File: CC2Guide-TxtfMapLosRoof.PDF
Format: PDF
Date: January 1st, 2006
Author: Mafi; closecombat2@claranet.de; http://members.fortunecity.de/closecombat2/
Last update: May 1st, 2006

Close Combat series of games

Map File Formats

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

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 (only public release is CCM version 3.1), The Road to Baghdad released in January 2004 (abbreviated: RtB), an updated CCM released to the USMC in summer 2005, CCRAFRegt released to UK forces in Feb. 2006). 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 MICK "XE5" CONMY (http://users.intrepid.net/~mcconmy/)and GERRY SHAW "TIN TIN" (http://www.organicbit.com/closecombat/) for their CC2- and CC3-file format informations and to PEKKA SAASTAMOINEN aka "CPL_FILTH" for his help on the CC3 roof file format. Please look at his homepage for further development on CC2-CC3-CC4-CC5-RtB-tools: http://www.student.oulu.fi/%7Epsaastam/ .

Map related Files of the CC Series of Games

This text is a compilation of already published or long-time known facts about CC map file formats and other details. CC experts wont find anything new inside it. For all other CC modders (like me) it is intended as a library.

All games of the Close Combat series store their map datas and map graphical datas in several files. File formats differ slightly between CC2 and the later releases, also the file names and the storing location differs. First of all you must know what is stored in what kind of file:

- **map data file:** contains numerical description of the map's size and its terrain structure (terrain type and height). In CC3 and newer it is also stored here the name of the map and further map descriptions (general terrain type, display color in the user interface). The file format is plain ASCII text, with TAB-seperated columns and CR-delimited lines in CC2 respectively CR+LF-delimited lines in CC3 or newer.
- **map los file:** contains line-of-sight informations for the entire map area. Identical file format throughout the whole CC series.

- **background graphics file:** stores the entire background of the map in 16-bit uncompressed RGB-format similiar to TARGA-graphics format. The game will use this graphic as a battleground.
- **overview graphics file:** stores a shrinked overview of the background graphics in the same format as the background graphics file. The game might uses this graphic in the deployment screen and perhaps when zooming out during battle.
- **monitoring graphics file:** stores a drastically shrinked minimap of the background graphics in the same format as the background graphics file. The game uses this graphic during battle for monitoring where the actual screen cutout is located on the map and as a "map icon" during map selection.
- **roof file:** stores multiple pairs of exterior and interior views of buildings together with coordinates informations. The interior view is pasted over the background graphic when a soldier enters a building, and the exterior view is pasted over the background when all soldiers have left a building during battle.
- **bridge file:** stores the graphic of the blown bridge and can contain in addition a graphic of the repaired bridge (Bailey Bridge). This kind of file is only available in CC2. During gameplay the bridge of your map can be blown by Axis troops. CC2 will do this by issuing a massive explosion, pasting shellholes over the bridge. When the game continues, some or all of these shellholes might disappear, because CC2's amount of displayable shellholes is limited. So the makers of CC2 needed this additional graphic to make the bridge blow effect visible throughout the whole battle. Repaired bridge graphic will be only pasted over the background graphic when XXX Corps reaches the bridge in Operation/Campaign play. This type of file is nearly identical with a CC2 roof file.
- **BTD file/Scenario file:** these files store informations about victory locations, forces setup and map connections. These files cannot be edited with 5CC.

The graphic files in CC2 use the Macintosh/Motorola byte order BIG Endian, the graphic files of CC3 and newer releases use the Intel byte order LITTLE Endian. Except for the roof file their file formats are mainly identical. The roof file of CC3 is much more complex than it is in CC2. The following table shows the main differences:

Short Synopsis of CC Map related Files

	CC2	CC3 and newer
folder to store map data file	../Data/Maps	../Maps
folder to store los file	../Data/Maps	../Maps
folder to store graphics files	../Graphics/Maps	../Maps
name of map data file	Map###	*.txt
map data file format	ASCII, TAB-seperated, CR-delimited lines	ASCII, TAB-seperated, CR+LF-delimited lines
name of los file	Map###.los	*.los
name of background graphics file	BGMap###	*.bgm
name of overview graphics file	OVMap###	*.ovm
name of monitoring graphics file	MMMap###	*.mmm
name of roof graphics file	Roof###	*.rfm
name of bridge graphics file	Bridg###	not available
byte order of the graphics files	BIG Endian	LITTLE Endian

CC2 Filename conventions

CC2 map files must have a fixed format. The map data file must always reside in a ../Data/Maps folder and its name must always start with the string "Map" followed by three digits. The number represented by these digits can range hypothetically from 100 to 999. This number is the so-called Map-slot. The Map-slot must not be identical with the Battle-slot, but in most cases it is. The Scenario file contains the number of the map to be used for this battle. The LOS (line-of-sight) file and the graphic files follow the same restrictions like the map data file (their name must contain the Map-slot number at the end). See the table above. For more details look into my guide "CC2Guide-NewBattlesMaps_v6.pdf".

