PES (Brother) Design format

Disclaimer:

This file is full of assumptions, deductions, etc. It is a clean room implementation based on works found in the credit section, and comparison of various PES files. Anyone is free to use this document and contribute provided the following is agreed to:

- No illegal contributions. If you have inside information (NDA or otherwise proprietary or legally protected information), I do not want it. By contributing you are stating that any information, techniques, etc. used are legal in the jurisdiction where you live and to the best of your knowledge and judgment is legal for you to share.
- This information is “AS IS.” You assume all liability for damage to your machine, materials, loss of money, damage and injury to your body (including, but not limited to, damage to hearing, eyesight, broken bones, loss of life, halitosis, etc.) or that of others from any machine malfunctions caused by any information, accurate or not, found here-in. THIS DOCUMENT IS CURRENTLY FULL OF A LOT OF GUESSES.

Explanation of Abbreviations :

HEX	Hexadecimal
DEC	Decimal
MSB	Most significant bit first
LSB	Least significant bit first
NN	Value varies
UNK	Unknown
FIXED	Fixed value

Explanation of datatypes:

Short	This is a 16 bit integer stored in Little Endian.
Int32	This is a 32 bit integer stored in Little Endian.

Float	This is a 32 bit IEEE 754 stored as a 4 bytes little endian

Like most embroidery formats the distance of 1 unit is equal to 1/10mm.

Understanding PES:

Ultimately PES is a hack on top of the PEC which allows older embroidery machines to be future compatible with a changing format. In order to read the file simply skip the first 8 characters these are depending on the version #PEC0001, #PES0001, #PES0020, #PES0030, #PES0040, #PES0050, #PES0055, #PES0060, any future version. This information is immediately followed by the location of the PEC code block, as a little-endian 32 bit integer. 512 (0x200) bytes beyond this the stitches are are encoded. This allows the format to be agnostic as to what version it is while allowing any supporting machine to do the relevant stitching. As embroidery machines are ultimately just a panograph hooked to a sewing machine. All that it needs are the control commands. Any reading of the file should likely be done in accordance with this, with the exception of special case applications.

Most PES data is stored in 2-4 byte LSB data, as it is intended to be read by various versions of PE-Design software. The PEC code block data is intended to be read by the embroidery machine and these therefore are have different encoding schemes. The data in the PES file contains things like the location it is displayed on the screen, the size of the hoop, if there is a grid, the colors used, whether it should be scaled or not, and some internal settings only relevant to the one software suite in question, and has no bearing on the actual stitches.

The PES data in later versions 5.5 and above contains various shapes and worker information as well as larger hoops and other thread information. The intent is that the format should be lossless. A major problem with embroidery and embroidery software is that the settings and information used to create an embroidery design can and is lost when it is converted to stitches. But, because the program itself knows how it converts these vector shapes into the given stitches saving this information allows it to be reloaded and altered, re-rendering the shapes into stitches and saving them out.

It is for this reason that many improperly converted PES files will load in software like Wilcom’s software or OESD, etc. Because these software suites load the color/ stitch data, whereas Brother’s software loads the format CEmb data directly.

File structure :

The file is structured therefore into two main sections. The PES section and the PEC section.

- PES
  - PES header
    - “#PES00NN” PES Version number. Int32 location of PEC block.
    - Data.
  - PES Blocks.
- PEC
  - PEC Header.
  - PEC StitchData
  - PEC graphics.

There is also a file format called PEC it consists of “#PEC0001” followed directly by the PEC data.

