

.XBE File Format 1.1

XBox Executable Documentation by Caustik (caustik@caustik.com)

This document is, afaik, the most precise .XBE documentation out there (other than official Microsoft documentation). The reason for this is the simple fact that I have done a lot of work with this type of file, and I am very familiar with the format. .XBE files are very similar to .EXE, so you won't have much of a problem using them if you have any sort of experience with .EXE files. Well, here we go..

Note: The author of this document is not responsible for anything you do with this information. This information is solely for entertainment purposes, and the author does not condone any illegal goals you may decide to use this information to achieve.

| [Image Header](#) | [Certificate](#) | [Section Header](#) | [Library Version](#) | [TLS](#) | [Logo Bitmap](#) | [C Source](#) |

.XBE Image Header			
Field Name	Description	Size	Offset
Magic Number	This field must always equal 0x48454258 ("XBEH")	0x0004	0x0000
Digital Signature	256 Bytes. This is where a game is signed. Only on officially signed games is this field worthwhile.	0x0100	0x0004
Base Address	Address at which to load this .XBE. Typically this will be 0x00010000.	0x0004	0x0104
Size of Headers	Number of bytes that should be reserved for headers.	0x0004	0x0108
Size of Image	Number of bytes that should be reserved for this image.	0x0004	0x010C
Size of Image Header	Number of bytes that should be reserved for image header.	0x0004	0x0110
TimeDate	Time and Date when this image was created. Standard windows format.	0x0004	0x0114
Certificate Address	Address to a Certificate structure, after the .XBE is loaded into memory.	0x0004	0x0118
Number of Sections	Number of sections contained in this .XBE.	0x0004	0x011C
Section Headers Address	Address to an array of SectionHeader structures, after the .XBE is loaded into memory.	0x0004	0x011C
Initialization Flags	Various flags for this .XBE file. Known flags are: MountUtilityDrive = 0x00000001 FormatUtilityDrive = 0x00000002 Limit64Megabytes = 0x00000004 DontSetupHarddisk = 0x00000008	0x0004	0x0124
Entry Point	Address to the Image entry point, after the .XBE is loaded into memory. This is where execution starts. This value is encoded with an XOR key. Considering this is far too weak to be considered security, I assume this XOR is a clever method for discerning between Debug/Retail .XBE files without adding another field to the .XBE header. The XOR key is dependant on the build: Debug = 0x94859D4B, Retail = 0xA8FC57AB To encode an entry point, you simply XOR the real entry point with either Debug or Retail key, depending on if you want the Xbox to see this as a Debug or Retail executable. To decode an entry point, you XOR with the debug key, then check if it is a valid entry point. If it is not, then you try again with the retail key. Note: The Kernel Image Thunk Address member of this header must also be encoded as described later in this document.	0x0004	0x0128
TLS Address	Address to a TLS (Thread Local Storage) structure.	0x0004	0x012C
PE Stack Commit	Copied from the PE file this .XBE was created from.	0x0004	0x0130
PE Heap Reserve	Copied from the PE file this .XBE was created from.	0x0004	0x0134
PE Heap Commit	Copied from the PE file this .XBE was created from.	0x0004	0x0138
PE Base Address	Copied from the PE file this .XBE was created from.	0x0004	0x013C
PE Size of Image	Copied from the PE file this .XBE was created from.	0x0004	0x0140

