

# GBH Map Format

Version 8.2  
18<sup>th</sup> August 1999  
©1997-1999 DMA Design Ltd

## **Table of Contents**

<b>1. TABLE OF CONTENTS</b>	<b>2</b>
<b>2. INTRODUCTION</b>	<b>3</b>
<b>3. MAP FORMAT</b>	<b>3</b>
<b>4. COMPRESSION</b>	<b>6</b>
<b>5. MAP ZONES</b>	<b>7</b>
<b>6. MAP OBJECTS</b>	<b>9</b>
<b>7. PSX MAPPING TABLE</b>	<b>10</b>
<b>8. TILE ANIMATION</b>	<b>10</b>
<b>9. LIGHTS</b>	<b>11</b>
<b>10. JUNCTION LIST</b>	<b>11</b>
<b>11. FILE FORMAT</b>	<b>12</b>
<b>12. DOCUMENT REVISION LIST</b>	<b>14</b>

## Introduction

This document describes the format of the map file used by GBH, and the format of the data in that file.

## Map Format

The GBH map is stored as a 3D array of *block\_info* structures. The size of this array is fixed at 256x256x8. This corresponds to x and y co-ordinates from 0 to 255 and z co-ordinates from 0 to 7. Note that the player is not allowed to go outwith the range of 1 to 254 in x and y and 1 to 6 in z, so that there is a one-block unused buffer all around the outside of the map.

At the roof of the map (z=7), no filled blocks are allowed.

At the floor of the map (z=0), blocks can have a lid but no sides, and cannot be slopes. They are automatically treated as water ( i.e. the player drowns if he lands on top of them ).

The whole array, in uncompressed form, looks like this :

```
struct map
{
    block_info city_scape[8][256][256];
};
```

The structure of each element of the array looks like this :

```
struct block_info
{
    UInt16 left, right, top, bottom, lid;
    UInt8 arrows;
    UInt8 slope_type;
}
```

Within this, the *arrows* bitmap looks like this :

<b>arrows</b>	
bit 0	green left
bit 1	green right
bit 2	green up
bit 3	green down
bit 4	red left
bit 5	red right
bit 6	red up
bit 7	red down

Green arrows are used to mark the normal road network. Dummy cars use them to see where they are allowed to drive (i.e. which side of the road to drive on, where the corners & junctions are, etc.). The green arrows must be set up perfectly for the game to function.

Red arrows are used to mark special routes which are only used in special cases, e.g. mission cars in a hurry or police cars on an emergency. They might mark shortcuts across parks or pavements, for example.

Note that green and red arrows do not have to be on road. It is allowed to place them on any ground type except air.

The *slope\_type* bitmap looks like this :

<b>slope_type</b>	
bits 0-1	ground type ( 0 = air, 1 = road, 2 = pavement, 3 = field )
bits 2-7	slope type (0 = none 1- 2 = up 26 low, high 3 - 4 = down 26 low, high 5 - 6 = left 26 low, high 7 - 8 = right 26 low, high 9 - 16 = up 7 low – high 17 - 24 = down 7 low – high 25 - 32 = left 7 low – high 33 - 40 = right 7 low – high 41 - 44 = 45 up,down,left,right 45 = diagonal, facing up left 46 = diagonal, facing up right 47 = diagonal, facing down left 48 = diagonal, facing down right 49 = 3 or 4-sided diagonal slope, facing up left 50 = 3 or 4-sided diagonal slope, facing up right 51 = 3 or 4-sided diagonal slope, facing down left 52 = 3 or 4-sided diagonal slope, facing down right 53 = partial block left 54 = partial block right 55 = partial block top 56 = partial block bottom 57 = partial block top left corner 58 = partial block top right corner 59 = partial block bottom right corner 60 = partial block bottom left corner 61 = partial centre block 16x16 62 = reserved for future use 63 = indicates slope in block above )

A slope code of 63 indicates not that this block is a slope but that the block above is a slope. This code must always be set in blocks which are underneath slope blocks (except if the block underneath is a diagonal). The ground type of the block with a type 63 slope must be 0 (air). It cannot itself be a slope. In the map, slope blocks must always be placed only in a complete group – e.g. a lower 26° part must always be connected to a higher 26° part.

Codes 45-48 are diagonals. A diagonal is a special triangular block, where one side is at a 45° angle, the other sides are straight and the lid is a triangle. The up or down side which the diagonal is facing towards is ignored – it takes the data from the left or right side. A limited form of collision with diagonal blocks is supported – see below. A diagonal is not a slope – there is no need for a code 63 underneath it.