CC3 (or newer) Filename conventions

CC3/CC4/CC5/CCM/RtB map files can have an individual name. The different map types will be indicated by the filename extension (MS-DOS like: dot followed by three characters). The map data file has always the filename extension ".txt" (it is in fact a plain ASCII file).

CC3 map filenames are limited to a length of 8 characters (MS-DOS compatible). In CC4/CC5 the map's filenames are also limited to 8 characters, because CC4's/CC5's Index.mpi file (which contain the list of usable maps) allows only entries up to a length of 8 characters).

In CCM/RtB the Index.mpi file format changed. The Index.mpi file can now store map names with a length of up to 23 characters. The longest original CCM v3.1 map name entry in the Index.mpi is "Big_Wonderland" with 14 characters.

Map Size Limit Synopsis

Version	max. map size...		
	...in pixel	...in deployment tiles	...in elevation tiles
CC2 (MAC & PC)	2280x2280	19x19	57x57
CC3	2880x2880	24x24	72x72
CC4	3000x3000[1]	25x25	75x75
CC5	3840x3840[2]	32x32	96x96
RtB	4800x4800	40x40	120x120
CCM (standard map)	4800x4800	40x40	120x120
CCM (BIG map)	19200x4800	160x40	480x120

In Feb. 2006 Demiurg reported at CSO forum that he was able to use maps in CC3 up to a size of 2960x2960 pixels, but this size does not fit to the Mega-tile (deployment tile) grid of CC.

CCRAFRegt uses the same map sizes as CCM, and the same is to be expected for future CC versions.

[1] david_Michael reported in 2006 that max. CC4 map size is 2880x2880 pixels like in CC3!

[2] In a thread of Feb. 2006 at CSO forum Buck_Compton reported the possible map size of 3840x3840 pixels for CC5. Before his report we all assumed that 3600x3600 is the allowed maximum.

A suitable TARGA-File Format

CC2 map file formats were first revealed by Adam "The Man" D'Arcy. He discovered that the graphical contents of the background map files are 16-bit uncompressed pixel data in Big Endian byte order. Making it accessible to a graphic editor was simple: changing the 16 byte header by a valid TARGA header. A long time people supposed that the graphics in the CC2 files are flipped. But this occured because the first CC2-modders used the wrong TARGA header. So we must first discuss the TARGA file format a little bit:

```
// TARGA header for our CC purposes
// all integers are coded in Little Endian!
Byte    00h    // ID length, set it to zero, meaning: no ID
               // if it is 0, then header length = 18,
               // if it is 1 than header length = 19, etc.
Byte    00h    // Color Map Type, set it to zero,
               // meaning: no color map
Byte    02h    // Image Type, set it to 2,
               // meaning: uncompressed, true-color image
Byte    00h    // 5 bytes: Color Map Specification,
Byte    00h    // must be set to zero (no color map used)
Byte    00h    //
Byte    00h    //
Byte    00h    //
Short   0000h  // 2 bytes: X-origin of image, set it to zero,
               // meaning: keep lower left corner X
Short   0000h  // 2 bytes: Y-origin of image, set it to zero,
               // meaning: keep lower left corner Y
Short   xx     // 2 bytes: image width in pixels
Short   yy     // 2 bytes: image height in lines
Byte    10h    // Color Depth, set it to 16, meaning: 16-bits per pixel
Byte    20h    // Image Descriptor, set it to 32,
               // meaning: no alpha-channel, top-to-bottom ordering,
               // left-to-right ordering of the pixels
data           // data-size = width * height * 2 bytes,
               // pixel-integers coded always in Little Endian
```

The "Image Descriptor" byte is the one defining the image orientation. Enter here 20hex = 32. Such a TARGA header has a size of 18 bytes. Because there is no color map defined, the size of the datas following this header can be calculated "data-size = width * height * 2" bytes. This is the same size of CC's 16-bit graphic datas after stripping the header.

Because this TARGA-header stores the image width and height in 2-bytes short integers, the maximum size of an image is limited to 65535x65535 pixels. This will be also the maximum size of a CC background image to be converted into such a TARGA format by several modding tools (65535x65535 pixels = 546x546 CC deployment tiles; the largest CC maps known today are CCM's big maps with a size of 160x40 deployment tiles, I think this limitation to short integers will be no serious problem at the moment).

