Portable Executable Format by Micheal J. O'Leary

PORTABLE EXECUTABLE FORMAT

Author: Micheal J. O'Leary

Preface

This document was edited and released by Microsoft Developer Support. It describes the binary portable executable format for NT. The information is provided at this point because we feel it will make the work of application development easier. Unfortunately, the information in this document may change before the final release of Windows NT. Microsoft is NOT committing to stay with these formats by releasing this document. Questions or follow-ups for any of the information presented here should be posted to CompuServe MSWIN32 forum, section 6.

--Steve Firebaugh
Microsoft Developer Support

Contents

1. Overview

```
    +-------------------+  <--+ <----- Base of Image Header
    ¦ DOS 2 Compatible  ¦     ¦
    ¦    EXE Header     ¦     ¦
    +-------------------¦     ¦
    ¦      unused       ¦     ¦
    +-------------------¦     ¦
    ¦  OEM Identifier   ¦     ¦
    ¦  OEM Info         ¦     ¦
    ¦                   ¦     ¦  DOS 2.0 Section
    ¦    Offset to      ¦     ¦  (for DOS compatibility only)
    ¦    PE Header      ¦     ¦
    +-------------------¦     ¦
    ¦   DOS 2.0 Stub    ¦     ¦
    ¦   Program &       ¦     ¦
    ¦   Reloc. Table    ¦     ¦
    +-------------------¦  <--+
    ¦      unused       ¦
    +-------------------¦  <--------- Aligned on 8 byte boundary
    ¦    PE Header      ¦
    +-------------------¦
    ¦   Object Table    ¦
    +-------------------¦
    ¦   Image Pages     ¦
    ¦     import info   ¦
    ¦     export info   ¦
    ¦     fixup info    ¦
    ¦     resource info¦
    ¦     debug info    ¦
    +-------------------+
```

Figure 1. A typical 32-bit Portable EXE File Layout

2. PE Header

```
    +-------------------------------------------------------+
0   ¦      SIGNATURE BYTES        ¦  CPU TYPE   ¦  # OBJECTS  ¦
    +-----------------------------+---------------------------¦
8   ¦       TIME/DATE STAMP       ¦         RESERVED          ¦
    +-----------------------------+---------------------------¦
16  ¦          RESERVED           ¦  NT HDR SIZE¦    FLAGS    ¦
```

```
     +---------------------------+---------------------------¦
24   ¦  RESERVED   ¦LMAJOR¦LMINOR¦         RESERVED          ¦
     +---------------------------+---------------------------¦
32   ¦          RESERVED         ¦         RESERVED          ¦
     +---------------------------+---------------------------¦
40   ¦       ENTRYPOINT RVA      ¦         RESERVED          ¦
     +---------------------------+---------------------------¦
48   ¦          RESERVED         ¦        IMAGE BASE         ¦
     +---------------------------+---------------------------¦
56   ¦       OBJECT ALIGN        ¦        FILE ALIGN         ¦
     +---------------------------+---------------------------¦
64   ¦  OS MAJOR   ¦  OS MINOR   ¦USER MAJOR   ¦USER MINOR   ¦
     +-------------+-------------+---------------------------¦
72   ¦ SUBSYS MAJOR¦ SUBSYS MINOR¦         RESERVED          ¦
     +---------------------------+---------------------------¦
80   ¦         IMAGE SIZE        ¦       HEADER SIZE         ¦
     +---------------------------+---------------------------¦
88   ¦       FILE CHECKSUM       ¦  SUBSYSTEM  ¦  DLL FLAGS  ¦
     +---------------------------+---------------------------¦
96   ¦   STACK RESERVE SIZE      ¦     STACK COMMIT SIZE     ¦
     +---------------------------+---------------------------¦
104  ¦   HEAP RESERVE SIZE       ¦     HEAP COMMIT SIZE      ¦
     +---------------------------+---------------------------¦
112  ¦         RESERVED          ¦  # INTERESTING RVA/SIZES  ¦
     +---------------------------+---------------------------¦
120  ¦   EXPORT TABLE RVA        ¦   TOTAL EXPORT DATA SIZE  ¦
     +---------------------------+---------------------------¦
128  ¦   IMPORT TABLE RVA        ¦   TOTAL IMPORT DATA SIZE  ¦
     +---------------------------+---------------------------¦
136  ¦  RESOURCE TABLE RVA       ¦  TOTAL RESOURCE DATA SIZE ¦
     +---------------------------+---------------------------¦
144  ¦  EXCEPTION TABLE RVA      ¦  TOTAL EXCEPTION DATA SIZE¦
     +---------------------------+---------------------------¦
152  ¦  SECURITY TABLE RVA       ¦  TOTAL SECURITY DATA SIZE ¦
     +---------------------------+---------------------------¦
160  ¦    FIXUP TABLE RVA        ¦  TOTAL FIXUP DATA SIZE    ¦
     +---------------------------+---------------------------¦
     ¦    DEBUG TABLE RVA        ¦  TOTAL DEBUG DIRECTORIES  ¦
     +---------------------------+---------------------------¦
     ¦  IMAGE DESCRIPTION RVA    ¦  TOTAL DESCRIPTION SIZE   ¦
     +---------------------------+---------------------------¦
     ¦   MACHINE SPECIFIC RVA    ¦   MACHINE SPECIFIC SIZE   ¦
     +---------------------------+---------------------------¦
     ¦  THREAD LOCAL STORAGE RVA ¦      TOTAL TLS SIZE       ¦
     +-------------------------------------------------------+
```

Figure 2. PE Header

Notes:

- o A VA is a virtual address that is already biased by the Image Base found in the PE Header. A RVA is a virtual address that is relative to the Image Base.

