KolibriOS eXecutable (KX) File Format & Loadtime Specification

version 0.0

by Kenshin

February 2021

Contents

Introduction

KolibriOS eXecutable File Format (KX) is an executable file format for KolibriOS operating system, which was developed to be as compact as possible and with capabilities of being improved and extended in future versions. This specification describes and defines structure of KX files and also defines principles of loading and executing such files. Though this document is not approved by any standards organization, any program, application or software component that works with KX files should follow statements of this specification to avoid incompatibility problems and similar issues.

KX file format was made for KolibriOS – a 32-bit operating system for personal computers with x86-compatible processor architecture written in flat assembler, so hereafter x86 and Asm-related terminology will be used. All offsets in this document are in hexadecimal form and are like *+0xnn/+0xnnnnnnnn* and so on. Other hex numbers use the same notation. Binary numbers are postfixed with «b» and look like nnb/nnnnnnnnb, etc. Decimal numbers have no prefix/postfix. The specification uses following data types or data size specifiers or data units:

- **utf8_str** – UTF-8 string
- **utf8z_str** – null-terminated UTF-8 string
- **byte** – unsigned 8-bit integer
- **word** – unsigned 16-bit integer
- **dword** – unsigned 32-bit integer
- **qword** – unsigned 64-bit integer
- **s_byte**, *__s_word__, **s_dword**, **s_qword** – signed byte/word/dword/qword

For all data in a KX header and a subheader following rules apply:

- 16/32/64-bit integers must be in LE (Little Endian) byte order
- strings must be in UTF-8 encoding with valid characters and without BOM (Byte Order Mark)

Other data and strings in KX file may have any encoding, byte order or aligning which software developers find appropriate for use.

KX File Format Specification

Every KX file consists of two or three parts: a header, an optional subheader and code/data. They stored in a file in quite obvious order. The header starts a file, after that (if used) the subheader goes, and then code/data. Simple KolibriOS applications/programs may use only the header, followed by the entry point of code (thus, entry point = 0x00000004) and the rest of code/data. The term ***code/data*** used throughout this document, because the Specification doesn't regulate how portions of code and data must be organized in an KX executable file.

KX Header

The header is what every KX file is started with. It has following structure:

Offset	Size and data type	Field name	Description
+0x00	utf8_str (2 bytes)	kx_magic	KX signature. Must be 0x584B ("KX").
+0x02	1 byte	kx_version	This byte has form XXXXYYYYb, where XXXX bits is major version number, and YYYY is minor version number. Must be 0 for current version.
+0x03	1 byte	kx_flags	Bit flags (see KX Flags section).

The header has size of 4 bytes. If the subheader flag is set, right after it subheader chunks are being stored.

KX Flags

The KX bit flags look like FEDCCBBAb, where:

- A – a subheader flag:
  - A = 0 – no subheader
  - A = 1 – the subheader is used
- BB – a subheader chunks flag:
  - BB = 00 – chunk_type/chunk_data_size is 8-bit
  - BB = 01 – chunk_type/chunk_data_size is 16-bit
  - BB = 10 – chunk_type/chunk_data_size is 32-bit
  - value 11 of BB flag is reserved

    - this flag doesn't have any meaning when A flag is cleared, however if A = 0, BB have to be zeroed too
- CC – common compression flag:
    - CC = 00 – no compression
    - CC = 01 – the kpack compression is used
    - other values of CC flag are reserved
- D – a subheader compression flag:
    - D = 0 – the subheader is not compressed
    - D = 1 – the subheader is compressed with compression defined by the CC flag
    - this flag doesn't have any meaning when A flag is cleared, however if A = 0, D should be zeroed too
    - this flag doesn't have any meaning when CC flag is cleared, however if CC = 00, D flag should be zeroed too
- E – code/data compression flag:
    - E = 0 – code/data are not compressed
    - E = 1 – code/data are compressed with compression defined by the CC flag
    - this flag doesn't have any meaning when CC flag is cleared, however if CC = 00, E flag should be zeroed too
- F – additional memory flag:
    - F = 0 – no additional memory
    - F = 1 – a loader will add 4 KB of memory for uninitialized data after the end of a loaded KX image (see KX Loadtime Specification chapter)

Data Compression

The KX file format supports data compression for a subheader and code/data. The header is never compressed. For current version there is one available compression method – kpack (default KolibriOS compression utility and method that is based on LZMA). Flags CC/D/E are set by a compression utility (not by a compiler/linker, nor manually).

