An virtual device drivers or some other applications for Microsoft Windows or IBM OS/2 operating system which uses 32-bits segments for 80386+ contains a combination of code and data or combination of code,data, and resources. The `LINEAR-EXECUTABLE` file such as a NEW-STYLE EXE file also contains two headers: an ^Tp236 {MS-DOS header} and a `LINEAR` EXE header.
The ^Tp236 {MS-DOS (old-style) executable-file header} contains four distinct parts: a collection of header information,a reserved section, a pointer to a `LINEAR` exe header, and a stub program. The following illustrations shows the MS-DOS executable-file header:

```
                              `Beginning of file`

                          ╓──────────────────────────────────╖
 Offset:           00h    ║         MS-DOS Header Info       ║
                          ╟──────────────────────────────────╢
                   20h    ║             Reserved             ║
                          ╟──────────────────────────────────╢
                   3Ch    ║          LE header offset        ║
                          ╟──────────────────────────────────╢
                   40h    ║                                  ║
                          ║         MS-DOS stub program      ║
                          ║                                  ║
                          ╟──────────────────────────────────╢
   Beginning of `LE` header║                                 ║
                          ║      .                           ║
                          ║      .                           ║
                          ║      .                           ║
```

If word value at offset 18h is 40h or greater, the dword value at 3Ch is an offset to a `LE` header

MS-DOS uses stub program to display a message if Windows or OS/2 has not been loaded when the user attempts to run a program.

The `LINEAR` executable-file header contains information that the loader requires for segmented executable files. This information includes the linker version number, data specified by linker, data specified by resource compiler, tables of segment data, tables of resource data, and so on. The following illustrations shows the LE file header:

```
                          ║                                  ║
                          ║   .                              ║
                          ║   .                              ║
                          ║   .                              ║
                          ╟──────────────────────────────────╢
                          ║                                  ║
    End of MS-DOS header  ║         MS-DOS stub program      ║
                          ║                                  ║
                          ╟──────────────────────────────────╢
 Beginning of `LE` header ║         Information block        ║
                          ╟──────────────────────────────────╢
                          ║           Object table           ║
                          ╟──────────────────────────────────╢
                          ║       Object page map table      ║
                          ╟──────────────────────────────────╢
                          ║  Object iterate data map table   ║
                          ╟──────────────────────────────────╢
                          ║          Resource table          ║
                          ╟──────────────────────────────────╢
                          ║        Resident-names table      ║
                          ╟──────────────────────────────────╢
                          ║           Entry  table           ║
                          ╟──────────────────────────────────╢
                          ║      Module directives table     ║
                          ╟──────────────────────────────────╢
                          ║         Fixup page table         ║
                          ╟──────────────────────────────────╢
                          ║         Fixup record table       ║
```

```
                    ║                                      ║
                    ╟──────────────────────────────────────╢
                    ║    Imported modules name table       ║
                    ╟──────────────────────────────────────╢
                    ║  Imported procedures name table      ║
                    ╟──────────────────────────────────────╢
                    ║      Per-page checksum table         ║
                    ╟──────────────────────────────────────╢
 Code and data segments                                    ║
                    ║    Code or Data Segment 1            ║
                    ║                                      ║
                    ╟──────────────────────────────────────╢
                    ║                                      ║
                    ║    Code or Data Segment 2            ║
                    ║                                      ║
                    ╟──────────────────────────────────────╢
                    ║                                      ║
                    ║        .                             ║
                    ║        .                             ║
                    ║        .                             ║
                    ║                                      ║
                    ║                                      ║
                    ║    Code or Data Segment n            ║
                    ║                                      ║
                    ╟──────────────────────────────────────╢
                    ║                                      ║
                    ║        Non-resident table            ║
                    ║                                      ║
                    ╚══════════════════════════════════════╝

                     `End of file`
```

