Appendix K
Segmented (New) .EXE File Header Format

Microsoft Windows requires much more information about a program than is available in the format of the .EXE executable file supported by MS-DOS. For example, Windows needs to identify the various segments of a program as code segments or data segments, to identify exported and imported functions, and to store the program's resources (such as icons, cursors, menus, and dialog-box templates). Windows must also support dynamically linkable library modules containing routines that programs and other library modules can call. For this reason, Windows programs use an expanded .EXE header format called the New Executable file header format. This format is used for Windows programs, Windows library modules, and resource-only files such as the Windows font resource files.

The Old Executable Header

The New Executable file header format incorporates the existing MS-DOS executable file header format. In fact, the beginning of a New Executable file is simply a normal MS-DOS .EXE header. The 4 bytes at offset 3CH are a pointer to the beginning of the New Executable header. (Offsets are from the beginning of the Old Executable header.)

Offset	Length (bytes)	Contents
00H	1	Signature byte *M*
01H	1	Signature byte *Z*
3CH	4	Offset of New Executable header from beginning of file

This normal MS-DOS .EXE header can contain size and relocation information for a non-Windows MS-DOS program that is contained within the .EXE file along with the Windows program. This program is run when the .EXE file is executed from the MS-DOS command line. Most Windows programmers use a standard program that simply prints the message *This program requires Microsoft Windows.*

The New Executable Header

The beginning of the New Executable file header contains information about the location and size of various tables within the header. (Offsets are from the beginning of the New Executable header.)

Offset	Length (bytes)	Contents
00H	1	Signature byte *N*
01H	1	Signature byte *E*
02H	1	LINK version number
03H	1	LINK revision number
04H	2	Offset of beginning of entry table relative to beginning of New Executable header
06H	2	Length of entry table
08H	4	32-bit checksum of entire contents of file, using zero for these 4 bytes
0CH	2	Module flag word (*see* below)
0EH	2	Segment number of automatic data segment (0 if neither SINGLEDATA nor MULTIPLEDATA flag is set in flag word)
10H	2	Initial size of local heap to be added to automatic data segment (0 if there is no local heap)
12H	2	Initial size of stack to be added to automatic data segment (0 for library modules)
14H	2	Initial value of instruction pointer (IP) register on entry to program
16H	2	Initial segment number for setting code segment (CS) register on entry to program
18H	2	Initial value of stack pointer (SP) register on entry to program (0 if stack segment is automatic data segment; stack should be set above static data area and below local heap in automatic data segment)

(more)

Offset	Length (bytes)	Contents
1AH	2	Segment number for setting stack segment (SS) register on entry to program (0 for library modules)
1CH	2	Number of entries in segment table
1EH	2	Number of entries in module reference table
20H	2	Number of bytes in nonresident names table
22H	2	Offset of beginning of segment table relative to beginning of New Executable header
24H	2	Offset of beginning of resource table relative to beginning of New Executable header
26H	2	Offset of beginning of resident names table relative to beginning of New Executable header
28H	2	Offset of beginning of module reference table relative to beginning of New Executable header
2AH	2	Offset of beginning of imported names table relative to beginning of New Executable header
2CH	4	Offset of nonresident names table relative to beginning of file
30H	2	Number of movable entry points listed in entry table
32H	2	Alignment shift count (0 is equivalent to 9)
34H	12	Reserved for expansion

The module flag word at offset 0CH in the New Executable header is defined as shown in Figure K-1.

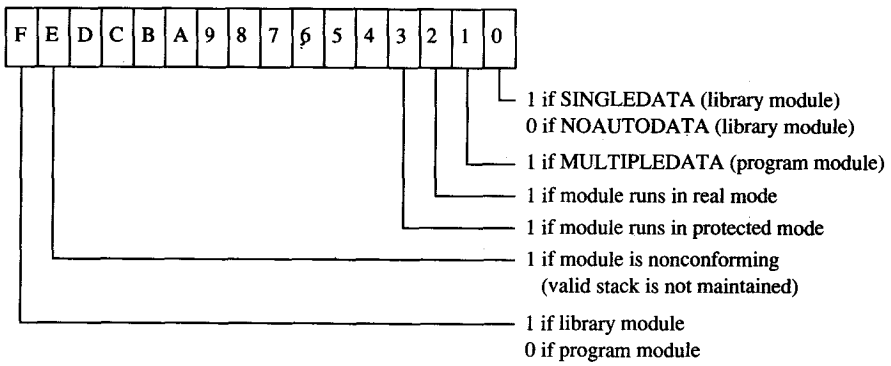

Figure K-1. The module flag word.

