

Windows 3.0 16-bit NE Executable Format

INF: Executable-File Header Format

[P_WinSDK]

3.00

WINDoWS

PSSoNLY | Windows 3 Developers Notes softlib ENDUSER

Summary:

Note: This article is part of a set of seven articles, collectively called the "Windows 3.00 Developer's Notes." More information about the contents of the other articles, and procedures for ordering a hard-copy set, can be found in the knowledge base article titled "INF: The Windows 3.00 Developer's Notes" (Q65260).

This article can be found in the Software/Data Library by searching on the word EXEFMT or S12688. EXEFMT was archived using the PKware file-compression utility.

More Information:

Microsoft defined the segmented executable file format for Windows applications and dynamic-link libraries (DLLs). This file format is also referred to as the New Executable Format. This new format is an extension of the existing MS-DOS .EXE format (old-style format). The purpose of the segmented executable format is to provide the information needed to support the dynamic linking and segmentation capabilities of the Windows environment.

An executable file contains Microsoft Windows code and data, or Windows code, data, and resources. Specific fields have been added to the old-style .EXE format header to indicate the existence of the segmented file format. The old-style header may contain a valid executable program, called a stub program, that will be executed if the program is run on MS-DOS (without Windows). This stub program usually prints a message indicating that Microsoft Windows is required to run the program. The segmented executable format extensions also begin with a header that describes the contents and location of the executable image in the file. The loader uses this header information when it loads the executable segments in memory.

oLD-STYLE HEADER EXTENSIONs

The old-style header contains information the loader expects for a DOS executable file. It describes a stub program (WINSTUB) the loader can place in memory when necessary, it points to the new-style header, and it contains the stub programs relocation table.

The following illustrates the distinct parts of the old-style executable format:

+-----+ +		
00h	old-style header info	
+-----+ +		
20h	Reserved	
+-----+ +		
3Ch	offset to segmented	
	.EXE header	
+-----+ +		
40h	Relocation table and	
	DoS stub program	

```

+-----+
| Segmented .EXE Header |
|       .               |
|       .               |
|       .               |
+-----+

```

The word at offset 18h in the old-style .EXE header contains the relative byte offset to the stub program's relocation table. If this offset is 40h, then the double word at offset 3Ch is assumed to be the relative byte offset from the beginning of the file to the beginning of the segmented executable header. A new-format .EXE file is identified if the segmented executable header contains a valid signature. If the signature is not valid, the file is assumed to be an old-style format .EXE file. The remainder of the old-style format header will describe a DoS program, the stub. The stub may be any valid program but will typically be a program that displays an error message.

```

=====
                        SEGMENTED EXE FORMAT
=====

```

Because Windows executable files are often larger than one segment (64K), additional information (that does not appear in the old-style header) is required so that the loader can load each segment properly. The segmented EXE format was developed to provide the loader with this information.

The segmented .EXE file has the following format:

```

+-----+
00h | old-style EXE |
|   Header       |
+-----+
20h | Reserved    |
+-----+
3Ch | offset to   | --- +
| Segmented Header|   |
+-----+
40h | Relocation Table|   |
| & Stub Program |   |
+-----+
|               |   |
+-----+
xxh | Segmented EXE | <--+
|   Header       |   |
+-----+
| Segment Table  |   |
+-----+
| Resource Table |   |
+-----+
| Resident Name  |   |
|   Table       |   |
+-----+
| Module Reference|   |
|   Table       |   |
+-----+
| Imported Names |   |
|   Table       |   |
+-----+
| Entry Table    |   |
+-----+
| Non-Resident   |   |
|   Name Table   |   |
+-----+

```

```

|   Seg #1 Data   |
|   Seg #1 Info   |
+-----+
      .
      .
      .
+-----+
|   Seg #n Data   |
|   Seg #n Info   |
+-----+

```

The following sections describe each of the components that make up the segmented EXE format. Each section contains a description of the component and the fields in the structures that make up that component.

Note: All unused fields and flag bits are reserved for future use and must contain 0 (zero) values.

```

=====
                        SEGMENTED EXE HEADER
=====

```

The segmented EXE header contains general information about the EXE file and contains information on the location and size of the other sections. The Windows loader copies this section, along with other data, into the module table in the system data. The module table is internal data used by the loader to manage the loaded executable modules in the system and to support dynamic linking.

The following describes the format of the segmented executable header. For each field, the offset is given relative to the beginning of the segmented header, the size of the field is defined, and a description is given.

offset	Size	Description
--------	------	-------------

00h	DW	Signature word. "N" is low-order byte. "E" is high-order byte.
02h	DB	Version number of the linker.
03h	DB	Revision number of the linker.

04h	DW	Entry Table file offset, relative to the beginning of
06h	DW	the segmented EXE header. Number of bytes in the entry table.
08h	DD	32-bit CRC of entire contents of file. These words are taken as 00 during the calculation.
0Ch	DW	Flag word.