PES Header v1

Address :	**Variable name :**	**Explanation :**
0000-0007	version	#PES followed by four bytes of ASCII version of format or ASCII of a pointed version (ex. 0001, 0055 for 5.5)
0008 - 000B	pecstart	4 Bytes, pointing to beginning of PEC codeblock stored in PES file, LSB first
000C-000D	hoop	00 00, means 100mm x 100mm hoop, 10 00 means 130mm 180mm hoop.
000E-000F		01 00 seems to be fixed and may mean scale to fit (UNVERIFIED). Changing it to zero seems to have no effect.
0010-0011	# of stitchgroups	This is the number of stitch groupings in the CEmbSewSeg and CEmbOne blocks. In save files made by brother this is most often equal the number of colors. In PES files made by others this is typically 1. If this number is zero, then the CEmbSewSeg and CEmbOne blocks are omitted. And there should be no stitches or they should be entirely made from the shapes defined in the shape-blocks.
0012-0015		ff ff 00 00 – Section break/termination

CSewSeg Block v1 (offsets are HEX from file start)

Address :	Variable name :	Explanation :
005F-0061		LSB int16_t length of structure name (07 00 for CSewSeg)
0061-0067		CSewSeg (43 53 65 77 53 65 67)
0068 – pecstart		This is CSewSeg Stitch Data. See below.
pecstart – fileend?		See PEC code block data

CEmbOne Block v1

Address :	Variable name :	Explanation :
0016-0017		LSB int16_t length of structure name (07 00 for CEmbOne)
0018-001E		CEmbOne (43 45 6d 62 4f 6e 65)
001F-0020	UNVERIFIED	This appears to be the minimum x of all stitches in CSewSeg stitch data
0021-0022	UNVERIFIED	This appears to be the minimum y of all stitches in CSewSeg stitch data
0023-0024	UNVERIFIED	This appears to be the maximum x of all stitches in CSewSeg stitch data
0025-0026	UNVERIFIED	This appears to be the maximum y of all stitches in CSewSeg stitch data

CembOne Block ends here, immediately goes into a Stitch Group

Stitch Group Data (offset vs. start of definition)

This element is found in both the Sew Segments Block and the EmbOne block. When in the the EmbOne Block it's preceded by a duplicate copy of the extents (minX, minY, maxX, maxY) and in the standard block format. And in Sew Segments in the given SewSegments format.

Address :	Variable name :	Explanation :
0000-0001	UNVERIFIED	Minimum x of all stitches in stitch group
0002-0003	UNVERIFIED	Minimum y of all stitches in stitch group
0004-0005	UNVERIFIED	Maximum x of all stitches in stitch group
0006-0007	UNVERIFIED	Maximum y of all stitches in stitch group

(Taken together these 4 shorts are the extents, the amount which each item extends in the various directions).

0008-0011		4 bytes: float 1, 00 00 80 3F (Scale X)

0012-0015	4 bytes: float 0, 00 00 00 00 (Skew_X)
0016-0019	4 bytes: float 0, 00 00 00 00 (Skew_Y)
0020-0023	4 bytes: float 1, 00 00 80 3F (Scale_Y)

(Taken together these 6 floats are the relevant factors of an AffineTransform or Matrix.)

0024-0027	4 bytes: float (350 + HoopWidth/2 - WidthOfDesign/2), 350 is the distance from 0,0 that the 130mm x 180mm hoop is stored, if we add half the HoopWidth (1300 / 2) and subtract half our design's width. The embroidery will be centered in the hoop.
0028-0031	4 bytes float (100 + HoopHeight/2 - HeightOfDesign/2), 100 is the distance from 0,0 that our 130mm x 180mm hoop is stored, if we add half the HoopHeight (1800 / 2) and subtract half our design's height. The embroidery will be centered in the hoop.
0032-0033	2 bytes: 01 00, unknown.
0034-0035	2 bytes: 0, X-location
0036-0037	2 bytes 0, Y-location
0038-0039	2 bytes width, WidthOfDesign
0040-0041	2 bytes height, WidthOfDesign.
0042-0045	4 bytes, 0, unknown.
0046-0049	4 bytes, 0 unknown.
0050-0051	2 bytes, number of segments + (2 * colorChanges), this is going to be the number of segments we will need to use.
0052-0055	4 bytes FF FF 00 00. End StitchGroup definition.

