**Asynchronous Programming with Async and Await (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/hh191443(d=printer,v=vs.110).aspx)

You can avoid performance bottlenecks and enhance the overall responsiveness of your application by using asynchronous programming. However, traditional techniques for writing asynchronous applications can be complicated, making them difficult to write, debug, and maintain.

Visual Studio 2012 introduces a simplified approach, async programming, that leverages asynchronous support in the .NET Framework 4.5 and the Windows Runtime. The compiler does the difficult work that the developer used to do, and your application retains a logical structure that resembles synchronous code. As a result, you get all the advantages of asynchronous programming with a fraction of the effort.

This topic contains the following sections.

* [Async Improves Responsiveness](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx#BKMK_WhentoUseAsynchrony)
* [Async Methods Are Easier to Write](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx#BKMK_HowtoWriteanAsyncMethod)
* [What Happens in an Async Method](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx#BKMK_WhatHappensUnderstandinganAsyncMethod)
* [API Async Methods](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx#BKMK_APIAsyncMethods)
* [Threads](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx#BKMK_Threads)
* [Async and Await](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx#BKMK_AsyncandAwait)
* [Return Types and Parameters](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx#BKMK_ReturnTypesandParameters)
* [Naming Convention](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx#BKMK_NamingConvention)
* [Related Topics](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx#BKMK_RelatedTopics)
* [Complete Example](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx#BKMK_CompleteExample)
* [Related Topics](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx#seeAlsoToggle)

This topic provides an overview of when and how to use async programming and includes links to support topics that contain details and examples.

[Async Improves Responsiveness](javascript:void(0))

Asynchrony is essential for activities that are potentially blocking, such as when your application accesses the web. Access to a web resource sometimes is slow or delayed. If such an activity is blocked within a synchronous process, the entire application must wait. In an asynchronous process, the application can continue with other work that doesn't depend on the web resource until the potentially blocking task finishes.

The following table shows typical areas where asynchronous programming improves responsiveness. The listed APIs from the .NET Framework 4.5 and the Windows Runtime contain methods that support async programming.

|  |  |
| --- | --- |
| **Application area** | **Supporting APIs that contain async methods** |
| Web access | [HttpClient](http://msdn.microsoft.com/en-us/library/system.net.http.httpclient.aspx) , [SyndicationClient](http://go.microsoft.com/fwlink/p/?LinkId=259441) |
| Working with files | [StorageFile](http://go.microsoft.com/fwlink/p/?LinkId=248220), [StreamWriter](http://msdn.microsoft.com/en-us/library/system.io.streamwriter.aspx), [StreamReader](http://msdn.microsoft.com/en-us/library/system.io.streamreader.aspx), [XmlReader](http://msdn.microsoft.com/en-us/library/system.xml.xmlreader.aspx) |
| Working with images | [MediaCapture](http://go.microsoft.com/fwlink/p/?LinkId=261839), [BitmapEncoder](http://go.microsoft.com/fwlink/p/?LinkId=261840), [BitmapDecoder](http://go.microsoft.com/fwlink/p/?LinkId=261841) |
| WCF programming | [Synchronous and Asynchronous Operations](http://go.microsoft.com/fwlink/p/?LinkID=192382) |
|  |  |

Asynchrony proves especially valuable for applications that access the UI thread because all UI-related activity usually shares one thread. If any process is blocked in a synchronous application, all are blocked. Your application stops responding, and you might conclude that it has failed when instead it's just waiting.

When you use asynchronous methods, the application continues to respond to the UI. You can resize or minimize a window, for example, or you can close the application if you don't want to wait for it to finish.

The async-based approach adds the equivalent of an automatic transmission to the list of options that you can choose from when designing asynchronous operations. That is, you get all the benefits of traditional asynchronous programming but with much less effort from the developer.

[Async Methods Are Easier to Write](javascript:void(0))

The [Async](http://msdn.microsoft.com/en-us/library/hh191564.aspx) and [Await](http://msdn.microsoft.com/en-us/library/hh191564.aspx) keywords in Visual Basic and the [async](http://msdn.microsoft.com/en-us/library/hh156513.aspx) and [await](http://msdn.microsoft.com/en-us/library/hh156528.aspx) keywords in C# are the heart of async programming. By using those two keywords, you can use resources in the .NET Framework or the Windows Runtime to create an asynchronous method almost as easily as you create a synchronous method. Asynchronous methods that you define by using async and await are referred to as async methods.

The following example shows an async method. Almost everything in the code should look completely familiar to you. The comments call out the features that you add to create the asynchrony.

You can find the complete example file at the end of this topic, and you can download the sample from [Async Sample: Example from "Asynchronous Programming with Async and Await"](http://go.microsoft.com/fwlink/?LinkID=261549) .

