

# **ADVANCED COROUTINES**

## **user guide**



# 1. Виды корутин в AdvancedCoroutines

## 1.1) Корутины привязанные к объекту

Корутины привязанные к объекту работают по принципу стандартных корутин Unity.

**Запуск корутины:**

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

- **Routine** - возвращаемый объект, аналог класса `Coroutine` в Unity
- **enumerator()** - объект типа `IEnumerator` или метод возвращающий `IEnumerator`
- **this** - ссылка на `MonoBehaviour`, к которому будет привязана корутина

**Примечание:** нельзя запустить корутину передав ей название метода строкой, как в стандартных корутинах Unity

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

**Остановка корутины:**

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

- **\_routine** - объект типа `Routine`, полученный при `CoroutineManager.StartCoroutine...`

**Примечание:** нельзя остановить корутину передав параметром название метода строкой или объект/метод `IEnumerator`

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

Остановка всех корутин, привязанных к `MonoBehaviour`:

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

- **this** - ссылка на `MonoBehaviour`, к которому привязана корутина

**Примечание:**

При уничтожении объекта `MonoBehaviour` (например при `Destroy(gameObject)`), все привязанные к нему корутины останавливаются

## 1.2) Самостоятельные корутины (Standalone Coroutines)

Корутины не привязанные к каким-либо объектам, продолжают работать даже при переходе между сценами

### Запуск standalone-корутины:

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

- **Routine** - возвращаемый объект, аналог класса Coroutine в Unity
- **enumerator()** - объект типа IEnumerator или метод возвращающий IEnumerator

**Примечание:** нельзя запустить корутину передав ей название метода строкой, как в стандартных корутинах Unity

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

### Остановка standalone-корутины

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

**\_routine** - объект типа Routine, полученный при CoroutineManager.StartStandaloneCoroutine

**Примечание:** нельзя остановить корутину передав параметром название метода строкой или объект/метод IEnumerator

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

**Примечание:** standalone-корутины могут быть остановлены только вручную

## 2. Класс Routine

При старте, методы `CoroutineManager.StartCoroutine()` или `CoroutineManager.StartStandaloneCoroutine()` возвращают объект типа **Routine**, который позволяет управлять корутиной. Он необходим для остановки корутин вручную и является уникальным идентификатором корутины

Корутину можно ставить на паузу вызвав метод `Pause()` у экземпляра `Routine`

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

Чтобы снять корутину с паузы нужно вызвать метод `Resume()` у экземпляра `Routine`

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

Чтобы проверить стоит ли корутина на паузе нужно вызвать метод `IsPaused()` у экземпляра `Routine`

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

Чтобы проверить является ли корутина уничтоженной, нужно вызвать статический метод `IsNull(Routine routine)` класса `Routine`

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

## 3. Работа с IEnumerator

### 3.1) Wait(float seconds)

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

Приостановит выполнение корутины на **sec** секунд

**Примечание:** нужно использовать вместо `yield return new WaitForSeconds(sec)`.

При использовании `WaitForSeconds` будет показана ошибка выполнения

### 3.2) Wait(Wait.WaitType)

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

Приостановит выполнение корутины до конца фрейма

**Примечание:** нужно использовать вместо `yield return new WaitForEndOfFrame()`.

При использовании `WaitForSeconds` будет показана ошибка выполнения

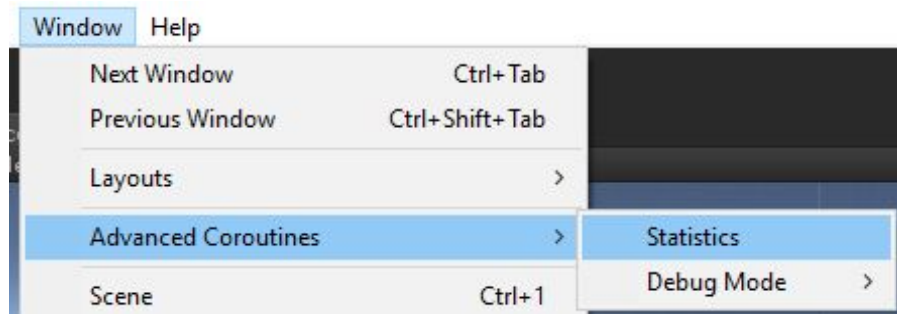
```
yield return new Wait(Wait.WaitType.ForEndOfUpdate);
```

Приостановит выполнение корутины по конца `Update()`.

