# MGL Avionics

Plates file format

Plates file version 1

Draft document dated April 11, 2014

## Data items

| | |
|---|---|
| Byte | Unsigned 8 bit value |
| Word | Unsigned 16 bit value |
| Smallint | Signed 16 bit value |
| Longint | Signed 32 bit value |

String	ASCII string. First byte is length of string. String follows. Note: This is not a “C” style string, there is no termination character as it is not needed.

### Position data format

The position format is used in all data files. This format allows usage of 32 bit integers while allowing position resolution to less than 3 feet.

Position latitude and longitude are stored as signed 32 bit integers.

Degrees are multiples of the value 180000 and any degree fraction is a multiple of 1/180000.

Degrees North and East are positive while degrees South and West are negative.

Example: North 45 degrees, 59 minutes, 30 seconds is 8278500.

Example: North 0 degrees, 30 minutes, 0 seconds is 90000.

## File format

The MGL Avionics plates file is a general purpose image file format containing one to many images. Images may have georeferencing information.

Images are stored in a runlength compressed format which tends to suit typical plates files well while allowing simple and fast decoding on the EFIS.

The basic idea behind a plates file is that each file contains all of the plates belonging to an airport and the name of that file would be the airports short ID prefixed with the letters “EG”. For example, the plates file for Los Angeles International would be EGKLAX.EPG.

In addition to this provision is made for a single additional file called EPINFO.EPG. This file can be used as a general purpose container for images not related to any particular airport. A use for it could be for example to contain flight level graphics or other information useful to the pilot.

The file format for either type is identical.

The file starts with a single copy of TGroupHeader.

```
 TGroupHeader = packed record
            IDString: string[3];
            GroupID: string[6];
            Version: byte;
            GroupName: string[63];
```

```
        NumberOfPlates: longint;
      end;
```

IDString contains the letters “EPG”.

GroupID contains up to six characters. This would normally be the short airport identifier or incase of the information file the text “INFO”.

Version contains the value 0x01

Groupname contains the long airport name, for example “Los Angeles International”.

NumberOf plates contains the number of plates following in this file.

The Group header is followed by “NumberOfPlates” entries of type TPlateHeader.

```
TPlateHeader = packed record
        PlateName: string[63];
        PlatePtr: longint;
       end;
```

PlateName contains the name of the plate, for example “Approach RWY 19”.

PlatePtr points to the absolute byte position of the TPlateDataHeader in the file.

```
  TPlateDataHeader = packed record
          Lat1,Long1,Lat2,Long2: Longint;
          NumberOfPages: word;
          PageNumber: word;
          GeoRef: boolean;
          Padding: byte;
          Width,Height: word;
        end;
```

If the image is georeferenced, the lat and long entries contain the coordinates of the top left and bottom right corners of the image. (Lat1,Long1 is top left, Lat2,Long2 is bottom right).

NumberOfPages contains the number of pages for this plate. This is currently not use and should be set to 0x01.

PageNumber should be set to 0x01

GeoRef should be set to 0x01 if the image is georeferenced, 0x00 if it is not.

Padding should be set to 0x00. It is used to align the following words.

Width and Height specify the size of the image in pixels.

The imageheader follows the PlateDataHeader immediately.

```
  TImageHeader = record
        Width,Height: word;
```

```
    Size: longint;
  end;
```

Width and height repeat the image size while the size entry contains the number of bytes the image is occupying in the file (the image is compressed so width and height cannot be used to determine this). The EFIS uses this to read the file effectively.

The image follows the ImageHeader immediately.

## The image format

A simple run length format is used based on a 256 byte fixed color palette. The palette details are given further on in this file.

Two runlengh modes are used: Verbatim and compress. Verbatim means copy a defined number of bytes as they are. Runlength means create a string of a fixed pallete entry for n-number of bytes.

Each of these modes can create up to 127 bytes of image data.

The mode byte is simple: The most significant bit (bit 7) if set means “runlength” and if it is not set it means “Verbatim”. Bits 0-6 are a count. The Value “0” is not legal and not used.

The encoder will scan the image in horizontal lines from left to right. As it does so it converts every pixels color to a palette index value. It will use a “nearest match” algorithm to pick the most suitable available color. If it finds adjoining pixels with the same index, it will encode these as “runlength”. If it finds adjoining pixels are different, it will encode them as “verbatim”. Usually, it needs at least three adjoining same pixels to encode as runlength.

Runlength example:

0x87 0x33    Seven pixels of index value 0x33

Verbatim example:

0x04 0x12 0x48 0x31 0xAF  Four pixels

Every new line should start with a new mode byte, do not wrap lines.