C#

[VB](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

// Three things to note in the signature:

// - The method has an async modifier.

// - The return type is Task or Task<T>. (See "Return Types" section.)

// Here, it is Task<int> because the return statement returns an integer.

// - The method name ends in "Async."

async Task<int> AccessTheWebAsync()

{

// You need to add a reference to System.Net.Http to declare client.

HttpClient client = new HttpClient();

// GetStringAsync returns a Task<string>. That means that when you await the

// task you'll get a string (urlContents).

Task<string> getStringTask = client.GetStringAsync("http://msdn.microsoft.com");

// You can do work here that doesn't rely on the string from GetStringAsync.

DoIndependentWork();

// The await operator suspends AccessTheWebAsync.

// - AccessTheWebAsync can't continue until getStringTask is complete.

// - Meanwhile, control returns to the caller of AccessTheWebAsync.

// - Control resumes here when getStringTask is complete.

// - The await operator then retrieves the string result from getStringTask.

string urlContents = await getStringTask;

// The return statement specifies an integer result.

// Any methods that are awaiting AccessTheWebAsync retrieve the length value.

return urlContents.Length;

}

If AccessTheWebAsync doesn't have any work that it can do between calling GetStringAsync and awaiting its completion, you can simplify your code by calling and awaiting in the following single statement.

C#

[VB](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

string urlContents = await client.GetStringAsync();

The following characteristics summarize what makes the previous example an async method.

* The method signature includes an Async or async modifier.
* The name of an async method, by convention, ends with an "Async" suffix.
* The return type is one of the following types:
  + [Task<TResult>](http://msdn.microsoft.com/en-us/library/dd321424.aspx) if your method has a return statement in which the operand has type TResult.
  + [Task](http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.aspx) if your method has no return statement or has a return statement with no operand.
  + Void (a [Sub](http://msdn.microsoft.com/en-us/library/831f9wka.aspx) in Visual Basic) if you're writing an async event handler.

For more information, see "Return Types and Parameters" later in this topic.

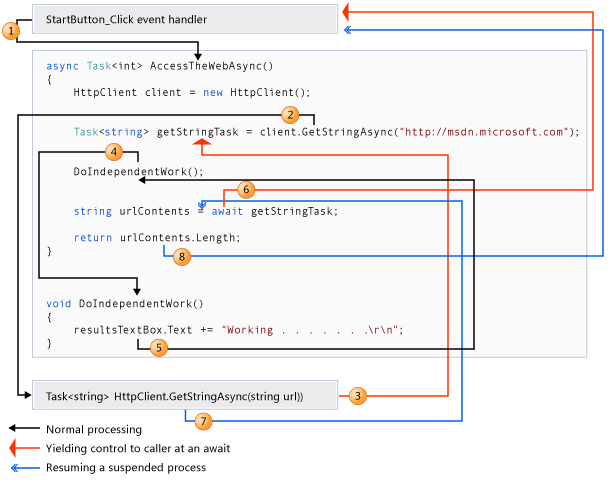
* The method usually includes at least one await expression, which marks a point where the method can't continue until the awaited asynchronous operation is complete. In the meantime, the method is suspended, and control returns to the method's caller. The next section of this topic illustrates what happens at the suspension point.

In async methods, you use the provided keywords and types to indicate what you want to do, and the compiler does the rest, including keeping track of what must happen when control returns to an await point in a suspended method. Some routine processes, such as loops and exception handling, can be difficult to handle in traditional asynchronous code. In an async method, you write these elements much as you would in a synchronous solution, and the problem is solved.

For more information about asynchrony in previous versions of the .NET Framework, see [TPL and Traditional .NET Framework Asynchronous Programming](http://msdn.microsoft.com/en-us/library/dd997423.aspx).

[What Happens in an Async Method](javascript:void(0))

The most important thing to understand in asynchronous programming is how the control flow moves from method to method. The following diagram leads you through the process.



The numbers in the diagram correspond to the following steps.

1. An event handler calls and awaits the AccessTheWebAsync async method.
2. AccessTheWebAsync creates an [HttpClient](http://msdn.microsoft.com/en-us/library/system.net.http.httpclient.aspx) instance and calls the [GetStringAsync](http://msdn.microsoft.com/en-us/library/system.net.http.httpclient.getstringasync.aspx) asynchronous method to download the contents of a website as a string.
3. Something happens in GetStringAsync that suspends its progress. Perhaps it must wait for a website to download or some other blocking activity. To avoid blocking resources, GetStringAsync yields control to its caller, AccessTheWebAsync.

GetStringAsync returns a [Task<TResult>](http://msdn.microsoft.com/en-us/library/dd321424.aspx) where TResult is a string, and AccessTheWebAsync assigns the task to the getStringTask variable. The task represents the ongoing process for the call to GetStringAsync, with a commitment to produce an actual string value when the work is complete.

