File: CC2Guide-SpriteFiles.PDF
Format: PDF
Date: February 21st, 2004
Author: Mafi; closecombat2@claranet.de; http://closecombat2.fortunecity.com/
Last revision v8: March 10th, 2010 – added CC2's style value inside the footer; WaR/TLD Terrain file

Close Combat 2 "A Bridge Too Far"

Sprite files of the Close Combat games

(PC- & Mac-version of CC2; also: CC1, CC3, CC4, CC5, CCM, RtB and newer)

What it is

"Close Combat - A Bridge Too Far" (abreviated CC2, ABTF, CC2-ABTF) was the second game of the CloseCombat-series created by Atomic and presented by Microsoft to the Mac-community. It was also the last game of this series for the MacOS. The series was then continued by SSI, UbiSoft and Destineer for PCs only (up to day CC3, CC4, CC5, CCM, The Road to Baghdad released in January 2004, abbreviated: RtB, CCM v2 released to the USMC in summer 2005, the re-releases by MatrixGames CoI, CCMT, WaR, TLD). CC2 was released in 1997 on a hybrid-CD, running on PCs and under the MacOS 7.5 up to 9.2.2 / MacOS X 10.2.8 / 10.3 / 10.4 (in Classic environment) as well. Later (localized) releases of CC2 were for PCs only. A trial demo of CC2 was also released in 1997.

Many thanks to ...

Many thanks to PEKKA SAASTAMOINEN aka "CPL_FILTH" for his great work. Without his program "SprTool.exe" the easy handling of sprite files of all CC games is nearly impossible and the following work could never be done by myself. Please look at his homepage for further development on CC2-CC3-CC4-CC5-RtB-tools: `http://www.student.oulu.fi/%7Epsaastam/` . He lend a hand in 2004 in building up my own tool "CC2Spriter" for MacOS and PC. Thanks for all the support, Cpl! Also many thanks to TIN TIN for his CC3-file format informations from his site at `http://www.organicbit.com/closecombat/` and to Nembo for further debugging infos on my CC2Spriter tool. And I have to thank Konrad for his SprPack tool for CC2/CC3 to understand better the Soldier/SoldierB files.

Sprite file basics

Sprite files in CC1 are used to store small graphics (commonly called "sprites") to represent vehicles and other map details during gameplay in different orientation or in an animation sequence. Later on this file format was used slightly modified in CC2 for the files "Explode", "Smoke", "Soldier", "SoldierB", "Terrain" and all vehicle shadow files ("VehS###" and "VehB###"). In CC3 most of the animation sequences moved into *.zfx files, leaving only the sprite files "Terrain", "Soldier" and "SoldierB". From CC4 onward (until now in "The Road to Baghdad" = RtB) all animation sequences are stored (as 24-bit graphics) in *.azp archives. The only sprite files remaining are again "Terrain", "Soldier" and "SoldierB". From CC4 onward the sprite file format changed again.

Sorry to say that I have not investigated the CC1-sprite file format completely. Here is what CPL_FILTH has reported to me and what I have found:

Sprite file format

```
// byte order in CC1, CC2, CC3: Mac-like (Motorola style) Big Endian
// byte order in CC4, CC5, RtB: PC-like Little Endian
//                              (Intel style reverse byte format)
// every sprite file is divided into 5 parts; I will refer to them as "sections"

// section: HEADER (8 bytes):
char(4)          // "SPRI" (CC1, CC2, CC3); "IRPS" (CC4, CC5, CCM, RtB)
long             // version-ID, always 00000001h in CC2, all Terrain-files
                 //        and in all *.nsd / *.zsd CC3 vehicle shadow files

// section: DIRECTORY (10 bytes):
short            // Directory-ID (2 bytes): 03E8h = 1000 (decimal)
short            // Number of sprite graphics in the "Sprite section" (2 bytes)
short            // Number of animation sequences in the "Static
                 //       animation section", called "Sequences" by Atomic
                 //       (2 bytes)
short            // Number of animation sequences in the "Direction-oriented
                 //       animation section", called "Rotosprites" by Atomic
                 //       (2 bytes)
short            // 2 bytes of unknown purpose. Cpl_Filth supposes that it
                 //       might be a color depth definition, but I think it
                 //       will be the maximum length of direction-based
                 //       animation sequences.

// section: SPRITES (of varying size):
short            // Sprite-section-ID (2 bytes): 03E9h = 1001 (decimal)
sprite_entry     // sprite entries, see seperate list
sprite_entry     // 3 different formats: CC1, CC2/CC3, CC4/CC5/CCM/RtB
sprite_entry
...

// section: STATIC ANIMATIONS (of varying size):
short            // StaticAnim-section-ID (2 bytes): 03EAh = 1002 (decimal)
staticseq_entry  // static animation sequence entries, see separate list
staticseq_entry
staticseq_entry
...

// section: DIRECTION ORIENTED ANIMATIONS (of varying size):
short            // DirectionAnim-section-ID (2 bytes): 03EBh = 1003 (decimal)
directseq_entry  // direction-oriented animation sequence entries,
directseq_entry  // see separate list
...

// end of file
```

