APPLIX

Words Technical Reference Guide

This manual was produced using Applixware.

Printed: April 2000

Contents

Tables

Preface

About This Manual

This manual describes aspects of Applix Words that advanced ELF programmers may need to use. The topics discussed are:

- The Applix Words file format. This is the format used to store an ASCII representation of a Words file on disk.
- Technical terms. This chapter is a high-level glossary.
- Fields. This contains a definition of every field that can be inserted into an Applix Words file.

Conventions Used in This Manual

The following typeface conventions are used throughout this manual:

Helvetica	Helvetica text indicates that this option or object appears in the document window. For example, "Type the name of the document in the File name entry area." File names and directories are also indicated by Helvetica text. For example, "The new

Applixware version is installed in your axlocal directory."

Applixware keys are printed in a Helvetica uppercase typeface. For example, "Press the TAB key."

Helvetica Bold

Bold Helvetica text indicates an option to choose or text to type. It usually appears in numbered steps as shown in the following example:

1. Type **2.5** in the Line spacing entry area.
2. Click **Apply**.

Italics

Words are italicized for emphasis or to draw your attention to a new term. For example, "*Do not* press the RETURN key," or "This action is called *word wrapping*."

Italic type is also used to indicate variable information, as in "Put Applixware in the /user/*your_name* directory."

Menu Name → Option Name	Whenever you see a reference to a menu option, the option is identified using the following notation: Menu Name → Option Name For example, "Choose **File → Save**."
OK and Apply	When numbered instructions are included in the text, we omit the final "Click **OK** or **Apply**" statement for brevity.

Applixware Window Environment and Interface

Consult your window manager and hardware documentation if you need information about how to operate in your window environment.

1 File Format

This chapter describes the generic ASCII file format of all Version 4.3 and later Applixware documents and the Applix Words internal ASCII file format. These formats consist of statements and properties that define all of the text, graphics, formatting, and layout constructs of an Applixware document.

The following topics are described:

- An explanation of the document conventions used in this chapter.
- A description of the Version 4.3/4.4/5.0 file format for Applix Words.
- A description of the supported Applix Words statements and their properties.

Statement Syntax

In the following listing of statements, these conventions apply:

`UPPERCASE`	Courier uppercase indicates an actual statement. At this point, turn to the "Statements Reference" later in this chapter to look up the exact syntax of the statement. Example: `DOCUMENT_BEGIN` is a self-contained statement and may be found in the "Statements Reference."
`italic`	Courier italic indicates a statement group. When you reach an italic entry, look further down the list to see the statements that make up the group. Example: The *`document`* group outlines the overall structure of the document file. Within the *`document`* structure, the *`definitions`* entry indicates a group of statements that defines items for the entire document. Look under *`definitions`* in the left column for the statements that make up the *`definitions`* group.
{ *`braces`* }	Braces indicate a required entry; that is, you must choose one of the indicated elements.
[*`brackets`*]	Brackets indicate an optional entry, which may or may not appear in the document.

	Example: [*`object_definition`*] This entry appears in the file only if one or more objects are included in the document.
`...` *(ellipse)*	Any entry followed by an ellipse (`...`) indicates that the statement or group may be repeated any number of times. Example: *`paragraph...`* This shows that the statement group can contain any number of paragraphs.
`vertical|bar`	A vertical bar between items indicates that the document entry includes one of the options. Example: *`object_definition`* `= OBJECT|GRAPHICS` This indicates that the *`object_definition`* section can include one of the keywords `OBJECT` or `GRAPHICS`.

File Header

The first line of the file is a header line that identifies the file as an Applixware file, specifies the encoding method and gives the Applixware version number.

The file header is:

```
*BEGIN1 datatype VERSION=version
       [ ENCODING=encoding] [C=content]
       [future data elements]
```

The last line of the file is

```
*END datatype
```

The following defines these elements:

content =	This field depends on the datatype field. It gives further information about the content of the file.
datatype =	`WORDS	GRAPHICS	SPREADSHEETS	QUERYDATA	MACROS	EQUATIONS	ASCII	BINARY` The significance of *datatype* is that you know how to process each of these types of contents. The data in the Applixware file is encoded to prevent accidental asterisk (*) lines from occurring. Datatype sections that are unknown to this version are skipped. ASCII and BINARY are two different encodings for arbitrary data, such as local foreign insets in an Applix Words application. This is used for foreign insets. ASCII provides a base level of readability in the Applixware file; however, it does not provide any more or less information than BINARY.
version =	Should be 430 for a file in Release 4.3 format or 440 for a file in Release 4.4 format (both are documented here). 311 was used for a file in Release 3 format.

[1] Was *START in releases prior to Release 4.

Embedded Data

For embedded data, use `*BEGIN` and `*END` to delimit the data.

Links

For each external link in the file, a `LINK` statement must appear somewhere in the file to permit a simple scan for links.

```
*LINK pathname
```

Maximum Line Length

Programs that write files should write short lines (at most 80 characters) for the convenience of people using text editors. Programs that read files should be able to handle lines as long as 4090 characters. If a statement needs to be longer than the desired line length, the continuation line format is used.

Continuation Lines

An application is free to define how continuation lines are handled within its data segment. The one restriction is that the first character of a line is never an asterisk.

In Words, if a statement is longer than the standard line length (80 characters for files written by Words), it is broken into multiple lines. A backslash is added to the end of each line (except for the last line). A space is added to the start of each line after the first line. The lines are rejoined and these backslashes and spaces are discarded when Words reads the file.

NOTE: If a Words document contains embedded Applixware objects, the application that owns the object determines the continuation line format for that object.

Wide and Multi-Byte Characters

8-bit and 16-bit characters should, be default, be encoded in the output file so that the files are 8-bit clean (that is, the file contains only 7-bit characters). Such files will have the token "`ENCODING=7BIT`" in their `*BEGIN` header lines (as would Applixware objects embedded within Applixware files).

Here are the algorithms for folding 8- and 16-bit characters into 7 bits.

For both algorithms, the carat character ’^’ is used as an escape character to signal the start of an 8- or 16-bit character. To output a real carat character, escape it with another carat (i.e. double it). For example, given the following string in a Words document:

```
Here is a single carat: ^
```

The following line would appear in the Words file:

```
<Text "Here is a single carat: ^^">
```

8-bit characters

Assume a hexadecimal system where ’a’ represents 0, ’b’ is 1, ... ’p’ is 15. An 8-bit character is represented as a carat character followed by a pair of these "hex letters". For example, the tab character (character code 9) is represented as "^aj", where a=0, j=9, and thus (0 * 16) + 9 = 9. The (US) cents character ("¢"), which is code 162, is represented as "^kc", where k=10, c=2, and thus (10 * 16) + 2 = 162.

Any 8-bit character larger or equal to 128, or smaller than 32 (but excluding character 10, the line break) is output in this format. Character code 10 is output as "\n". Any other character is output in its ASCII representation.

16-bit characters

For character codes less than 256, the 8-bit system is used as described above. For character codes greater than 256, a system is used where space is 0, '!' is 1, ... underscore is 63. However, the backquote (code 96) is used to represent 2 instead of the double quote. The overall representation is of the form ^XYY, where X is a 6-bit value in this system, and Y is a 5-bit value. Thus the largest possible 16-bit code (65535) would be represented as "^_??", where _=63, ?=31, and thus (63 * 1024) + (31 * 32) + 31 = 65535. The character code 256 is not used in this system.

Note there is no intersection between the codes used to represent 8-bit values (a-p) and the codes used to represent 16-bit values (space to underscore, plus back quote). Following an unescaped carat you should either see a pair of 8-bit codes or a triple of 16-bit codes.

Escape characters

An application is free to support escape characters, in addition to the carat as described above.

In a Words file, the backslash character '\' is used as an escape character within quoted strings. Backslashes and double quotes must be escaped. "\n" represents a line break.

The following table describes the supported escape sequences in quoted strings within a Words file:

Table 1-1 Escape Sequences in Quoted Strings

Appearance in File	Meaning
\\	Backslash within string
\"	Double quote within string
\n	Newline within string

Comment Lines

Lines that start with ** are comments; they have no inherent semantics. However, they are treated as significant in that applications will preserve them.

A comment line can contain any legal Applix character. The comment cannot be longer than one line.

Comments can be inserted at the beginning of any line within a document. When a document is stored by an Applix application, all Applixware comments are written to the stored image immediately following the `*BEGIN` line of the document. Comments within embedded data remain within that data and are not moved to the start of the file.

Third party developers and users can attach their own semantics and additional syntax to comments. Applix strongly suggests that developers adhere to the following conventions:

- A structured comment will consist of the following sequence:

  ```
  ** "keyword" string
  ```

  `keyword` is an agreed upon sequence of case sensitive characters enclosed in quotes and *`string`* is any sequence of legal characters terminated by the end of the line.

- All Applix keywords begin with one of the following strings: ax, Ax, or AX. These strings are reserved for Applix's use.

Applix Words Statements

This section describes Applix Words statements. Most statements consist of:

1. A left angle bracket
2. A keyword
3. A sequence of items
4. A terminating right angle bracket

When an item appears, it must appear in the order specified. For example:

```
< keyword
   item1
   item2
>
```

Items

There are three types of items that can be used in statement syntax:

`Literal text`	Literal text is indicated by Courier text. It is to be specified exactly as shown.
`Variables`	Variables are represented by Courier italic text, followed by the type of the variable in parenthesis. A number or string of the appropriate type should be provided.
`Property reference`	A *property reference* is represented by mixed-case Courier italic. The property is either described immediately below the statement syntax description

or in the "Supported Properties" section later in this chapter.

Unless otherwise indicated, the items within the statements are optional and have a default value when not specified.

Example Statement Definition

```
< ice_cream                /* abbrev: i_c */
   jimmies|nuts
   scoops:numscoops(int)
   flavor:flavorName
>
flavorName = "Vanilla"|"Chocolate"
```

The following table describes each of the components of the statements:

Table 1-2 Statement Elements

Element	Meaning
`<`	Opening bracket
`ice_cream`	May substitute a short form here: for example, `i_c`
`jimmies | nuts`	Can specify either of these literals
`scoops:`*`numscoops(int)`*	Literal followed by variable
`flavor:`*`flavorName`*	Mixed case, no type: must be a property
`>`	Closing bracket
`flavorName` `= "Vanilla" | "Chocolate"`	The property definition. If the property definition is not included immediately below the statement definition, look up the property in the "Supported Properties" section later in this chapter.

Using this definition, we can derive the following legal statements:

```
<ice_cream>
```

Uses all defaults.

```
<i_c scoops:1>
```

Uses the keyword abbreviation.

```
<ice_cream jimmies scoops:2 flavor:"Vanilla">
```

Includes a literal (jimmies), a variable (scoops), and a property (flavor). Spaces around the colons are optional.

If an item is omitted, default values are used. The default is either described in a comment or can be assumed to be one of the following:

- Zero
- Not present
- FALSE

Variable Types

The following variable types are used in this chapter:

Table 1-3 Variable Types

Type	Meaning
float	Floating point number. This is a number greater than or equal to zero unless otherwise indicated.
int	An integer. This is a number greater than or equal to zero unless otherwise indicated.

Table 1-3 Variable Types (cont.)

Type	Meaning
mils	Units of 1/1000th of an inch. This is a positive integer unless otherwise indicated.
points	Text size (72 points to the inch). This should be a positive integer.

string	A text string in double quotes; for example, `"Vanilla"`. The backslash (\) is used as an escape character in quoted strings in a Words file. Double quotes (") must be escaped, as are backslashes used in the text. The newline character (code 10) is represented as "`\n`". See "Wide and Multi-byte Characters" earlier in this chapter for a discussion of the treatment of other 8-bit and special characters.
	
	As an example, given the following string in a Words document:
	`The character "\" is a backslash`
	The following line would appear in the Words file:
	`<Text "The character \"\\\" is a backslash">`

All other units are described in context.

Sequence of Statements in Words

The following example displays the sequence of statements in a Words document. These statements are explained in detail in the sections following this example, listed in alphabetical order:

```
document = DOCUMENT_BEGIN
         definitions
         FLOW_BEGIN
         [ contents ]...
         paragraph   /* Minimum flow contents is one
                        PARA */
         SECTION
         FLOW_END
         [ secondary_flow ]...
         [ object_definition ]...
         variables
         DOCUMENT_END

definitions =  STYLES_BEGIN
         STYLE            /* Top-level style */
         [ STYLE|SERIES|COLOR|glossary ]...
         STYLES_END

glossary =     GLOSSARY_BEGIN
         [ contents|material ]...
         GLOSSARY_END

contents =     paragraph|row

paragraph = [ material ]...
         PARA

material =     field|TEXT|PICTURE|break|MARKER2|
         ERROR|EQUATION

row =    ROW_START
         cell...          /* Last cell specially
                             marked as such */
cell =   paragraph.../* No frames or
```

[2] MARKER beads did not exist until release 4.

```
                               section/column/page breaks
                               allowed */
                            /* Cell must end with PARA */
           CELL_END

field =    FIELD_BEGIN
           material         /* Field method - defines field;
                               see Chapter 3, "Field Method
                               Syntax." */

           LINK             /* Required for linked objects;
                               omit for others */
           FIELD_VALUE
                            /* Field value - contents of
                              field */
           [ contents|material ]...
           FIELD_END

break =    LINE_BREAK|COLUMN_BREAK|SECTION|PAGE_BREAK

secondary_flow =
           hdr_ftr|footnote

object_definition =
           OBJECT|GRAPHICS

variables =     VARS_BEGIN
           [ DOC_VARIABLE ]...
           VARS_END

hdr_ftr =HDRFTR_BEGIN
           [ contents ]...
           paragraph        /* Must end with PARA */
           HDRFTR_END

footnote =      FOOTNOTE_BEGIN
           field            /* ...of type Footnote
                               Numbering */
           [ contents ]...
           paragraph        /* Must end with PARA */
           FOOTNOTE_END
```

NOTE: The TEXT statements that make up a field method must contain specific strings. Field methods are described in Chapter 3.

CELL_END

```
< cell_end                        /* abbrev: CE */
   width:width(mils)
   [ center|bottom ]
   [ leftCellMargin:margin(mils) ]     /* abbrev: lm */
   [ rightCellMargin:margin(mils) ]   /* abbrev: rm */
   [ topCellMargin:margin(mils) ]     /* abbrev: tm */
   [ bottomCellMargin:margin(mils) ]  /* abbrev: bm */
   [ borderAttributes ]
   [ locked ]
   [ id:cell_id(string) ]
   [ nextId:cell_id(string) ]
   [ doubleClickMacro:macro(string) ]
   [ enterMacro:macro(string) ]
   [ exitMacro:macro(string) ]
   [ maxChars:maxChars(int) ]
   [ nextIfFull ]
   [ case:caseCode(int) ]
   [ promptText:promptText(string) ]
   lastCellInRow                  /* abbrev: last */
>
```

The following table describes the items in the `CELL_END` statement:

Table 1-4 CELL_END Comments

Element	Notes
borderAttributes	Border and shading properties of the cell. See "Supported Statements"
case: *caseCode(int)*	Causes the case of text typed into the cell in Forms mode to be altered as follows: 0 = No change (the default) 1 = Force lowercase 2 = Force uppercase This item did not exist in releases prior to release 4.1, and is meaningful only in Forms mode.

Table 1-4 CELL_END Comments (cont.)

Element	Notes
`center|bottom`	Vertical alignment of material in cell; omit for default (top)
`doubleClickMacro:` *`macro (string)`*	The name of an ELF macro that will be executed when a user double clicks upon a cell in Forms mode This item did not exist in releases prior to release 4, and is meaningful only in Forms mode.
`enterMacro:` *`macro (string)`*	The name of an ELF macro that will be executed when the cursor moves into a cell in Forms mode This item did not exist in releases prior to release 4, and is meaningful only in Forms mode.
`exitMacro:` *`macro (string)`*	The name of an ELF macro that will be executed when the cursor leaves a cell while in Forms mode This item did not exist in releases prior to release 4, and is meaningful only in Forms mode.
`id:` *`cell_id (string)`*	An arbitrary string identifying a cell This item did not exist in releases prior to release 4.
`lastCellInRow`	This must be specified for the last cell (and only the last cell) in each row

Table 1-4 CELL_END Comments (cont.)

Element	Notes
`leftCellMargin, rightCellMargin, topCellMargin, bottomCellMargin:` *`margin (mils)`*	Space between left / right / top / bottom border of the cell and material within the cell
`locked`	This only exists if a cell is locked (which means the cell cannot be selected, entered, or edited in Forms mode) This item did not exist in releases prior to release 4, and is meaningful only in Forms mode.
`maxChars:` *`maxChars(int)`*	Maximum number of characters allowed to be typed into the cell in Forms mode. This item did not exist in releases prior to release 4.1, and is meaningful only in Forms mode.
`nextID:` *`cell_id (string)`*	The id of a cell to which the "Go To Next Cell" command will move the cursor when in Forms mode This item did not exist in releases prior to release 4, and is meaningful only in Forms mode.
`nextIfFull`	If present, and if a `maxChars` value is specified, when the cursor is in the cell in Forms mode and the cell becomes "full" (as defined by `maxChars`), the cursor is moved to the cell with the nextID cell id. This item did not exist in releases prior to release 4.1, and is meaningful only in Forms mode.

Table 1-4 CELL_END Comments (cont.)

Element	Notes
`promptText:` *`promptText(string)`*	When the cursor enters the cell, this text will be displayed on the status line at the bottom of the Words window. This item did not exist in releases prior to release 4.1, and is meaningful only in Forms mode.
`width (mils)`	Overall width of cell, in mils

COLOR

```
< color
   name(string):C:M:Y:K
>
```

Table 1-5 COLOR Comments

Element	Meaning
C:*M*:*Y*:*K*	Required. The *C*, *M*, *Y*, and *K* values are integers between 0 and 255 representing the cyan, magenta, yellow, and black values respectively of the color

COLUMN_BREAK

```
< column_break >
```

DOC_VARIABLE

```
< variable
   name(string)            /* Required */
   elfData
>
```

DOCUMENT_BEGIN

```
*BEGIN WORDS VERSION=vnum ENCODING=encoding
<Applix Words>
< Globals
   levelIndent:indent(mils)
   hyphMethod: hyphCode
   headerMargins:header(mils)
   footerMargins:footer(mils)
   changeBarPos:barPos
   [ facingPages ]
>
```

If the beads were contained in a formatted Macro Editor document, the first statement would be as follows:

```
*BEGIN MACROS VERSION=vnum ENCODING=encoding
```

The following table describes items in the DOCUMENT_BEGIN statement:

Table 1-6 DOCUMENT_BEGIN Comments

Element	Notes
changeBarPos:*barPos*	*barPos* is an integer controlling the placement of change bars in the document. The code numbers corresponding to the WP#CB#POS# defines in wp_.am are as follows: 0 = left margin 1 = right margin 2 = outer margin
facingPages	If present, the document is in "facing pages" mode; the main effect is the treatment of the left and right page margins (see the Section bead, below). Note this does not imply the document will be printed in double-sided mode.

Table 1-6 DOCUMENT_BEGIN Comments (cont.)

Element	Notes
`footerMargins:`*`footer(mils)`*	Distance between the bottom of the page and the bottom of the footer, if any
`headerMargins:`*`header(mils)`*	Distance between the top of the page and the top of the header, if any
`hyphMethod:`*`hyphcode`*	*`hyphCode`* is an integer indicating the method used to hyphenate words in the dictionary. The code numbers corresponding to the WP#HYPHENATE# defines in wp_.am are as follows: 0 = use rules for hyphenation 1 = look in dictionary; if not found, use the rules 2 = don't hyphenate. 3 = look in dictionary only 4 = use profiled value (wpSpellEnglishOpts for English; wpSpellFrenchOpts for French, etc.; profiles use the HYPH_ defines in wp_.am). This option is supported as of release 4.4.
`levelIndent:`*`indent(mils)`*	Extra indent per para level.

