.WPM Macro File Formats for DOS, VAX, and UNIX WordPerfect

All files created by WPCorp products begin with a 16-byte file prefix that identifies the file type and the product and product version that created it.

The macro file prefix has the following structure:

Field	Size	Value for WP5.1 Document
WPCorp File ID	4 bytes	þ1 (0xFF),“WPC”
Start of Document	long	File position of start of macro
Product Type	1 byte	1
File Type	1 byte	1
Major Version #	1 byte	1 for WP5.1 (0 for WP5.0 .WPM files)
Minor Version #	1 byte	1 for WP5.1 (0 for WP5.0 .WPM files)
Encryption Key	short	0 = not encrypted
Reserved	short	0

The WPCorp file ID field is the same for all files produced by WPCorp products.

The start of the macro field is a long integer offset from the beginning of the file. The macro ends with two null terminators.

The product type field can be one of the following:

Value	Product
1	WordPerfect
2	Shell
3	Notebook
4	Calculator
5	File Manager
6	Calendar
7	Program Editor
8	Macro Editor
9	PlanPerfect
10 (0xA)	DataPerfect
11 (0xB)	Mail
12 (0xC)	Printer (PTR.EXE)
13 (0xD)	Scheduler
14 (0xE)	WordPerfect Office
15 (0xF)	DrawPerfect
16 (0x10)	LetterPerfect

The file type for macros is “1” (refer to Document File Formats for a list of other file types).

The major and minor version values may change depending on different product versions. If the product uses WP5.0 type macro commands, major and minor version values are "0." If the product uses enhanced WP5.1 type macro commands, major and minor version values are "1."

The Encryption key for macros is zero (0). Macro files cannot be encrypted within WordPerfect 5.0 or 5.1. If you want the code for your product to be confidential, produce a program that uses a third-party interface instead of WordPerfect macros.

For current WordPerfect products, the reserved field is zero (0).

The following is information about the macro prefix structure that comes after the 16-byte file prefix.

The *special header index* is 10 bytes long and has the following structure:

Offset	Size	Usage
0	short	Packet type for header index (65531 [0xFFFB])
2	short	# of indexes (including header); .WPM macros contain four indexes (only the first and second are currently used)
4	short	Size of index block (# of indexes * 10)
6	long	File position of next index block from beginning of prefix (0, since a macro file needs only one index)

Other indexes in the index block are also 10 bytes long and have the following structure (only the first index following the special header index is currently used in .WPM macros):

Offset	Size	Usage
0	short	Packet type (1 for macros)
2	long	Length of macro description (1 if no macro description exists, 0 if the macro was created originally in ME50.EXE [or ED.EXE] rather than in WP)
6	long	File position for start of description

Note: *If the macro is created in ME50.EXE (or ED.EXE), the "pointer to description" and "pointer start of macro" are the same if no macro description is entered in ME50.EXE (or ED.EXE).*

Structure of Macro Commands (how codes and characters are represented in macro files)

Normal Characters

High Byte	Low Byte	Meaning
0	32–126 (0x20–0x7E)	ASCII characters

Extended Characters

High Byte	Low Byte	Meaning
Char set	Char #	"Char set" represents the character set # (0 – 12 [0xC]). The "char #" represents the character # within a particular character set.

Note: *For more information on WordPerfect characters and WordPerfect character sets, see* Appendix P *in* WordPerfect Reference.