## The Pallete

The pallete is typically created programatically.

Here is the Pascal code to create the palette:

```
var
  ColorTable: array[0..255] of longint;
  NumberOfColors: word;
const
  DEFC: array[0..15,0..2] of byte =
    ((200,150,50),
     (211,165,72),
     (217,179,90),
     (229,194,108),
     (239,213,133),
     (247,231,160),
     (245,250,171),
     (252,253,208),
     ($80,$80,$80),
     ($FF,0,0),
     ($0,$FF,0),
     ($FF,$FF,0),
     (0,0,$FF),
     ($FF,0,$FF),
     (0,$FF,$FF),
     ($FF,$FF,$FF));

procedure MakePallete;
var
  i: word;
  red,green,blue: byte;
  VRed,VGreen,VBlue,c: longint;
begin
  //The first 16 colors are standard Windows/DOS colors
 Colortable[0]:=clBlack;
```

```
Colortable[1]:=clMaroon;
Colortable[2]:=clGreen;
Colortable[3]:=clOlive;
Colortable[4]:=clNavy;
Colortable[5]:=clPurple;
Colortable[6]:=clTeal;
Colortable[7]:=clGray;
Colortable[8]:=clSilver;
Colortable[9]:=clRed;
Colortable[10]:=clLime;
Colortable[11]:=clYellow;
Colortable[12]:=clBlue;
Colortable[13]:=clFuchsia;
Colortable[14]:=clAqua;
Colortable[15]:=clWhite;
i:=16;
for red:=0 to 13 do
  for green:=0 to 4 do
    for blue:=0 to 2 do
    begin
      Vblue:=blue*128;
      if VBlue>255 then VBlue:=255;
      Vgreen:=green*64;
      if Vgreen>255 then Vgreen:=255;
      Vred:=red*20;
      if Vred>255 then Vred:=255;
      c:=(Vblue shl 16)+(Vgreen shl 8)+Vred;
      if (c<$FFFFFF) and (c>0) then
      begin
        if i<256 then ColorTable[i]:=c;
        inc(i);
      end;
    end;
for red:=2 to 15 do
```

```
  begin
    if i<256 then c:=(red*16 shl 16)+(red*16 shl 8)+red*16;
    colortable[i]:=c;
    inc(i);
  end;
  for red:=0 to 7 do
  begin
    if i<256 then c:=(DEFC[red,2] shl 16)+(DEFC[red,1] shl 8)+DEFC[red,0];
    colortable[i]:=c;
    inc(i);
  end;
  NumberOfColors:=i;
end;
```

**A suitable color match algorithm is here:**

```
function ChangeToNearestColor(v: longint): longint;
var
  Error,r,g,b,r1,g1,b1,diff: longint;
  x,n: word;
begin
  Error:=$FFFFFF;
  x:=0;
  for n:=0 to NumberOfColors-1 do
  begin
    b:=v shr 16 and $FF;
    g:=v shr 8 and $FF;
    r:=v and $FF;
    b1:=ColorTable[n] shr 16 and $FF;
    g1:=ColorTable[n] shr 8 and $FF;
    r1:=ColorTable[n] and $FF;
    diff:=abs(sqr(r-r1))+abs(sqr(g-g1))+abs(sqr(b-b1));
    if diff<Error then
    begin
      Error:=Diff; x:=n;
    end;
  end;
  result:=ColorTable[x];
  SelectedColor:=x;
end;
```

