Courses IPCx, C1: Bitmap, Code Comments

In `bitmap1Doc.h` in front of `class CBitmap1Doc : public CDocument`

`#include < vector >` //declares the dynamic arrays of the Standard Template Library STL. The blanks inside the < > clause around the word `vector` may be suppressed. They are just necessary in a HTML-Document otherwise HTML treats the word `vector` as HTML-Tag.

`BITMAPFILEHEADER FH;` //structure of overall length = 14 bytes containing 5 variables (see MSDN and C6_Bitmap_FAQs)

`BYTE IBytes[1200];` //untyped space for 1200 bytes for BitmapInfoHeader+Palette. You don't know the types of BitmapInfoHeaders you will read and you don't know if there will be palettes or not. You reserve 1200 bytes for the worst case: BITMAPV5HEADER (=136 byte) plus 256 palette entries (=1024 bytes). 1160 bytes are wasted in case of traditional 24-Bit-Bitmaps but you can probably afford that.

`BITMAPINFOHEADER* pIH;` //typed pointer allows the access to biSize, biWidth etc.

`BITMAPINFO* pI;` //This typed pointer is needed by function `StretchDIBits`. Structure BITMAPINFO (see MSDN and C6_Bitmap_FAQs) is longer than BITMAPINFOHEADER but it contains in its first part the structure BITMAPINFOHEADER completely . It allows access to both the BitmapInfoHeader and (if `biClrUsed > 0`) the palette. If `biClrUsed > 0` then `StretchDIBits` automatically accesses and uses the palette with the help of this pointer.

`std::vector< BYTE > Pixel;` //name of a dyn. array of bytes aimed to store the original pixels

`int nClicks;` //counter for the no. of mouse clicks

In `bitmap1Doc.cpp` inside the constructor `CBitmap1Doc()` you have to initialize 4 variables:

`memset( IBytes, 0, sizeof(IBytes) );` // clear the 1200 untyped bytes in order to mark the absence of an image. Function `CBitmap1View::OnDraw()` reads from here if you already loaded an image or not.

`pIH = (BITMAPINFOHEADER*) IBytes;` //The typed pointer points now to the the head of IBytes[1200]

`pI = (BITMAPINFO*) IBytes;` //The typed pointer points now to the the head of IBytes[1200] as `pIH` does.

`nClicks = 0;` //Reset the mouse click counter.

In `bitmap1Doc.cpp` inside `Serialize(CArchive& ar)` inside the `else` clause: Code for reading an image from the hard disk

`ar.Read( & FH, sizeof(BITMAPFILEHEADER) );` //Read 14 bytes from the harddisk.

`if ( FH.bfType != 'MB') { forget_it(); return;` //The first 2 bytes form the reversed string of BM.

`if ( FH.bfSize <= 54 ) { forget_it(); return;` //A bitmap file contains at least 55 bytes.

`if ( FH.bfOffBits < 54 ) { forget_it(); return;` //The shortest possible headers need 54 bytes.

`int nBytesInfo = FH.bfOffBits - sizeof(BITMAPFILEHEADER);` //That is the space left for the BitmapInfoHeader and palette.

`int nBytesPixel = FH.bfSize - FH.bfOffBits;` //That is the space for all the pixels.

`ar.Read( IBytes, nBytesInfo );` //Read BitmapInfoHeader+palette from harddisk

`if ( !(pIH->biBitCount == 8 || pIH->biBitCount ==24) ) { forget_it(); return; }` //Ignore all 1, 4, 16 and 32 bit bitmaps. `!` = logical NOT and `||` = logical OR.

`Pixel .resize( nBytesPixel );` //Keep free the necessary space for all the original pixels in main memory.

`PixelBinary.resize( nBytesPixel );` //Double the space for storing the black and white pixels.

`ar.Read( &Pixel.front(), nBytesPixel );` //Read all original pixels from the harddisk into the main memory starting at the first adress of the dynamic byte array named `Pixel`.

In `bitmap1Doc.cpp` inside `void CBitmap1Doc::forget_it()` //Small private member function of `CBitmap1Doc`

`memset( IBytes, 0, sizeof(IBytes) );` //If this was not a reasonable bitmap, then clear all 1200 bytes of IBytes to zero.

`for ( int i=0; i < 10; i++ ) MessageBeep(-1);` //This is a loud protest against misuse.

In `bitmap1View.cpp` inside `void CBitmap1View::OnDraw(CDC* pDC)` //

`CBitmap1Doc* pDoc = GetDocument();` //This line was prepared by Visual Studio. Keep it. It give us a pointer to reach the variables which have been declared in class `CBitmap1Doc`

`if ( !pDoc->pIH->biSize ) { pDC->TextOut(0,0,"Open a *.BMP file !"); return; }` //Advice to users who do not know what they should do after they started the program.

`CString s[11];` //Declaration of an array of 11 strings.

`CRect R; GetClientRect( R );` //Find out the current dimensions of the client area that could be covered with the image.

