## ♦ WordPerfect Document File Format

This manual provides the detailed documentation necessary for developers to interpret the contents of WordPerfect® files and to create new WordPerfect files. The documentation covers the prefix packets and function codes for WordPerfect version 7. Definitions of prefix packets and function codes that will appear in future releases of WordPerfect are also included. Details of the file header are given in this introductory section, which also compares WordPerfect 6.*x* and WordPerfect 5.1 file formats.

Information about the SET file used by WordPerfect for DOS is included at the end of the manual.

## ♦ Manual Conventions

As you read this manual, you should understand the following conventions to help you locate and interpret the information you need.

### Decimal and Hexadecimal Values

Computers perform operations and handle data in binary form, which can be readily represented with hexadecimal numbers. Conversion between decimal and hexadecimal values is not obvious. In this document, values will generally be shown as decimal numbers followed by the hexadecimal equivalent in parentheses. The hexadecimal value will be represented as number that begins with "0x" followed by the actual value, such as (0xFF). In most cases values are unsigned and exceptions to this rule will be noted.

### Text Characters

The character <0 (0x00)> has special meaning as the null character and is always deleted by WordPerfect. All values from <1 (0x01)> to <127 (0x7F)> are characters and are mappable to WordPerfect extended characters. For a description of these characters, see *Default Extended International Characters* under *Single-Byte Characters and Functions* later in this section.

### Size Definitions

Sizes are referred to as bytes, short integers (sometimes abbreviated as *short*), or long integers (sometimes abbreviated as *long*). Depending on the environment and operating system, these terms can mean different things. Fields are depicted with the field name enclosed in brackets. The brackets indicate the size of the field. Use the table below to match the bracket types with the size and terms they represent.

| Terms | Names | Brackets | Size |
|---|---|---|---|
| Byte | character (char), byte | <> | 8 bits |
| Short | short integer, word | [] | 16 bits |
| Long | long integer, double word | {} | 32 bits |
| Quadword | | ⟦⟧ | 64 bits |

The byte sequence of all multi-byte data types that are larger than a byte follows the Intel convention of placing the least-significant byte first. The following block diagrams show this coding format for short integer, long integer, and quadword.

| Screen | **JXT524** | |
|---|---|---|

**Fields with Bit Flags**
Some fields have bit flags. Individual bits are specified by a bit number. The range of bits for a byte value is from 0 to 7, with bit 0 the rightmost or least significant bit, and bit 7 the leftmost or most significant bit.

**Function Codes**
The brackets shown in the table under *Size Definitions* above are used to describe the size of the individual fields within a packet or a function code. Unless otherwise specified, byte values, 16-bit short values, and 32-bit long values are unsigned. If a field is variable in length, it is represented with " x ?" following the field.

Examples:

If a field contains 5 bytes, it is represented as:

► <byte field description> x 5◄

If a field contains an indefinite number of short values, such as a null-terminated word string, it is represented as:

► [short field description] x ?◄

If a field contains 2 long values, it is shown as:

► {long field description} x 2◄

**Indentation**
Indentation is used in this document to distinguish levels of detail and to signal something unique about the information. Most flag fields require a definition of

the meaning of each bit used in the flag. The definition will be indented under the flag field to give a visual indication that it contains additional information about the previous field. Some data fields may or may not exist in a particular instance of a function. Generally a flag bit is used to indicate whether or not these fields are present. These field definitions will be indented to show that they may not exist in a specific instance of the function. An example follows:

```
►  <function> <subfunction> [size] <flags = 0 or PRFXID>
   If the prefix ID bit (PRFXID) is set, the following information exists:
         [number of PIDs]
         [first PID]
         . . .
         [last PID]
   [data field 1]
   <flag field>
         bit 0:       1 = more data follows
         If bit 0 is set, the following data exists:
   [data field 2]
   [data field 3]
         bit 1:       1 = meaning of bit 1
   {data field 4}
   [size] <function>◄
```

The above example illustrates how indentation is used to give visual clues to the data content. If the function flags byte is 0, the PID information is omitted. Bits 0 and 1 are defined for the flag field and in this case bits 2-7 are not used. Bit 0 indicates whether or not data fields 2 and 3 are present.

**WordPerfect Word Strings**

Some fields in packets and in functions hold text that is marked as *WP word*, or *word* strings. The reference to *word* is to the Intel assembly language term for an unsigned short integer. In this format each character of a string takes up one short integer. The high byte is the number of the WordPerfect character set. The low byte contains an offset value into the character set that represents the position of the actual character.

