Rewarding Learning

AS1 LEVEL

FACT FILES

Software Systems Development

For first teaching from September 2013

For first AS award in Summer 2014

For first A Level award in Summer 2015

Introduction to Object Oriented Development Part (vii) – Managing Input/Output

software systems development

FACT FILE

software systems development

Part (vii) – Managing Input/Output

Learning Outcomes

Students should be able to:

- input data from the command line prompt control screen output
- evaluate the role of different file types in an object oriented environment including:-
  - text files;
  - binary files;
  - object files (Serialization)

Content

Input Data from the Command Line Prompt

Use the methods of class Console

Console.read()
Console.Readline()
Console.Write()
Console.Writeline()

Use **Console.SetCursorPosition**(left, top); //to position the cursor on screen before input or output.

e.g. Console.SetCursorPosition(5, 2);

Use Console.Clear() to clear the screen buffer.

Console - Input

Data is input as a string and parsed to the required format, for example number and date.

Console.Read() - terminated by white space
Console.ReadLine() - terminated by return

Examples of input:-

```
DateTime dob;
string str;
int number;
bool ok = false;
//prompt for input

do{
        ok = false;
        Console.SetCursorPosition(10, 2); Console.WriteLine("Type an integer:");
        // Read string from console
        Console.SetCursorPosition(30, 2); str = Console.ReadLine();
        if (int.TryParse(str, out number)) // Try to parse the string as an integer
        {
                ok = true;
        }
        else
        {
                Console.SetCursorPosition(10, 20);
                Console.WriteLine("Error: Not an integer!  Please re-enter");
        }
}while(!ok);
//Clear error message
Console.SetCursorPosition(10, 20); Console.WriteLine("                          ");


//Enter date
do{
        ok = false;
        Console.SetCursorPosition(10, 4);
        Console.WriteLine("Enter the order date and time (dd/mm/yyyy hh:mm
AM/PM)");
        Console.SetCursorPosition(66, 4);
        if (DateTime.TryParse(Console.ReadLine(), out dob))
                ok = true;
        else
        {
                ok = false;
                Console.SetCursorPosition(10, 20);
                Console.WriteLine("Error: Not in date format!  Please re-enter");
            }
        }while(!ok);
        //Clear error message
        Console.SetCursorPosition(10, 20); Console.WriteLine("                    ");

        Console.SetCursorPosition(10, 6);
        Console.WriteLine("  " + dob.ToShortDateString());
```

Console Output

Console.Write(…)
Console.WriteLine(…)

There are many overloads.

```
Console.WriteLine(6);                    //integer
Console.WriteLine("Type an integer:");   //string
Console.WriteLine(ok);                   //boolean
```

Elements can be concatenated using String.Concat or by using the overloaded + operator.

```
Console.WriteLine(string.Concat(str, 6));
Console.WriteLine("Type an integer:" + 6);
```

Include \n or \t in the string to take a new line or tab.
Tabs work best with similar length strings.

A very useful method is to use a **format string** for more precise spacing and formatting of elements.

{0} gives the position in the list, of the argument to be output and a letter denotes the format.

```
Console.WriteLine("{0}, {1}, {2}",    value1,    value2,
value3);
```

Sample formats:-

```
Console.WriteLine(
    "(C) Currency: . . . . . {0:C}\n" +
    "(D) Decimal:. . . . . {0:D}\n" +
    "(F) Fixed point:. . . . . {1:F}\n" +
    "(G) General:. . . . . {0:G}\n" , 123,123.65);


Console.WriteLine("Standard DateTime Format Specifiers");
 Console.WriteLine(
    "(d) Short date: . . . . . {0:d}\n" +
    "(D) Long date:. . . . . {0:D}\n" +
    "(t) Short time: . . . . . {0:t}\n" +
    "(T) Long time:. . . . . {0:T}\n" +
    "(f) Full date/short time: . . {0:f}\n" ,dob);
```

```
Console.WriteLine("Bill total:\t{0,8:c}", billTotal);       //  width of 8 for the output value
```

Screen Design.

Be consistent with screen design.

For example error messages should be displayed at the bottom of the screen. Write a function which will clear this area of the screen and call when appropriate.

If the file contents are extensive display a section at a time and use a function to continue with the display of the next section.

