# TagWrite 3.0
for Windows

Template Designer’s Guide
Command Reference

ZANDAR Corporation

TagWrite 3.0 For Windows
Template Designer's Guide
Command Reference

ZANDAR Corporation, creator of TagWrite, is a pioneer in the development of SGML tools and other tagging applications. Our products specialize not only in tagging but also in untagging SGML and reformatting it for display and other uses.

ZANDAR offers additional software tools and services to expand your SGML application. An expanded tool set is available to automatically and bi-directionally transform Rich Text Format (RTF) tables and fully formatted SGML CALS tables including complete table declaration, support for spanning cells, grid and box, and all character formatting. Extensions can be developed to fully transform tables between SGML and virtually any other markup.

For SGML and other tagging/untagging development services and software tools, contact ZANDAR Corporation through our telephone listing in Burlington, Vermont, USA. For information on development services and software call our sales-only line at 800-639-5818. (Sorry, no support calls can be accepted. For support, contact Corel Technical Support at 1-800-818-1848).

ZANDAR Corporation
P.O. Box 467
Jericho, Vermont
05465

Table of Contents

OVERVIEW

Command Reference Guide

The Command Reference is divided into two parts and provides complete definitions for every command used in the TagWrite Template Language.

Part 1 briefly describes each of these topics:

- Tokens
- Supertokens
- Frequency Indicators
- Logical Operators
- Reserved Characters
- Parenthetical Expressions
- Counters

Part II of the Command Reference explains in detail each command of the Template Language. Example Template rules are provided where necessary to enhance the descriptions.

Conceptual and specific instructions on how to create Templates are contained in the Template Designer’s Guide. If you are new to TagWrite Template design, you should read Chapters 1 through 7 in the Template Designer’s Guide. Tutorials are included in chapters 4 and 7. Do not rely on this Command Reference as an introduction to TagWrite. It will be most useful to you after you understand how the program functions. Refer also to Chapter 8, “Special Techniques For RTF (Rich Text Format) and WordPerfect” in the Template Designer’s Guide. Information about special handling for Microsoft Word (RTF) and WordPerfect files.

PART 1

Overview

Part 1 briefly describes each of these topics:

- Tokens
- Supertokens
- Frequency Indicators
- Logical Operators
- Reserved Characters
- Parenthetical Expressions
- Counters

Tokens

TagWrite Tokens represent specific individual units of information contained in the Microsoft RTF, WordPerfect, or ASCII file. There are three categories of Token information, Codes Styles and Alphanumeric characters. They are described below.

Codes

- "Codes" are word processor format commands.

  A "code" is a single-byte, multi-byte or binary character that is understood only by a specific word processor. TagWrite Tokens represent these word processing codes allowing TagWrite to read and write the word processing file format.

  Some codes, like {center} or {tab}, are used with WordPerfect and Microsoft Word RTF. Others, like [PLAIN] or [RESET] are used only with RTF. Some, like [CR] and [LF] are used only with straight ASCII files.

Styles

- TagWrite considers a word processor Style to be a Token if your version of TagWrite supports the Style Module.

Alphanumeric characters

- The {space} character and all ASCII alphanumeric characters that are entered in the TagWrite Template Editor from the standard keyboard are tokens.

A TagWrite Token is one kind of tool used in the Top line of a Template rule to search for a pattern of information in a text file. The pattern may contain one or more word processor format commands, a Style if your version of TagWrite supports styles, one or more ASCII characters, or a combination of these components.

A Token is used in the Bottom line of a rule to write into the output file individual ASCII characters, word processor format commands (including styles if supported by your version of TagWrite), or a combination of these components.

By selecting "Codes" from the Template Editor Menu Bar, a Codes column appears on the right hand side of the Template Editor Screen.

TagWrite is delivered with a robust list of Tokens. New Tokens representing other word processing format information can be added to the list using the TagWrite Codes Editor explained in Appendix A. Since you can make new Tokens, there is no definitive list.

Token Samples

Following is a sample of Tokens. The first part lists some Tokens that are used with both WordPerfect and Microsoft Word RTF. The second part lists some Tokens used only with RTF. The third part lists some straight ASCII tokens that are not easily entered from the keyboard.

