# **Control flow in async programs (C#)**

* 07/20/2015
* 9 minutes to read

You can write and maintain asynchronous programs more easily by using the async and await keywords. However, the results might surprise you if you don't understand how your program operates. This topic traces the flow of control through a simple async program to show you when control moves from one method to another and what information is transferred each time.

In general, you mark methods that contain asynchronous code with the [async (C#)](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/async) modifier. In a method that's marked with an async modifier, you can use an [await (C#)](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/await) operator to specify where the method pauses to wait for a called asynchronous process to complete. For more information, see [Asynchronous Programming with async and await (C#)](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/).

The following example uses async methods to download the contents of a specified website as a string and to display the length of the string. The example contains the following two methods.

startButton\_Click, which calls AccessTheWebAsync and displays the result.

AccessTheWebAsync, which downloads the contents of a website as a string and returns the length of the string. AccessTheWebAsync uses an asynchronous [HttpClient](https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient) method, [GetStringAsync(String)](https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.getstringasync" \l "System_Net_Http_HttpClient_GetStringAsync_System_String_), to download the contents.

Numbered display lines appear at strategic points throughout the program to help you understand how the program runs and to explain what happens at each point that is marked. The display lines are labeled "ONE" through "SIX." The labels represent the order in which the program reaches these lines of code.

The following code shows an outline of the program.

C#Copy

public partial class MainWindow : Window

{

// . . .

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

{

// ONE

Task<int> getLengthTask = AccessTheWebAsync();

// FOUR

int contentLength = await getLengthTask;

// SIX

resultsTextBox.Text +=

$"\r\nLength of the downloaded string: {contentLength}.\r\n";

}

async Task<int> AccessTheWebAsync()

{

// TWO

HttpClient client = new HttpClient();

Task<string> getStringTask =

client.GetStringAsync("https://msdn.microsoft.com");

// THREE

string urlContents = await getStringTask;

// FIVE

return urlContents.Length;

}

}

Each of the labeled locations, "ONE" through "SIX," displays information about the current state of the program. The following output is produced:

outputCopy

ONE: Entering startButton\_Click.

Calling AccessTheWebAsync.

TWO: Entering AccessTheWebAsync.

Calling HttpClient.GetStringAsync.

THREE: Back in AccessTheWebAsync.

Task getStringTask is started.

About to await getStringTask & return a Task<int> to startButton\_Click.

FOUR: Back in startButton\_Click.

Task getLengthTask is started.

About to await getLengthTask -- no caller to return to.

FIVE: Back in AccessTheWebAsync.

Task getStringTask is complete.

Processing the return statement.

Exiting from AccessTheWebAsync.

SIX: Back in startButton\_Click.

Task getLengthTask is finished.

Result from AccessTheWebAsync is stored in contentLength.

About to display contentLength and exit.

Length of the downloaded string: 33946.

## **Set up the program**

You can download the code that this topic uses from MSDN, or you can build it yourself.

**Note**

To run the example, you must have Visual Studio 2012 or newer and the .NET Framework 4.5 or newer installed on your computer.

### **Download the program**

You can download the application for this topic from [Async Sample: Control Flow in Async Programs](https://code.msdn.microsoft.com/Async-Sample-Control-Flow-5c804fc0). The following steps open and run the program.

Unzip the downloaded file, and then start Visual Studio.

On the menu bar, choose ****File**** > ****Open**** > ****Project/Solution****.

Navigate to the folder that holds the unzipped sample code, open the solution (.sln) file, and then choose the ****F5**** key to build and run the project.

### **Create the program Yourself**

The following Windows Presentation Foundation (WPF) project contains the code example for this topic.

To run the project, perform the following steps:

Start Visual Studio.

On the menu bar, choose ****File**** > ****New**** > ****Project****.

The ****New Project**** dialog box opens.

Choose the ****Installed**** > ****Visual C#**** > ****Windows Desktop**** category, and then choose ****WPF App**** from the list of project templates.

Enter AsyncTracer as the name of the project, and then choose the ****OK**** button.

The new project appears in ****Solution Explorer****.

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

If the tab isn’t visible, open the shortcut menu for MainWindow.xaml in ****Solution Explorer****, and then choose ****View Code****.

In the ****XAML**** view of MainWindow.xaml, replace the code with the following code.

