Microsoft Word 6.0 Binary File Format

REVISION HISTORY

12/02/93	Updated structures and sprm table for Windows Word 6.0 format
10/25/91	Reformatted document, removed revision marks and completed the summary of changes from Windows Word 1.x to 2.0 format.
5/10/91	Updated structures and sprm table for Windows Word 2.0 format.
1/23/90	Corrected offsets with the definition of the FIB
6/16/89	Updated structure definitions
1/9/89	Document Created

TABLE OF CONTENTS

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

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 Words binary data.

summary information stream

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

object stream

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

CP (**C**haracter **P**osition):

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

FC(**F**ile **C**haracter position):

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

PLCF(**PL**ex of **C**ps(or F**C**s) stored in **F**ile):

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 CP**s or **FC**s 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** (**PL**ex of **C**ps containing **P**ie**c**e **D**escriptors).The piece table relates a logical character number, called a **CP** (**C**haracter **P**osition), to a physical location within a Word file (an **FC**). The array of **CP**s in the **plcfpcd** defines a partitioning of the Word document into disjoint pieces. The second array is an array of **PCD**s (**P**ie**c**e **D**escriptors) which is in 1-to-1 correspondence to the array of **CP**s 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 **CP**s that is less than the character **CP**. Then reference the **PCD** with that index in the array of **PCD**s. 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 the actual file offset of the character.

sprm (**S**ingle **PR**operty **M**odifier):

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. The operand is omitted for **sprms** whose opcodes completely specify the values that must be stored in the property data structure. A synonym used for **sprm** in some data structure definitions is **prl** (**pr**operty modifiers stored in a **l**ist).

grpprl (group of prls):

A grpprl is a data structure that records a set of **sprm**s. The 0th **sprm** is recorded at offset 0 of the structure. Any succeeding **sprm**s are recorded immediately after the end of the preceding **sprm** . To traverse a **grpprl** 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 grpprl** (or **papx** or **sepx)**" used later in this document means to fetch the 0th **sprm** recorded in the **grpprl** and perform the action for that sprm, fetch the first **sprm** and perform its action, and continue this procedure until all **sprms** in the **grpprl** (or **papx** or **sepx)** have been processed.

prm (**PR**operty **M**odifier):

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 **grpprl** which records the user's formatting changes as a group of **sprm**s. 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. 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.

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** (**c**ount of **CP**s) to record how many characters are stored in the text stream. When a file is stored in non-complex format, the **fc** and **ccp** allow an initial piece table to be constructed when the file is read.

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.

FIB (**F**ile **I**nformation **B**lock):

The header of a Windows 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 (**CH**aracter **P**roperties)

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

CHPX (**Ch**aracter **P**roperty E**X**ception)

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 (**PA**ragraph **P**roperties)

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

PAPX (**PA**ragraph **P**roperty E**X**ception)

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 **grpprl** 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 **ISTD**s (**I**ndexes to **ST**yle **D**escriptors). 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. **CHPX**s recorded for the text within the paragraph and **PAPX**s 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 **FC**s 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 **FC**s that locate the properties of the run that begins at a particular **FC.**

In **paragraph FKP**s an array of **BX** structures follows the array of **FC**s in one to one correspondence with the array of **FC**s. 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 **CHPX**s if the **FKP** stores character properties or a group of **PAPX**s if the **FKP** stores paragraph and table properties.

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 **FC**s 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 **Bx**s (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 **FC**s in the Word file. In Word files, a **PLC**, the **plcfbte** (**PL**ex of F**C**s containing **B**in **T**able **E**ntries) is maintained. It records the association between a particular range of **FC**s and the **PN** (**P**age **N**umber) 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 **FKP**s recorded in ascending **FC** order, as are all of the **PAPX FKPS**. In a non-complex document, at least the first **FKP** page number will be recorded so that the beginning of the consecutive range of pages may be located. However, the bin table may be incomplete because of resource constraints placed on Word's save procedures.

If a **plcfbte** is incomplete, the page numbers of the first **n FKP**s will be recorded but the last **m FKPs** would not be represented. The complete **plcfbte** may be reconstructed by the reader because the total number of **CHPX FKPs** and **PAPX FKPs** is recorded in the **FIB.** When a reader notices that the number of entries in a **plcfbte** is less than the number of FKP pages that was recorded in the **FIB**, the reader must locate the last PN recorded in the **plcfbte**, call it pnLast. If the number of missing page entries is **m**, the reader would have to read pages pnLast+1 through pnLast+m and record the first **fc** stored in each of the tables plus the last **fc** of page pnLast+1 to produce a complete **plcfbte**.

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 **SEPX**s stored in a Word file. The array of **CP**s in the **plcfsed** records the boundaries of sections in the Word document . The second array in the **plcf**, an array of **SED**s (**SE**ction **D**escriptors), is in 1-to-1 correspondence to the array of **CP**s. 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 **CP**s in the **PLCSED** 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 84 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. 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 (ie. 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. Similarly, the **BKL** structure stored in the **plcbkl** consists of a single index which specifies which **plcbkf** marks the beginning of the bookmark.

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. For Windows Word, a picture may be a Window's metafile, a bitmap or a reference to a TIFF file**.** Beginning at the position recorded in chp.fcPic, a header data structure, the PIC, will be stored. If the picture is a Window's metafile or a bitmap, the metafile or bitmap will immediately follow the PIC. If the picture is a TIFF file, the filename of the TIFF file will be recorded immediately following the PIC.

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 seperator 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.

drawing object

REVIEW Dave

A drawing 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 drawing objects. The native data for the drawing object my be obtained by taking the CP for the special character and using this to find the corresponding entry in the **plcfdoa**. An entry in this plc consists of an FC pointing to the **DO** structure and a **ctxbx**, which is the count of text boxes in the drawing object. Text for the textboxes is stored separately in the textbox subdocument of the main or header document. The textbox subdocument contains a **plctxbx** where the text from CP n to CP n+1 in the subdocument is the text which is contained in the n^{th} textbox of the superior document. Ordering of textboxes is based upon CP order of the **DOs** in the superior document, and order of the textboxes within the **DO** itself. For example, if a document contains 1 **DO** at CP 500 which contains 3 textboxes and a **DO** at CP 600 which contains 10 textboxes, then the text for the 4th textbox in the second **DO** would be stored at the CP specified by the 6th entry in the **plctxbx**.

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 (*fTrue*, *TRUE*) or 0 (*fFalse*, *FALSE*). (eg. fWidowControl, fShadow)
l	used to name a 4 byte integer value (a long). (eg. 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. (eg. 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.

dxa used to name a variable that contains the horizontal distance of an object measured from some reference point expressed in twips. (eg. 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. (eg. pap.dyaAbs is the vertical distance of the top of a paragraph from a reference frame declared in the pap).

dxp 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"). (eg. dxpSpace)

dyp 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. (eg.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. (eg. 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. (eg. 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. (eg. a grpprl 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. (eg. grpfIhdt 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 (ie. 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.

WORD AND DOCFILES

Word 6.0 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. To access data within a Word binary file, the file must be opened using the OLE 2.0 docfile APIs.

A word docfile consists of a main stream, a summary information 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 streams contain binary data for embedded objects. Word has no knowledge of the contents of these streams; this information is accessed and manipulated though the OLE 2.0 APIs. The main stream of the Word docfile contains all other binary data. The majority of this document describes the contents of the main stream.

FORMAT OF THE SUMMARY INFO STREAM IN A WORD FILE

Summary information is stored with the stream named ”SummaryInformation”. This summary information consists of the following elements:

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.

group of SEPXs
SEPXs immediately follow the text and are concatenated one after the other. A SEPX may not span a 512-byte page boundary. If a SEPX will not fit in the space that remains in a page from recording previous text or SEPXs, space is skipped to allow the SEPX to start on a page boundary. A SEPX is guaranteed to be less than 512 bytes in length. If all sections in the document have default properties, no SEPXs would be stored.

pictures
Word picture structures immediately follow the preceding text/SEPXs and are concatenated one after the other if the document contains pictures.

embedded objects-native data
Word embedded object structures immediately follow the preceding text/SEPXs/picture and are concatenated one after the other if the document contains embedded objects.

FKPs for CHPs
The first CHP FKP begins at the first 512-byte boundary after the last byte of text\SEPX\picture\embedded objects written. The remaining CHP FKPs are recorded in the 512-byte pages that immediately follow.

FKPs for PAPs
The first PAP FKP is written in the 512-byte page that immediately follows the page used to record the last CHP FKP. The remaining PAP FKPs are recorded in the 512-byte pages that follow.

stsh (style sheet)
The style sheet is written at the beginning of the 512-byte page that immediately follows the last PAP FKP. This is recorded in all Windows 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.

plcfandRef (annotation reference position table)
Written immediately after the **plcffndTxt** if the document contains annotations.

plcfandTxt (annotation text position table)
Written immediately after the **plcfandRef**.if the document contains footnotes.

plcfsed (section table)
Written immediately after the previously recorded table. Recorded in all Windows Word documents.

plcfphe (paragraph height table)
Written immediately after the **plcfsed**, if paragraph heights have been recorded.

plcfpgd (page table)
Written immediately after the previously recorded table, if page boundary information is recorded.

sttbGlsy (glossary name string table)
Written immediately after the previously recorded table, if the document stored is a glossary.

plcfglsy (glossary entry text position table)
Written immediately after the **sttbGlsy**, 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 Windows Word documents.

plcfbtePapx (bin table for PAP FKPs)
Written immediately after the **plcfbteChpx**. This is recorded in all Windows Word documents.

sttbfFn (table of font name strings)
Written immediately after the **plcfbtePapx**. This is recorded in all Windows Word documents. 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[1].

plcffldMom(table of field positions and statuses for main document)
Written immediately after the **sttbfFn** 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.

plcffldMcr(table of field positions and statuses for macro subdocument)
Written immediately after the previously recorded table, if the macro subdocument contains fields.

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.

cmds (recording of command data structures)
Written immediately after the previously recorded table, if special commands are linked to this document.

plcfmcr (macro text position table -- delimits boundaries of text for macros stored in macro subdocument)
Written immediately after the previously recorded table, if a macro subdocument is recorded.

sttbfMcr (table of macro name strings)
Written immediately after the **plcfmcr**, if a macro subdocument is recorded.

PrEnv (data structures recording the print environment for document)
Written immediately after the previously recorded table, if a print environment is recorded for the document.

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

dop (document properties record)
Written immediately after the end of previously recorded structure.. This is recorded in all Windows Word documents.

sttbfAssoc(table of associated strings)

Autosave source(name of original)
Written immediately after the sttbfAssoc table. This field only appears in autosave files. These files are normal Word for Windows document 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 for Windows document.

---

[1]In the Winword 1.x format, the names of the first three fonts were omitted from the table and assumed to be "Tms Rmn" (for ftc = 0), "Symbol", and "Helv". In WinWord 2.0, the names for all fonts are included explitly in the table. It is still true that ftc = 0 represents the "best" Roman PS font on the system, ftc = 1 represents the Symbol font, and ftc = 2 represents the "best" Swiss (Sans Serif) PS font available.

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.
Group of SEPXs stored during last full save
Pictures stored during last full save
Embedded Ojbects stored during last full save
Drawing Objects stored during last full save
FKPs for CHPs during last full save
The first CHP FKP begins at the first 512-byte boundary after the last byte of text\SEPX\picture\embedded object written. The remaining CHP FKPs are recorded in the 512-byte pages that immediately follow.
FKPs for PAPs during last full save
The first PAP FKP is written in the 512-byte page that immediately follows the page used to record the last CHP FKP. The remaining PAP FKPs are recorded in the 512-byte pages that follow.

STSH (if style sheet has not grown since last full save)

Any text, SEPXs, pictures, embedded objects, or drawing objects stored during first fast save
Any CHP FKPs stored during first fast save
Any PAP FKPs stored during first fast save

Any text, SEPXs,pictures, embedded objects, or drawing objects stored during second fast save
Any CHP FKPs stored during second fast save
Any PAP FKPs stored during second fast save

...

Any text, SEPXs, pictures, embedded objects, or drawing objects stored during nth fast save
Any CHP FKPs stored during nth fast save
Any PAP FKPs stored during nth fast save
stsh (if style sheet has grown since last full save)
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.
plcfandRef (annotation reference position table)
Written immediately after the **plcffndTxt** if the document contains annotations.
plcfandTxt (annotation text position table)
Written immediately after the **plcfandRef**.if the document contains footnotes.
plcfsed (section table)
Written immediately after the previously recorded table. Recorded in all Windows Word documents.
plcfphe (paragraph height table)
Written immediately after the **plcfsed**, if paragraph heights have been recorded.
plcfpgd (page table)
Written immediately after the previously recorded table, if page boundary information is recorded.
sttbGlsy (glossary name string table)
Written immediately after the previously recorded table, if the document stored is a glossary.
plcfglsy (glossary entry text position table)
Written immediately after the **sttbGlsy**, 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 Windows Word documents.

plcfbtePapx (bin table for PAP FKPs)

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

sttbfFn (table of font name strings)

Written immediately after the **plcfbtePapx**. This is recorded in all Windows Word documents. 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[1] .

sttbRMark (table of Author names for Revision Marking)

Written immediately after the **plcfbtePapx** if revision marking is being tracked in the document. (**REVIEW** davidlu Each record in the sttb stores a 2-byte length extra portion, which contains undefined data. David, no definition of an sttb is given in this document, thus no definition of ”extra” data in an sttb is given.)

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

Written immediately after the **sttbfFn** 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.

plcffldMcr(table of field positions and statuses for macro subdocument)

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

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.

cmds (recording of command data structures)

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

plcfmcr (macro text position table -- delimits boundaries of text for macros stored in macro subdocument)

Written immediately after the previously recorded table, if a macro subdocument is recorded.

sttbfMcr (table of macro name strings)

Written immediately after the **plcfmcr**, if a macro subdocument is recorded.

