# Cancel a list of tasks

Article • 03/19/2025

You can cancel an async console application if you don't want to wait for it to finish. By following the example in this topic, you can add a cancellation to an application that downloads the contents of a list of websites. You can cancel many tasks by associating the CancellationTokenSource instance with each task. If you select the `Enter` key, you cancel all tasks that aren't yet complete.

This tutorial covers:

- ✓ Creating a .NET console application
- ✓ Writing an async application that supports cancellation
- ✓ Demonstrating signaling cancellation

## Prerequisites

- The latest .NET SDK
- Visual Studio Code editor
- The C# DevKit

### Create example application

Create a new .NET Core console application. You can create one by using the dotnet new console command or from Visual Studio. Open the *Program.cs* file in your favorite code editor.

### Replace using directives

Replace the existing `using` directives with these declarations:

C#

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;
```

# Add fields

In the `Program` class definition, add these three fields:

C#

```
static readonly CancellationTokenSource s_cts = new
CancellationTokenSource();

static readonly HttpClient s_client = new HttpClient
{
    MaxResponseContentBufferSize = 1_000_000
};

static readonly IEnumerable<string> s_urlList = new string[]
{
    "https://learn.microsoft.com",
    "https://learn.microsoft.com/aspnet/core",
    "https://learn.microsoft.com/azure",
    "https://learn.microsoft.com/azure/devops",
    "https://learn.microsoft.com/dotnet",
    "https://learn.microsoft.com/dynamics365",
    "https://learn.microsoft.com/education",
    "https://learn.microsoft.com/enterprise-mobility-security",
    "https://learn.microsoft.com/gaming",
    "https://learn.microsoft.com/graph",
    "https://learn.microsoft.com/microsoft-365",
    "https://learn.microsoft.com/office",
    "https://learn.microsoft.com/powershell",
    "https://learn.microsoft.com/sql",
    "https://learn.microsoft.com/surface",
    "https://learn.microsoft.com/system-center",
    "https://learn.microsoft.com/visualstudio",
    "https://learn.microsoft.com/windows",
    "https://learn.microsoft.com/maui"
};
```

The CancellationTokenSource is used to signal a requested cancellation to a CancellationToken. The `HttpClient` exposes the ability to send HTTP requests and receive HTTP responses. The `s_urlList` holds all of the URLs that the application plans to process.

# Update application entry point

The main entry point into the console application is the `Main` method. Replace the existing method with the following:

C#

```
static async Task Main()
{
    Console.WriteLine("Application started.");
    Console.WriteLine("Press the ENTER key to cancel...\n");

    Task cancelTask = Task.Run(() =>
    {
        while (Console.ReadKey().Key != ConsoleKey.Enter)
        {
            Console.WriteLine("Press the ENTER key to cancel...");
        }

        Console.WriteLine("\nENTER key pressed: cancelling downloads.\n");
        s_cts.Cancel();
    });

    Task sumPageSizesTask = SumPageSizesAsync();

    Task finishedTask = await Task.WhenAny(new[] { cancelTask,
sumPageSizesTask });
    if (finishedTask == cancelTask)
    {
        // wait for the cancellation to take place:
        try
        {
            await sumPageSizesTask;
            Console.WriteLine("Download task completed before cancel request
was processed.");
        }
        catch (TaskCanceledException)
        {
            Console.WriteLine("Download task has been cancelled.");
        }
    }

    Console.WriteLine("Application ending.");
}
```

The updated `Main` method is now considered an Async main, which allows for an asynchronous entry point into the executable. It writes a few instructional messages to the console, then declares a Task instance named `cancelTask`, which will read console key strokes. If the `Enter` key is pressed, a call to CancellationTokenSource.Cancel() is made. This will signal cancellation. Next, the `sumPageSizesTask` variable is assigned from the `SumPageSizesAsync` method. Both tasks are then passed to Task.WhenAny(Task[]), which will continue when any of the two tasks have completed.

The next block of code ensures that the application doesn't exit until the cancellation has been processed. If the first task to complete is the `cancelTask`, the `sumPageSizeTask` is awaited. If it was cancelled, when awaited it throws a

System.Threading.Tasks.TaskCanceledException. The block catches that exception, and prints a message.

## Create the asynchronous sum page sizes method

Below the `Main` method, add the `SumPageSizesAsync` method:

```
C#

static async Task SumPageSizesAsync()
{
    var stopwatch = Stopwatch.StartNew();

    int total = 0;
    foreach (string url in s_urlList)
    {
        int contentLength = await ProcessUrlAsync(url, s_client,
s_cts.Token);
        total += contentLength;
    }

    stopwatch.Stop();

    Console.WriteLine($"\nTotal bytes returned:  {total:#,#}");
    Console.WriteLine($"Elapsed time:          {stopwatch.Elapsed}\n");
}
```

The method starts by instantiating and starting a Stopwatch. It then loops through each URL in the `s_urlList` and calls `ProcessUrlAsync`. With each iteration, the `s_cts.Token` is passed into the `ProcessUrlAsync` method and the code returns a Task<TResult>, where `TResult` is an integer:

