Contents

1 Introduction

This is a manual for the installation and use of the application ‘PF4File’. The program is a Windows application, for creating and viewing a .pf4 file. This is an emerging file format which currently uses up to 256 distinct colours. With this program one can create a .pf4 file from a .bmp file and conversely.

The pf4 specification is a lossless compression which results in file sizes of 1/6 to 1/15 (or more) of the equivalent bitmap image.

The .pf4 specification will be published in due course, as will the specification for video files. This application permits the user to gain familiarity with the format and its advantages.

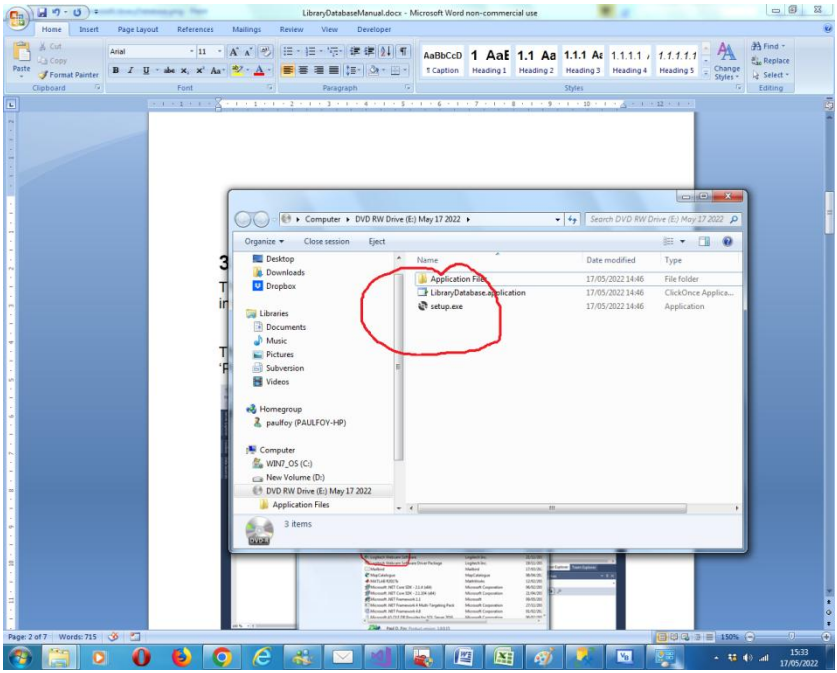
2 Pre-requisites.

1. A PC running Windows 7 or above.

2. A USB stick or optical drive containing the program setup files, together with this manual (available online).

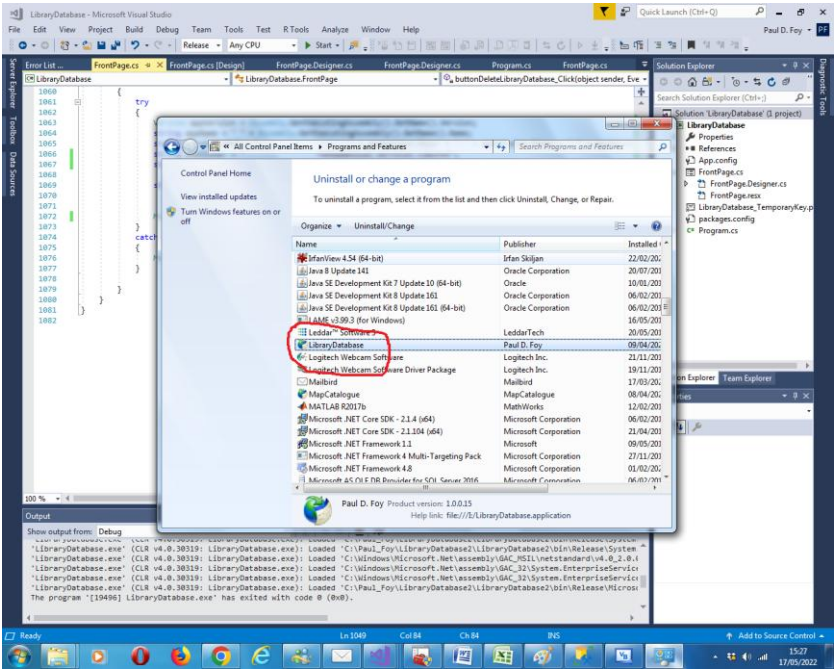
3 Installation/Removal

The program is installed by inserting the supplied stick or disc into the PC and running the ‘setup.exe’ program on it.



