UdonSharp Nitpicks

> I hope you enjoy reading this - but keep in mind that this isn't quite finished ✌

Here's some things which I consider good practice when writing U# code, in no particular order. These suggestions are *somewhat* personal, so please let me know what you think in the comments below the article! ❤

`UdonSyncMode`

By default, the sync mode of an UdonBehaviour is chosen in the Unity inspector. Which is fine, until you accidently set it to the wrong sync mode. Yikes!

Instead, use U#'s `UdonSyncMode` attribute to make sure your `UdonBehaviour` always uses the sync mode you want.

```
using UdonSharp;
using UnityEngine;

[UdonBehaviourSyncMode(BehaviourSyncMode.Manual)]
public class Foo : UdonSharpBehaviour
{
  // ...
}
```

> This will prevent you from changing the sync mode on different GameObjects reusing the same U# script.
> But... does anyone actually do that? 🤔

`[SerializeField]`

If you make one of your variables `public`, it'll be accessible from the Unity inspector. It'll *also* make it accessible from any other `UdonBehaviour`. Most variables *aren't* accessed outside their own class, so making variables `public` makes your classes needlessly messy.

The solution? Keep those variables `private` instead, and just add `[SerializeField]` like so:

```
[SerializeField] private string foo;
```

> You can also do the opposite: To hide a public variable from the inspector, use [HideInInspector].

`[Tooltip]` and `[Header]`

U# scripts can have long and messy inspectors. Why not make them look nicer with the `[Header]` and `[Tooltip]` attributes?

```
[Header("References")]
[Tooltip("Reference to the mirror GameObject.")]
[SerializeField] private GameObject mirror;
[Tooltip("Transform to move the mirror to.")]
[SerializeField] private Transform targetTransform;
```

Cache your `LocalPlayer`

If you're doing something with `Networking.LocalPlayer` frequently, consider caching the reference in start:

```
private VRCPlayerApi localPlayer;

private void Start()
{
  localPlayer = Networking.LocalPlayer;
}
```

... and re-using that reference instead of calling `Networking.LocalPlayer` directly.

```
private void Update()
{
  Debug.Log(localPlayer.GetPosition();
}
```

`OnDeserialization()` and `[FieldChangeCallback]`

Call `OnDeserialization` as owner

When the owner of a script uses `RequestSerialization()`, all players *except* the owner will execute `OnDeserialization()`. But if you override `OnDeserialization()` in your script, you can have the owner call it manually. This is useful if you'd like the owner of the script to execute `OnDeserialization()`, to replicate the same behaviour as remote clients.

For example, here is a simple script which the master can use to increase a `score` variable. The `scoreText` is updated for everyone in `OnDeserialization`.

```
[UdonSynced] private int score;

[SerializeField] private TextMeshProUGUI scoreText;

// The script owner can call IncreaseScore anywhere to increase the score by 1.
public void IncreaseScore()
{
  score++;
  RequestSerialization();
  OnDeserialization()
}

public override void OnDeserialization()
{
  scoreText.text = score.ToString();
}
```

`[FieldChangeCallback]` instead of `OnDeserialization()`

The approach above works fine for simple scripts. However, if you script has multiple synced variables, your `OnDeserialization()` method may become quite large. Consider using `[FieldChangeCallback]` with a Property instead. It is similar to the Udon Graph's `OnVariableChanged` node. For example:

```
[UdonSynced, FieldChangeCallback(nameof(Score))] private int score;

private bool Score
{
  get => score;
  set
  {
    score = value;
    scoreText.text = score.ToString();
  }
}

[SerializeField] private TextMeshProUGUI scoreText;

public void IncreaseScore()
{
  Score++;
  RequestSerialization();
}
```

`[FieldChangeCallback]` execution order issues

> I've lost a lot of time identifying this issue. Please learn from my mistakes!

If you have multiple networked variables, watch out when using `[FieldChangeCallback]`. When a `[FieldChangeCallback]` happens, **other networked variables may not be updated yet.** In other words, if your networked variables depend on each other, you may want use `OnDeserialization()` instead.

Notably, the owner of the script will not suffer from this issue. If you're seeing no issues as an object's owner, but outdated variables on remote clients, you may be running into issues with `[FieldChangeCallback]`'s execution order!

What else?

There's more I'd like to write about... Let me know if any of these sound interesting!

- Avoid Update(), use delayed events instead.
- Did you know that FixedUpdate() depends on your monitor's refresh rate? Yeah.
- Use synced variables, avoid networked events, if you can.
- FieldChangeCallback is great.
- Name your classes and methods properly.

Revision #4
Created 25 May 2022 22:59:15 by
Updated 20 September 2022 21:29:30 by Admin