For example KeyPress()

displays the message :- press any key to continue
clears the screen and
outputs an appropriate heading.

NOTE It can save a lot of time and effort to design the data input screen or update screen as a text file, read it in to the program as a stream of characters and display on screen.

Console.SetCursorPosition is then only required for the input of the values.

File Input/Output With Streams From System.IO namespace

File Processing allows data to be stored permanently onto backing store which can be read back at a later date as required. Text files are used for character based data such as simple reports. Binary files are used for user-defined types - objects.

The **File** denotes the Path and name of the file to be used and the FileMode.

File.OpenRead	Opens file for reading
File.Open	Opens file for writing with a following mode
FileMode.Create	Opens file and overwrites existing file if present
FileMode.CreateNew	Opens file and throws exception if file already exists
FileMode.Append	Opens file and moves to end of file for writing

The **Stream** objects allow data to be written to and from backing storage in different formats.

Two such formats are **character** and **binary**:

StreamReader and **StreamWriter** transfer **character** based data from/to text files.

BinaryFormatter allows transfer of **Serializable** data (user defined types) as a simple binary data stream.

Serialization stores the object member variable values to disk. Deserialization is the reverse of serialization. It is a process of reading objects from a file where they have been stored.

NOTE File-handling operations should always be protected with the appropriate exception-handling code, for example, IOException, SerializationException

The Stream objects are capable of one or more of the following:

Reading - The transfer of data from a stream into a data structure;

Writing - The transfer of data from a data structure into a stream;

Seeking - The querying and modifying of the current position within the stream.

Transferring Character Based Data

- StreamReader and **StreamWriter**

Data can be streamed as characters using the StreamReader and the StreamWriter, effectively a character layer on top of a raw byte FileStream object.

The following example shows how a report is created from an array of car objects using the WriteLine method of the StreamWriter object.

```
  static void Main(string[] args)
  {
                    Car[] cars = new Car[3];
                     PopulateCars(cars);
   //write to character based file -  CarReport.txt
                    WriteToReport(cars);
                    Console.ReadLine();
  }

public static void PopulateCars(Car[] cars)
{
   cars[0] = new Car("Vauxhall", "RUI 2345", 10);
   cars[1] = new Car("Audi A3", "SUI 6655", 5);
   cars[2] = new Car("Ford", "PUI 9381", 2);
}

public static void WriteToReport(Car[] cars)
{
   FileStream s = new FileStream("CarReport.txt", FileMode.Create);
   StreamWriter sw;

   try
   {
      // write report
      sw = new StreamWriter(s);
      sw.WriteLine();
      sw.WriteLine("{0,40}", "Car Report");
      sw.WriteLine();
      sw.WriteLine();
      for (int x = 0; x < cars.Length; x++)
      {
```

```
            sw.WriteLine(  cars[x].toString()  );
        }
        sw.WriteLine();
        sw.WriteLine("       Number of Makes  - {0}", cars.Length);
        sw.Close();
      }
      catch (Exception e)
      {
        Console.WriteLine(  e.Message );
      }

  }
```

Output from the program

```
                    Car Report


Vauxhall          :  RUI 2345  010 in stock
Audi A3           :  SUI 6655  005 in stock
Ford              :  PUI 9381  002 in stock

    Number of Makes  - 3
```

NB The text file CarReport.txt can be found in the ProjIO/ProjIO/bin/debug folder

Reading from a text file uses the StreamReader :-

```
FileStream s = new FileStream("Menu.txt", FileMode.Open);
StreamReader sr;
string str;
while( ( str = sr.ReadLine()  )  != null)
{
        Console.Writeline( str);
}
Sr.Close();
```

Transfer Of Binary Data For User Defined Types

BinaryFormatter allows transfer of **Serializable** data. The Binary Formatter object is more suited to smaller files.

The user class must be defined as Serializable eg.