See also :   MS-DOS old-style ^Tp236 {EXE File Header}

|===

`LE Header Information Block Layout`

The `information block` in the LE header contains the linker version number, length of various tables that further describe the executable file, the offsets from the beginning of the header to the beginnig of these tables, the heap and stack sizes, and so on. The following list summarizes the contents of the header `information block` (the locations are relative to the beginning of the block):

```
Offset Size Contents
▀▀▀▀▀▀ ▀▀▀▀ ▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

                ┌───────┐
 +0      2      │4Ch 45H│  Specifies the signature word 'LE'
                ├───┬───┘
 +2      1      │   │      Byte order:(00h = little-endian, nonzero = big-endian)
                ├───┤
 +3      1      │   │      Word order:(00h = little-endian, nonzero = big-endian)
                ├───┼───┬───┬───┐
 +4      4      │Exe format lev │  Executable format level
                ├───┼───┴───┴───┘
 +8      2      │CPU typ│  CPU type:
                └───┴───┘    01h - Intel 80286 or upwardly compatible
                             02h - Intel 80386 or upwardly compatible
                             03h - Intel 80486 or upwardly compatible
                             04h - Intel 80586 or upwardly compatible
                             20h - Intel i860 (N10) or compatible
                             21h - Intel "N11" or compatible
                             40h - MIPS Mark I ( R2000, R3000) or compatible
                             41h - MIPS Mark II ( R6000 ) or compatible
                             42h - MIPS Mark III ( R4000 ) or compatible
                ┌───────┐
 +0Ah    2      │OS Type│  Target operating system:
```

```
                └───┴───┘   01h - OS/2
                            02h - Windows
                            03h - DOS 4.x
                            04h - Windows 386
                ┌───┬───┬───┬───┐
    +0Ch    4   │Module version │ Module version.
                ├───┼───┼───┼───┤
    +10h    4   │Module Type Flg│ Module type flags
                ├───┼───┼───┼───┤
    +14h    4   │ Memory Pages  │ Number of memory pages
                ├───┼───┼───┼───┤
    +18h    4   │Init CS object │ Initial object CS number
                ├───┼───┼───┼───┤
    +1Ch    4   │ Init Offset   │ Initial EIP
                ├───┼───┼───┼───┤
    +20h    4   │Init SS object │ Initial object SS number
                ├───┼───┼───┼───┤
    +24h    4   │ Init ESP Offs │ Initial ESP
                ├───┼───┼───┼───┤
    +28h    4   │Mem Page size  │ Memory page size
                ├───┼───┼───┼───┤
    +2Ch    4   │ Last page Byts│ Bytes on last page
                ├───┼───┼───┼───┤
    +30h    4   │ Fixup size    │ Fixup section size
                ├───┼───┼───┼───┤
    +34h    4   │ Fixup checksum│ Fixup section checksum
                ├───┼───┼───┼───┤
    +38h    4   │Loader sect siz│ Loader section size
                ├───┼───┼───┼───┤
    +3Ch    4   │Loader checksum│ Loader section checksum
                ├───┼───┼───┼───┤
    +40h    4   │Object table of│ Offset of object table
                ├───┼───┼───┼───┤
    +44h    4   │Obj table entr │ Object table entries
                ├───┼───┼───┼───┤
    +48h    4   │Obj page map   │ Object page map offset
                ├───┼───┼───┼───┤
    +4Ch    4   │Obj iter dat mp│ Object iterate data map offset
                ├───┼───┼───┼───┤
    +50h    4   │Resource offset│ Resource table offset
                ├───┼───┼───┼───┤
    +54h    4   │Resource entr  │ Resource  table entries
                ├───┼───┼───┼───┤
    +58h    4   │Resident name  │ Resident names table offset
                ├───┼───┼───┼───┤
    +5Ch    4   │Entry table ofs│ Entry table offset
                ├───┼───┼───┼───┤
    +60h    4   │Module direct  │ Module directives table offset
                ├───┼───┼───┼───┤
    +64h    4   │Module dir entr│ Module directives entries
                ├───┼───┼───┼───┤
    +68h    4   │Fixup page tabl│ Fixup page table offset
                ├───┼───┼───┼───┤
    +6Ch    4   │Fixup rec table│ Fixup record table offset
                ├───┼───┼───┼───┤
    +70h    4   │Imported module│ Imported modules name table offset
                ├───┼───┼───┼───┤
    +74h    4   │Imported mod cn│ Imported modules count
                ├───┼───┼───┼───┤
    +78h    4   │Imported proc  │ Imported procedure name table offset
                ├───┼───┼───┼───┤
    +7Ch    4   │Per-page checks│ Per-page checksum table offset
                ├───┼───┼───┼───┤
    +80h    4   │Data pages offs│ Data pages offset from top of file
                ├───┼───┼───┼───┤
    +84h    4   │Preload page cn│ Preload page count
                ├───┼───┼───┼───┤
    +88h    4   │Non-resid table│ Non-resident names table offset from top of file
                ├───┼───┼───┼───┤
```

```
  +8Ch    4   │Non-resid size │ Non-resident names table length
              ├───┼───┼───┼───┤
  +90h    4   │Non-res checksm│ Non-resident names table checksum
              ├───┼───┼───┼───┤
  +94h    4   │Auto data obj  │ Automatic data object
              ├───┼───┼───┼───┤
  +98h    4   │Debug info offs│ Debug information offset
              ├───┼───┼───┼───┤
  +9Ch    4   │Debug inf size │ Debug information length
              ├───┼───┼───┼───┤
  +A0h    4   │Preload pg numb│ Preload instance pages number
              ├───┼───┼───┼───┤
  +A4h    4   │Demand pg numb │ Demand instance pages number
              ├───┼───┼───┼───┤
  +A8h    4   │Extra head aloc│ Extra heap allocation
              ├───┼───┼───┼───┤
  +ACh    4   │  Unknown  │   │  ???
              └───┴───┴───┴───┘
```

