Directory Class

Source	http://samples.gotdotnet.com/quickstart/howto/doc/Directory.aspx

This sample illustrates how to use the `DirectoryInfo` **class to create a directory listing. This is a great way to quickly list files for users or reports, or to find basic information about a specific directory. When used in combination with** `FileInfo`**, you can get all the information you need about files and directories in a specific directory.**

The ability to look over files and subdirectories for a given directory is essential for many programming tasks. This QuickStart shows you how to use the `FileInfo` **and** `DirectoryInfo` **classes. Why not the File and Directory classes? This QuickStart intends to display information such as the size, and creation date for a given file. In this case, it is easier to use** `FileInfo` **and** `DirectoryInfo` **objects. The** `File` **class focuses on the use of non-instantiated (static in C#, or shared in Visual Basic) methods, while the** `FileInfo` **class offers methods based on an instance of a** `FileInfo` **object.** `Directory` **and** `DirectoryInfo` **are similar.**

The first task explains how create an instance of the `DirectoryInfo` **object (based on the current directory), then find the fully-qualified path of that directory. Next, you can make a table of all the files and directories inside our current directory, so it will print the full path of the current directory at the top of that table. We use the** `GetFullPath` **method of the Path class to determine the fully-qualified path of our directory.**

```
//Do not forget your using statements at the top of your code.
using System;
using System.IO;
//...

// make an object which represents our current directory. The dot (".")
represents this directory
DirectoryInfo dir = new DirectoryInfo(".");

// make the header for our table, letting the user know the path we are in...
Console.WriteLine("Following is a listing for directory: {0}",
Path.GetFullPath(dir.ToString()));
```

The objects inside the directory can be files or directories. You can iterate through the directory twice, looking for files first, and directories next. An alternate solution is to use the `FileSystemInfo` **object, which can represent a** `FileInfo` **or a** `DirectoryInfo` **object. This means you only have to iterate through the collection once. Then, call the** `GetFileSystemInfos` **method of the object created in the code above, which returns an array of** `FileSystemInfo` **objects. You can iterate through that array with the** `foreach` **(For Each in Visual Basic), as the following code demonstrates.**

```
// loop through our array of FileSysteminfo objects
foreach(FileSystemInfo fsi in dir.GetFileSystemInfos()) {
```

```
        // ... the code in the following samples goes here.

}
```

So now you can see what kind of object each element of the `FileSystemInfo` **array is, and process it accordingly. First test the instance of the** `FileSystemObject` **to see if it is a file. If it is, create a new instance of the** `FileInfo` **object to represent it. This enables you to call the methods particular to that kind of object. If you do not do this, you could not call properties such as** `Length`**, which does not exist in** `FileSystemInfo`**. In contrast,** `CreationTime` **is a property of the** `FileSystemInfo` **object, which is why you can deal with it before you enter your** `If` **statement, regardless of whether you have a file or a directory.**

Notice that you are shortening the name you display to the user, primarily to save screen space. Because there can be file access or other related exceptions, execute this code in a `Try` **block, to make sure it does not end abnormally. Put this code together with the code in the previous examples, and you have your program (once inside a method, of course). Note that this only demonstrates how to process a file.**

```
try {
        DateTime creationTime = fsi.CreationTime;
        int subLength = 25;

        if (fsi is FileInfo) {    // check to see if the current object is a
file...

                FileInfo f = (FileInfo)fsi;

                // this if statement simply ensures that we do not shorten the
                // name of the file too much!
                if (f.Name.Length < subLength)
                        subLength = f.Name.Length;
                }

                string name = f.Name.Substring(0, subLength);
                 long size = f.Length;

                // format the output to the screen
                Console.WriteLine("{0, -25} {1,-12:N0} {2, -12} {3,-20:g}",
                                name, (size + " KB").PadLeft(12), "File",
creationTime);
        }
        else { // it must be a directory
                // ...
        }
} catch (Exception) {} // ignore errors such as 'file being used' etc...
```

Summary

You can working with files and directories using `FileInfo` **and** `DirectoryInfo`**, although you can also use** `File` **and** `Directory` **as alternatives. By working with the Info-based objects, you need an instance of the class, but you can easily obtain specific information about a file or directory (such as size, or creation date).**

~~~ End of Article ~~~