

GBH Style Format

Version 4.6
16th November, 1999
©1997-1999 DMA Design Ltd

1. Table of Contents

<i>1. Table of Contents</i>	<i>2</i>
<i>2. Introduction</i>	<i>3</i>
<i>3. Palette Data</i>	<i>3</i>
<i>4. Tiles</i>	<i>4</i>
<i>5. Sprites</i>	<i>4</i>
<i>6. Deltas</i>	<i>5</i>
<i>7. Fonts</i>	<i>6</i>
<i>8. Cars</i>	<i>6</i>
<i>9. Map Objects</i>	<i>8</i>
<i>10. Recycling Info</i>	<i>9</i>
<i>11. PSX Tiles</i>	<i>9</i>
<i>12. File Format</i>	<i>9</i>
<i>13. Document Revision List</i>	<i>11</i>

2. Introduction

This document defines the file and data format used by the style file for GBH.

3. Palette Data

3.1 Palette Index

Every graphic in the game uses an individual 256-colour palette. This has associated with it a virtual palette number. This can be anything from 0 to 16383. However, only 1024 physical palettes are actually stored. The palette index is used to map virtual palette numbers onto physical palettes.

The palette index looks like this :

```
struct palette_index
{
    UInt16 phys_palette[16384];
};
```

Thus, for every virtual palette number, there is an unsigned 16-bit integer which stores the physical palette number.

3.2 Physical Palettes

The physical palettes themselves are stored in order in 64 palette pages of 64K each. Each page contains 64 palettes of 256 doublewords (1024 bytes) each. The palettes are stored in doubleword-wide columns down each page. The entry for each colour is a doubleword containing the actual colour information in BGRA order.

Any unused physical palettes are filled with zeros.

Note that colour 0 of each palette is always transparent. This means that graphics must not use colour 0 for anything other than transparent areas. Graphics which do not have any transparent areas must not use colour 0 at all.

3.3 Remap Palettes

Various auxilliary remap palettes are stored towards the end of the virtual palette space. There are specific areas for cars, peds, code objects, map objects, fonts and user graphics. Each type of remap has a virtual palette number which is relative to the start of its own area.

3.4 Palette Bases

Palette base data allows the game to calculate the virtual palette offset for the start of the set of palettes for each of the various types of graphics in GBH. In the style file, it stores the number of physical palettes which there are for each set. The palette base data looks like this:

```
struct palette_base
{
    UInt16 tile;
    UInt16 sprite;
    UInt16 car_remap;
    UInt16 ped_remap;
    UInt16 code_obj_remap;
    UInt16 map_obj_remap;
    UInt16 user_remap;
    UInt16 font_remap;
};
```

To calculate the base values, each value is set to the total of the values before it, with the first one set to 0.

4. Tiles

All tiles in GBH are stored as uncompressed 64x64 pixel 256-colour graphics. 992 tiles are stored, in 63 pages of 64K each. Each page is 256 x 256 pixels.

No distinction is made between side tiles and lid tiles.

The tiles are numbered 0 to 991, and stored in that order. Note that tile numbers 992 to 1023 are used internally by the game (for asynchronous animation).

1 virtual colour palette is stored per tile.

The virtual palette number for a tile is:
tile number + tile palette base

5. Sprites

5.1 Sprite Graphics Data

Sprites are stored as uncompressed 256-colour graphics upto 32 pages of 64K each. Each page is 256x256 pixels. The maximum size of a sprite is 128x64 and the minimum is 2x2. The width and height can have any value inbetween as long it is an even number.

5.2 Sprite Index

An index stores the width & height of each sprite and its position in the sprite graphics pages. The index entry for a sprite looks like this :

```
struct sprite_entry
{
    UInt32 ptr;
    UInt8 w,h;
    UInt16 pad;
};
```

ptr is a pointer relative to the start of the sprite graphics data.

One index entry exists for every sprite.

