**Topic 2 – Introduction to File IO**

**Objective 1 : To work with directories. Write and read to/from a Text File**

**Objective 2: To Read an XML File and display to screen**

**Objective 3 : To Read a JSON File and convert to Memory objects.**

**Objective 4: To Download a File from Internet and Save to Disk**

The System.IO namespace has various classes that are used for performing numerous operations with files and directories, such as creating and deleting files, reading from or writing to a file, setting directory information.

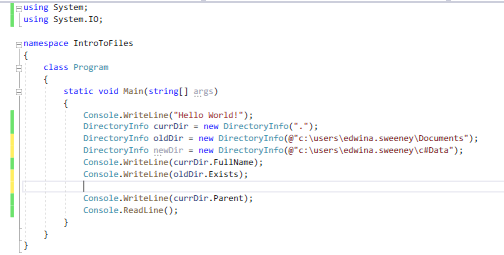
**Tip: Have a look at the Microsoft Docs and all the classes available under System.IO.**

**Objective 1 – Looking at the DirectoryInfo Class**

Create a console application called IntroToFiles and add the following code;

The following allows us to gather information regarding our directories using the System.IO namespace and the DirectoryInfo class.

<https://docs.microsoft.com/en-us/dotnet/api/system.io.directoryinfo?view=net-5.0>



Task 1 -Create a method which will read in a directory name and then create that directory in the current location. Write the name of the directory to standard output. Show code and output of program in box below.

public static void ReadAndWrite(DirectoryInfo currDir)

{

Console.WriteLine("Enter Directory Name:");

string newDir = Console.ReadLine();

if (!System.IO.Directory.Exists(currDir.FullName + newDir))

System.IO.Directory.CreateDirectory(currDir.FullName + @"\" + newDir);

else

Console.WriteLine("Directory already exists.");

}

**Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated**

Writing to files and reading from files requires an encoding and decoding mechanism.

**Task 2 -** What encoding mechanism is used widely today?

ASCII,BinHex,Unicode,UTF-8, UTF-16 are some examples.

The most common would be UTF-8 because it can handle emojis and Japanese characters etc

**Objective 2 : Reading and Writing Text Files using System.IO Namespace**

StreamReader and StreamWriter are found in the System.IO namespace. Both classes are useful when you want to read or write character-based data. Each class returns either a StreamReader or a StreamWriter object which is an idisposable object which must be managed by the application.

Diagram

Description automatically generated

Continuing on using the same project, add the following static method;

Writing to a new file using StreamWriter, we use the WriteLineAsync method to write strings to the file stream.

Timeline

Description automatically generated

The ***using*** statement is a very useful and allows us to write to a streamWriter object and then dispose when completed.

Adjust this method and apply the TAP model to make this an asynchronous method instead.

**Task 3**

public async static void CreateFile(DirectoryInfo currDir)

{

Console.WriteLine("Enter File Name:");

var filename = Console.ReadLine();

try

{

if (filename != null)

{

FileStream fs = new FileStream(filename, FileMode.OpenOrCreate);

using (StreamWriter writer = new StreamWriter(fs, Encoding.UTF8))

{

await writer.WriteLineAsync("this is my new file");

await writer.WriteLineAsync("These string characters will be encoded");

await writer.WriteLineAsync("The end of my file");

}

}

else

Console.WriteLine("File was not created.");

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

}

**Reading from a file using the StreamReader class**

For this example, we will use the StreamReader class and the ReadLineAsync method until it reaches null or end of file. StreamReader will read from the Filestream until it reaches the end of file.

Graphical user interface, text, application, email

Description automatically generated

Call this method from the Main method using code below;

**Graphical user interface, text, application

Description automatically generated**

There are numerous ways to open a file for reading and writing through the File class and System.IO namespace.

However, these do not work for Windows Universal Apps. It is recommended that you use the Windows Storage Namespace.

**Objective 2: Working with XML Files using XDocument and XElement Classes**

XML Files

XML is short for eXtensible Markup Language. It is a very widely used format for exchanging data, mainly because it's easily readable for both humans and machines. XML is made up of tags, attributes and values. The C# language has built-in functions or classes to deal with XML files and its contents.