The program can be removed from the PC, by using the ‘Program & Features’ menu from within Control Panel.

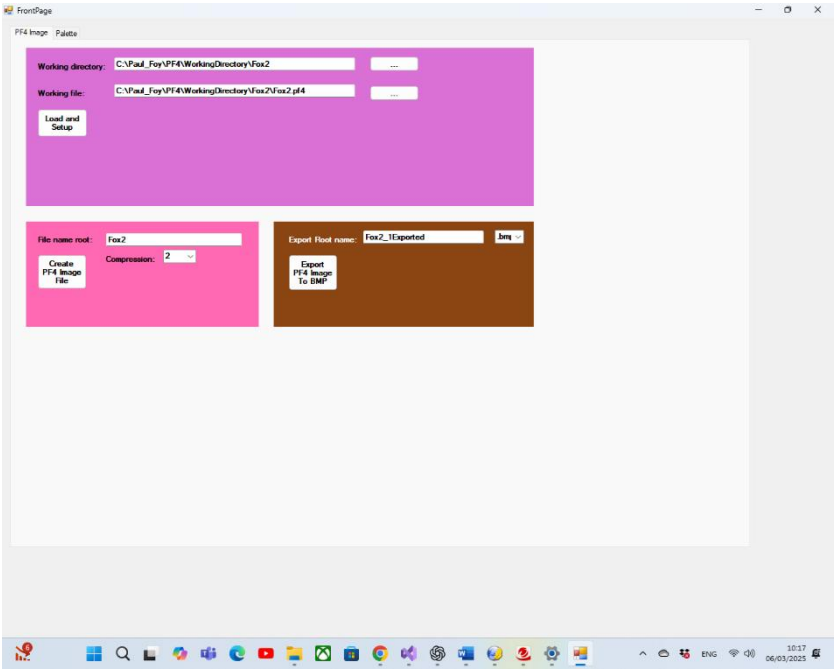
The program is robust to erroneous parameter input, but not completely fool proof.



4 PF4File

The application has currently two tabs:

4.1 PF4Image



4.1.1 Top panel

The text box **Working directory** is the directory where files created will be saved. This must be selected before the application will function.

The text box **Working file** is the file that the newly created .pf4 file is based upon.

The button **Load and Setup** loads either the .pf4 or the .bmp file into memory and displays it in the Palette.

4.1.2 Bottom left panel

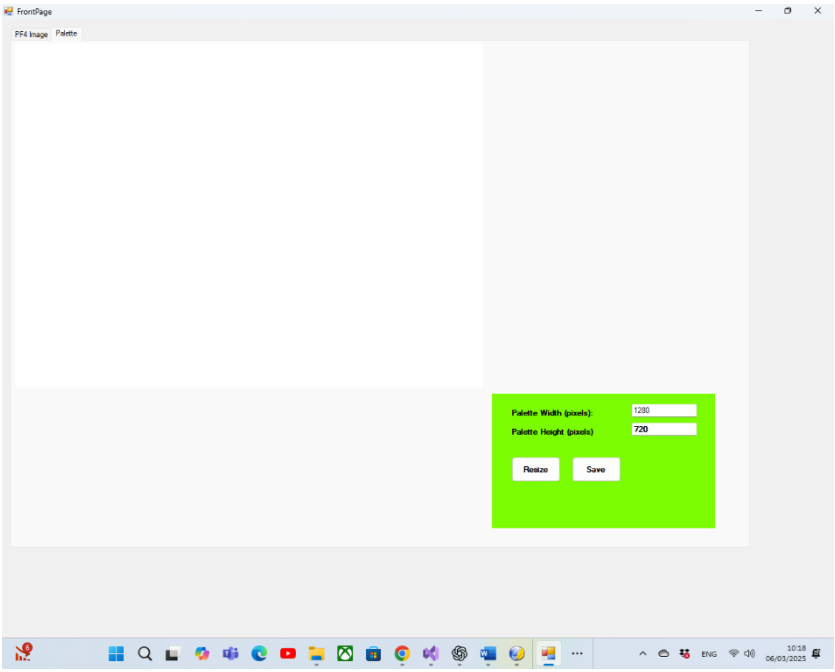
A .pf4 file with the root name of **File name root** is created with compression of **Compression** and saved in the **Working directory**. This is the function of the button **Create PF4 Image File**. Compression c reduces the width and height of the image by a factor of $1/c$ on saving ($1 \leq c \leq 11$).