DOCUMENT_END

```
< end_document >
*END WORDS
```

If the beads were contained in a formatted Macro Editor document, the last statement would be:

```
*END MACROS
```

EQUATION

See "Equation Statements" later in this chapter for more information.

ERROR

Contains compilation error message for an ELF macro document. This information is not written to or read from a file.

FIELD_BEGIN

```
< start_field              /* abbrev: S_F */
   [ { botPage|endDoc } footnote footNoteId(string) ]
   [ date:dateStamp ]
   [ docType:docTypeCode ]
   [ appType:appTypeCode ]
   [ editable ]
   [ outDated ]
>
```

Table 1-7 FIELD_BEGIN Comments

Element	Meaning
`appType:`*`appTypeCode`*	Only used for linked or embedded object fields; *`appTypeCode`* is described in the `OBJECT` statement
`{ botPage|endDoc } footnote` *`footNoteId(string)`*	Only used for footnote reference fields. *`footNoteId`* is a string that identifies the matching footnote body. `botPage` is used for footnotes that are to be placed at the bottom of the page; `endDoc` for footnotes that are to be placed at the end of the document.

Table 1-7 FIELD_BEGIN Comments (cont.)

Element	Meaning
`date:`*`dateStamp`*	Only used for linked object fields; *`dateStamp`* is the time the linked file was last modified This is the same format as the time returned by DATE_LAST_MODIFIED@
`docType:`*`docTypeCode`*	Only used for linked or embedded object fields; *`docTypeCode`* is described in the `OBJECT` statement
`editable`	If present, the field value can be edited; that is, it is unprotected
`endDoc footnote` *`name(string)`*	Only used for end-of-document footnote reference field
`outDated`	If present, the field value is reevaluated when the document is read

The `FIELD_BEGIN` statement is followed by the field method. See Chapter 3 for information.

FIELD_END

```
< end_field >            /* abbrev: < E_F > */
```

FIELD_VALUE

```
< field_value >          /* abbrev: < FV > */
```

FLOW_BEGIN

```
< start_flow >
```

FLOW_END

```
< end_flow >
```

FOOTNOTE_BEGIN

```
< start_footnote
   footNoteId
>
```

Table 1-8 FOOTNOTE_BEGIN Comments

Element	Meaning
footNoteId	A string that identifies the matching footnote reference field

FOOTNOTE_END

```
< end_footnote >
```

GLOSSARY_BEGIN

```
< start_glossary           /* abbrev: < S_G */
   name(string)
   [ live ]
>
```

Table 1-9 GLOSSARY_BEGIN Comments

Element	Meaning
live	If specified, matching references change when the glossary’s contents change
name(string)	Required; name of glossary

GLOSSARY_END

```
< end_glossary >           /* abbrev: < E_G > */
```

GRAPHICS

```
< graphics
   name(string)
   [ changedFlag ]
>
asciiData
```

The graphics bead is used as a temporary container for the actual graphics data in a linked graphics file. This information is discarded on save and recreated when the document is read back in. Thus the Graphics bead does not usually appear in the Words file.

Table 1-10 GRAPHICS Comments

Element	**Meaning**
asciiData	Contents of graphics
name(string)	Unique name for this data

HDRFTR_BEGIN

```
< start_hdrftr
   hdrftrname(string)
>
```

Table 1-11 HDRFTR_BEGIN Comments

Element	**Meaning**
hdrftrname(string)	This name matches the name in the *hdrFtrReference* statement. The name should be of the format "_AX_HF_#", where "#" is a number; e.g. "_AX_HF_1".

HDRFTR_END

```
< end_hdrftr>
```

LINE_BREAK

```
\n
```

The line break is represented by a backslash-n in the file.

LINK

```
*LINK pathname
```

Table 1-12 LINK Comments

Element	Meaning
pathname	The name of the external file to which you are linking

MARKER

```
<marker
   markername(string)
>
```

Creates a named location in a file. This bead was not written to file during a Save operation in releases prior to release 4.

OBJECT

```
< object
   name(string)
   docType:docTypeCode
   appType:appTypeCode
   [ cvtMacro:macro(string) ]
   [ appMacro:macro(string) ]
   [ allowUnreferenced ]
   [ changedFlag ]
>
asciiData|binaryData       /* Contents of object */
```

Table 1-13 OBJECT Comments

Element	Meaning
`allowUnreferenced`	If present, this object will be preserved at save time even if not referenced in the document. If absent, this object will not be written to the output file if it is not referenced within the document (default behavior).
`appMacro:` *`macro(string)`*	The name of the ELF macro to execute when the object is double-clicked upon (not needed for an Applixware object). Note this value will be overridden by a similar specification in the object field that references this object, if the field has such a specification.
`appType:`*`appTypeCode`*	An integer representing the Applix application matching the object. For example, if the object is a graphics object of any type (e.g. GIF, JPEG, Applixware Graphics), the value will be 2, representing "Applixware Graphics". See the *`appTypeCode`* section below for more information. Note this value will be overridden by a similar specification in the object field that references this object, if the field has such a specification.

Table 1-13 OBJECT Comments (cont.)

Element	**Meaning**
`docMacro:` *`macro(string)`*	The name of the ELF macro to be used when converting the object into Applixware format (not needed for an Applixware object). Note this value will be overridden by a similar specification in the object field that references this object, if the field has such a specification.
`docType:`*`docTypeCode`*	Is an integer representing the origin of the object. The same coding system is used by the RECOGNIZE_FILE@ macro Some of the common codes are: 0 = ASCII 1 = DCA 35 = HTML 66 = GIF 108 = Excel XLS 4.0 200 = Applix Words 201 = Applix Graphics 203 = Applix Spreadsheets 211 = Applix Data See recgfil_.am for more codes. Note this value will be overridden by a similar specification in the object field that references this object, if the field has such a specification.
`name(string)`	This required field matches the name in an object field method

PAGE_BREAK

```
< page_break
   [ odd_page | even_page ]
>
```

Table 1-14 PAGE_BREAK Comments

Element	Meaning
odd_page | even_page	If odd_page is specified, the material following the page break will begin on the next odd page. If even_page is specified, it will begin on the next even page. If nothing is specified, the material will begin on the next page, regardless of whether it is an odd or even page (default behavior).

PARA

```
< para                     /* abbrev: < P */
   styleName(string)
   [ borderAttributes ]
   [ paraAttributes ]
   [ tabDefinitions ]
   [ frameAttributes ]
   [ textAttributes ]
   [ changedFlag ]
>
```

Table 1-15 PARA Comments

Element	Meaning
changedFlag	For change bar operations only, If this attribute is used, it always occurs just before the bead's concluding ">"

Table 1-15 PARA Comments (cont.)

Element	Meaning
styleName	The name of the style controlling the inherited attributes of this paragraph

PICTURE

```
< picture
   name(string)
   [ builtIn ]
   appId:appTypeCode
   [ proportional ]
   [ clipMode:clipModeCode(int)
   [ xScale:xscale(float) ]
   [ yScale:yscale(float) ]
   [ xOffset:xoffset(mils) ]
   [ yOffset:yoffset(mils) ]
   [ srcWidth:srcWidth(mils) ]
   [ srcHeight:srcHeight(mils) ]
   [ picSource:"elfStringArray" ]
   [ width:displayWidth(mils)/ ]
   [ height:displayHeight(mils) ]
   [ changedFlag ]
>
```

Table 1-16 PICTURE Comments

Element	Meaning
`builtIn`	Obsolete as of 4.3.

Table 1-16 PICTURE Comments (cont.)

Element	Meaning
`appId:`*`appTypeCode`*	An integer representing the Applix application matching the inset. For example, if the inset is a graphics object of any type (e.g. GIF, JPEG, Applixware Graphics), the value will be 2, representing "Applixware Graphics". See the *`appTypeCode`* section below for more information. Note this value will be overridden by a similar specification in the inset field that references this object, if the field has such a specification.

Table 1-16 PICTURE Comments (cont.)

Element	Meaning
`clipMode:` *`clipModeCode`*	This controls how the picture is scaled or clipped. Possible values are: 0: This is the default if "clipMode" is not specified. If the picture is in a frame, the picture is scaled to the size of the frame. If the frame has auto width or height or both, the corresponding width / height of the picture is not scaled, i.e., it is displayed at full size. In release 4.3 and before, if the picture is not in a frame, the picture is not scaled, but may be clipped if it is larger than the margins. As of release 4.4, the picture will be sized so its width exactly fits the column or cell it falls within[3], and its height adjusted proportinately, even if the `proportional` flag is absent. However, the 4.3 behavior can be preserved (i.e. the picture only resizes if it is in a frame) by setting the `wpNoInsetScaleToFit` profile to 1. 1: The picture is clipped if it is larger than the frame (if the picture is within a frame) or the margins (if the picture is not in a frame).

[3]Note in release 4.3 and before, a `clipmode` of 0 for an inset not in a frame was ignored (treated as a 1), since only framed insets were allowed to resize. Some documents from 4.3 and before may have this flag improperly set to 0. When the 4.4 release reads a document from release 4.3 or earlier, if an unframed inset has a `clipmode` flag set to 0, it is reset to 1. This will prevent unwanted resizing of unframed insets from older releases.

Table 1-16 PICTURE Comments (cont.)

Element	Meaning
	Note in release 4.2, a graphics inset in a cell was not clipped to the cell margins. As of release 4.3, any picture inset in a cell is clipped to the cell margins. As of release 4.4, the release 4.2 behavior can be recovered (i.e. an inset will not be clipped to the cell margins) by setting the wpNoClip profile to 1. The picture is scaled using the "xScale" and "yScale" attributes of the `PICTURE` bead, if any.
`height:`*`displayHeight (mils)`*	Only used for filters. `height` and `width` define the display size of the picture, taking into account the source specification (or offsets and source size) and the scale factor. `height` and `width` are included only to provide conversion compatibility with foreign formats that require this information. Words will recalculate these values when the file is read and the object first displayed.
`name(string)`	This is a required field. For an embedded object, *name* matches the name in an object or graphic bead containing the data being displayed. For a linked inset, *name* represents the (absolute or relative) filename of the entity. As of release 4.3, name can be a URL.
`proportional`	If omitted, `xScale` and `yScale` can differ

Table 1-16 PICTURE Comments (cont.)

Element	Meaning
`picSource:"`*`elfStringArray`*`"`	As of release 4.3, this format is used to indicate what portion of the inset to display. For a Graphics inset, this might describe a rectangular area within the object; for a Spreadsheet, a cell range, etc. This format represents an ELF array which will be different for each object type. For a Graphics inset, see gr_inset_source@ in graphic_.am. For a Spreadsheet inset, see ss_inset_source@ in spsheet_.am. For a Words inset, see wp_inset_source@ in wp_.am. For a Graphics inset only, if no `picSource` is specified, and any combination of `xOffset, yOffset, srcWidth`, and `srcHeight` are specified, these values are used to manufacture a `picSource.`
`srcHeight:` *`srcHeight(mils)`*	For a graphics inset only. If omitted, the entire height of the source object is used `xOffset, yOffset, srcWidth`, and `srcHeight` define the area of the source material used by the picture. As of 4.3, all insets will write their source specification data in `picSource:"`*`elfStringArray`*`"` format.
`srcWidth:` *`srcWidth(mils)`*	For a graphics inset only. If omitted, the entire width of the source object is used. See `srcHeight` for more information.
`width:` *`displayWidth (mils)`*	Only used for filters; see `height` for more information

Table 1-16 PICTURE Comments (cont.)

Element	Meaning
`xOffset:` *`xoffset(mils)`*	For a graphics inset only. See `srcHeight` for more information
`xScale:` *`xscale(float)`*	If omitted, a default value of 1 is used `xScale` is the horizontal scale factor by which the source material will be scaled
`yOffset:` *`yoffset(mils)`*	For a graphics inset only. See `srcHeight` for more information
`yScale:` *`yscale(float)`*	If omitted, a default value of 1 is used `yScale` is the vertical scale factor by which the source material will be scaled

ROW_START

```
< row_start                /* abbrev: < RS */
   [ justifyLeft|justifyRight|justifyCenter ]
   [ leftIndent:indent(mils) ]
   [ topRowMargin:margin(mils) ]
   [ bottomRowMargin:margin(mils) ]
   [ height:height(mils)|minHeight: height(mils)|
         maxHeight:height(mils) ]
   [ heading ]
>
```

Table 1-17 ROW_START Comments

Element	Meaning
`bottomRowMargin:` *`margin(mils)`*	Amount of whitespace below the row (before the next row or paragraph). Assumed to be zero if omitted.

Table 1-17 ROW_START Comments (cont.)

Element	Meaning
`heading`	If present and this is the first row in the table, this row will be repeated as the first row on subsequent pages which contain rows of the same table This is a Release 4 feature

`height:`*`height(mils)`*`|`
`minHeight:`*`height(mils)`*`|`
`maxHeight:`*`height(mils)`*
	If omitted, `row` is as high as the tallest cell; otherwise, the heights are as follows:
	`height` exactly this tall
	`minHeight` at least this tall
	`maxHeight` at most this tall

`justifyLeft|justifyRight|` `justifyCenter`	The default is left justified
`leftIndent`	If omitted, the row begins at the left column or margin
`topRowMargin:` *`margin(mils)`*	Amount of whitespace above the row (after the preceding row or paragraph). Assumed to be zero if omitted.

SECTION

```
< section
   [ next_column|next_page|even_page|odd_page ]
   [ landscape ]
   pageWidth:width(mils)
   pageHeight: height(mils)
   [ paperSize:paperCode(int) ]
   [ leftMargin:margin(mils) ]
   [ rightMargin:margin(mils) ]
   [ topMargin:margin(mils) ]
   [ bottomMargin:margin(mils) ]
   [ bindingMargin:margin(mils) ]
   [ columns:columns(1-19) ]
   [ gutterWidth:gutter(mils) ]
   [ oddHeader:hdrFtrReference ]
   [ evenHeader:hdrFtrReference ]
   [ firstHeader:hdrFtrReference ]
   [ lastHeader:hdrFtrReference ]
   [ oddFooter:hdrFtrReference ]
   [ evenFooter:hdrFtrReference ]
   [ firstFooter:hdrFtrReference ]
   [ lastFooter:hdrFtrReference ]
   [ sectNumFmt:numstyle ]
   [ sectNumCtl:sectnumctl ]
   [ sectNumVal:sectnum(int) ]
   [ sectPageSep:separator(string) ]
   [ pageNumFmt:numstyle ]
   [ pageNumCtl:pagenumctl ]
   [ pageNumVal:pagenum(int) ]
   [ printSourceTray:sourceTray(int) ]
   [ printDestTray:destTray(int) ]
   [ duplex:duplex(int) ]
>
```

The section and page numbering properties control the appearance of the page number fields that fall into the section (that is, before this section bead but after any preceding section beads). Page numbering consists of a page number, optionally preceded by a section number. Within the section, the page numbers increment, but the section number does not.

Table 1-18 SECTION Comments

Element	**Meaning**
`bindingMargin: margin(mils)`	If the document is in "facing pages" mode, this margin is added to the inner margin of each page. If the document is not in "facing pages" mode, this margin is added to the left margin of each page.
`columns:columns(1-19)`	Single column if omitted

Table 1-18 SECTION Comments (cont.)

Element	Meaning
`duplex:`*`duplex(int)`*	Indication of whether the physical printing should occur on one or both sides of the paper. Note this is not related to the "facing pages" mode. The codes are: 0 = simplex (single-sided) print (the default) 1 = duplex (double-sided), long edge 2 = duplex (double-sided), short edge On a sheet of paper printed in duplex-long, the paper must be rotated about the long (vertical) axis in order to view the back right side up. In duplex-short, the paper must be rotated about the short (horizontal) axis to view the back right side up. This item did not exist in releases prior to release 4.2.
`gutterWidth:`*`gutter(mils)`*	The space left between columns, if the section has more than one column.
`landscape`	If omitted, the pages in the section are printed in normal (portrait) mode. If present, pages are printed in landscape mode. In landscape mode, all text and graphics is rotated 90 degrees with respect to the physical paper.

Table 1-18 SECTION Comments (cont.)

Element	Meaning

`leftMargin:`*`margin(mils)`*
`rightMargin:`*`margin(mils)`*
`topMargin:`*`margin(mils)`*
`bottomMargin:`*`margin(mils)`*
	The left, right, top, and bottom margins; that is, the whitespace between the main flow rectangle and the edge of the paper. Note the header and footer ignore the top and bottom margins.
	
	The margins are place relative to the direction of the text. In a portrait (normal) section of US Letter size, the top and bottom margins run along the short (8.5 inch) sides of the paper. In a landscape section of the same page size, the top and bottom margins run along the long (11 inch) sides of the paper.
	
	If the document is in "facing pages" mode, `leftMargin` represents the inner margin (i.e. the left margin of an odd page and the right margin of an even page) and `rightMargin` represents the outer margin.

`next_column` | `next_page` | `even_page` | `odd_page` | `Continuous`	If omitted, the default is `Continuous`

`oddHeader:`*`hdrFtrReference`*`...`
`lastFooter:`*`hdrFtrReference`*
	The status of the headers and footers on the first, odd, even, and last pages of the section. See *`hdrFtrReference`* for more information.

`pageWidth:`*`width(mils)`*`,`
`pageHeight:`*`height(mils)`*
	Actual paper (page) size for pages that fall into this section. Will override `paperSize` (see below) if a mismatch occurs.
	
	Note that these values are not swapped when the document moves between portrait and landscape mode

Table 1-18 SECTION Comments (cont.)

Element	Meaning
`pageNumCtl:`*`pagenumctl`*	If omitted, increment last page in previous section *`pagenumctl`* is a Boolean controlling the value of the first page in the section. The code is as follows: 0 = increment last page in previous section 1 = use *`pagenum`* value
`pageNumFmt:`*`numstyle`*	If omitted, Arabic numbering is used
`pageNumVal`:*`pagenum(int)`*	If this is used, the page number to a specific value *`pagenum`* is an integer used as the value of the first page number in the section only when *`pagenumctl`* is set to 1

Table 1-18 SECTION Comments (cont.)

Element	Meaning
`paperSize:`*`paperCode(int)`*	Code indicating desired paper size, as displayed in the Page Setup dialog box. Must be kept in synch with `pageWidth` and `pageHeight` (see above) as they control the actual values used for the page size. The code values are as follows: 1 = US Letter 2 = US Tabloid 3 = US Ledger 4 = US Legal 5 = US Statement 6 = US Executive 7 = US Envelope 10 8 = US Envelope 9 9 = US Envelope 6 10 = A3 11 = A4 12 = A5 13 = B4 14 = B5 15 = Envelope C4 16 = Envelope C5 17 = Envelope DIN
`printDestTray:` *`destTray(int)`*	Printer tray number into which printed paper is put during a print. Zero represents the default destination tray. This item did not exist in releases prior to release 4.2.

Table 1-18 SECTION Comments (cont.)

