**Delegates, Events, and Lambdas**

**Creating a Web File Downloader**

1. Change to the **Chapter03** folder and create a new console app, called **Activity01**, using the CLI **dotnet** command:

source\Chapter03>dotnet new console -o Activity01

1. Open **Chapter03\ Activity01.csproj** and replace the entire file with these settings:

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net6.0</TargetFramework>

</PropertyGroup>

</Project>

1. Open **Activity01\Program.cs** and clear the contents.
2. You will need to include the following **using** statements:

using System;

using System.IO;

using System.Net;

using System.Threading;

1. You will use **Chapter03.Activity01** as the namespace:

namespace Workshop3.Activity

{

1. Add the progress reporting class (**DownloadProgressChangedEventArgs** was suggested as a name) to help report the progress percentage and  
   bytes received:

public class DownloadProgressChangedEventArgs

{

public DownloadProgressChangedEventArgs(int

progressPercentage, long bytesReceived)

{

ProgressPercentage = progressPercentage;

BytesReceived = bytesReceived;

}

public long BytesReceived { get; init; }

public int ProgressPercentage { get; init;}

}

1. Now add the **WebClientAdapter** class (which hides the internal usage of **WebClient**):

public class WebClientAdapter

{

public event EventHandler DownloadCompleted;

public event EventHandler<DownloadProgressChangedEventArgs> DownloadProgressChanged;

public event EventHandler<string> InvalidUrlRequested;

This will act as a **publisher**, so you need to add the three events. Notice that **DownloadCompleted** is based on **EventHandler**, as you do not need to pass any extra information when publishing the event, whereas the other two events do include extra details.

1. Add a **DownloadFile** method is added, which is passed a **url** string (as entered by the user) and a destination filename:

public IDisposable DownloadFile(string url, string

destination)

{

if (!Uri.TryCreate(url, UriKind.Absolute, out var uri))

{

InvalidUrlRequested?.Invoke(this, url);

return null;

}

You need to convert the **url** string into a **Uri** class using **Uri.TryCreate**. You have specified **UriKind.Absolute**, as you have a fully qualified web address. If this call fails, you invoke the **InvalidUrlRequested** event.

1. Create a **WebClient** instance used to initiate the actual file download:

var client = new WebClient();

**WebClient** has a **DownloadFileCompleted** event that you subscribe to, to indicate when the download is complete.

1. Define a lambda, which you use to publish your own **DownloadCompleted** event. Remember, you are trying to hide your internal use of **WebClient** with your own events:

client.DownloadFileCompleted += (sender, args) =>

DownloadCompleted?.Invoke(this, EventArgs.Empty);

1. The **WebClient** class' **DownloadProgressChanged** event is subscribed to so that you are given progress updates, which you adapt and publish via your event of the same name:

client.DownloadProgressChanged += (sender, args) =>

DownloadProgressChanged?.Invoke(this,

new DownloadProgressChangedEventArgs(args.

ProgressPercentage, args.BytesReceived));

1. Finally, call **WebClient.DownloadFileAsync**, passing in the **URI** and the file destination to start the download reque

client.DownloadFileAsync(uri, destination);

return client;

}

}

1. Now add the console app that uses **WebClientAdapter**. Define a new **Program** class with the standard **void Main** entry point:

public class Program

{

public static void Main()

{

1. Add a **do** loop that prompts for a URL to be entered:

string input;

do {

Console.WriteLine("Enter a URL:");

input = Console.ReadLine();

if (!string.IsNullOrEmpty(input))

1. Create a temporary filename to store the download based on the source URL, so you extract the last **/** symbol, which is appended to **Path.GetTempPath**:

{

string destination;

var lastSlash = input.LastIndexOf("/");

if (lastSlash > -1)

{

destination = Path.Join(Path.GetTempPath(),

input.Substring(lastSlash + 1));

} else {

destination = Path.GetTempFileName();

}

If there is no trailing **/** symbol, you ask for a filename using **Path. GetTempFileName**.