The algorithm for Compression and Decompression will be described in due course. It is a lossless one.

4.1.3 Bottom right panel

The button **Export PF4 Image To BMP** exports the .pf4 file created to a .bmp and saves it in the **Working directory**. **Export root name** is the filename root.

4.2 Pallet



This tab contains a picture box control for receiving the created image. The image may be resized with a new or original size by using the **Palette Width (pixels)**, **Palette Height (pixels)** text boxes and **Resize** button. The image is saved with the **Save** button.

5 File Byte Structure of a .pf4 File

This section describes the file byte structure of a .pf4 file as created by this application. It also permits the creation of a .pf4 file by an independent program which can then be loaded by this application.

A file '`inputfile`' would be read into an array of bytes by a line of C# code such as in Table 1

```
byte[] pf4Data = File.ReadAllBytes(inputfile);
```

Table 1

Similarly an array of bytes would by written to file, '`filename`' by a line of C# code such as in Table 2.

```
File.WriteAllBytes(filename, byteArray);
```

Table 2

`byteArray` is a sequence of bytes having the following structure (and in the order stated):

[FileType][NumberOfColours][Compression][Colours][ImageWidth][ImageHeight][SpatialTrackLengths][SpatialTracks]

[FileType] – 1 byte, set to 0 (to indicate a static image as opposed to a video).

[NumberOfColours] – 1 byte, set to the number of distinct colours in the image.

[Compression] – 1 byte, set to the value of c, as described in section 4.1.2.

[Colours] – 3 x NumberOfColours bytes. For each colour 3 bytes are supplied representing the red, green and blue values (in that order consecutively and one colour followed by another).

[ImageWidth] – 4 bytes, an integer representing the width (in pixels) *of the compressed image*.

[ImageHeight] – 4 bytes, an integer representing the height (in pixels) *of the compressed image*.

The next two fields are used to describe how the structure of the compressed image is described.

If w and h are the width and height of the original image then $\bar{w} = w/2^c$ and $\bar{h} = h/2^c$ are the width and height of the compressed image respectively. $\bar{w}\bar{h}$ SpatialTracks are saved each having SpatialTrackLength, where each SpatialTrack consists of up to $2^c 2^c$ *trails* as described subsequently.

Colours, of a single pixel, are stored by their *index number* (corresponding to a number between 1 and 255) as in their location in the [Colours] array, and not by RGB values.

SpatialTrackLengths and SpatialTracks will firstly be described, by example, by referring to the 16 pixels corresponding to the case where $c = 4$.

1	2	5	10
3	4	7	12
6	8	9	14
11	13	15	16

Table 3

The 16 pixels are cycled through in the order of the numbering. Trails are tuples representing the number of occurrences of the colour and the colour index in that order. The pixels are cycled through until they are exhausted. They may be up to 16 trails. The idea is that the

ordering of Table 1, is most likely to achieve an efficiency in the storage because similar colours are most likely to be found corresponding to this sequence.

Thus the trails and the byte storage for a cell with the (index of) colours of Table 4 is

0	0	12	15
0	0	12	13
12	12	15	14
15	13	14	14

Table 4 (Colours with index values 12, 13, 14,15)

(4,0),(4,12),(3,15),(2,13),(3,14)

Figure 1

Note that the sum of the values for the first ordinate is 16. Thus we have:

[SpatialTrackLengths] – This is a byte representing the number of bytes in all the trails of a SpatialTracks. Thus in Figure 1 it is 10. There is an entry for each of the $\bar{w}\bar{h}$ spatial tracks and the order is from left to right top to bottom.

[SpatialTracks] – this is a sequence of bytes representing the spatial track such as in Figure 1. There is a sequence for each of the $\bar{w}\bar{h}$ spatial tracks and the order is from left to right top to bottom.

5.1 Code Fragmets

Some code fragments are given to assist in writing code both to create a .pf4 file and to read in from one.