Element	Meaning
`printSourceTray:` *`sourceTray(int)`*	Printer tray number from which paper is drawn during a print. Zero represents the default source tray. This item did not exist in releases prior to release 4.2.
`sectNumCtl`:*`sectnumctl`*	If omitted, section numbering is the same as the previous section *`sectnumctl`* is an integer indicating the method of determining the value of the section numbering. The code numbers, corresponding to the `WP#SECT#NUM#` defines in wp_.am, are as follows: 0 = same value as previous section 1 = increment previous section’s value 2 = use *`sectnum`* value.

Table 1-18 SECTION Comments (cont.)

Element	Meaning
`sectNumFmt`:*`numstyle`*	If omitted, no section numbering *`numstyle`* is an integer representing the section or page numbering format The code numbers, corresponding to the `WP#NUMBERING#` defines in wp_.am, are as follows: -1 = no numbering 0 = Arabic 1 = uppercase letters 2 = lowercase letters 3 = uppercase Roman 4 = lowercase Roman Numbers larger than 1000 indicate document variables that define a macro for a custom series format (1000+N corresponds to document variable `UserSeriesN`) Refer to the Utilities→ Edit Numbered Series → Custom Series dialog box for more information
`sectNumVal`:*`sectnum(int)`*	If used, the section number is forced to this value *`sectnum`* is the section numbering value when *`sectnumctl`* is set to 2
`sectPageSep`:*`separator(string)`*	If omitted, the page separator is a hyphen *`separator`* is the string separating the section and page numbering; it is only displayed if both section and page numbering are displayed (that is, both *`numstyles`* are not -1)

SERIES

```
< series
   name(string)
   n0 n1 n2 n3 n4 n5 n6 n7 n8 n9
   [ leader:leader(string) ]
   [ trailer:trailer(string) ]
   [ leaders0:leader (string) ]
   [ leaders1:leader (string) ]
   [ leaders2:leader (string) ]
   [ leaders3:leader (string) ]
   [ leaders4:leader (string) ]
   [ leaders5:leader (string) ]
   [ leaders6:leader (string) ]
   [ leaders7:leader (string) ]
   [ leaders8:leader (string) ]
   [ leaders9:leader (string) ]
   [ trailers0:trailer (string) ]
   [ trailers1:trailer (string) ]
   [ trailers2:trailer (string) ]
   [ trailers3:trailer (string) ]
   [ trailers4:trailer (string) ]
   [ trailers5:trailer (string) ]
   [ trailers6:trailer (string) ]
   [ trailers7:trailer (string) ]
   [ trailers8:trailer (string) ]
   [ trailers9:trailer (string) ]
   [ multiLevel ]          /* Display single level
                          only if omitted */
   [ sep1:separator (string) ]
   [ sep2:separator (string) ]
   [ sep3:separator (string) ]
   [ sep4:separator (string) ]
   [ sep5:separator (string) ]
   [ sep6:separator (string) ]
   [ sep7:separator (string) ]
   [ sep8:separator (string) ]
   [ sep9:separator (string) ]
   [ from1:level to show from(0-1) ]
   [ from2:level to show from(0-2) ]
   [ from3:level to show from(0-3) ]
   [ from4:level to show from(0-4) ]
   [ from5:level to show from(0-5) ]
   [ from6:level to show from(0-6) ]
   [ from7:level to show from(0-7) ]
   [ from8:level to show from(0-8) ]
```

```
               [ from9:level to show from(0-9) ]
            >
```

Table 1-19 SERIES Comments

Element	**Meaning**
name(string)	Name of the series.
n0...n9 (int)	Integers representing the format of the levels of the series. The level of an instance of the series is, by default, determined by the level of the paragraph the series field falls within. The code numbers, corresponding to the `WP#NUMBERING#` defines in wp_.am, are as follows: -1 = no numbering 0 = Arabic 1 = uppercase letters 2 = lowercase letters 3 = uppercase Roman 4 = lowercase Roman Numbers larger than 1000 indicate document variables which define a macro for a custom series format (1000+N corresponds to document variable `UserSeriesN`); refer to the Utilities → Edit Numbered Series → Custom Series dialog box for more information
`leader:` *leader (string)*	If present, the string to be displayed before the series number(s) or letter(s).
`trailer:` *trailer (string)*	If present, the string to be displayed after the series number(s) or letter(s).

Table 1-19 SERIES Comments (cont.)

Element	Meaning
`leaders0...leaders9:` *`leader (string)`*	Strings representing individual leaders for particular levels of the series. These can be used as an alternative to, or in addition to, the global leader. If a series at a particular level has a level-dependent leader as well as a global leader, the global leader precedes the level-dependent leader. This feature was implemented in release 4.0
`trailers0...trailers9:` *`trailer (string)`*	Strings representing individual trailers for particular levels of the series. These can be used as an alternative to, or in addition to, the global trailer. If a series at a particular level has a level-dependent trailer as well as a global trailer, the global trailer follows the level-dependent trailer. This feature was implemented in release 4.0
`multilevel`	If omitted, only display single level
`from1...from9:` *`level to show from (int)`*	Numbers representing, for a particular level, which higher levels to display. For example, if the formats for levels 0, 1, and 2 are "A", "a", and "i" respectively, the following would be displayed at level 2: Flag — displayed from2:2 — i from2:1 — a.i from2:0 — A.a.i These flags are only serviced when the `multilevel` flag is set.

Table 1-19 SERIES Comments (cont.)

Element	Meaning
`sep1...sep9:` *`separator (string)`*	Strings representing individual separators between particular levels of the series. These separators are only displayed when the `multilevel` flag is set, and the `from` flag for the given level specifies the display of higher levels. The separator for a given level precedes the string for that level, if the next-higher level is also being displayed.

STYLE

```
< style
   name(string)
   parent name(string)
   [ nextStyle name(string) ]
   [ borderAttributes ]
   [ frameAttributes ]
   [ glossary name(string)]  /* Style has no leading
                             glossary if omitted*/
   [ paraAttributes ]
   [ tabDefinitions ]
   [ textAttributes ]
>
```

Table 1-20 STYLE Comments

Element	Meaning
`glossary` *`name(string)`*	If present, name of glossary to be inserted before each instance of a paragraph that uses this style (e.g. a glossary containing a bullet, or a numbered series field).

Table 1-20 STYLE Comments (cont.)

Element	Meaning
name(string)	Name of current style. All styles in a document must have unique names. The first style in a document should be the top-level style, from which all other styles should be directly or indirectly descended.
nextStyle *name(string)*	Name of style to be used when user puts cursor at the end of a paragraph of current style and hits return, thus creating a new, empty paragraph. If omitted, the current style will be used.
parent *name(string)*	This is required except for the top-level style where it is omitted The top-level style does not have itself for a parent

STYLES_BEGIN

```
< start_styles >
```

STYLES_END

```
< end_styles >
```

TEXT

```
< text                       /* abbrev: < T */
   text(string)
   [ textAttributes ]
   [ changedFlag ]
>
```

Table 1-21 TEXT Comments

Element	Meaning
text(string)	This cannot be empty; if it is, the bead is discarded
changedFlag	(Change bar operations only) If present, it always occurs just before the ">"

VARS_BEGIN

```
< start_vars >
```

VARS_END

```
< end_vars >
```

Supported Properties

This section describes the properties of the supported statements. The properties are listed in alphabetical order. Generally, properties are optional and have a default value when they are omitted. The formatting conventions from the "Supported Statements" section are also used here.

appTypeCode

appTypeCode is an integer representing an Applix application type. The same coding system is used by the AX_APP_TYPE_FROM_DOC_TYPE@ macro; some of the common codes are:

1= Words
2= Graphics

3= Spreadsheets
4= Macros
5= Audio
6= Bitmap
10= Data
12 = Equations
14 = HTML Author

See app_ids_.am for more codes.

asciiData

```
<start_data ascii >
   asciiData
<end_data>
```

asciiData is printable ASCII characters. Newlines are ignored. To make a newline in the data, use \n. The maximum line length is 1000 characters. The following are the other escape sequences:

`\t`	tab
`\<`	left angle bracket
`\>`	right angle bracket
`\\`	backslash
`\*`	asterisk

See the section on Wide and Multi-byte Characters earlier in this chapter for a discussion of the treatment of other 8-bit and special characters.

In Release 4.0 and beyond, ASCII data that extends across multiple lines should be in continuation line format, with a backslash at the end of each line (except for the last line), and a space at the beginning of each line (except the first line). For example:

```
<start_data ascii >
Here is the first line.\nHere is the second line; no\
 te the backslash and leading space. Here is the las\
 t line which has no final backslash.
<end_data>
```

Prior to Release 4.0, continuation line format was not used for ASCII data. For example:

```
<start_data ascii >
Here is the first line in 3.11 format.\nHere is th
e second line; note there are no trailing backslas
hes or leading spaces in this version.
<end_data>
```

It is important that the format of the ASCII data be appropriate for the version number in the document header (use continuation lines when the version number is 400 or larger).

binaryData

```
<start_data binary >
   hexData
<end_data>
```

hexData is any binary data where bytes are represented by two hexadecimal digits. New lines are ignored. The maximum line length is 1000 characters.

borderAttributes

See the sections CELL_END, PARA, and STYLE.

```
bottomBorder:thickness(mils)
                          :double(bool):color(string)
                          /* or bB:*/
dropShadow:shadingStyle
horizontal:thickness(mils) :double(bool):color(string)
                          /* or hB:*/
horizontalMargin:margin(mils)
leftBorder:thickness(mils):double(bool):color(string)
                          /* or lB :*/
rightBorder:thickness(mils) :double(bool):color(string)
                          /* or rB :*/
```

```
shading:shadingStyle:foregroundcolor(string):backgroun-
   dcolor(string)
topBorder:thickness(mils) :double(bool):color(string)
                           /* or tB :*/
verticalBorder:thickness(mils)
                           :double(bool):color(string)
                           /* or vB:*/
verticalMargin:margin(mils)
```

shadingStyle is an integer representing the desired shading pattern, as defined by the `WP#SHADING#` defines in wp_.am. `WP#SHADING#NONE`, or integer value 18, is the default.

changedFlag

See the sections PARA and TEXT.

```
Changes:added | deleted | changed
```

This property controls the type of change bar to be displayed adjacent to the line containing the statement. If omitted, no change bar is displayed.

docTypeCode

See OBJECT for information on docTypeCodes.

elfData

See DOC_VARIABLE and elfDataArray.

```
value(int) | value(float) | value(string) | binaryData |
                           elfDataArray
```

elfDataArray

See elfData in the previous section.

```
<
   elfData...
>
```

elfString

See PICTURE and elfStringArray.

```
value(int) | value(float) | value(string) | ~ | elfDataArray
```

Table 1-22 elfString Comments

Element	Meaning
~	The tilde character '~' represents a NULL element in an array.

elfStringArray

See elfString in the previous section.

```
<
   elfString...
>
```

frameAttributes

See the PARA and STYLE sections.

```
bottomFrameMargin: margin(mils)
frameMargin: margin(mils) */
height:height(mils) | minHeight:height(mils) | maxHeight:
                           height(mils)
jumpOver | background       /* Frame type; omit for
                           default (flow around) */
leftFrameMargin: margin(mils)
noFrame                    /* If present, para has no
                           frame */
rightFrameMargin: margin(mils)
topFrameMargin: margin(mils)
width:framewidth(mils)
xpos:position(mils)
xposMarginRelative | xposColumnRelative
```

```
xposTypeCenter | xposTypeRight | xposTypeOutside |
                           xposTypeInside
ypos:position(mils)
yposMarginRelative | yposParaRelative
yposTypeCenter | yposTypeBottom
```

Table 1-23 frameAttributes Comments

Element	Meaning
`frameMargin:` *`margin(mils)`*	If present, it is the value to which all four margins are set
`height:`*`height(mils)`* | `minHeight:`*`height(mils)`*	If omitted, the height will be *autosized*
`jumpOver` | `background`	Frame type; omit for the default, which is flow around
`noFrame`	If present, the paragraph does not have a frame
`width:`*`framewidth(mils)`*	If omitted, the width will be *autosized*
`xposMarginRelative` | `xposColumnRelative`	Relative to (horizontally); omit for default, which is page
`xposTypeCenter` | `xposTypeRight` | `xposTypeOutside` | `xposTypeInside`	Horizontal alignment; omit for the default, which is left align
`yposMarginRelative` | `yposParaRelative`	If omitted, the height is *autosized*
`yposTypeCenter` | `yposTypeBottom`	Vertical alignment; omit for default, which is top align

hdrFtrReference

```
inherit | none | Normal:hdrftrName(string) |
   UseGlossary: glossaryName(string)
```

Table 1-24 hdrFtrReference Comments

Element	Meaning
glossaryName	Match the name in a GLOSSARY_BEGIN statement
hdrftrName	Match the name in a HDRFTR_BEGIN statement. The name should be of the format "_AX_HF_#", where # is a number; For example: "_AX_HF_1"
inherit	Inherit from the previous section
none	Do not use a header or footer of this type in this section

paraAttributes

Note that if properties are missing, then the values are inherited.

firstIndent:*indent(mils)*
hyphMinFrag:*frag(# of chars)*
hyphZone:*zone(mils)*
indentToLevel|no-indentToLevel
justifyLeft|justifyRight|justifyCenter|justifyFull
leftIndent:*indent(mils)*
level:*level(0-10)*
lineSpacing:*spacing (mils)*
lineSpacingMode:*mode*
lineSpacingCountHeight:*value (number, mils)*
lineSpacingSep:*value (mils)*
no-keepWith|keepWith
postParaSpacing:*spacing(mils)*
preParaSpacing:*spacing(mils)*
rightIndent:*indent(mils)*
spellcheck|no-spellcheck

Table 1-25 paraAttributes Comments

Element	Meaning
`indentToLevel`	If indenting, add an extra indent of `level * indent-per-level-amount`; See DOCUMENT_BEGIN for more information
`level`	Paragraph indent level. Values range from 0 to 9.
`lineSpacing`	*Spacing* is a number specifying leading, in mils. Retained for compatibility with versions earlier than 4.4.2. Additional space allocated to lines for single-line spacing, to emulate multiple-line spacing.
`lineSpacingMode`	Retained for backward compatibility with Version 4.4.1 and earlier. *Mode* is a number specifying line spacing mode. 0: Auto (line height set by contents) 1: Exact 2: At-least.
`lineSpacingMode2`	Retained for backward compatibility with Version 4.4.1 and earlier. *Mode* is a number specifying line spacing mode. 4: At-least 5: Exact 6: Multiple-line

Table 1-25 paraAttributes Comments (cont.)

Element	Meaning
`lineSpacing-CountHeight`	"Multiple-line" spacing: *Value* is number of lines, in 1000*ths* of lines (1000 = single). "Exact" mode: *Value* is line height in mils. "At Least" mode: *Value* is minimum line height in mils.
`lineSpacingSep`	"Multiple-line" mode only. *Value* is an amount in mils added to space between bottom and top of consecutive lines.

tabDefinitions

See PARA, STYLE

```
localTabs
[tabStop]...
```

Table 1-26 tabDefinitions Comments

Element	Meaning
`localTabs`	If omitted, all tabs are inherited; otherwise, no tabs are inherited
`tabStop`	Set of tabs if tabs are not inherited

tabStop

See tabDefinitions

A tabstop definition consists of one of the following:

```
centerTab:position(mils)) [:leaders(string) ] /* abbrev
                              cT */
                                      /* abbrev dT */
```

```
decimalTab:position:alignChar(string) [:leaders(string)
                                     ]
leftTab:position(mils) [:leaders(string) ]
                                              /* abbrev: lT */
rightTab:position(mils) [:leaders(string) ]
                                               /* abbrev rT */
```

Table 1-27 tabStop Comments

Element	Meaning
alignChar	A string containing a single character; the matching character in the text following a tab will align at this tab stop An *alignChar* is required in a decimal tab
leaders	The string that is replicated along the length of the tab up to the tabbed text; if omitted, the tab has no leader

textAttributes

See PARA, STYLE

Note that if properties are missing, values are inherited.

```
bold|no-bold
color:name(string)
face: name(string)
hyphenate|no-hyphenate
italic|no-italic
no-underline|underline|double-underline|word-
   underline |double-word-underline|underline-no-tabs
   |double-underline-no-tabs
position:offset (points)
size: size(points)
strikethru|no-strikethru
```

Table 1-28 textAttributes Comments

Element	Meaning
offset	A positive number is a superscript; a negative number is a subscript

Equation File Format

Equations uses two different formats. One format is used when the equation is contained as an embedded object; the second is used when the object is localized.

The following is the sequence of statements used when Equation information is stored as an embedded object.

```
document = DOCUMENT_BEGIN
         [ EQUATION-SETTINGS ]
         equation
         DOCUMENT_END

equation = EQUATION_BEGIN
         [ TEMPLATE ]
         EQUATION_END

subequations = [ equation ] ...
```

The following format is used when an Equation object is localized:

```
equationBead =
         EQUATION_BEAD_BEGIN
         [ EQUATION-SETTINGS ]
         equation
         EQUATION_BEAD_END
```

Equation Statements

DOCUMENT_BEGIN

```
*BEGIN[4] EQUATIONS VERSION=vnum ENCODING=encoding
<APPLIX EQUATIONS>
```

DOCUMENT_END

```
<end_document>
*END EQUATIONS
```

EQUATION_BEAD_END

```
>
```

EQUATION_BEAD_START

```
<equation
```

EQUATION_BEGIN

```
<eqn_begin
      [ pileAlign ]
      [ matrixRowAlign ]
      [ matrixColAlign ]
 >
```

EQUATION_END

```
<eqn_end>
```

[4] Was *START in releases prior to Release 4.

EQUATION_SETTINGS

```
fullSize: fontSize(points)
subSize: fontSize(points)
subSubSize: fontSize(points)
symbolSize: fontSize(points)
subSymbolSize: fontSize(points)
lineSpace: percent|amount(points)
matrixRowSpace: percent|amount(points)
matrixColSpace: percent|amount(points)
subscriptDepth: percent|amount(points)
superscriptHeight: percent|amount(points)
limitHeight: percent|amount(points)
```

Only amounts that differ from their default values are stored in a file. A negative spacing attribute value represents a relative value in percent of the *`fontHeight`* of the font with a `size=fullSize`.

The following are the default value for each of these equation settings:

Table 1-29 Equation Setting Defaults

Element	Default Value
`fullSize`	12 points
`subSize`	8 points
`subSubSize`	6 points
`symbolSize`	24 points
`subSymbolSize`	18 points
`lineSpace`	-150
`matrixRowSpace`	-150
`matrixColSpace`	-100
`subscriptDepth`	-40

Table 1-29 Equation Setting Defaults (cont.)

Element	Default Value
`superscriptHeight`	-20
`limitHeight`	-20

TEMPLATE

```
<tmpl
      template-type
      [ xnudge: amount(mils) ]
      [ ynudge: amount(mils) ]
>
subequations...
```

The number of subequations depends on the type of template.

Equation Properties

matrixColAlign

`leftAlign | centerAlign | rightAlign | equalsAlign | markerAlign | decimalAlign`

Only a `matrixColAlign` attribute that differs from the default is saved in a file. The default value is `leftAlign`.

matrixRowAlign

`topAlign | centerAlign | bottomAlgin | baselineAlign`

Only a `matrixRowAlign` attribute that differs from the default is saved in a file. The default value is `topAlign`.

pileAlign

```
leftAlign|centerAlign|rightAlign|equalsAlign|
markerAlign|decimalAlign
```

Only a `pileAlign` attribute that differs from the default is saved in a file. The default value is `leftAlign`.

symbol

```
symbol { string|char:asciiCode(num)
               | space:amount(mils)|alignMarker }  |
relSymbol { string|char:asciiCode(num) }
|
greekSymbol {string|char:asciiCode(num) }
```

A symbol template is a simple template. That is, it is a template whose contents do not contain any subequations. All adjacent symbol templates with similar properties are normally combined to form string.

