Files

Legacy File Handling

- You have probably utilized files with legacy libraries such as:
  1. The C run-time library
  2. C++ Iostream library
  3. MFC CFile class
  4. WIN32 CreateFile operations etc.
- The .NET FCL provides a new and simple to use class library for file handling.

File Streams

- *FileStream* is the basic class to perform *raw* file I/O.
- *System.Object*
  System.MarshalByRefObject
  System.IO.Stream
  System.IO.FileStream
- The *Stream* class is not necessarily associated with a file and is used more generally.
- The constructor for *FileStream* is very simple:

```
public FileStream(string path,
  FileMode mode);
```

FileMode Enumeration

Member	*Description*
Append	Opens the file if it exists and seeks to the end of the file, or creates a new file.
Create	Specifies that the operating system should create a new file. If the file already exists, it will be overwritten.
CreateNew	Specifies that the operating system should create a new file.
Open	Specifies that the operating system should open an existing file.
OpenOrCreate	Specifies that the operating system should open a file if it exists; otherwise, a new file should be created.
Truncate	Specifies that the operating system should open an existing file. Once opened, the file should be truncated so that its size is zero bytes.

Reading a File

```
int ReadByte(); //read a single byte
int Read(byte[] array, int offset,
     int count);
```

- *ReadByte* returns a value between 0 and 255 for the next byte.
- *Read* trys to read *count* bytes and the actual number of bytes read is returned.

Writing a File

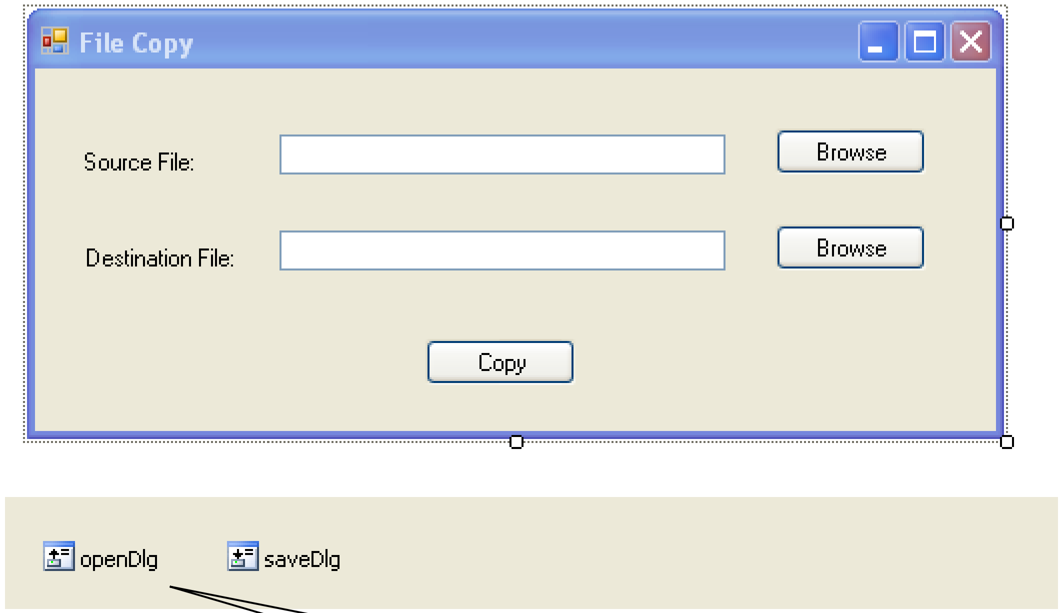
- The corresponding methods for writing are as follows:

```
void WriteByte(byte value);
void Write(byte[] array, int offset, int
    count);
```

