

Microsoft Word 97 Binary File Format

Microsoft Word 97 (aka Version 8) for Windows and Macintosh. From the Office book, found in the Microsoft Office Development section in the [MSDN Online Library](#). HTMLified June 1998. Revised Aug 1 1998, added missing Definitions section. Revised Dec 21 1998, added missing Document Properties (section).

Contents

- [Note](#)
- [Word and Docfiles](#)
- [Definitions](#)
- [Naming Conventions](#)
- [Format of the Summary Info Stream in a Word File](#)
- [Format of the Main Stream in a Word Non-Complex File](#)
- [Format of the Main Stream in a Complex File](#)
- [Format of the Table Stream](#)
- [Format of the Data Stream](#)
- [FIB](#)
- [Text](#)
- [Character and Paragraph Formatting Properties](#)
- [Bin Tables](#)
- [Stylesheet](#)
 - [STSHI](#)
 - [STD](#)
- [List Tables](#)
 - [LST records and the rglst](#)
 - [List Names and the stbListNames](#)
 - [LFO records and the pllfo](#)
 - [Paragraph List Formatting](#)
- [SPRM Definitions](#)
- [Complex File Format](#)
 - [Algorithm to determine the bounds of a paragraph containing a certain character in a complex file](#)
 - [Algorithm to determine paragraph properties for a paragraph in a complex file](#)
 - [Algorithm to determine table properties for a table row in a complex file](#)
 - [Algorithm to determine the character properties of a character in a complex file](#)
 - [Algorithm to determine the section properties of a section in a complex file](#)
 - [Algorithm to determine the pic of a picture in a complex file.](#)
- [Footnotes & Endnotes](#)
- [Headers and Footers](#)
- [Page Table](#)
- [Glossary Files](#)
- [Routing Slip](#)
- [Autosummary](#)
- [STTBFASSOC \(Table of Associated Strings\)](#)
- [Structure Definitions](#)
 - [AnnoTation Reference Descriptor \(ATRD\)](#)
 - [Autonumbered List Data Descriptor \(ANLD\)](#)
 - [Autonumber Level Descriptor \(ANLV\)](#)
 - [AutoSummary Analysis \(ASUMY\)](#)
 - [AutoSummary Info \(ASUMYI\)](#)
 - [Bin Table Entry \(BTE\)](#)
 - [BreaK Descriptor \(BKD\)](#)
 - [BookMark First descriptor \(BKF\)](#)

- [BookMark Lim descriptor \(BKL\)](#)
- [Border Code \(BRC\)](#)
- [Border Code for Windows Word 1.0 \(BRC10\)](#)
- [Character Properties \(CHP\)](#)
- [Character Property Exceptions \(CHPX\)](#)
- [Date and Time \(internal date format\) \(DTTM\)](#)
- [Drop Cap Specifier\(DCS\)](#)
- [Drawing Object Grid \(DOGRID\)](#)
- [Document Properties \(DOP\)](#)
- [Document Typography Info \(DOPTYPOGRAPHY\)](#)
- [Field Descriptor \(FLD\)](#)
- [File Shape Address \(FSPA\)](#)
- [Font Family Name \(FFN\)](#)
- [File Information Block \(FIB\)](#)
- [Footnote Reference Descriptor \(FRD\)](#)
- [Formatted Disk Page for CHPXs \(CHPX FKP\)](#)
- [Formatted Disk Page for PAPXs \(PAPX FKP\)](#)
- [List LeVeL \(on File\) \(LVLF\)](#)
- [Line Spacing Descriptor \(LSPD\)](#)
- [LiST Data \(on File\) \(LSTF\)](#)
- [List Format Override \(LFO\)](#)
- [List Format Override for a single LeVeL \(LFOLVL\)](#)
- [Outline LiST Data \(OLST\)](#)
- [Number Revision Mark Data \(NUMRM\)](#)
- [Page Descriptor \(PGD\)](#)
- [Paragraph Height \(PHE\)](#)
- [Paragraph Properties \(PAP\)](#)
- [Paragraph Property Exceptions \(PAPX\)](#)
- [Picture Descriptor \(on File\) \(PICF\)](#)
- [Piece Descriptor \(PCD\)](#)
- [Plex of CPs stored in File \(PLCF\)](#)
- [Property Modifier\(variant 1\) \(PRM\)](#)
- [Property Modifier\(variant 2\) \(PRM\)](#)
- [Routing Slip \(RS\)](#)
- [Routing Recipient \(RR\)](#)
- [Section Descriptor \(SED\)](#)
- [Section Properties \(SEP\)](#)
- [Section Property Exceptions \(SEPX\)](#)
- [Shading Descriptor \(SHD\)](#)
- [Tab Descriptor \(TBD\)](#)
- [Table Cell Descriptors \(TC\)](#)
- [Table Autoformat Look sPecifier \(TLP\)](#)
- [Table Properties \(TAP\)](#)
- [TeXtBoX Story \(FTXBXS\)](#)
- [WoRk Book \(WKB\)](#)
- [Appendix A - Reading a Macintosh PICT Graphic](#)
- [Appendix B - Calculation of font \(FTC\) and language \(LID\)](#)

Note

Many of the structures written in Word files differ slightly from the corresponding structures Word uses internally. The file-specific version of a structure is typically named by adding a preceding or (more often) trailing F. For example, Word uses internally a PLC (PLex of Cps), but writes to files a PLCF (PLex of Cps in File). Many discussions in this document use the name of the internal structure when the file-specific structure is what is really being referred to. The reader should remember that the name of a seemingly undefined structure type may simply be missing a leading or trailing F.

Word and Docfiles

Word 97 is an OLE 2.0 application. A Word binary file is a docfile and Word binary data is written into streams within the docfile using the OLE 2.0 docfile APIs. These streams are stored in the file as linked lists of file blocks and this data cannot be reliably accessed by using the operating system Open APIs. To access data within a Word binary file, the file must be opened using the OLE 2.0 docfile APIs, and it must be read with the appropriate docfile APIs.

A word docfile consists of a main stream, a summary information stream, a table stream, a data stream, and 0 or more object streams which contain private data for OLE 2.0 objects embedded within the Word document. The summary information stream is described in the section immediately following this one. The object storages contain binary data for embedded objects. Word has no knowledge of the contents of these storages; this information is accessed and manipulated through the OLE 2.0 APIs.

The majority of this document describes the contents of the main stream and the table stream.

Definitions

OLE 2.0:

Object Linking and Embedding 2.0

API (Application Programming Interface):

A set of libraries, functions, definitions, etc. which describe an interface to a programming environment or model.

docfile:

An OLE 2.0 compatible multi-stream file. Word files are docfiles.

page (or sector):

512 byte segment of the main stream within a Word docfile that begins on a 512-byte boundary. (bytes 0-511 are in page 0, bytes 512-1023 are in page 1, etc.). In Word data structures, an unsigned two-byte integer page number is given the acronym **PN** (for **P**age **N**umber).

document:

A named, multi-linked list of data structures, representing an ordered stream of text with properties that was produced by a user of Microsoft Word

stream:

The physical encoding of a Word document 's text and sub data structures in a random access stream within a docfile.

main stream:

The stream within a Word docfile containing the bulk of Word's binary data.

table stream:

The stream within a Word docfile containing the various plcf's and tables that describe a documents structures.

data stream:

The stream within a Word docfile containing various data that hang off of characters in the main stream. For example, binary data describing in-line pictures and/or formfields.

summary information stream:

The stream within a Word docfile containing the document summary information.

object storage:

A storage containing binary data for an embedded OLE 2.0 object.

CP (Character Position):

A four-byte integer which is the position coordinate of a character of text within the logical text stream of a document.

FC(File Character position):

A four-byte integer which is the byte offset of a character (or other object) from the beginning of a stream of the docfile. Before a file has been edited(i.e. in a full saved Word document), CPs can be transformed into FCs by adding the FC coordinate of the beginning of a document's text stream to the CP. After a file has been edited (i.e. in a fast-saved Word document), the mapping from CP to FC is recorded in the **piece table** (see below)

XCHAR(eXtended CHARacter set):

A data type which defines a "character". Each XCHAR corresponds to a character in the document, where "character" is defined as a glyph, regardless of whether it is a single-byte or double-byte character. With Word6/FE, Word95/FE, Word97/all and future versions of Word, this is defined as a 16-bit integer corresponding to the Unicode character code of the glyph.

PLF(PLex stored in File):

A data structure consisting of an array of structures preceded by a long count of structures.

PLCF(PLex of Cps(or FCs) stored in File):

A data structure consisting of two parallel arrays that allows a relation to be established between a certain CP position in the document text stream (or FC position in a file) and an arbitrary data structure. It consists of an array of **n+1** CPs or FCs followed by an array of **n** instances of a particular arbitrary data structure. In typical usage, the **nth** CP or FC of the **PLCF** is in one-to-one correspondence with the **nth** instance of the arbitrary data structure, with the **n+1st** CP or FC marking the limit of the **nth** instance's influence. When a **PLCF** is used to record a partitioning of the document's text stream or a partitioning of the bytes stored in a file, the 0th CP/FC stored in the **PLCF** will be 0. When a **PLCF** is used to record the location of certain marks or links within the document text stream, the 0th CP/FC stored in the **PLCF** will record the position of the 0th mark or link. To properly interpret a **PLCF** stored in a Word file, the length of the stored **PLCF** and the length of the arbitrary data structure stored in the **PLCF** must be known. The length of the stored **PLCF** is recorded in the **FIB**. The lengths of the data structures stored in **PLCFs** within Word files are listed later in this document.

piece table:

The **piece table** is a data structure that describes the logical sequence of characters in a Word document and records recent changes to the formatting of a Word document. It is stored in a Word file as a **PLCF** named the **plcfpcd** (**PLex** of Cps containing **Piece Descriptors**).The piece table relates a logical character number, called a **CP** (**Character Position**), to a physical location within a Word file (an **FC**). The array of **CPs** in the **plcfpcd** defines a partitioning of the Word document into disjoint pieces. The second array is an array of **PCDs** (**Piece Descriptors**) which is in 1-to-1 correspondence to the array of **CPs** that records the physical location in the Word file where the corresponding piece begins. To find the physical location of a particular logical character in a Word document, take the **CP** coordinate of that character within the document and find the piece that contains that character. This is done by finding the index of the largest **CP** in the array of **CPs** that is less than the character **CP**. Then reference the **PCD** with that index in the array of **PCDs**. The **FC** stored in the **PCD** gives the position of the beginning of the piece in the file. Finally, add the offset of the desired character from the beginning of its piece to the **FC** of the beginning of the piece. This gives a "virtual" file offset of the character. If the second most significant bit is clear, then this indicates the actual file offset of the unicode character (two bytes). If the second most significant bit is set, then the actual address of the codepage-1252 compressed version of the unicode character (one byte), is actually at the offset indicated by clearing this bit and

dividing by two.

sprm (Single PProperty Modifier):

An instruction to modify one or more properties within one of the property defining data structures (**CHP, PAP, TAP, SEP, or PIC**). It consists of an operation code which identifies the field(s) to be changed, and an operand which gives the value that a particular field is changed to or else which is a parameter to a procedure which will change the field or fields. A **prl** (property modifiers stored in a list) is a **sprm** plus its operand.

grppl (group of prls):

A **grppl** is a data structure that records a set of **sprms**. The 0th **sprm** is recorded at offset 0 of the structure. Any succeeding **sprms** are recorded immediately after the end of the preceding **sprm**. To traverse a **grppl** and locate the **sprms** recorded within it, it's necessary to fetch the opcode of the first **sprm**, lookup the length of the **sprm** with that opcode, use that length to skip past the first **sprm**, fetch the opcode of the second **sprm**, lookup the length of that **sprm**, use the length to skip the second **sprm**, and so on. See the table in the "**SPRM Definition**" topic to determine the length of a **sprm**.

The phrase "**apply the sprms of a grppl** (or **papx** or **sepx**)" used later in this document means to fetch the 0th **sprm** recorded in the **grppl** and perform the action for that **sprm**, fetch the first **sprm** and perform its action, and continue this procedure until all **sprms** in the **grppl** (or **papx** or **sepx**) have been processed.

prm (PProperty Modifier):

A field in piece table entries that records how the properties of text within a piece were changed to reflect user formatting operations. The **prm** usually contains an index to a **grppl** which records the user's formatting changes as a group of **sprms**. If the user has made only a small change to formatting that can be expressed as a single 2 or 1-byte **sprm**, that **sprm** is stored within the **prm**.

STTBF (STring TaBle stored in File)

Word has many tables of strings that are stored as Pascal type strings. STTBFs consist of an optional short containing 0xFFFF, indicating that the strings are extended character strings, a short indicating how many strings are included in the string table, another short indicating the size in bytes of the extra data stored with each string and each string followed by the extra data. Non-extended character Pascal strings begin with a single byte length count which describes how many characters follow the length byte in the string. If *pst* is a pointer to an array of characters storing a Pascal style string then the length of the string is $*pst+1$. In an STTBF Pascal style strings are concatenated one after another until the length of the STTBF recorded in the FIB is exhausted. Extra data associated with a string may also be stored in an *sttbf*. When extra data is stored for an STTBF, it is written at the end of each string. For example: The extra data for an STTBF consists of a short. If the string "Cat" were stored, the actual entry in the string table would consist of a length byte containing 3 (3 for "Cat") followed by the bytes 'C' 'a' 't', followed by the 2 bytes containing the short. Extended character strings are stored just the same, except they have a double byte length count and each extended character occupies two bytes.

full-saved (or non-complex) file:

A Word file in which the physical order of characters stored in the file is identical to the logical order of characters in the document that the file represents. The text stream of a non-complex file can be described by an **fc** (an offset from the beginning of the file) to mark where the text begins and a **ccp** (count of CPs) to record how many characters are stored in the text stream. Due to unicode compression to code page 1252, all files (simple and complex) now contain a piece table. However, a full-saved piece table will not have property modifiers (**prms**) and all text in the file will be referenced by the piece table.

fast-saved (or complex) file:

A Word file in which the physical order of characters stored in the file does not match the logical order of characters in the document that the file represents. A **piece table** must be stored in the file to describe the text stream of the document. Due to unicode compression to code page 1252, all files (simple and complex) now contain a piece table.

FIB (File Information Block):

The header of a Word file. Begins at offset 0 in file. Gives the beginning offset and lengths of the document's text stream and subsidiary data structures within the file. Also stores other file status information.

paragraph

A contiguous sequence of characters within the text stream of a document that is delimited by a paragraph mark, cell mark, row mark, or a section mark (These are special characters described later in this document).

run of text

A contiguous sequence of characters within the text stream of a document that have the same character formatting properties. A single run may cross paragraph boundaries and may encompass the entire document.

section

A contiguous sequence of paragraphs within the text stream of a document that is delimited by a section mark or by the final paragraph mark at the end of a document. Users frequently treat sections as the equivalent of a chapter in a book. The boundaries of sections mark locations where the layout rules for a document (number of columns, text of headers and footers to use, whether page numbers should be displayed, etc.) are changed.

paragraph style

A named set of character and paragraph properties that can be associated with any number of **paragraphs** in a Word document's text stream. A **paragraph style** provides a set of character and paragraph property defaults for the text of any paragraph tagged with that style. When a new paragraph is created and given a particular style, newly typed text is given the character and paragraph properties of that style unless the user makes an exception to the paragraph style definition by performing other editing operations.

CHP (CHaracter Properties)

The data structure describing the character properties of a run of text.

CHPX (Character Property EXception)

A data structure which describes how a particular **CHP** differs from a reference **CHP**. In Win Word 6.0, the **CHPX** simply consists of a **grpprl** which is applied to the reference **CHP** to produce the originally encoded **CHP**. By applying a **CHPX** to the character properties (**CHP**) inherited by a particular paragraph from its **style**, it is possible to reconstitute the **CHP** for the portion of the character run that intersects that paragraph

character style

A named character property exception that can be associated with any number of runs of text in a Word document's text stream. When a run of text is tagged with a particular **character style**, a **chpx** recorded for the character style is applied to the character properties that are defined for the paragraph style of the paragraph that contains the text. This means that the character style can change one or more of the character property field settings specified by the paragraph style of a paragraph to a particular setting without changing the value of any other field.

PAP (PARagraph Properties)

The data structure which describes the properties of a particular paragraph.

PAPX (PARagraph Property EXception)

A data structure describing how a particular paragraph's properties differ from the paragraph properties of the style assigned to the paragraph. By applying a **PAPX** to the paragraph properties (**PAP**) inherited by a particular paragraph from its **style**, it is possible to reconstitute the **PAP** for that paragraph. The **PAPX** contains an **ISTD** (a style code to identify the style in control of the paragraph and a **grpprl** which specifies how the style's paragraph properties must be changed to produce the paragraph

properties of the paragraph.

table row:

A contiguous sequence of paragraphs within the text stream of a document that is partitioned into subsequences of paragraphs called **cells**. The last paragraph of each cell is terminated by a special paragraph mark called a **cell mark**. Following the cell mark that ends the last cell of a table row, the table row is terminated by a special paragraph mark called a **row mark**. When Word displays a table row, it assigns a rectangular shaped display area to each cell in the row. All of the cell display area's top's are aligned at the same vertical position on a page. The leftmost display area in a table row is assigned to the 0th cell of the row; the next display area to the right is assigned to the 1st cell of the row, etc. The text of the cell is wrapped to fit its display area. As more text is added to the cell, the cell display area extends downward. A set of table properties that determine how many cells are in a row, where the horizontal boundaries of cell display areas are, and what borders are drawn around each cell in the table is stored for the **row mark** that marks the end of the table row.

TAP (Table Properties):

The data structure which describes the properties of a single table row. The information in the **TAP** for a table row is stored in a Word file as a list of sprms that modify a **TAP** which has been cleared to zeros. This list of table sprms is appended to the **grppl** of paragraph sprms that is recorded in the **PAPX** for the **row mark** that delimits the end of a **table row**.

STSH (STyle SHeet)

A data structure which represents every style defined within the Word document. The **STSH** records a unique name string for every style and associates each name with a particular **CHP** and/or a **PAP**. The indexes used to refer to individual styles are called **ISTDs** (Indexes to **STyle Descriptors**). Every **PAPX** for every paragraph recorded in a document contains an **ISTD** which identifies the style from which a paragraph inherited its default character and paragraph properties. **CHPXs** recorded for the text within the paragraph and **PAPXs** recorded for the paragraph itself encode changes that the user has made with respect to the style's default properties.

FKP (Formatted disk Page):

A data structure that fits in one 512-byte page that encodes either the character properties or the paragraph properties of a certain portion of a Microsoft Word file. An **FKP** consists of four components:

- 1) a count of the number of runs or paragraphs described by the page.
- 2) an array of **FCs** recorded in ascending order demarcating the boundaries between runs or paragraphs that are recorded adjacent to one another in the Word file.
- 3) In **character FKP**s an array of offsets within the **FKP** in one to one correspondence with the array of **FCs** that locate the properties of the run that begins at a particular **FC**.

In **LVC FKP**s an array of offsets within the **FKP** in one to one correspondence with the array of **FCs** that locate the **LVCXs** that describe the run that begins at a particular **FC**.

In **paragraph FKP**s an array of **BX** structures follows the array of **FCs** in one to one correspondence with the array of **FCs**. Each **BX** begins with an offset that locates the properties of the paragraph that begins at a particular **FC**. The remainder of the **BX** contains a **PHE** structure that encodes information about the height of the paragraph that begins at that **FC**.

- 4) a group of **CHPXs** if the **FKP** stores character properties, a group of **PAPXs** if the **FKP** stores paragraph and table properties, or a group of **LVCXs** if the **FKP** stores paragraph level and numbering cache information

To find the **CHPX/PAPX** corresponding to a particular character in a document, calculate the **FC** coordinate for that character. Then search through the **bin table** (see next entry) for the type of property you want to produce, to find the **FKP** in the document stream whose array of **FCs** encompasses the **FC** of the document character.

Then search within the **FKP** to find the index of the largest **FC** entry that is less than or equal to the **FC** of the document

character. Use this index to look up an offset in the array of offsets (for **character FKPs**) or look up an offset in the array of **Bxs** (for **paragraph FKPs**) within the **FKP**. Add this offset to the beginning address of the **FKP** in memory. This will be the first byte of the desired **CHPX/PAPX**.

bin table

Each **FKP** can be viewed as bucket or **bin** that contains the properties of a certain range of **FCs** in the Word file. In Word files, a **PLC**, the **plcfbte** (**PLex** of **FCs** containing **Bin Table Entries**) is maintained. It records the association between a particular range of **FCs** and the **PN** (**Page Number**) of the **FKP** that contains the properties for that **FC** range in the file. In a **complex (fast-saved)** Word document, **FKP** pages are intermingled with pages of text in a random pattern which reflects the history of past fast saves. In a complex document, a **plcfbteChpx** which records the location of every **CHPX FKP** must be stored and a **plcfbtePapx** which records the location of every **PAPX FKP** must be stored. In a **non-complex, full-saved** document, all of the **CHPX FKPS** are recorded in consecutive 512-byte pages with the **FKPs** recorded in ascending **FC** order, as are all of the **PAPX FKPS**. A **plcfbteLvex** serves the same purpose for **LVCX FKPS**.

In a full save document, the **plcfbte's** may not have been able to be expanded during the save process due to a lack of RAM. In that situation, the **plcfbte's** will be interspersed with the property pages in a linked list of **FBD** pages.

SEP(Section Properties)

The data structure describing the properties of a particular section.

SEPX(SEction Property EXceptions)

A data structure describing how the properties of a particular section differ from a Word-defined standard **SEP**. As in the **PAPX**, the differences between the **SEP** for a section and the standard **SEP** are encoded as list of **sprms** that describe how the standard **SEP** can be transformed into the section's **SEP**. By applying a **SEPX's** **sprms** to the standard **SEP**, it is possible to reconstitute the **SEP** for that section.

The **PLCFSED**, a data structure stored in a Word file, records the locations of all **SEPXs** stored in a Word file. The array of **CPs** in the **plcfSED** records the boundaries of sections in the Word document. The second array in the **plcf**, an array of **SEDs** (**Section Descriptors**), is in 1-to-1 correspondence to the array of **CPs**. Each **SED** stores the beginning **FC** of the **SEPX** that records the properties for a section. If the **FC** stored in a **SED** is -1, the section properties of the section are exactly equal to the standard section properties.

The **SEP** for a particular section may be constructed if a **CP** of a character in that section is known. First search the array of **CPs** in the **PLCFSED** for the index of the largest **CP** that is less than or equal to the **CP** of the character. Use this index to locate the **SED** in the **plcfSED** which describes the section. The **FC** stored in the **SED** is the offset from the beginning of the Word file at which the **SEPX** is stored. If the stored **FC** is equal to 0xFFFFFFFF, then the **SEP** for the section is exactly equal to the standard **SEP** (see **SEP** structure definition) Otherwise, read the **SEPX** into memory and create a copy of the standard **SEP**. Finally, apply the **sprms** stored in the **SEPX** to the standard **SEP** to produce the **SEP** for a section.

DOP (Document Properties)

The data structure describing properties that apply to the document as a whole.

sub-document

A separate logical stream of text with properties for which correspondences with the main document text are maintained. Word's headers/footers, footnotes, endnotes, macro procedure text, annotation text, and text within textboxes are kept in separate subdocuments. Each subdocument has its own **CP** coordinate space. In other words, data structures are stored in Word files that are components of these subdocuments. These data structures contain **CP** coordinates whose 0 point is the beginning of the subdocument text stream instead of the beginning of the main document text stream.

In **full-saved documents**, a simple calculation with values stored in the **FIB** produces the file offset of the beginning of the subdocument text streams (if they exist). The length of these streams is also stored.

In **fast-saved documents**, the **piece tables** of subdocuments are concatenated to the end of the main document piece table. In this case, to identify the beginning of subdocument text, you must sum the length of the main document text stream with the lengths of any subdocument text streams stored ahead of the subdocument (information stored in the **FIB**) and treat this sum as a **CP** coordinate. To retrieve the text of the subdocument, you must do lookups in the piece table, starting with the piece that contains the beginning **CP** coordinate, to find the physical location of each piece of the subdocument text stream.

field

A field is a two-part structure that may be recorded in the CP stream of a document. The first part of the structure contains **field codes** which instruct Window's Word to insert text into the second part of the structure, the **field result**. Fields in Window's Word are used to insert text from an external file or to quote another part of a document, to mark index and table of contents entries and produce indexes and tables of contents, maintain DDE links to other programs, to produce dates, times, page numbers, sequence numbers, etc. There are 91 different field types.

A **field begin mark** delimits the beginning of a field and precedes any of the field codes stored in the field. The end of the field codes and the beginning of the field result is marked with the **field separator** and the field result and the field itself are terminated by a **field end mark**.

The CP locations of the field begin mark, field separator, and field end mark are recorded in **plcfld** data structures that are maintained for the main document and all of the subdocuments of the main document whenever a field is inserted or edited. A field can be **dead**, in which case it has no field separator, no field result, and no entry in the **plcfld**. (See the definition of the FLD structure for a list of possible dead field code strings.) An array of two-byte **FLD** structures is stored in the **plcfld** in one-to-one correspondence with the CP entries recorded. An **FLD** associated with a **field begin mark** records the type of the field. An **FLD** associated with the **field end mark** records the current status of the field (i.e. whether the result is dirty or has been edited, whether the result has been locked, etc.)

Fields may be nested. 20 levels of nesting are permitted.

bookmark

A **bookmark** associates a user definable name with a range of text within a document. A bookmark is frequently used as an operand in **field code** instructions within a field. In Window's Word a bookmark is represented by three parallel data structures, the **sttbBkmk**, the **plcbkf** and the **plcbkl**. The **sttbBkmk** is a string table which contains the name of each bookmark that is defined. The **plcbkf** records the beginning CP position of each bookmark. The **plcbkl** records the limit CP position that delimits the end of a bookmark. Since bookmarks may be nested within one another to any level, the **BKF** structure stored in the **plcbkf** consists of a single index which specifies which **plcbkl** marks the end of the bookmark. The **BKL** structure is not written to the file, and the **plcbkl** contains only CPs.

picture

A picture is represented in the document text stream as a special character, an ASCII 1 whose CHP has the fSpec bit set to 1. The file location of the picture in the Word binary file is stored in the character's CHP in **chp.fcPic**. The **fcPic** is a byte offset into the data stream. Beginning at the position recorded in **chp.fcPic**, a header data structure, the **PIC**, will be stored. If the picture is a reference to a TIFF file, a Picture file or an Office shape file, the name of the file will be recorded immediately following the **PIC** in a Pascal style string. If the picture is an Office shape, a Window's metafile or a bitmap, the shape, metafile or bitmap will immediately follow the **PIC**. Pictures that are a reference to an Office shape file will include both the filename and the shape in that order. Pictures inserted with Word97 are in the new Office shape format (documented elsewhere). However, pictures can be copied from older files into newer ones and their old format will persist until the picture is edited or displayed.

Some files (including all files created by Word for the Macintosh) may store Macintosh PICT pictures as well. In this case, the **PIC** structure is immediately followed by a standard Windows metafile depicting a large "x", so that older readers expecting only a metafile after the **PIC** will just display this "x". If a reader detects this standard "x" metafile, it can extract the sizes of the standard "x" metafile and the Macintosh PICT picture that follows it from an early portion of this "x" metafile. Please see Appendix B for a discussion of this technique.

embedded object

The native data for Embedded objects (OBJs) is stored similarly to pictures (PICs). To locate the native data for Embedded objects, scan the plc of field codes for the mother, header, footnote and annotation, textbox and header textbox documents (fib.PlcffldMom/Hdr/Ftn/Atn/Txbx/HdrTxbx). For each separator field, get the chp.

If chp.fSpec=1 and chp.fObj=1, then this separator field has an associated embedded object. The file location of the object data is stored in chp.fcObj. At the specified location an object header is stored followed by the native data for the object. See the **_OBJHEADER** structure.

If chp.fOle2=1, then this separator field has an associated OLE2 object. The fcPic will be a unique integer that specifies the name of the object's sub-storage instead of an offset into the data stream.

office art object

An office art object is represented in the document stream as a special character, an ASCII 8, which has chp.fSpec set to 1 for the run of text containing the character. Only main documents and header documents contain office art objects. The native data for the office art object may be obtained by taking the CP for the special character and using this to find the corresponding entry in the **plcspa**. An entry in this plc consists of a **FSPA** structure, which is described elsewhere in this document.

Office art objects can have text attached to them. Text for the textboxes is stored separately in the textbox subdocument of the main or header document. The textbox subdocument contains a **plctxbxs** where the text from CP n to CP n+1 in the subdocument is the text which is contained in a textbox as specified in the **TXBXS** structure for this nth entry in the **plctxbxs**. Textboxes can be linked in chains of up to 32 textboxes. Ordering of textboxes in the subdocument is completely unrelated to the document structure due to the nature of textbox linking. To find the text for a given office art object, the **TXID** property (a long: high word is itxbxs+1, low word is the sequence number) must be fetched from the office art data for the shape. This contains an index (itxbxs) into **plctxbxs** and a sequence number in the chain of linked textboxes. The text for the entire chain of linked textboxes is stored from the CP itxbxs to CP itxbxs+1 of plctxbxs. The **plctxbxBkd** describes the "page table" within textbox stories (where the textboxes in each linked textbox chain are thought of as "pages"). So, for each entry in the plctxbxs there is a corresponding entry in the **plctxbxBkd** at the same CP, and there may be additional entries in the **plctxbxBkd** to describe the breaks from one textbox to the next in linked textbox chains.

Note

In this document, bit 0 is the low-order bit. Structures are described as they would be declared in C for the Intel architecture. When numbering bytes in a word from low offset towards high offset, two-byte integers will have their least significant eight bits stored in byte 0 and most significant eight bits in byte 1. If bit 31 is the most significant bit in a four-byte integer, bits 31 through 24 will be stored in byte 3 of a four-byte integer, bits 23 through 16 will be stored in byte 2, bits 15 through 8 will be stored in byte 1, and bits 7 through 0 will be stored in byte 0.

Naming Conventions

The names in Word data structures usually consist of a lower case sequence of characters followed by an optional upper case modifier. The following tags are used in the lower case parts of field names to document the data type of a field:

f	used to name a flag (a variable containing a Boolean value). Usually the object referred to will contain either 1 (<i>fTrue</i> , <i>TRUE</i>) or 0 (<i>fFalse</i> , <i>FALSE</i>). (e.g. fWidowControl, fShadow)
l	used to name a 4 byte integer value (a long). (e.g. lcb)
w	used to name a 2 byte integer value (a short).
b	used to name a 1 byte integer value
cp	used to name a variable that contains a character position within the document. always a 4 byte quantity.
fc	used to name a variable that contains an offset from the beginning of a file. always a 4 byte quantity.

xa	used to name a variable that contains a width of an object imaged on screen or on hard copy that is measured in units of 1/1440 of an inch. This unit which is one-twentieth of a point size (1/20 * 1/72") is called a twip in this documentation. (e.g. xaPage is the width of a page).
ya	used to name a variable that contains a height of an object imaged on screen or on hard copy that is measured in twips.
dxaf	used to name a variable that contains the horizontal distance of an object measured from some reference point expressed in twips. (e.g. pap.dxaLeft is the distance of the left boundary of a paragraph measured from the left margin of the page)
dya	used to name a variable that contains the vertical distance of an object measured from some reference point expressed in twips. (e.g. pap.dyaAbs is the vertical distance of the top of a paragraph from a reference frame declared in the pap).
dxfp	used to name a variable that contains the horizontal distance of an object measured from some reference point expressed in Macintosh pixel units (1/72"). (e.g. dxfpSpace)
dypf	used to name a variable that contains the vertical distance of an object measured from some reference point expressed in Macintosh pixel units (1/72").
rg	prefix used to signify that the data structure being defined is an array. (e.g. rgb (an array of bytes), rgcp (an array of CPs), rgfc (an array of FCs), rgfoo (an array of foos).
i	prefix used to signify that an integer value is used as an index into an array. (e.g. itbd is an index into rgtbd, itc is an index into rgtc.)
c	prefix used to signify that an integer value is a count of some number of objects. (e.g. a cb is a count of bytes, a cl is a count of lines, ccol is a count of columns, a cpe is a count of picture elements.)
grp	prefix used to name an array of bytes that contains one or more copies of a variable length data structure with the instances of the data structure stored one after the other in the array. (e.g. a grppl is a array of bytes that stores a group of prls.)
grpf	prefix used to name an integer or byte value whose bits are used as flags. (e.g. grpflhdt is a group of flags that records the types of headers that are stored for a particular section of a document).

The two following modifiers are used occasionally in this documentation:

First	Means that variable marks the first of a range of objects. For example, cpFirst would mark the first character position of a range of characters in a document. fcFirst would mark the file offset of the first byte of a range of bytes stored in a file.
Lim	Means the variable marks the limit of a range of objects (i.e. is the index of the last object in a range plus 1). For example, cpLim would be the limit CP of a range of characters in a document. fcLim would be the limit file offset of a range of bytes stored in a file.

Format of the Summary Info Stream in a Word File

The summary information for a Word document is stored in two structured storage streams, SummaryInformation and DocumentSummaryInformation. Information on the layout of the SummaryInformation stream can be found in Appendix B of the OLE 2 Programmers Reference.

Format of the Main Stream in a Word Non-Complex File

The main stream of a Word docfile (non-complex format) consists of the Word file header (FIB), the text, and the formatting information.

FIB

Stored at beginning of page 0 of the file. fib.fComplex will be set to zero.

text of body, footnotes, headers

Text begins at the position recorded in fib.fcMin.

FKPs for CHPs, PAPs and LVCs

The first FKP begins at a 512-byte boundary after the last byte text written.. The remaining FKPs are recorded in the 512-byte pages that immediately follow. The FKPs for CHPs PAPs and LVCs are interleaved. Previous versions of Word wrote them in contiguous chunks. The hplcfbte's of the three flavors (CHP, PAP and LVC) are used to find the relevant FKP of the appropriate type.

group of SEPXs

SEPXs immediately follow the FKPs and are concatenated one after the other. SEPXs are no longer guaranteed to start on a page boundary if it would span a boundary if placed immediately after the preceding SEPX.

Format of the Main Stream in a Complex File

The main stream of a Word binary file (complex format) consists of the Word file header (FIB), the text, and the formatting information.

FIB**Text of body, footnotes, headers stored during last full save**

Text begins at the position recorded in fib.fcMin.

FKPs for CHPs, PAPs and LVCs

The first FKP begins at a 512-byte boundary after the last byte text written.. The remaining FKPs are recorded in the 512-byte pages that immediately follow. The FKPs for CHPs PAPs and LVCs are interleaved. Previous versions of Word wrote them in contiguous chunks. The hplcfbte's of the three flavors (CHP, PAP and LVC) are used to find the relevant FKP of the appropriate type.