Tokens Used With WordPerfect 5.x and MS RTF

Although the TagWrite Template editor displays the same TagWrite Token for many functions in WordPerfect and RTF, the internal byte and bit information handled by TagWrite is completely different for WordPerfect and RTF. TagWrite handles these differences automatically.

[BLD] — a {bold on}

[BLDOFF] — a {bold off}

[CENTER] — a {center}

[HNL] — a {hard new line}. A {hard new line} is created in a wordprocessor every time the Enter key is pressed. A {hard new line} is used in a word processed file most commonly to indicate the end of a paragraph of text.

[HPg] — a {hard page break}. Used to force a page break in the text before the normal end-of-page.

[H**^**] — a {hard space} (also known as a “non-breaking” space). This is a special character in each word processor, and it is not the same as an ordinary space. The {hard space} is commonly used to create a space between text characters that will not break the line at the right margin and will not collapse when the line is justified.

[INDT] — an {indent}. A paragraph formatting code that indents the first and all subsequent lines of text. Used to create an indented list or “step”. TagWrite is not concerned with the actual measurement of space of each indent. Actual spacing will vary depending on the settings established in your word processor. [INDT] is not the same as [TAB].

[ITAL] — an {italics on}

[ITALOFF] — an {italics off}

[SmCAP] — a {small caps on}

[SmCAP/] — a {small caps off}

[STRKTHRU] — a {strikethrough on}

[STRKOFF] — a {strikethrough off}

[TAB] — a {tab}. Tab is recorded differently in WordPerfect, Microsoft Word RTF, and ASCII. It is, therefore, not a standard character, and TagWrite must use a [TAB] Token in the Codes column. TagWrite is concerned with the presence of a tab and the number of tabs present, but it is not concerned with the actual spacing of a {tab} which depends on the tab stops set in your word processor. {tab} is not the same as a Microsoft Word {first line indent}.

[ULIN] — an {underline on}

[ULINOFF] — an {underline off}

Other Tokens are described in the following part of this chapter.

Tokens Used With RTF Only

[PLAIN] — used in the Bottom line before writing back styles.

[RESET] — used in the Bottom line before writing back styles.

[1stIND] — a {first-line indent}. Not the same as a {tab} code. Used only with Microsoft Word to indent the first line of text in a paragraph. Generally used in the Bottom line to write back a First Line Indent. Use [STab] Supertoken in the Top line.

[HIDE] — Begin “hidden text”

[HIDEOFF] — End of “hidden text”

[FLDRSLT] — Start of Field Result Group

[/GRP] — End of all named groups

Examples of ASCII Tokens

[CR][LF] — The {carriage return/linefeed} combination is used in a straight ASCII file to indicate the end of a line or paragraph of text. [CR] and [LF] can be used separately.

[**^**] — a {space}. This Token is used to indicate the normal {space} character which is keyentered with the Spacebar.

[BELL] — a {bell}. The ASCII character (hex 07) for a "beep" or {bell} is commonly used as a typesetting delimiter by the U.S. Government Printing Office. [BELL] illustrates that you can make a TagWrite Token for any ASCII character.

Supertokens

Supertokens are a TagWrite proprietary invention. They are designed to be started by specific characters or commands and ended by others.

Once started, the Supertoken absorbs the start-up character and all subsequent characters and word processing format commands until it finds the assigned character or format command that signals a stop. The character or format command that stops a Supertoken is not included in the string that is absorbed by the Supertoken.

A Supertoken is used in the Bottom line of a rule to write back the group of characters or format commands to the output document.

The following is a list of the Supertokens delivered with TagWrite Version 3.0:

[TeXt+#] — The text plus number Supertoken is a general, workhorse Supertoken used to capture letters, punctuation and numbers, and certain word processing character format commands in a string of any length.

[FILTxT] — Another major workhorse. Used for intraparagraph tagging to "filter text" from character formats like {bold}, {italic}, {underline}, and used in table tagging.

[GRPTxT] — Used in RTF applications to pick up text within known groups.

[NO] — The number Supertoken captures one number. The default settings of TagWrite define a number as any combination of digits, hyphens, and commas, not separated by spaces, periods or alphabetic characters. One occurrence of the [NO] Supertoken will be fulfilled by any of the following:

12
123
12,345
12-345

[KEEP] — The [KEEP] Supertoken is started by everything in the input document except {hard new line} and is ended by a {hard new line}. [KEEP] generally is used with the ultimate Escape Rule in the main or body Element. Also often used with untagging.