Automatic mathematical function recognition is performed upon symbols.

To qualify for a function name, all the letters in the function should exist as symbol templates with no attribute exceptions. In addition, they must all be adjacent.

Non-printable characters are stored with a `char:asciiCode` description.

`AlignMarker` and `spaces` (which are also simple templates) are treated as symbol templates.

`greekSymbol` and `relSymbol` are derived from symbol templates; however, their default font is Symbol.

symbolAttributes

```
no-bold|bold
no-italic|italic
face: fontName(name)
size: fontSize(pts)
```

Only attributes which differ from their default values are stored in a file. The default values for these properties are:

Table 1-30 symbolAttribute Defaults

Element	Default Value
`no-bold|bold`	`no-bold`
`no-italic|italic`	`italic`
`face:` *`fontName(name)`*	`Times`
`size:` *`fontSize(pts)`*	`fullSize`

templateType

```
symbol [ symbol-attributes ] |
     paren|ltparen|rtparen |
     brace|ltbrace|rtbrace |
     bracket|ltbracket|rtbracket |
     vert|ltvert|rtvert |
     dblvert|ltdblvert|rtdblvert |
     floor|ltfloor|rtfloor |
     ceil|ltceil|rtceil|
     ltltbracket|rtrtbracket|oppBracket |
     parenBracket|bracketParen |
     underbar|dblunderbar |
overbar|dbloverbar |
over|divide|slash|by|vertslash |
sqrt|nroot |
sub|sup|subsup|ltsub|ltsup|ltsubsup |
script|scriptup|scriptdn| scriptupdn |
sum|sumup|sumdn|sumupdn|sumsup|sumsub|
               sumsubsup |
prod|produp|proddn|produpdn|prodsup|prodsub|
               prodsubsup |
cup|cupup|cupdn|cupupdn|cupsup|cupsub|
cupsubsup |
cap|capup|capdn|capupdn|capsup|capsub|
capsubsup |
int|intup|intdn|intupdn|intsup|intsub|
intsubsup |
tint|tintup|tintdn|tintupdn|tintsup|tintsub|
tintsubsup |
thint|thintup|thintdn|thintupdn|thintsup|
thintsub |
               thintsubsup |
oint|ointup|ointdn|ointupdn|ointsup|ointsub|
ointsubsup |
ltarrowup|rtarrowup|ltrtarrowup |
ltarrowdn|rtarrowdn|ltrtarrowdn |
matrix matrixRows:number MatrixCols: number
      [matrixRowAlign: topAlign|centerAlign|
bottomAlign|baselineAlign ]
      [matrixColAlign: leftAlign|centerAlign|
rightAlign|equalsAlign|markerAlign|decimalAlign
]
      [equalRowHeights] [equalColWidths] |
eqn_break
```

More Information on Templates

The following table describes various template types and the number of subequations they contain.

Table 1-31 Template Descriptions

Types	#	Description
symbol	0	symbol, relSymbol, greekSymbol, space, alignMarker are members of this template, which are simple templates
eqn_break	1	Piles of equations are built with eqn_break template as the separator between equations
fences	1	paren, ltparen, rtparen, brace, ltbrace, rtbrace, bracket, ltbracket, rtbracket, vert, ltvert, rtvert, dblvert, ltdblvert, rtdblvert, underbar, dblunderbar, overbar, dbloverbar, floor, ltfloor, rtfloor, ceil, ltceil, rtceil, ltltbracket, rtrtbracket, and oppBracket are members of the fences template The size of the subequation is fullsize
fraction	2	over, divide, slash, by and vertslash are members of the fraction template The order of equations stored in files is: [*numerator*][*denominator*] The size of both subequations is fullSize

Table 1-31 Template Descriptions (cont.)

Types	#	Description
sqrt	*	sqrt(1), nroot(2) are members of the sqrt template The number in parentheses indicates the number of subequations these templates contain. The order of equations stored in files is: [*body*][*nroot*] Size of '*body*' equation is fullSize The size of '*nroot*' equation is subSize
script	*	script(1), scriptup(2), scriptdn(2) and scriptupdn(3) are members of the script templates. The number in parenthesis indicates the number of subequations these templates contain The order of equations stored in files is : [*body*][*up limit*][*down limit*] The size of '*body*' equation is fullSize The size of both the *limit* equations is subSize

Table 1-31 Template Descriptions (cont.)

Types	#	Description

summation	*	sum(1), sumup(2), sumdn(2), sumupdn(3), sumsup(2), sumsub(2), sumsubsup(3)
		prod(1), produp(2), proddn(2), produpdn(3), prodsup(2), prodsub(2), prodsubsup(3)
		cup(1), cupup(2), cupdn(2), cupupdn(3), cupsup(2), cupsub(2), cupsubsup(3)
		cap(1), capup(2), capdn(2), capupdn(3), capsup(2), capsub(2), capsubsup(3)
		int(1), intup(2), intdn(2), intupdn(3), intsup(2), intsub(2), intsubsup(3)
		tint(1), tintup(2), tintdn(2), tintupdn(3), tintsup(2), tintsub(2), tintsubsup(3)
		thint(1), thintup(2), thintdn(2), thintupdn(3), thintsup(2), thintsub(2), thintsubsup(3)
		oint(1), ointup(2), ointdn(2), ointupdn(3), ointsup(2), ointsub(2), ointsubsup(3)
		are the members of the summation template
		
		The order of equations stored is:
		[*body*][*up/sup limit*][*down/sub limit*]
		
		The size of '*body*' equation is fullSize and size of both the *limit* equations is subSize

Table 1-31 Template Descriptions (cont.)

Types	#	Description
subscript/ superscript	*	sub(1), sup(1), subsup(2), ltsub(1), ltsup(1) , and ltsubsup(2) are members of the subscript / superscript template The order of equations stored is: [*superscript*][*subscript*] The size of both the *script* equations is subSize
label-arrows	1	ltarrowup, rtarrowup, ltrtarrowup, ltarrowdn, rtarrowdn, ltrtarrowdn are members of the label-arrows template Size of the subequation is subSize
matrix	*	Total number of subequations inside a matrix of *m* rows and *n* cols is *m* x *n* The order of equations stored in files is row major ; that is, elements in the top row are stored first from left-column to right-column and then the second row and then the third row and so on All subequations are created with fullSize

*   An asterisk in the second column indicates the number of subequations is variable and is described in ’description’ column.

2 Technical Terms

This chapter describes the technical terms for Words document elements, giving tips for usage and sample macros.

Technical Terms

The terms are described in the following sections in alphabetical order. The terms are:

Bead
Bead Number
Cell
Column, Table
Current Selection
Cursor
Embedded Object
Evaluation
Field
Field Method
Field Value
Flow
Footnote
Frame
Glossary Definition and Glossary Field
Header and Footer
Inherited Attributes
Inset
Local Attributes
Linked Object
Localize
Location
Marker Bead
Main Flow
Multi-Cell Selection
Object
Object Bead
Offset
Paragraph and Paragraph Bead
Paragraph Marker
Range
Row
Section
Selection
Series Definition & Series Field
Simple Selection
Style
Table
Tag
Text Bead
Undo and Redo

Bead

The *bead* is the basic unit of a information within a Words document and represents the different kinds of information and attributes . For example, a paragraph that contains simple text is stored as a sequence of two beads:

1. A text bead containing the text of the paragraph
2. A paragraph bead naming the paragraph's style

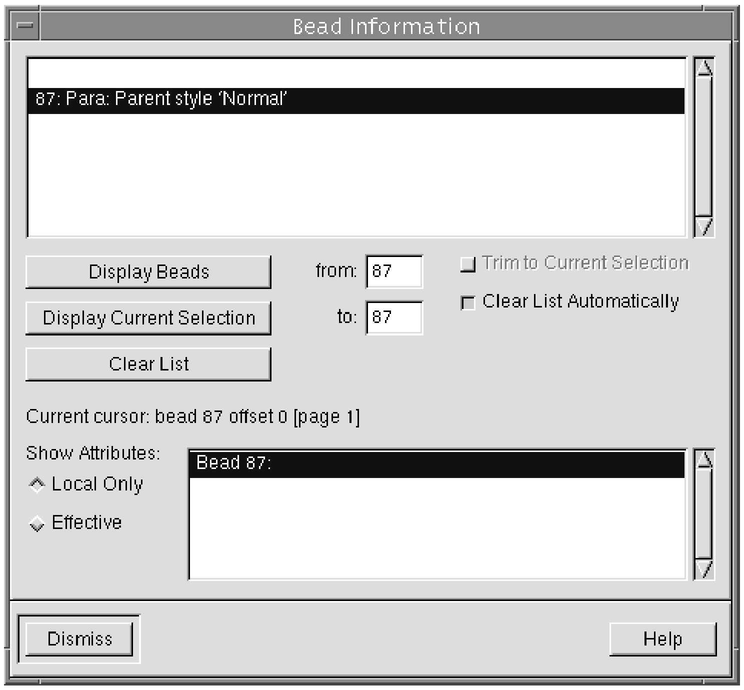
A Words document can be conceived as being a linear sequence of beads. All the material that makes up a document is contained within beads.

The .aw file generated by Words represents a sequence of tokens that is converted into an equivalent bead sequence when the file is opened.

The best way to understand beads is as follows:

- Invoke the Bead Information dialog box.

  It is listed on the Help pulldown within the menubar editor as follows: Help → Document Information → Bead Information; however, it is not displayed in the default menubar.



87: Para: Parent style 'Normal'

from: 87

to: 87

Trim to Current Selection

Clear List Automatically

Current cursor: bead 87 offset 0 [page 1]

Show Attributes:
Local Only
Effective

Bead 87:

- Select various parts of the document
- Examine the bead information displayed in the dialog box

This dialog box can also be useful when debugging some ELF macros. The ELF programmer must be very careful when dealing directly with beads as deleting or inserting the wrong type of bead in the wrong place can cause Words to exit precipitously.

Fortunately, most of the macros that you'll write will use the higher-level Words macros. These macros allow you to bypass bead manipulation macros. They will also detect and prevent most of the problems that can occur when you directly manipulate beads.

Related Macros:

WP_BEAD_INFO_DLG@
WP_COPY_BEADS@
WP_DELETE_BEAD@
WP_DELETE_BEADS@
WP_DELETE_BEADS_IN_RANGE@
WP_GET_BEAD@
WP_GET_BEADS@
WP_GET_BEADS_IN_RANGE@
WP_INSERT_BEAD@
WP_INSERT_BEADS@
WP_MOVE_BEADS@
WP_REPLACE_BEAD@
WP_SWAP_BEADS@
WP_SPLIT_BEAD@

Bead Number

The sequence of beads that makes up a Words document is numbered starting at one and continuing monotonically until the last bead in the document. There are no gaps in this sequence.

Many ELF macros access beads by their numbers.

As a document is edited, beads are added or deleted; the number associated with a bead will change as numbering gaps are eliminated. This means that your macros must use tags or markers to keep track of a bead if an operation can shift bead numbers.

Related definitions:

Beads; *Marker Beads*; *Tags*

Cell

A cell is a rectangular area that can contain document information. A series of adjacent cells make up a row; a series of adjacent rows makes up a table.

The beads that make up a cell are:

1. Any document material (see below)
2. A Paragraph bead
3. A Cell End bead

A cell has the following properties:

- Width
- Left, right, top, and bottom margins that separate the contents of the cell from its bounds
- Border and shading information
- Form properties (beginning at Release 4)
- Whether or not this is the last cell in the row

A cell can contain any document material except:

- Other cells
- Framed paragraphs
- Section, column, and page breaks

Macros that deal with column and cell numbers use a zero-based system; that is, the first cell in a row is cell 0, the second cell is cell 1, and so on.

Related User Interface:

Table ➝ Cell Attributes
Table ➝ Insert Cells, Table ➝ Delete Cells
Table ➝ Make Row Fit
Table ➝ Merge ➝ Cells Into First [Wide] Cell
Table ➝ Select ➝ Cell
Table ➝ Split ➝ Cell

Related ELF formats:

format wp_cell_attrs@
    see wp_.am in *install_dir*/axdata/elf

Related Macros:

WP_CELL_IS_SELECTED@
WP_COPY_CELL@
WP_DELETE_CELL@
WP_DELETE_CELLS@
WP_FIND_BEAD_OF_CELL@
WP_FIND_NTH_CELL@
WP_GET_CELL@
WP_GET_RANGE_OF_CELL@
WP_GET_RANGE_OF_CELLS@
WP_GET_SELECTED_CELL_ATTRS@
WP_GOTO_FIRST_CELL@
WP_ILLEGAL_FOR_TABLE@
WP_INSERT_CELLS@
WP_IS_BEAD_IN_CELL@
WP_NUM_CELLS_IN_ROW@
WP_SELECT_NEXT_CELL@
WP_SELECT_PREV_CELL@
WP_SET_CELL_ATTRS@
WP_SET_CELL_MARGINS@
WP_SET_CELL_VERT_ALIGN@
WP_SET_CELL_WIDTH@
WP_SET_SELECTED_CELL_ATTRS@
WP_SET_SELECTED_CELL_MARGINS@

WP_SET_SELECTED_CELL_V_ALIGN@
WP_SET_SELECTED_CELL_WIDTH@
WP_SHIFT_CELLS@
WP_SWAP_CELLS@

Related Definitions:

Bead; Column, Table; Row; Table

Column, Table

A set of cells that occupy the same logical position in adjacent rows (that is, within a table) are referred to as a column. For example, the third cell in every row of a table represents the third column. It could occur that some rows in this table do not have three cells; a column does not necessarily have a cell in each row.

Macros that deal with column and cell numbers use a zero-based system; that is, the first cell in a row is cell 0, the second cell is cell 1, and so on.
Related User Interface:

Table → Insert Cells
Table → Select Column

Related Macros:

WP_COL_NUM@
WP_GET_RANGE_OF_COLUMN@
WP_GET_RANGE_OF_NTH_COLUMN@

Related Definitions:

Cell; Row; Table

Current Selection

At all times, either a cursor or a single selection is displayed in the document. Either of these can be referred to as the *current selection*. Most operations deal with the current selection.

Related Macros:

WP_GET_CURRENT_RANGE@
WP_HAVE_SELECTION@
WP_LOCATION_OF_CURSOR@
WP_SELECT_RANGE@
WP_SET_CURSOR@

Related Definitions:

Cursor; Selection

Cursor

The current selection can be a cursor (which by default appears as an *I-bar*). The cursor can only be placed in a location where it is legal to type. Any operation that inserts material (typing, pasting, inserting a field, and so on) will insert this material at the cursor.

Related Macros:

WP_FORCE_CURSOR@
WP_LOCATION_OF_CURSOR@
WP_RANGE_IS_CURSOR@
WP_SET_CURSOR@

Related Definitions:

Current Selection; Location; Range

Embedded Object

An embedded object is an object representing (usually) non-Words material placed within the document. This material is completely stored within the document; in contrast, a linked object has its information stored in an external file.

An embedded object is defined by an embedded object field. The field value contains the beads that represent the foreign material. For a picture object, this will be a single Bitmap bead; for an object in Words material format, this will be the set of beads (e.g. Text, Paragraph, Row Start, Cell End) required to represent the object.

The field method contains a unique string (for example, "object2"). These strings correspond to object beads that are stored after the main flow. The object bead contains the embedded material in its native format. Double-clicking upon the object will bring up the foreign application (appropriate for this type of material) to edit it.

Note that it is the contents of the object bead, not the field value, that is given to the foreign application; any edits to the field value are lost in this process. (This occurs because the Words beads representing the material in the field value may not be rich enough to preserve all the foreign material's information.)

For example, a Spreadsheet is represented by a table; this table has no knowledge of cell formulas. By retaining the original Spreadsheet material, you can edit it in Spreadsheets again. It is helpful to think of the field value of an object as just a "view" or "snapshot" of the foreign material.

As of release 4.3, a Words document can contain an embedded Words object. This is most useful in picture format.

Related User Interface:

Format ➝ Embedded Objects
Format ➝ Graphics

Insert → Object → [all]

Related Macros:

WP_INSERT_AUDIO_INSET@
WP_INSERT_EQUATION_INSET@
WP_INSERT_GR_INSET@
WP_INSERT_GR_INSET_IN_FRAME@
WP_INSERT_QUERY_INSET@
WP_INSERT_SS_INSET@
WP_OPEN_INSET@

Related Definitions:

Bead; Field; Field Value; Linked Object; Main Flow; Object; Picture; Table

Evaluation

When a field is evaluated, Words uses the field method to generate a new field value. This new value replaces the old field value.

Related User Interface:

Utilities → Field Editing

Related Macros:

WP_EVAL_FIELDS@
WP_EVAL_ALL_FIELDS@

Related Definitions:

Field; Field Method; Field Value

Field

A field is a special type of document material that consists of a *field method* and a *field value*, only one of which is visible at a time. Evaluating the field method generates the field value.

The beads that make up a field are as follows:

1. The Field Start bead
2. Beads for the field method (usually a single text bead)
3. The Field Split bead
4. Beads for the field value
5. The Field End bead

If a selection includes the start or the end of a field, the selection is expanded to include the entire field.

Beginning at Release 3.2 of Applixware, the cursor can either be just before the Field Start bead or just before the first bead of the field value when the user is in normal viewing mode. While the bead position is different, the viewing position is identical. Applix Words lets you know which position you are in by displaying a message in the status line at the bottom of the screen. This message will indicate if the the cursor is "Before" or "In" the field. Similarly, a cursor that is before the Field End bead is indicated as "In" the field.

Related Macros:

WP_EVAL_ALL_FIELDS@
WP_EVAL_FIELDS@
WP_FIND_FIELD_OF_BEAD@
WP_FIND_MATCHING_FIELD_END@
WP_FIND_MATCHING_FIELD_SPLIT@
WP_FIND_MATCHING_FIELD_START@

WP_GET_FIELD_METHOD@
WP_GET_FIELD_NESTING_LEVEL@
WP_INSERT_FIELD@
WP_SELECT_FIELD_VALUE_OF_FIELD@
WP_UNFIELD@

Related Definitions:

Bead; Evaluation; Field Method; Field Value; Selection

Field Method

A field method is a set of instructions in the form of a text string that, when evaluated, generate a field value. The field method always starts with the field type (for example, date or page_number). It is usually followed by a number of flags or parameters.

In normal viewing mode, field methods are not visible; turn on View → Field Methods to see them. While in this mode, the corresponding field values are no longer visible. Field methods are displayed in curly braces (for example, {date 0}). When they are visible, field methods can be edited just like normal text; a field with an edited method is re-evaluated when you return to normal viewing mode.

Related User Interface:

View → Field Methods

Related Macros:

WP_GET_FIELD_METHOD@

Related Definition:

Field Value

Field Value

A field value is a set of document material generated by the evaluation of the corresponding field method. In normal viewing mode, field values look like any other document material. Most fields are *protected*, which means you cannot put the cursor into the field value nor can the selection include only part of the field value. The field can be unprotected using the Utilities ➝ Field Editing ➝ Protect/Unprotect Fields command.

Before Release 4 of Words, the cursor could not be placed at the very beginning or end of the field value. (This is just after the Field Start bead or just before the Field End bead.)

When a field is evaluated, the current field value is discarded and replaced by a new field value. Only local text attributes that were consistent across the entire field value are preserved.