The next problem for the first CC2 modders was, that the pixel datas in the CC2 files are coded in Big Endian (Motorola style, MacOS-like). Graphic datas of CC3 or newer are coded in Little Endian (Intel-style, MS-DOS/Win-like). Of course it is much easier to convert a CC3 graphic into TARGA than a CC2 graphic.

BGMap###- / BGM-File Format

In CC2 these files have always a filename beginning with "BGMap" followed by three digits (possible range 100 – 999). In CC3 or newer these files have always the filename extension ".bgm".

File format of a CC2 background map file:

```
// BGMap### header of CC2
// all integers are coded in Big Endian!
Char(4)  MAPI   // 4 bytes: ID, always "MAPI",
                // but other values will cause no trouble
Short    0002h  // 2 bytes: unknown purpose, is always 2,
                // perhaps color depth descriptor
Short    0000h  // 2 bytes: unknown purpose, is always 0
Long     xxxx   // 4 bytes: image width in pixels
Long     yyyy   // 4 bytes: image height in lines
data            // data-size = width * height * 2 bytes,
                // pixel-integers coded always in Big Endian
```

File format of a CC3-or-newer background map file:

```
// *.bgm header of CC3/CC4/CC5/CCM/RtB
// all integers are coded in Little Endian!
Char(4)  MAPI   // 4 bytes: ID, always "MAPI",
                // but other values will cause no trouble
Long     iiii   // 4 bytes: data-size counted in bytes
Long     xxxx   // 4 bytes: image width in pixels
Long     yyyy   // 4 bytes: image height in lines
data            // data-size = width * height * 2 bytes,
                // pixel-integers coded always in Little Endian
```

The CC2 file is completely coded in Big Endian. CC3 or newer files are completely encoded in Little Endian. Both file types have the same 4 byte header ID = "MAPI". In CC3 this header ID is followed by the data's size counted in bytes. In CC2 the purpose of these fifth and sixth byte of the header is unknown, but their value interpreted as a Little Endian integer represents the value 0002hex. Such a small map in CC3 is impossible (would mean data size = 2 bytes = 1 pixel). So you can use the fifth and sixth byte of a BGMap### file to check if it is a CC2 map or not.

OVMap###- & MMMap###- / OVM- & MMM-File Format

In CC2 these files have always a filename beginning with "OVMap" / "MMMap" followed by three digits (possible range 100 – 999). In CC3 or newer these files have always the filename extension ".ovm" / ".mmm".

File format of a CC2 overview and monitoring map file:

```
// OVMap### / MMMap### header of CC2
// all integers are coded in Big Endian!
Byte(4)  00000000h // 4 bytes: ID, always set to zero bytes,
                   // but other values will cause no trouble
Long     iiii      // 4 bytes: data-size counted in bytes
Long     xxxx      // 4 bytes: image width in pixels
Long     yyyy      // 4 bytes: image height in lines
data               // data-size = width * height * 2 bytes,
                   // pixel-integers coded always in Big Endian
```

File format of a CC3-or-newer overview and monitoring map file:

```
// *.ovm / *.mmm header of CC3/CC4/CC5/CCM/RtB
// all integers are coded in Little Endian!
Byte(4)  00000000h // 4 bytes: ID, always set to zero bytes,
                   // but other values will cause no trouble
Long     iiii      // 4 bytes: data-size counted in bytes
Long     xxxx      // 4 bytes: image width in pixels
Long     yyyy      // 4 bytes: image height in lines
data               // data-size = width * height * 2 bytes,
                   // pixel-integers coded always in Little Endian
```

The CC2 file is completely coded in Big Endian. CC3 or newer files are completely encoded in Little Endian. Both file types have the same 4 byte header ID completely set to zero bytes. In both CC2 and CC3 this header ID is followed by the data's size counted in bytes. You can only try to determine if it is a Big Endian coded file by checking if the data-size read is a multiple of the image width you have read (or by checking for negative values).

Roof###- / RFM-File Format

Roof files are a collection of graphic pairs representing the exterior and interior view of buildings bundled with their coordinates. At runtime CC will paste the interior view over the background graphic when a soldier enters the building (= is inside the roof's coordinate-rectangle (CC2) or -polygon (CC3 or newer)). When the last soldier has left the building, CC will paste over the exterior view (and will not switch back to the original background graphic). CC2 and CC3 roof files have different header IDs, so they can be identified easily. CC2 roof files are completely coded in Big Endian and their filename starts always with "Roof". CC3-or-newer roof files are completely coded in Little Endian and have always the filename extension ".rfm".