{<TeXt>] — The angle bracket text Supertoken is started by any alphanumeric or punctuation characters except for the left and right angle brackets. It is stopped by a left or right angle bracket or a {hard new line}. Used most often to capture letters, punctuation and numbers or typesetting codes delimited by angle brackets.

[CF] — The character format Supertoken captures the word processing character format commands for {bold on}, {underline on}, {italics on}, {small caps on}, and {strikethrough on}. Most often used in the Top line rule with an asterisk (*) in order to capture any format commands in any order and any frequency.

[PF] — The paragraph format Supertoken captures paragraph formats like {indent}, {center}, {tab}, {flush right} and others. Most often used in the Top line rule with an asterisk (*) in order to capture any paragraph format commands in any order and any frequency.

[CF + PF] — Combined power of [PF] and [CF] will capture be started by any paragraph or character format and capture all formatting information until stopped by alphanumeric characters. Most often used with Style tokens.

[[TxT/all] — The text all Supertoken (short for "text stopped by all other characters") captures one single upper or lower case letter and is used to remove alphabetic ordering (alphabetic numbering) from paragraphs and lists so that the alphabetic ordering system can be replaced by the correct tagname. It is started by any one alphabetic character and is stopped by all alphanumeric and punctuation characters, as well as {space}, {tab} and all other character and paragraph formatting commands.

TagWrite Special Codes

Two codes are not true Tokens or Supertokens. These codes are built into TagWrite and you cannot change their settings.

START — START is a marker that begins all Templates. It is used only (and always) in the Current Element of Rule 0. It is hard coded into the Current Element of Rule 0 of the Template Editor and cannot be edited.

[ALL] — [ALL] is used only on the Bottom line of a rule. It returns all letters, punctuation, numbers, Tokens, and Supertokens captured by the Top rule.

Reserved Characters In the Template Editor

Reserved characters exist only in the Template Editor. So far as the typist working in the word processor is concerned, TagWrite has **no** reserved characters. The end-user of TagWrite (“the typist”) can use any character or format command to create the original document in the word processor.

Characters are “reserved” in the Template Editor because they can have two uses: One use is to give special meaning within template rules. The second use is as an ordinary text character.

Top Line Reserved Characters

These characters:

’ * + ? | ( ) [ ] ^

are special TagWrite Template Language symbols, and therefore are reserved characters when used in the Top line of a rule in the Template Editor.

Bottom Line Reserved Characters

These characters:

’ [ ] ^

are reserved characters in Bottom line rules in the Template Editor.

▶ ▶ To use a reserved character as text:

- To use a reserved character as text in the Top or Bottom line of a rule, you must enclose the reserved character in single quotes with no spaces between the single quote and the character. For example, by enclosing the asterisk between single quotes, as shown:

  ’ * ’

  TagWrite interprets this asterisk as actual text rather than as a frequency indicator.

  The only exception is the single quote itself. To use a single quote as text in the Top or Bottom line, simply use two single quotes in a row, as shown below:

  ’ ’

Frequency Indicators

Frequency Indicators are tools that define the number of times a Token, Supertoken, or parenthetical expression can occur in the Top line of a rule and still fulfill the rule. There are three Frequency Indicators with different functions. They are the asterisk, the plus sign, and the question mark.

***** — The asterisk indicator means zero or more occurrences. The asterisk indicates that a Token, Supertoken, character, **or** parenthetical expression may appear next in the text stream, **or** it may appear more than once without limit, or it may not appear at all.

+ — The plus sign indicator means one or more occurrences. It indicates that the Token, character, or parenthetical expression **must** appear next in the text stream at least once, but it may appear more than once without limit.

? — The question mark indicator means zero or one occurrence. The question mark indicates that the Token, character, or parenthetical expression may not appear or it may appear one time.

Use Only One Frequency Indicator At A Time

Use of the Frequency Indicator is optional depending upon the requirements of your Top rule. You can place any one of the three Frequency Indicators directly after a Token, Supertoken, character, or parenthetical expression in the Top line of a Template rule.

