##
BLP Spezifikation
von : Maik Gollnick
##

###Einleitung###
Diese Spezifikation soll jedem helfen, der das BLP-Format von Blizzard, besser verstehen will. Ich fand viel zu wenig Informationen im Internet und nur wenige "Anleitungen"(meist nur kurz und Stichpunktartig) konnten mir helfen das BLP-Format zu verstehen. Meine Spezifikation, sowie jede die ich gelesen hab, wurde ohne die Hilfe von Blizzard erstellt und erarbeitet.

Quellen:

MagosBlpFormat.txt von **Magnus Ostberg** (aka Magos)
http://magos.thejefffiles.com/War3ModelEditor/MagosBlpFormat.txt

Unbekannter Autor:
txt-Datei auf meiner Seite

"BLP FILE SPEC" von **BlacKDlcK**
txt-Datei auf meiner Seite

Unbekannter Autor:
txt-Datei auf meiner Seite

BLP0/1 Header

##
BLP0/1 Header (allgemein)
##

Der blp0/1 - header ist immer 156 bytes lang, d.h. 28 bytes Kennungs-daten, dann 64 bytes Mipmapoffset - Daten und 64 bytes Mipmapsize-Daten.

Wenn das blp mit jpeg - Daten kodiert vorliegt, dann folgen nach den 156 bytes weitere 4 bytes mit der jpeg - header - size. Danach folgt der eigentliche jpeg - header (weiteres dazu später).

Wenn das blp mit einer Farbpalette kodiert wurde, dann folgt diese direkt nach den 156 bytes.

VORSICHT :
Beim dekodieren einer blp - Datei sind die mipmapoffsets und die mipmapsizes oft falsch angeben. Meistens stimmt nur der erste mipmapoffset, alles andere ist falsch. Daher ist es wichtig sich beim Dekodieren und beim Kodieren sich seine eigenen Offsets bzw.

Sizes zu schreiben und sie im header zu hinterlassen (Beispiele dazu weiter unten).

BLP0 und BLP1:

Eine blp1 – Datei besteht nur aus einem File und endet jeweils mit *.blp. Im Gegensatz dazu hat eine blp0 – Datei ein File für den Header und (wenn vorhanden) für die Palette, dieses File endet mit *.blp. Danach folgen im schlimmsten Fall 1..n Files, die die Mipmap – Daten enthalten, diese enden jeweils mit *.b01 .. *.b0n.

Beispiel:(mit 3 Mipmaps)

example.blp - enthält 156 bytes Header + 1024 bytes Palette = Gesamt : 1180 bytes
example.b01 - enthält 1tes Mipmap, also (width * height) bytes
example.b02 - enthält 2tes Mipmap, also (width * height)/2 bytes
example.b03 - enthält 3tes Mipmap, also (width * height)/4 bytes

```
#############################################################################
# BLP0/1 Header (kurz)                                                      #
#############################################################################
```

4 Bytes (char):	BLP0/1
4 Bytes (int):	Kompression (0 für jpeg, 1 für palette)
4 Bytes (int):	Alpha (8 blp hat Alpha, ungleich 8 blp hat keins)
4 Bytes (int):	Breite des Bildes (width)
4 Bytes (int):	Höhe des Bildes (height)
4 Bytes (int):	Bildtyp (3, 4, 5)
4 Bytes (int):	Bilduntertyp (ist immer 1)
64 Bytes (int):	Mipmapoffsets (4 Bytes = 1 Mipmapoffset)
64 Bytes (int):	Mipmapsizes (4 Bytes = 1 Mipmapsize)

```
#############################################################################
# BLP0/1 Header (ausführlich)                                               #
#############################################################################
```

4 Bytes (char): BLP0 oder BLP1

Die ersten 4 bytes repräsentieren die Kennung einer blp - Datei. Es handelt sich dabei um die Zeichen "BLP0" oder “BLP1” als bytes. Um sie als chars zu bekommen maskiert man das jeweilige byte mit 0xff und erhält einen Integer im Wertebereich 0 - 255, diesen castet man auf char und erhält das entsprechende Zeichen.

4 Bytes (int): Kompression (0 für jpeg, 1 für palette)

Diese 4 bytes müssen als Integer eingelesen werden (näheres dazu weiter unter :D). Man bekommt dann entweder eine 0 oder eine 1 als Wert. Wenn man eine 0 bekommt dann sind die Bild - daten dieses blps als jpeg - Daten vorhanden. Bei einer 1 als Wert bestehen die Daten aus einer Farbpalette und entweder nur

aus einer folgenden Indexliste oder aus einer Indexliste und
einer Alphaliste.

