

Re: file class

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-08/msg01905.html>

- *From:* "MikeJ" <vettes_n_jets@xxxxxxxxxxxxxx>
 - *Date:* Tue, 14 Aug 2007 19:14:14 GMT
-

No
I meant to replay to another question below...
sorry
MJ

"Ignacio Machin (.NET/ C# MVP)" <machin TA laceupsolutions.com> wrote in message news:e7gaDSq3HHA.2752@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

Hi,

Is this a question?

"MikeJ" <vettes_n_jets@xxxxxxxxxxxxxx> wrote in message [news:XMmwi.57408\\$5j1.28126@xxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:XMmwi.57408$5j1.28126@xxxxxxxxxxxxxxxxxxxxxxxxxxxx)

make a While loop

```
ofs = TextFileServer("somefile")
string srow
while (ofs=false)
{
srow=ofs.getRow();
Console.WriteLine(srow);
}
```

below is the actual class have fun just copy and past it should work
ok for ya
its also work in progress

```
using System;
using System.IO;

using System.Text;
using System.Windows.Forms;
using System.Data;
```

Re: file class

```
public class TextFileServer
{

protected object[,] _aStru;
private object[] _aColumns; //text row broke up into
columns
private int _ColumnCount=0; // column count
private string _Delimiter; //column delimiter
// private long iReccount;
private String _FileName;

protected string _cEol; //end of line

private long _Fsize; //File size
private FileStream _Fs; //file stream
private long[] _aOffsets; //array of row offsets

private byte[] aBytes; // for Reading Rows uses
in GetRow
protected int nRecordLen; // how long is a record
(line)
private long iRecno; //current Record number
protected string cRecord; //actual Row (columns not
removed)

private bool _RemoveQuotes = true;
private int _FixedRowSize=0; //fixed Bytes to Read

private int CombinedColumnLength;

/// <summary>
/// Base Constructure if used
/// filename and all required parameters must be set up by user
/// then call BeginRead()
/// </summary>
public TextFileServer()
{
this._ColumnCount = 0;//this._aStru.GetLength(0);
this.CombinedColumnLength = 0; //78;
}
/// <summary>
///
/// </summary>
/// <param name="FileName"></param>
public TextFileServer(string FileName)
{

this._FileName = FileName;
this._cEol = "\r\n";
this.BeginRead();
```

Re: file class

```
}  
/// <summary>  
/// File name and End of line Marker String  
/// </summary>  
/// <param name="FileName"></param>  
/// <param name="ceol"></param>  
public TextFileServer(string FileName,string ceol)  
{  
  
    this._FileName = FileName;  
    this._cEol = ceol;  
    this.BeginRead();  
  
}  
/// <summary>  
/// Filename and Array of Structure  
/// ColumnName, Type, Length, Decimal  
/// object[] 2 Dimensional  
/// </summary>  
/// <param name="FileName"></param>  
/// <param name="aStru"></param>  
public TextFileServer(string FileName, object[,] aStru)  
{  
  
    this.Structure= aStru;  
    this._FileName = FileName;  
    this.BeginRead();  
  
}  
/// <summary>  
/// filename  
/// Structure for columns object[] 2 dimensional Name,type,length,dec  
/// column Delimiter string  
/// </summary>  
/// <param name="FileName"></param>  
/// <param name="aStru"></param>  
/// <param name="delimiter"></param>  
public TextFileServer(string FileName, object[,] aStru,string  
delimiter)  
{  
  
    this._aStru = aStru;  
    this._Delimiter = delimiter;  
    this._FileName = FileName;
```

