# SUBROUTINES

**MugHeadStudios**

## OVERVIEW

*Note that this documentation only details what is necessary for interfacing with the package. If you have any further questions, please contact me, or check the* ***.cs*** *files themselves which have been heavily commented.*

**What are sub routines and why would I need them?**

To put it simply once created, a sub routine will run in the background performing an action until a condition is met. Fleshing this idea out further we can build many complex tasks that can run concurrently or within a chain of events, while adding features that makes awkward development much easier to write and code that is easier to read at a glance.

**How does it work?**

Many varieties of tasks can be defined by 2 simple things: a condition and an action. This package provides all of the background work by testing and running these tasks according to the current frame rate while giving a simple interface to create them. The package comes with the static class named SubRoutines which contains a variety of common understandable methods to easily interface with such as "Wait", "While" and "RunFor", which act as a great foundation for any type of time / condition based task.

**How does it run at the game's frame rate?**

The first time a SubRoutine is created it will instantiate a single supporting GameObject (SubRoutinesGameObject) which holds a list of currently running SubRoutines and runs each one during the standard Unity Update method frame. This way all SubRoutines are being checked and ran in sync with the game's frame rate which means we can be time specific within our conditions and actions using Time.deltaTime or other time sensitive properties.

## Table Of Contents

*Please note if you see any mistakes please contact me using one of the options at the end of this document.*

# DOCUMENTATION

## 1. Prerequisites

There are none, but it is recommended you include the MugHeadStudios namespace for quick access to the SubRoutines class.

```
using MugHeadStudios;
```

## 2. Available Methods (Using the SubRoutines Static Class)

The SubRoutines class is both public and static, it simply holds a collection of methods to easily start new SubRoutines or chain them together. All available methods are explained below:

**2a. Always(Action action)**
Start invoking an action every frame.

```
SubRoutines.Always(() => {
    // Execute code every frame
});
```

**2b. Wait(float time, Action action)**
Wait for specified time and then invoke an action.

```
SubRoutines.Wait(1f, () => {
    // Execute code after specified time
});
```

**2c. RunFor(float time, Action<float> action, Action callback = null)**
Invoke an action for a specified time passing percentage progress (t), then invoke the callback if passed.

```
SubRoutines.RunFor(float time, t => {
    // Execute code until time elapsed. t is the current progression between 0 - 1
}, () => {
    // Execute callback code once time elapsed
});
```

**2d. When(Func<bool> condition, Action action)**
Wait until condition becomes true and then invoke the action.

```
SubRoutines.When(() => condition == true, () => {
    // Execute code once condition becomes true
});
```

**2e. While(Func<bool> condition, Action action, Action callback)**
The foundation of all other SubRoutines, it tests against the given condition every frame and runs the action while true, once the condition becomes false it runs the callback and finally disposes of the SubRoutine.

```
SubRoutines.While(() => condition == true, () => {
    // Execute code while condition is true
}, () => {
    // Execute callback code once condition is false
});
```

**2f. RunNextFrame(Action action)**
Waits one frame and then invokes action.

```
SubRoutines.RunNextFrame(() => {
    // Executes code on the next frame
});
```

**2g. RunAfterXFrames(int numFrames, Action action)**
Waits numFrames frames and then invokes action.

```
SubRoutines.RunAfterXFrames(60, () => {
    // Executes code after X frames
});
```

**2h. RunForXFrames(int numFrames, Action<int> action, Action callback)**
Invoke an action for a specified number of frames (numFrames), passing the current number of frames elapsed into action (f), then invokes callback on the last frame.

```
SubRoutines.RunForXFrames(60, f => {
    // Executes code every frame for X frames. f is the number of frames passed
}, () => {
    // Execute callback code once all frames have elapsed
});
```

**2i. Do(Action action)**
Invoke an action immediately. Note that this is only ever useful within a chain of events, example below.

```
SubRoutines.Do(() => {
    // Executes code immediately
});
```

**2j. Repeat(int repeat, float interval, Action<int> action, Action callback)**
Repeat invoking an action a specified number of times (repeat), passing the current number of repeats into action (r), then invokes callback after the last repeat.

```
SubRoutines.Repeat(10, 0.3f, r => {

    // Execute code passing "r" as current number of repeats

}, () => {

    // Execute callback after last repeat

});
```

