Profile Persistence in HCE

Serialising the Blam.sav Binary

Emilian Roman

Web: https://github.com/yumiris

January 12, 2018

DRAFT

Table of contents

1. Video Configuration

1.1. Resolution

The resolutions are stored in uint16_t variables:

```
struct resolution {
    uint16_t width;
    uint16_t height;
}
```

Read below for an insight into what happens when you forget that a byte can have a maximum value of 255.

1.1.1. Calculation

The width and height are each represented by two uint8 variables, A & B, whose values are calculated using the following formula:

```
unsigned int a = x / (2 ^ 8); // 2 ^ 8 = 256 or 0x100
unsigned int b = x % (2 ^ 8); // x = width (or height)
```

- The dividend is the width or height value, which can be an uint8 variable of a value up to 2^15 (32768 or 0x8000).
- The divisor can be represented with the magic unsigned integer of 2^8.
- Quotient ‘‘A’’ is the result *without* a remainder when dividing by 2^8.
- The decimal values are stored as their hexadecimal equivalent.

For example, 1920x1080 is represented in the blam.sav with the following values:

Width

Code	Dec	Hex
`int a = 1920 / (2 ^ 8);`	7	07
`int b = 1920 % (2 ^ 8);`	128	80

Height

Code	Dec	Hex
`int c = 1080 / (2 ^ 8);`	04	04
`int d = 1080 % (2 ^ 8);`	56	38

1.1.2. Retrieval

To get back the values from the blam.sav with the two variables, the following formula can be used: `unsigned int x = (a * (2 ^ 8)) + b;`

To get 1920 and 1080 back, we should use 128 & 7 and 56 & 4, respectively:

Width

Code	Result
`int x = (7 * (2 ^ 8)) + 128;`	1920

Height

Code	Result
`int y = (4 * (2 ^ 8)) + 56;`	1080

1.1.3. Offsets

Width

Formula	Variable	Address
X / (2 ^ 8)	A	0x00000A69
X % (2 ^ 8)	B	0x00000A68

Height

Formula	Variable	Address
Y / (2 ^ 8)	C	0x00000A6B
Y % (2 ^ 8)	D	0x00000A6A

1.2. Effects

1.2.1. Introduction

This section covers video effects that can be turned on and off. They're stored as boolean values in the binary file, represented by uint8 variables. Effects, in this context, refer to the following graphical options:

- Specular - Effects and luster to objects
- Shadows - Dynamic shadows
- Decals - Bullet holes and blood

1.2.2. Offsets

Option	Address
Specular	0x00000A70
Shadows	0x00000A71
Decals	0x00000A72

1.3. Framerate, Particles & Qualities

1.3.1. Introduction

This section covers video options that can have different values to signify their current state. In the binary files, they're stored as uint8. Options that are "levels" include the following:

- Frame Rate
- Particles
- Texture Quality

1.3.2. States

Frame Rate

State	Value
VSync Off	0x0
VSync On	0x1
30 FPS	0x2

Particles

State	Value
None	0x0
Low	0x1
Full	0x2

Texture Quality

State	Value
Low	0x0
Medium	0x1
High	0x2

Enumerators could be used for representing these states:

```
// enumerators
enum framerate_t { vsync_on, no_vsync, locked };
enum particles_t { none, low, full };
enum qualities_t { low, medium, high }

// handling the chosen framerate
framerate_t chosen_framerate;
chosen_framerate = vsync_off;

if (chosen_framerate == vsync_off)
{
    unsigned int framerate = 1; // 0x1
    // write the framerate to the file
}
```

1.3.3. Offsets

Option	Address
Frame Rate	0x00000A6F
Particles	0x00000A73
Texture Quality	0x00000A74

2. Audio Configuration

2.1. Volumes

2.1.1. Introduction

The volumes that are stored in the binary file are the following:

- Master
- Effects
- Music

The volumes for each option are stored using uint8 variables with a value between 0x00 and 0x0A, where 0x00 = 0 = Mute, and 0x0A = 10 = Maximum volume.

2.1.2. Offsets

Volume	Address
Master	0x00000B78
Effects	0x00000B79
Music	0x00000B7A

2.2. Quality & Variety

The audio quality and variety levels both have three possible states, represented by uint8 variables.