Re: file class

```
this.BeginRead();

}
public TextFileServer(string FileName, int FixedRecordLen)
{

this._FixedRowSize=FixedRecordLen;
this.nRecordLen = this._FixedRowSize;
this._cEol = null;

this._FileName = FileName;
this.BeginRead();

}
/// <summary>
/// Call this if you Construct class TextFileServer()
/// the other constructors call this themselves
/// </summary>
public void BeginRead()
{

this.CombinedColumnLength = 0; //78;
this._Fs= new FileStream(this._FileName, FileMode.Open,
FileAccess.Read, FileShare.ReadWrite);
//this._Fs = dvFileIO.FileOpen(this._FileName);
/*
if (! (this._aStru == null))
{
this._ColumnCount = this._aStru.GetLength(0);
this._aColumns = new object[this._ColumnCount];
for (int x = 0; x < this._ColumnCount; x++)
{
this.CombinedColumnLength += (int)this._aStru[x, 2];

}
}
*/
//find end of line markers

if (this._cEol == null) this._cEol = "\r\n";
//get line information
this.FindEol();
//this.FileName = FileName;
// this._cEol = cEol;
// this.nEol = cEol.Length;
//open file
//oFs = new FileStream(this.FileName, FileMode.Open,
FileAccess.Read);
```

Re: file class

```
//get filesize
this._Fsize = this._Fs.Seek(0, SeekOrigin.End);
//move back to the top
this._Fs.Seek(0, SeekOrigin.Begin);
this.iRecno = 0;

//this.iBytes2Read = this.nRecordLen;
/*
for (int x = 0; x < aStru.GetLength(0); x++)
{
this.iBytes2Read+= (Int32)aStru[x,2];
}
*/
// add to buffer if not fixed width
/*
if (cEol != "")
{

this.iBytes2Read = this.nRecordLen + 100; // add 100 bytes to
make sure we read enough
}
*/
this.LoadOffSets();

}
//-----
/// <summary>
/// Finds End of line Markers in Files and reads a few lines to
determine a possible
/// widest line in file
/// </summary>
public void FindEol()
{
this._Fs.Seek(0, SeekOrigin.Begin);
string cBuffer;
int npos;
byte[] aBytes = new byte[255];
char[] Sep = new Char[] { ' ' };
long iBytesRead;
iBytesRead = this._Fs.Read(aBytes, 0, aBytes.Length);
// we have a buffer with more than 1 line find eol and keep the
longest line
cBuffer = ASCIIEncoding.ASCII.GetString(aBytes);
this._Fs.Seek(0, SeekOrigin.Begin);
//End of line built from constructor or user manual
if ((npos = cBuffer.IndexOf(this._cEol)) > 0)
{
```

Re: file class

```
//default just get Recordlen
Sep = this._cEol.ToCharArray();
}

//below this point is all known endof line markers based on No
//end of line marker set during constructor

else if ((npos = cBuffer.IndexOf("\r\n")) > 0) //13/10
{
    this._cEol = "\r\n";

    Sep = new char[] { '\r', '\n' };

}
else if ((npos = cBuffer.IndexOf("\r")) > 0) //13 only
{

    this._cEol = "\r";

    Sep = new char[] { '\r' };

}
else if ((npos = cBuffer.IndexOf("\n")) > 0) //10 only
{

    this._cEol = "\n";

    Sep = new char[] { '\n' };

}
else if ((npos = cBuffer.IndexOf("~")) > 0) //ansi files
{
    this._cEol = "~";

    this.nRecordLen = 106;
    npos = -1;

}
// no end of line markers found set to a fixed length hope for the
bst
else
{
    npos = -1;
    this._cEol = "";

    this.nRecordLen = 78;
}
}
```

Re: file class

```
//find the longest line from the Buffer we read in
if (npos > 0)
{
string[] ainfo = cBuffer.Split(Sep);
this.nRecordLen = 0;
for (int x = 0; x < ainfo.Length; x++)
{
if (ainfo[x].Length > this.nRecordLen)
{
this.nRecordLen = ainfo[x].Length;
}
}
}
//check astru (combinded column lengthes vs nrecordlen)
if (this.CombinedColumnLength>=this.nRecordLen)
{
this.nRecordLen=this.CombinedColumnLength;
}

}

/// <summary>
/// this is the workhorse it loads all the end of line byte positions
/// from the file into the _aoffsets long[] array
/// </summary>
public void LoadOffSets()
{
// get filesize

long iSeekBytes;
int iBytes2Read;
long iBytesRead;

int ipos;
int iWidest; // widest row in file
string oBufferIn="";
// this.ctrlz = (char)26; //ctrl_z end of file marker for some
files

//this.iBytes2Read=1024;

long nCntr;
long nOffset;

iSeekBytes = 0;
//aproximate how many array elements we need
// i want more than required always, so there is only 2 resizes
```