See also : NE Header Information Block Layout
|===

`LE Header Information Block Flags Layout`

```
 3 3 2 2  2 2 2 2    2 2 2 2  1 1 1 1
 1 0 9 8  7 6 5 4    3 2 1 0  9 8 7 6
 ■ ■ ■ ■  ■ ■ ■ ■    ■ ■ ■ ■  ■ ■ ■ ■
 └─┴─┴─┴──┴─┴─┴─┴────┴─┴─┴─┴──┴─┴─┴─┴───  Reserved

 1 1 1 1  1 1
 5 4 3 2  1 0 9 8    7 6 5 4  3 2 1 0
 ■ ■ ■ ■  ■ ■ ■ ■    ■ ■ ■ ■  ■ ■ ■ ■
 │   │      │ │ │        │ │    │
 │   │      └─┼─┘        │ │    └────2─ Initialization ( Only for DLL ):
 │   │        │          │ │             0: Global
 │   │        │          │ │             1: Per-Process
 │   │        │          │ │
 │   │        │          │ └─────────4─ 1:No internal fixup in exe image
 │   │        │          └───────────5─ 1:No external fixup in exe image
 │   │        └─────────────────8,9,10─   0 - Unknown
 │   │                                    1 - Incompatible with PM windowing
 │   │                                    2 -   Compatible with PM windowing
 │   │                                    3 - Uses PM windowing API
 │   │
 │   └──────────────────────────────13─ 1: Module not loadable
 └──────────────────────────────────15─ 1: Module is DLL rather then program
```