You cannot place more than one Frequency Indicator after the same Token in the same rule. For example, you cannot use a **+** and follow it immediately with an * asterisk.

Frequency Indicators Are Plain Text In Bottom Line

Frequency Indicators have no special function in the Bottom rule. In the Bottom line, these symbols are treated as plain text and will be written out to the tagged file literally.

Frequency Indicator and Supertoken

Caution: If you use an asterisk with a Supertoken in the Top line of a rule, you cannot place that Supertoken in the Bottom line of the rule. The asterisk allows that the Supertoken in the Top may not be fulfilled. TagWrite does not allow an empty (null) Supertoken to be placed in a Bottom rule, thus, if the Top is not fulfilled, and the Supertoken is written in the Bottom line, TagWrite will not function correctly.

Three Supertokens in particular commonly use a Frequency Indicator:

[CF]

[PF]

[CF + PF]

[STab]

For example, [CF] is used with an asterisk [CF]* to capture any and all character formatting that may be embedded in the text. See Part 2 of this guide for complete descriptions of the other Supertokens.

Logical Operators

AND

The AND expression is implied when you place Token codes, text characters, Counters, symbols, etc. next to each other in a rule. For example,

[CENTER][HNL]

in a Top line rule indicates that a {center} formatting code AND a {hard new line} are both required if the rule is to be satisfied.

Caution: Commas should never be used to delimit Tokens as part of a logical AND expression. Commas in the Top rule will be treated as text. Use a comma in a rule only if you want that rule specifically to search for a comma.

OR

The “pipe” or vertical bar character **|** represents the OR expression.

The OR **|** means that one Token or another *in any order* must appear in the text if the Top rule is to be fulfilled.

The **|** sign is entered without any space between the two tokens or expressions it connects. For example:

([CENTER] **|** [ITAL])

in a Top line rule indicates that either a {center} formatting code OR an {italics on} code must appear in the text for the rule to be satisfied.

Enclose OR Expressions in Parentheses

Every OR expression must be enclosed in parentheses. The parentheses delimit the membership in the OR statement.

Caution: Parentheses are required with an OR statement.

There is no limit (except the limit of 98 characters per line in a rule) to the number of OR expressions which may be built into each rule.

Frequency Indicators with OR Expressions. You may use frequency indicators with OR structures in parentheses.

The frequency indicator may be attached to any item within a parenthetical statement. For example, the following expression,

([INDT]* **|** [CENTER])

indicates zero or more occurrences of {indent}, OR one {center}.

The frequency indicator may be attached to the parenthetical statement itself. For example, the following expression,

([INDT] | [CENTER])*

indicates zero or more occurrences of {indent} OR {center}.

Parenthetical Expressions

Parentheses are used to group three kinds of expressions: OR statements, ASCII Text Strings, and Combined ASCII Text and Token/Supertoken strings.

Every OR Expression in Parentheses

Every OR expression must always be placed in parentheses. If the parentheses are missing, the Top rule will not function properly.

ASCII Text Strings

Any ASCII text string of alphanumeric and punctuation characters meant to be a coherent group should be placed within parens. For example,

(Address:)

would indicate a search for the word "Address" and the colon **:** character.

Combination Text and Token/Supertoken Strings

Any ASCII text string, in combination with tokens or Supertokens, meant to be a coherent group should be placed within parens. For example,

(Date:[TAB])

would indicate a search for the word "Date", the colon ":" character, and one {tab}.

Frequency Indicators With Parenthetical Expressions

Enclosing a string of text within parentheses allows you to treat the string as a group rather than as individual characters. Grouping adds clarity to your rules and allows the use of Frequency Indicators which refer to the group. Frequency Indicators placed after parenthetical expressions indicate the number of times the contents of the expression can occur.

The & Ampersand Designates a Style Token

The & Ampersand is used to designate that a Token is a style Token. For example:

[&STYLE]

The & Ampersand is not a reserved character in that it can be used freely in rules; however, the & ampersand has a special meaning when used to start a Token name.

The manual titled “Preparing a Styles Application” supplied with the TagWrite Styles Module describes the use of the & Ampersand.