Re: file class

```
//1st = create the long[], 2nd = Resize back to what was needed
//take recordlen and divide by 6 , take that result subtract it
from recordlen
int i=0;
int r=0;
long nSize=0;

if (this._FixedRowSize==0)
{
i = (this.nRecordLen / 6); //--50;

r = this.nRecordLen -i ;
nSize = this._Fsize / r;
iBytes2Read = this.nRecordLen + (this.nRecordLen / 2);

}
else
{
this.nRecordLen=this._FixedRowSize;
nSize = this._Fsize / this.nRecordLen;
iBytes2Read = this.nRecordLen;
}
// empty file this just stops crash...oops
if (nSize <= 0)
{
nSize = 1;

}

//long nSize =
this.Fsize/(this.iBytes2Read-Math.Min(this.iBytes2Read,this.iBytes2Read/4));
dvConsole.Print(DateTime.Now.ToString()+" Array Size " +
nSize.ToString()+" Bytes2Read "+iBytes2Read.ToString());

this._aOffsets = new long[nSize];

byte[] aReadBytes =new Byte[iBytes2Read];

nCnt = 0;
iSeekBytes = 0;

this._aOffsets[nCnt] = 0;
iWidest = 0; //holds the widest row in the file;
ipos = 0;
long nCr = 0;

while ((iBytesRead = this._Fs.Read(aReadBytes, 0, iBytes2Read)) >
0)
{
```

Re: file class

```
if ((nCntr % 5000) == 0)
{
Application.DoEvents();
}

oBufferIn = UTF8Encoding.ASCII.GetString(aReadBytes);

//ALWAYS FIND END OF LINE UNLESS THIS HAS A VALUE
GREATER THAN
ZERO

//class Not Constructed with fixed Size rows
if (this._FixedRowSize ==0)
{
ipos = oBufferIn.IndexOf(this._cEol);
//check if embedded chr 10 is in the file as line feeds

if ((ipos > 0) && (oBufferIn.IndexOf("\n") == 0))
{
iSeekBytes += 1;
}

else if (ipos > 0)
{

nCr++;
// oBufferOut = oBufferIn.Substring(0, ipos - 1);
iSeekBytes += ipos + (this._cEol.Length) ; //nEol);
//get past chr(10)
// Console.ReadKey();
}

else
{
// oBufferOut = oBufferIn;
iSeekBytes += oBufferIn.Length;
ipos = oBufferIn.Length;
}
}
//we have fixed Row Size
else
{

iSeekBytes += oBufferIn.Length;
ipos = oBufferIn.Length;
// dvConsole.PrintCR(oBufferIn.Substring(0,40)+" iseek
"+iSeekBytes.ToString());
```

Re: file class

Re: file class

```
}
//grab the offset noffset
nOffset = (long)this._Fs.Seek(iSeekBytes, SeekOrigin.Begin);
// *****
// save the widest row ipos is where endof line found
if (ipos>0 && iWidest<ipos) iWidest=ipos;
// *****

// count Rows
nCtr++;
//set the offset to the array element the else part of the if
should never happend
//but somtimes it will we have to add a new element to the
array (resize)
if (nCtr < nSize) // use nsize instead of property -> Length
this._aOffsets.Length)
{
this._aOffsets[nCtr] = nOffset;
}
else
{
nSize++;
Array.Resize(ref this._aOffsets,this._aOffsets.Length +
1);
this._aOffsets[nCtr] = nOffset;
}

// Console.ReadKey();
}

// resize the Array to actual Size Needed
if (nCtr < this._aOffsets.Length)
{

Array.Resize(ref this._aOffsets, (int)nCtr);

}

dvConsole.Print(DateTime.Now.ToString()+" Alen " +
this._aOffsets.Length.ToString());
//load a 1st row and columns
//set Widest Row base on local variable
if (this._FixedRowSize == 0 && this.CombinedColumnLength <
iWidest)
{
this.CombinedColumnLength = iWidest;
}
}
```