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

```
using System.Text;


namespace ProjIO
{
   [Serializable]
   class Car
   {
      private string make;
      private string regNo;
      private int qty;

      public Car(string make, string regNo, int qty)
      {
         Make = make;
         RegNo = regNo;
         Qty = qty;
      }
      public string Make
      {
         set { make = value; }
         get { return make; }
      }
      public string RegNo
      {
         set { regNo = value; }
         get { return regNo; }
      }
      public int Qty
      {
         set { qty = value; }
         get { return qty; }
      }
      public string toString()
      {
         return String.Format("   {0,-20}  :  {1}  {2:D3} in stock ", make, regNo, qty);
      }
   }
}
```

The project must include the following namespaces to support the Input/Output process.

```
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
```

The following code reads and writes an **array object** to a file.

```
class Program
   {
      static void Main(string[] args)
      {
         Car[] cars = new Car[3];
         PopulateCars(cars);


         WriteToFile(cars);
         ReadFromFile(cars);


         //display car details

         Console.WriteLine("\t\t\tCar Details for {0} makes\n\n", cars.Length);
         foreach (Car car in cars)
         {
            Console.WriteLine(car.toString());
         }

         Console.ReadLine();
      }

      public static void PopulateCars(Car[] cars)
      {

         cars[0] = new Car("Vauxhall", "RUI 2345", 10);
         cars[1] = new Car("Audi A3", "SUI 6655", 5);
         cars[2] = new Car("Ford", "PUI 9381", 2);

      }
      public static void WriteToFile(Car[] cars)
      {
         Stream sw;
         BinaryFormatter bf = new BinaryFormatter();

         try
         {
            // write cars array object to file
            sw = File.Open("Cars.bin", FileMode.Create);
            bf.Serialize(sw, cars);
            sw.Close();
         }
         catch (SerializationException e)
         {
            Console.WriteLine("" + e.Message);
         }

      }
```

```
        public static void ReadFromFile(Car[] cars)
        {
            Stream sr;
            BinaryFormatter bf = new BinaryFormatter();
            //read car details from file to array object
            try
            {
                sr = File.OpenRead("Cars.bin");
                cars = (Car[])bf.Deserialize(sr);

                sr.Close();
            }
            catch (Exception e)
            {
                Console.WriteLine("" + e.Message);
            }
        }
}
```

The binary file Cars.bin can be found in the ProjIO/ProjIO/bin/debug folder

Where an array of objects is created large enough to allow new objects to be added to it, the array should be read from and written to as **individual** objects.

In reading a file the current position of the stream pointer is compared to the length of the stream. If less, a car object is read from the file, and the count of car objects incremented.

```
static void Main(string[] args)
        {
            int size = 0;
            Car[] cars = new Car[20];

            //read existing car file into car array and return number of cars
            size =  ReadFromFile(cars);

            // add another car to the array
            cars[size] = new Car("Audi A4", "RUI 9934", 5);
            size++;

            //display car details
            Console.WriteLine("\t\t\tCar Details for {0} makes\n\n", size);
            for (int x = 0; x < size; x++ )
            {
                Console.WriteLine(cars[x].toString());
            }

            //write expanded array back to file
            WriteToFile(cars, size);

            Console.Read();
        }

        public static void WriteToFile(Car[] cars,  int size)
        {
```

```
        Stream sw;
        BinaryFormatter bf = new BinaryFormatter();

        try
        {
          // Use Create to open file and write first car details to file
          //  - will overwrite previous existing details
           sw = File.Open("Cars.bin", FileMode.Create);
           bf.Serialize(sw, cars[0]);
           sw.Close();
           //open as append and write rest of array
           sw = File.Open("Cars.bin", FileMode.Append);
           for (int x = 1; x < size; x++)
           {
              bf.Serialize(sw, cars[x]);
           }
           sw.Close();
        }
        catch (IOException e)
        {
           Console.WriteLine("" + e.Message);
        }

      }

     public static int ReadFromFile(ref Car[] cars)
      {
                //The array address is passed by reference (default- array address is
                passed by value)
                //  as the read puts data into an area in memory  and
                //  then the array address must be changed to point to this new area.
        int size=0;
        Stream sr;
        try
        {
           sr = File.OpenRead("Cars.bin");
           BinaryFormatter bf = new BinaryFormatter();
           //read car details from file – while stream position  is less than stream
length
           try
           {
              while (sr.Position < sr.Length)
              {
                 cars[size] = (Car)bf.Deserialize(sr);
                 size++;
              }
              sr.Close();
           }
           catch (SerializationException e)
           {
              sr.Close();
              return size;
           }
           return size;
        }
        catch (IOException e)
```

```
                {
                    cars[size] = (Car)bf.Deserialize(sr);
                    size++;
                }
                sr.Close();
            }
            catch (SerializationException e)
            {
                sr.Close();
                return size;
            }
            return size;
        }
        catch (IOException e)
        {
            Console.WriteLine("\n\n\tFile not found" +e.Message);
        }
        return size;
    }
```

Updating Files

Data can be read, updated and added to a binary file directly. The position of an object within a file can be determined from the **stream pointer.** The **seek** command moves the stream pointer to a given position on the file.