Related User Interface:

Utilities ➝ Field Editing ➝ Protect/Unprotect Fields
Utilities ➝ Field Editing ➝ Select Field Value

Related Macros:

WP_SELECT_FIELD_VALUE_OF_FIELD@

Related Definitions:

Cursor; Evaluation; Field Method; Selection

Flow

A flow is a self-contained group of document material. A document always contains a main flow. It also includes separate flows for each header, footer, or footnote. In general, any low-level operation that begins in one flow cannot continue into another.

Every flow ends with a paragraph bead that cannot be deleted.

The current selection cannot extend across multiple flows.

Certain operations are only available within the main flow; for example, if your cursor is in a footnote body, you cannot insert a footnote. Some operations that start in the main flow (for example, Find & Replace and Spellcheck) continue into header, footer, and footnote flows.

Related Macros:

WP_GET_BEAD_FLOW_BOUNDS@
WP_GET_DOC_BOUNDS@
WP_GET_FLOW_BOUNDS@
WP_IN_FOOTNOTE@
WP_IN_HDRFTR@
WP_IN_MAIN_FLOW@
WP_SELECT_TO_FLOW_END@

Related Definitions:

Current Selection; Main Flow; Paragraph Bead

Footnote

A footnote has two parts:

- The *footnote reference*, which is a field that controls the footnote indicator (usually a superscripted number) in the document's main flow. This field cannot be placed in any other flow. See Chapter 3 for more information on this field.
- The *footnote body*, which defines a separate flow. This flow consists of a footnote body field followed by the document material

representing the contents of the footnote. It concludes with a paragraph bead. The footnote body field refers back to the matching footnote reference field and duplicates the footnote indicator. See Chapter 3 for more information on this field.

A footnote can either be defined as "bottom-of-page" or "end-of-document". A separate series is used for each type’s numbering. For bottom-of-page footnotes, the footnote bodies are displayed in a rectangular area below the main flow’s rectangle and above the footer rectangle (if it exists). A horizontal bar is also displayed between the main flow rectangle and the footnote rectangle.

Related Macros:

WP_IN_FOOTNOTE@
WP_INSERT_FOOTNOTE@
WP_VALIDATE_FOOTNOTE_LOCATION@

Related Definitions:

Field; Flow; Main Flow; Paragraph Bead

Form and Form Mode (Beginning at Version 4)

A form is a table that whose purpose is to have a user fill-in values in cells. Some of the cells have special properties that are only activated in Form mode.

Cell properties that Forms mode makes use of are as follows:

Cell Id	A name that identifies a cell
Next Cell Id	The Cell Id of the cell that logically follows this cell in the form.
Locked	A flag that indicates if the form user can navigate into this cell

Double-click macro	The name of the ELF macro invoked when a user double-clicks in this cell while in Form mode
Entry macro, Exit macro	The name of the ELF macro invoked when the cursor enters or exits this cell in Form mode.

A form is created as a table. The form's designer will use the commands with the Table Form → Editor dialog box to describe the form's behavior when the document is placed in Form mode. In addition, ELF macros can be written if you need to control the form's behavior or validate the information typed by users.

The Table → Form Mode command turns Form mode on and off. If a document is saved in Form mode, it will be in Form mode when it is reopened.

When a document is in Form mode:

- Locked calls cannot be selected or deleted
- Any macros set for the cell are run
- The Goto Next / Previous Cell macros use the Next Cell Id property of the current cell
- The user cannot directly drag the cell borders.

Related Macros:

WP_GET_FORM_MODE@
WP_SET_FORM_MODE@
WP_TOGGLE_FORM_MODE@

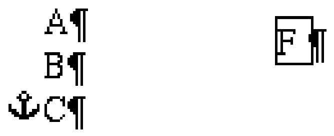
Related Definitions:

Cell

Frame

A frame is usually thought of as being a rectangular container which can hold document material. However, it is really a set of paragraph attributes that imposes an alternate display position and margins for the paragraph. Adjacent paragraphs with equivalent frame properties appear to be merged into a single frame.

Although framed material may appear anywhere on the page, its *logical* location (in terms of beads) is indicated by the frame anchor symbol (which is visible if View ➝ Format Characters is on). For example:



In this illustration, paragraphs A, B, and C are "normal" and paragraph F is framed. The frame anchor is visible to the left of paragraph C. This indicates the contents of paragraph F actually falls between paragraphs B and C regardless of where paragraph F is actually displayed. Thus if you selected paragraphs B and C, paragraph F is also selected; a search operation would examine paragraph B, then F, then C.

If you unframed paragraph F, its contents would appear after paragraph B. By convention, the framed paragraph is anchored to the paragraph it follows; that is, F is anchored to B.

Framed material always appears on the same page as the paragraph to which it is anchored. Using the Format ➝ Frame dialog box, you can control where the contents of the frame are displayed:

- Relative to the paragraph to which it is anchored
- Relative to the page upon which it falls

Related User Interface:

Insert → Frame
Format → Frame

Related ELF formats:

format wp_frame@
    see wp_.am in axdata/elf

Related Macros:

WP_APPLY_FRAME@
WP_FORMAT_FRAME@
WP_GET_ACTUAL_FRAME_SIZE@
WP_GET_FRAME_ATTRS@
WP_NEW_FRAME@
WP_SELECT_RANGE_FRAME@

Related Definitions:

Beads

Glossary Definition & Glossary Field

A glossary definition is a set of document material that is assigned a unique name. This information is stored before the main flow.

A glossary field displays the contents of the glossary's definition as its field value. See "Glossary Field" in Chapter 3 for more information.

A style definition can use a glossary. If a style using a glossary is applied to a paragraph, the glossary field is placed at the beginning of the paragraph.

Related Definitions:

Field; Field Value; Style

Header & Footer

A header or footer can contain almost any type of document material. This material is not considered to be in the main flow of the document. Instead, the header or footer defines a separate flow for the material. This flow is displayed in a rectangular area at the top or bottom of the page (use View → Header/Footer Boundaries to see these rectangles).

A header or footer definition is attached to a section. This means that a header or footer is only defined within its section. (A section's definition may allow it to use the same header or footer information as a previous section). Material in a header or footer is repeated across multiple pages within its section according to its type:

- *Default* (for example, Quick Header / Footer): all pages
- *First*: first page of section only
- *Last*: last page of section only
- *Odd/Even*: odd or even pages of section only

Only one set of beads represents the material in a header or footer, even if the material appears on more than one page. For example, if a default header appears on all pages in the document and you put your cursor into the header on page 3, you are not really "on" page 3; you are in the header flow.

A Find & Replace or Spellcheck operation begun at this point will operate only within the header flow; it will not examine page 3 or any other page. Edits to the header do not just alter it on page 3; the edits are visible on all pages upon which this header is displayed.

The page number and cross reference fields, when placed in a header or footer, are reevaluated on each page displaying a header or footer. This means that the header and footer's information can differ from page to page. All other material in the header and footer appears the same on every page.

Page number and cross reference field reevaluation occurs at display time; the field value is not altered. This means that the field value of the page number and cross reference fields are irrelevant. (Page number field values are always irrelevant; cross-reference field values are irrelevant when they are a header or footer).

Related ELF formats:

```
format wp_hdrftr@
    see wp_.am in axdata/elf
```

Related Macros:

```
WP_GET_CURRENT_HDRFTR_INFO@
WP_GET_HDRFTR_FROM_NAME@
WP_GET_HDRFTR_INFO@
WP_GET_HDRFTR_MARGINS@
WP_GOTO_EVEN_FOOTER@
WP_GOTO_EVEN_HEADER@
WP_GOTO_FIRST_FOOTER@
WP_GOTO_FIRST_HEADER@
WP_GOTO_LAST_FOOTER@
WP_GOTO_LAST_HEADER@
WP_GOTO_ODD_FOOTER@
WP_GOTO_ODD_HEADER@
WP_IN_HDRFTR@
WP_SELECT_CURRENT_HDRFTR@
WP_SET_CURRENT_HDRFTR_INFO@
WP_SET_HDRFTR_INFO@
WP_SET_HDRFTR_MARGINS@
```

Related Definitions:

Bead; Field Value; Flow; Main Flow

Inherited Attributes

In document material, an attribute that is marked as *inherited* derives its value from the setting in the material's paragraph style. Changes to the style's definition for this attribute alter the effective (visible) appearance of this attribute. In contrast, local attributes are unaffected by changes to the style.

Inherited attributes can also be set in styles.

Related Definitions:

Local Attributes; Style

Inset

An *inset* is another name for an *object*.

Related Definitions:

Object

Level

Level is an attribute of a paragraph. It controls the display of series fields within the paragraph. It can also control the indentation of the paragraph. In the user interface, levels are numbered from one (the "highest" to 10 (the "lowest"); internally, levels are zero-based (thus zero through 9). Zero-based numbers are used in this definition.

The Demote macro lowers the level (increases the level number) of a paragraph. The Promote macro raises the level (decreases the level number) of a paragraph.

Level controls the display of series fields. The series definition indicates a format for each level; for example, Arabic numbering for level 1, capital letters for level 2, and small letters for level 3. If the series definition indicates that multiple levels are not being displayed, only the format for that level is visible. For example, assume that the series defined above is used as a glossary for the following paragraphs:

4. This is a level 1 paragraph

    A. This is a level 2 paragraph

        a. This is a level 3 paragraph

The series field can, however, override the paragraph level. See Chapter 3 for more information.

The level can also alter the indenting of a paragraph. Every paragraph has a first and left indent. If levels are affecting the indent, then the following value is added to the first and left indent:

*level_number * level_indent*

The *level_indent* is set using the Format → Document Settings dialog box. The level indent is only applied to the individual paragraph contents if the Level Affects Indent paragraph attribute is set TRUE for the paragraph, which is the default.

If paragraph is indented due to its level, this extra indentation is not shown in the Paragraph Settings dialog box indent entry fields. Also, if the user desires that certain paragraphs be given a different level (for example, to affect any embedded series fields) without affecting their indents, the "Level Affects Indent" toggle can be turned off.

Related Definitions:

Paragraph; Series

Linked Object

A linked object is an object whose contents are contained in a separate file (Words or non-Words). The linked object provides a view of the file's current contents. Contrast this with an embedded object: an embedded object stores this material completely within the document.

A linked object is defined by a linked object field. The field value contains the beads that represent the foreign material. For a picture object, this will be a single Bitmap bead; for an object in Words material format, this will be the set of beads (e.g. Text, Paragraph, Row Start, Cell End) required to represent the object.

The field method contains the filename that contains the linked information. By default, this is the absolute pathname; however, it can be edited (in field method mode or from the Format → Linked Object dialog box) and turned into a relative pathname. A URL can also be used, if Applixware is properly configured to access the Web. Double-clicking upon the object invokes the foreign application. Note that it is a filename that is presented to the foreign application, not the field value. This means that edits made to the field value will be lost. See "Embedded Object" earlier in this chapter for more information.

When you double-click on a linked object, edit the object in its parent window, and save, the changes are immediately reflected in the containing Words document. The situation is different if you independently edit a file that happens to be linked into a containing Words document. For an Applixware document stored locally, if you edit it with an Applixware application under the same Applixware session, the changes again should be reflected immediately. Under any other circumstance (e.g. a non-Applixware file, or any file edited with a non-Applixware application), the link polling interval determines the frequency that linked files are examined to detect modification. By default, this interval is 5 minutes. This default can be overridden by setting the profile variable wpLinkPollSeconds to the desired number of seconds. A link to a URL, however, will never update automatically.

When a linked file is deleted, the field value will contain an error message.

The presence of linked files should be considered when a Words document is copied from one place to another.

A Words file can be linked to a Words document. The field value for this object does not contain an entire document; rather, it contains (approximately) the main flow of the linked Words file.

A linked object can be localized (converted into an embedded object). This is done through the Format ➝ Object ➝ Object Properties dialog box.

Related User Interface:

Insert ➝ Object From File ➝ [all]
Format ➝ Object ➝ Object Properties

Related Macros:

WP_CHANGE_LINK_INFO@
WP_CHILD_LINK@
WP_GET_DOC_LINKS_INFO@
WP_GET_LINKS_INFO@
WP_INSERT_LINK_FILE_FIELD@
WP_IS_CHILD@
WP_SET_DOC_LINKS_INFO@
WP_SET_LINKS_INFO@

Related Definitions:

Bead; Embedded Object; Evaluation; Field; Field Method; Field Value; Localization; Object, Picture

Local Attributes

In document material, an attribute that is set locally overrides the setting in the material’s paragraph style. If, at a later time, a change is made to the style’s definition for this attribute, this change will not affect this material; however, it will affect material that inherits this attribute. Local attributes can also be set in styles, which follow the same inheritance rules. See "Style" later in this chapter for more information.

For text and paragraph attributes, an array of flags is maintained (one for each specific attribute such as bold, size, or line spacing) that indicates whether the attribute is local or inherited. The following attributes are either all inherited or all local:

- Tabs
- Border and shading
- Frame attributes
- Leading glossary
- Line Spacing

Editing any of these attributes within a paragraph forces those attributes to become local for that paragraph or style.

Related User Interface:

Format → Borders & Shading
Format → Character Settings
Format → Demote
Format → Frame
Format → Paragraph Settings
Format → Promote
Format → Tab Settings

Related Definitions:

Inherited Attributes; Style

Localize

A linked object can be localized into an embedded object. After an object is localized, it no longer depends upon (or updates to stay in synch with) an external file. Instead, its contents at the time of the localization are copied into the Words document.

Related Macros:

WP_LOCALIZE_ALL_LINKS@
WP_LOCALIZE_LINK@

Related Definitions:

Embedded object; Linked Object; Object

Location

A location identifies a unique position within a document; for example, the position of the cursor. A location has two components:

- A bead number
- An offset

The user interface does not allow the cursor to be placed in every potential location. Some locations are invisible; other locations may be located in inaccessible flows.

Locations can be expressed in two formats: "start format" and "end format". Suppose bead 50 is a text bead containing the string "CAT", and bead 51 is a paragraph bead. The location between the beads can be expressed in start format as bead 51, offset 0 or in end format as bead 50, offset 3. (See "Offset" later in this chapter.) A cursor’s location is always expressed in start format; end format is only used for expressing the end of a range.

Related Macros:

WP_GET_BEAD_END@
WP_ILLEGALIZE_LOCATION@
WP_LEGALIZE_LOCATION@

WP_LOCATION_IN_RANGE@

Related Definitions:

Bead Number; Cursor; Flow; Offset; Paragraph Bead; Range; Text Bead

Marker Bead

A marker bead marks a position in a file with a unique string identifier.

In ELF macros that add or delete beads, you cannot store raw bead numbers of beads that come after the point at which material is inserted or deleted because bead numbers are altered to close up gaps in the numbering system. One of the ways to keep track of a location in a file is to use a marker bead.

Marker beads are inserted before or after a bead using the WP_INSERT_MARKER_BEAD@ macro. You can retrieve this location using the WP_GET_MARKER_BEAD@ macro.

Note that a generic Delete operation will not delete marker beads in the selected range. Also, inserting and deleting marker beads takes much more computer resources than inserting and deleting tags.

Beginning at Applixware version 4, markers can be saved within a file. Before version 4, marker beads were removed from the file before it was saved.
Related Macros:

WP_DELETE_SELECTION_MARKS@
WP_GET_SELECTION_MARKS@
WP_IN_SELECTION_MARKS@
WP_INSERT_MARKER_BEAD@
WP_PRESERVE_SELECTION@
WP_RESTORE_SELECTION@
WP_SET_SELECTION_MARKS@

Related Definitions:

Bead; Bead Number; Tag

Main Flow

The main flow contains the primary contents of the document. This flow does not include headers, footers, footnotes, style definitions, glossaries, and so on.

The main flow ends with a paragraph bead followed by a section bead. Neither of these beads can be deleted; even an "empty document" has these two beads.

Related Macros:

WP_IN_MAIN_FLOW@

Related Definitions:

Bead; Flow

Multi-Cell Selection

A selection is a range that indicates what material is selected. Usually, this range is all that is needed to identify the selected material—all material that falls within this range is considered to be selected (and is highlighted on the display). However, a special case exists when table cells or columns are selected. For example, suppose you drag the mouse from within cell 2 to cell 15:

cell 1	cell 2	cell 3	cell 4
cell 5	cell 6	cell 7	cell 8
cell 9	cell 10	cell 11	cell 12

cell 13	cell 14	cell 15	cell 16

Cells 2, 3, 6, 7, 10, 11, 14, and 15 are highlighted. The current selection is reported as beginning with cell 2 and ending with cell 15. However, this falsely implies that cells 4, 5, 8, 9, 12, and 13 are also selected.

For a multi-cell selection, the width of the selection in terms of number of cells is also recorded.

In the above example, the selection is 2 cells wide. Since the selection begins with cell 2, you now know that the second and third cell of each row in the selection is selected.

The status line at the bottom of the Words window indicates a multi-cell selection. In this case, it would contain the string "2 Cols". No status messages is displayed when a non-tabular selection is made.

When writing macros that operates upon the current selection, you must plan for multi-cell selections. The best way to do this is to use the WP_GET_NEXT_SELECTION@ macro. This macro can be called in a loop and will return the selected part of each row in the selection on each cycle. In the above example, each iteration of the loop would return a different range as follows:

- Cell 2 to cell 3
- Cell 6 to cell 7
- Cell 10 to cell 11
- Cell 14 to cell 15

If the current selection is a simple selection, the range is returned on the first iteration of the loop.

A multi-cell selection can only occur when one or more cells in a single table are selected. Such a selection can never contain a Row Start bead, material from outside a table, or material from multiple tables. Even if all the cells in a row are selected, the row itself is not considered to be selected unless the Row Start bead is selected.

Dragging from cell 1 to cell 4 in the above table generates a multi-cell selection of width 4. Note that the row is not selected. Clicking to the left of the row (to the left of the page margin) or dragging from the previous paragraph into the row, selects the entire row within a simple selection. Certain functions (most notably Delete) will behave differently in these cases.

Related Macros:

WP_GET_NEXT_SELECTION@
WP_NUM_COLS_SELECTED@
WP_SELECTION_IS_MULTIPLE@
WP_GET_NUM_SELECTIONS@
WP_SELECT_RANGE_CELLS@

Related Definitions:

Bead; Current Selection; Range; Selection; Simple Selection

Object

An *object* is a portion of self-contained data, displayed within a Words document. This data can consist of Words material, or data from another Applixware application, or material in a foreign format (e.g. GIF or ASCII). An object is either an *embedded object* or a *linked object*. An embedded object exists entirely and only within the containing Words file. In contrast, the linked object represents the current view of an external, independent file.

An object either exists as a *picture*, or as Words material. A picture object is displayed in single rectangle that may be no larger than a single Words page. The rendering is accomplished by the *parent application* (e.g. Applixware Spreadsheets, for a spreadsheet inset) and consequently appears exactly as it would in the parent application window. If the object is not a picture, it must be converted into Words material (text, frames, and tables). It is not limited to a single rectangle, however, and thus can span multiple pages.

Various object types can be inset as pictures and / or Words material, as follows:

Table 2-1 Object Types

Parent Application	**Permitted Object Types:**	
	Picture:	**Words Material:**
Words	Yes	Yes
Macros	Yes	Yes
HTML Author	Yes	Yes
Equation	Yes	No
Graphics	Yes	No
Spreadsheets	Yes	Yes
Data	No	Yes

As the table indicates, Graphics and Equation objects must be pictures; Data objects must be Words material, and the remaining object types can be either.

Embedded and linked objects are represented by fields. The field method contains a reference to the source of the material, and the field value contains the beads representing this material. For a picture object, the field value will contain only a single Bitmap bead. For an object in Words material format, the field value will contain the entire representation of the object (e.g. a table for a spreadsheet). Note the representation of non-Words material in Words is not exact; for example, the table representation of a spreadsheet does not contain the cell formulas.