Re: file class

10

Re: file class

```
else
{
this.CombinedColumnLength = this._FixedRowSize;
}
this.iRecno = this._aOffsets.Length - 1;

} // method
/// <summary>
/// Set filename if Constructed with no params
/// </summary>
public string Filename
{
set
{
this._FileName = value;

}
get
{
return this._FileName;
}
}
/// <summary>
/// Set the End of Line marker
/// use when construction with no params
/// </summary>
public string Crlf
{

set
{
this._cEol = value;
}
}
/// <summary>
/// Delimiter use for Column Delimiter
/// </summary>
public string Delimiter
{

set
{
if (value == "")
{
this._Delimiter = null;
// this.FixedSize = true;
}
else
{
this._Delimiter = value;
}
}
}
```

Re: file class

```
}
}
/// <summary>
/// Sets up the Properties for a Column Structure based on a object[]
/// Name,type,length,dec
/// 2 dimensional array {{"name","C",20,0},
/// {"Id","N",10,0}}
///
/// </summary>
public object Structure
{
    set
    {
        this._aStru=(object[,])value;
        if (!(this._aStru == null))
        {
            this._ColumnCount = this._aStru.GetLength(0);

            this._aColumns = new object[this._ColumnCount];
            for (int x = 0; x < this._ColumnCount; x++)
            {
                this.CombinedColumnLength += (int)this._aStru[x, 2];
            }
        }
    }
    get
    {
        return this._aStru;
    }
}
/// <summary>
/// Some Files to be parsed based on columns have or may not have
/// Quoted Character columns
/// i Default to Remove Quotes true
/// </summary>
public bool StripQuotes
{
    set
    {
        this._RemoveQuotes = value;
    }
}
/// <summary>
/// Returns column count based on Array structure
/// </summary>
public int ColumnCount
```

Re: file class

12

Re: file class

```
{
get
{

return this._ColumnCount;
}
set
{
this._ColumnCount = value;
}
}
/// <summary>
/// Set a Fixed Row Size
/// </summary>
public int FixedRowSize
{
set
{
this._FixedRowSize=value;
}
}
/// <summary>
/// Check for End of File should be used external with While loop
/// </summary>
public bool Eof
{
get
{
if (this.iRecno >= (this._aOffsets.Length))
{

return true;
}

return false;
}
}
/// <summary>
/// check for beginning of file
/// </summary>
public bool Bof
{
get
{
if (this.iRecno < 0)
{
this.iRecno = 0;
return true;
}
return false;
}
```

Re: file class

13

Re: file class

```
}
}
/// <summary>
/// Moves to Next Row in file
/// </summary>
/// <returns></returns>
public long Skip()
{
return this.Skip(1);
}
/// <summary>
/// Moves to Next Row in File use this instead of Skip
/// </summary>
/// <param name="n2Skip"></param>
/// <returns></returns>
public long Skip(int n2Skip)
{

if (n2Skip >= 0)
{
this.iRecno += n2Skip;
}
else
{
long iRec = Math.Abs(n2Skip);
this.iRecno -= iRec;
}

return this.iRecno;

}
public long GoTop()
{

this.iRecno = 0;
return this.iRecno;

}
/// <summary>
/// go to specified row pos in a file
/// </summary>
/// <param name="Recno"></param>
/// <returns></returns>
public long GoTo(long Recno)
{
if (Recno > (this._aOffsets.Length - 1))
{
this.iRecno = this._aOffsets.Length - 1;
}
else
{
```

Re: file class

Re: file class

```
this.iRecno = (Recno-1);
}
return this.iRecno;
}
/// <summary>
/// get current row position in file
/// </summary>
public long RowNum
{
    get
    {
        return iRecno+1;
    }
}
/// <summary>
/// go to last row in file
/// </summary>
/// <returns></returns>
public long GoBottom()
{

    this.iRecno = this._aOffsets.Length - 1;
    return this.iRecno;
}
/// <summary>
/// total count of rows in file
/// </summary>
/// <returns></returns>
public long RowCount()
{
    return this._aOffsets.Length;
}

/// <summary>
/// Getrow does the actually reading of 1 row from the table
/// if columns are set then they will be created also
/// </summary>
/// <returns></returns>
public string GetRow()
{

    long nStart=this._aOffsets[iRecno];
    //byte[] aReadBytes;

    this._Fs.Seek(this._aOffsets[this.iRecno], SeekOrigin.Begin);
    int nCnt=0;

    //if (this.iRecno==0)
    //{
    // nCnt = (int)(this._aOffsets[this.iRecno + 1] -
    this._aOffsets[this.iRecno]);
```