The segment table

This table contains one 8-byte record for every code and data segment in the program or library module. Each segment has an ordinal number associated with it. For example, the first segment has an ordinal number of 1. These segment numbers are used to reference the segments in other sections of the New Executable file. (Offsets are from the beginning of the record.)

Offset	Length (bytes)	Contents
00H	2	Offset of segment relative to beginning of file after shifting value left by alignment shift count
02H	2	Length of segment (0000H for segment of 65536 bytes)
04H	2	Segment flag word (*see* below)
06H	2	Minimum allocation size for segment; that is, amount of space Windows reserves in memory for segment (0000H for minimum allocation size of 65536 bytes)

The segment flag word is defined as shown in Figure K-2.

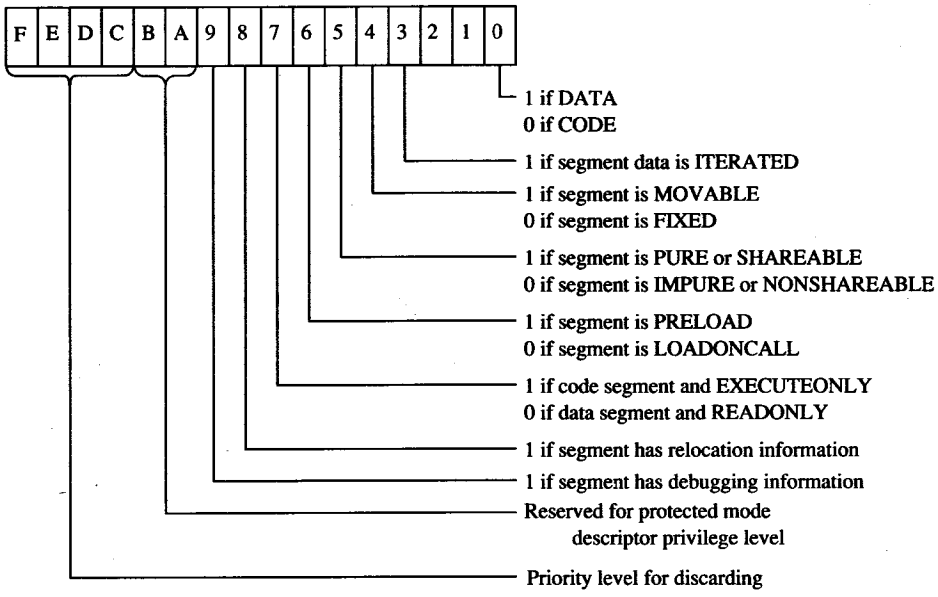

Figure K-2. The segment flag word.

The resource table

Resources are segments that contain data but are not included in a program's normal data segments. Resources are commonly used in Windows programs to store menus, dialog-box templates, icons, cursors, and text strings, but they can also be used for any type of read-only data. Each resource has a type and a name, both of which can be represented by either a number or an ASCII name.

The resource table begins with a resource shift count used for adjusting other values in the table. (Offsets are from the beginning of the table.)

Offset	Length (bytes)	Contents
00H	2	Resource shift count

This is followed by one or more resource groups, each defining one or more resources. (Offsets are from the beginning of the group.)

Offset	Length (bytes)	Contents
00H	2	Resource type (0 if end of table) If high bit set, type represented by predetermined number (high bit not shown): 1 Cursor 2 Bitmap 3 Icon 4 Menu template 5 Dialog-box template 6 String table 7 Font directory 8 Font 9 Keyboard-accelerator table If high bit not set, type is ASCII text string and this value is offset from beginning of resource table, pointing to 1-byte value with number of bytes in string followed by string itself.
02H	2	Number of resources of this type
04H	4	Reserved for run-time use
08H	12 each	Resource description

Each resource description requires 12 bytes. (Offsets are from the beginning of the description.)

