Microsoft®
Operating System/2

Programmer's Toolkit

Programmer's Reference

Version 1.0

Microsoft Corporation

Document No. 060060014-100-R00-0388
Part No. 01887

D.1 Introduction

This appendix describes the format of the MS OS/2 executable files. This file format is used for both programs and dynamic-link libraries. The linker creates the executable file by using object files, run-time and import libraries, and a module-definition file. Figure D.1 shows the executable file format:

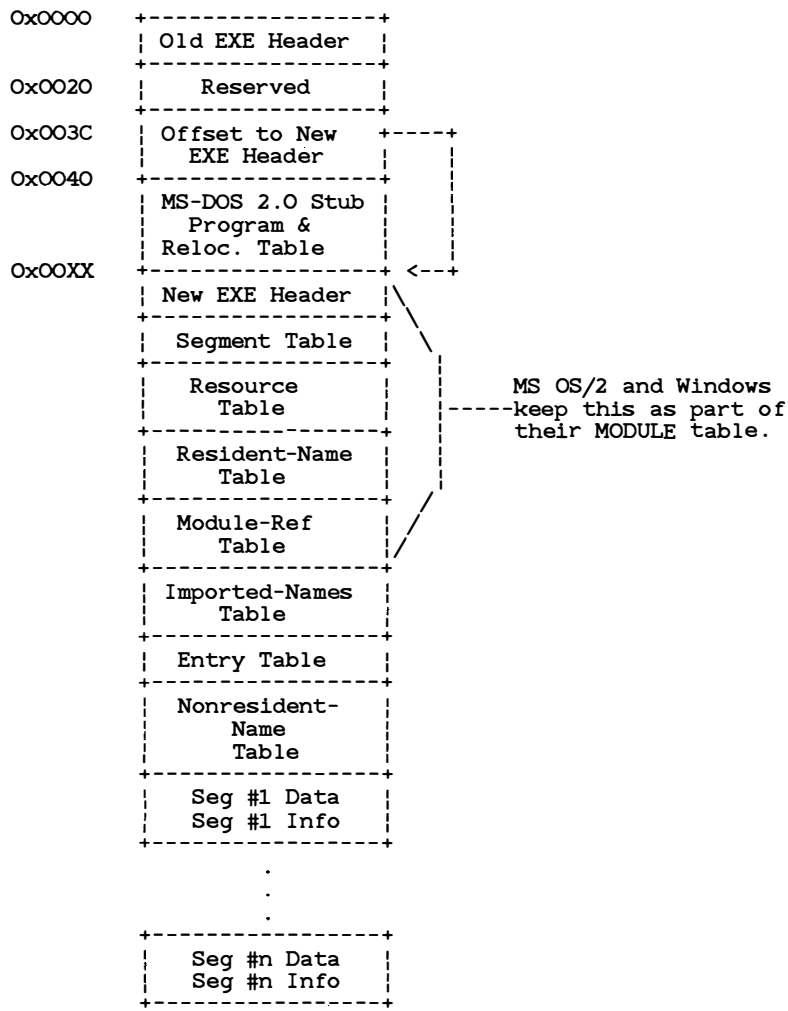

Figure D.1 Executable-File Format

D.2 MS-DOS Executable Header

An MS OS/2 executable file starts with a slightly modified MS-DOS executable header followed by the MS-DOS program data. The MS OS/2 executable file is a concatenation of an MS-DOS program and an MS OS/2 program. The MS-DOS program is usually a small stub program that displays an error message if a user loads and runs the executable file in MS-DOS. However, the linker can substitute a complete MS-DOS program for the stub.

In an MS-DOS executable file, the word at offset 0x0018 contain the relative byte offset to the relocation table. In an MS OS/2 executable file, this word is set to 0x0040 to indicate that the format of the executable file is for MS OS/2. In such cases, the double word at offset 0x003C is the relative byte offset from the beginning of the MS OS/2 executable file to the beginning of the executable file header.

The following list shows the contents of the various fields in the MS-DOS program portion of the MS OS/2 executable file:

Field	Description
0x0000–0x001F	Contains the header for the MS-DOS executable file.
0x0018	Contains the value 0x0040 to indicate an MS OS/2 executable file.
0x0020–0x003B	Contains reserved values.
0x003C–0x003F	Contains a double word that specifies the offset from the beginning of the MS OS/2 file to the start of the MS OS/2 executable file header.
0x0040	Contains the first byte of the MS-DOS program. The length is defined in the MS-DOS executable file header.