The Header

The header of all sprite files is always 8 bytes long: first four bytes are used as a header-ID. For the games CC1, CC2 and CC3 the header-ID is "SPRI". For CC4, CC5, CCM and RtB the header-ID is "IRPS", indicating the reverse byte format. The next 4 bytes contain represent always the value 1, we suppose that it is some kind of version-ID. Only the vehicle shadow files (*.spr-files) introduced by CC4 (and used up to now by RtB) can have here 11 different values: 00000001h, AE7800h, AE7A00h, AE8400h, AE9E00h, EE7A00h (airplanes in CC4/CC5), FE9700h, FEA600h, FEBD00h, 12E7800h, 12E7A00h. Meaning of these values is still unknown.

The Directory

The second section contains some kind of directory. The start of this section is indicated by the value 1000 (in decimal) encoded in a short integer (2 bytes). The real encoding of this value depends on the selected byte format: CC1, CC2, CC3 sprite files use here the two bytes 03h and E8h. For CC4, CC5, CCM, RtB sprite files the byte sequence is E8h and 03h. All following values are encoded in short integers, too: the directory contains the numbers of entries in the following sections: the number of sprite graphics in the "Sprite section", the number of animation sequences in the "Static animations section" and the number of animation sequences in the "Direction oriented animations section". The last entry in the directory is of unknown purpose. In CC1, CC2 and CC3 it varies between 8 and 16. In the vehicle shadow files of CC3 (the *.nsd / *.zsd files stored in the file "Shadows.zfx") this value is always 64 (identical to the length of the one and only direction based animation sequence). And in CC4-RtB it is always 63. In the "Soldier" / "SoldierB" sprite files this value is always 0. The only excepton from the rule are the *.spr files: the values 63 or 64 are possible. It might be some kind of color depth (as supposed by Cpl_Filth) or a further value concerning the "Direction oriented animations section" (I suppose it is the "maximum length of sequences").

"Sprites section"

This section always starts with the short integer indicator 1001 (decimal). Internal encoding of this indicator depends on used byte format (see above). For every sprite graphic a seperate entry is used. The principle is always the same: each sprite entry contains of three parts: the header (describing the image size and hotspots), the line offset table and the pixel datas in 16-bit RGB-color (since CC2. CC1 stores them in some kind of 8-bit).

Now it is time to eliminate a CC-myth: the pixels are organized left-to-right, top-to-bottom. The supposing by Cpl_Filth (and others before him) that CC games store their graphics flipped is a result from adding a wrong TARGA-header to the extracted graphics. But in fact all CC graphics are stored top-to-bottom in the original files. The flipping in the graphics editors is a result of a wrong TARGA-header created by the older tools (like SprTool.exe)!

There are three different "sprite_entry" formats used:

CC1: similiar to the CC2 format. Fewer pixel datas than in CC2 because the pixels are encoded in 8-bit relating to one of CC1's color look-up tables (CLUTs). Will be discussed at the end of this guide.

CC2 and CC3: "sprite_entry" format (as discribed first by CPL_FILTH):

```
// header: 12 bytes
short sprite_width      // 2 bytes
short sprite_height
short hotspot_X
short hotspot_Y
long  sprite_data_size  // 4 bytes: number of bytes
                        //          of sprite data to follow
                        // (= size_of_offsetable + size_of_pixeldata)
// line offset table
short line_offset       // 2 bytes for each line of the graphic.
short line_offset       // offset is counted from start of pixeldata
short line_offset       // that means: first entry for a line containing
short line_offset       // pixels must be 0000h.
...                     // special in CC2/CC3: value FFFFh indicates
                        // that complete line is tranparent, no bytes
                        // will be in the pixeldata table for this line.
// pixeldata
data                    // with special run length encoding
...
```

The special encoding for the "pixeldata" is as follows (as originally written by Cpl_Filth):
the sprite pixel format is sort of a variation on run length encoding. The basic idea is this : a 2 byte code indicating pixel type, 2 bytes to indicate length of pixel run , and possibly (pixel run length) * 2 bytes of color data.

