Wait! - Overview

Wait! is a Unity coroutine helper that lets you use declarative syntax to write coroutines without the fuss of IEnumerator or yield syntax for simple timing tasks.

```
using SL.Wait;

Wait.Seconds(10,
  () => {
    Debug.Log("Waited 10 seconds");
  }
).Start();
```

Features

- Wait for Frames, Seconds, SecondsByFrame, or until a specified condition is satisfied.
- Repeat your wait any number of times or indefinitely.
- Easily chain waits together. (eg. wait until 5 seconds after a condition is complete)
- Group waits to stop, pause, or resume them all together.
- Specify an action that runs when ever you start the wait.
- Run coroutines in any script, not just MonoBehaviors (thanks to **More Effective Coroutines**).

Installation

It's free! so just grab it from the **Unity Asset Store** and add "using SL.Wait;" to any script.

Dependancies

More Effective Coroutines - FREE

Wait! is effectively a wrapper that makes More Effective Coroutines easier to use. As such, it also needs to be installed for Wait! to work. Any version should work fine but we use the Free

version linked here.

wait.Start()

This method needs to be called to begin the Wait. If it is not called then nothing will happen. It is most often used as a chained function call at the end of the Wait setup.

```
using SL.Wait;

Wait.Seconds(10,
  () => {
    Debug.Log("Waited 10 seconds");
  }
).Start();
```

wait.Start() - delayed

You can save the Wait object and start it at a later time instead of needing to start it immediately. This is particularly handy for chained or grouped waits.

```
using SL.Wait;

Wait timing = Wait.Seconds(10,
  () => {
    Debug.Log("Waited 10 seconds");
  }
);

// Do some other stuff, start it from some other script, or how ever you'd like.

timing.Start();
```

wait.Stop()

You can stop the wait early by calling Stop on the instance.

```
using SL.Wait;

Wait timing = Wait.Seconds(10,
  () => {
    Debug.Log("Waited 10 seconds");
  }
).Start();
```

```
// Somewhere else...

timing.Stop();
```

wait.Pause()

You can pause the wait mid-run by calling Pause on the instance.

```
using SL.Wait;

Wait timing = Wait.Seconds(10,
  () => {
    Debug.Log("Waited 10 seconds");
  }
).Start();

// Somewhere else...

timing.Pause();
```

wait.Resume()

You can resume checking the wait by calling Resume on the instance.

```
using SL.Wait;

Wait timing = Wait.Seconds(10,
  () => {
    Debug.Log("Waited 10 seconds");
  }
).Start();

// Somewhere else after .Pause() has been called...

timing.Resume();
```

Static Constructors

There are four static constructors that help clean up the syntax. They function the same as the instance constructors, they just look nicer. The equivalent instance syntax is below.

```
using SL.Wait;

Wait timing = Wait.Frames(10, () => { ... });
Wait timing = Wait.Seconds(10, () => { ... });
Wait timing = Wait.SecondsWhilePaused(10, () => { ... });
Wait timing = Wait.For(() => { ... }, () => { ... });
```

Instances

If you favor instance syntax that's fine too. The equivalent static syntax is above.

```
using SL.Wait;

Wait timing = new Wait().ForFrames(10).Then(() => { ... });
Wait timing = new Wait().ForSeconds(10).Then(() => { ... });
Wait timing = new Wait().ForSecondsByFrame(10).Then(() => { ... });
Wait timing = new Wait().Until(() => { ... }).Then(() => { ... });
```

Chained Waits (Don't do this)

The syntax allows you to chain multiple wait types but it will only use the last one specified. If you want to run one wait after another see the "Chain" section.

```
using SL.Wait;

// ForSeconds() will overwrite the wait type and be used instead of Frames.
// Then() will overwrite the constructor function.

Wait timing = Wait.Frames(300, () => {
  Debug.Log("This won't run, it's been overwritten by the next Then() call.");
}).ForSeconds(10).Then(() => {
  Debug.Log("Waits 10 seconds, not 300 frames.");
});
```