C#Copy

<Window

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" mc:Ignorable="d" x:Class="AsyncTracer.MainWindow"

Title="Control Flow Trace" Height="350" Width="592">

<Grid>

<Button x:Name="startButton" Content="Start

" HorizontalAlignment="Left" Margin="250,10,0,0" VerticalAlignment="Top" Width="75" Height="24" Click="startButton\_Click" d:LayoutOverrides="GridBox"/>

<TextBox x:Name="resultsTextBox" HorizontalAlignment="Left" TextWrapping="Wrap" VerticalAlignment="Bottom" Width="576" Height="265" FontFamily="Lucida Console" FontSize="10" VerticalScrollBarVisibility="Visible" Grid.ColumnSpan="3"/>

</Grid>

</Window>

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

Add a reference for [System.Net.Http](https://docs.microsoft.com/en-us/dotnet/api/system.net.http).

In ****Solution Explorer****, open the shortcut menu for MainWindow.xaml.cs, and then choose ****View Code****.

In MainWindow.xaml.cs, replace the code with the following code.

C#Copy

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;

// Add a using directive and a reference for System.Net.Http;using System.Net.Http;

namespace AsyncTracer

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

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

{

// The display lines in the example lead you through the control shifts.

resultsTextBox.Text += "ONE: Entering startButton\_Click.\r\n" +

" Calling AccessTheWebAsync.\r\n";

Task<int> getLengthTask = AccessTheWebAsync();

resultsTextBox.Text += "\r\nFOUR: Back in startButton\_Click.\r\n" +

" Task getLengthTask is started.\r\n" +

" About to await getLengthTask -- no caller to return to.\r\n";

int contentLength = await getLengthTask;

resultsTextBox.Text += "\r\nSIX: Back in startButton\_Click.\r\n" +

" Task getLengthTask is finished.\r\n" +

" Result from AccessTheWebAsync is stored in contentLength.\r\n" +

" About to display contentLength and exit.\r\n";

resultsTextBox.Text +=

$"\r\nLength of the downloaded string: {contentLength}.\r\n";

}

async Task<int> AccessTheWebAsync()

{

resultsTextBox.Text += "\r\nTWO: Entering AccessTheWebAsync.";

// Declare an HttpClient object.

HttpClient client = new HttpClient();

resultsTextBox.Text += "\r\n Calling HttpClient.GetStringAsync.\r\n";

// GetStringAsync returns a Task<string>.

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

resultsTextBox.Text += "\r\nTHREE: Back in AccessTheWebAsync.\r\n" +

" Task getStringTask is started.";

// AccessTheWebAsync can continue to work until getStringTask is awaited.

resultsTextBox.Text +=

"\r\n About to await getStringTask and return a Task<int> to startButton\_Click.\r\n";

// Retrieve the website contents when task is complete.

string urlContents = await getStringTask;

resultsTextBox.Text += "\r\nFIVE: Back in AccessTheWebAsync." +

"\r\n Task getStringTask is complete." +

"\r\n Processing the return statement." +

"\r\n Exiting from AccessTheWebAsync.\r\n";

return urlContents.Length;

}

}

}

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

The following output appears:

outputCopy

ONE: Entering startButton\_Click.

Calling AccessTheWebAsync.

TWO: Entering AccessTheWebAsync.

Calling HttpClient.GetStringAsync.

THREE: Back in AccessTheWebAsync.

Task getStringTask is started.

About to await getStringTask & return a Task<int> to startButton\_Click.

FOUR: Back in startButton\_Click.

Task getLengthTask is started.

About to await getLengthTask -- no caller to return to.

FIVE: Back in AccessTheWebAsync.

Task getStringTask is complete.

Processing the return statement.

Exiting from AccessTheWebAsync.

SIX: Back in startButton\_Click.

Task getLengthTask is finished.

Result from AccessTheWebAsync is stored in contentLength.

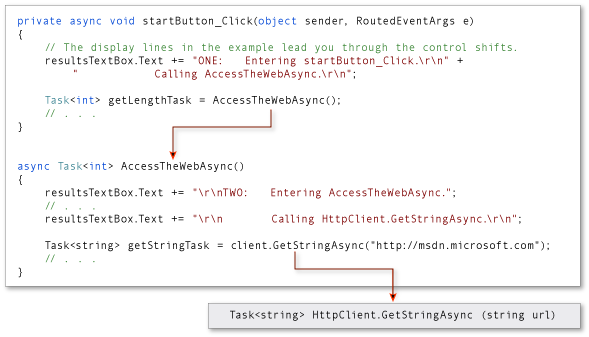
About to display contentLength and exit.

Length of the downloaded string: 33946.