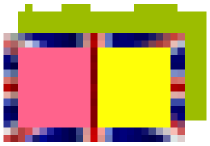
The codes we have seen are :

00h	transparent pixels.
FFh	color pixels, this is followed by both the length and the color values for the pixels.
F5h	shadow pixels.
F7h	"filling area" color, for example used by the crosshairs for the area which will change ist color depending upon th ereachability of th etarget.
F9h	2nd shadow. Not yet really revealed for what the game it uses.
C0h -> C6h:	these only come across in the soldier file, some sort of a color-by-numbers chart (mask) for the exe to colorize the sprites in at runtime.
EDh	end of line, means all pixels remaining for this line are transparent.

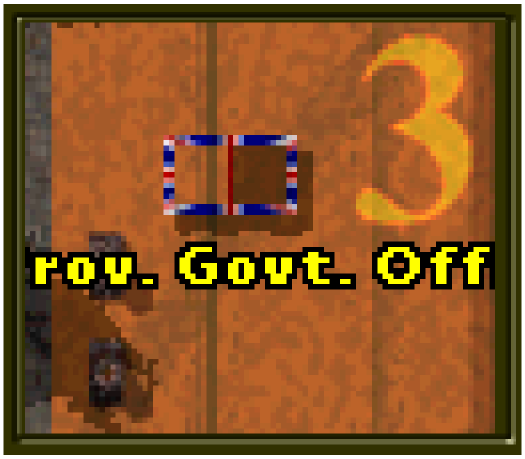
CPL_FILTH used the following example: 00 06 FF 02 12 34 56 78 F5 01 ED would mean 6 white pixels, followed by two pixels with the color values 0x1234 and 0x5678 respectively, a shadow pixel and white pixels until the end of the line.

No pixel data is given for lines that are indicated as fully transparent (FFFFh) in the "line offset table". All pixel data runs top to bottom, left to right. In CC2 and CC3 all pixel data are encoded as 16-bit short integer. And you must obey: the sequence "special code byte" followed by "number of pixels" is not changed when the byte format of the entire file is changed into Little Endian for CC4/CC5/CCM/RtB files.

In CC1 you will only encounter the special codes 00h, FFh, F5h and EDh. In CC1 the pixel data are encoded as 8-bit bytes. These bytes are index numbers pointing to an entry in one of CC1's color look-up tables (CLUTs).



An example for using the "shadow" pixel area (green), the "filling area" pixels (pink), the "2nd shadow" pixels (yellow) and the "tranparent" pixels (white): the resulting effects of "filling area" and "transparent" are the same as it is for both shadow pixel types.



CC4, CC5, CCM and RtB: "sprite_entry" format has changed:

```
// header: 12 bytes
short sprite_width      // 2 bytes
short sprite_height
short hotspot_X
short hotspot_Y
long  sprite_data_size  // 4 bytes: number of bytes
                        //          of sprite data to follow
                        // (= size_of_offsetable + size_of_pixeldata)
// line offset table
short line_offset       // 2 bytes for each line of the graphic.
short line_offset       // offset is counted from start of pixeldata
short line_offset       // that means: first entry for a line containing
short line_offset       // pixels must be 0000h.
...                     // Terrain files only: no special meaning of
                        //                     value FFFFh anymore!
// pixeldata
data                    // with special run length encoding
...
// terminator byte
byte  zero_byte         // Terrain files only: a terminating zero byte!
```

The special encoding for the "pixeldata" is nearly the same as in CC2/CC3, but in the "Terrain" files now all lines can be reached via the "line offset table" and all lines contain definitions for all pixels (even for the last transparent pixels at the end in every line). **The "Soldier"/"SoldierB" files have the same "pixeldata" encoding like in CC2/CC3 except for the byte order:**

00h	transparent pixels.
FFh	color pixels, this is followed by both the length and the color values for the pixels.
F5h	shadow pixels.
F7h	"filling area" color, for example used by the crosshairs for the area which will change ist color depending upon th ereachability of th etarget.
F9h	NO LONGER USED in the "Terrain" files.
C0h -> C6h:	these only come across in the soldier file, probably sort of a color-by-numbers chart for the exe to color the sprites in.
EDh	end of line.

In CC4 and newer all pixel data are encoded as 16-bit short integer, too. And you must obey: the sequence "special code byte" followed by "number of pixels" is not changed even now, where the byte format of the entire file is changed into Little Endian for this CC4/CC5/RtB files. The main difference (which will have an effect upon encoding/decoding programs) is that every entry is now terminated by a seperate zero byte ("Terrain" files only), except for the"Soldier"/"SoldierB" files! Decoding the entries requires exact respecting of the size of the entry as coded in the "sprite_data_size" value.