Material from a foreign application may be inset as an embedded or linked object within Words. In order to accomplish this, a filter must exist to convert from the foreign format into one of the Applixware application formats. Once this is done, the object is subject to the requirements of the above table. For example, an Excel spreadsheet is converted to an Applixware Spreadsheet, and can be inset as either a picture object or as Words material, while a GIF file would be converted into an Applixware Graphics and inset only as a picture.

If direct editing of such foreign material is desired, you must provide an ELF macro that will bring up the foreign application when the user double-clicks on the object in Words.

Related Definitions:

Bead; Embedded Object; Field; Field Method; Field Value; Linked Object; Picture; Table

Object Bead

The object bead can contain non-Words material in an encoded format. An embedded object, which provides a view of non-Words material, refers to an object bead by means of a unique name.

Object beads are placed after the main flow of the document. By default, if an embedded object is deleted, its object bead is also deleted (when the document is saved); this behavior can be overridden. This means that a Words document can contain a "library" of objects, not all of which are visible.
Related User Interface:

Format → Objects

Related Macros:

WP_ADD_OBJECT@
WP_CONVERT_OBJECT@

WP_DELETE_OBJECT@
WP_GENERATE_NEW_OBJECT_NAME@
WP_GET_OBJECTS_INFO@
WP_OBJECT_ALLOW_UNREF@
WP_OBJECT_FILTER_MACRO@
WP_OBJECT_LAUNCH_MACRO@
WP_OBJECT_IS_REFERENCED@
WP_OBJECT_NAME@
WP_SAVE_OBJECT@

Related Definitions:

Bead; Embedded Object; Field; Main Flow

Offset

An offset is a position within a bead and defines a part of a location. The only bead that has multiple offset positions is the text bead. For example, if a text bead contains the string "CAT", offset zero represents the position before the ’C’, and offset one is between the ’C’ and ’A’. Offset three is after the ’T’. This last position is also offset zero of the following bead.

No other beads has an offset; if you specify an offset, offset zero is before the bead and offset one is after it.

Related Macros:

WP_GET_BEAD_END@
Related Definitions:

Bead; Location; Text Bead

Paragraph and Paragraph Bead

All basic document material (primarily text and graphics) must be in a paragraph; a cursor can only be placed in a paragraph.

The beads that make up a paragraph are:

- 1. Any document material
- 2. Paragraph bead

The paragraph bead contains the following information:

- The name of the paragraph’s style
- Local paragraph attributes such as line spacing or justification
- Local text attributes such as size or color ; these apply to the display of the paragraph marker only—text beads in the paragraph will have their own local text attributes
- Local tabs; if they are specified, none of the style’s tabs are used
- Local frame attributes
- Local border and shading attributes

A paragraph contains every bead that comes before it up to the preceding paragraph bead or start of flow. Paragraphs do not extend across multiple flows or cells; however, they can extend across sections. Every cell and flow ends in a paragraph. However, the main flow has a section following the last paragraph.

Related Macros:

WP_FIND_BEAD_OF_PARA@
WP_FIND_FIRST_IN_PARA@
WP_FIND_LAST_PARA@

WP_FIND_NEXT_PARA@
WP_FIND_PARA_OF_STYLE@
WP_GET_CURRENT_PARA_RANGE@
WP_GET_FLAGGED_PARA_ATTRS@
WP_SET_FLAGGED_PARA_ATTRS@
WP_GET_PARA_SETTINGS@
WP_SET_PARA_SETTINGS@
WP_GET_RANGE_OF_PARA@
WP_RESET_PARA_SETTINGS@
WP_SELECT_CURRENT_PARA@
WP_SELECT_WHOLE_PARAS@

Related Definitions:

Bead; Cell; Cursor; Flow; Main Flow; Paragraph Marker; Section; Style; Text Bead

Paragraph Marker

Paragraph beads are represented by the paragraph marker or pilcro symbol (¶). This symbol is visible at the end of each paragraph if the View ➝ Format Characters option is turned on.

Related User Interface:

View ➝ Format Characters

Related Definitions:

Paragraph Beads

Picture

An *picture* is a special type of *object*, which itself is a portion of self-contained data, displayed within a Words document. This data can consist of Words material, or data from another Applixware application, or material in a foreign format (e.g. GIF or ASCII).

A picture object is rendered into a single rectangle (clipped to a single Words page) within the Words document by the appropriate Applixware application. For example, a spreadsheet object (whether in Applixware Spreadsheets format, or a foreign format such as Excel) is rendered by the Applixware Spreadsheets application. Thus the object should appear in Words exactly as it does in the parent application. In contrast, an object in Words material format can span multiple pages, but the conversion into Words material may make it look somewhat different than it does in its parent application.

All picture objects may be scaled larger or smaller. In addition, the user interface allows a portion of the object to be displayed. For example, a specific range of paragraphs or pages can be displayed for a Words picture inset, and a specific cell range for a Spreadsheet picture inset. This functionality is available through the Format ➝ Object ➝ Applix Graphics, Applix Spreadsheets, and Applix Words dialog boxes.

Picture objects may be converted to Words material format, and vice versa, via the Format ➝ Object ➝ Object Properties dialog box, subject to the requirements of the Object Types table in the Object section. Prior to release 4.3, only graphics objects could be pictures. If a 4.3 document is opened by an earlier version of Applixware Words, all other picture objects will be interpreted as Words material.

Related User Interface:

Format ➝ Object ➝ Applix Graphics
Format ➝ Object ➝ Applix Spreadsheets
Format ➝ Object ➝ Applix Words
Format ➝ Object ➝ Object Properties
Related ELF formats:

format wp_inset_info@
    see wp_.am in axdata/elf

Related Macros:

WP_INSERT_LINK_FILE_FIELD@
WP_INSERT_EMBEDDED_OBJ_FIELD@
WP_GET_OBJECT_PROPERTIES@

Related Definitions:

Embedded Object; Linked Object; Object

Range

A range uniquely identifies a portion of the document. It consists of a start and end location; the start location is in "start format" and the end location is in "end format".

Many ELF functions accept a range as an argument; this range indicates the portion of the document upon which it will operate.

The current selection is expressible as a range. If the current selection is a cursor, the start and end locations both contain the cursor location (and in this special case, the end location is in "start format").

For example, suppose bead 50 is a text bead containing the string "CAT", and bead 51 is a paragraph bead.

- If the word "CAT" is selected, the current selection would have a range of bead 50, offset 0 to bead 50, offset 3 —or in compact notation (50,0 to 50,3)
- If "CAT" and the paragraph marker were selected, the selection would be (50,0 to 51,1)
- If there was a cursor in front of the ’C’, the selection would be (50,0 to 50,0)
- A cursor in front of the ’A’ would be (50,1 to 50,1)

- A cursor in front of the paragraph marker would be (51,0 to 51,0)

A selection contain cells may require additional information. See multi-cell selection.

Related Macros:

WP_LOCATION_IN_RANGE@
WP_RANGE_IS_CURSOR@

Related Definitions:

Current Selection; Cursor; Location; Multi-cell Selection; Paragraph Bead; Paragraph Marker; Text Bead

Redo

See "Undo and Redo" later in this chapter.

Row and Row Start Bead

A row consists of one or more horizontally adjacent cells. A set of vertically adjacent rows makes up a table.

The beads that make up a row are:

- Row Start bead
- Beads for a least one cell, as described in the *Cell* definition earlier in this chapter

The information contained in the Row Start bead includes:

- The row’s offset from the left margin and justification (left, center, or right)

- The top and bottom row margins; these margins separate the row from the material such as other rows that are above and below it
- *Growth policy* and height (if needed) for the row, as follows:

  All cells in the row have the same height; they may have a fixed height or grow to fit the tallest material in any of the cells, and so on

Related User Interface:

Table → Insert Cells
Table → Make Row Fit
Table → Merge Rows
Table → Row Attributes
Table → Select Row
Table → Split Row Before [After] Cell

Related ELF formats:

format wp_row_attrs@
    see wp_.am in axdata/elf

Related Macros:

WP_COPY_ROW@
WP_GET_FIRST_ROW@
WP_GET_LAST_ROW@
WP_GET_NEXT_ROW@
WP_GET_PREV_ROW@
WP_GET_RANGE_OF_ROW@
WP_GET_ROW@
WP_GET_SELECTED_ROW_ATTRS@
WP_INSERT_ROWS@
WP_NUM_CELLS_IN_ROW@
WP_NUM_ROWS_IN_TABLE@
WP_PUT_RANGE_IN_ROW@
WP_REMOVE_ROW@

WP_ROW_NUM@
WP_SET_ROW_ALIGNMENT@
WP_SET_ROW_ATTRS@
WP_SET_ROW_HEIGHT@
WP_SET_ROW_HEIGHT_TYPE@
WP_SET_ROW_INDENT@
WP_SET_ROW_MARGINS@
WP_SET_SELECTED_ROW_ALIGNMENT@
WP_SET_SELECTED_ROW_ATTRS@
WP_SET_SELECTED_ROW_HEIGHT@
WP_SET_SELECTED_ROW_HT_TYPE@
WP_SET_SELECTED_ROW_INDENT@
WP_SET_SELECTED_ROW_MARGINS@
WP_SPLIT_ROW@
WP_MERGE_ROW@

Related Definitions:

Bead: Cell; Table

Section

A section is a high-level document structure that controls the "page layout" of the material that falls within it.

The beads that make up a section are:

- Document material
- Section bead

The section bead contains the following information:

- Page size and orientation (landscape or portrait)
- Left, right, top, and bottom margins
- Number of columns and column gutters

- Header and footer information
- Page numbering format

A section is considered to contain every bead that comes before it until the preceding section bead or start of flow. Sections cannot extend across flows.

Section attributes do not apply to specific pages; rather, they apply to all material of a section. That is, there is no way to force page 3 to have 2 columns. Instead, a section can set the document material it affects into 2 columns. This 2-column information will then be displayed on pages. As editing operations move this material to different pages, those pages will also be formatted by the section.

By default, the Main flow ends with a section bead (following the final paragraph bead). This section is not displayed even if View → Breaks is on.

In a simple document, as long as all page formatting applies to the entire document, this will be the only section.

When some document material is selected and page attributes are applied, it turns out that *two* section beads need to be inserted. For example:

- Preceding document material
- Selected document material
- Following document material
- Section bead

In the above example, the existing section bead controls the page formatting of all the indicated material. Suppose you put the "selected document material" into two columns. You will end up with the following:

- Preceding document material
- Section bead S1
- Selected document material—2 columns
- Section bead S2
- Following document material
- Section bead (original)

Section bead S2 puts the selected document material into two columns. Section bead S1, a copy of the original, terminating section bead, keeps the preceding document material in its original format; that is, it is as determined by the original section bead.

The Use Section Properties toggle on the Insert → Linked Object → Applix Words dialog box determines whether this final section bead is copied into the containing document (along with the rest of the document being linked).

Related User Interface:

Format → Columns
Format → Header and Footer
Format → Page Numbering
Format → Page Setup
Format → Section Type
Insert → Break
Insert → Quick Header [Footer]

Related Definitions:

Bead: Flow; Main Flow; Paragraph Bead

Selection

The current selection can be a selection or it can be a cursor. If it is a selection, the selected material appears in inverse video. Any operation that inserts material (typing, pasting, inserting a field, and so on) replaces the selected material with the new material.

Applixware 3.0 uses a single-selection model. That is, only one selection can exist at a time. Thus, an operation such as global search and replace must find and replace successive instances of the target material, rather than finding and selecting all the matches and then replacing the selections.

A selection is described by a range. This range indicates exactly which material is selected. Usually, this range is sufficient to identify the selected material. The only special case is when table cells or columns are selected.

Related Macros:

```
WP_SELECT_RANGE@
WP_SELECT_RANGE_CELLS@
```

Related Definitions:

Current Selection: Cursor; Multi-cell Selection; Range

Series Definition & Series Field

A series definition is a bead that defines the method by which an ordered set of strings is generated. This definition includes:

- The format of the string for each level; ELF macros can be used to generate such formats as January, February, ...
- Characters that separate the level
- Optional leading and trailing string
- Whether one or more levels should be displayed

The series field generates the desired string using the series definition and then displays it as its field value. By default, the value of this string is one greater than the previous series field value of the same type. However, the user can override this default in a variety of ways. See "Series Field" in Chapter 3.

Related Definitions:

Bead; Field; Field Value

Simple Selection

A simple selection is a selection described by a range such that all material that falls within the range is in fact selected. If a simple selection contains table material, it contains entire rows. Contrast this form of selection with Multi-cell Selection.

Related Macros:

WP_SELECTION_IS_MULTIPLE@
WP_SELECT_RANGE@

Related Definitions:

Multi-cell Selection; Range; Selection

Style

A style is a set of attributes that defines the initial look of a paragraph and its contents. A document contains at least one (and usually several) style definitions. The initial paragraph attributes inherit the style's attribute definition.

The style is defined by a single bead called the style bead. Style beads reside before the main flow. An end user can access these style definitions by using the Format → Style dialog box.

A style definition contains the following information:

- Paragraph attributes (for example, line spacing, justification)
- Text attributes (for example, size, color)
- Tabs
- Frame attributes
- Border and shading attributes
- Glossary

Style definitions are arranged in a hierarchy (a "tree structure") as follows.

- There is only one top-level style; it is usually named "Normal"
- All other styles are either children of the top-level style, or children of its children, and so on
- Every style has one parent; the only exception is the top-level style, which has no parent
- Any style can either set its own unique attributes or inherit them from its parent.

- The top level style has all its attributes set locally; that is, no attributes are inherited

This tree structure determines the visible, or effective, attributes of paragraph material. For example, suppose some 13-point text exists in a document.

- The text may have this size attribute set locally
- The text size may be the point size defined explicitly for this paragraph's style
- The text size may be the point size defined implicitly for this paragraph's size by inheritance from a parent style

For every attribute, the attribute's definition can come from one of three places:

- From the text or paragraph
- From the paragraph's style
- From a parent of the paragraph's style

Searching "upwards" in the tree, the first non-inherited value found specifies the attribute to be used. Since the top-level style has all local attributes, an explicitly set attribute must exist. If document material inherits an attribute from a style, and the style's definition changes, the look of the displayed material will be automatically updated.

Related User Interface:

Format → Apply Style
Format → Style

Related Definitions:

Bead; Main Flow

Table

A table is a set of vertically adjacent rows. A table has no special beads or properties beyond those contained in the rows. If rows are separated by a paragraph that is not in a cell, the rows are not adjacent. This means the rows are not in the same table.

Related User Interface:

Table ➝ Convert Table to Text
Table ➝ Convert Text to Table
Table ➝ Insert Table
Table ➝ Select Table
Table ➝ Split ➝ Table Before [After] Row

Related Macros:

WP_GET_FIRST_ROW@
WP_GET_LAST_ROW@
WP_GET_RANGE_OF_TABLE@
WP_INSERT_TABLE@
WP_INSERT_TABLE_AT_LOC@
WP_IS_BEAD_IN_TABLE@
WP_NUM_ROWS_IN_TABLE@

Related Definitions:

Bead; Paragraph; Row

Tag

A tag marks a location in a file with a unique string identifier.

In ELF macros that add or delete beads, you cannot store raw bead numbers of beads that come after the point at which material is inserted or deleted because bead numbers are altered to close up gaps in the numbering system. One of the ways to keep track of a location in a file is by using a tag.

- A location can be "tagged" using the WP_ADD_TAG@ macro
- It can be retrieved using the WP_GET_TAG_LOCATION@ macro

If a tagged location is deleted, the tag "moves forward" to the next location. This could cause tags to overlap. For example, assume that a paragraph contains more than one tag and the paragraph is deleted. In this case, all the tags would point to the first location in the next paragraph. In addition, the original tag ordering is lost.

If ordering is important, use marker beads. However, inserting and deleting marker beads is more (computationally) expensive than using tags.

Tags are not saved into the .aw file.

The WP_TAG_INFO_DLG@ dialog box can help you debug an ELF macro that uses tags. (It is listed on the Help pulldown within the menubar editor as follows: Help → Document Information → Tag Information; however, it is not displayed in the distributed menubar.) It contains a list of all the current tags and their corresponding locations.

Related Macros:

WP_TAG_SELECTION@
WP_TAG_RANGE@
WP_RESTORE_TAGGED_SELECTION@
WP_RETRIEVE_TAGGED_RANGE@
WP_TAG_INFO_DLG@

Related Definitions:

Bead; Bead Number; Location; Marker Beads

Text Bead

All text in a document is stored in text beads. A text bead contains the following information:

- The text
- The text's local text attributes (for example, size, color)

Every character in the text bead has the same displayed attributes. If a change in a displayed attribute occurs, then you know that the information is associated with another text bead. For example, a paragraph that contains a bold word somewhere within the paragraph must contain at least 3 text beads; the middle one indicates the bold attribute and the leading and trailing beads that do not have this attribute.

Text beads must not be empty.

Every text bead is associated with a paragraph; the paragraph's style determines the inherited text attributes.

Related Macros:

WP_GET_FLAGGED_TEXT_ATTRS@
WP_SET_FLAGGED_TEXT_ATTRS@
WP_GET_TEXT_ATTRS@
WP_SET_TEXT_ATTRS@
WP_RESET_TEXT_ATTRS@

Related Definitions:

Bead; Local Attributes; Paragraph; Style

Undo and Redo

Every time a bead is changed or deleted, the original bead is first preserved. When a bead is added, the new bead is added to the "undo" list. The Undo function reverses the operation (restoring a deleted bead or the original version of a changed bead and removing an added bead). Redo undoes the Undo in a symmetric fashion. Only bead-level information is recorded. This means that other changes such as changing a View command cannot be undone.

Data about bead-level operations is stored in a temporary file. The number of operations that can be undone is limited by the size of this file.

Storing this data clearly has a performance cost. The macro WP_UNDO_ACTIVATE@ can turn Undo recording off and back on. You would shut off recording when running an ELF macro that does not give the user an opportunity to Undo.

The ELF macro writer should nest the highest-level user interface between WP_UNDO_START@ and WP_UNDO_END@, if the user is to be able to reverse the action with a single Undo command. Otherwise, the user will be forced to Undo the individual commands making up your macro.

Macros should catch their own errors and issue a WP_UNDO_END@ (assuming a prior WP_UNDO_START@ is in effect). If this is not done, and a macro throws an error and exits while a WP_UNDO_START@ is in effect, the Undo nesting will be unbalanced. The Undo file will be discarded when this is detected, denying the user the chance to Undo any changes.

Related User Interface:

Edit → Redo
Edit → Undo

Related Macros:

WP_REDO@
WP_UNDO@
WP_UNDO_ACTIVATE@
WP_UNDO_CLEAR@
WP_UNDO_END@
WP_UNDO_START@

Related Definitions:

Bead

3 Field Method Syntax

This chapter describes the syntax of field methods. The following fields are discussed:

Conditional Variable	Macro
Cross Reference	Macro String
Cross Reference Source	Make Index
Date	Make Table of Contents
Entry	Merge
Footnote	Page Count
Footnote Body	Page Number
Glossary	Plain
HTML Tag	Printer Code
Hyperlink	Series
Hypertarget	Soft Hyphen
Index	Time
Link	Variable String
Local Inset	

Field Methods

A field method is the part of the field structure that contains the material (usually simple text) that identifies and defines the field. A process called *evaluation* analyzes this field method and generates a field value, which is usually what the user sees in the document. For example, the date field contains a method that identifies the field type (date), and contains a code indicating the desired format. The displayed field value is the current date in the desired format.