Understand that some of these items must only be set once. If you set a position, you do not need to apply the transformation with the Translate_X and Translate_Y. It is also acceptable to set most of the values to zero and **only** use the translation of the matrix to set the position.

PEC Code Block (offsets are in decimal)

Address :	**Variable name :**	**Explanation :**
pecstart 0 - 18	UNK_NN	Label (often the name of a file prior to the .EXT). Start with “LA:” (4C 41 3A). Any empty value is a space (HEX 20).
pecstart 19	UNK_FIXED	HEX 0d (carriage return)
pecstart 20 - 31	UNK (FIXED?)	spaces (HEX 20)
pecstart 32 - 35	UNK_FIXED	ff 00 06 26

pecstart 36 - 40	UNK (FIXED?)	Spaces (HEX 20), seen Hex 64 (This to pecstart 48 seems to supposed to be spaces (HEX 20) – does it have a meaning?)
pecstart 41 - 44	UNK (FIXED?)	20 00 20 00 (space null space null, is this a marker? Or is this someone writing spaces as 16 bit ints?), also seen 20 20 20 20 (the all 20's is in professionally designed items)
pecstart 45 - 47	UNK (FIXED?)	20 20 20
pecstart 48		No. of colors in file (0 is 1 color, so always add 1)
pecstart 49 - 49 + number of colors above (at least + 173)		This is the index into some magical color table. Color changes in stitch data refer to this table, which then bounces elsewhere. The offset is a 0-start count of the color change. The 1 byte index value is the brother color #. Number of colors (pecstart 48) is one byte. Is the high bit in the first byte (&80) meaning that the next byte is a color above 125/127? (Baby Lock high end machines have 125 possible color changes.) I haven't even see colors yet with high bit set. NOTE: It appears that this goes to 511. Any unused entry is HEX 20
Pecstart 512 - 513	UNK (FIXED)	HEX 00 00
pecstart 514 - 516	graphic	3 Bytes pointing to beginning of pixel graphic, LSB first, this is an offset from the byte previous to the start of this. (+513 gives you the data after the ff 00 ending of the stitches, + 512 gives you the 00)
pecstart 517 - 519	UNK (FIXED)	3 Bytes, 31 FF F0
pecstart 520 - 521		2 Bytes, x-size of design, LSB
pecstart 522 - 523		2 Bytes, y-size of design, LSB
pecstart 524 - 525	UNK (FIXED)	HEX E0 01
pecstart 526 - 527	UNK (FIXED)	HEX B0 01 NOTE: B0 happens to be the second byte of the color change code. I don't know if that means anything.
pecstart 528 - 531	UNK_NN	NOTE: The first and third bytes always seem to be close (exact, or off by 1 or 2). They don't seem to be mathematically related to the size of the design.

pecstart 532		Beginning of stitch data Stitch data is in PEC Code Block Stitch Data format. See Below.
pecstart graphic + 512		End of stitch data: To compute how far to read for stitch data add 512 to graphic and pecstart, this is a file offset for last byte of stitch data (00).
pecstart graphic + 513	PEC image	There are N sections, where N = number of colors + 1. An image for each thread color + 1 for the overall thumbnail. Each section has 228 bytes. There are no section marks. Each section comes one after the other. Each section is a pixel image with 38 rows and 48 columns, each bit = one pixel, drawn from top left to bottom right. For every byte out of these 228 bytes –> convert into binary form –> reverse the order of bits (The bigs are Big Endian, most machines are Little Endian) –> replace 1 for a pixel. In order to view the image, display 6 bytes per line, 38 lines in total, and voila! Each pixel image is framed with 1 and 0′s. This is the reason probably the images look delimited by marks, at byte level. Or, in other words, The outermost rectangle (1st row, last row, 1st column, last column) is made of 0's. If you look at the next inside rectangle (2nd row, 2nd to last row, 2nd column and 2nd to last column) you will see mostly 1's with the exception of the corners which make the rectangle round. For instance, upper-left corner is always: 0000000 0000111 0001000 001*0*xxx 010xxxx 010xxxx 010xxxx It is possible that there is another inner "rounded rectangle" frame made of 0's.