Offset	Length (bytes)	Contents
00H	2	Offset of resource relative to beginning of file after shifting left by resource shift count
02H	2	Length of resource after shifting left by resource shift count
04H	2	Resource flag word (*see* below)
06H	2	Resource name If high bit set, represented by a number; otherwise, type is ASCII text string and this value is offset from beginning of resource table, pointing to 1-byte value with number of bytes in string followed by string itself.
08H	4	Reserved for run-time use

The resource flag word is defined as shown in Figure K-3.

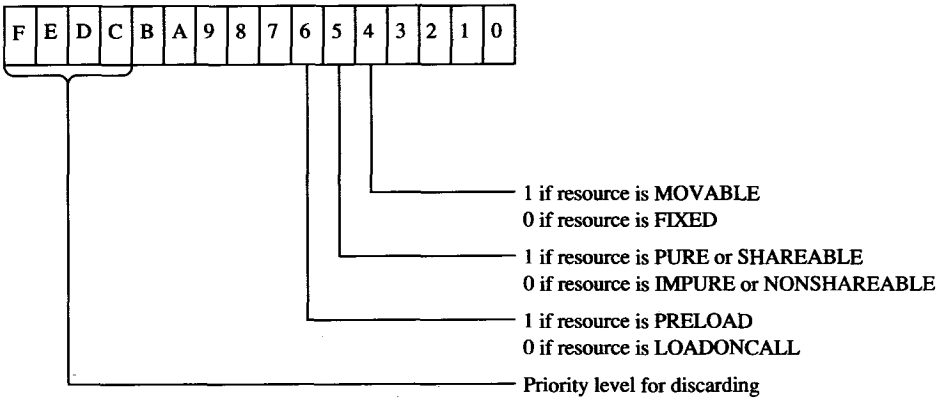

Figure K-3. The resource flag word.

The resident names table

This table contains a list of ASCII strings. The first string is the module name given in the module definition file. The other strings are the names of all exported functions listed in the module definition file that were not given explicit ordinal numbers or that were explicitly specified in the file as resident names. (Exported functions with explicit ordinal numbers in the module definition file are listed in the nonresident names table.)

Each string is prefaced by a single byte indicating the number of characters in the string and is followed by a word (2 bytes) referencing an element in the entry table, beginning at 1. The word that follows the module name is 0. (Offsets are from the beginning of the record.)

Offset	Length (bytes)	Contents
00H	1	Number of bytes in string (0 if end of table)
01H	*n*	ASCII string, not null-terminated
n+1	2	Index into entry table

The module reference table

The module reference table contains 2 bytes for every external module the program uses. These 2 bytes are an offset into the imported names table.

The imported names table

The imported names table contains a list of ASCII strings. These strings are the names of all other modules that are referenced through imported functions. The strings are prefaced with a single byte indicating the length of the string.

For most Windows programs, the imported names table includes KERNEL, USER, and GDI, but it can also include names of other modules, such as KEYBOARD and SOUND. (Offsets are from the beginning of the record.)

Offset	Length (bytes)	Contents
00H	1	Number of bytes in name string
01H	*n*	ASCII name string, not null-terminated

These strings do not necessarily start at the beginning of the imported names table; the names are referenced by offsets specified in the module reference table.

The entry table

This table contains one member for every entry point in the program or library module. (Every public FAR function or procedure in a module is an entry point.) The members in the entry table have ordinal numbers beginning at 1. These ordinal numbers are referenced by the resident names table and the nonresident names table.

LINK versions 4.0 and later bundle the members of the entry table. Each bundle begins with the following information. (Offsets are from the beginning of the bundle.)

Offset	Length (bytes)	Contents
00H	1	Number of entry points in bundle (0 if end of table)
01H	1	Segment number of entry points if entry points in bundle are in single fixed segment; 0FFH if entry points in bundle are in movable segments

For a bundle containing entry points in fixed segments, each entry point requires 3 bytes. (Offsets are from the beginning of the entry description.)