An example of reading in from a byte array to populate the fields would be as in Figure 2

```
//Create PF4 object from disk file.
int iCount = 0;  //This is the number of bytes read.
byte[] pf4Data = File.ReadAllBytes(inputfile);
FileType = pf4Data[iCount];
iCount += 1;
NumberOfColours = pf4Data[iCount];
iCount += 1;
Compression = pf4Data[iCount];
iCount += 1;
//3 bytes per colour
Colours = new byte[3 * NumberOfColours];
for (int i = 0; i < NumberOfColours; i++)
{
    Colours[3 * i] = pf4Data[iCount + 3 * i];
    Colours[3 * i + 1] = pf4Data[iCount + 3 * i + 1];
    Colours[3 * i + 2] = pf4Data[iCount + 3 * i + 2];
}
iCount += (3 * NumberOfColours);
//2 bytes for the width
byte[] bytesInInt = new byte[sizeof(int)];
for (int i = 0; i < sizeof(int); i++)
{
    bytesInInt[i] = pf4Data[iCount + i];
}
ImageWidth = BitConverter.ToInt32(bytesInInt, 0);
iCount += sizeof(int);
//2 bytes for the height
int imageWidth = ImageWidth;
bytesInInt = new byte[sizeof(int)];
for (int i = 0; i < sizeof(int); i++)
{
    bytesInInt[i] = pf4Data[iCount + i];
}
```

```
ImageHeight = BitConverter.ToInt32(bytesInInt,0);
int imageHeight = ImageHeight;
iCount += sizeof(int);
//1 byte to store each pixel's colour
if (FileType == 0)
{
    //we have a static image

    //The spatial track lengths
    SpatialTrackLengths = new byte[imageWidth,
imageHeight];
    int totalSpatialTrackLength = 0;
    for (int n = 0; n < imageHeight; n++)
    {
        for (int m = 0; m < imageWidth; m++)
        {
            SpatialTrackLengths[m, n] = pf4Data[iCount
+ (n * imageWidth + m)];

            totalSpatialTrackLength +=
SpatialTrackLengths[m, n];
        }
    }
    iCount += (imageWidth * imageHeight);

    //the spatial tracks themselves
    SpatialTracks = new List<byte>[imageWidth,
imageHeight];
    int iSpatialTrackCount = 0;
    for (int n = 0; n < imageHeight; n++)
    {
        for (int m = 0; m < imageWidth; m++)
        {
            SpatialTracks[m, n] = new List<byte>();
            for (int l = 0; l < (SpatialTrackLengths[m,
n]); l++)
            {
                SpatialTracks[m, n].Add(pf4Data[iCount
+ l]);
            }
```

```
            iCount += SpatialTrackLengths[m, n];
            iSpatialTrackCount += SpatialTracks[m,
n].Count;
        }
    }

}
```

Figure 2

An example of creating the fields of the .pf4 file ready for writing would be as in Figure 3

```
Boolean error = false;


List<Color> colors = new List<Color>();
Boolean tooMany = findColoursInImage(ref bitmap, ref colors);
if (!tooMany)
{
   //save the image to a pf4 file
   //Blocks are
   //[FileType][NumberOfColours][Compression][Colours][ImageWidth]
   //          [ImageHeight][SpatialTrackLengths][SpatialTracks]

   //[FileType] - size 1
   FileType = 0;

   //[NumberOfColours] - size 1
   NumberOfColours = (byte)colors.Count;

   //[Compression] - size 1
   Compression = compression;

   //[Colours] - size 3 * NumberOfColours
   Colours = new byte[3 * colors.Count];
   for (int i = 0; i < colors.Count; i++)
```

```
    {
        Colours[3 * i] = colors[i].R;
        Colours[3 * i + 1] = colors[i].G;
        Colours[3 * i + 2] = colors[i].B;
    }

    int factor = Convert.ToInt32(Math.Pow(2, compression));
    //[ImageWidth] - size 4
    ImageWidth = bitmap.Width / factor;
    //[ImageHeight] - size 4
    ImageHeight = bitmap.Height / factor;

    //get the spatial tracks and their lengths for each block
    //blocks are f = 2^compression squares and there are (w/f)*(h/f) of
them,
    //making up the image.
    SpatialTracks = new List<byte>[bitmap.Width / factor,
bitmap.Height / factor];
    SpatialTracks = getSpatialTracks(bitmap, compression);

    //populate the spatial track lengths
    SpatialTrackLengths = new byte[bitmap.Width / factor,
bitmap.Height / factor];
    for (int m = 0; m < (bitmap.Width / factor); m++)
    {
        for (int n = 0; n < (bitmap.Height / factor); n++)
```

```
        {
            SpatialTrackLengths[m, n] = (byte)(SpatialTracks[m, n].Count);
        }
    }



}
else
{
    error = true;
}
```

Figure 3

Paul D. Foy
Mathematical Services

March 2025