WordPerfect word strings require that all characters in the string have 16-bit values including any null terminator. However, byte strings of 8-bit characters can have 16-bit characters embedded within the string. This is accomplished with a function code that shows the beginning of a text block using 16-bit characters, and the same function code is repeated to show the end of the block. For the format of this code, see the Extended Character function 240 (0xF0) under *Fixed-Length Multi-Byte Functions*.

**Units of Measure**

WPU stands for WordPerfect Unit, which is one 1200th of an inch. Dimensions are usually given in WordPerfect Units.

WPFP stands for WordPerfect Fixed Point Value. This is an unsigned 16-bit

number which represents a fraction between 0 and 1. 0x8000 is equal to 0.5 and 0xFFFF is treated as 1.0. It is used to specify a percentage value or a fraction. It is also used as the fractional part of WPSP.

WPSP is used to specify spacing values. WPSP denotes a 32-bit value composed of a 16-bit fraction (WPFP) and a 16-bit integer in that order. The fractional value is always positive. The associated integer value is signed which allows the values to be added as though they were one 32-bit value. For example, to code the number −3.75 the integer would be −4 and the fraction would be +0.25 (0x4000). When the integer and fraction are added, the result is −3.75.

PSU stands for Printer Scalable Unit, which is in 10,000ths of the point size of the font.

Font point sizes are given in 3600ths of an inch and are denoted as 3600ths.

**RGB, RGBS, RGBT**

RGB is used to mean the percent of red, percent of green, and percent of blue in specifying colors. Each color takes one byte with a range from 0 to 255 (0xFF) where 255 is 100%. The numeric value of 127 (0x7F) is calculated to be 50%. This 3-byte field in a function definition will be represented as:

► <color (RGB)> x 3◄

RGBS includes the three bytes of color information above and adds one byte for percent of shading. Shading also has a range of 0 to 255 (0xFF) where 255 is 100%. This 4-byte field in a function definition will be represented as:

► <color (RGBS)> x 4◄

RGBT includes the three bytes of color information above and adds one byte for percent of transparency. Transparency also has a range of 0 to 255 (0xFF) where 255 is 100%. This 4-byte field in a function definition will be represented as:

► <color (RGBT)> x 4 ◄

## ♦ WordPerfect Document File Formats

### WP5.*x* and WP6.*x* File Formats Compared

The 16-byte file header for WordPerfect 6.*x* (WP6.*x*) has essentially the same format as WP5.*x* with different values for major and minor versions of the document. The document is still separated into the document prefix and the document area which follows the prefix.

In WP5.*x*, the file header is always 16 bytes long. In WP6.*x* the file header can be

16 bytes or longer. When the file header is longer than 16 bytes, it is called an extended file header.

In WP5.*x* documents, the file prefix is in blocks of five indexes with the relative data following each index block. In WP6.*x* there is only one index block, which contains all the indexes needed for the document. Prefix or packet data follows the index block. References to prefix IDs refer to indexes in the prefix.

The values for most single-byte function codes have changed for WP6.*x*. The structure of the variable-length multi-byte function codes has also changed.

Fixed-length multi-byte function codes start at 240 (0xF0) for WP6.*x* (instead of 192 (0xC0) as in WP5.1), but the structure is basically the same as WP5.1. There are some fixed-length function codes which only appear as subfunctions within other function codes. These will be referred to as *embedded subfunctions*. An example is the End-of-Line Group subfunctions, function 208 (0xD0), subfunctions 128 (0x80) to 141 (0x8D). They can only appear in the non-deletable part of the End-of-Line Group function codes, function 208 (0xD0), subfunctions 0 (0x00) to 28 (0x1C). End-of-Line Group embedded subfunctions are valid only when found within an End-of-Line Group function code. Some End-of-Line Group subfunctions are fixed length.

**WP5.*x* and WP6.*x* Function Codes Compared**

In WP5.*x*, the size field (short integer value) of a function code is the size of the function minus 4. In WP6.*x*, the size field is the size of the entire function. The size field still appears at the beginning and the end of the function code.

In WP5.*x* documents, a variable-length function code begins with group, subgroup, and the size. It ends with the size, subgroup, and group.

| **5.*x* Format** | **☺!!Size** |
|---|---|
| <group> | byte |
| <subgroup> | byte |
| [size] | short |
| ... | function data |
| ... | function data |
| [size] | short |
| <subgroup> | byte |
| <group> | byte |

In WP6.*x* documents, a variable-length function code begins with group, subgroup, and the size. It ends with the size and group.

| 6.*x* Format | ☺ !!Size |
|---|---|
| <group> | byte |
| <subgroup> | byte |
| [size] | short |
| ... | ... |
| ... | non-deletable function data. |
| ... | deletable function data (if it exists) |
| [size] | short |
| <group> | byte |

For more detail, see *Variable-Length Multi-Byte Functions* later in this section.