It is just a visual thing. This frame is always the same, for all images. So the pixels of the image (x), without the frame, are bound between rows 4-35 and between columns 4-45. I think it would be good if this can be checked again because I'm not sure if the bit (4,4) is always 0 or it can be used by the image. I think it always stays 0.

Note: All counts in this explanation (PEC image) start at 1, not at 0.

CSewSeg Stitch Data (all values uint16_t LSB, unless stated)

Type	Variable name :	Explanation :
	Color_Change_Indicator	Used to identify if BLOCK CHANGE is real. (The first of these encountered in the file is used to tell if a block ending is a real one or something else. If the first and current match, it is a block change.) Is this always 0?
HEADER	color_index	2 bytes for color index, not sure where this indexes to, maybe same as PEC data.
	stitches	2 bytes, this is how many stitches to expect in the segment, including the initial jump/position stitch (does not include “junk” stitches that are part of termination).
	x	Absolute x position where to move or to stitch from current location.
	y	Absolute y position where to move or to stitch from current location.
STITCHES	NOTES: 1 x and y must come in pairs (x,y). 2 Also, to do a jump stitch (better called move without stitch) duplicate the (x,y) pair to cause it to move and then stitch from that location without stitching to it. Sometimes it comes as a triplicate instead of duplicate. When 2 and when 3 are unknown. 3 First and second stitch on some files are a jump stitch to position. Others, which seem to be professionally digitized, do not do this for the first stitch and it appears a new block means the first stitch is actually a position code, although some terminations duplicate this stitch even in those same files, so is it or is it not necessary.	
BLOCK CHANGE		X of stitch is 32771 (03 80) and Y of stitch is the same as the first color_change_indicator in the file.

NORMAL BLOCK TERMINATION, NOT INCLUDED IN STITCH COUNT (NOT IN PROFESSIONALLY DIGITIZED DESIGNS?)	Unless otherwise noted, these are fixed values. I am calling them stitches only because it follows the same 2 2 byte values as a stitch. stitch 1 x: 1, y: 0 stitch 2 x: 12 (NN: Color of block which is ending), y: 32769 (01 80) stitch 3 x: 875 (UNK_NN), y: 901 (UNK_NN) stitch 4 x: 1096 (UNK_NN), y: 1114 (UNK_NN) stitch 5 same as stitch 3 stitch 6 same as stitch 4 stitch 7 x: 0, y: 16256 (80 3F) stitch 8 x: 0, y: 0 stitch 9 x: 0, y: 0 stitch 10 x: 0, y: 16256 stitch 11 x: 0, y: 0 stitch 12 x: 0, y: 0 stitch 13 x: 1, y: 875 (same as s3 x) stitch 14 x: 1114 (same as s4 y), y: 221 (UNK_NN) stitch 15 x: 213 (UNK_NN), y: 0 stitch 16 x: 0, y: 0 stitch 17 x: 0, y: 1
NORMAL BLOCK TERMINATION, NOT INCLUDED IN STITCH COUNT	This block termination only seems to exist if there are jump stitches within the block in question, and it is not the first block. The last line replaces the first copy of the double stitch (position and then stitch) found as the part of the next block. stitch x: 32771, y: 1 (probably not fixed, but anything but color_change_indicator) stitch x: 52 (NN: Color of the block that is ending), y: 1 stitch x: 1191 (NN), y: 914 (NN)
LAST BLOCK TERMINATION, COMPLETELY FIXED (NOT IN PROFESSIONALLY DIGITIZED DESIGNS?)	Unless otherwise noted, these are fixed values. stitch x: 1, y: 0 stitch x: 20 or 24 (NN: Color of the block that is ending), y: 0 stitch x: 0, y: 16716 (4C 41)
LAST BLOCK TERMINATION, COMPLETELY FIXED	The FAKE_STITCHES seems to be the count between that line and the 0, 16716 pair. stitch x: 7 (NN: FAKE_STITCHES), y: 0 stitch x: 45 (UNK_NN) , y: 1 (UNK_NN) stitch x: 29 (UNK_NN), y: 2 (UNK_NN) stitch x: 58 (UNK_NN), y: 5 (UNK_NN)