Group of SEPXs stored during last full save

Any text, stored during first fast save
 Any FKPs stored during first fast save
 Any SEPXs stored during first fast save
 Any text, stored during second fast save
 Any FKPs stored during second fast save
 Any SEPXs stored during second fast save
 ...
 Any text, stored during nth fast save
 Any FKPs stored during nth fast save
 Any SEPXs stored during nth fast save

Format of the Table Stream

Word stores various plcfs and tables with the stream named either "0Table" or "1Table". Ordinarily a file will contain only one table stream. However, in some unusual circumstances (e.g. crash during file save) a file might have two table streams. In that case the bit field fWhichTblStm in the FIB should be used to determine which table stream to read. If fWhichTblStm is 0, then the FIB refers to the stream named "0Table", and if fWhichTblStm is 1, then the FIB refers to the stream name "1Table".

stbfUssr

Undocumented undo / versioning data

plcupcRgbuse

Undocumented undo / versioning data

plcupcUsp

Undocumented undo / versioning data

uskf

Undocumented undo / versioning data

stsh (style sheet)

Written immediately after the previous table. This is recorded in all Word documents.

plcffndRef (footnote reference position table)

Written immediately after the **stsh** if the document contains footnotes

plcffndTxt (footnote text position table)

Written immediately after the **plcffndRef** if the document contains footnotes

pgdFtn (footnote text page description table)

Written immediately after the **plcffndTxt** if the document contains footnotes

bkdFtn (footnote text break descriptor table)

Written immediately after the **pgdFtn** if the document contains footnotes.

plcfendRef (endnote reference position table)

Written immediately after the previously recorded table if the document contains endnotes

plcfendTxt (endnote text position table)

Written immediately after the **plcfendRef** if the document contains endnotes

pgdEdn (endnote text page description table)

Written immediately after the **plcfendTxt** if the document contains endnotes

bkdEdn (endnote text break descriptor table)

Written immediately after the **pgdEdn** if the document contains endnotes

plcftxbxTxt (text box link table)

Written immediately after the previously recorded table if the document contains textboxes

plcftxbxBkd (text box break descriptor table)

Written immediately after the **plcftxbxTxt** if the document contains textboxes

plcfhdrtxbxTxt (header text box link table)

Written immediately after the previously recorded table if the header subdocument contains textboxes

plcfhdrtxbxBkd (header text box break descriptor table)

Written immediately after the **plcfhdrtxbxTxt** if the header subdocument contains textboxes.

grpXstAtnOwners (annotation owner table)

Written immediately after the previously recorded table if the document contains annotations.

plcfandRef (annotation reference position table)

Written immediately after the **grpXstAtnOwners** if the document contains annotations

plcfandTxt (annotation text position table)

Written immediately after the **plcfandRef** if the document contains annotations.

plcfsed (section table)

Written immediately after the previously recorded table. Recorded in all Word documents

pgdMother (page description table)

Written immediately after the **plcfsed** in all Word documents

bkdMother (break descriptor table)

Written immediately after the **pgdMother** in all Word documents

plcfphe (paragraph height table)

Written after the previously recorded table, if paragraph heights have been recorded. Only written during a fast save.

plcfsea (private)

PLCF reserved for private use by Word.

plcflvc (list and outline level table)

Written immediately after the previously recorded table during fast save only.

plcasumy (AutoSummary analysis)

Written immediately after the previously recorded table, if the document stored is in AutoSummary view mode.

sttbGlsy (glossary name string table)

Written immediately after the previously recorded table, if the document stored is a glossary.

sttbGlsyStyle (glossary style name string table)

Written immediately after **sttbGlsy**, if the document stored is a glossary.

plcflglsy (glossary entry text position table)

Written immediately after the previously recorded table, if the document stored is a glossary.

plcfhdd (header text position table)

Written immediately after the previously recorded table, if the document contains headers or footers.

plcfbteChpx (bin table for CHP FKPs)

Written immediately after the previously recorded table. This is recorded in all Word documents.

plcfbtePapx (bin table for PAP FKPs)

Written immediately after the **plcfbteChpx**. This is recorded in all Word documents.

plcfbteLvc (bin table for LVC FKPs)

Written immediately after the **plcfbtePapx**. This is recorded in all Word documents.

sttbfRMark (revision mark author string table)

Written immediately after **plcfbteLvc**, if the document contains revision marks.

PlcffldMom (table of field positions and statuses for main document)

Written immediately after the previously recorded table if the main document contains fields.

PlcffldHdr (table of field positions and statuses for header subdocument)

Written immediately after the previously recorded table, if the header subdocument contains fields.

PlcffldFtn (table of field positions and statuses for footnote subdocument)

Written immediately after the previously recorded table, if the footnote subdocument contains fields.

PlcffldAtn (table of field positions and statuses for annotation subdocument)

Written immediately after the previously recorded table, if the annotation subdocument contains fields.

PlcffldEdn (table of field positions and statuses for endnote subdocument)

Written immediately after the previously recorded table, if the endnote subdocument contains fields.

PlcffldTxbx (table of field positions and statuses for textbox subdocument)

Written immediately after the previously recorded table, if the textbox subdocument contains fields.

plcOcx (ocx position table)

Written immediately after the previously recorded table, if the document contains ole controls.

Undocumented.

PlcffldHdrTxbx (table of field positions and statuses for textbox subdocument of header subdocument)

Written immediately after the previously recorded table, if the textbox subdocument of the header subdocument contains fields.

dggInfo (office drawing information)

Written immediately after the previously recorded table. Format is described in the Office drawing group format document.

plcspaMom (office drawing table)

Written immediately after the previously recorded table, if the document contains office drawings.

plcspaHdr (header office drawing table)

Written immediately after the previously recorded table, if the header subdocument contains office drawings.

sttbfBkmk (table of bookmark name strings)

Written immediately after the previously recorded table, if the document contains bookmarks.

plcfBkmkf (table recording beginning CPs of bookmarks)

Written immediately after the **sttbfBkmk**, if the document contains bookmarks.

plcfBkmkl (table recording limit CPs of bookmarks)

Written immediately after the **plcfBkmkf**, if the document contains bookmarks.

sttbfAtnBkmk (table of annotation bookmark string names)

Written immediately after the previously recorded table, if the document contains annotations with bookmarks.

plcfAtnbkf (table recording beginning CPs of bookmarks in the annotation subdocument)

Written immediately after the **sttbfAtnBkmk** previously recorded table, if the document contains annotations with bookmarks.

plcfAtnbkl (table recording limit CPs of bookmarks in the annotation subdocument)

Written immediately after the **plcfAtnbkf** previously recorded table, if the document contains annotations with bookmarks.

plcfspl (spelling state table)

Written immediately after the previously recorded table. Records state of spell checking in a PLCF of SPLS structures.

plcfgram (grammar state table)

Written immediately after the previously recorded table. Records state of grammar checking in a PLCF of SPLS structures.

plcfwkb (work book document partition table)

Written immediately after the previously recorded table, if the document is a master document.

formFldSttbs (form field dropdown string tables)

Written immediately after the previously recorded table, if the document contains form field dropdown controls.

sttbCaption (caption title string table)

Written immediately after the previously recorded table, if the document contains captions.

sttbAutoCaption (auto caption string table)

Written immediately after the previously recorded table, if the document contains auto captions.

sttbFnm (filename reference string table)

Written immediately after the previously recorded table, if the document references other documents.

sttbSavedBy (last saved by string table)

Written immediately after the previously recorded table.

plcflst (list formats)

Written immediately after the end of the previously recorded, if there are any lists defined in the document.

This begins with a short count of LSTF structures followed by those LSTF structures.

This is immediately followed by the allocated data hanging off the LSTFs. This data consists of the array of LVLs for each LSTF. (Each LVL consists of an LVLf followed by two grpprls and an XST.)

plflfo (more list formats)

Written immediately after the end of the **plcflst** and its accompanying data, if there are any lists defined in the document. This consists first of a PL of LFO records, followed by the allocated data (if any) hanging off the LFOs. The allocated data consists of the array of LFOLVLFs for each LFO (and each LFOLVLF is immediately followed by some LVLs).

sttbListNames (more list formats)

Written immediately after the end of the **plflfo** and its accompanying data, if there are any lists defined in the document. This is a string table containing the list names for each list. It is parallel with the **plcflst**, and may contain null strings if the corresponding LST does not have a list name.

hplgosl (grammar option settings)

Written immediately after the previously recorded table. This undocumented structure maps LID and grammar checker type to grammar checking options.

stwUser (macro user storage)**routeSlip (mailer routing slip)**

Written immediately after the previously recorded table, if this document has a mailer routing slip.

cmds (recording of command data structures)

Written immediately after the previously recorded table, if special commands are linked to this document.

prDrvr (printer driver information)

Written immediately after the previously recorded table, if a print environment is recorded for the document.

prEnvPort (print environment in portrait mode)

Written immediately after the previously recorded table, if a portrait mode print environment is recorded for this document.

prEnvLand (print environment in landscape mode)

Written immediately after the previously recorded table, if a landscape mode print environment is recorded for this document.

wss (window state structure)

Written immediately after the end of previously recorded structure, if the document was saved while a window was open.

pms (print merge state)

Written immediately after the previously recorded table, if information about the print / mail merge state is recorded for the document

clx (encoding of the sprm lists for a complex file and piece table for a any file)

Written immediately after the end of previously recorded structure. This is recorded in all Word documents.

sttbffn (table of font name strings)

Written immediately after the **clx**. This is recorded in all Word documents. The **sttbffn** is an sttb where each string is instead an FFN structure (note that just as for a pascal-style string, the first byte in the FFN records the total number of bytes not counting the count byte itself). The names of the fonts correspond to the ftc codes in the CHP structure. For example, the first font name listed corresponds is the name for ftc = 0.

stbtmbd (true type font embedding string table)

Written immediately after the end of previously recorded structure if document contains embedded true type fonts.

dop (document properties record)

Written immediately after the end of previously recorded structure. This is recorded in all Word documents

sttbAssoc (table of associated strings)

autosaveSource (name of original)

Written immediately after the **sttbAssoc** table. This field only appears in autosave files. These files are normal Word documents in every other way. Also, autosaved files are typically in the complex file format except that we don't overwrite the tables (plcf*, etc.). I.e., an autosaved file is typically longer than the equivalent Word document.

Format of the Data Stream

pictures

Word picture structures are concatenated one after the other if the document contains pictures.

embedded objects-native data

Word embedded object structures are concatenated one after the other if the document contains embedded objects.

huge PAPXs

The grpprls from PAPXs which are too large to fit in an FKP are concatenated one after the other as necessary.

FIB

The FIB contains a "magic word" and pointers to the various other parts of the file, as well as information about the length of the file. The FIB starts at the beginning of the file. The FIB is [defined](#) in the structure definition section of this document.

Text

The text of the file starts at fib.fcMin. fib.fcMin is usually set to the next 128 byte boundary after the end of the FIB. The text in a Word document is ASCII text with the following restrictions (ASCII codes given in decimal):\

- **Paragraph ends** are stored as a single Carriage Return character (ASCII 13). No other occurrences of this character sequence are allowed.
- **Hard line breaks** which are not paragraph ends are stored as ASCII 11. Other line break or word wrap information is not stored.
- **Breaking hyphens** are stored as ASCII 45 (normal hyphen code); **Non-required hyphens** are ASCII 31. **Non-breaking hyphens** are stored as ASCII 30.
- **Non-breaking spaces** are stored as 160. Normal **spaces** are ASCII 32.
- **Page breaks** and **Section marks** are ASCII 12 (normal form feed); if there's an entry in the section table, it's a section mark, otherwise it's a page break.
- **Column breaks** are stored as ASCII 14.
- **Tab** characters are ASCII 9 (normal).
- The **field begin mark** which delimits the beginning of a field is ASCII 19. The **field end mark** which delimits the end of a field is ASCII 21. The **field separator**, which marks the boundary between the preceding field code text and following field expansion text within a field, is ASCII 20. The **field escape character** is the '\ character which also serves as the **formula mark**.
- The **cell mark** which delimits the end of a cell in a table row is stored as ASCII 7 and has the fInTable paragraph property set to fTrue (pap.fInTable == 1).
- The **row mark** which delimits the end of a table row is stored as ASCII 7 and has the fInTable paragraph property and fTtp paragraph property set to fTrue (pap.fInTable == 1 && pap.fTtp == 1).

The following ASCII codes are treated as "special" characters when they have the character property *special* on (chp.fSpec ==

1):

ASCII Code	Special Character
0	Current page number
1	Picture
2	Autonumbered footnote reference.
3	Footnote separator character
4	Footnote continuation character
5	Annotation reference
6	Line number
7	Hand Annotation picture (Generated in Pen Windows)
8	Drawn object
10	Abbreviated date (e.g. "Wed, Dec 1, 1993")
11	Time in hours:minutes:seconds
12	Current section number
14	Abbreviated day of week (e.g. "Thu" for "Thursday")
15	Day of week (e.g. "Thursday")
16	Day short (e.g. "9" for the ninth day of the month)
22	Hour of current time with no leading zero
23	Hour of current time (two digit with leading zero when necessary)
24	Minute of current time with no leading zero
25	Minute of current time(two digit with leading zero when necessary)
26	Seconds of current time
27	AM/PM for current time
28	Current time in hours:minutes:seconds in old format
29	Date M (e.g. "December 2, 1993")
30	Short Date (e.g. "12/2/93")
33	Short Month (e.g. "12" to represent "December")
34	Long Year (e.g. "1993")
35	Short Year (e.g. "93")
36	Abbreviated month (e.g. "Dec" to represent "December")
37	Long month (e.g. "December")
38	Current time in hours:minutes (e.g. "2:01")
39	Long date (e.g. "Thursday, December 2, 1993")
41	Print Merge Helper field

Note

The end of a section is also the end of a paragraph. The last character of a section is a section mark which stands in place of the paragraph mark normally required to end a paragraph. An exception is made for the last character of a document which is always a paragraph mark although the end of a document is always an implicit end of section.

If !fib.fComplex, the document text stream is represented by the text beginning at fib.fcMin up to (but not including) fib.fcMac. Otherwise, the document is represented by the piece table stored in the file in the data beginning at .fib.fcClx.

The document text stream includes text that is part of the main document, plus any text that exists for the footnote, header, macro, or annotation subdocuments. The sizes of the main document and the header, footnote, macro and annotation subdocuments are stored in the fib, in variables fib.ccpText, fib.ccpFtn, fib.ccpHdr, fib.ccpMcr, fib.ccpEdn, fib.ccpTxbx, fib.ccpHdrTxbx and fib.ccpAtn respectively. In a non-complex file, this means that the text of the main document begins at fib.fcMin in the file and continues through fib.fcMin + fib.ccpText; that the text of the footnote subdocument begins at fib.fcMin + fib.ccpText and extends to fib.fcMin + fib.ccpText + fib.ccpFtn; that the text of the header subdocument begins at fib.fcMin + fib.ccpText + fib.ccpFtn and extends to fib.fcMin + fib.ccpText + fib.ccpFtn + fib.ccpHdr; that the text of the annotation subdocument begins at .fib.fcMin + fib.ccpText + fib.ccpFtn + fib.ccpHdr and extends to fib.fcMin + fib.ccpText + fib.ccpFtn + fib.ccpHdr + ccpAtn; that the text of the endnote subdocument begins at .fib.fcMin + fib.ccpText + fib.ccpFtn + fib.ccpHdr + ccpAtn and extends to fib.fcMin + fib.ccpText + fib.ccpFtn + fib.ccpHdr + fib.ccpEdn; that the text of the textbox subdocument begins at .fib.fcMin + fib.ccpText + fib.ccpFtn + fib.ccpHdr + fib.ccpAtn + fib.ccpEdn and extends to fib.fcMin + fib.ccpText + fib.ccpFtn + fib.ccpHdr + fib.ccpEdn + fib.ccpTxbx and that the text of the header textbox subdocument begins at .fib.fcMin + fib.ccpText + fib.ccpFtn + fib.ccpHdr + fib.ccpAtn + fib.ccpEdn + fib.ccpTxbx and extends to fib.fcMin + fib.ccpText + fib.ccpFtn + fib.ccpHdr + fib.ccpEdn + fib.ccpTxbx + fib.ccpHdrTxbx.

In a complex, fast-saved file, the main document text must be located by examining the piece table entries from the 0th piece table entry through the piece table entry that describes cp = fib.ccpText.

A footnote subdocument's text must be located by examining the piece table entries beginning with the one that describes cp=fib.ccpText through the entry that describes cp = fib.ccpText + fib.ccpFtn.

A header subdocument's text must be located by examining the piece table entries beginning with the one that describes cp=fib.ccpText + ccpFtn through the entry that describes cp = fib.ccpText + fib.ccpFtn + fib.ccpHdr.

An annotation subdocument's text must be located by examining the piece table entries beginning with the one that describes cp=fib.ccpText + ccpFtn + fib.ccpHdr through the entry that describes cp = fib.ccpText + fib.ccpFtn + fib.ccpHdr + fib.ccpAtn.

An endnote subdocument's text must be located by examining the piece table entries beginning with the one that describes cp=fib.ccpText + ccpFtn + fib.ccpHdr + fib.ccpAtn through the entry that describes cp = fib.ccpText + fib.ccpFtn + fib.ccpHdr + fib.ccpAtn + fib.ccpEdn

A textbox subdocument's text must be located by examining the piece table entries beginning with the one that describes cp=fib.ccpText + ccpFtn + fib.ccpHdr + fib.ccpAtn + fib.ccpEdn through the entry that describes cp = fib.ccpText + fib.ccpFtn + fib.ccpHdr + fib.ccpAtn + fib.ccpEdn + fib.ccpTxbx

A header textbox subdocument's text must be located by examining the piece table entries beginning with the one that describes cp=fib.ccpText + ccpFtn + fib.ccpHdr + fib.ccpAtn + fib.ccpEdn + fib.ccpTxbx through the entry that describes cp = fib.ccpText + fib.ccpFtn + fib.ccpHdr + fib.ccpAtn + fib.ccpEdn + fib.ccpTxbx + fib.ccpHdrTxbx

Character and Paragraph Formatting Properties

Character and paragraph properties in Word documents are stored in a compressed format. The information that is stored on disk is not the actual properties of a particular sequence of text but the difference of the properties of a sequence from some reference property.

The **PAP** is a data structure that holds uncompressed paragraph property information; the **CHP** (pronounced like "chip") is a

structure that holds uncompressed character property information. Each paragraph in a Word document inherits a default set of paragraph and character properties from one of the **paragraph styles** recorded in the style sheet data structure (**STSH**).

A particular **PAP** is converted into its compressed form, the **PAPX**, by first comparing the **pap** for a paragraph with the **pap** stored in the style sheet for the paragraph's style. Any properties in the paragraph's **PAP** that are different from those stored in the style sheet **PAP** are encoded as a list of **sprms (grpprl)**. **sprms** express how the content of the style sheet **PAP** should be transformed to create the properties for the paragraph. A **PAPX** is a variable-length data structure that begins with a count of words that encodes the **PAPX** length. It contains a **istd** (index to style descriptor) which specifies which style entry in the style sheet contains the default paragraph and character properties for the paragraph, paragraph height information, and the list of difference **sprms**. If the only difference between the paragraph's **PAP** and the style's **PAP** were in the justification code field, which is one byte long, one two-byte **sprm**, **sprmPjc**, would be generated to express that difference; thus the total **PAPX** size would be 5 bytes. This is better than 54-1 compression since the total size of a **PAP** is 274 bytes.

To convert a **CHP** for a sequence of characters contained within a single paragraph into its compressed form, the **CHPX**, it's first necessary to know the **paragraph style** that is assigned to the paragraph containing those characters and any character style that may be tagging the character run. The character properties inherited from the paragraph style are moved into a buffer. If the **chp.istd** of the **chp** to be compressed is not **istdNormalChar**, the changes recorded for that character style are applied to buffer. Then the character properties of the character sequence are compared with the character properties generated using the paragraph's style and the run's character style. Any properties in the paragraph's **CHP** that are different from those stored in the generated **CHP** are encoded as a list of **sprms (grpprl)**. The **sprms** express how the content of the **CHP** generated from the paragraph and character styles should be transformed to create the character properties for the text run. A **CHPX** is a variable-length data structure that begins with a count of words that encodes the **CHPX** length followed by the list of difference **sprms**.

If one of the bit fields in the **CHP** to be compressed such as **fBold** is different from the reference **CHP**, you would build a difference **sprm** using **sprmCFBold** in the first byte and the bytes pattern **0x81** in the second byte which signifies that the value of the bit in the **CHP** to be compressed is of opposite value from the value stored in the reference **CHP**. If there was no difference, **sprmCFBold** would not be recorded in the **grpprl** to be generated. If there were difference in a field larger than a single bit such as the **chp.hps**, a **sprmCHps** would be generated to record the value of **chp.hps** in the **chp** to be compressed. If the **chp.hps** were equal in both the **chp** to be compressed and the reference **CHP**, **sprmCHps** would not be recorded in the **grpprl** that is generated. If a sequence of characters has the same character properties and the sequence spans more than one paragraph, it's necessary to examine each paragraph's properties and to generate a different **CHPX** every time there is a change of style.

In Word documents, the fundamental unit of text for which character exception information is kept is the **run of exception text**, a contiguous sequence of characters stored on disk that all have the same exception properties with respect to their underlying style character properties. Each run would have an entry recorded in a **CHPX FKP**. If a user never changed the character properties inherited from the styles used in his document and did a complete save of his document, although each of those styles may have different properties, the entire document stream would be one large **run of exception text** and one **CHPX** would suffice to describe the character properties of the entire document.

The fundamental unit of text for which paragraph properties are recorded is the **paragraph**. Every paragraph has an entry recorded in a **PAPX FKP**.

The **CHPX FKP** and the **PAPX FKP** have similar physical structures. An **FKP** is a 512-byte data structure that is stored in one page of a Word file. At offset 511 is a 1-byte count named **crun**, which is a count of runs of exception text for **CHPX FKP**s and which is a count of paragraphs in **PAPX FKP**s. Beginning at offset 0 of the **FKP** is an array of **crun+1 FC**s, named **rgfc**, which records the beginning and limit **FC**s of **crun** runs of exception text or paragraphs.

For **CHPX FKP**s, immediately following **fkp.rgfc** is a byte array of **crun** word offsets to **CHPX**s from the beginning of the **FKP**. This byte array, named **rgb**, is in 1-to-1 correspondence with the **rgfc**. The **ith rgb** gives the word offset of the exception property that belongs to the run\paragraph whose beginning

For **PAPX FKPS**s, immediately following the **fkp.rgfc** is an array of 13 byte entries called **BX**s. This array called the **rgbx** is in 1-to-1 correspondence with the **rgfc**. The first byte of the **ith BX** entry contains a single byte field which gives the word offset of the **PAPX** that belongs to the paragraph whose beginning in **FC** space is **rgfc[i]** and whose limit is **rgfc[i+1]** in **FC** space. The last 12 bytes of the **ith BX** entry contain a **PHE** structure that stores the current paragraph height of the paragraph whose beginning in **FC** space is **rgfc[i]** and whose limit is **rgfc[i+1]** in **FC** space.

The fact that the offset to property stored in the **rgb** or **rgbx** is a word offset implies that **CHPX**s and **PAPX**s are stored in **FKP**s beginning on word boundaries. Since the values stored in the **rgb/rgbx** allow random access throughout the **FKP**, space within an **FKP** can be conserved by storing the offset of the same physical **CHPX/PAPX** in **rgb/rgbx** entries when several runs or paragraphs in the **FKP** have the same properties. Word uses this optimization.

An **rgb** or **rgbx[]b** value of 0 is used in another optimization. When a **rgb** or **rgbx[]b** value of 0 is stored in an **FKP**, it means that instead of referring to a particular **CHPX/PAPX** in the **FKP** the 0 value is a signal that the reader should construct for itself a commonly encountered predefined set of properties.

For **CHPX FKP**s a 0 **rgb** value means that the properties of the run of text were exactly equal to the character properties inherited from the style of the paragraph it was in. For **PAPX FKP**s, a 0 **rgbx[]b** value means that the paragraph's properties were exactly equal to the paragraph properties of the Normal style ($stc == 0$) and the paragraph contained 1 line of 240 pixels, with a column width of 7980 dxas.

When new entries are added to an **FKP**, there must be unallocated space in the middle of the **FKP** equal to 5 bytes for **CHPX**s (size of an **FC** plus size of one-byte word offset) or 11 bytes for **PAPX**s (size of an **FC** plus the size of a seven byte **BX** entry), plus the size of the new **CHPX** or **PAPX** if the property being added is not already recorded in the **FKP** and is not the property coded with a 0 **rgb/rgbx[]b** value. To add a new property in a **CHPX FKP**, existing **rgb** entries are moved four bytes to the right in the **FKP**. To add a new property in a **PAPX FKP**, existing **rgbx** entries are moved four bytes to the right in the **FKP**. The new **FC** is added at the end of the **rgfc**. The new **CHPX** or **PAPX** is recorded on a 2-byte boundary before the previously recorded properties stored at the end of the block. The word offset of the beginning of the **CHPX** or **PAPX** is stored as the last entry of the relocated **rgb/rgbx[]b**, and finally, the **crun** stored at offset 511 is incremented. In Word '97, **PAPX**s can be generated which are too large to fit in an **FKP**. In such a case, the **grpprl** of the **PAPX** is written to the data stream and a **PAPX** is stored in an **FKP** with that **grpprl** replaced by a **sprmPHugePapx**.

Bin Tables

A bin table (**plcfbte**) partitions the total extent of the Word file that contains text characters into a set of contiguous intervals marked by a **fcFirst** and an **fcLim**. The **fcFirst** for the **nth** interval would be **plcfbte.rgfc[n]** and the **fcLim** for the **nth** interval would be **plcfbte.rgfc[n+1]**. Associated with each interval is a **BTE**. A **BTE** holds a four-byte **PN** (page number) which identifies the **FKP** page in the file which contains the formatting information for that interval. A **CHPX FKP** further partitions an interval into runs of exception text. A **PAPX FKP** in a non-complex, full-saved file, partitions the text within intervals into paragraphs. If a file is in complex format (has been fast-saved), the **PAPX FKP** only records the **FC**s within the text that are preceded by a paragraph mark. Even though a sequence of text may be physically located between two paragraph end marks, it may reside in a paragraph different from the one defined by the following paragraph end mark, because the text may have been moved by the user into a different paragraph. In the logical text stream represented by the document's piece table, the paragraph mark that follows the moved text is stored in a non-adjacent physical location in the file.

Stylesheet

A stylesheet is a collection of styles. In Word, each document has its own stylesheet.

A style is a set of formatting information collected together and given a name. Word 6.0 supports paragraph and character styles, previous versions supported only paragraph styles. Character styles have just character formatting, paragraph styles have both character and paragraph formatting. The style sheet establishes a correspondence between a style code and a style definition.

Note that the storage and behavior of styles has changed radically since Word 2 for Windows, beginning with nFib 63. Some of the differences are:

- Character styles are supported.
- The style code is called an **istd**, rather than an **stc**.
- The **istd** is a short, where the **stc** was a byte.
- The range of the **istd** is 0-4095, where 4095 is the null style. The range of the **stc** was 0-256, with 222 as the null style.
- **PAPX**'s have a short **istd** at the beginning, rather than a byte **stc**.
- **CHPX**'s are a **grpprl**, not a **CHP**.

- Many other changes...

This document describes only the final Word 6.0 version of the stylesheet, not the Word 2.x version.

The styles for a document (both paragraph and character styles) are stored in an array in each document. When new styles are created, they are added to the end of the array. The array can have unused slots. Some slots at the beginning of the array are reserved for specific styles, whether they have been created yet or not. Paragraph and character styles are stored in the same array. Each document has a separate array, so the same style will usually have a different `istd` in two different documents. Thus style matching between documents must be done by name (or by `sti` if the styles are built-in.)

Styles are usually referred to using an `istd`. The `istd` is an index into an array of STD's (STyle Descriptions). A (doc, `istd`) pair uniquely identifies a style because it tells which style in which array.

Parts of a style (for more information, see the STD structure below):

- `sti`: A style identifier. Built-in styles have an `sti` that indicates which built-in style they are. User-defined styles all have `stiUser`.
- `sgc`: The type of style, either paragraph or character.
- `istdBase`: The style that this style is based on.
- `istdNext`: The style that should be applied after this one.
- `stzName`: The name of a style, unique within its stylesheet.
- `UPX`: The difference between this style and the one it is based on.
- `UPE`: The properties of this style (a PAP, CHP, and/or `grppl`).

Every paragraph has a paragraph style. Every character has a character style. The default paragraph style is Normal (`stiNormal`, `istdNormal`). The default character style is Default Paragraph Font (`stiNormalChar`, `istdNormalChar`).

The formatting of a paragraph (the PAP) and a character (the CHP) depend on the paragraph and character styles applied to them, as well as any additional formatting stored in the FKPs. The PAP and CHP are constructed in a layered fashion:

For a PAP:

- An initial PAP is determined by getting the PAP from the paragraph's style.
- Any paragraph formatting stored in the file (the FKP `papx`'s) is then applied to that PAP.

For a CHP:

- An initial CHP is determined by getting the CHP from the paragraph's style.
- Properties from the character's style (the `UPX.chpx.grppl`) are then applied to that CHP.
- Any character formatting stored in the file (the FKP `chpx`'s) is the applied to that CHP.

Note that the resulting PAP and CHP have fields that indicate what style was applied: `PAP.istd`, `CHP.istd`.

Stylesheet File Format

The style sheet (STSH) is stored in the file in two parts, a STSHI and then an array of STDs. The STSHI contains general information about the following stylesheet, including how many styles are in it. After the STSHI, each style is written as an STD. Both the STSHI and each STD are preceded by a `ushort` that indicates their length.

Field	Size	Comment
<code>cbStshi</code>	2 bytes	size of the following STSHI structure
STSHI	(<code>cbStshi</code>)	Stylesheet Information

Then for each style in the stylesheet (stshi.cstd), the following is stored:		
cbStd	2 bytes	size of the following STD structure
STD	(cbStd)	the style description

STSHI:

The STSHI structure has the following format:

```
// STSHI: STyleSHeet Information, as stored in a file
// Note that new fields can be added to the STSHI without invalidating
// the file format, because it is stored preceded by it's length.
// When reading a STSHI from an older version, new fields will be zero.
typedef struct _STSHI
{
    ushort   cstd;                // Count of styles in stylesheet
    ushort   cbSTDBaseInFile;     // Length of STD Base as stored in a file
    BF       fStdStylenamesWritten : 1; // Are built-in stylenames stored?
    BF       : 15;                // Spare flags
    ushort   stiMaxWhenSaved;     // Max sti known when this file was written
    ushort   istdMaxFixedWhenSaved; // How many fixed-index istds are there?
    ushort   nVerBuiltInNamesWhenSaved; // Current version of built-in stylenames
    FTC      rgftcStandardChpStsh[3]; // ftc used by StandardChpStsh for this document
} STSHI;
```

The cb preceding the STSHI in the file is the length of the STSHI as stored in the file. The current definition of the STSHI structure might be longer or shorter than that stored in the file, the stylesheet reader routine needs to take this into account.

stshi.cstd: The number of styles in this stylesheet. There will be stshi.cstd (cbSTD, STD) pairs in the file following the STSHI. Note that styles can be empty, i.e. cbSTD == 0.

stshi.cbSTDBaseInFile: The STD structure (see below) is divided into a fixed-length "base", and a variable length part. The stshi.cbSTDBaseInFile indicates the size in bytes of the fixed-length base of the STD as it was written in this file. If the STD base is grown in a future version, the file format doesn't change, because the stylesheet reader can discard parts it doesn't know about, or use defaults if the file's STD is not as large as it was expecting. (Currently, stshi.cbSTDBaseInFile is 8.)

stshi.fStdStylenamesWritten: Previous versions of Word did not store the style name if the style was a built-in style; Word 6.0 does, for compatibility with future versions. Note that the built-in stylenames may need to be "regenerated" if the file is opened in a different language or if stshi.nVerBuiltInNamesWhenSaved doesn't match the expected value.

stshi.stiMaxWhenSaved: This indicates the last built-in style known to the version of Word that saved this file.

stshi.istdMaxFixedWhenSaved: Each array of styles has some fixed-index styles at the beginning. This indicates the number of fixed-index positions reserved in the stylesheet when it was saved.

stshi.nVerBuiltInNamesWhenSaved: Since built-in stylenames are saved with the document, this provides an way to see if the saved names are the same "version" as the names in the version of Word that is loading the file. If not, the built-in stylenames need to be "regenerated", i.e. the old names need to be replaced with the new.

stshi.rgftcStandardChpStsh: This is the default fonts for this stylesheet. The first is for Ascii characters (0-127), the second is for Far East characters, and the third is the default font for non-Far East, non-Ascii text. See notes on sprmCRgftcX for details.

STD:

The style description is stored in an STD structure as follows:

```
// STD: STyle Definition
// The STD contains the entire definition of a style.
```

```

// It has two parts, a fixed-length base (cbSTDBase bytes long)
// and a variable length remainder holding the name, and the upx and upe
// arrays (a upx and upe for each type stored in the style, std.cupx)
// Note that new fields can be added to the BASE of the STD without
// invalidating the file format, because the STSHI contains the length
// that is stored in the file. When reading STDs from an older version,
// new fields will be zero.
typedef struct _STD
{
    // Base part of STD:
    ushort   sti : 12;           /* invariant style identifier */
    ushort   fScratch : 1;      /* spare field for any temporary use,
                                always reset back to zero! */
    ushort   fInvalHeight : 1; /* PHEs of all text with this style are wrong */
    ushort   fHasUpe : 1;       /* UPEs have been generated */
    ushort   fMassCopy : 1;     /* std has been mass-copied; if unused at
                                save time, style should be deleted */
    ushort   sgc : 4;           /* style type code */
    ushort   istdBase : 12;     /* base style */
    ushort   cupx : 4;          /* # of UPXs (and UPEs) */
    ushort   istdNext : 12;     /* next style */
    ushort   bchUpe;           /* offset to end of upx's, start of upe's */

    ushort   fAutoRedef : 1;    /* auto redefine style when appropriate */
    ushort   fHidden : 1;       /* hidden from UI? */
    ushort : 14;                /* unused bits */

    // Variable length part of STD:
    XCHAR    xstzName[2];       /* sub-names are separated by chDelimStyle */
    /* char   grupx[]; */
    /* the UPEs are not stored on the file; they are a cache of the based-on
       chain */
    /* char   grupe[]; */
} STD;

```

The cb preceding each STD is the length of the data, which includes all of the STD except the grupe array (which is derived after the file is read in, by building each UPE from the base style UPE plus the exceptions in the UPX.) A cb of zero indicates an empty slot in the style array, i.e. no style has that istd. Note that the STD structure may be longer or shorter than the one stored in the file, stshi.cbSTDBaseInFile indicates the length of the base of the STD (up to stzName) as stored in the file. The stylesheet reader routine has to take this into account.

