Offset	Size	Name	Description	
00h	WORD	ne_magic	Signature word NEMAGIC	
On-disk				
02h	BYTE	ne_ver	Version number of the linker	
03h	BYTE		ne_rev	Revision number of the linker
In-memory				
02h	WORD	count	Usage count (ne_ver/ne_rev on disk)	
04h	WORD	ne_enttab	Entry Table file offset, relative to the beginning of the segmented EXE header	
On-disk				
06h	WORD	ne_cbenttab	Number of bytes in the entry table	
In-memory				
06h	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

                                   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 */
  WORD  ne_heap;                    /* Initial size, in bytes, of dynamic
heap added to the

                                   data segment. This value is zero if no
initial local

                                   heap is allocated */
  WORD  ne_stack;                   /* 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 */
```

```
  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
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 */
  WORD  ne_cseg;                    /* Number of entries in the Segment
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;                /* Number of movable entries in the Entry
Table */
  WORD  ne_align;                /* Logical sector alignment shift count,
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

Offset	Size	Name	Description
00h	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
02h	WORD	ns_cbseg	Length of the segment in the file, in bytes. Zero means 64K
04h	WORD	ns_flags	Flag word
06h	WORD	ns_minalloc	Minimum allocation size of the segment, in bytes. Total size of the segment. Zero means 64K

In-memory segment entry

Offset	Size	Name	Description
00h	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
02h	WORD	ns1_cbseg	Length of the segment in the file, in bytes. Zero means 64K
04h	WORD	ns1_flags	Flag word
06h	WORD	ns1_minalloc	Minimum allocation size of the segment, in bytes. Total size of the segment. Zero means 64K
08h	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;
```

};