stitch x: 52 (UNK_NN), y: 8 (UNK_NN)
stitch x: 4 (UNK_NN), y: 11 (UNK_NN)
stitch x: 21 (UNK_NN), y: 14 (UNK_NN)
stitch x: 23 (NN: Color of the block that is ending), y: 0
stitch x: 0, y: 16716

NOTES: Common to both terminations seems to be the first stitch (x: NN: FAKE_STITCHES, y: 0), the second to last (x: NN: Color of the block that is ending), y:0) and the last stitch (x: 0, y: 16716)

PEC Code Block Stitch Data (all values are single bytes, stitches/jump stitches are relative to current position)

Type	Byte values:	No. of Bytes:	Explanation
Color Change	kx=254, ky=176, NN	3	Byte following ky gives color No. NN seems to always follow the pattern of 2,1,2,1...
Jump Stitch	128 <= kx <= 254, ky	2	lower four bytes (nibble) of kx is multiplication factor for jump stitch ky, direction of jump is determined as follows:
	(kx and 15) <= 7		jump in positive direction. length = ky + (kx and 15) x 256 C: if(kx & 0x80) ky + ((kx & 15) << 8)
	(kx and 15) >=8		jump in negative direction. length = (ky-256) + ((kx and 15)-15) x 256 C: same as positive, except if result & 2048, -=4096
Regular stitch data	kx,ky	2	0 <= kx <= 63 : positive 64 <= kx <= 127 : negative, kx=kx-128 C: if (kx & 64) kx -=128 0 <= ky <= 63 : positive 64 <= ky<= 127 : negative, ky=ky-128 (ky &64) C: if (ky & 0x80) ky -=128
End of stitch data	kx = 255,	2	

ky = 0

PES Block v6

Address :	Variable name :	Explanation :
0000-0007	version	#PES0060
0008 - 000B	pecstart	4 Bytes, pointing to beginning of PEC codeblock stored in PES file, LSB first
000C-000D	hoop	00 00, means 100mm x 100mm hoop, 10 00 means 130mm 180mm hoop.
000E-000F	UNK_NN	
0010-0011	# of stitchgroups	This is the number of stitch groupings in the CEmbSewSeg and CEmbOne blocks. In save files made by brother this is most often equal the number of colors. In PES files made by others this is typically 1. If this number is zero, then the CEmbSewSeg and CEmbOne blocks are omitted. And there should be no stitches or they should be entirely made from the shapes defined in the shape-blocks.
0069		DEC 10 (HEX 0a) (length of EMBROIDERY)
006A-0075		EMBROIDERY (45 4d 42 52 4f 49 44 45 52 59 00) Always followed by 15 bytes.
0075-0083	UNK_NN	
0084		DEC 10 (HEX 0a) (length of EMBROIDERY)
0085-008E		EMBROIDERY (45 4d 42 52 4f 49 44 45 52 59 00) Always followed by 15 bytes.
008F-009E	UNK_NN	
009F		DEC 10 (HEX 0a) (length of EMBROIDERY)
00A0-00AB		EMBROIDERY (45 4d 42 52 4f 49 44 45 52 59 00) Always followed by 15 bytes.
00AC-00B9	UNK_NN	
00BA		DEC 10 (HEX 0a) (length of EMBROIDERY)
00BB-00C5		EMBROIDERY (45 4d 42 52 4f 49 44 45 52 59 00) Always followed by 15 bytes.
00C6-00D3	UNK_NN	
00D4		DEC 10 (HEX 0a) (length of EMBROIDERY)
00D5-00DF		EMBROIDERY (45 4d 42 52 4f 49 44 45 52 59 00) Always followed by 15 bytes.
00E0-00FD	UNK_NN	
00FE		DEC 10 (HEX 0a) (length of EMBROIDERY)
00FF-00FA		EMBROIDERY (45 4d 42 52 4f 49 44 45 52 59 00) All others are followed by 15 bytes, this one bye the 01 00 termination