- o An RVA in the PE Header which has a value of zero indicates the field isn't used.

```
  o  Image pages are aligned and zero padded to a File Align
     boundary.  The bases of all other tables and structures must be
     aligned on DWORD (4 byte) boundary.  Thus, all VA's and RVA's
     must be on a 32 bit boundary. All table and structure fields
     must be aligned on their "natural" boundaries, with the possible
     exception of the Debug Info.

SIGNATURE BYTES = DB * 4.
Current value is "PE/0/0". Thats PE followed by two zeros (nulls).

CPU TYPE = DW CPU Type.
This field specifies the type of CPU compatibility required by this
image to run.  The values are:

  o  0000h __unknown

  o  014Ch __80386

  o  014Dh __80486

  o  014Eh __80586

  o  0162h __MIPS Mark I (R2000, R3000)

  o  0163h __MIPS Mark II (R6000)

  o  0166h __MIPS Mark III (R4000)

# OBJECTS = DW Number of object entries.
This field specifies the number of entries in the Object Table.

TIME/DATE STAMP = DD Used to store the time and date the file was
created or modified by the linker.

NT HDR SIZE = DW This is the number of remaining bytes in the NT
header that follow the FLAGS field.

FLAGS = DW Flag bits for the image.
The flag bits have the following definitons:

  o  0000h __Program image.

  o  0002h __Image is executable.
     If this bit isn't set, then it indicates that either errors
     where detected at link time or that the image is being
     incrementally linked and therefore can't be loaded.

  o  0200h __Fixed.
     Indicates that if the image can't be loaded at the Image Base,
     then don't load it.

  o  2000h __Library image.


LMAJOR/LMINOR = DB Linker major/minor version number.
```

```
ENTRYPOINT RVA = DD Entrypoint relative virtual address.
The address is relative to the Image Base.  The address is the
starting address for program images and the library initialization
and library termination address for library images.

IMAGE BASE = DD The virtual base of the image.
This will be the virtual address of the first byte of the file (Dos
Header).  This must be a multiple of 64K.

OBJECT ALIGN = DD The alignment of the objects. This must be a power
of 2 between 512 and 256M inclusive. The default is 64K.

FILE ALIGN = DD Alignment factor used to align image pages.  The
alignment factor (in bytes) used to align the base of the image pages
and to determine the granularity of per-object trailing zero pad.
Larger alignment factors will cost more file space; smaller alignment
factors will impact demand load performance, perhaps significantly.
Of the two, wasting file space is preferable.  This value should be a
power of 2 between 512 and 64K inclusive.

OS MAJOR/MINOR = DW OS version number required to run this image.

USER MAJOR/MINOR # = DW User major/minor version number.
This is useful for differentiating between revisions of
images/dynamic linked libraries.  The values are specified at link
time by the user.

SUBSYS MAJOR/MINOR # = DW Subsystem major/minor version number.

IMAGE SIZE = DD The virtual size (in bytes) of the image.
This includes all headers.  The total image size must be a multiple
of Object Align.

HEADER SIZE = DD Total header size.
The combined size of the Dos Header, PE Header and Object Table.

FILE CHECKSUM = DD Checksum for entire file.  Set to 0 by the linker.

SUBSYSTEM = DW NT Subsystem required to run this image.
The values are:

  o  0000h __Unknown

  o  0001h __Native

  o  0002h __Windows GUI

  o  0003h __Windows Character

  o  0005h __OS/2 Character

  o  0007h __Posix Character

DLL FLAGS = DW Indicates special loader requirements.
This flag has the following bit values:
  o  0001h __Per-Process Library Initialization.

  o  0002h __Per-Process Library Termination.

  o  0004h __Per-Thread Library Initialization.

  o  0008h __Per-Thread Library Termination.

All other bits are reserved for future use and should be set to zero.

STACK RESERVE SIZE = DD Stack size needed for image.
The memory is reserved, but only the STACK COMMIT SIZE is committed.
The next page of the stack is a 'guarded page'. When the application
hits the guarded page, the guarded page becomes valid, and the next
page becomes the guarded page. This continues until the RESERVE SIZE
is reached.

STACK COMMIT SIZE = DD Stack commit size.

HEAP RESERVE SIZE = DD Size of local heap to reserve.

HEAP COMMIT SIZE = DD Amount to commit in local heap.

# INTERESTING VA/SIZES = DD Indicates the size of the VA/SIZE array
that follows.

EXPORT TABLE RVA = DD  Relative Virtual Address of the Export Table.
This address is relative to the Image Base.

IMPORT TABLE RVA = DD  Relative Virtual Address of the Import Table.
This address is relative to the Image Base.

RESOURCE TABLE RVA = DD  Relative Virtual Address of the Resource
Table. This address is relative to the Image Base.

EXCEPTION TABLE RVA = DD  Relative Virtual Address of the Exception
Table. This address is relative to the Image Base.

SECURITY TABLE RVA = DD  Relative Virtual Address of the Security
Table. This address is relative to the Image Base.

FIXUP TABLE RVA = DD  Relative Virtual Address of the Fixup Table.
This address is relative to the Image Base.

DEBUG TABLE RVA = DD  Relative Virtual Address of the Debug Table.
This address is relative to the Image Base.

IMAGE DESCRIPTION RVA = DD  Relative Virtual Address of the
description string specified in the module definiton file.

MACHINE SPECIFIC RVA = DD  Relative Virtual Address of a machine
specific value. This address is relative to the Image Base.

TOTAL EXPORT DATA SIZE = DD  Total size of the export data.
```