See also : NE Header Information Block Flags Layout
|===

`LE Header Object Table Layout`

The object table contains information that describes each segment in an executable file. This information includes segment length, segment type, and segment-relocation data. The following list summarizes the values found in in the segment table (the locations are relative to the beginning of each entry):

```
Offset Size Contents
══════ ════ ═══════════════════════════════════════════════════════════════════

              ┌───┬───┬───┬───┐
+0h       4   │ Virt segm size│ Virtual segment size in bytes
              ├───┼───┼───┼───┤
+4h       4   │Reloc base addr│ Relocation base address
              ├───┼───┼───┼───┤
```

```
+8h      4   │ Object flags   │ Object flags
             ├───┼───┼───┼───┤
+Ch      4   │ Page map index │ Page map index
             ├───┼───┼───┼───┤
+10h     4   │ Page map entr  │ Page map entries
             ├───┼───┼───┼───┤
+14h     4   │    Unknown     │ ???
             └───┴───┴───┴───┘
```

See also :
|==

`LE Header Object Flags Layout`

```
 3 3 2 2  2 2 2 2    2 2 2 2  1 1 1 1
 1 0 9 8  7 6 5 4    3 2 1 0  9 8 7 6
 ■ ■ ■ ■  ■ ■ ■ ■    ■ ■ ■ ■  ■ ■ ■ ■
 └─┴─┴─┴──┴─┴─┴─┴────┴─┴─┴─┴──┴─┴─┴─┴───  Reserved

 1 1 1 1  1 1
 5 4 3 2  1 0 9 8    7 6 5 4  3 2 1 0
 ■ ■ ■ ■  ■ ■ ■ ■    ■ ■ ■ ■  ■ ■ ■ ■
 │ │ │ │    │ └┬┘    │ │ │ │  │ │ │ └─0─  1: Readable
 │ │ │ │    │  │     │ │ │ │  │ │ └───1─  1: Writable
 │ │ │ │    │  │     │ │ │ │  │ └─────2─  1: Executable
 │ │ │ │    │  │     │ │ │ │  └───────3─  1: Resource
 │ │ │ │    │  │     │ │ │ └──────────4─  1: Discardable
 │ │ │ │    │  │     │ │ └────────────5─  1: Shared
 │ │ │ │    │  │     │ └──────────────6─  1: Preloaded
 │ │ │ │    │  │     └────────────────7─  1: Invalid
 │ │ │ │    │  └──────────────────8,9──  Type: 00 - Normal
 │ │ │ │    │                                  01 - Zero-filled
 │ │ │ │    │                                  10 - Resident
 │ │ │ │    │                                  11 - Resident/contiguous
 │ │ │ │    └────────────────────────10─  "RESIDENT/LONG_LOCABLE"
 │ │ │ │
 │ │ │ └─────────────────────────────12─  "16:16_ALIAS"
 │ │ └───────────────────────────────13─  1: "BIG" (USE32)
 │ └─────────────────────────────────14─  1: Conforming
 └───────────────────────────────────15─  "Object_i/o_privilage_level"
```

See also :
|==

`LE Header Resident-Name Table Layout`

The `resident-name` table contains strings that identify exported functions in the exe file. As the name implies, these strings are resident in system memory and never discarded. The `resident-name` strings are case-sensitive and are not null-terminated. The following list summarizes the values found in in the `resident-name` table (the locations are relative to the beginning of each entry):

```
Offset Size Contents
▀▀▀▀▀▀ ▀▀▀▀ ▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

             ┌───┐
+0h      1   │Siz│     Specifies the length of a string.If there are no more
             └───┘       strings in the table, this value is zero.
             ┌── ──── ──── ──── ─── ──┐
+1h      N   │  String                │ Specifies the `resident-name` text.
             ├───┬────┬───────────────┘
+N+01h   2   │Index   │  Specifies an ordinal number, that identifies the string.
             └───┴────┘  This number is an index into the entry table.
```

The first string in the resident-name table is the module name.

See also : LE Header Information Block Layout
|==

`LE Header Entry-Table Layout`

The `entry table` contains bundles of entry points from exe file (the linker generates each bundle). The numbering system for these ordinal values is 1-based -- that is, the ordinal value corresponding to the first entry point is 1.

The linker generates the densest possible bundles under the restriction that it cannot reorder the entry points. This restriction is necessary because other exe files may refer to entry points within a given bundle by their ordinar values.