Closing a File

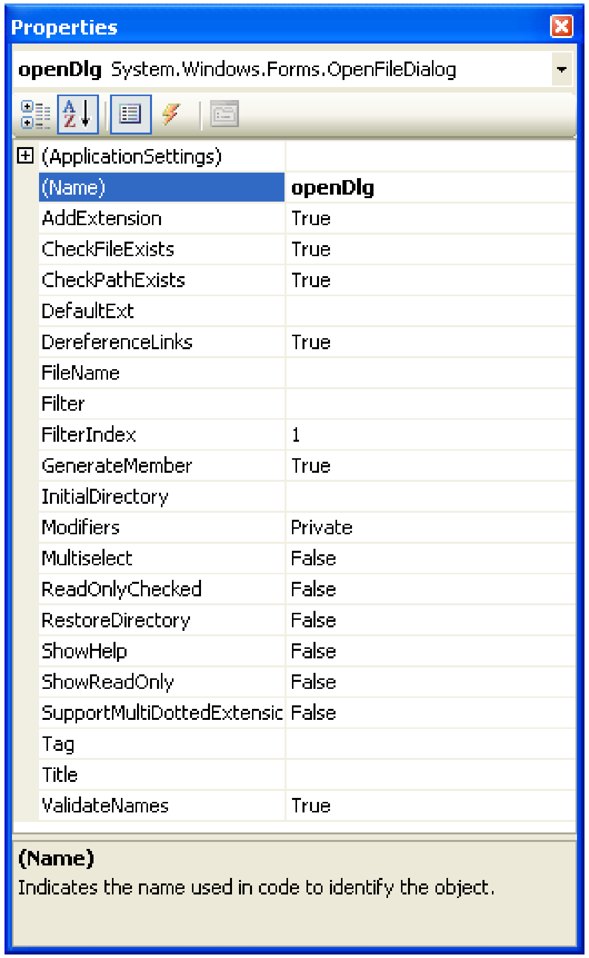
- Call *Close* to complete the reading or writing of the file and ensure it is completely written to the destination in the case of writing.
- This also frees up the resource within the operating system.

File Copy Example



These are the dialogs dragged from the toolbox.

File Open Dialog Properties



Properties

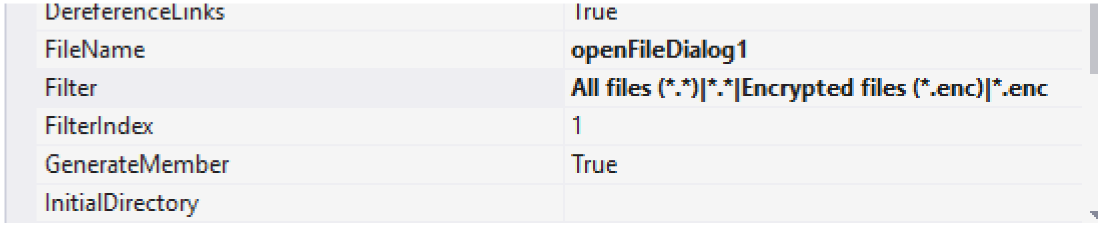
openDlg System.Windows.Forms.OpenFileDialog

⊞ (ApplicationSettings)	
(Name)	**openDlg**
AddExtension	True
CheckFileExists	True
CheckPathExists	True
DefaultExt	
DereferenceLinks	True
FileName	
Filter	
FilterIndex	1
GenerateMember	True
InitialDirectory	
Modifiers	Private
Multiselect	False
ReadOnlyChecked	False
RestoreDirectory	False
ShowHelp	False
ShowReadOnly	False
SupportMultiDottedExtensic	False
Tag	
Title	
ValidateNames	True

(Name)
Indicates the name used in code to identify the object.

File Filter

- Add pairs separated by vertical bars.
- Each pair has a description and wildcard pattern



DereferenceLinks	True
FileName	**openFileDialog1**
Filter	**All files (*.*)|*.*|Encrypted files (*.enc)|*.enc**
FilterIndex	1
GenerateMember	True
InitialDirectory	

File Copy Example Code

```
FileCopy - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace FileCopy
{
    public partial class FileCopy : Form
    {
```

File Copy Example Code – Contd.

```
public FileCopy()
{
    InitializeComponent();
}
private void browseSource_Click(object sender,
       EventArgs e)
{
    if(openDlg.ShowDialog(this)==DialogResult.OK)
        source.Text=openDlg.FileName;
}
private void browseDestination_Click(object
        sender, EventArgs e)

{
```

Copyright © Thomas P. Skinner

File Copy Example Code – Contd.

```
            if (saveDlg.ShowDialog(this) ==
                    DialogResult.OK)
                 destination.Text = saveDlg.FileName;
         }
         private void copy_Click(object sender, EventArgs
                   e)
         {
             FileStream infile=null, outfile=null;
             byte[] buffer = new byte[512];
             int count;
             if (source.Text == "" || destination.Text ==
                        "")
             {
```

File Copy Example Code – Contd.

```
                MessageBox.Show("Both source and
                   estination files must be specified!");
                return;
            }
             if (source.Text == destination.Text)
            {
                MessageBox.Show("Source and destination
                    file can't be the same!");
                return;
            }
          try
            {
```