Note: Do not build a Token starting with an & Ampersand unless the TagWrite Styles Module is installed. You will know if the Styles Module is installed because the Codes Editor will be titled “Edit Codes and Styles”. You **cannot build a valid Style Token without the TagWrite Style module.**

Counters

Counters allow you to insert incremented or constant numeric or alphabetic values into the output document. Counters are used only in the Bottom line of a Rule. Typically Counters are used:

- In the creation of tagged table data where each column and/or row must be indicated with unique tagnames.
- To untag a file and restore sequential numbering or alphabetization of lists.
- To increment paragraph number codes as required for some SGML applications.

Counters are explained in detail in the Template Designer’s Guide, Chapter 10, “Counters.”

PART 2

Overview

Part 2 of this Command Reference Guide explains in detail each command of the Template Language, including each TagWrite Token, Supertoken and frequency indicator. Example Template rules are provided where necessary to enhance the descriptions.

Conventions

Each TagWrite function is discussed systematically. Following is a list of discussion categories and conventions followed throughout the remainder of this chapter:

Definition – Defines the symbol, Token, Supertoken, or Counter name and explains its use in Template rules.

Supertoken Code No. – Gives the Hexadecimal Supertoken code number as it was delivered in TagWrite and as it appears in CODES.DAT.

Started by – Lists the characters or format commands which trigger (start) the Supertoken.

Stopped by – Lists the characters or format commands which terminate (stop) the Supertoken.

Starts – Lists the Supertokens which a Token starts.

Stops – Lists the Supertokens which a Token stops.

Examples – Includes typical examples of how to use the symbol, Token, Supertoken, or Counter.

Notes – Provides operational caveats and describes special circumstances that need to be considered when writing rules.

[ ] – Throughout the following text, square brackets **[ ]** are used to denote TagWrite Token names

{ } – "Curly brackets" { } delimit word processor "codes" (word processor format commands).

Distinguishing Tokens from a Wordprocessor Code

In most cases TagWrite has a Token that corresponds to the word processor "code". It will not be uncommon to see, for example, the TagWrite Token name like [BLD] in the same paragraph as the format code {bold on}. It should be understood that **[BLD]** is the name of the TagWrite Token, while {**bold on**} refers to the actual digital code in your word processed file that turns on bold formatting.

Tagnames

We have used one convention for tagnames in the sample Templates found throughout the following text. Tagnames are delimited by the left and right angle brackets, such as:

    <tagname>

We have followed this convention since the left and right angle brackets are commonly used delimiters for SGML and many kinds of typesetting software.

Note: If your typesetting software uses a different naming convention, simply substitute the appropriate naming convention when you build your Templates. You are **not** bound to the use of any one style for representing tagnames.

& Ampersand [&style]

The **&** Ampersand is used to designate that a Token is a Style Token.

Style Tokens are used with Template applications that capture or write WordPerfect or Microsoft Word (RTF) styles.

The creation of Style Tokens requires the TagWrite Styles Module. See the manual titled “Preparing A Styles Application” supplied with the Styles Module for a complete description of Style Tokens, their creation, and their use.

* (asterisk)

Definition:

An asterisk symbol in the Top line of a rule captures zero or more occurrences of the preceding Token, Supertoken, character, or parenthetical expression. The asterisk is a powerful tool because it allows that something may occur zero times which is another way of saying that, with an asterisk, you can account for the possibility that the item may not occur. The potential absence of something may be as important as its presence.

An asterisk symbol in the Bottom line of a rule is treated as plain text.

Examples:

Example 1

Use the asterisk symbol to find format commands such as {bold} which may or may not be present in the paragraph. With the asterisk present, the segment of the rule is fulfilled if the Token is present or not present.

The following rule is fulfilled if zero or more {bold} commands are entered before an {indent} at the beginning of a paragraph.

Current Element			Next Element
0	body		body
		[BLD] * [INDT][TeXt + #][HNL]+	
	10	<list1>[TeXt + #][HNL]	

In the above example, if a {bold} word processing code is not found, TagWrite will try to fulfill the remainder of the rule as if the [BLD]* were never present in the Top rule. If a {bold} code is found, then the asterisk will tell TagWrite to absorb the {bold} code. The {bold} code will be captured and will not become a part of [TeXt + #]. The {Bold} code, therefore, will not appear in the Bottom line output because we picked it off in the Top, and, in this example, decided not to write it back in the Bottom.