The virtual palette number for the default palette for a sprite is:
sprite_number + sprite palette base

5.3 Sprite Bases

Various different game items often refer to sprites. Each type of game item refers to its own sprites using a relative sprite number, which should be unaffected by changes to other game item sprite numbers. The base for each game item is an offset to the start of the sprite numbers. The sprite base data looks like this :

```
struct sprite_base
{
    UInt16 car;
    UInt16 ped;
    UInt16 code_obj;
    UInt16 map_obj;
    UInt16 user;
    UInt16 font;
};
```

Note that in the file these numbers store simply the number of sprites of each type. To get the correct base values, they must be added together to produce cumulative totals : to calculate the base values, each value is set to the total of the values before it, with the first one set to 0.

6. Deltas

6.1 Delta Store

Deltas are used to modify sprites. Each delta contains a stream of instructions which are designed to be used to modify the appearance of a particular sprite, e.g. to add a dent to the wing of a car. Any sprite can have deltas added. However, each delta works with only one sprite. A sprite can have up to 32 deltas.

Deltas have this format :

```
offset ( 2 bytes )
length ( 1 byte )
data ( length bytes )
```

This is repeated as many times as is necessary to represent the differences between the original sprite and the changed one which the delta is for.

Note that the offset is always relative to the last position used (initially zero). It must take account of the fact that the sprites are stored in 256-pixel wide pages.

Chunks of data will not go over the end of a line. A new one will start for each line.

Deltas are stored consecutively, with all the deltas for the same sprite grouped together in sets.

6.2 Delta Index

A delta index entry appears for each set of deltas. A set of deltas is all those deltas which apply to a single sprite. A delta index entry looks like this:

```
struct delta_entry
{
    UInt16 which_sprite;
    UInt8 delta_count;
    UInt8 pad;
    UInt16 delta_size[variable - see delta_count];
};
```

which_sprite is the sprite number of the sprite to which this delta is to be applied.

delta_size is an array of *delta_count* unsigned 16-bit ints, detailing the size in bytes of each of the deltas in this set. The order of the deltas in the index is the same as the order of the deltas in the delta store.

There is no need to associate palettes with deltas – a delta always uses the same palette as the sprite.

7. Fonts

The graphics for fonts are simply sprites – they are stored in among the other sprites. Each font character has a width and height which is stored in the usual sprite index. The font characters have virtual palette numbers which follow the same rules as all other sprites.

7.1 Font Bases

To enable individual fonts and font characters to be easily accessed, font bases are stored.

These look like this :

```
struct font_base
{
    UInt16 font_count;
    UInt16 base[variable - see font_count]
};
```

Note that in the file these numbers store simply the number of characters in each font. To get the correct base values, they must be added together to produce cumulative totals : to calculate the base values, each value is set to the total of the values before it, with the first one set to 0.

The overall sprite number for a font is :

(character code - first character) + font sprite base + font base

8. Cars

Various parameters are stored for each car in GBH. A car info structure is stored for each distinct type of car. The car info structure for one car model looks like this:

```
struct car_info
{
    UInt8 model;
    UInt8 sprite;
    UInt8 w,h;
    UInt8 num_remaps;
    UInt8 passengers;
    UInt8 wreck;
    UInt8 rating;
    Int8 front_wheel_offset, rear_wheel_offset;
    Int8 front_window_offset, rear_window_offset;
    UInt8 info_flags;
    UInt8 info_flags_2;
    UInt8 remap[variable - see num_remaps];
    UInt8 num_doors;
    door_info doors[variable - see num_doors];
};
```

w and *h* are the width and height of the car in pixels. These are required here because they may be different from the width and height of the car for collision purposes.

model is the car model number. Every distinct type of car has a unique *model* number.

sprite is the relative car sprite number. At least one sprite is stored for every car. The sprite number for each car is simply: car sprite number + car sprite base. In practice, the relative sprite number is actually filled in here by the game when the style is loaded. The style file only needs to store here the number of sprites used by the car (0 or 1). If a car has 0 sprites, it shares the graphic of the preceding one.

passengers is the number of passengers which this car can carry (not including the driver).

wreck is the wreck graphic number to use when this car is wrecked (0-8, or 99 if can't wreck).

rating is the quality rating for this car – used to decide how often it is created in different areas of the city. Values are :

1	bad
2	bad x 2
3	bad x 3
11	average
12	average x 2
13	average x 3
21	good
22	good x 2
23	good x 3
99	not recycled

front_wheel_offset and *rear_wheel_offset* are the distances in pixels from the centre of the car to the front axle, and to the back axle, respectively.

front_window_offset and *rear_window_offset* are the distances in pixels from the centre of the car to the front window, and to the back window, respectively.

remap stores a list of virtual palette numbers, representing all of the alternative palettes which can sensibly be applied to this car. Note that these palette numbers are relative to the start of the car remap palette area.