Codes 49-52 are diagonal slopes. There are 2 types of diagonal slope – 3-sided and 4-sided. 3-sided diagonal slopes are shapes with a triangular base going up to a point. They are designed to be placed on top of diagonals to provide spires. These are identified by having a lid tile number of 1023 – which is a dummy graphic number. The top, bottom, left & right sides are used to store the sides (with the top or bottom ignored as required)  
4-sided diagonal slopes are identified by the fact that they have a normal (or zero) lid tile number. These are the inverse shape of the 3-sided diagonal slopes and are used to fill in the corners between slopes. They have a triangular lid and an inverted triangle diagonal side which goes down to a point at the low corner. Note that the small triangular sides under this are not drawn at all.

A limited form of collision with 3-sided diagonal slopes is supported – see below. There is no need for a code 63 underneath.

Codes 53-56 are partial blocks. These are like normal cube blocks except only 24 pixels wide, instead of 64. The graphics are cropped to fit. There is no need for a code 63 underneath. As with diagonals, there is no collision support, so they must be kept separate from the player.

Codes 57-60 are partial corner blocks. These are like normal cube blocks except only 24 pixels wide and 24 pixels long, instead of 64. The graphics are cropped to fit. There is no need for a code 63 underneath. As with diagonals, there is no collision support, so they must be kept separate from the player.

The left, right, top & bottom 16-bit values store a bitmap like this :

bits 0-9	tile graphic number (0-1023)
bit 10	wall (1 = collide, 0 = no collide)
bit 11	bullet wall (1 = collide, 0 = no collide)
bit 12	flat (1 = flat, 0 = not flat)
bit 13	flip (1 = flip, 0 = no flip )
bit 14-15	rotation code (0=0°, 1=90°, 2=180°, 3=270°)

The lid 16-bit value stores a bitmap like this :

bits 0-9	tile graphic number (0-1023)
bit 10-11	lighting level (0-3)
bit 12	flat (1 = flat, 0 = not flat)
bit 13	flip (1 = flip, 0 = no flip )
bit 14-15	rotation code (0=0°, 1=90°, 2=180°, 3=270°)

*tile graphic number* simply indicates which of the possible 1024 tile graphics to draw on this surface. It serves as an index into the tile information in the style file. A value of 0 means leave it blank. 992-1022 are reserved for internal use by the game engine. 1023 is used as a dummy tile number to mark 3-sided diagonal slopes.

*wall* indicates whether or not a car, ped or object should collide with this tile.

*bullet wall* indicates whether or not a bullet should collide with this tile.

Note that only (*wall*=1,*bullet wall*=1), (*wall*=1, *bullet wall*=0) and (*wall*=0, *bullet wall*=0) are allowed. (*wall*=0, *bullet wall*=1) is not allowed as you cannot have a wall which stops bullets but not peds.

*wall* and *bullet wall* must be 0 for diagonals – the game does not support collision with these.

It is not allowed to set the *bullet* or *bullet wall* bit if the *tile graphic number* is 0 or if the tile is opposite a flat, as this would cause an invisible wall.

The *bullet* or *bullet wall* bits must be set at the outside edges of a slope ( to stop the player getting on to the slope from the side. They must not be set at the ends of a slope block, except at the highest end of the highest block.

*flat* indicates whether or not this tile should be treated as a flat. This means that it gets drawn transparently, and (except for a lid ) the tile opposite is used as the graphic for the reverse side.

If both matching sides of a block (i.e. top and bottom or left and right) are flat, then both tiles are drawn at both positions.

Flat slopes and diagonals are allowed.

*flip* indicates whether or not this tile is to be drawn flipped left to right.

*rotation code* describes a rotation to turn this tile by when drawing it. Any combination of flipping and rotation is allowed.

Note that flipping and rotation is not supported on the sides of slopes.

*lighting level* marks which shading level to apply to a lid tile. 0 is normal brightness. 1-3 are increasing levels of darkness.

### **Diagonals Collision**

The game supports a limited form of collision with diagonals only. It does this by creating a temporary dummy invisible object whenever it encounters a diagonal block. This means that collision with the bottom of a diagonal block works – but collision with the top of it (i.e. walking on top of it) does not. Players must be prevented from reaching the top part of diagonal blocks ( or even throwing things up there ).

Diagonals cannot be flat, and the diagonal wall will always be collidable whatever the wall and bullet wall bits are set to. The tiles behind the diagonal wall must have graphics and wall bits and bullet wall bits, and the tiles in front must not have them.

All this applies to 3-sided diagonal slopes as well (collision-wise, these are treated the same as diagonal blocks) – but not to 4-sided diagonal slopes.