0000h = NoAUToDATA

0001h = SINGLEDATA (Shared automatic data segment)

0002h = MULTIPLEDATA (Instanced automatic data segment)

2000h = Errors detected at link time, module will not load.

8000h = Library module.

The SS:SP information is invalid, CS:IP points to an initialization procedure that is called with AX equal to the module handle. This initialization procedure must perform a far return to the caller, with AX not equal to zero to indicate success, or AX equal to zero to indicate failure to initialize. DS is set to the library's data segment if the SINGLEDATA flag is set. otherwise, DS is set to the caller's data segment.

A program or DLL can only contain dynamic links to executable files that have this library module flag set. one program cannot dynamic-link to another program.

0Eh DW Segment number of automatic data segment.
This value is set to zero if SINGLEDATA and MULTIPLEDATA flag bits are clear, NoAUToDATA is indicated in the flags word.

A Segment number is an index into the module's segment table. The first entry in the segment table is segment number 1.

10h DW Initial size, in bytes, of dynamic heap added to the

		data segment. This value is zero if no initial local heap is allocated.
12h	DW	Initial size, in bytes, of stack added to the data segment. This value is zero to indicate no initial stack allocation, or when SS is not equal to DS.
14h	DD	Segment number:offset of CS:IP.
18h	DD	Segment number:offset of SS:SP. If SS equals the automatic data segment and SP equals zero, the stack pointer is set to the top of the automatic data segment just below the additional heap area.

```

+ ----- +
| additional dynamic heap |
+-----+ <- SP
|      additional stack      |
+ ----- +
| loaded auto data segment |
+-----+ <- DS, SS

```

1Ch	DW	Number	of	entries in the Segment Table.
1Eh	DW	Number	of	entries in the Module Reference Table.
20h	DW	Number	of	bytes in the Non-Resident Name Table.
22h	DW	Segment Table file offset, relative to the beginning of the segmented EXE header.		
24h	DW	Resource Table file offset, relative to the beginning of the segmented EXE header.		
26h	DW	Resident Name Table file offset, relative to the beginning of the segmented EXE header.		
28h	DW	Module Reference Table file offset, relative to the beginning of the segmented EXE header.		
2Ah	DW	Imported Names Table file offset, relative to the beginning of the segmented EXE header.		
2Ch	DD	Non-Resident Name Table offset, relative to the beginning of the file.		
30h	DW	Number of movable entries in the Entry Table.		
32h	DW	Logical sector alignment shift count, log(base 2) of the segment sector size (default 9).		
34h	DW	Number of resource entries.		
36h	DB	Executable type, used by loader.		

02h = WINDOWS

37h-3Fh DB Reserved, currently 0's.

=====

SEGMENT TABLE

=====

The segment table contains an entry for each segment in the executable file. The number of segment table entries are defined in the segmented EXE header. The first entry in the segment table is segment number 1. The following is the structure of a segment table entry.

Size Description

DW Logical-sector offset (n byte) to the contents of the segment data, relative to the beginning of the file. Zero means no file data.

DW Length of the segment in the file, in bytes. Zero means 64K.

DW Flag word.

0007h = TYPE_MASK Segment-type field.
0000h = CODE Code-segment type.
0001h = DATA Data-segment type.
0010h = MOVEABLE Segment is not fixed.
0040h = PRELOAD Segment will be preloaded; read-only if this is a data segment.

0100h = RELOCINFO Set if segment has relocation records.
F000h = DISCARD Discard priority.

DW Minimum allocation size of the segment, in bytes. Total size of the segment. Zero means 64K.

=====

RESOURCE TABLE

=====

The resource table follows the segment table and contains entries for each resource in the executable file. The resource table consists of an alignment shift count, followed by a table of resource records. The resource records define the type ID for a set of resources. Each resource record contains a table of resource entries of the defined type. The resource entry defines the resource ID or name ID for the resource. It also defines the location and size of the resource. The following describes the contents of each of these structures:

Size Description

DW Alignment shift count for resource data.

A table of resource type information blocks follows. The following is the format of each type information block:

DW Type ID. This is an integer type if the high-order bit is set (8000h); otherwise, it is an offset to the type string, the offset is relative to the beginning of the resource table. A zero type ID marks the end of the resource type information blocks.

DW Number of resources for this type.

DD Reserved.

A table of resources for this type follows. The following is the format of each resource (8 bytes each):

DW File offset to the contents of the resource data, relative to beginning of file. The offset is in terms of the alignment shift count value specified at beginning of the resource table.

DW Length of the resource in the file (in bytes).

DW Flag word.

0010h = MOVEABLE Resource is not fixed.
0020h = PURE Resource can be shared.
0040h = PRELOAD Resource is preloaded.

DW Resource ID. This is an integer type if the high-order

bit is set (8000h), otherwise it is the offset to the resource string, the offset is relative to the beginning of the resource table.

DD Reserved.

Resource type and name strings are stored at the end of the resource table. Note that these strings are NoT null terminated and are case sensitive.

DB Length of the type or name string that follows. A zero value indicates the end of the resource type and name string, also the end of the resource table.