This is possible only if objects written to the file are all the same length. This applies to the string fields. The relevant classes must ensure that all string fields are padded to the same length. This is taken care of by the **Property** methods and is hidden from the user.

The Car class in the example must be modified as :-

```
[Serializable]
    class Car
    {
        // for use with files that can be updated using seek
        // maximum String lengths allowed :  make -20 chars, regNo -10 chars

        private string make;
        private string regNo;
        private int qty;

        public Car(string make, string regNo, int qty)
        {
            Make = make;
            RegNo = regNo;
            Qty = qty;
        }
        public string Make
        {
            set
            {       make = value;
                    make = Pad(make, 20);
            }
            get { return make; }
        }
```

```
public string RegNo
{
   set { regNo = value;
         regNo = Pad(regNo, 10); }
   get { return regNo; }
}
public int Qty
{
   set { qty = value; }
   get { return qty; }
}
public string toString()
{
   return String.Format("  {0,-25} {1,-25}  {2} \n", make, regNo, qty);
}

private string Pad(string s, int max)
{
   // truncate string if too long, pad string to maximum length if shorter
   s = s.Trim();
   if (s.Length > max)

      return s.Substring(0, max);
   else
   {
      for (int x = s.Length; x < max; x++)
      {
         s = s+ " ";
      }
      return s;
   }
}
```

Update Data On File

The file open statement indicates that the file will be both read from and written to.

```
Stream  sr = File.Open("MoreCars.dat", FileMode.Open, FileAccess.ReadWrite);
```

Read the file sequentially until the required object is found or the end of the file is reached. Use sr.Position and sr.Length to check when end of file is reached, for example,

```
while (sr.Position < sr.Length && found == false)
{
   pos = sr.Position;                  // note position of object in file
   car = (Car)bf.Deserialize(sr);      //reads car object
```

Modify the object if found and use sr.Seek to move to the object's position in the file stream.

```
sr.Seek(pos, SeekOrigin.Begin);       // number of bytes from start of file stream
```

The stream must be closed to ensure that the stream is flushed and the object updated correctly.

The following code shows how the car details can be modified and written back to the same position in the file. The position of an object is noted before the car details are read and this is used by the seek method later to write the modified object back.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Collections;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

namespace ProjIO
{


    class Program
    {
        // operations using file only
        // display file contents
        // update car details :- seek car object, update and write back to file

        static void Main(string[] args)
        {
            string reqRegNo;
            bool found;
            /*
            //Create file initially from array to write individually as fixed length Car objects
            //Strings e.g. make and RegNo are padded to their maximum size to ensure
           fixed length objects
            //See modified Car class to ensure this

            int size = 0;
            Car[] cars = new Car[20];
            size = PopulateCars(cars);
            WriteToFile(cars, size);
             */

            // display car details from file
            Console.Clear();
            Console.SetCursorPosition(5, 2);
            Console.WriteLine("\t\t\tCar Details \n\n");
            Console.WriteLine(String.Format("  {0,-25} {1,-22} {2,-12}\n", "Make", "Reg
            No.", "No in Stock"));
            displayCars();
```

```
    // Update car details
    Console.SetCursorPosition(5, 15);
    Console.Write("   Enter Reg No to be updated : ");
    reqRegNo = Console.ReadLine();

    //find car in file and update
    found = updateCar(reqRegNo);
    if (found)
    {
        Console.WriteLine("\n\t\t\t Updated Car Details \n\n");
        displayCars();

    }
    else
        Console.WriteLine(" Reg No not found in Car file ");
    Console.Read();
}

public static void DisplayCars()