1. Because getStringTask hasn't been awaited yet, AccessTheWebAsync can continue with other work that doesn't depend on the final result from GetStringAsync. That work is represented by a call to the synchronous method DoIndependentWork.
2. DoIndependentWork is a synchronous method that does its work and returns to its caller.
3. AccessTheWebAsync has run out of work that it can do without a result from getStringTask. AccessTheWebAsync next wants to calculate and return the length of the downloaded string, but the method can't calculate that value until the method has the string.

Therefore, AccessTheWebAsync uses an await operator to suspend its progress and to yield control to the method that called AccessTheWebAsync. AccessTheWebAsync returns a Task(Of Integer) or Task<int> to the caller. The task represents a promise to produce an integer result that's the length of the downloaded string.

|  |
| --- |
| **NoteNote** |
| If GetStringAsync (and therefore getStringTask) is complete before AccessTheWebAsync awaits it, control remains in AccessTheWebAsync. The expense of suspending and then returning to AccessTheWebAsync would be wasted if the called asynchronous process (getStringTask) has already completed and AccessTheWebSync doesn't have to wait for the final result. |

Inside the caller (the event handler in this example), the processing pattern continues. The caller might do other work that doesn't depend on the result from AccessTheWebAsync before awaiting that result, or the caller might await immediately. The event handler is waiting for AccessTheWebAsync, and AccessTheWebAsync is waiting for GetStringAsync.

1. GetStringAsync completes and produces a string result. The string result isn't returned by the call to GetStringAsync in the way that you might expect. (Remember that the method already returned a task in step 3.) Instead, the string result is stored in the task that represents the completion of the method, getStringTask. The await operator retrieves the result from getStringTask. The assignment statement assigns the retrieved result to urlContents.
2. When AccessTheWebAsync has the string result, the method can calculate the length of the string. Then the work of AccessTheWebAsync is also complete, and the waiting event handler can resume. In the full example at the end of the topic, you can confirm that the event handler retrieves and prints the value of the length result.

If you are new to asynchronous programming, take a minute to consider the difference between synchronous and asynchronous behavior. A synchronous method returns when its work is complete (step 5), but an async method returns a task value when its work is suspended (steps 3 and 6). When the async method eventually completes its work, the task is marked as completed and the result, if any, is stored in the task.

For more information about control flow, see [Control Flow in Async Programs (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/hh873191.aspx).

[API Async Methods](javascript:void(0))

You might be wondering where to find methods such as GetStringAsync that support async programming. The .NET Framework 4.5 contains many members that work with async and await. You can recognize these members by the "Async" suffix that’s attached to the member name and 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). For example, the System.IO.Stream class contains methods such as [CopyToAsync](http://msdn.microsoft.com/en-us/library/system.io.stream.copytoasync.aspx), [ReadAsync](http://msdn.microsoft.com/en-us/library/system.io.stream.readasync.aspx), and [WriteAsync](http://msdn.microsoft.com/en-us/library/system.io.stream.writeasync.aspx) alongside the synchronous methods [CopyTo](http://msdn.microsoft.com/en-us/library/system.io.stream.copyto.aspx), [Read](http://msdn.microsoft.com/en-us/library/system.io.stream.read.aspx), and [Write](http://msdn.microsoft.com/en-us/library/system.io.stream.write.aspx).

The Windows Runtime also contains many methods that you can use with async and await in Windows Store apps. For more information and example methods, see [Quickstart: using the await operator for asynchronous programming](http://go.microsoft.com/fwlink/?LinkId=248545), [Asynchronous programming (Windows Store apps)](http://go.microsoft.com/fwlink/?LinkId=259592), and [WhenAny: Bridging between the .NET Framework and the Windows Runtime (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/jj635140.aspx).

[Threads](javascript:void(0))

Async methods are intended to be non-blocking operations. An await expression in an async method doesn’t block the current thread while the awaited task is running. Instead, the expression signs up the rest of the method as a continuation and returns control to the caller of the async method.

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. The method runs on the current synchronization context and uses time on the thread only when the method is active. You can use [Task.Run](http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.run.aspx) to move CPU-bound work to a background thread, but a background thread doesn't help with a process that's just waiting for results to become available.

The async-based approach to asynchronous programming is preferable to existing approaches in almost every case. In particular, this approach is better than [BackgroundWorker](http://msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker.aspx) for IO-bound operations because the code is simpler and you don't have to guard against race conditions. In combination with [Task.Run](http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.run.aspx), async programming is better than [BackgroundWorker](http://msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker.aspx) for CPU-bound operations because async programming separates the coordination details of running your code from the work that Task.Run transfers to the threadpool.