The variable-length part of the STD actually has three variable-length subparts, the xstzName, the grupx, and the grupe. Since this doesn't fit well into a C structure declaration, some processing is needed to figure out where one part stops and the next part begins. An important note is that all variable-length parts and subparts of the STD begin on EVEN-BYTE OFFSETS within the STD, even if the length of the preceding variable-length part was odd.

std.sti: The sti is an identifier which built-in style this is, or stiUser for a user-defined style. An sti is intended to be permanent through versions of Word, although new sti's may be added in new versions. The sti definitions are:

```

// standard sti codes - these are invariant identifiers for built-in styles
// and must remain the same (i.e. don't renumber them, or old files will be
// messed up.)
// NOTE: sti and istd are the same for Normal and level styles
// If you want to define a new built-in style:
// 1) Decide if you really need one--it will exist in all future versions!
// 2) Add a new sti below. You can take the first available slot.
// 3) Change stiMax, and stiPapMax or stiChpMax
// 4) Add entry to _dnsti, and the two ids's in strman.pp
// 5) Add case in GetDefaultUpdForSti
// 6) Change cstiMaxBuiltinDependents if necessary
// If you want to change the definition of a built-in style
// 1) In order to make WinWord 2 documents that use the style look like
// they did in WinWord 2, add a case in GetDefaultUpdForSti to handle
// fOldDef. This definition will be used when converting WinWord 2

```

```

//      stylesheets.
//  2) If you change the name of a built-in style, increment nVerBuiltInNames
#define stiNormal      0      // 0x0000

#define stiLev1        1      // 0x0001
#define stiLev2        2      // 0x0002
#define stiLev3        3      // 0x0003
#define stiLev4        4      // 0x0004
#define stiLev5        5      // 0x0005
#define stiLev6        6      // 0x0006
#define stiLev7        7      // 0x0007
#define stiLev8        8      // 0x0008
#define stiLev9        9      // 0x0009
#define stiLevFirst    stiLev1
#define stiLevLast     stiLev9

#define stiIndex1      10     // 0x000A
#define stiIndex2      11     // 0x000B
#define stiIndex3      12     // 0x000C
#define stiIndex4      13     // 0x000D
#define stiIndex5      14     // 0x000E
#define stiIndex6      15     // 0x000F
#define stiIndex7      16     // 0x0010
#define stiIndex8      17     // 0x0011
#define stiIndex9      18     // 0x0012
#define stiIndexFirst  stiIndex1
#define stiIndexLast   stiIndex9

#define stiToc1        19     // 0x0013
#define stiToc2        20     // 0x0014
#define stiToc3        21     // 0x0015
#define stiToc4        22     // 0x0016
#define stiToc5        23     // 0x0017
#define stiToc6        24     // 0x0018
#define stiToc7        25     // 0x0019
#define stiToc8        26     // 0x001A
#define stiToc9        27     // 0x001B
#define stiTocFirst    stiToc1
#define stiTocLast     stiToc9

#define stiNormIndent  28     // 0x001C
#define stiFtnText     29     // 0x001D
#define stiAtnText     30     // 0x001E
#define stiHeader      31     // 0x001F
#define stiFooter      32     // 0x0020
#define stiIndexHeading 33     // 0x0021
#define stiCaption      34     // 0x0022
#define stiToCaption   35     // 0x0023
#define stiEnvAddr     36     // 0x0024
#define stiEnvRet      37     // 0x0025
#define stiFtnRef      38     // 0x0026 char style
#define stiAtnRef      39     // 0x0027 char style
#define stiLnn         40     // 0x0028 char style
#define stiPgn         41     // 0x0029 char style
#define stiEdnRef      42     // 0x002A char style
#define stiEdnText     43     // 0x002B
#define stiToa         44     // 0x002C
#define stiMacro       45     // 0x002D
#define stiToaHeading  46     // 0x002E
#define stiList        47     // 0x002F
#define stiListBullet  48     // 0x0030
#define stiListNumber  49     // 0x0031
#define stiList2       50     // 0x0032
#define stiList3       51     // 0x0033
#define stiList4       52     // 0x0034
#define stiList5       53     // 0x0035

```

```

#define stiListBullet2 54 // 0x0036
#define stiListBullet3 55 // 0x0037
#define stiListBullet4 56 // 0x0038
#define stiListBullet5 57 // 0x0039
#define stiListNumber2 58 // 0x003A
#define stiListNumber3 59 // 0x003B
#define stiListNumber4 60 // 0x003C
#define stiListNumber5 61 // 0x003D
#define stiTitle 62 // 0x003E
#define stiClosing 63 // 0x003F
#define stiSignature 64 // 0x0040
#define stiNormalChar 65 // 0x0041 char style
#define stiBodyText 66 // 0x0042
#define stiBodyText2 67 // 0x0043
#define stiListCont 68 // 0x0044
#define stiListCont2 69 // 0x0045
#define stiListCont3 70 // 0x0046
#define stiListCont4 71 // 0x0047
#define stiListCont5 72 // 0x0048
#define stiMsgHeader 73 // 0x0049
#define stiSubtitle 74 // 0x004A
#define stiSalutation 75 // 0x004B
#define stiDate 76 // 0x004C
#define stiBodyText1I 77 // 0x004D
#define stiBodyText1I2 78 // 0x004E
#define stiNoteHeading 79 // 0x004F
#define stiBodyText2 80 // 0x0050
#define stiBodyText3 81 // 0x0051
#define stiBodyTextInd2 82 // 0x0052
#define stiBodyTextInd3 83 // 0x0053
#define stiBlockQuote 84 // 0x0054
#define stiHyperlink 85 // 0x0055 char style
#define stiHyperlinkFollowed 86 // 0x0056 char style
#define stiStrong 87 // 0x0057 char style
#define stiEmphasis 88 // 0x0058 char style
#define stiNavPane 89 // 0x0059 char style
#define stiPlainText 90 // 0x005A
#define stiMax 91 // number of defined sti's

#define stiUser 0x0ffe // user styles are distinguished by name
#define stiNil 0x0fff // max for 12 bits

```

See below for the names of these styles.

std.stc: The type of each style is indicated by std.sgc. The two types currently in use are:

sgcPara	1	// A paragraph style
sgcChp	2	// A character style

More style types may exist in the future, so styles of an unknown type should be discarded.

std.istdBase: The style that this style is based on. A style is always based on another style or the null style (istdNil). Following a "chain" of based-on styles will always end at the null style, because a based-on chain cannot have a loop in it. A style can have up to 11 "ancestors" in its based-on chain, including the null style. A style's definition is built up from the style that it is based on. See std.cupx, std.grupx, std.grupe.

std.istdNext: The style that should be applied after this one. For a paragraph style, this is the style that is applied when Enter is pressed at the end of a paragraph. For a character style, the next style is essentially ignored, but should be the same as the current style.

std.xstzName: The name of the style, including aliases. The name is stored as an xstz (preceded by a length byte, followed by a

null-terminator.) A style name can contain multiple "aliases", separated by commas. Aliases are alternate names for the same style (e.g. a style named "a,b,c" has three aliases, and can be referred to by "a", "b", or "c", or any combination.) WinWord 2.x did not have aliases, but MacWord 5.x did. If a style is a built-in style, the built-in stylename is always stored first.

All names (and aliases) must be unique within a stylesheet (e.g. styles "a,b" and "b,c" should not exist in the same stylesheet, as "b" matches multiple stylenames.)

A stylename (including all its aliases and comma separators) can be up to 253 characters long. So the xstz format of that name can be up to 255 characters. Stylenames are case sensitive.

The built-in stylenames (corresponding to each sti above) are defined for each language version of Word. For the USA, the names are:

```
// These are the names of the built-in styles as we want to present them
// to the user.
Normal
Heading 1
Heading 2
Heading 3
Heading 4
Heading 5
Heading 6
Heading 7
Heading 8
Heading 9
Index 1
Index 2
Index 3
Index 4
Index 5
Index 6
Index 7
Index 8
Index 9
TOC 1
TOC 2
TOC 3
TOC 4
TOC 5
TOC 6
TOC 7
TOC 8
TOC 9
Normal Indent
Footnote Text
Annotation Text
Header
Footer
Index Heading
Caption
Table of Figures
Envelope Address
Envelope Return
Footnote Reference
Annotation Reference
Line Number
Page Number
Endnote Reference
Endnote Text
Table of Authorities
Macro Text
TOA Heading
List
List 2
```

List 3
 List 4
 List 5
 List Bullet
 List Bullet 2
 List Bullet 3
 List Bullet 4
 List Bullet 5
 List Number
 List Number 2
 List Number 3
 List Number 4
 List Number 5
 Title
 Closing
 Signature
 Default Paragraph Font
 Body Text
 Body Text Indent
 List Continue
 List Continue 2
 List Continue 3
 List Continue 4
 List Continue 5
 Message Header
 Subtitle
 Salutation
 Date
 Body Text First Indent
 Body Text First Indent 2
 Note Heading
 Body Text 2
 Body Text 3
 Body Text Indent 2
 Body Text Indent 3
 Block Text
 Hyperlink
 Followed Hyperlink
 Strong
 Emphasis
 Document Map
 Plain Text

std.cupx: This is the number of UPXs in the std.grupx array. See below.

std.grupx: This is an array of variable-length UPXs, with std.cupx UPXs in the array. This array begins after the variable-length xstzName field, at the next even-byte offset within the STD. A UPX (Universal Property eXception) describes the difference in formatting of this style as compared to its based-on style. The UPX structure looks like this:

```

typedef union _UPX
{
    struct
    {
        uchar grppr1[cbMaxGrppr1StyleChpx];
    } chpx;
    struct
    {
        ushort istd;
        uchar grppr1[cbMaxGrppr1StylePapx];
    } papx;
    uchar rgb[1];
} UPX;

```

Each UPX stored in a file is not a complete UPX, rather it is a UPX with all trailing zero bytes lopped off, and preceded by a ushort length field. So it is stored like:

Field	Size	Comment
cbUPX	2 bytes	size of the following UPX structure
UPX	(cbUPX)	Nonzero prefix of a UPX structure

Each UPX begins on an even-byte offset within the STD, even if the length of the previous UPX (cbUPX) was odd.

The meaning of each UPX depends on the style type (std.sgc). For a paragraph style, std.cupx is 2. The first UPX is a paragraph UPX (UPX.papx) and the second UPX is a character UPX (UPX.chpx). For a character style, std.cupx is 1, and that UPX is a character UPX (UPX.chpx). Note that new UPXs may be added in the future, so std.cupx might be larger than expected. Any UPXs past those expected should be discarded.

The grpprl within each UPX contains the differences of this property type for this style from the UPE of that property type for the based on style. For example, if two paragraph styles, A and B, were identical except that B was bold where A was not, and B was based on A, B would have two UPXs, where the paragraph UPX would have an empty grpprl, and the character UPX would have a bold sprm in the grpprl. Thus B looks just like A (since B is based on A), with the exception that B is bold.

std.grupe: This is an array (group) of variable-length UPEs. These are not stored in the file! Rather, they are constructed using the std.istdBase and std.grupx fields. A UPE (Universal Property Expansion) describes the "end-result" of the property formatting, i.e. what the style looks like. The UPE structure is the non-zero prefix of a UPD structure. The UPD structure looks like this:

```
typedef union _UPD
{
    PAP pap;
    CHP chp;
    struct
    {
        ushort istd;
        uchar cbGrpprl;
        uchar grpprl[cbMaxGrpprlStyleChpx];
    } chpx;
} UPD;
```

The std.grupe and std.grupx arrays are similar: there is one UPE for each UPX, and internally they are stored similarly (a length ushort followed by a non-zero prefix), though remember that the UPEs are not stored in the file. The meaning of each UPE depends on the style type (std.sgc). For a paragraph style, the first UPE is a PAP (UPE.pap). The second UPE is a CHP (UPE.chp). For a character style, the first UPE is a CHPX (UPE.chpx).

The UPEs for a style are constructed by taking the UPEs from the based-on style, and applying the UPXs to them. Obviously, if the UPEs for the based-on style haven't yet been constructed, that style's UPE needs to be constructed first. Eventually by following the based-on chain, a style will be based on the null style (istdNil). The UPEs for the null style are predefined:

- The UPE.pap for the null style is all zeros, except fWidowControl which is 1, dyaLine which is 240, and fMultLinespace which is 1.
- The UPE.chp for the null style is all zeros, except istd which is 10 (istdNormalChar), hps which is 20, lid which is 0x0400, and ftc which is set to the STSHI.ftcStandardChpStsh.
- The UPE.chpx for the null style has an istd of zero, a cbGrpprl of zero (and an empty grpprl).

So, for a paragraph style, the first UPE is a UPE.pap. It can be constructed by starting with the first UPE from the based-on style (std.istdBase), and then applying the first UPX (UPX.papx) in std.grupx to that UPE. To apply a UPX.papx to a UPE.pap, set UPE.pap.istd equal to UPX.papx.istd, and then apply the UPX.papx.grpprl to UPE.pap. Similarly, the second UPE is a UPE.chp. It can be constructed by starting with the second UPE from the based-on style, and then applying the second UPX (UPX.chpx) in std.grupx to that UPE. To apply a UPX.chpx to a UPE.chp, apply the UPX.chpx.grpprl to UPE.chp. Note that a UPE.chp for a paragraph style should always have UPE.chp.istd == istdNormalChar.

For a character style, the first (and only) UPE (a UPE.chpx) can be constructed by starting with the first UPE from the based-on

style (std.istdBase), and then applying the first UPX (UPX.chpx) in std.grupx to that UPE. To apply a UPX.chpx to a UPE.chpx, take the grpprl in UPE.chpx.grpprl (which has a length of UPE.chpx.cbGrpprl) and merge the grpprl in UPX.chpx.grpprl into it. Merging grpprls is a tricky business, but for character styles it is easy because no prls in character style grpprls should interact with each other. Each prl from the source (the UPX.chpx.grpprl) should be inserted into the destination (the UPE.chpx.grpprl) so that the sprm of each prl is in increasing order, and any prls that have the same sprm are replaced by the prl in the source. UPE.chpx.cbGrpprl is then set to the length of resulting grpprl, and UPE.chpx.istd is set to the style's istd.

List Tables

Word 97 stores its paragraph numbering information very differently from Word 6.0. In Word 6.0, all information for a paragraph was stored in that paragraph's pap.anld. In Word 97, the pap only contains two values: a short ilfo and a byte ilvl, which indicate which list the paragraph belongs to and which level of that list it should be, respectively. The ilfo is actually an index into one of the the document's list tables: the pllfo, and the paragraph gets most of its information about appearance from the list tables.

There are three list tables in a word document: the rglst, the hpllfo, and the hsttbListNames. They will be described below in greater detail, and the precise formats of several of these structures (the LSTF, LVLf, LFO, and LFOLVL) are listed in the appendix.

LST records and the rglst

The LST structure is where most of the list appearance data is stored. An LST consists of two main parts: (1) an LSTF, which is stored on disk and contains formatting properties which apply to the entire list, such as whether the list is simple or multilevel, the list's unique list index and template code, the istd's (see Stylesheet above) of the styles (if any) that each level in the list is linked to, and a number of Word 6 compatibility option; (2) an array of LVL structures, which describe the appearance of each individual level in the LST.

A LVL structure contains two parts to it: (1) an LVLf, which stores all static data such as the start-at value for the list level, the numbering type (arabic or roman), the alignment (left, right or centered) of the number, and several Word 6.0 compatibility options; and (2) a set of pointers to variable length data: (a) a grpprlChpx, which gives character formatting to the paragraph number text itself, (b) a grpprlPapx, which gives paragraph formatting to the paragraph containing the number, such as indenting and tab information, and (c) the number text itself.

Word writes out the rglst as the **plcflst** by writing out, first, a short integer containing the number of LST structures to be written. It then enumerates through the rglst, writing out each LSTF structure. It then enumerates through the rglst again, deciding, for each LST, whether it has one level (LSTF.fSimpleList) or nine levels (!LSTF.fSimpleList). It then writes the appropriate number of LVL structures as described below.

When Word writes out an LVL structure, it first writes out the LVLf, followed by the grpprlPapx (of LVLf.cbGrpprlPapx bytes in length), followed by the grpprlChpx (of length LVLf.cbGrpprlChpx), and an XCHAR string with the number text, preceded by an XCHAR containing the string's length.

List Names and the sttbListNames

The string table containing the List Names is by far the least significant of the three list tables. Most lists do not have names, and the names are only useful to users of the macro language. If this list has a name, however, it will be in this table: the table is a parallel array with the **rglst** above, and will contain an empty string for any list which does not have a list name.

LFO records and the pllfo

The LFO structure serves primarily as a level of indirection between the paragraph and the LST, but also can be used to override certain features of the list formats (LFO stands for List Format Override). An LFO consists of two main parts: (1) the List ID of the list (LST record) to which this LFO belongs, and an array of overrides to the formatting in that LST. For the vast majority of LFOs, there are no overrides, but if there are any, they reside in an array of LFOLVL structures -- one LFOLVL per level of the

LST to be overridden. An LFOLVL contains a set of flags to indicate whether just the start-at value of the LST is overridden, or whether just the formatting is overridden, or both, as well as either a start-at value or a pointer to a LVL record, depending upon the values of the flags. Note that if the LFOLVL says the start-at value should be overridden, what that means is that the FIRST paragraph in the document with this LFO should have a number equal exactly to that start-at value, but any subsequent paragraphs should just follow the previous paragraph in the sequence. Also, if LFOLVL.fFormatting and LFOLVL.fStartAt are *both* true (rare) then LFOLVL.iStartAt should be ignored in favor of the iStartAt value from the corresponding LVL.

Word writes out the pllfo first by writing out a PL of LFO structures. It then enumerates through each LFO to figure out how many LFOLVLs each one has (LFO.clfolvl), and writes out, in order, each LFOLVL structure followed by its corresponding LVL structure (if LFOLVL.fFormatting is set).

Paragraph List Formatting

Given a paragraph and its corresponding PAP, the following process must be followed to find out the paragraph's list information:

- Using the pap.ilfo, look up the LFO record in the pllfo with that (1-based) index.
- Using the LFO, and the pap.ilvl, check to see if there are any overrides for this particular level. If so, and if the override pertains to both formatting and start-at value, use the LVL record from the correct LFOLVL in the LFO, and skip to step 5.
- If the override does not pertain to either formatting or start-at value, we must look up the LST for this list. Using the LFO's List ID, search the rglst for the LST with that List ID.
- Now, take from this LST any information (formatting or start-at value) we still need after consulting the LFO.
- Once we've got the correct LVL record, apply the lvl.grprlPapx to the PAP. It may adjust the indents and tab settings for the paragraph.
- Use the other information in the LVL, such as the start at, number text, and grprlChpx, to determine the appearance of the actual paragraph number text.

SPRM Definitions

A **sprm** is an instruction to modify one or more properties within one of the property defining data structures (**CHP**, **PAP**, **TAP**, **SEP**, or **PIC**). A **sprm** is a two-byte opcode at offset 0 which identifies the operation to be performed. If necessary information for the operation can always be expressed with a fixed length parameter, the fixed length parameter is recorded immediately after the opcode beginning at offset 2. The length of a fixed length sprm is always 2 plus the size of the sprm's parameter. If the parameter for the sprm is variable length, the count of bytes of the following parameter is stored in the byte at offset 2, followed by the parameter at offset 3.

Three sprms -- sprmPChgTabs, sprmTDefTable, and sprmTDefTable10 -- can be longer than 256 bytes. The method for calculating the length of sprmPChgTabs is recorded below with the description of the sprm. For sprmTDefTable and sprmTDefTable10, the length of the parameter plus 1 is recorded in the two bytes beginning at offset 2.

For all other variable length sprms, the total length of the sprm is the count recorded at offset 2 plus three (2 for the sprm + 1 for the count byte). The parameter immediately follows the count.

The sprm value encodes information on the size of the operand, the type of sprm (PAP, CHP, etc), and whether the sprm requires special handling (in cases where a property value isn't simply replaced).

Sprm bits (0 = low)	Value	Details
0-8	ispmid	unique identifier within sgc group

9	fSpec	sprm requires special handling
10-12	sgc	sprm group; type of sprm (PAP, CHP, etc)
13-15	spra	size of sprm argument (see following table for values)

sgc value	type of sprm
1	PAP
2	CHP
3	PIC
4	SEP
5	TAP

spra value	operand size
0	1 byte (operand affects 1 bit)
1	1 byte
2	2 bytes
3	4 bytes
4	2 bytes
5	2 bytes
6	variable length -- following byte is size of operand
7	3 bytes

When parsing a grpprl, you can use the sprm's sprm value to determine how many bytes are used by that sprm; it also enables you to skip over sprms you don't handle.

Unless otherwise noted, when a sprm is applied to a property the sprm's parameter changes the old value of the property in question to the value stored in the sprm parameter.

Name	sprm	Property Modified	Parameter	Parameter size
sprmPIstd	0x4600	pap.istd	istd (style code)	short
sprmPIstdPermute	0xC601	pap.istd	permutation vector (see below)	variable length
sprmPIncLvl	0x2602	pap.istd, pap.lvl	difference between istd of base PAP and istd of PAP to be produced (see below)	byte
sprmPJc	0x2403	pap.jc	jc (justification)	byte
sprmPFSideBySide	0x2404	pap.fSideBySide	0 or 1	byte
sprmPFKeep	0x2405	pap.fKeep	0 or 1	byte
sprmPFKeepFollow	0x2406	pap.fKeepFollow	0 or 1	byte
sprmPFPageBreakBefore	0x2407	pap.fPageBreakBefore	0 or 1	byte
sprmPBrcl	0x2408	pap.brcl	brcl	byte

sprmPBrcp	0x2409	pap.brcp	brcp	byte
sprmPIlvl	0x260A	pap.ilvl	ilvl	byte
sprmPIlfo	0x460B	pap.ilfo	ilfo (list index)	short
sprmPFNoLineNumb	0x240C	pap.fNoLnn	0 or 1	byte
sprmPChgTabsPapx	0xC60D	pap.itbdMac, pap.rgdxaTab, pap.rgtbd	complex - see below	variable length
sprmPDxaRight	0x840E	pap.dxaRight	dxa	word
sprmPDxaLeft	0x840F	pap.dxaLeft	dxa	word
sprmPNest	0x4610	pap.dxaLeft	dxa-see below	word
sprmPDxaLeft1	0x8411	pap.dxaLeft1	dxa	word
sprmPDyaLine	0x6412	pap.lspd	an LSPD, a long word structure consisting of a short of dyaLine followed by a short of fMultLinespace - see below	long
sprmPDyaBefore	0xA413	pap.dyaBefore	dya	word
sprmPDyaAfter	0xA414	pap.dyaAfter	dya	word
sprmPChgTabs	0xC615	pap.itbdMac, pap.rgdxaTab, pap.rgtbd	complex - see below	variable length
sprmPFInTable	0x2416	pap.fInTable	0 or 1	byte
sprmPFTtp	0x2417	pap.fTtp	0 or 1	byte
sprmPDxaAbs	0x8418	pap.dxaAbs	dxa	word
sprmPDyaAbs	0x8419	pap.dyaAbs	dya	word
sprmPDxaWidth	0x841A	pap.dxaWidth	dxa	word
sprmPPc	0x261B	pap.pcHorz, pap.pcVert	complex - see below	byte
sprmPBrcTop10	0x461C	pap.brcTop	BRC10	word
sprmPBrcLeft10	0x461D	pap.brcLeft	BRC10	word
sprmPBrcBottom10	0x461E	pap.brcBottom	BRC10	word
sprmPBrcRight10	0x461F	pap.brcRight	BRC10	word
sprmPBrcBetween10	0x4620	pap.brcBetween	BRC10	word
sprmPBrcBar10	0x4621	pap.brcBar	BRC10	word
sprmPDxaFromText10	0x4622	pap.dxaFromText	dxa	word
sprmPWwr	0x2423	pap.wr	wr (see description of PAP for definition)	byte
sprmPBrcTop	0x6424	pap.brcTop	BRC	long
sprmPBrcLeft	0x6425	pap.brcLeft	BRC	long
sprmPBrcBottom	0x6426	pap.brcBottom	BRC	long
sprmPBrcRight	0x6427	pap.brcRight	BRC	long
sprmPBrcBetween	0x6428	pap.brcBetween	BRC	long

sprmPBrcBar	0x6629	pap.brcBar	BRC	long
sprmPFNoAutoHyph	0x242A	pap.fNoAutoHyph	0 or 1	byte
sprmPWHeightAbs	0x442B	pap.wHeightAbs	w	word
sprmPDcs	0x442C	pap.dcs	DCS	short
sprmPShd	0x442D	pap.shd	SHD	word
sprmPDyaFromText	0x842E	pap.dyaFromText	dya	word
sprmPDxaFromText	0x842F	pap.dxaFromText	dxa	word
sprmPFLocked	0x2430	pap.fLocked	0 or 1	byte
sprmPFWidowControl	0x2431	pap.fWidowControl	0 or 1	byte
sprmPRuler	0xC632			variable length
sprmPFKinsoku	0x2433	pap.fKinsoku	0 or 1	byte
sprmPFWordWrap	0x2434	pap.fWordWrap	0 or 1	byte
sprmPFOverflowPunct	0x2435	pap.fOverflowPunct	0 or 1	byte
sprmPFTopLinePunct	0x2436	pap.fTopLinePunct	0 or 1	byte
sprmPFAutoSpaceDE	0x2437	pap.fAutoSpaceDE	0 or 1	byte
sprmPFAutoSpaceDN	0x2438	pap.fAutoSpaceDN	0 or 1	byte
sprmPWAlignFont	0x4439	pap.wAlignFont	iFa (see description of PAP for definition)	word
sprmPFframeTextFlow	0x443A	pap.fVertical pap.fBackward pap.fRotateFont	complex (see description of PAP for definition)	word
sprmPISnapBaseLine	0x243B	obsolete: not applicable in Word97 and later versions		byte
sprmPANld	0xC63E	pap.anld		variable length
sprmPFPropRMark	0xC63F	pap.fPropRMark	complex (see below)	variable length
sprmPOutLvl	0x2640	pap.lvl	has no effect if pap.istd is < 1 or is > 9	byte
sprmPFBiDi	0x2441			byte
sprmPFNumRMIns	0x2443	pap.fNumRMIns	1 or 0	bit
sprmPCrLf	0x2444			byte
sprmPNumRM	0xC645	pap.numrm		variable length
sprmPHugePapx	0x6645	see below	fc in the data stream to locate the huge grppl (see below)	long
sprmPFUsePgsuSettings	0x2447	pap.fUsePgsuSettings	1 or 0	byte
sprmPFAdjustRight	0x2448	pap.fAdjustRight	1 or 0	byte

sprmCFRMarkDel sprmCFRMark	0x0800 0x0801	chp.fRMarkDel chp.fRMark	1 or 0 1 or 0	bit bit
sprmCFFldVanish	0x0802	chp.fFldVanish	1 or 0	bit
sprmCPicLocation	0x6A03	chp.fcPic and chp.fSpec	see below	variable length, length recorded is always 4
sprmCIbstRMark	0x4804	chp.ibstRMark	index into sttbRMark	short
sprmCDttmRMark	0x6805	chp.dttmRMark	DTTM	long
sprmCFData	0x0806	chp.fData	1 or 0	bit
sprmCIidsIRMark	0x4807	chp.idsIRMReason	an index to a table of strings defined in Word 6.0 executables	short
sprmCChs	0xEA08	chp.fChsDiff and chp.chse	see below	3 bytes
sprmCSymbol	0x6A09	chp.fSpec, chp.xchSym and chp.ftcSym	see below	variable length, length recorded is always 4
sprmCFOle2	0x080A	chp.fOle2	1 or 0	bit
sprmCIIdCharType	0x480B	obsolete: not applicable in Word97 and later versions		
sprmCHighlight	0x2A0C	chp.fHighlight, chp.icoHighlight	ico (fHighlight is set to 1 iff ico is not 0)	byte
sprmCObjLocation	0x680E	chp.fcObj	FC	long
sprmCFFtcAsciSymb	0x2A10			
sprmCIstd	0x4A30	chp.istd	istd, see stylesheet definition	short
sprmCIstdPermute	0xCA31	chp.istd	permutation vector (see below)	variable length
sprmCDefault	0x2A32	whole CHP (see below)	none	variable length
sprmCPlain	0x2A33	whole CHP (see below)	none	0
sprmCKcd	0x2A34			
sprmCFBold	0x0835	chp.fBold	0,1, 128, or 129 (see below)	byte
sprmCFItalic	0x0836	chp.fItalic	0,1, 128, or 129 (see below)	byte
sprmCFStrike	0x0837	chp.fStrike	0,1, 128, or 129 (see below)	byte
sprmCFOutline	0x0838	chp.fOutline	0,1, 128, or 129 (see below)	byte
sprmCFShadow	0x0839	chp.fShadow	0,1, 128, or 129 (see below)	byte
sprmCFSmallCaps	0x083A	chp.fSmallCaps	0,1, 128, or 129 (see below)	byte
sprmCFCaps	0x083B	chp.fCaps	0,1, 128, or 129 (see below)	byte
sprmCFVanish	0x083C	chp.fVanish	0,1, 128, or 129 (see below)	byte

sprmCFtcDefault	0x4A3D		ftc, only used internally, never stored in file	word
sprmCKul	0x2A3E	chp.kul	kul	byte
sprmCSizePos	0xEA3F	chp.hps, chp.hpsPos	(see below)	3 bytes
sprmCDxaSpace	0x8840	chp.dxaSpace	dxa	word
sprmCLid	0x4A41		only used internally never stored	word
sprmCIco	0x2A42	chp.ico	ico	byte
sprmCHps	0x4A43	chp.hps	hps	byte
sprmCHpsInc	0x2A44	chp.hps	(see below)	byte
sprmCHpsPos	0x4845	chp.hpsPos	hps	byte
sprmCHpsPosAdj	0x2A46	chp.hpsPos	hps (see below)	byte
sprmCMajority	0xCA47	chp.fBold, chp.fItalic, chp.fSmallCaps, chp.fVanish, chp.fStrike, chp.fCaps, chp.rgftc, chp.hps, chp.hpsPos, chp.kul, chp.dxaSpace, chp.ico, chp.rglid	complex (see below)	variable length, length byte plus size of following grppl
sprmCIss	0x2A48	chp.iss	iss	byte
sprmCHpsNew50	0xCA49	chp.hps	hps	variable width, length always recorded as 2
sprmCHpsInc1	0xCA4A	chp.hps	complex (see below)	variable width, length always recorded as 2
sprmCHpsKern	0x484B	chp.hpsKern	hps	short
sprmCMajority50	0xCA4C	chp.fBold, chp.fItalic, chp.fSmallCaps, chp.fVanish, chp.fStrike, chp.fCaps, chp.ftc, chp.hps, chp.hpsPos, chp.kul, chp.dxaSpace, chp.ico,	complex (see below)	variable length
sprmCHpsMul	0x4A4D	chp.hps	percentage to grow hps	short
sprmCYsri	0x484E	chp.ysri	ysri	short
sprmCRgFtc0	0x4A4F	chp.rgftc[0]	ftc for ASCII text (see below)	short
sprmCRgFtc1	0x4A50	chp.rgftc[1]	ftc for Far East text (see below)	short
sprmCRgFtc2	0x4A51	chp.rgftc[2]	ftc for non-Far East text (see below)	short
sprmCCharScale	0x4852			
sprmCFDStrike	0x2A53	chp.fDStrike		byte
sprmCFImprint	0x0854	chp.fImprint	1 or 0	bit
sprmCFSpec	0x0855	chp.fSpec	1 or 0	bit

sprmCFObj sprmCPropRMark	0x0856 0xCA57	chp.fObj chp.fPropRMark, chp.ibstPropRMark, chp.dttmPropRMark	1 or 0 Complex (see below)	bit variable length always recorded as 7 bytes
sprmCFEmboss	0x0858	chp.fEmboss	1 or 0	bit
sprmCSfxText	0x2859	chp.sfxText	text animation	byte
sprmCFBiDi	0x085A			
sprmCFDiacColor	0x085B			
sprmCFBoldBi	0x085C			
sprmCFItalicBi	0x085D			
sprmCFtcBi	0x4A5E			
sprmCLidBi	0x485F			
sprmCIcoBi	0x4A60			
sprmCHpsBi	0x4A61			
sprmCDispFldRMark	0xCA62	chp.fDispFldRMark, chp.ibstDispFldRMark, chp.dttmDispFldRMark	Complex (see below)	variable length always recorded as 39 bytes
sprmCIbstRMarkDel	0x4863	chp.ibstRMarkDel	index into stbRMark	short
sprmCDttmRMarkDel	0x6864	chp.dttmRMarkDel	DTTM	long
sprmCBrc	0x6865	chp.brc	BRC	long
sprmCShd	0x4866	chp.shd	SHD	short
sprmCIdslRMarkDel	0x4867	chp.idslRMReasonDel	an index to a table of strings defined in Word 6.0 executables	short
sprmCFUsePgsuSettings	0x0868	chp.fUsePgsuSettings	1 or 0	bit
sprmCCpg	0x486B			word
sprmCRgLid0	0x486D	chp.rglid[0]	LID: for non-Far East text	word
sprmCRgLid1	0x486E	chp.rglid[1]	LID: for Far East text	word
sprmCI dctHint	0x286F	chp.idctHint	IDCT: (see below)	byte
sprmPicBrcL	0x2E00	pic.brcl	brcl (see PIC structure definition)	byte
sprmPicScale	0xCE01	pic.mx, pic.my, pic.dxaCropLeft, pic.dyaCropTop pic.dxaCropRight, pic.dyaCropBottom	Complex (see below)	length byte plus 12 bytes
sprmPicBrcTop	0x6C02	pic.brcTop	BRC	long
sprmPicBrcLeft	0x6C03	pic.brcLeft	BRC	long
sprmPicBrcBottom	0x6C04	pic.brcBottom	BRC	long
sprmPicBrcRight	0x6C05	pic.brcRight	BRC	long

sprmSensPgn	0x3000	sep.cnsPgn	cns	byte
sprmSiHeadingPgn	0x3001	sep.iHeadingPgn	heading number level	byte
sprmSOlstAnm	0xD202	sep.olstAnm	OLST	variable length
sprmSDxaColWidth	0xF203	sep.rgdxaColWidthSpacing	complex (see below)	3 bytes
sprmSDxaColSpacing	0xF204	sep.rgdxaColWidthSpacing	complex (see below)	3 bytes
sprmSFEvenlySpaced	0x3005	sep.fEvenlySpaced	1 or 0	byte
sprmSFProtected	0x3006	sep.fUnlocked	1 or 0	byte
sprmSDmBinFirst	0x5007	sep.dmBinFirst		word
sprmSDmBinOther	0x5008	sep.dmBinOther		word
sprmSBkc	0x3009	sep.bkc	bkc	byte
sprmSFTitlePage	0x300A	sep.fTitlePage	0 or 1	byte
sprmSCcolumns	0x500B	sep.ccolM1	# of cols - 1	word
sprmSDxaColumns	0x900C	sep.dxaColumns	dxa	word
sprmSFAutoPgn	0x300D	sep.fAutoPgn	obsolete	byte
sprmSNfcPgn	0x300E	sep.nfcPgn	nfc	byte
sprmSDyaPgn	0xB00F	sep.dyaPgn	dya	short
sprmSDxaPgn	0xB010	sep.dxaPgn	dya	short
sprmSFPgnRestart	0x3011	sep.fPgnRestart	0 or 1	byte
sprmSFEndnote	0x3012	sep.fEndnote	0 or 1	byte
sprmSLnc	0x3013	sep.lnc	lnc	byte
sprmSGprfIhdt	0x3014	sep.grpfIhdt	grpfihdt (see Headers and Footers topic)	byte
sprmSNLnnMod	0x5015	sep.nLnnMod	non-neg int.	word
sprmSDxaLnn	0x9016	sep.dxaLnn	dxa	word
sprmSDyaHdrTop	0xB017	sep.dyaHdrTop	dya	word
sprmSDyaHdrBottom	0xB018	sep.dyaHdrBottom	dya	word
sprmSLBetween	0x3019	sep.fLBetween	0 or 1	byte
sprmSVjc	0x301A	sep.vjc	vjc	byte
sprmSLnnMin	0x501B	sep.lnnMin	lnn	word
sprmSPgnStart	0x501C	sep.pgnStart	pgn	word
sprmSBOrientation	0x301D	sep.dmOrientPage	dm	byte
sprmSBCustomize	0x301E			
sprmSXaPage	0xB01F	sep.xaPage	xa	word
sprmSYaPage	0xB020	sep.yaPage	ya	word
sprmSDxaLeft	0xB021	sep.dxaLeft	dxa	word

sprmSDxaRight	0xB022	sep.dxaRight	dxa	word
sprmSDyaTop	0x9023	sep.dyaTop	dya	word
sprmSDyaBottom	0x9024	sep.dyaBottom	dya	word
sprmSDzaGutter	0xB025	sep.dzaGutter	dza	word
sprmSDmPaperReq	0x5026	sep.dmPaperReq	dm	word
sprmSPropRMark	0xD227	sep.fPropRMark, sep.ibstPropRMark, sep.dttmPropRMark	complex (see below)	variable length always recorded as 7 bytes
sprmSFBiDi	0x3228			
sprmSFFacingCol	0x3229			
sprmSFRTLGutter	0x322A			
sprmSBrcTop	0x702B	sep.brcTop	BRC	long
sprmSBrcLeft	0x702C	sep.brcLeft	BRC	long
sprmSBrcBottom	0x702D	sep.brcBottom	BRC	long
sprmSBrcRight	0x702E	sep.brcRight	BRC	long
sprmSPgbProp	0x522F	sep.pgbProp		word
sprmSDxtCharSpace	0x7030	sep.dxtCharSpace	dxt	long
sprmSDyaLinePitch	0x9031	sep.dyaLinePitch	dya	long
sprmSClm	0x5032			
sprmSTextFlow	0x5033	sep.wTextFlow	complex (see below)	short
sprmTJc	0x5400	tap.jc	jc	word (low order byte is significant)
sprmTDxaLeft	0x9601	tap.rgdxaCenter (see below)	dxa	word
sprmTDxaGapHalf	0x9602	tap.dxaGapHalf, tap.rgdxaCenter (see below)	dxa	word
sprmTFCantSplit	0x3403	tap.fCantSplit	1 or 0	byte
sprmTTableHeader	0x3404	tap.fTableHeader	1 or 0	byte
sprmTTableBorders	0xD605	tap.rgbrcTable	complex(see below)	24 bytes
sprmTDefTable10	0xD606	tap.rgdxaCenter, tap.rgtc	complex (see below)	variable length
sprmTDyaRowHeight	0x9407	tap.dyaRowHeight	dya	word
sprmTDefTable	0xD608	tap.rgtc	complex (see below)	
sprmTDefTableShd	0xD609	tap.rgshd	complex (see below)	
sprmTTlp	0x740A	tap.tlp	TLP	4 bytes
sprmTFBiDi	0x560B			
sprmTHTMLProps	0x740C			

sprmTSetBrc	0xD620	tap.rgtc[].rgbrc	complex (see below)	5 bytes
sprmTInsert	0x7621	tap.rgdxaCenter, tap.rgtc	complex (see below)	4 bytes
sprmTDelete	0x5622	tap.rgdxaCenter, tap.rgtc	complex (see below)	word
sprmTDxaCol	0x7623	tap.rgdxaCenter	complex (see below)	4 bytes
sprmTMerge	0x5624	tap.fFirstMerged, tap.fMerged	complex (see below)	word
sprmTSplit	0x5625	tap.fFirstMerged, tap.fMerged	complex (see below)	word
sprmTSetBrc10	0xD626	tap.rgtc[].rgbrc	complex (see below)	5 bytes
sprmTSetShd	0x7627	tap.rgshd	complex (see below)	4 bytes
sprmTSetShdOdd	0x7628	tap.rgshd	complex (see below)	4 bytes
sprmTTextFlow	0x7629	tap.rgtc[].fVertical tap.rgtc[].fBackward tap.rgtc[].fRotateFont	0 or 1 0 or 1 0 or 1	word
sprmTdiagLine	0xD62A			
sprmTVertMerge	0xD62B	tap.rgtc[].vertMerge	complex (see below)	variable length always recorded as 2 bytes
sprmTVertAlign	0xD62C	tap.rgtc[].vertAlign	complex (see below)	variable length always recorded as 3 byte

sprmPIstdPermute (opcode 0xC601) is a complex sprm which is applied to a piece when the style codes of paragraphs within a piece must be mapped to other style codes. It has the following format:

Field	Size	Comment
sprm	short	opcode(==0xC601)
cch	byte	count of bytes (not including sprm and cch)
fLongg	byte	always 0
fSpare	byte	always 0
istdFirst	unsigned short	index of first style in range to which permutation stored in rgistd applies
istdLast	unsigned short	index of last style in range to which permutation stored in rgistd applies
rgistd[]	unsigned short	array of istd entries that records the mapping of istds for text copied from a source document to istds that exists in the destination document after the text has been pasted

To interpret sprmPIstdPermute, first check if pap.istd is greater than the istdFirst recorded in the sprm and less than or equal to the istdLast recorded in the sprm. If not, the sprm has no effect. If it is, pap.istd is set to rgistd[pap.istd - istdFirst]. sprmPIstdPermute is only stored in **grpprls** linked to a piece table. It should never be recorded in a PAPX.

sprmPInclvl (opcode 0x2602) is applied to pieces in the piece table that contain paragraphs with style codes (istds) greater than or equal to 1 and less than or equal to 9. These style codes identify heading levels in a Word outline structure. The sprm causes a set of paragraphs to be changed to a new heading level. The sprm is three bytes long and consists of the sprm code and a one

byte two's complement value.