♦ **WordPerfect 7 File Format**

This section contains an overview of the file format for WordPerfect 7 documents. Detailed information about the packet and function formats follows the overview.

**File Structure**
A WordPerfect 7 document file is a binary file containing four distinct areas: the file header, the index area, the packet data area, and the document area. Everything preceding the document area is called the prefix area.

| Host Def Table | **JXT525** | |
|---|---|---|
| **WordPerfect 7 DOCUMENT** | WPCorp File Header<br>(May be a 16-byte file header or an extended file header longer than 16 bytes) | PREFIX |
| | Index Area<br>(Indexes point to packets in the packet data area) | |
| | Packet Data Area<br>(Data pointed to by indexes in the index area) | |
| | Document Area<br>(This includes text; single-byte functions; variable-length multi-byte functions; and fixed-length multi-byte functions) | |

The following table shows the components of a document file in more detail. The shaded area of the table is the file header.

| Host Def Table | JXT526 | |
|---|---|---|

| -1 | 'W' | 'P' | 'C' | Pointer to Document Area | Product | File Type | MjrV | MnrV | Encrypt | Ptr to Index Area | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x02 | 0x00 | # Indexes | | Reserved x 10 | | | | | | Flags | Packet Type |
| Packet Use Count | | Hidden Use Count | | Packet Data Length | | Pointer to Data 0 | | | Flags | Packet Type | Pkt Use Count |
| Hidden Use Count | | Packet Data Length | | Pointer to Data 1 | | . . . . . | | | | | |
| Data 0 Data 0 Data 0 Data 0 | | | | | | | | | | | |
| Data 1 Data 1 Data 1 Data 1 | | | | | | | | | | | |
| . . . . . | | | | | | | | | | | |
| Prefix Continued | | | | | | | | | | | |
| Document Area | | | | | | | | | | | |

### Generic File Prefix

A generic WordPerfect 7 prefix is 30 bytes long. The first 16 bytes comprise the standard WordPerfect 16-byte header. The last 14 bytes comprise the index header. There is no packet data area. The following table describes the generic prefix:

| Host Def Table | JXT527 | |
|---|---|---|

| -1 | 'W' | 'P' | 'C' | Pointer to document Area = 30 | Product | File Type | MjrV | MnrV | Encrypt | Ptr to Index Area |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x02 | 0x00 | # Indexes 1 | | Reserved x 10 | | | | | | Document Area |
| Document Area Continued | | | | | | | | | | |

The pointer to the document area is 30 bytes long (16-byte prefix header + 14-byte index block). The number of indexes is 1 for the index header only.

Example of Generic Header (hex dump):

```
FF 57 50 43 1E 00 00 00 01 0A 02 00 00 00 00 00 .WPC............
02 00 01 00 00 00 00 00 00 00 00 00 00 00       ..............
```

### File Header Format

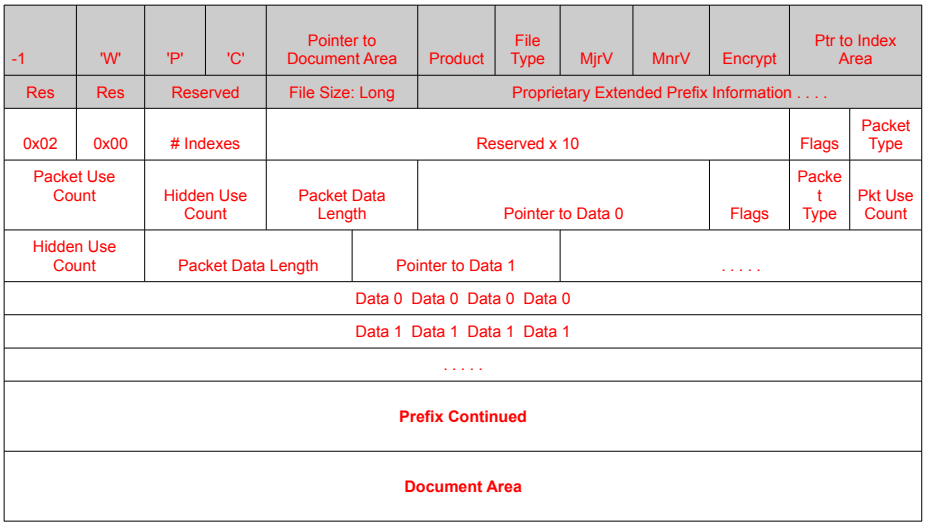
The file header specific to WordPerfect can be 16 bytes or longer. If the file header is longer than 16 bytes, it is called an *extended file header*. The 16 bytes in a normal file header and the first 16 bytes of an extended file header contain identical information. The format of the file header is:

| Format | Description |
|---|---|
| <−1 (0xFF)> <87 (0x57)> <80 (0x50)> <67 (0x43)> | −1,"WPC". Always the first four bytes of a WordPerfect document file |
| {Pointer to Document Area} | Long pointer to document area (the absolute offset from the beginning of the file) |
| <Product type> | Product type (see *Product Type Field* below) WordPerfect program = 1 |
| <File type> | File type (see *Product-Specific File Types* below) |
| <Major version> | Major version of file. WP6.0 = 2 |
| <Minor version> | Minor version of file. WP6.0 = 0, WP6.1 = 1. |
| [Encryption] | If nonzero, document is encrypted |
| [pointer to index area] | Offset of index header (a value of 0-15 implies an offset of 16) |
| <reserved> | Beginning of extended file header |
| <reserved> | |
| [reserved] | |
| {file size} | File size, not including pad characters at EOF |

**Important:** If an extended header exists and an application developer changes the file external to WordPerfect, the developer must update the {file size} long value to reflect the new file size.◄

The following table gives a graphical representation of the WP7 extended file header:

| Host Def Table | **JXT528** | |
|---|---|---|




## File ID Field

The File ID fields are the first 4 bytes of a file and have the same values for all files produced by Corel® products (excluding pre-WP5.0 files). It is displayed as −1,"WPC" or “FF 57 50 43” in hexadecimal values. If you look at a WordPerfect file that is 5.0 or later in any binary editor, you can see the ID in the first 4 bytes of the file as described. If you do not see this ID, the file is not a WordPerfect 5.0 or later file.

## Pointer to Document Area

The Pointer to Document area field is a long value that begins at offset 4 in the 16-byte file header. This is a pointer to the beginning of the document area. Short integers (two bytes) and long integers (four bytes) are saved in a byte-reversed order.

If any codes such as margins, tabs, font changes, or a specific page size are inserted at the beginning of a document, those codes appear in the document area before the actual text.

## Product Type Field

The Product Type field is 1 byte in length and is the 9th byte from the beginning of the file. It contains a value that identifies the Corel software product used to create the file. The product type for WordPerfect (the word processor) is 1.

## File Type Field

The File Type field is 1 byte in length and is the 10th byte from the beginning of the file. The value depends on the Product Type associated with the file. The first ten values (0–9) are reserved for general purpose files that have application across all Corel products. Values 10 and above are available for product-specific file types. See *Product-Specific File Types* below.

## General-Purpose File Types

These general-purpose file types are documented for information only. These file types are reserved for Corel software.

| Value | File |
|---|---|
| 1 | Macro file |
| 2 | Help file |
| 3 | Keyboard definition file |
| 4 | VAX* keyboard definition file added for WP5.1 (3-30-90) |

**Product-Specific File Types**

| Value | File |
|---|---|
| 10 (0x0A) | WordPerfect document |
| 11 (0x0B) | Dictionary file |
| 12 (0x0C) | Thesaurus file |
| 13 (0x0D) | Block |
| 14 (0x0E) | Rectangular block |
| 15 (0x0F) | Column block |
| 16 (0x10) | Printer resource file (.PRS) |
| 17 (0x11) | Setup file [contains the system values for WP{WP}.SET (Setup values, Shift-F1)] |
| 18 (0x12) | Reserved |
| 19 (0x13) | Printer resource file (.ALL) |
| 20 (0x14) | Display resource file (.DRS) |
| 21 (0x15) | Overlay file (WP.FIL) |
| 22 (0x16) | WordPerfect graphic file (.WPG) |
| 23 (0x17) | Hyphenation code module |
| 24 (0x18) | Hyphenation data module |
| 25 (0x19) | Macro resource file (.MRS) |
| 26 (0x1A) | Graphics screen-driver file (.VRS) |
| 27 (0x1B) | Hyphenation lex module |

The following file types were added to the WordPerfect file type for WP7:

| Value | File |
|---|---|
| 28 (0x1C) | Printer Q codes (used by VAX/DG) |
| 29 (0x1D) | Speller code module word list |
| 30 (0x1E) | WP.QRS file (WP5.1 equation resource file) |
| 31 (0x1F) | Reserved |
| 32 (0x20) | VAX.SET |
| 33 (0x21) | Speller code module rules |
| 34 (0x22) | Dictionary rules |
| 35 (0x23) | Reserved |
| 36 (0x24) | .WPD files |
| 37 (0x25) | WordPerfect Rhymer™ word file (TSR product) |
| 38 (0x26) | Rhymer pronunciation file |
| 39 (0x27) | Reserved |
| 40 (0x28) | Reserved |
| 41 (0x29) | WP51.INS file (install options file) |
| 42 (0x2A) | Mouse driver for WP5.1 |
| 43 (0x2B) | UNIX® setup file for WP5.0 |
| 44 (0x2C) | Macintosh* WP2.0 document |
| 45 (0x2D) | VAX file (WP4.2 document) |
| 46 (0x2E) | External speller code module (WP5.1) (This file type is set aside for third parties to create their own speller code modules in WP5.1 DOS) |
| 47 (0x2F) | External speller dictionary (This file type is set aside so third parties can create their own dictionaries .LEX files to be read by their speller code.) |
| 48 (0x30) | Macintosh SOFT graphics file (SOFT (Sequential Object Format) graphics file for the Macintosh WP.) |
| 49 (0x31) | Reserved |
| 50 (0x37) | Reserved |
| 51 (0x38) | WPWin 5.1 Application Resource Library added for WPWin 5.1 |

**Major Version and Minor Version Fields**

These version numbers are the major and minor version numbers of the file format itself, not of the program creating the file. For WP6.*x* document files the major version byte is 2. The minor version byte is 0 for WP6.0 files and 1 for

WP6.1 files.

An application that expects the 2.*x* version of this file format should not fail unless the major number (not the minor number) of the file prefix is different than expected. A major revision to the file format should be rejected. An application written for WP7 should not be expected to read incompatible file formats, such as WP5.0.

**Encryption Field**
If this word value is non-zero, the file is encrypted and nothing beyond the file header (the first 16 bytes of the file) will be intelligible to an application program.

**Pointer to Index Area**
This is the offset from the beginning of the file to the index header. If this value is less than 16 (0x10), the file header is a normal 16-byte file header and the actual offset of the index area is 16 (0x10). If the file header is extended, this value will be greater than 16 and will be the valid offset.

**Reserved**
Four bytes at the beginning of the extended file header are reserved.

**File Size**
This 32-bit integer field contains the total length of the file. This does not include any pad characters added to files by some operating systems (DOS does not use pad characters). If this field exists, it must be kept accurate. If an application changes the file external to WordPerfect, it must update this field to reflect the new file size.

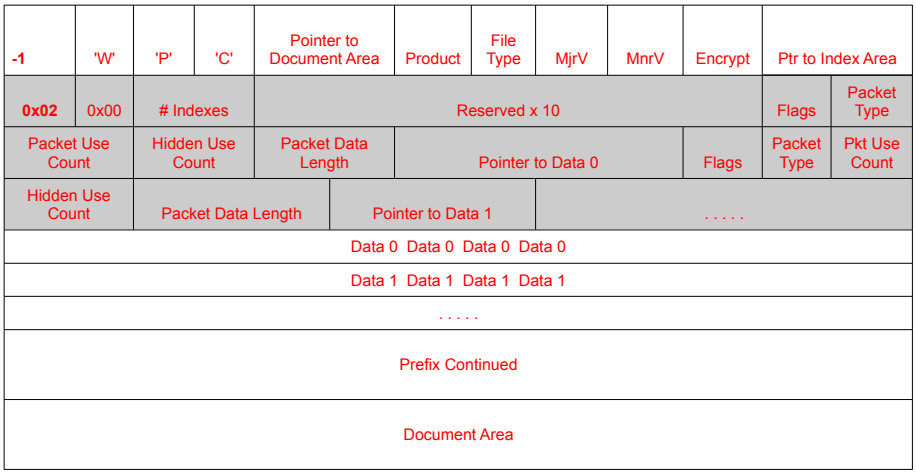
**Index and Packet Data Areas**

The prefix index area comes immediately after the file header. The index area contains indexes which point to data in the packet data area. The packet data area follows the index area in the file. Packets of structured data make up the packet data area. See *Packet Data Formats* later in this section for examples. The packet data contains information such as font descriptions that may be used many times in the document but which is not part of the actual content of the document. Packet data is referenced from other packets and from the document area though the indexes.

Unlike WP5.*x* documents, which have index blocks of 5 indexes each, WP7 documents have only one index area. The WP7 index area contains all the indexes needed for the document.

