGUI’s call to Send() is a little more involved.

```
if (Event.current.type == EventType.KeyUp)
{
	var keyCode = Event.current.keyCode;
	string ctrl = GUI.GetNameOfFocusedControl();

	if (ctrl == "Name")
	{
		if (keyCode == KeyCode.Return)
		{
			// Enter key pressed on the input field for the player's
		nickname -- change the player's name.
			TNManager.playerName = mName;
			if (Application.isPlaying) GUI.FocusControl("Chat Window");
		}
	}
	else if (ctrl == "Chat Input")
	{
		if (keyCode == KeyCode.Return)
		{
			Send();
			if (Application.isPlaying) GUI.FocusControl("Chat Window");
		}
	}
	else if (keyCode == KeyCode.Return)
	{
		// Enter key pressed -- give focus to the chat input
```

```
			if (Application.isPlaying) GUI.FocusControl("Chat Input");
		}
		else if (keyCode == KeyCode.Slash)
		{
			mInput = "/";
			if (Application.isPlaying) GUI.FocusControl("Chat Input");
		}
}
```

If a keyboard key was released, the focus of the input was “Chat Input”, and the key pressed was Return then, and only then, does the Send() function get called.

```
void Send ()
{
	if (!string.IsNullOrEmpty(mInput))
	{
		mInput = mInput.Trim();

		if (mInput == "/get") PrintConfig(TNManager.serverData);
		else if (mInput.StartsWith("/get "))
	PrintConfig(mInput.Substring(5));
		else if (mInput.StartsWith("/set "))
	TNManager.SetServerData(mInput.Substring(5));
		else tno.Send("OnChat", Target.All, TNManager.playerID, mInput);

		mInput = "";
	}
}
```

If `mInput` has a value then it moves on. There are a couple of commands which can be performed with the keywords */get* and */set* which will be detailed later, but for now just look at the final “else” statement.

`mInput` is sent to the RFC function named “OnChat” on all clients connected to the network.

```
[RFC] void OnChat (int playerID, string text)
{
	// Figure out who sent the message and add their name to the text
	Player player = TNManager.GetPlayer(playerID);
	Color color = (player.id == TNManager.playerID) ? Color.green :
Color.white;
	AddToChat("[" + player.name + "]: " + text, color);
}
```

OnChat receives the player id number and the text that they wrote on every client connected to the channel. If the player id matches this client’s player id, then display the text in green vs white. Append the player name to the text and then add it to the chat. Note that the tno.Send is

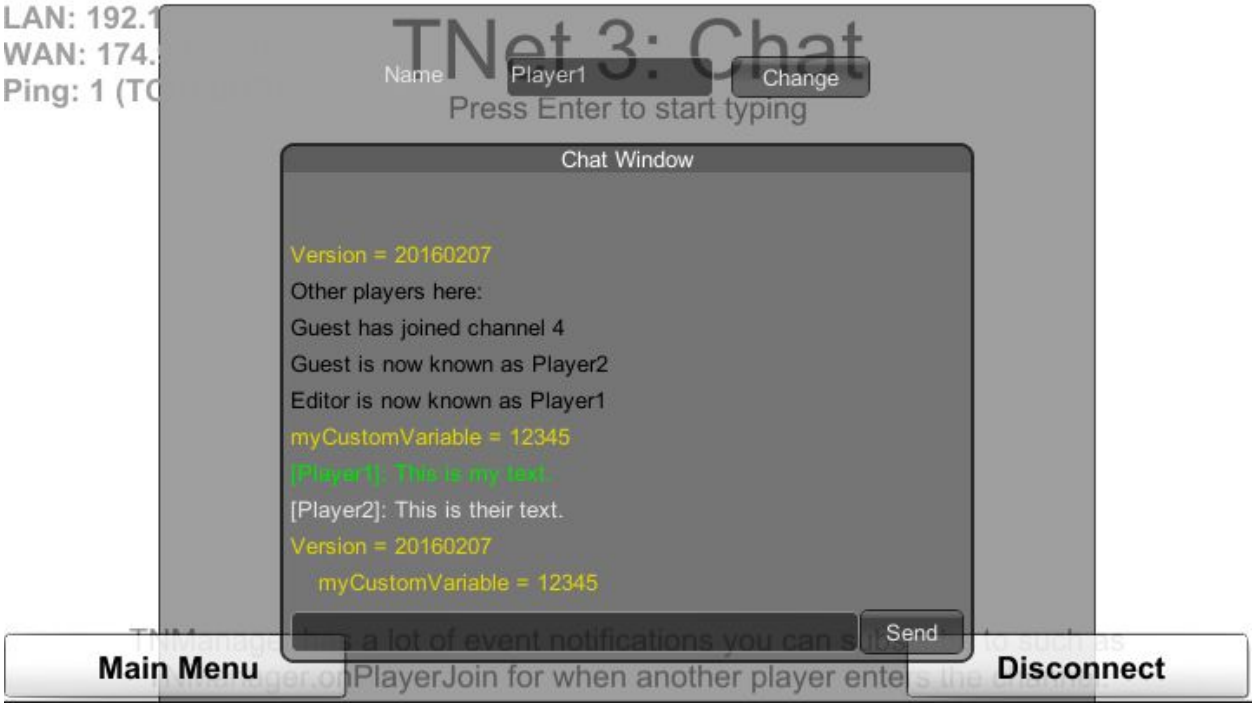
called with the target of “Target.All”, meaning it’s not saved on the server. As such, the chat is immediate -- nothing is stored on the server.

With the exception of trapping various other events (Look the OnEnable function if you’re interested) the chat is now functional.

Server Side Variables

Previously the */get* and */set* commands were mentioned. They work similar to a console command available in some games and will write variables to the server. The three commands which are supported are:

- /get - Returns a list of all set variables
- /get variableName - Returns the contents of one server side variable.
- /set path = value - Sets a variable to a specific value.



The variables are written to a file on the server named server.dat.config and are stored in plain text. Only those players that have authenticated as administrators can set server data.

When */set* is run, the string to the right of the "/set " is passed to TNManager like this:

```
TNManager.SetServerData(mInput.Substring(5));
```

There are two versions of the function, one which takes a string "path = value" and another which takes two separate strings. It's up to you, which one you would like to use.

The /set command will parse the text as a path similar to what you'd use when searching for a file, so calling `/set Path/To/Node = 123` will create a hierarchy like this when you do `Debug.Log(TNManager.serverData)`:

```
Version
    Path
        To
            Node = 123
```

```
TNManager.onSetServerData += OnSetServerOption;
```

Setting the server data will trigger a callback. If the callback is set up in the OnEnable function, then all clients will have the ability to react to the change immediately. In this case, the change is printed to the chat window.

```
void OnSetServerOption (string path, DataNode node)
{
    PrintConfig(path, node);
}
```

PrintConfig loops through all of the child nodes in the DataNode and adds them in the colour yellow to the chat.

/get and */get variableName* are very similar in functionality. */get* passes the entirety of `TNManager.serverData` to Print Config, whereas */get path* passes only the *path* portion of the node to the function GetServerData.

```
PrintConfig(path, TNManager.GetServerData(path));
```

Like the /set command, /get can natively parse paths when the server data has hierarchy to consider. For example `/get Path/To/Node` will effectively call TNManager.serverData.GetHierarchy("Path/To/Node") then print its value -- which, with the above /set example will be "123".

Note that the /set command is persistent, so the next time you start your server the previously set values will be loaded automatically as soon as you connect to that server. Since the chat example prints the serverData in OnJoinChannel, you will see the old values printed right away.