`int FHeader, IHeader, dx, dy, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r;` //Definitions of 23 integers

```
int xSize = pDoc->pIH->biWidth;
int ySize = pDoc->pIH->biHeight;
```
//The names `xSize` and `ySize` replace from here on the complicated `BITMAPINFOHEADER` names.

`SetStretchBltMode( pDC->GetSafeHdc(), COLORONCOLOR );` //To prevent from color chaos after zoom down in High Color Mode (see FAQs)

`switch ( pDoc->nClicks % 4 )` //Display four differnt window contents with cyclic change on every mouse click.

```
case 0: // first screen: what has been found in the headers
```

```
FHeader = sizeof(BITMAPFILEHEADER);
IHeader = sizeof(BITMAPINFOHEADER);
```
//Length of both headers must be identical to the lengths of the appropriate predefined data structures

```
a = pDoc->FH.bfSize;
```
//Specifies the size, in bytes, of the bitmap file.

```
b = pDoc->FH.bfOffBits;
```
//Specifies the offset, in bytes, from the BITMAPFILEHEADER structure to the bitmap bits.

```
c = pDoc->pIH->biSize;
```
//Specifies the number of bytes required by the structure.

```
d = xSize;
```
//Specifies the width of the bitmap, in pixels.
Windows 98, Windows 2000: If biCompression is BI_JPEG or BI_PNG, the biWidth member specifies the width of the decompressed JPEG or PNG image file, respectively.

```
e = ySize;
```
//Specifies the height of the bitmap, in pixels. If biHeight is positive, the bitmap is a bottom-up DIB and its origin is the lower-left corner. If biHeight is negative, the bitmap is a top-down DIB and its origin is the upper-left corner. If biHeight is negative, indicating a top-down DIB, biCompression must be either BI_RGB or BI_BITFIELDS. Top-down DIBs cannot be compressed.
Windows 98, Windows 2000: If biCompression is BI_JPEG or BI_PNG, the biHeight member specifies the height of the decompressed JPEG or PNG image file, respectively.

```
f = pDoc->pIH->biPlanes;
```
//Specifies the number of planes for the target device. This value must be set to 1.

```
g = pDoc->pIH->biBitCount;
```
//Specifies the number of bits-per-pixel. The biBitCount member of the BITMAPINFOHEADER structure determines the number of bits that define each pixel and the maximum number of colors in the bitmap.

```
h = pDoc->pIH->biCompression;
```
//Specifies the type of compression for a compressed bottom-up bitmap (top-down DIBs cannot be compressed). This member can be one of the following values.
`BI_RGB, BI_RLE4, BI_BITFIELDS, BI_JPEG, BI_PNG`.

```
i = pDoc->pIH->biSizeImage;
```
//Specifies the size, in bytes, of the image. This may be set to zero for BI_RGB bitmaps.
Windows 98, Windows 2000: If biCompression is BI_JPEG or BI_PNG, biSizeImage indicates the size of the JPEG or PNG image buffer, respectively.

j = pDoc->pIH->biXPelsPerMeter; //Specifies the horizontal resolution, in pixels-per-meter, of the target device for the bitmap. An application can use this value to select a bitmap from a resource group that best matches the characteristics of the current device.

```
k = pDoc->pIH->biYPelsPerMeter;
```
//Specifies the vertical resolution, in pixels-per-meter, of the target device for the bitmap.

```
l = pDoc->pIH->biClrUsed;
```
//Specifies the number of color indexes in the color table that are actually used by the bitmap. If this value is zero, the bitmap uses the maximum number of colors corresponding to the value of the biBitCount member for the compression mode specified by biCompression.
If biClrUsed is nonzero and the biBitCount member is less than 16, the biClrUsed member specifies the actual number of colors the graphics engine or device driver accesses. If biBitCount is 16 or greater, the biClrUsed member specifies the size of the color table used to optimize performance of the system color palettes. If biBitCount equals 16 or 32, the optimal color palette starts immediately following the three DWORD masks.
When the bitmap array immediately follows the BITMAPINFO structure, it is a packed bitmap. Packed

bitmaps are referenced by a single pointer. Packed bitmaps require that the biClrUsed member must be either zero or the actual size of the color table.

`m = pDoc->pIH->biClrImportant;` //Specifies the number of color indexes that are required for displaying the bitmap. If this value is zero, all colors are required.

`n = b - FHeader - IHeader;` //Byte-length of the palette

`o = a - b;` //Byte-length of all pixels

`p = d * e;` //no. of pixels

`q = FHeader + IHeader + n + o;` //Must be identical to `a`.