00FB-00FC	01 00 (is this the same termination seen in v1 where this is 01 00 for professional and the number of colors in PEC otherwise?
00FD-0100	ff ff 00 00 – Section break/termination

NOTES: It appears that the 64 00 (LSB DEC 100) is the offset to the beginning of CEmbOne section. Whether or not all of them are static I do not know. There are 6.

It appears that the 80 3F section from CEmbOne v1 (maybe different number of zeros around it) is the same. It repeats.

Maybe 0a (45 4d 42 52 4f 49 44 45 52 59 00) ends a section, not begins it?!?! Just a random thought.

It is fairly certain that CEmbOne v1 is used for PES v6, but is at a different offset. It appears that CSewSeg v1 is used for PES v6, but I haven't verified it yet.

It appears that after the PEC code block and whatever structured data (starting after the last stitch), in PES v1, there appears to be repetitive data in v6. It may have some variation.

Credits

A special thanks for the individual who put up: http://www.achatina.de/sewing/main/TECHNICL.HTM It is the original basis to this document.

A special thanks to all those who provided code and information at: https://bugs.launchpad.net/inkscape/+bug/247463 as it provides some clarification to the achatina.de site above.

A special thanks to Linus Torvalds for for his blog post and code, found respectively at: http://torvalds-family.blogspot.com/2010/01/embroidery-gaah.html and http://git.kernel.org/?p=linux/kernel/git/torvalds/pesconvert.git;a=summary

Thank you to Nathan Crawford for: http://www.njcrawford.com/programs/embroidery-reader/pes0001/ and http://www.njcrawford.com/programs/embroidery-reader/csewfigseg/

Thank you to Josh Varga, author of Emboridery Modder, for so very much enlightenment on this file format.

Thank you Madalina for her work on PEC image format and her permission to include it. Her original work can be found at: http://madamade.com/wp/?p=10

Thank you to David Olsen for his massive contributions to this document.

The following is work from David Olsen which has not yet been merged with the above document. I include it now for the use of those interested before I have been able to massage it in. It may be ready as is, but will require updating information above.

PES Blocks:

All PES blocks follow the same general format.

C- General Block Format

Address :	**Variable name :**	**Explanation :**
2 bytes	length	Short: length of structure name
length bytes	name	Name of the block in ASCII.
NN	block data	Data of various length.
2 bytes	break.	ff ff 00 00 – This appears to be section break/ termination

Types of PES blocks include:

CEmbOne, CSewSeg, CEmbCirc, CEmbRect, CEmbLine, CEmbPunch, CSewFigSeg, CEmbNText, CLetter

The blocks are omitted if there is no relevant data in them (including the sew segments block, if the number of stitchgroups is set to 0). If there are multiple types of the same object the objects are placed inline in the their respective block. There is never more than one instance of the block type.

The CEmbOne Block:

The CEmbOne block is in the general block format, and contains the first stitchgroup data entity. When loaded into PE-Design this is the bounding box of the data, the data associated with that bounding box, the affine transformation, the location, size, etc. This defines the first such block. Additional stitchgroups are defined inline within the CEmbSewSeg block data.

This block starts with the standard block header for #CEmbOne and is then followed by:

Short: minX

Short: minY

Short: maxX

Short maxY

StitchGroup Data.

Block termination FF FF 00 00

The extends for the block are duplicated in the Stitchgroup data. So the block will always have two sets of the extends in a row to begin with. But, one of these is naturally in the CEmbOne block and the other in the StitchGroup.

See: StitchGroup Format.