2.2.1. States

Quality

State	Value
Low	0x0
Normal	0x1
High	0x2

Variety

State	Value
Low	0x1
Medium	0x2
High	0x3

Like the video-related states, enumerators can be used for a higher-level representation of the audio quality & variety states:

```
// enumerators
enum quality_t { low, normal, high };
enum variety_t { low, medium, high };

// handling the chosen audio quality
quality_t chosen_quality;
chosen_quality = high;

if (chosen_quality == high)
{
    unsigned int quality = 2; // 0x2
    // write the quality to the file
}
```

2.2.2. Offsets

Option	Address
Sound Quality	0x00000B7D
Sound Variety	0x00000C7F

2.3. Enhancements

Enhancements in this context refer to Hardware Acceleration and Environmental Sound options, both which are binary values stored as uint8.

2.3.1. Switches

Acceleration

Switch	Value
No	0x0
Yes	0x1

Environmental

Switch	Value
Off	0x0
EAX	0x1

2.3.2. Offsets

Enhancement	Address
Hardware Acceleration	0x00000B7C
Environmental Sound	0x00000B7B

3. Network Configuration

This section covers the network options, which include the connection type and server/client connection ports.

3.1. Connection Types

3.1.1. States

HCE uses the following connection types for determining the maximum number of players in a self-hosted server. The selected value is stored in an uint8 variable.

Category	Type	Value
DSL	Poor	0x1
	Normal	0x2
	Best	0x3
~	T1/LAN	0x4
	56k	0x0

3.2. Connection Ports

3.2.1. Introduction

Ports are stored in uint16 variables with a maximum value of (2 ^ 16) - 1, or 65535.

Calculation and retrieval is done the exact same way as video resolutions, in the uint16_t format. For a more elaborate explanation of the formulae, check the Resolutions subsection in the Video Configuration section.

- Calculation:

```
unsigned int a = x / (2 ^ 8); // x = port
unsigned int b = x % (2 ^ 8);
```

- Retrieval:

```
unsigned int x = (a * (2 ^ 8)) + b; // a = port / 0x100; b = port % 0x100
```

3.3. Offsets

Server Port

Formula	Variable	Address
x / (2 ^ 8)	A	0x00001003
x % (2 ^ 8)	B	0x00001002

Client Port

Formula	Variable	Address
y / (2 ^ 8)	C	0x00001005
y % (2 ^ 8)	D	0x00001004

Option	Address
Connection Type	0x00000FC0

4. Player Customisation

4.1. Profile Name

The profile name is stored as a hex representation of its UTF-16 equivalent. The plain string value can be a maximum of 11 characters, and the caracters can only be ASCII characters.

4.1.1. Encoding

A simple way to represent it is to add null characters between each character.

Plain Value	Encoded Value	Encoded Value in Hexadecimal
New001	N.e.w.0.0.1.	4E 00 65 00 77 00 30 00 30 00 31 00

Since the plain value string can be a maximum length of 11 characters, the whole array can be a maximum length of 22 indices. A simple implementation in Python:

```python
def encode(name: str) -> bytearray:
    # plain value
    n = name

    # encoded value in hex
    # length would be name.length * 2
    y = {}

    # current loop iteration
    i = 0

    # for each character,
    # append its hex equivalent
    # and append null after it
    for c in n:
        y[i] = hex(ord(c))
        y[i + 1] = ’\0’
        i = i + 2

    return y


if __name__ == ’__main__’:
    print(encode(’New001’))
```

4.1.2. Decoding

The naive way to decode the name is to loop through the encoded name array and convert the hex value to a string. In Python, it’s unsurprisingly simple:

```python
def decode(name: bytearray) -> str:
    y = name
    n = ’’

    for c in y:
        n += chr(c)

    return n


if __name__ == ’__main__’:
    array = [
        0x4E,  # N
        0x00,
        0x65,  # e
        0x00,
        0x77,  # w
        0x00,
        0x30,  # 0
        0x00,
        0x30,  # 0
        0x00,
        0x31,  # 1
        0x00
    ]

    print(decode(array))
```

4.2. Player Colour

The player’s colour in online free-for-all game modes (e.g. slayer, oddball, etc.) is stored as an uint8, and can be one of the following 18 values:

Colour	Value
Aqua	0x0C
Black	0x01
Blue	0x03
Brown	0x0E
Cobalt	0x0A
Crimson	0x02
Cyan	0x09
Gold	0x05
Green	0x06

Colour	Value
Maroon	0x10
Orange	0x0B
Peach	0x11
Rose	0x07
Sage	0x0D
Snow	0x00
Steel	0x04
Tan	0x0F
Violet	0x08

4.3. Offsets

Option	Address
Name	0x00000002
Colour	0x0000011A