Alt Characters

High Byte	Low Byte	Meaning
255 (0xFF)	0 þ 9	0 – 9
128 (0x80)	97 (0x61)	
128 (0x80)	107 (0x6B)	=
254 (0xFE)	65 - 90 (0x41 þ 0x5A)	A – Z

Control Characters

High Byte	Low Byte	Meaning
128 (0x80)	0	^@ — Compose (cannot be entered by third-party interface)
128 (0x80)	1	^A
128 (0x80)	2	^B — Page #
128 (0x80)	3	^C — Merge from console
128 (0x80)	4	^D — Merge date
128 (0x80)	5	^E — Merge end of record
128 (0x80)	6	^F — Merge field
128 (0x80)	7	^G — Merge macro
128 (0x80)	8	^H — Home
128 (0x80)	9	^I — Tab
128 (0x80)	10 (0xA)	^J — Enter
128 (0x80)	11 (0xB)	^K — Delete to end of line
128 (0x80)	12 (0xC)	^L — Delete to end of page
128 (0x80)	13 (0xD)	^M — Search value for [SRt]
128 (0x80)	14 (0xE)	^N — Merge next record
128 (0x80)	15 (0xF)	^O — Merge output prompt
128 (0x80)	16 (0x10)	^P — Merge primary filename
128 (0x80)	17 (0x11)	^Q — Merge quit
128 (0x80)	18 (0x12)	^R — Merge return
128 (0x80)	19 (0x13)	^S — Merge secondary filename
128 (0x80)	20 (0x14)	^T — Merge text to printer
128 (0x80)	21 (0x15)	^U — Merge update the screen
128 (0x80)	22 (0x16)	^V — Ignore meaning of following code
128 (0x80)	23 (0x17)	^W — Up
128 (0x80)	24 (0x18)	^X — Right and search wildcard—same as typical ? wildcard
128 (0x80)	25 (0x19)	^Y — Left
128 (0x80)	26 (0x1A)	^Z — Down
128 (0x80)	27 (0x1B)	^[— Escape
128 (0x80)	28 (0x1C)	^\
128 (0x80)	29 (0x1D)	^]
128 (0x80)	30 (0x1E)	^^ — Reset keyboard map (cannot be entered by macro or DOS third-party interface)
128 (0x80)	31 (0x1F)	^_

Note: *These control characters apply to WP5.0 and also work in WP5.1. However, WP5.1 has an enhanced set of merge commands (see* Appendix K *in* WordPerfect Reference*).*

Function Keys

High Byte	Low Byte	Meaning
128 (0x80)	32 (0x20)	Cancel
128 (0x80)	33 (0x21)	Forward Search
128 (0x80)	34 (0x22)	Help
128 (0x80)	35 (0x23)	Indent
128 (0x80)	36 (0x24)	List Files
128 (0x80)	37 (0x25)	Bold
128 (0x80)	38 (0x26)	Exit
128 (0x80)	39 (0x27)	Underline
128 (0x80)	40 (0x28)	Merge Return
128 (0x80)	41 (0x29)	Save
128 (0x80)	42 (0x2A)	F11
128 (0x80)	43 (0x2B)	F12

Shift + Function Keys

High Byte	Low Byte	Meaning
128 (0x80)	44 (0x2C)	Setup
128 (0x80)	45 (0x2D)	Backward Search
128 (0x80)	46 (0x2E)	Switch
128 (0x80)	47 (0x2F)	Left/Right Indent
128 (0x80)	48 (0x30)	Date/Outline
128 (0x80)	49 (0x31)	Center
128 (0x80)	50 (0x32)	Print
128 (0x80)	51 (0x33)	Format
128 (0x80)	52 (0x34)	Merge Commands
128 (0x80)	53 (0x35)	Retrieve
128 (0x80)	54 (0x36)	Shift-F11
128 (0x80)	55 (0x37)	Shift-F12

Alt + Function Keys

High Byte	Low Byte	Meaning
128 (0x80)	56 (0x38)	Thesaurus
128 (0x80)	57 (0x39)	Search/Replace
128 (0x80)	58 (0x3A)	Reveal Codes
128 (0x80)	59 (0x3B)	Block
128 (0x80)	60 (0x3C)	Mark Text
128 (0x80)	61 (0x3D)	Flush Right
128 (0x80)	62 (0x3E)	Columns/Table
128 (0x80)	63 (0x3F)	Style
128 (0x80)	64 (0x40)	Graphics
128 (0x80)	65 (0x41)	Macro
128 (0x80)	66 (0x42)	Alt-F11
128 (0x80)	67 (0x43)	Alt-F12