The `entry-table` data is organized by bundle, each of which begins with a 2-byte header. The first byte of the header specifies the number of entries in the bungle (a value of 00h designates the end of the table). The second byte specifies flags. The third and forth byte specified object number.

```
Offset Size Contents
▀▀▀▀▀▀ ▀▀▀▀ ▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

          ┌───┐
+0h      1 │Siz│     Number of entries in this bungle
          ├───┤
+1h      1 │Ind│     Bungle flags :
          └───┘     7 6 5 4   3 2 1 0
                    ■ ■ ■ ■   ■ ■ ■ ■
                                   │ └0─  1:Valid entry,   0:Zero entry
                                   └───1─  1:32-bits entry, 0:16-bits entry
          ┌───┬───┐
+2h      2 │Index  │ Object index
          └───┴───┘
          ┌───────────────────┐
+4h 3 or 5 │  Entry 1          │
          ├───────────────────┤
+?? 3 or 5 │  Entry 2          │
          ├── ── ─── ─────────┤
+?? 3 or 5 │  Entry N          │
          └───────────────────┘
```

Each entry consists of 3 or 5 bytes and has the following form:

```
Offset Size Contents
▀▀▀▀▀▀ ▀▀▀▀ ▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀

          ┌───┐
+0h      1 │Flg│ Specifies a byte value.This value can be a combination of
          └───┘  the following bits:
                 7 6 5 4  3 2 1 0
                 ■ ■ ■ ■  ■ ■ ■ ■
                 └─┴─┴─┬─┘    │ └─  1: Entry is exported
                       │      └───  1: The segment uses a global (shared)
                       │               data segment.
                       │
                       └──────────  If Code segment these bits specify the
                                       number of words that compose the
                                       stack.At the time of the ring
                                       transitions, these words must be
                                       copied from one ring to the other.
          ┌───┬───┐
+1   2 or 4 │Offset │ Specifies the segment offset. ( Word or Dword depending
          └───┴───┘ on bit 1 bungle flags
```

See also : LE Header Information Block Layout

|==

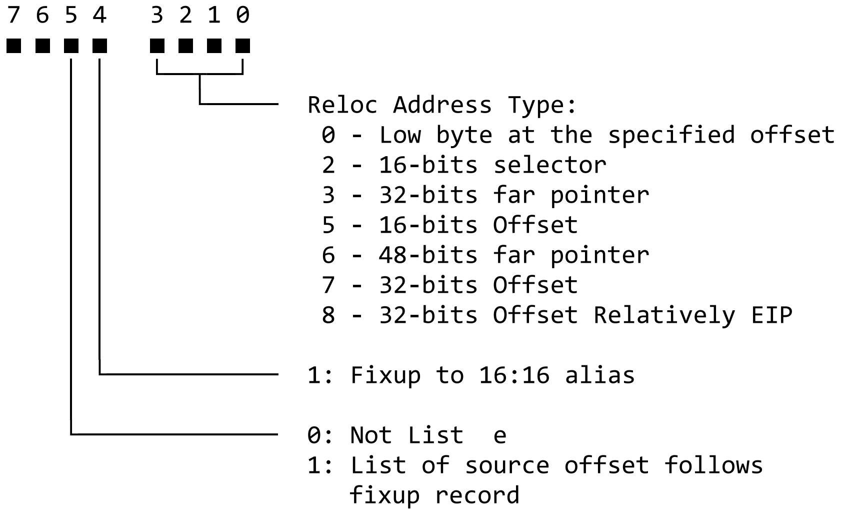
`LE Header Fixup Record  Table Layout`

Code and data segments follow the LE header. Some of code segments may contain calls to function in other segments and may,therefore, require relocation data to resolve those references.This relocation data is stored in a fixup record table.A relocation item is a collection of bytes specifying the following information:

- Address type (Segment only,offset only,segment and offset)
- Relocation type (internal reference, imported ordinal, imported name)
- Segment number or ordinal identifier (for internal references)
- Reference-table index or function ordinal number (for imported ordinal)
- Reference-table index or name-table offset (for imported names)