Known bugs inside the original files

There are some graphical bugs inside the original "Terrain" files: in the CC2 file the sprite #61 has 8 bytes too much (in one pixel line), CC3's Terrain file contains 206 bytes too much, and the CC4/CC5/CCM/RtB "Terrain" file also has a minor graphical bug inside. If you extract the contents of the CC4/CC5/CCM/RtB "Terrain" file and rebuild it with my tool, you will get a 10 bytes larger file, but it will work correctly with the original program. Investigations in late 2005 revealed that in the "newly dug trench" sprites (for example in sprite #450, line 45) some white pixels are encoded as color pixels with the 16-bit color value 7FFFh instead coding them as transparent pixels. This saves some bytes.

"Static animation section" = "Sequences"

This section always starts with the short integer indicator 1002 (decimal). Atomic called the entries within this section "Sequences". The reason why I call it "static" is because most of the animation sequences in here are defined for non-moving objects on the map (trees, VL-flags) and CC4 and newer really dont use the sequences defined here for animations anymore. This (and the next) section contains only lists (sequences) of sprite numbers referring to the graphical sprites of the "sprites section". The format of each entry "staticseq_entry“ is:

```
// no header
short number_of_sprites // 2 bytes, how many frames are in the sequence
short style_indicator   // 2 bytes of style/type purpose
short unknown_1         // 2 bytes, always zero in CC2
short unknown_2         // 2 bytes, always zero in CC2
data                    // 2 bytes for each number in the sequence
                        //     data size = number_of_sprites * 2
```

The meaning of the 2 bytes of the "style_indicator" value and the 4 following bytes are still not completely clear to me. Here is a list of values I have found:

1100h: regular sprite sequence in CC2/CC3
1200h: regular sprite sequence in CC2/CC3
2100h: not used animations in the CC2 file "Explode"
2200h: disabled entry in CC2/CC3 (?)
2220h: disabled entry in CC2 (?)
0001h: regular entry in CC4/CC5/RtB (perhaps meaning: show only 1 sprite ??)
FFF2h: always used in CC4-RtB *.spr files

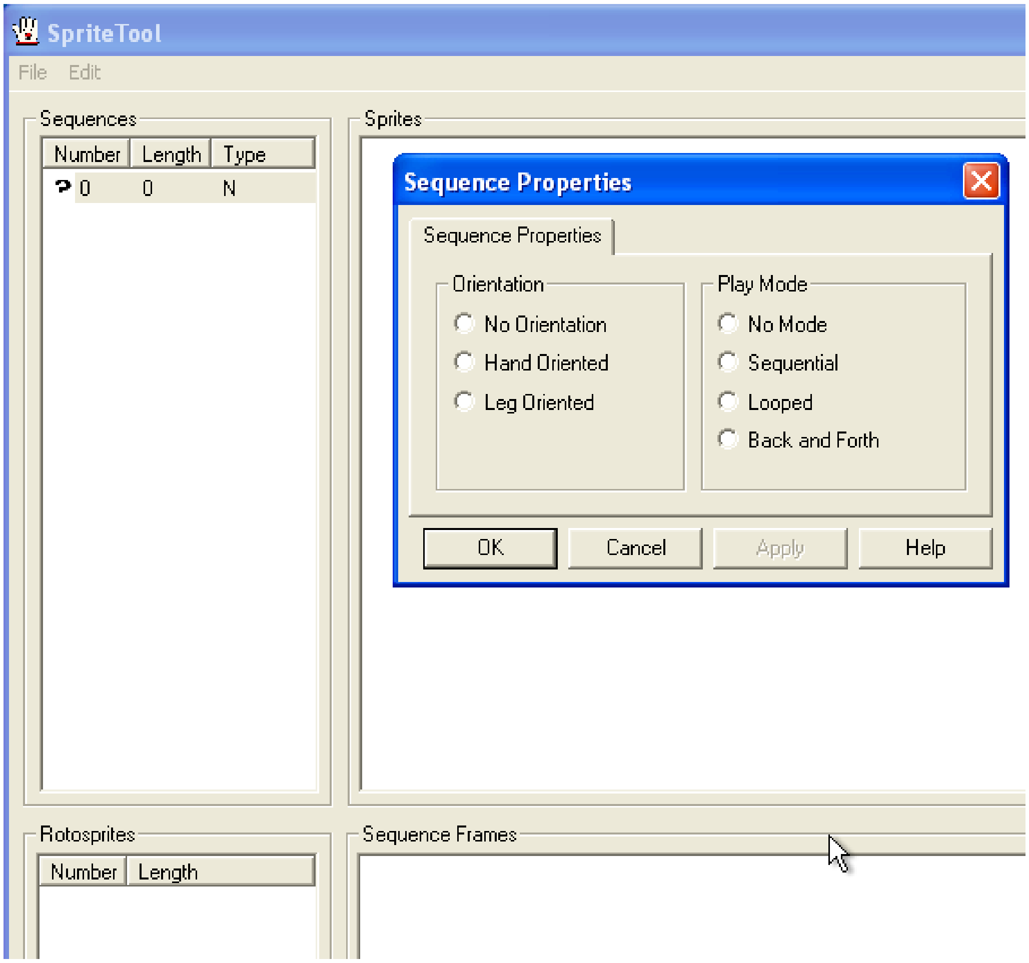
In CC5-"Terrain" file the value of "unknown_1" is always 8C2Dh, the value of "unknown_2" is always 0042h. No idea what it will mean. Changing the values of "unknown_1" and "unknown_2" seems to have no effect during gameplay. In all CC4/CC5/CCM/RtB *.spr files the value of "unknown_1" is always F51Ch, the value of "unknown_2" is always 0068h.