**2k. Chain(params SubRoutine[])**
Invoke a chain of SubRoutines, waiting for one to finish before moving on to the next. Below is showing an example of the Do and Wait SubRoutines being used but any type will work. Explained further in **section 5**.

```
SubRoutines.Chain(
    SubRoutines.Do(() => /* Executes immediately */),
    SubRoutines.Wait(1f, () => /* 1 second has passed */),
    SubRoutines.Wait(1f, () => /* 2 seconds have passed */),
    SubRoutines.Wait(1f, () => /* 3 seconds have passed */)
    // Continue to pass any number of SubRoutine objects
);
```

**2l. GetInstance()**
Return the SubRoutinesGameObject instance.

```
SubRoutines.GetInstance()
```

**2m. RemoveSubRoutine(SubRoutine subRoutine)**
Remove a SubRoutine.

```
SubRoutines.RemoveSubRoutine()
```

**2n. RemoveAllSubRoutines()**
Remove all SubRoutines.

```
SubRoutines.RemoveAllSubRoutines()
```

## 3. SubRoutine Class

The core sub routine object. SubRoutine objects should not be created manually but are created indirectly through the SubRoutines static class methods.

### 3a. Properties

**Condition**: The condition that will be tested each frame.

```
Public Func<bool> condition {get; set;}
```

**Action**: The action that will be invoked each frame while condition is true.

```
Public Action action {get; set;}
```

**Callback**: The callback to invoke once condition becomes false.

```
Public Action callback {get; set;}
```

**Paused**: If true then we will not check the condition, or invoke action or callback during the Run method.

```
Public bool paused {get; protected set;}
```

**Chained routine**: If not null then chainedRoutine will be un-paused immediately after this SubRoutine has completed.

```
Public SubRoutine chainedRoutine {get; protected set;}
```

### 3b. Methods

**Constructor**: Sets corresponding properties and immediately adds itself to the SubRoutinesGameObejct list of running routines.

```
public SubRoutine(Func<bool> _condition, Action _action, Action _callback = null)
```

**Pause**: Sets the paused property. Note that SubRoutines are automatically paused when added to a chain, and automatically un-paused when the last routine within the chain has finished. But this can be overridden with this method if necessary.

```
public SubRoutine Pause(bool pause)
```

**ChainedRoutine**: Sets the past SubRoutine as the next in the chain by setting the chainedRoutine property. The passed SubRoutine will be paused automatically and un-paused once this SubRoutine is complete.

```
public SubRoutine ChainedRoutine(SubRoutine subRoutine)
```

**Run**: Should never be called manually, the Run method is used each frame by the current instance of SubRoutinesGameObject. If this routine is paused then we bail out immediately before continuing, otherwise we invoke the condition. If the condition returns true then we invoke the current action property and return true, if the condition returns false then we invoke the callback, un-pause chainedRoutine if it is set, then finally return false. By returning false the SubRoutinesGameObject knows this routine has completed and disposes of it.

```
public void Run()
```

## 4. SubRoutinesGameObject Class (Explained)

A MonoBehaviour GameObject that holds all current SubRoutines running each one, each frame. We use a GameObject to do this so we can keep in perfect sync with Unity’s standard Update method, in this way Time.deltaTime and other time related properties will always be correct during a SubRoutine action or callback.

Note that the SubRoutinesGameObject class exists with the MugHeadStudios.Internal namespace and should not be interacted with publicly, in this case here are some important notes to bear in mind about this object:

The SubRoutinesGameObject…

- should only be used internally
- keeps a list of all currently running SubRoutines named runningSubRoutines
- works as a singleton, and should only ever be interacted with using its instance property
- automatically disposes of SubRoutine objects once their condition returns false

## 5. Chains

A chain is useful when you need an array of SubRoutines to run in order, waiting for each one to finish before moving to the next. There are two ways to do this:

**1.** Using the SubRoutines.Chain static method (preferred).

```
SubRoutines.Chain(
    // Pass any number of SubRoutine objects
);
```

**2.** Using the SubRoutines.ChainedRoutine method.

```
SubRoutines.Wait(1f)
.ChainedRoutine(/* Pass a single SubRoutine object */)
.ChainedRoutine(/* Pass another SubRoutine object... */);
```

On chaining a SubRoutine it will automatically become paused and only resume once the routine in the chain before has completed.

## 6. Conclusion

Thank you for using MugHeadStudios SubRoutines, if you have any suggestions, notice any mistakes within the documentation, or bugs in the package, please let me know as it is very much appreciated.

## 7. Contact

Craig Dennis (Zephni)
**MugHeadStudios**

**Contact me by email:** zephni@hotmail.co.uk

**Discord:** https://discord.gg/S7Y3RyUpGh