4 Bytes (int): Alpha (8 blp hat Alpha, ungleich 8 blp hat keins)

Wieder 4 bytes die als Integer einzulesen sind. Bekommt man dabei
eine 8 als Wert, dann besitzt das blp einen Alphakanal. Alles
ungleich 8 bedeutet, dass kein Alphakanal vorhanden ist.

4 Bytes (int): Breite des Bildes (width)

Die Breite des Bildes wird hier in Pixel angeben.

4 Bytes (int): Höhe des Bildes (height)

Die Höhe des Bildes wird ebenfalls in Pixel angeben.

4 Bytes (int): Bildtyp (3, 4, 5)

Der Bildtyp hat nur Bedeutung wenn die Bilddaten in einer
Farbpalettenkompression vorhanden sind. Das heißt nicht, dass
die Werte nicht angeben sind, wenn es sich um ein jpeg handelt.
Sonder sie werden dann nur interpretiert oder eben nicht.

3 und 4 :
Die beiden Zahlen geben jeweils die gleiche Struktur an,
warum genau weiß ich nicht, vielleicht kommt es daher das
es schon so viele blp - Formate vorher gab und man sie
einfach nie geändert/angepasst hat. Also erhält man eine
3 oder 4 so folgt nach der Palette eine Indexliste der
Größe width*height (width = Breite des Bildes, height = Höhe
des Bildes) und direkt nach der Indexliste kommt eine
Alphaliste ebenfalls der Größe width*height. Die Werte
in der Indexliste verweisen auf die jeweilige Stelle in der
Palette (warum ? erkläre ich später). In der Alpha list stehen
die Alphawerte, diese ist auch vorhanden falls das blp gar
kein Alpha besitz, sie wird dann halt nich verwendet.

5 :
Hier folgt nach der Palette nur eine Indexliste mit der
selben Eigenschaft wie bei 3 und 4.

4 Bytes (int): Bilduntertyp (ist immer 1)

Keine Ahnung was diese 1 bedeutet, Blizzard benutz
sie vielleicht als Kennung oder zum dekodieren aber
genau weiß ich es auch nicht. WICHTIG : Sie muss immer
angeben werden, wie alle anderen Werte im header.

64 Bytes (16xint): Mipmapoffset

Ein blp kann höchstens 16 Mipmaps haben, der Offset gibt an ab wo die Bilddaten beginnen. Hier sind wirklich die Bilddaten gemeint und nicht etwa bei einer Palettenkodierung der Anfang der Palette, sonder erst danach.

Wie bestimmt man die Offsets selbst?

Also man nimmt die Größe des blp - headers, das sind ja immer 156 bytes, und rechnet bei

Palettenkodierung (Bildtyp 3 oder 4):

156 + 1024 = Offset 1tes Mipmap
156 + 1024 + width*height*2 = Offset 2tes Mipmap
156 + 1024 + (width/2)*(height/2)*2 = Offset 3tes Mipmap
156 + 1024 + (width/4)*(height/4)*2 = Offset 4tes Mipmap
...

Palettenkodierung (Bildtyp 5):

156 + 1024 = Offset 1tes Mipmap
156 + 1024 + width*height = Offset 2tes Mipmap
156 + 1024 + (width/2)*(height/2) = Offset 3tes Mipmap
156 + 1024 + (width/4)*(height/4) = Offset 4tes Mipmap
...

JPGkodierung(schwierig):

160 = Offset 1tes Mipmap
160 + 2 + jpgDaten(Größe) = Offset 2tes Mipmap
...

Erklärung:

156(blp-header) + 4(jpg-header-size) + 2(Marker:FFD8) + ?(jpgDaten)
Die Anordnung der jpgDaten sind eigentlich genau wie in jeder herkömmlichen jpgDatei, außer das der Header nur 2 bytes lang ist, also die beiden Marker FFD8 besitz.Ihr könnt natürlich statt den 2 bytes auch immer den vollen jpg-header reinschreiben, dass mach eure blp - Datei nur etwas länger. Wie man die Größe berechnet kann man einer jpg - Spezifikation entnehmen, ich denke es gibt da auch feste Werte. Eine gute Spezifikation gibt es auf http://de.wikipedia.org/wiki/JPEG.

64 Bytes (16xint): Mipmapsize

Wie bestimmt man die Size selbst?

Palettenkodierung (Bildtyp 3 oder 4):

width*height*2 = Size 1tes Mipmap
(width/2)*(height/2)*2 = Size 2tes Mipmap
(width/4)*(height/4)*2 = Size 3tes Mipmap
...