Re: file class

15

Re: file class

```
// nCnt--=1;
// aReadBytes = new Byte[nCnt];
// this._Fs.Read(aReadBytes, 0, nCnt - 1);
//}
if (this.iRecno>-1 && (this.iRecno<this._aOffsets.Length-1))
{
nCnt = (int)(this._aOffsets[this.iRecno + 1] -
this._aOffsets[this.iRecno]);
nCnt -= 1;
this.aBytes = new Byte[nCnt];
this._Fs.Read(this.aBytes, 0, nCnt - 1);
this.cRecord = UTF8Encoding.ASCII.GetString(this.aBytes);
}
else
{
//last row in file
nCnt = this.CombinedColumnLength; //this.nRecordLen - 1;
//nCnt = this.iBytes2Read-1;
aBytes = new Byte[nCnt];
this._Fs.Read(this.aBytes, 0, nCnt - 1);
this.cRecord = UTF8Encoding.ASCII.GetString(this.aBytes);
this.cRecord = this.cRecord.Substring(0,
this.cRecord.IndexOf(this._cEol) - 1);

}
// byte[] aReadBytes = new Byte[nCnt];
// this._Fs.Read(aReadBytes, 0, nCnt-1);
//this.cRecord = UTF8Encoding.ASCII.GetString(this.aBytes);

//pad the record to meet column specifications if columns
Specified
if (this._ColumnCount > 0)
{
this.cRecord =
this.cRecord.PadRight(this.CombinedColumnLength);

// fill the Columns
//*****

int npos = 0;
int x = 0;
//dvConsole.Print("Fill columns "+this.cRecord.Length);
if (this._Delimiter == null)
{

//dvConsole.PrintCR(this.cRecord);
for (x = 0; x < this._ColumnCount; x++)
{
```

Re: file class

```
this._aColumns[x] = this.cRecord.Substring(npos,
(int)this._aStru[x, 2]);
// dvConsole.PrintCR("ccol "+this._aColumns[x]);
npos += (int)this._aStru[x, 2];
//dvConsole.Print("acols "+this._aColumns[x]);
}

}
else
{
npos = 0;
int nSplit = 0;
string cCol = "";
string cRow;
if (this._RemoveQuotes)
{
cRow = this.cRecord.Replace("'", ' ');
}
else
{
cRow = this.cRecord;
}
for (x = 0; x < this.ColumnCount; x++)
{
nSplit = cRow.IndexOf(this._Delimiter);
//dvConsole.Print("nsplit " + nSplit.ToString());
if (nSplit == -1) // last column
{
cCol = cRow.Substring(0);
}
else
{
cCol = cRow.Substring(0, nSplit);

cRow = cRow.Replace(cCol + this._Delimiter, "");
}

cCol = cCol.Trim();

this._aColumns[x] = cCol.PadRight((int)this._aStru[x,
2], ' ');

// dvConsole.PrintCr("ccol " + cCol);
// dvConsole.Print(cRow);
//this._aColumns[x] = cCol;
//SIZE EACH COLUMN

}
}
```

Re: file class

17

Re: file class

```
}
//
*****

return this.cRecord;
}

/// <summary>
/// Retrieve a Column by Column Position
/// </summary>
/// <param name="nPos"></param>
/// <returns></returns>
public object ColumnGet(int nPos)
{

return (this.ColumnGet(this._aStru[nPos], 0).ToString());
}
/// <summary>
/// Gets column Position by column Name
/// </summary>
/// <param name="cName"></param>
/// <returns></returns>
public object ColumnGet(string cName)
{
int x;
object oRet = null;

for (x = 0; x < this.ColumnCount; x++)
{

if (this._aStru[x, 0].ToString() == cName.ToString())
{

oRet = this._aColumns[x];
}
}

return oRet;
}
public void Close()
{
this._aOffsets = null;
this._Fs.Close();
}
} //fileserver
```

Re: file class