Example 2

The following rule is fulfilled whether or not the typist entered a {bold} code before the {tab} code, after it, or both. The objective of this example is to ensure that {bold} is removed from the tagged file regardless of how many {bold} codes were entered and regardless of where the {bold} code might appear in the original word processed file.

	Current Element		Next Element
	body		**body**
		[BLD] * [TAB][BLD] * [TeXt + #][HNL]+	
10		**<list1>[TAB][TeXt + #][HNL]**	

Example 3

Another way to use the asterisk is to find periods between section numbers in the numbering system of an outlined document like a technical manual.

In the example below, which we used in “Advanced Template Techniques”, the number must be in the input text, but the period is optional. Also, the [CF + PF] cleanup Supertoken and the [PF] Supertoken written with an asterisk allow that paragraph and character formatting may optionally occur.

1	**body**		**paranext**
		[CF+PF]* [NO](.)* [NO](.)* [NO](.)* ([^] | [PF])* [TeXt + #] [HNL]+	
20		**<para2>**	
2	**body**		**paranext**
		[CF+PF]* [NO](.)* [NO](.)* ([^] | [PF])* [TeXt + #] [HNL]+	
19		**<para1>**	
3	**body**		**paranext**
		[CF+PF] [NO](.)* ([^] | [PF])	
18		**<para0>**	

Notes:

No Supertoken In Bottom Rule If Asterisk is in Top

Caution: If you use an asterisk in the Top line of a rule with a Supertoken (i.e. [STab]*), then you can **not** write the Supertoken back in the Bottom rule. The asterisk stands for “zero or more occurrences.” This means that the Supertoken stands a chance of not being fulfilled in the Top, and TagWrite can not write back an empty Supertoken in the Bottom line.

Regular Tokens OK In Bottom

There is **not** a similar problem associated with the use of an asterisk after a regular TagWrite Token (i.e. [HNL]* or [BLD]*). You may always choose to write any Token in the Bottom line of the rule regardless of the content of the Top line.

+ (plus symbol)

Definition:

A plus symbol in the Top line of a rule captures one or more occurrences of the preceding Token, Supertoken, character, or parenthetical expression.

A plus symbol in the Bottom line of a rule is treated as plain text.

Example:

Most typists enter two {hard new line} codes at the end of each paragraph. In fact, your key entry conventions may dictate this. However, the typist may inadvertently enter one, two or even three {hard new line}s. By using the plus symbol, you can fulfill the rule and filter out the unwanted {hard new line}s.

The following rule tags the input paragraph no matter how many extra {hard new line}s are contained at the end of the paragraph in the word processed file. We collect all of the {hard new line}s in the Top rule, and write back only one {hard new line} in the Bottom. (If you wanted to write back two {hard new line}s, you would simply add an additional [HNL] Token to the Bottom.)

Current Element		Next Element
body		body
	[TAB][TeXt + #][HNL]+	
10	**<para1>[TeXt + #][HNL]**	

? (question mark)

Definition:

A question mark symbol in the Top line of a rule captures zero or one occurrence of the preceding parenthetical expression, Token, or Super-token from the input document.

A question mark symbol in the Bottom line of a rule is treated as plain text in the output document.

Notes:

Use the question mark to find format commands which, according to key entry conventions, may not occur at all, or should occur no more than one time in a paragraph.

( ) (parentheses)

Definition:

Parentheses are used to group text characters, OR (I) constructs, Tokens, and Supertokens included in the Top line of a rule. One character or a string included in a parenthetical expression is considered to be a single entity. The contents of a parenthetical statement may be modified by a frequency indicator.

You may group parenthetical statements within parenthetical statements using double or triple sets of parentheses.

All OR (I) expressions should be enclosed in parens for clarity of purpose. OR statements can be modified by frequency indicators.

Parentheses and parenthetical expressions included in the Bottom line of a rule are taken as plain text.