If `pap.stc` is < 1 or > 9 , `sprmPIncLvl` has no effect. Otherwise, if the value stored in the byte has its highest order bit off, the value is a positive difference which should be added to `pap.istd` and `pap.lvl` and then `pap.stc` should be set to $\min(\text{pap.istd}, 9)$. If the byte value has its highest order bit on, the value is a negative difference which should be sign extended to a word and then subtracted from `pap.istd` and `pap.lvl`. Then `pap.stc` should be set to $\max(1, \text{pap.istd})$. `sprmPIncLvl` is only stored in **grpprls** linked to a piece table.

`sprmPIlfo` (opcode 0x460B) sets the `pap.ilfo`. Its argument, an `ilfo`, is an index into the document's `hpllfo`, which contains the list data for that paragraph, describing the appearance of the automatic number at the beginning of the paragraph. A value of zero means that the paragraph is not numbered, and a value of 2047 indicates that this paragraph came from a pre-Word 97 file so the formatting information is still stored in the `pap.anld` and the paragraph should be converted to Word 97 format.

`sprmPIlvl` (opcode 0x260A) sets the `pap.ilvl`. It takes an index (0 through 8) which indicates which level of a multilevel list this paragraph belongs to. For simple (one-level lists) or unnumbered paragraphs, this value should always be zero.

`sprmPANld` (opcode ...) is currently only used for compatibility with pre-Word 97 docs. It sets the `pap.anld`, which before Word 97 described the automatic number at the beginning of any numbered paragraph. Now we use it only long enough to put the data into the document's list table (`rglst`) and set the `pap.ilfo` to point to the proper entry in the list table. The `pap.anld` is only relevant if `pap.ilfo` is equal to 2047 (see `sprmPIlfo` above).

The `sprmPChgTabsPapx` (opcode 0xC60D) is a complex `sprm` that describes changes in tab settings from the underlying style. It is only stored as part of `PAPXs` stored in **FKPs** and in the **STSH**. It has the following format:

Field	Size	Comment
<code>sprm</code>	short	opcode
<code>cch</code>	byte	count of bytes (not including <code>sprm</code> and <code>cch</code>)
<code>itbdDelMax</code>	byte	number of tabs to delete
<code>rgdxaDel</code>	<code>int[itbdDelMax]</code>	array of tab positions for which tabs should be deleted
<code>itbdAddMax</code>	byte	number of tabs to add
<code>rgdxaAdd</code>	<code>int[itbdAddMax]</code>	array of tab positions for which tabs should be added
<code>rgtbdAdd</code>	<code>byte[itbdAddMax]</code>	array of tab descriptors corresponding to <code>rgdxaAdd</code>

When `sprmPChgTabsPapx` is interpreted, the `rgdxaDel` of the `sprm` is applied first to the `pap` that is being transformed. This is done by deleting from the `pap` the `rgdxaTab` entry and `rgtbd` entry of any tab whose `rgdxaTab` value is equal to one of the `rgdxaDel` values in the `sprm`. It is guaranteed that the entries in `pap.rgdxaTab` and the `sprm`'s `rgdxaDel` and `rgdxaAdd` are recorded in ascending `dxa` order.

Then the `rgdxaAdd` and `rgtbdAdd` entries are merged into the `pap`'s `rgdxaTab` and `rgtbd` arrays so that the resulting `pap.rgdxaTab` is sorted in ascending order with no duplicates.

`sprmPNest` (opcode 0x4610) causes its operand, a two-byte `dxa` value to be added to `pap.dxaLeft`. If the result of the addition is less than 0, 0 is stored into `pap.dxaLeft`. It is used to shift the left indent of a paragraph to the right or left. `sprmPNest` is only stored in **grpprls** linked to a piece table.

`sprmPDyaLine` (opcode 0x6412) moves a 4 byte `LSPD` structure into `pap.lspd`. Two short fields are stored in this data structure. The first short in the structure is named `lspd.dyaLine` and the second is named `lspd.fMultLinespace`. When `lspd.fMultLinespace` is 0, the magnitude of `lspd.dyaLine` specifies the amount of space that will be provided for lines in the paragraph in `twips`. When `lspd.dyaLine` is positive, Word will ensure that AT LEAST the magnitude of `lspd.dyaLine` will be reserved on the page for each line displayed in the paragraph. If the height of a line becomes greater than `lspd.dyaLine`, the size calculated for that line will be reserved on the page. When `lspd.dyaLine` is negative, Word will ensure that EXACTLY the magnitude of `lspd.dyaLine` (-

lspd.dyaLine) will be reserved on the page for each line displayed in the paragraph. When lspd.fMultLinespace is 1, Word will reserve for each line the (maximal height of the line*lspd.dyaLine)/240.

The sprmPChgTabs (opcode 0xC615) is a complex sprm which describes changes tab settings for any paragraph within a piece. It is only stored as part of a **grpprl** linked to a piece table. It has the following format:

Field	Size	Comment
sprm	short	opcode
cch	byte	count of bytes (not including sprm and cch)
itbdDelMax	byte	number of tabs to delete
rgdxaDel	int[itbdDelMax]	array of tab positions for which tabs should be deleted
rgdxaClose	int[itbdDelMax]	array of tolerances corresponding to rgdxaDel where each tolerance defines an interval around corresponding rgdxaDel entry within which all tabs should be removed
itbdAddMax	byte	number of tabs to add
rgdxaAdd	int[itbdAddMax]	array of tab positions for which tabs should be added
rgtbdAdd	byte[itbdAddMax]	array of tab descriptors corresponding to rgdxaAdd

itbdDelMax and itbdAddMax are defined to be equal to 50. This means that the largest possible instance of sprmPChgTabs is 354. When the length of the sprm is greater than or equal to 255, the cch field will be set equal to 255. When cch == 255, the actual length of the sprm can be calculated as follows: length = 2 + itbdDelMax * 4 + itbdAddMax * 3.

When sprmPChgTabs is interpreted, the rgdxaDel of the sprm is applied first to the pap that is being transformed. This is done by deleting from the pap the rgdxaTab entry and rgtbd entry of any tab whose rgdxaTab value is within the interval [rgdxaDel[i] - rgdxaClose[i], rgdxaDel[i] + rgdxaClose[i]] It is guaranteed that the entries in pap.rgdxaTab and the sprm's rgdxaDel and rgdxaAdd are recorded in ascending dxa order.

Then the rgdxaAdd and rgtbdAdd entries are merged into the pap's rgdxaTab and rgtbd arrays so that the resulting pap.rgdxaTab is sorted in ascending order with no duplicates.

The sprmPPc (opcode 0x261B) is a complex sprm which describes changes in the pap.pcHorz and pap.pcVert. It is able to change both fields' contents in parallel. It has the following format:

b10	b16	field	type	size	bitfield	comments
0	0	sprm	short			opcode
2	2		short	:4	F0	reserved
		pcVert	short	:2	0C	if pcVert == 3, pap.pcVert should not be changed. Otherwise, contains new value of pap.pcVert.
		pcHorz	short	:2	03	if pcHorz == 3, pap.pcHorz should not be changed. Otherwise, contains new value of pap.pcHorz.

Length of sprmPPc is three bytes.

sprmPPc is interpreted by moving pcVert to pap.pcVert if pcVert != 3 and by moving pcHorz to pap.pcHorz if pcHorz != 3. sprmPPc is stored in PAPX FKP's and also in **grpprls** linked to piece table entries.

sprmPPPropRMark (opcode 0xC63F) is interpreted by moving the first parameter byte to pap.fPropRMark, the next two bytes to pap.ibstPropRMark, and the remaining four bytes to pap.dttmPropRMark.

sprmPHugePapx is stored in PAPX FKP's in place of the grpprl of a PAPX which would otherwise be too big to fit in an FKP (as of this writing, 488 bytes is the size of the largest PAPX which can fit in an FKP). The parameter fc gives the location of the grpprl in the data stream. The first word at that fc counts the number of bytes in the grpprl (not including the byte count itself). A sprmPHugePapx should therefore only be found in a PAPX FKP and should be the only sprm in that PAPX's grpprl.

sprmCPicLocation (opcode 0x6A03) is used ONLY IN CHPX FKP's. This sprm moves the 4-byte operand of the sprm into the chp.fcPic field. It simultaneously sets chp.fSpec to 1. This sprm is also when the chp.lTagObj field that is unioned with chp.fcPic is to be set for OLE objects.

sprmCChs (opcode 0xEA08) is used to record a character set id for text that was pasted into the Word document that used a character set different than Word's default character set. When chp.fChsDiff is 0, the character set used for a run of text is the default character set for the version of Word that last saved the document. When chp.fChsDiff is 1, chp.chse specifies the character set used for this run of text. This sprm's operand is 3 bytes. When this sprm is interpreted, the first byte of the operand is moved to chp.fChsDiff and the remaining word is moved to chp.chse.

sprmCSymbol (opcode 0x6A09) is used to specify the font and the character that will be used within that font to display a symbol character in Word. This sprm's operand is 4 bytes. The first 2 hold the font code; the last 2 hold a character specifier. When this sprm is interpreted, the font code is moved to chp.ftcSym and the character specifier is moved to chp.xchSym and chp.fSpec is set to 1.

sprmCIstdPermute (opcode 0xCA31) (which has the same format as sprmPIstdPermute (opcode 0xC601)) is a complex sprm which is applied to a piece when the style codes for character styles tagging character runs within a piece must be mapped to other style codes. It has the following format:

Field	Size	Comment
sprm	short	opcode(==0xCA31)
cch	byte	count of bytes (not including sprm and cch)
fLongg	byte	always 0
fSpare	byte	always 0
istdFirst	unsigned short	index of first style in range to which permutation stored in rgstd applies
istdLast	unsigned short	index of last style in range to which permutation stored in rgstd applies
rgstd[]	unsigned short	array of istd entries that records the mapping of istds for text copied from a source document to istds that exists in the destination document after the text has been pasted

To interpret sprmCIstdPermute, first check if chp.istd is greater than the istdFirst recorded in the sprm and less than or equal to the istdLast recorded in the sprm. If not, the sprm has no effect. If it is, chp.istd is set to rgstd[chp.istd - istdFirst] and any chpx stored in that rgstd entry is applied to the chp. sprmCIstdPermute is only stored in **grpprls** linked to a piece table. It should never be recorded in a CHPX.

Note that it is possible that an istd may be recorded in the rgstd that refers to a paragraph style. This will no harmful consequences since the istd for a paragraph style should never be recorded in chp.istd.

sprmCDefault (opcode 0x2A32) clears the fBold, fItalic, fOutline, fStrike, fShadow, fSmallCaps, fCaps, fVanish, kul and ico fields of the chp to 0. It was first defined for Word 3.01 and had to be backward compatible with Word 3.00 so it is a variable length sprm whose count of bytes is 0. It consists of the sprmCDefault opcode followed by a byte of 0. sprmCDefault is stored only in **grpprls** linked to piece table entries.

sprmCPlain (opcode 0x2A33) is used to make the character properties of runs of text equal to the style character properties of the paragraph that contains the text. When Word interprets this sprm, the style sheet CHP is copied over the original CHP preserving the fSpec setting from the original CHP. sprmCPlain is stored only in **grpprls** linked to piece table entries.

sprms 0x0835 through 0x083C (sprmCFBold through sprmCFVanish) set single bit properties in the CHP. When the parameter of the sprm is set to 0 or 1, then the CHP property is set to the parameter value.

When the parameter of the sprm is 128, then the CHP property is set to the value that is stored for the property in the style sheet. CHP When the parameter of the sprm is 129, the CHP property is set to the negation of the value that is stored for the property in the style sheet CHP. sprmCFBold through sprmCFVanish are stored only in **grppls** linked to piece table entries.

sprmCSizePos (opcode 0xEA3F) is a five-byte sprm consisting of the sprm opcode and a three byte parameter. The sprm has the following format:

b10	b16	field	type	size	bitfield	comments
0	0	sprm	short			opcode
2	2	hpsSize	short	:8	FF	when != 0, contains new size of chp.hps
3	3	cInc	short	:7	FE	contains the number of font levels to increase or decrease size of chp.hps as a twos complement value.
		fAdjust	short	:1	01	when == 1, means that chp.hps should be adjusted up/down by one font level for super/subscripting change
4	4	hpsPos	short	:8	FF	when != 128, contains super/subscript position as a twos complement number

When Word interprets this sprm, if hpsSize != 0 then chp.hps is set to hpsSize. If cInc is != 0, the cInc is interpreted as a 7 bit twos complement number and the procedure described below for interpreting sprmCHpsInc is followed to increase or decrease the chp.hps by the specified number of levels. If hpsPos is != 128, then chp.hpsPos is set equal to hpsPos. If fAdjust is on , hpsPos != 128 and hpsPos != 0 and the previous value of chp.hpsPos == 0, then chp.hps is reduced by one level following the method described for sprmCHpsInc. If fAdjust is on, hpsPos == 0 and the previous value of chp.hpsPos != 0, then the chp.hps value is increased by one level using the method described below for sprmCHpsInc.

sprmCHpsInc(opcode 0x2A44) is a three-byte sprm consisting of the sprm opcode and a one-byte parameter. Word keeps an ordered array of the font sizes that are defined for the fonts recorded in the system file with each font size transformed into an hps. The parameter is a one-byte twos complement number. Word uses this number to calculate an index in the font size array to determine the new hps for a run. When Word interprets this sprm and the parameter is positive, it searches the array of font sizes to find the index of the smallest entry in the font size table that is greater than the current chp.hps. It then adds the parameter minus 1 to the index and maxes this with the index of the last array entry. It uses the result as an index into the font size array and assigns that entry of the array to chp.hps.

When the parameter is negative, Word searches the array of font sizes to find the index of the entry that is less than or equal to the current chp.hps. It then adds the negative parameter to the index and does a min of the result with 0. The result of the min function is used as an index into the font size array and that entry of the array is assigned to chp.hps. sprmCHpsInc is stored only in **grppls** linked to piece table entries.

sprmCHpsPosAdj (opcode 0x2A46) causes the hps of a run to be reduced the first time text is superscripted or subscripted and causes the hps of a run to be increased when superscripting/subscripting is removed from a run. The one byte parameter of this sprm is the new hpsPos value that is to be stored in chp.hpsPos. If the new hpsPos is not equal 0 (meaning that the text is to be super/subscripted), Word first examines the current value of chp.hpsPos to see if it is equal to 0. If so, Word uses the algorithm described for sprmCHpsInc to decrease chp.hps by one level. If the new hpsPos == 0 (meaning the text is not super/subscripted), Word examines the current chp.hpsPos to see if it is not equal to 0. If it is not (which means text is being restored to normal position), Word uses the sprmCHpsInc algorithm to increase chp.hps by one level. After chp.hps is adjusted, the parameter value is stored in chp.hpsPos. sprmCHpsPosAdj is stored only in **grppls** linked to piece table entries.

The parameter of sprmCMajority (opcode 0xCA47) is itself a list of character sprms which encodes a criterion under which certain fields of the chp are to be set equal to the values stored in a style's CHP. Bytes 0 and 1 of sprmCMajority contains the opcode, byte 2 contains the length of the following list of character sprms. . Word begins interpretation of this sprm by applying the stored character sprm list to a standard chp. That chp has chp.istd = istdNormalChar. chp.hps=20, chp.lid=0x0400 and chp.ftc = 4. Word then compares fBold, fItalic, fStrike, fOutline, fShadow, fSmallCaps, fCaps, ftc, hps, hpsPos, kul, qpsSpace

and ico in the original CHP with the values recorded for these fields in the generated CHP.. If a field in the original CHP has the same value as the field stored in the generated CHP, then that field is reset to the value stored in the style's CHP. If the two copies differ, then the original CHP value is left unchanged. sprmCMajority is stored only in **grpprls** linked to piece table entries.

sprmCHpsInc1 (opcode 0xCA4A) is used to increase or decrease chp.hps by increments of 1. This sprm is interpreted by adding the two byte increment stored as the opcode of the sprm to chp.hps. If this result is less than 8, the chp.hps is set to 8. If the result is greater than 32766, the chp.hps is set to 32766.

sprmCMajority50 (opcode 0xCA4C) has the same format as sprmCMajority and is interpreted in the same way.

sprmCPropRMark (opcode 0xCA57) is interpreted by moving the first parameter byte to chp.fPropRMark, the next two bytes to chp.ibstPropRMark, and the remaining four bytes to chp.dttmPropRMark.

sprmCDispFldRMark (opcode 0xCA62) is interpreted by moving the first parameter byte to chp.fDispFldRMark, the next two bytes to chp.ibstDispFldRMark, the next four bytes to chp.dttmDispFldRMark, and the remaining 32 bytes to chp.xstDispFldRMark.

sprmCRgftc0 (opcode 0x4A4F), sprmcCRgftc1(opcode 0x4A50), and sprmCRgftc2 (opcode 0x4A4F) are used to specify the fonts that are available for use with text. Rgftc0 specifies the font used for characters from U+0000 -> U+007F. Rgftc1 specifies the font to be used for Far East characters, and Rgftc2 specifies the font to be used for all other text. See appendix C for details on how the font is calculated.

sprmCRglid0 (opcode 0x486D) and sprmCRglid1 (opcode 0x486E) are used to specify the languages that are available for use with the text in this run. sprmCRglid1 specifies the language for Far East text, sprmCRglid0 specifies the language for all other text. See Appendix C for details on the language is calculated.

sprmCIctHint (opcode 0x286F) specifies a script bias for the text in the run. For Unicode characters that are shared between Far East and non-Far East scripts, this property determines what font and language the character will use. When this value is 0, text properties bias towards non-Far East properties. When this value is 1, text properties bias towards Far East properties. See Appendix C for details on the calculation of font and language properties.

sprmPicScale (opcode 0xCE01) is used to scale the x and y dimensions of a Word picture and to set the cropping for each side of the picture. The sprm begins with the two-byte opcode, followed by the length of the parameter (always 12) stored in a byte. The 12-byte long operand consists of an array of 6 two-byte integer fields. The 0th integer contains the new setting for pic.mx. The 1st integer contains the new setting for pic.my. The 2nd integer contains the new setting for pic.dxaCropLeft. The 3rd integer contains the new setting for pic.dyaCropTop. The 4th integer contains the new setting for pic.dxaCropRight. The 5th integer contains the new setting of pic.dxaCropBottom. sprmPicScale is stored only in **grpprls** linked to piece table entries.

sprmSPropRMark (opcode 0xD227) is interpreted by moving the first parameter byte to sep.fPropRMark, the next two bytes to sep.ibstPropRMark, and the remaining four bytes to sep.dttmPropRMark.

sprmSTextFlow (opcode 0x5033) represents the textflow to be applied to this section. Possible values are:

0	horizontal, non-@font
1	top to bottom, @font
2	bottom to top, non-@font
3	top to botton, non-@font
4	horizontal, @-font

sprmTDxaLeft (opcode 0x9601) is called to adjust the x position within a column which marks the left boundary of text within the first cell of a table row. This sprm causes a whole table row to be shifted left or right within its column leaving the horizontal width and vertical height of cells in the row unchanged. Bytes 0-1 of the sprm contains the opcode, and the new dxa position, call it dxaNew, is stored as an integer in bytes 2 and 3. Word interprets this sprm by adding dxaNew - (rgdxaCenter[0] +

tap.dxaGapHalf) to every entry of tap.rgdxaCenter whose index is less than tap.itcMac. sprmTDxaLeft is stored only in **grpprls** linked to piece table entries.

sprmTDxaGapHalf (opcode 0x9602) adjusts the white space that is maintained between columns by changing tap.dxaGapHalf. Because we want the left boundary of text within the leftmost cell to be at the same location after the sprm is applied, Word also adjusts tap.rgdxCenter[0] by the amount that tap.dxaGapHalf changes. Bytes 0-1 of the sprm contains the opcode, and the new dxaGapHalf, call it dxaGapHalfNew, is stored in bytes 2 and 3. When the sprm is interpreted, the change between the old and new dxaGapHalf values, tap.dxaGapHalf - dxaGapHalfNew, is added to tap.rgdxCenter[0] and then dxaGapHalfNew is moved to tap.dxaGapHalf. sprmTDxaGapHalf is stored in PAPXs and also in **grpprls** linked to piece table entries.

sprmTTableBorders (opcode 0xD605) sets the tap.rgbrcTable. The sprm is interpreted by moving the 24 bytes of the sprm's operand to tap.rgbrcTable.

sprmTDefTable10 (opcode 0xD606) is an obsolete version of sprmTDefTable (opcode 0xD608) that was used in WinWord 1.x. Its contents are identical to those in sprmTDefTable, except that the TC structures contain the obsolete structures BRC10s.

sprmTDefTable (opcode 0xD608) defines the boundaries of table cells (tap.rgdxaCenter) and the properties of each cell in a table (tap.rgtc). Bytes 0 and 1 of the sprm contain its opcode. Bytes 2 and 3 store a two-byte length of the following parameter. Byte 4 contains the number of cells that are to be defined by the sprm, call it itcMac. When the sprm is interpreted, itcMac is moved to tap.itcMac. itcMac cannot be larger than 32. In bytes 5 through $5+2*(itcMac + 1) - 1$, is stored an array of integer dxa values sorted in ascending order which will be moved to tap.rgdxaCenter. In bytes $5+2*(itcMac + 1)$ through byte $5+2*(itcMac + 1) + 10*itcMac - 1$ is stored an array of TC entries corresponding to the stored tap.rgdxaCenter. This array is moved to tap.rgtc. sprmTDefTable is only stored in PAPXs.

sprmTDefTableShd (opcode 0xD609) is similar to sprmTDefTable, and compliments it by defining the shading of each cell in a table (tap.rgshd). Bytes 0 and 1 of the sprm contain its opcode. Bytes 2 and 3 store a two-byte length of the following parameter. Byte 4 contains the number of cells that are to be defined by the sprm, call it itcMac. itcMac cannot be larger than 32. In bytes 5 through $5+2*(itcMac + 1) - 1$, is stored an array of SHDs. This array is moved to tap.rgshd. sprmTDefTable is only stored in PAPXs.

sprmTSetBrc (opcode 0xD620) allows the border definitions(BRCs) within TCs to be set to new values. It has the following format:

b10	b16	field	type	size	bitfield	comments
0	0	sprm	short			opcode 0xD620
2	2	count	byte			number of bytes for operand
3	3	itcFirst	byte			the index of the first cell that is to have its borders changed.
4	4	itcLim	byte			index of the cell that follows the last cell to have its borders changed
5	5		short	:4	F0	reserved
		fChangeRight	short	:1	08	=1 when tap.rgtc[].brcRight is to be changed
		fChangeBottom	short	:1	04	=1 when tap.rgtc[].brcBottom is to be changed
		fChangeLeft	short	:1	02	=1 when tap.rgtc[].brcLeft is to be changed
		fChangeTop	short	:1	01	=1 when tap.rgtc[].brcTop is to be changed
6	6	brc	BRC			new BRC value to be stored in TCs.

This sprm changes the brc fields selected by the fChange* flags in the sprm to the brc value stored in the sprm, for every tap.rgtc entry whose index is greater than or equal to itcFirst and less than itcLim. sprmTSetBrc is stored only in **grpprls** linked to piece table entries.

sprmTInsert (opcode 0x7621) inserts new cell definitions in an existing table's cell structure. Bytes 0 and 1 of the sprm contain

the opcode. Byte 2 is the index within tap.rgdxaCenter and tap.rgtc at which the new dxacenter and tc values will be inserted. Call this index itcInsert. Byte 3 contains a count of the cell definitions to be added to the tap, call it ctc. Bytes 4 and 5 contain the width of the cells that will be added, call it dxacol. If there are already cells defined at the index where cells are to be inserted, tap.rgdxaCenter entries at or above this index must be moved to the entry ctc higher and must be adjusted by adding $ctc * dxacol$ to the value stored. The contents of tap.rgtc at or above the index must be moved $10 * ctc$ bytes higher in tap.rgtc. If itcInsert is greater than the original tap.itcMac, itcInsert - tap.ctc columns beginning with index tap.itcMac must be added of width dxacol (loop from itcMac to itcMac+itcInsert-tap.ctc adding dxacol to the rgdxaCenter value of the previous entry and storing sum as dxacenter of new entry), whose TC entries are cleared to zeros. Beginning with index itcInsert, ctc columns of width dxacol must be added by constructing new tap.rgdxaCenter and tap.rgtc entries with the newly defined rgtc entries cleared to zeros. Finally, the number of cells that were added to the tap is added to tap.itcMac. sprmTInsert is stored only in **grpprls** linked to piece table entries.

sprmTDelete (opcode 0x5622) deletes cell definitions from an existing table's cell structure. Bytes 0 and 1 of the sprm contain the opcode. Byte 2 contains the index of the first cell to delete, call it itcFirst. Byte 3 contains the index of the cell that follows the last cell to be deleted, call it itcLim. sprmTDelete causes any rgdxaCenter and rgtc entries whose index is greater than or equal to itcLim to be moved to the entry that is itcLim - itcFirst lower, and causes tap.itcMac to be decreased by the number of cells deleted. sprmTDelete is stored only in **grpprls** linked to piece table entries.

sprmTDxaCol (opcode 0x7623) changes the width of cells whose index is within a certain range to be a certain value. Bytes 0 and 1 of the sprm contain the opcode. Byte 2 contains the index of the first cell whose width is to be changed, call it itcFirst. Byte 3 contains the index of the cell that follows the last cell whose width is to be changed, call it itcLim. Bytes 4 and 5 contain the new width of the cell, call it dxacol. This sprm causes the itcLim - itcFirst entries of tap.rgdxaCenter to be adjusted so that $tap.rgdxaCenter[i+1] = tap.rgdxaCenter[i] + dxacol$. Any tap.rgdxaCenter entries that exist beyond itcLim are adjusted to take into account the amount added to or removed from the previous columns. sprmTDxaCol is stored only in **grpprls** linked to piece table entries.

sprmTMerge (opcode 0x5624) merges the display areas of cells within a specified range. Bytes 0 and 1 of the sprm contain the opcode. Byte 2 contains the index of the first cell that is to be merged, call it itcFirst. Byte 3 contains the index of the cell that follows the last cell to be merged, call it itcLim. This sprm causes tap.rgtc[itcFirst].fFirstMerged to be set to 1. Cells in the range whose index is greater than itcFirst and less than itcLim have tap.rgtc[].fMerged set to 1. sprmTMerge is stored only in **grpprls** linked to piece table entries.

sprmTSplit (opcode 0x5625) splits the display areas of merged cells into their originally assigned display areas. Bytes 0 and 1 of the sprm contain the opcode. Byte 2 contains the index of the first cell that is to be split, call it itcFirst. Byte 3 contains the index of the cell that follows the last cell to be split, call it itcLim. This sprm clears tap.rgtc[].fFirstMerged and tap.rgtc[].fMerged for all rgtc entries \geq itcFirst and $<$ itcLim. sprmTSplit is stored only in **grpprls** linked to piece table entries.

SprmTSetBrc10 (opcode 0xD626) has the same format as SprmTSetBrc but uses the old BRC10 structure.

sprmTSetShd (opcode 0x7627) allows the shading definitions (SHDs) within a tap to be set to new values. Bytes 0 and 1 of the sprm contain the opcode. Byte 2 contains the index of the first cell whose shading is to be changed, call it itcFirst. Byte 3 contains the index of the cell that follows the last cell whose shading is to be changed, call it itcLim. Bytes 4 and 5 contain the SHD structure, call it shd. This sprm causes the itcLim - itcFirst entries of tap.rgshd to be set to shd. sprmTSetShd is stored only in **grpprls** linked to piece table entries.

sprmTSetShdOdd (opcode 0x7628) is identical to sprmTSetShd, but it only changes the rgshd for odd indices between itcFirst and itcLim. sprmTSetShdOdd is stored only in **grpprls** linked to piece table entries.

sprmTVertMerge (opcode 0xD62B) changes the vertical cell merge properties for a cell in the tap.rgtc[]. Bytes 0 and 1 of the sprm contain the opcode. Byte 2 contains the index of the cell whose vertical cell merge properties are to be changed. Byte 3 codes the new vertical cell merge properties for the cell, a 0 clears both fVertMerge and fVertRestart, a 1 sets fVertMerge and clears fVertRestart, and a 3 sets both flags. sprmTVertMerge is stored only in **grpprls** linked to piece table entries.

sprmTVertAlign (opcode 0xD62C) changes the vertical alignment property in the tap.rgtc[]. Bytes 0 and 1 of the sprm contain the opcode. Byte 2 contains the index of the first cell whose shading is to be changed, call it itcFirst. Byte 3 contains the index of the cell that follows the last cell whose shading is to be changed, call it itcLim. This sprm causes the vertAlign properties of the

itcLim - itcFirst entries of tap.rgtc[] to be set to the new vertical alignment property contained in Byte 4. sprmTVertAlign is stored only in **grpprls** linked to piece table entries.

Complex File Format

There are some differences between the file format of a full saved document and that of a fast saved document. In previous versions of Word, one of the differences was the necessity of the "complex" table information. In Word '97 and later, the fcClx always indicates the location of the "complex" table information and it is always necessary to determine the location and contents of text and properties. This arises due to unicode and unicode compression.

fcClx is the fc where the complex part of the file begins, and cbClx is the size (in bytes) of the complex part. The complex part of the file contains a group of **grpprls** that encode formatting changes made by the user and a piece table (**plcfpcd**). The piece table is needed because the text of the document is not stored contiguously in the file after a fast save.

The complex part of a file (**CLX**) is composed of a number of variable-sized blocks of data. Recorded first are any **grpprls** that may be referenced by the **plcfpcd** (if the **plcfpcd** has no **grpprl** references, no **grpprls** will be recorded) followed by the **plcfpcd**. Each block in the complex part is prefaced by a **clxt (clx type)**, which is a 1-byte code, either 1 (meaning the block contains a **grpprl**) or 2 (meaning this is the **plcfpcd**). A **clxtGrpprl (1)** is followed by a 2-byte cb which is the count of bytes of the **grpprl**. A **clxtPlcfpcd (2)** is followed by a 4-byte lcb which is the count of bytes of the piece table. A full saved file will have no clxtGrpprl's. So the formats of the two types of blocks are:

clxt = 1	clxtGrpprl
cb	count of bytes in grpprl
grpprl	see " Definitions " for description of grpprl; a grpprl can contain sprms modifying character, paragraph, table, section or picture properties

or

clxt = 2	clxtPlcfpcd
lcb	count of bytes in piece table
plcfpcd	piece table

The entire CLX would look like this, depending on the number of grpprl's:

```
clxtGrpprl
cb
grpprl (0th grpprl)
clxtGrpprl
cb
grpprl (1st grpprl)
...
clxtPlcfpcd
cb
plcfpcd
```

When the **prm** in **pcds** stored in the **plcfpcd**, contains an **igrpprl** (index to a grpprl), the index stored is the order in which that grpprl was stored in the **CLX**.

Algorithm to determine the bounds of a paragraph containing a certain character in a complex file

When a document is recorded in non-complex format, the bounds of the paragraph that contains a particular character can be found by calculating the **FC** coordinate of the character, searching the bin table to find an FKP page that describes that **FC**, fetching that FKP, and then searching the FKP to find the interval in the **rgfc** that encloses the character. The bounds of the interval are the **fcFirst** and **fcLim** of the containing paragraph. Every character greater than or equal to **fcFirst** and less than **fcLim** is part of the containing paragraph.

