

# Asynchronous programming in C#: How and why to use it?

## Nagib Sabbag Filho

FIAP (Faculty of Informatics and Administration Paulista) Avenida Paulista, 1106 - 7º andar - Bela Vista, São Paulo, Brazil.

e-mail: profnagib.filho@fiap.com.br

PermaLink: <https://leaders.tec.br/article/fac61a>

jul 22 2024

---

### Abstract:

Asynchronous programming in C# improves the efficiency and scalability of applications by allowing the execution of multiple tasks concurrently without blocking the main thread.

### Key words:

asynchronous programming, C#, async, await, asynchronous tasks, performance, parallelism, concurrent processing, awaiter

---

## Introduction to Asynchronous Programming in C#

Asynchronous programming in C# is a relevant technique that allows a program to execute multiple tasks concurrently without blocking the main thread. This is extremely useful in situations where it is necessary to handle intensive input/output (I/O) operations, such as network requests, file reading, among others.

### Why use asynchronous programming in C#?

The use of asynchronous programming in C# brings several benefits, such as improved application responsiveness, better utilization of system resources, and the ability to create more scalable and efficient applications. Furthermore, asynchronous programming is essential in applications that need to handle a large volume of simultaneous requests, such as web servers and real-time processing systems.

### How to use asynchronous programming in C#?

There are several options for working asynchronously. In the context of this article, we will focus exclusively on C# and specific keywords. One way to use asynchronous programming in C# is to utilize the keywords `async` and `await` in methods that perform asynchronous operations. For example, when making an HTTP request in a C# application, we can use the `HttpClient` class along with the `async` and `await` keywords to make the operation asynchronous:

```
private async Task < string > GetHttpContentAsync(string url)
{
    using (HttpClient client = new HttpClient())
    {
        HttpResponseMessage response = await client.GetAsync(url);
        return await response.Content.ReadAsStringAsync();
    }
}
```

Thus, the `GetHttpContentAsync` method will perform the HTTP request asynchronously, without blocking the main thread of the program.

### Practical example of asynchronous programming in C#

Let's consider a more complex example, where we need to make multiple HTTP requests asynchronously and wait for all requests to be completed before continuing the process. In this case, we can use the `Task.WhenAll` method to wait for the completion of all asynchronous tasks:

```
private async Task < List < string > > GetMultipleHttpContentsAsync(List < string >
urls)
{
    List < Task < string > > tasks = new List < Task < string > > ();
    foreach (string url in urls)
    {
        tasks.Add(GetHttpContentAsync(url));
    }
    string[] results = await Task.WhenAll(tasks);
    return results.ToList();
}
```

In this example, the `GetMultipleHttpContentsAsync` method will make multiple HTTP requests asynchronously and wait for all of them to complete before returning the final result.

## Other ways to work asynchronously with C#

In addition to using the `async` and `await` keywords, there are other approaches and patterns for working with asynchronous programming in C#. Let's explore some of these options:

### Using `Task.Run` for CPU-bound operations

For operations that require intensive CPU processing, we can use the `Task.Run` method to execute these tasks on a separate thread, freeing the main thread to continue executing other operations:

```
private async Task < int > PerformCpuBoundOperationAsync()
{
    return await Task.Run(() =>
    {
        // Simulating an operation that requires intensive CPU processing
        int result = 0;
        for (int i = 0; i < 1000000; i++)
        {
            result += i;
        }
        return result;
    });
}
```

### Using `IAsyncEnumerable` for asynchronous stream processing

The `IAsyncEnumerable` interface allows asynchronous iteration over data collections, which is useful for processing data streams that arrive asynchronously:

```
private async IAsyncEnumerable GenerateSequenceAsync()
{
    for (int i = 0; i < 10; i++)
    {
        await Task.Delay(1000); // Simulates an asynchronous operation
        yield return i;
    }
}
```

```
    }  
}  
  
public async Task ProcessSequenceAsync()  
{  
    await foreach (var number in GenerateSequenceAsync())  
    {  
        Console.WriteLine(number);  
    }  
}
```

In this example, the sequence of numbers is generated asynchronously and processed as values become available, without blocking the main thread.

## Using `Parallel.ForEachAsync` for parallel processing

In scenarios where we need to process a large set of data in parallel, we can use the `Parallel.ForEachAsync` method to distribute tasks across multiple threads:

```
private async Task ProcessDataInParallelAsync(List data)  
{  
    await Parallel.ForEachAsync(data, async (item, cancellationToken) =>  
    {  
        await Task.Delay(100); // Simulates an asynchronous operation  
        Console.WriteLine(item);  
    });  
}
```

This method distributes the processing of the items in the data list across multiple threads, allowing operations to be executed in parallel and asynchronously.

## Considerations on performance and best practices

When working with asynchronous programming, it's important to follow some best practices to ensure optimized performance and avoid common problems. Make sure to:

Avoid excessive use of `async` and `await` in operations that do not need to be asynchronous.

Use `ConfigureAwait(false)` in libraries and components that do not depend on the synchronization context to avoid deadlocks.

Monitor and handle exceptions properly in asynchronous methods.

Take advantage of parallelization when appropriate to improve performance.

## Conclusion

Asynchronous programming in C# is a powerful technique that allows for improved efficiency and scalability of applications, especially in scenarios where it is necessary to handle intensive input/output operations. By using the `async` and `await` keywords, as well as other asynchronous approaches, it is possible to make asynchronous operations simple and intuitive. Therefore, consider using asynchronous programming in your C# projects for optimized performance and a smoother user experience.

## References

For more information on asynchronous programming in C#, check the following official references:

MICROSOFT. Official Microsoft Documentation on Asynchronous Programming in C#. Available at: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/>. Accessed on: July 23, 2024.

MICROSOFT. Documentation for the HttpClient Class. Available at: <https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient>. Accessed on: July 23, 2024.

MICROSOFT. Task-Based Asynchronous Programming Patterns (TAP). Available at: <https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/task-based-asynchronous-pattern-tap>. Accessed on: July 23, 2024.

MICROSOFT. Documentation on Parallel.ForEachAsync. Available at: <https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel.foreachasync>. Accessed on: July 23, 2024.

MICROSOFT. Documentation on IEnumerableAsync. Available at: <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-8#iasyncenumerable>. Accessed on: July 23, 2024.

---

Nagib Filho is a University Professor and Tech Manager. He has a track record of achievements in technical and agile certifications, including MCSD, MCSA, and PSM1. He holds a postgraduate degree in IT Management from SENAC and an MBA in Software Technology from USP. Nagib has taken extension programs at MIT and the University of Chicago. Other achievements include authoring a peer-reviewed article on chatbots, presented at the University of Barcelona.