

Offset	Size	Name	Description
00h	WORD	e_magic	0x4d, 0x5a. This is the “magic number” of an EXE file. The first byte of the file is 0x4d and the second is 0x5a.
02h	WORD	e_cblk	The number of bytes in the last block of the program that are actually used. If this value is zero, that means the entire last block is used (i.e. the effective value is 512).
04h	WORD	e_cp	Number of blocks in the file that are part of the EXE file. If [02-03] is non-zero, only that much of the last block is used.
06h	WORD	e_crlc	Number of relocation entries stored after the header. May be zero.
08h	WORD	e_cparhdr	Number of paragraphs in the header. The program's data begins just after the header, and this field can be used to calculate the appropriate file offset. The header includes the relocation entries. Note that some OSs and/or programs may fail if the header is not a multiple of 512 bytes.
0Ah	WORD	e_minalloc	Number of paragraphs of additional memory that the program will need. This is the equivalent of the BSS size in a Unix program. The program can't be loaded if there isn't at least this much memory available to it.
0Ch	WORD	e_maxalloc	Maximum number of paragraphs of additional memory. Normally, the OS reserves all the remaining conventional memory for your program, but you can limit it with this field.
0EH	WORD	e_ss	Relative value of the stack segment. This value is added to the segment the program was loaded at, and the result is used to initialize the SS register.
10h	WORD	e_sp	Initial value of the SP register.
12h	WORD	e_csum	Word checksum. If set properly, the 16-bit sum of all words in the file should be zero. Usually, this isn't filled in.
14h	WORD	e_ip	Initial value of the IP register.
16h	WORD	e_cs	Initial value of the CS register, relative to the segment the program was loaded at.
18h	WORD	e_lfarlc	Offset of the first relocation item in the file.
1Ah	WORD	e_ovno	Overlay number. Normally zero, meaning that it's the main program.
1Ch	WORD	e_res[ERES1WDS]	

/* In-disk and In-memory module structure. See 'Windows Internals' p. 219 */

```
struct new_exe {
```

```
    WORD ne_magic;                                /* Signature word EMAGIC */
union {
    struct {
        BYTE ne_ver;                            /* Version number of the linker */
        BYTE ne_rev;                            /* Revision number of the linker */
    };
    WORD count;                               /* Usage count (ne_ver/ne_rev on disk) */
};
WORD ne_enttab;                                /* Entry Table file offset, relative to the
beginning of                                         the segmented EXE header */
union {
```

```
WORD ne_cbenttab;           /* Number of bytes in the entry table */
WORD next;                  /* Selector to next module */
};

union {
    DWORD          ne_crc;   /* 32-bit CRC of entire contents of file.
                                These words are taken as 00 during the
                                calculation */
    struct {
        WORD      dgroup_entry; /* Near ptr to segment entry for DGROUP */
        WORD      fileinfo;     /* Near ptr to file info (OFSTRUCT) */
    };
};

WORD ne_flags;              /* Flag word */
WORD ne_autodata;           /* Segment number of automatic data
                                segment.
                                This value is set to zero if SINGLEDATA
                                and
                                NOAUTODATA is
                                indicated in the flags word.
```

module's segment table is segment

WORD ne_heap; /* Initial size, in bytes, of dynamic
 data segment. This value is zero if no
 heap is allocated */
heap added to the initial local

WORD ne_stack; /* Initial size, in bytes, of stack
 segment. This value is zero to indicate
 stack allocation, or when SS is not equal
 to DS */
added to the no initial to DS */

DWORD ne_csip; /* Segment number:offset of CS:IP */
DWORD ne_sssp; /* Segment number:offset of SS:SP.
 If SS equals the automatic data segment
 zero, the stack pointer is set to the top
 additional heap area.

```
+-----+
| additional dynamic heap |
+-----+ <- SP
```

```

        |   additional stack   |
+-----+
        | loaded auto data segment |
+-----+ <- DS, SS */
        /* Number of entries in the Segment

WORD ne_cseg;
Table */
        WORD ne_cmod;           /* Number of entries in the Module
Reference Table */

WORD ne_cbnrestab;          /* Number of bytes in the Non-Resident
Name Table */
        WORD ne_segtab;         /* Segment Table file offset, relative to
the beginning
                                of the segmented EXE header */

WORD ne_rsrctab;            /* Resource Table file offset, relative to
the beginning
                                of the segmented EXE header */

WORD ne_restab;             /* Resident Name Table file offset, relative
to the
                                beginning of the segmented EXE header */
        WORD ne_modtab;          /* Module Reference Table file offset,
relative to the
                                beginning of the segmented EXE header */

WORD ne_imptab;              /* Imported Names Table file offset,
relative to the
                                beginning of the segmented EXE header */

DWORD ne_nrestab;            /* Non-Resident Name Table offset,
relative to the
                                beginning of the file */
        WORD ne_cmovent;
Table */
        WORD ne_align;           /* Number of movable entries in the Entry
log(base 2) of
                                the segment sector size (default 9) */
        WORD ne_cres;             /* Number of resource entries */
        BYTE ne_exetyp;           /* Executable type, used by loader.
02h = WINDOWS */
        BYTE ne_flagsothers;       /* Operating system flags */
        char ne_res[NERESBYTES];    /* Reserved */