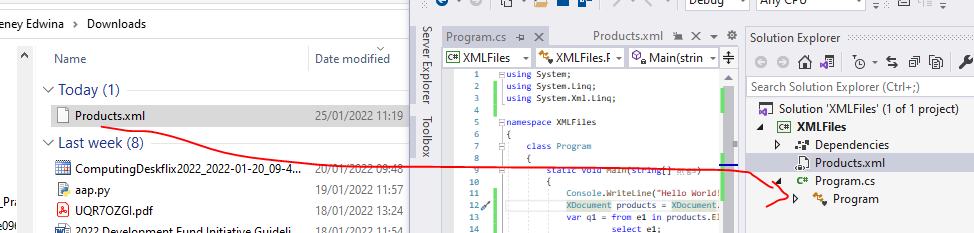
There are mainly two methods for reading XML with C#: The XmlDocument class and the XmlReader class. **XmlDocument** reads the entire XML content into memory and then you can navigate through all the elements.

**XmlReader** reads each element into memory one at a time. Very useful for very large XML files where you don’t want to consume too much memory.

**Task 1 Create a new a static method to existing project.**

Download XML file Products.xml from BB into the Downloads folder.

Drag file from Downloads folder to Project solution and change properties of file to ‘Copy if newer’.



Insert the following code; We will load an XML from local disk and display all elements on screen. We also need the using System.Xml.Linq library to use the XML classes.

Graphical user interface, text, application

Description automatically generated

At the top, we have Catalog which is the **Root** element. Data for all Catalog Products can be stored in child elements. Products is an element which has child elements of **id**, **name** etc. Season is an **attribute** of Product. **Season** could also be added as a child element. There are no hard rules when adding data to an XML document.



Lets print only the root node of the document.

Graphical user interface, text, application, email

Description automatically generated

Text

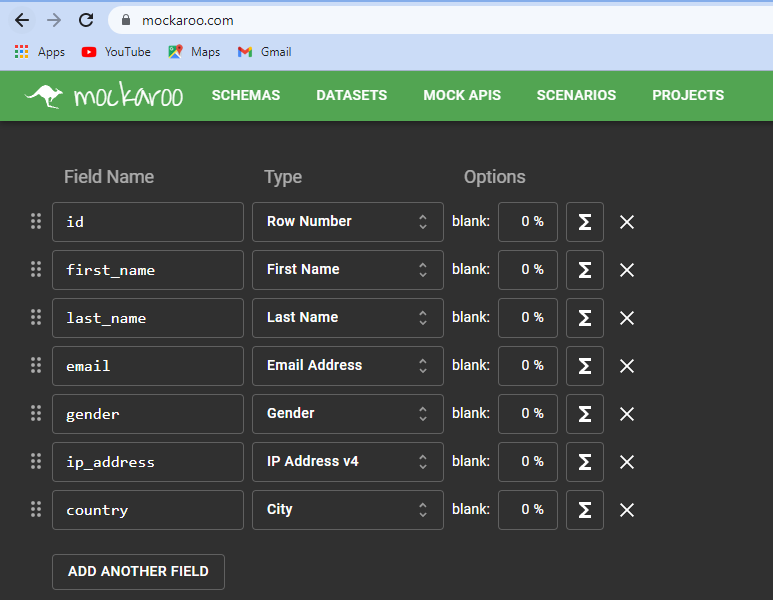
Description automatically generated

**Task 4** – Investigate what is the best approach for reading very large XML files into memory. Place your answer below.