File format of a CC2 roof file (as reported by Mick "xe5" Conmy):

```
// Roof### header of CC2
// all integers are coded in Big Endian!
// header size = (16 + (16 * roof-pairs)) bytes
Char(4)  ROOF    // 4 bytes: ID, always "ROOF"
Long     iiii    // 4 bytes: number of entries (number of roof-pairs)
Long     00000000h // 4 bytes of unknown purpose, always 0
Long     00000000h // 4 bytes of unknown purpose, always 0
for ( each roofpair )  // 16 bytes for each roof pair
         Short   // 2 bytes:x1 coordinate of roof images for this entry
         Short   // 2 bytes:x2 coordinate of roof images for this entry
         Short   // 2 bytes:y1 coordinate of roof images for this entry
         Short   // 2 bytes:y2 coordinate of roof images for this entry
         Long    // 4 bytes: offset of exterior image data from start of data
         Long    // 4 bytes: offset of interior image data from start of data
data             // standard CC format graphical data, 16-bit color-depth
                 // coded in Big Endian
roof0_exterior_data
roof0_interior_data
roof1_exterior_data
roof1_interior_data
roof2_exterior_data
roof2_interior_data
etc.
```

Other than the CC3 roof file format, CC2 uses offsets from top of the first graphics data entry. That means that the first offset value will be always 00000000hex.

Coordinates in CC2 roof files means: x1,y1 = upper left corner of the image's position on the map and x2,y2 = lower left corner of the image's position there. x2 – x1 = image width, and y2 – y1 = image height. These coordinates for the images in the header entries are counted from (0, 0) = upper left corner of the map.

CC2 maps without roofs have the roof file omitted, in other words: the roof file is optional in CC2.

File format of a CC3 roof file (as reported by "Cpl_Filth"):

```
// *.rfm header of CC3/CC4/CC5/CCM/RtB
// all integers are coded in Big Endian!
// header size = (16 + (132 * roof-pairs)) bytes
Long     E002F001h // 4 bytes: ID, always E002F001hex
Long     iiii   // 4 bytes: number of entries (number of roof-pairs)
Long     00000000h // 4 bytes of unknown purpose, always 0
Long     00000000h // 4 bytes of unknown purpose, always 0
for ( each roofpair ) // 132 bytes for each roof pair
      Long 00000002h    // number of roof images for this entry, usually 2
      Long iiii   // number_of_vertices, maximum is 12
      // vertices table, fixed size of 12 entries
      for ( i = 0 ; i < number_of_vertices ; i++ )
         {
         Long vertex.i.x-coordinate
         Long vertex.i.y-coordinate
         }
      for ( i = number of vertices ; i < 12 ; i++ )
         {
         // CDCDCDCDhex indicates this vertex is not used
         Long CDCDCDCDh
         Long CDCDCDCDh
         }
      // coordinates of surrounding rectangle
      Long iiii   // 2*roof_width = size of pixel line in bytes
      Long xxxx // roof_top_left_x-coordinate
      Long yyyy // roof_top_left_y-coordinate
      Long xxxx // roof_bottom_right_x-coordinate
      Long yyyy // roof_bottom_right_y-coordinate
      Long eeee // offset_of_exterior_data from beginning of file
      Long iiii // offset_of_interior_data // ditto
data            // standard CC format graphical data, 16-bit color-depth
                // coded in Little Endian
roof0_exterior_data
roof0_interior_data
roof1_exterior_data
roof1_interior_data
roof2_exterior_data
roof2_interior_data
etc.
```

Other than the CC2 roof file format, CC3 roof files use offsets from top of file.

The "`2*roof_width`" value determines the width of the stored graphics. "`2*roof_width`" is the size of a pixel line counted in bytes. To get the width of the images, you must divide this value by 2. The visible width is equal or less than this value. The visible width can be calculated by "`roof_bottom_right_x-coordinate`" - "`roof_top_left_x-coordinate`". The original roof files of CC3 contain some black colored pixels at the right side of the visible width. Roof file graphics of CC3 or newer contain white colored pixels outside the area defined by the vertices table. This table is of fixed size (12 entries). These white pixels are not pasted over the background graphic at runtime.

In original CC4 the Schneeifel-East map has no roof file. So I think that in CC3 and CC4 a roof file is optional like in CC2. Two original CC5 maps, five original CCM maps and three original RoadToBaghdad maps have roof files with no roof entries (CC5: Branvll.rfm and Hill140.rfm; RtB: Marsh.rfm, Snow1.rfm and Wadi.rfm). These files have the "number of roof entries" set to zero. The header of these empty roof files is followed by 16 dummy bytes with value CDhex.