The CSewSeg Block:

The CSewSeg block stores a sequence of segment lists, in shorts, based on their position within the window defined in the stitchgroup. The first stitchgroup is defined in CEmbOne Block. All data stored in this block is stored in the form of a header, body, and end. The end is omitted with regard to the final segment. See, Segment Format.

Segment Format:

Short, either 0 or 1, 0 means jump stitches or non-stitching event. 1 means stitching event. Within the stitchgroup, this data must alternate between 0 and 1.

Short, color. The index of the color code.

Short, length. The length of this segment. If this is 01 80 0x8001 (-32767) then this is defining the next stitchgroup.

Stitches are defined in groups of 2 shorts.

Absolute value of the position X relative to the location of the stitchgroup.

Absolute value of the position Y relative to the location of the stitchgroup.

Short, -32765, in bytes: 03 80.

If defined inline with the segments the StitchGroup is 00 00 (non-sewing), NN 00 (color), 01 80 (-32767 for length). This is followed by the StitchGroup Format. This will have 1 set of extend data.

StitchGroup Format:

This element is found in both the Sew Segments Block and the EmbOne block. When in the the EmbOne Block it's preceded by a duplicate copy of the extents (minX, minY, maxX, maxY) and in the standard block format. And in Sew Segments in the given SewSegments format.

Short: minX

Short: minY

Short: maxX

Short: maxY
(Taken together these 4 shorts are the extents, the amount which each item extends in the various directions).

Float: Scale_X
Float: Skew_X
Float: Skew_Y
Float: Scale_Y
Float: Transform_X
Float: Transform_Y
(Taken together these 6 floats are the relevant factors of an AffineTransform or Matrix.)

Short: 1, unknown.

Short, X-position

Short, Y-position.

Short, width

Short, height

Int32, 0, unknown.

Int32, 0, unknown.

Understand that some of these items must only be set once. If you set a position, you do not need to apply the transformation with the Translate_X and Translate_Y. It is also acceptable to set most of the values to zero and *only* use the translation of the matrix to set the position.

File structure:

PES 0001 structure is typically:

- PES header.
- CEmbOneBlock
- CSewSeg Block
  - Contains one or more CSewSeg Stitch Data Blocks, and Stitch Groups.
- PEC code block
  - PEC Header.
  - Contains PEC Code Block Stitch Data.
  - Graphics

Example Write Routine:

We can read PES and PEC for all versions by reading the color data and stitch data from the PEC block. But, the typical use case is going to be writing a successful PES file.

[all writes should be done Little Endian].

We must write the following things to disk.

8 bytes: "#PES0001"

4 bytes: Location of PEC block.

2 bytes: 1 (scale to fit)

2 bytes: 1 (hoop size, 130mm 180mm)

2 bytes: 1 (we are going to write only one stitchblock object).

4 bytes: FF FF 00 00, end the header.

2 bytes: 7 (length of text "EmbOne")

7 bytes: "EmbOne"

2 bytes: 0, minX (we're passing on this)

2 bytes: 0, minY (we're passing on this)

2 bytes: 0, maxX (we're passing on this)

2 bytes: 0, maxY (we're passing on this).

(Starting StitchGroup definition)

2 bytes: 0, minX (we're passing on this)

2 bytes: 0, minY (we're passing on this)

2 bytes: 0, maxX (we're passing on this)

2 bytes: 0, maxY (we're passing on this).

4 bytes: float 1, 00 00 80 3F (Scale X)

4 bytes: float 0, 00 00 00 00 (Skew_X)

4 bytes: float 0, 00 00 00 00 (Skew_Y)

4 bytes: float 1, 00 00 80 3F (Scale_Y)

4 bytes: float (350 + HoopWidth/2 - WidthOfDesign/2), 350 is the distance from 0,0 that the 130mm x 180mm hoop is stored, if we add half the HoopWidth (1300 / 2) and subtract half our design's width. The embroidery will be centered in the hoop.