The View → Field Methods menu option controls whether all field values (the default) or field methods are displayed.

When a field is created through the user interface, the dialog box settings generate the field method automatically. However, if the user wants to edit the field method or add a feature not supported by the dialog box interface, a knowledge of the proper field method syntax is necessary.

Field Evaluation

All fields are evaluated as follows:

- When the field is first created
- When the field method is edited
- When the field is selected and the Tools → Field Editing → Evaluate Fields menu item is used

Any other circumstances that trigger a field's evaluation are listed in the field descriptions that follow.

Field Method Syntax Conventions

The following conventions are used in this chapter:

- **Bold Helvetica** represents literal text, and needs to be specified exactly as indicated.
- *Italicized Helvetica* represents variable information, either to be specified by the user, or selected from a limited list of options.
- Items in [square brackets] are optional.
- Items in "double quotes" should be specified with double quotes.
- Items separated by the "or" symbol (|) offer a set of alternatives. Unless otherwise indicated, only one may be selected.
- If your string contains a double-quote character (") or a backslash character ('\"), you must type a backslash *escape* character before it. For example: \" or \\.

Error Handling

If the evaluation of a field method generates an error, the error is usually placed in the field method. This can be due to improper field method syntax, as follows:

ERROR: Too few field arguments	A required part of the field syntax is missing
???	New field has not yet been evaluated or some undefined problem
Missing a quoted string argument	A flag was specified in the field method that is should be followed by a quoted string; however, no string was detected

Errors that are specific to one or more fields are documented with the field.

Common Flags

The *evalFlags* and the *textStyle* flags can be used in many or all of the fields.

evalFlags

evalFlags = **-readEval | -printEval | -saveEval | -noEval | -emptyValue**

The *evalFlags* force the reevaluation of a field under the following circumstances:

Table 3-1 evalFlags

Flag	Meaning
-readEval	Evaluate field when the document is opened
-printEval	Evaluate field when the document is printed
-saveEval	Evaluate field when the document is saved
-noEval	If evaluation occurs, it has no effect; the Field Start contents are not changed

Table 3-1 evalFlags (cont.)

Flag	Meaning
-emptyValue	If evaluation occurs, an empty field value is generated; the method is ignored -noEval has higher precedence than **-emptyValue**; that is, if -noEval is used, the existing contents do not change **NOTE:** This option is available beginning at Applixware version 4
-preserveVal	If evaluation occurs, only the attributes of the field can change (e.g. shrink to fit, text attributes). The actual text in the field value does not change. **NOTE:** This option is available beginning at Applixware version 4.3

One or more of these flags can be specified in the same field.

textStyle Flag

textStyle = **-textStyle** "*styleName*"

The *textStyle* flag applies the text attributes of the specified style to the text in the field value. *styleName* is the name of the style (as it appears in the Style pulldown and the Style dialog box) and must be enclosed in double quotes.

Conditional Variable Field

Description When evaluated, the Conditional Variable field examines the desired document variable; if it is not empty or zero, the remaining material in the field method is copied into the field value.

The Conditional Variable field is reevaluated whenever the referenced document variable changes in value.

Format { **if_var** "*varName*" *material* [*evalFlags*] }

Arguments The *varName* is the name of the document variable in question (as displayed in the Utilities ➝ Special Fields ➝ Set Document Variables dialog box); it must be enclosed in double quotes. It is evaluated as:

Variable Type	Conditional Status
Number	TRUE if variable exists and is non-zero
String	TRUE if variable exists and is not the empty string
Array	TRUE if the variable exists and has an array size of at least one item
anything else	Puts an error message in the field value

If the variable does not exist, it is FALSE. The remainder of the field method is the material copied into the field value if appropriate.

Interface Tools ➝ Special Fields ➝ Conditional Upon Variable

Cross Reference Field

Description If the Cross Reference field is in the main flow then, when it is evaluated, the Cross Reference field copies the field value of the matching Cross Reference Source field into its own field value. For a Cross Reference field in a header or footer, the material from the matching Cross Reference Source field is only displayed; the beads are not actually copied. See "Cross Reference" in Chapter 2 for more information.

The Cross Reference field is reevaluated if the matching Cross Reference Source field has the "live" property, and the material in the Cross Reference source field is edited.

Format { **xref** *"xrefName"* [*directionFlag*] *pageFlag* [*evalFlags*] [*textStyle*] }

Arguments The *xrefName* flag provides a unique name for the material in the field value; the same name is used by Cross Reference fields to access this material. It must be enclosed in double quotes.

The *directionFlag* and *pageFlag* are used when more than one matching Cross Reference Source field can exist in the document. These flags help identify which Cross Reference Source field should be used. For example, in a report with multiple sections, each section may define a section name in a Cross Reference Source field with a common name and repeat the section name within the section (for example, in a header) by using a Cross Reference field.

The *directionFlag* indicates the direction the search for the matching Cross Reference Source field should take:

Table 3-2 directionFlags

directionFlag	**Effect on search**
-prev	Starting at the Cross Reference field, search backwards for the first matching Cross Reference Source field encountered; stop at the top of the document
-next	Starting at the Cross Reference field, search forwards for the first matching Cross Reference Source field encountered; stop at the bottom of the document
(none)	First look backwards; if a matching Cross Reference Source field is not encountered, look forwards

The *pageFlag* is ignored unless the Cross Reference field is in a header or footer. In this case, it is required. If a Cross Reference field is in a header or footer, the matching Cross Reference Source field is searched for within the main flow of the document, not within the header or footer. If the same header or footer is visible on more than one page, the search is carried out for each page.

For every page, the *pageFlag* controls whether the search should begin at the top or the bottom of the page, as follows:

Table 3-3 pageFlag

pageFlag	**Effect on search**
-pageTop	Start searching at the top of the page upon which the header or footer falls in the direction indicated by the *directionFlag*
-pageBottom	Start searching at the bottom of the page upon which the header or footer falls in the direction indicated by the *directionFlag*

Error Handling "No such cross-reference source field":
The source field could not be located.

Interface Tools → Cross Reference → Cross Reference

Cross Reference Source Field

Description The Cross Reference Source field indicates that its field value is available to be cross-referenced. No action is taken when the field is evaluated.

Format { **xref_source** *"xrefName"* [**-live**] }

Arguments The *xrefName* flag provides a unique name for the material in the field value. The same name is used by Cross Reference fields to access this material. It must be enclosed in double quotes.

The **-live** flag, if specified, indicates that when the cross reference source material is edited, the matching Cross Reference fields should copy the updated material.

Interface Tools → Cross Reference → Cross Reference Source

Date Field

Description When evaluated, the Date field generates a string representing the current date in the specified date format.

Format { **date -format** *"formatStr"* **|** *formatCode* [*evalFlags*] [*textStyle*] }

Arguments The **-format** *"formatStr"* flag and value pair specify the exact format of the date to be displayed in the field value. The string consists of a mixture of day, month, and year codes with miscellaneous text. For example, the string "Mmm d, yyyy" would yield (for example) "August 24, 1993"; note the preservation of the spaces and comma. A list of the day, month and year codes can be found in the file datetime.sp under the Applixware release directory, or in help under DATE_FORMAT@ macro. Note the use of *formatStr* is new for release 4.1. For earlier releases, the *formatCode* should be used as described below.

The *formatCode* is an integer representing a date format, as shown below. Release 4.0 and earlier will ignore the *formatStr* and use the *formatCode*. Note that it is acceptable to specify both a *formatCode* and a *formatStr*; earlier releases will use the former and later releases the latter.

Table 3-4 Date Formats

formatCode	**Matching *formatStr***	**Display**
0	mm/dd/yy	08/24/93
1	Mm dd, yyyy	Aug 24, 1993
2	dd.mm.yy	24.08.93
3	dd Mm yy	24 Aug 1993
4	Mmm dd, yyyy	August 24, 1993
5	yyyy-mm-dd	1993-08-24
6	yy-mm-dd	93-08-24
7	yyyy mm dd	1993 08 24
8	yy mm dd	93 08 24
9	yyyymmdd	19930824
10	yymmdd	930824
11	dd/mm/yy	24/08/93
12	dd.mm.yyyy	24.08.1993

Interface Insert → Date & Time

Entry Field

Description The Entry field, like the Form cell, provides a dynamic and constrained area for the entry of text when Words is in Forms mode. No action is taken when the field is evaluated. The Entry field is displayed as an area lightly shaded in gray; this shading does not print. This field is available in release 4.1.

Format { **entry** *"fieldId"* [**-locked**] [**-minWidth** *width*] [**-maxChars** *num*]
[**-nextIfFull**] [**-nextId** *"nextId"*] [**-case** *caseFlag*]
[**-underline**] [**-promptText** *"text"*]
[**-dblClickMacro** *"macro"*] [**-enterMacro** *"macro"*]
[**-exitMacro** *"macro"*]}

Arguments The *fieldId* is a required string identifying this particular Entry field. The same name is used by **-nextId** flags in other Entry fields and nextID values in Form cells to access this field. It must be enclosed in double quotes.

The **-locked** flag, if specified, indicates that the user cannot navigate into, select, edit, or delete this field when in Forms mode.

The **-minWidth** *width* flag and value pair, if specified, indicate the minimum displayed width of the Entry field. *width* should be an integer in mils (there are 1000 mils per inch). If no minimum width is specified, an empty Entry field will take up no width and will be difficult for the end user to find. Note this flag is ignored if the field splits across multiple lines.

The **-maxChars** *num* flag and value pair, if specified, indicate the maximum number of characters that can be typed into the field in

Forms mode. When this maximum number is reached, Words beeps and then behaves in a manner dependent on the **-nextIfFull** flag, as described below. If **-maxChars** is omitted, there is no limit to the number of characters that can be entered into an Entry field.

The **-nextIfFull** flag, if specified, indicates that when an Entry field is full, the cursor should be moved to the end of the next Entry field or Form cell. The "next" object is identified by *nextId*, as described below. This action only takes place in Forms mode, in the case where a maximum number of characters has been specified for the Entry field (via the **-maxChars** flag, as described above), and this limit has been reached. If the **-nextIfFull** flag is omitted, the cursor will remain in the full Entry field.

The **-nextId** *"nextId"* flag and value pair, if specified, indicate the id of the Entry field or Form cell to which the cursor should be moved if the user performs the Goto Next Cell operation (by default, this occurs when the tab key is hit in Forms mode), or when the field fills up (as specified by the **-maxChars** and **-nextIfFull** flags, above). If the **-nextId** flag is omitted, the "next" object is the next Entry field or Form cell following the current Entry field.

The **-case** *caseFlag* flag and value pair, if specified, indicate an override to the case of text typed into the Entry field in Forms mode. The *caseFlag* value is an integer as follows:

Table 3-5 caseFlag

caseFlag	**Effect on typed text**
0	No effect
1	Convert to all lowercase
2	Convert to all uppercase

The **-underline** flag, if specified, draws an underline across the entire width of the field, including the empty portion of the field (only relevant if **-minWidth** is specified as above). Unlike the gray shading of the Entry field, the underline does print.

The **-promptText** *"text"* flag and value pair, if specified, indicate the text that should appear on the status line at the bottom of the Words window when the cursor is in the Entry field in Forms mode.

The **-dblClickMacro** *"macro"* flag and value pair, if specified, indicate the macro to be executed when the Entry field is double-clicked on in Forms mode. The specified macro will receive two arguments: the bead number of the Field Start bead of the Entry field, and its *fieldId*.

The **-enterMacro** *"macro"* flag and value pair, if specified, indicate the macro to be executed when the cursor enters the Entry field. The specified macro will receive two arguments as described above.

The **-exitMacro** *"macro"* flag and value pair, if specified, indicate the macro to be executed when the cursor leaves the Entry field. The specified macro will receive two arguments as described above.

Interface

Tools → Forms → Entry Fields and Form Cells

Footnote Field

Description

When evaluated, the Footnote field generates a string representing the current footnote number. This number is displayed in a superscripted position at the field location. A "matching" Footnote Body field should exist elsewhere in the document.

Format

{ **footnote** [**-endDoc**] [*autoNumbering* **|** *textFlag "mark"*] [*evalFlags*] [*textStyle*] }

Arguments

The **-endDoc** flag, if present, indicates that the matching footnote body is placed at the end of the document. If it does not exist (which is the default), the footnote body is placed at the bottom of the page that contains the footnote field.

By default, the Footnote$ and FootnoteEndDoc$ series are used to generate numbers for footnotes. The number that is displayed is one more than the previous footnote of the same type. However, you can modify or override value as follows:

- The *autoNumbering* flag can consist of one or both of the following options:

-set *number*	Forces the value to the specified integer *number*
-series "seriesName"	Overrides the default series with the indicated series

- Alternatively, the *textFlag* can directly specify the footnote symbol or string to use as follows:

-text "*mark*"	*mark* must contain normal (Latin) characters
-dtext "*mark*"	*mark* is displayed in Dingbats typeface
-stext "*mark*"	*mark* is displayed in Symbol typeface

For example, to create an asterisk-style footnote, use the following:

```
{ footnote -text "*" }
```

Interface Insert ➝ Footnote

Footnote Body Field

Description When evaluated, the Footnote Body field generates a string representing a copy of the "matching" Footnote field (that is, the footnote number or mark).

Format { **footnote_body** [*evalFlags*] [*textStyle*] }

Error Handling "No such footnote": Cannot find the matching footnote field.

Interface Insert ➝ Footnote

Glossary Field

Description When evaluated, the Glossary field copies the contents of the matching glossary definition.

If the glossary definition has a "live" property, the field is reevaluated when the glossary definition changes.

Format { **glossary** *"glossaryName"* [*evalFlags*] [*textStyle*] }

Arguments The *glossaryName* flag indicates the name of the glossary definition, as displayed in the Inset ➝ Glossary ➝ Define Glossary and Insert ➝ Glossary ➝ Glossary dialog boxes. It should be enclosed in double quotes.

Error Handling "glossaryName: No such glossary": The glossary definition specified in the glossary field does not exist.

Interface Insert ➝ Glossary ➝ Define Glossary

Insert ➝ Glossary ➝ Glossary

HTML Tag Field

Description The HTML Tag field provides a place to store HTML tags that the HTML to Words import filter cannot convert into Words material. The Words to HTML export filter converts these fields back into the HTML material. The field value will display the HTML tag only in the HTML Author application when in Edit mode and the View HTML Tags option is on. The field value is empty under all other circumstances. This field is available starting in release 4.2.

Format { **html_tag** **-text** *"htmlText"* [*textStyle*] }

Arguments The **-text** *"htmlText"* flag and value pair specify the the HTML material. For an HTML tag, this should include the leading and terminating angle brackets

By convention, the HTML to Words import filter specifies a *textStyle* of html_unknown_text for all HTML Tag fields, so that these fields will be more easily identified.

Interface Insert → HTML Tag (HTML Author only)

Hyperlink Field

Description When triggered, the Hyperlink macro moves the cursor to the first matching Hypertarget field in the document. It first searches the main flow, followed by the flows for footnotes, headers, and footers).

The Hyperlink field is not a separate field type; rather, it is a specific instance of the Macro field.

Format { **macro "wp_hyperlink@" -pass** "*targetName*" **-pass** "*linkFile*"... }

Arguments As described above, the Hyperlink field is actually a Macro field. When it is triggered, the ELF macro WP_HYPERLINK@ is called and *targetName* and *linkFile* are passed to it. Both *targetName* and *linkFile* may be specified, or one may be left as the empty string (**-pass** "")

If *linkFile* is not specified, this macro moves the cursor to the Hypertarget field in the current document matching the *targetName.*

If *linkFile* is specified, if it is a document matching the current application type (a Words document if running Words, or an HTML document if running the HTML Author), the specified document is opened in the current window. The the document does not match the current application, a new application window is opened and the specified document loaded into that window. As of release 4.3, it is permissible for *linkFile* to be a URL. If *targetName* is specified as well, the cursor is moved to the matching Hypertarget field; if not, the cursor is left at the beginning of the document.

The remaining arguments are described in the Macro field section.

Interface Insert → Hyperlink

Hypertarget Field

Description The Hypertarget field marks a location in the text. As of release 4.2, it is allowed to contain text. Prior to this release, its field value was always empty, and no action was ever taken when the field was evaluated.

Format { **hyper_target** *"targetName"* [*textFlag* *"textStr"*] [*evalFlags*] [*textStyle*]}

Arguments The *targetName* should be a unique string, enclosed in double quotes. When a matching Hyperlink field is double-clicked, the cursor is moved in front of the Hypertarget field.

The rest of the flags are only serviced in release 4.2 and later.

The *textFlag* and *textStr* pair determine what, if anything, is displayed in the field value as follows:

-text "*textStr*"	*textStr* is considered to consist of normal (Latin) characters.
-dtext "*textStr*"	*textStr* is displayed in Dingbats typeface.
-stext "*textStr*"	*textStr* is displayed in Symbol typeface.

Interface Insert → Hypertarget

Index Field

Description The Index field contains text that can be placed in an index. The field value is always empty and no action is taken when the field is evaluated.

Format { **index** *"indexText"* [**-span**] [**-see** *"seeAlsoText"*] }

Arguments The *indexText* is the string to place in the index. A colon character separates entries that have multiple levels; for example, "Animals:mammals:cats". The string must be enclosed in double quotes.

The **-span** flag indicates that multiple entries are shown as a range of pages rather than individual pages. For example, if you did not use this flag for a set of Index Entry fields containing identical *indexText* appears on pages 17, 18, and 19, the index entry would contain the page references: "16, 17, 18". However, if all of these Index Entry fields included the **-span** flag, the page reference would be "16-18".

The **-see** flag indicates that, instead of a page number, the *seeAlsoText* is to be included in the index, preceded by "See" or "See also". For example:

{ index "Animals:mammals:cats -see "felines" }

This field generates the entry "*See also* felines" under the "cats" entry, unless it is the only item under "cats", in which case it generates "*See* felines" next to "cats"

Interface Tools → Book Building → Index Entry

Link Field

Description When evaluated, the Link field converts the indicated, external file into a format appropriate for display, and places the necessary material (either a Picture bead, or the results of the conversion) into the field value. When triggered, the link field opens the indicated file in its "parent application". At this time, a user can edit it. Evaluation and triggering of the field are separate actions.

The field value of a link field is not saved when the document is saved (except for unprotected, non-picture insets). This means that the field is reevaluated when the document is opened. In addition, the field is reevaluated when the linked file is edited.

Format { **link** "*linkFile*" [**-docType** *type*] [**-appType** *"type"*]
[**-picture**] [**-cvt** *"filterName"*] [**-app** "*filterName*"]

[**-view** "*viewName*" [**-ssObject | -ssView | -ssRange**]]
[**-shrinkToFit**] [**-lastSection**]
[**-htmlAnchor** *"anchorURL"*]
[*evalFlags*] [*textStyle*] }

Arguments

The *linkFile* is the absolute or relative pathname of the file being linked. It should be enclosed in double quotes. The *linkFile* is required. As of release 4.3, it is permissible for *linkFile* to be a URL.