When a document is recorded in complex format, a piece that was originally part of one paragraph can be copied or moved within a different paragraph. To find the beginning of the paragraph containing a character in a complex document, it's first necessary to search for the piece containing the character in the piece table. Then calculate the **FC** in the file that stores the character from the piece table information. Using the **FC**, search the FCs FKP for the largest FC less than the character's FC, call it **fcTest**. If the character at **fcTest-1** is contained in the current piece, then the character corresponding to that FC in the piece is the first character of the paragraph. If that FC is before or marks the beginning of the piece, scan a piece at a time towards the beginning of the piece table until a piece is found that contains a paragraph mark. This can be done by using the end of the piece FC, finding the largest FC in its FKP that is less than or equal to the end of piece FC, and checking to see if the character in front of the FKP FC (which must mark a paragraph end) is within the piece. When such an FKP FC is found, the FC marks the first byte of paragraph text.

To find the end of a paragraph for a character in a complex format file, again it is necessary to know the piece that contains the character and the FC assigned to the character. Using the FC of the character, first search the FKP that describes the character to find the smallest FC in the **rgfc** that is larger than the character FC. If the FC found in the FKP is less than or equal to the limit FC of the piece, the end of the paragraph that contains the character is at the FKP FC minus 1. If the FKP FC that was found was greater than the FC of the end of the piece, scan piece by piece toward the end of the document until a piece is found that contains a paragraph end mark. It's possible to check if a piece contains a paragraph mark by using the FC of the beginning of the piece to search in the FKPs for the smallest FC in the FKP **rgfc** that is greater than the FC of the beginning of the piece. If the FC found is less than or equal to the limit FC of the piece, then the character that ends the paragraph is the character immediately before the FKP FC.

A special procedure must be followed to locate the last paragraph of the main document text when footnote or header/footer text is saved in a Word file (i.e. when **fib.ccpFtn** != 0 or **fib.ccpHdr** != 0).

In this case the CP of that paragraph mark is **fib.ccpText** + **fib.ccpFtn** + **fib.ccpHdr** + **fib.ccpMcr** + **fib.ccpAtn** and the limit CP of the entire **plcfpcd** is **fib.ccpText** + **fib.ccpFtn** + **fib.ccpHdr** + **fib.ccpMcr** + **fib.ccpAtn** + 1.

Algorithm to determine paragraph properties for a paragraph in a complex file

Having found the index **i** of the FC in an FKP that marks the character stored in the file immediately after the paragraph's paragraph mark, it is necessary to use the word offset stored in the first byte of the **fkp.rgbx[i - 1]** to find the PAPX for the paragraph. Using **papx.istd** to index into the properties stored for the style sheet, the paragraph properties of the style are copied to a local PAP. Then the **grpprl** stored in the PAPX is applied to the local PAP, and **papx.istd** along with **fkp.rgbx.phe** are moved into the local PAP. The process thus far has created a PAP that describes what the paragraph properties of the paragraph were at the last full save. Now it's necessary to apply any paragraph **sprms** that were linked to the piece that contains the paragraph's paragraph mark. If **pcd.prm.fComplex** is 0, **pcd.prm** contains 1 **sprm** which should only be applied to the local PAP if it is a paragraph **sprm**. If **pcd.prm.fComplex** is 1, **pcd.prm.igrpprl** is the index of a **grpprl** in the CLX. If that **grpprl** contains any paragraph **sprms**, they should be applied to the local PAP. After applying all of the **sprms** for the piece, the local PAP contains the correct paragraph property values.

Algorithm to determine table properties for a table row in a complex file

To determine the table properties for a table row in a complex file, scan paragraph-by-paragraph toward the end of the table row, until a paragraph is found that has **pap.fTtp** set to 1. This paragraph consists of a single row end character. This row end character is linked to the table properties of the row. To create the TAP for the table row, clear a local TAP to zeros. Then the PAPX for the row end character must be fetched from an FKP, and the table **sprms** that are stored in this PAPX must be applied to the local TAP. The process thus far has created a TAP that describes what the table properties of the table row were at the last full save. Now apply any table **sprms** that were linked to the piece that contains the table row's row end character. If **pcd.prm.fComplex** is 0, **pcd.prm** contains 1 **sprm** which should be applied to the local TAP if it is a table **sprm**. If

pcd.prm.fComplex is 1, pcd.prm.igrpprl is the index of a grpprl in the CLX. If that grpprl contains any table sprms, apply them to the local TAP. After all of the sprms for the piece are applied, the local TAP contains the correct table property values for the table row.

Algorithm to determine the character properties of a character in a complex file

It is first necessary to fetch the paragraph properties of the paragraph that contains the character. The pap.istd of the fetched properties specifies which style sheet entry provides the default character properties for the character. The character properties recorded in the style sheet for that style are copied into a local CHP. Then, the piece containing the character is located in the piece table (plcfpcd) and the fc of the character is calculated. Using the character's FC, the page number of the CHPX FKP that describes the character is found by searching the bin table (hplcfbteChpx). The CHPX FKP stored in that page is fetched and then the rgfc in the FKP is searched to locate the bounds of the run of exception text that encompasses the character. The CHPX for that run is then located within the FKP, and the CHPX is applied to the contents of the local CHP. The process thus far has created a CHP that describes what the character properties of the character were at the last full save. Now apply any character sprms that were linked to the piece that contains the character. If pcd.prm.fComplex is 0, pcd.prm contains 1 sprm which should be applied to the local CHP if it is a character sprm. If pcd.prm.fComplex is 1, pcd.prm.igrpprl is the index of a grpprl in the CLX. If that grpprl contains any character sprms, apply them to the local CHP. After applying all of the sprms for the piece, the local CHP contains the correct properties for the character.

Characters that are within the same piece, same paragraph, and same run of exception text are guaranteed to have the same properties. This fact can be used to construct a scanner that can return the limit CPs and properties of a sequence of characters that all have the same properties.

Algorithm to determine the section properties of a section in a complex file

To determine which section a character belongs to and what its section properties are, it is necessary to use the CP of the character to search the **plcfsed** for the index **i** of the largest CP that is less than or equal to the character's CP. plcfsed.rgcp[**i**] is the CP of the first character of the section and plcfsed.rgcp[**i+1**] is the CP of the character following the section mark that terminates the section (call it cpLim). Then retrieve plcfsed.rgsed[**i**]. The FC in this SED gives the location where the SEPX for the section is stored. Then create a local SEP with default section properties. If the sed.fc != 0xFFFFFFFF, then the sprms within the SEPX that is stored at offset sed.fc must be applied to the local SEP. The process thus far has created a SEP that describes what the section properties of the section at the last full save. Now apply any section sprms that were linked to the piece that contains the section's section mark. If pcd.prm.fComplex is 0, pcd.prm contains 1 sprm which should be applied to the local SEP if it is a section sprm. If pcd.prm.fComplex is 1, pcd.prm.igrpprl is the index of a grpprl in the CLX. If that grpprl contains any section sprms, they should be applied to the local SEP. After applying all of the section sprms for the piece, the local SEP contains the correct section properties.

Algorithm to determine the pic of a picture in a complex file.

The picture sprms contained in the prm's grpprl apply to any picture characters within the piece that have their chp.fSpec character == fTrue. The picture properties for a picture (the PIC described in the Structure Definitions) are derived by fetching the PIC stored with the picture and applying to that PIC any picture sprms linked to the piece containing the picture special character.

Footnotes & Endnotes

In Word the text of footnotes and endnotes is anchored to a particular position within the document's main text, the location of its footnote/endnote reference. The following discussion only describes footnotes, with endnotes being handled identically except that the endnote data structures contain the "edn" abbreviation where footnote data structures contain the "fnd" abbreviation. There is a structure referenced by the fib, the plcffndRef, which records the locations of the footnote references within the main text address space and another structure referenced by the fib, the plcffndTxt, which records the beginning locations of corresponding footnote text within the footnote text address space. The footnote text characters in a full saved file begin at offset fib.fcMin + fib.ccpText and extends till fib.fcMin + fib.ccpText + fib.ccpFtn. In a complex fast-saved document, the footnote text begins at CP fib.ccpText and extends till fib.ccpText + fib.ccpFtn. To find the location of the **ith** footnote

reference in the main text address space, look up the **ith** entry in the **plcffndRef** and find the location of the text corresponding to the reference within the footnote text address space by looking up the **ith** entry in the **plcffndTxt**.

When there are **n** footnotes, the **plcffndTxt** structure consists of **n+2** CP entries. The CP entries mark the beginning character position within the footnote text address space of the footnote text for the footnotes defined for the file. The beginning CP of the text of the **ith** footnote is the **ith** CP within the **plcffndTxt**. The limit CP of the text of the **ith** footnote is the **i+1st** CP within the **plcffndTxt**.

The last character of footnote text for a footnote (i.e. the character at limit CP - 1) is always a paragraph end(ASCII 13). If there are **n** footnotes, the **n+2nd** CP entry value is always 1 greater than the **n+1st** CP entry value. A paragraph end (ASCII 13) is always stored at the file position marked by the **n+1st** CP value.

When there are **n** footnotes, the **plcffndRef** structure consists of **n+1** CP entries followed by **n** integer flags, named **fAuto**. The **ith** CP in the **plcffndRef** corresponds to the **ith** **fAuto** flag. The CP entries give the locations of footnote references within the main text address space. The **n+1th** CP entry contains the value **fib.ccpText + fib.ccpFtn + fib.ccpHdr + 1**. The **fAuto** flag contains 1 whenever the footnote reference name is auto-generated by Word.

When a footnote reference name is automatically generated by Word, Word generates the name by adding 1 to the index number of the reference in the **plcffndRef** and translating that number to ASCII text. When the footnote reference is auto generated, the character at the main text CP position for the footnote reference should be a footnote reference character (ASCII 5) which has a **chp** recorded with **chp.fSpec = 1**.

The number of footnotes stored in a Word binary file can be found by dividing **fib.cbPlcffndTxt** by 4 and subtracting 1.

Headers and Footers

The header and footer text characters in a full saved file begin at offset **fib.fcMin + fib.ccpText + fib.ccpFtn** and extend till **fib.fcMin + fib.ccpText + fib.ccpFtn + fib.ccpHdr**. In a complex fast-saved document, the footnote text begins at CP **fib.ccpText + fib.ccpFtn** and extends till **fib.ccpText + fib.ccpFtn + fib.ccpHdr**. The **plcfhdd**, a table whose location and length within the file is stored in **fib.fcPlcfhdd** and **fib.cbPlcfhdd**, describes where the text of each header/footer begins. If there are **n** headers/footers stored in the Word file, the **plcfhdd** consists of **n + 2** CP entries. The beginning CP of the **ith** header/footer is the **ith** CP in the **plcfhdd**. The limit CP (the CP of character 1 position past the end of a header/footer) of the **ith** header/footer is the **i + 1 st** CP in the **plcfhdd**. Note that at the limit CP - 1, Word always places a **chEop** as a placeholder which is never displayed as part of the header/footer. This allows Word to change an existing header/footer to be empty.

If there are **n** header/footers, the **n+2nd** CP entry value is always 1 greater than the **n+1st** CP entry value. A paragraph end (ASCII 13) is always stored at the file position marked by the **n+1st** CP value.

The transformation in a full saved file from a header/footer CP to an offset from the beginning of a file (**fc**) is **fc = fib.fcMin + ccpText + ccpFtn + cp**.

In Word, headers/footers can be defined for a document that:

- will act as a separator between main text and footnote text
- will print below footnote text on a page when footnote text must be continued on a succeeding page (continuation separator)
- will print above footnote text on a page when the text must be continued from a previous page (continuation notice)
- will act as a separator between main text and endnote text
- will print below endnote text on a page when endnote text must be continued on a succeeding page (continuation separator)
- will print above endnote text on a page when the text must be continued from a previous page (continuation notice)

Also for each section defined for the document, distinct headers can be defined for printing on odd-numbered/right facing pages, even-numbered /left facing pages and the first page of a section. Similarly for each document section, distinct footers can be defined for printing on odd-numbered/right facing pages, even-numbered/left facing pages and the first page of a section.

The **plcfhdd** contains an entry for each kind of header or footer. (The **grpflhdt** is no longer used to find entries in the **plcfhdd**.) Indices in the **plcfhdd** are as follows:

0	header for even pages
1	header for odd pages
2	footer for even pages
3	footer for odd pages
4	header for first page of section
5	footer for first page of section
6	footnote separator
7	footnote continuation separator
8	footnote continuation notice
9	endnote separator
10	endnote continuation separator
11	endnote continuation notice

Page Table

Page table information is optional data which is not always stored in a Word binary file. It may be stored for the main text, footnote text and endnote text. The fib contains three FCPGD structures (fcpgdMother, fcpgdFtn, fcpgdEdn) which point to where the data is stored. Each fcpgd points to a PLF of PGD structures and a PLCF of BKD structures. The PLF of PGD descriptors contains n entries where n is the number of pages in the associated text stream. The PLC of BKDs contains $\geq n$ entries where each entry describes a single break (page break or otherwise) within the text stream. Each BKD is associated with a PGD and contains an ipgd which is an index into the PLF of PGDs. To find the CP range of a given page, traverse the BKDs searching for the first and last BKD which refer to the given page. The CP range of these BKDs is the CP range of the page.

Third-party creators of Word files should not attempt to create fcpgd structures. They can only be created properly using Word's page layout routines. If a Word document is edited in any way, the fcpgds in the fib should be filled with 0s.

Glossary Files

A Word glossary file is a normal Word binary file with two supplemental files, the **sttbfglsy**, the **sttbglstyle** and the **plcfglsty**, also stored in the file. The **sttbfglsy** contains a list of the names of glossary entries, the **sttbglstyle** contains a list of the stylenames for every autotext entry, and the **plcfglsty** contains a table of beginning positions within the text address space of the file of the text of glossary entries.

The **sttbfglsy** begins with an integer count of bytes of the size of the sttbfglsy (includes the size of the integer count of bytes). If there are **n** glossary entries defined, there will follow **n** Pascal-type strings (string preceded by length byte) concatenated one after the other which store glossary entry names. The glossary entry names must be sorted in case-insensitive ascending order. (i.e. **a** and **A** are treated as equal). Also the names **date** and **time** must be included in the list of names. The name of the **ith** glossary entry is the **ith** name defined in the **sttbfglsy**. The extra field in each entry contains an index on the **sttbglstyle** that indicates the stylename of the first paragraph in **plcfglsty**.

The **sttbglstyle** is not sorted and has no duplicates. Each entry has an extra field indicating how many autotext entries have

that style.

If there are **n** glossary entries, the **plcflgsy**, will consist of **n+2** CP entries. The **ith** CP entry will contain the location of the beginning of the text for the **ith** glossary entry. The **i+1st** CP entry will contain the limit CP of the **ith** glossary entry. The character at a CP position of limit CP - 1 is always a paragraph mark. The **n+2nd** CP entry always contains fib.ccpText + fib.ccpFtn + fib.ccpHdr + 1 if there are headers, footers or footnotes stored in the glossary and contains fib.ccpText + fib.ccpFtn + fib.ccpHdr otherwise. The **n+1st** CP entry is always 1 less than the value of the **n+2nd** entry.

The text for the **time** and **date** entries will always be a single paragraph mark (ASCII 13).

Routing Slip

A routing slip is stored in the main document stream as an RS (Routing Slip) structure followed by a set of variable length data. After the RS are 4 null terminated strings. Each string is preceded by a short containing the string length (including the null terminator). The strings are: the subject, the message text, status and title. Following these strings are a variable number (rs.cRecip) of Routing Recipient (RR) records. Each RR is immediately followed by a variable number (rr.cb) of bytes containing private data, which is in turn followed by a null terminated string containing the recipient name.

Autosummary

For a document for which AutoSummary View is active (specified in the ASUMYI), the plcfasumy records the result of the last AutoSummary analysis. Each ASUMYI in the PLCF gives the AutoSummary level for the text starting at the corresponding CP. The level must be non-negative and no greater than the upper bound specified in the ASUMYI. The ASUMYI specifies the current summary view level. In emphasize view mode, all text at and below the current summary view level is highlighted. In reduce view mode, all text above the current summary view level is hidden.

STTBFASSOC (Table of Associated Strings)

The following are indices into a table of associated strings:

ibst	index	description
ibstAssocFileNext	0	unused
ibstAssocDot	1	filename of associated template
ibstAssocTitle	2	title of document
ibstAssocSubject	3	subject of document
ibstAssocKeyWords	4	keywords of document
ibstAssocComments	5	comments of document
ibstAssocAuthor	6	author of document
ibstAssocLastRevBy	7	name of person who last revised the document
ibstAssocDataDoc	8	filename of data document
ibstAssocHeaderDoc	9	filename of header document
ibstAssocCriteria1	10	packed string used by print merge record selection
ibstAssocCriteria2	11	packed string used by print merge record selection
ibstAssocCriteria3	12	packed string used by print merge record selection
ibstAssocCriteria4	13	packed string used by print merge record selection

ibstAssocCriteria5	14	packed string used by print merge record selection
ibstAssocCriteria6	15	packed string used by print merge record selection
ibstAssocCriteria7	16	packed string used by print merge record selection
ibstAssocMax	17	maximum number of strings in string table

The format of the ibstAssocCriteriaX strings are as follows:

```
int  cbIbstAssoc:8;           // BYTE 0  size of ibstAssocCriteriaX string
int  fCompOr:1;             // BYTE 1  set if cond is an or cond
int  iCompOp:7;            // BYTE 1  index of Comparison Operator
char  stMergeField[];       // Name of MergeField
char  stCompInfo[];        // User Supplied Comparison Information
```

Both stMergeField and stCompInfo are variable length character arrays preceded by a length byte.

Structure Definitions

AnnoTation Reference Descriptor (ATRD)

b10	b16	field	type	size	bitfield	comments
0	0	xstUsrInItl	XCHAR[10]			pascal-style string holding initials of annotation author
20	14	ibst	short			index into GrpXstAtnOwners
22	16	ak	short	:2	0003	unused
			short	:14	FFFC	unused
24	18	grfbmc	uns short			unused
26	1A	lTagBkmk	long			when not -1, this tag identifies the annotation bookmark that locates the range of CPs in the main document which this annotation references.

cbATRD (count of bytes of ATRD) is 30 (decimal), 1E(hex).

Autonumbered List Data Descriptor (ANLD)

b10	b16	field	type	size	bitfield	comments
0	0	nfc	unsigned char			number format code 0 Arabic numbering 1 Upper case Roman 2 Lower case Roman 3 Upper case Letter 4 Lower case letter 5 Ordinal
1	1	cxchTextBefore	unsigned char			offset into anld.rgch that is the limit of the text that will be displayed as the prefix of the autonumber text
2	2	cxchTextAfter	unsigned char			anld.cxchTextBefore will be the beginning offset of the text in the anld.rgch that will be displayed as the suffix of an autonumber. The sum of anld.cxchTextBefore + anld.cxchTextAfter will be the limit of the autonumber suffix in anld.rgch
3	3	jc	uns char	:2	03	justification code

						0 left justify 1 center 2 right justify 3 left and right justify
		fPrev	uns char	:1	04	when ==1, number generated will include previous levels (used for legal numbering)
		fHang	uns char	:1	08	when ==1, number will be displayed using a hanging indent
		fSetBold	uns char	:1	10	when ==1, boldness of number will be determined by andl.fBold.
		fSetItalic	uns char	:1	20	when ==1, italicness of number will be determined by andl.fItalic
		fSetSmallCaps	uns char	:1	40	when ==1, andl.fSmallCaps will determine whether number will be displayed in small caps or not.
		fSetCaps	uns char	:1	80	when ==1, andl.fCaps will determine whether number will be displayed capitalized or not
4	4	fSetStrike	uns char	:1	01	when ==1, andl.fStrike will determine whether the number will be displayed using strikethrough or not.
		fSetKul	uns char	:1	02	when ==1, andl.kul will determine the underlining state of the autonumber.
		fPrevSpace	uns char	:1	04	when ==1, autonumber will be displayed with a single prefixing space character
		fBold	uns char	:1	08	determines boldness of autonumber when andl.fSetBold == 1.
		fItalic	uns char	:1	10	determines italicness of autonumber when andl.fSetItalic == 1.
		fSmallCaps	uns char	:1	20	determines whether autonumber will be displayed using small caps when andl.fSetSmallCaps == 1.
		fCaps	uns char	:1	40	determines whether autonumber will be displayed using caps when andl.fSetCaps == 1.
		fStrike	uns char	:1	80	determines whether autonumber will be displayed using caps when andl.fSetStrike == 1.
5	5	kul	uns char	:3	07	determines whether autonumber will be displayed with underlining when andl.fSetKul == 1.
		ico	uns char	:5	F1	color of autonumber
6	6	ftc	short			font code of autonumber
8	8	hps	uns short			font half point size (or 0=auto)
10	A	iStartAt	uns short			starting value (0 to 65535)
12	C	dxaIndent				width of prefix text (same as indent)
14	E	dxaSpace	uns short			minimum space between number and paragraph
16	10	fNumber1	uns char			number only 1 item per table cell
17	11	fNumberAcross	uns char			number across cells in table rows(instead of down)
18	12	fRestartHdn	uns char			restart heading number on section boundary
19	13	fSpareX	uns char			unused(should be 0)
20	14	rgxch	array of 32			characters displayed before/after autonumber

XCHARs

cbANLD (count of bytes of ANLD) is 84 (decimal), 54(hex).

Autonumber Level Descriptor (ANLV)

b10	b16	field	type	size	bitfield	comments
0	0	nfc	unsigned char			number format code 0 Arabic numbering 1 Upper case Roman 2 Lower case Roman 3 Upper case Letter 4 Lower case letter 5 Ordinal
1	1	cxchTextBefore	unsigned char			offset into anld.rgxch that is the limit of the text that will be displayed as the prefix of the autonumber text
2	2	cxchTextAfter	unsigned char			anld.cxchTextBefore will be the beginning offset of the text in the anld.rgxch that will be displayed as the suffix of an autonumber. The sum of anld.cxchTextBefore + anld.cxchTextAfter will be the limit of the autonumber suffix in anld.rgxch
3	3	jc	uns char	:2	03	justification code 0 left justify 1 center 2 right justify 3 left and right justify
		fPrev	uns char	:1	04	when ==1, number generated will include previous levels (used for legal numbering)
		fHang	uns char	:1	08	when ==1, number will be displayed using a hanging indent
		fSetBold	uns char	:1	10	when ==1, boldness of number will be determined by anld.fBold.
		fSetItalic	uns char	:1	20	when ==1, italicness of number will be determined by anld.fItalic
		fSetSmallCaps	uns char	:1	40	when ==1, anld.fSmallCaps will determine whether number will be displayed in small caps or not.
		fSetCaps	uns char	:1	80	when ==1, anld.fCaps will determine whether number will be displayed capitalized or not
4	4	fSetStrike	uns char	:1	01	when ==1, anld.fStrike will determine whether the number will be displayed using strikethrough or not.
		fSetKul	uns char	:1	02	when ==1, anld.kul will determine the underlining state of the autonumber.
		fPrevSpace	uns char	:1	04	when ==1, autonumber will be displayed with a single prefixing space character
		fBold	uns char	:1	08	determines boldness of autonumber when anld.fSetBold == 1.
		fItalic	uns char	:1	10	determines italicness of autonumber when anld.fSetItalic == 1.
		fSmallCaps	uns char	:1	20	determines whether autonumber will be displayed using small caps when anld.fSetSmallCaps == 1.
		fCaps	uns char	:1	40	determines whether autonumber will be displayed using caps when

		fStrike	uns char	:1	80	and.fSetCaps == 1. determines whether autonumber will be displayed using caps when and.fSetStrike == 1.
5	5	kul	uns char	:3	07	determines whether autonumber will be displayed with underlining when and.fSetKul == 1.
		ico	uns char	:5	F1	color of autonumber
6	6	ftc	short			font code of autonumber
8	8	hps	uns short			font half point size (or 0=auto)
10	A	iStartAt	uns short			starting value (0 to 65535)
12	C	dxaIndent				width of prefix text (same as indent)
14	E	dxaSpace	uns short			minimum space between number and paragraph

cbANLV (count of bytes of ANLV) is 16 bytes (decimal), 10 bytes (hex).

AutoSummary Analysis (ASUMY)

b10	b16	field	type	size	bitfield	comments
0	0	lLevel	long			AutoSummary level

cbASUMY (count of bytes of ASUMY) is 4 bytes.

AutoSummary Info (ASUMYI)

b10	b16	field	type	size	bitfield	comments
0	0	fValid	short	:1	0001	true iff the ASUMYI is valid
		fView	short	:1	0002	true iff AutoSummary View is active
		iViewBy	short	:2	000C	Display method for AutoSummary View: 0 = Emphasize in current doc 1 = Reduce doc to summary 2 = Insert into doc 3 = Show in new document
		fUpdateProps	short	:1	0010	true if we should update File Properties summary information after the next summarization
		reserved	short	:11	FFE0	reserved
2	2	wDlgLevel	short			Dialog summary level
4	4	lHighestLevel	long			upper bound for lLevel for sentences in this document
8	8	lCurrentLevel	long			show document sentences at or below this level

cbASUMYI (count of bytes of ASUMYI) is 12 bytes (decimal), C bytes (hex).

Bin Table Entry (BTE)

b10	b16	field	type	size	bitfield	comments
0	0	pn	long	:22		AutoSummary level
		unused	long	:10		unused

cbBTE (count of bytes of BTE) is 4 bytes.

Break Descriptor (BKD)

b10	b16	field	type	size	bitfield	comments
0	0	ipgd	short			except in textbox BKD, index to PGD in plfpgd that describes the page this break is on.
0	0	itxbxs	short			in textbox BKD,
2	2	dcpDepend	short			number of cp's considered for this break; note that the CP's described by cpDepend in this break reside in the next BKD
		icol	uns short	:8	00FF	
		fTableBreak	uns short	:1	0100	when 1, this indicates that this is a table break.
		fColumnBreak	uns short	:1	0200	when 1, this indicates that this is a column break.
		fMarked	uns short	:1	0400	used temporarily while word is running.
		fUnk	uns short	:1	0800	in textbox BKD, when == 1 indicates cpLim of this textbox is not valid
		fTextOverflow	uns short	:1	1000	in textbox BKD, when == 1 indicates that text overflows the end of this textbox

cbBKD (count of bytes of BKD) is 6.

Bookmark First descriptor (BKF)

b10	b16	field	type	size	bitfield	comments
0	0	ibkl	short			index to BKL entry in plcfbkl that describes the ending position of this bookmark in the CP stream.
2	2	itcFirst	uns short	:7	007F	when bkf.fCol is 1, this is the index to the first column of a table column bookmark.
		fPub	uns short	:1	0080	when 1, this indicates that this bookmark is marking the range of a Macintosh Publisher section.
		itcLim	uns short	:7	7F00	when bkf.fCol is 1, this is the index to limit column of a table column bookmark.
		fCol	uns short	:1	8000	when 1, this bookmark marks a range of columns in a table specified by [bkf.itcFirst, bkf.itcLim).

cbBKF (count of bytes of BKF) is 4.

Bookmark Lim descriptor (BKL)

The **BKL** is no longer stored in the **plcfbkl** or **plcfatnbkl**, and is instead reconstructed from the **plcfbkf** or **plcfatnbkf** when the file is opened.

b10	b16	field	type	size	bitfield	comments
0	0	ibkf	short			index to BKF entry in plcfbkf that describes the beginning position of this bookmark in the CP stream. If the bkl.ibkf is negative, add on the number of bookmarks recorded in the hplcbkf to the bkl.ibkf to calculate the index to the BKF that corresponds to this entry.

cbBKL (count of bytes of BKL) is 2.

Border Code (BRC)

The **BRC** is a substructure of the **CHP**, **PAP**, **PIC**, **SEP**, **TAP** and **TC**. See also the obsolete BRC10 structure.

b10	b16	field	type	size	bitfield	comments
0	0	dptLineWidth	short	:8	00FF	width of a single line in 1/8 pt, max of 32 pt.
		brcType	short	:8	FF00	border type code: 0 none 1 single 2 thick 3 double 5 hairline 6 dot 7 dash large gap 8 dot dash 9 dot dot dash 10 triple 11 thin-thick small gap 12 thick-thin small gap 13 thin-thick-thin small gap 14 thin-thick medium gap 15 thick-thin medium gap 16 thin-thick-thin medium gap 17 thin-thick large gap 18 thick-thin large gap 19 thin-thick-thin large gap 20 wave 21 double wave 22 dash small gap 23 dash dot stroked 24 emboss 3D 25 engrave 3D codes 64 - 230 represent border art types and are used only for page borders.
2	2	ico	short	:8	00FF	color code (see chp.ico)
		dptSpace	short	:5	1F00	width of space to maintain between border and text within border. Must be 0 when BRC is a substructure of TC. Stored in points.
		fShadow	short	:1	2000	when 1, border is drawn with shadow. Must be 0 when BRC is a substructure of the TC

		fFrame	short	:1	4000	
			short	:1	8000	reserved

cbBRC (count of bytes of BRC) is 4.

Border Code for Windows Word 1.0 (BRC10)

b10	b16	field	type	size	bitfield	comments
0	0	dxpLine2Width	short	:3	0007	width of second line of border in pixels
		dxpSpaceBetween	short	:3	0038	distance to maintain between both lines of border in pixels
		dxpLine1Width	short	:3	01C0	width of first border line in pixels
		dxpSpace	short	:5	3E00	width of space to maintain between border and text within border. Must be 0 when BRC is a substructure of the TC.
		fShadow	short	:1	4000	when 1, border is drawn with shadow. Must be 0 when BRC10 is a substructure of the TC.
		fSpare	short	:1	8000	reserved

The seven types of border lines that Windows Word 1.0 supports are coded with different sets of values for dxpLine1Width, dxpSpaceBetween, and dxpLine2 Width.

The border lines and their brc10 settings follow:

line type	dxpLine1Width	dxpSpaceBetween	dxpLine2Width
no border	0	0	0
single line border	1	0	0
two single line border	1	1	1
fat solid border	4	0	0
thick solid border	2	0	0
dotted border	6 (special value meaning dotted line)	0	0
hairline border	7(special value meaning hairline)	0	0

When the **no border** settings are stored in the BRC, brc.fShadow and brc.dxpSpace should be set to 0.

cbBRC10 (count of bytes of BRC10) is 2.

Character Properties (CHP)

The **CHP** is never stored in Word files. It is the result of decompression operations applied to **CHPX**s

The **CHPX** is stored in **CHPX FKPS** and within the **STSH**

Note

When a **CHPX** is stored in an **FKP** it is prefixed by a one-byte count of bytes that records the size of the non-zero prefix of the **CHPX**. Since the count of bytes must begin on an even boundary within the **FKP** followed by the non-zero prefix, it's

guaranteed that the int and FC fields of the CHPX are aligned on an odd-byte boundary. Using normal integer or long load instructions will cause address errors on a 68000. The best technique for reconstituting the CHPX is to move the non-zero prefix to the beginning of a local instance of a CHPX that has been cleared to zeros.

b10	b16	field	type	size	bitfield	comment
0	0	fBold	short	:1	0001	text is bold when 1 , and not bold when 0.
		fItalic	short	:1	0002	italic when 1, not italic when 0
		fRMarkDel	short	:1	0004	when 1, text has been deleted and will be displayed with strikethrough when revision marked text is to be displayed
		fOutline	short	:1	0008	outlined when 1, not outlined when 0
		fFldVanish	short	:1	0010	used internally by Word
		fSmallCaps	short	:1	0020	displayed with small caps when 1, no small caps when 0
		fCaps	short	:1	0040	displayed with caps when 1, no caps when 0
		fVanish	short	:1	0080	when 1, text has "hidden" format, and is not displayed unless fPagHidden is set in the DOP
1	1	fRMark	short	:1	0100	when 1, text is newly typed since the last time revision marks have been accepted and will be displayed with an underline when revision marked text is to be displayed
		fSpec	short	:1	0200	character is a Word special character when 1, not a special character when 0
		fStrike	short	:1	0400	displayed with strikethrough when 1, no strikethrough when 0
		fObj	short	:1	0800	embedded object when 1, not an embedded object when 0
		fShadow	short	:1	1000	character is drawn with a shadow when 1; drawn without shadow when 0
		fLowerCase	short	:1	2000	character is displayed in lower case when 1. No case transformation is performed when 0. This field may be set to 1 only when chp.fSmallCaps is 1.
		fData	short	:1	4000	when 1, chp.fcPic points to an FFDATA, the data structure binary data used by Word to describe a form field. The bit chp.fData may only be 1 when chp.fSpec is also 1 and the special character in the document stream that has this property is a chPicture (0x01).
		fOle2	short	:1	8000	when 1, chp.ITagObj specifies a particular object in the object stream that specifies the particular OLE object in the stream that should be displayed when the chPicture fSpec character that is tagged with the fOle2 is encountered. The bit chp.fOle2 may only be 1 when chp.fSpec is also 1 and the special character in the document stream that has this property is a chPicture (0x01).
2	2	fEmboss	short	:1	0001	text is embossed when 1 and not embossed when 0
		fImprint	short	:1	0002	text is engraved when 1 and not engraved when 0
		fDStrike	short	:1	0004	displayed with double strikethrough when 1, no double strikethrough when 0
		fUsePgsuSettings	short	:1	0008	

4	4		short long	:12	FFF0	Reserved Reserved
8	8	ftc	short			no longer stored
10	A	ftcAscii(rgftc[0])	short			font for ASCII text
12	C	ftcFE(rgftc[1])	short			font for Far East text
14	E	ftcOther(rgftc[2])	short			font for non-Far East text
16	10	hps	unsigned short			font size in half points
18	12	dxSpace	long			space following each character in the run expressed in twip units.
22	16	iss	short	:3	0007	superscript/subscript indices 0 means no super/subscripting 1 means text in run is superscripted 2 means text in run is subscripted
		kul	short	:4	0078	underline code: 0 none 1 single 2 by word 3 double 4 dotted 5 hidden 6 thick 7 dash 8 dot (not used) 9 dot dash 10 dot dot dash 11 wave
		fSpecSymbol	short	:1	0080	used by Word internally, not stored in file
23	17	ico	short	:5	1F00	color of text: 0 Auto 1 Black 2 Blue 3 Cyan 4 Green 5 Magenta 6 Red 7 Yellow 8 White 9 DkBlue 10 DkCyan 11 DkGreen 12 DkMagenta 13 DkRed 14 DkYellow 15 DkGray 16 LtGray
			short	:1	2000	reserved
		fSysVanish	short	:1	4000	used by Word internally, not stored in file
		hpsPos	short	:1	8000	reserved

24	18		short			super/subscript position in half points; positive means text is raised; negative means text is lowered.
26	1A	lid	LID			language identification code (no longer stored here, see rglid below)
			Language ID			Language Name
			0x0400			No Proofing
			0x0401			Arabic
			0x0402			Bulgarian
			0x0403			Catalan
			0x0404			Traditional Chinese
			0x0804			Simplified Chinese
			0x0405			Czech
			0x0406			Danish
			0x0407			German
			0x0807			Swiss German
			0x0408			Greek
			0x0409			U.S. English
			0x0809			U.K. English
			0x0c09			Australian English
			0x040a			Castilian Spanish
			0x080a			Mexican Spanish
			0x040b			Finnish
			0x040c			French
			0x080c			Belgian French
			0x0c0c			Canadian French
			0x100c			Swiss French
			0x040d			Hebrew
			0x040e			Hungarian
			0x040f			Icelandic
			0x0410			Italian
			0x0810			Swiss Italian
			0x0411			Japanese
			0x0412			Korean
			0x0413			Dutch
			0x0813			Belgian Dutch
			0x0414			Norwegian - Bokmal
			0x0814			Norwegian - Nynorsk
			0x0415			Polish
			0x0416			Brazilian Portuguese
			0x0816			Portuguese
			0x0417			Rhaeto-Romanic
			0x0418			Romanian
			0x0419			Russian
			0x041a			Croato-Serbian (Latin)
			0x081a			Serbo-Croatian (Cyrillic)
			0x041b			Slovak
			0x041c			Albanian
			0x041d			Swedish
			0x041e			Thai
			0x041f			Turkish
			0x0420			Urdu
			0x0421			Bahasa
			0x0422			Ukrainian
			0x0423			Byelorussian
			0x0424			Slovenian
			0x0425			Estonian
			0x0426			Latvian
			0x0427			Lithuanian
			0x0429			Farsi
			0x042D			Basque
			0x042F			Macedonian
			0x0436			Afrikaans
			0x043E			Malaysian

28	1C	lidDefault(rglid[0])	LID			language for non-Far East text
30	1E	lidFE(rglid[1])	LID			language for Far East text
32	20	idct	unsigned char			not stored in file
33	21	idctHint	unsigned char			Identifier of Characte type
						0 -> shared chars get non-FE props
						1 -> shared chars get FE props
						(see Appendix C)
34	22	wCharScale	unsigned short			
36	24	fcPic	FC			offset in data stream pointing to beginning of a picture when character is a picture character (character is 0x01 and chp.fSpec is 1)
36	24	fcObj	FC			offset in data stream pointing to beginning of a picture when character is an OLE1 object character (character is 0x20 and chp.fSpec is 1, chp.fOle2 is 0)
36	24	lTagObj	unsigned long			long word tag that identifies an OLE2 object in the object stream when the character is an OLE2 object character. (character is 0x01 and chp.fSpec is 1, chp.fOle2 is 1)
40	28	ibstRMark	short			index to author IDs stored in hsttbFRMark. used when text in run was newly typed when revision marking was enabled
42	2A	ibstRMarkDel	short			index to author IDs stored in hsttbFRMark. used when text in run was deleted when revision marking was enabled
44	2C	dtmRMark	DTTM			Date/time at which this run of text was entered/modified by the author. (Only recorded when revision marking is on.)
48	30	dtmRMarkDel	DTTM			Date/time at which this run of text was deleted by the author. (Only recorded when revision marking is on.)
52	34		short			reserved
54	36	istd	unsigned short			index to character style descriptor in the stylesheet that tags this run of text When istd is istdNormalChar (10 decimal), characters in run are not affected by a character style. If chp.istd contains any other value, chpx of the specified character style are applied to CHP for this run before any other exceptional properties are applied.
56	38	ftcSym	short			when chp.fSpec is 1 and the character recorded for the run in the document stream is chSymbol (0x28), chp.ftcSym identifies the font code of the symbol font that will be used to display the symbol character recorded in chp.xchSym. chp.ftcSym is an index into the rgffn structure.
58	3A	xchSym	XCHAR			when chp.fSpec is 1 and the character recorded for the run in the document stream is chSymbol (0x28), the character stored chp.xchSym will be displayed using the font specified in chp.ftcSym.

