Waitress

(Anyone can comment)

Please check online documentation for most updated version:
https://docs.google.com/document/d/1eGd3DuOe4Bs70O_AKImTLOUisa0GbBIIKVDYpvnvL_o

Index

What is Waitress?

Waitress provides few helper classes which extend functionality of Unity's original waiting classes such as WaitForSeconds and WaitForEndOfFrame.

IMPORTANT FEATURES: All waitress classes can be **used in regular (non-coroutine) methods** and they are **cancelable** UNLIKE Unity's original waiting classes.

The extended classes are

- **Timer:** Waits for seconds. It supports **UnscaledTime** too.
- **Framer:** Waits for end of frame or next frame.
- **FixedFramer:** Waits for next fixed frame.

Waitress also provides a **"Conditioner"** class which can wait for a specific boolean condition you define.

Some of the extended waiting classes(Timer and Conditioner) also support **"repeating"** functionality.

How to use

(iOS only) After you import the asset package, please set your project's API compatibility level to full .NET. Don't use the subset level.

Timer class

Timer class lets you wait for a duration and it replaces the Unity's WaitForSeconds class by extending it.

First of all, you have to create an instance of Timer class before starting wait process.
Then you can call Wait method to start the waiting process. You can also provide a "done" handler so you get notified when the waiting is done.

C# lambda notation can help you define done event handler method instead of defining a separate "named" method.

```
new Timer().Wait(3, Done: (Sender, E) =>
{
	// Your done handler code here
});
```

Canceling the timer

You can store the Timer instance into a variable if you wish to cancel it anytime.

```
MyTimer = new Timer();
MyTimer.Wait(3, Done: (Sender, E) =>
{
   if (E.Canceled)
   {
      // Your cancel handler code here
   }
   else
   {
      // Your done handler code here
   }
});
```

Call Cancel method when you wish to cancel the timer:

```
MyTimer.Cancel();
```

Please see the example scripts in the waitress folder.

Waiting for unscaled time

It’s very common for unity game developers to set the Unity’s Time.timeScale property to set their game speed or even to pause the game by setting it to zero. When that happens, the developer may also want the game to wait for a while in real time for UI animations or similar stuff. Unfortunately, Unity’s WaitForSeconds class doesn’t provide a solution to this problem. Luckly, you can use Waitress’ Timer class to wait for seconds in real time (unscaled time). You just need to set the UnscaledTime parameter in Wait method call to true.

```
new Timer().Wait(3, UnscaledTime: true, Done: (Sender, E) =>
{
      // Your done handler code here
});
```

Framer and FixedFramer classes

Framer class lets you wait for the next frame. Unity's way of doing this is yielding a null value in the coroutines but for this reason, you have to use it in a coroutine. You can use Framer in any method even if it's not a coroutine.

```
new Framer().Wait(Done: (Sender, E) =>
{
	// Your done handler code here
});
```

You can set the EndOfCurrentFrame parameter to true for waiting for the end of the frame.

```
new Framer().Wait(true, (Sender, E) =>
{
	// Your done handler code here
});
```

FixedFramer class lets you wait for the next fixed frame just like Unity's WaitForFixedFrame class. Unlike WaitForFixedFrame , you can use FixedFramer in non-coroutine methods because it provides a done event handler.

```
new FixedFramer().Wait((Sender, E) =>
{
	// Your done handler code here
});
```

Conditioner class

Conditioner class lets you wait for a boolean condition which you define.

In your script class, you can define a boolean condition variable:

```
	bool SampleCondition = false;
```

This Conditioner call will wait until SampleCondition is true.

```
new Conditioner().Wait(() => SampleCondition, Done: (Sender, E) =>
{
	// Your done handler code here
});
```

Using Waitress classes in coroutines

If you want to block execution of coroutines while you were waiting for something using waitress classes, you can still yield the coroutine just like Unity's wait classes.

```
IEnumerator CoroutineMethod()
{
	print("Coroutine paused");

	yield return new Conditioner().Wait(() => SampleCondition);
	// Coroutine is paused until conditioner is done or canceled

	yield return new Timer().Wait(Duration);
	// Coroutine is paused until timer is done or canceled

	print("Coroutine resumed");
 }
```

How does it work?

Waitress classes wraps the Unity's waiting classes into internal coroutines. When you want to wait for something, waitress classes starts that wrapping coroutine so you don't have to implement custom coroutines. When you want to cancel the waiting process, waiter classes just stop these internal coroutines.

Since waiting coroutines are wrapped in waitress classes, you can use them in regular(non-coroutine methods) unlike Unity's original waiting classes. Original waiting classes HAVE to be used in coroutines only.
Waitress classes notifies you via the "done" event handler you provided when the waiting process is done.

Troubleshooting

1. PostSharp Serialization error on iOS runtime

You may encounter this runtime error on iOS platform if your project's API compatibility level is .NET subset. Please use full .NET profile as your compatibility level. Also note that this rule only applies to iOS platform.