The shaded area of the following table is a representation of the index area:

| | | |
|---|---|---|
| Host Def Table | **JXT529** | |

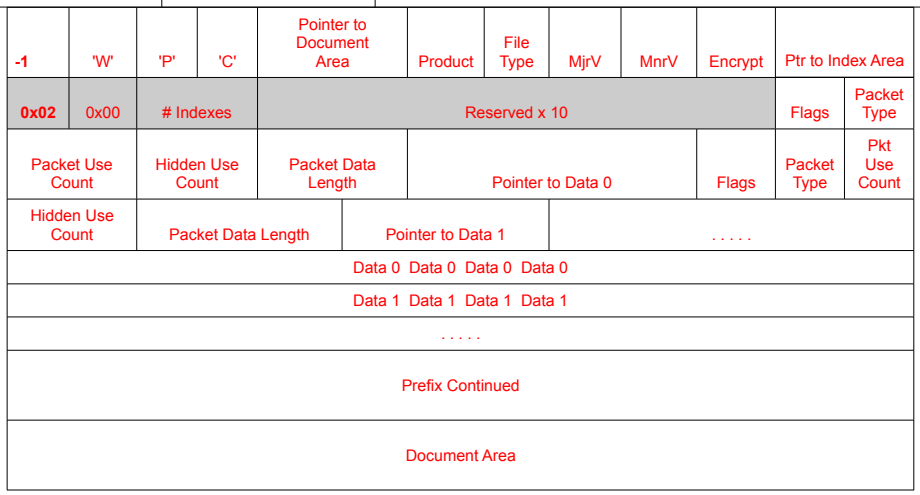



In a WP7 index area, the first index is the index header. The index header tells how many indexes are in the index area. The format of the index header is:

► <flags byte = 2>
<reserved = 0>
[number of indexes in index block]
<reserved = 0> x 10◄

The shaded area of the following table represents the index header.

| Host Def Table | **JXT530** | |
|---|---|---|




Indexes which follow the index header have the following format:

| **Format** | **Description** |
|---|---|
| <flags> | Bit 0: (1) set means the packet contains IDs for child packets (see next table). |

| Format | Description |
|---|---|
| | Bit 1: (2) set means the packet contains WordPerfect character set mappable text. |
| | Bit 2: (4) set means the maximum valid use count is 1. |
| | Bit 3: (8) set means the valid use count is 1 when no functions reference it. |
| <packet type> | Indicates the type of data that this index points to. |
| [packet use count] | The number of document functions that reference this prefix data. |
| [hidden count] | The number of deleted document functions that reference this prefix data. |
| {size of data packet} | The number of bytes of data in the packet. |
| {ptr to data packet} | Offset from beginning of file. |

If bit 0 of the index flags byte is set, it means the data packet has child IDs. If a child ID exists, the first part of the data for the specified packet has the following structure:

| Format | Description |
|---|---|
| [number of child IDs] | |
| [ID 1] | first child prefix ID reference |
| [ID 2] | second child prefix ID reference |
| ... | ... |
| [ID n] | last child prefix ID reference |

If bit 1 of the index flags byte is set, then the following data structure will be the next thing in the packet. If bit 0 (the child bit) is not set, then the following data structure is the first thing in the data packet.

►
[number of text blocks]
{relative offset of text block within packet}
{size of first text block}
{size of second text block}
...
{size of last text block}◄

If bit 0 and bit 1 of the index flags byte are not set, neither of the previous two data structures appear in the packet data.

Example:

```
00000000: FF 57 50 43 B7 1B 00 00  01 0A 83 00 00 00 00 00 #WPC╖.....â.....
00000010: 02 00 1F 00 00 00 00 00  00 00 00 00 00 00|0B 30 ......▒. .......0
00000020: 02 00 00 00 28 00 00 00  53 1A 00 00|00 09 00 00 ....(...S.......
00000030: 01 00 06 00 00 00 7B 1A  00 00|08 02 01 00 00 00 ......{. ........
00000040: 10 00 00 00 81 1A 00 00 |08 23 01 00 00 00 09 01 ....ü....#......
00000050: 00 00 91 1A 00 00|00 55  08 00 00 00 17 00 00 00 ..æ....U ........
00000060: 9A 1B 00 00|0B 30 03 00  00 00 C0 00 00 00 B0 02 Ü....0.. ..└...░.
00000070: 00 00|0B 30 05 00 00 00  78 00 00 00 70 03 00 00 ...0....x...p...
...
...
...
Prefix ID ref Text Block pointers
000002B0: 01 00 07 00|04 00 2C 00  00 00 4A 00 00 00 00 00 ......,. ..J.....
000002C0: 00 00 00 00 00 00 4A 00  00 00|03 0C 77 1E 2A 00 ......J. ....w.*.
000002D0: 72 00 75 00 73 00 73 00  00 00 00 00|D4 1A 1D 00 r.u.s.s. ....└...
000002E0: 80 01 05 00 08 00 58 02  EC 38 00 00 58 02 00 00 Ç.....X. ∞8..X...
000002F0: 58 02 05 00 50 50 1D 00  D4 D4 1B 1D 00 80 01 05 X...PP.. └└...Ç..
00000300: 00 08 00 58 02 EC 38 00  00 58 02 00 00 58 02 05 ...X.∞8. .X...X..
00000310: 00 58 02 1D 00 D4 D4 18  10 00 00 03 00 00 00 00 .X...└└. ........
00000320: 00 00 00 10 00 D4|D4 1A  1D 00 80 01 05 00 08 00 .....└└. ..Ç.....
00000330: 58 02 EC 38 00 00 58 02  00 00 58 02 05 00 50 50 X.∞8..X. ..X...PP
00000340: 1D 00 D4 D4 1B 1D 00 80  01 05 00 08 00 58 02 EC ..└└...Ç .....X.∞
00000350: 38 00 00 58 02 00 00 58  02 05 00 58 02 1D 00 D4 8..X...X ...X...└
00000360: D4 18 10 00 00 03 00 00  00 00 00 00 00 10 00 D4 └.............└
```