PE Checksum	Copied from the PE file this .XBE was created from.	0x0004	0x0144
PE TimeDate	Copied from the PE file this .XBE was created from.	0x0004	0x0148
Debug PathName Address	Address to the debug pathname (i.e. "D:\Nightlybuilds\011026.0\code\build\xbox\Release\simpsons.exe")	0x0004	0x014C
Debug FileName Address	Address to the debug filename (i.e. "simpsons.exe")	0x0004	0x0150
Debug Unicode FileName Address	Address to the debug unicode filename (i.e. L"simpsons.exe")	0x0004	0x0154
Kernel Image Thunk Address	<p>Address to the Kernel Image Thunk Table, after the .XBE is loaded into memory. This is how .XBE files import kernel functions and data.</p> <p>This value is encoded with an XOR key. Considering this is far too weak to be considered security, I assume this XOR is a clever method for discerning between Debug/Retail .XBE files without adding another field to the .XBE header. The XOR key is dependant on the build:</p> <p style="padding-left: 40px;">Debug = 0xEFB1F152, Retail = 0x5B6D40B6</p> <p>To encode a kernel thunk address, you simply XOR the real address with either Debug or Retail key, depending on if you want the Xbox to see this as a Debug or Retail executable.</p> <p>To decode a kernel thunk address, you XOR with the debug key, then check if it is a valid address. If it is not, then you try again with the retail key.</p> <p>The Kernel Thunk Table itself is simply an array of pointers to Kernel imports. There are 366 possible imports, and the table is terminated with a zero dword (0x00000000). Typically the values in this table can be generated with the following formula:</p> <p style="padding-left: 40px;">KernelThunkTable[v] = ImportThunk + 0x80000000;</p> <p>so, for example, the import PsCreateSystemThreadEx, which has a thunk value of 255 (0xFF) would be...</p> <p style="padding-left: 40px;">KernelThunkTable[v] = 0xFF + 0x80000000; // (0x800000FF)</p> <p>When the .XBE is loaded by the OS (or the CXBX Emulator), all kernel imports are replaced by a valid function or data type address. In the case of CXBX, the import table entry at which (KernelThunkTable[v] & 0x1FF == 0xFF) will be replaced by &cxbx_PsCreateSystemThreadEx (which is a wrapper function).</p> <p>Note: The Entry Point member of this header must also be encoded as described earlier in this document.</p>	0x0004	0x0158
Non-Kernel Import Directory Address	Address to the Non-Kernel Import Directory. It is typically safe to set this to zero.	0x0004	0x015C
Number of Library Versions	Number of Library Versions pointed to by Library Versions Address.	0x0004	0x0160
Library Versions Address	Address to an array of <u>LibraryVersion</u> structures, after the .XBE is loaded into memory.	0x0004	0x0164
Kernel Library Version Address	Address to a <u>LibraryVersion</u> structure, after the .XBE is loaded into memory.	0x0004	0x0168
XAPI Library Version Address	Address to a <u>LibraryVersion</u> structure, after the .XBE is loaded into memory.	0x0004	0x016C
Logo Bitmap Address	Address to the Logo Bitmap (Typically a "Microsoft" logo). The format of this image is described here . This field can be set to zero, meaning there is no bitmap present.	0x0004	0x0170
Logo Bitmap Size	Size (in bytes) of the Logo Bitmap data. The format of this image is described here .	0x0004	0x0174

.XBE Certificate			
Field Name	Description	Size	Offset
Size of Certificate	Number of bytes that should be reserved for this certificate.	0x0004	0x0000
TimeDate	Time and Date when this certificate was created. Standard windows format.	0x0004	0x0004
Title ID	Title ID for this application. This field doesn't appear to matter with unsigned code, so it can be set to zero.	0x0004	0x0008
Title Name	Title name for this application (i.e. L"The Simpsons Road Rage"). This buffer contains enough room for 40 Unicode characters.	0x0050	0x000C

Alternate Title IDs	Alternate Title IDs (16 4-byte DWORDs) for this certificate. These do not appear to matter with unsigned code (or signed code, for that matter), so they can all be set to zero.	0x0040	0x005C
Allowed Media	Allowed media types for this .XBE. The following values are known: <pre>#define XBEIMAGE_MEDIA_TYPE_HARD_DISK 0x00000001 #define XBEIMAGE_MEDIA_TYPE_DVD_X2 0x00000002 #define XBEIMAGE_MEDIA_TYPE_DVD_CD 0x00000004 #define XBEIMAGE_MEDIA_TYPE_CD 0x00000008 #define XBEIMAGE_MEDIA_TYPE_DVD_5_RO 0x00000010 #define XBEIMAGE_MEDIA_TYPE_DVD_9_RO 0x00000020 #define XBEIMAGE_MEDIA_TYPE_DVD_5_RW 0x00000040 #define XBEIMAGE_MEDIA_TYPE_DVD_9_RW 0x00000080 #define XBEIMAGE_MEDIA_TYPE_DONGLE 0x00000100 #define XBEIMAGE_MEDIA_TYPE_MEDIA_BOARD 0x00000200 #define XBEIMAGE_MEDIA_TYPE_NONSECURE_HARD_DISK 0x40000000 #define XBEIMAGE_MEDIA_TYPE_NONSECURE_MODE 0x80000000 #define XBEIMAGE_MEDIA_TYPE_MEDIA_MASK 0x0FFFFFFF</pre>	0x0004	0x009C
Game Region	Game region for this .XBE. For example: <pre>#define XBEIMAGE_GAME_REGION_NA 0x00000001 #define XBEIMAGE_GAME_REGION_JAPAN 0x00000002 #define XBEIMAGE_GAME_REGION_RESTOFWORLD 0x00000004 #define XBEIMAGE_GAME_REGION_MANUFACTURING 0x80000000</pre>	0x0004	0x00A0
Game Ratings	Game ratings for this .XBE. It is typically safe to set this to 0xFFFFFFFF.	0x0004	0x00A4
Disk Number	Disk Number. Typically zero.	0x0004	0x00A8
Version	Certificate Version.	0x0004	0x00AC
LAN Key	16-byte LAN Key. An unsigned .XBE can just zero these out.	0x0010	0x00B0
Signature Key	16-byte Signature Key. An unsigned .XBE can just zero these out.	0x0010	0x00C0
Alternate Signature Keys	16 x 16-byte Signature Keys. An unsigned .XBE can just zero these out.	0x0100	0x00D0