Bridg###-File Format

CC2's Bridg### files can contain one or two images (blown bridge only, or blown bridge and repaired bridge). The blown bridge image will be pasted over the map after the Axis-user (or Axis-AI) has pressed the "Blow bridge" button. The second (repaired bridge) image will be pasted over the background graphic at runtime only in operation / campaign play when XXX Corps has reached the bridge. File format of Bridg### files:

```
// Bridg### header of CC2
// datas encoded in Big Endian = Motorola style, MacOS compatible
Char(4)  BRDG    // 4 bytes: ID, always "BRDG"
Long     iiii    // 4 bytes: number of images in this Bridge-file
Long     00000000h // unknown, always zero
Long     00000000h // unknown, always zero
for ( each bridge image )  // 16 bytes for each bridge image
       Short     // two bytes: upper-left horizontal position on map
       Short     // two bytes: lower-right horizontal position on map
       Short     // two bytes: upper-left vertical position on map
       Short     // two bytes: lower-right vertical position on map
       Long      // offset of image data from top of data
       Long      // index of this image (1 or 2)
//followed by the pixel datas (16 bit) for each image, Big Endian
data1    // required (one image at least is necessary)
data2    // optional
```

The coordinates for the images in the header entries **are counted from (0, 0) = upper left corner of the map**. If you take the coordinates of your "blown clipouts" to create your Bridge-file, ensure that your graphics editor program uses also (0,0)-based corrdinates. You are free in the calculation of the coordinates for the upper-left position of the image(s) on the map as long as the image can be entirely pasted into the map during runtime.

The lower-right coordinates for the the header entries must be calculated as "lower-right-horizontal-pos = upper-left-horizontal-pos + width-of-image" and "lower-right-vertical-pos = upper-left-vertical-pos + height-of-image" (or much more mathematical: x2 = x1 + width; y2 = y1 + height). If the lower-right coordinates are less than the upper-left coordinates, then no image will be pasted in during runtime when pressing the "Blow Bridge"-button (the same as for the Roof###-files).

In case of the Bridg### files the coordinates and size of the blown bridge image can be different from the coordinates and size of the repaired bridge image.

CC2 maps which dont need blown bridge image(s) have the Bridg### file omitted, in other words: the Bridg### file is optional in CC2. Bridg### files are special files for CC2 only. For more details please read my guide "CC2Guide-Bridg-files_v6.pdf".

Map###- / TXT-Data-File Format

The terrain type and elevation encoding is done in plain ASCII files for all CC versions since CC2. These map data files have filenames starting with "Map###" (for CC2) or the filename extension ".txt" (CC3 or newer). They contain numerical description of the map's size and its terrain structure (terrain type and height). In CC3 and newer it is also stored here the name of the map and further map descriptions (general terrain type, display color in the user interface). The file format is plain ASCII text, with TAB-seperated columns and CR-delimited lines in CC2 respectively CR+LF-delimited lines in CC3 or newer. These files can be edited perfectly with MS-Excel.

File format of a CC2 map data file:

- first line: ID, always the string "41",
- second line: width (in elevation tiles) followed by the string "X Max",
- third line: height (in elevation tiles) followed by the string "Y Max",
- forth line: comment line starting with the string "Idx", followed by TAB + string "E0" + TAB ... + TAB + "E15" + TAB + "Elev",
- fifth line: data start indicator "&",
- data lines, number of data lines = width * height (counted in elevation tiles)
- last line: data end indicator "#",
- perhaps additional data lines (bailey bridge datas),
- perhaps second data end indicator "#".

Every data line starts with the index number of the elevation tile defined by this line. The index numbering starts at 0. The elevation tiles are counted sequentally. The additional data lines are repeating these index number to indicate which elevation tiles they will replace when the bridge is repaired (by XXX Corps) in CC2.

The index number will be followed by 16 TAB seperated terrain numbers (possible range 0 – 255, as defined in th ebase file "Elements") and in the 18th column TAB seperated the elevation (values greater than 0).

File format of a CC3/CC4/CC5/CCM/RtB map data file:

- first line: ID, always the string "10",
- second line: can contain the map's name,
- third line: width (in elevation tiles) followed by the string "X Max",
- forth line: height (in elevation tiles) followed by the string "Y Max",
- fifth line: map terrain type descriptor,
- sixth line: map name color descriptor,
- seventh line: comment line starting with the string "Idx", followed by TAB + string "E0" + TAB ... + TAB + "E15" + TAB + "Elev",
- eight line: data start indicator "&",
- data lines, number of data lines = width * height (counted in elevation tiles)
- last line: data end indicator "#",

The data line format is similiar to the one of CC2, except for the fact that there are 16 elevation definitions per line (in CC2: only one elevation value).