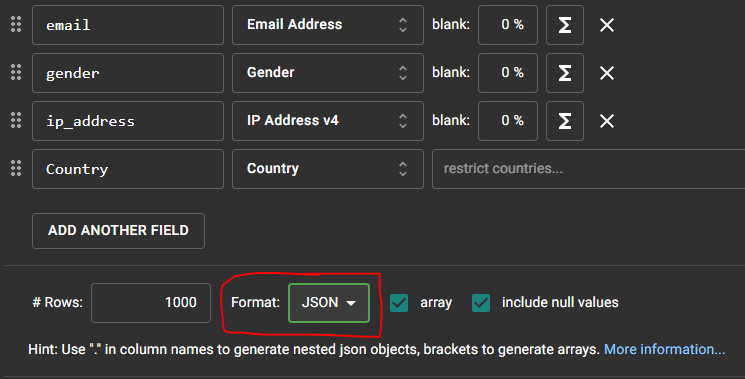
The best approach is to use StreamReader so as not to populate the app with to much data as that could freeze the app or take up to much resources.

**Objective 3: Working with JSON Files**

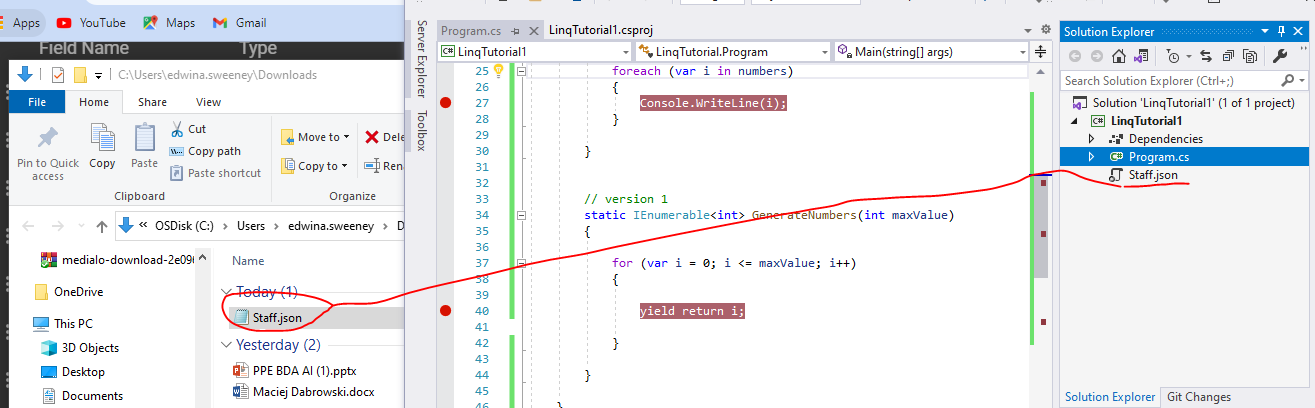
Go to Mockaroo.com and generate a JSON data file which resembles that of staff members.



Add field country and select Country as field type (not city as shown above). Select JSON as the format of the file.



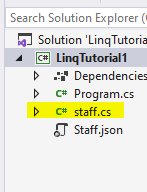
Download data, and rename file as staff.json



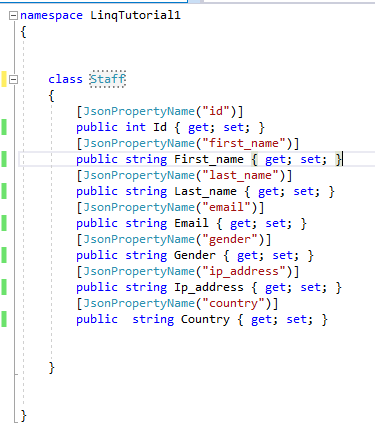
Now that we have data, we need to read the contents into the application memory, and deserialize the JSON data.

Staff Class

Create a class called staff and define all properties of the staff object. Right-click on the Project solution and add a class item called staff.cs.



Inside the class file add the following definition of the Staff class. To set the name of individual properties of the JSON file, use the [JsonPropertyName] attribute.

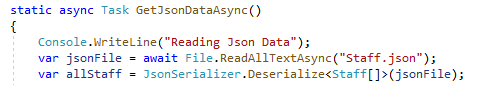


Read File and Deserialize Data into Memory Objects

Create a static method GetJsonDataAsync and use the File method ReadAllTextAsync to read the contents of the file.