PrEnv (data structures recording the print environment for document)

Written immediately after the previously recorded table, if a print environment is recorded for the 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/mail merge state information structure)

Written immediately after the end of previously recorded structure, (**REVIEW** davidlu;stevebu;jayb)

sttbEmbeddedFonts (table of font name strings for Embedded True Type Fonts stored in the file)

Written immediately after the end of the previously recorded structure, if Embedded True Type Fonts were stored in the document when it was saved.

rgfcEmbeddedFonts (array of FCs bounding the Embedded font data)

---

[1] In the Winword 1.x format, the names of the first three fonts were omitted from the table and assumed to be "Tms Rmn" (for ftc = 0), "Symbol", and "Helv". In WinWord 2.0, the names for all fonts are included explitly in the table. It is still true that ftc = 0 represents the "best" Roman PS font on the system, ftc = 1 represents the Symbol font, and ftc = 2 represents the "best" Swiss (Sans Serif) PS font available.

Written immediately after the end of the **sttbEmbeddedFonts**, if the file contains an **sttbEmbeddedFonts**. The binary data for the embedded font corresponding to font n in **sttbEmbeddedFonts** is stored in the main stream at file position rgfc[n], and has a length of rgfc[n+1] - rgfc[n].

Clx (encoding of the sprm lists and piece table for a complex file)
Written immediately after the end of previously recorded structure. This is recorded in all complex Windows Word documents.

dop (document properties record)
Written immediately after the end of previously recorded structure.. This is recorded in all Windows Word documents.

sttbfAssoc(table of associated strings)

Autosave source (documented above)

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 and fits within the first page 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):

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 (eg. "Wed, Dec 1, 1993")
11 Time in hours:minutes:seconds
12 Current section number
14 Abbreviated day of week (eg. "Thu" for "Thursday")
15 Day of week (eg. "Thursday")
16 Day short (eg. "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 (eg. "December 2, 1993")
30 Short Date (eg. "12/2/93")
33 Short Month (eg. "12" to represent "December")
34 Long Year (eg. "1993")
35 Short Year (eg. "93")
36 Abbreviated month (eg. "Dec" to represent "December")
37 Long month (eg. "December")
38 Current time in hours:minutes (eg. "2:01")
39 Long date (eg. "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.ccpHdrTxbox 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 andthat 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 grrprl 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 grrprl 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 FKPs, 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 FKPSs**, immediately following the **fkp.rgfc** is an array of 7 byte entries called **BXs**. 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 six 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 CHPXs (size of an **FC** plus size of one-byte word offset) or 11 bytes for PAPXs (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.

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 two-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 FCs 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 one type of 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 WinWord 2, 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.[2] 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.[3] Paragraph and character styles are stored in the same array. Each document has a separate array, so the same style will usually[4] 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.

[2] The DOD.hplhqstd is a handle to a plex (array) of hq's (handles) to std's (style descriptions).
[3] Istd (slot) 0 is Normal. Istd 1-9 are Heading 1-9. Istd 10 is Default Paragraph Font. Istd 11-14 are reserved. So the first non-fixed index is 15 (see stshi.istdMaxFixedWhenSaved.)
[4] Those styles in fixed locations in the stylesheet will have the same istd's in all documents.

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 grpprl).

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:

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

For a CHP:

1. An initial CHP is determined by getting the CHP from the paragraph's style.
2. Properties from the character's style (the UPX.chpx.grpprl) are then applied to that CHP.
3. 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*
cbStshi	2 bytes	size of the following STSHI structure[5]
STSHI	(cbStshi)	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     ftcStandardChpStsh;              // ftc used by StandardChpStsh for this document
```

[5] For early versions of Word 6 files (versions prior to nFib 67), this field was not written. The cbStshi to use for those file versions is 4 bytes.

```
    } 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, ie. 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", ie. the old names need to be replaced with the new.

stshi.ftcStandardChpStsh: This is the default font for this stylesheet.

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 */

    // Variable length part of STD:
    uchar   stzName[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, ie. 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 stzName, 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 (ie. 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 stiMax         75    // 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.stzName: The name of the style, including aliases. The name is stored as an stz (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 (eg. 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 (eg. 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 stz format of that name can be up to 255 characters.