4 bytes float (100 + HoopHeight/2 - HeightOfDesign/2), 100 is the distance from 0,0 that our 130mm x 180mm hoop is stored, if we add half the HoopHeight (1800 / 2) and subtract half our design's height. The embroidery will be centered in the hoop.

2 bytes: 1, unknown.

2 bytes: 0, X-location (we're passing on this).

2 bytes 0, Y-location (we're passing on this).

2 bytes width, WidthOfDesign

2 bytes height, WidthOfDesign.

4 bytes, 0, unknown.

4 bytes, 0 unknown.

2 bytes, number of segments + (2 * colorChanges), this is going to be the number of segments we will need to use.

(End StitchGroup definition)

4 bytes FF FF 00 00, end block.

2 bytes: 7, length of "CSewSeg"

7 bytes: "CSewSeg"

(Writing Segment Information)
For these, we must ensure our segments are flagged 0,1,0,1,0,1,0. That our jumps are 0, that our stitches are 1, that our color changes are 0, we pad with jumps and stitches to nowhere to maintain this. We assume that we jump to the location, then color change, then stitch as relevant.

(Color Change, Initialize)

2 bytes: 1 this is a stitch.

2 bytes color, color index.

2 bytes 2, needs a length of at least two to not be a color change.

2 byes: X, current position X.

2 bytes: Y, current position Y.

2 byes: X, current position X.

2 bytes: Y, current position Y.

2 bytes 0x8001, 01 80. End Segment

2 bytes: 0, non-stitching.

2 bytes, color, color index.

2 bytes, 2, needs a length of at least two to not be a color change.

2 byes: X, current position X.

2 bytes: Y, current position Y.

2 byes: X, current position X.

2 bytes: Y, current position Y.

2 bytes 0x8001, 01 80. End Segment

(We didn't call a color change event, but we ensured that the initial color is established)

(Color Change, Middle)

2 bytes: 0, we are not sewing this

2 bytes: Color, color index. Index of old color.

2 byte: 1, length of stitches.

2 byes: X, current position X.

2 bytes: Y, current position Y.

2 bytes 0x8001, 01 80. End Segment

2 bytes: 1, we are sewing this.

2 bytes: Color, color index. Index of new color.

2 bytes: 2, length two.

2 byes: X, current position X.

2 bytes: Y, current position Y.

2 byes: X, current position X.

2 bytes: Y, current position Y.

2 bytes 0x8001, 01 80. End Segment

(We invoked a color change, and then a sewable event that went nowhere, to make sure the color is established.)

(Jump)

2 bytes, 0, we are not stitching.

2 bytes, Color, whatever color we're on.

2 bytes, length, the number of stitches in this jump, including start and end (minimum 2).

NN bytes, X position, Y position (relative to the position given to the StitchGroup description).

2 bytes 0x8001, 01 80. End Segment

(Stitch)

2 bytes, 1, we are stitching.

2 bytes, Color, whatever color we're on.

2 bytes, length, the number of stitches in this block, including start and end (minimum 2).

NN bytes, X position, Y position (relative to the position given to the StitchGroup description).

2 bytes 0x8001, 01 80. End Segment

(End Segments)

We now write the color change log.

4 bytes: Number of color changes

For each color change:

2 bytes: Color Change, old index value.

2 bytes: Section at which this change occurred. (Note this must account for the initialization and the padding sections we added.

(Starting PEC elements).

Write PEC Elements.

(Write graphics)

1 copy of blank.

N copies of blank. (where N is the number of colors used).

Where blank is defined as:

```
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0xF0, (byte) 0xFF, (byte) 0xFF, (byte) 0xFF, (byte) 0xFF, (byte) 0x0F,
(byte) 0x08, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x10,
(byte) 0x04, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x20,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x02, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x40,
(byte) 0x04, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x20,
(byte) 0x08, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x10,
(byte) 0xF0, (byte) 0xFF, (byte) 0xFF, (byte) 0xFF, (byte) 0xFF, (byte) 0x0F,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00};
```