**async (C# Reference)**

**Visual Studio 2013**

[Other Versions](javascript:;)

http://i.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [Visual Studio 2012](http://msdn.microsoft.com/en-us/library/hh156513(d=printer,v=vs.110).aspx)

By using the async modifier, you specify that a method, [lambda expression](http://msdn.microsoft.com/en-us/library/bb397687.aspx), or [anonymous method](http://msdn.microsoft.com/en-us/library/0yw3tz5k.aspx) is asynchronous. If you use this modifier on a method or expression, it's referred to as an async method.

C#

public async Task<int> ExampleMethodAsync()

{

// . . . .

}

If you're new to asynchronous programming, you can find an introduction in [Asynchronous Programming with Async and Await (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/hh191443.aspx).

An async method provides a convenient way to do potentially long-running work without blocking the caller's thread. The caller (say, M1) of an async method can resume its work without waiting for the async method to finish. However, M1 typically uses the await keyword so that it returns immediately, allowing M1’s caller to resume work or return to the thread’s synchronization context (or message pump).

string contents = await contentsTask;

The 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, because that situation might indicate an error. See [Compiler Warning (level 1) CS4014](http://msdn.microsoft.com/en-us/library/hh873131.aspx).

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](javascript:void(0))

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 length of a downloaded website. The code is suitable for a Windows Presentation Foundation (WPF) app or Windows Store app that you create in Visual Studio 2013; see the code comments for setting up the app.

C#

// You can run this code in Visual Studio 2013 as a WPF app or a Windows Store app.

// You need a button (StartButton) and a textbox (ResultsTextBox).

// Remember to set the names and handler so that you have something like this:

// <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="TextBox" 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, and

// 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, and

// add using directives for System.Net.Http and System.Threading.Tasks.

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}\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;

}

// Output:

// Preparing to finish ExampleMethodAsync.

// Length: 53292

|  |
| --- |
| **Important noteImportant** |
| For more information about tasks and the code that executes while waiting for a task, see [Asynchronous Programming with Async and Await (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/hh191443.aspx). For a full WPF example that uses similar elements, see [Walkthrough: Accessing the Web by Using Async and Await (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/hh300224.aspx). You can download the walkthrough code from [Developer Code Samples](http://go.microsoft.com/fwlink/?LinkId=255191). |

[Return Types](javascript:void(0))

An async method can have a return type of [Task](http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.aspx), [Task<TResult>](http://msdn.microsoft.com/en-us/library/dd321424.aspx), or [void](http://msdn.microsoft.com/en-us/library/yah0tteb.aspx). The method can't declare any [ref](http://msdn.microsoft.com/en-us/library/14akc2c7.aspx) or [out](http://msdn.microsoft.com/en-us/library/t3c3bfhx.aspx) parameters, but it can call methods that have such parameters.

You specify Task<TResult> as the return type of an async method if the [return](http://msdn.microsoft.com/en-us/library/1h3swy84.aspx) 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.

For more information and examples, see [Async Return Types (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/hh524395.aspx).

**Async Return Types (C# and Visual Basic)**

**Visual Studio 2013**

[Other Versions](javascript:;)

http://i.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [Visual Studio 2012](http://msdn.microsoft.com/en-us/library/hh524395(d=printer,v=vs.110).aspx)

Async methods have three possible return types: [Task<TResult>](http://msdn.microsoft.com/en-us/library/dd321424.aspx), [Task](http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.aspx), and void. In Visual Basic, the void return type is written as a [Sub](http://msdn.microsoft.com/en-us/library/831f9wka.aspx) procedure. For more information about async methods, see [Asynchronous Programming with Async and Await (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/hh191443.aspx).

Each return type is examined in one of the following sections, and you can find a full example that uses all three types at the end of the topic.

|  |
| --- |
| **NoteNote** |
| To run the example, you must have Visual Studio 2012, Visual Studio 2013, [Visual Studio Express 2012 for Windows Desktop](http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-for-windows-desktop), Visual Studio Express 2013 for Windows, or the .NET Framework 4.5 or 4.5.1 installed on your computer. |

This topic contains the following sections.

