# ADVANCED COROUTINES

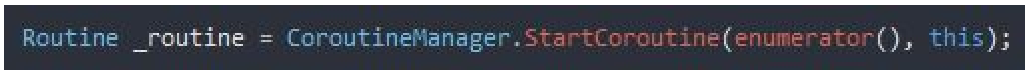
user guide

## 1. Coroutines type in `AdvancedCoroutines`

### 1.1) Coroutines, attached to the object

Coroutines attached to the object works on the principle of standard Unity coroutines .

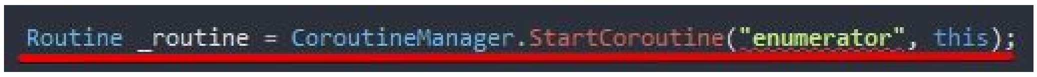
#### Launching coroutine:



```
Routine _routine = CoroutineManager.StartCoroutine(enumerator(), this);
```

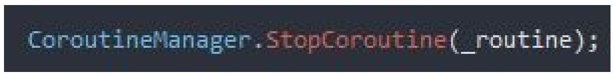
- **`Routine`** - returning object, simular to `Unity Coroutine` class
- **`enumerator()`** - `IEnumerator-like` object or method returning `IEnumerator`
- **`this`** - link to `MonoBehaviour` to which coroutine will be attached

**Note**: coroutine can't be started by passing a method name as a string like in standard `Unity` coroutines



```
Routine _routine = CoroutineManager.StartCoroutine("enumerator", this);
```

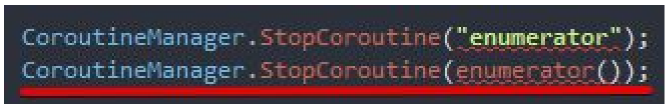
#### Stopping coroutine:



```
CoroutineManager.StopCoroutine(_routine);
```

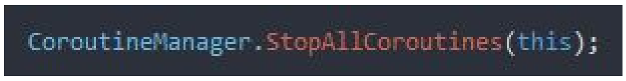
- **`_routine`** - `Routine`-like object received by `CoroutineManager.StartCoroutine`

**Note:** coroutine can't be stopped by passing a method name as a string or object/method IEnumerator to StopCoroutine method like in standard `Unity` coroutines



```
CoroutineManager.StopCoroutine("enumerator");
CoroutineManager.StopCoroutine(enumerator());
```

Stop of all coroutines attached to `MonoBehaviour`:



```
CoroutineManager.StopAllCoroutines(this);
```

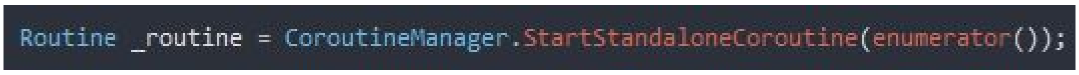
- **`this`** - link to `MonoBehaviour`, which coroutine attached to

**Note:**
Destroying a **`MonoBehaviour`** object (e.x. by **`Destroy(gameObject)`**) stops all coroutines attached to it

## 1.2) Standalone Coroutines

Coroutines unattached to any oblects continue their work even between scenes

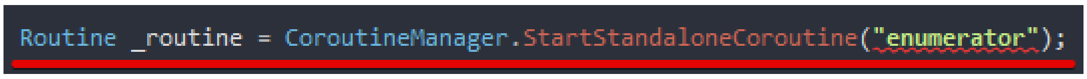
### Launching standalone coroutine:



```
Routine _routine = CoroutineManager.StartStandaloneCoroutine(enumerator());
```

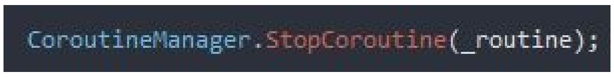
- **`Routine`** - returning object, simular to `Unity Coroutine` class
- **`enumerator()`** - IEnumerator-like object or method returning `IEnumerator`