Changing the values of "style_indicator" does have effect, at least for the "Terrain"-file of CC2. I made a test with modified CC2-"Terrain"-files in 2010, changing the "style_indicator" of the British VL-flag animation sequence, and got the following result:

1000h: not supported = game crashes when sequence reaches it’s end.
1100h: "Sequential" = do the animation one time, stopping at it’s end,
1200h: "Looped" = perform the animation continously,
1300h: "Back and Forth" = perform the animation and loop back at it’s end.
1400h: not supported = game crashes when sequence reaches it’s end.
2100h: not supported = game crashes when loading the file "Terrain".
2200h: not supported = game crashes when loading the file "Terrain".

I believe that the value "1000h" might be supported as "no animation" in other sprite sequences, but this was not further investigated by me.

Looking at the "style_indicator" as a whole, I believe that only the first byte of it is used in CC2 and CC3. This first byte can be splitted into two nibbles: the four higher bits might have a special meaning, and the four lower bits might have a different meaning. Look at the following screenshot I got from a SpriteTool running on a WinPC (coded in 1998, updated the last time in 2005): this screenshot shows the possible "Properties" of a "Sequence":



The higher nibble might refer to "Orientation", and the lower nibble of the first byte might refer to "Play Mode". Three of the four possible "Play Mode" values can be set in CC2's "Terrain"-file by modding the "style_indicator" as shown above. I have no idea how this concept was organized later on in CC4/CC5-"Terrain"-files. The EXE-engines of these games generate no animations at all.

"Direction-oriented animation section" = "Rotosprites"

This section always starts with the short integer indicator 1003 (decimal). It contains "orientation" sequences to store individual references to sprites for each direction (8 directions or a multiple of 8). Atomic was calling these sequences "Rotosprites". I will refer to them as "Direction-oriented animations", although they are no animations at all.

An example: the CC2 file "Explode" contains 7 "Rotosprite"-sequences for gun muzzle fire, each sequence contains 8 sprites, one for every direction. So the graphics of each sequence will not be displayed "in a sequence", but the sequence will be treated by the game like an array of reference numbers. The format of each entry "directseq_entry" is:

```
// no header
short number_of_sprites // 2 bytes, how many numbers are in the sequence
short style_indicator   // 2 bytes of style/type purpose
short unknown_1         // 2 bytes, always zero in CC2 / CC3
data                    // 2 bytes for each number in the sequence
                        //     data size = number_of_sprites * 2
```

In CC2 the "style_indicator" and "unknown_1" is always 0000h. In CC5-"Terrain" file the "style_indicator" is 0001h and "unknown_1" is 8C2Dh. In all cases it looks like that both values have no effect on the gameplay. In all CC4/CC5/CCM/RtB *.spr files the value of "style_indicator" is always FFF2h, the value of "unknown_1" is always F51Ch.