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
```

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

std.grupx: This is an array[6] of variable-length UPXs, with std.cupx UPXs in the array. This array begins after the variable-length stzName 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 grpprl[cbMaxGrpprlStyleChpx];
            } chpx;
    struct
            {
            ushort istd;
            uchar grpprl[cbMaxGrpprlStylePapx];
            } 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[7], 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-

[6] More accurately a ”group”, because each of the elements (UPXs) in the array is variable-length.
[7] Note that the UPX.papx contains both a grpprl and an istd. Even if the grpprl is empty, the istd is still needed.

result" of the property formatting, ie. 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 the with 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.

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** always begins with a one 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 1. The length of a fixed length sprm is always 1 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 1.

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 1.

For variable length sprms, the total length of the sprm is the count recorded at offset 1 plus two. The parameter immediately follows the count.

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	*op code*	*Property Modified*	*Parameter*	*Parameter size*
sprmPIstd	2	pap.istd	istd (style code)	short
sprmPIstdPermute	3	pap.istd	permutation vector (see below)	variable length
sprmPIncLv1	4	pap.istd	difference between istd of base PAP and istd of PAP to be produced (see below)	byte
sprmPJc	5	pap.jc	jc (justification)	byte
sprmPFSideBySide	6	pap.fSideBySide	0 or 1	byte
sprmPFKeep	7	pap.fKeep	0 or 1	byte
sprmPFKeepFollow	8	pap.fKeepFollow	0 or 1	byte
sprmPPageBreakBefore	9	pap.fPageBreakBefore	0 or 1	byte
sprmPBrcl	10	pap.brcl	brcl	byte
sprmPBrcp	11	pap.brcp	brcp	byte
sprmPAnld	12	pap.anld	anld	variable length (the length of an ANLD structure)
sprmPNLvlAnm	13	pap.nLvlAnm	nn	byte
sprmPFNoLineNumb	14	pap.fNoLnn	0 or 1	byte
sprmPChgTabsPapx	15	pap.itbdMac, pap.rgdxaTab, pap.rgtbd	complex - see below	variable length
sprmPDxaRight	16	pap.dxaRight	dxa	word
sprmPDxaLeft	17	pap.dxaLeft	dxa	word
sprmPNest	18	pap.dxaLeft	dxa-see below	word
sprmPDxaLeft1	19	pap.dxaLeft1	dxa	word
sprmPDyaLine	20	pap.lspd	an LSPD, a long word structure consisting of a short of dyaLine followed by a short of fMultLinespace - see below	long
sprmPDyaBefore	21	pap.dyaBefore	dya	word
sprmPDyaAfter	22	pap.dyaAfter	dya	word
sprmPChgTabs	23	pap.itbdMac, pap.rgdxaTab, pap.rgtbd	complex - see below	variable length
sprmPFInTable	24	pap.fInTable	0 or 1	byte
sprmPTtp	25	pap.fTtp	0 or 1	byte
sprmPDxaAbs	26	pap.dxaAbs	dxa	word
sprmPDyaAbs	27	pap.dyaAbs	dya	word
sprmPDxaWidth	28	pap.dxaWidth	dxa	word
sprmPPc	29	pap.pcHorz, pap.pcVert	complex - see below	byte

sprmPBrcTop10	30	pap.brcTop	BRC10	word
sprmPBrcLeft10	31	pap.brcLeft	BRC10	word
sprmPBrcBottom10	32	pap.brcBottom	BRC10	word
sprmPBrcRight10	33	pap.brcRight	BRC10	word
sprmPBrcBetween10	34	pap.brcBetween	BRC10	word
sprmPBrcBar10	35	pap.brcBar	BRC10	word
sprmPFromText10	36	pap.dxaFromText	dxa	word
sprmPWr	37	pap.wr	wr (see description of PAP for definition	byte
sprmPBrcTop	38	pap.brcTop	BRC	word
sprmPBrcLeft	39	pap.brcLeft	BRC	word
sprmPBrcBottom	40	pap.brcBottom	BRC	word
sprmPBrcRight	41	pap.brcRight	BRC	word
sprmPBrcBetween	42	pap.brcBetween	BRC	word
sprmPBrcBar	43	pap.brcBar	BRC	word
sprmPFNoAutoHyph	44	pap.fNoAutoHyph	0 or 1	byte
sprmPWHeightAbs	45	pap.wHeightAbs	w	word
sprmPDcs	46	pap.dcs	DCS	short
sprmPShd	47	pap.shd	SHD	word
sprmPDyaFromText	48	pap.dyaFromText	dya	word
sprmPDxaFromText	49	pap.dxaFromText	dxa	word
sprmPFLocked	50	pap.fLocked	0 or 1	byte
sprmPFWidowControl	51	pap.fWidowControl	0 or 1	byte
sprmPRuler	52			
sprmCFStrikeRM	65	chp.fRMarkDel	1 or 0	bit
sprmCFRMark	66	chp.fRMark	1 or 0	bit
sprmCFFldVanish	67	chp.fFldVanish	1 or 0	bit
sprmCPicLocation	68	chp.fcPic and chp.fSpec	see below	variable length, length recorded is always 4
sprmCIbstRMark	69	chp.ibstRMark	index into sttbRMark	short
sprmCDttmRMark	70	chp.dttm	DTTM	long
sprmCFData	71	chp.fData	1 or 0	bit
sprmCRMReason	72	chp.idslRMReason	an index to a table of strings defined in Word 6.0 executables	short
sprmCChse	73	chp.fChsDiff and chp.chse	see below	3 bytes
sprmCSymbol	74	chp.fSpec, chp.chSym and chp.ftcSym	see below	variable length, length recorded is always 3
sprmCFOle2	75	chp.fOle2	1 or 0	bit
sprmCIstd	80	chp.istd	istd, see stylesheet definition	short
sprmCIstdPermute	81	chp.istd	permutation vector (see below)	variable length
sprmCDefault	82	whole CHP (see below)	none	variable length
sprmCPlain	83	whole CHP (see below)	none	0
sprmCFBold	85	chp.fBold	0,1, 128, or 129 (see below)	byte
sprmCFItalic	86	chp.fItalic	0,1, 128, or 129 (see below)	byte

sprmCFStrike	87	chp.fStrike	0,1, 128, or 129 (see below)	byte
sprmCFOutline	88	chp.fOutline	0,1, 128, or 129 (see below)	byte
sprmCFShadow	89	chp.fShadow	0,1, 128, or 129 (see below)	byte
sprmCFSmallCaps	90	chp.fSmallCaps	0,1, 128, or 129 (see below)	byte
sprmCFCaps	91	chp.fCaps	0,1, 128, or 129 (see below)	byte
sprmCFVanish	92	chp.fVanish	0,1, 128, or 129 (see below)	byte
sprmCFtc	93	chp.ftc	ftc	word
sprmCKul	94	chp.kul	kul	byte
sprmCSizePos	95	chp.hps, chp.hpsPos	(see below)	3 bytes
sprmCDxaSpace	96	chp.dxaSpace	dxa	word
sprmCLid	97	chp.lid	LID	word
sprmCIco	98	chp.ico	ico	byte
sprmCHps	99	chp.hps	hps	byte
sprmCHpsInc	100	chp.hps	(see below)	byte
sprmCHpsPos	101	chp.hpsPos	hps	byte
sprmCHpsPosAdj	102	chp.hpsPos	hps (see below)	byte
sprmCMajority	103	chp.fBold, chp.fItalic, chp.fSmallCaps, chp.fVanish, chp.fStrike, chp.fCaps, chp.ftc, chp.hps, chp.hpsPos, chp.kul, chp.dxaSpace, chp.ico, chp.lid	complex (see below)	variable length,length byte plus size of following grpprl
sprmCIss	104	chp.iss	iss	byte
sprmCHpsNew50	105	chp.hps	hps	variable width, length always recorded as 2
sprmCHpsInc1	106	chp.hps	complex (see below)	variable width, length always recorded as 2
sprmCHpsKern	107	chp.hpsKern	hps	short
sprmCMajority50	108	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	109	chp.hps	percentage to grow hps	short
sprmCCondHyhen	110	chp.ysri	ysri	short
sprmCFSpec	117	chp.fSpec	1 or 0	bit
sprmCFObj	118	chp.fObj	1 or 0	bit
sprmPicBrcl	119	pic.brcl	brcl (see PIC structure definition)	byte
sprmPicScale	120	pic.mx, pic.my, pic.dxaCropleft, pic.dyaCropTop pic.dxaCropRight, pic.dyaCropBottom	complex (see below)	length byte plus 12 bytes
sprmPicBrcTop	121	pic.brcTop	BRC	word
sprmPicBrcLeft	122	pic.brcLeft	BRC	word
sprmPicBrcBottom	123	pic.brcBottom	BRC	word

sprmPicBrcRight	124	pic.brcRight	BRC	word
sprmSScnsPgn	131	sep.cnsPgn	cns	byte
sprmSiHeadingPgn	132	sep.iHeadingPgn	heading number level	byte
sprmSOlstAnm	133	sep.olstAnm	OLST	variable length
sprmSDxaColWidth	136	sep.rgdxaColWidthSpacing	complex (see below)	3 bytes
sprmSDxaColSpacing	137	sep.rgdxaColWidthSpacing	complex (see below)	3 bytes
sprmSFEvenlySpaced	138	sep.fEvenlySpaced	1 or 0	byte
sprmSFProtected	139	sep.fUnlocked	1 or 0	byte
sprmSDmBinFirst	140	sep.dmBinFirst		word
sprmSDmBinOther	141	sep.dmBinOther		word
sprmSBkc	142	sep.bkc	bkc	byte
sprmSFTitlePage	143	sep.fTitlePage	0 or 1	byte
sprmSCcolumns	144	sep.ccolM1	# of cols - 1	word
sprmSDxaColumns	145	sep.dxaColumns	dxa	word
sprmSFAutoPgn	146	sep.fAutoPgn	obsolete	byte
sprmSNfcPgn	147	sep.nfcPgn	nfc	byte
sprmSDyaPgn	148	sep.dyaPgn	dya	short
sprmSDxaPgn	149	sep.dxaPgn	dya	short
sprmSFPgnRestart	150	sep.fPgnRestart	0 or 1	byte
sprmSFEndnote	151	sep.fEndnote	0 or 1	byte
sprmSLnc	152	sep.lnc	lnc	byte
sprmSGprfIhdt	153	sep.grpfIhdt	grpfihdt (see Headers and Footers topic)	byte
sprmSNLnnMod	154	sep.nLnnMod	non-neg int.	word
sprmSDxaLnn	155	sep.dxaLnn	dxa	word
sprmSDyaHdrTop	156	sep.dyaHdrTop	dya	word
sprmSDyaHdrBottom	157	sep.dyaHdrBottom	dya	word
sprmSLBetween	158	sep.fLBetween	0 or 1	byte
sprmSVjc	159	sep.vjc	vjc	byte
sprmSLnnMin	160	sep.lnnMin	lnn	word
sprmSPgnStart	161	sep.pgnStart	pgn	word
sprmSBOrientation	162	sep.dmOrientPage	dm	byte
sprmSBCustomize	163			
sprmSXaPage	164	sep.xaPage	xa	word
sprmSYaPage	165	sep.yaPage	ya	word
sprmSDxaLeft	166	sep.dxaLeft	dxa	word
sprmSDxaRight	167	sep.dxaRight	dxa	word
sprmSDyaTop	168	sep.dyaTop	dya	word
sprmSDyaBottom	169	sep.dyaBottom	dya	word
sprmSDzaGutter	170	sep.dzaGutter	dza	word
sprmSDMPaperReq	171	sep.dmPaperReq	dm	word
sprmTJc	182	tap.jc	jc	word (low order byte is significant)
sprmTDxaLeft	183	tap.rgdxaCenter (see below)	dxa	word
sprmTDxaGapHalf	184	tap.dxaGapHalf, tap.rgdxaCenter (see below)	dxa	word
sprmTFCantSplit	185	tap.fCantSplit	1 or 0	byte
sprmTTableHeader	186	tap.fTableHeader	1 or 0	byte
sprmTTableBorders	187	tap.rgbrcTable	complex(see below)	12 bytes

sprmTDefTable10	188	tap.rgdxaCenter, tap.rgtc	complex (see below)	variable length
sprmTDyaRowHeight	189	tap.dyaRowHeight	dya	word
sprmTDefTable	190	tap.rgtc	complex (see below)	
sprmTDefTableShd	191	tap.rgshd	complex (see below)	
sprmTTlp	192	tap.tlp	TLP	4 bytes
sprmTSetBrc	193	tap.rgtc[].rgbrc	complex (see below)	5 bytes
sprmTInsert	194	tap.rgdxaCenter,tap.rgtc	complex (see below)	4 bytes
sprmTDelete	195	tap.rgdxaCenter, tap.rgtc	complex (see below)	word
sprmTDxaCol	196	tap.rgdxaCenter	complex (see below)	4 bytes
sprmTMerge	197	tap.fFirstMerged, tap.fMerged	complex (see below)	word
sprmTSplit	198	tap.fFirstMerged, tap.fMerged	complex (see below)	word
sprmTSetBrc10	199	tap.rgtc[].rgbrc	complex (see below)	5 bytes
sprmTSetShd	200	tap.rgshd	complex (see below)	4 bytes
sprmMax	208			

The paragraph sprms used to encode paragraph properties in a PAPX are: sprmPJc, sprmPFSideBySide, sprmPFKeep, sprmPFKeepFollow, sprmPFPageBreakBefore, sprmPBrcp, sprmPPc, sprmPBrcl, sprmPNLvelAnm, sprmPFNoLineNumb, sprmPFSideBySide, sprmPDxaRight, sprmPDxaLeft., sprmPDxaLeft1, sprmPDyaLine, sprmPDyaBefore, sprmPDyaAfter, sprmPFNoAutoHyph, sprmPFInTable, sprmPFTtp, sprmPDxaAbs, sprmPDyaAbs, sprmPDxaWidth, sprmPBrcTop, sprmPBrcLeft, sprmPBrcBottom, sprmPBrcRight, sprmPBrcBetween, sprmPBrcBar, sprmPDxaFromText, sprmPDyaFromText, sprmPWr, sprmPWHeightAbs, sprmPShd, sprmPDcs, sprmPAnld and sprmPChgTabsPapx.

The table sprms used to encode table properties in a PAPX stored in a PAPX FKP are: sprmTJc, sprmTDxaGapHalf, sprmTDyaRowHeight, sprmTDefTableShd, and sprmTDefTable.

The section sprms used to encode section properties in a SEPX are:
sprmSBkc, sprmSFTitlePage, sprmSCcolumns, sprmSNfcPgn, sprmSPgnStart, sprmSFAutoPgn, sprmSDyaPgn, sprmSDxaPgn, sprmSFPgnRestart, sprmSFEndnote, sprmSLnc, sprmSGrpfIhdt, sprmSNLnnMod, sprmSDxaLnn, sprmSDyaHdrTop, sprmSDyaHdrBottom.

sprmPIstdPermute (opcode 3) 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	byte	opcode(==3)
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 4) is applied to pieces in the piece table that contain paragraphs with style codes (istds) greater than or equal to1 and less than or equal to9. 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 two 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 from pap.istd and then pap.stc should be set to min(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. Then pap.stc should be set to max(1, pap.istd). sprmPIncLvl is only stored in **grpprls** linked to a piece table.

The sprmPAnld (opcode 12) sets the pap.anld which is a data structure which describes what Word will display as an automatically generated sequence number at the beginning of an autonumbered paragraph. See the description of the ANLD in the data structure descriptions.

The sprmPChgTabsPapx (opcode 15) 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*
sprm	byte	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
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

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 18) 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 20) 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 23) 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	byte	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 29) 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:

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	sprm	byte			opcode
1	1		int	:4	F0	reserved
		pcVert	int	:2	0C	if pcVert ==3, pap.pcVert should not be changed. Otherwise, contains new value of pap.pcVert.
		pcHorz	int	:2	03	if pcHorz==3, pap.pcHorz should not be changed. Otherwise, contains new value of pap.pcHorz.

Length of sprmPPc is two 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 FKPs and also in **grpprls** linked to piece table entries.

sprmCPicLocation (opcode 68) is used ONLY IN CHPX FKPs. This sprm moves the 4 bytes of data stored at offset 2 in 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.

sprmCChse (opcode 73) 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. When this sprm is interpreted, the byte at offset 1 in the sprm is moved to chp.fChsDiff and the word stored at offset 2 is moved to chp.chse.

sprmCSymbol (opcode 74) is used to specify the font and the character that will be used within that font to display a symbol character in Word. The length byte recorded at offset 1 in this sprm will always be 3. When this sprm is

interpreted the two byte font code recorded at offset 2 is moved to chp.ftcSym, the single byte character specifier recorded at offset 4 is moved to chp.chSym and chp.fSpec is set to 1.

sprmCIstdPermute (opcode 81) (which has the same format as sprmPIstdPermute (opcode 3)). 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	byte	opcode(==81)
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 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 rgistd 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 82) 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 83) 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 85 through 92 (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 **grpprls** linked to piece table entries.

sprmCSizePos (opcode 95) is a four byte sprm consisting of the sprm opcode and a three byte parameter. The sprm has the following format:

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	sprm	byte			opcode
1	1	hpsSize	int	:8	FF	when != 0, contains new size of chp.hps
2	2	cInc	int	:7	FE	contains the number of font levels to increase or decrease size of chp.hps as a twos complement value.
		fAdjust	int	:1	01	when == 1, means that chp.hps should be adjusted up/down by one font level for super/subscripting change
3	3	hpsPos	int	: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 100) is a two-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 **grpprls** linked to piece table entries.

sprmCHpsPosAdj (opcode 102) causes the hps of a run to be reduced the first time 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 **grpprls** linked to piece table entries.