### **Compression**

The memory required by the uncompressed map is 256x256x8x12 bytes = 6MB . Clearly, this is too large to be practical in-game. In addition, the map stores a lot of unnecessary data, such as hidden faces, etc. For these reasons, a compressed form of the map is required.

There are two types of compressed map : CMAP(16-bit) & DMAP(32-bit). CMAP was the original format but the PC version maps have outgrown this, so will now use DMAP.

The PSX version can use DMAP for development but must use CMAP in the final version.

After compression, the map is stored in 3 data areas :

```
CMAP version :
struct compressed_map
{
    UInt16 base [256][256];
    UInt16 column_words;
    UInt16 column[variable size - see column_words];
    UInt16 num_blocks;
    block_info block[variable size - see num_blocks];
};

DMAP version :
struct compressed_map
{
    UInt32 base [256][256];
    UInt32 column_words;
    UInt32 column[variable size - see column_words];
    UInt32 num_blocks;
    block_info block[variable size - see num_blocks];
};
```

#### **base**

- a 2D array of 16-bit/32-bit unsigned ints, where each int stores a word/dword offset into *column* – that is, the index of the start of the column which sits at that square on the ground.

## **column**

- a variable length array of words/dwords. For each column, the format is:

```
CMAP version :
  struct col_info
  {
    UInt8 height;
    UInt8 offset;
    UInt16 blockd[variable size - see height];
  };

DMAP version :
  struct col_info
  {
    UInt8 height;
    UInt8 offset;
    UInt16 pad;
    UInt32 blockd[variable size - see height];
  };
```

Here, *height* is the height of the column (0 to 7).

*offset* is the number of empty blocks at the bottom (0 to *height*).

*blockd* is a variable length array of block numbers, with *blockd*[0] storing the block at *z=offset*. Each block number is a reference to the block data stored in *block*.

In a CMAP, there is a limit of 65536 words of column data. In a DMAP, the limit is 131072 dwords.

## **block**

- a variable length array of *block\_info* structures, containing every distinct combination of faces & types required for the level. Each individual block is addressed by its block number.

Note that block 0 is always all zero – it represents an air block with no graphics.

In a CMAP, there is a limit of 65536 distinct blocks in the world. In a DMAP, the limit is 131072 blocks.

## **Hidden Surface Removal**

As part of the compression process, all hidden surfaces of the map are removed. This means that all tiles which cannot be seen by the player, because they are on the inside of a building or on the outside of the world, for example, have their face bitmap set to zero.

## **Map Zones**

Map zones are used in the game to identify named areas for various purposes.

Map zone information stores the name, size and position of a variable number of zones within the map. The structure of a single map zone entry is :

```
struct map_zone
{
  UInt8 zone_type;
  UInt8 x,y;
  UInt8 w,h;
  UInt8 name_length;
  Char name[0 to 255 - see name_length];
};
```

(*x,y*) is the position of the top left corner of the zone (in block co-ordinates).

*w* and *h* are the width and height of the zone in blocks. Only rectangular zones are allowed.

*name* is a variable-length string whose length is stored in *name\_length*. Valid lengths are 0 to 255 characters. There is no NULL terminator.

Valid values of the *zone\_type* field are :

zone_type	
0	general purpose
1	navigation
2	traffic light
3	unused
4	unused
5	arrow blocker
6	railway station (platform)
7	bus stop (pavement)
8	general trigger
9	unused
10	information
11	railway station entry point
12	railway station exit point
13	railway stop point
14	gang
15	local navigation
16	restart
17	unused
18	unused
19	unused
20	arrest restart

*general purpose* zones are used simply to mark an area so that it can be referred to by that name in mission scripts.

*navigation* zones are used to store the correct geographical names of areas within the city. When the player enters a new navigation zone, its name is displayed on screen. When a police radio message reports a crime, the enclosing navigation zone is used to describe the area. Every block in the city must be within some navigation zone. They can overlap, in which case the smaller area zone gets priority. It is not allowed for equally sized zones to overlap. The navigation zone names are not translated in foreign versions of the game – they always have the same name. For large areas, the game appends North, South, etc. to the displayed name, so as to give the player a better idea of where he is.

*traffic light* zones are used to mark junctions where traffic lights are required. The game will automatically construct a traffic light system at each traffic light zone. It is not allowed to place a traffic light zone on an area which is not a valid junction.

Traffic light zones must not be on slopes, and must not be within one block of the end of a slope. Traffic light zones must be at least a screen away from each other.

*arrow blocker* zones are used to mark zones in the map where gang arrows should not be drawn even if within a gang zone.