All visible doors in cars in GBH must be openable. A list of *num_doors* door structures is stored for each car, where a door structure entry looks like this :

```
struct door_info
{
    Int8 rx, ry;
};
```

rx and *ry* are the position relative to the centre of the car where a ped graphic should be drawn when on the last frame of getting into the car (or the first frame of getting out of the car) via this door. This is normally the position of the outer edge of the inside of the car when the door is open.

There is one special case here. If *rx* is greater than 64 (or less than -64) then 64 must be subtracted (or added) before *rx* is used. When this happens, it indicates that peds should enter/exit the car at this door by simply walking straight in, rather than by going through the sit-down/stand-up animation which they use at other doors. This is used, for example, for the sliding doors on a train.

info_flags is a bitmap with the following fields:

info_flags			
bit	value	name	meaning
0	0x01	ped_jump	1 if this car is too high for a ped to jump, else 0
1	0x02	emerg_lights	1 if this car has emergency lights (e.g. police car), else 0
2	0x04	roof_lights	1 if this car has roof lights (come on with headlights), else 0
3	0x08	cab	1 if this car can be used as an artic cab, else 0
4	0x10	trailer	1 if this car can be used as an artic trailer, else 0
5	0x20	forhire_lights	1 if this car has forhire lights (e.g. taxi) else 0
6	0x40	roof_decal	1 if this car has a roof decal (e.g. TV van) else 0
7	0x80	rear_emerg_lights	1 if this car has rear emergency lights (else 0)

NOTE:

- no car can have more than one out of *emerg_lights*, *roof_lights*, *forhire_lights* & *roof decal*.
- no car can be both *cab* and *trailer*
- a car with *rear emerg lights* must have *emerg lights* as well

info_flags_2 is a bitmap with the following fields:

info_flags_2			
bit	value	name	meaning
0	0x01	collide_over	1 if this car can drive over other cars, else 0
1	0x02	popup	1 if this car has popup headlights, else 0

8.1 Car Delta Usage

Cars are the biggest user of deltas in GBH. They use the deltas for the relevant sprite in a standard order, as follows :

car delta usage	
delta number	usage
0	rear right dent
1	rear left dent
2	front left dent
3	front right dent
4	windscreen damage
5	left brake light
6	left headlight
7-10	left front door
11-14	left back door/FBI light anim
15	emerg/roof lights/decal
16	emerg lights
17	right rear emerg light
18	left rear emerg light
22(mirror)	right brake light
23(mirror)	right headlight
24-27(mirror)	right front door
28-31(mirror)	right back door

Note that these directions are relative to the car. The delta numbers are relative to the start of the set of deltas for that sprite. Deltas 22 to 31 are mirrors of 5 to 14 – they are not actually stored in the style file.

The racing car with a number on the roof uses deltas 11-20 for the numbers.

9. Map Objects

A very small amount of type information is stored for each type of map object. The object info structure for one object model looks like this :

```
struct object_info
{
    UInt8 model;
    UInt8 sprites;
};
```

model is the object model number. Every distinct type of object has a unique *model* number. Objects placed in the map using the editor are represented by a model number.

sprites is the number of sprites stored for this model.

Many other properties of objects are required for GBH, but these are stored in code header files, not in the style file, because they are more likely to be updated by programmers than by artists, and the editor does not need to know about them.

Note that there are two different kinds of object – map objects and code objects. Map objects (e.g. bins and hot dog stands) can be placed by the editor and hence have their information stored here. Code objects (e.g. blood and skidmarks) can only be placed by the code so no information is stored for them here.