Control + Function Keys

High Byte	Low Byte	Meaning
128 (0x80)	68 (0x44)	Shell
128 (0x80)	69 (0x45)	Spell
128 (0x80)	70 (0x46)	Screen
128 (0x80)	71 (0x47)	Move
128 (0x80)	72 (0x48)	Text In/Out
128 (0x80)	73 (0x49)	Tab Align
128 (0x80)	74 (0x4A)	Footnote
128 (0x80)	75 (0x4B)	Font
128 (0x80)	76 (0x4C)	Merge/Sort
128 (0x80)	77 (0x4D)	Define Macro
128 (0x80)	78 (0x4E)	Ctrl-F11
128 (0x80)	79 (0x4F)	Ctrl-F12

WordPerfect Key Commands

High Byte	Low Byte	Meaning
128 (0x80)	80 (0x50)	Backspace
128 (0x80)	81 (0x51)	Delete Right
128 (0x80)	82 (0x52)	Delete Word
128 (0x80)	83 (0x53)	Word Right
128 (0x80)	84 (0x54)	Word Left
128 (0x80)	85 (0x55)	Home, Home, Right (End key)
128 (0x80)	86 (0x56)	Home, Home, Left
128 (0x80)	87 (0x57)	Reserved
128 (0x80)	88 (0x58)	Go To
128 (0x80)	89 (0x59)	PgUp
128 (0x80)	90 (0x5A)	PgDn
128 (0x80)	91 (0x5B)	Screen Down (“+” on numeric keypad)
128 (0x80)	92 (0x5C)	Screen Up (“þ” on numeric keypad)
128 (0x80)	93 (0x5D)	Typeover
128 (0x80)	94 (0x5E)	Left Margin Release (Shift-Tab)
128 (0x80)	95 (0x5F)	Hard Page (Ctrl-Enter)
128 (0x80)	96 (0x60)	Soft Hyphen
128 (0x80)	97 (0x61)	Hard Hyphen
128 (0x80)	98 (0x62)	Hard Space

High Byte	Low Byte	Meaning
252 (0xFC)	58 (0x3A)	{~} (hard tilde) *added for WP5.1*
252 (0xFC)	11 (0xB)	{;}comment~ (comment — do not execute)
252 (0xFC)	1	{ASSIGN}var~value~ (assign value to variable)
252 (0xFC)	2	{BELL} (sound bell)
128 (0x80)	108 (0x6C)	{Block Append} *only in WP5.1*
128 (0x80)	110 (0x6E)	{Block Copy} *only in WP5.1*
128 (0x80)	109 (0x6D)	{Block Move} *only in WP5.1*
252 (0xFC)	3	{BREAK} (leave current if, for, or while)
252 (0xFC)	4	{CALL}label~ (call subroutine in same file)
252 (0xFC)	5	{CANCEL OFF} (ignore cancel)
252 (0xFC)	6	{CANCEL ON} (allow cancel)
252 (0xFC)	7	{CASE}val~case1~label1~c2~l2~...~~ (goto label val=case)
252 (0xFC)	8	{CASE CALL}val~c1~l1~c2~l2~...~~ (call label val=case)
252 (0xFC)	9	{CHAIN}file~ (execute after current macro)
252 (0xFC)	10 (0xA)	{CHAR}var~message~ (input single character into var)
252 (0xFC)	12 (0xC)	{DISPLAY OFF} (turn off display)
252 (0xFC)	13 (0xD)	{DISPLAY ON} (turn on display)
252 (0xFC)	14 (0xE)	{ELSE} (else portion of IF)
252 (0xFC)	15 (0xF)	{END FOR} (end of FOR/FOR EACH loop) *added for WP5.1*
252 (0xFC)	16 (0x10)	{END IF} (end of IF command)
252 (0xFC)	17 (0x11)	{END WHILE} (end of WHILE loop) *added for WP5.1*
252 (0xFC)	18 (0x12)	{FOR}var~start~stop~step~ (for loop END FOR) *added for WP5.1*
252 (0xFC)	19 (0x13)	{FOR EACH}var~value~value~...~~ *added for WP5.1* (loop with var assigned to each value)
252 (0xFC)	20 (0x14)	{GO}label~ (jump to the label)
252 (0xFC)	21 (0x15)	{IF}value~ (true if value !0 ELSE 0 END IF)
252 (0xFC)	47 (0x2F)	{IF EXISTS}var~ (true if a value is assigned)
252 (0xFC)	51 (0x33)	{INPUT}message~ (Prompt, input until Merge R) *added for WP5.1*
128 (0x80)	104 (0x68)	{Item Down} *only for WP5.1*
128 (0x80)	101 (0x65)	{Item Left} *only for WP5.1*
128 (0x80)	102 (0x66)	{Item Right} *only for WP5.1*
128 (0x80)	103 (0x67)	{Item Up} *only for WP5.1*
252 (0xFC)	56 (0x38)	{KTON}key~ (convert key to #) *added for WP5.1*
252 (0xFC)	57 (0x39)	{LEN}var~ (get length of variable) *added for WP5.1*
252 (0xFC)	22 (0x16)	{LABEL}label~ (label for CALL, GO, etc.)