{
    Stream sr;
    Car car;
    try
    {
        sr = File.OpenRead("MoreCars.dat");
        BinaryFormatter bf = new BinaryFormatter();
        //read car details from file
        try
        {
            while (sr.Position < sr.Length)
            {
                car = (Car)bf.Deserialize(sr);
                Console.WriteLine(car.toString());
            }
        }
        catch (SerializationException e)
        {
            Console.WriteLine("\n\t"+ e.Message);
        }
        sr.Close();
        sr.Dispose();
    }
    catch (IOException e)
    {
        Console.WriteLine("\n\n\t" +e.Message);
    }

}

public static bool UpdateCar(string reqRegNo)
{
```

```
    //open file for both read and write
    long pos = -1;
    bool found = false;
    Car car;
    Stream sr;
    try
    {
        sr = File.Open("MoreCars.dat", FileMode.Open, FileAccess.ReadWrite);
        BinaryFormatter bf = new BinaryFormatter();
        //read car details from file
        try
        {
            while (sr.Position < sr.Length && found == false)
            {
                pos = sr.Position;                  // note position in file
                car = (Car)bf.Deserialize(sr);      //reads car object
                // Console.WriteLine(" object position   " + pos + car.toString());
                // change Make in stock if found


                if (car.RegNo.Trim().CompareTo(reqRegNo.Trim()) == 0)
                {
                    found = true;
                    //enter changes and validate  :-
                    car.Make = "Test increased length";
                    // write back to position on file
                    sr.Seek(pos, SeekOrigin.Begin); //from start of file seek position of
                    car object
                    bf.Serialize(sr, car);
                }
            }
            sr.Close();
        }
        catch (SerializationException e)
        {
            Console.WriteLine("\n\t" + e.Message);
        }
        sr.Close();
        sr.Dispose();
    }
    catch (IOException e)
    {
        Console.WriteLine("\n\n\tFile not found" + e.Message);
    }

    return found;
}

public static int PopulateCars(Car[] cars)
{

    cars[0] = new Car("Vauxhall", "RUI 2345", 10);
    cars[1] = new Car("Audi A3", "SUI 6655", 5);
    cars[2] = new Car("Ford", "PUI 9381", 2);
    return 3;
```

```
    }
    public static void WriteToFile(Car[] cars, int size)
    {
        Stream sw;
        BinaryFormatter bf = new BinaryFormatter();
        try
        {
            sw = File.Open("MoreCars.dat", FileMode.Create);

            for (int x = 0; x < size; x++)
            {
                Car c = new Car(cars[x].Make, cars[x].RegNo, cars[x].Qty);
                bf.Serialize(sw, c);
            }
            sw.Close();

        }
        catch (IOException e)
        {
            Console.WriteLine("" + e.Message);
        }

    }
}
```

Questions

Q1

a. Write a method void pressKey(int row, int col) which will :-
Display "Press any key to continue" message at the co-ordinates passed;
On key entry clear the message.

b. Write a method void errMessage(string message, int row, int col) which will:-
Display an error message at the co-ordinates passed;
Call method pressKey(int row, int col) which will display the continue message;
Clear the error message.

Q2 Using the following class design for stock item details.

```
class       Stock

int stockNo;
            string stockName;
            int qtyInStock;
double unitCost;
```

a. Create a text file to hold the prompts for a data entry screen for the Stock object.

b. (i) Write a program to enter details of 5 items of stock to an array. Make use of the input screen created in 2a.

(ii) Write the stock details to a binary file "Stock.dat".

Sample file contents :-

1006	Oak carver	12	299.50
1007	Pine carver	25	125.80
1008	Oak table large	3	759.99

c. Using the Binary File, Stock.dat, created above :-

(i) Write C# code to produce a report "StockReport.txt" detailing the stock.

The report should show :-
An appropriate heading
The details for each of the stock items including:-
the total cost = (qty * unitCost) + vat)
VAT applied at 20%.
total cost with VAT
and finally:
the total number of stock Items,
the total value of the stock.

Your program should display an error message if the stock file is not found.

(ii) Write the C# code to append an item of stock to the file. Prompt for, and enter the new data through the Console

Q3 What modifications would you make to the class Stock if updating was required?

Q4 Use the modified class Stock run program 2 b to create a fixed length file.
(delete the existing file, stock.dat).

Write a program which will update the stockName of a stock item:

Prompt for, and enter the reqStockName through the Console;
Find and display the stock details on screen;
Prompt for, and enter the new stockName through the Console;
Update the stockName to file;
Print an error message if the stock item does not exist on file.