**Note**: coroutine can't be started by passing a method name as a string like in standard `Unity` coroutines



```
Routine _routine = CoroutineManager.StartStandaloneCoroutine("enumerator");
```

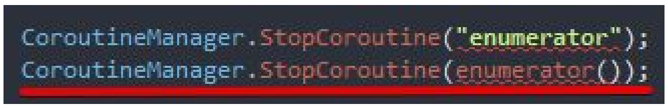
### Stopping standalone coroutine



```
CoroutineManager.StopCoroutine(_routine);
```

**`_routine`** - `Routine`-like object, received by `CoroutineManager.StartStandaloneCoroutine`

**Note:** coroutine can't be stopped by passing a method name as a string or object/method IEnumerator to StopCoroutine method like in standard `Unity` coroutines



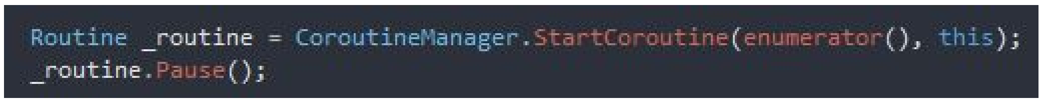
```
CoroutineManager.StopCoroutine("enumerator");
CoroutineManager.StopCoroutine(enumerator());
```

**Note:** standalone-coroutines can be stopped only in a manual way

## 2. Routine class

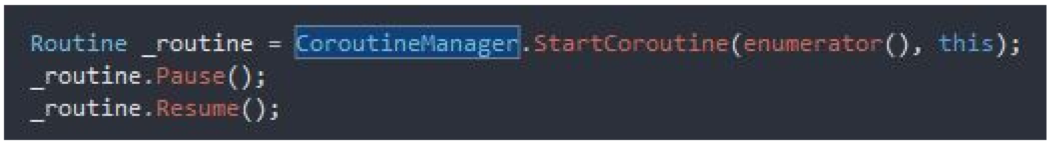
Methods **CoroutineManager.StartCoroutine()** or **CoroutineManager.StartStandaloneCoroutine()** return **Routine** object that allows to control coroutine. It is required to stop coroutines manually and this is unique identifier of coroutine.

Coroutine can be paused by calling Routine’s method **Pause()**



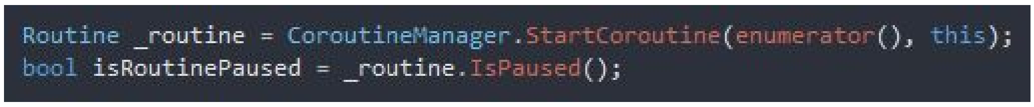
```
Routine _routine = CoroutineManager.StartCoroutine(enumerator(), this);
_routine.Pause();
```

To resume coroutine you need to call Routine’s method **Resume()**



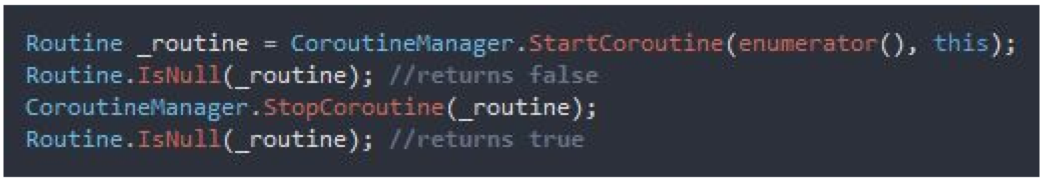
```
Routine _routine = CoroutineManager.StartCoroutine(enumerator(), this);
_routine.Pause();
_routine.Resume();
```

To check whether coroutine is paused you need to call Routine’s method **IsPaused()**



```
Routine _routine = CoroutineManager.StartCoroutine(enumerator(), this);
bool isRoutinePaused = _routine.IsPaused();
```