High Byte	Low Byte	Meaning
252 (0xFC)	23 (0x17)	{LOOK}var~ (tstkb type of {CHAR}) *added for WP5.1*
252 (0xFC)	44 (0x2C)	{Macro Commands} (application program access to macro commands)
252 (0xFC)	48 (0x30)	{MENU OFF} (menu display off) *added for WP5.1* (only for WP5.1)
252 (0xFC)	49 (0x31)	{MENU ON} (menu display on) *added for WP5.1* (only in WP5.1)
252 (0xFC)	54 (0x36)	{MID}var~offset~count~ (get substring) *added for WP5.1*
252 (0xFC)	24 (0x18)	{NEST}file~ (execute now)
252 (0xFC)	25 (0x19)	{NEXT} (loop in while, for, for each) *added for WP5.1*
252 (0xFC)	55 (0x37)	{NTOK}value~ (convert # to key) *added for WP5.1*
252 (0xFC)	27 (0x1B)	{ON CANCEL}action~ ~(continue), {QUIT}~, {RETURN}~, {GO}label~, {CALL}label~
252 (0xFC)	28 (0x1C)	{ON ERROR}action~ ~(continue), {QUIT}~, {RETURN}~, {GO}label~, {CALL}label~
252 (0xFC)	29 (0x1D)	{ON NOT FOUND}action~ ~(continue), {QUIT}~, {RETURN}~, {GO}label~, {CALL}label~
252 (0xFC)	46 (0x2E)	{ORIGINAL KEY} (original keyboard contents of last key entered)
128 (0x80)	100 (0x64)	{Para Down} *only for WP5.1*
128 (0x80)	99 (0x63)	{Para Up} *only for WP5.1*
252 (0xFC)	30 (0x1E)	{PAUSE} (unprompted user input)
252 (0xFC)	59 (0x3B)	{PAUSE KEY}key~ *only for WP5.1*
252 (0xFC)	31 (0x1F)	{PROMPT}message~ (prompt on bottom line)
252 (0xFC)	32 (0x20)	{QUIT} (stop all macro execution)
252 (0xFC)	33 (0x21)	{RESTART} (quit after current level)
252 (0xFC)	34 (0x22)	{RETURN} (leave current execution level)
252 (0xFC)	35 (0x23)	{RETURN CANCEL} (leave current gen CANCEL cond)
252 (0xFC)	36 (0x24)	{RETURN ERROR} (leave current gen ERROR cond)
252 (0xFC)	37 (0x25)	{RETURN NOT FOUND} (leave current gen NOT FND cond)
252 (0xFC)	26 (0x1A)	{SHELL MACRO}name~ (execute a shell macro) *added for WP5.1*
252 (0xFC)	38 (0x26)	{SPEED}100ths~ (time to wait between each char)
252 (0xFC)	41 (0x29)	{STATE} (get value of the state variable)
252 (0xFC)	50 (0x32)	{STATUS PROMPT}message~ (message for status line) *added for WP5.1*
252 (0xFC)	45 (0x2D)	{STEP OFF} (turn off single step)
252 (0xFC)	39 (0x27)	{STEP ON} (turn on single step)
252 (0xFC)	53 (0x35)	†{SYSTEM}systemvar~ (reference system data) *added for WP5.1*