Wait.Frames()

Waits the specified number of frames before executing it's function.

```
using SL.Wait;

Wait.Frames(300,
```

```
  () => {
    Debug.Log("Waited 300 frames");
  }
).Start();

// or

Wait.Frames(300).Then(
  () => {
    Debug.Log("Waited 300 frames");
  }
).Start();
```

Wait.Seconds()

Waits the specified number of seconds before executing it's function. Uses internal timing methods which stop when the game is paused.

```
using SL.Wait;

Wait.Seconds(10,
  () => {
    Debug.Log("Waited 10 seconds");
  }
).Start();

// or

Wait.Seconds(10).Then(
  () => {
    Debug.Log("Waited 10 seconds");
  }
).Start();
```

Wait.SecondsWhilePaused()

Waits the specified number of seconds before executing it's function. This checks every frame to see if the time is elapsed so it will run even while the game is paused.

```
using SL.Wait;

Wait.SecondsWhilePaused(10,
  () => {
    Debug.Log("Waited 10 seconds");
  }
).Start();
```

```
// or

Wait.SecondsWhilePaused(10).Then(
  () => {
    Debug.Log("Waited 10 seconds");
  }
).Start();
```

Wait.For()

Waits until the given condition is satisfied before running the given function.

This will check the condition every frame, it's up to you to keep the check reasonable and performant.

```
using SL.Wait;

Wait.For(
  () => {
    return someVariable == anotherVariable;
  },
  () => {
    Debug.Log("Waited until the condition returned true.");
  }
).Start();

// or

Wait.For(
  () => {
    return someVariable == anotherVariable;
  }
).Then(
  () => {
    Debug.Log("Waited until the condition returned true.");
  }
).Start();
```

wait.Until()

The instance version of adding a condition to any wait.

This will check the condition every frame, it's up to you to keep the check reasonable and performant.

```
using SL.Wait;

Wait timing = new Wait().Until(
  () => {
    return someVariable == anotherVariable;
  }
).Then(
  () => {
    Debug.Log("Waited until the condition returned true.");
  }
).Start();
```

wait.Repeat()

Repeats the wait the specified number of times. If you enter 0 it will repeat indefinitely (see below).

```
using SL.Wait;

Wait.Seconds(2,
  () => {
    // This will output 5 times, every 2 seconds.
    Debug.Log("Waited 2 seconds");
  }
).Repeat(5).Start();
```

Repeat Indefinitely

Passing in 0 to Repeat() will repeat indefinitely. It is up to you to call timing.Stop() at some point, either in the wait function or anywhere else.

```
using SL.Wait;

Wait timing = Wait.Seconds(2);

timing.Then(() => {
  if (someCondition) {
    timing.Stop();
  }
```

```
  // This will output every 2 seconds until .Stop() is called.
  Debug.Log("Waited 2 seconds");
})

// Repeat indefinitely
timing.Repeat(0)

timing.Start();
```

wait.Chain()

Runs a second Wait object after the current one finishes. Unlike other methods you may call .Chain() as many times as you want on the same Wait object. (see wait.Chain() - Sequence below)

```
using SL.Wait;

Wait next_wait = Wait.Frames(600,
  () => {
    Debug.Log("Waited 600 frames after waiting 3 seconds");
  }
);

Wait chain_wait = Wait.Seconds(3,
  () => {
    Debug.Log("Waited 3 seconds, chaining to next_wait");
  }
).Chain(next_wait).Start();
```

Chain Sequence

Runs multiple Wait objects in sequence. Unlike other methods you may call .Chain() as many times as you want on the same Wait object without overwriting. The chained waits will run in sequence in the order in which you add them with .Chain().

```
using SL.Wait;

Wait timing_1 = Wait.Frames(600,
  () => {
    Debug.Log("Waited 3 seconds, then 600 frames - chaining to next_wait");
  }
);

Wait timing_2 = Wait.Seconds(5,
```

```
  () => {
    Debug.Log("Waited 3 seconds, then 600 frames, then 5 seconds.");
  }
);

Wait chain_wait = Wait.Seconds(3,
  () => {
    Debug.Log("Waited 3 seconds - chaining to next_wait");
  }
).Chain(timing_1).Chain(timing_2).Start();
```