1. Pass the input URL and destination filename to a **Download** method. Define that next:

Download(input, destination);

}

} while (input != string.Empty);

}

1. Next make the **Download** method create a new instance of **WebClientAdapter**:

private static void Download(string url, string destination)

{

var client = new WebClientAdapter();

You need a way to wait for the requested download to finish before you leave this **Download** method.

1. Now use the **ManualResetEventSlim** to signal that an operation has finished:

var waiter = new ManualResetEventSlim();

using (waiter)

{

As with **WebClient**, the **ManualResetEventSlim** class implements the **IDisposible** interface, which means it is effective to wrap it in a **using** statement to ensure that memory and other resources are cleaned up after use.

1. Subscribe the **WebClientAdaptor.InvalidUrlRequested** event to using a lambda:

client.InvalidUrlRequested += (sender, args) =>

{

var oldColor = Console.BackgroundColor;

Console.BackgroundColor = ConsoleColor.DarkRed;

Console.WriteLine($"Invalid URL {args}");

Console.BackgroundColor = oldColor;

};

This makes a note of the current console background color, which temporarily changes to red before writing a warning message.

1. Subscribe to the **DownloadProgressChanged** event, so that you can offer progress reports as the download progresses:

client.DownloadProgressChanged += (sender, args) =>

{

Console.WriteLine( $"Downloading...{args.

ProgressPercentage}% complete ({args.BytesReceived:N0} bytes)");

};

1. For the final event, call **waiter.Set**, which will signal to the **waiter** that you have finished:

client.DownloadCompleted += (sender, args) =>

{

Console.WriteLine($"Downloaded to {destination});

waiter.Set();

};

1. Next call **WebClientAdapter.DownloadFile**, and you are returned another **IDisposable** object:

Console.WriteLine($"Downloading {url}…”);

var request = client.DownloadFile(url, destination);

if (request == null)

return;

If the web address turns out to be an invalid one, then this will be null, so you do not need to proceed.

1. Wrap the **request** object in a **using** statement:

using (request)

{

if (!waiter.Wait(TimeSpan.FromSeconds(10D)))

{

Console.WriteLine($"Timedout downloading {url}");  
 }

}

}

}

}

}

This helps you to call **waiter.Wait** to pause for up to 10 seconds or until you receive a **waiter.Set** call triggered via the download event.

1. Run the console app with various download requests to produce this output:

Enter a URL:

https://www1.ncdc.noaa.gov/pub/data/swdi/stormevents/csvfiles/

StormEvents\_details-ftp\_v1.0\_d1950\_c20170120.csv.gz

Downloading https://www1.ncdc.noaa.gov/pub/data/swdi/stormevents/

csvfiles/StormEvents\_details-ftp\_v1.0\_d1950\_c20170120.csv.gz...

Downloading...73% complete (7,758 bytes)

Downloading...77% complete (8,192 bytes)

Downloading...100% complete (10,597 bytes)

Downloaded to C:\Temp\StormEvents\_details-ftp\_v1.0\_d1950\_c20170120.

csv.gz

Enter a URL:

https://www1.ncdc.noaa.gov/pub/data/swdi/stormevents/csvfiles/

StormEvents\_details-ftp\_v1.0\_d1954\_c20160223.csv.gz

Downloading https://www1.ncdc.noaa.gov/pub/data/swdi/stormevents/

csvfiles/StormEvents\_details-ftp\_v1.0\_d1954\_c20160223.csv.gz...

Downloading...29% complete (7,758 bytes)

Downloading...31% complete (8,192 bytes)

Downloading...54% complete (14,238 bytes)

Downloading...62% complete (16,384 bytes)

Downloading...84% complete (22,238 bytes)

Downloading...93% complete (24,576 bytes)

In this activity, you planned to investigate patterns in US storm events. To do this, you downloaded storm event datasets from online sources for later analysis. The National Oceanic and Atmospheric Administration were accessed from https://www1.ncdc.noaa. gov/pub/data/swdi/stormevents/csvfiles.

You created a .NET Core console app that allowed a web address to be entered, the contents of which are downloaded to a local disk. To be as user-friendly as possible, the application used events that signal when an invalid address is entered, the progress of a download, and when it completes