* [Task(T) Return Type](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx#BKMK_TaskTReturnType)
* [Task Return Type](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx#BKMK_TaskReturnType)
* [Void Return Type](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx#BKMK_VoidReturnType)
* [Complete Example](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx#BKMK_Example)
* [Related Topics](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx#seeAlsoToggle)

[Task(T) Return Type](javascript:void(0))

The [Task<TResult>](http://msdn.microsoft.com/en-us/library/dd321424.aspx) return type is used for an async method that contains a [Return](http://msdn.microsoft.com/en-us/library/2e34641s.aspx) (Visual Basic) or [return](http://msdn.microsoft.com/en-us/library/1h3swy84.aspx) (C#) statement in which the operand has type TResult.

In the following example, the TaskOfT\_MethodAsync async method contains a return statement that returns an integer. Therefore, the method declaration must specify a return type of Task(Of Integer) in Visual Basic or Task<int> in C#.

C#

[VB](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

// TASK<T> EXAMPLE

async Task<int> TaskOfT\_MethodAsync()

{

// The body of the method is expected to contain an awaited asynchronous

// call.

// Task.FromResult is a placeholder for actual work that returns a string.

var today = await Task.FromResult<string>(DateTime.Now.DayOfWeek.ToString());

// The method then can process the result in some way.

int leisureHours;

if (today.First() == 'S')

leisureHours = 16;

else

leisureHours = 5;

// Because the return statement specifies an operand of type int, the

// method must have a return type of Task<int>.

return leisureHours;

}

When TaskOfT\_MethodAsync is called from within an await expression, the await expression retrieves the integer value (the value of leisureHours) that's stored in the task that's returned by TaskOfT\_MethodAsync. For more information about await expressions, see [Await Operator (Visual Basic)](http://msdn.microsoft.com/en-us/library/hh156570.aspx) or [await (C# Reference)](http://msdn.microsoft.com/en-us/library/hh156528.aspx).

The following code calls and awaits method TaskOfT\_MethodAsync. The result is assigned to the result1 variable.

C#

[VB](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

// Call and await the Task<T>-returning async method in the same statement.

int result1 = await TaskOfT\_MethodAsync();

You can better understand how this happens by separating the call to TaskOfT\_MethodAsync from the application of Await or await, as the following code shows. A call to method TaskOfT\_MethodAsync that isn't immediately awaited returns a Task(Of Integer) or Task<int>, as you would expect from the declaration of the method. The task is assigned to the integerTask variable in the example. Because integerTask is a [Task<TResult>](http://msdn.microsoft.com/en-us/library/dd321424.aspx), it contains a [Result](http://msdn.microsoft.com/en-us/library/dd321468.aspx) property of type TResult. In this case, TResult represents an integer type. When Await or await is applied to integerTask, the await expression evaluates to the contents of the [Result](http://msdn.microsoft.com/en-us/library/dd321468.aspx) property of integerTask. The value is assigned to the result2 variable.

|  |
| --- |
| **Caution noteCaution** |
| The [Result](http://msdn.microsoft.com/en-us/library/dd321468.aspx) property is a blocking property. If you try to access it before its task is finished, the thread that's currently active is blocked until the task completes and the value is available. In most cases, you should access the value by using Await or await instead of accessing the property directly. |

C#

[VB](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

// Call and await in separate statements.

Task<int> integerTask = TaskOfT\_MethodAsync();

// You can do other work that does not rely on integerTask before awaiting.

textBox1.Text += String.Format("Application can continue working while the Task<T> runs. . . . \r\n");

int result2 = await integerTask;

The display statements in the following code verify that the values of the result1 variable, the result2 variable, and the Result property are the same. Remember that the Result property is a blocking property and shouldn't be accessed before its task has been awaited.

C#

[VB](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

// Display the values of the result1 variable, the result2 variable, and

// the integerTask.Result property.

textBox1.Text += String.Format("\r\nValue of result1 variable: {0}\r\n", result1);

textBox1.Text += String.Format("Value of result2 variable: {0}\r\n", result2);

textBox1.Text += String.Format("Value of integerTask.Result: {0}\r\n", integerTask.Result);

[Task Return Type](javascript:void(0))

Async methods that don't contain a return statement or that contain a return statement that doesn't return an operand usually have a return type of [Task](http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.aspx). Such methods would be void-returning methods ([Sub](http://msdn.microsoft.com/en-us/library/831f9wka.aspx) procedures in Visual Basic) if they were written to run synchronously. If you use a Task return type for an async method, a calling method can use an await operator to suspend the caller's completion until the called async method has finished.