CC1 sprites

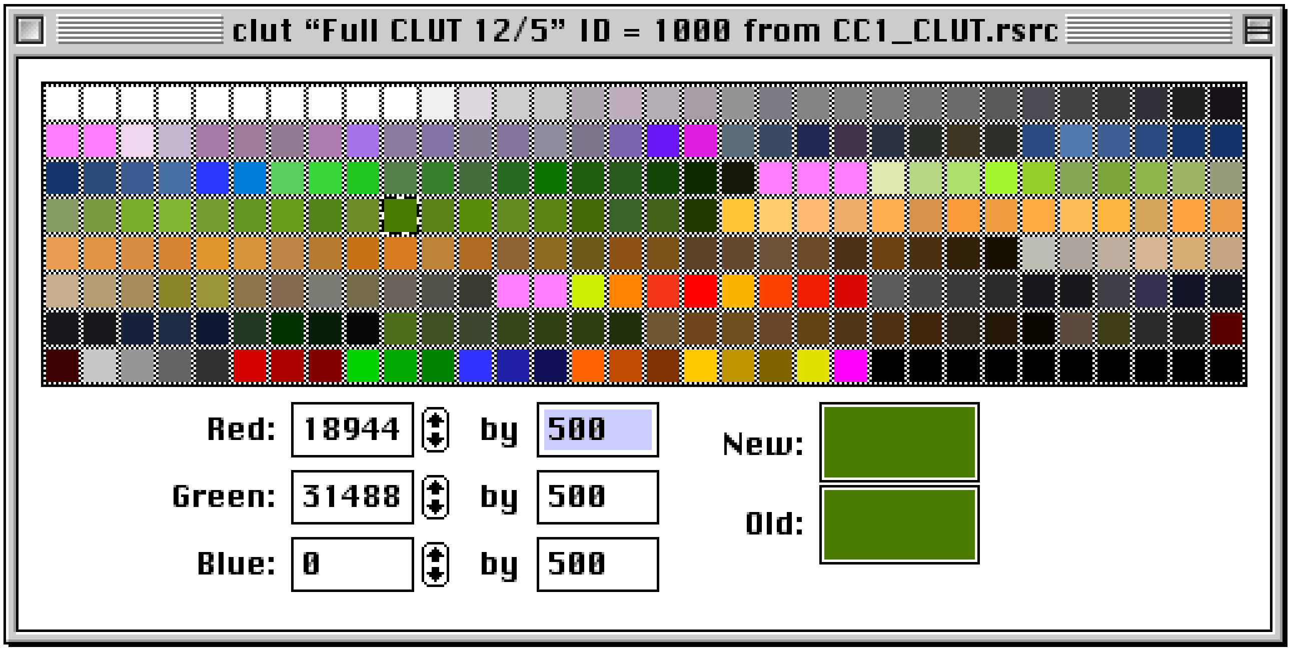
In CC1 the pixel datas are encoded as 8-bit bytes, since CC2 the pixel datas are encoded as 16-bit short integers. Sorry to say that CC1 sprite files have the same header like CC2 sprite files. If you look only at the header, you cannot determine if it is a CC1 or CC2 sprite file. The only key is to check if the sprite_data_size entry fits to the detected number of bytes when reading the data contents as 16-bit pixel datas. If the number of bytes read exceeds the file size or the sprite_data_size entry, you are probably reading a CC1 file.

As stated already, in CC1 the pixel datas are encoded as 8-bit bytes. These bytes are index numbers pointing to an entry in one of CC1's color look-up tables (CLUTs). The MacOS version of CC1 contains 6 CLUTs, numbered from 1000 to 1005. The CLUT with index number 1000 is named "Full CLUT 12/5" and this is the one used by the sprites. Here is the list of CC1's CLUTs, their names indicate their intended use I think:

- 1000 = "Full CLUT 12/5"
- 1001 = "MS Logo CLUT"
- 1002 = "Atomic Logo CLUT"
- 1003 = "CC CLUT"
- 1004 = "Movie CLUT"
- 1005 = "AtomLogo CLUT"

Converting CC1 sprite datas into graphics files requires translating the 8-bit pixel data into truecolor values using the CLUT. But it is also possible to export the datas as 8-bit graphics, in this case the CLUT must be incorporated into the graphics file (TARGA file format is recommended in this case). The shadow indicating pixels (special code F5h) might then use a non-used color at the end of the CLUT. Transparency indicating pixels (special code 00h) might use a non-used (white) color at the beginning of the CLUT. Another way exporting 8-bit pixel sprite datas is to export them in two files: color-datas and masks.

CC1 vehicle sprite datas contain the vehicle texture (the vehicle image) together with the shadow definition in one graphic. The turreted CC1 vehicles have 3 direction-oriented animation sequences: one for the vehicle's hull (together with shadow), one for the turret and a third one for the turret's shadow.



clut “Full CLUT 12/5” ID = 1000 from CC1_CLUT.rsrc

Red: 18944 by 500

Green: 31488 by 500

Blue: 0 by 500

Picture: CC1’s CLUT 1000 (MacOS version). Highlighted is the main color of the Sherman tank (color #105).

