C# How-to 8: Perform a directory listing

In a previous page on this site, I showed how to search a directory recursively to find a file, using the Windows Scripting Host from Visual Basic. It's at least as easy to do that from C#, which is the topic for this page.

To do this we can use the DirectoryInfo and FileInfo objects. This page shows how to use these to return a list of files or folders in a given directory.

1. Create class to return a file list

Create a new project (I called it DirLister) and add a new class (.cs) file to it, which you might call ListFiles.cs. This class should use the System, System.IO, and System.Collections namespaces, and needs one class-level variable of type DirectoryInfo, which I called m_dir. The constructor for this class takes one parameter, the name of the folder from which a list of files or sub-folders is to be returned. The code looks like this so far:

```
using System;
using System.IO;
using System.Collections;

namespace DirLister
{
public class ListFiles {
private DirectoryInfo m_dir;

// constructor takes the path of the folder
// to return a file or folder list from
public ListFiles(string sFolder) {
m_dir = new DirectoryInfo(sFolder);
}
}
}
```

2. Get a list of files or folders

The list of files we get must be held in something, and for this we can use an ArrayList - basically an array which expands dynamically as items are added to it. We now have a DirectoryInfo object, and the GetFiles() method of this object returns a list of files - not the file names, but a collection of FileInfo objects, from which we can get the file name. All we have to do is call GetFiles() on our m_dir object, store the names in an ArrayList, and return this list. The code is:

```
public ArrayList ShowFiles() {
// lists all the files in a folder
ArrayList alFiles = new ArrayList();
foreach (FileInfo f in m_dir.GetFiles("*.*")) {
alFiles.Add(f.Name);
}
return alFiles;
}
```

The code to return a list of folders is identical, except that instead of the GetFiles() method, we call the (you guessed it) GetDirectories() method of the DirectoryInfo object. This returns a collection of DirectoryInfo objects, from which we can get the folder name. The code is like this:

```
public ArrayList ShowFolders() {
// lists all sub-folders in a folder
```

```
ArrayList alFolders = new ArrayList();
foreach (DirectoryInfo dinfo in m_dir.GetDirectories("*.*")) {
alFolders.Add(dinfo.FullName);
}
return alFolders;
}
```

So now our class has two methods which will return either a list of files in a folder, or a list of sub-folders. We can use these functions recursively to list all files and folders (and their files and sub-folders) in a given directory.

3. List all files and folders in a folder

I won't show the entire code here, just the relevant parts. For now, assume we have a form with a list box on it called lisFiles, and some method of getting a folder path - it doesn't matter exactly what. Then we need some code to add all the files in a folder to the list box, which could look like this:

```
private void ListAllFiles(string startDir) {
ArrayList ar = new ArrayList();

ListFiles lf = new ListFiles(startDir);
ar = lf.ShowFiles();
foreach (string fName in ar) {
lisFiles.Items.Add(fName.ToLower());
}
}
```

What happens here? First, we create an ArrayList to contain the returned list of files. Then we create a new ListFiles object (this is the class we created at the start of this page), passing its constructor the starting directory name. We then call its ShowFiles() method, which returns a list of file names. Finally, we add each one of these to the list box. We can use this function to display a list of files from any folder. (The setting of the file name to lower case is purely a kludge to distinguish files from folders in the list box. In a real-world application, you'd do this in a much nicer way.)

The full list of folders, files, and sub-folders and their files is a little trickier. Each folder may or may not contain files, but it may also contain sub-folders. If it does, we want to drill down into each sub-folder (and their sub-folders if there are any) until we reach the point where there are no more sub-folders. This is where the recursion comes in. The function looks like this:

```
private void ListAllFolders(string startDir) {
ArrayList ar = new ArrayList();

ListAllFiles(startDir);
ListFiles lf = new ListFiles(startDir);
ar = lf.ShowFolders();
foreach (string folName in ar) {
lisFiles.Items.Add(folName.ToUpper());
ListAllFiles(folName);
ListAllFolders(folName);
}
}
```

The first thing we do is call ListAllFiles() with the starting directory name as its parameter to display all the files in that folder. If we don't do this, we'll miss the files in this top-level folder. Then we call the ShowFolders() method of the ListFiles object to get a list of sub-folder path names, if any. For each of those sub-folders, if there are any, we add the folder name to the list box, first making it upper case to distinguish it

from ordinary files. Then we get the list of files in that sub-folder, and finally (the recursion) we call ListAllFolders() again with the name of the sub-folder as the parameter. This process will then repeat itself, drilling deeper and deeper until there are no more sub-folders in that tree.

There's only one final problem. The DirectoryInfo.GetFiles() and .GetDirectories() methods could throw an exception if we pass the name of a directory which doesn't exist. So we should catch this error as in this code fragment:

```
try {
ar = lf.ShowFiles();
foreach (string fName in ar) {
lisFiles.Items.Add(fName.ToLower());
}
}
catch (System.IO.DirectoryNotFoundException) {
MessageBox.Show("The directory you entered was not found.",
"Directory not found", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation);
}
```

A similar try...catch block would be added to both the ListAllFiles and ListAllFolders functions we defined above. If a DirectoryNotFound exception occurs, we simply show the user an appropriate message.

I think you'll agree that there isn't much code when you consider the functionality. You can download a demo application which shows how it works from the link below. This includes the ListFiles class which you can re-use in your own projects.

Download a simple demonstration of recursive file listing (39K)