TOTAL IMPORT DATA SIZE = DD Total size of the import data.

TOTAL RESOURCE DATA SIZE = DD Total size of the resource data.

TOTAL EXCEPTION DATA SIZE = DD Total size of the exception data.

TOTAL SECURITY DATA SIZE = DD Total size of the security data.

TOTAL FIXUP DATA SIZE = DD Total size of the fixup data.

TOTAL DEBUG DIRECTORIES = DD Total number of debug directories.

TOTAL DESCRIPTION SIZE = DD Total size of the description data.

MACHINE SPECIFIC
SIZE = DD A machine specific value.

3. Object Table

The number of entries in the Object Table is given by the # Objects field in the PE Header. Entries in the Object Table are numbered starting from one. The object table immediately follows the PE Header. The code and data memory object entries are in the order chosen by the linker. The virtual addresses for objects must be assigned by the linker such that they are in ascending order and adjacent, and must be a multiple of Object Align
in the PE header.

Each Object Table entry has the following format:

```
    +-------------------------------------------------------+
    ¦                      OBJECT NAME                      ¦
    +-------------------------------------------------------¦
    ¦       VIRTUAL SIZE        ¦           RVA             ¦
    +---------------------------+---------------------------¦
    ¦      PHYSICAL SIZE        ¦      PHYSICAL OFFSET      ¦
    +---------------------------+---------------------------¦
    ¦         RESERVED          ¦         RESERVED          ¦
    +---------------------------+---------------------------¦
    ¦         RESERVED          ¦       OBJECT FLAGS        ¦
    +-------------------------------------------------------+
```

Figure 3. Object Table

OBJECT NAME = DB * 8 Object name. This is an eight-byte null-padded ASCII string representing the object name.

VIRTUAL SIZE = DD Virtual memory size. The size of the object that will be allocated when the object is loaded. Any difference between PHYSICAL SIZE and VIRTUAL SIZE is zero filled.

RVA = DD Relative Virtual Address. The virtual address the object is currently relocated to, relative to the Image Base. Each Object's virtual address space consumes a multiple of Object Align (power of 2

```
between 512 and 256M inclusive. Default is 64K), and immediately
follows the previous Object in the virtual address space (the virtual
address space for a image must be dense).

PHYSICAL SIZE = DD Physical file size of initialized data.  The size
of the initialized data in the file for the object.  The physical
size must be a multiple of the File Align field in the PE Header, and
must be less than or equal to the Virtual Size.

PHYSICAL OFFSET = DD Physical offset for object's first page.  This
offset is relative to beginning of the EXE file, and is aligned on a
multiple of the File Align field in the PE Header.  The offset is
used as a seek value.

OBJECT FLAGS = DD Flag bits for the object.  The object flag bits
have the following definitions:

  o  000000020h __Code object.

  o  000000040h __Initialized data object.

  o  000000080h __Uninitialized data object.

  o  040000000h __Object must not be cached.

  o  080000000h __Object is not pageable.

  o  100000000h __Object is shared.

  o  200000000h __Executable object.

  o  400000000h __Readable object.

  o  800000000h __Writeable object.

All other bits are reserved for future use and sho
uld be set to zero.

4. Image Pages

The Image Pages section contains all initialized data for all
objects.  The seek offset for the first page in each object is
specified in the object table and is aligned on a File Align
boundary.  The objects are ordered by the RVA.  Every object begins
on a multiple of Object Align.



5. Exports

A typical file layout for the export information follows:

    +-----------------------+
    ¦     DIRECTORY TABLE   ¦
    +-----------------------¦
    ¦     ADDRESS TABLE     ¦
    ¦                       ¦
    ¦                       ¦
    ¦                       ¦
    ¦                       ¦
    ¦                       ¦
    ¦                       ¦
    +-----------------------¦
    ¦     NAME PTR TABLE    ¦
    ¦                       ¦
    ¦                       ¦
    ¦                       ¦
    +-----------------------¦
    ¦     ORDINAL TABLE     ¦
    ¦                       ¦
    ¦                       ¦
    ¦                       ¦
    +-----------------------¦
    ¦     NAME STRINGS      ¦
    ¦                       ¦
    ¦                       ¦
    +-----------------------+
```