Each relocation item consist of:

Offset	Size	Contents
+0h	1	ATp Specify relocation addres type:

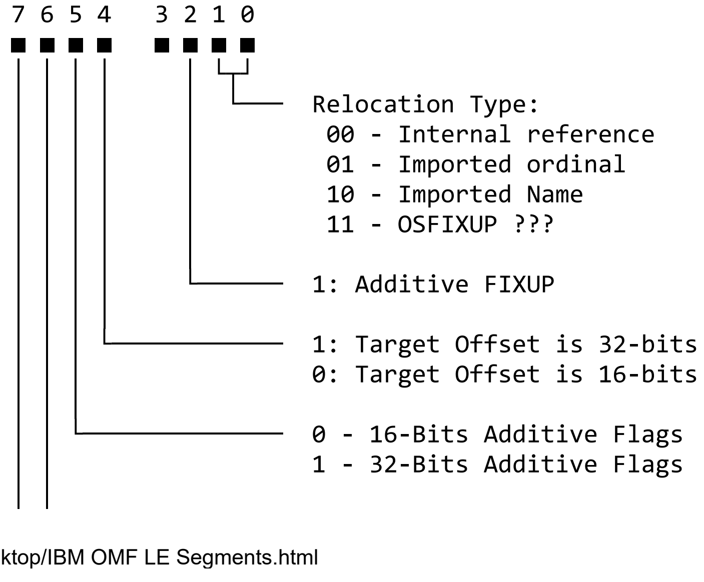


Reloc Address Type:
 0 - Low byte at the specified offset
 2 - 16-bits selector
 3 - 32-bits far pointer
 5 - 16-bits Offset
 6 - 48-bits far pointer
 7 - 32-bits Offset
 8 - 32-bits Offset Relatively EIP

1: Fixup to 16:16 alias

0: Not List e
1: List of source offset follows
   fixup record

Offset	Size	Contents
+1h	1	Rtp Specify relocation type:



Relocation Type:
 00 - Internal reference
 01 - Imported ordinal
 10 - Imported Name
 11 - OSFIXUP ???

