**The 7th Guest VDX**

From XentaxWiki

Jump to: [navigation](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#mw-head), [search](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#p-search)

**Contents**

* [1 VDX](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#VDX)
  + [1.1 Preface](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#Preface)
  + [1.2 Format Specifications](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#Format_Specifications)
  + [1.3 Block handling](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#Block_handling)
  + [1.4 Block types](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#Block_types)
    - [1.4.1 Type 0x00 (?)](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#Type_0x00_.28.3F.29)
    - [1.4.2 Type 0x20 (still image)](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#Type_0x20_.28still_image.29)
    - [1.4.3 Type 0x25 (animation)](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#Type_0x25_.28animation.29)
    - [1.4.4 Type 0x80 (sound)](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#Type_0x80_.28sound.29)
  + [1.5 Notes and Comments](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#Notes_and_Comments)
  + [1.6 Compatible Programs](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#Compatible_Programs)
  + [1.7 See Also](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/The_7th_Guest_VDX#See_Also)

**VDX**

* **Format Type**: Multimedia file \ Video file
* [**Endian Order**](https://web.archive.org/web/20230518100437/http:/en.wikipedia.org/wiki/Endianness): Little Endian

**Preface**

Extracting files from the \*.gjd sources in The 7th Guest will yield almost excusively \*.vdx files. These files represent some sort of multimedia container format. The importance of this will become clear as soon as one becomes aware of the fact that the GROOVIE engine used in The 7th Guest (and in a modified variant later in its successor, The 11th Hour) is basically a media player capable of running game scripts.  
Taking into account this design decision, the way the VDX format has been designed comes more or less naturally: A VDX file consists of several blocks, where each block can carry a different type of data (sound, image, animation), determined by a value in that particular block's header structure. This allows e. g. interleaved video/audio sequences.

**Format Specifications**

**// main VDX header**

**uint16 {2}   - Identifier (0x92, 0x67)**

**byte {6}     - *Unknown***

This main header is followed by an arbitrary number of blocks until the file ends. Each block has the following block header:

**// block header**

**byte {1}     - Block Type**

**byte {1}     - *Unknown*(possibly a play command for the GROOVIE player?)**

**uint32 {4}   - Data Size**

**byte {1}     - Length Mask**

**byte {1}     - Length Bits**

**byte {X}     - Block Data**

**Block handling**

Each block can optionally be compressed using a variant of the [LZSS](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/LZSS) algorithm. Whether a block has been compressed can be determined by the Length Mask and Length Bits values of the block header structure. If both are zero, the block has not been compressed and can be extracted by simply copying over the following block data. The size of the data directly corresponds to the Data Size entry in the block's header.  
  
If the block has been compressed (i. e. the block header entries Length Mask and Length Bits are non-zero), decompress the block data (again, the data size can be read from the Data Size entry) by applying the [LZSS](https://web.archive.org/web/20230518100437/https:/wiki.xentax.com/index.php/LZSS) decompression scheme using the following parameters:

* History buffer size N = 1 **shl** (16 - Length Bits), addressed from zero onwards
* Window size F = 1 **shl** (Length Bits), addressed from zero onwards
* Initial history buffer writing position: N - F
* Initial history buffer filling: probably arbitrary, 0x00 seems to work
* Compression threshold: 3
* Buffer references are relative to the current buffer write position.

**Block types**

The Block Type header value determines the type of data stored within the block (i. e. sound, image, animation ...). Currently, four types of blocks are known.

**Type 0x00 (?)**

This block type mostly (always?) seems to have zero data size and has only set the second byte of the header to a non-zero value. The values 0x67 and 0x77 have been observed for this byte. This might indicate some kind of play command to the GROOVIE player, but this is only an assumption.

**Type 0x20 (still image)**

This block contains bitmapped image data. This data has a header itself:

**// image data**

**uint16 {2}   - Horizontal Block Number**

**uint16 {2}   - Vertical Block Number**

**uint16 {2}   - Colour Depth (only 8 observed)**

**byte {768}   - Palette Data (in R/G/B byte format for 256 colours, as observed)**

**byte {X}     - Image Data**

Interestingly, *every* \*.vdx file seems to start with such a still image in full game screen proportions (and will not contain any further still image blocks). This holds true even for otherwise pure sound files, the block will then simply contain an empty (i. e. black or similarly uni-coloured) picture.  
The image contained in this block also has to be decompressed to convert it to a standard bitmap format. The image data consists of a sequence of blocks (Horizontal Block Number \* Vertical Block Number), each having the following format:

**// image block**

**byte {1}     - Colour 1**

**byte {1}     - Colour 0**

**uint16 {2}   - Colour Map**

Every such data block represents a 4x4 pixel image block within the decompressed picture, starting from the top left to the bottom right, line-wise.  
Such a 4x4 pixel image block will only contain two colours in the resulting image, determined by the Colour 1 and Colour 0 entries above. The bits of the Colour Map tell which colour should be placed where (starting again from the top left, choosing Colour 1 if and only if the appropriate bit in the Colour Map is set to 1).