The parameter of sprmCMajority (opcode 103) 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. Byte 0 of sprmCMajority contains the opcode, byte 1 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=0x0400and chp.ftc = 4. Wordthen comparesfBold, 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 106) is used to increase or decrease chp.hps by increments of 1. This sprm is interpreted by adding the two byte increment stored at byte 2 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 108) has the same format as sprmCMajority and is interpreted in the same way.

sprmPicScale (opcode 120) 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 one 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.

sprmTDxaLeft (opcode 183) 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. Byte 0 of the sprm contains the opcode, and the new dxa position, call it dxaNew, is stored as an integer in bytes 1 and 2. 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 184) 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. Byte 0 of the sprm contains the opcode, and the new dxaGapHalf, call it dxaGapHalfNew, is stored in bytes 1 and 2. When the sprm is interpreted, the change between the old and new dxaGapHalf values, tap.dxaGapHalf - dxaGapHalfNew, is added to tap.rgdxaCenter[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 187) sets the tap.rgbrcTable. The sprm is interpreted by moving 12 bytes beginning at byte 1 of the sprm to tap.rgbrcTable.

sprmTDefTable10 (opcode 188) is an obsolete version of sprmTDefTable (opcode 154) 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 190) defines the boundaries of table cells (tap.rgdxaCenter) and the properties of each cell in a table (tap.rgtc). The 0th byte of the sprm contains its opcode. Bytes 1 and 2 store a two-byte length of the following paramter. Byte 3 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 4 through 4+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 4+ 2*(itcMac + 1) through byte 4+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 191) is similar to sprmTDefTable, and compliments it by defining the shading of each cell in a table (tap.rgshd). The 0th byte of the sprm contains its opcode. Bytes 1 and 2 store a two-byte length of the following paramter. Byte 3 contains the number of cells that are to be defined by the sprm, call it itcMac. itcMac cannot be larger than 32. In bytes 4 through 4+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 193) allows the border definitions(BRCs) within TCs to be set to new values. It has the following format:

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	sprm	byte			opcode 193
1	1	itcFirst	byte			the index of the first cell that is to have its borders changed.
2	2	itcLim	byte			index of the cell that follows the last cell to have its borders changed
3	3		int	:4	F0	reserved
		fChangeRight	int	:1	08	=1 when tap.rgtc[].brcRight is to be changed
		fChangeBottom	int	:1	04	=1 when tap.rgtc[].brcBottom is to be changed
		fChangeLeft	int	:1	02	=1 when tap.rgtc[].brcLeft is to be changed
		fChangeTop	int	:1	01	=1 when tap.rgtc[].brcTop is to be changed
4	4	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 194) inserts new cell definitions in an existing table's cell structure. The 0th byte of the sprm contains the opcode Byte 1 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 2 contains a count of the cell definitions to be added to the tap, call it ctc. Bytes 3 and 4 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 195) deletes cell definitions from an existing table's cell structure. The 0th byte of the sprm contains the opcode. Byte 1 contains the index of the first cell to delete, call it itcFirst. Byte 2 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 196) changes the width of cells whose index is within a certain range to be a certain value. The 0th byte of the sprm contains the opcode. Byte 1 contains the index of the first cell whose width is to be changed, call it itcFirst. Byte 2 contains the index of the cell that follows the last cell whose width is to be changed, call it itcLim. Bytes 3 and 4 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 197) merges the display areas of cells within a specified range. The 0th byte of the sprm contains the opcode. Byte 1 contains the index of the first cell that is to be merged, call it itcFirst. Byte 2 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 198) splits the display areas of merged cells into their originally assigned display areas. The 0th byte of the sprm contains the opcode. Byte 1 contains the index of the first cell that is to be split, call it itcFirst. Byte 2 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 >= itcFirst and < itcLim. sprmTSplit is stored only in **grpprls** linked to piece table entries.

SprmTSetBrc10 (opcode 199) has the same format as SprmTSetBrc but uses the old BRC10 structure.

sprmTSetShd (opcode 200) allows the shading definitions(SHDs) within a tap to be set to new values. The 0th byte of the sprm contains the opcode. Byte 1 contains the index of the first cell whose shading is to be changed, call it itcFirst. Byte 2 contains the index of the cell that follows the last cell whose shading is to be changed, call it itcLim. Bytes 3 and 4 contain the SHD structure, call it shd. This sprm causes the itcLim - itcFirst entries of tap.rgshd to be set to shd. sprmTDxaCol is stored only in **grpprls** linked to piece table entries.

COMPLEX FILE FORMAT

The complex file format is used when a file is fast-saved. A complex file has fib.fComplex set to 1. In a complex file, 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 **grpprl**s 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 t**ype), which is a 1-byte code, either 1 (meaning the block contains a **grpprl**) or 2 (meaning this is the **plcfpcd**). In both cases, the **clxt** is followed by a 2-byte cb which is the count of bytes of the **grpprl** or the piece table. 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
cb	count of bytes in piece table
plcfpcd	piece table

The entire CLXwould 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 **pcd**s 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 movedwithin 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 (ie. 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 pargraph 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

In Windows Word the text of a footnote is anchored to a particular position within the document's main text , the location of its footnote reference. 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 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 coresponding 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 (ie. 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 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 Windows Word, headers/footers can be defined for a document that:

1) will act as a separator between main text and footnote text

2) will print below footnote text on a page when footnote text must be continued on a succeeding page (continuation separator)

3) will print above footnote 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.

Within the document and the section properties of a document (the DOP and SEP) is a field, the **grpfIhdt**, which enumerates which of the header/footer types are defined for the document or for a particular section. The grpfIhdt in both the DOP and SEP is treated as a group of bit flags stored within a character field with a flag assigned to every type of header/footer that is possible to define for DOPs and SEPs. When a bit is on, it signifies that the header/footer type corresponding to the bit is defined for the document or for a particular section.

Definition of the bits of dop.grpfIhdt:

Bit position	
7	footnote separator defined when == 1 (fTrue).
6	footnote continuation separator defined when == 1 (fTrue).
5	footnote continuation notice defined when == 1 (fTrue).

Definition of the bits of sep.grpfIhdt:

Bit position	
7	header for even pages defined when == 1 (fTrue).
6	header for odd pages defined when == 1 (fTrue).
5	footer for even pages defined when == 1 (fTrue).
4	footer for odd pages defined when == 1 (fTrue).
3	header for first page of section defined when == 1 (fTrue).
2	footer for first page of section defined when == 1 (fTrue).

Given that a particular footnote separator exists, one can locate the text for that separator using the following algorithm:

Initially set ihdd (index into plcfhdd) to 0.
Scan bits 7, 6, and 5 of the dop.grpfIhdt in order looking for bit == 1 while you have not yet reached the bit corresponding to the separator whose text is to be located. For each such bit ==1 add 1 to ihdd.
The value of ihdd that results is the index into plcfhdd that can be used to access the text of the separator.

Given that a particular header/footer exists for a particular section, one can locate the text for that header/footer using the following algorithm:

initially set ihdd (index into plcfhdd) to 0.
scan bits 7, 6, and 5 of the dop.grpfIhdt looking for bit == 1 and add 1 to ihdd for each such bit == 1.
Examine the sep.grpfIhdt of each section preceding the section of the header/footer to be located in ascending section number order, scanning bits 7, 6, 5, 4, 3, and 2 of the sep.grpfIhdt in order, adding 1 to ihdd for each bit == 1.
For the section of the header/footer to be located, scan bits 7, 6, 5, 4, 3, and 2 of the sep.grpfIhdt in order looking for bit == 1 while you have not yet reached the bit corresponding to the header/footer to be located. For each such bit ==1 add 1 to ihdd.
The value of ihdd that results is the index into plcfhdd that can be used to access the text of the header/footer.

PAGE TABLE

The **plcfpgd**, referenced by the fib, gives the location of page breaks within a Word document and may optionally be saved in a Word binary file. If there are **n** page breaks calculated for a document, the plcfpgd would consist of **n+1** CP entries followed by **n** PGD entries.

Third-party creators of Windows Word files should not attempt to create a plcfpgd. It can only be created properly using Windows Word's page layout routines. If a Windows Word document is edited in any way, the plcfpgd should be deleted by setting fib.cbPlcfpgd to 0.

If there are **n** pages breaks recorded for the document stored, the **n+1st** CP stored in the array of CPs for the plcfpgd will have the value fib.ccpText + fib.ccpFtn + fib.ccpHdr + 1 if the document contains footnotes or header/footers and will have the value fib.ccpText + fib.ccpFtn + fib.ccpHdr if the document contains no subdocuments.

GLOSSARY FILES

A Word glossary file is a normal Word binary file with two supplemental files, the **sttbfglsy** and the **plcfglsy**, also stored in the file. The **sttbfglsy** contains a list of the names of glossary entries, and the **plcfglsy** 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. (ie. **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.**

If there are **n** glossary entries, the **plcfglsy**, 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).

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

Autonumbered List Data Descriptor (ANLD)

b_{10}	b_{16}	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	cbTextBefore	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	cbTextAfter	unsigned char			anld.cbTextBefore will be the beginning offset of the text in the anld.rgchthat will be displayed as the suffix of an autonumber. The sum of anld.cbTextBefore + anld.cbTextAfter 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 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 anld.fSetCaps == 1.
		fStrike	uns char	:1	80	determines whether autonumber will be displayed using caps when anld.fSetStrike == 1.

5	5	kul	uns char :3	07	determines whether autonumber will be displayed with underlining when anld.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	rgchAnld	array of 32 chars		characters displayed before/after autonumber

*cbANLD (count of bytes of ANLD) is 52 (decimal), 34(hex).

Autonumber Level Descriptor (ANLV)

b_{10}	b_{16}	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	cbTextBefore	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	cbTextAfter	unsigned char			anld.cbTextBefore will be the beginning offset of the text in the anld.rgchthat will be displayed as the suffix of an autonumber. The sum of anld.cbTextBefore + anld.cbTextAfter 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 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 anld.fSetCaps == 1.
		fStrike	uns char	:1	80	determines whether autonumber will be displayed using caps when anld.fSetStrike == 1.

5	5	kul	uns char	:3	07	determines whether autonumber will be displayed with underlining when anld.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 is 16 bytes (decimal), 10 bytes (hex).
).

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

BooKmark First descriptor (BKF)

b_{10}	b_{16}	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 is 4.

BooKmark Lim descriptor (BKL)

b_{10}	b_{16}	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 boomarks recorded in the hplcbkf to the bkl.ibkf to calculate the index to the BKF that corresponds to this entry.

cbBKL is 2.

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

Border Code (BRC)

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	dxpLineWidth	int	:3	0007	When dxpLineWidth is 0, 1, 2, 3, 4, or 5, this field is the width of a single line of border in units of 0.75 points. Each line in the border is this wide (e.g. a double border is three lines). Must be nonzero when brcType is nonzero. When dxpLineWidth is 6, it means that the border line is dotted. When dxpLineWidth is 7, it means the border line is dashed.
		brcType	int	:2	0018	border type code 0 none 1 single 2 thick 3 double
		fShadow	int	:1	0020	when 1, border is drawn with shadow. Must be 0 when BRC is a substructure of the TC
		ico	int	:5	07C0	color code (see chp.ico)
		dxpSpace	int	:5	F800	width of space to maintain between border and text within border. Must be 0 when BRC is a substructure of the TC. Stored in points for Windows.

Border Code for Windows Word 1.0 (BRC10)

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	dxpLine2Width	int	:3	0007	width of second line of border in pixels
		dxpSpaceBetween	int	:3	0038	distance to maintain between both lines of border in pixels
		dxpLine1Width	int	:3	01C0	width of first border line in pixels
		dxpSpace	int	: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	int	:1	4000	when 1, border is drawn with shadow. Must be 0 when BRC10 is a substructure of the TC.
		fSpare	int	: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.

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.)

Character Properties (CHP)

b10	b16	field	type	size	bitfield	comment
0	0	fBold	int	:1	0001	text is bold when 1 , and not bold when 0.
		fItalic	int	:1	0002	italic when 1, not italic when 0