High Byte	Low Byte	Meaning
252 (0xFC)	40 (0x28)	{TEXT}var~message~ (assign string to the var)
252 (0xFC)	52 (0x34)	{VARIABLE}var~ (reference variable contents) *added for WP5.1*
252 (0xFC)	42 (0x2A)	{WAIT}10ths~ (time for single delay)
252 (0xFC)	43 (0x2B)	{WHILE}value~ (while !0 loop END WHILE) *added for WP5.1*

† If the cursor rests on a function code, the macro command and {SYSTEM}right~ subcommand return the first two bytes of that function code.

Below is a hex dump of a .WPM macro that does the following:

{DISPLAY OFF}{Search}{Text In/Out}c{Search}{ON NOT FOUND}{TEXT}
No·More·Comments~{PAUSE}~

The description of this macro is “Search for Comment.”

```
FF 57 50 43 4B 00 00 00 01 01 01 01 00 00 00 00   ?WPCK@@@AAAA@@@@
FB FF 04 00 28 00 00 00 00 00 01 00 13 00 00 00   þ?D@(@@@@@A@S@@@
38 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   8@@@@@@@@@@@@@@@
00 00 00 00 00 00 00 00 53 65 61 72 63 68 20 66   @@@@@@@@Search f
6F 72 20 43 6F 6D 6D 65 6E 74 00 0C FC 21 80 48   or Comment@Lþ!ÇH
80 63 00 21 80 1D FC 28 FC 4E 00 6F 00 20 00 4D   Çc@!Ç]þ(þN@o@ @M
00 6F 00 72 00 65 00 20 00 0D 00 43 00 6F 00 6D   @o@r@e@ @M@C@o@m
00 6D 00 65 00 6E 00 74 00 73 00 7E 00 1E FC 7E   @m@e@n@t@s@~@^þ~
00 00 00                                          @@@
```

Note: *Normally, 10 (0xA) (hard return), 13 (0xD) (soft return), 9 (tab), and 2 (a WP flag to check if a macro is being edited) do not appear on the screen. However, they do appear when you parse.*

Detection of WordPerfect 4.2 Files by WP5.0 and WP5.1

Many developers place WordPerfect 4.2 format and merge codes into ASCII text files so that WordPerfect 5.x can read and convert the files on the fly.

Once the file has gone through this conversion process, it has the format and codes of any WP5.x file (including the prefix header). A secondary (merge) file is the one exception to this. In WordPerfect 5.x, the conversion process does not create a secondary file that is compatible with named fields unless the WP4.2 secondary file was created with named fields in mind (see *Format of WordPerfect 4.2 Named Secondary Files* below).

A discussion of problems that can occur during the detection process used on these modified (to WP4.2) ASCII text files follows. WordPerfect 5.0 and 5.1 handle this process differently.

Note: *Because of the way ASCII files are made on the VAX, there is very little similarity between them and the WP4.2 format. The following information is unnecessary if you are working on a VAX VMS system.*

WordPerfect 5.x documents have a prefix area (see the *WordPerfect Prefix Area* subsection) just above the body of the document. This prefix explicitly tells WordPerfect 5.x which of the two versions (WP5.0 or WP5.1) created the document.