60	3C	idslRMReason	short			an index to strings displayed as reasons for actions taken by Word's AutoFormat code
62	3E	idslReasonDel	short			an index to strings displayed as reasons for actions taken by Word's AutoFormat code
64	40	ysr	unsigned character			hyphenation rule 0 No hyphenation 1 Normal hyphenation 2 Add letter before hyphen 3 Change letter before hyphen 4 Delete letter before hyphen 5 Change letter after hyphen 6 Delete letter before the hyphen and change the letter preceding the deleted character
65	41	chYsr	unsigned character			the character that will be used to add or change a letter when chp.ysr is 2,3, 5 or 6
66	42	cpg	unsigned short			
68	44	hpsKern	unsigned short			kerning distance for characters in run recorded in half points
70	46	icoHighlight	short	:5	001F	highlight color (see chp.ico)
		fHighlight	short	:1	0020	when 1, characters are highlighted with color specified by chp.icoHighlight.
		kcd	short	:3	01C0	
		fNavHighlight	short	:1	0200	used internally by Word
		fChsDiff	short	:1	0400	
		fMacChs	short	:1	0800	
		fFtcAsciSym	short	:1	1000	
			short	:3	E000	Reserved
72	48	fPropMark	unsigned short			when 1, properties have been changed with revision marking on
74	4A	ibstPropRMark	short			index to author IDs stored in hsttbFRMark. used when properties have been changed when revision marking was enabled
76	4C	dtmPropRMark	DTTM			Date/time at which properties of this were changed for this run of text by the author. (Only recorded when revision marking is on.)
80	50	sfxtText	unsigned char			text animation: 0 no animation 1 Las Vegas lights 2 background blink 3 sparkle text 4 marching ants 5 marchine red ants 6 shimmer
81	51		unsigned char			reserved
82	52		unsigned			reserved

			char			
83	53		unsigned short			reserved
85	55		short			reserved
87	57		DTTM			reserved
91	5B	fDispFldRMark	byte			(Only valid for ListNum fields). When 1, the number for a ListNum field is being tracked in xstDispFldRMark -- if that number is different from the current value, the number has changed.
92	5C	ibstDispFldRMark	short			Index to author IDs stored in hsttbfrMark. used when ListNum field numbering has been changed when revision marking was enabled
94	5E	dttmDispFldRMark	DTTM			The date for the ListNum field number change
98	62	xstDispFldRMark	XCHAR[16]			The string value of the ListNum field when revision mark tracking began
130	82	shd	SHD			shading
132	84	brc	BRC			border

cbCHP (count of bytes of CHP) is 136 (decimal), 88(hex).

The standard CHP is all zeros except:

hps	20 half-points
fcPic	-1
istd	10 (the standard character style)
lidDefault, lidFE	0x0400 (no proofing)
wCharScale	100
fUsePgsuSettings	-1

Character Property Exceptions (CHPX)

The **CHPX** is stored within **Character FKP**s and within the **STSH** in **STD**s for **paragraph style** and **character style** entries.

b10	b16	field	type	size	bitfield	comments
0	0	cb	byte			count of bytes of following data in CHPX.
1	1	grppl	character array			a list of the sprms that encode the differences between CHP for a run of text and the CHP generated by the paragraph and character styles that tag the run.

Date and Time (internal date format) (DTTM)

b10	b16	field	type	size	bitfield	comment
0	0	mint	short	:6	003F	minutes (0-59)
		hr	short	:5	07C0	hours (0-23)

		dom	short	:5	F800	days of month (1-31)
2	2	mon	short	:4	000F	months (1-12)
		yr	short	:9	1FF0	years (1900-2411)-1900
		wdy	short	:3	E000	weekday Sunday=0 Monday=1 Tuesday=2 Wednesday=3 Thursday=4 Friday=5 Saturday=6

cbDTTM (count of bytes of DTTM) is 4.

Drop Cap Specifier(DCS)

b10	b16	field	type	size	bitfield	default value	comment
0	0	fdct	short	:3	0007	0	drop cap type 0 no drop cap 1 normal drop cap 2 drop cap in margin
			short	:5	00F8	0	count of lines to drop
1	1		short	:8			reserved

cbDCS (count of bytes of DCS) is 2.

Drawing Object Grid (DOGRID)

The drawing object grid is Far East only, and it sets up a grid in which Far Eastern characters are displayed (one character per grid square).

b10	b16	field	type	size	bitfield	comment
0	0	xaGrid	short			x-coordinate of the upper left-hand corner of the grid
2	2	yaGrid	short			y-coordinate of the upper left-hand corner of the grid
4	4	dxaGrid	short			width of each grid square
6	6	dyaGrid	short			height of each grid square
8	8	dyGridDisplay	short	:7	007F	the number of grid squares (in the y direction) between each gridline drawn on the screen. 0 means don't display any gridlines in the y direction.
		fTurnItOff	short	:1	0080	suppress display of gridlines
		dxGridDisplay	short	:7	7F00	the number of grid squares (in the x direction) between each gridline drawn on the screen. 0 means don't display any gridlines in the y direction.
		fFollowMargins	short	:1	8000	if true, the grid will start at the left and top margins and ignore xaGrid and yaGrid.

cbDOGRID (count of bytes of DOGRID) is 10 bytes (decimal), A bytes (hex).

Document Properties (DOP)

Each version of Word, the DOP gets a little bit larger. Shown below are three different versions of the DOP: for nFib values < 103, for nFib values between 103 and 105, and for nFib values > 105.. Winword 97 and later products write files with nFib > 105. Word 6.0 for the Macintosh writes files with nFib == 103 or 104. The compatibility options (copts) section was grown (to add more compatibility options in the Tools/Options/Compatibility dialog) and copied to the end of the DOP, so for files with nFib >= 103, the first copts section should be ignored (and the analogous fields in the new copts section used instead), whereas files with nFib < 103 will have DOP's without the new copts section. See below for the addition.

b10	b16	field	type	size	bitfield	default value	comment
0	0	fFacingPages	short	:1	0001	0	1 when facing pages should be printed
		fWidowControl	short	:1	0002	1	1 when widow control is in effect. 0 when widow control disabled.
		fPMHMainDoc	short	:1	0004	0	1 when doc is a main doc for Print Merge Helper, 0 when not; default=0
		grfSuppression	short	:2	0018	0	Default line suppression storage; 0= form letter line suppression; 1= no line suppression; default=0. No longer used.
		fpc	short	:2	0060	1	footnote position code 0 print as endnotes 1 print at bottom of page 2 print immediately

							beneath text
			short	:1	0080	0	unused
1	1	grpflhdt	short	:8	FF00	0	No longer used.
2	2	rncFtn	short	:2	0003	0	restart index for footnotes 0 don't restart note numbering 1 restart for each section 2 restart for each page
		nFtn	short	:14	FFFC	1	initial footnote number for document
4	4	fOutlineDirtySave	short	:1	0001		when 1, indicates that information in the hplcpad should be refreshed since outline has been dirtied
			short	:7	00FE		reserved
5	5	fOnlyMacPics	short	:1	0100		when 1, Word believes all pictures recorded in the document were created on a Macintosh
		fOnlyWinPics	short	:1	0200		when 1, Word believes all pictures recorded in the document were created in Windows
		fLabelDoc	short	:1	0400		when 1, document was created as a print merge labels document
		fHyphCapitals	short	:1	0800		when 1, Word is allowed to hyphenate words that are capitalized. When 0, capitalized may not be hyphenated
		fAutoHyphen	short	:1	1000		when 1, Word will hyphenate newly typed text as a background task
		fFormNoFields	short	:1	2000		
		fLinkStyles	short	:1	4000		when 1, Word will merge styles from its template
		fRevMarking	short	:1	8000		when 1, Word will mark revisions as the document is edited
6	6	fBackup	short	:1	0001		always make backup when document saved when 1.
		fExactCWords	short	:1	0002		when 1, the results of the last Word Count execution (as recorded in several DOP fields) are still exactly correct.
		fPagHidden	short	:1	0004		when 1, hidden document contents are displayed.
		fPagResults	short	:1	0008		when 1, field results are displayed, when 0 field codes are displayed.
		fLockAtn	short	:1	0010		when 1, annotations are locked for editing

		fMirrorMargins	short	:1	0020	swap margins on left/right pages when 1.
			short	:1		reserved
		fDfltTrueType	short	:1	0080	when 1, use TrueType fonts by default (flag obeyed only when doc was created by WinWord 2.x)
7	7	fPagSuppressTopSpacing	short	:1	0100	when 1, file created with SUPPRESSTOPSPACING=YES in win.ini. (flag obeyed only when doc was created by WinWord 2.x).
		fProtEnabled	short	:1	0200	when 1, document is protected from edit operations
		fDispFormFldSel	short	:1	0400	when 1, restrict selections to occur only within form fields
		fRMView	short	:1	0800	when 1, show revision markings on screen
		fRMPrint	short	:1	1000	when 1, print revision marks when document is printed
			short	:1		reserved
		fLockRev	short	:1	4000	when 1, the current revision marking state is locked
		fEmbedFonts	short	:1	8000	when 1, document contains embedded TrueType fonts
8	8	copts.fNoTabForInd	short	:1	0001	compatibility option: when 1, don't add automatic tab stops for hanging indent
		copts.fNoSpaceRaiseLower		:1	0002	compatibility option: when 1, don't add extra space for raised or lowered characters
		copts.fSuppressSpbfAfterPageBreak:		:1	0004	compatibility option: when 1, suppress the paragraph Space Before and Space After options after a page break
		copts.fWrapTrailSpaces		:1	0008	compatibility option: when 1, wrap trailing spaces at the end of a line to the next line
		copts.fMapPrintTextColor		:1	0010	compatibility option: when 1, print colors as black on non-color printers
		copts.fNoColumnBalance		:1	0020	compatibility option: when 1, don't balance columns for Continuous Section starts
		copts.fConvMailMergeEsc		:1	0040	
		copts.fSupressTopSpacing		:1	0080	compatibility option: when 1, suppress extra line spacing at top of page
		copts.fOrigWordTableRules		:1	0100	compatibility option: when 1, combine table borders like Word 5.x for the Macintosh
		copts.fTransparentMetafiles		:1	0200	compatibility option: when 1, don't blank area between metafile pictures
		copts.fShowBreaksInFrames		:1	0400	compatibility option: when 1, show hard page

		copts.fSwapBordersFacingPgs		:1	0800	compatibility option: when 1, swap left and right pages on odd facing pages
					F000	reserved
10	A	dxaTab	uns short			720 twips default tab width
12	C	wSpare	uns short			
14	E	dxaHotZ	uns short			width of hyphenation hot zone measured in twips
16	10	cConsecHypLim	uns short			number of lines allowed to have consecutive hyphens
18	12	wSpare2	uns short			reserved
20	14	dtmCreated	DTTM			date and time document was created
24	18	dtmRevised	DTTM			date and time document was last revised
28	1C	dtmLastPrint	DTTM			date and time document was last printed
32	20	nRevision	int			number of times document has been revised since its creation
34	22	tmEdited	long			time document was last edited
38	26	cWords	long			count of words tallied by last Word Count execution
42	2A	cCh	long			count of characters tallied by last Word Count execution
46	2E	cPg	int			count of pages tallied by last Word Count execution
48	30	cParas	long			count of paragraphs tallied by last Word Count execution
52	34	rncEdn	short	:2	0003	restart endnote number code 0 don't restart endnote numbering 1 restart for each section 2 restart for each page
		nEdn	short	:14	FFFC	beginning endnote number
54	36	epc	short	:2	0003	endnote position code 0 display endnotes at end of section 3 display endnotes at end of document
		nfcFtnRef	short	:4	003C	number format code for auto footnotes 0 Arabic 1 Upper case Roman 2 Lower case Roman 3 Upper case Letter 4 Lower case Letter
		nfcEdnRef	short	:4	03C0	number format code for auto endnotes 0 Arabic 1 Upper case Roman 2 Lower case Roman 3 Upper case Letter

						4 Lower case Letter
		fPrintFormData	short	:1	0400	only print data inside of form fields
		fSaveFormData	short	:1	0800	only save document data that is inside of a form field.
		fShadeFormData	short	:1	1000	shade form fields
				:2	6000	reserved
		fWCFtnEdn	short	:1	8000	when 1, include footnotes and endnotes in word count
56	38	cLines	long			count of lines tallied by last Word Count operation
60	3C	cWordsFtnEnd	long			count of words in footnotes and endnotes tallied by last Word Count operation
64	40	cChFtnEdn	long			count of characters in footnotes and endnotes tallied by last Word Count operation
68	44	cPgFtnEdn	short			count of pages in footnotes and endnotes tallied by last Word Count operation
70	46	cParasFtnEdn	long			count of paragraphs in footnotes and endnotes tallied by last Word Count operation
74	4A	cLinesFtnEdn	long			count of paragraphs in footnotes and endnotes tallied by last Word Count operation
78	4E	lKeyProtDoc	long			document protection password key, only valid if dop.fProtEnabled, dop.fLockAtn or dop.fLockRev are 1.
82	52	wvkSaved	short	:3	0007	document view kind 0 Normal view 1 Outline view 2 Page View
		wScaleSaved	short	:9	0FF8	zoom percentage
		zkSaved	short	:2	3000	zoom type 0 None 1 Full page 2 Page width
		fRotateFontW6	short	:1	4000	This is a vertical document (Word 6/95 only)
		iGutterPos	short	:1	8000	Gutter position for this doc: 0 => side; 1 => top

In a file with nFib < 103-for example, documents created with Word 6.0 for Windows-the DOP would end here. This DOP would have a cbDOP of 84, and a cwDOP of 42.

Files with nFib >= 103, the compatibility options (copts) section at offset 8 was copied here and expanded. Options marked " (see above)" hold the same value that the same-named field in the old copts section above had in files with nFib < 103.

84	54	fNoTabForInd	uns long	:1	00000001	(see above)
		fNoSpaceRaiseLower		:1	00000002	(see above)
		fSupressSpbfAfterPageBreak		:1	00000004	(see above)
		fWrapTrailSpaces		:1	00000008	(see above)
		fMapPrintTextColor		:1	00000010	(see above)
		fNoColumnBalance		:1	00000020	(see above)
		fConvMailMergeEsc		:1	00000040	(see above)
		fSupressTopSpacing		:1	00000080	(see above)
		fOrigWordTableRules		:1	00000100	(see above)
		fTransparentMetafiles		:1	00000200	(see above)
		fShowBreaksInFrames		:1	00000400	(see above)
		fSwapBordersFacingPgs		:1	00000800	(see above)
				:4	0000F000	(reserved)
		fSuppressTopSpacingMac5		:1	00010000	Suppress extra line spacing at top of page like MacWord 5.x
		fTruncDxaExpand		:1	00020000	Expand/Condense by whole number of points.
		fPrintBodyBeforeHdr		:1	00040000	Print body text before header/footer
		fNoLeading		:1	00080000	Don't add leading (extra space) between rows of text
				:1	00100000	(reserved)
		fMWSmallCaps		:1	00200000	Use larger small caps like MacWord 5.x
				:10	FFC00000	(reserved)

For this expanded DOP, cbDOP = 88 and cwDOP = 44.

For files with nFib > 105, the DOP has a number of additional fields:

88	58	adt	short			Autoformat Document Type: 0 for normal. 1 for letter, and 2 for email.
90	5A	doptypography	DOPTYPOGRAPHY			see DOPTYPOGRAPHY
400	190	dogrid	DOGRID			see DOGRID
410	19A	reserved	short	:1	0001	Always set to zero when writing files
		lvl	short	:4	001E	Which outline levels are showing in outline view (0 => heading 1 only, 4 => headings 1 through 5, 9 => all levels showing)
		fGramAllDone	short	:1	0020	Doc has been completely grammar checked
		fGramAllClean	short	:1	0040	No grammer errors exist in doc
		fSubsetFonts	short	:1	0080	if you are doing font embedding, you should only embed the characters in the font that are used in the document
		fHideLastVersion	short	:1	0100	Hide the version created for autoversion

	fHtmlDoc	short	:1	0200	This file is based upon an HTML file
	reserved	short	:1	0400	Always set to zero when writing files
	fSnapBorder	short	:1	0800	Snap table and page borders to page border
	fIncludeHeader	short	:1	1000	Place header inside page border
	fIncludeFooter	short	:1	2000	Place footer inside page border
	fForcePageSizePag	short	:1	4000	Are we in online view
	fMinFontSizePag	short	:1	8000	Are we auto-promoting fonts to >= hpsZoonFontPag?
412	19C	fHaveVersions	short	:1	0001 versioning is turned on
		fAutoVersion	short	:1	0002 autoversioning is enabled
		reserved	short	:14	FFFC Always set to zero when writing files
414	19E	asumyi	ASUMYI		Autosummary info
426	1AA	cChWS	long		Count of characters with spaces
430	1AE	cChWSFtnEdn	long		Count of characters with spaces in footnotes and endnotes
434	1B2	grfDocEvents	long		
438	1B6	fVirusPrompted	long	:1	0001 Have we prompted for virus protection on this doc?
		fVirusLoadSafe	long	:1	0002 If prompted, load safely for this doc?
		KeyVirusSession30	long	:30	FFFC Random session key to sign above bits for a Word session.
442	1BA	Spare	30 bytes		Spare
472	1D8	reserved	long		Always set to zero when writing files
476	1DC	reserved	long		Always set to zero when writing files
480	1E0	cDBC	long		Count of double byte characters
484	1E4	cDBCFTnEdn	long		Count od double byte characters in footnotes and endnotes
488	1E8	reserved	long		Always set to zero when writing files
492	1EC	nfcFtnRef	short		number format code for auto footnote references 0 Arabic 1 Upper case Roman 2 Lower case Roman 3 Upper case Letter 4 Lower case Letter
494	1EE	nfcEdnRef	short		number format code for auto endnote references 0 Arabic 1 Upper case Roman 2 Lower case Roman 3 Upper case Letter 4 Lower case Letter
496	1F0	hpsZoonFontPag	short		minimum font size if fMinFontSizePag is true
498	1F2	dywDispPag	short		height of the window in online view during last repag

For this expanded DOP, cbDOP = 500 and cwDOP = 250.

Document Typography Info (DOPTYOGRAPHY)

These options are Far East only, and are accessible through the Typography tab of the Tools/Options dialog.

b10	b16	field	type	size	bitfield	comment
0	0	fKerningPunct	short	:1	00000001	true if we're kerning punctuation
		iJustification	short	:2	00000006	Kinsoku method of justification: 0 = always expand 1 = compress punctuation 2 = compress punctuation and kana.
		iLevelOfKinsoku	short	:2	00000018	Level of Kinsoku: 0 = Level 1 1 = Level 2 2 = Custom
		f2on1	short	:1	00000020	2-page-on-1 feature is turned on.
		reserved	short	:10	0000FFC0	reserved
2	2	cchFollowingPunct	short			length of rgxchFPunct
4	4	cchLeadingPunct	short			length of rgxchLPunct
6	6	rgxchFPunct	XCHAR[101]			array of characters that should never appear at the start of a line
208	D0	rgxchLPunct	XCHAR[51]			array of characters that should never appear at the end of a line

cbDOPTYOGRAPHY (count of bytes of DOPTYOGRAPHY) is 310 bytes (decimal), 136 (hex)..

Field Descriptor (FLD)

b10	b16	field	type	size	bitfield	comment
0	0	ch	char	:5	1F	type of field boundary the FLD describes: 19 field begin mark 20 field separator mark 21 field end mark
			char	:3	E0	reserved
						variant used when fld.ch == 19 (field begin mark)
1	1	flt	char			field type (see flt table below)
						variant used when fld.ch == 21 (field end mark)
1	1	fDiffer	char	:1	01	ignored for saved file
		fZombieEmbed	char	:1	02	==1 when result still believes this field is an EMBED or LINK field
		fResultDirty	char	:1	04	==1 when user has edited or formatted the result. == 0 otherwise.
		fResultEdited	char	:1	08	==1 when user has inserted text into or deleted text from the result.
		fLocked	char	:1	10	==1 when field is locked from recalc
		fPrivateResult	char	:1	20	==1 whenever the result of the field is never to be shown.

	fNested	char	:1	40	==1 when field is nested within another field
	fHasSep	char	:1	80	==1 when field has a field separator

flt value	live/dead	field type
1		unknown keyword
2	live	possible bookmark (syntax matches bookmark name)
3	live	bookmark reference
4	dead	index entry
5	live	footnote reference
6	live	Set command (for Print Merge)
7	live	If command (for Print Merge)
8	live	create index
9	dead	table of contents entry
10	live	Style reference
11	dead	document reference
12	live	sequence mark
13	live	create table-of-contents
14	live	quote Info variable
15	live	quote Title variable
16	live	quote Subject variable
17	live	quote Author variable
18	live	quote Keywords variable
19	live	quote Comments variable
20	live	quote Last Revised By variable
21	live	quote Creation Date variable
22	live	quote Revision Date variable
23	live	quote Print Date variable
24	live	quote Revision Number variable
25	live	quote Edit Time variable
26	live	quote Number of Pages variable
27	live	quote Number of Words variable
28	live	quote Number of Characters variable
29	live	quote File Name variable
30	live	quote Document Template Name variable
31	live	quote Current Date variable
32	live	quote Current Time variable

33	live	quote Current Page variable
34	live	evaluate expression
35	live	insert literal text
36	live	Include command (Print Merge)
37	live	page reference
38	live	Ask command (Print Merge)
39	live	Fill-in command to display prompt (Print Merge)
40	live	Data command (Print Merge)
41	live	Next command (Print Merge)
42	live	NextIf command (Print Merge)
43	live	SkipIf (Print Merge)
44	live	inserts number of current Print Merge record
45	live	DDE reference
46	live	DDE automatic reference
47	live	Inserts Glossary Entry
48	live	sends characters to printer without translation
49	live	Formula definition
50	live	Goto Button
51	live	Macro Button
52	live	insert auto numbering field in outline format
53	live	insert auto numbering field in legal format
54	live	insert auto numbering field in Arabic number format
55	live	reads a TIFF file
56	live	Link
57	live	Symbol
58	live	Embedded Object
59	live	Merge fields
60	live	User Name
61	live	User Initial
62	live	User Address
63	live	Bar code
64	live	Document variable
65	live	Section
66	live	Section pages
67	live	Include Picture

68	live	Include Text
69	live	File Size
70	live	Form Text Box
71	live	Form Check Box
72	live	Note Reference
73	live	Create Table of Authorities
74	dead	Mark Table of Authorities Entry
75	live	Merge record sequence number
76	either	Macro
77	dead	Private
78	live	Insert Database
79	live	Autotext
80	live	Compare two values
81	live	Plug-in module private
82	live	Subscriber
83	live	Form List Box
84	live	Advance
85	live	Document property
86	live	
87	live	OCX
88	live	Hyperlink
89	live	AutoTextList
90	live	List element
91	live	HTML control

Since dead fields have no entry in the plcffld, the string in the field code must be used to determine the field type. All versions of Word '97 use English field code strings, except French, German, and Spanish versions of Word. The strings for all languages for all possible dead fields are listed below.

flt value	English string	French string	German string	Spanish string	field type
4	XE	EX	XE	E	index entry
9	TC	TE	INHALT	TC	table of contents entry
11	RD	RD	RD	RD	document reference
74	TA	TA	TA	TA	Mark Table of Authorities Entry
76					Macro
77	PRIVATE	PRIVE	PRIVATE	PRIVATESPA	Private

File Shape Address (FSPA)

b10	b16	field	type	size	bitfield	comment
0	0	spid	long			Shape Identifier. Used in conjunction with the office art data (found via fcDggInfo in the FIB) to find the actual data for this shape.
4	4	xaLeft	xa			left of rectangle enclosing shape relative to the origin of the shape
8	8	yaTop	ya			top of rectangle enclosing shape relative to the origin of the shape
12	C	xaRight	xa			right of rectangle enclosing shape relative to the origin of the shape
16	10	yaBottom	ya			bottom of the rectangle enclosing shape relative to the origin of the shape
20	14	fHdr	uns short	:1	0001	1 in the undo doc when shape is from the header doc, 0 otherwise (undefined when not in the undo doc)
		bx	uns short	:2	0006	x position of shape relative to anchor CP 0 relative to page margin 1 relative to top of page 2 relative to text (column for horizontal text; paragraph for vertical text) 3 reserved for future use
		by	uns short	:2	0018	y position of shape relative to anchor CP 0 relative to page margin 1 relative to top of page 2 relative to text (paragraph for horizontal text; column for vertical text)
		wr	uns short	:4	01E0	text wrapping mode 0 like 2, but doesn't require absolute object 1 no text next to shape 2 wrap around absolute object 3 wrap as if no object present 4 wrap tightly around object 5 wrap tightly, but allow holes 6-15 reserved for future use
		wrk	uns short	:4	1E00	text wrapping mode type (valid only for wrapping modes 2 and 4) 0 wrap both sides 1 wrap only on left 2 wrap only on right 3 wrap only on largest side
		fRcaSimple	uns short	:1	2000	when set, temporarily overrides bx, by, forcing the xaLeft, xaRight, yaTop, and yaBottom fields to all be page relative.
		fBelowText	uns short	:1	4000	1 shape is below text 0 shape is above text
		fAnchorLock	uns short	:1	8000	1 anchor is locked 0 anchor is not locked
22	16	cTxbx	long			count of textboxes in shape (undo doc only)

cbFSPA (count of bytes of FSPA) is 26 (decimal), 1A (hex).

Font Family Name (FFN)

b10	b16	field	type	size	bitfield	comment
0	0	cbFfnM1	uns char			total length of FFN - 1.

1	1	prq	uns char	:2	03	pitch request
		fTrueType	uns char	:1	04	when 1, font is a TrueType font
			uns char	:1	08	reserved
		ff	uns char	:3	70	font family id
			uns char	:1	80	reserved
2	2	wWeight	short			base weight of font
4	4	chs	uns char			character set identifier
5	5	ixchSzAlt	uns char			index into ffn.szFfn to the name of the alternate font
6	6	panose	PANOSE			
16	10	fs	FONTSIGNATURE			
40	28	xszFfn	XCHAR[]			zero terminated string that records name of font. Possibly followed by a second xsz which records the name of an alternate font to use if the first named font does not exist on this system. Maximal size of xszFfn is 65 characters.

File Information Block (FIB)

b10	b16	field	type	size	bitfield	comment
-----	-----	-------	------	------	----------	---------

Definition of type: "FIBFCLCB"

Decimal	Hex	Field	Type	Bitfield	Comments
0	0x0000	fc	long		
4	0x0004	lcb	ulong		

Definition of type: "FIB"

In Word version 8, the FIB is reorganized to make future extension easier, and to make it easier to make backward compatible file format changes. The FIB now consists of four substructures: the header and three arrays. The FIB header, is unchanged from past versions. The second part is an array of 16-bit "shorts", most of which were present in earlier versions in different locations. The third part is an array of 32-bit longs, many of which were scattered through the previous version FIB. Finally, there is an array of FC/LCB pairs, which were divided into several disjoint arrays in the previous FIB. Future versions of Word will add entries to the three arrays, so readers of the FIB must be careful to skip over any entries in each array that were not present in the version for which the reader was designed. Writers of the FIB must write exactly as many entries as was defined for the nFib value they put in the FIB.

The **FIBFCLCB** structure, used in the last array in the FIB:

Decimal	Hex	Name	Type	Bitfield Size	Bitfield Mask	Comments
0	0x0000	fc	long			File position where data begins.
4	0x0004	lcb	ulong			Size of data. Ignore fc if lcb is zero.

The **FIB** structure itself:

Dec	Hex	Name	Type	Size	Bitfield	Comments
-----	-----	------	------	------	----------	----------

					Mask	
0	0x0000	fibh	FIBH			Beginning of the FIB header
0	0x0000	wIdent	ushort			magic number
2	0x0002	nFib	ushort			FIB version written. This will be >= 101 for all Word 6.0 for Windows and after documents.
4	0x0004	nProduct	ushort			product version written by
6	0x0006	lid	ushort			language stamp -- localized version In pre-WinWord 2.0 files this value was the nLocale. If value is < 999, then it is the nLocale, otherwise it is the lid.
8	0x0008	pnNext	short			
10	0x000A	fDot	ushort	:1	0x0001	Set if this document is a template
		fGlsy	ushort	:1	0x0002	Set if this document is a glossary
		fComplex	ushort	:1	0x0004	when 1, file is in complex, fast-saved format .
		fHasPic	ushort	:1	0x0008	set if file contains 1 or more pictures
		cQuickSaves	ushort	:4	0x00F0	count of times file was quicksaved
		fEncrypted	ushort	:1	0x0100	Set if file is encrypted
		fWhichTblStm	ushort	:1	0x0200	When 0, this fib refers to the table stream named "0Table", when 1, this fib refers to the table stream named "1Table". Normally, a file will have only one table stream, but under unusual circumstances a file may have table streams with both names. In that case, this flag must be used to decide which table stream is valid.
		fReadOnlyRecommended	ushort	:1	0x0400	Set when user has recommended that file be read read-only
		fWriteReservation	ushort	:1	0x0800	Set when file owner has made the file write reserved
		fExtChar	ushort	:1	0x1000	Set when using extended character set in file
		fLoadOverride	ushort	:1	0x2000	REVIEW
		fFarEast	ushort	:1	0x4000	REVIEW
		fCrypto	ushort	:1	0x8000	REVIEW
12	0x000C	nFibBack	ushort			This file format it compatible with readers that understand nFib at or above this value.
14	0x000E	lKey				File encrypted key, only valid if fEncrypted.
18	0x0012	envr	uchar			environment in which file was created 0 created by Win Word 1 created by Mac Word
19	0x0013	fMac	uchar	:1	0x01	when 1, this file was last saved in the Mac environment
		fEmptySpecial	uchar	:1	0x02	
		fLoadOverridePage	uchar	:1	0x04	

		fFutureSavedUndo	uchar	:1	0x08	
		fWord97Saved	uchar	:1	0x10	
		fSpare0	uchar	:3	0xFE	
20	0x0014	chs	ushort			Default extended character set id for text in document stream. (overridden by chp.chse) 0 by default characters in doc stream should be interpreted using the ANSI character set used by Windows 256 characters in doc stream should be interpreted using the Macintosh character set.
22	0x0016	chsTables	ushort			Default extended character set id for text in internal data structures 0 by default characters stored in internal data structures should be interpreted using the ANSI character set used by Windows 256 characters stored in internal data structures should be interpreted using the Macintosh character set.
24	0x0018	fcMin	long			file offset of first character of text. In non-complex files a CP can be transformed into an FC by the following transformation: $fc = cp + fib.fcMin$.
28	0x001C	fcMac	long			file offset of last character of text in document text stream + 1
32	0x0020	csw	ushort			Count of fields in the array of "shorts"
34	0x0022	rgsw				Beginning of the array of shorts
34	0x0022	wMagicCreated				unique number Identifying the File's creator 0x6A62 is the creator ID for Word and is reserved. Other creators should choose a different value.
36	0x0024	wMagicRevised				identifies the File's last modifier
38	0x0026	wMagicCreatedPrivate				private data
40	0x0028	wMagicRevisedPrivate				private data
42	0x002A	pnFbpChpFirst_W6	short			not used
44	0x002C	pnChpFirst_W6	short			not used
46	0x002E	cpnBteChp_W6	short			not used
48	0x0030	pnFbpPapFirst_W6	short			not used
50	0x0032	pnPapFirst_W6	short			not used
52	0x0034	cpnBtePap_W6	short			not used
54	0x0036	pnFbpLvcFirst_W6	short			not used
56	0x0038	pnLvcFirst_W6	short			not used
58	0x003A	cpnBteLvc_W6	short			not used
60	0x003C	lidFE	short			Language id if document was written by Far East version of Word (i.e. FIB.fFarEast is on)

62	0x003E	clw	ushort		Number of fields in the array of longs
64	0x0040	rglw			Beginning of the array of longs
64	0x0040	cbMac	long		file offset of last byte written to file + 1.
68	0x0044	lProductCreated			contains the build date of the creator. 10695 indicates the creator program was compiled on Jan 6, 1995
72	0x0048	lProductRevised			contains the build date of the File's last modifier
76	0x004C	ccpText	long		length of main document text stream
80	0x0050	ccpFtn	long		length of footnote subdocument text stream
84	0x0054	ccpHdd	long		length of header subdocument text stream
88	0x0058	ccpMcr	long		length of macro subdocument text stream, which should now always be 0.
92	0x005C	ccpAtn	long		length of annotation subdocument text stream
96	0x0060	ccpEdn	long		length of endnote subdocument text stream
100	0x0064	ccpTxbx	long		length of textbox subdocument text stream
104	0x0068	ccpHdrTxbx	long		length of header textbox subdocument text stream.
108	0x006C	pnFbpChpFirst	long		when there was insufficient memory for Word to expand the plcfbte at save time, the plcfbte is written to the file in a linked list of 512-byte pieces starting with this pn
112	0x0070	pnChpFirst	long		the page number of the lowest numbered page in the document that records CHPX FKP information
116	0x0074	cpnBteChp	long		count of CHPX FKPs recorded in file. In non-complex files if the number of entries in the plcfbteChpx is less than this, the plcfbteChpx is incomplete.
120	0x0078	pnFbpPapFirst	long		when there was insufficient memory for Word to expand the plcfbte at save time, the plcfbte is written to the file in a linked list of 512-byte pieces starting with this pn
124	0x007C	pnPapFirst	long		the page number of the lowest numbered page in the document that records PAPX FKP information
128	0x0080	cpnBtePap	long		count of PAPX FKPs recorded in file. In non-complex files if the number of entries in the plcfbtePapx is less than this, the plcfbtePapx is incomplete.
132	0x0084	pnFbpLvcFirst	long		when there was insufficient memory for Word to expand the plcfbte at save time, the plcfbte is written to the file in a linked list of 512-byte pieces starting with this pn
136	0x0088	pnLvcFirst	long		the page number of the lowest numbered page in the document that records LVC FKP information
140	0x008C	cpnBteLvc	long		count of LVC FKPs recorded in file. In non-complex files if the number of entries in the plcfbtePapx is less than this, the plcfbtePapx is incomplete.
144	0x0090	fcIslandFirst	long		

