Corel® PHOTO-PAINT®
CPT file format specification

draft version: 0.01

1. Overview

CPT file format serves as a native container for layered images made in Corel PHOTO-PAINT[1] (from now on I will refer to Corel PHOTO-PAINT as PP, concatenated together with version when appropriate, i.e. PP6 meaning “Corel PHOTO-PAINT 6.0” etc). It is functionally comparable to Adobe's PhotoShop PSD format or GIMP's XCF. CPT files, as opposed to “flat” bitmap formats like PNG hold several types of data and information: layers (stacked up bitmaps covering each other, together with information about their position, alpha channel, transparency, etc.), better known in Corel's terminology as *objects* (I am going to use this one rather than Adobe's layers*);* lenses (dynamically cast bitmap effects on limited area of image); masks, ICC color profile, grid and rulers information, etc.

The list above is by no means complete; for more detailed description see paragraph 3.

1.1. File format versions

CPT format has a few versions. Corel PHOTO-PAINT default output format version, as well as ability to read given format version, depends on PP version itself. Detailed information follows:

1. **CPT version 6.0** is produced by PP6 and is completely different from later versions. It is a TIFF based file format. Object information is saved; several applications able to view images in this format are available [TODO: place list together with URLs here]. It can be considered obsolete. No more analysis on this format has been done.
2. **CPT version 7.0** is produced by PP7 and PP8. In case of PP8 this is the default output format unless [TODO: check which] functionality is used.
3. **CPT version 7.01** is produced probably by PP7 or PP8. Code executed by PP's filter checks against such file version, however I have been unable to produce such file myself (or find one) [TODO: needs more investigation].
4. **CPT version 8.0** is produced by PP8 when [TODO: check which] functionality is used. It seems to a slightly modified variant of PP7. It seems to be produced by PP8 only [TODO: check how PP9+ behaves when saving to CPT7 format: does it produce CPT8 or CPT7? On which circumstances?]
5. **CPT version 9.0** is produced by PP9 and all later versions.

[TODO: check how the situations looks with PPX4]

Following table covers versions read (left column) and written (right colum) by different versions of PHOTO-PAINT (black dot = true, white dot = false, hyphen = not tested / don't know):

PP ▼ CPT ►	6.0		7.0		7.01[2]		8.0		9.0	
6	●	●	○	○	○	○	○	○	○	○
7	●	●	●	●	●	-	-	○	○	○
8	●	-	●	●	●	-	●	●	○	○
9	●	-	●	-	●	-	●	○	●	●
10	●	-	●	-	●	-	●	○	●	●
11	●	-	●	-	●	-	●	○	●	●
12	●	○	●	●	●	-	●	○	●	●
X3	●	-	●	-	●	-	●	○	●	●
X4	-	-	-	-	-	-	-	-	-	-

Table I: Summary of CPT versions read and written by different PP versions

1 All brands and product names used throughout this document have only informative character and might be trademarks or registered trademarks of their respective owners, even if they are not marked as such.

2 Values in left column are probably a good guess (see CPT version 7.01 remarks).

2. Specification

All information provided here describes CPT 7.0 and newer formats.

[TODO: describe CPT mask files]

As mentioned earlier, PP file format evolved throughout time, thus each of the subformats differs and has to be described separatly. However, both subformats share several fields and properties in common; those are described below.

2.1. Tables, formulas

Value column is always expressed in hexadecimal, unless stated other explicitly.

2.1.1. Color models

Table below describes available **color models**. Value is a byte value used by PP to identify model, Bits means BPP (bits per pixel) in given color model.

Value	Bits	Color model
0x01	24	RGB
0x03	32	CMYK
0x05	8	grayscale
0x06	1	black and white
0x0A	8	RGB (indexed)
0x0B	24	LAB
0x0C	48	RGB
0x0E	16	grayscale

Tabela II: CPT color models

2.1.2. Measure units

Table below describes available **measure units** and their values (it is not a mistake; some of the PHOTO-PAINT's measure units values seems to be a bit odd / non-standard / inadequate, however, it is possible it's my erronous analysis / calculation / rounding errors. Especially cicero and didot units seem to have strange scaling values).

Value	Unit	Scale factor
0x01	inch	25.4
0x02	mm	1.0
0x03	pica; point	25.4 / 6.0
0x04	point	25.4 / 72.0
0x05	cm	10.0
0x06	pixel[3]	25.4
0x0C	cicero; didot	4.5118699999999997
0x0D	didot	0.37591999999999998

Tabela III: CPT measure units

Finally, scale factor needs to be divided by a measure of 10^4. Pixel unit is a special case, which requires to be further multiplied by DPI value. [TODO: check if this is a rounding bug or bug in PP. Add some Wikipedia info for measure units lengths and conversion]. (Note: the letter C and D in hex means cicero and didot :-)).

