C# Async, Await

Async, await. Many methods do not immediately return. A method may need to query an external source. This takes time—and other code could run.

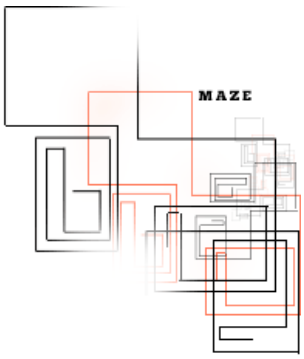
With async and await, we formalize and clarify how asynchronous, non-blocking methods begin and end. An async method can return only void or a Task.

Task:
A Task returns no value (it is void). A Task<int> returns an element of type int. This is a generic type.

Void

Main:
The Main method cannot be async. It cannot use the await keyword. It must start an async method with the Task class.

Note:
An async method will be run synchronously if it does not contain the await keyword.



This program uses the async and await keywords to asynchronously run a method. The program begins a long-running method (HandleFileAsync).

First:
We create a Task instance with the ProcessDataAsync method as the argument. We Start this task, and Wait for it to finish.

Messages:
The method displays a status message after it starts. When it ends, the results are displayed.

ProcessDataAsync:

► C# Linq
► C# Keywords
► C# Tutorial

In ProcessDataAsync, we call the HandleFileAsync method. We write a status message to the screen.

HandleFileAsync:

In HandleFileAsync, we use the StreamReader type and await the ReadToEndAsync method. We perform some computations.

Based on:

.NET 4.5

C# program that uses async, await, Task

```
using System;
using System.IO;
using System.Threading.Tasks;

class Program
{
    static void Main()
    {
        // Create task and start it.
        // ... Wait for it to complete.
        Task task = new Task(ProcessDataAsync);
        task.Start();
        task.Wait();
        Console.ReadLine();
    }

    static async void ProcessDataAsync()
    {
        // Start the HandleFile method.
        Task<int> task = HandleFileAsync("C:\\enable1.txt");

        // Control returns here before HandleFileAsync returns.
        // ... Prompt the user.
        Console.WriteLine("Please wait patiently " +
            "while I do something important.");

        // Wait for the HandleFile task to complete.
        // ... Display its results.
        int x = await task;
        Console.WriteLine("Count: " + x);
    }

    static async Task<int> HandleFileAsync(string file)
    {
        Console.WriteLine("HandleFile enter");
        int count = 0;

        // Read in the specified file.
        // ... Use async StreamReader method.
        using (StreamReader reader = new StreamReader(file))
        {
            string v = await reader.ReadToEndAsync();

            // ... Process the file data somehow.
            count += v.Length;

            // ... A slow-running computation.
            //     Dummy code.
            for (int i = 0; i < 10000; i++)
            {
                int x = v.GetHashCode();
```

```
                if (x == 0)
                {
                    count--;
                }
            }
        }
        Console.WriteLine("HandleFile exit");
        return count;
    }
}
```

Output: initial

```
HandleFile enter
Please wait patiently while I do something important.
```

Output: final

```
HandleFile enter
Please wait patiently while I do something important.
HandleFile exit
Count: 1916146
```

Above, the slow computation done in HandlFileAsync is for demonstration. If you change the path to a large text file that exists on your computer, the program should work.

Note:

We can do something (such as write a message) after the async method starts. This is not possible with synchronous methods.

Simple example. This program runs a computation asynchronously on every line entered in the console. It keeps accepting lines even when computations are running.

Action:

A lambda expression is specified as the argument to Task.Run. This is an action delegate.

Action

Allocate:

This method does a slow-running computation. But when run asynchronously, it does not cause the program to freeze.

Result:

Many user inputs can be handled while the computation is running. Each Allocate() call finishes at its own pace.

C# program that uses async computation

```
using System;
using System.Threading.Tasks;
```

```
class Program
{
    static void Main()
    {
        while (true)
        {
            // Start computation.
            Example();
            // Handle user input.
            string result = Console.ReadLine();
            Console.WriteLine("You typed: " + result);
        }
    }

    static async void Example()
    {
        // This method runs asynchronously.
        int t = await Task.Run(() => Allocate());
        Console.WriteLine("Compute: " + t);
    }

    static int Allocate()
    {
        // Compute total count of digits in strings.
        int size = 0;
        for (int z = 0; z < 100; z++)
        {
            for (int i = 0; i < 1000000; i++)
            {
                string value = i.ToString();
                if (value == null)
                {
                    return 0;
                }
                size += value.Length;
            }
        }
        return size;
    }
}
```

Output

```
hello
You typed: hello
good
You typed: good
day
You typed: day
Compute: 588889000
friend
You typed: friend
Compute: 588889000
Compute: 588889000
Compute: 588889000
Compute: 588889000
```

Main method. The async keyword cannot be used on the Main method. So we will need to add a second method before using an await call.

C# program that causes compile-time error

```
using System;
```

```
using System.Threading.Tasks;

class Program
{
    static async void Main()
    {
    }
}
```

Output

```
error CS4009: 'Program.Main()': an entry point cannot be marked
    with the 'async' modifier
```

A pattern. Async and await are a code pattern—they allow methods to asynchronously run. They are a form of syntactic sugar. They make code that uses threads easier to read.

Complexity. With async and await, the compiler helps with asynchronous code. We return a Task or void from an async method. Visual Studio reports warnings or errors on incorrect methods.

Types (StreamReader, HttpClient) contain "Async" methods. These should be called with the await keyword. And the await keyword must be used within an async method.

StreamReader
HttpClient

Task.Start:
The first async method call can occur with the Task Start method. This is an instance method.

Also:
Event handlers can be used with async methods. This is not currently shown here.

Asynchronous. This term does not mean multithreaded code.
By default,
code written with async
and await is single-threaded. But threaded code works well here.
Threads

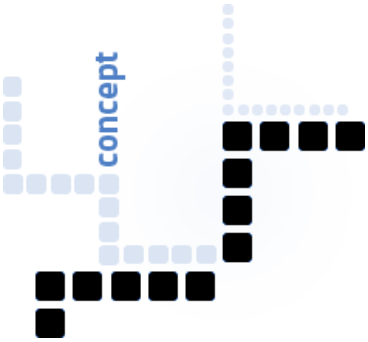
Task.Run. With the Task.Run method,
we can make code that uses async
and await multithreaded. Asynchronous code is code that returns upon completion.

And:
Other code can execute (even on the same thread) after an asynchronous task has started.

Note:
Thanks to Donnie Karns for pointing out that async and await code statements are not by default run on another thread.

The async and await keywords don't cause additional threads to be created. Async methods don't require multithreading because an async method doesn't run on its own thread.

Async, await: MSDN



Concept. Programs are full of methods that do not immediately return.
Sometimes an external slowdown,
as from a network,
is the cause,
not processor usage.

With these keywords, we run methods in an asynchronous way. Threads are optional. This style of code is more responsive. A network access can occur with no program freeze.