		fRMarkDel	int	:1	0004	when 1, text has been deleted and will be displayed with strikethrus when revision marked text is to displayed
		fOutline	int	:1	0008	outlined when 1, not outlined when 0
		fFldVanish	int	:1	0010	<needs work>
		fSmallCaps	int	:1	0020	displayed with small caps when 1, no small caps when 0
		fCaps	int	:1	0040	displayed with caps when 1, no caps when 0
		fVanish	int	:1	0080	
1	1	fRMark	int	: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	int	:1	0200	character is a Word special character when 1, not a special character when 0
		fStrike	int	:1	0400	displayed with strikethrough when 1, no strikethrough when 0
		fObj	int	:1	0800	embedded object when 1, not an embedded object when 0
		fShadow	int	:1	1000	character is drawn with a shdow when 1; drawn without shadow when 0
		fLowerCase	int	: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	int	:1	4000	when 1, chp.fcPic points to an FFDATA the data structure binary data used by Word to describe a form field. 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	int	:1	8000	when 1, chp.lTagObj 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. 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		int	:16	FFFF	Reserved
4	4	ftc	short			font code. The ftc is an index into the rgffn structure. The rgffn entry indexed by ftc describes the font that will be used to display the run of text described by the CHP.
6	6	hps	unsigned short			font size in half points
8	8	dxaSpace	short			space following each character in the run expressed in twip units.
10	A	iss	int	:3	0007	superscript/subscript indices 0 means no super/subscripting 1 means text in run is superscripted 2 means text in run is subscripted
			int	:3	0038	reserved
		fSysVanish	int	:1	0040	used by Word internally, not stored in file
			int	:1	0080	reserved
11	B	ico	int	: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
		kul	int	:3	E000	underline code: 0 none 1 single 2 by word 3 double 4 dotted 5 hidden
12	C	hpsPos	short			super/subscript position in half points; positive means text is raised; negative means text is lowered.
14	E	lid	LID			language identification code

Language Name	Language ID
Arabic	0x0401
Bulgarian	0x0402
Catalan	0x0403
Traditional Chinese	0x0404
Simplified Chinese	0x0804

				Czech 0x0405 Danish 0x0406 German 0x0407 Swiss German 0x0807 Greek 0x0408 U.S. English 0x0409 U.K. English 0x0809 Austalian English 0x0c09 Castilian Spanish 0x040a Mexican Spanish 0x080a Finnish 0x040b French 0x040c Belgian French 0x080c Canadian French 0x0c0c Swiss French 0x100c Hebrew 0x040d Hungarian 0x040e Icelandic 0x040f Italian 0x0410 Swiss Italian 0x0810 Japanese 0x0411 Korean 0x0412 Dutch 0x0413 Belgian Dutch 0x0813 Norwegian - Bokmal 0x0414 Norwegian - Nynorsk 0x0814 Polish 0x0415 Brazilian Portuguese 0x0416 Portuguese 0x0816 Rhaeto-Romanic 0x0417 Romanian 0x0418 Russian 0x0419 Croato-Serbian (latin) 0x041a Serbo-Croatian (cyrillic) 0x081a Slovak 0x041b Albanian 0x041c Swedish 0x041d Thai 0x041e Turkish 0x041f Urdu 0x0420 Bahasa 0x0421
16	10	fcPic	FC	offset in document stream pointing to beginning of a picture when character is a picture character (character is 0x01 and chp.fSpec is 1)
16	10	fcObj	FC	offset in document 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)
16	10	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)
20	14	ibstRMark	short	index to author IDs stored in hsttbfRMark. used when text in run was newly typed or

				deleted when revision marking was enabled
22	16	dttmRMark	DTTM	Date/time at which this run of text was entered/modified by the author. (Only recorded when revision marking is on.)
26	1A	short		reserved
28	1C	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.
30	1E	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.chSym. Just like chp.ftc, chp.ftcSym is an index into the rgffn structure.
32	20	chSym	unsigned char	when chp.fSpec is 1 and the character recorded for the run in the document stream is chSymbol (0x28), the character stored chp.chSym will be displayed using the font specified in chp.ftcSym.
33	21	fChsDiff	unsigned char	when 1, the character set used to interpret the characters recorded in the run identified by chp.chse is different from the native character set for this document which is stored in fib.chse.
34	22	idslRMReason	short	an index to strings displayed as reasons for actions taken by Word's AutoFormat code
36	24	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
37	25	chYsr	unsigned character	the character that will be used to add or change a letter when chp.ysr is 2,3, 5 or 6
38	26	chse	unsigned short	extended character set id 0 characters in run should be interpreted using the ANSI set used by Windows 256 characters in run should be interpreted using the Macintosh character set.
40	28	hpsKern	unsigned short	kerning distance for characters in run recorded in half points

*cbCHP (count of bytes of CHP) is 42 (decimal), 2A(hex).

Character Property Exceptions (CHPX)

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

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	cb	byte			count of bytes of following data in CHPX.
1	1	grpprl	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.

The following sprms may be recorded in a **CHPX**:

sprm	fields in CHP altered by sprm
sprmCFSpec	**chp.fSpec**
sprmCSymbol	**chp.chSym, chp.ftcSym**
sprmCPicLocation	**chp.fcPic**
sprmCFStrikeRM	**chp.fRMarkDel**
sprmCFRMark	**chp.fRMark**
sprmCFFldVanish	**chp.fFldVanish**
sprmClbstRMark	**chp.ibstRMark**
sprmCDttmRMark	**chp.dttmRMark**
sprmCRMReason	**chp.idslRMReason**
sprmClstd	**chp.istd**
sprmCFBold	**chp.fBold**
sprmCFItalic	**chp.fItalic**
sprmCFStrike	**chp.fStrike**
sprmCFOutline	**chp.fOutline**
sprmCFShadow	**chp.fShadow**
sprmCFSmallCaps	**chp.fSmallCaps**
sprmCFCaps	**chp.fCaps**
sprmCFVanish	**chp.fVanish**
sprmCFtc	**chp.ftc**
sprmCKul	**chp.kul**
sprmCDxaSpace	**chp.dxaSpace**
sprmCLid	**chp.lid**
sprmClco	**chp.ico**
sprmCHps	**chp.hps**
sprmCHpsPos	**chp.hpsPos**
sprmClss	**chp.iss**
sprmCFData	**chp.fData**
sprmCFObj	**chp.fObj**
sprmCFOle2	**chp.fOle2**
sprmCYsri	**chp.ysri**
sprmCHpsKern	**chp.hpsKern**
sprmCChse	**chp.chse, chp.fChsDiff**

chpx.cb is equal to (1 + sizeof(chpx.grpprl)) .

Date and Time (internal date format) (DTTM)

b10	b16	field	type	size	bitfield	comment
0	0	mint	unsigned	:6	003F	minutes (0-59)
		hr	unsigned	:5	07C0	hours (0-23)
		dom	unsigned	:5	F800	days of month (1-31)
2	2	mon	unsigned	:4	000F	months (1-12)
		yr	unsigned	:9	1FF0	years (1900-2411)-1900
		wdy	unsigned	:3	E000	weekday, Sunday=0, Monday=1, Tuesday=2, Wednesday=3, Thursday=4, Friday=5, Saturday=6

Drop Cap Specifier(DCS)

b_{10}	b_{16}	field	type	size	bitfield	default value	comment
0	0	fdct	int	:3	0007	0	drop cap type 0 no drop cap 1 normal drop cap 2 drop cap in margin
			int	:5	00F8	0	count of lines to drop
1	1		int	:8			reserved

Document Properties (DOP)

b_{10}	b_{16}	field	type	size	bitfield	default value	comment
0	0	fFacingPages	int	:1	0001	0	1 when facing pages should be printed
		fWidowControl	int	:1	0002	1	1 when widow control is in effect. 0 when widow control disabled.
		fPMHMainDoc	int	:1	0004	0	1 when doc is a main doc for Print Merge Helper, 0 when not; default=0
		grfSuppression	int	:2	0018	0	Default line suppression storage; 0= form letter line suppression; 1= no line suppression; default=0
		fpc	int	:2	0060	1	footnote position code 0 print as endnotes 1 print at bottom of page 2 print immediately beneath text
			int	:1	0080	0	unused
1	1	grpfIhdt	int	:8	FF00	0	specification of document headers and footers. See explanation under Headers and Footers topic.
2	2	rncFtn	int	:2	0003	0	restart index for footnotes 0 don't restart note numbering 1 restart for each section 2 restart for each page
		nFtn	int	:14	FFFC	1	initial footnote number for document
4	4	fOutlineDirtySave	int	:1	0001		when 1, indicates that information in the hplcpad should be refreshed since outline has been dirtied
			int	:7	00FE		reserved
5	5	fOnlyMacPics	int	:1	0100		when 1, Word believes all pictures recorded in the document were created on a Macintosh
		fOnlyWinPics	int	:1	0200		when 1, Word believes all pictures recorded in the document were created in Windows
		fLabelDoc	int	:1	0400		when 1, document was created as a print merge labels document
		fHyphCapitals	int	:1	0800		when 1, Word is allowed to hyphenate words that are capitalized. When 0, capitalized may not be hyphenated
		fAutoHyphen	int	:1	1000		when 1, Word will hyphenate newly typed text as a background task
		fFormNoFields	int	:1	2000		
		fLinkStyles	int	:1	4000		when 1, Word will merge styles from its template

		fRevMarking	int	:1	8000		when 1, Word will mark revisions as the document is edited
6	6	fBackup	int	:1	0001		always make backup when document saved when 1.
		fExactCWords	int	:1	0002		
		fPagHidden	int	:1	0004		
		fPagResults	int	:1	0008		
		fLockAtn	int	:1	0010		when 1, annotations are locked for editing
		fMirrorMargins	int	:1	0020		swap margins on left/right pages when 1.
		fReadOnlyRecommended	int	:1	0040		user has recommended that this doc be opened read-only when 1
		fDfltTrueType	int	:1	0080		when 1, use TrueType fonts by default (flag obeyed only when doc was created by WinWord 2.x)
7	7	fPagSuppressTopSpacing	int	:1	0100		when 1, file created with SUPPRESSTOPSPACING=YES in win.ini. (flag obeyed only when doc was created by WinWord 2.x).
		fProtEnabled	int	:1	0200		when 1, document is protected from edit operations
		fDispFormFldSel	int	:1	0400		when 1, restrict selections to occur only within form fields
		fRMView	int	:1	0800		when 1, show revision markings on screen
		fRMPrint	int	:1	1000		when 1, print revision marks when document is printed
		fWriteReservation	int	:1	2000		
		fLockRev	int	:1	4000		when 1, the current revision marking state is locked
		fEmbedFonts	int	:1	8000		when 1, document contains embedded True Type fonts
8	8	copts.fNoTabForInd	int	:1	0001		compatability option: when 1, don't add automatic tab stops for hanging indent
		copts.fNoSpaceRaiseLower		:1	0002		compatability option: when 1, don't add extra space for raised or lowered characters
		copts.fSupressSpbfAfterPageBreak:		:1	0004		compatability option: when 1, supress the paragraph Space Before and Space After options after a page break
		copts.fWrapTrailSpaces		:1	0008		compatability option: when 1, wrap trailing spaces at the end of a line to the next line
		copts.fMapPrintTextColor		:1	0010		compatability option: when 1, print colors as black on non-color printers

		copts.fNoColumnBalance		:1	0020		compatability option: when 1, don't balance columns for Continuous Section starts
		copts.fConvMailMergeEsc		:1	0040		
		copts.fSupressTopSpacing		:1	0080		compatability option: when 1, supress extra line spacing at top of page
		copts.fOrigWordTableRules		:1	0100		compatability option: when 1, combine table borders like Word 5.x for the Macintosh
		copts.fTransparentMetafiles		:1	0200		compatability option: when 1, don't blank area between metafile pictures
		copts.fShowBreaksInFrames		:1	0400		compatability option: when 1, show hard page or column breaks in frames
		copts.fSwapBordersFacingPgs		:1	0800		compatability option: when 1, swap left and right pages on odd facing pages
					F000		reserved
10	A	dxaTab	uns			720 twips	default tab width
12	C	wSpare	uns				
14	E	dxaHotZ	uns				width of hyphenation hot zone measured in twips
16	10	cConsecHypLim	uns				number of lines allowed to have consecutive hyphens
18	12	wSpare2	uns			reserved	
20	14	dttmCreated	DTTM				date and time document was created
24	18	dttmRevised	DTTM				date and time document was last revised
28	1C	dttmLastPrint	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	int	:2	0003		restart endnote number code 0 don't restart endnote numbering 1 restart for each section 2 restart for each page
		nEdn	int	:14	FFFC		beginning endnote number
54	36	epc	int	:2	0003		endnote position code 0 display endnotes at end of

						section 3 display endnotes at end of document
		nfcFtnRef	int	: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	int	: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
55	37	fPrintFormData	int	:1	0400	only print data inside of form fields
		fSaveFormData	int	:1	0800	only save document data that is inside of a form field.
		fShadeFormData	int	: 1	1000	shade form fields
				:2	6000	reserved
		fWCFtnEdn	int	: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	int	:3	0007	document view kind 0 Normal view 1 Outline view 2 Page View
		wScaleSaved	int	:9	0FF8	

zkSaved	int	:2	3000	

cbDOP is 84.
cwDOP is 42.

Drawing Object (Word) (DO)

b10	b16	field	type	size	bitfield	comment
0	0	fc	long			FC pointing to drawing object data
0	0	dok	uns short			Drawn Object Kind, currently this is always 0
2	2	cb	short			size (count of bytes) of the entire DO
4	4	bx	uns char			x position relative to anchor CP
5	5	by	uns char			y position relative to anchor CP
6	6	dhgt	short			height of DO
8	8	fAnchorLock	uns short	:1	0001	1 if the DO anchor is locked
10	a	rgdp				variable length array of drawing primitives

Drawing Primitive Header (Word) (DPHEAD)

b10	b16	field	type	size	bitfield	comment
0	0	dpk	uns short			Drawn Primitive Kind REVIEW davebu 0x0000 = start of grouping of primitives 0x0001 = line 0x0002 = textbox 0x0003 = rectangle 0x0004 = arc 0x0005 = elipse 0x0006 = polyline 0x0007 = callout textbox 0x0008 = end of grouping of primitives 0x0009 = sample primitve holding default values
2	2	cb	short			size (count of bytes) of this DP
4	4	xa	short			These 2 points describe the rectangle
6	6	ya	short			enclosing this DP relative to the origin of
8	8	dxa	short			the DO
10	a	dya	short			

Drawing Primitive (Word) (DP)

b10	b16	field	type	size	bitfield	comment
0	0	dphead	DPHEAD	12		Common header for a drawing primitive
			DP data for a line			
12	c	xaStart	short			starting point for line
14	e	yaStart	short			
12	c	xaEnd	short			ending point for line
14	e	yaEnd	short			
16	10	lnpc	long			LiNe Property Color -- RGB color value
20	14	lnpw	short			line property weight in twips
22	16	lnps	short			line property style 0 Solid 1 Dashed 2 Dotted 3 Dash Dot 4 Dash Dot Dot 5 Hollow
24	18	eppsStart	uns short	:2	0003	Start EndPoint Property Style 0 None 1 Hollow 2 Filled
		eppwStart	uns short	:2	000c	Start EndPoint Property Weight
		epplStart	uns short	:2	0030	Start EndPoint Property length
26	1a	eppsEnd	uns short	:2	0003	End EndPoint Property Style
		eppwEnd	uns short	:2	000c	End EndPoint Property Weight
		epplEnd	uns short	:2	0030	End EndPoint Property length
28	1c	shdwpi	short			Shadow Property Intensity REVIEW davebu
30	1e	xaOffset	short			x offset of shadow
32	20	yaOffset	short			y offset of shadow
		DP data for a textbox (DPTXBX)				
12	c	lnpc	long			LiNe Property Color -- RGB color value
16	10	lnpw	short			line property weight in twips
18	12	lnps	short			line property style See description above in the DP data for a line
20	14	dlpcFg	long			FiLl Property Color ForeGround -- RGB color value
24	18	dlpcBg	long			FiLl Property Color BackGround -- RGB color value