3 Read further for more information.

2.1.3. ICC profiles

Corel PHOTO-PAINT allows embedding ICC profile information into file. Table below represents meaning of values for different profiles [TODO: description in this section needs to be clarified] and is valid for CPT 9.0 (PP9+):

Value int32_t	ICC internal profile type
0	sRGB
1	Fraser (1998)
2	SMTPE-240M
3	NTSC (1953)
4	PAL
5	SECAM
6	Barco – D50
7	Barco – D65
-1	(embedded profile)

Tabela IV: CPT internal profile types

If value -1 is encountered, it means ICC profile is non-standard and provided as an embedded file. [TODO: check if PPX3 and PPX4 extend this list in any way. For which PP versions this table is valid?]:

2.1.4. Character encodings

CPT uses two types of encodings for text:

- 8-bit (narrow characters) encoding (further referenced as *ANSI*). Actually, this encoding is not strictly defined as such; Corel PHOTO-PAINT uses Windows' current system encoding when saving files, so this might result in text being CP1250 as well as CP1252 encoded, etc. PP's behavior is to allow user to select encoding when file is being opened, so any import plugin needs to mimic this behavior (however, one can suggest an encoding to user based on some autodetection algorithm).
- 16-bit (wide characters) encoding (further referenced as *UCS2*). This encoding is exactly UCS-2LE (Windows “wide” character).

CPT9 files have text embedded using both encodings; lower versions use only ANSI variant. Both encodings character widths are constant. [TODO: describe somewhere a bug in PP when writing long *UCS2* comments together with embedding ICC profile].

I will call *ANSI* or *UCS2 string* an array of 8- or 16-bit (respectively) characters terminated with a NULL char (0x00 or 0x0000).

2.1.5. DPI calculation

DPI value written in CPT file header uses somewhat specific unit. In theory, to get the real DPI value, one should use formula:

$$\text{real\_dpi} = \text{dpi} \cdot 25.4 / 10^6$$

where 25.4 is a number of milimeters in 1 inch. However, above formula has given me incorrect results in almost all cases. Algorithm in C, which results in correct values is given below:

```
const double cpt_dpi_scale = 25.399986284007403 / 1000000
dpi = lround((double) (header->dpi_h) * cpt_dpi_scale);
```

This certainly needs clarification.

Minimum and maximum (real) values of DPI for CPT images are equal 10 and 10000, respectively.

2.2. File master blocks

CPT file is primarly organized in something I have called Master Blocks (MB):

Offset uint32_t	Master Block type	Master Block name
0x0000	mandatory	File header
0x0100	optional	ICC profile
. . .	optional	UCS-2 comment
blocks_array_offset	mandatory	Image blocks offsets array
. . .	mandatory	Image block 1
. . .	optional	Image block 2
. . .	optional	...
. . .	optional	Image block n

Tabela V: CPT Master Blocks

Optional Master Blocks are placed between File header and Image blocks offsets array. ICC profile and UCS-2 comment are CPT9 specific and their presence is marked by File header **flags** field. Each optional field seems to be recognizable by it's magic. ICC profile is a variable-length MB.

Notice: for some reason CPT if both ICC profile and UCS-2 comment are present, always puts those two MBs in order given in table above.

2.2.1. File header

Corel PHOTO-PAINT's file header is fixed length (256 bytes), mandatory master block starting at file offset 0. The block is constructed like this:

Offset uint8_t	Type	Len	Field, description
0x00	uint8_t	8	**magic** – constant length CPT file identifier. Allowed values: • “CPT7FILE” (means CPT 7.0 or 7.01) • “CPT8FILE” (CPT 8.0) • “CPT9FILE” (CPT 9.0)
0x08	uint32_t	1	**color_model** – image color model used.
0x0C	uint32_t	1	**palette_length** – number of colors * 3 in case of 8-bit RGB paletted image[4]; 0 otherwise. Allowed values: 0 or [1..256] * 3.
0x10	uint32_t	2	**reserved** – seems to be always 0.
0x18	uint32_t	1	**dpi_h** – image horizontal DPI in CPT DPI units. See DPI calculation for details. Allowed values (after scaling): [10..10000] or 0 (for CPT mask files only).
0x1C	uint32_t	1	**dpi_v** – image vertical DPI in CPT DPI units. Allowed values (after scaling): [10..10000] or 0 (for CPT mask files only).
0x20	uint32_t	2	**reserved** – seems to be always 0.
0x28	uint32_t	1	**blocks_count** – number of CPT_Block blocks. Allowed values: > 0 (upper limit unknown, it is probably tied up with upper limit of objects[5]).
0x2C	uint32_t	1	**unknown** – seems to be always 0x10000[6].
0x30	uint16_t	1	**reserved** – seems to be always 0.
0x32	uint16_t	1	**flags** – CPT file flags. Lower [TODO: verify] 8 bits seem to be exact CPT format version (?). Values[7] met: • 0x01 (files with “CPT7FILE” magic) • 0x8C (“CPT8FILE” magic) • 0x94 (“CPT9FILE” magic) Upper [TODO: verify] 8 bits seem to be file flags interpreted as follows (boolean values, 1 = true): • bit 0: ICC profile block present • bit 1: UCS-2 file comment present • bits 2-7: unknown, seem to be always 0
0x34	uint32_t	1	**blocks_array_offset** – CPT file offset where CPT_Block offsets array is placed. This value is always 0 for CPT7 and CPT8 files created by PP7-PP8 (see Block table paragraph for calculation algorithm) and non-zero for CPT9 files and CPT7 files created by PP9+.
0x38	uint32_t	1	**reserved** – seems to be always 0.
0x3C	uint8_t	112	**comment** – file comment as *ANSI string*.