1: Additive FIXUP

1: Target Offset is 32-bits
0: Target Offset is 16-bits

0 - 16-Bits Additive Flags
1 - 32-Bits Additive Flags

```
                         │ └────────────────  1 - 16-bit Object/Module ordinal
                         │                    0 -  8-bit Object/Module ordinal
                         │
                         │
                         └──────────────────  1 -  8-bit Import Ordinal
                                              0 -  Bit 4 Toggles Between 16 and
                                                   32 ordinal




╔═ If Bit 5 of relocation addres type equal 0 ══════════════════════════════╗
║                                                                           ║
║             ┌───┬───┐                                                     ║
║+2h      2   │RelOffs│Specify the offset of the relocation item whithin    ║
║             └───┴───┘ current page ( See fixup page table )               ║
║                                                                           ║
║---For internal reference ------                                           ║
║             ┌───┐                                                         ║
║+4h      1   │Ind│     Specify Target segment number.                      ║
║             └───┘                                                         ║
║                                                                           ║
║---For imported ordinal-----------------------                             ║
║             ┌───┐                                                         ║
║+4h      1   │Ind│  Imported module-name index.See imported modules name table
║             ├───┼─ ─┐                                                     ║
║+5h   1 or 2 │Ordinal│ Ordinal value Depending on Bit 7 of Relocation type   ║
║             └───┴───┘                                                     ║
║             ┌ ──┬ ───┬─ ──┬─ ┐                                            ║
║+6(7)  2 (4) │ Abs Add Value  │ This field present if Bit 2 of Relocation  ║
║             └ ──┴─ ──┴─ ──┴─ ┘  Type Set to 1. Its size 2 or 4 bytes depending
║                                 on Bit 4 of Relocation Type.              ║
║             ┌ ─ ┬ ─ ┐                                                     ║
║+n        2  │Extra  │ Present if bit 4 Relocation Type Set to 1           ║
║             └─ ─┴ ──┘                                                     ║
║                                                                           ║
║---For imported name--------------------------                             ║
║                                                                           ║
║             ┌───┐                                                         ║
║+4h      1   │Ind│  Imported module-name index.See imported modules name table
║             ├───┼───┐                                                     ║
║+6h      2   │Offset │ Offset of name in imported procedure names table.   ║
║             └───┴───┘                                                     ║
║             ┌ ──┬ ───┬─ ──┬─ ┐                                            ║
║+8     2 (4) │ Abs Add Value  │ This field present if Bit 2 of Relocation  ║
║             └ ──┴─ ──┴─ ──┴─ ┘  Type Set to 1. Its size 2 or 4 bytes depending
║                                 on Bit 4 of Relocation Type.              ║
║                                                                           ║
║             ┌ ─ ┬ ─ ┐                                                     ║
║+n        2  │Extra  │ Present if bit 4 Relocation Type Set to 1           ║
║             └─ ─┴ ──┘                                                     ║
║                                                                           ║
╚═══════════════════════════════════════════════════════════════════════════╝
╔═ If Bit 5 of relocation addres type equal 1 ══════════════════════════════╗
║                                                                           ║
║             ┌───┐                                                         ║
║+2h      1   │Cnt│   Offset Counter                                        ║
║             └───┘                                                         ║
║                                                                           ║
║---For internal reference ------                                           ║
║             ┌───┐                                                         ║
║+4h      1   │Ind│     Specify Target segment number.                      ║
║             └───┘                                                         ║
║             ┌───┬───┐                                                     ║
║+5h    2*N   │RelOffs│Specify the offset of the relocation item whithin    ║
║             ├───┼───┤ current page ( See fixup page table )               ║
║             │   │   │                                                     ║
║            ├───┼───┤
║                ...
║            ├───┼───┤
║            │   N   │
║            └───┴───┘
║
║---For imported ordinal------------------------
║            ┌───┐
║+4h      1  │Ind│  Imported module-name index.See imported modules name table
║            ├───┼ ─ ┐
║+5h   1 or 2│Ordinal│  Ordinal value Depending on Bit 7 of Relocation type
║            └───┴───┘
║            ┌ ──┬ ──┬─ ─┬─ ┐
║+6(7)  2 (4)│ Abs Add Value │  This field present if Bit 2 of Relocation
║            └ ──┴ ──┴─ ─┴─ ┘   Type Set to 1. Its size 2 or 4 bytes depending
║                               on Bit 4 of Relocation Type.
║            ┌ ─ ┬ ─ ┐
║+n        2 │Extra  │  Present if bit 4 Relocation Type Set to 1
║            └─ ─┴ ──┘
║            ┌───┬───┐
║+n+2   2*N  │RelOffs│Specify the offset of the relocation item whithin
║            ├───┼───┤ current page ( See fixup page table )
║            │   │   │
║            ├───┼───┤
║                ...
║            ├───┼───┤
║            │   N   │
║            └───┴───┘
║
║---For imported name---------------------------
║
║            ┌───┐
║+4h      1  │Ind│  Imported module-name index.See imported modules name table
║            ├───┼───┐
║+6h      2  │Offset │  Offset of name in imported procedure names table.
║            └───┴───┘
║            ┌ ──┬ ──┬─ ─┬─ ┐
║+8     2 (4)│ Abs Add Value │  This field present if Bit 2 of Relocation
║            └ ──┴ ──┴─ ─┴─ ┘   Type Set to 1. Its size 2 or 4 bytes depending
║                               on Bit 4 of Relocation Type.
║
║            ┌ ─ ┬ ─ ┐
║+n        2 │Extra  │  Present if bit 4 Relocation Type Set to 1
║            └─ ─┴ ──┘
║            ┌───┬───┐
║+n+2   2*N  │RelOffs│Specify the offset of the relocation item whithin
║            ├───┼───┤ current page ( See fixup page table )
║            │   │   │
║            ├───┼───┤
║                ...
║            ├───┼───┤
║            │   N   │
║            └───┴───┘
║
╚══════════════════════════════════════════════════════════════════════════╝
```

See also : LE Header Information Block Layout

|===

`LE Header Fixup Page  Table Layout`

In the LE header fixup records table are array of fixup records and offset into fixup records are relative to the current page. Fixup page table serves to identify fixup records into code and data segments offset.