The first bolded block of data is an index to the data type 48 (0x30). The flags byte is 11 (0x0B) or 1011 binary, so bit 0 and bit 1 are set. Bit 0 specifies that there are child references; bit 1 specifies that text blocks exist.

The second bolded block is the packet data pointed to by the bolded index. The first word or the count of child index references specifies that there is only one child index referenced. The second word is the index ID for the child index. The third word is the number of text blocks in this packet of data. Following the third word are five long values. The first long value is a pointer to where the text data begins. The rest of the long values are sizes of the respective text blocks. See the format of *Packet Type 48 (0x30)*.

When a reference is made to a prefix packet ID, remember that a prefix ID is not the same as a packet type. *Prefix ID* refers to the index sequence of the packet's index in the index block. *Packet type* refers to the purpose and structure of the packet's data. Packet IDs are unique to each packet in the prefix area. There can be several packets with the same type value.

## Document Area

The document area contains the actual text of the document along with all of the formatting function codes required to create and format the desired document. These function codes may be a single byte or may be many bytes in length. The multi-byte functions may be fixed or variable in length depending on the particular function. These functions may reference data in the packet data area by specifying a prefix index which in turn points to the packet data. This prefix index reference is called a prefix ID or PID.

### Single-Byte Functions

Single-byte functions range from 128 (0x80) through 207 (0xCF). In the following example a soft Hyphen at End-of-Line function byte is inserted in the word “comment.”

Example: com**<131 (0x83)>**ment

See *Single-Byte Functions* later in this manual.

**Variable-Length Multi-Byte Functions**

The codes for variable-length multi-byte functions 208 (0xD0) through 239 (0xEF) appear twice each time the function is invoked. The first occurrence is the begin gate (beginning of the function code) and a second occurrence is the end gate (end of the function code).

The orientation of a function specifies what this function applies to. For example, if this function is specific to the page format, the orientation is page type; if it is specific to the line format, the orientation is line type.

Each begin gate is followed by a subgroup byte, a value of size short (16 bits), and a function flag byte. An example of font code structure follows:

| Format | Description |
|---|---|
| <212 (0xD4)> | Begin gate group byte |
| <26 (0x1A)> | Subgroup byte |
| [31 (0x001F)] | Size short |
| <128 (0x80)> | Flags byte (high bit set means prefix IDs follow) |
| <1 (0x01)> | Number of prefix IDs which follow = 1 |
| [10 (0x000A)] | Prefix ID for desired font. Index ten points to the desired font information |
| ... | |
| ... | |
| [31 (0x001F)] | Size short |
| <212 (0xD4)> | End gate group byte |

Any of the variable-length function codes may reference one or more prefix IDs. For example, the font change function code (see *Font Face Change* under *Variable-Length Multi-Byte Functions* later in this section) references the prefix ID that contains the desired font data. Variable-length functions that do not currently reference a prefix ID may reference one or more prefix IDs in the future.

**Important:** Document parsing programs must allow for prefix ID references in every variable-length function code.◄

When the flags byte has the high bit set, there is prefix data associated with the function. The byte following the flags byte contains the number of prefix IDs that are referenced in the function.

Following the byte that gives the number of prefix IDs is a short value for each individual prefix ID. Refer to *Variable-Length Multi-Byte Functions* later in this section for more information.

Next is a short value showing the size of the non-deletable data. Any non-deletable function data comes after the size value. The data in variable-length multi-byte functions is divided into two main parts: a non-deletable portion and a deletable portion. The non-deletable portion of a function code is the documented part of the function and should not change from one interim release of WP7 to another. If data is added to the non-deletable part of a function code, it is added to the end of the non-deletable data.

Deletable data directly follows the non-deletable data. The size of the deletable data can be variable for each function. Deletable data is not documented, since it is specific to the formatter of WordPerfect 7. It can be platform specific, language

specific, version specific, and so forth. It is subject to change at any time. Deletable data may or may not be present in a variable-length function code within files created in WordPerfect. Document files created outside of WordPerfect should contain only non-deletable data.