KX Subheader

The subheader is an optional part of a KX file. Actually the subheader is a linear array of chunks. Chunks offer a very flexible way to organize structure of the file. By using them programmers could choose those features of KX files which are necessary. Each chunk has its header (chunk_type + chunk_data_size) and followed by its data. Typical structure for a chunk with an 8-bit header is like this :

Offset	Size and data type	Field name	Description
+0x00	1 byte	chunk_type	Unique type of the chunk.
+0x01	1 byte	chunk_data_size	Chunk data size in bytes.
+0x02	chunk_data_size bytes	chunk_data	Data. Depends on chunk type.

with a 16-bit header:

Offset	Size and data type	Field name	Description
+0x00	1 word	chunk_type	Unique type of the chunk.
+0x02	1 word	chunk_data_size	Chunk data size in bytes.
+0x04	chunk_data_size bytes	chunk_data	Data. Depends on chunk type.

with a 32-bit header:

Offset	Size and data type	Field name	Description
+0x00	1 dword	chunk_type	Unique type of the chunk.
+0x04	1 dword	chunk_data_size	Chunk data size in bytes.
+0x08	chunk_data_size bytes	chunk_data	Data. Depends on chunk type.

For convenience, hereafter chunks with the 8-bit header will be used in the structure descriptions.

Even if the subheader is empty (no chunks were defined), it is terminated by special chunk -1 (type 0xFF/0xFFFF/0xFFFFFFFF). It has only chunk_type field.

Some chunks has their own dependencies (i.e. they depend on other chunks), some shouldn't or must not be used when other are already defined, some could be used regardless of any other chunk. See more details in the specific chunk description.

Chunks

There is a list of available for version 0.0 chunks:

- type 0x00 – default entry point
- type 0x20 – general executable file attributes

- type 0x40 – main thread stack size
- type 0x60 – initial process memory size
- type 0x80 – single dynamic library import table
- type 0xA0 – brief (short) description

Type 0x00 – Default Entry Point

Defines the default entry point of code. The structure for a chunk with an 8-bit header:

Offset	Size and data type	Field name	Description
+0x00	1 byte	chunk_type	Must be 0x00.
+0x01	1 byte	chunk_data_size	Must be 4.
+0x02	1 dword	chunk_data	The offset of an entry point (from beginning of the file).

There must be just one chunk of this type in the subheader, but if it's two of them or more, a KX loader should use a first found chunk and load such a file in spite of the fact that the subheader is not properly formed (but a loader can show a warning – see KX Loadtime Specification chapter).

Type 0x20 – General Executable File Attributes

This chunk defines the essential attributes of a KX file. The structure for a chunk with an 8-bit header:

Offset	Size and data type	Field name	Description
+0x00	1 byte	chunk_type	Must be 0x20.
+0x01	1 byte	chunk_data_size	Must be 12.
+0x02	12 bytes	chunk_data	General executable file attributes structure. See description below.

General executable file attributes:

Offset	Size and data type	Field name	Description
+0x00	1 word	kx_gen_atrbs.xf_type	KX executable file type. For version 0.0 only type 0 is supported – a regular KolibriOS 32-bit application/program.
+0x02	1 word	kx_gen_atrbs.ui_type	Specifies which user interface

			type this app/program uses. See the description below.
+0x04	1 dword	kx_gen_atrbs.xf_flags	Executable file flags (don’t confuse with KX flags). Reserved and must be zeroed.
+0x08	1 dword	kx_gen_atrbs.min_ker_rev	This field fixes minimal SVN revision of KolibriOS kernel that supports this app/program and its features. If it’s needed to run on all kernel versions, must be set to 0.

ui_type can have following values: 0 – undefined UI type, 1 – GUI, 2 – TUI (Text-based User Interface) via X-SHELL interface, other values are reserved.

There must be just one chunk of this type in the subheader, but if it’s two of them or more, a KX loader should use a first found chunk and load such a file in spite of the fact that the subheader is not properly formed (but a loader can show a warning – see KX Loadtime Specification chapter).

Type 0x40 – Main Thread Stack Size

Defines the main thread stack size. The structure for a chunk with an 8-bit header:

Offset	Size and data type	Field name	Description
+0x00	1 byte	chunk_type	Must be 0x40.
+0x01	1 byte	chunk_data_size	Must be 4.
+0x02	1 dword	chunk_data	The size of stack for the main thread of a created process.

There must be just one chunk of this type in the subheader, but if it’s two of them or more, a KX loader should use a first found chunk and load such a file in spite of the fact that the subheader is not properly formed (but a loader can show a warning – see KX Loadtime Specification chapter).

Type 0x60 – Initial Process Memory Size

Defines the initial process memory size. The structure for a chunk with an 8-bit header:

Offset	Size and data type	Field name	Description
+0x00	1 byte	chunk_type	Must be 0x60.
+0x01	1 byte	chunk_data_size	Must be 4.
+0x02	1 dword	chunk_data	The size of memory which is allocated for a created process.

There must be just one chunk of this type in the subheader, but if it’s two of them or more, a KX loader should use a first found chunk and load such a file in spite of the fact that the subheader is not properly formed (but a loader can show a warning – see KX Loadtime Specification chapter).