```

};

On-disk segment entry struct new_seg { WORD ns_sector; / Logical-sector offset (n byte) to the contents of the segment data, relative to the beginning of the file. Zero means no file data */ WORD ns_cbseg; /* Length of the segment in the file, in bytes. Zero means 64K */ WORD ns_flags; /* Flag word */ WORD ns_minalloc; /* Minimum allocation size of the segment, in bytes. Total size of the segment. Zero means 64K */ }; In-memory segment entry struct new_seg1 {*

```

WORD ns1_sector;           /* Logical-sector offset (n byte) to the
contents of the segment
                                data, relative to the beginning of the
file. Zero means no

```

```
        file data */
WORD ns1_cbseg;           /* Length of the segment in the file, in
bytes. Zero means 64K */
WORD ns1_flags;           /* Flag word */
WORD ns1_minalloc;        /* Minimum allocation size of the
segment, in bytes. Total size
of the segment. Zero means 64K */
WORD ns1_handle;          /* Selector or handle (selector - 1) of
segment in memory */
```

};

struct new_segdata {

```
union {
    struct {
        WORD      ns_niter;
        WORD      ns_nbytes;
        char     ns_iterdata;
    } ns_iter;
    struct {
        char     ns_data;
    } ns_noniter;
} ns_union;
```

};

struct new_rlcinfo {

```
WORD nr_nreloc;
```

};

struct new_rlc {

```
char      nr_stype;
char      nr_flags;
WORD    nr_soff;
union {
    struct {
        char      nr_segno;
        char      nr_res;
        WORD    nr_entry;
    } nr_intref;
    struct {
        WORD    nr_mod;
        WORD    nr_proc;
    } nr_import;
    struct {
        WORD    nr_ostype;
        WORD    nr_osres;
```

```

        } nr_osfix;
    } nr_union;
};

#define NR_STYPE(x) (x).nr_stype #define NR_FLAGS(x) (x).nr_flags #define NR_SOFF(x) (x).nr_soff
#define NR_SEGNO(x) (x).nr_union.nr_intref.nr_segno #define NR_RES(x) (x).nr_union.nr_intref.nr_res
#define NR_ENTRY(x) (x).nr_union.nr_intref.nr_entry #define NR_MOD(x)
(x).nr_union.nr_import.nr_mod #define NR_PROC(x) (x).nr_union.nr_import.nr_proc #define
NR_OSTYPE(x) (x).nr_union.nr_osfix.nr_ostype #define NR_OSRES(x) (x).nr_union.nr_osfix.nr_osres

#define NRSTYP 0x0f #define NRSBYT 0x00 #define NRSSEG 0x02 #define NRSPTR 0x03 #define
NRSOFF 0x05 #define NRPTR48 0x06 #define NROFF32 0x07 #define NRSOFF32 0x08

#define NRADD 0x04 #define NRRTYP 0x03 #define NRRINT 0x00 #define NRRORD 0x01 #define
NRRNAM 0x02 #define NRROSF 0x03 #define NRICHAIN 0x08

#if (EXE386 == 0)

#define RS_LEN(x) (x).rs_len #define RS_STRING(x) (x).rs_string #define RS_ALIGN(x) (x).rs_align
#define RT_ID(x) (x).rt_id #define RT_NRES(x) (x).rt_nres #define RT_PROC(x) (x).rt_proc
#define RN_OFFSET(x) (x).rn_offset #define RN_LENGTH(x) (x).rn_length #define RN_FLAGS(x)
(x).rn_flags #define RN_ID(x) (x).rn_id #define RN_HANDLE(x) (x).rn_handle #define RN_USAGE(x)
(x).rn_usage

#define RSORDID 0x8000

#define RNMOVE 0x0010 #define RNPURE 0x0020 #define RNPRELOAD 0x0040 #define RNDISCARD
0xF000

#define NE_FFLAGS_LIBMODULE 0x8000

struct rsrc_string {

    char    rs_len;
    char    rs_string[1];
};

struct rsrc_typeinfo {

    WORD    rt_id;
    WORD    rt_nres;
    DWORD           rt_proc;
};

struct rsrc_nameinfo {

    WORD    rn_offset;
    WORD    rn_length;
};

```

```
WORD rn_flags;
WORD rn_id;
WORD rn_handle;
WORD rn_usage;

};

struct new_rsrc {

WORD rs_align;
struct rsrc_typeinfo rs_typeinfo;

};

#endif

#pragma pack(pop)

#ifndef __cplusplus /* extern "C" */ #endif

#endif
```

From:
<https://ftp.osfree.org/doku/> - osFree wiki



Permanent link:
<https://ftp.osfree.org/doku/doku.php?id=en:docs:tk:formats:newexe&rev=1726998016>

Last update: **2024/09/22 09:40**