## **Trace the program**

### **Steps ONE and TWO**

The first two display lines trace the path as startButton\_Click calls AccessTheWebAsync, and AccessTheWebAsync calls the asynchronous [HttpClient](https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient) method [GetStringAsync(String)](https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.getstringasync" \l "System_Net_Http_HttpClient_GetStringAsync_System_String_). The following image outlines the calls from method to method.



The return type of both AccessTheWebAsync and client.GetStringAsync is [Task<TResult>](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task-1). For AccessTheWebAsync, TResult is an integer. For GetStringAsync, TResult is a string. For more information about async method return types, see [Async Return Types (C#)](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/async-return-types).

A task-returning async method returns a task instance when control shifts back to the caller. Control returns from an async method to its caller either when an await operator is encountered in the called method or when the called method ends. The display lines that are labeled "THREE" through "SIX" trace this part of the process.

### **Step THREE**

In AccessTheWebAsync, the asynchronous method [GetStringAsync(String)](https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.getstringasync" \l "System_Net_Http_HttpClient_GetStringAsync_System_String_) is called to download the contents of the target webpage. Control returns from client.GetStringAsync to AccessTheWebAsync when client.GetStringAsync returns.

The client.GetStringAsync method returns a task of string that’s assigned to the getStringTask variable in AccessTheWebAsync. The following line in the example program shows the call to client.GetStringAsync and the assignment.

C#Copy

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

You can think of the task as a promise by client.GetStringAsync to produce an actual string eventually. In the meantime, if AccessTheWebAsync has work to do that doesn't depend on the promised string from client.GetStringAsync, that work can continue while client.GetStringAsync waits. In the example, the following lines of output, which are labeled "THREE," represent the opportunity to do independent work

outputCopy

THREE: Back in AccessTheWebAsync.

Task getStringTask is started.

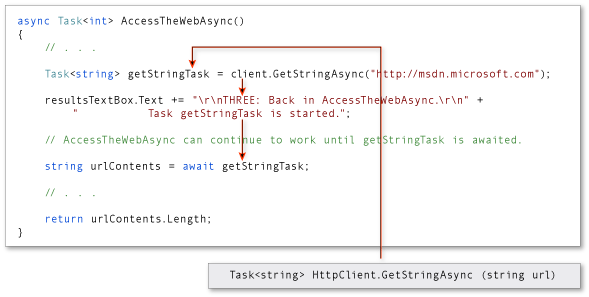
About to await getStringTask & return a Task<int> to startButton\_Click.

The following statement suspends progress in AccessTheWebAsync when getStringTask is awaited.

C#Copy

string urlContents = await getStringTask;

The following image shows the flow of control from client.GetStringAsync to the assignment to getStringTask and from the creation of getStringTask to the application of an await operator.



The await expression suspends AccessTheWebAsync until client.GetStringAsync returns. In the meantime, control returns to the caller of AccessTheWebAsync, startButton\_Click.

**Note**

Typically, you await the call to an asynchronous method immediately. For example, the following assignment could replace the previous code that creates and then awaits getStringTask: string urlContents = await client.GetStringAsync("https://msdn.microsoft.com");

In this topic, the await operator is applied later to accommodate the output lines that mark the flow of control through the program.

### **Step FOUR**

The declared return type of AccessTheWebAsync is Task<int>. Therefore, when AccessTheWebAsync is suspended, it returns a task of integer to startButton\_Click. You should understand that the returned task isn’t getStringTask. The returned task is a new task of integer that represents what remains to be done in the suspended method, AccessTheWebAsync. The task is a promise from AccessTheWebAsync to produce an integer when the task is complete.

The following statement assigns this task to the getLengthTask variable.

C#Copy

Task<int> getLengthTask = AccessTheWebAsync();

As in AccessTheWebAsync, startButton\_Click can continue with work that doesn’t depend on the results of the asynchronous task (getLengthTask) until the task is awaited. The following output lines represent that work.

outputCopy

FOUR: Back in startButton\_Click.

Task getLengthTask is started.

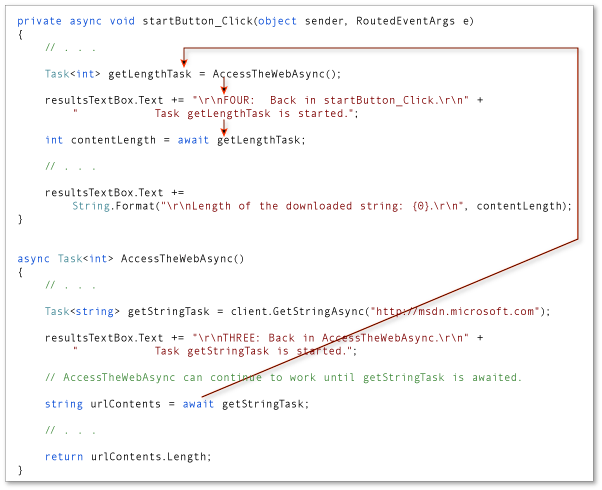
About to await getLengthTask -- no caller to return to.

