Instant Search

C# Example Pages

FileInfo gets information about a file. It retrieves information about a specific file or directory from the file system. The FileInfo type provides a host of methods and properties. These help us detect the status of a file.

Attributes. Every file on the Windows File system stores a set of attributes that tell you certain things about the file. You can detect its visibility, whether it is a directory, and if it is read-only.

Note:

The Attributes property returns an enumerated constant that is encoded as enum flags. You can individually test these flags.

Enum Flags

Based on: .NET (2019)

C# program that uses Attributes

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        // Get Attributes for file.
        FileInfo info = new FileInfo("C:\\file.txt");
        FileAttributes attributes = info.Attributes;
        Console.WriteLine(attributes);

        // Get Attributes for directory.
        info = new FileInfo("C:\\");
        attributes = info.Attributes;
        Console.WriteLine(attributes);
    }
}
```

Output

```
Archive
Hidden, System, Directory
```

Possible values for FileAttributes. There are many different flags on the FileAttributes enum. Some of them, such as ReadOnly, can be retrieved in other ways. This is true for many methods in the .NET Framework.

FileAttributes values

```
FileAttributes.Archive
FileAttributes.Compressed
FileAttributes.Device
FileAttributes.Directory
FileAttributes.Encrypted
FileAttributes.Hidden
FileAttributes.Normal
FileAttributes.NotContentIndexed
FileAttributes.Offline
FileAttributes.ReadOnly
FileAttributes.ReparsePoint
FileAttributes.SparseFile
FileAttributes.System
FileAttributes.Temporary
```

ReadOnly and IsReadOnly method. If you are trying to determine if a file is read-only, you can use the IsReadOnly method instead of the FileAttributes.ReadOnly constant. This may result in easier-to-understand code.

Times. The Windows file system keeps track of a file's creation time, its last access, and its last write time. We can get this information with the FileInfo type. We access the CreationTime, LastAccessTime and LastWriteTime properties.

File.GetLastWriteTimeUtc

Utc time:

You can also get the creation, last access, and last write times in Coordinated Universal Time.

C# program that uses Time properties

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        FileInfo info = new FileInfo("C:\\file.txt");

        DateTime time = info.CreationTime;
        Console.WriteLine(time);

        time = info.LastAccessTime;
        Console.WriteLine(time);

        time = info.LastWriteTime;
        Console.WriteLine(time);
    }
}
```

Output

```
7/17/2010 9:48:48 AM
7/17/2010 9:48:48 AM
8/18/2010 4:48:27 PM
```

Directory. Every file on the file system is stored in a containing directory. You can quickly access this DirectoryInfo with the Directory property. Alternatively, you can get just the name of the directory with the DirectoryName string property.

Directory

C# program that uses Directory property

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        // Get file info.
        FileInfo info = new FileInfo("C:\\file.txt");

        // Access parent directory.
        DirectoryInfo dir = info.Directory;
        Console.WriteLine(dir.Name);

        // Get directory name.
        string name = info.DirectoryName;
        Console.WriteLine(name);
    }
}
```

Output

```
C:\
C:\
```

Exists. You can create a FileInfo for a file that does not exist—no exception will be thrown. To see if the file does not exist, you can test the result of the Exists property. It is true or false.

Bool

File.Exists:

This method has the same result as the Exists property. It is easier to call when we have no convenient FileInfo.

File.Exists

C# program that uses Exists property

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        // Get file info.
        FileInfo info = new FileInfo("C:\\does-not-exist.txt");
        bool exists = info.Exists;
        Console.WriteLine(exists);
    }
}
```

Output

```
False
```

Name, extension. Typically, you should use the Path type to get file name parts such as the name or extension. But the FileInfo type provides ways to get some of these parts as well. You can get the Name, the FullName or the Extension.