Figure 4. Export File Layout

5.1 Export Directory Table

The export information begins with the Export Directory Table which describes the remainder of the export information. The Export Directory Table contains address information that is used to resolve fixup references to the entry points within this image.

```
    +-----------------------------------+
    ¦            EXPORT FLAGS           ¦
    +-----------------------------------¦
    ¦           TIME/DATE STAMP         ¦
    +-----------------------------------¦
    ¦  MAJOR VERSION  ¦   MINOR VERSION ¦
    +-----------------------------------¦
    ¦             NAME RVA              ¦
    +-----------------------------------¦
    ¦           ORDINAL BASE            ¦
    +-----------------------------------¦
    ¦           # EAT ENTRIES           ¦
    +-----------------------------------¦
    ¦            # NAME PTRS            ¦
    +-----------------------------------¦
    ¦         ADDRESS TABLE RVA         ¦
    +-----------------------------------¦
    ¦        NAME PTR TABLE RVA         ¦
    +-----------------------------------¦
    ¦         ORDINAL TABLE RVA         ¦
    +-----------------------------------+
```

Figure 5. Export Directory Table Entry

EXPORT FLAGS = DD Currently set to zero.

TIME/DATE STAMP = DD Time/Date the export data was created.

MAJOR/MINOR VERSION = DW A user settable major/minor version number.

NAME RVA = DD Relative Virtual Address of the Dll asciiz Name.
This is the address relative to the Image Base.

ORDINAL BASE = DD First valid exported ordinal.
This field specifies the starting ordinal number for the export
address table for this image. Normally set to 1.

# EAT ENTRIES = DD Indicates number of entries in the Export Address
Table.

# NAME PTRS = DD This indicates the number of entries in the
Name Ptr Table (and parallel Ordinal Table).

ADDRESS TABLE RVA = DD Relative Virtual Address of the Export Address
Table.
This address is relative to the Image Base.

NAME TABLE RVA = DD Relative Virtual Address of the Export Name Table
Pointers.
This address is relative to the beginning of the Image Base. This
table is an array of RVA's with # NAMES entries.

ORDINAL TABLE RVA = DD Relative Virtual Address of Export Ordinals
Table Entry.
This address is relative to the beginning of the Image Base.

5.2 Export Address Table

The Export Address Table contains the address of exported entrypoints
and exported data and absolutes. An ordinal number is used to index
the Export Address Table. The ORDINAL BASE must be subracted from the
ordinal number before indexing into this table.

Export Address Table entry formats are described below:

```
    +------------------------------------+
    ¦            EXPORTED RVA            ¦
    +------------------------------------+
```

Figure 6. Export Address Table Entry

EXPORTED RVA = DD Export address.
This field contains the relative virtual address of the exported
entry (relative to the Image Base).

5.3 Export Name Table Pointers

The export name table pointers array contains address into the Export
Name Table. The pointers are 32-bits each, and are relative to the
Image Base. The pointers are ordered lexically to allow binary
searches.

5.4 Export Ordinal Table

The Export Name Table Pointers and the Export Ordinal Table form two parallel arrays, separated to allow natural field alignment. The export ordinal table array contains the Export Address Table ordinal numbers associated with the named export referenced by corresponding Export Name Table Pointers.

The ordinals are 16-bits each, and already include the Ordinal Base stored in the Export Directory Table.

5.5 Export Name Table

The export name table contains optional ASCII names for exported entries in the image. These tables are used with the array of Export

Name Table Pointers and the array of Export Ordinals to translate a procedure name string into an ordinal number by searching for a matching name string. The ordinal number is used to locate the entry point information in the export address table.

Import references by name require the Export Name Table Pointers table to be binary searched to find the matching name, then the corresponding Export Ordinal Table is known to contain the entry point ordinal number. Import references by ordinal number provide the fastest lookup since searching the name table is not required.

Each name table entry has the following format:

```
    +------------------------------------+
    ¦ ASCII STRING ::: ::::::::   '\0'  ¦
    +------------------------------------+
```

Figure 7. Export Name Table Entry

ASCII STRING = DB ASCII String.
The string is case sensitive and is terminated by a null byte.

6. Imports

A typical file layout for the import information follows:

```
    +------------------------+
    ¦     DIRECTORY TABLE    ¦
    ¦                        ¦
    ¦                        ¦
    ¦                        ¦
    +------------------------¦
    ¦    NULL DIR ENTRY      ¦
    +------------------------+

    +------------------------+
    ¦   DLL1 LOOKUP TABLE    ¦
```

```
      ¦                        ¦
      +------------------------¦
      ¦         NULL           ¦
      +------------------------+

      +------------------------+
      ¦   DLL2 LOOKUP TABLE    ¦
      ¦                        ¦
      +------------------------¦
      ¦         NULL           ¦
      +------------------------+

      +------------------------+
      ¦   Dll3 LOOKUP TABLE    ¦
      ¦                        ¦
      +------------------------¦
      ¦         NULL           ¦
      +------------------------+

      +------------------------+
      ¦    HINT-NAME TABLE     ¦
      ¦                        ¦
      +------------------------+

      +------------------------+
      ¦   DLL1 ADDRESS TABLE   ¦
      ¦                        ¦
      +------------------------¦
      ¦         NULL           ¦
      +------------------------+

      +------------------------+
      ¦   DLL2 ADDRESS TABLE   ¦
      ¦                        ¦
      +------------------------¦
      ¦         NULL           ¦
      +------------------------+

      +------------------------+
      ¦   DLL3 ADDRESS TABLE   ¦
      ¦                        ¦
      +------------------------¦
      ¦         NULL           ¦
      +------------------------+
```