WordPerfect 4.2 files have *no prefix* information, so there is no direct way for WordPerfect 5.x to determine if the file to be retrieved is an ASCII text file or a WP4.2 file.

If you want to use the method mentioned above to create documents, secondary files, etc., and to have compatibility between both WP5.0 and WP5.1, consider the following information:

þ WordPerfect 5.0 scans through the document file to look for specific WP4.2 codes or combinations of codes that are unique to WP4.2. If it finds any, it considers the file a WP4.2 document. Otherwise, it is considered an ASCII text file.

Note: *Because of the end-of-line marker used by UNIX, any 5.x version or greater of the WordPerfect program (WP5.0, 5.1, 5.2+ or 7C) will use this method of detection. The rest of the information in this list is unnecessary if you are working on a UNIX system.*

> *However, in the earlier version of WP5.1 for UNIX, a 4.2 merge file would come into a merge with no problems, but if you tried to view it as a document, WP would convert it as if it were an ASCII file. To avoid this, insert a WP4.2 Bold On code followed by a WP4.2 Bold Off code between the Ctrl R code and the 10 (0xA) End of Line code on the first line of the file.*

þ WordPerfect 5.1 moves to the end-of-line marker of the first line in the file and checks to see what is there. DOS-ASCII text files have a carriage return code (13 [0xD]) and a linefeed code (10 [0xA]) at the end of each line. If WP5.1 finds this combination at the end of the first line in the document, it assumes this is an ASCII text file no matter what other codes it finds. If it

only finds one of the two possible codes at the end of the line, it imports the file as a WP4.2 file.

þ To ensure total compatibility with both WP5.0 and WP5.1, you must have the correct end-of-line marker (usually the 10 [0xA], hard return code), as well as the correct WP4.2 format or merge codes.

Note: *WP5.0 versions before 9/23/88 did not automatically accept WP4.2 files during a merge.*

Format of WordPerfect 4.2 Named Secondary Files
A "named" secondary file is a secondary file that has its fields in the primary file identified as field names instead of field numbers.

WordPerfect 4.2 secondary files had the capacity for named fields because of their use in the earlier versions of the WordPerfect Library Notebook program. To take advantage of this capacity to make named secondary merge files out of ASCII text files, convert the ASCII files to the Wordperfect 4.2 format. The format of a WP4.2 secondary file is much simpler and requires fewer codes than a comparable WP5.1 file.

The only difference between a standard WP4.2 secondary file and a named WP4.2 secondary file is the name definitions at the top of the file.

Named fields are defined with the following format:

Screen View	**Hex Codes Used**
^N{HRt}	14 (0xE),10 (0xA)
{Bold}Name{bold}{HRt}	157 (0x9D),Name,156 (0x9C),10 (0xA)
{Bold}Address{bold}{HRt}	157 (0x9D),Address,156 (0x9C),10 (0xA)
^R{HRt}	18 (0x12),10 (0xA)
^E{Hpg}	5,12 (0xC)

The example above defines two fields. The first field is named "Name" and the second field is named "Address."

The hard returns at the end of each line are uaws for readability rather than for the name definitions. The actual data fields would begin after the hard page code. As with any other secondary merge file, each field ends with a ^R and each record ends with a ^E.

When the secondary merge file is imported into WP5.x as part of a merge, it will be converted to the 5.x format and used with the standard, named fields in the 5.x primary file.

Importing Lotus Files Into WordPerfect

A Lotus 1-2-3 worksheet file that WordPerfect 5.1 can accept must include the following two records, and each record must contain the correct information.

Record Type 0
Version Information
Currently, WordPerfect accepts only three versions of Lotus worksheets. They are listed below with their corresponding version field values. The field values are not shown in reverse memory order.

