await operator (C# reference)

11/08/2019 • 3 minutes to read •

In this article

The `await` operator suspends evaluation of the enclosing async method until the asynchronous operation represented by its operand completes. When the asynchronous operation completes, the `await` operator returns the result of the operation, if any. When the `await` operator is applied to the operand that represents already completed operation, it returns the result of the operation immediately without suspension of the enclosing method. The `await` operator doesn't block the thread that evaluates the async method. When the `await` operator suspends the enclosing async method, the control returns to the caller of the method.

In the following example, the HttpClient.GetByteArrayAsync method returns the `Task<byte[]>` instance, which represents an asynchronous operation that produces a byte array when it completes. Until the operation completes, the `await` operator suspends the `DownloadDocsMainPageAsync` method. When `DownloadDocsMainPageAsync` gets suspended, control is returned to the `Main` method, which is the caller of `DownloadDocsMainPageAsync`. The `Main` method executes until it needs the result of the asynchronous operation performed by the `DownloadDocsMainPageAsync` method. When GetByteArrayAsync gets all the bytes, the rest of the `DownloadDocsMainPageAsync` method is evaluated. After that, the rest of the `Main` method is evaluated.

C# Copy

```
using System;
using System.Net.Http;
using System.Threading.Tasks;

public class AwaitOperator
{
    public static async Task Main()
    {
        Task<int> downloading = DownloadDocsMainPageAsync();
```

```
        Console.WriteLine($"{nameof(Main)}: Launched downloading.");

        int bytesLoaded = await downloading;
        Console.WriteLine($"{nameof(Main)}: Downloaded {bytesLoaded} bytes.");
    }

    private static async Task<int> DownloadDocsMainPageAsync()
    {
        Console.WriteLine($"{nameof(DownloadDocsMainPageAsync)}: About to start
downloading.");

        var client = new HttpClient();
        byte[] content = await
client.GetByteArrayAsync("https://docs.microsoft.com/en-us/");

        Console.WriteLine($"{nameof(DownloadDocsMainPageAsync)}: Finished
downloading.");
        return content.Length;
    }
}
// Output similar to:
// DownloadDocsMainPageAsync: About to start downloading.
// Main: Launched downloading.
// DownloadDocsMainPageAsync: Finished downloading.
// Main: Downloaded 27700 bytes.
```

The preceding example uses the async Main method, which is possible beginning with C# 7.1. For more information, see the await operator in the Main method section.

> **Note**
>
> For an introduction to asynchronous programming, see **Asynchronous programming with async and await**. Asynchronous programming with `async` and `await` follows the **task-based asynchronous pattern**.

You can use the `await` operator only in a method, lambda expression, or anonymous method that is modified by the async keyword. Within an async method, you cannot use the `await` operator in the body of a synchronous function, inside the block of a lock statement, and in an unsafe context.

The operand of the `await` operator is usually of one of the following .NET types: Task, Task<TResult>, ValueTask, or ValueTask<TResult>. However, any awaitable expression can be the operand of the `await` operator. For more information, see the Awaitable expressions section of the C# language specification.

Beginning with C# 8.0, you can use the `await foreach` statement to consume an asynchronous stream of data. For more information, see the Asynchronous streams section of the What's new in C# 8.0 article.

The type of expression `await t` is `TResult` if the type of expression `t` is Task<TResult> or ValueTask<TResult>. If the type of `t` is Task or ValueTask, the type of `await t` is `void`. In both cases, if `t` throws an exception, `await t` rethrows the exception. For more information about exception handling, see the Exceptions in async methods section of the try-catch statement article.

The `async` and `await` keywords are available in C# 5 and later.

await operator in the Main method

Beginning with C# 7.1, the Main method, which is the application entry point, can return `Task` or `Task<int>`, enabling it to be async so you can use the `await` operator in its body. In earlier C# versions, to ensure that the `Main` method waits for the completion of an asynchronous operation, you can retrieve the value of the Task<TResult>.Result property of the Task<TResult> instance that is returned by the corresponding async method. For asynchronous operations that don't produce a value, you can call the Task.Wait method. For information about how to select the language version, see C# language versioning.

C# language specification

For more information, see the Await expressions section of the C# language specification.

See also

- C# reference
- C# operators
- async
- Task asynchronous programming model
- Asynchronous programming
- Async in depth
- Walkthrough: accessing the Web by using async and await
- Tutorial: Generate and consume async streams using C# 8.0 and .NET Core 3.0

Is this page helpful?

Yes No