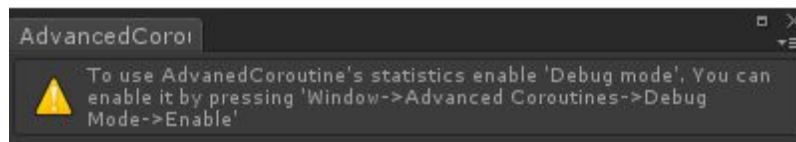
**Примечание:** корутина возобновит выполнение в `LateUpdate()`

## 4. Окно статистики

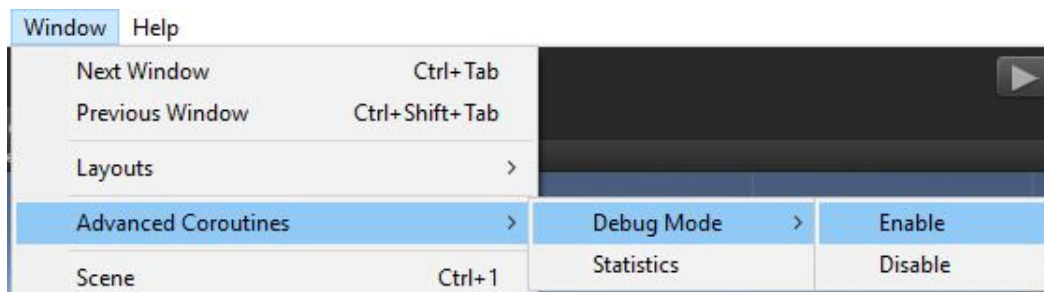
Чтобы открыть окно статистики нужно зайти в **Window->Advanced Coroutines->Statistics**



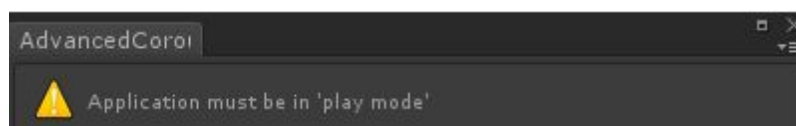
Если появится окно статистики со следующим сообщением



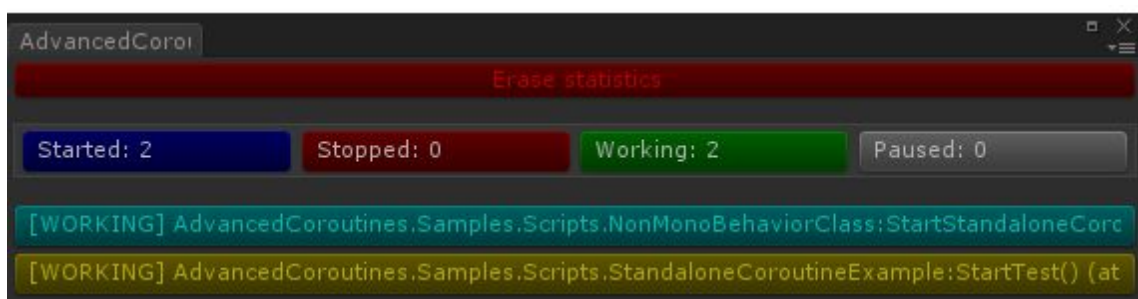
Перейдите в **Window->Advanced Coroutines->Debug Mode** и нажмите **Enable**. Это активирует директиву препроцессора, которая скрывает сбор статистики



Сбор статистики работает только в активном приложении, иначе вы увидите следующее сообщение



Работающее окно статистики выглядит так



Erase statistics	Очищает накопившуюся статистику
Started: 2	общее число запущенных корутин
Stopped: 0	общее число остановленных корутин
Working: 2	число активных (работающих) корутин
Paused: 0	общее число приостановленных корутин
[WORKING] AdvancedCoroutines.Samples.Scripts.NonMonoBehavior	активная standalone-корутина
[WORKING] AdvancedCoroutines.Samples.Scripts.StandaloneCorouti	активная корутина
[PAUSED] AdvancedCoroutines.Samples.Scripts.TimeCoroutineExam	приостановленная корутина
[WORKING] AdvancedCoroutines.Samples.Scripts.StandaloneCorol 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:\buildslz	развернутый вид активной корутины. Открывается нажатием на активную корутину. Отображает стек вызова корутины.

## 5. Расширение функционала

Для расширения функционала корутин необходимо дополнить статический метод `ExtentionMethod(object o)` класса `AdvancedCoroutinesExtention`

**Пример:** Создаем класс `MyAsyncResourceLoader`, с полем `IsDone`, которое станет true когда необходимые ресурсы будут загружены  
Теперь необходимо познакомить `AdvancedCoroutines` с нашим классом, для этого добавим следующий код в `ExtentionMethod(object o)`

**Внимание:** расширения можно добавлять в метод `ExtentionMethod` в любом месте до строки `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");
}
```