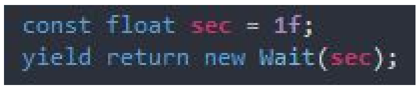
To check whether coroutine is destroyed you need to call  static method **IsNull(Routine routine)** of `Routine` class



```
Routine _routine = CoroutineManager.StartCoroutine(enumerator(), this);
Routine.IsNull(_routine); //returns false
CoroutineManager.StopCoroutine(_routine);
Routine.IsNull(_routine); //returns true
```

## 3. Working with IEnumerator

### 3.1) `Wait(float seconds)`



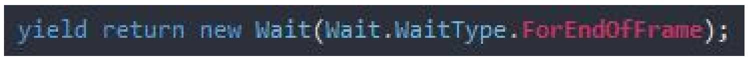
```
const float sec = 1f;
yield return new Wait(sec);
```

Suspends the coroutine execution for **`sec`** seconds using scaled time

**Note**: you need to use **`Wait(sec)`** instead of **`yield return new WaitForSeconds(sec)`**.

Using WaitForSeconds will throw an exception

### 3.2) `Wait(Wait.WaitType)`

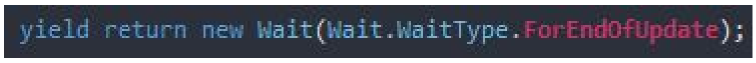


```
yield return new Wait(Wait.WaitType.ForEndOfFrame);
```

Waits until the end of the frame after all cameras and GUI is rendered, just before displaying the frame on screen

**Note**: you need to use **`Wait(Wait.WaitType.ForEndOfFrame)`** instead of **`yield return new WaitForEndOfFrame(sec)`**.

Using WaitForEndOfFrame will throw an exception



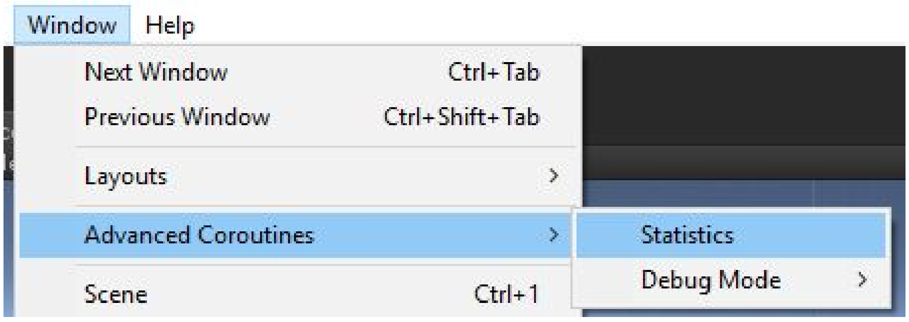
```
yield return new Wait(Wait.WaitType.ForEndOfUpdate);
```

Waits until the end of the Update method

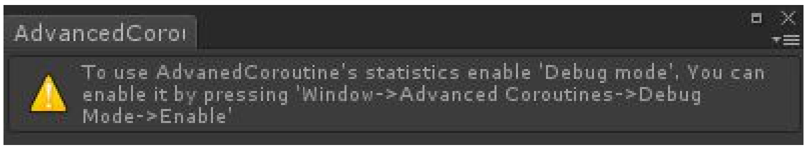
**Note**: coroutine resumes execution in `LateUpdate()`

## 4. Statistics window

To open a statistics window, you need to go to `Window->Advanced Coroutines->Statistics`