Progress in startButton\_Click is suspended when getLengthTask is awaited. The following assignment statement suspends startButton\_Click until AccessTheWebAsync is complete.

C#Copy

int contentLength = await getLengthTask;

In the following illustration, the arrows show the flow of control from the await expression in AccessTheWebAsync to the assignment of a value to getLengthTask, followed by normal processing in startButton\_Click until getLengthTask is awaited.



### **Step FIVE**

When client.GetStringAsync signals that it’s complete, processing in AccessTheWebAsync is released from suspension and can continue past the await statement. The following lines of output represent the resumption of processing.

outputCopy

FIVE: Back in AccessTheWebAsync.

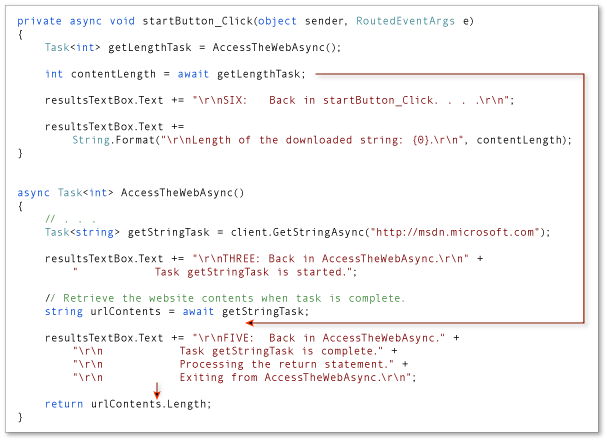
Task getStringTask is complete.

Processing the return statement.

Exiting from AccessTheWebAsync.

The operand of the return statement, urlContents.Length, is stored in the task that AccessTheWebAsync returns. The await expression retrieves that value from getLengthTask in startButton\_Click.

The following image shows the transfer of control after client.GetStringAsync (and getStringTask) are complete.



AccessTheWebAsync runs to completion, and control returns to startButton\_Click, which is awaiting the completion.

### **Step SIX**

When AccessTheWebAsync signals that it’s complete, processing can continue past the await statement in startButton\_Async. In fact, the program has nothing more to do.

The following lines of output represent the resumption of processing in startButton\_Async:

outputCopy

SIX: Back in startButton\_Click.

Task getLengthTask is finished.

Result from AccessTheWebAsync is stored in contentLength.

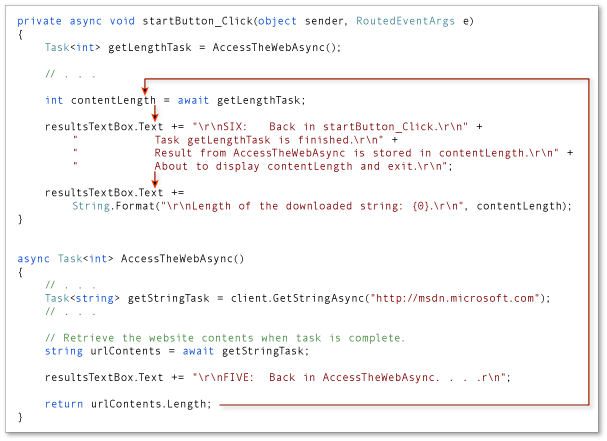
About to display contentLength and exit.

The await expression retrieves from getLengthTask the integer value that’s the operand of the return statement in AccessTheWebAsync. The following statement assigns that value to the contentLength variable.

C#Copy

int contentLength = await getLengthTask;

The following image shows the return of control from AccessTheWebAsync to startButton\_Click.



## **See also**

* [Asynchronous Programming with async and await (C#)](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/)
* [Async Return Types (C#)](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/async-return-types)
* [Walkthrough: Accessing the Web by Using async and await (C#)](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/walkthrough-accessing-the-web-by-using-async-and-await)
* [Async Sample: Control Flow in Async Programs (C# and Visual Basic)](https://code.msdn.microsoft.com/Async-Sample-Control-Flow-5c804fc0)

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/control-flow-in-async-programs>