28	1c	flpp	short			FiLl Property Pattern REVIEW davebu
30	1e	shdwpi	short			Shadow Property Intensity
32	20	xaOffset	short			x offset of shadow
34	22	yaOffset	short			y offset of shadow
36	24	fRoundCorners	uns short	:1	0001	1 if the textbox has rounded corners
36	24	zaShape	uns short	:15	000e	REVIEW davebu
38	26	dzaInternalMargin	short			REVIEW davebu
		DP data for a rectangle				
12	c	lnpc	long			LiNe Property Color -- RGB color value
16	10	lnpw	short			line property weight in twips
18	12	lnps	short			line property style See description above in the DP data for a line
20	14	dlpcFg	long			FiLl Property Color ForeGround -- RGB color value

24	18	dlpcBg	long		FiLl Property Color BackGround -- RGB color value
28	1c	flpp	short		FiLl Property Pattern REVIEW davebu
30	1e	shdwpi	short		Shadow Property Intensity
32	20	xaOffset	short		x offset of shadow
34	22	yaOffset	short		y offset of shadow
36	24	fRoundCorners	uns short :1	0001	1 if the textbox has rounded corners
36	24	zaShape	uns short :15	000e	REVIEW davebu
		DP data for an arc			
12	c	lnpc	long		LiNe Property Color -- RGB color value
16	10	lnpw	short		line property weight in twips
18	12	lnps	short		line property style See description above in the DP data for a line
20	14	dlpcFg	long		FiLl Property Color ForeGround -- RGB color value
24	18	dlpcBg	long		FiLl Property Color BackGround -- RGB color value
28	1c	flpp	short		FiLl Property Pattern REVIEW davebu
30	1e	shdwpi	short		Shadow Property Intensity
32	20	xaOffset	short		x offset of shadow
34	22	yaOffset	short		y offset of shadow
36	24	fLeft	uns short :8	00ff	REVIEW davebu
36	24	fUp	uns short :8	ff00	REVIEW davebu
		DP data for an elipse			
12	c	lnpc	long		LiNe Property Color -- RGB color value
16	10	lnpw	short		line property weight in twips
18	12	lnps	short		line property style See description above in the DP data for a line
20	14	dlpcFg	long		FiLl Property Color ForeGround -- RGB color value
24	18	dlpcBg	long		FiLl Property Color BackGround -- RGB color value
28	1c	flpp	short		FiLl Property Pattern REVIEW davebu
30	1e	shdwpi	short		Shadow Property Intensity
32	20	xaOffset	short		x offset of shadow
34	22	yaOffset	short		y offset of shadow
		DP data for a polyline (DPPOLYLINE)			
12	c	lnpc	long		LiNe Property Color -- RGB color value
16	10	lnpw	short		line property weight in twips
18	12	lnps	short		line property style See description above in the DP data for a line
20	14	dlpcFg	long		FiLl Property Color ForeGround -- RGB color value
24	18	dlpcBg	long		FiLl Property Color BackGround -- RGB color value
28	1c	flpp	short		FiLl Property Pattern REVIEW davebu
30	1e	eppsStart	uns short :2	0003	Start EndPoint Property Style

					0 None 1 Hollow 2 Filled
		eppwStart	uns short :2	000c	Start EndPoint Property Weight
		epplStart	uns short :2	0030	Start EndPoint Property length
32	20	eppsEnd	uns short :2	0003	End EndPoint Property Style
		eppwEnd	uns short :2	000c	End EndPoint Property Weight
		epplEnd	uns short :2	0030	End EndPoint Property length
34	22	shdwpi	short		Shadow Property Intensity
36	24	xaOffset	short		x offset of shadow
38	26	yaOffset	short		y offset of shadow
40	28	fPolygon	uns short :1	0001	1 if this is a polygon
40	28	cpt	uns short :15	00fe	count of points
42	2a	xaFirst	short		These are the endpoints of the first line.
44	2c	yaFirst	short		
46	2e	xaEnd	short		
48	30	yaEnd	short		
50	32	rgpta[]	short		An array of xa,ya pairs for the remaining points

DP data for a callout textbox

12	c			REVIEW davebu flags
14	e	dzaOffset	short	REVIEW davebu
16	10	dzaDescent	short	REVIEW davebu
18	12	dzaLength	short	REVIEW davebu
20	14	dpheadTxbx	DPHEAD	DPHEAD for a textbox
32	20	dptxbx	DP	DP for a textbox
60	4c	dpheadPolyLine	DPHEAD	DPHEAD for a Polyline
72	48	dpPolyLine	DP	DP for a polyline

DP data for a sample primitive holding default values

12	c	lnpc	long		LiNe Property Color -- RGB color value
16	10	lnpw	short		line property weight in twips
18	12	lnps	short		line property style See description above in the DP data for a line
20	14	dlpcFg	long		FiLl Property Color ForeGround -- RGB color value
24	18	dlpcBg	long		FiLl Property Color BackGround -- RGB color value
28	1c	flpp	short		FiLl Property Pattern REVIEW davebu
30	1e	eppsStart	uns short :2	0003	Start EndPoint Property Style 0 None 1 Hollow 2 Filled
		eppwStart	uns short :2	000c	Start EndPoint Property Weight
		epplStart	uns short :2	0030	Start EndPoint Property length
32	20	eppsEnd	uns short :2	0003	End EndPoint Property Style
		eppwEnd	uns short :2	000c	End EndPoint Property Weight
		epplEnd	uns short :2	0030	End EndPoint Property length
34	22	shdwpi	short		Shadow Property Intensity
36	24	xaOffset	short		x offset of shadow
38	26	yaOffset	short		y offset of shadow
42	2a	dzaOffset	short		REVIEW davebu
44	2c	dzaDescent	short		REVIEW davebu

46	2e	dzaLength	short			REVIEW davebu
48	30	fRoundCorners	uns short	:1	0001	1 if the textbox has rounded corners
48	30	zaShape	uns short	:15	000fe	REVIEW davebu
50	32	dzaInternalMargin	short			REVIEW davebu

Embedded Object Properties (OBJHEADER)

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	lcb	long			length of object (including this header)
4	4	cbHeader	int			length of this header (for future use)
6	6	icf	int			index to clipboard format of object

Field Descriptor (FLD)

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	ch	byte			type of field boundary the FLDdescribes. 19 field begin mark 20 field separator 21 field end mark
variant used when fld.ch == 19 (field begin mark)						
1	1	flt	byte			field type see flt table below
variant used when fld.ch == 21 (field end mark)						
1	1	fDiffer	int	:1	01	ignored for saved file
		fZombieEmbed	int	:1	02	==1, when result still believes this field is an EMBED or LINK field
		fResultDirty	int	: 1	04	== 1, when user has edited or formatted the result. ==0 otherwise
		fResultEdited	int	:1	08	==1, when user has inserted text into or deleted text from the result.
		fLocked	int	:1	10	==1, when field is locked from recalc
		fPrivateResult	int	:1	20	==1, whenever the result of the field is never to be shown.
		fNested	int	: 1	40	==1, when field is nested within another field
		fHasSep	int	:1	80	==1, when field has a field separator

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

29	quote File Name variable
30	quote Document Template Name variable
31	quote Current Date variable
32	quote Current Time variable
33	quote Current Page variable
34	evaluate expression
35	insert literal text
36	Include command (Print Merge)
37	page reference
38	Ask command (Print Merge)
39	Fillin command to display prompt (Print Merge)
40	Data command (Print Merge)
41	Next command (Print Merge)
42	NextIf command (Print Merge)
43	SkipIf (Print Merge)
44	inserts number of current Print Merge record
45	DDE reference
46	DDE automatic reference
47	Inserts Glossary Entry
48	sends characters to printer without translation
49	Formula definition
50	Goto Button
51	Macro Button
52	insert auto numbering field in outline format
53	insert auto numbering field in legal format
54	insert auto numbering field in arabic number format
55	reads a TIFF file
56	Link
57	Symbol
58	Embedded Object
59	Merge fields
60	User Name
61	User Initial
62	User Address
63	Bar code
65	Section
66	Section pages
67	Include Picture
68	Include Text
69	File Size
70	Form Text Box
71	Form Check Box
72	Note Reference
73	Create Table of Authorities
74	Mark Table of Authorities Entry
75	Merge record sequence number
76	Macro
77	Private
78	Insert Database
79	Autotext
80	Compare two values
81	Plug-in module private
82	Subscriber
83	Form List Box
84	Advance

File Drawn Object Address (Word) (FDOA)

b10	b16	field	type	size	bitfield	comment
0	0	fc	long			FC pointing to drawing object data
4	4	ctxbx	short			count of textboxes in the drawing object

Font Family Name (FFN)

b10	b16	field	type	size	bitfield	comment
0	0	cbFfnM1	uns char			total length of FFN - 1.
1	1	prg	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	ibszAlt	uns char			index into ffn.szFfn to the name of the alternate font
6	6	szFfn	char[]			zero terminated string that records name of font. Possibly followed by a second sz which records the name of an alternate font to use if the first named font does not exist on this system. Maximal size of szFfn is 65 characters.

File Information Block (Windows Word) (FIB)

b10	b16	field	type	size	bitfield	comment
0	0	wIdent	int			magic number
2	2	nFib	int			FIB version written
4	4	nProduct	int			product version written by
6	6	lid	int			language stamp---localized version; In pre-WinWord2.0 files this value was the nLocale. If value is < 999, then it is the nLocale, otherwise it is the lid.
8	8	pnNext	PN			
10	A	fDot	uns	:1	0001	
		fGlsy	uns	:1	0002	
		fComplex	uns	:1	0004	when 1, file is in **complex, fast-saved format.**
		fHasPic	uns	:1	0008	file contains 1 or more pictures
		cQuickSaves	uns	:4	00F0	count of times file was quicksaved
11	B	fEncrypted	uns	:1	0100	1 if file is encrypted, 0 if not
			uns	:1	0200	reserved
		fReadOnlyRecommended	uns	:1	0400	=1 when user has recommended that file be read read-only
		fWriteReservation	uns	:1	0800	=1, when file owner has made the file write reserved
		fExtChar	uns	:1	1000	=1, when using extended character set in file
		*	uns	:3	E000	unused
12	C	nFibBack	uns			
14	E	lKey	long			file encrypted key, only valid if fEncrypted.
18	12	envr	unsigned char			environment in which file was created 0 created by Win Word 1 created by Mac Word
19	13					reserved
20	14	chse	unsigned short			default extended character set id for text in document stream. (overidden 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	16	chseTables	uns			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	18	fcMin	FC			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	1C	fcMac	FC			file offset of last character of text in document text stream + 1

32	20	cbMac	FC	file offset of last byte written to file + 1.
36	24	fcSpare0	FC	reserved
40	28	fcSpare1	FC	reserved
44	2C	fcSpare2	FC	reserved
48	30	fcSpare3	FC	reserved
52	34	ccpText	CP	length of main document text stream
56	38	ccpFtn	CP	length of footnote subdocument text stream
60	3C	ccpHdd	CP	length of header subdocument text stream
64	40	ccpMcr	CP	length of macro subdocument text stream
68	44	ccpAtn	CP	length of annotation subdocument text stream
72	48	ccpEdn	CP	length of endnote subdocument text stream
76	4C	ccpTxbx	CP	length of textbox subdocument text stream
80	50	ccpHdrTxbx	CP	length of header textbox subdocument text stream Note: when ccpFtn == 0 and ccpHdr == 0 and ccpMcr == 0 and ccpAtn == 0 and ccpEdn ==0 and ccpTxbx == 0 and ccpHdrTxbx == 0, then fib.fcMac = fib.fcMin+ fib.ccpText. If either ccpFtn != 0 or ccpHdd != 0 or ccpMcr != 0 or ccpAtn != 0 or ccpEdn !=0 or ccpTxbx != 0 or ccpHdrTxbx == 0, then fib.fcMac = fib.fcMin + fib.ccpText + fib.ccpFtn + fib.ccpHdd + fib.ccpMcr + fib.ccpAtn + fib.ccpEdn + fib.ccpTxbx + fib.ccpHdrTxbx + 1. The single character stored beginning at file position fib.fcMac - 1 must always be a CR character (ASCII 13).
84	54	ccpSpare2	CP	reserved
88	58	fcStshfOrig	FC	file offset of original allocation for STSH in file. During fast save Word will attempt to reuse this allocation if STSH is small enough to fit.
92	5C	lcbStshfOrig	long	count of bytes of original STSH allocation

96	60	fcStshf	FC	file offset of STSH in file.
100	64	lcbStshf	long	count of bytes of current STSH allocation
104	68	fcPlcffndRef	FC	file offset of footnote reference PLC. CPs in PLC are relative to main document text stream and give location of footnote references. The structure stored in this plc, called the **FRD** (**f**ootnote **r**eference **d**escriptor) is two byte long.
108	6C	lcbPlcffndRef	long	count of bytes of footnote reference PLC == 0 if no footnotes defined in document.
112	70	fcPlcffndTxt	FC	file offset of footnote text PLC. CPs in PLC are relative to footnote subdocument text stream and give location of beginnings of footnote text for correspondings 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
116	74	lcbPlcffndTxt	long	count of bytes of footnote text PLC. == 0 if no footnotes defined in document
120	78	fcPlcfandRef	FC	file offset of annotation reference PLC. The CPs recorded in this PLC give the offset of annotation references in the main document.
124	7C	lcbPlcfandRef	long	count of bytes of annotation reference PLC.
128	80	fcPlcfandTxt	FC	file offset 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.
132	84	lcbPlcfandTxt	long	count of bytes of the annotation text PLC
136	88	fcPlcfsed	FC	file offset of section descriptor PLC. CPs in PLC are relative to main document. The length of the **SED** is 12 bytes.
140	8C	lcbPlcfsed	long	count of bytes of section descriptor PLC.
144	90	fcPlcfpad	FC	file offset of paragraph descriptor PLC for main document which is used by Word's Outline view. CPs in PLC are relative to main document. The length of the **PGD** is 8 bytes.
148	94	lcbPlcfpad	long	count of bytes of paragraph descriptor PLC. ==0 if file was never viewed in Outline view. Should not be written by third party creators of Word files.
152	98	fcPlcfphe	FC	file offset of PLC of paragraph heights. CPs in PLC are relative to main document text stream. Only written for fies in **complex** format. Should not be written by

				third party creators of Word files. The **PHE** is 6 bytes long.
156	9C	lcbPlcfphe	long	count of bytes of paragraph height PLC. ==0 when file is **non-complex**.
160	A0	fcSttbfglsy	FC	file offset of glossary string table. This table consists of pascal style strings (strings stored prefixed with a length byte) concatenated one after another.
164	A4	lcbSttbfglsy	long	count of bytes of glossary string table. == 0 for non-glossary documents. !=0 for glossary documents.
168	A8	fcPlcfglsy	FC	file offset 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 sttbfglsy. No structure is stored in this PLC. There will be **n+1 FC** entries in this PLC when there are **n** glossary entries.
172	AC	lcbPlcfglsy	long	count of bytes of glossary PLC. == 0 for non-glossary documents. !=0 for glossary documents.
176	B0	fcPlcfhdd	FC	byte offset of header PLC. CPs are relative to header subdocument and mark the beginnings of individual headers in the header subdoc. 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.
180	B4	lcbPlcfhdd	long	count of bytes of header PLC. == 0 if document contains no headers
184	B8	fcPlcfbteChpx	FC	file offset of character property bin table.PLC. FCs in PLC are file offsets. Describes text of main document and all subdocuments. The **BTE** is 2 bytes long.
188	BC	lcbPlcfbteChpx	long	count of bytes of character property bin table PLC.
192	C0	fcPlcfbtePapx	FC	file offset of paragraph property bin table.PLC. FCs in PLC are file offsets. Describes text of main document and all subdocuments. The **BTE** is 2 bytes long.
196	C4	lcbPlcfbtePapx	long	count of bytes of paragraph property bin table PLC.
200	C8	fcPlcfsea	FC	file offset of PLC reserved for private use. The **SEA** is 6 bytes long.