A suitable routine to transfer such a CLUT into a picture is shown here (coded in RealBasic 5.2):

```
Sub ExportCLUTasPicture(ResourceID As Integer)
  const maxentries = 256
  const boxsize = 16

  // color look up table (CLUT) of CC1
  dim clut As String
  dim CC1red(maxentries) As Integer
  dim CC1green(maxentries) As Integer
  dim CC1blue(maxentries) As Integer
  dim myNumberOfEntries As Integer
  dim myPic As Picture
  dim i As Integer

  // the MacOS resource fork must be inside this program
  Dim rf as ResourceFork
  rf=App.ResourceFork

  clut = rf.GetResource("clut", ResourceID)  // get CLUT from "clut"-resource

  // the first 8 bytes of clut are the header
  // byte 7+8 contain last entry's index, encoded in BIG Endian
  myNumberOfEntries = (AscB(MidB(clut,7,1)) * 256) + AscB(MidB(clut,8,1)) + 1
  if (myNumberOfEntries > maxentries) then
    myNumberOfEntries = maxentries
  end

  // then follows the clut table: each entry is
  //     8 bytes long: index number (2 bytes in BIG endian, counted from 0),
  //     red, green, blue (each 2 bytes coded in LITTLE endian)

  for i = 0 to myNumberOfEntries - 1
   CC1red(i)=AscB(MidB(clut,(8*(i+1))+3,1))+(AscB(MidB(clut,(8*(i+1))+4,1))*256)
 CC1green(i)=AscB(MidB(clut,(8*(i+1))+5,1))+(AscB(MidB(clut,(8*(i+1))+6,1))*256)
```

```
  CC1blue(i)=AscB(MidB(clut,(8*(i+1))+7,1))+(AscB(MidB(clut,(8*(i+1))+8,1))*256)
  next


  // create picture as 32-bit truecolor
  if (myNumberOfEntries MOD 16) = 0 then
    myPic = NewPicture(boxsize*16, boxsize*(myNumberOfEntries \ 16), 32)
  else
    myPic = NewPicture(boxsize*16, boxsize*((myNumberOfEntries \ 16) + 1), 32)
  end

  if (myPic <> NIL) and (myPic.Graphics <> NIL) then
    // fill entire pic black
    myPic.Graphics.ForeColor = RGB(0,0,0)  // Black
    myPic.Graphics.FillRect(0, 0, myPic.Width, myPic.Height)
    // draw the color-boxes
    for i = 0 to myNumberOfEntries - 1
      // draw the box
      myPic.Graphics.ForeColor = RGB(CC1red(i), CC1green(i), CC1blue(i))
      myPic.Graphics.FillRect(((i MOD 16) * boxsize) + 1,
                              ((i \ 16) * boxsize) + 1,
                              boxsize - 2, boxsize - 2)
    next
    if ExportPicture(myPic) then
      MsgBox "Your picture was saved."
    else
      MsgBox "Your picture was not saved."
    end
  end
End Sub
```

Synopsis of the "Header" and "Directory"

Game	File	Byte order	Header-ID	Sprites	Static anim.	Direction anim.	ColorDepth? Directions?
CC1 (Win95)	**EXPLODE**	PC	IRPS	476	9	31	16
CC1 (Win95)	**AXRIFLE**	PC	IRPS	1240	54	155	8
CC1 (Win95)	**GIRIFLE**	PC	IRPS	1240	54	155	8
CC1 (Win95)	**SMOKE**	PC	IRPS	392	31	8	8
CC1 (Win95)	**TERRAIN**	PC	IRPS	181	100	1	16
CC1 (Win95)	**Vehicle Type 1 (turreted)**	PC	IRPS	48	6	3	16
	Vehicle Type 2	PC	IRPS	16	2	1	16
	Vehicle Type 3	PC	IRPS	16	1	1	16
CC1 (Mac)	**Explosion Sprites**	Mac	SPRI	476	9	31	16
CC1 (Mac)	**Germ Rifle Sprites**	Mac	SPRI	1240	54	155	8

Game	File	Byte order	Header-ID	Sprites	Static anim.	Direction anim.	ColorDepth? Directions?
CC1 (Mac)	**GI Rifle Sprites**	Mac	SPRI	1240	54	155	8
CC1 (Mac)	**Smoke Sprites**	Mac	SPRI	392	31	8	8
CC1 (Mac)	**Terrain Sprites**	Mac	SPRI	181	100	1	16
CC1 (Mac)	**Vehicle Type 1 (turreted)**	Mac	SPRI	48	6	3	16
	Vehicle Type 2	Mac	SPRI	16	2	1	16
	Vehicle Type 3	Mac	SPRI	16	1	1	16
CC2	**VehB###** **VehS###** **(turretless)**	Mac	SPRI	32	2	1	8
CC2	**VehB###** **VehS###** **(turreted)**	Mac	SPRI	64	3	2	8
CC2	**Explode**	Mac	SPRI	588	13	31	16
CC2	**Smoke**	Mac	SPRI	620	43	8	8
CC2 / CC3 / CoI	**Soldier**	Mac	SPRI	3088	127	387	0
CC2 / CC3 / CoI	**SoldierB**	Mac	SPRI	1992	127	250	0
CC2	**Terrain**	Mac	SPRI	257	78	1	16
CC3 / CoI	**Terrain**	Mac	SPRI	448	165	2	16
CC3 / CoI	***.nsd**	Mac	SPRI	64	1	1	64
CC3 / CoI	***.zsd**	Mac	SPRI	64	1	1	64
CC4	**Terrain**	PC	IRPS	468	176	2	63
CC5	**Terrain**	PC	IRPS	474	177	2	63
RtB	**Terrain**	PC	IRPS	503	179	2	63
CCM / CCMT	**Terrain**	PC	IRPS	500	177	2	63
CC:WaR / CC:TLD	**Terrain**	PC	IRPS	500	248	2	63
CC4/CC5/CCM/RtB	**Soldier**	PC	IRPS	3088	127	387	0
CC4/CC5/CCM/RtB	**SoldierB**	PC	IRPS	1992	127	250	0
CC4/CC5/CCM/RtB	***.spr**	PC	IRPS	64	1	1	63 or 64