**Type 0x25 (animation)**

These blocks contain animated (i. e. video) sequences. They are based on the (always-existing) still image of the 0x20 block as some sort of delta frames.

The data block is roughly structured as follows:

**// delta frame**

**uint16 {2}   - Local Palette Size**

**byte {32}    - Palette Bit Field**

**byte {X}     - Palette Entries (RGB)**

**byte {X}     - Image Data**

Please note that the first value always exists, whereas the next two fields can only be found when the local palette size is greater than zero!

The current palette is changed according to the Palette Bit Field. Each position in this 256 bits wide field represents one palette entry. If a bit is set, the new colours for the referenced palette entry can be found in the Palette Entries, which is a sequence of RGB colours following the bit field. Note that there are exactly as many bits set as there are RGB values following the bit field.

After the palette has been adapted, the last frame can be changed according to the Image Data. This data basically consists of a sequence of byte opcodes and a varying number of parameters following each opcode. Just as the still image in the block of type 0x20, the changes are performed on a picture divided into 4x4 pixel blocks. The changes start at the top left block.

The opcodes can be devided into several classes according to their function:

* 0x00..0x5f -- Followed by two parameters, which are hereby referred to as colours c1 and c0. The current block will be altered in the same fashion the still image in the 0x20 block is decompressed. The Colour Map used is extracted from a predefined array of uint16 values containing 96 entries. The correct entry can be identified by directly using the opcode as the element index. The map of predefined values is given here byte-wise:

$00, $c8, $80, $ec, $c8, $fe, $ec, $ff, $fe, $ff, $00, $31, $10, $73, $31, $f7,

$73, $ff, $f7, $ff, $80, $6c, $c8, $36, $6c, $13, $10, $63, $31, $c6, $63, $8c,

$00, $f0, $00, $ff, $f0, $ff, $11, $11, $33, $33, $77, $77, $66, $66, $cc, $cc,

$f0, $0f, $ff, $00, $cc, $ff, $76, $00, $33, $ff, $e6, $0e, $ff, $cc, $70, $67,

$ff, $33, $e0, $6e, $00, $48, $80, $24, $48, $12, $24, $00, $12, $00, $00, $21,

$10, $42, $21, $84, $42, $00, $84, $00, $88, $f8, $44, $00, $32, $00, $1f, $11,

$e0, $22, $00, $4c, $8f, $88, $70, $44, $00, $23, $11, $f1, $22, $0e, $c4, $00,

$3f, $f3, $cf, $fc, $99, $ff, $ff, $99, $44, $44, $22, $22, $ee, $cc, $33, $77,

$f8, $00, $f1, $00, $bb, $00, $dd, $0c, $0f, $0f, $88, $0f, $f1, $13, $b3, $19,

$80, $1f, $6f, $22, $ec, $27, $77, $30, $67, $32, $e4, $37, $e3, $38, $90, $3f,

$cf, $44, $d9, $4c, $99, $4c, $55, $55, $3f, $60, $77, $60, $37, $62, $c9, $64,

$cd, $64, $d9, $6c, $ef, $70, $00, $0f, $f0, $00, $00, $00, $44, $44, $22, $22

* 0x60 -- 16 parameters which represent individual colours. The 16 pixels of the current image block will be filled by these values.
* 0x61 -- No parameters. This opcode simply skips one line as a carriage return would do, i. e. the new position will be 4 pixels (1 block) further down on the left border of the frame.
* 0x62..0x6b -- No parameters. These opcodes skip blocks within a line. The current position is moved Opcode - 0x62 blocks to the right.
* 0x6c..0x75 -- One parameter defining a colour. The next Opcode - 0x6b blocks in the current line will be solidly filled with the defined colour.
* 0x76..0x7f -- Opcode - 0x75 parameters. Each defines a colour and is used to solidly fill a block within the current line.
* 0x80..0xff -- 3 Parameters. The opcode itself and the following byte represent a Colour Map, the next two are the colours c1 and c0, respectively. The block will be coloured according to the known method.

**Type 0x80 (sound)**

These blocks contain raw sound data sampled with 22,050 Hz, 8 bits, mono.

**Notes and Comments**

* Usually, the sound of an animation file can be extracted by simply appending all raw sound blocks found within the \*.vdx file and outputting them to a properly formatted \*.wav file.
* As indicated by the length of a default audio frame, the video sequences are played at a rate of 15 fps.
* The play commands within each block are probably for synchronization of audio and video frames. From the observed files synchronization can be achieved by repeating the last video frame as soon as a block with type 0x00 and play command 0x67 is encountered. The play commands within other blocks don't seem to be necessary for the purpose of correct extraction, but further research might prove this wrong.