C# reading to and writing from files

C# uses System.IO to provide access to file input and output routines. Care must be taken to when working with files as it is easy to crash the system.

The following routine writes the current date and time to a file in the directory F. If the file does not exist the program will create it. If the file does exist any current data will be overwritten.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            String input;
            input = Console.ReadLine();
            // create a writer and open the file
            TextWriter tw = new StreamWriter("f:\\date.txt");
            // write a line of text to the file
            tw.WriteLine(DateTime.Now);
            // close the stream
            tw.Close();

        }
    }
}
```

Note that if the '`using System.IO;`' line is commented out the TextWriter and StreamWriter will throw errors because C# no longer recognises what they are. This can be overcome by using the following line but it is easier to import System.IO as several objects from this set are used in file access.

```
System.IO.TextWriter tw =new System.IO.StreamWriter("f:\\date.txt");
```

The line `DateTime.Now` will write the current date and time to file. Any String can be written to file with the same method, such as `tw.WriteLine("data to save")`. The file name can also be passed to as a String, `System.IO.StreamWriter(nameOfString)` but the path specified must be valid.

The following code will open the file and write its contents to the console. The file must exist for this to work.

```
TextReader tr = new StreamReader("f:\\date.txt");
Console.WriteLine(tr.ReadLine());
tr.Close();
```

If the file does not exist a FileNotFoundException is raised. The TextReader and TextWriter streams must be closed. If they are not it is possible that data will not be correctly read from or written to the file. There is also the chance of closing a stream that is already closed or opening a stream that is already open. Both of these evants will cause an error.

Adding new data to an existing file is called appending data. This requires a minor modification to the StreamWriter code. The append action can also be used to create a file that does not exist and put data into it.

```
TextWriter tw = new StreamWriter("f:\\date.txt",true);
tw.WriteLine(DateTime.Now);
tw.Close();
```

The file contents might now look something like the following but the TextReader code will only return 1 line of the file.

```
30/09/2011 15:19:37
30/09/2011 15:39:54
30/09/2011 15:43:21
```

This is because Readline() only reads one line. The correct TextReader method to call is ReadToEnd().

```
TextReader tr = new StreamReader("f:\\date.txt");
String text = tr.ReadToEnd();
Console.WriteLine(text);
tr.Close();
```

Practice

Create a C# program that uses these 3 routines by asking the user for a keyboard input to choose the file routine to run.

```
Enter w to write to a new file
Enter r to read from a file
Enter a to append to a file
r
30/09/2011 15:19:37
30/09/2011 15:39:54
30/09/2011 15:43:21
30/09/2011 15:51:43
```

Allow the user to input the data to be saved to file and the path and name of the file.

The File class will allow entire files to be manipulated such as copying and deleting. There will be an error if the file to be copied or deleted does not exist.

```
File.Copy("f:\\date.txt", "f:\\date.bak");
```

Catching errors

As file access programming regularly throws up errors it is best to handle them. The best way is through a try catch block.

Here the try goes around all of the file handling routines. This is followed by the catch block. The error is output to the console here but the program continues to run, it does not crash.

```
try
{
	File.Copy("f:\\date.txt", "f:\\date.bak");
}
catch (FileNotFoundException f)
{
	String err = f.ToString();
	Console.WriteLine("Error: " + err);
}
```

In the code above if there is no `"f:\\date.txt"` file the code will still run without crashing. There can be multiple catch clauses for a single try each picking up a different error.

```
try
{
	File.Copy("f:\\date.txt", "f:\\date.bak");
}
catch (FileNotFoundException f)
{
	String err = f.ToString();
	Console.WriteLine("Error: " + err);
}
catch (Exception e)
{
	Console.WriteLine("Something else went wrong");
	Console.WriteLine(e.ToString());
}
```