In the following example, async method Task\_MethodAsync doesn't contain a return statement. Therefore, you specify a return type of Task for the method, which enables Task\_MethodAsync to be awaited. The definition of the Task type doesn't include a Result property to store a return value.

C#

[VB](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

// TASK EXAMPLE

async Task Task\_MethodAsync()

{

// The body of an async method is expected to contain an awaited

// asynchronous call.

// Task.Delay is a placeholder for actual work.

await Task.Delay(2000);

// Task.Delay delays the following line by two seconds.

textBox1.Text += String.Format("\r\nSorry for the delay. . . .\r\n");

// This method has no return statement, so its return type is Task.

}

Task\_MethodAsync is called and awaited by using an await statement instead of an await expression, similar to the calling statement for a synchronous Sub or void-returning method. The application of an await operator in this case doesn't produce a value.

The following code calls and awaits method Task\_MethodAsync.

C#

[VB](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-6)

// Call and await the Task-returning async method in the same statement.

await Task\_MethodAsync();

As in the previous [Task<TResult>](http://msdn.microsoft.com/en-us/library/dd321424.aspx) example, you can separate the call to Task\_MethodAsync from the application of an await operator, as the following code shows. However, remember that a Task doesn't have a Result property, and that no value is produced when an await operator is applied to a Task.

The following code separates calling Task\_MethodAsync from awaiting the task that Task\_MethodAsync returns.

C#

[VB](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-7)

// Call and await in separate statements.

Task simpleTask = Task\_MethodAsync();

// You can do other work that does not rely on simpleTask before awaiting.

textBox1.Text += String.Format("\r\nApplication can continue working while the Task runs. . . .\r\n");

await simpleTask;

[Void Return Type](javascript:void(0))

The primary use of the void return type (Sub procedures in Visual Basic) is in event handlers, where a void return type is required. A void return also can be used to override void-returning methods or for methods that perform activities that can be categorized as "fire and forget." However, you should return a Task wherever possible, because a void-returning async method can't be awaited. Any caller of such a method must be able to continue to completion without waiting for the called async method to finish, and the caller must be independent of any values or exceptions that the async method generates.

The caller of a void-returning async method can't catch exceptions that are thrown from the method, and such unhandled exceptions are likely to cause your application to fail. If an exception occurs in an async method that returns a [Task](http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.aspx) or [Task<TResult>](http://msdn.microsoft.com/en-us/library/dd321424.aspx), the exception is stored in the returned task, and rethrown when the task is awaited. Therefore, make sure that any async method that can produce an exception has a return type of [Task](http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.aspx) or [Task<TResult>](http://msdn.microsoft.com/en-us/library/dd321424.aspx) and that calls to the method are awaited.

For more information about how to catch exceptions in async methods, see [try-catch (C# Reference)](http://msdn.microsoft.com/en-us/library/0yd65esw.aspx) or [Try...Catch...Finally Statement (Visual Basic)](http://msdn.microsoft.com/en-us/library/fk6t46tz.aspx).

The following code defines an async event handler.

C#

[VB](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-8)

// VOID EXAMPLE

private async void button1\_Click(object sender, RoutedEventArgs e)

{

textBox1.Clear();

// Start the process and await its completion. DriverAsync is a

// Task-returning async method.

await DriverAsync();

// Say goodbye.

textBox1.Text += "\r\nAll done, exiting button-click event handler.";

}

[Complete Example](javascript:void(0))

The following Windows Presentation Foundation (WPF) project contains the code examples from this topic.

To run the project, perform the following steps:

1. Start Visual Studio.
2. On the menu bar, choose File, New, Project.

The New Project dialog box opens.

1. In the Installed, Templates category, choose Visual Basic or Visual C#, and then choose Windows. Choose WPF Application from the list of project types.
2. Enter AsyncReturnTypes as the name of the project, and then choose the OK button.

