How to: Enumerate directories and files

Article • 09/15/2021

Enumerable collections provide better performance than arrays when you work with large collections of directories and files. To enumerate directories and files, use methods that return an enumerable collection of directory or file names, or their DirectoryInfo, FileInfo, or FileSystemInfo objects.

If you want to search and return only the names of directories or files, use the enumeration methods of the Directory class. If you want to search and return other properties of directories or files, use the DirectoryInfo and FileSystemInfo classes.

You can use enumerable collections from these methods as the IEnumerable<T> parameter for constructors of collection classes like List<T>.

The following table summarizes the methods that return enumerable collections of files and directories:

Expand table

To search and return	Use method
Directory names	Directory.EnumerateDirectories
Directory information (DirectoryInfo)	DirectoryInfo.EnumerateDirectories
File names	Directory.EnumerateFiles
File information (FileInfo)	DirectoryInfo.EnumerateFiles
File system entry names	Directory.EnumerateFileSystemEntries
File system entry information (FileSystemInfo)	DirectoryInfo.EnumerateFileSystemInfos
Directory and file names	Directory.EnumerateFileSystemEntries

Note

Although you can immediately enumerate all the files in the subdirectories of a parent directory by using the AllDirectories option of the optional SearchOption enumeration, UnauthorizedAccessException errors may make the enumeration incomplete. You can catch these exceptions by first enumerating directories and then enumerating files.

Examples: Use the Directory class

The following example uses the Directory.EnumerateDirectories(String) method to get a list of the top-level directory names in a specified path.

C#

```csharp
using System;
using System.Collections.Generic;
using System.IO;

class Program
{
    private static void Main(string[] args)
    {
        try
        {
            // Set a variable to the My Documents path.
            string docPath =
Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

            List<string> dirs = new List<string>(Directory.Enumerate-
Directories(docPath));

            foreach (var dir in dirs)
            {
                Console.WriteLine($"
{dir.Substring(dir.LastIndexOf(Path.DirectorySeparatorChar) + 1)}");
            }
            Console.WriteLine($"{dirs.Count} directories found.");
        }
        catch (UnauthorizedAccessException ex)
        {
            Console.WriteLine(ex.Message);
        }
        catch (PathTooLongException ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}
```

The following example uses the Directory.EnumerateFiles(String, String, SearchOption) method to recursively enumerate all file names in a directory and subdirectories that match a certain pattern. It then reads each line of each file and displays the lines that contain a specified string, with their filenames and paths.

C#

```
using System;
using System.IO;
using System.Linq;

class Program
{
    static void Main(string[] args)
    {
        try
        {
            // Set a variable to the My Documents path.
            string docPath =

Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

            var files = from file in
Directory.EnumerateFiles(docPath, "*.txt",
SearchOption.AllDirectories)
                        from line in File.ReadLines(file)
                        where line.Contains("Microsoft")
                        select new
                        {
                            File = file,
                            Line = line
                        };

            foreach (var f in files)
            {
                Console.WriteLine($"{f.File}\t{f.Line}");
            }
            Console.WriteLine($"{files.Count().ToString()} files
found.");
        }
        catch (UnauthorizedAccessException uAEx)
        {
            Console.WriteLine(uAEx.Message);
        }
        catch (PathTooLongException pathEx)
        {
            Console.WriteLine(pathEx.Message);
        }
    }
}
```

Examples: Use the DirectoryInfo class

The following example uses the DirectoryInfo.EnumerateDirectories method to list a collection of top-level directories whose CreationTimeUtc is earlier than a certain DateTime value.

C#

```
using System;
using System.IO;

namespace EnumDir
{
    class Program
    {
        static void Main(string[] args)
        {
            // Set a variable to the Documents path.
            string docPath =
Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

            DirectoryInfo dirPrograms = new DirectoryInfo(docPath);
            DateTime StartOf2009 = new DateTime(2009, 01, 01);

            var dirs = from dir in dirPrograms.EnumerateDirectories()
            where dir.CreationTimeUtc > StartOf2009
            select new
            {
                ProgDir = dir,
            };

            foreach (var di in dirs)
            {
                Console.WriteLine($"{di.ProgDir.Name}");
            }
        }
    }
}
// </Snippet1>
```

The following example uses the DirectoryInfo.EnumerateFiles method to list all files whose Length exceeds 10MB. This example first enumerates the top-level directories, to catch possible unauthorized access exceptions, and then enumerates the files.

C#

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        // Set a variable to the My Documents path.
        string docPath =
Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

        DirectoryInfo diTop = new DirectoryInfo(docPath);
        try
        {
            foreach (var fi in diTop.EnumerateFiles())
            {
                try
                {
                    // Display each file over 10 MB;
                    if (fi.Length > 10000000)
                    {
                        Console.WriteLine($"
{fi.FullName}\t\t{fi.Length.ToString("N0")}");
                    }
                }
                catch (UnauthorizedAccessException unAuthTop)
                {
                    Console.WriteLine($"{unAuthTop.Message}");
                }
            }

            foreach (var di in diTop.EnumerateDirectories("*"))
            {
                try
                {
                    foreach (var fi in di.EnumerateFiles("*",
SearchOption.AllDirectories))
                    {
                        try
                        {
                            // Display each file over 10 MB;
                            if (fi.Length > 10000000)
                            {
                                Console.WriteLine($"
{fi.FullName}\t\t{fi.Length.ToString("N0")}");
                            }
                        }
                        catch (UnauthorizedAccessException unAuth-
File)
                        {
                            Console.WriteLine($"unAuthFile: {unAuth-
File.Message}");
                        }
                    }
                }
                catch (UnauthorizedAccessException unAuthSubDir)
                {
                    Console.WriteLine($"unAuthSubDir: {unAuthSub-
Dir.Message}");
                }
            }
        }
        catch (DirectoryNotFoundException dirNotFound)
        {
            Console.WriteLine($"{dirNotFound.Message}");
        }
        catch (UnauthorizedAccessException unAuthDir)
        {
            Console.WriteLine($"unAuthDir: {unAuthDir.Message}");
        }
        catch (PathTooLongException longPath)
        {
            Console.WriteLine($"{longPath.Message}");
        }
    }
}
```

See also

- File and stream I/O