DB ASCII text of the type or name string.

RESIDENT-NAME TABLE

The resident-name table follows the resource table, and contains this module's name string and resident exported procedure name strings. The first string in this table is this module's name. These name strings are case-sensitive and are not null-terminated. The following describes the format of the name strings:

Size Description

DB Length of the name string that follows. A zero value indicates the end of the name table.

DB ASCII text of the name string.

DW ordinal number (index into entry table). This value is ignored for the module name.

MODULE-REFERENCE TABLE

The module-reference table follows the resident-name table. Each entry contains an offset for the module-name string within the imported-names table; each entry is 2 bytes long.

Size Description

DW offset within Imported Names Table to referenced module name string.

IMPORTED-NAME TABLE

The imported-name table follows the module-reference table. This table contains the names of modules and procedures that are imported by the

executable file. Each entry is composed of a 1-byte field that contains the length of the string, followed by any number of characters. The strings are not null-terminated and are case sensitive.

Size Description

DB Length of the name string that follows.

DB ASCII text of the name string.

```
=====
                        ENTRY TABLE
=====
```

The entry table follows the imported-name table. This table contains bundles of entry-point definitions. Bundling is done to save space in the entry table. The entry table is accessed by an ordinal value. ordinal number one is defined to index the first entry in the entry table. To find an entry point, the bundles are scanned searching for a specific entry point using an ordinal number. The ordinal number is adjusted as each bundle is checked. When the bundle that contains the entry point is found, the ordinal number is multiplied by the size of the bundle's entries to index the proper entry.

The linker forms bundles in the most dense manner it can, under the restriction that it cannot reorder entry points to improve bundling. The reason for this restriction is that other .EXE files may refer to entry points within this bundle by their ordinal number. The following describes the format of the entry table bundles.

Size Description

DB Number of entries in this bundle. All records in one bundle are either moveable or refer to the same fixed segment. A zero value in this field indicates the end of the entry table.

DB Segment indicator for this bundle. This defines the type of entry table entry data within the bundle. There are three types of entries that are defined.

000h = Unused entries. There is no entry data in an unused bundle. The next bundle follows this field. This is used by the linker to skip ordinal numbers.

001h-0FEh = Segment number for fixed segment entries. A fixed segment entry is 3 bytes long and has the following format.

DB Flag word.

01h = Set if the entry is exported.

02h = Set if the entry uses a global (shared) data segments.

The first assembly-language instruction in the entry point prologue must be "MOV AX,data segment number". This may be set only for SINGLEDATA library modules.

DW offset within segment to entry point.

0FFh = Moveable segment entries. The entry data contains the segment number for the entry points. A moveable segment entry is 6 bytes long and has the following format.

DB Flag word.

01h = Set if the entry is exported.

02h = Set if the entry uses a global (shared) data segments.

INT 3FH.

DB Segment number.

DW offset within segment to entry point.

NONRESIDENT-NAME TABLE

The nonresident-name table follows the entry table, and contains a module description and nonresident exported procedure name strings. The first string in this table is a module description. These name strings are case-sensitive and are not null-terminated. The name strings follow the same format as those defined in the resident name table.

PER SEGMENT DATA

The location and size of the per-segment data is defined in the segment table entry for the segment. If the segment has relocation fixups, as defined in the segment table entry flags, they directly follow the segment data in the file. The relocation fixup information is defined as follows:

Size Description

DW Number of relocation records that follow.

A table of relocation records follows. The following is the format of each relocation record.

DB Source type.

0Fh = SOURCE_MASK

00h = LoBYTE

02h = SEGMENT

03h = FAR_ADDR (32-bit pointer)

05h = OFFSET (16-bit offset)

DB Flags byte.

03h = TARGET_MASK

00h = INTERNALREF

01h = IMPORTORDINAL

02h = IMPORTNAME

03h = OSFIXUP

04h = ADDITIVE

DW offset within this segment of the source chain.
If the ADDITIVE flag is set, then target value is added to the source contents, instead of replacing the source and following the chain. The source chain is an 0FFFFh terminated linked list within this segment of all references to the target.

The target value has four types that are defined in the flag byte field. The following are the formats for each target type:

INTERNALREF

DB Segment number for a fixed segment, or 0FFh for a movable segment.

DB 0

DW offset into segment if fixed segment, or ordinal number index into Entry Table if movable segment.

IMPoRTNAME

DW Index into module reference table for the imported module.

DW offset within Imported Names Table to procedure name string.

IMPoRToRDINAL

DW Index into module reference table for the imported module.

DW Procedure ordinal number.

oSFIXUP

DW operating system fixup type.

Floating-point fixups.

0001h = FIARQQ, FJARQQ

0002h = FISRQQ, FJSRQQ

0003h = FICRQQ, FJCRQQ

0004h = FIERQQ

0005h = FIDRQQ

0006h = FIWRQQ

DW 0

=====

Microsoft is a registered trademark and Windows is a trademark of Microsoft Corporation.

Additional reference words: 3.0