Tutorial No. 1

http://www.dotnetperls.com/async

Many methods do not immediately return. A method may need to query an external source. This takes time.
With **async** and**await**, we formalize and clarify how asynchronous, non-blocking methods begin and end.

> **Tip:**An async method can return only void or a Task.
> A Task returns no value.
> A Task<int> returns an element of type int.

Void

> **Tip 2:**The Main method cannot be async.
> It cannot use the await keyword.
> It must start an async method with the Task class.

> **Tip 3:**An async method will be run synchronously if it does not contain the await keyword.

Based on:

.NET 4.5

Example

This program uses the async and await keywords to asynchronously run a method. The program begins a long-running method (HandleFileAsync). It displays a status message after this method starts.

> **And:**When the method ends, the results of the computation are written to the screen.

The only difference between this program and a synchronous one is that we can do something (such as write a message) after

the async method starts. It avoids blocking control flow. Statements continue as normal.

To start, we create a Task instance with the ProcessDataAsync method as the argument. We Start this task, and Wait for it to finish. In ProcessDataAsync, we call the HandleFileAsync method. We write a status message to the screen.

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

Program that uses async, await, Task: C#

```
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
```

In this example, the slow computation done in HandleFileAsync is only for demonstration. It does nothing useful. If you change

the string "C:\enable1.txt" to a large text file that exists on your computer, the program should work.

Discussion

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. And this in turn makes that code less prone to flaws.

> **However:**There are many complexities in using async and await. We must return the a Task or void type from an async method.
>
> > **And:**Async methods that are incorrect will cause Visual Studio to report warnings or errors.

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. The first async method call can occur with the Task Start method.

StreamReaderHttpClient

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

Asynchronous

Asynchronous code is not multithreaded code.
By default,
code written with async
and await is single-threaded. With the Task.Run method, we can make it 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

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 async and await, we run methods asynchronously—threads are optional.