```
C#

int total = 0;
foreach (string url in s_urlList)
{
    int contentLength = await ProcessUrlAsync(url, s_client, s_cts.Token);
    total += contentLength;
}
```

## Add process method

Add the following `ProcessUrlAsync` method below the `SumPageSizesAsync` method:

C#

```
static async Task<int> ProcessUrlAsync(string url, HttpClient client,
CancellationToken token)
{
    HttpResponseMessage response = await client.GetAsync(url, token);
    byte[] content = await response.Content.ReadAsByteArrayAsync(token);
    Console.WriteLine($"{url,-60} {content.Length,10:#,#}");

    return content.Length;
}
```

For any given URL, the method will use the `client` instance provided to get the response as a `byte[]`. The CancellationToken instance is passed into the HttpClient.GetAsync(String, CancellationToken) and HttpContent.ReadAsByteArrayAsync() methods. The `token` is used to register for requested cancellation. The length is returned after the URL and length is written to the console.

## Example application output

Console

```
Application started.
Press the ENTER key to cancel...

https://learn.microsoft.com                                   37,357
https://learn.microsoft.com/aspnet/core                       85,589
https://learn.microsoft.com/azure                            398,939
https://learn.microsoft.com/azure/devops                      73,663
https://learn.microsoft.com/dotnet                            67,452
https://learn.microsoft.com/dynamics365                       48,582
https://learn.microsoft.com/education                         22,924

ENTER key pressed: cancelling downloads.

Application ending.
```

# Complete example

The following code is the complete text of the *Program.cs* file for the example.

C#

```
using System.Diagnostics;
```

```
class Program
{
    static readonly CancellationTokenSource s_cts = new
CancellationTokenSource();

    static readonly HttpClient s_client = new HttpClient
    {
        MaxResponseContentBufferSize = 1_000_000
    };

    static readonly IEnumerable<string> s_urlList = new string[]
    {
            "https://learn.microsoft.com",
            "https://learn.microsoft.com/aspnet/core",
            "https://learn.microsoft.com/azure",
            "https://learn.microsoft.com/azure/devops",
            "https://learn.microsoft.com/dotnet",
            "https://learn.microsoft.com/dynamics365",
            "https://learn.microsoft.com/education",
            "https://learn.microsoft.com/enterprise-mobility-security",
            "https://learn.microsoft.com/gaming",
            "https://learn.microsoft.com/graph",
            "https://learn.microsoft.com/microsoft-365",
            "https://learn.microsoft.com/office",
            "https://learn.microsoft.com/powershell",
            "https://learn.microsoft.com/sql",
            "https://learn.microsoft.com/surface",
            "https://learn.microsoft.com/system-center",
            "https://learn.microsoft.com/visualstudio",
            "https://learn.microsoft.com/windows",
            "https://learn.microsoft.com/maui"
    };

    static async Task Main()
    {
        Console.WriteLine("Application started.");
        Console.WriteLine("Press the ENTER key to cancel...\n");

        Task cancelTask = Task.Run(() =>
        {
            while (Console.ReadKey().Key != ConsoleKey.Enter)
            {
                Console.WriteLine("Press the ENTER key to cancel...");
            }

            Console.WriteLine("\nENTER key pressed: cancelling
downloads.\n");
            s_cts.Cancel();
        });

        Task sumPageSizesTask = SumPageSizesAsync();

        Task finishedTask = await Task.WhenAny(new[] { cancelTask,
sumPageSizesTask });
        if (finishedTask == cancelTask)
```

```
        {
            // wait for the cancellation to take place:
            try
            {
                await sumPageSizesTask;
                Console.WriteLine("Download task completed before cancel
request was processed.");
            }
            catch (OperationCanceledException)
            {
                Console.WriteLine("Download task has been cancelled.");
            }
        }

        Console.WriteLine("Application ending.");
    }

    static async Task SumPageSizesAsync()
    {
        var stopwatch = Stopwatch.StartNew();

        int total = 0;
        foreach (string url in s_urlList)
        {
            int contentLength = await ProcessUrlAsync(url, s_client,
s_cts.Token);
            total += contentLength;
        }

        stopwatch.Stop();

        Console.WriteLine($"\nTotal bytes returned:  {total:#,#}");
        Console.WriteLine($"Elapsed time:          {stopwatch.Elapsed}\n");
    }

    static async Task<int> ProcessUrlAsync(string url, HttpClient client,
CancellationToken token)
    {
        HttpResponseMessage response = await client.GetAsync(url, token);
        byte[] content = await response.Content.ReadAsByteArrayAsync(token);
        Console.WriteLine($"{url,-60} {content.Length,10:#,#}");

        return content.Length;
    }
}
```

## See also

- CancellationToken
- CancellationTokenSource
- Asynchronous programming with async and await (C#)