The **-docType** flag specifies one of the document type codes (an integer) specified in the recgfil_.am file. This code represents the input file type of the file being linked. Note this can represent an Applixware file type or a foreign file type. Use 166 (AX_DOC_TYPE_UNKNOWN) for document types unknown to Applixware (in this case, the -cvt flag must be specified as described below)

The **-appType** flag indicates the Applixware application (via a quoted string) corresponding to the type of material in the linked file. For example, Applixware Graphics must be specified for a GIF file. The values that you can use are specified within app_ids_.am. Here are some of the values:

Table 3-6 appTypeCode Flags

values	Application Type
"Words_"	Applix Words
"Graphics_"	Applix Graphics
"Spreadsheets_"	Applix Spreadsheets
"Macros_"	Applix Macros
"Icons_"	Applix Bitmap
"Equations_"	Applix Equation
"DATA_"	Applix Data

Table 3-6 appTypeCode Flags (cont.)

"HTML_Authoring_"	HTML

If the **-picture** flag is present, the linked file is treated as a *picture object*. Otherwise, the linked file is represented by Words material. See the Object section in Chapter 2 for more details.

The **-cvt** flag names the macro that converts the foreign file into an Applixware file. Note the conversion is done automatically for foreign documents that Applixware can automatically recognize and import. You only need specify a macro if the file cannot be recognized.

The **-app** flag names the macro that is to be called when you double-click on the linked file. If omitted, a standard macro will be called based upon the doctype of the file (e.g. WP_START_INSET@ for an Applixware Words file, WP_START_FRAME20_INSET@ for a Frame 2.0 file, etc.).

If the **-view** flag is present and the linked file is an Applix Spreadsheets file, *viewName* specifies a named view, range, or chart; only the appropriate material is displayed in the link. If the linked file is a non-Applixware file, *viewName* is passed as an argument to the *filterName* conversion filter described above. Note that as of release 4.3, a view name specified in the picSource element of the Picture bead (for a Spreadsheet picture inset) will override *viewName*, if any. The 4.3 release does however redundantly maintain the **-view** flag when outputting the Words file for the benefit of previous releases.

As of release 4.4, the **-ssObject**, **-ssView**, and **-ssRange** flags modify the **-view** flag by indicating whether the *viewName* specifies a chart, named view, or named range respectively. Again, the data specified in the picSource element of the Picture bead, if any, will override this information. Release 4.3 and before will ignore these flags and take their best guess as to what *viewName* represents.

If the **-shrinkToFit** flag is present, and the linked file represents a non-picture Spreadsheet or Data object, the resulting table (representing the view of the Spreadsheet or Data object) can be shrunk horizontally to fit within the current margins. This flag has no effect for a picture inset.

If the **-lastSection** flag is present, and the linked file represents a non-picture Words document, the final section break of the main flow of the linked document is included in the field value. If the **-lastSection** flag is absent, the final section break is omitted.

If a simple document is being linked (into the containing or "master" document), this final section may control the entire page setup, column, and header/footer information for the linked document. Thus, this flag controls whether these attributes should be preserved when the small document is linked within the master document, or whether the material in the linked document should take on these attributes from the section in the master document into which it falls. For a Words document inset as a picture, this flag has no effect; the section attributes of the linked document do not interact in any way with those of the containing document.

The **-htmlAnchor** *"anchorURL"* flag and value pair, if present, indicates to the HTML Author as well as the Words to HTML export filter that the linked graphic is in fact a hyperlink to another HTML document. In the HTML Author, in View mode, double-clicking on the graphic will jump to the indicated document. This feature is available starting in release 4.2, and only for an Applixware that has the ability to fetch URLs. See the release 4.2 HTML documentation for more details.

The **-htmlImg** was used only in release 4.2; it is now obsolete. The value following this flag should now be specified as the *linkFile* as described above..

Error Handling

"Cannot evaluate circular link":

The named link appears more than one; for example, A links to B links to C links to A is a circular reference.

"No such Chart, View, or Range in SS doc"
: The indicated *viewName* does not appear in the linked Spreadsheet.

Interface Insert ➝ Object from File ➝ [*Application*]

Local Inset Field

Description When evaluated, the Local Inset field converts the locally-stored data object into a format appropriate for Words and includes it as the field value. When triggered, the Object field opens the "parent application" of the object, and allows the user to edit the object. Evaluation and triggering of the field are separate actions.

The field value of a protected, non-graphic object is not saved when the document is saved. This means that the field is reevaluated when the document is opened.

Format { **local_inset** "*objectName*" [**-docType** *type*] [**-appType** *"type"*]
[**-picture**] [**-cvt** *"filterName"*] [**-app** "filtername"]
[**-view** "*viewName*" [**-ssObject | -ssView | -ssRange**]]
[**-shrinkToFit**] [**-lastSection**]
[**-htmlAnchor** *"anchorURL"*]
[*evalFlags*] [*textStyle*] }

Arguments *objectName* is a string that uniquely identifies the object stored within the document. This string is automatically generated when the object is inset into the document. By default, *objectName* is a string in the series "object1", "object2", and so on.

The stored objects are accessed using the Advanced button within the Format → Object → Object Properties dialog box. A given object may be in any Applixware or foreign format. As of release 4.3, Applixware Words insets are supported within Words.

The **-docType, -appType, -picture, -cvt, -app, -view, -ssObject, -ssView, -ssRange, -shrinkToFit, -lastSection, and -htmlAnchor** flags work largely as described in "Link Field" above; simply substitute "data stored within the containing Words document" in place of "linked file" where applicable.

Interface Insert → New Object → [*Application*]

Macro Field

Description When evaluated, the Macro field displays the desired text string. When triggered using a double-click, the macro field executes an ELF macro. Evaluation and triggering are separate actions.

Format { **macro** *"macroName"* [**-pass** *"argStr"*] *textFlag* "*textStr*" |
-textButton "*textStr*" [**-buttonWidth** *width*]] |
-glossary *"glossaryName"* [*evalFlags*] [*textStyle*] }

Arguments The *macroName* is the name of the ELF macro to be executed when the field is triggered. The name should be enclosed in double quotes.

The **-pass** flag indicates that the following argStr is to be passed to the ELF macro. The bead number of the Field Start bead of the Macro field is also passed to the ELF macro (so the macro can determine, if necessary, which field activated it). Any number of flag and argument pairs can be specified, with the following effects:

Table 3-7 -pass Packaging

# flag/arg pairs specified	**Arguments received by the ELF macro**
None	1 argument: Field Start bead number
One	2 arguments: *argStr* as a single string, followed by Field Start bead number
More than one	2 arguments: Array of *argStr* strings, followed by Field Start bead number

The *textFlag* and *textStr* pair determine what is displayed in the field value (unless explicitly written to do so, the ELF macro has no effect on this) as follows:

-text "*textStr*"	*textStr* is considered to consist of normal (Latin) characters.
-dtext "*textStr*"	*textStr* is displayed in Dingbats typeface.
-stext "*textStr*"	*textStr* is displayed in Symbol typeface.

As an alternative to *textFlag*, the **-textButton** flag can be used to place the subsequent *textStr* string in a graphical representation of a button. The string is considered to consist of normal (Latin) characters. A double-click on the button will trigger the macro. By default, the button will be sized just large enough to contain the specified string. If a wider button is desired, use the **-buttonWidth** *width* flag and value pair to force a wider button; *width* is an integer representing the desired button width, in mills (there are 1000 mils to the inch). Note this value is ignored if it would make the button too narrow to contain the text.

Another alternative to *textFlag* is the **-glossary** *"glossaryName"* flag and value pair. If specified, the contents of glossary *glossaryName* are displayed as the field value.

Interface Tools → Special Fields → Macro Button

Macro String Field

Description When evaluated, the Macro String field executes an ELF macro and copies the returned string into the field value.

Format { **macro_string** *"macroName"* [**-pass** *"argStr"*] [**-doubleClick**]
[*evalFlags*] [*textStyle*] }

Arguments The *macroName* is the name of the ELF macro to be executed when the field is evaluated. The name must be enclosed in double quotes.

The **-pass** flag indicates that the following *argStr* is sent to the ELF macro. The bead number of the Field Start bead of the Macro String field is also passed to the ELF macro (so the macro can determine, if necessary, which field activated it). Any number of flag and argument pairs can be specified, with the following effects:

Table 3-8 -pass Packaging

# flag/arg pairs specified	**Arguments received by the ELF macro**
None	1 argument: Field Start bead number
One	2 arguments: *argStr* as a single string, followed by Field Start bead number
More than one	2 arguments: Array of *argStr* strings, followed by Field Start bead number

The called macro should return a string that is placed in the field value. If the macro throws an error, it is placed in the field value.

If the **-doubleClick** flag is present, double-clicking on the current field value reevaluates the field.

Interface Tools ➝ Special Fields ➝ Macro String

Make Index Field

Description When evaluated, the Make_Index field generates a document’s index by collecting the contents of all the Index fields in the document.

Format { **make_index** [*groupSeparators*] [*evalFlags*] [*textStyle*] }

Arguments The *groupSeparators* variable, if present, indicates how alphabetic sections of the index will be separated, as follows:

Table 3-9 groupSeparators

Flag	Meaning
-letterHeader	Separate the index entries with each letter of the alphabet (using the indexHeader$ style)
-blankHeader	Separate each alphabetic group of index entries with a blank line
-noHeader	Run all index entries together as a single group

Interface Tools ➝ Book Building ➝ Create Index

Make Table of Contents Field

Description When evaluated, the Table of Contents field generates a table of contents by collecting the text contents of certain paragraphs.

Format { **make_TOC -styles** "*styleList*" *beforePageNum*
[**-maxLevel** *levelNumber*] **[-noPageNums** "*styleList2*"]
[**-stripTabs**] [*evalFlags*] [*textStyle*] }

Arguments The **-styles** flag precedes the *styleList*, which is a comma-separated list of styles. The text content of each paragraph of those styles that appear in the main flow of the document is placed in the table of contents. The *styleList* must be enclosed in double quotes.

The *beforePageNum* indicates the text separating the paragraph text from the page number within the table of contents entries, as follows:

Table 3-10 beforePageNum Flags

Flag	Meaning
-dotTab	Insert a right-justified, dotted tab between the paragraph text and the page number This is the default if no flags are included
-plainTab	Insert a right-justified tab between the paragraph text and the page number
-noTab	Insert three blank spaces between the paragraph text and the page number

The **-maxLevel** flag precedes the integer *levelNumber* that indicates the maximum level number for entries in the table of contents. If specified, only the text contents of paragraphs (of the specified styles) with this number or lower will appear in the table of contents. Note level 1 is the minimum, and default, paragraph level. If absent, all paragraphs of the specified styles will appear in the table of contents. Note this option might be used, for example, for creating a table of contents for the higher levels of an outline document.

The **-noPageNums** flag precedes the *styleList2* parameter, which is a comma-separated list of styles. If present, this list should be a subset of the *styleList*. Table of contents entries for these styles are not followed by a page number. The *styleList2* should be enclosed in double quotes.

The optional **-stripTabs** flag, if present, removes the tab stops from the table of contents styles and replaces the tab characters in the table of contents paragraphs with two spaces each (however, if the user specifies a tab before the page number, this tab and tab stop will not be removed). This flag may be useful in cases where the page number does not align to the right margin, due to a mismatch between tabs and tab stops in the table of contents paragraphs.

Interface Tools → Book Building → Create Table of Contents

Merge Field

Description When evaluated during a merge operation, the Merge field copies the contents of a cell of the current row in the merge data file. Manual evaluation has no effect, however.

Format { **merge** *"fieldName"* [**-rowAdvance** *advanceNumber*]
[-rowOffset *offsetNumber*] [**-usePara**] [*textStyle*] }

Arguments The *fieldName* identifies a column of the merge data table. The column is identified by matching *fieldName* with the text in the top row of the table. The cell in the current row of the data table that falls into this column is copied into the field value of the Merge field. The *fieldName* must be enclosed in double quotes.

The **-rowAdvance** flag forces the current row of the merge data table to be advanced by *advanceNumber* rows, where *advanceNumber* is a positive or negative integer. (The Print Merge operation automatically advances the current row by one on each print cycle.) This action occurs before the cell of the data table is identified and copied.

The **-rowOffset** flag temporarily advances the current row of the merge data table by *offsetNumber* rows, where *offsetNumber* is a positive or negative integer. This action occurs before the field is evaluated, and is reversed immediately afterwards. This option allows, for example, reference to the previous or next record while the current record is being processed.

Both the **-rowAdvance** and **-rowOffset** flags can be specified for a single field. The sum of *advanceNumber* and *offsetNumber* will alter the current row before field evaluation; only *offsetNumber* will be undone after the evaluation.

If the **-usePara** flag is absent (the default), the contents of the appropriate merge data table cell, excluding the final paragraph marker, are copied into the field value.

If the **-usePara** flag is present, the entire cell including the final paragraph marker are copied. However, if the cell only contains an empty paragraph, nothing is copied. This feature can be used to avoid blank lines due to empty data table cells (in a document that uses successive lines of data from the table). If a column in the data table can contain an empty cell, instead of following the merge field with a paragraph marker (that will force a blank line), omit the paragraph marker and include the **-usePara** flag for this field.

Error Handling

The following messages may be inserted into the field value:

Missing a numeric argument	The **-rowAdvance** flag was not immediately followed by the integer *advanceNumber*, or the **-rowOffset** flag was not immediately followed by the integer *offsetNumber*
Record not found	The merge data table has run out of records, or the **-rowAdvance** or **-rowOffset** flags have specified a record beyond the range of the table
Specified field name not in data table	The *fieldName* was not found in the first row of the merge data table
Short row in data table	The current row of the merge data table is too short and does not contain a cell in the desired column

Interface

Insert → Merge Field

Page Count Field

Description When evaluated, the Page Count field generates the number of pages in the document. As the document is edited, Page Count fields are reevaluated.

Format { **page_count** }

Interface Insert → Page Numbering → Page Count

Page Number Field

Description When evaluated, the Page Number field generates the formatted page number. While value is displayed and printed, it is not inserted into the field value. In a header or footer, a Page Number field can appear on more than one page and it displays a different value on each page. As the document is edited, Page Number fields are reevaluated.

The page number format is defined by the current section’s definition.

Format { **page_number** }

Arguments None.

Interface Insert → Page Numbering → Page Number

Plain Field

Description The Plain field provides a method of enclosing arbitrary material within a field. No action is taken when the field is evaluated, other than servicing *textStyle*, if any. This field was implemented in release 4.3. There is currently no user interface to create this type of field; it must be created via an ELF macro.

Format { **plain** [*textStyle*] }

Arguments None.

The Plain field is primarily intended as a simple way to apply a text style to some material, with no other side effects.

Interface None

Printer Code Field

Description The Printer Code field provides a method of passing special codes to the printer during a print operation. The field value is always empty, and no action is taken when the field is evaluated. This field was implemented in release 4.1. There is currently no user interface to create this type of field; it must be created via an ELF macro.

Format { **printer_code** [*printerType*] [**-text** *"text"* **|** **-ascii** *code*]... }

Arguments The *printerType* variable indicates the type of printer for which the code is intended, as follows:

Table 3-11 printerType

Flag	Meaning
-pcl	A PCL printer (the default, if omitted)
-postscript	A Postscript printer
-gdi	A GDI (Windows) printer

A series of printer codes can follow the *printerType*, as described below.

Table 3-12 Printer codes

Flag	Meaning of value
-text	"*text*" = a quoted string of Latin characters, representing printer codes.

Table 3-12 Printer codes

-ascii	*code* = an integer representing a printer code

Interface None

Series Field

Description When evaluated, the Series field generates a string representing the next item of an ordered series, at the appropriate level.

Format { **series** "*seriesName*" [*valueFlag*] [**-noDisplay**]
[*levelNumber*] [*evalFlags*] [*textStyle*] }

Arguments The *seriesName* is the name of the series as it is displayed in the Edit Numbered Series dialog box. The name must be enclosed in double quotes.

The *valueFlag* indicates the number used to generate the formatted string. If omitted (the default), the value is one more than the previous instance of this series. Otherwise, *valueFlag* can be one of the following:

Table 3-13 valueFlags

Flag	Meaning
-noIncrement	Uses the same value as as the previous instance of this series This could be used, for example, in a document consisting of numbered subsections where the subsection number needs to be displayed within the contents of the subsection (for example, for a figure number)
-set *number*	Forces the value to the specified integer *number* For example, if the series "Outline" used the A, B, C numbering style, { series "Outline" -set 4 } would generate ’D’

The **-noDisplay** flag, if present, causes the generated field value to be empty. However, the number is "still in the document" as far as subsequent series fields are concerned. For example, this feature could be used to reset the numbering of lists at given points in the document, as follows:

Table 3-14 View Field Method Modes

View Field Method Mode	Normal Display
Engines:{ series "List" -set 0 -nodisplay}	Engines:
{ series "List" }Thomas	1. Thomas
{ series "List" }James	2. James
Coaches:{ series "List" -set 0 -nodisplay}	Coaches:
{ series "List" }Anna	1. Anna
{ series "List" }Claribel	2. Claribel

If the optional **-level** flag is omitted, the paragraph's level determines the level within the series definition to use. If this flag is present, the integer *levelNumber* specifies a level to use in place of the paragraph's level.

For example, if you have a single-level series appear in paragraphs that had levels larger than one (to cause them to indent), you would want the series field always to be evaluated at level one. For example:

```
{ series "List" -level 1 }
```

Interface Tools → Insert Numbered Field

Soft Hyphen Field

Description The Soft Hyphen field marks a preferred hyphenation point. Its field value is always empty. No action is taken when the field is evaluated.

If a word containing a Soft Hyphen field needs to be hyphenated, it will be hyphenated at the field location, if possible.

Format { **-** }

Arguments None.

Interface CTRL - (hyphen) Key combination

Time Field

Description When evaluated, the Time field generates a string representing the current time in the desired time format.

Format { **time -format** *"formatStr"* **|** *formatCode* [*evalFlags*] [*textStyle*] }

Arguments The **-format** *"formatStr"* flag and value pair specify the exact format of the time to be displayed in the field value. The string consists of a mixture of hour, minute, second, and am / pm codes with miscellaneous text. For example, the string "hh:mi" would yield (for example) "09:34"; note the preservation of the colon. A list of the hour, minute, second, and am / pm codes can be found in the file datetime.sp under the Applixware release directory, or in help under the description of the DATE_FORMAT@ macro. Note the use of *formatStr* is new for release 4.1. For earlier releases, the *formatCode* should be used as described below.

The *formatCode* is an integer representing a time format, as shown below. Release 4.0 and earlier will ignore the *formatStr* and use the *formatCode*. Note that it is acceptable to specify both a *formatCode* and a *formatStr*; earlier releases will use the former and later releases the latter.

Table 3-15 Time Formats

formatCode	**Matching *formatStr***	**Display**
0	HH:mi:ss	14:58:20

Table 3-15 Time Formats

1	hh:mi pm	2:58 pm
2	HH:mi	14:58
3	12hh:mipm	2:58pm
4	HH.mi	14.58
5	HHmi	1458

Interface Insert → Date & Time

Variable String Field

Description When evaluated, the Variable String field generates a string representing the contents of the desired document variable.

The Variable String field is reevaluated whenever the referenced document variable changes value.

Format { **var_string** "*varName*" [*evalFlags*] [*textStyle*] }

Arguments *varName* is the name of the document variable (as displayed in the Tools → Special Fields → Set Document Variables dialog box). It must be enclosed in double quotes. The way in which the variable is represented depends upon its *type* and is as follows:

Table 3-16 Variable Types

Variable type	Representation
Number or String	The variable’s value
One-dimensional array	A separate paragraph is generated for each array item
Two-dimensional array	A table is generated with a row for each item in the major array and a cell for each item in the secondary arrays
Anything else	The following error is displayed: "ERROR: Variable is of incorrect type"

An empty string is generated if varName does not exist.

Interface Tools → Special Fields → Variable String

Index