Figure 8. Import File Layout

6.1 Import Directory Table

The import information begins with the Import Directory Table which describes the remainder of the import information. The Import Directory Table contains address information that is used to resolve fixup references to the entry points within a DLL image. The import directory table consists of an array of Import Directory Entries, one entry for each DLL this image references. The last directory entry is empty (NULL) which indicates the end of the directory table.

An Import Directory Entry has the following format:

```
    +-----------------------------------+
 0  ¦            IMPORT FLAGS           ¦
    +-----------------------------------¦
 4  ¦           TIME/DATE STAMP         ¦
    +-----------------------------------¦
 8  ¦  MAJOR VERSION  ¦   MINOR VERSION ¦
    +-----------------------------------¦
 12 ¦              NAME RVA             ¦
    +-----------------------------------¦
 16 ¦      IMPORT LOOKUP TABLE RVA      ¦
    +-----------------------------------¦
 20 ¦      IMPORT ADDRESS TABLE RVA     ¦
    +-----------------------------------+
 24
```
Figure 9. Import Directory Entry

IMPORT FLAGS = DD Currently set to zero.

TIME/DATE STAMP = DD Time/Date the import data was pre-snapped or
zero if not pre-snapped.

MAJOR/MINOR VERSION = DW The major/minor version number of the dll
being referenced.

NAME RVA = DD Relative Virtual Address of the Dll asciiz Name.
This is the address relative to the Image Base.

IMPORT LOOKUP TABLE RVA
 = DD This field contains the address of the
start of the import lookup table for this image. The address is
relative to the beginning of the Image Base.

IMPORT ADDRESS TABLE RVA = DD This field contains the address of the
start of the import addresses for this image. The address is
relative to the beginning of the Image Base.

6.2 Import Lookup Table

The Import Lookup Table is an array of ordinal or hint/name RVA's for
each DLL. The last entry is empty (NULL) which indicates the end of
the table.

The last element is empty.

```
     3                                 0
     1
    +-Ê---------------------------------+
    ¦0¦   ORDINAL#/HINT-NAME TABLE RVA  ¦
    +-ð---------------------------------+
```

Figure 10. Import Address Table Format

```
ORDINAL/HINT-NAME TABLE RVA = 31-bits (mask = 7fffffffh) Ordinal
Number or Name Table RVA.
If the import is by ordinal, this field contains a 31 bit ordinal
number.  If the import is by name, this field contains a 31 bit
address relative to the Image Base to the Hint-Name Table.

O = 1-bit (mask = 80000000h) Import by ordinal flag.

  o  00000000h __Import by name.

  o  80000000h __Import by ordinal.

6.3 Hint-Name Table

The Hint-Name Table format follows:

    +-----------------------------------+
    ¦       HINT      ¦ ASCII STRING |||¦
    +--------+--------+--------+--------¦
    ¦||||||||||||||||¦  '\0'     PAD   ¦
    +-----------------------------------+


    The PAD field is optional.

Figure 11.  Import Hint-Name Table

HINT = DW Hint into Export Name Table Pointers.
The hint value is used to index the Export Name Table Pointers array,
allowing faster by-name imports.  If the hint is incorrect, then a
binary search is performed on the Export Name Ptr Table.

ASCII STRING = DB ASCII String.
The string is case sensitive and is terminated by a null byte.

PAD = DB Zero pad byte.
A trailing zero pad byte appears after the trailing null byte if
necessary to align the next entry on an even boundary.

The loader overwrites the import address table when loading the image
with the 32-bit address of the import.



6.4 Import Address Table

The Import Address Table is an array of addresses of the imported
routines for each DLL. The last entry is empty (NULL) which indicates
the end of the table.

7. Thread Local Storage

Thread local storage is a special contiguous block of data. Each
thread will gets its own block upon creation of the thread.

The file layout for thread local storage follows:
    +-----------------------+
    ¦     DIRECTORY TABLE   ¦
    +-----------------------+
    +-----------------------+
    ¦        TLS DATA       ¦
    +-----------------------+
    +-----------------------+
    ¦      INDEX VARIABLE   ¦
    +-----------------------+
    +-----------------------+
    ¦   CALLBACK ADDRESSES  ¦
    +-----------------------+
```

Figure 12. Thread Local Storage Layout

7.1 Thread Local Storage Directory Table

The Thread Local Storage Directory Table contains address information that is used to describe the rest of TLS.

The Thread Local Storage Directory Table has the following format:

```
    +-----------------------------------+
    ¦       START DATA BLOCK VA         ¦
    +-----------------------------------¦
    ¦        END DATA BLOCK VA          ¦
    +-----------------------------------¦
    ¦             INDEX VA              ¦
    +-----------------------------------¦
    ¦         CALLBACK TABLE VA         ¦
    +-----------------------------------+
```

Figure 13. Thread Local Storage Directory Table

START DATA BLOCK VA = DD Virtual Address of the start of the thread local storage data block.