Copyright © Thomas P. Skinner

File Copy Example Code – Contd.

```
      infile = new FileStream(source.Text,
         FileMode.Open);
      outfile = new FileStream(destination.Text,
         FileMode.Create);
      while((count=infile.Read(buffer, 0,
          512))>0)
          outfile.Write(buffer, 0, count);
   }
   catch (Exception ee)
   {
       MessageBox.Show("Copy failed!\n" +
          ee.Message);
   }
```

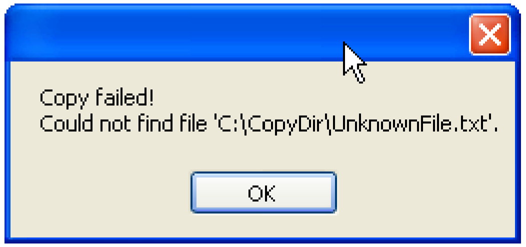
File Copy Example Code – Contd.

```
            finally
            {
                if(infile!=null) infile.Close();
                if(outfile!=null) outfile.Close();
            }
        }
    }
}
```

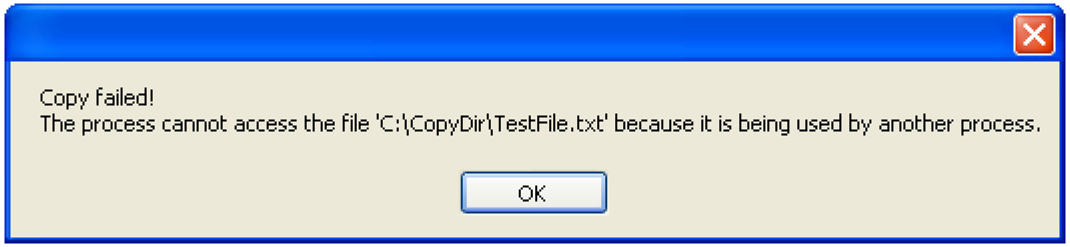
Example Notes

- The save dialog prompts if the file exists.
- The OS remembers the last files selected for a given application.
- We need to check for missing file names and setting the destination the same as the source.
- A *try/catch* is used to give friendly feedback to the user.

Friendly Messages



```
Copy failed!
Could not find file 'C:\CopyDir\UnknownFile.txt'.
```



```
Copy failed!
The process cannot access the file 'C:\CopyDir\TestFile.txt' because it is being used by another process.
```

Copyright © Thomas P. Skinner

The Copy Operation

```
infile = new FileStream(source.Text,
  FileMode.Open);
outfile = new
  FileStream(destination.Text,
  FileMode.Create);
while((count=infile.Read(buffer, 0,
  512))>0)
    outfile.Write(buffer, 0, count);
```

Closing the Files

```
if(infile!=null) infile.Close();
if(outfile!=null) outfile.Close();
```

- It is necessary to test to make sure the file was actually opened before attempting to close it.
- It is also important to note that I explicitly initialized both *infile* and *outfile* to *null*. If you neglect to do this the compiler will give you an error   indicating that you have attempted to use an unassigned local variable.
- This is a feature of C# that helps prevent errors that have plagued programmers in the past. Previously compilers and languages made it difficult or impossible to make such a test.

Flush

- *Flush* can be called at any time a file is open to ensure that all buffered data is actually output to the device.
- Use *Flush* if you are concerned about as much data being written to the device as possible in case of a system crash.
- Note that this does not guarantee that the disk file is readable subsequent to a crash.
- *Flush* can also be used with streams that associated with a network connection.

Seek

- *Seek* allows random access to any part of a file.
- `long Seek(long offset, SeekOrigin origin);`
- Offset is in bytes.

Member name	*Description*
Begin	Specifies the beginning of a stream.
Current	Specifies the current position within a stream.
End	Specifies the end of a stream.

Stream Readers and Writers

- The *StreamReader* and *StreamWriter* classes make working with text files very easy since we can work with characters or character strings rather than raw bytes.
- An important issue when working with character data is what character set we wish to use.
- Unicode is the standard for character strings in .NET, but ASCII is standard for Windows text files.
- The default for these classes is UTF-8 (Unicode Transformation Format) which is neither ASCII or Unicode.
- If you restrict your characters to the ASCII character set then the UTF-8 will be ASCII compatible.
- See http://www.unicode.org for details.