148	0x0094	fcIslandLim	long		
152	0x0098	cfclcb	ushort		Number of fields in the array of FC/LCB pairs.
154	0x009A	rgfclcb			Beginning of array of FC/LCB pairs.
154	0x009A	fcStshfOrig	long		file offset of original allocation for STSH in table stream. During fast save Word will attempt to reuse this allocation if STSH is small enough to fit.
158	0x009E	lcbStshfOrig	ulong		count of bytes of original STSH allocation
162	0x00A2	fcStshf	long		offset of STSH in table stream.
166	0x00A6	lcbStshf	ulong		count of bytes of current STSH allocation
170	0x00AA	fcPlcffndRef	long		offset in table stream of footnote reference PLCF of FRD structures. CPs in PLC are relative to main document text stream and give location of footnote references.
174	0x00AE	lcbPlcffndRef	ulong		count of bytes of footnote reference PLC== 0 if no footnotes defined in document.
178	0x00B2	fcPlcffndTxt	long		offset in table stream of footnote text PLC. CPs in PLC are relative to footnote subdocument text stream and give location of beginnings of footnote text for corresponding references recorded in plcffndRef. No structure is stored in this plc. There will just be n+1 FC entries in this PLC when there are n footnotes
182	0x00B6	lcbPlcffndTxt	ulong		count of bytes of footnote text PLC. == 0 if no footnotes defined in document
186	0x00BA	fcPlcfandRef	long		offset in table stream of annotation reference ATRD PLC. The CPs recorded in this PLC give the offset of annotation references in the main document.
190	0x00BE	lcbPlcfandRef	ulong		count of bytes of annotation reference PLC.
194	0x00C2	fcPlcfandTxt	long		offset in table stream of annotation text PLC. The Cps recorded in this PLC give the offset of the annotation text in the annotation sub document corresponding to the references stored in the plcfandRef. There is a 1 to 1 correspondence between entries recorded in the plcfandTxt and the plcfandRef. No structure is stored in this PLC.
198	0x00C6	lcbPlcfandTxt	ulong		count of bytes of the annotation text PLC
202	0x00CA	fcPlcfsed	long		offset in table stream of section descriptor SED PLC. CPs in PLC are relative to main document.
206	0x00CE	lcbPlcfsed	ulong		count of bytes of section descriptor PLC.
210	0x00D2	fcPlcpad	long		no longer used
214	0x00D6	lcbPlcpad	ulong		no longer used
218	0x00DA	fcPlcfphe	long		offset in table stream of PHE PLC of paragraph heights. CPs in PLC are relative to main document text stream. Only written for files in complex format. Should not be written by third party creators of Word

					files.
222	0x00DE	lcbPlcfphe	ulong		count of bytes of paragraph height PLC. ==0 when file is non-complex .
226	0x00E2	fcSttbfglsty	long		offset in table stream of glossary string table. This table consists of Pascal style strings (strings stored prefixed with a length byte) concatenated one after another.
230	0x00E6	lcbSttbfglsty	ulong		count of bytes of glossary string table. == 0 for non-glossary documents. !=0 for glossary documents.
234	0x00EA	fcPlcflsty	long		offset in table stream of glossary PLC. CPs in PLC are relative to main document and mark the beginnings of glossary entries and are in 1-1 correspondence with entries of sttbfglsty. No structure is stored in this PLC. There will be n+1 FC entries in this PLC when there are n glossary entries.
238	0x00EE	lcbPlcflsty	ulong		count of bytes of glossary PLC.== 0 for non-glossary documents. !=0 for glossary documents.
242	0x00F2	fcPlcfhdd	long		byte offset in table stream of header HDD PLC. CPs are relative to header subdocument and mark the beginnings of individual headers in the header subdocument. No structure is stored in this PLC. There will be n+1 FC entries in this PLC when there are n headers stored for the document.
246	0x00F6	lcbPlcfhdd	ulong		count of bytes of header PLC. == 0 if document contains no headers
250	0x00FA	fcPlcfbteChpx	long		offset in table stream of character property bin table.PLC. FCs in PLC are file offsets in the main stream. Describes text of main document and all subdocuments.
254	0x00FE	lcbPlcfbteChpx	ulong		count of bytes of character property bin table PLC.
258	0x0102	fcPlcfbtePapx	long		offset in table stream of paragraph property bin table.PLC. FCs in PLC are file offsets in the main stream. Describes text of main document and all subdocuments.
262	0x0106	lcbPlcfbtePapx	ulong		count of bytes of paragraph property bin table PLC
266	0x010A	fcPlcfsea	long		offset in table stream of PLC reserved for private use. The SEA is 6 bytes long.
270	0x010E	lcbPlcfsea	ulong		count of bytes of private use PLC.
274	0x0112	fcSttbfffn	long		offset in table stream of font information STTBF. The sttbfffn is a STTBF where is string is actually an FFN structure. The n th entry in the STTBF describes the font that will be displayed when the chp.ftc for text is equal to n . See the FFN file structure definition.
278	0x0116	lcbSttbfffn	ulong		count of bytes in sttbfffn.
282	0x011A	fcPlcffldMom	long		offset in table stream to the FLD PLC of field positions in the main document. The CPs point to the

					beginning CP of a field, the CP of field separator character inside a field and the ending CP of the field. A field may be nested within another field. 20 levels of field nesting are allowed.
286	0x011E	lcbPlcffldMom	ulong		count of bytes in plcffldMom
290	0x0122	fcPlcffldHdr	long		offset in table stream to the FLD PLC of field positions in the header subdocument.
294	0x0126	lcbPlcffldHdr	ulong		count of bytes in plcffldHdr
298	0x012A	fcPlcffldFtn	long		offset in table stream to the FLD PLC of field positions in the footnote subdocument.
302	0x012E	lcbPlcffldFtn	ulong		count of bytes in plcffldFtn
306	0x0132	fcPlcffldAtn	long		offset in table stream to the FLD PLC of field positions in the annotation subdocument.
310	0x0136	lcbPlcffldAtn	ulong		count of bytes in plcffldAtn
314	0x013A	fcPlcffldMcr	long		no longer used
318	0x013E	lcbPlcffldMcr	ulong		no longer used
322	0x0142	fcSttbfkmmk	long		offset in table stream of the STTBF that records bookmark names in the main document
326	0x0146	lcbSttbfkmmk	ulong		
330	0x014A	fcPlcfbkf	long		offset in table stream of the PLCF that records the beginning CP offsets of bookmarks in the main document. See BKF structure definition
334	0x014E	lcbPlcfbkf	ulong		
338	0x0152	fcPlcfbkl	long		offset in table stream of the PLCF that records the ending CP offsets of bookmarks recorded in the main document. No structure is stored in this PLCF.
342	0x0156	lcbPlcfbkl	ulong		
346	0x015A	fcCmnds	long		offset in table stream of the macro commands. These commands are private and undocumented.
350	0x015E	lcbCmnds	ulong		undocument size of undocument structure not documented above
354	0x0162	fcPlcmcr	long		no longer used
358	0x0166	lcbPlcmcr	ulong		
362	0x016A	fcSttbfmcr	long		no longer used
366	0x016E	lcbSttbfmcr	ulong		
370	0x0172	fcPrDrvr	long		offset in table stream of the printer driver information (names of drivers, port, etc.)
374	0x0176	lcbPrDrvr	ulong		count of bytes of the printer driver information (names of drivers, port, etc.)
378	0x017A	fcPrEnvPort	long		offset in table stream of the print environment in portrait mode.
382	0x017E	lcbPrEnvPort	ulong		count of bytes of the print environment in portrait

					mode.
386	0x0182	fcPrEnvLand	long		offset in table stream of the print environment in landscape mode.
390	0x0186	lcbPrEnvLand	ulong		count of bytes of the print environment in landscape mode.
394	0x018A	fcWss	long		offset in table stream of Window Save State data structure. WSS contains dimensions of document's main text window and the last selection made by Word user.
398	0x018E	lcbWss	ulong		count of bytes of WSS. ==0 if unable to store the window state. Should not be written by third party creators of Word files.
402	0x0192	fcDop	long		offset in table stream of document property data structure.
406	0x0196	lcbDop	ulong		count of bytes of document properties.
410	0x019A	fcSttbfAssoc	long		offset in table stream of STTBF of associated strings. The strings in this table specify document summary info and the paths to special documents related to this document. See documentation of the STTBFASSOC.
414	0x019E	lcbSttbfAssoc	ulong		
418	0x01A2	fcClx	long		offset in table stream of beginning of information for complex files. Consists of an encoding of all of the prms quoted by the document followed by the plcpcd (piece table) for the document.
422	0x01A6	lcbClx	ulong		count of bytes of complex file information == 0 if file is non-complex .
426	0x01AA	fcPlcfpgdFtn	long		not used
430	0x01AE	lcbPlcfpgdFtn	ulong		
434	0x01B2	fcAutosaveSource	long		offset in table stream of the name of the original file. fcAutosaveSource and cbAutosaveSource should both be 0 if autosave is off.
438	0x01B6	lcbAutosaveSource	ulong		count of bytes of the name of the original file.
442	0x01BA	fcGrpXstAtnOwners	long		offset in table stream of group of strings recording the names of the owners of annotations stored in the document
446	0x01BE	lcbGrpXstAtnOwners	ulong		count of bytes of the group of strings
450	0x01C2	fcSttbfAtnbkmk	long		offset in table stream of the sttbf that records names of bookmarks for the annotation subdocument
454	0x01C6	lcbSttbfAtnbkmk	ulong		length in bytes of the sttbf that records names of bookmarks for the annotation subdocument
458	0x01CA	fcPlcdoaMom	long		no longer used
462	0x01CE	lcbPlcdoaMom	ulong		
466	0x01D2	fcPlcdoaHdr	long		no longer used

470	0x01D6	lcbPlcdoaHdr	ulong		
474	0x01DA	fcPlcspaMom	long		offset in table stream of the FSPA PLC for main document. == 0 if document has no office art objects.
478	0x01DE	lcbPlcspaMom	ulong		length in bytes of the FSPA PLC of the main document.
482	0x01E2	fcPlcspaHdr	long		offset in table stream of the FSPA PLC for header document. == 0 if document has no office art objects.
486	0x01E6	lcbPlcspaHdr	ulong		length in bytes of the FSPA PLC of the header document.
490	0x01EA	fcPlcfAtnbkf	long		offset in table stream of BKF (bookmark first) PLC of the annotation subdocument
494	0x01EE	lcbPlcfAtnbkf	ulong		length in bytes of BKF (bookmark first) PLC of the annotation subdocument
498	0x01F2	fcPlcfAtnbkl	long		offset in table stream of BKL (bookmark last) PLC of the annotation subdocument
502	0x01F6	lcbPlcfAtnbkl	ulong		length in bytes of PLC marking the CP limits of the annotation bookmarks. No structure is stored in this PLC.
506	0x01FA	fcPms	long		offset in table stream of PMS (Print Merge State) information block. This contains the current state of a print merge operation
510	0x01FE	lcbPms	ulong		length in bytes of PMS. ==0 if no current print merge state. Should not be written by third party creators of Word files.
514	0x0202	fcFormFldSttbs	long		offset in table stream of form field Sttbf which contains strings used in form field dropdown controls
518	0x0206	lcbFormFldSttbs	ulong		length in bytes of form field Sttbf
522	0x020A	fcPlcfendRef	long		offset in table stream of endnote reference PLCF of FRD structures. CPs in PLCF are relative to main document text stream and give location of endnote references.
526	0x020E	lcbPlcfendRef	ulong		
530	0x0212	fcPlcfendTxt	long		offset in table stream of PlcfendRef which points to endnote text in the endnote document stream which corresponds with the plcfendRef. No structure is stored in this PLC.
534	0x0216	lcbPlcfendTxt	ulong		
538	0x021A	fcPlcffldEdn	long		offset in table stream to FLD PLCF of field positions in the endnote subdoc
542	0x021E	lcbPlcffldEdn	ulong		
546	0x0222	fcPlcfpgdEdn	long		not used
550	0x0226	lcbPlcfpgdEdn	ulong		
554	0x022A	fcDggInfo	long		offset in table stream of the office art object table data.

					The format of office art object table data is found in a separate document.
558	0x022E	lcbDggInfo	ulong		length in bytes of the office art object table data
562	0x0232	fcSttbfRMark	long		offset in table stream to STTBF that records the author abbreviations for authors who have made revisions in the document.
566	0x0236	lcbSttbfRMark	ulong		
570	0x023A	fcSttbCaption	long		offset in table stream to STTBF that records caption titles used in the document.
574	0x023E	lcbSttbCaption	ulong		
578	0x0242	fcSttbAutoCaption	long		offset in table stream to the STTBF that records the object names and indices into the caption STTBF for objects which get auto captions.
582	0x0246	lcbSttbAutoCaption	ulong		
586	0x024A	fcPlcfwkb	long		offset in table stream to WKB PLCF that describes the boundaries of contributing documents in a master document
590	0x024E	lcbPlcfwkb	ulong		
594	0x0252	fcPlcfspl	long		offset in table stream of PLCF (of SPLS structures) that records spell check state
598	0x0256	lcbPlcfspl	ulong		
602	0x025A	fcPlcftxbxTxt	long		offset in table stream of PLCF that records the beginning CP in the text box subdoc of the text of individual text box entries. No structure is stored in this PLCF
606	0x025E	lcbPlcftxbxTxt	ulong		
610	0x0262	fcPlcffldTxbx	long		offset in table stream of the FLD PLCF that records field boundaries recorded in the textbox subdoc.
614	0x0266	lcbPlcffldTxbx	ulong		
618	0x026A	fcPlcfhdrtxbxTxt	long		offset in table stream of PLCF that records the beginning CP in the header text box subdoc of the text of individual header text box entries. No structure is stored in this PLC.
622	0x026E	lcbPlcfhdrtxbxTxt	ulong		
626	0x0272	fcPlcffldHdrTxbx	long		offset in table stream of the FLD PLCF that records field boundaries recorded in the header textbox subdoc.
630	0x0276	lcbPlcffldHdrTxbx	ulong		
634	0x027A	fcStwUser	long		Macro User storage
638	0x027E	lcbStwUser	ulong		
642	0x0282	fcSttbttmbd	long		offset in table stream of embedded true type font data.
646	0x0286	cbSttbttmbd	ulong		

650	0x028A	fcUnused	long		
654	0x028E	lcbUnused	ulong		
658	0x0292	rgpgdbkd	FCPGD		beginning of array of fcPgd / fcBkd pairs
658	0x0292	fcPgdMother	long		offset in table stream of the PLF that records the page descriptors for the main text of the doc.
662	0x0296	lcbPgdMother	ulong		
666	0x029A	fcBkdMother	long		offset in table stream of the PLCF that records the break descriptors for the main text of the doc.
670	0x029E	lcbBkdMother	ulong		
674	0x02A2	fcPgdFtn	long		offset in table stream of the PLF that records the page descriptors for the footnote text of the doc.
678	0x02A6	lcbPgdFtn	ulong		
682	0x02AA	fcBkdFtn	long		offset in table stream of the PLCF that records the break descriptors for the footnote text of the doc.
686	0x02AE	lcbBkdFtn	ulong		
690	0x02B2	fcPgdEdn	long		offset in table stream of the PLF that records the page descriptors for the endnote text of the doc.
694	0x02B6	lcbPgdEdn	ulong		
698	0x02BA	fcBkdEdn	long		offset in table stream of the PLCF that records the break descriptors for the endnote text of the doc.
702	0x02BE	lcbBkdEdn	ulong		
706	0x02C2	fcSttbfIntlFld	long		offset in table stream of the STTBF containing field keywords. This is only used in a small number of the international versions of word. This field is no longer written to the file for nFib >= 167.
710	0x02C6	lcbSttbfIntlFld	ulong		Always 0 for nFib >= 167.
714	0x02CA	fcRouteSlip	long		offset in table stream of a mailer routing slip.
718	0x02CE	lcbRouteSlip	ulong		
722	0x02D2	fcSttbSavedBy	long		offset in table stream of STTBF recording the names of the users who have saved this document alternating with the save locations.
726	0x02D6	lcbSttbSavedBy	ulong		
730	0x02DA	fcSttbFnm	long		offset in table stream of STTBF recording filenames of documents which are referenced by this document.
734	0x02DE	lcbSttbFnm	ulong		
738	0x02E2	fcPlcfLst	long		offset in the table stream of list format information.
742	0x02E6	lcbPlcfLst	ulong		
746	0x02EA	fcPifLfo	long		offset in the table stream of list format override information.
750	0x02EE	lcbPifLfo	ulong		

754	0x02F2	fcPlcftxbxBkd	long		offset in the table stream of the textbox break table (a PLCF of BKDs) for the main document
758	0x02F6	lcbPlcftxbxBkd	ulong		
762	0x02FA	fcPlcftxbxHdrBkd	long		offset in the table stream of the textbox break table (a PLCF of BKDs) for the header subdocument
766	0x02FE	lcbPlcftxbxHdrBkd	ulong		
770	0x0302	fcDocUndo	long		offset in main stream of undocumented undo / versioning data
774	0x0306	lcbDocUndo	ulong		
778	0x030A	fcRgbuse	long		offset in main stream of undocumented undo / versioning data
782	0x030E	lcbRgbuse	ulong		
786	0x0312	fcUsp	long		offset in main stream of undocumented undo / versioning data
790	0x0316	lcbUsp	ulong		
794	0x031A	fcUskf	long		offset in table stream of undocumented undo / versioning data
798	0x031E	lcbUskf	ulong		
802	0x0322	fcPlcupcRgbuse	long		offset in table stream of undocumented undo / versioning data
806	0x0326	lcbPlcupcRgbuse	ulong		
810	0x032A	fcPlcupcUsp	long		offset in table stream of undocumented undo / versioning data
814	0x032E	lcbPlcupcUsp	ulong		
818	0x0332	fcSttbGlsyStyle	long		offset in table stream of string table of style names for glossary entries
822	0x0336	lcbSttbGlsyStyle	ulong		
826	0x033A	fcPlgosl	long		offset in table stream of undocumented grammar options PL
830	0x033E	lcbPlgosl	ulong		
834	0x0342	fcPlcocx	long		offset in table stream of undocumented ocx data
838	0x0346	lcbPlcocx	ulong		
842	0x034A	fcPlcfbteLvc	long		offset in table stream of character property bin table.PLC. FCs in PLC are file offsets. Describes text of main document and all subdocuments.
846	0x034E	lcbPlcfbteLvc	ulong		
850	0x0352	ftModified	FILETIME		
850	0x0352	dwLowDateTime	ulong		
854	0x0356	dwHighDateTime	ulong		
858	0x035A	fcPlcflvc	long		offset in table stream of LVC PLCF

862	0x035E	lcbPlcflvc	ulong		size of LVC PLCF, ==0 for non-complex files
866	0x0362	fcPlcasumy	long		offset in table stream of autosummary ASUMY PLCF.
870	0x0366	lcbPlcasumy	ulong		
874	0x036A	fcPlcfgram	long		offset in table stream of PLCF (of SPLS structures) which records grammar check state
878	0x036E	lcbPlcfgram	ulong		
882	0x0372	fcSttbListNames	long		offset in table stream of list names string table
886	0x0376	lcbSttbListNames	ulong		
890	0x037A	fcSttbUssr	long		offset in table stream of undocumented undo / versioning data
894	0x037E	lcbSttbUssr	ulong		

cbFIB (count of bytes of FIB) is 898 (decimal), 382 (hex).

Note

If a table does not exist in the file, its cb in the FIB is zero and its fc is equal to that of the following table (the latter equality is irrelevant, as the cb should be used to determine existence of the table).

Footnote Reference Descriptor (FRD)

The FRD is stored in both the plcffndRef and the plcfendRef

offset (base 10)	field	type	size	bitfield	comments
0		nAuto	short		if > 0, the note is an automatically numbered note, otherwise it has a custom mark

Formatted Disk Page for CHPXs (CHPX FKP)

offset (base 10)	field	type	size	bitfield	comments
0		rgfc	array of FCs		Each FC is the limit FC of a run of exception text.
4*(fkp.crun+1)	rgb	array of bytes			an array of bytes where each byte is the word offset of a CHPX . If the byte stored is 0, there is no difference between run's character properties and the style's character properties.
5*fkp.crun+4		unused space			As new runs/paragraphs are recorded in the FKP , unused space is reduced by 5 if CHPX is already recorded and is reduced by 5+sizeof(CHPX) if property is not already recorded.
511-sizeof(grpchpx)	grpchpx	array of bytes			grpchpx consists of all of the CHPXs stored in FKP concatenated end to end. Each CHPX is prefixed with a count of bytes which records its length.
511		crun	byte		count of runs for CHPX FKP ,

The **CHP** is never stored in a Word file. It is derived by expanding stored **CHPXs**.

Formatted Disk Page for **PAPXs (PAPX FKP)**

offset (base 10)	field	type	size	bitfield	comments
0	rgfc	FC[fkp.crun+1]			Each FC is the limit FC of a paragraph (i.e. points to the next character past an end of paragraph mark). There will be fkp.crun+1 recorded in the FKP.
4*(fkp.crun+1)	rgbx	BX[fkp.crun]			an array of the BX data structure. The ith BX entry in the array describes the paragraph beginning at fkp.rgfc[i]. The BX is a 13 byte data structure. The first byte of each BX is the word offset of the PAPX recorded for the paragraph corresponding to this BX. .. If the byte stored is 0, this represents a 1 line paragraph 15 pixels high with Normal style (stc == 0) whose column width is 7980 dxas. The last 12 bytes of the BX is a PHE structure which stores the current paragraph height for the paragraph corresponding to the BX. If a plcph has an entry that maps to the FC for this paragraph, that entry's PHE overrides the PHE stored in the FKP.11*fkp.crun+4 unused space. As new runs/paragraphs are recorded in the FKP , unused space is reduced by 17 if CHPX/PAPX is already recorded and is reduced by 17+sizeof(PAPX) if property is not already recorded.
511-sizeof(grppapx)	grppapx	array of bytes			grppapx consists of all of the PAPXs stored in FKP concatenated end to end. Each PAPX begins with a count of words which records its length padded to a word boundary.
511	crun	byte			count of paragraphs for PAPX FKP .

The **PAP** is never stored in a Word file. It is derived by expanding stored **PAPXs**.

List LeVeL (on File) (LVLF)

b10	b16	field	type	size	bitfield	comments
0	0x00	iStartAt	long	4		start at value for this list level
4	0x04	nfc	byte	1		number format code (see anld.nfc for a list of options)
5	0x05	jc	uns char	:2	0x03	alignment (left, right, or centered) of the paragraph number.
		fLegal	uns char	:1	0x04	true (==1) if the level turns all inherited numbers to arabic, false if it preserves their number format code (nfc)
		fNoRestart	uns char	:1	0x08	true if the level's number sequence is not restarted by higher (more significant) levels in the list
		fPrev	uns char	:1	0x10	Word 6 compatibility option: equivalent to anld.fPrev (see ANLD)
		fPrevSpace	uns char	:1	0x20	Word 6 compatibility option: equivalent to anld.fPrevSpace (see ANLD)
		fWord6	uns char	:1	0x40	true if this level was from a converted Word 6 document. If it is true, all of the Word 6 compability options become valid; otherwise they are ignored.

6	0x06	rgbxchNums[9]	array	9		contains the character offsets into the LVL's XST of the inherited numbers of previous levels. This array should be zero terminated unless it is full (all 9 levels full). The XST contains place holders for any paragraph numbers contained in the text of the number, and the place holder contains the lvl of the inherited number, so lvl.xst[lvl.rgbxchNums[0]] == the level of the first inherited number in this level.
15	0x0F	ixchFollow	uns char	1		the type of character following the number text for the paragraph: 0 == tab, 1 == space, 2 == nothing.
16	0x10	dxaSpace	long	4		Word 6 compatibility option: equivalent to anld.dxaSpace (see ANLD)
20	0x14	dxaIndent	long	4		Word 6 compatibility optino: equivalent to anld.dxaIndent (see ANLD)
24	0x18	cbGrpprlChpx	byte	1		length, in bytes, of the LVL's grpprlChpx
25	0x19	cbGrpprlPapx	byte	1		length, in bytes, of the LVL's grpprlPapx
26	0X1A	reserved	short	2		reserved

Line Spacing Descriptor (LSPD)

b10	b16	field	type	size	bitfield	comments
0	0	dyaLine	short			see description of sprmPDyaLine for description of the meaning of dyaLine
2	2	fMultLinespace	short			see description of sprmPDyaLine in the Sprm Definitions section for description of the meaning of dyaLine and fMultLinespace fields.

cbLSPD (count of bytes of LSPD) is 4.

LiST Data (on File) (LSTF)

b10	b16	field	type	size	bitfield	comments
0	0x00	lsid	long	4		Unique List ID
4	0x04	tplc	long	4		Unique template code
8	0x08	rgistd[9]	array	18		Array of shorts containing the istd's linked to each level of the list, or istdNil (4095) if no style is linked.
26	0x1A	fSimpleList	uns char	:1	0x01	true if this is a simple (one-level) list; false if this is a multilevel (nine-level) list.
		fRestartHdn	uns char	:1	0x02	Word 6 compatibility option: true if the list should start numbering over at the beginning of each section
		reserved	uns char	:6	0xFC	reserved
27	0x1B	reserved	uns char	byte		reserved

List Format Override (LFO)

b10	b16	field	type	size	bitfield	comments
0	0x0	lsid	long	4		List ID of corresponding LSTF (see LSTF)
4	0x4	reserved	long	4		reserved

8	0x8	reserved	long	4		reserved
12	0xC	clfolvl	uns char	1		count of levels whose format is overridden (see LFOLVL)
13	0xD	reserved	array	3		reserved

List Format Override for a single LeVeL (LFOLVL)

b10	b16	field	type	size	bitfield	comments
0	0	iStartAt	long	4		start-at value if fFormatting == false and fStartAt == true. (if fFormatting == true, the start-at is stored in the LVL)
4	4	ilvl	uns char	:4	0x0F	the level to be overridden
		fStartAt	uns char	:1	0x10	true if the start-at value is overridden
		fFormatting	uns char	:1	0x20	true if the formatting is overridden (in which case the LFOLVL should contain a pointer to a LVL)
		reserved	uns char	:2	0xC0	reserved
5	5	reserved	array	3		reserved

Outline LiST Data (OLST)

b10	b16	field	type	size	bitfield	comments
0	0	rganlv[9]	ANLV			an array of 9 ANLV structures describing how heading numbers should be displayed for each of Word's 9 outline heading levels
144	90	fRestartHdr	uns char			when ==1, restart heading on section break
145	91	fSpareOlst2	uns char			reserved
146	92	fSpareOlst3	uns char			reserved
147	93	fSpareOlst4	uns char			reserved
148	94	rgxch[32]	array of 32 XCHARs			text before/after number

cbOLST (count of bytes of OLST) is 212(decimal), D4(hex).

Number Revision Mark Data (NUMRM)

The NUMRM structure is used to track revision marking data for paragraph numbers, and is stored in the PAP for each numbered paragraph. When revision marking tracking is turned on, we fill out the NUMRM for each number with the data required to recreate the number's text. Then at display time, that string is compared with the current paragraph number string, and displayed as changed (old deleted, current inserted) if the strings differ. The string construction algorithm is the same as for an LVL structure.

b10	b16	field	type	size	bitfield	comment
0	0	fNumRM	uns char	1		True if this paragraph was numbered when revision mark tracking

						was turned on
1	1	Spare	uns char	1		
2	2	ibstNumRM	short	2		index to author IDs stored in hsttbfrMark for the paragraph number change
4	4	dtmNumRM	DTTM	4		Date of the paragraph number change
8	8	rgbxchNums[9]	uns char[9]	9		Index into NUMRM.xst of the locations of paragraph number place holders for each level (see LVL.rgxchNums)
17	11	rgnfc[9]	uns char[9]	9		Number Format Code for the paragraph number place holders for each level (see LVL.nfc)
26	1A	Spare	short	2		
28	1C	PNBR	int [9]	36		Numerical value for each level place holder in NUMRM.xst.
64	40	xst	XCHAR[32]	64		The text string for the paragraph number, containing level place holders

cbNUMRM (count of bytes of NUMRM) is 128 (decimal), 80 (hex).

Page Descriptor (PGD)

b10	b16	field	type	size	bitfield	comments
0	0	*	short	:4	000F	
		fGhost	short	:2	0030	redefine fEmptyPage and fAllFtn. true when blank page or footnote only page
		*	short	:10	FFC0	
0	0	fContinue	short	:1	0001	1 only when footnote is continued from previous page
		fUnk	short	:1	0002	1 when page is dirty (i.e. pagination cannot be trusted)
		fRight	short	:1	0004	1 when right hand side page
		fPgnRestart	short	:1	0008	1 when page number must be reset to 1.
		fEmptyPage	short	:1	0010	1 when section break forced page to be empty.
		fAllFtn	short	:1	0020	1 when page contains nothing but footnotes
			short	:1	0040	unused
		fTableBreaks	short	:1	0080	table breaks have been calculated for this page.
		fMarked	short	:1	0100	used temporarily while word is running.
		fColumnBreaks	short	:1	0200	column breaks have been calculated for this page.
		fTableHeader	short	:1	0400	page had a table header at the end
		fNewPage	short	:1	0800	page has never been valid since created, must recalculate the bounds of this page. If this is the last page, this PGD may really represent many pages.
		bkc	short	:4	F000	section break code
2	2	lnn	uns short			line number of first line, -1 if no line numbering
4	4	pgn	uns			page number as printed

			short			
6	6	dym	long			

cbPGD (count of bytes of PGD) is 10.

Paragraph Height (PHE)

The **PHE** is a substructure of the **PAP** and the **PAPX FKP** and is also stored in the **PLCFPHE**.

b10	b16	field	type	size	bitfield	comments
0	0	fSpare	short	:1	0001	reserved
		fUnk	short	:1	0002	PHE entry is invalid when == 1
		fDiffLines	short	:1	0004	when 1, total height of paragraph is known but lines in paragraph have different heights.
		*	short	:5	00F8	reserved
		clMac	short	:8	FF00	when fDiffLines is 0 is number of lines in paragraph
2	2		short			reserved
4	4	dxaCol	long			width of lines in paragraph
8	8	dymLine	long			when fDiffLines is 0, is height of every line in paragraph in pixels
8	8	dymHeight	long			when fDiffLines is 1, is the total height in pixels of the paragraph

If the PHE is stored in a PAP whose fTtp field is set (non-zero), the following structure is used:

b10	b16	field	type	size	bitfield	comments
0	0	fSpare	short	:1	0001	reserved
		fUnk	short	:1	0002	PHE entry is invalid when == 1
		dcpTtpNext	short	:30		if not == 0, used as a hint when finding the next row
4	4	dxaCol	long			
8	8	dymTableHeight	long			height of table row

cbPHE (the count of bytes of PHE) is 12.

If there is no paragraph height information stored for a paragraph, all of the fields in the **PHE** are set to 0. If a paragraph contains more than 127 lines, the clMac, dylLine variant cannot be used, so fDiffLines must be set to 1 and the total size of the paragraph stored in dylHeight. If a paragraph height is greater than 32767 twips, the height cannot be represented by a **PHE** so all fields of the **PHE** must be set to 0.

If a new Word file is created, the **PHE** of every **papx fkp** entry created to describe the paragraphs of the file should be set to 0. If a Word file is altered in place (a character of the file changed to a new character or a property changed), the paragraph containing the change must have its **papx.phe** field set to 0. If this paragraph is in a table row, the **PHE** in the **papx** at the end of the row (indicated by flnTable) must also be set to 0.