Palettenkodierung (Bildtyp 5):
width*height = Size 1tes Mipmap
(width/2)*(height/2) = Size 2tes Mipmap
(width/4)*(height/4) = Size 3tes Mipmap
...
JPGkodierung(schwierig):

Eine blp-Datei kann nur 16 Mipmap haben das macht bei allen 16 Mipmaps, die gespeichert werden, einen Platzbedarf von 2^16 = 65536 bytes. Man kann leicht mit den jpgDaten diesen Wert überschreiten, also **VORSICHT!!!**

BLP Data

##
BLP Data (allgemein)
##

Es kann sein das bei einigen blps, zwischen dem header und den eigentlichen Daten, 0 stehen. Diese haben keine weitere Bedeutung und können gelöscht werden.

Aber woher weis man dann wo die Daten anfangen?

Ganz einfach das erste mipmapoffset im header stimmt eigentlich immer. Wenn man das hat kann man die anderen ganz einfach bestimmen.

Bei jpeg kodierten Daten ist es **"nur"** wichtig beim dekodieren den Anfang richtig zu erwischen. Wenn der stimmt kann man **"nichts"** mehr falsch machen auch wenn man zuviel Bytes einliest. Denn jpeg benutz feste Sprungmarken.

##
BLP Data (jpeg - Daten)
##

Wie schon erwähnt folgen nach den ersten 156 bytes weitere 4 bytes, welche die jpeg - header Größe angeben. Dieser Wert beträgt meistens 10. Dann folgt der jpeg header, er ist so groß wie davor es ausgelesen wurde.

Beispiel:
Man liest von 156 - 160 die 4 bytes als Integer aus. Dieser Wert sagt einem wie groß der folgende header ist. Ist also der Wert beispielsweise 10, dann ist der header 10 bytes lang und beginnt bei 160 und geht bis 170.

Blizzards blp braucht keinen vollständigen jpg - header sondern nur die beiden Marken **FF D8**. Danach können die anderen Daten direkt folgen,

daher steht die jpg - header - size meist nur provisorisch im blp.
Nach diesem Integer folgen meist die beiden Marker FF D8 und danach der Rest. Man kann also nichts falsch machen, wenn man nach dem Integer alles andere einliest.

WICHTIG!
Die Farbinformationen in den blps werden nicht in **RGBA** abgespeichert, sondern in **BGRA**. Das bedeutet euer ausgelesenes jpg wird in blau gehalten sein :D. Um das zu ändern muss man sich entweder einen jpg - Dekodierer selber schreiben oder einen vorgebenen benutzen.

##
BLP Data (Palette - Daten)
##

Nach den ersten 156 bytes folgt bei einer Palettenkodierung gleich die Palette selbst. Das heißt das blp hat 8-bit und somit 2^8 = 256 Farben. Eine Farbe wird in 4 byte gespeichert, d.h.

1 byte = Blau (unsigned int 0-255)
1 byte = Grün (unsigned int 0-255)
1 byte = Rot (unsigned int 0-255)
1 byte = Alpha* (unsigned int 0-255)

Das macht 4 * 256 = 1024 Plätze in der Palette.

WICHTIG!
Die Farbinformationen werden wieder in BGRA gespeichert, also so wie oben. Die Palette hat immer eine Größe von 1024, auch wenn euer zu speicherndes Bild z.B. nur 60 Farben hat. Die restlichen Indizes in der Palette sind dann halt mit 0 belegt.

Nach der Palette folgen die Bilddaten und zwar genau width*height bytes. Jedes byte verweist auf einen Platz in der Palette aber nicht etwa index * 4 sondern nur der Index (unsigned int 0-255).

Bei einem Bildtyp von 3 und 4 bestehen die Bilddaten einmal aus einer Indexliste, also die Liste mit den Verweisen auf die Indizes in der Palette und aus einer Alphaliste, welche die Alphawerte für jeden Bildpunkt enthält. Die Größe der Bilddaten ist also width*height*2.

Bei einer 5 als Bildtyp bestehen die Bilddaten nur aus einer Indexliste, also ist die Größe der Daten width*height. Die Alphawerte stehen in der Palette, wie oben gezeigt, wobei der Wert aber inventiert ist,d.h. Alpha* = 255 - Alpha.

##
Mipmaps

##

Was ist ein Mipmap?

Ein Mipmap ist eigentlich nichts weiter als die Bilddaten eines Bildes.

Warum hat eine blp - Datei mehere Mipmaps?

Das erste Mipmap mit der Größe width*height ist das eigentliche Bild. Danach folgen mehrere Mipmaps die jedes mal kleiner sind als das Mipmap davor und zwar genau um den Faktor 2.

Beispiel:
width = 128
height = 64

Mipmap - Folge:
128x64, 64x32, 32x16, 16x8, 8x4, 4x2, 2x1, 1x1

Die kleineren Mipmaps werden z.B. für Vorschaubilder benutz oder Iconbilder. Man muss sich natürlich überlegen ob man ein Bild der Größe 1x1 gebrauchen kann :D. Ich denke 4 oder 8 ist eine gute Grenze aber wer will kann sie auch höher wählen. Es reicht also auch eine Mipmap - Folge der Form

128x64, 64x32, 32x16, 16x8, 8x4

oder

128x64, 64x32, 32x16, 16x8

aus.