204	CC	lcbPlcfsea	uns	count of bytes of private use PLC.
208	DO	fcSttbfffn	FC	file offset of font information STTBF. 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.
212	D4	lcbSttbfffn	long	count of bytes in sttbfffn.
216	D8	fcPlcffldMom	FC	offset in doc stream to the 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.
220	DC	lcbPlcffldMom	long	
224	E0	fcPlcffldHdr	FC	offset in doc stream to the PLC of field positions in the header subdocument.
228	E4	lcbPlcffldHdr	long	
232	E8	fcPlcffldFtn	FC	offset in doc stream to the PLC of field positions in the footnote subdocument.
236	EC	lcbPlcffldFtn	long	
240	F0	fcPlcffldAtn	FC	offset in doc stream to the PLC of field positions in the annotation subdocument.
244	F4	lcbPlcffldAtn	long	
248	F8	fcPlcffldMcr	FC	offset in doc stream to the PLC of field positions in the macro subdocument.
252	FC	lcbPlcffldMcr	long	
256	100	fcSttbfbkmk	FC	offset in document stream of the STTBF that records bookmark names in the main document
260	104	lcbSttbfbkmk	long	
264	108	fcPlcfbkf	FC	offset in document stream of the PLCF that records the beginning CP offsets of bookmarks in the main document. See BKF structure definition
268	10C	lcbPlcfbkf	long	
272	110	fcPlcfbkl	FC	offset in document stream of the PLCF that records the ending CP offsets of bookmarks recorded in the main document. See the BKL structure definition.
276	114	lcbPlcfbkl	long	
280	118	fcCmds	FC	
284	11C	lcbCmds	long	
288	120	fcPlcmcr	FC	
292	124	lcbPlcmcr	long	
296	128	fcSttbfmcr	FC	
300	12C	lcbSttbfmcr	long	

304	130	fcPrDrvr	FC	file offset of the printer driver information (names of drivers, port etc...)
308	134	lcbPrDrvr	long	count of bytes of the printer driver information (names of drivers, port etc...)
312	138	fcPrEnvPort	FC	file offset of the print environment in portrait mode.
316	13C	lcbPrEnvPort	long	count of bytes of the print environment in portrait mode.
320	140	fcPrEnvLand	FC	file offset of the print environment in landscape mode.
324	144	lcbPrEnvLand	long	count of bytes of the print environment in landscape mode.
328	148	fcWss	FC	file offset of **W**indow **S**ave **S**tate data structure. **WSS** contains dimensions of document's main text window and the last selection made by Word user.
332	14C	lcbWss	long	count of bytes of WSS. ==0 if unable to store the window state. Should not be written by third party creators of Word files.
336	150	fcDop	FC	file offset of document property data structure.
340	154	lcbDop	long	count of bytes of document properties.
344	158	fcSttbfAssoc	FC	offset to 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.
348	15C	cbSttbfAssoc	long	
352	160	fcClx	FC	file of offset of beginning of information for **complex** files. Consists of an encoding of all of the **prm**s quoted by the document followed by the **plcpcd** (piece table) for the document.
356	164	lcbClx	long	count of bytes of complex file information. == 0 if file is **non-complex**.
360	168	fcPlcfpgdFtn	FC	file offset of page descriptor PLC for footnote subdocument. CPs in PLC are relative to footnote subdocument. Should not be written by third party creators of Word files.
364	16C	lcbPlcfpgdFtn	long	count of bytes of page descriptor PLC for footnote subdocument. ==0 if document has not been paginated. The length of the **PGD** is 8 bytes.
368	170	fcAutosaveSource	FC	file offset of the name of the original file. fcAutosaveSource and cbAutosaveSource should both be 0 if autosave is off.

372	174	lcbAutosaveSource	long	count of bytes of the name of the original file.
376	178	fcGrpStAtnOwners	FC	group of strings recording the names of the owners of annotations stored in the document
380	17C	lcbGrpStAtnOwners	long	count of bytes of the group of strings
384	180	fcSttbfAtnbkmk	FC	file offset of the sttbf that records names of bookmarks in the annotation subdocument
388	184	lcbSttbfAtnbkmk	long	length in bytes of the sttbf that records names of bookmarks in the annotation subdocument
392	188	wSpare4Fib		
394	18A	pnChpFirst	PN	the page number of the lowest numbered page in the document that records CHPX FKP information
396	18C	pnPapFirst	PN	the page number of the lowest numbered page in the document that records PAPX FKP information
398	18E	cpnBteChp	PN	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.
400	190	cpnBtePap	PN	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.
402	192	fcPlcfdoaMom	FC	file offset of the **FDOA** (drawn object) PLC for main document. ==0 if document has no drawn objects. The length of the **FDOA** is 6 bytes.
406	196	lcbPlcfdoaMom	long	length in bytes of the **FDOA** PLC of the main document
410	19A	fcPlcfdoaHdr	FC	file offset of the **FDOA** (drawn object) PLC for the header document. ==0 if document has no drawn objects. The length of the **FDOA** is 6 bytes.
414	19E	lcbPlcfdoaHdr	long	length in bytes of the **FDOA** PLC of the header document
418	1A2	fcUnused1	FC	
422	1A6	lcbUnused1	long	
426	1AA	fcUnused2	FC	
430	1AE	lcbUnused2	long	
434	1B2	fcPlcfAtnbkf	FC	file offset of BKF (bookmark first) PLC of the annotation subdocument
438	1B6	lcbPlcfAtnbkf	long	length in bytes of BKF (bookmark first) PLC of the annotation subdocument
442	1BA	fcPlcfAtnbkl	FC	file offset of BKL (bookmark last) PLC of the annotation subdocument
446	1BE	lcbPlcfAtnbkl	long	length in bytes of BKL (bookmark first) PLC of the annotation subdocument
450	1C2	fcPms	FC	file offset of PMS (Print Merge State) information block
454	1C6	lcbPMS	long	length in bytes of PMS
458	1CA	fcFormFldSttbf	FC	file offset of form field Sttbf which contains strings used in form field dropdown controls

462	1CE	lcbFormFldSttbf	long	length in bytes of form field Sttbf
466	1D2	fcPlcfendRef	FC	file offset of PlcfendRef which points to endnote references in the main document stream
470	1D6	lcbPlcfendRef	long	
474	1DA	fcPlcfendTxt	FC	file offset of PlcfendRef which points to endnote text in the endnote document stream which corresponds with the plcfendRef
478	1DE	lcbPlcfendTxt	long	
482	1E2	fcPlcffldEdn	FC	offset to PLCF of field positions in the endnote subdoc
486	1E6	lcbPlcffldEdn	long	
490	1EA	fcPlcfpgdEdn	FC	offset to PLCF of page boundaries in the endnote subdoc.
494	1EE	lcbPlcfpgdEdn	long	
498	1F2	fcUnused3	FC	
502	1F6	lcbUnused3	long	
506	1FA	fcSttbfRMark	FC	offset to STTBF that records the author abbreviations for authors who have made revisions in the document.
510	1FE	lcbSttbfRMark	long	
514	202	fcSttbfCaption	FC	offset to STTBF that records caption titles used in the document.
518	206	lcbSttbfCaption	long	
522	20A	fcAutoCaption	FC	
526	20E	lcbAutoCaption	long	
530	212	fcPlcfwkb	FC	offset to PLCF that describes the boundaries of contributing documents in a master document
534	216	lcbPlcfwkb	long	
538	21A	fcUnused4	FC	
542	21E	lcbUnused4	long	
546	222	fcPlcftxbxTxt	FC	offset in doc stream of PLCF that records the beginning CP in the text box subdoc of the text of individual text box entries
550	226	lcbPlcftxbxTxt	long	
554	22A	fcPlcffldTxbx	FC	offset in doc stream of the PLCF that records field boundaries recorded in the textbox subdoc.
558	22E	lcbPlcffldTxbx	long	
562	232	fcPlcfHdrtxbxTxt	FC	offset in doc stream of PLCF that records the beginning CP in the header text box subdoc of the text of individual header text box entries
566	236	lcbPlcfHdrtxbxTxt	long	
570	23A	fcPlcffldHdrTxbx	FC	offset in doc stream of the PLCF that records field boundaries recorded in the header textbox subdoc.
574	23E	lcbPlcffldHdrTxbx	long	
578	242	fcStwUser	FC	Macro User storage
582	246	lcbStwUser	long	
586	24A	fcSttbttmbd	FC	
590	24E	lcbSttbttmbd	long	
594	252	fcPlcunused	FC	
598	256	lcbUnused	long	
602	25A	fcpgdMother.fcPgd	FC	

606	25E	fcpgdMother.lcbPgd	long
610	262	fcpgdMother.fcBkd	FC
614	266	fcpgdMother.lcbBkd	long
616	26A	fcpgdFtn.fcPgd	FC
620	26E	rgfcpgdFtn.lcbPgd	long
624	272	rgfcpgdFtn.fcBkd	FC
628	276	rgfcpgdFtn.lcbBkd	long
632	27A	fcpgdFtn.fcPgd	FC
636	27E	rgfcpgdFtn.lcbPgd	long
640	282	rgfcpgdFtn.fcBkd	FC
644	286	rgfcpgdFtn.lcbBkd	long
648	28A	fcSttbfIntlFld	FC
652	28E	lcbSttbfIntlFld	long
656	292	fcRouteSlip	FC
660	296	lcbRouteSlip	long
664	29A	fcSttbSavedBy	FC
668	29E	lcbSttbSavedBy	long
672	2A2	fcSttbFnm	FC
676	2A6	lcbSttbFnm	long

cbFIB is 682.
cwFIB is 341.

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).

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 **CHPX**s 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 **CHPX**s.

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 (ie. 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 seven 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 six bytes of the BX is a PHE structure which stores the current paragraph height for the paragraph corresponding to the BX. If a plcfphe has an entry that maps to the FC for this paragraph, that entry's PHE overides 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 11 if CHPX/PAPX is already recorded and is reduced by 11+sizeof(PAPX) if property is not already recorded.
511-sizeof(grppapx)	grppapx	array of bytes			grppapx consists of all of the **PAPX**s 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 **PAPX**s.

Line Spacing Descriptor (LSPD)

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	dyaLine	short			see description of sprmPDyaLine for description of the meaning of dyaLine
2	2	fMultLinespace	short			

cbLSPD is 4.

see description of sprmPDyaLine in the Sprm Definitions section for description of the meaning of dyaLine and fMultLinespace fields.

Outline LiST Data (OLST)

b_{10}	b_{16}	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	rgch[64]	array of 64 chars			text before/after number

cbOLST is 212(decimal), D4(hex).

Page Descriptor (PGD)

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	*	int	:5	001F	
		fGhost	int	:2	0060	redefine fEmptyPage and fAllFtn. true when blank page or footnote only page
		*	int	:9	FF10	
0	0	fContinue	int	:1	0001	1 only when footnote is continued from previous page
		fUnk	int	:1	0002	1 when page is dirty (ie. pagination cannot be trusted)
		fRight	int	:1	0004	1 when right hand side page
		fPgnRestart	int	:1	0008	1 when page number must be reset to 1.
		fEmptyPage	int	:1	0010	1 when section break forced page to be empty.
		fAllFtn	int	:1	0020	1 when page contains nothing but footnotes
		fColOnly	int	:1	0040	
		fTableBreaks	int	:1	0080	
		fMarked	int	:1	0100	
		fColumnBreaks	int	:1	0200	
		fTableHeader	int	:1	0400	
		fNewPage	int	:1	0800	
		bkc	int	:4	F000	section break code
2	2	lnn	uns			line number of first line, -1 if no line numbering
4	4	pgn	uns short			page number as printed

cbPGD (count of bytes of PGD) is 6(decimal),6(hex).

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

Paragraph Height (PHE)

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	fSpare	int	:1	0001	reserved
		fUnk	int	:1	0002	phe entry is invalid when == 1
		fDiffLines	int	:1	0004	when 1, total height of paragraph is known but lines in paragraph have different heights.
		*	int	:5	00F8	reserved
		clMac	int	:8	FF00	when fDiffLines is 0 is number of lines in paragraph
2	2	dxaCol	int			width of lines in paragraph
4	4	dylLine	int			when fDiffLines is 0, is height of every line in paragraph.in pixels
4	4	dylHeight	uns			when fDiffLines is 1, is the total height in pixels of the paragraph

cbPHE (the count of bytes in a PHE) is 6 (decimal), 6(hex).

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 Windows Word file is created, the **PHE** of every **papx fkp** entrycreated to describe the paragraphs of the file should be set to 0. If a Windows 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.

Paragraph Properties (PAP)

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	istd	uns char			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	int	:1	0001	
		fBrLnBelow	int	:1	0002	
		fUnused	int	:2	0006	reserved
		pcVert	int	: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 coortindtes 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	int	: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 superceded by the newly defined brcLeft, brcTop, etc. fields. They remain in the PAP for compatibility with MacWord 3.0 */				
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	nLvlAnm	uns char	auto list numbering level (0 = nothing)
11	B	fNoLnn	uns char	no line numbering for this para. (makes this an exception to the section property of line numbering)
12	C	fSideBySide	uns char	when 1, paragraph is a side by side paragraph
14	E	dxaRight	int	indent from right margin (signed).
16	10	dxaLeft	int	indent from left margin (signed)
18	12	dxaLeft1	int	first line indent; signed number relative to dxaLeft
20	14	lspd	LSPD	line spacing descriptor
24	18	dyaBefore	uns	vertical spacing before paragraph (unsigned)
26	1A	dyaAfter	uns	vertical spacing after paragraph (unsigned)
28	1C	phe	PHE	height of current paragraph.
34	22	fAutoHyph	uns char	when 1, text in paragraph may be auto hyphenated
35	23	fWidowControl	uns char	when 1, Word will prevent widowed lines in this paragraph from being placed at the beginning of a page
36	24	fInTable	char	when 1, paragraph is contained in a table row
37	25	fTtp	char	when 1, paragraph consists only of the row mark special character and marks the end of a table row.
38	26	ptap	*TAP*	used internally by Word
40	28	dxaAbs	int	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 refence 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
42	2A	dyaAbs	int			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.
44	2C	dxaWidth	int			when not == 0, paragraph is constrained to be dxaWidth wide, independent of current margin or column setings.
46	2E	brcTop	BRC	d		specification for border above paragraph
48	30	brcLeft	BRC			specification for border to the left of paragraph
50	32	brcBottom	BRC			specification for border below paragraph
52	34	brcRight	BRC			specification for border to the right of paragraph
54	36	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.
56	38	brcBar	BRC			specification of border to place on outside of text when facing pages are to be displayed.
58	3A	dxaFromText	int			horizontal distance to be maintained between an absolutely positioned paragraph and any non-absolute positioned text
60	3C	dyaFromText	int			vertical distance to be maintained between an absolutely positioned paragraph and any non-absolute positioned text
62	3E	wr	char			Wrap Code for absolute objects
63	3F	fLocked	char			when 1, paragraph may not be editted
64	40	dyaHeight	int	:15	7FFF	height of abs obj; 0 == Auto
		fMinHeight	int	:1	8000	0 = Exact, 1 = At Least
66	42	shd	SHD			shading
68	44	dcs	DCS			drop cap specifier (see DCS definition)
70	46	anldPap	ANLD			autonumber list descriptor (see ANLD definition)
122	7A	itbdMac	int			number of tabs stops defined for paragraph. Must be >= 0 and <= 50.
124	7C	rgdxaTab	int[itbdMax]			array of positions of itbdMac tab stops. itbdMax == 50
224	E0	rgtbd	char[itbdMax]			array of itbdMac tab descriptors

cbPAP (count of bytes of PAP) is 274 (decimal), 112(hex)

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