**A typical encoder extracted from the MGL Plates application (for reference) is here:**

```
procedure ConvertCanvasToMIF(Filename: string; Width,Height: longint;
                             ClipEnable: boolean;
                             ClipX1,ClipY1,ClipX2,ClipY2: longint);
var
  ln,c,ptr,RL,same,notsame,n,i,ww,ImageSize,Size,HDPtr,
  ClipHeight,ClipWidth,xt,yt,x,y,SpanX,SpanY: longint;
  v,mode,a1,a2,a3,a4,a5: byte;
  First: Boolean;
  s,OldS: string;
  Ratio: double;
begin
  OrgWidth:=width;
  OrgHeight:=height;
  if clipenable then
  begin
    if ClipX1<Width then
      if ClipY1<Height then
        if ClipX2<Width then
          if ClipY2<Height then
            if ClipX1<ClipX2 then
              if ClipY1<ClipY2 then
    begin
      ClipWidth:=ClipX2-ClipX1;
      ClipHeight:=ClipY2-ClipY1;
      with LBitMap.Canvas do
      begin
        yt:=ClipY1;
        for y:=0 to ClipHeight-1 do
        begin
          xt:=ClipX1;
          for x:=0 to ClipWidth-1 do
          begin
            Pixels[x,y]:=Pixels[xt,yt]; inc(xt);
          end;
          inc(yt);
        end;
      end;
      Width:=ClipWidth;
      Height:=ClipHeight;
    end;
  end;
  ImageHeader.Width:=Width;
  ImageHeader.Height:=Height;
  ImageHeader.Size:=0;
  ImageWidth:=Width;
  ImageHeight:=Height;
  ExpWidth:=Width;
  ExpHeight:=height;
  GetMem(RawImagePointer,ImageWidth*ImageHeight);
  with LBitMap.Canvas do
  begin
    for ln:=0 to ImageHeight-1 do
    begin
      for c:=0 to ImageWidth-1 do
      begin
        Pixels[c,ln]:=ChangeToNearestColor(Pixels[c,ln]);
        RawImagePointer^[ln*ImageWidth+c]:=SelectedColor;
      end;
    end;
  end;
  AssignFile(MIFFile,ExePath+'Temp\EXP.TMP');
```

```
Rewrite(MIFFile,1);
BlockWrite(MIFFile,ImageHeader,4);
HDPtr:=FilePos(MIFFile);
BlockWrite(MIFFile,ImageHeader.Size,4);
OutputSize:=0;
for ln:=0 to ImageHeight-1 do
begin
  mode:=0; //verbatim
  Ptr:=0;
  Same:=0; NotSame:=0;
  c:=0;
  First:=true;
  while c<ImageWidth do
  begin
    if mode=0 then //Verbatim
    begin
      a1:=RawImagePointer^[ln*ImageWidth+c];
      if c<ImageWidth-3 then
      begin
        a2:=RawImagePointer^[ln*ImageWidth+c+1];
        a3:=RawImagePointer^[ln*ImageWidth+c+2];
      end else
      begin
        a2:=$FF; a3:=$00;
      end;
      if (a1=a2) and (a1=a3) then
      begin
        if mode=0 then
        begin
          if not first then if (NotSame=0) then dec(ptr);
          if NotSame>0 then
            CompressedLine[RL]:=NotSame;
          Same:=3;
          c:=c+3;
          mode:=1;
          RL:=ptr;
          inc(ptr);
          CompressedLine[ptr]:=a1;
          inc(ptr);
          notsame:=0;
          First:=false;
        end;
      end else
      begin
        if First and (NotSame=0) then
        begin
          First:=false;
          RL:=ptr;
          inc(ptr);
        end;
        CompressedLine[ptr]:=a1;
        inc(ptr);
        inc(c);
        inc(NotSame);
        if NotSame=127 then
        begin
          CompressedLine[RL]:=$7F;
          NotSame:=0;
          RL:=ptr;
          inc(ptr);
        end;
      end;
    end else
    begin  //mode is same
      a5:=RawImagePointer^[ln*ImageWidth+c];
      if a5=a1 then
      begin
        inc(same);
        if same=127 then
        begin
          CompressedLine[RL]:=$FF;
          RL:=ptr;
          inc(ptr);
          CompressedLine[ptr]:=a1;
          inc(ptr);
          same:=0;
        end;
        inc(c);
      end else
      begin
        if (mode=1) and (same=0) then dec(ptr,2) else CompressedLine[RL]:=Same or $80;
        RL:=ptr;
        inc(ptr);
        inc(c);
        CompressedLine[ptr]:=a5;
        inc(ptr);
        a1:=a5;
        if c<ImageWidth-1 then
        begin
          a2:=RawImagePointer^[ln*ImageWidth+c];
          a3:=RawImagePointer^[ln*ImageWidth+c+1];
        end else
        begin
          a2:=$FF; a3:=$00;
        end;
        if (a1=a2) and (a1=a3) then
        begin
          mode:=1; //remains 1
          NotSame:=0;
          same:=1;
        end else
        begin
```

```
          mode:=0; //Verbatim
          NotSame:=1;
          Same:=0;
        end;
      end;
    end;
  end;
  if (mode=1) and (same>0) then
  begin
    CompressedLine[RL]:=Same or $80;
  end else
  if (mode=0) and (NotSame>0) then
  begin
    CompressedLine[RL]:=NotSame;
  end;
  if (mode=1) and (same=0) then dec(ptr,2);
  BlockWrite(MIFFILE,CompressedLine,ptr);
  OutputSize:=OutputSize+ptr;
 end;
 seek(MIFFile,HDPtr);
 BlockWrite(MIFFile,OutputSize,4);
 CloseFile(MIFFile);
 if assigned(RawImagePointer) then FreeMem(RawImagePointer);
end;
```