StreamWriter

- Constructors:

```
StreamWriter(string path);
StreamWriter(string path, bool append);
StreamWriter(string path, bool append, Encoding
  encoding);
```

- Example to force ASCII:

```
StreamWriter writer = new StreamWriter(mypath, false,
         Encoding.ASCII);
```

TextWriter

- *StreamWriter* inherits from *TextWriter*, an abstract class that is used elsewhere in the FCL.
- *TextWriter* allows formatting strings the same way as using the *Format* method of the *string* class. (Console.WriteLine as well)
- Example:

```
writer.Write("A simple string.");
writer.Write("Count = {0}", count);
```

Line Terminators

- This is always an issue when interoperting between Unix/Linus, Windows and Mac OS for example.
- Windows uses CR LF (0d, 0a)as the line terminator. (LF is \n)
- The *NewLine* property can be used to change this string.

A StreamReader Example

- This example displays text from a file in a scrollable panel.
  1. Change the form's title.
  2. Drag a menu strip to the form.
  3. Add a *File* item and an *Open* item under the *File* item.
  4. Add an event handler for the *Open* item.
  5. Drag a panel to the form.
  6. Set the panel's *AutoScroll* property to *true*.
  7. Set the panel's *Dock* property to *Fill*.
  8. Add a *Paint* event handler for the panel.
  9. Drag a *FileOpenDIalog* to the form.
  10. Set the *Filter* property to "Text Files|*.txt|All Files|*.*".

Example

```
ReadFile - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Collections;
using System.IO;
namespace ReadFile
{
    public partial class Form1 : Form
```

Example – Contd.

```
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void openToolStripMenuItem_Click(object
             sender, EventArgs e)
        {
            StreamReader infile = null;
            string line;
            if (openDlg.ShowDialog() == DialogResult.OK)
            {
                lines = new ArrayList();
```

Example – Contd.

```
try
{
    infile = new
        StreamReader(openDlg.FileName);
    while((line=infile.ReadLine())!=null)
        lines.Add(line);
}
catch
{
    MessageBox.Show("Can't open file!");
    return;
}
```

Example – Contd.

```
                finally
                {
                    if (infile != null) infile.Close();
                }
                Invalidate(true);
            }
        }
        ArrayList lines;
        private void panel_Paint(object sender,
                PaintEventArgs e)
        {
            int dy = 0;
            float maxh = 0, maxw = 0;
```

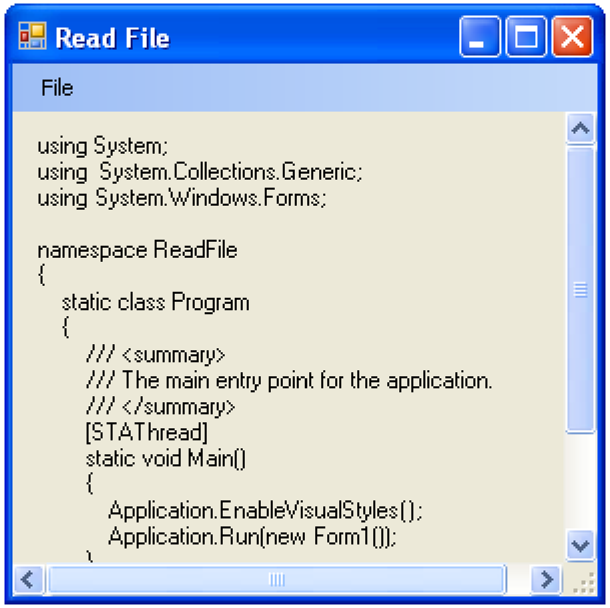
Example – Contd.

```
SizeF extent;
Graphics g = e.Graphics;
if (lines == null) return;
//compute size of text
foreach (string s in lines)
{
    extent = g.MeasureString(s, Font);
    maxh += extent.Height;
    maxw = Math.Max(maxw, extent.Width);
}
maxw += 50; maxh += 50;
panel.AutoScrollMinSize = (new SizeF(maxw,
       maxh)).ToSize();
```

Example – Contd.

```
          g.TranslateTransform(panel.AutoScrollPosition.X,
                panel.AutoScrollPosition.Y);
            foreach (string s in lines)
            {
                g.DrawString(s, Font, Brushes.Black, 10,
                        10 + dy);
                dy += Font.Height;
            }
        }
    }
}
```

Output



```
using System;
using  System.Collections.Generic;
using System.Windows.Forms;

namespace ReadFile
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new Form1());
```

String Writers

- The *StringWriter* class works just like a *StreamWriter* except that the output goes to a *string* rather than a file.
- We can then write formatted output using the *Write* and *WriteLine* methods.

```
int count = 1234;
sw.WriteLine("Count is equal to {0}”, count);
string s = sw.ToString();
```

- An alternative is the *StringBuilder* class that I discuss in a later chapter.