D.3 New Executable Header

The new executable header defines the location and size of the various tables and segments in the executable file. The MS OS/2 loader uses this header to create a module table for each program and each dynamic-link library. Specifically, it uses the fields that start at offset 0x0008.

Many fields in the executable header use segment numbers to identify segments in the program or library. A segment number is an index into the module's segment table. The first entry in the segment table is number 1.

In the following list, 0x0000 is specified as the starting offset of the executable file. The actual starting offset depends on the length of the MS-DOS program at the beginning of the executable file. The executable file header always starts immediately after the end of the MS-DOS program.

Field	Description
0x0000	Specifies the signature of the MS OS/2 executable file. It is 0x454E (that is, the letters "NE" for "new executable").
0x0002	Contains the linker version and the revision number. The linker version is in the low-order byte, the revision number is in the high-order byte.
0x0004	Specifies the offset from the beginning of the file to the entry table.
0x0006	Specifies the number of bytes in the entry table.
0x0008–0x000A	Specifies the CRC-32 of the entire contents of the file (with the following words taken as zero during the calculation).
0x000C	Specifies the flag word. It can be a combination of the following values:

Value	Meaning
0x0000	No automatic data segment.
0x0001	Single data segment (nonshared).
0x0002	Multiple data segments (shared).
0x0004	Run in real mode.
0x0008	Run in protected mode.
0x2000	Errors detected at link time.
0x4000	Non-conforming program (a valid stack is not maintained).
0x8000	Library module.

The 0x0001 and 0x0002 values cannot be used together. The 0x0001 value is required for programs or libraries that export functions for dynamic linking.

0x000E	Specifies the segment number of the automatic data segment. This field is set to zero if the module has no automatic data segment.
0x0010	Specifies the initial size in bytes of the dynamic heap added to the data segment. This field is set to zero if there is no heap.

0x0012	Specifies the initial size in bytes of the stack added to the data segment. This field is set to zero if there is no stack.
0x0014–0x0016	Specifies the starting address of the program or of the library's initialization function. The low-order word is the offset (**IP**); the high-order word is the segment number of the starting segment (**CS**). The address specifies the entry point of the initialization function only if the executable file defines a library module.
0x0018–0x001A	Specifies the starting address of the stack. The low-order word is the offset (**SP**); the high-order word is the segment number of the stack segment (**SS**). If the number of the stack segment is the same as the number of the automatic data segment and the offset is zero, the loader creates a stack in the automatic data segment and sets the starting address to be the address of the last word in that stack. This field is ignored if the executable file defines a library module.
0x001C	Specifies the number of entries in the segment table.
0x001E	Specifies the number of bytes in the nonresident-name table.
0x0020	Specifies the offset of the segment table relative to the beginning of the new executable header.
0x0022	Specifies the offset of the resource table relative to the beginning of the new executable header.
0x0024	Specifies the offset of the resident-name table relative to the beginning of the new executable header.
0x0026	Specifies the offset of the module-reference table relative to the beginning of the new executable header.
0x0028	Specifies the offset of the imported-names table relative to the beginning of the new executable header.
0x002A–0x002C	Specifies the offset of the nonresident-name table relative to the beginning of the file.
0x002E	Specifies the number of movable entry points.
0x0030	Specifies the shift count of the logical sector alignment. The alignment specifies the byte boundary on which a segment starts. It is expressed as an exponent of 2; for the default alignment of 512 bytes, the shift count 9 is given.
0x0032–0x003E	Specifies reserved values.

The system loader creates a stack for a program if the stack segment is the same as the automatic data segment. The loader adds the stack to the end of the data segment, setting the stack pointer to the top of the automatic data segment. If the data segment also has a heap area, the stack is between the data and heap.

D.4 Segment Table

The segment table defines the location, size, and type of the code and data segments of the module. The segment table contains one or more entries. Each entry specifies the location of the code and data segments in the executable file, the size of the segment data in the file, the size of the segment when loaded into memory, and a flag word that specifies the segment type. The number of segment entries in the table is specified by field 0x001C in the executable file header.

Many fields in the executable file header use segment numbers to identify segments in this table. Segment number 1 identifies the first segment table entry, number 2 the second, and so on.