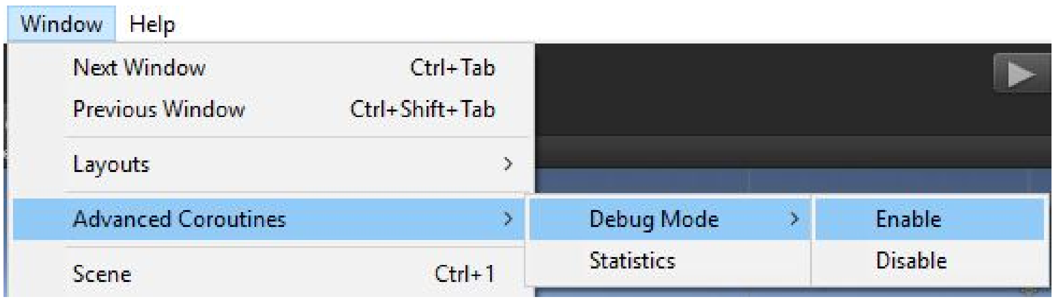


If statistics window displayed with following message,

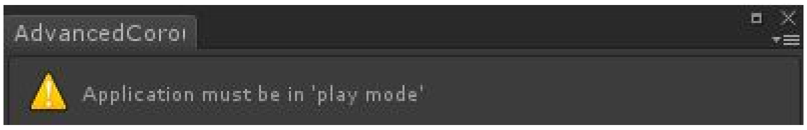


To use AdvanedCoroutine's statistics enable 'Debug mode'. You can enable it by pressing 'Window->Advanced Coroutines->Debug Mode->Enable'

go to `Window->Advanced Coroutines->Debug Mode` and press `Enable`. This action enables preprocessor directive, which hide gathering of statistics

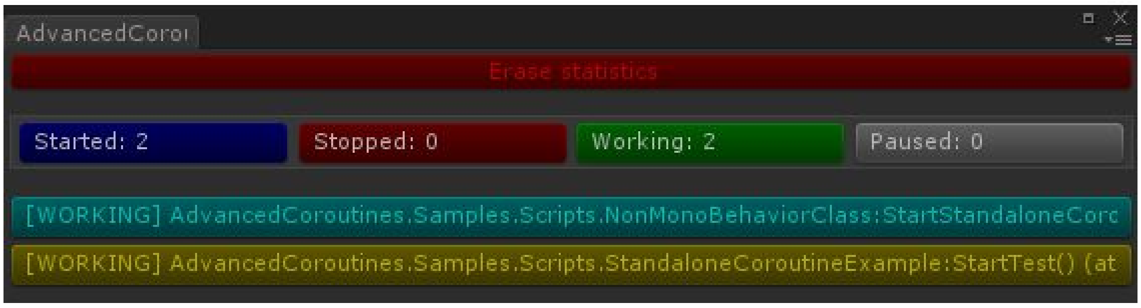


Statistics are collected only in “Play mode”, in “Editor mode” you’ll see the following message



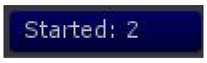
Application must be in 'play mode'

Operating statistics window is as follows:

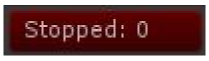


```
[WORKING] AdvancedCoroutines.Samples.Scripts.NonMonoBehaviorClass:StartStandaloneCorc
[WORKING] AdvancedCoroutines.Samples.Scripts.StandaloneCoroutineExample:StartTest() (at
```

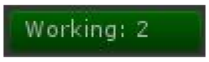
Erases collected statistics



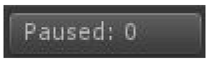
the total number of running coroutines



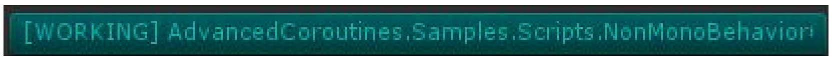
the total number of stopped coroutines



the total number of working coroutines

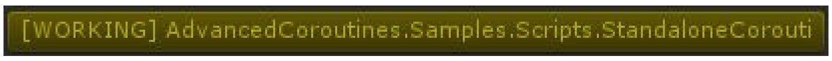


the total number of paused coroutines



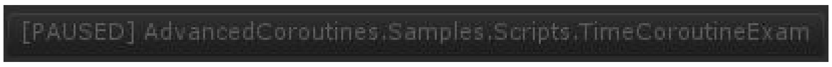
```
[WORKING] AdvancedCoroutines.Samples.Scripts.NonMonoBehavior
```