Paragraph Properties (PAP)

b10	b16	field	type	size	bitfield	comments

0	0	istd	uns short			index to style descriptor . This is an index to an STD in the STSH structure
2	2	jc	uns char			justification code 0 left justify 1 center 2 right justify 3 left and right justify
3	3	fKeep	uns char			keep entire paragraph on one page if possible
4	4	fKeepFollow	uns char			keep paragraph on same page with next paragraph if possible
5	5	fPageBreakBefore	uns char			start this paragraph on new page
6	6	fBrLnAbove	short	:1	0001	
		fBrLnBelow	short	:1	0002	
		fUnused	short	:2	0006	reserved
		pcVert	short	:2	0030	vertical position code. Specifies coordinate frame to use when paragraphs are absolutely positioned. 0 vertical position coordinates are relative to margin 1 coordinates are relative to page 2 coordinates are relative to text. This means: relative to where the next non-APO text would have been placed if this APO did not exist.
		pcHorz	short	:2	00C0	horizontal position code. Specifies coordinate frame to use when paragraphs are absolutely positioned. 0 horiz. position coordinates are relative to column. 1 coordinates are relative to margin 2 coordinates are relative to page

/* the brcp and brcl fields have been superseded by the newly defined brcLeft, brcTop, etc. fields. They remain in the PAP for compatibility with MacWord 3.0 */

b10	b16	field	type	size	bitfield	comments
7	7	brcp	uns char			rectangle border codes 0 none 1 border above 2 border below 15 box around 16 bar to left of paragraph
8	8	brcl	uns char			border line style 0 single 1 thick 2 double 3 shadow
9	9					reserved
10	A	ilvl	uns char			when non-zero, list level for this paragraph
11	B	fNoLnn	uns char			no line numbering for this paragraph. (makes this an exception to the section property of line numbering)

12	C	ilfo	short			when non-zero, (1-based) index into the pllfo identifying the list to which the paragraph belongs
14	E	nLvlAnm	uns char			no longer used
15	F					reserved
16	10	fSideBySide	uns char			when 1, paragraph is a side by side paragraph
17	11					reserved
18	12	fNoAutoHyph	uns char			when 0, text in paragraph may be auto hyphenated.
19	13	fWidowControl	uns char			when 1, Word will prevent widowed lines in this paragraph from being placed at the beginning of a page
20	14	dxaRight	long			indent from right margin (signed).
24	18	dxaLeft	long			indent from left margin (signed)
28	1C	dxaLeft1	long			first line indent; signed number relative to dxaLeft
32	20	lspd	LSPD			line spacing descriptor
36	24	dyaBefore	uns long			vertical spacing before paragraph (unsigned)
40	28	dyaAfter	uns long			vertical spacing after paragraph (unsigned)
44	2C	phe	PHE			height of current paragraph.
56	38	fCrLf	uns char			
57	39	fUsePgsuSettings	uns char			
58	3A	fAdjustRight	uns char			
59	3B					reserved
60	3C	fKinsoku	uns char			when 1, apply kinsoku rules when performing line wrapping
61	3D	fWordWrap	uns char			when 1, perform word wrap
62	3E	fOverflowPunct	uns char			when 1, apply overflow punctuation rules when performing line wrapping
63	3F	fTopLinePunct	uns char			when 1, perform top line punctuation processing
64	40	fAutoSpaceDE	uns char			when 1, auto space FE and alphabetic characters
65	41	fAtuoSpaceDN	uns char			when 1, auto space FE and numeric characters
66	42	wAlignFont	uns short			font alignment 0 Hanging 1 Centered 2 Roman 3 Variable 4 Auto
68	44	fVertical	short	:1	0001	
		fBackward	short	:1	0002	
		fRotateFont	short	:1	0004	
			short	:13	FFF8	reserved
70	46					reserved

72	48	fInTable	char			when 1, paragraph is contained in a table row
73	49	fTtp	char			when 1, paragraph consists only of the row mark special character and marks the end of a table row.
74	4A	wr	byte			Wrap Code for absolute objects
75	4B	fLocked	byte			when 1, paragraph may not be edited
76	4C	ptap	*TAP*			used internally by Word
80	50	dxaAbs	long			when positive, is the horizontal distance from the reference frame specified by pap.pcHorz. 0 means paragraph is positioned at the left with respect to the reference frame specified by pcHorz. Certain negative values have special meaning: -4 paragraph centered horizontally within reference frame -8 paragraph adjusted right within reference frame -12 paragraph placed immediately inside of reference frame -16 paragraph placed immediately outside of reference frame
84	54	dyaAbs	long			when positive, is the vertical distance from the reference frame specified by pap.pcVert. 0 means paragraph's y-position is unconstrained. Certain negative values have special meaning: -4 paragraph is placed at top of reference frame -8 paragraph is centered vertically within reference frame -12 paragraph is placed at bottom of reference frame.
88	58	dxaWidth	long			when not == 0, paragraph is constrained to be dxaWidth wide, independent of current margin or column settings.
92	5C	brcTop	BRC			specification for border above paragraph
96	60	brcLeft	BRC			specification for border to the left of paragraph
100	64	brcBottom	BRC			specification for border below paragraph
104	68	brcRight	BRC			specification for border to the right of paragraph
108	6C	brcBetween	BRC			specification of border to place between conforming paragraphs. Two paragraphs conform when both have borders, their brcLeft and brcRight matches, their widths are the same, they both belong to tables or both do not, and have the same absolute positioning props.
112	70	brcBar	BRC			specification of border to place on outside of text when facing pages are to be displayed.
116	74	dxaFromText	long			horizontal distance to be maintained between an absolutely positioned paragraph and any non-absolute positioned text
120	78	dyaFromText	long			vertical distance to be maintained between an absolutely positioned paragraph and any non-absolute positioned text
124	7C	dyaHeight	short	:15	7FFF	height of abs obj; 0 == Auto
		fMinHeight	short	:1	8000	0 = Exact, 1 = At Least
126	7E	shd	SHD			shading
128	80	dcs	DCS			drop cap specifier (see DCS definition)
130	82	lvl	char			
131	83	fNumRMIns	char			

132 216	84 D8	anld fPropRMark	ANLD short			autonumber list descriptor (see ANLD definition) when 1, properties have been changed with revision marking on
218	DA	ibstPropRMark	short			index to author IDs stored in hsttbfRMark. used when properties have been changed when revision marking was enabled
220	DC	dttmPropRMark	DTTM			Date/time at which properties of this were changed for this run of text by the author. (Only recorded when revision marking is on.)
224	E0	numrm	NUMRM			paragraph numbering revision mark data (see NUMRM)
352	160	itbdMac	short			number of tabs stops defined for paragraph. Must be >= 0 and <= 64.
354	162	rgdxaTab	short[itbdMax]			array of positions of itbdMac tab stops. itbdMax == 64
482	1E2	rgtbd	char[itbdMax]			array of itbdMac tab descriptors

cbPAP (count of bytes of PAP) is 610 (decimal), 262(hex)

The standard PAP is all zeros except:

fWidowControl	1
fMultLineSpace	1
dyaLine	240 twips
lvl	9

Paragraph Property Exceptions (PAPX)

The **PAPX** is stored within **FKPs** and within the **STSH**.

b10	b16	field	type	size	bitfield	comments
0	0	cb	byte			count of bytes of following data in PAPX. The first byte of a PAPX is a count of bytes when a PAPX is stored in a STSH. Count of bytes is used because only paragraph sprms are stored in a STSH PAPX.
0	0	cw	byte			count of words for this byte and the following data in PAPX. The first byte of a PAPX is a count of words when PAPX is stored in an FKP. If this value is 0, it is a 'pad' byte and the count is stored in the following byte, Count of words is used because PAPX in an FKP can contain paragraph and table sprms.
1	1	(cw)	byte			if previous byte is 0, this is the count of words of following data in PAPX (not including this and previous 'pad' byte)
1/2	1/2	istd	uns short			index to style descriptor of the style from which the paragraph inherits its paragraph and character properties
3/4	3/4	grpprl	character array			a list of the sprms that encode the differences between PAP for a paragraph and the PAP for the style used. When a paragraph bound is also the end of a table row, the PAPX also contains a list of table sprms which express the difference of table row's TAP from an empty TAP that has been cleared to zeros. The table sprms are recorded in the list after all of the paragraph sprms. See Sprms definitions for list of sprms that are used in PAPXs.

For calculating papx.cw when storing in a FKP: For even-sized grpprl's, the grpprl plus the istd and cw bytes will be an even
number of bytes, so we store the count of words for all three elements in papx.cw. For odd-sized grpprl's, the three elements will

be an odd number of bytes, which can't be represented with a count of words; so, we store a 'pad' byte of 0 at the beginning (in the normal cw location), followed by a count that is the size of the grpprl and istd byte only (since that's an even number of bytes). In either case, papx.cw is immediately followed by the istd and grpprl.

Picture Descriptor (on File) (PICF)

b10	b16	field	type	size	bitfield	comments
0	0	lcb	long			number of bytes in the PIC structure plus size of following picture data which may be a Window's metafile, a bitmap, or the filename of a TIFF file. In the case of a Macintosh PICT picture, this includes the size of the PIC, the standard "x" metafile, and the Macintosh PICT data. See Appendix B for more information.
4	4	cbHeader	unsigned			number of bytes in the PIC (to allow for future expansion).
6	6	mfp.mm	short			
8	8	mfp.xExt	short			
10	A	mfp.yExt	short			
12	C	mfp.hMF	short			

If a Windows metafile is stored immediately following the PIC structure, the mfp is a Window's METAFILEPICT structure. When the data immediately following the PIC is a TIFF filename, mfp.mm == 98 If a bitmap is stored after the pic, mfp.mm == 99

When the PIC describes a bitmap, mfp.xExt is the width of the bitmap in pixels and mfp.yExt is the height of the bitmap in pixels..

b10	b16	field	type	size	bitfield	comments
14	E	bm	BITMAP (14 bytes)			Window's bitmap structure when PIC describes a BITMAP.
14	E	rcWinMF	rc (rectangle - 8 bytes)			rect for window origin and extents when metafile is stored -- ignored if 0
28	1C	dxaGoal	short			horizontal measurement in twips of the rectangle the picture should be imaged within.
30	1E	dyaGoal	short			vertical measurement in twips of the rectangle the picture should be imaged within.

when scaling bitmaps, dxaGoal and dyaGoal may be ignored if the operation would cause the bitmap to shrink or grow by a non-power-of-two factor

b10	b16	field	type	size	bitfield	comments
32	20	mx	uns short			horizontal scaling factor supplied by user expressed in .001% units.
34	22	my	uns short			vertical scaling factor supplied by user expressed in .001% units.

for all of the Crop values, a positive measurement means the specified border has been moved inward from its original setting and a negative measurement means the border has been moved outward from its original setting.

b10	b16	field	type	size	bitfield	comments
36	24	dxaCropLeft	short			the amount the picture has been cropped on the left in twips.
38	26	dyaCropTop	short			the amount the picture has been cropped on the top in twips.
40	28	dxaCropRight	short			the amount the picture has been cropped on the right in twips.
42	2A	dyaCropBottom	short			the amount the picture has been cropped on the bottom in twips.
44	2C	brcl	short	:4	000F	Obsolete, superseded by brcTop, etc. In WinWord 1.x, it was the type of border to place around picture 0 single 1 thick 2 double 3 shadow
		fFrameEmpty	short	:1	0010	picture consists of a single frame
		fBitmap	short	:1	0020	==1, when picture is just a bitmap
		fDrawHatch	short	:1	0040	==1, when picture is an active OLE object
		fError	short	:1	0080	==1, when picture is just an error message
		bpp	short	:8		bits per pixel 0 unknown 1 monochrome 4 VGA
46	2E	brcTop	BRC			specification for border above picture
50	32	brcLeft	BRC			specification for border to the left of picture
54	36	brcBottom	BRC			specification for border below picture
58	3A	brcRight	BRC			specification for border to the right of picture
62	3E	dxaOrigin	short			horizontal offset of hand annotation origin
64	40	dyaOrigin	short			vertical offset of hand annotation origin
66	42	cProps	short			unused
68	44	rgb				variable array of bytes containing Window's metafile, bitmap or TIFF file filename.

Piece Descriptor (PCD)

b10	b16	field	type	size	bitfield	comment
0	0	fNoParaLast	short	:1	0001	when 1, means that piece contains no end of paragraph marks.
		fPaphNil	short	:1	0002	used internally by Word
		fCopied	short	:1	0004	used internally by Word
		*	short	:5		
1	1	fn	short	:8	FF00	used internally by Word
2	2	fc	FC			file offset of beginning of piece. The size of the ith piece can be determined by

in **Complex File Format** topic).

b10	b16	field	type	size	bitfield	comment
0	0	fComplex	short	:1	0001	set to 1 for variant 2
		igrpprl	short	:15	FFFE	index to a grpprl stored in CLX portion of file.

cbPRM (count of bytes of PRM) is 2.

Routing Slip (RS)

b10	b16	field	type	size	bitfield	comments
0	0	fRouted	short			when 1, document has been routed to at least one recipient.
2	2	fReturnOrig	short			when 1, document should be routed to the originator after it has been routed to all recipients.
4	4	fTrackStatus	short			when 1, a status message is sent to the originator each time the document is forwarded to a recipient on the routing list.
6	6	fDirty	short			unused(should be 0)
8	8	nProtect	short			document protection while routing 0 recipients can make changes to the document and all changes are untracked. 1 recipients can add annotations and make changes to the document. Any changes are tracked by revision marks, and revision marking cannot be turned off. 2 recipients can only add annotations to the document. 3 recipients can enter information only in form fields.
10	A	iStage	short			index of the current recipient.
12	C	delOption	short			when 0, document is routed to each recipient in turn. when 1, document is routed to all recipients simultaneously.
14	E	cRecip	short			count of recipients.

cbRS (count of bytes of RS) is 16 (decimal), 10 (hex).

Routing Recipient (RR)

b10	b16	field	type	size	bitfield	comments
0	0	cb	short			count of bytes of private system data
2	2	cbSzRecip	short			count of bytes in recipient string (including null terminator).

cbRR (count of bytes of RR) is 4.

Section Descriptor (SED)

b10	b16	field	type	size	bitfield	comments
0	0	fn	short			used internally by Word
2	2	fcSepx	FC			file offset in main stream to beginning of SEPX stored for section. If sed.fcSepx ==

					0xFFFFFFFF, the section properties for the section are equal to the standard SEP (see SEP definition).
6	6	fnMpr	short		used internally by Word
8	8	fcMpr	FC		points to offset in FC space of main stream where the Macintosh Print Record for a document created on a Mac will be stored

cbSED (count of bytes of SED) is 12 (decimal), C (hex).

Section Properties (SEP)

b10	b16	field	type	size	bitfield	comments
0	0	bkc	uns char			break code: 0 No break 1 New column 2 New page 3 Even page 4 Odd page
1	1	fTitlePage	uns char			set to 1 when a title page is to be displayed
2	2	fAutoPgn	char			only for Mac compatibility, used only during open, when 1, sep.dxaPgn and sep.dyaPgn are valid page number locations
3	3	nfcPgn	uns char			page number format code: 0 Arabic 1 Roman (upper case) 2 Roman (lower case) 3 Letter (upper case) 4 Letter (lower case)
4	4	fUnlocked	uns char			set to 1, when a section in a locked document is unlocked
5	5	cnsPgn	uns char			chapter number separator for page numbers
6	6	fPgnRestart	uns char			set to 1 when page numbering should be restarted at the beginning of this section
7	7	fEndNote	uns char			when 1, footnotes placed at end of section. When 0, footnotes are placed at bottom of page.
8	8	lnc	char			line numbering code: 0 Per page 1 Restart 2 Continue
9	9	grpflhdt	char			specification of which headers and footers are included in this section. See explanation in Headers and Footers topic. No longer used.
10	A	nLnnMod	uns short			if 0, no line numbering, otherwise this is the line number modulus (e.g. if nLnnMod is 5, line numbers appear on line 5, 10, etc.)
12	C	dxaLnn	long			distance of

16	10	dxapgn	short		when fAutoPgn ==1, gives the x position of auto page number on page in twips (for Mac compatibility only)
18	12	dypgn	short		when fAutoPgn ==1, gives the y position of auto page number on page in twips (for Mac compatibility only)
20	14	fLBetween	char		when ==1, draw vertical lines between columns
21	15	vjc	char		vertical justification code 0 top justified 1 centered 2 fully justified vertically
	3	bottom justified			
22	16	dmBinFirst	uns short		bin number supplied from windows printer driver indicating which bin the first page of section will be printed.
24	18	dmBinOther	uns short		bin number supplied from windows printer driver indicating which bin the pages other than the first page of section will be printed.
26	1A	dmPaperReq	uns short		dmPaper code for form selected by user
28	1C	brcTop	BRC		top page border
32	20	brcLeft	BRC		left page border
36	24	brcBottom	BRC		bottom page border
40	28	brcRight	BRC		right page border
44	2C	fPropRMark	short		when 1, properties have been changed with revision marking on
46	2E	ibstPropRMark	short		index to author IDs stored in hsttbFRMark. used when properties have been changed when revision marking was enabled
48	30	dtmPropRMark	DTTM		Date/time at which properties of this were changed for this run of text by the author. (Only recorded when revision marking is on.)
52	34	dxtCharSpace	long		
56	38	dyaLinePitch	long		
60	3C	clm	uns short		
	62	3E	short		reserved
64	40	dmOrientPage	uns char		orientation of pages in that section. set to 0 when portrait, 1 when landscape
65	41	iHeadingPgn	uns char		heading number level for page number
66	42	pgnStart	uns short		user specified starting page number.
68	44	lnnMin	short		beginning line number for section
70	46	wTextFlow	uns short		

72	48		short			reserved
74	4A	pgbProp	short			page border properties
74	4A	pgbApplyTo	short	:3	0007	page border applies to: 0 all pages in this section 1 first page in this section 2 all pages in this section but first 3 whole document (all sections)
		pgbPageDepth	short	:2	0018	page border depth: 0 in front 1 in back
		pgbOffsetFrom	short	:3	00E0	page border offset from: 0 offset from text 1 offset from edge of page
			short	:8	FF00	reserved
76	4C	xaPage	uns long			default value is 12240 twipswidth of page
80	50	yaPage	uns long			default value is 15840 twipsheight of page
84	54	xaPageNUp	uns long			used internally by Word
88	58	yaPageNUp	uns long			used internally by Word
92	5C	dxaLeft	uns long			default value is 1800 twipsleft margin
96	60	dxaRight	uns long			default value is 1800 twipsright margin
100	64	dyaTop	long			default value is 1440 twipstop margin
104	68	dyaBottom	long			default value is 1440 twipsbottom margin
108	6C	dzaGutter	uns long			default value is 0 twips gutter width
112	70	dyaHdrTop	uns long			y position of top header measured from top edge of page.
116	74	dyaHdrBottom	uns long			y position of bottom header measured from top edge of page.
120	78	ccolM1	short			number of columns in section - 1.
122	7A	fEvenlySpaced	char			when == 1, columns are evenly spaced. Default value is 1.
123	7B		char			reserved
124	7C	dxaColumns	long			distance that will be maintained between columns
128	80	rgdxaColumnWidthSpacing array of XA				array of 89 longs that determine bounds of irregular width columns
484	1E4	dxaColumnWidth	long			used internally by Word
488	1E8	dmOrientFirst	uns			

b10	b16	field	type	size	bitfield	comments
0	0	icoFore	short	:5	001F	foreground color (see chp.ico)
		icoBack	short	:5	03E0	background color (see chp.ico)
		ipat	short	:6	FC00	shading pattern (see ipat table below)

ipat	pattern
0	Automatic
1	Solid
2	5 Percent
3	10 Percent
4	20 Percent
5	25 Percent
6	30 Percent
7	40 Percent
8	50 Percent
9	60 Percent
10	70 Percent
11	75 Percent
12	80 Percent
13	90 Percent
14	Dark Horizontal
15	Dark Vertical
16	Dark Forward Diagonal
17	Dark Backward Diagonal
18	Dark Cross
19	Dark Diagonal Cross
20	Horizontal
21	Vertical
22	Forward Diagonal
23	Backward Diagonal
24	Cross
25	Diagonal Cross
35	2.5 Percent
36	7.5 Percent
37	12.5 Percent
38	15 Percent

39	17.5 Percent
40	22.5 Percent
41	27.5 Percent
42	32.5 Percent
43	35 Percent
44	37.5 Percent
45	42.5 Percent
46	45 Percent
47	47.5 Percent
48	52.5 Percent
49	55 Percent
50	57.5 Percent
51	62.5 Percent
52	65 Percent
53	67.5 Percent
54	72.5 Percent
55	77.5 Percent
56	82.5 Percent
57	85 Percent
58	87.5 Percent
59	92.5 Percent
60	95 Percent
61	97.5 Percent
62	97 Percent

cbSHD (count of bytes of SHD) is 2.

Tab Descriptor (TBD)

The **TBD** is a substructure of the **PAP**.

b10	b16	field	type	size	bitfield	comments
0	0	jc	byte	:3	07	justification code 0 left tab 1 centered tab 2 right tab 3 decimal tab 4 bar
		tlc	byte	:3	38	tab leader code 0 no leader

						1 dotted leader 2 hyphenated leader 3 single line leader
		*	byte	:2	C0	4 heavy line leader reserved

cbTBD (count of bytes of TBD) is 1.

Table Cell Descriptors (TC)

The **TC** is a substructure of the **TAP**.

b10	b16	field	type	size	bitfield	comments
0	0	rgf	short	0	0	
		fFirstMerged	short	:1	0001	set to 1 when cell is first cell of a range of cells that have been merged. When a cell is merged, the display areas of the merged cells are consolidated and the text within the cells is interpreted as belonging to one text stream for purposes of calculating line breaks.
		fMerged	short	:1	0002	set to 1 when cell has been merged with preceding cell.
		fVertical	short	:1	0004	set to 1 when cell has vertical text flow
		fBackward	short	:1	0008	for a vertical table cell, text flow is bottom to top when 1 and is bottom to top when 0.
		fRotateFont	short	:1	0010	set to 1 when cell has rotated characters (i.e. uses @font)
		fVertMerge	short	:1	0020	set to 1 when cell is vertically merged with the cell(s) above and/or below. When cells are vertically merged, the display area of the merged cells are consolidated. The consolidated area is used to display the contents of the first vertically merged cell (the cell with fVertRestart set to 1), and all other vertically merged cells (those with fVertRestart set to 0) must be empty. Cells can only be merged vertically if their left and right boundaries are (nearly) identical (i.e. if corresponding entries in rgdxaCenter of the table rows differ by at most 3).
		fVertRestart	short	:1	0040	set to 1 when the cell is the first of a set of vertically merged cells. The contents of a cell with fVertStart set to 1 are displayed in the consolidated area belonging to the entire set of vertically merged cells. Vertically merged cells with fVertRestart set to 0 must be empty.
		vertAlign	short	:2	0180	specifies the alignment of the cell contents relative to text flow (e.g. in a cell with bottom to top text flow and bottom vertical alignment, the text is shifted horizontally to match the cell's right boundary): 0 top 1 center 2 bottom
		fUnused	short	:7	FE00	reserved
2	2	wUnused	uns short			reserved
4	4	rgbrc	BRC[cbrcTc]			notational convenience for referring to brcTop, brcLeft, etc. fields.
4	4	brcTop	BRC			specification of the top border of a table cell
8	8	brcLeft	BRC			specification of left border of table row

12	12	brcBottom	BRC			specification of bottom border of table row
16	16	brcRight	BRC			specification of right border of table row.

cbTC (count of bytes of TC) is 20(decimal), 14(hex).

Table Autoformat Look sPecifier (TLP)

b10	b16	field	type	size	bitfield	comments
0	0	itl	short			index to Word's table of table looks (see itl table below)
2	2	fBorders	short	:1	0001	when ==1, use the border properties from the selected table look
		fShading	short	:1	0002	when ==1, use the shading properties from the selected table look
		fFont	short	:1	0004	when ==1, use the font from the selected table look
		fColor	short	:1	0008	when ==1, use the color from the selected table look
		fBestFit	short	:1	0010	when ==1, do best fit from the selected table look
		fHdrRows	short	:1	0020	when ==1, apply properties from the selected table look to the header rows in the table
		fLastRow	short	:1	0040	when ==1, apply properties from the selected table look to the last row in the table
		fHdrCols	short	:1	0080	when ==1, apply properties from the selected table look to the header columns of the table
		fLastCol	short	:1	0100	when ==1, apply properties from the selected table look to the last column of the table

itl	table look
0	(none)
1	Simple 1
2	Simple 2
3	Simple 3
4	Classic 1
5	Classic 2
6	Classic 3
7	Classic 4
8	Colorful 1
9	Colorful 2
10	Colorful 3
11	Columns 1
12	Columns 2
13	Columns 3
14	Columns 4

15	Columns 5
16	Grid 1
17	Grid 2
18	Grid 3
19	Grid 4
20	Grid 5
21	Grid 6
22	Grid 7
23	Grid 8
24	List 1
25	List 2
26	List 3
27	List 4
28	List 5
29	List 6
30	List 7
31	List 8
32	3D Effects 1
33	3D Effects 2
34	3D Effects 3
35	Contemporary
36	Elegant
37	Professional
38	Subtle1
39	Subtle2

cbTLP (count of bytes of TLP) is 4.

Table Properties (TAP)

b10	b16	field	type	size	bitfield	comments
0	0	jc	short			justification code. specifies how table row should be justified within its column. 0 left justify 1 center 2 right justify
2	2	dxapGapHalf	long			measures half of the white space that will be maintained between text in adjacent columns of a table row. A dxapGapHalf width of white space will be maintained on both sides of a column boundary.

6	6	dyaRowHeight	long			when greater than 0, guarantees that the height of the table will be at least dyaRowHeight high. When less than 0, guarantees that the height of the table will be exactly absolute value of dyaRowHeight high. When 0, table will be given a height large enough to represent all of the text in all of the cells of the table. Cells with vertical text flow make no contribution to the computation of the height of rows with auto or at least height. Neither do vertically merged cells, except in the last row of the vertical merge. If an auto height row consists entirely of cells which have vertical text direction or are vertically merged, and the row does not contain the last cell in any vertical cell merge, then the row is given height equal to that of the end of cell mark in the first cell.
10	A6	fCantSplit	uns char			when 1, table row may not be split across page bounds
11	B	fTableHeader	uns char			when 1, table row is to be used as the header of the table
12	C	tlp	TLP			table look specifier (see TLP definition)
16	10	lwHTMLProps	long			reserved for future use
20	14	fCaFull	short	:1	0001	used internally by Word
		fFirstRow	short	:1	0002	used internally by Word
		fLastRow	short	:1	0004	used internally by Word
		fOutline	short	:1	0008	used internally by Word
		*	short	:12	FFE0	reserved
22	16	itcMac	short			count of cells defined for this row. ItcMac must be ≥ 0 and less than or equal to 64.
24	18	dxaAdjust	long			used internally by Word
28	1C	dxaScale	long			used internally by Word
32	20	dxsInch	int			used internally by Word
36	24	rgdxaCenter	short[itcMax + 1]			rgdxaCenter[0] is the left boundary of cell 0 measured relative to margin.. rgdxaCenter[$\text{tap.itcMac} - 1$] is left boundary of last cell. rgdxaCenter[tap.itcMac] is right boundary of last cell.
166	A6	rgdxaCenterPrint	short[itcMax + 1]			used internally by Word
296	128	rgtc	TC[itcMax]			array of table cell descriptors
1576	628	rgshd	SHD[itcMax]			array of cell shades
1704	6A81D4	rgbrcTable	BRC[6]			array of border defaults for cells

cbTAP (count of bytes of TAP) is 1728 (decimal), 6C0(hex).

TeXtBoX Story (FTXBXS)

b10	b16	field	type	size	bitfield	comments

0	0	cTxbx	long			when not fReusable, counts the number of textboxes in this story chain
0	0	iNextReuse	long			when fReusable, the index of the next in the linked list of reusable FTXBXSs
4	4	cReusable	long			if fReusable, counts the number of reusable FTXBXSs follow this one in the linked list
8	8	fReusable	short			this FTXBXS is not currently in use
10	A		long			reserved
14	E	lid	long			Shape Identifier (see FSPA) for first Office Shape in textbox chain.
18	12	txidUndo	long			

cbFTXBXS (count of bytes of FTXBXS) is 22 (decimal), 16 (hex).

Work Book (WKB)

b10	b16	field	type	size	bitfield	comments
0	0	fn	short			
2	2	grfwkb	uns short			
4	4	lvl	short			
6	6	fnpt	short	:4	000F	
		fnpd	short	:12	FFF0	
8	8	doc	long			unused

cbWKB (count of bytes of WKB) is 12 (decimal), C (hex).

Appendix A - Reading a Macintosh PICT Graphic

As described under "picture" in the Definition section of this document, some pictures in Word documents are stored as Macintosh PICT graphics, particularly in files created by Word for the Macintosh. All pictures, including these, are stored as a block of binary data attached to a special chPic character in the text stream. This block always begins with a PIC structure. (Please see the "picture" definition mentioned above for more information on general picture-reading.)

Normal graphics follow the PIC structure with a single Office shape, Windows metafile, bitmap, or TIFF representation, as described in the "picture" definition section. Macintosh PICT graphics have a standard, unchanging Windows metafile after the PIC which always depicts an "x", followed by the actual Macintosh PICT picture. This is for backward-compatibility with older readers, which expect to find a Windows metafile after the PIC structure. These readers will simply display the fixed "x" image. In the Macintosh PICT case, the PIC structure's lcb field represents the size of the entire picture data block, including the PIC itself, the "x" metafile and the Macintosh PICT data. (see the description of the PIC structure in the Structure Definitions section of this document.)

To distinguish between normal and Macintosh PICT graphics, a reader needs to detect the presence of the special "x" metafile. The bytes below are in an early portion of the "x" metafile.

```
unsigned char rgbWmfXBegin[] =
{
    '\x14', '\x00', '\x00', '\x00', '\x26', '\x06', '\x0F', '\x00', '\x1E', '\x00',
    '\xFF', '\xFF', '\xFF', '\xFF', '\x04', '\x00', '\x14', '\x00', '\x00', '\x00',
    '\x57', '\x6F', '\x72', '\x64', '\x0E', '\x00', '\x4D', '\x69', '\x63', '\x72',
    '\x6F', '\x73', '\x6F', '\x66', '\x74', '\x20', '\x57', '\x6F', '\x72', '\x64',
    '\x0E', '\x00', '\x00', '\x00', '\x26', '\x06', '\x0F', '\x00', '\x12', '\x00',
```

```
'\x57', '\x6F', '\x72', '\x64', '\xFF', '\xFF', '\x08', '\x00', '\x00', '\x00'
```

```
/* "x" wmf and PICT data sizes immediately follow as 2 four-byte longs */
};
```

```
#define cbMETAHDR 18 // size of a standard Windows metafile header
#define cbWmfXBegin 60 // length of this beginning section of the x metafile
```

After reading the PIC structure from the picture data block, the reader should skip cbMETAHDR bytes (the size of a standard Windows metafile header). It should then compare the next cbWmfXBegin bytes in the picture data block against the bytes in the rgbWmfXBegin array above. If they do not match, the picture is a normal picture -- Windows metafile, bitmap or TIFF.

If they *do* match, then the reader should read the next 8 bytes in the picture data block as two 4-byte "long"s (Intel 80x86 byte order). These numbers are the sizes (in bytes) of the "x" metafile and the Macintosh PICT data, respectively. The size of the "x" metafile is measured from its start immediately after the PIC structure. It is possible for the PICT's size to be zero. In this case, there is no PICT data, and the reader may use the "x" Windows metafile as the picture's representation.

Appendix B - Calculation of font (FTC) and language (LID)

Certain Unicode characters are shared between Far East and non-Far East scripts requiring the calculation of font and language based on the Unicode character code and the chp.idctHint property.

Characters are classified into one of four groups, ASCII, Far East, floating, and non-Far East. Properties are calculated as follows:

Character type	Font (ftc)	Language (lid)
ASCII	sprmCRgftc0	sprmCRglid0
non-Far East	sprmCRgftc2	sprmCRglid0
Far East	sprmCRgftc1	sprmCRglid1
shared character	sprmCRgftc2 if chp.idctHint is 0 sprmCRgftc1 if chp.idctHint is 1	sprmCRglid0 if chp.idctHint is 0 sprmCRglid1 if chp.idctHint is 1

The table below defines the classification of various ranges of Unicode characters:

Unicode subrange	character range	Classification
usrBasicLatin	0x20 -> 0x7f	ASCII
usrLatin1	0xa0 -> 0xff	some shared (see notes below)
usrLatinXA	0x100 -> 0x17f	some shared (see notes below)
usrLatinXB	0x180 -> 0x24f	some shared (see notes below)
usrIPAExtensions	0x250 -> 0x2af	some shared (see notes below)
usrSpacingModLetters	0x2b0 -> 0x2ff	shared
usrCombDiacritical	0x300 -> 0x36f	shared
usrBasicGreek	0x370 -> 0x3cf	shared
usrGreekSymbolsCop	0x3d0 -> 0x3ff	non-Far East
usrCyrillic	0x400 -> 0x4ff	shared
usrArmenian	0x500 -> 0x58f	non-Far East

usrBasicHebrew	0x5d0 -> 0x5ff	non-Far East
usrHebrewXA	0x590 -> 0x5cf	non-Far East
usrBasicArabic	0x600 -> 0x652	non-Far East
usrArabicX	0x653 -> 0x6ff	non-Far East
usrDevangari	0x900 -> 0x97f	non-Far East
usrBengali	0x980 -> 0x9ff	non-Far East
usrGurmukhi	0xa00 -> 0xa7f	non-Far East
usrGujarati	0xa80 -> 0xaff	non-Far East
usrOriya	0xb00 -> 0xb7f	non-Far East
usrTamil	0x0b80 -> 0x0bff	non-Far East
usrTelugu	0x0c00 -> 0x0c7f	non-Far East
usrKannada	0x0c80 -> 0x0cff	non-Far East
usrMalayalam	0x0d00 -> 0x0d7f	non-Far East
usrThai	0x0e00 -> 0x0e7f	non-Far East
usrLao	0x0e80 -> 0x0eff	non-Far East
usrBasicGeorgian	0x10d0 -> 0x10ff	non-Far East
usrGeorgianExtended	0x10a0 -> 0x10cf	non-Far East
usrHangulJamo	0x1100 -> 0x11ff	non-Far East
usrLatinExtendedAdd	0x1e00 -> 0x1eff	shared
usrGreekExtended	0x1f00 -> 0x1fff	non-Far East
usrGeneralPunct	0x2000 -> 0x206f	shared
usrSuperAndSubscript	0x2070 -> 0x209f	shared
usrCurrencySymbols	0x20a0 -> 0x20cf	shared
usrCombDiacriticsS	0x20d0 -> 0x20ff	shared
usrLetterlikeSymbols	0x2100 -> 0x214f	shared
usrNumberForms	0x2150 -> 0x218f	shared
usrArrows	0x2190 -> 0x21ff	shared
usrMathematicalOps	0x2200 -> 0x22ff	shared
usrMiscTechnical	0x2300 -> 0x23ff	shared
usrControlPictures	0x2400 -> 0x243f	shared
usrOpticalCharRecog	0x2440 -> 0x245f	shared
usrEnclosedAlphanum	0x2460 -> 0x24ff	shared
usrBoxDrawing	0x2500 -> 0x257f	shared
usrBlockElements	0x2580 -> 0x259f	shared
usrGeometricShapes	0x25a0 -> 0x25ff	shared
usrMiscDingbats	0x2600 -> 0x26ff	shared

usrDingbats	0x2700 -> 0x27bf	shared
usrCJKSymAndPunct	0x3000 -> 0x303f	Far East
usrHiragana	0x3040 -> 0x309f	Far East
usrKatakana	0x30a0 -> 0x30ff	Far East
usrBopomofo	0x3100 -> 0x312f	Far East
usrHangulCompatJamo	0x3130 -> 0x318f	Far East
usrCJKMisc	0x3190 -> 0x319f	Far East
usrEnclosedCJkLtMnth	0x3200 -> 0x32ff	Far East
usrCJKCompatibility	0x3300 -> 0x33ff	Far East
usrHangul	0xac00 -> 0xd7a3	Far East
usrReserved1		
usrReserved2		
usrCJKUnifiedIdeo	0x4e00 -> 0x9fff	Far East
usrPrivateUseArea	0xe000 -> 0xf8ff	shared
usrCJKCompatibilityIdeographs	0xf900 -> 0xfaff	Far East
usrAlphaPresentationForms	0xfb00 -> 0xfb4f	shared
usrArabicPresentationFormsA	0xfb50 -> 0xfdff	shared
usrCombiningHalfMarks	0xfe20 -> 0xfe2f	Far East
usrCJKCompatForms	0xfe30 -> 0xfe4f	Far East
usrSmallFormVariants	0xfe50 -> 0xfe6f	Far East
usrArabicPresentationFormsB	0xfe70 -> 0xfefe	shared
usrHFWidthForms	0xff00 -> 0xffef	Far East
usrSpecials	0xfff0 -> 0xffff	non-Far East

The table below describes the behavior of the unicode subrange usrLatin1. Shared characters are marked in this table with a 1, while characters marked with a 0 are considered "non-Far East". All other characters in this unicode subrange are considered "non-Far East".

```
// 0 1 2 3 4 5 6 7 8 9 a b c d e f
   0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, // 0x00a0-0x00af
   1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, // 0x00b0-0x00bf
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // 0x00c0-0x00cf
   0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, // 0x00d0-0x00df
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // 0x00e0-0x00ef
   0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, // 0x00f0-0x00ff
};
```

The table below describes the behavior of the unicode range usrLatinXA. Shared characters are marked in this table with a 1, while characters marked with a 0 are considered "non-Far East". All other characters in this unicode subrange are considered "non-Far East".

```
// 0 1 2 3 4 5 6 7 8 9 a b c d e f
   1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // 0x0100-0x010f
   0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, // 0x0110-0x011f
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, // 0x0120-0x012f
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // 0x0130-0x013f
0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, // 0x0140-0x014f
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // 0x0150-0x015f
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, // 0x0160-0x016f
```

In `usrLatinXB` shared characters are `0x192`, `0x1FA`, `0x1FB`, `0x1FC`, `0x1FD`, `0x1FE` and `0x1FF`. All other characters in this unicode subrange are considered "non-Far East".

In `usrIPAExtensions` shared characters are `0x251`, and `0x261`.

An optimization is available. If the Far East font `chp.ftcFE` is 0 and `chp.idctHint` is 0 and `chp.ftcAscii` is equal to `chp.ftcOther`, the font is `chp.ftcAscii` and the language is `chp.lidDefault`.