Paragraph Property Exceptions (PAPX)

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	cw	byte			count of words of following data in PAPX. The first byte of a PAPX is a count of words when PAPX is stored in an FKP. Count of words is used because PAPX in an FKP can contain paragraph **and** table sprms.
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.
1	1	istd	byte			index to style descriiptor of the style from which the paragraph inherits its paragraph and character properties
3	3	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.

papx.cw is equal to (3 + sizeof(grpprl) + 1) / 2. If the size of the grpprl is odd, a byte of zero is stored immediately after the grpprl to pad the PAPX so its length in bytes is papx.cw * 2.

Picture Descriptor (PIC)

b_{10}	b_{16}	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.
4	4	cbHeader	unsigned			number of bytes in the PIC (to allow for future expansion).
6	6	mfp.mm	int			
8	8	mfp.xExt	int			
10	A	mfp.yExt	int			
12	C	mfp.hMF	int			

If a Windows metafiles is stored immediatelly 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..

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	int			horizontal measurement in twips of the rectangle the picture should be imaged within.
30	1E	dyaGoal	int			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

32	20	mx	uns			horizontal scaling factor supplied by user expressed in .001% units.
34	22	my	uns			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.

36	24	dxaCropLeft	int			the amount the picture has been cropped on the left in twips.
38	26	dyaCropTop	int			the amount the picture has been cropped on the top in twips.
40	28	dxaCropRight	int			the amount the picture has been cropped on the right in twips.
42	2A	dyaCropBottom	int			the amount the picture has been cropped on the bottom in twips.
44	2C	brcl	int	: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	int	:1	0010	picture consists of a single frame
		fBitmap	int	:1	0020	==1, when picture is just a bitmap
		fDrawHatch	int	:1	0040	==1, when picture is an active OLE object
		fError	int	:1	0080	==1, when picture is just an error message
		bpp	int	:8		bits per pixel 0 unknown

						1 monochrome 4 VGA
46	2E	brcTop	BRC			specification for border above picture
48	30	brcLeft	BRC			specification for border to the left of picture
50	32	brcBottom	BRC			specification for border below picture
52	34	brcRight	BRC			specification for border to the right of picture
54	36	dxaOrigin	int			horizontal offset of hand annotation origin
56	38	dyaOrigin	int			vertical offset of hand annotation origin
58	3A	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	int	:1	0001	when 1, means that piece contains no end of paragraph marks.
		fPaphNil	int	:1	0002	used internally by Word
		fCopied	int	:1	0004	used internally by Word
		*	int	:5		
1	1	fn	int	: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 subtracting rgcp[**i**] of the containing **plcfpcd** from its rgcp[**i+1**].
6	6	prm	PRM			contains either a single sprm or else an index number of the grpprl which contains the sprms that modify the properties of the piece.
8	8	cbPCD				

Plex of CPs stored in File (PLCF)

offset (in decimal)	field	type	comment
0	rgfc	FC[]	given that the size of PLCF is cb and the size of the structure stored in plc is cbStruct, then the number of structure instances stored in PLCF, iMac is given by (cb -4)/(4 + cbStruct) The number of FCs stored in the PLCF will be iMac + 1.
4*(iMac+1)	rgstruct	struct[]	array of some arbitrary structure.

cbPLC (count of bytes of a PLC) is iMac(4 + cbStruct) + 4.

The **PRM** has two variants. In the first variant, the **PRM** records a single one or two byte **sprm** whose opcode is less than 128.

Property Modifier(variant 1) (PRM)

b10	b16	field	type	size	bitfield	comment
0	0	fComplex	int	:1	0001	set to 0 for variant 1
		sprm	int	:7	00FE	sprm opcode
		val	int	:8	FF00	sprm's second byte if necessary

In the second variant, prm.fComplex is 1, and the rest of the structure records an index to a grpprl stored in the **CLX** (described in **Complex File Format** topic).

Property Modifier(variant 2) (PRM)

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

Section Descriptor (SED)

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	fSwap	int	:1	0001	runtime flag, indicates whether orientation should be changed before printing. 0 indicates no change, 1 indicates orientation change.
		fUnk	int	:1	0002	used internally by Windows Word
		fn	int	:14	FFFC	used internally by Windows Word
2	2	fcSepx	FC			file offset 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	int			used internally by Windows Word
8	8	fcMpr	FC			points to offset in FC space where the Macintosh Print Record for a document created on a Mac will be stored

cbSED is 12 (decimal)), C (hex).

Section Properties (SEP)

b_{10}	b_{16}	field	type	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	ccolM1	int	number of columns in section - 1.
4	4	dxaColumns	uns	distance that will be maintained between columns
6	6	fAutoPgn	char	only for Mac compatability, used only during open, when 1, sep.dxaPgn and sep.dyaPgn are valid page number locations
7	7	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)
8	8	pgnStart	uns	user specified starting page number.
10	A	fUnlocked	uns char	set to 1, when a section in a locked document is unlocked
11	B	cnsPgn	uns char	chapter number separator for page numbers
12	C	fPgnRestart	uns char	set to 1 when page numbering should be restarted at the beginning of this section
13	D	fEndNote	uns char	when 1, footnotes placed at end of section. When 0, footnotes are placed at bottom of page.
14	E	lnc	char	line numbering code: 0 Per page 1 Restart 2 Continue
15	F	grpfIhdt	char	specification of which headers and footers are included in this section. See explanation in Headers and Footers topic.
16	10	nLnnMod	uns	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.)
18	12	dxaLnn	int	distance of
20	14	dyaHdrTop	uns	y position of top header measured from top edge of page.
22	16	dyaHdrBottom	uns	y position of top header measured from top edge of page.
24	18	dxaPgn	int	when fAutoPgn ==1, gives the x position of auto page number on page in twips (for Mac compatabilty only)
26	1A	dyaPgn	int	when fAutoPgn ==1, gives the y position of auto page number on page in twips (for Mac compatabilty only)
28	1C	fLBetween	char	when ==1, draw vertical lines between columns
29	1D	vjc	char	vertical justification code 0 top justified 1 centered 2 fully justified vertically 3 bottom justified

30	1E	lnnMin	int	beginning line number for section
32	20	dmOrientPage	uns char	orientation of pages in that section. set to 0 when portrait, 1 when landscape
33	21	iHeadingPgn	char	heading number level for page number
34	22	xaPage	uns	default value is 12240 twips width of page
36	24	yaPage	uns	default value is 15840 twips height of page
38	26	dxaLeft	uns	default value is 1800 twips left margin
40	28	dxaRight	uns	default value is 1800 twips right margin
42	2A	dyaTop	int	default value is 1440 twips top margin
44	2C	dyaBottom	int	default value is 1440 twips bottom margin
46	2E	dzaGutter	uns	default value is 0 twips gutter width
48	30	dmBinFirst	uns	bin number supplied from windows printer driver indicating which bin the first page of section will be printed.
50	32	dmBinOther	uns	bin number supplied from windows printer driver indicating which bin the pages other than the first page of section will be printed.
52	34	dmPaperReq	uns short	dmPaper code for form selected by user
54	36	fEvenlySpaced	char	when == 1, columns are evenly spaced. Default value is 1.
55	37			reserved
56	38	dxaColumnWidth	uns	used internally by Word
58	3A	rgdxaColumnWidthSpacing	array of XA	array of 89 Xas that determine bounds of irregular width columns
236	EC	olstAnm	OLST	multilevel autonumbering list data (see OLST definition)

cbSEP (count of bytes of SEP) is 448(decimal), 1C0(hex).

The standard SEP is all zeros except as follows:

bkc	2
dyaPgn	720 twips (equivalent to .5 in)
dxaPgn	720 twips
fEndnote	1 (True)
fEvenlySpaced	1 (True)
xaPage	12240 twips
yaPage	15840 twips
dyaHdrTop	720 twips
dyaHdrBottom	720 twips
dmOrientPage	1 (portrait orientation)

Section Property Exceptions (SEPX)

b_{10}	b_{16}	field	type	size	bitfield	comment
0	0	cb	byte			count of bytes in remainder of **SEPX.**
1	1	grpprl	char[]			list of sprms that encodes the differences between the properties of a section and Word's default section properties.

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

Tab Descriptor (TBD)

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	jc	int	:3	07	justification code 0 left tab 1 centered tab 2 right tab 3 decimal tab 4 bar
		tlc	int	:3	38	tab leader code 0 no leader 1 dotted leader 2 hyphenated leader 3 single line leader 4 heavy line leader
		*	int	:2	C0	reserved

cbTBD (count of bytes of a tab descriptor) is 1.

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

Table Cell Descriptors (TC)

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	rgf	int			
0	0	fFirstMerged	int	: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	int	:1	0002	set to 1 when cell has been merged with preceding cell.
		fUnused	int	:14	FFFC	reserved
2	2	rgbrc	BRC[cbrcTc]			notational convenience for referring to brcTop, brcLeft, etc fields.
2	2	brcTop	BRC			specification of the top border of a table cell
4	4	brcLeft	BRC			specification of left border of table row
6	6	brcBottom	BRC			specification of bottom border of table row
8	8	brcRight	BRC			specification f right border of table row.

cbTC (count of bytes of a TC) is 10(decimal), A(hex).

Table Autoformat Look sPecifier ()

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	itl	short			index to Word's table of table looks
2	2	fBorders	int	:1	0001	when ==1, use the border properties from the selected table look
		fShading	int	:1	0002	when ==1, use the shading properties from the selected table look
		fFont	int	:1	0004	when ==1, use the font from the selected table look
		fColor	int	:1	0008	when ==1, use the color from the selected table look
		fBestFit	int	:1	0010	when ==1, do best fit from the selected table look
		fHdrRows	int	:1	0020	when ==1, apply properties from the selected table look to the header rows in the table
		fLastRow	int	:1	0040	when ==1, apply properties from the selected table look to the last row in the table
		fHdrCols	int	:1	0080	when ==1, apply properties from the selected table look to the header columns of the table
		fLastCol	int	:1	0100	when ==1, apply properties from the selected table look to the last column of the table

Table Properties (TAP)

b_{10}	b_{16}	field	type	size	bitfield	comments
0	0	jc	int			justification code. specifies how table row should be justified within its column. 0 left justify 1 center 2 right justify
2	2	dxaGapHalf	int			measures half of the white space that will be maintained between text in adjacent columns of a table row. A dxaGapHalf width of white space will be maintained on both sides of a column boundary.
4	4	dyaRowHeight	int			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.
6	6	fCantSplit	uns char			when 1, table row may not be split across page bounds
7	7	fTableHeader	uns char			when 1, table row is to be used as the header of the table
8	8	tlp	TLP			table look specifier (see TLP definition)
12	C	fCaFull	int	:1	0001	used internally by Word
		fFirstRow	int	:1	0002	used internally by Word
		fLastRow	int	:1	0004	used internally by Word
		fOutline	int	:1	0008	used internally by Word
		*	int	:12	FFE0	reserved
14	E	itcMac	int			count of cells defined for this row. ItcMac must be >= 0 and less than or equal to 32.
16	10	dxaAdjust	int			used internally by Word
18	12	rgdxaCenter	int[itcMax + 1]			rgdxaCenter[0] is the left boundary of cell 0 measured relative to margin.. rgdxaCenter[tap.itcMac - 1] is left boundary of last cell. rgdxaCenter[tap.itcMac] is right boundary of last cell.
84	54	rgtc	TC[itcMax]			array of table cell descriptors
404	194	rgshd	SHD[itcMax]			array of cell shades
468	1D4	rgbrcTable	BRC[6]			array of border defaults for cells

cbTAP (count of bytes of a TAP) is 480 (decimal), 1E0(hex).

Appendix A - Changes from version 1.x to 2.0

Changes to Structures

BRC

The previously defined BRC was renamed BRC10, and a new BRC was defined with new fields and field names.

CHP

The size of the CHP changed from 16 to 32 bits, with some spare bits added.

The fStrike, hpsPos, & fSysVanish fields were moved within the CHP. A new field, fRMarkDel, is located where fStrike used to be.

The fsLid and lid fields were added for the language identification code.

The types of several fields were changed. The ftc field was changed from an unsigned integer to a WORD. The hps field was changed from an unsigned char to a WORD. The fnPic field was changed from an unsigned integer to a BYTE.

The fObj and fcObj fields were added for managing embedded objects.

DOP

fWide removed
irmBar is a BYTE rather than an int
rgwSpare uns[2] became wSpare2 uns and wSpare3 uns
fPMHMainDoc, grfSuppression,fKeepFileFormat, fDfltTrueType, and fPagSuppressTopSpacing added

DTTM

FIB

Password Protection added
    fEncrypted and lKey added for file encryption
Print Environment & orientation changes
    fcPrEnv & cbPrEnv were removed.
    fcPrDrv & cbPrDrv----------\
    fcPrEnvPort & cbPrEnvPort ------- were added to FIB
    fcPrEnvLand & cbPrEnvLand----/
Autosave added
    fcAutosaveSource
    cbAutosaveSource
nLocale changed to lid

_OBJHEADER

PAP

Frames
    added dyaFromText, wr, dyaHeight, fMinHeight
    When converting 1.x documents with Absolutely Positioned Objects set the old dxaFromText (Distance from text) to both dxaFromText and dyaFromText.
Shading
    added shd
Auto numbering
    added nfcSeqNumb and nnSeqNumb

PIC

(at the end of the structure before the variable length array)
brcTop BRC
brcLeft BRC
brcBottom BRC
brcRight BRC
dxaOrigin, dyaOrigin

SEP

removed fAutoPgn changed to bUnused1
Added Page Orientation stuff
    morPage
    bUnused2
Added Printer Environment
    dmBinFirst
    dmBinOther

DOP to SEP

Page Dimensions & Margin stuff
    xaPage
    yaPage

dxaLeft
dxaRight
dyaTop
dyaBottom
dxaGutter in DOP renamed dzaGutter in SEP

SED

fSpare (reserved) changed to fSwap (runtime flag for landscape/portrait orientation)

TAP

wSpare1
wSpare2
wSpare3
wSpare4
wSpare5

TAP

Shading

rgshd[itchMax] SHD

TC

Border

rgbrc, brcTop, brcLeft, brcBottom, brcRight were int, now they are BRC.

Other changes

sttbfAssoc

Indices to the associated string table and descriptions of strings were added.

sttbfFn

The fonts written in the font string table and the indexing were changed.

REVIEW DavidLu

FonT Code Link field (FTCL)

b_{10}	b_{16}	field	type	size	bitfield	comments
12	b	fEmbedLoad	int	:1	0001	1 if embedded fonts were stored in the file.
		wLicense	int	:3	000e	Licensing permissions 0 - font is installable 4 - font is print preview 8 - font is editable

Index of Changes from version 1.x to 2.0