Tabela VI: CPT File Header

...

2.2.2. ICC profile

ICC profile is variable length, optional block. The block is constructed like this:

2.3. CPT7, CPT8

It seems that the main difference between CPT7 / CPT8 format and CPT9 format is not the absence of optional MBs, but how Image blocks are constructed.

[TODO: ...]

4 Notice: on paletted images, CPT_Palette is always constant 768-bytes block anyway, so this field is not a “length” actually.
5 Would need a machine with plenty of RAM and good CPU to test it...
6 If manually set bigger, PP refuses to open such file displaying “Out of memory” error. Smaller values seem not to have any special effect.
7 Moreover, in original PP's filter plugin, PHOTO-PAINT makes bitwise AND of this value and 0xF0 and tests if result is equal 0x90. Then, if magic is equal “CPT7FILE” this file is considered as CPT 7.01.

2.3.1. CPT7 Image Block

[TODO: ...]

2.4. CPT9

[TODO: ...]

2.4.1. CPT9 Image Block

Each Image Block (IB) in CPT9 is constructed as follows:

Offset uint32_t	Image Block element
0x0000	IB header
0x003C	IB chunk 1
. . .	...
. . .	IB chunk n
. . .	IB data

Offset is relative to IB beginning. [TODO: ...]

Some of the IBs are *Objects*, however, not each IB is an *Object*.

IB header:

Offset uint8_t	Type	Len	Field, description
0x00	uint32_t	1	**width** – block (object?) width in pixels
0x04	uint32_t	1	**heigth** – block (object?) width in pixels
0x08	uint32_t	1	**tile_width** – tile[8] (???) width in pixels
0x0C	uint32_t	1	**tile_height** – tile (???) width in pixels
0x10	uint32_t	1	**bpp** – number of bits per pixel. It probably holds information of bpp per *Object* (it is possible to have different bpp per *Object* – eg. pasted *Objects* can have different bpp, which is probably converted (upcasted?) by PP on the fly)
0x14	uint32_t	1	**unknown0** – seems to be always 0, needs further research
0x18	uint32_t	1	**unknown1** – 1, 4, 8, maybe some other, various values, needs further research
0x1C	uint32_t	1	**unknown2** – 0, 1 or 2. It looks like 1 indicates IB is an *Object*.
0x20	uint32_t	1	**chunk_area_size** – probably, in bytes, needs further research
0x24	uint32_t	1	**palette_size** – not sure, can be always 0 for **bpp** = 8-bit
0x28	uint32_t	5	**unknown2** – probably reserved, seems to be always 0.

Chunk recognition legend:

chunk	Just guessing a name...
chunk	Slight idea what it is, some stuff recognized
chunk	Almost know what it is, most stuff recognized
chunk	Complete

8 I have no idea what are this dimensions for. “Tile” term is just a placeholder. Usually “tile” is 144x151 pixels, but sometimes 256x256 pixels or (probably if less than 144x151) the same as **width** and **height** fields.

Chunk name	Chunk description
aext	
anaw	... name (?) (*UCS2 string*)
anam	... name (?) (*ANSI string*)
aovr	
bnam	Background name (*ANSI string*)
bnwm	Background name (*UCS2 string*)
clpa	Clipping area (?)
docs	
duot	Duotone stuff (?)
grid	Grid information (?)
guid	
lres	
lthm	Zajebiście tajemnicze pole :-P
nmpa	
nozz	
nupa	
obln	
oduo	Object duotone stuff (?)
oinf	Object information (name as *ANSI string*, name as *UCS2 string*, some other unknown information)
olex	
olns	Lens information (?)
osdw	
ot10	OpenType stuff (?)
ot12	OpenType stuff (?)
othm	
otpp	
otx9	
otxt	
roid	
path	Path information (name as *ANSI string*, some other unknown information)
psdp	
pthw	Path information (name as *UCS2 string*, some other unknown information)
pthx	
tgpa	
titl	Title (of what? ;-)
urla	URL information (?) (*ANSI string*)
urlc	URL information (?) (*ANSI string*)
urls	URL information (?) (*ANSI string*)
urlt	URL information (?) (*ANSI string*)
urwa	URL information (?) (*UCS2 string*)

`urwc`	URL information (?) (*UCS2 string*)
`urws`	URL information (?) (*UCS2 string*)
`urwt`	URL information (?) (*UCS2 string*)
`vbai`	VBA macro (?)
`vbax`	VBA macro (?)
`viac`	
`vrhs`	
`vset`	
`wkpa`	