The new project appears in Solution Explorer.

1. In the Visual Studio Code Editor, choose the MainWindow.xaml tab.

If the tab is not visible, open the shortcut menu for MainWindow.xaml in Solution Explorer, and then choose Open.

1. In the XAML window of MainWindow.xaml, replace the code with the following code.

C#

[VB](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-9)

<Window x:Class="AsyncReturnTypes.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="MainWindow" Height="350" Width="525">

<Grid>

<Button x:Name="button1" Content="Start" HorizontalAlignment="Left" Margin="214,28,0,0" VerticalAlignment="Top" Width="75" HorizontalContentAlignment="Center" FontWeight="Bold" FontFamily="Aharoni" Click="button1\_Click"/>

<TextBox x:Name="textBox1" Margin="0,80,0,0" TextWrapping="Wrap" FontFamily="Lucida Console"/>

</Grid>

</Window>

A simple window that contains a text box and a button appears in the Design window of MainWindow.xaml.

1. In Solution Explorer, open the shortcut menu for MainWindow.xaml.vb or MainWindow.xaml.cs, and then choose View Code.
2. Replace the code in MainWindow.xaml.vb or MainWindow.xaml.cs with the following code.

C#

[VB](http://msdn.microsoft.com/en-us/library/hh524395(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-10)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace AsyncReturnTypes

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

// VOID EXAMPLE

private async void button1\_Click(object sender, RoutedEventArgs e)

{

textBox1.Clear();

// Start the process and await its completion. DriverAsync is a

// Task-returning async method.

await DriverAsync();

// Say goodbye.

textBox1.Text += "\r\nAll done, exiting button-click event handler.";

}

async Task DriverAsync()

{

// Task<T>

// Call and await the Task<T>-returning async method in the same statement.

int result1 = await TaskOfT\_MethodAsync();

// Call and await in separate statements.

Task<int> integerTask = TaskOfT\_MethodAsync();

// You can do other work that does not rely on integerTask before awaiting.

textBox1.Text += String.Format("Application can continue working while the Task<T> runs. . . . \r\n");

int result2 = await integerTask;

// Display the values of the result1 variable, the result2 variable, and

// the integerTask.Result property.

textBox1.Text += String.Format("\r\nValue of result1 variable: {0}\r\n", result1);

textBox1.Text += String.Format("Value of result2 variable: {0}\r\n", result2);

textBox1.Text += String.Format("Value of integerTask.Result: {0}\r\n", integerTask.Result);

// Task

// Call and await the Task-returning async method in the same statement.

await Task\_MethodAsync();

// Call and await in separate statements.

Task simpleTask = Task\_MethodAsync();

// You can do other work that does not rely on simpleTask before awaiting.

textBox1.Text += String.Format("\r\nApplication can continue working while the Task runs. . . .\r\n");

await simpleTask;

}

// TASK<T> EXAMPLE

async Task<int> TaskOfT\_MethodAsync()

{

// The body of the method is expected to contain an awaited asynchronous

// call.

// Task.FromResult is a placeholder for actual work that returns a string.

var today = await Task.FromResult<string>(DateTime.Now.DayOfWeek.ToString());

// The method then can process the result in some way.

int leisureHours;

if (today.First() == 'S')

leisureHours = 16;

else

leisureHours = 5;

// Because the return statement specifies an operand of type int, the

// method must have a return type of Task<int>.

return leisureHours;

}

// TASK EXAMPLE

async Task Task\_MethodAsync()

{

// The body of an async method is expected to contain an awaited

// asynchronous call.

// Task.Delay is a placeholder for actual work.

await Task.Delay(2000);

// Task.Delay delays the following line by two seconds.

textBox1.Text += String.Format("\r\nSorry for the delay. . . .\r\n");

// This method has no return statement, so its return type is Task.

}

}

}

1. Choose the F5 key to run the program, and then choose the Start button.

The following output should appear.

Application can continue working while the Task<T> runs. . . .

Value of result1 variable: 5

Value of result2 variable: 5

Value of integerTask.Result: 5

Sorry for the delay. . . .

Application can continue working while the Task runs. . . .

Sorry for the delay. . . .

All done, exiting button-click event handler.