Copyright © Thomas P. Skinner

Binary Readers and Writers

- The *BinaryReader* and *BinaryWriter* classes provide a general binary read and write.
- These classes are more flexible than reading and writing *raw* binary with the *FileStream* class.

Issues

- Format of the binary data, e.g. floating point.
- Big endian vs. little endian.
- Precision – int is not always 32 bits.
- The example to follow is one from an earlier chapter in which we drew small ellipses. We will now save the points so we can reload them and redraw the client area.

Example

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace BinaryFile1
{
    public partial class Form1 : Form
```

Copyright © Thomas P. Skinner

Example

```
{
       private ArrayList coordinates = new ArrayList();
       private const int magic = 123456;  //the magic
                                  // number

       public Form1()
       {
           InitializeComponent();
       }
       protected override void OnPaint(PaintEventArgs e)
       {
           const int WIDTH = 20;
           const int HEIGHT = 20;
```

Example

```
            Graphics g = e.Graphics;
            foreach (Point p in this.coordinates)
            {
                g.FillEllipse(Brushes.Black,
                    p.X - WIDTH / 2, p.Y - WIDTH / 2,
                        WIDTH, HEIGHT);
            }
        }
        protected override void
                OnMouseClick(MouseEventArgs e)
        {
            {
```

Example

```
                if (e.Button == MouseButtons.Left)
                {
                    Point p = new Point(e.X, e.Y);
                    this.coordinates.Add(p);
                    this.Invalidate();
                }
                if (e.Button == MouseButtons.Right)
                {
                    this.coordinates.Clear();
                    this.Invalidate();
                }
            }
        }
```

Example

```
                private void
                  openToolStripMenuItem_Click(object
                      sender,
                EventArgs e)
        {
            if (openFileDialog1.ShowDialog(this) ==
                DialogResult.OK)
            {
                FileStream stream = new
                   FileStream(openFileDialog1.FileName,
                       FileMode.Open);
                BinaryReader reader = new
                       BinaryReader(stream);
                //check for magic number
                try
                {
```

Example

```
int i = reader.ReadInt32();
    if (i != magic)
        throw (new Exception());
}
catch
{
    MessageBox.Show("Not the correct file
         format!");
    reader.Close();
    return;
}
coordinates.Clear();
int x, y;
```

Copyright © Thomas P. Skinner

Example

```
bool done = false;
while (!done)
{
    try
    {
        x = reader.ReadInt32();
        y = reader.ReadInt32();
        Point p = new Point(x, y);
        coordinates.Add(p);
    }
    catch
    {
```

Example

```
                        done = true;
                    }
                }
                reader.Close();
                Invalidate();
            }
        }