Version	Field Value
1, 1A	1028 (0x404)
2.0	1030 (0x406)
3.0	4096 (0x1000)

Record Type 6
Dimension Information
For WordPerfect to read the worksheet, this record must have all of the following information (record type and record length fields are not shown).

Offset	Size	Meaning
0	short	Beginning column
2	short	Beginning row
4	short	End column
6	short	End row

WordPerfect Font Procedure

This procedure has two parts: first, finding the base font; and second, using the AFC Chain procedure to find the actual font used at any point in the base font's range.

Finding the Base Font
The format described in the *Toolkit* for Packet Type 15 (0xF) is the format for the first font used in a file, not the complete packet. This format is repeated each time a font is added to a document. For each font in a document, Packet Type 15 (0xF) has a corresponding 86byte block of information on that font. This is header prefix information. For simplicity, we will want to start this chain of events in the actual body of the document.

When a Font Change code (Function 209 [0xD1], subfunction 1) is found in a document, the number in the Matched Font number field is used as a pointer into Packet Type 15 (0xF). If no Font Change code is present, or to find the initial font before the first Font Change code, assume a matched font number of zero (0). To use this Matched Font number, multiply the number by 86 (to represent the 86byte blocks in Packet Type 15 [0xF]). The result is the actual offset position of the first field in this font block of information. This is the connection between the Font Change function code in the body of the document and the font information in Packet Type 15 (0xF).

In the Font Change code, get the Matched Font Hash Value number. The number should match the Hash of Descriptor field in Packet Type 15 (0xF). If they do not match, this document was saved with the Fast Save option turned on. In this case, send an error message to the user. The information in a document file saved under Fast Save does not permit the safe determination of font information or let you be certain of valid answers concerning the font.

Note: *On the VAX VMS version of WP5.1, this check of hash values is unnecessary. On VMS, with "Fast Save" set to "Yes," WordPerfect will totally delete packets 15 (0xF), 7, and any PS (Proportional Spacing) Table packets that may be in the prefix area at the time of the save. The only codes left intact will be the Font Change codes in the document area.*

To find the name of the font, go to the Pointer to Font String Pool of Font Name field in the corresponding block of Packet Type 15 (0xF). The value in this field is an exact offset into Packet Type 7. This points directly to the beginning of the string that represents the name of the font.

The functions necessary to find the base font are now complete. However, the base font may not be the actual font used to print the document. To determine the actual font, consider the second part of the font procedure, the AFC chain sequence.

AFC Chaining
This procedure modifies the base font for changes in orientation, character set, and attributes, *in that order.* For example, suppose you have a base font of Courier 10pt, but to bold you have to change fonts to Courier 10pt Bold. After you finish bolding, move back to the original 10pt font.

This constitutes AFC chaining.

AFC Chain for Orientation
If, in the body of the document, you find the Form code (Func. 208 [0xD0] Subfunc. 11 [0xB]), look at its New Effective Orientation field and find out if the value there is a 0 (zero), 1, 2, or 3 (meaning Portrait, Landscape, Reverse Portrait, or Reverse Landscape). With that information, look at the Orientation AFC List field of the 86-byte base font block in Packet Type 15 (0xF). Each byte in this "table" represents one entry in the list, and the list is ordered from 0–3. Move to the position in the table marked by the value found from the Form code. Inside this entry you will find a byte value that, when multiplied by 86, points to a new 86-byte block of font information in Packet Type 15 (0xF). This new block represents the information for the font used for this new orientation. If no new font is needed to change to a particular orientation, the value in the table entry is þ1 (0xFF). You may find a "special case" third value in this table entry. The value is þ2 (0xFE). When it is found in a table entry, it means that there is a font that could be chained to for a particular orientation, but it is currently unavailable.

AFC Chain for Character set
This procedure is basically the same as the AFC chain for orientation, except there are more than four entries in the table.

If you encounter an Extended Character code (192 [0xC0]), look for the value (0–12 [0xC]) in the Character Set field in that code.