`r = o-d*e*g/8;` //Must be zero, otherwise empty bytes are attached.

```
s[0].Format( "a) %d Bytes BITMAPFILEHEADER", FHeader );
s[2].Format( "b) %d Bytes BITMAPINFOHEADER", IHeader );
s[1].Format( " ->Type=BM, Size=%d, OffBits=%d", a, b);
s[3].Format( " ->Size=%d, Width=%d, Height=%d, Planes=%d", c, d, e, f);
s[4].Format( " ->BitCount=%d, Compression=%d, SizeImage=%d", g, h, i );
s[5].Format( " ->XPelsPerMeter=%d, YPelsPerMeter=%d", j, k );
s[6].Format( " ->ClrUsed=%d, ClrImportant=%d", l, m );
s[7].Format( "c) %d Bytes in der Palette = %d RGBQUADs", n, n/4 );
s[8].Format( "d) %d Bytes für %d Pixel", o, p );
s[9].Format( "a) + b) + c) + d) = %d Bytes = %0.3f KiloBytes",q, (double)q/1024.
);
```

`s[10].Format( "Press the left mouse button !" );` //Advice to users who do not know what to do.

//Put some text together with the variables into the 11 strings. The first parameter of the format method always is a string itself, the so-called format-control string. It contains percent signs as place-holders for variables followed by the letter d (indicating integers). All following parameters are variables. Their order and integer data type correspond to the percent-d place-holders of the format-control string.

`for ( i = 0; i < 11; i++ ) pDC->TextOut(0,i*15,s[i]);` //Print out the 11 strings in form of 11 lines on the client area.

`break;` //This is a goto (=jump) statement to the end of the switch-clause. When you forget it, the program will execute the next case too.

`case 1:` //second screen: the image in its original size

`StretchDIBits( pDC->GetSafeHdc(),` //Draw the image into the graphics board and on the screen. The first parameter has to be a handle to the Device Context.

`0, 0, xSize, ySize,` //Parameters 2,3,4,5 of `StretchDIBits` indicate the rectangular destination dimensions.

`0, 0, xSize, ySize,` //Parameters 6,7,8,9 of `StretchDIBits` indicate the rectangular original source dimensions.

`&(pDoc->Pixel.front()), pDoc->pI` //Parameter 10 has to point to the first pixel and parameter 11 must be a typed pointer to a BitmapInfo-structure (BitmapInfoheader plus palette).

`DIB_RGB_COLORS, SRCCOPY );` //Parameters 12 and 13 are constants which influence the use of colors and transparency (see MSDN).

`break;` //This is a goto (=jump) statement to the end of the switch-clause. When you forget it, the program will execute the next case too.

`pDC->TextOut(0,R.Height()-20,"Press the left mouse button !");` //Advice to users who do not know what to do.

`case 2:` //third screen: the image zoomed to fill the complete client area

`StretchDIBits( pDC->GetSafeHdc(),` //Draw the image into the graphics board and on the screen. The first parameter has to be a handle to the Device Context.

`0, 0,R.Width(), R.Height(),` //Parameters 2,3,4,5 of `StretchDIBits` indicate the rectangular destination dimensions.

`0, 0, xSize, ySize,` //Parameters 6,7,8,9 of `StretchDIBits` indicate the rectangular original source dimensions.

`&(pDoc->Pixel.front()), pDoc->pI` //Parameter 10 has to point to the first pixel and parameter 11 must be a typed pointer to a BitmapInfo-structure (BitmapInfoheader plus palette).

`DIB_RGB_COLORS, SRCCOPY );` //Parameters 12 and 13 are constants which influence the use of colors and transparency (see MSDN).

`break;` //This is a goto (=jump) statement to the end of the switch-clause. When you forget it, the program will execute the next case too.

`pDC->TextOut(0,R.Height()-20,"Change window size ! Press the left mouse button !");` //Advice to users who do not know what to do.

`case 3:` //fourth screen: the image zooms down in 20 steps

```
dx = R.Width() / 20;
dy = R.Height() / 20;
```
//width and height of 20 down steps

`for ( i = 0; i < 20; i++ )` //go to any step

`StretchDIBits( pDC->GetSafeHdc(),` //Draw the image into the graphics board and on the screen. The first parameter has to be a handle to the Device Context.

`0, 0,R.Width() - i*dx, R.Height() - i*dy,` //Parameters 2,3,4,5 of `StretchDIBits` indicate the rectangular destination dimensions. They shrink at each step.

`0, 0, xSize, ySize,` //Parameters 6,7,8,9 of `StretchDIBits` indicate the rectangular original source dimensions.

`&(pDoc->Pixel.front()), pDoc->pI` //Parameter 10 has to point to the first pixel and parameter 11 must be a typed pointer to a BitmapInfo-structure (BitmapInfoheader plus palette).

`DIB_RGB_COLORS, SRCCOPY );` //Parameters 12 and 13 are constants which influence the use of colors and transparency (see MSDN).

`break;` //This is a goto (=jump) statement to the end of the switch-clause. When you forget it, the program will execute the next case too.

`pDC->TextOut(0,R.Height()-20,"Change window size ! Press the left mouse button !");` //Advice to users who do not know what to do.

In `void CBitmap1View::OnLButtonDown(UINT nFlags, CPoint point)` //The user presses the left mouse button.

`CBitmap1Doc* pDoc = GetDocument();` //The following statement needs a pointer to the Document class

`pDoc->nClicks++;` //Increment the mouse click counter which lives in the Document class.

`Invalidate();` //Ask the operating system to redraw the content of the client area.