.XBE Section Header			
Field Name	Description	Size	Offset
Section Flags	Various flags for this .XBE section. Known flags are: <pre>writable = 0x00000001 Preload = 0x00000002 Executable = 0x00000004 Inserted File = 0x00000008 Head Page Read Only = 0x00000010 Tail Page Read Only = 0x00000020</pre>	0x0004	0x0000
Virtual Address	Address of memory to load this section at.	0x0004	0x0004
Virtual Size	Number of bytes in memory to fill with this section.	0x0004	0x0008
Raw Address	File address where this section resides in the .XBE file.	0x0004	0x000C
Raw Size	Number of bytes of this section that exist in the .XBE file.	0x0004	0x0010
Section Name Address	Address to the name for this section, after the .XBE is loaded into memory.	0x0004	0x0014
Section Name Reference Count	It is typically safe to set this to zero.	0x0004	0x0018
Head Shared Page Reference Count Address	It is typically safe to set this to point to a 2-byte WORD in memory with value zero.	0x0004	0x001C
Tail Shared Page Reference Count Address	It is typically safe to set this to point to a 2-byte WORD in memory with value zero.	0x0004	0x0020
Section Digest	20-byte digest for this section. For unsigned .XBE files, it is safe to set this to zeros.	0x0014	0x0024

.XBE Library Version			
Field Name	Description	Size	Offset

Library Name	8-byte name of this library. (i.e. "XAPILIB")	0x0008	0x0000
Major Version	Major version for this library (2-byte WORD).	0x0002	0x0008
Minor Version	Minor version for this library (2-byte WORD).	0x0002	0x000A
Build Version	Build version for this library (2-byte WORD).	0x0002	0x000C
Library Flags	Various flags for this library. The fields are: QFEVersion = 0x1FFF (13-Bit Mask) Approved = 0x6000 (02-Bit Mask) Debug Build = 0x8000 (01-Bit Mask) To see how these can easily be modified, view the C structs included in this document.	0x0004	0x0124

.XBE TLS			
Field Name	Description	Size	Offset
Data Start Address	Address, after the .XBE is loaded into memory, of this .XBE's TLS Data.	0x0004	0x0000
Data End Address	Address, after the .XBE is loaded into memory, of the end of this XBE's TLS Data.	0x0004	0x0004
TLS Index Address	Address, after the .XBE is loaded into memory, of this XBE's TLS Index.	0x0004	0x0008
TLS Callback Address	Address, after the .XBE is loaded into memory, of this XBE's TLS Callback.	0x0004	0x000C
Size of Zero Fill	Size of Zero Fill	0x0004	0x0010
Characteristics	Various TLS characteristics.	0x0004	0x0014