        private void saveToolStripMenuItem_Click(object
                 sender, EventArgs e)
        {
            if (saveFileDialog1.ShowDialog(this) ==
                    DialogResult.OK)
```

Example

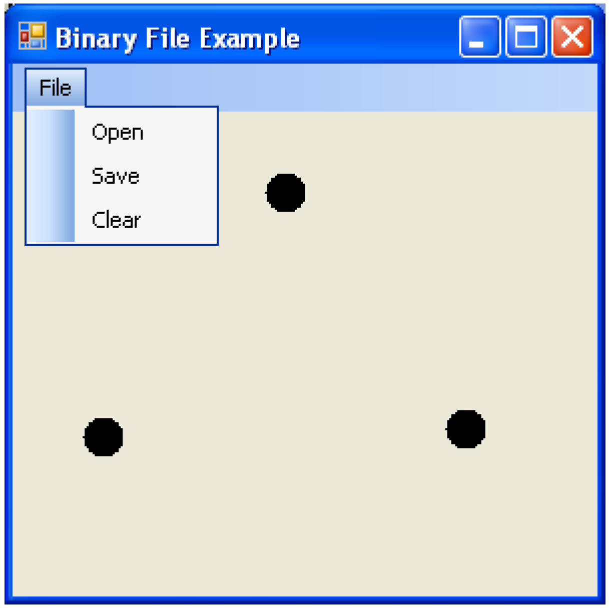
```
{
                FileStream stream = new
                    FileStream(saveFileDialog1.FileName,
                            FileMode.Create);
                BinaryWriter writer = new
                    BinaryWriter(stream);
                //Write magic number
                writer.Write(magic);
                //write the point collection
                foreach (Point p in coordinates)
                {
                    writer.Write(p.X);
                    writer.Write(p.Y);
                }
```

Example

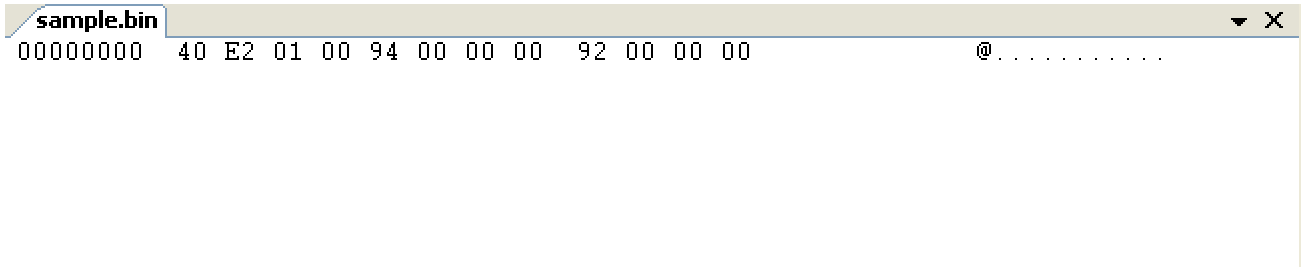
```
                writer.Close();
            }
        }

        private void clearToolStripMenuItem_Click(object
           sender, EventArgs e)
        {
            coordinates.Clear();
            Invalidate();
        }
    }
}
```

Copyright © Thomas P. Skinner

Menu



The Actual Binary File Data



```
00000000  40 E2 01 00 94 00 00 00  92 00 00 00          @...........
```

Working with Files

- We often perfomr the following operations:
  1. Delete a file
  2. Rename a file
  3. Move/copy a file
  4. Determine a file's attributes
  5. Change a files attributes
- Two classes provide a plethora of operations.

The File Class

- As an alternative to constructor the *File* class can be used.
- These are the static methods you can use:

Method	*Description*
AppendText	Creates a StreamWriter in append mode.
CreateText	Creates a StreamWriter.
Open	Opens a FileStream.
OpenRead	Opens an existing file for reading using a FileStream.
OpenText	Opens a text file for reading using a StreamReader.
OpenWrite	Opens an existing file for writing using a FIleStream.

Additional Methods

Method	*Description*
Copy	Copies a file.
Delete	Deletes a file.
Exists	Tests if a file exists.
Move	Move a file (does not overwrite.)
Replace	Replaces an existing file, deletes the original file, and makes a backup of the replaced file.

Moving a File

```
if(!File.Exists(file1))
     MessageBox.Show("Source file does not exist!");
else if(File.Exists(file2)
     MessageBox.Show("Destination file already
  exists!");
else File.Move(file1, file2);
```

The FileInfo Class

Method	*Description*
AppendText	Open a text file for appending.
Create	Create a FileStream.
CreateText	Create a text file.
Delete	Delete a file.
MoveTo	Move a file.
Open	Open a FileStream.
OpenRead	Open an existing file as a FileStream for reading.
OpenText	Open a text file.
OpenWrite	Creates a write only FileStream.
Replace	Replaces a file.

FileInfo Properties

FileInfo Properties	*Description*
CreationTime	The file's creation time.
Exists	True if file exists.
FullName	The full path name of the file.
Extension	The file's extension.
IsReadOnly	True if the file is read only.
LastAccessTime	The last access time.
Length	The files length in bytes.
Name	The file's name.

Working with Directories (Folders)

- Often we need to manipulate entire directories rather than files.
- .NET makes this easy.

The Directory Class

Method	*Description*
CreateDirectory	Creates a directory including all the directories in the path.
Delete	Deletes a specified directory.
Exists	Determines is a directory exists.
GetCurrentDirectory	Gets the current working directory of the application.
GetDirectories	Gets the names of subdirectories in a specified directory.
GetDirectoryRoot	Return volume and/or root information for a path.
GetFiles	Returns the files in the specified directory.
GetLogicalDrives	Gets the logical drives on the computer in the form <driveletter>:\.
GetParent	Gets the parent directory of the specified directory.
Move	Moves a file or directory to a new location.
SetCurrentDirectory	Changes the current working directory to the path specified.

Examples

- Create a directory