Offset	Length (bytes)	Contents
00H	1	Entry-point flag byte (*see* below)
01H	2	Offset of entry point in segment

For bundles containing entry points in movable segments, each entry point requires 6 bytes. (Offsets are from the beginning of the entry description.)

Offset	Length (bytes)	Contents
00H	1	Entry-point flag byte (*see* below)
01H	2	Interrupt 3FH instruction: CDH 3FH
03H	1	Segment number of entry point
04H	2	Offset of entry-point segment

The entry-point flag byte is defined as shown in Figure K-4.

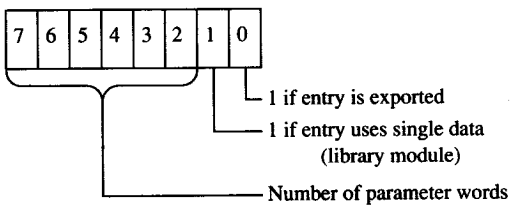

Figure K-4. The entry-point flag.

The nonresident names table

This table contains a list of ASCII strings. The first string is the module description from the module definition file. The other strings are the names of all exported functions listed in the module definition file that have ordinal numbers associated with them. (Exported functions without ordinal numbers in the module definition file are listed in the resident names table.)

Each string is prefaced by a single byte indicating the number of characters in the string and is followed by a word (2 bytes) referencing a member of the entry table, beginning at 1. The word that follows the module description string is 0. (Offsets are from the beginning of the table.)

Offset	Length (bytes)	Contents
00H	1	Number of bytes in string (0 if end of table)
01H	*n*	ASCII string, not null-terminated
n+1	2	Index into entry table

The code and data segment

Following the various tables in the New Executable file header are the code and data segments of the program or library module.

If the code or data segment is flagged in the segment flag word as ITERATED, the segment is organized as follows. (Offsets are from the beginning of the segment.)

Offset	Length (bytes)	Contents
00H	2	Number of iterations of data
02H	2	Number of bytes of data
04H	*n*	Data

Otherwise, the size of the segment data is given by the length of the segment field in the segment table.

If the segment is flagged in the segment flag word as containing relocation information, then the relocation table begins immediately after the segment data. Windows uses the relocation table to resolve references within the segments to functions in other segments in the same module and to imported functions in other modules. (Offsets are from the beginning of the table.)

Offset	Length (bytes)	Contents
00H	2	Number of relocation items

Each relocation item requires 8 bytes. (Offsets are from the beginning of the relocation item.)

Offset	Length (bytes)	Contents
00H	1	Type of address to insert in segment: 01H Offset only 02H Segment only 03H Segment and offset

(more)

Offset	Length (bytes)	Contents
01H	1	Relocation type: 00H Internal reference 01H Imported ordinal 02H Imported name If bit 2 set, relocation type is additive (*see* below)
02H	2	Offset of relocation item within segment

The next 4 bytes depend on the relocation type. If the relocation type is an internal reference to a segment in the same module, these bytes are defined as follows. (Offsets are from the beginning of the relocation item.)

Offset	Length (bytes)	Contents
04H	1	Segment number for fixed segment; 0FFH for movable segment
05H	1	0
06H	2	If MOVABLE segment, ordinal number referenced in entry table; if FIXED segment, offset into segment

If the relocation type is an imported ordinal to another module, then these bytes are defined as follows. (Offsets are from the beginning of the relocation item.)

Offset	Length (bytes)	Contents
04H	2	Index into module reference table
06H	2	Function ordinal number

Finally, if the relocation type is an imported name of a function in another module, these bytes are defined as follows. (Offsets are from the beginning of the relocation item.)

Offset	Length (bytes)	Contents
04H	2	Index into module reference table
06H	2	Offset within imported names table to name of imported function

If the ADDITIVE flag of the relocation type is set, the address of the external function is added to the contents of the address in the target segment. If the ADDITIVE flag is not set, then the target contains an offset to another target within the same segment that requires the same relocation address. This defines a chain of target addresses that get the same address. The chain is terminated with a –1 entry.

Charles Petzold