Chain Function

This will run the chained function in the same frame as the .Then() function but after the .Then() function is complete.

```
using SL.Wait;

Wait chain_wait = Wait.Seconds(3,
  () => {
    Debug.Log("Waited 3 seconds, chaining to next_wait");
  }
).Chain(
  () => {
    Debug.Log("Runs after the main Wait function.");
  }
).Start();
```

wait.Tag()

By default a Wait object is tagged with a random guid. You can change the tag so that you may reference it without a held instance through the WaitManager. Only Waits that are actively waiting can be accessed in the WaitManager.

```
using SL.Wait;

string tagName = "myTiming";

Wait.For(
  () => characterIsInPosition,
  () => { Bark("Hi there!"); }
).Tag(tagName).Start();

// Stop from anywhere without passing the instance around.
```

```
WaitManager.StopTag(tagName);
```

wait.Group()

By default a Wait object is not in any group. You can add it to a group by passing in any string. You can then Stop, Pause, and Resume all grouped objects through the WaitManager all at once.

```
using SL.Wait;

string groupName = "characterGroup";

Wait.For(
  () => CharacterIsInPosition,
  () => { Bark("Hi there!"); }
).Group(groupName).Start();

Wait.Seconds(
  10,
  () => WarpCharacter
).Group(groupName).Start();

// Stop from anywhere without passing the instances around.
WaitManager.StopGroup(groupName);
```

Wait Manager

The wait manager is a static class that lets you access currently waiting Wait objects more easily. You can access them via tag, group, or grab every single one to Stop, Pause, or Resume them.

WaitManager.Stop()

Stop all active Wait coroutines.

```
using SL.Wait;

WaitManager.Stop();
```

WaitManager.Pause()

Pause all active Wait coroutines.

```
using SL.Wait;

WaitManager.Pause();
```

WaitManager.Resume()

Resume all active Wait coroutines.

```
using SL.Wait;

WaitManager.Resume();
```

WaitManager.StopTag()

Stop the Wait coroutine with the specified tag.

```
using SL.Wait;

string tagName = "myTiming";

WaitManager.StopTag(tagName);
```

WaitManager.PauseTag()

Pause the Wait coroutine with the specified tag.

```
using SL.Wait;

string tagName = "myTiming";

WaitManager.PauseTag(tagName);
```

WaitManager.ResumeTag()

Resume the Wait coroutine with the specified tag.

```
using SL.Wait;

string tagName = "myTiming";

WaitManager.ResumeTag(tagName);
```

WaitManager.StopGroup()

Stop all active Wait coroutines in the given group.

```
using SL.Wait;

string groupName = "timingGroup";

WaitManager.StopGroup(groupName);
```

WaitManager.PauseGroup()

Pause all active Wait coroutines in the given group.

```
using SL.Wait;

string groupName = "timingGroup";

WaitManager.PauseGroup(groupName);
```

WaitManager.ResumeGroup()

Resume all active Wait coroutines in the given group.

```
using SL.Wait;

string groupName = "timingGroup";

WaitManager.ResumeGroup(groupName);
```

wait.OnStart()

Runs the specified function whenever .Start() is called. This is mostly useful for Wait objects that you define in one place and start somewhere else.

```
using SL.Wait;

Wait timing = Wait.Seconds(2,
  () => {
    // This will output 5 times, every 2 seconds.
    Debug.Log("Waited 2 seconds");
  }
).OnStart(
  () => {
    // This will output only once, when Start() is called.
    Debug.Log("Coroutine Started!");
  }
).Repeat(5);

// Somewhere else
timing.Start();
```