```
Directory.CreateDirectory(@"\sub1");
```

- Get the files in a directory

```
string[] files =
  Directory.GetFiles(@"\windows");
foreach (string s in files)
      Console.WriteLine(s);
```

Examples – Contd.

- Get all the directories

```
string[] dirs =
  Directory.GetDirectories(@"\win
  dows");
foreach (string s in dirs)
     Console.WriteLine(s);
```

The DirectoryInfo Class

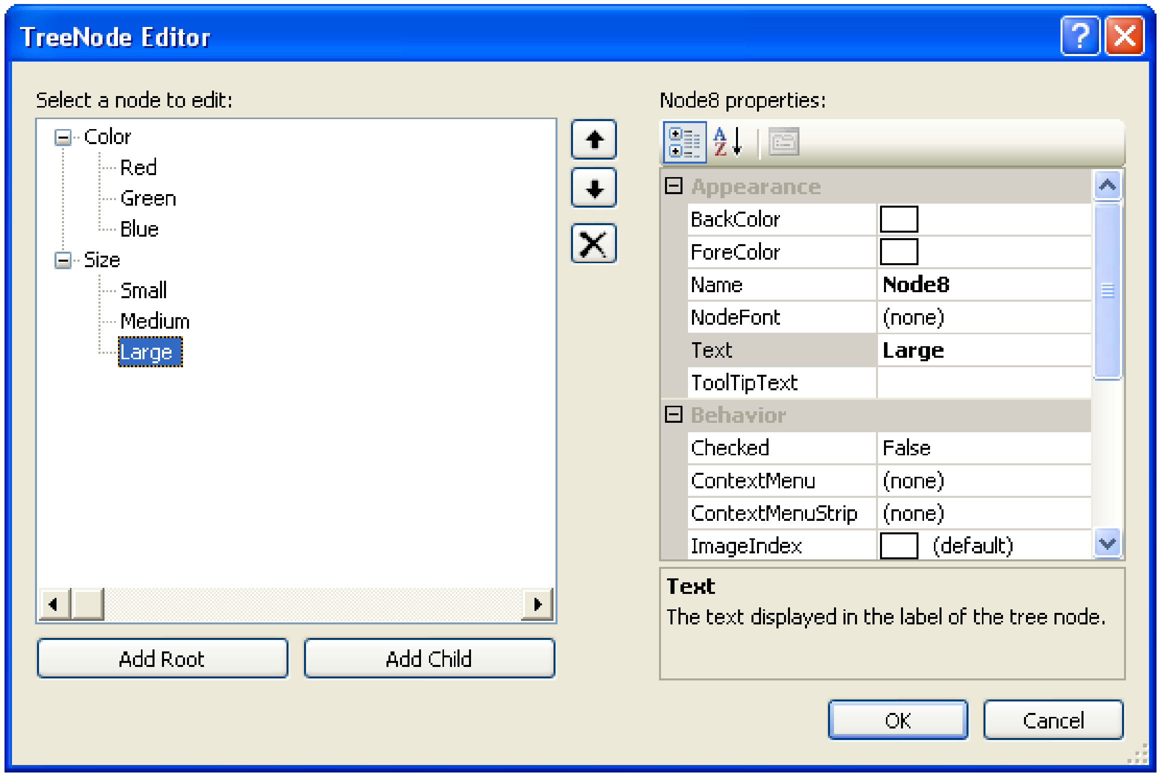
Method	*Description*
Create	Creates a directory.
CreateSubdirectory	Creates a subdirectory
Delete	Deletes the directory.
GetDirectories	Gets the subdirectorties of the associated directory.
GetFiles	Gets the files in the directory.
MoveTo	Moves a directory.

The Path Class

- Allows us to parse a path

Method	*Description*
ChangeExtension	Change a file's extension.
GetDirectoryName	Get the directory name part of a path.
GetExtension	Gets the file extension is it exists.
GetFileName	Gets the file name part of a path including the extension.
GetFileNameWithoutExtension	As above but strips the extension.
GetFullPath	Gets the absolute path for a file.
GetPathRoot	Gets the root of a path.

The TreeView Control



Appearance	
BackColor	⬜
ForeColor	⬜
Name	**Node8**
NodeFont	(none)
Text	**Large**
ToolTipText	
Behavior	
Checked	False
ContextMenu	(none)
ContextMenuStrip	(none)
ImageIndex	⬜ (default)

Text
The text displayed in the label of the tree node.

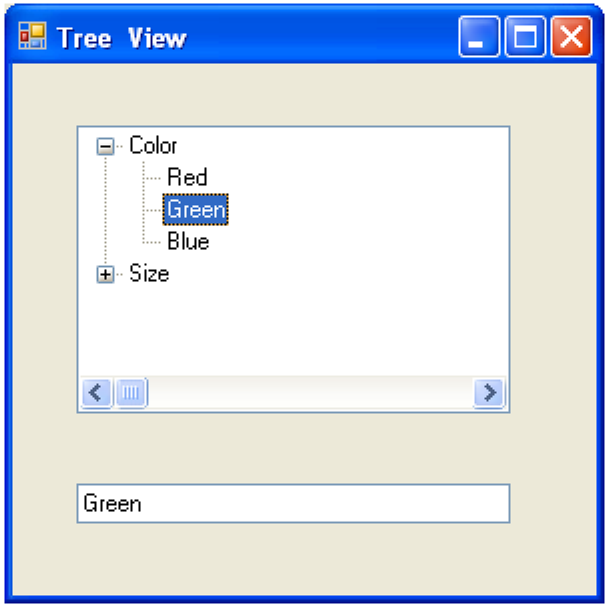
Example