Path

C# program that uses file name properties

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        // Get file info.
        FileInfo info = new FileInfo("C:\\file.txt");

        string name = info.Name;
        string fullName = info.FullName;
        string extension = info.Extension; // Has period

        Console.WriteLine(name);
        Console.WriteLine(fullName);
        Console.WriteLine(extension);
    }
}
```

Output

```
file.txt
C:\file.txt
.txt
```

Length. How many bytes are in a file? The Length property on the FileInfo type provides a way to effectively determine this. It returns a figure in bytes, not megabytes or kilobytes. You may need to convert the value.

FileInfo Length

Sort Files, Sizes

Directory Size

Convert Bytes, Megabytes

C# program that uses Length property

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        FileInfo info = new FileInfo("C:\\a");
        long value = info.Length;
        Console.WriteLine(value);
    }
}
```

Output

```
5320683
```

MoveTo. You can use the FileInfo type to rename (move) a file. This should reduce copying in the file system over creating a duplicate file and deleting the original. This improves performance.

Also:

CopyTo and Replace provide parallel functionality to the MoveTo method. Instead of renaming, CopyTo makes a copy of the file.

And:

The Replace method allows you to copy a file to an existing location and also make a backup file.

C# program that uses MoveTo method on FileInfo

```
using System.IO;

class Program
{
   static void Main()
   {
      // Write the specified file with some text.
      File.WriteAllText("C:\\test1.txt", "A");

      // Create a FileInfo instance for the specified path.
      // ... Then move the specified file to a new file path.
      FileInfo info = new FileInfo("C:\\test1.txt");
      info.MoveTo("C:\\test2.txt");
   }
}
```

Results

```
1. One file is created.
2. The file is renamed to a new name.
```

You can test this program by compiling as a console application in Visual Studio. To do this, paste the text into the Program.cs file and compile it. Then open the Windows Explorer after the program is executed.

Note:

The test1.txt file in the C: volume directory is erased and the result is a test2.txt file with the contents of "A".

Text. If your FileInfo points to a text file, it is a good idea to use the AppendText, CreateText, and OpenText methods to acquire a StreamWriter or StreamReader instance.

You should wrap these in the using-resource-acquisition statement.

Using

In my view:

Directly creating a StreamWriter or Reader is a simpler pattern, one more common, and it should be preferred.

StreamWriter

StreamReader

C# program that uses text properties

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        // This file will have a new line at the end.
        FileInfo info = new FileInfo("C:\\file.txt");
        using (StreamWriter writer = info.AppendText())
        {
            writer.WriteLine("Line");
        }

        // This file will have the word New in it.
        info = new FileInfo("C:\\new.txt");
        using (StreamWriter writer = info.CreateText())
        {
            writer.WriteLine("New");
        }

        // Get a StreamReader with OpenText.
        using (StreamReader reader = info.OpenText())
        {
            Console.WriteLine(reader.ReadToEnd());
        }
    }
}
```

Output

```
New
```

FileStream. Sometimes, we deal with a non-text file and want to use types such as BinaryReader or BinaryWriter on a file. To do this from a FileInfo instance, use the Create and Open methods, and then create the BinaryReader from the FileStream.
BinaryReader
BinaryWriter

Security. The FileInfo type provides a way to retrieve and store security information on the file system. This is most important for multi-user servers that have restricted information. These methods are not covered in detail here.

Security methods

```
GetAccessControl
SetAccessControl

GetLifetimeService
InitializeLifetimeService
```

Next, the Encrypt and Decrypt methods are not available on all Windows computers. If you choose to use them, you can wrap them in exception-handling in the chance that the computer does not support them.

Error caused by using Encrypt when unsupported

```
Unhandled Exception: System.IO.IOException: The request is not supported.

  at System.IO.__Error.WinIOError(Int32 errorCode, String maybeFullPath)
  at System.IO.File.Encrypt(String path)
  at System.IO.FileInfo.Encrypt()
  at Program.Main() in C:\...Program.cs:line 10
```

Remoting. We don't cover it here. But the Remoting capability in the .NET Framework provides support for the FileInfo type. We can use the CreateObjectRef methods to enable remoting on a FileInfo instance.

Summary. By providing these properties and methods, the FileInfo type makes it easy to get information from the file system about a file. You can use this information to make important branches in your program's logic.

Dot Net Perls