[Async and Await](javascript:void(0))

If you specify that a method is an async method by using an [Async](http://msdn.microsoft.com/en-us/library/hh191564.aspx) or [async](http://msdn.microsoft.com/en-us/library/hh156513.aspx) modifier, you enable the following two capabilities.

* The marked async method can use [Await](http://msdn.microsoft.com/en-us/library/hh156570.aspx) or [await](http://msdn.microsoft.com/en-us/library/hh156528.aspx) to designate suspension points. The await operator tells the compiler that the async method can't continue past that point until the awaited asynchronous process is complete. In the meantime, control returns to the caller of the async method.

The suspension of an async method at an await expression doesn't constitute an exit from the method, and finally blocks don’t run.

* The marked async method can itself be awaited by methods that call it.

An async method typically contains one or more occurrences of an await operator, but the absence of await expressions doesn’t cause a compiler error. If an async method doesn’t use an await operator to mark a suspension point, the method executes as a synchronous method does, despite the async modifier. The compiler issues a warning for such methods.

Async , async, Await, and await are contextual keywords. For more information and examples, see the following topics:

* [Async (Visual Basic)](http://msdn.microsoft.com/en-us/library/hh191564.aspx)
* [async (C# Reference)](http://msdn.microsoft.com/en-us/library/hh156513.aspx)
* [Await Operator (Visual Basic)](http://msdn.microsoft.com/en-us/library/hh156570.aspx)
* [await (C# Reference)](http://msdn.microsoft.com/en-us/library/hh156528.aspx)

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

In .NET Framework programming, an async method typically returns a [Task](http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.aspx) or a [Task<TResult>](http://msdn.microsoft.com/en-us/library/dd321424.aspx). Inside an async method, an await operator is applied to a task that's returned from a call to another async method.

You specify [Task<TResult>](http://msdn.microsoft.com/en-us/library/dd321424.aspx) as the return type if the method 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 that specifies an operand of type TResult.

You use Task as the return type if the method has no return statement or has a return statement that doesn't return an operand.

The following example shows how you declare and call a method that returns a [Task<TResult>](http://msdn.microsoft.com/en-us/library/dd321424.aspx) or a [Task](http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.aspx).

C#

[VB](http://msdn.microsoft.com/en-us/library/hh191443(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

// Signature specifies Task<TResult>

async Task<int> TaskOfTResult\_MethodAsync()

{

int hours;

// . . .

// Return statement specifies an integer result.

return hours;

}

// Calls to TaskOfTResult\_MethodAsync

Task<int> returnedTaskTResult = TaskOfTResult\_MethodAsync();

int intResult = await returnedTaskTResult;

// or, in a single statement

int intResult = await TaskOfTResult\_MethodAsync();

// Signature specifies Task

async Task Task\_MethodAsync()

{

// . . .

// The method has no return statement.

}

// Calls to Task\_MethodAsync

Task returnedTask = Task\_MethodAsync();

await returnedTask;

// or, in a single statement

await Task\_MethodAsync();

Each returned task represents ongoing work. A task encapsulates information about the state of the asynchronous process and, eventually, either the final result from the process or the exception that the process raises if it doesn't succeed.

An async method can also be a Sub method (Visual Basic) or have a void return type (C#). This return type is used primarily to define event handlers, where a void return type is required. Async event handlers often serve as the starting point for async programs.

An async method that’s a Sub procedure or that has a void return type can’t be awaited, and the caller of a void-returning method can't catch any exceptions that the method throws.

An async method can't declare [ByRef](http://msdn.microsoft.com/en-us/library/c84t73c2.aspx) parameters in Visual Basic or [ref](http://msdn.microsoft.com/en-us/library/14akc2c7.aspx) or [out](http://msdn.microsoft.com/en-us/library/t3c3bfhx.aspx) parameters in C#, but the method can call methods that have such parameters.

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

Asynchronous APIs in Windows Runtime programming have one of the following return types, which are similar to tasks:

* [IAsyncOperation](http://go.microsoft.com/fwlink/p/?LinkId=261896), which corresponds to [Task<TResult>](http://msdn.microsoft.com/en-us/library/dd321424.aspx)
* [IAsyncAction](http://go.microsoft.com/fwlink/p/?LinkId=261897), which corresponds to [Task](http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.aspx)
* [IAsyncActionWithProgress](http://go.microsoft.com/fwlink/p/?LinkId=261898)
* [IAsyncOperationWithProgress](http://go.microsoft.com/fwlink/p/?LinkID=259454)

For more information and an example, see [Quickstart: using the await operator for asynchronous programming](http://go.microsoft.com/fwlink/p/?LinkId=248545).

[Naming Convention](javascript:void(0))

By convention, you append "Async" to the names of methods that have an Async or async modifier.

You can ignore the convention where an event, base class, or interface contract suggests a different name. For example, you shouldn’t rename common event handlers, such as Button1\_Click.