```
TreeView1 - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace TreeView1
{
    public partial class Form1 : Form
    {
        public Form1()
```

Example – Contd.

```
        {
            InitializeComponent();
        }
        private void treeView1_AfterSelect(object sender,
              TreeViewEventArgs e)
        {
            selection.Text = e.Node.Text;
        }
    }
}
```

Output



Green

Directory Tree Example

```
DirectoryTree1 - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace DirectoryTree1
{
    public partial class Form1 : Form
    {
```

Directory Tree Example

```
public Form1()
{
    InitializeComponent();
}
private void display_Click(object sender,
       EventArgs e)
{
    tree.Nodes.Clear();
    try
    {
        tree.BeginUpdate();
        tree.Nodes.Add(path.Text);
        AddDirs(tree.Nodes[0], true);
```

Directory Tree Example

```
        tree.EndUpdate();
        }
        catch
        {
            return;
        }
    }
    private void tree_BeforeExpand(object sender,
         TreeViewCancelEventArgs e)
    {
        tree.BeginUpdate();
        AddDirs(e.Node, true);
```

Copyright © Thomas P. Skinner

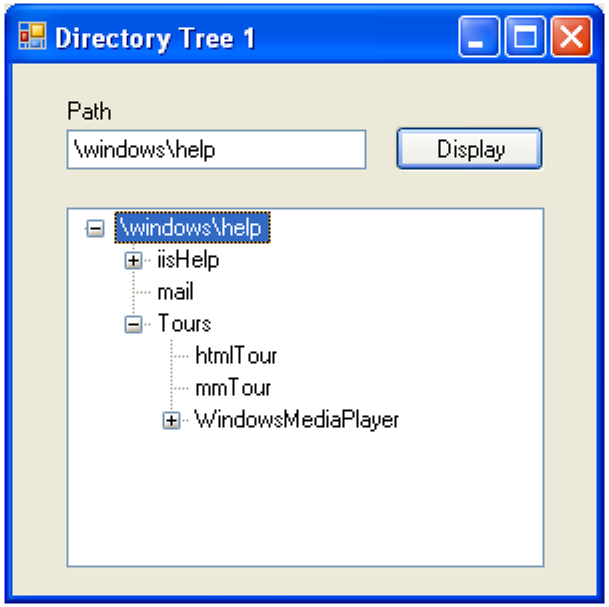
Directory Tree Example

```
          tree.EndUpdate();
        }
        private void AddDirs(TreeNode node, bool recurse)
        {
            node.Nodes.Clear();
            try
            {
                string[] dirs =
                  Directory.GetDirectories(node.FullPath);
                foreach (string dir in dirs)
                {
                    if(recurse)
```

Directory Tree Example

```
    AddDirs(node.Nodes.Add(Path.GetFileName(dir)),false);
    else node.Nodes.Add(Path.GetFileName(dir));
              }
            }
            catch
            {
                return;
            }
        }
    }
}
```

Output



```
\windows\help
  iisHelp
  mail
  Tours
    htmlTour
    mmTour
    WindowsMediaPlayer
```

Copyright © Thomas P. Skinner