CC3 introduced three additional lines: now the second line can contain the map's full (or abbreviated) name. But in original CC3 most maps have here an empty entry. The 5th line contains a general map terrain type descriptor, a number ranging from 0 to 4:

- 0 = standard map,
- 1 = light mud,
- 2 = deep mud,
- 3 = light snow,
- 4 = deep snow.

If you change this number from a zero to a 3 or 4 the units in CC3 will be in winter camouflage gear. The 6th line directly below this one indicates whether the title of the map (in the scenario editor) will be printed in white or green (CC3 only!, as reported by "KWP").[3] I will refer to this line as a "map name color descriptor":

- 0 = white,
- 1 = green.

The original CC4 maps have here always the general map terrain type descriptor set to 3 = light snow and the value zero for the map name color descriptor. Only two original CC4 maps have a map's name entry (full length: Hotton and Marche).

In CC5 maps the 5th and 6th lines will always contain the value zero. The map's name entry contains in several cases the name abbreviated or misspelled. As reported by Senior Drill at the CSO forum in early March 2006 the "...use of the number in the fifth line of a map .txt header has been turned off or disabled in CC5".

In CCM v3.1 and RtB nearly all maps will contain in the 5th and 6th lines the value zero like in CC5, even if they are explicit snow terrain maps (CCM v3.1: Montanyan#; RtB: Snow#). But different to CC5 some CCM and RtB maps have a map name color descriptor 1 = green: the CCM maps Big_Urbania.txt, 29Palms2.txt and the RtB map Wadi.txt. The meaning of the 5th line value changed in CMM (perhaps valid for CCM release of summer 2005 only):[4]

- 0 = standard map,
- 1 = mud,
- 2 = light snow,
- 3 = deep snow,
- 4 = desert.

In CCM v3.1 and RtB the map's name entry contains always the full map name.

---

[3] This is what "KWP" wrote at CSO forum in Nov. 2005: "...Look at the number directly UNDER the Y max value. This will show a number in the range 0 - 4. 0 = standard map. 1 = light mud. 2 = deep mud. 3 = light snow. 4 = deep snow. If you change this number from a zero to a 3 or 4 the units will be in winter camouflage gear. Also, the number directly below this one indicates whether the title of the map (in the scenario editor) will be printed in white or green. 0 = white. 1 = green."

[4] This is what Senior Drill wrote at CSO forum in March 2006: "The use of the number in the fifth line of a map .txt header has been turned off or disabled in CC5. It was reenabled in CCM. In CCRAF, one further step was taken, though in the easiest manner possible, and not necessarily the best manner. ... In CCM, the definition of the map codes were changed in the code to be "normal", mud, light snow, heavy snow and desert. So while vehicles could have all three camoflague patterns, soldiers still were limited to just the two of woodland and snow."

How to get winter / desert camouflage for your troops

In CC3 you can change the map's general map terrain type descriptor as discussed above. For CCM v3.1 and RtB the map's name defines which camo you will get for your vehicles:

Camouflage type you want to get	CCM: filename must start with	RtB: filename must start with
green / grey	default for all other maps	not available
desert / sand	29Palms	default for all other maps
snow	Montanyan	Snow

These filenames can be followed by a digit (examples: CCM: 29Palms1.txt, 29Palms2.txt, 29Palms3.txt; RtB: Snow1.txt, Snow2.txt). The filename templates are encoded inside the executables (CCMarines3.1.exe, RoadToBaghdad.exe) and can be patched (but the length is fixed).

MPI-File Format

This file format is used by CC4/CC5/CCM/RtB for the file Index.mpi. This file controls which maps are available to the game. You can have more maps inside your ../Maps folder, but only those listed in the Index.mpi file will be available for play. The file Index.mpi contains the filenames without the filename extensions.

Two different formats:

CC4/CC5:

```
// Index.mpi file of CC4/CC5
// no header
for every map-entry
      char(8) // ASCII of the map's name, padded with zero bytes
      byte    0    // terminating zero byte
next map-entry
```

CC4 has 43 maps listed here, CC5 has 44 maps listed here.