active standalone coroutine



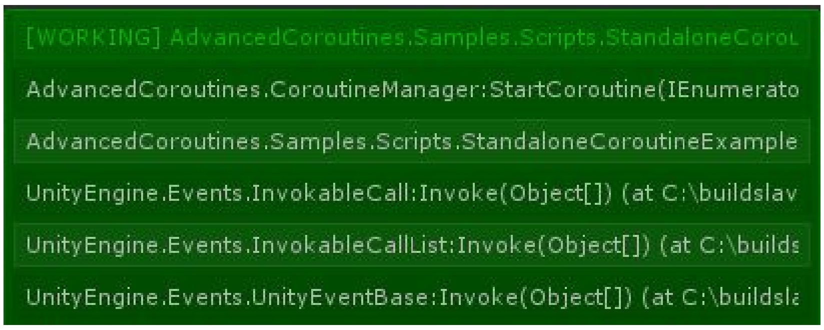
```
[WORKING] AdvancedCoroutines.Samples.Scripts.StandaloneCorouti
```

active coroutine



```
[PAUSED] AdvancedCoroutines.Samples.Scripts.TimeCoroutineExam
```

paused coroutine



```
[WORKING] AdvancedCoroutines.Samples.Scripts.StandaloneCorou
AdvancedCoroutines.CoroutineManager:StartCoroutine(IEnumerato
AdvancedCoroutines.Samples.Scripts.StandaloneCoroutineExample
UnityEngine.Events.InvokableCall:Invoke(Object[]) (at C:\buildslav
UnityEngine.Events.InvokableCallList:Invoke(Object[]) (at C:\builds
UnityEngine.Events.UnityEventBase:Invoke(Object[]) (at C:\buildsla
```

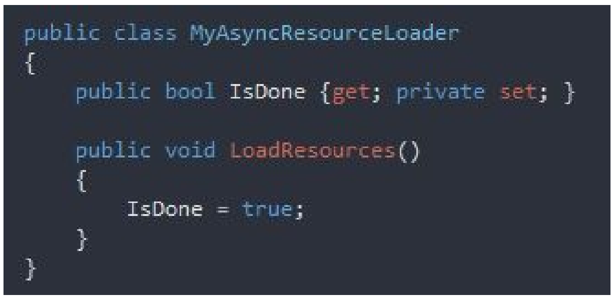
expanded active coroutine. It opens by clicking on working coroutine. It displayed the coroutine call stack.