END DATA BLOCK VA = DD Virtual Address of the end of the thread local storage data block.

INDEX VA = DD Virtual Address of the index variable used to access the thread local storage data block.

CALLBACK TABLE VA = DD Virtual Address of the callback table.

7.2 Thread Local Storage CallBack Table

The Thread
Local Storage Callbacks is an array of Virtual Address of functions to be called by the loader after thread creation and thread termination. The last entry is empty (NULL) which indicates the end of the table.

The Thread Local Storage CallBack Table has the following format:

```
    +-----------------------------------+
    ¦            FUNCTION1 VA          ¦
    +----------------------------------¦
    ¦            FUNCTION2 VA          ¦
    +----------------------------------¦
    ¦                                  ¦
    +----------------------------------¦
    ¦                NULL              ¦
    +----------------------------------+
```

Figure 14. Thread Local Storage CallBack Table

8. Resources

Resources are indexed by a multiple level binary-sorted tree structure. The overall design can incorporate 2**31 levels, however, NT uses only three: the highest is TYPE, then NAME, then LANGUAGE.

A typical file layout for the resource information follows:

```
    +-----------------------+
    ¦  RESOURCE DIRECTORY   ¦
    ¦                       ¦
    ¦                       ¦
    ¦                       ¦
    +-----------------------¦
    ¦     RESOURCE DATA     ¦
    ¦                       ¦
    ¦                       ¦
    ¦                       ¦
    ¦                       ¦
    ¦                       ¦
    ¦                       ¦
    +-----------------------+
```

Figure 15. Resource File Layout

The Resource directory is made up of the following tables:

8.1 Resource Directory Table

```
+----------------------------------+
¦           RESOURCE FLAGS         ¦
+----------------------------------¦
¦           TIME/DATE STAMP        ¦
+----------------------------------¦
¦  MAJOR VERSION  ¦   MINOR VERSION ¦
+-----------------+----------------¦
¦    # NAME ENTRY ¦  # ID ENTRY     ¦
+----------------------------------¦
¦       RESOURCE DIR ENTRIES        ¦
+----------------------------------+
```

Figure 16. Resource Table Entry

```
RESOURCE FLAGS = DD Currently set to zero.

TIME/DATE  STAMP = DD Time/Date the resource data was created by the
resource compiler.

MAJOR/MINOR VERSION = DW  A user settable major/minor version number.

# NAME ENTRY = DW The number of name entries.
This field contains the number of entries at the beginning of the
array of directory entries which have actual string names associated
with them.

# ID ENTRY = DW The number of ID integer entries.
This field contains the number of 32-bit integer IDs as their names
in the array of directory entries.

The resource directory is followed by a variable length array of
directory entries.  # NAME ENTRY is the number of entries at the
beginning of the array that have actual names associated with each
entry.  The entires are in ascending order, case insensitive strings.
# ID ENTRY identifies the number of entries that have 32-bit integer
IDs as their name.  These entries are also sorted in ascending order.

This structure allows fast lookup by either name or number, but for
any given resource entry only one form of lookup is supported, not
both. This is consistent with the syntax of the .RC file and the .RES
file.



The array of directory entries have the following format:
 3                                0
 1
+------------------------------------+
¦          NAME RVA/INTEGER ID       ¦
+-Ê----------------------------------¦
¦E¦     DATA ENTRY RVA/SUBDIR RVA    ¦
+-ð----------------------------------+

Figure 17.  Resource Directory Entry


INTERGER ID = DD ID.
This field contains a integer ID field to identify a resource.

NAME RVA = DD Name RVA address.
This field contains a 31-bit address relative to the beginning of the
Image Base to a Resource Directory String Entry.

E = 1-bit (mask 80000000h) Unescape bit.
This bit is zero for unescaped Resource Data Entries.

DATA RVA = 31-bits (mask 7fffffffh) Data entry address.
This field contains a 31-bit address relative to the beginning of the
Image Base to a Resource Data Entry.
```

```
E = 1-bit (mask 80000000h) Escape bit.
This bit is 1 for escaped Subdirectory Entry.

DATA RVA = 31-bits (mask 7fffffffh) Directory entries.
This field contains a 31-bit address relative to the beginning of the
Image Base to Subdirectory Entry.



Each resource directory string entry has the following format:
+------------------------------------+
¦      LENGTH     ¦ UNICODE STRING  ¦
+--------+--------+--------+--------¦
¦                                   ¦
+------------------------------------+

Figure 18.  Resource Directory String Entry


LENGTH = DW Length of string.

UNICODE STRING = DW UNICODE String.

All of these string objects are stored together after the last
resource directory entry and before the first resource data object.
This minimizes the impact of these variable length objects on the
alignment of the fixed size directory entry objects. The length needs
to be word aligned.



Each Resource Data Entry has the following format:

    +------------------------------------+
    ¦               DATA RVA             ¦
    +------------------------------------¦
    ¦                SIZE                ¦
    +------------------------------------¦
    ¦               CODEPAGE             ¦
    +------------------------------------¦
    ¦               RESERVED             ¦
    +------------------------------------+

Figure 19.  Resource Data Entry



DATA RVA = DD Address of Resource Data.
This field contains 32-bit virtaul address of the resource data
(relative to the Image Base).

SIZE = DD Size of Resource Data.
This field contains the size of the resource data for this resource.

CODEPAGE = DD Codepage.
```

RESERVED = DD Reserved - must be zero.