Move to the current base font block in Packet Type 15 (0xF). Find the Char AFC List field of that block. As before, each entry in the table is one byte long. However, there is a maximum possibility of 16 entries in the list. Only the first 13 are now used. Move to the position in the table indicated by the value found in the Extended Character code. The value found at this table position is (when multiplied by 86) the pointer to the new 86-byte block in Packet Type 15 (0xF). This is the new font used to print the character. After printing the character, it returns to the current base font. As before, if no new font exists, the entry value in the table is þ1 (0xFF), and if there is a font to chain to, but it is currently unavailable, the value of þ2 (0xFE) will be in the entry.

AFC Chain for Attributes
This procedure is similar to those described above. Find an Attributes On code in the document. The number in the Attribute Type field (0–15 [0þ0xF]) is the pointer into the table.

The table is the AFC List field, found in the current base font block in Packet Type 15 (0xF). Like the other two tables, the value found in the correct table entry, when multiplied by 86, points to a new font block in Packet Type 15 (0xF). If no new font block exists, the value is þ1 (0xFF), and if the font exists, but is not available now, the value will be þ2 (0xFE).

AFC Chain Order
As mentioned before, if at any time two or more of the three codes above are in the same base font area of a document and regardless of their order in the document file, process the codes in the following order:

Orientation
Character Set
Attributes

There are no exceptions to this rule. Otherwise, you will chain to an incorrect font block and will identify a font incorrectly.

Backtracking
WordPerfect does not backtrack through the AFC chain and neither should you. WordPerfect returns directly back to the base font and restarts the chain.

Minimum Information Needed by WordPerfect to Create Fonts
This section is for those who are coding programs to create WordPerfect documents outside of WordPerfect.

Despite all the information that goes into a font selection, the only thing your program needs to consider is creating the correct Font Change code. WordPerfect will create the correct Packet Type 7 and Packet Type 15 (0xF) information.

There are two parts to a Font Change code: the code and its font descriptor. The Font Change code tells WordPerfect to change fonts and, in the case of WordPerfect 5.1, what the point size of the font should be. The font descriptor tells WordPerfect which font you want to use for this change.

There are two general ways developers tend to produce these codes. In both methods, the developer usually doesn't create the Font Change codes from scratch. Because of all of the information involved, it is easier to let WordPerfect do the work for you. For either font change creation method, start WordPerfect. Select a printer definition (you don't need to have the printer) that best suits the fonts you want to support. Select that as the default printer, then go to a blank editing screen.

Method 1
In this method, the Font Change codes are treated as one unit. Choose each of the fonts you want to use and select the initial point size while you are in WordPerfect. To help you in the next steps, you can put a character after each Font Change code that WordPerfect creates.

When you finish putting all of the Font Change codes into the document, give it a file name and save it. You can use a binary editor to extract each of the Font Change codes and put them into your program so that your program can reference the name of each font with a particular Font Change code.

Method 2
In this method, the font is considered in the same two parts mentioned in method 1: the font descriptor and the font code. However, the font code information is created by the program based on a generalized template.
As in method 1, the developer creates all of the font changes in WordPerfect. Instead of extracting each function code, the developer extracts each font descriptor from the Font Change codes.

If you use method 2, there is one important thing to keep in mind: If you change the point size (in the Font Change code) for one of these fonts, you must always change the "Cell Height" and "Optimal Width" fields of the corresponding font descriptor as well. This keeps compatibility with all releases of WordPerfect 5.0 and many early releases of WordPerfect 5.1.

If you create the document outside of WordPerfect, you may want to add fonts. To create the information in the Font Change code in the easiest way, first enter WordPerfect. Then, select the default printer that best fits the fonts you want to use. Using WordPerfect, create a document that contains the Font Change codes for each of the fonts. Make copies of all the Font Change codes and have the program that creates the document use these function code segments as templates for each of the particular fonts.