## 5. Expansion of functionality

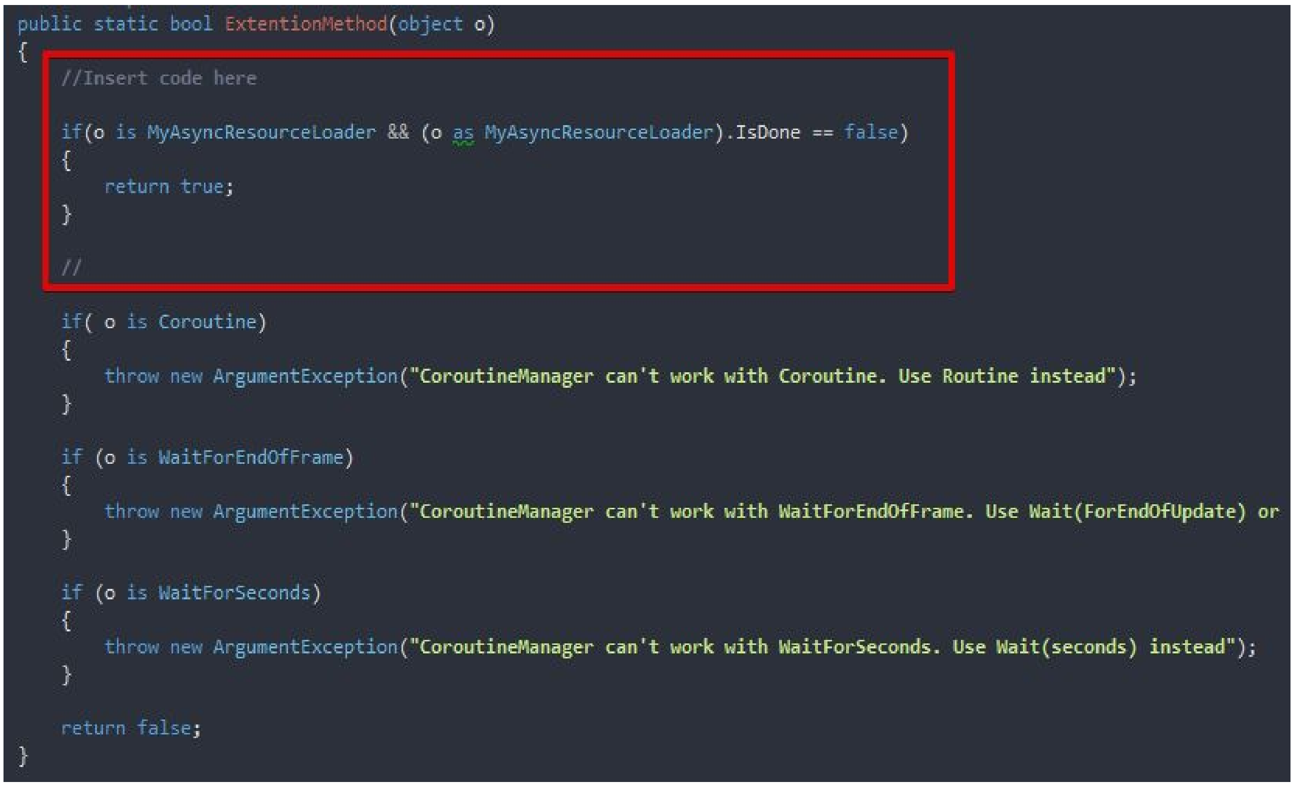
For expansion of functionality of coroutine you need to complement static method `ExtentionMethod(object o)` of `AdvancedCoroutinesExtention` class.

**Example:** Create class `MyAsyncResourceLoader`, with field `IsDone`, that becomes true when required resources will be loaded
Now we need to introduce `AdvancedCoroutines` to our class. For this we add the following code in `ExtentionMethod(object o)`

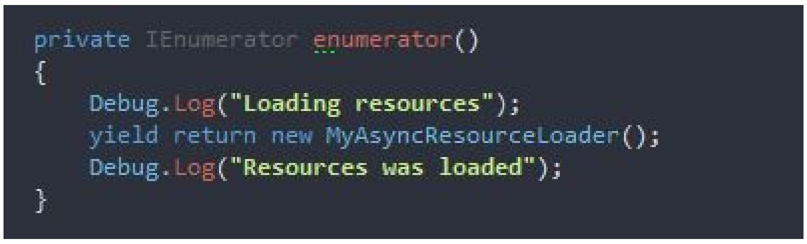
**Note:** extensions can be added in method `ExtentionMethod` in any place up to line `return false;`



```
public class MyAsyncResourceLoader
{
    public bool IsDone {get; private set; }

    public void LoadResources()
    {
        IsDone = true;
    }
}
```



```
public static bool ExtentionMethod(object o)
{
    //Insert code here

    if(o is MyAsyncResourceLoader && (o as MyAsyncResourceLoader).IsDone == false)
    {
        return true;
    }

    //

    if( o is Coroutine)
    {
        throw new ArgumentException("CoroutineManager can't work with Coroutine. Use Routine instead");
    }

    if (o is WaitForEndOfFrame)
    {
        throw new ArgumentException("CoroutineManager can't work with WaitForEndOfFrame. Use Wait(ForEndOfUpdate) or
    }

    if (o is WaitForSeconds)
    {
        throw new ArgumentException("CoroutineManager can't work with WaitForSeconds. Use Wait(seconds) instead");
    }

    return false;
}
```



```
private IEnumerator enumerator()
{
    Debug.Log("Loading resources");
    yield return new MyAsyncResourceLoader();
    Debug.Log("Resources was loaded");
}
```