.XBE Logo Bitmap
Encoding Algorithm
<pre> // ***** // * standard typedefs // ***** typedef signed int sint; typedef unsigned int uint; typedef char int08; typedef short int16; typedef long int32; typedef unsigned char uint08; typedef unsigned short uint16; typedef unsigned long uint32; typedef signed char sint08; typedef signed short sint16; typedef signed long sint32; // ***** // * func : import_logo (100x17 8bit grayscale -> LogoBitmap format) // ***** void cxbx_xbe::import_logo(const uint08 x_gray[100*17]) { char *rle = get_logo(); // get raw logo data bytes // calculate maximum bitmap size supported by the existing file uint32 max_size = m_header.m_logo_bitmap_size; while(rle[max_size] == 0) max_size++; if(rle == 0) { if(get_error() == 0) set_error("logo bitmap could not be imported (not enough space in file?)", false); return; } // clear old bitmap data area { for(uint32 x=0;x<max_size;x++) </pre>

```

        rle[x] = 0;
    }

    uint32 offs = 0;

    for(uint32 v=1;v<100*17;offs++)
    {
        char color = x_gray[v] >> 4;

        uint32 len = 1;

        while(++v<100*17-1 && len < 1024 && color == x_gray[v] >> 4)
            len++;

        if(offs >= max_size)
        {
            set_error("not enough room to write logo bitmap", true);
            return;
        }

        logo_rle *cur = (logo_rle *)&rle[offs];

        if(len <= 7)
        {
            cur->m_eight.m_type1 = 1;
            cur->m_eight.m_len = len;
            cur->m_eight.m_data = color;
        }
        else
        {
            cur->m_sixteen.m_type1 = 0;
            cur->m_sixteen.m_type2 = 1;
            cur->m_sixteen.m_len = len;
            cur->m_sixteen.m_data = color;
            offs++;
        }
    }

    m_header.m_logo_bitmap_size = offs;
}

```

Decoding Algorithm

```

// *****
// * standard typedefs
// *****
typedef signed int    sint;
typedef unsigned int  uint;

typedef char          int08;
typedef short         int16;
typedef long          int32;

typedef unsigned char uint08;
typedef unsigned short uint16;
typedef unsigned long  uint32;

typedef signed char   sint08;
typedef signed short  sint16;
typedef signed long   sint32;

// *****
// * func : export_logo (LogoBitmap format -> 100x17 8bit grayscale)
// *****
void xbe::export_logo(uint08 x_gray[100*17])
{
    // in that rare case that we have no logo bitmap, we should
    // just clear it to black.
    memset(x_gray, 0, 100*17);

    uint32 leng = m_header.m_logo_bitmap_size;

    char *rle = get_logo();

    if(rle == 0 || get_error())
        return;

    uint32 o = 0;

    for(uint32 v=0;v<leng;v++)
    {
        uint32 len = 0, data = 0;

        logo_rle *cur = (logo_rle *)&rle[v];
    }
}

```

```

if(cur->m_eight.m_type1)
{
    len    = cur->m_eight.m_len;
    data   = cur->m_eight.m_data;
}
else
{
    if(cur->m_sixteen.m_type2)
    {
        len    = cur->m_sixteen.m_len;
        data   = cur->m_sixteen.m_data;
        v      += 1;
    }
}

for(uint32 j=0;j<len;j++)
{
    o++;

    if(o < 100*17)
        x_gray[o] = (char)(data << 4); // could use (int)(data * 15.0 + .5);
}
}
}

```

.XBE Structures (From CXBX Source)

C++ Source

```

// Note: This source is compatible with MSVC 6.0 or higher, and is public domain.
// *****
// * standard typedefs
// *****
typedef signed int    sint;
typedef unsigned int  uint;

typedef char          int08;
typedef short         int16;
typedef long          int32;

typedef unsigned char uint08;
typedef unsigned short uint16;
typedef unsigned long uint32;

typedef signed char   sint08;
typedef signed short  sint16;
typedef signed long   sint32;

// *****
// * class : xbe
// *****
class xbe
{
public:

    #pragma pack(1)
    struct header
    {
        uint32 m_magic;                // magic number [should be "XBEH"]
        uint08 m_digsig[256];          // digital signature
        uint32 m_base;                 // base address
        uint32 m_sizeof_headers;       // size of headers
        uint32 m_sizeof_image;         // size of image
        uint32 m_sizeof_image_header; // size of image header
        uint32 m_timedate;             // timedate stamp
        uint32 m_certificate_addr;     // certificate address
        uint32 m_sections;             // number of sections
        uint32 m_section_headers_addr; // section headers address

        struct init_flags
        {
            uint m_mount_utility_drive : 1; // mount utility drive flag
            uint m_format_utility_drive : 1; // format utility drive flag
            uint m_limit_64mb           : 1; // limit development kit run time memory to 64mb flag
            uint m_dont_setup_harddisk  : 1; // don't setup hard disk flag
            uint m_unused               : 4; // unused (or unknown)
            uint m_unused_b1            : 8; // unused (or unknown)
            uint m_unused_b2            : 8; // unused (or unknown)
            uint m_unused_b3            : 8; // unused (or unknown)
        }m_init_flags;

        uint32 m_entry;                // entry point address
    }
};