CCM/RtB:

```
// Index.mpi file of CCM/RtB
// no header
for every map-entry
      char(23)// ASCII of the map's name, padded with zero bytes
      byte    0    // terminating zero byte
next map-entry
```

CCM and RtB have room for 44 maps reserved here, unused names are completely filled with zero bytes. CCM v3.1 lists 20 map names, RtB has 14 maps listed here. Map limit of CCM/RtB seems to be 20 maps, although there is room for 44 maps in the Index.mpi.

LOS-File Format

The LOS (line-of-sight) files (filename extension ".los") have an identical file format throughout the CC series (CC2 to RtB). LOS files have no header. First description of their format was published by Vince Viaud in June 1998. His tool CClos.exe is the gold-standard for LOS-generating.

The Line-Of-Sight concept of the CC series of games:

When setting up the troops for a battle the AI must avoid to place them in open area without cover. The unit placement for human player and AI is limited to the deployment tiles defined in the corresponding files (in CC2: Scenario files in the ../Data/Battles/#### folders; in CCM/RtB the files in the folder ../Games/Battles). Each deployment tile (= Mega tile = LOS tile) is a square of 120x120 pixels. Their coordinates are counted from left to right/top to bottom starting with 0,0 in the upper left edge of the map.

The LOS file contain informations about the visibility from one deployment tile to every other deployment tile of the map. This information is stored binary: LOS true or false (1 or 0). A correct calculated LOS file should contain as a minimum LOS=true from every deployment tile to itself. But thats not all. The LOS file contain this visibility for 4 different situations:

1. soldier viewing soldier (SvS),
2. soldier viewing vehicle (SvV),
3. vehicle viewing soldier (VvS),
4. vehicle viewing vehicle (VvV).

Every "LOS situation" will be stored as a seperate bitfield. CPL_FILTH refered to this "LOS situation" as a "row", I think we can also call it a "LOS layer". CPL_FILTH is the one who described this best in the header file of his CClos.exe version:

"... each row is a bitfield (0 can't see, 1 can see) from that Mega tile to every other Mega tile,
the 4 rows correspond to

- soldier viewing soldier,
- soldier viewing vehicle,
- vehicle viewing soldier,
- vehicle viewing vehicle,

vehicle is used for standing soldier. LOS files are only used for strategic AI and therefore have no effect on 2 player games. A file with all zeros will be valid but the AI wouldn't play worth a shit."

When storing the bitfield it will be padded up with zero bits to get a multiple of 8 bits (to store it in bytes). There will be 4 bitfields for every deployment tile. The bitfields of the upper-left deployment tile will be stored first in the file, then follows the second tile etc. The storing order is like the coordinates from left to right, top to bottom. A short piece of source code in Pseudo-PASCAL (intended for loading a LOS file) will illustrate this:

```
Var     InputBytesPerRow : Integer;                          (* size of a bitfield (in bytes) *)
        InputMegaTilesWidth, InputMegaTilesHeight : Integer; (* will get their values from map *)
        t, row : Integer;                                        (* loop variables *)
        iStream : File;                                          (* the LOS file will read *)
        rowRead : array[0..MaxInt] of Byte;                  (* for storing an entire bitfield *)
        actualInputByte, actualInputBit : Byte;                  (* to retrieve single bits *)
        LOSMegaTile : array[0..MaxInt] of Record
                                  Views : array[0..MaxInt] of Record
                                          SoldierViewingSoldier : Boolean;
                                          SoldierViewingVehicle : Boolean;
```

```
                                        VehicleViewingSoldier : Boolean;
                                        VehicleViewingVehicle : Boolean;
                                    end;
                                end;

InputBytesPerRow := (InputMegaTilesWidth * InputMegaTilesHeight) DIV 8;
if ((InputBytesPerRow MOD 8) = 0) then InputBytesPerRow = InputBytesPerRow + 1;

for t := 0 to ((InputMegaTilesWidth * InputMegaTilesHeight) -1) do
begin      (* read all 4 rows of each mega-tile of source file *)
        for row := 0 to 3 do
        begin
                if not iStream.EOF then
                begin     (* read one entire row of this MegaTile *)
                        rowRead = iStream.ReadBytesFromFile(InputBytesPerRow);
                        (* each row contains 1 bit for every mega tile *)
                        for i = 0 to ((InputMegaTilesWidth * InputMegaTilesHeight) -1) do
                        begin
                                actualinputbyte = rowRead[ (i DIV 8) ]  (* index starts at 0 *)
                                actualinputbit = BitWiseShiftRight( actualinputbyte, (i MOD 8) ) AND 1
                                case row of
                                0 : LOSMegaTiles[ t ].Views[ i ].SoldierViewingSoldier := (actualinputbit = 1)
                                1 : LOS.MegaTiles[ t ].Views[ i ].SoldierViewingVehicle := (actualinputbit = 1)
                                2 : LOS.MegaTiles[ t ].Views[ i ].VehicleViewingSoldier := (actualinputbit = 1)
                                3 : LOS.MegaTiles[ t ].Views[ i ].VehicleViewingVehicle := (actualinputbit = 1)
                                end; (* case *)
                        end; (* i *)
                end; (* EOF checking *)
        end;  (* row *)
end; (* InputMegaTile *)
```