Byte order "Mac" = Big Endian, "PC" = Little Endian.

Looking at the table above shows that the sprite files "Soldier" / "SoldierB" remained unchanged since CC2 except for the changing of the byte order. The new function of the file and a new sprite sequence in the CC3-"Terrain" file survived until today, only the byte order changed and new sprites and animation sequences were added at the end of the lists. The functionality of all other CC2 sprite files went into FX-files for visual effects (improving them from 16-bit to 24-bit datas with 8-bit-alpha channel).

And sorry to say: since CC3 the game will not display any animation at all even if you have defined a sequence larger than 1 entry in this section. But in CC2 you can animate the VL-flags (like the original game does) and in addition the trees in the "Terrain" file. Animating other objects out of this file will fail like in all newer game versions.

Synopsis of the "Static animation section"

Game	File	style_indicator	unknown_1	unknown_2
CC2	Terrain	varying	0000h	0000h
CC3 / CoI	Terrain	varying	0000h	0000h
CC4	Terrain	0001h	0000h	0000h
CC5	Terrain	0001h	8C2Dh	0042h
CCM / CCMT	Terrain	0001h	8C2Dh	0042h
RtB	Terrain	0001h	CDABh	BADCh
CC:WaR	Terrain	0001h 0001h 2100h	8C2Dh 8C2Dh 0000h	0042h 0000h 0000h
CC:TLD	Terrain	0001h 0001h 2100h	8C2Dh 8C2Dh 0000h	0042h 0000h 0000h

If you look into the "Static animation section" entries of the file "Terrain" of "The Road to Baghdad" and read the byte sequence, you will see the sequence "...ABCDDCAB..." for the unknown byte area: I think this shows clearly that these bytes are not used by the program, they are simply filled up with dummy datas.

STM file format (CC4-CC5-CCM-RtB and newer)

Used for the TerrainA.stm file in the CC versions since CC4. The whole file is encoded in Little Endian (PC/Intel like). The TerrainA.stm file contains nearly the same graphics like the Terrain file but in 24-bit RGB-color with 8-bit-alpha-channel. No animation sequences are defined here.

```
// header
char(4)     // 4 bytes ASCII, the string “ALPH” = sprites with alpha-channel
long  // 4 bytes, containing value “2”
long  // 4 bytes, number of sprites in this file

// directory with sprite definition entries
// each entry 20 bytes of the format:
long  // 4 bytes, HotSpot X
long  // 4 bytes, HotSpot Y
long  // 4 bytes, image width
long  // 4 bytes, image height
long  // 4 bytes, offset of pixel datas (from top of file)

// sprite graphics data
// 4 bytes per pixel: blue, green, red, alpha
// line orientation is top-to-bottom, pixel orientation is left-to-right
```

We must use Cpl_Filths algorithm of making 32-bit TARGAS after reading the CC values myBlue, myGreen, myRed, myAlpha:

```
if ( ( 8 * myBlue - 1 ) > 0 ) then
  myBlue = (8 * myBlue) -1
end
if ( ( 8 * myGreen - 1 ) > 0 ) then
  myGreen = (8 * myGreen) -1
end
if ( ( 8 * myRed - 1 ) > 0 ) then
  myRed = (8 * myRed) -1
end
if ( myAlpha <> 0 ) then
  myAlpha = (8 * myAlpha) -1
end
```

To write them to a valid 32-bit color-depth TARGA file, the sequence must be changed:

```
OutFile.WriteByte(myRed)
OutFile.WriteByte(myGreen)
OutFile.WriteByte(myBlue)
OutFile.WriteByte(myAlpha)
```

MAFI

closecombat2@claranet.de
http://closecombat2.fortunecity.com/
http://cc2revival.npage.de/
http://www.ftf.claranet.de/
http://www.closecombat2.claranet.de
http://www.afrika.claranet.de/
http://www.dieppe.claranet.de/
http://www.cc2.claranet.de/
http://www.mappa.claranet.de/
http://members.fortunecity.de/closecombat2/
http://www.geocities.com/cc2revival/