Each entry in the segment table has the following fields:

Field	Description
0x0000	Specifies the logical sector offset to the contents of the segment data relative to the beginning of the file. The actual offset is computed by multiplying this offset with the logical sector size as defined by field 0x0030 in the executable file header. This field is zero if there is no file data.
0x0002	Specifies the length in bytes of the segment in the file. It is zero if the segment is 65,536 bytes.
0x0004	Specifies the flag word. It can be a combination of the following values:

Value	Meaning
0x0000	Code-segment type.
0x0001	Data-segment type.
0x0008	Segment data is iterated.
0x0010	Segment is movable.
0x0020	Segment can be shared.

	0x0040	Segment is not demand loaded.
	0x0080	Segment is execute-only if code, or read-only if data.
	0x0100	Set if segment has relocation records.
	0x0200	Set if segment has debug information.
	0x0C00	Reserved for 286 DPL bits.
	0xF000	Discard priority.
0x0006	Specifies the minimum allocation size of the segment in bytes. This may be larger than the size of the segment in the file. If this field is zero, the minimum size is 65,536 bytes.	

D.5 Resource Table

The resource table specifies the location, size, type, and name of resources in the module. Resources are additional data that may be loaded from the executable file as needed by a program or library. Figure D.2 shows the form of the resource table:

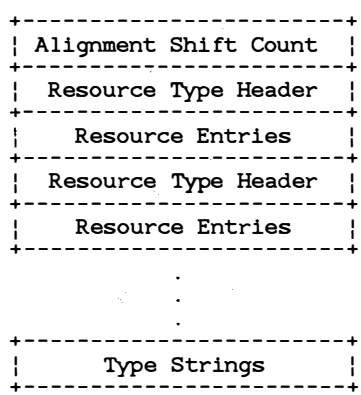

Figure D.2 Resource Table

The first word in the resource table specifies the alignment shift count for resource data. The alignment shift count is an exponent of 2 that defines the number of bytes in each alignment sector. The location of a resource in the executable file is computed by multiplying its alignment offset by the alignment sector size.

The resource table contains two or more resource-type headers. Each header has the following fields:

Field	Description
0x0000	Specifies the type identifier. It is an integer type if the high-order bit is set (0x8000). Otherwise, it is an offset to a type string, relative to the beginning of the resource table. If this field is zero, it specifies the end of the resource records.
0x0002	Specifies the number of resources for the type.
0x0004–0x0006	Specifies a reserved value.

The resource-type header is immediately followed by the specified number of resource entries. Each resource entry has the following fields:

Field	Description
0x0000	Specifies the alignment offset to the contents of the resource data, relative to the beginning of the file. The offset is given in terms of alignment units specified at the beginning of the resource table.
0x0002	Specifies the length in bytes of the resource in the file.
0x0004	Specifies the flag word. It can be a combination of the following values:

Value	Meaning
0x0010	Resource is movable.
0x0020	Resource can be shared.
0x0040	Resource is not demand loaded.

0x0006	Specifies the resource identifier. It is an integer type if the high-order bit is set (0x8000). Otherwise, this field is the offset to the resource string relative to the beginning of the resource table.
0x0008–0x000A	Specifies a reserved value.

Resource type and name strings are stored at the end of the resource table. The first byte of each string specifies the length of the string in bytes. If it is zero, it specifies the end of the resource table. The strings can contain any characters. They are not null-terminated.

D.6 Module-Reference Table

The module-reference table specifies the location of the names of the modules imported by the module. The names are character strings and are stored in the imported-names table.

The table contains one or more entries. Each entry specifies the offset (within the imported-names table) to the module-name string. Each entry has a unique module reference. A module reference is an index into the module-reference table. Module reference 1 identifies the first entry, 2 the second, and so on. Other tables use module references to identify the name of the module that contains a given imported function.

D.7 Entry-Point Table

The entry-point table defines the segment, offset, and type of entry points used in the module. The table contains one or more bundles. Each bundle describes the entry points of a given segment. Each bundle starts with the following fields:

Field	Description
0x0000	Specifies the number of entries in this bundle. All records in one bundle either are movable or refer to the same fixed segment. This field is zero if there are no more bundles in the entry table.
0x0001	Specifies the segment indicator for this bundle. If it is 0xFF, it specifies a movable segment, and the segment number is in the following 6-byte entry. If it is any other value (0x00 is not used), it is the segment number of a fixed segment.