You can calculate the size of the LOS file to check if it is will fit to your map:

number_of_tiles = number_of_depl_tiles_horizontal * number_of_depl_tiles_vertical (in bits)
size_LOS_layer = (number_of_tiles DIV 8)
+ 1 if (number_of_tiles MOD 8) <> 0) (in bytes)
size_LOS_entry_for_each_tile = 4 * size_LOS_layer (in bytes)
size_of_LOS_file = number_of_tiles * size_LOS_entry_for_each_tile

Example: a map of 10 * 3 deployment tiles will have a total of 30 deployment tiles. Therefore you must have 30 bits for every layer, rounded up to the next multiple of 8 will be 32 bits. 32 bits will be strored in 4 bytes. You will need these 4 bytes for each LOS layer. So you will need 4 * 4 = 16 bytes for each deployment tile to store all 4 LOS-layers. Total size of the LOS file will be 30 * 16 = 540 bytes.

Custom LOS files are a little bit larger, because CClos.exe is appending some generating-info bytes.

LOS-file format of CC2/CC3/CC4/CC5/CCM/RtB:

```
// LOS-file of CC2..RtB
// no header
for every deployment tile
      data0   // SvS bitfield, padded with zero bits
      data1   // SvV bitfield, padded with zero bits
      data2   // VvS bitfield, padded with zero bits
      data3   // VvV bitfield, padded with zero bits
next deployment tile
```

TXTF-File Format

Although this file format is not used for map related files, I will discuss it here because it is nearly identical to the BGMap###-/BGM-File format. In CC2 these files are used to store vehicle and flamethrower graphics. They are stored in the CC2 folder ../Graphics/Textures and their filename always starts with "Txtr" followed by three digits ranging from "000" to "173". These files are completely encoded in Big Endian. White pixels (indicating transparency) are coded with the value 7FFFhex.

File format of a CC2 txtf-file:

```
// texture header of CC2
// all integers are coded in Big Endian!
Char(4)  txtf   // 4 bytes: ID, always "txtf",
Short    0001h  // 2 bytes: unknown purpose, is always 1,
                // perhaps color depth descriptor
Short    0000h  // 2 bytes: unknown purpose, is always 0
Long     xxxx   // 4 bytes: image width in pixels
Long     yyyy   // 4 bytes: image height in lines
data            // data-size = width * height * 2 bytes,
                // pixel-integers coded always in Big Endian
```

File format of a CC3 txtf-file, as it is used by the files inside the FT.zfx, Guns.zfx and Wrecks.zfx archives:

```
// texture header of CC3 guns and wrecks
// all integers are coded in Little Endian!
Char(4)  txtf   // 4 bytes: ID, always "txtf",
Short    0000h  // 2 bytes: unknown purpose, is always 0
Short    0002h  // 2 bytes: unknown purpose, is always 2,
                // perhaps color depth descriptor
Long     xxxx   // 4 bytes: image width in pixels
Long     yyyy   // 4 bytes: image height in lines
Long     xxxx   // 4 bytes: X hotspot for shadow/turret rotation
Long     yyyy   // 4 bytes: Y hotspot for shadow/turret rotation
data            // data-size = width * height * 2 bytes,
                // pixel-integers coded always in Little Endian
                // padded up with zero bytes to a given file size
```

The file format description is NOT identical with the one you can find at GERRY SHAW "TIN TIN"'s site! He is describing a 26 byte long header. I'm sure the header is only 24 bytes long! The CC3 or newer files are completely encoded in Little Endian.

This CC3-txtf file format survived in CC4/CC5/CCM/RtB for special txtf-files inside the file "FT.azp". The files inside "FT.azp" have no filename extension. Contents of these FT/FL files: 1 small graphical texture for flamethrower animation (similiar to CC3's "FT.zfx" archive).

Mafi
closecombat2@claranet.de

http://www.geocities.com/cc2revival/
http://members.fortunecity.de/closecombat2/
http://www.closecombat2.claranet.de/
http://www.ftf.claranet.de/