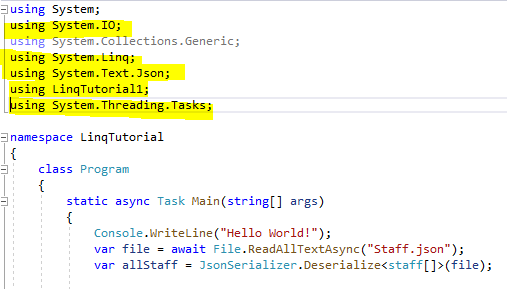
What are the consequences of reading the entire file into a string called jsonFile.

The deserialization will convert the JSON elements into an array of Staff objects, called allStaff.



You will need the following namespaces;

**System.Text.Json** is the new JSON library which is superseding NewtonsoftJson library, so use this library for all serialisation and deserialization tasks. Also, need the Linq library, to iterate through all the objects in the array.



Now we have the data deserialized into a Staff array, we can use a LINQ expression to iterate through the array of Json entries ;

var all = allStaff.Select(s=>s);

The Select statement will select each object of the Staff array (allStaff) and project them into a new form. The new form is a collection type of IEnumerable of type Staff.

So ***all*** is the **IEnumerable** of type **Staff**

IEnumerable is

1. an interface
2. a forward only cursor
3. a read only cursor
4. will return false if you get to end.
5. cannot manipulate IEnumerable.

Graphical user interface, text, application, email

Description automatically generated

**Task 5**

Convert the GetJsonData into an asynchronous task and call from the main method. List the steps and show code below;

I changed the method stamp to “async Task” then added “await” to the main method call. After that I then added an “await” key word in front of the File.ReadAllText. Then updated the File method to ReadAllTextAsync().

public static async Task GetJsonData()

{

Console.WriteLine("Reading JSON Data");

var jsonFile = await File.ReadAllTextAsync("../../../Staff.json");

var allStaff = JsonSerializer.Deserialize<Staff[]>(jsonFile);

if(allStaff != null)

{

var all = allStaff.Select(s => s);

foreach (var staff in all)

Console.WriteLine($"The staff member {staff.First\_Name} {staff.Last\_Name} is from {staff.Country}");

}

else

Console.WriteLine("File is empty");

}

**Task 6 -** Adjust the GetJsonDataAsync which will convert the staff memory objects to create an XML file. Paste code here.

public static async Task GetJsonDataAsync()

{

Console.WriteLine("Reading Json Data");

var jsonFile = await File.ReadAllTextAsync("../../../Staff.json");

var allStaff = JsonSerializer.Deserialize<Staff[]>(jsonFile);

System.Xml.Serialization.XmlSerializer serializer = new System.Xml.Serialization.XmlSerializer(typeof(Staff));

System.IO.FileStream file = System.IO.File.Create("../../../Staff.xml");

serializer.Serialize(file, allStaff);

file.Close();

}

**Objective 4: Download File from Internet using a HttpClient**

The simply way how to download a file from the Internet is to use HttpClient class and its method GetAsync. This method has two parameters, first is the url of the website you want to download the file from and the second parameter is to return the HTTP Get response code (which is optional).

Graphical user interface, text, application

Description automatically generated

Streams is a convenient way to channel information from network to local hard disk and vice versa.

Associated with HTTP GET requests are response codes;

200 – ok successful

Task 7

List the purpose of the following lines of code. Write your answer below the line of code.

1. var fname = Path.GetFileName(url);

it gets and stores the file name that was located at that location into the fname variable

1. using Stream stream = await resp.Content.ReadAsStreamAsync();

Serialize the HTTP content and returns a stream that represents the content as an asynchronous operation – (VS studio 2022 definetion.) The await keyword corresponds to the Async methods and Tasks.

1. using FileStream fs = File.Create(fname);

we create a FileStream object that then creates a file with the name that was stored in the above fname variable

1. await stream.CopyToAsync(fs);

Asynchronously reads the bytes from the current stream and writes them to another stream. – (VS 2022 )

**Task 8**

**Create a method which will display a menu to allow us to;**

**1.Create a New File**

**2.Read File**

**3.Write To File**

**4.Delete File**

**5.Quit Program**

**Show code and execution of program here.**

**End of Tutorial**