Each resource data entry describes a leaf node in the resource directory tree. It contains an address which is relative to the beginning of Image Base, a size field that gives the number of bytes of data at that address, a CodePage that should be used when decoding code point values within the resource data. Typically for new applications the code page would be the unicode code page.

8.2 Resource Example

The following is an example for an app. which wants to use the following data as resources:

TypeId#	NameId#	Language ID	Resource Data
00000001	00000001	0	00010001
00000001	00000001	1	10010001
00000001	00000002	0	00010002
00000001	00000003	0	00010003
00000002	00000001	0	00020001
00000002	00000002	0	00020002
00000002	00000003	0	00020003
00000002	00000004	0	00020004
00000009	00000001	0	00090001
00000009	00000009	0	00090009
00000009	00000009	1	10090009
00000009	00000009	2	20090009

Then the Resource Directory in the Portable format looks like:

```
Offset         Data
0000:   00000000 00000000 00000000 00030000  (3 entries in this directory)
0010:   00000001 80000028
    (TypeId #1, Subdirectory at offset 0x28)
0018:   00000002 80000050     (TypeId #2, Subdirectory at offset 0x50)
0020:   00000009 80000080     (TypeId #9, Subdirectory at offset 0x80)
0028:   00000000 00000000 00000000 00030000  (3 entries in this directory)
0038:   00000001 800000A0     (NameId #1, Subdirectory at offset 0xA0)
0040:   00000002 00000108     (NameId #2, data desc at offset 0x108)
0048:   00000003 00000118     (NameId #3, data desc at offset 0x118)
0050:   00000000 00000000 0000000000040000  (4 entries in this directory)
0060:   00000001 00000128     (NameId #1, data desc at offset 0x128)
0068:   00000002 00000138     (NameId #2, data desc at offset 0x138)
0070:   00000003 00000148     (NameId #3, data desc at offset 0x148)
0078:   00000004 00000158     (NameId #4, data desc at offset 0x158)
0080:   00000000 00000000 00000000 00020000  (2 entries in this directory)
0090:   00000001 00000168     (NameId #1, data desc at offset 0x168)
0098:   00000009 800000C0     (NameId #9, Subdirectory at offset 0xC0)
00A0:   00000000 00000000 00000000 00020000  (2 entries in this directory)
00B0:   00000000 000000E8     (Language ID 0, data desc at offset 0xE8
00B8:   00000001 000000F8     (Language ID 1, data desc at offset 0xF8
00C0:   00000000 00000000 00000000 00030000  (3 entries in this directory)
00D0:   00000001 00000178     (Language ID 0, data desc at offset 0x178
00D8:   00000001 00000188     (Language ID 1, data desc at offset 0x188
00E0:   00000001 00000198     (Language ID 2, data desc at offset 0x198

00E8:   000001A8  (At offset 0x1A8, for TypeId #1, NameId #1, Language id #0
        00000004  (4 bytes of data)
        00000000  (codepage)
        00000000  (reserved)
00F8:   000001AC  (At offset 0x1AC, for TypeId #1, NameId #1, Language id #1
        00000004  (4 bytes of data)
        00000000  (codepage)
        00000000  (reserved)
0108:   000001B0  (At offset 0x1B0, for TypeId #1, NameId #2,
        00000004  (4 bytes of data)
        00000000  (codepage)

        00000000  (reserved)
0118:   000001B4  (At offset 0x1B4, for TypeId #1, NameId #3,
        00000004  (4 bytes of data)
        00000000  (codepage)
        00000000  (reserved)
0128:   000001B8  (At offset 0x1B8, for TypeId #2, NameId #1,
        00000004  (4 bytes of data)
        00000000  (codepage)
        00000000  (reserved)
0138:   000001BC  (At offset 0x1BC, for TypeId #2, NameId #2,
        00000004  (4 bytes of data)
        00000000  (codepage)
        00000000  (reserved) 0
148:   000001C0  (At offset 0x1C0, for TypeId #2, NameId #3,
        00000004  (4 bytes of data)
        00000000  (codepage)
        00000000  (reserved)
0158:   000001C4  (At offset 0x1C4, for TypeId #2, NameId #4,
        00000004  (4 bytes of data)
        00000000  (codepage)
        00000000  (reserved)
0168:   000001C8  (At offset 0x1C8, for TypeId #9, NameId #1,
        00000004  (4 bytes of data)
        00000000  (codepage)
        00000000  (reserved)
0178:   000001CC  (At offset 0x1CC, for TypeId #9, NameId #9, Language id #0
        00000004  (4 bytes of data)
        00000000  (codepage)
        00000000  (reserved)
0188:   000001D0  (At offset 0x1D0, for TypeId #9, NameId #9, Language id #1
        00000004  (4 bytes of data)
        00000000  (codepage)
        00000000  (reserved)
0198:   000001D4  (At offset 0x1D4, for TypeId #9, NameId #9, Language id #2
        00000004  (4 bytes of data)
        00000000  (codepage)
        00000000  (reserved)

And the data for t
he resources will look like:
01A8:          00010001
01AC:          10010001
```

```
01B0:          00010002
01B4:          00010003
01B8:          00020001
01BC:          00020002
01C0:          00020003
01C4:          00020004
01C8:          00090001
01CC:          00090009
01D0:          10090009
01D4:          20090009
```