*railway station* zones are used to mark the location of railway stations. The platform zone marks where the platform is, i.e. where peds should stand. This must be adjacent to a valid train route. The stop point marks where trains should stop. The entry and exit points mark the points on the track where trains enter and exit the station.

*bus stop* zones are used to mark the location of bus stops. The zone is placed on the pavement and it marks where the peds should stand. Buses will stop at the nearest road piece (error if none close). There are no specific bus routes – buses are simply created at random and stop at bus stops whenever they pass them.

*general trigger* zones are used to mark the position of general purpose triggers. These are used by mission scripts to set off an event when the player enters an area of the map. Normally, some sort of event will be associated with the general trigger zone in the mission script.

*Information* zones are used to mark an area which has different information values from the area round about it. The actual information is set up using the mission script.

*gang* zones are used to mark the areas used by particular gangs. The name of the zone is the name of the gang. Any one gang may have many zones.

*local navigation* zones are used, like *navigation* zones, to mark the names of places which the player can visit. Local navigation zones will tend to be smaller than main navigation zones. On-screen, the local navigation zone is shown before the main navigation zone – e.g. “Johnny’s Strip Club, Hackenslash” would be displayed if the player is in the local navigation zone “Johnny’s Strip Club” and the main navigation zone “Hackenslash”. The rules for overlapping for local navigation zones are the same as for main navigation zones. However, unlike main navigation zones, local navigation zones do not have to cover the whole city.

*restart* zones are the places where the player is restarted after he dies . There must be at least one restart zone in a map. They can be any size . When the player dies, he is restarted at the nearest restart zone ( except if it is a mult-player game and another player is too close ) .

*arrest restart* zones mark the places where the player is taken to after he has been arrested. They must be only one block in size.

Named zones in the city need not have different names to each other. However, it should be noted that there could be confusion when a mission script refers to a zone by name if that name is not unique.

Zones must be sorted by order of area ( smallest first ) .

## **Station Names**

Railway station names start with a 5-letter code which is unique to that railway line. Following that, they have a code which is different for each station. This code is a digit 0-9 ( the order of which determines the order of visiting). The entry point, exit point and stop point for the station have the same name.

## **Map Objects**

Map objects are items, such as bins, piles of rubbish & trees which can be positioned on top of any valid ground block in the map.

An object entry looks like this:

```
struct
{
    Fix16 x,y;
    Ang8 rotation;
    UInt8 object_type;
} map_object;
```

(x,y) is the position of the object in (1.8.7) fixed-point block co-ordinates. The upper 9 bits of each number stores the block across the map ( with a sign ), and the lower 7 bits stores the position within the block. No z co-ordinate is specified, as the object is assumed to be sitting on top of the ground at whatever height that happens to be. An object cannot be placed where there is only air or water or on top of blocks without proper collision, such as diagonals.

Note that, although floating point co-ordinates are used internally by the game, co-ordinates such as these which may be stored in a file must be fixed point, or else console versions could not use the file.

*rotation* is the rotation of the object, stored as an *Ang8*, which maps the 360° of a circle onto values from 0 to 255, with 0 representing 0° and 128 representing 180°.

*object\_type* is a code which identifies the type of object. It serves as an index into the object information in the style file.

Note that map objects cannot be used to store parked cars.

## PSX Mapping Table

The PSX mapping table is used to map PC tile numbers onto PSX graphics.

The mapping table looks like this :

```
struct
{
    UInt8 tile_number;
    UInt8 undefined;
} mapping_entry;

mapping_entry mapping_table[1024];
```

The table is indexed by PC tile numbers [0-1023]. *tile\_number* gives the PSX tile number to use. *undefined* will be used to store information on lighting, etc.

## Tile Animation

The map can store tile animation data which is used to produce fixed background animations like moving water or flickering neon signs.

This tile animation data is stored in a list of *tile\_animation* structures. Each of these looks like this :

```
struct
{
    UInt16 base;
    UInt8 frame_rate;
    UInt8 repeat;
    UInt8 anim_length;
    UInt8 unused;
    UInt16 tiles[0 to 255 - see anim_length];
} tile_animation;
```

*base* is the base tile of the animation. This marks the tile as an animating tile.

*frame\_rate* is the number of game frames that each frame of the animation is displayed for.

*repeat* is the number of times the animation will be played. 0 means play forever.

*anim\_length* is the number of tiles in the animation.

*tiles* is a variable length array of UInt16s that stores the tile numbers that make up the animation.