For movable segments, the entry record has the following fields:

Field	Description
0x0000	Specifies the flags. If is it 0x01, the entry point is exported. If it is 0x02, the entry point uses a shared data segment.
0x0001	Specifies an **int 3Fh** instruction.
0x0003	Specifies the number of the entry-point segment.
0x0004	Specifies the offset to the entry point.

For fixed segments, the entry record has the following fields:

Field	Description
0x0000	Specifies the flags. If is it 0x01, the entry point is exported. If it is 0x02, the entry point uses a shared data segment.
0x0001	Specifies the offset to the entry point.

For movable and fixed entry points that use shared data segments, the following instruction must be the first instruction in the prologue of the entry point:

mov ax, *ds-value*

This flag may be set only for SINGLEDATA library modules.

Each entry point has a unique ordinal value. The ordinal value is an index into the entry-point table. The first entry point described in the table has ordinal value 1, the second has 2, and so on. When the loader searches for an entry point, it scans the bundles until it finds the segment that contains the entry point. It then multiplies the ordinal value by the entry size to index the proper entry.

The linker forms bundles in the densest manner it can, under the restriction that it cannot reorder entry points to improve bundling. The reason for this restriction is that other executable files may refer to entry points within this bundle by ordinal value instead of by name.

D.8 Resident- and Nonresident-Name Tables

The resident- and nonresident-name tables specify the names of the entry points in the module. The tables contain one or more entries. Each entry has the form shown in Figure D.3:

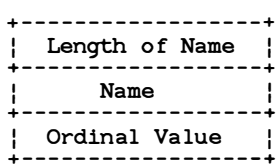

Figure D.3 Resident- and Nonresident-Name Table Entry Format

The first field is a byte that specifies the length of the string. If the field is zero, it specifies the end of the table. The second field is the name, a character string. The string is not null-terminated. The last field is the entry-point ordinal value.

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

D.9 Imported-Names Table

The imported-names table contains the names of modules that contain entry points imported by the module. The table contains one or more entries. Each entry starts with a byte that specifies the length of the name in bytes. If the byte is zero, it specifies the end of the table. The byte is followed immediately by the name, a character string. The string is not null-terminated.

D.10 Segments

The segments contain the data that is to be loaded into memory for the program or library. The format of the data depends on the segment type, as specified by field 0x0004 in the segment table.

If the segment is not iterated, it consists of the number of bytes specified in field 0x0002 in the segment table. If the segment has iterated data, the segment has the following fields:

Field	Description
0x0000	Specifies the number of iterations.
0x0002	Specifies the number of bytes of data.
0x0004	Specifies the data bytes to be repeated.

If the segment has relocation information, the relocation information consists of one or more relocation items. The relocation information starts with a word that specifies the number of relocation items. Each relocation item has the following fields:

Field	Description
0x0000	Specifies the relocation type. It can be one of the following values: **Value** **Meaning** 0x02 16-bit segment selector or address. 0x03 32-bit address. 0x05 16-bit address offset.
0x0001	Specifies the relocation flags. It can be one of the following values: **Value** **Meaning** 0x00 Internal reference. 0x01 Imported by ordinal. 0x02 Imported by name. 0x04 Additive reference.
0x0002	Specifies the offset within the segment of the source chain. If the additive flag is set, then add the target value to the source contents instead of replacing the source and following the chain. The source chain is a linked list within this segment of all references to the target. The list ends with the value 0xFFFF.

The source for each relocation has a specific format based on the relocation flags. The following lists define each of the source formats.

An INTERNALREF source has the following fields:

Field	Description
0x0000	Specifies the segment number for the fixed segment, or 0xFF if the segment is movable.
0x0001	Specifies a reserved value. It must be zero.
0x0002	Specifies the ordinal value of the entry point if the segment is movable. Otherwise, it specifies the offset into the segment.

An IMPORTNAME source has the following fields:

Field	Description
0x0000	Specifies the module reference. For more information, see Section D.6.
0x0002	Specifies the offset within the imported-names table to the function name.

An IMPORTORDINAL source has the following fields:

Field	Description
0x0000	Specifies the module reference. For more information, see Section D.6.
0x0002	Specifies the ordinal value for the imported function.

If a segment has debugging information, the first word specifies the number of bytes of debugging information. The remaining bytes are the actual debugging information.