Fixup page table is array of dwords. Number of dwords is number of pages plus 1.Each dword contains offset into Fixup Record Table of first fixup in the current page. Last dword contains size of fixup record table in bytes.I.e. substraction contains dword+1 with current dword is fixup table size for current page.

For example: Number of page is 4.

1	0	Offset of fixup for 1 page
2	5	Offset of fixup for 2 page
3	5	Offset of fixup for 3 page
4	0Ch	Offset of fixup for 4 page
5	13h	Size of fixup record table.

First page have fixup records at offset 0, its size is 5-0 = 5 bytes.
Second page hasn't fixup,because its size is 5-5=0 bytes.
Third page have fixup records at offset 5, its size is 0C-5 = 7 bytes.
Forth page have fixup records at offset 0Ch, its size is 13h-0Ch = 7 bytes.

See also : LE Header Information Block Layout
|==

`LE Header Imported-modules Name Table Layout`

The `imported-modules name` table contains the names of modules that the exe file imports. Each entry contains two parts: a single byte that specifies the length of the string and the string itself. The strings in this table are not null-terminated.

Offset	Size	Contents	
+0h	1	Siz	Specifies the length of a string
+1h	N	String	Specifies the string text.

The first byte in `imported-modules name` table is zero. First name begins from offset +1.

See also : NE Header Information Block Layout
|==

`LE Header Imported-procedures Name Table Layout`

The `imported-procedures name` table contains the names of procedures that the exe file imports. Each entry contains two parts: a single byte that specifies the length of the string and the string itself. The strings in this table are not null-terminated.

Offset	Size	Contents	
+0h	1	Siz	Specifies the length of a string
+1h	N	String	Specifies the string text.

See also : NE Header Information Block Layout
|==

`LE Header Nonresident-Name Table Layout`

The `nonresident-name` table contains strings that identify exported functions in the exe file. As the name implies, these strings are not always resident in system memory and discardable. The `nonresident-name` strings are case-sensitive and are not null-terminated. The following list summarizes the values found in in the `nonresident-name` table (the locations are relative to the beginning of each entry):

Offset	Size	Contents	
+0h	1	Siz	Specifies the length of a string.If there are no more strings in the table, this value is zero.
+1h	N	String	Specifies the `nonresident-name` text.
+N+01h	2	Index	Specifies an ordinal number, that identifies the string. This number is an index into the entry table.

The first name that appearsin the `nonresident-name` table is the module description string (which was specified in the module-definition file).

See also : LE Header Information Block Layout

|===

`LE Header Object Page Map Table`

The `object page map` table contains location of each page into exe file.This table consists of Dwords. Each dword correspond to one page in exe file. Number of page is set in LE Header Information Block at offset +14h.

Offset	Size	Contents	
+0h	2	HighPag	High page Number
+2	1	Low	Low page Number
+4	1	FLG	Page FLAGS:

```
    7 6 5 4  3 2 1 0
    ■ ■ ■ ■  ■ ■ ■ ■
    └─┘          └─┘
     │            └──────  11 - Last page in file
     │
     └───────────────────  Page Type: 00 - Legal
                                      01 - Iterated
                                      10 - Invalid
                                      11 - Zero filled
```

To compute page offset into file necessary:
(HighPageNumber+LowPageNumber-1)*PageSize+FirstPageOffset

See also : LE Header Information Block Layout