Auch ungerade Bildgrößen funktionieren, indem man einfach runter rundet.

Beispiel: (das Beispiel ist aus Magos BLP - Spezifikation)
width = 24
height = 17

Mipmap - Folge:
24x17, 12x8, 6x4, 3x2, 1x1

WICHTIG!
Egal welche Folge ihr wählt, es muss immer eine Mipmap - Folge vorhanden sein, also wählt eure Grenze nicht zu hoch :D.

BLP2 Header

##
BLP2 Header (allgemein)
##

Der blp2 - header ist immer 148 bytes lang, d.h. 20 bytes Kennungs-
daten, dann 64 bytes Mipmapoffset - Daten und 64 bytes Mipmapsize-
Daten.

##
BLP2 Header (kurz)
##

4 Bytes (char):	BLP2
4 Bytes (int):	Kompression (0 für jpeg, 1 für palette oder dxt)
4 Bytes (char):	Kennungen :
	char[0] = 2 -> DDS-Format
	char[1] = 8 -> DXT3
	char[1] != 8 -> DXT1
4 Bytes (int):	Breite des Bildes (width)
4 Bytes (int):	Höhe des Bildes (height)
64 Bytes (int):	Mipmapoffsets (4 Bytes = 1 Mipmapoffset)
64 Bytes (int):	Mipmapsizes (4 Bytes = 1 Mipmapsize)

##
BLP2 Header (ausführlich)
##

4 Bytes (char): BLP2

Die ersten 4 bytes repräsentieren die Kennung einer
blp - Datei. Es handelt sich dabei um die Zeichen "BLP2”
als bytes. Um sie als chars zu bekommen maskiert man das
jeweilige byte mit 0xff und erhält einen Integer im Wertebereich
0 - 255, diesen castet man auf char und erhält das
entsprechende Zeichen.

4 Bytes (int): Kompression (0 für jpeg, 1 für palette oder dxt)

Man bekommt dann entweder eine 0 oder eine 1 als Wert.
Wenn man eine 0 bekommt dann sind die Bild -
daten dieses blps als jpeg - Daten vorhanden. Bei einer 1 als
Wert bestehen die Daten entweder aus einer Farbpalette mit
folgender Indexlist oder aus einem DXT Kodierten Format.
Das DXT – Format bedeutet Direct X Texture Format und wird
von Microsofts DDS – Format unterstützt. Man unterscheidet
zwischen DXT1, DXT2, Dxt3, DXT4 und DXT5, wobei in einem
blp2 – File nur DXT1 oder DXT3 zu finden ist. Gute Spezifikationen gibt es unter :
http://en.wikipedia.org/wiki/S3_Texture_Compression#DXT1
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/Getting_Started.asp

4 Bytes (char): Kennung

Wenn das erste Byte eine als char interpretiert eine 2 ist, dann liegen die Bilddaten im DDS – Format vor. Das zweite Byte entscheidet nun welche der Kompressionen vorliegt. Handelt es sich dabei um eine 8, dann liegt ein DXT3 kodiertes Bild vor, alles ungleich 8 bedeutet DXT1.

4 Bytes (int): Breite des Bildes (width)

Die Breite des Bildes wird hier in Pixel angeben.

4 Bytes (int): Höhe des Bildes (height)

Die Höhe des Bildes wird ebenfalls in Pixel angeben.

64 Bytes (16xint): Mipmapoffset

Die Angabe zu JPEG bzw. Paletten – Mipmaps sind die Selben wie bei einem blp0 oder blp1. Bei einer DXT1 - Komprimierung beträgt die Größe des ersten Mipmaps (width * height)/2, wenn es sich um ein Bild mit 24-bit handelt. Der Offset wäre hier also 1172 (**Achtung** : man schreibt, obwohl es keine Palette gibt, eine leere Palette rein, also nur Nullen). Bei DXT3 beträgt die Größe width * height, auch hier beginnt der erste Offset bei 1172.

64 Bytes (16xint): Mipmapsize

Diese wurden schon unter **Mipmapoffset** angegeben.

```
################################################################################
# BLP2 Bilddaten (kurz)                                                        #
################################################################################
```

DXT1 und DXT3 kann man aus den oben angegebenen Links entnehmen.
(ich schreibe demnächst auch noch eine eigene Spezifikation)

JPEG und Palette sind ja aus BLP0/1 bekannt.

```
################################################################################
# OpenSource : BLPConvert                                                      #
# von : Maik Gollnick                                                          #
# Quellcode auf meiner Seite                                                   #
################################################################################
```