10. Recycling Info

A list of car model numbers is stored in the style file to show what cars should be included in the recycling. This list uses one byte per entry, has no more than 64 entries, is in ascending order of model number, and is terminated by the sentinel value 255.

11. PSX Tiles

The PSX tile format is as follows :

256x256 page of 256 colour 32x32 tiles	(65536 bytes)
256x256 page of 256 colour 32x32 tiles	(65536 bytes)
256x256 page of 16 colour 32x32 tiles	(32768 bytes)
256x256 page of 16 colour 32x32 tiles	(32768 bytes)
256 byte table containing palette number for each tile	(256 bytes)
Number of 256 colour palettes	(1 byte)
Number of 16 colour palettes	(1 byte)
256 colour palettes (15 bits BGR, 1 bit Alpha)	(512 bytes)
16 colour palettes (15 bits BGR, 1 bit Alpha)	(32 bytes)

NOTE:

The 256 byte palette table may be scrapped if we have yet another 1024 byte look-up table - for the tile palettes - as well as a pc to psx tile reduction table.

12. File Format

All of the above data is stored on disk in a single style file, which is used by both the game and the editor and must be kept up to date. The file consists of a header and a series of named chunks (which can be in any order).

Any program which uses the style file should load in the chunks it needs and ignore the others.

Any program which both loads and saves style files must make sure that all chunks are transferred.

The file header looks like this :

style file header		
name	size	notes
file_type	Char[4]	“GBST” – code for GBH style file
version_code	UInt16	- style file format version – currently 600

Each chunk in the file is preceded by a header which looks like this :

chunk header		
name	size	notes
chunk_type	Char[4]	code name for the chunk type
chunk_size	UInt32	size of the chunk in bytes (not including this header)

The chunks in the map file are :

map chunks	
chunk name	description
PALX	palette index
PPAL	physical palettes
PALB	palette base
TILE	tiles
SPRG	sprite graphics
SPRX	sprite index
SPRB	sprites bases
DELS	delta store
DELX	delta index
FONB	font base
CARI	car info
OBJI	map object info
PSXT	PSX tiles
RECY	car recycling info

Note that the chunk system is exactly the same as that used by the GBH map file.

13. Document Revision List

version	date	author	description
1.00	27/10/97	KRH	first draft
1.10	17/11/97	KRH	extra sprite types & remap palettes
1.20	19/11/97	KRH	fonts
1.30	28/11/97	KRH	EPAL & better base description
1.40	10/02/98	KRH	sprites in object_info changed
2.00	18/03/98	KRH	adjust alignment of structures to match PS. Remove tile remaps. Car deltas updated
2.10	22/04/98	KRH	extra information on door positions, removed EPAL
2.20	12/05/98	KRH	added user remaps, extra car info
2.50	25/06/98	KRH	car remap table changed to UInt8, added tile remaps
3.0	09/07/98	KRH & JC	add PSXT chunk
3.1	18/08/98	KRH	update delta numbers, and add info_flags to car_info
3.2	14/09/98	KRH	add window offset data to car_info
3.3	17/09/98	KRH	add forhired_lights to car_info_flags
3.4	05/11/98	KRH	992 tiles
3.5	15/12/98	KRH	add roof_decals to car_info_flags & deltas
3.6	05/02/99	KRH	add rear_emerg_lights to car_info_flags & deltas
3.7	24/03/99	KRH	add info_flags_2 to car_info
3.8	28/04/99	KRH	add FBI to info_flags_2 + wreck to car_info, and update delta numbers
3.9	03/05/99	KRH	add rating to car_info & change sprite number counting & remove FBI flag
4.0	12/05/99	KRH	more sprite pages
4.1	02/06/99	KRH	remove bulletproof flag
4.2	22/06/99	KRH	add RECY chunk
4.3	29/06/99	KRH	add collide_over bit
4.4	08/07/99	KRH	add logo offset to car_info
4.5	12/07/99	KRH	remove logo offset from car_info
4.6	16/11/99	KRH	Correct no. virtual palettes per tile (now 1, not 4)