Type 0x80 – Single Dynamic Library Import Table

Defines an import table for one dynamic library. The structure for a chunk with an 8-bit header:

Offset	Size and data type	Field name	Description
+0x00	1 byte	chunk_type	Must be 0x80.
+0x01	1 byte	chunk_data_size	Variable.
+0x02	n bytes	chunk_data	An import table, see the description below.

An import table for a library has this form:

Offset	Size and data type	Field name	Description
+0x00	1 dword	lib_flags	Library flags. See description below.
+0x04	1 dword	lib_filename_ptr	A pointer to utf8z_str with the filename of a dynamic library. The filename can be just a base name of a file (relative to the working directory) or an absolute filename.
…	…	…	This area is used for aligning and is filled by padding bytes.
…	1 dword	symb_0_name_ptr	Must be aligned on a 4 byte boundary. A pointer to utf8z_str with the name of an imported symbol (e.g. a function name or a public variable name). It’s actually an absolute offset of a string (from

			beginning of the file).
...	1 dword	symb_1_name_ptr	A pointer to utf8z_str with the name of an imported symbol (e.g. a function name or a public variable name). It’s actually an absolute offset of a string (from beginning of the file).
...	...	...	...
...	1 dword	symb_n_name_ptr	A pointer to utf8z_str with the name of an imported symbol (e.g. a function name or a public variable name). It’s actually an absolute offset of a string (from beginning of the file).
...	n bytes	lib_symbs_raw	An array of raw utf8z_str strings with symbol names. Any symb_*_name_ptr must point to a string from this area.

The 4 low bits of library flags look like CCBAb, where:

- A – library loading flag:
  - A = 0 – mandatory library loading (if a library cannot be loaded, the loading of a KX file fails)
  - A = 1 – nonmandatory library loading (if a library cannot be loaded, the loading of a KX file proceeds)
- B – library linking flag
  - B = 0 – mandatory library linking (if some of imported symbols cannot be linked, the loading of a KX file fails)
  - B = 1 – nonmandatory library linking (if some of imported symbols cannot be linked, pointers to linked procedures/data are set to -1 and the loading of a KX file proceeds)
- CC – library loading/linking status flag:
  - this flag is set by a KX loader after trying and loading/linking a dynamic library
  - CC = 00 – a library successfully loaded and linked
  - CC = 01 – a library successfully loaded, but some symbols are not linked
  - CC = 10 – the loading of a library failed

- CC = 11 – the linking of a library failed

Other bits of library flags are reserved and must be cleared.

The amount of chunks of such type is not limited. It’s recommended to use only Latin alphabet letters, digits and underscores in symbol names, though using national characters or any valid Unicode chars is not prohibited at all.

Type 0xA0 – Brief (Short) Description

Defines the brief description of an app/program (for example, "fasm 1.73.27 – flat assembler"). The structure for a chunk with an 8-bit header:

Offset	Size and data type	Field name	Description
+0x00	1 byte	chunk_type	Must be 0xA0.
+0x01	1 byte	chunk_data_size	Variable.
+0x02	1 dword	chunk_data	utf8z_str with a description.

A description string length (including the terminating zero) must be < 256 bytes. The string can contain any valid Unicode characters. Also it’s possible to use CR and LF characters for line breaking.

There must be just one chunk of this type in the subheader, but if it’s two of them or more, the code that is parsing a KX file should use a first found chunk in spite of the fact that the subheader is not properly formed.

KX Loadtime Specification

Any program code that loads a KX file for execution should correspond to loadtime specification details of this chapter.

Loadtime

Before running a KX application/program gets its own execution environment:

- initialized process heap
- stack of size defined by a type 0x40 chunk, else (if that chunk doesn't exist) 4 KB of stack space (initialized with zero bytes)
- ESP initialized with pointer to top of this stack
- sets EIP to the value defined by a type 0x00 chunk, else (if that chunk doesn't exist) EIP = 0x00000004
- EAX = pointer to PRDB (Pre-Runtime Data Block)
- EBX, ECX, EDX, ESI, EDI and EBP are cleared
- CF, PF, AF, ZF, SF, DF and OF of EFLAGS are cleared, other flags are undefined
- segment registers are undefined
- a KX image is loaded at 0x00000000 address, if it was packed with kpack, data is uncompressed

PRDB (Pre-Runtime Data Block)

PRDB has following structure:

Offset	Size and data type	Field name	Description
+0x000	1 dword	prdb_type	Must be 0 for currrent version.
+0x004	1 dword	prdb_size	Total size of a PRDB in bytes. For current KX version it's 32.
+0x008	1 dword	prdb_pid	PID of current process.
+0x00C	1 dword	prdb_ppid	Parent PID (PID of a loader).
+0x010	1 dword	reserved	Reserved. Must be 0.
+0x014	1 dword	reserved	Reserved. Must be 0.

+0x018	1 dword	prdb_filename_ptr	Pointer to a UTF8Z string with a filename of KX file.
+0x01C	1 dword	prdb_params_ptr	Pointer to a UTF8Z string with command line parameters.