## Lights

Lighting information is stored in an array of `map_light` structures. Each of these looks like this :

```
struct
{
    UInt32  ARGB;
    Fix16  x,y,z;
    Fix16  radius;
    UInt8  intensity;
    UInt8  shape;
    UInt8  on_time;
    UInt8  off_time;
} map_light;
```

*ARGB* is the colour of the light, with 8 bits each for alpha, red, green & blue. Note that alpha is not used and should be set to zero.

*(x,y,z)* is the position of the light in (1.8.7) fixed point world co-ordinates ( as with map objects ).

*radius* is the radius of the light in blocks, again in (1.8.7) fixed point. The maximum is 8. Fractions are allowed.

*intensity* is the strength of the light, from 0 (weakest) to 255 (strongest) .

*on\_time*, *off\_time* and *shape* are used to define animation of the light.

*on\_time* is how many game cycles (0-255) the light should stay on for. *off\_time* is how many game cycles (0-255) the light should stay off for. If *on\_time* is zero then the light is not animated.

If *shape* is not zero then it is the random variance for *on\_time* & *off\_time*, i.e. it is the maximum number of additional game cycles (0-255) which might be added to the on or off time of the light as it animates. Note that *shape+on\_time* and *shape+off\_time* must not exceed 255.

Only spherical lights are supported.

## Junction List

This is a list of junction information used in the routefinder mid-game. Generated by the editor from the map, it contains junction information, horizontal segments, vertical segments, the number of junctions, the number of horizontal segments, and finally, the number of vertical segments.

```
struct
{
    link  north;
    link  south;
    link  east;
    link  west;

    search_type  junc_type;
    UInt8  min_x;
    UInt8  min_y;
    UInt8  max_x;
    UInt8  max_y;
} junction
```

*north*, *south*, *west*, *west* are 4 byte descriptions of the 'links' to other junctions.

*search\_type* defines what arrows are involved with this junction.

*(min\_x , min\_y) (max\_x, max\_y)* are a pair of UInt8 coordinates describing the size of the junction.

```
struct
{
    UInt16 junction_num1;
    UInt16 junction_num2;
    UInt8 min_x;
    UInt8 min_y;
    UInt8 max_x;
    UInt8 max_y;
} segment
```

*junction\_num1*, *junction\_num2* are the junctions at either end of this segment.  
(*min\_x*, *min\_y*) (*max\_x*, *max\_y*) are a pair of UInt8 coordinates describing the area covered by this segment.

At the moment, these arrays are padded out to their maximum sizes – this may be changed in time.

## File Format

All of the above data is stored on disk in a single map 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 map file should load in the chunks it needs and ignore the others.

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

Any program which saves map data must ensure that the compressed map is up to date with respect to the uncompressed map.

The file header looks like this :

<b>map file header</b>		
name	size	notes
file_type	Char[4]	“GBMP” – code for GBH map file
version_code	UInt16	- map file format version – currently 500

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

<b>chunk header</b>		
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 :

<b>map chunks</b>	
chunk name	description
UMAP	uncompressed map
CMAP	compressed map (16-bit)
DMAP	compressed map (32-bit)
ZONE	map zones
MOBJ	map objects
PSXM	PSX mapping table
ANIM	tile animation
LGHT	Lights
RGEN	Junction List

## Document Revision List

Version	Date	Author	Comments
1.00	14/10/1997	KRH	first draft
2.00	16/10/1997	KRH	added zones, routes, objects, diagonals & changed map orientation
3.0	23/10/97	KRH	compression, file format, table of contents
4.0	27/10/97	KRH	ped routefinding
5.0	25/02/98	KRH	cmap & col_info & both sides flat updates, pedr removed
5.5	25/06/98	KRH	New zone types, new block types, lid lighting
5.6	30/06/98	KRH	Correct zone types
6.0	09/07/98	KRH	add PSXM chunk
6.1	03/09/98	KRH	add gang zones & local navigation zones, relax rule on unique zone names
6.2	09/09/98	KRH	add restart zones
6.3	17/09/98	KRH	add tile animation
6.4	25/09/98	KRH	changed tile animation format
6.5	17/12/98	KRH	partial centre blocks, inverse diag slopes, no rotation on slope sides, railway setup
7.0	07/01/99	KRH	add lighting information & change fixed point format to 1.8.7
7.5	26/01/99	KRH	change roadblocks & add animated lights
7.6	02/02/99	KRH	diagonals collision
7.7	23/02/99	BB	Junction list chunk info added
7.8	01/04/99	KRH	arrest restart zones
8.0	14/06/99	KRH	DMAPs & shading level added, roadblocks & routes removed
8.1	24/06/99	KRH	remove remap animation
8.2	18/08/99	KRH	arrow blocker zones