async (C# Reference)

05/21/2017 • 4 minutes to read • Contributors all

In this article

Example
Return Types
See also

Use the `async` modifier to specify that a method, lambda expression, or anonymous method is asynchronous. If you use this modifier on a method or expression, it's referred to as an *async method*. The following example defines an async method named `ExampleMethodAsync`:

C#

```
public async Task<int> ExampleMethodAsync()
{
    // . . . .
}
```

If you're new to asynchronous programming or do not understand how an async method uses the `await` keyword to do potentially long-running work without blocking the caller's thread, read the introduction in Asynchronous Programming with async and await. The following code is found inside an async method and calls the HttpClient.GetStringAsync method:

C#

```
string contents = await httpClient.GetStringAsync(requestUrl);
```

An async method runs synchronously until it reaches its first `await` expression, at which point the method is suspended until the awaited task is complete. In the meantime, control returns to the caller of the method, as the example in the next section shows.

If the method that the `async` keyword modifies doesn't contain an `await` expression or statement, the method executes synchronously. A compiler warning alerts you to any async methods that don't contain `await` statements, because that situation might indicate an error. See Compiler Warning (level 1) CS4014.

The `async` keyword is contextual in that it's a keyword only when it modifies a method, a lambda expression, or an anonymous method. In all other contexts, it's interpreted as an identifier.

Example

The following example shows the structure and flow of control between an async event handler, `StartButton_Click`, and an async method, `ExampleMethodAsync`. The result from the async method is the number of characters of a web page. The code is suitable for a Windows Presentation Foundation (WPF) app or Windows Store app that you create in Visual Studio; see the code comments for setting up the app.

You can run this code in Visual Studio as a Windows Presentation Foundation (WPF) app or a Windows Store app. You need a Button control named `StartButton` and a Textbox control named `ResultsTextBox`. Remember to set the names and handler so that you have something like this:

XAML Copy

```
<Button Content="Button" HorizontalAlignment="Left" Margin="88,77,0,0"
VerticalAlignment="Top" Width="75"
        Click="StartButton_Click" Name="StartButton"/>
<TextBox HorizontalAlignment="Left" Height="137" Margin="88,140,0,0"
TextWrapping="Wrap"
         Text="&lt;Enter a URL&gt;" VerticalAlignment="Top" Width="310"
Name="ResultsTextBox"/>
```

To run the code as a WPF app:

- Paste this code into the `MainWindow` class in MainWindow.xaml.cs.
- Add a reference to System.Net.Http.
- Add a `using` directive for System.Net.Http.

To run the code as a Windows Store app:

- Paste this code into the `MainPage` class in MainPage.xaml.cs.
- Add using directives for System.Net.Http and System.Threading.Tasks.

C# Copy

```
private async void StartButton_Click(object sender, RoutedEventArgs e)
{
    // ExampleMethodAsync returns a Task<int>, which means that the method
    // eventually produces an int result. However, ExampleMethodAsync returns
    // the Task<int> value as soon as it reaches an await.
    ResultsTextBox.Text += "\n";

    try
    {
        int length = await ExampleMethodAsync();
        // Note that you could put "await ExampleMethodAsync()" in the next
line where
        // "length" is, but due to when '+=' fetches the value of
ResultsTextBox, you
        // would not see the global side effect of ExampleMethodAsync setting
the text.
        ResultsTextBox.Text += String.Format("Length: {0:N0}\n", length);
    }
    catch (Exception)
    {
        // Process the exception if one occurs.
    }
}

public async Task<int> ExampleMethodAsync()
{
    var httpClient = new HttpClient();
    int exampleInt = (await
httpClient.GetStringAsync("http://msdn.microsoft.com")).Length;
    ResultsTextBox.Text += "Preparing to finish ExampleMethodAsync.\n";
    // After the following return statement, any method that's awaiting
    // ExampleMethodAsync (in this case, StartButton_Click) can get the
    // integer result.
    return exampleInt;
}
// The example displays the following output:
// Preparing to finish ExampleMethodAsync.
// Length: 53292
```

Important

For more information about tasks and the code that executes while waiting for a task, see Asynchronous Programming with async and await. For a full WPF example that uses similar elements, see Walkthrough: Accessing the Web by Using Async and Await.

Return Types

An async method can have the following return types:

- Task
- Task<TResult>
- void, which should only be used for event handlers.
- Starting with C# 7.0, any type that has an accessible `GetAwaiter` method. The `System.Threading.Tasks.ValueTask<TResult>` type is one such implementation. It is available by adding the NuGet package `System.Threading.Tasks.Extensions`.

The async method can't declare any in, ref or out parameters, nor can it have a reference return value, but it can call methods that have such parameters.

You specify `Task<TResult>` as the return type of an async method if the return statement of the method specifies an operand of type `TResult`. You use `Task` if no meaningful value is returned when the method is completed. That is, a call to the method returns a `Task`, but when the `Task` is completed, any `await` expression that's awaiting the `Task` evaluates to `void`.

You use the `void` return type primarily to define event handlers, which require that return type. The caller of a `void`-returning async method can't await it and can't catch exceptions that the method throws.

Starting with C# 7.0, you return another type, typically a value type, that has a `GetAwaiter` method to minimize memory allocations in performance-critical sections of code.

For more information and examples, see Async Return Types.

See also

- AsyncStateMachineAttribute
- await
- Walkthrough: Accessing the Web by Using Async and Await
- Asynchronous Programming with async and await