9. Fixup Table

The Fixup Table contains entries for all fixups in the image. The Total Fixup Data Size in the PE Header is the number of bytes in the fixup table. The fixup table is broken into blocks of fixups. Each block represents the fixups for a 4K page.

Fixups that are resolved by the linker do not need to be processed by the loader, unless the load image can't be loaded at the Image Base specified in the PE Header.

9.1 Fixup Block

Fixup blocks have the following format:

```
    +-----------------------------------+
    ¦             PAGE RVA              ¦
    +-----------------------------------¦
    ¦            BLOCK SIZE             ¦
    +-----------------------------------¦
    ¦   TYPE/OFFSET   ¦   TYPE/OFFSET   ¦
    +-----------------+-----------------¦
    ¦   TYPE/OFFSET   ¦       ...       ¦
    +-----------------------------------+
```

Figure 20. Fixup Block Format

To apply a fixup, a delta needs to be calculated. The 32-bit delta is the difference between the preferred base, and the base where the image is actually loaded. If the image is loaded at its preferred base, the delta would be zero, and thus the fixups would not have to be applied. Each block must start on a DWORD boundary. The ABSOLUTE fixup type can be used to pad a block.

PAGE RVA = DD Page RVA. The image base plus the page rva is added to each offset to create the virtual address of where the fixup needs to be applied.

BLOCK SIZE = DD Number of bytes in the fixup block. This includes the PAGE RVA and SIZE fields.

TYPE/OFFSET is defined as:

```
     1    1          0
```

```
  5    1
 +----Ê------------+
 ¦TYPE¦   OFFSET   ¦
 +----ð------------+
```

Figure 21. Fixup Record Format

TYPE = 4-bit fixup type. This value has the following definitions:

o 0h __ABSOLUTE. This is a NOP. The fixup is skipped.

o 1h __HIGH. Add the high 16-bits of the delta to the 16-bit field at Offset. The 16-bit field represents the high value of a 32-bit word.

o 2h __LOW. Add the low 16-bits of the delta to the 16-bit field at Offset. The 16-bit field represents the low half value of a 32-bit word. This fixup will only be emitted for a RISC machine when the image Object Align isn't the default of 64K.

o 3h __HIGHLOW. Apply the 32-bit delta to the 32-bit field at Offset.

o 4h __HIGHADJUST. This fixup requires a full 32-bit value. The high 16-bits is located at Offset, and the low 16-bits is located in the next Offset array element (this array element is included in the SIZE field). The two need to be combined into a signed variable. Add the 32-bit delta. Then a dd 0x8000 and store the high 16-bits of the signed variable to the 16-bit field at Offset.

o 5h __MIPSJMPADDR.

All other values are reserved.

10. Debug Information

The debug information is defined by the debugger and is not controlled by the portable EXE format or linker. The only data defined by the portable EXE format is the Debug Directory Table.

10.1 Debug Directory

The debug directory table consists of one or more entries that have the following format
:

```
 +------------------------------------+
 ¦             DEBUG FLAGS            ¦
 +------------------------------------¦
 ¦           TIME/DATE STAMP          ¦
 +------------------------------------¦
 ¦  MAJOR VERSION  ¦  MINOR VERSION   ¦
 +------------------------------------¦
```

```
¦               DEBUG TYPE             ¦
+-------------------------------------¦
¦               DATA SIZE              ¦
+-------------------------------------¦
¦               DATA RVA               ¦
+-------------------------------------¦
¦               DATA SEEK              ¦
+-------------------------------------+
```

Figure 22. Debug Directory Entry

```
DEBUG FLAGS = DD Set to zero for now.

TIME/DATE STAMP = DD Time/Date the debug data was created.

MAJOR/MINOR VERSION = DW Version stamp.
This stamp can be used to determine the version of the debug data.

DEBUG TYPE = DD Format type.
To support multiple debuggers, this field determines the format of
the debug informati
on. This value has the following definitions:

  o  0001h __Image contains COFF symbolics.

  o  0001h __Image contains CodeView symbolics.

  o  0001h __Image contains FPO symbolics.

DATA SIZE = DD The number of bytes in the debug data. This is the
size of the actual debug data and does not include the debug
directory.

DATA RVA = DD The relative virtual address of the debug data. This
address is relative to the beginning of the Image Base.

DATA SEEK = DD The seek value from the beginning of the file to the
debug data.

If the image contains more than one type of debug information, then
the next debug directory will immediately follow the first debug
directory.
```

1:1