Pertinent information for application developers is documented in the non-deletable portion of the function code. To skip over the deletable data, use the size field to move from the beginning of the current function to the next existing function.

Each end gate is preceded by a size value (short), which should always be the same value as the size encountered at the beginning of the function. The size of the function is the total size of the function including begin and end gates.

| **Format** | **Example** |
|---|---|
| <group> | <209 (0xD1)> |
| <subgroup> | <2(0x02)> |
| [size] | [14 (0x000E)] |
| <flags> | <0 (0x00)> |
| <number of prefix IDs> | (absent since bit 7 of flags = 0) |
| [prefix IDs] x ? | (absent) |
| [size of non-deletable information] | [2 (0x02)] |
| <non-deletable information> x ? | [18 (0x0012)] |
| <undocumented deletable data> x ? | [2 (0x0002)] |
| [size] | [14 (0x000E)] |
| <group> | <209 (0xD1)> |

The above function appears in this documentation as follows.

```
►       <group> <subgroup> [size] <flags>
                        <number of prefix IDs>
                        [prefix IDs...]
                [size of non-deletable information]
                <non-deletable information> x ?
                <undocumented deletable data> x ?
        [size] <group>◄
```

For other examples, see *Variable-Length Multi-Byte Functions* later in this manual.

When creating WP7 documents external to WP7, create the ending size (short

value) and the group byte directly after the non-deletable information. No deletable information should exist in documents created external to WP7.

**Function Flags Byte**
Variable-length multi-byte functions always contain a flags byte to indicate the presence of prefix IDs and to indicate how the function is used. See *Variable-Length Multi-Byte Functions* later in this manual for a full description of the flags byte.

**Fixed-Length Multi-Byte Functions**
The codes for fixed-length multi-byte functions 240 (0xF0) through 255 (0xFF) always appear twice. The first occurrence is the begin gate, and a second occurrence is the end gate. The length of each function is fixed and listed after the function code.

Example: can**<240 (0xF0)><28 (0x1C)><4 (0x04)><240 (0xF0)>**t

| **Format** | **Meaning** |
|---|---|
| <240 (0xF0)> | Function code for WordPerfect character |
| <28 (0x1C)> | Character number |
| <4 (0x04)> | Character set |
| <240 (0xF0)> | Function code end gate for WordPerfect character |

Document formatting is accomplished by embedding function codes in the text of a document. A function is any byte greater than 127 (0x7F).

## ♦ Character Map File Format

The character map file begins with the standard WP7 file header. The remainder of the file is unique to the character map as defined below.

| **Size and Format** | **Meaning** |
|---|---|
| <0xFF> <0x57> <0x50> <0x43> | −1,"WPC". First four bytes of a WordPerfect document file |
| {pointer to data} | Long pointer to data area (absolute offset from the beginning of the file) |
| <product type> | WordPerfect program = 1 |
| <file type> | File type. WP7 Character Map file = 49 |
| <major version> | Major version of file WP7 = 2 |

| Size and Format | Meaning |
|---|---|
| <minor version> | Minor version of file WP6.0 = 0, WP6.1 = 1. |
| [encryption] | If non-zero, then document is encrypted |
| [reserved] | Change label when used |
| [size of map information] | Begin of map information |
| <flag byte> | Bit 7 is set (0x80) for all 2-byte mappings. 2-byte conversions are done in code. There is no map name or map table saved in the map file. |
| | 0x80 = Japanese mapping |
| | 0x81 = Korean mapping |
| | 0x82 = Chinese mapping |
| | 0x83 = Taiwanese mapping |
| [pointer to map name] | Offset from start of map information |
| [pointer to convert table] | Table for converting document characters to WordPerfect characters |
| [map name] x ? | Null-terminated WordPerfect word string of indeterminate length |
| [minimum value] | Minimum value of mappable characters (greater than 0) |
| [maximum value] | Maximum value of mappable characters (less than or equal 127) |
| [number of mappable chars] | Maximum − minimum + 1 |
| [size of table] | Total size of the map table |
| [char #, char set] | Document characters are mapped into the range 1-127. The 1-byte document character to WordPerfect character mapping table is essentially a lookup table. It consists of 127 entries, each one word in length. The first byte of the first word specifies the document character number and the second byte specifies the WP character set or lookup table in which the character is contained. The first character in the lookup table is mapped to document character number 1. The second character in the lookup table is mapped to character number two and so on. |

If the {Pointer to Data} field is greater than 16 bytes, then the header is an extended file header. See *File Header Format* earlier in this section.