```

```

uint32 m_tls_addr; // thread local storage directory address
uint32 m_pe_stack_commit; // size of stack commit
uint32 m_pe_heap_reserve; // size of heap reserve
uint32 m_pe_heap_commit; // size of heap commit
uint32 m_pe_base_addr; // original base address
uint32 m_pe_sizeof_image; // size of original image
uint32 m_pe_checksum; // original checksum
uint32 m_pe_timedate; // original timedate stamp
uint32 m_debug_pathname_addr; // debug pathname address
uint32 m_debug_filename_addr; // debug filename address
uint32 m_debug_unicode_filename_addr; // debug unicode filename address
uint32 m_kernel_image_thunk_addr; // kernel image thunk address
uint32 m_nonkernel_import_dir_addr; // non kernel import directory address
uint32 m_library_versions; // number of library versions
uint32 m_library_versions_addr; // library versions address
uint32 m_kernel_library_version_addr; // kernel library version address
uint32 m_xapi_library_version_addr; // xapi library version address
uint32 m_logo_bitmap_addr; // logo bitmap address
uint32 m_logo_bitmap_size; // logo bitmap size
}
m_header;

struct certificate
{
    uint32 m_size; // size of certificate
    uint32 m_timedate; // timedate stamp
    uint32 m_titleid; // title id
    uint16 m_title_name[40]; // title name (unicode)
    uint32 m_alt_title_id[0x10]; // alternate title ids
    uint32 m_allowed_media; // allowed media types
    uint32 m_game_region; // game region
    uint32 m_game_ratings; // game ratings
    uint32 m_disk_number; // disk number
    uint32 m_version; // version
    uint08 m_lan_key[16]; // lan key
    uint08 m_sig_key[16]; // signature key
    uint08 m_title_alt_sig_key[16][16]; // alternate signature keys
}
m_certificate;

struct section_header
{
    struct flags // flags
    {
        uint m_writable : 1; // writable flag
        uint m_preload : 1; // preload flag
        uint m_executable : 1; // executable flag
        uint m_inserted_file : 1; // inserted file flag
        uint m_head_page_ro : 1; // head page read only flag
        uint m_tail_page_ro : 1; // tail page read only flag
        uint m_unused_a1 : 1; // unused (or unknown)
        uint m_unused_a2 : 1; // unused (or unknown)
        uint m_unused_b1 : 8; // unused (or unknown)
        uint m_unused_b2 : 8; // unused (or unknown)
        uint m_unused_b3 : 8; // unused (or unknown)
    }m_flags;
    uint32 m_virtual_addr; // virtual address
    uint32 m_virtual_size; // virtual size
    uint32 m_raw_addr; // file offset to raw data
    uint32 m_sizeof_raw; // size of raw data
    uint32 m_section_name_addr; // section name address
    uint32 m_section_reference_count; // section reference count
    uint16 *m_head_shared_ref_count_addr; // head shared page reference count address
    uint16 *m_tail_shared_ref_count_addr; // tail shared page reference count address
    uint08 m_section_digest[20]; // section digest
}
*m_section_header;

struct library_version
{
    char m_name[8]; // library name
    uint16 m_major_version; // major version
    uint16 m_minor_version; // minor version
    uint16 m_build_version; // build version
    struct flags // flags
    {
        uint16 m_qfe_version : 13; // QFE Version
        uint16 m_approved : 2; // Approved? (0:no, 1:possibly, 2:yes)
        uint16 m_debug_build : 1; // Is this a debug build?
    }m_flags;
}
*m_library_version, *m_kernel_version, *m_xapi_version;

struct tls // thread local storage
{
    uint32 m_data_start_addr; // raw start address
    uint32 m_data_end_addr; // raw end address
}

```

```
uint32 m_tls_index_addr;           // tls index address
uint32 m_tls_callback_addr;       // tls callback address
uint32 m_sizeof_zero_fill;        // size of zero fill
uint32 m_characteristics;         // characteristics
}
*m_tls;
}
```