Examples:

Example 1

This example places [BLD] and [ULIN] Tokens together in an OR statement. The parenthetical expression portion of the rule is fulfilled if either a {bold} or an {underline} format command is encountered before the {tab}. Notice that each "or" construct is and must be contained in parentheses. If the parenthetical grouping is not correct, the logical operation of "or" is unclear or bizarre.

```
Current                                                  Next
Element                                                  Element
--------------------------------------------------------------------
     body                                                  body
        ([BLD] | [ULIN])[TAB][TeXt + #][HNL]
   10   <head1>[TeXt + #][HNL]
```

Example 2

The following example captures the word "chapter" at the beginning of a paragraph. In this limited example, we require that the word "chapter" must be typed in lower case, exactly as it appears in the parenthetical expression, or the rule will not be fulfilled. In practice, we are not likely to want to write a rule in this way, but it is a useful example

```
Current                                                  Next
Element                                                  Element
--------------------------------------------------------------------
0    body                                                  body
        (chapter)[TeXt + #][HNL]
   00   <chaptitle>[TeXt + #][HNL]
```

Example 3

The following example captures the word “chapter” at the beginning of a paragraph for all upper case letters, all lower case letters, or mixed upper and lower case letters. This is a robust way of writing this rule, and it illustrates the use of double parentheses. You may enclose parenthetical expressions within parenthetical expressions. In practice there is no limit to the level of parenthetical nesting. However, common sense dictates keeping the statement as simple as possible.

Current Element			Next Element
0	body		body
		((C)|(c))((hapter)|(HAPTER))[TeXt + #][HNL]	
	10	<haptitle>[TeXt + #][HNL]	

Notes:

If your parens are not correctly matched, you will receive an error message when using Error Check. This is a critical error. If you fail to correct the Error Check message and your parens do not match, TagWrite may fail to run, and you may not receive a specific error message at runtime.

| (vertical bar)

Definition:

A vertical bar represents the Boolean or construct in the Top line of a rule. It allows for the condition that one or another component of the OR expression be fulfilled. An OR expression may include one or more OR (|) segments, for example

([TAB] | [INDT] | [CENTER])

A vertical bar in the Bottom line of a rule is treated as plain text and will be written literally to the output document.

Examples:

Example 1:

The following example shows you how to write a rule to capture a paragraph which begins with the word "chapter." The rule captures two variations of the word form and is fulfilled if the word has initial caps, or is all uppercase.

Current Element		Next Element
0	body ((Chapter) | (CHAPTER))[TeXt + #][HNL]+	body
00	<CHAPTER^TITLE>[TeXt + #][HNL]	

Example 2:

The following example presents an efficient way to generalize the previous rule.

Current Element		Next Element
0	body ((C) | (c))((hapter) | (HAPTER))[TeXt + #][HNL]	body
10	<chaptitle>[TeXt + #][HNL]	

Notes:

In the examples, notice that each OR construct is contained in parens.

There is no limit to the number of or constructs in each rule.

Each component of the OR expression may be modified by a frequency indicator, and the whole OR expression may be modified by a frequency indicator. The following examples illustrate the flexibility of the combined use of parens, the OR operator (**|**), and the asterisk (*****). You can invent many variations to meet specific needs.

([TAB] | [INDT])* means that any number of {tab} OR {indent} codes may occur in any order and satisfy the rule, but if neither of the Tokens occur, then the rule will also be satisfied.

([TAB]* | [INDT]) means that a {tab} code may or may not occur, OR that it may occur many times. The expression also specifies that an {indent} code can occur once and still satisfy the rule. However, since there is no asterisk modifying the entire group, the occurrence of either one or more {tab}s or one {indent] or both is required. That is, at least one {tab} or one {indent} must occur if the Rule is to be satisfied.

[-] (hyphen)

Definition:

The hyphen Token is used to represent the word processor {hyphen} which is entered into a word processed file by pressing the hyphen key.

In the Template, use the hyphen Token [-] to capture the hyphen that has been keyentered in the word processed document.

Note On Different Hyphens

WordPerfect and Microsoft Word each produce several different kinds of hyphens. The WordPerfect and RTF codes for handling these hyphens are mapped differently.

The TagWrite Template Editor has tried to simplify handling hyphens by using one hyphen Token [-].

Caution: In the Template Rule, unless you fully understand the WordPerfect or RTF native file formats, do **not** attempt to represent the word processor's hyphen by pressing the hyphen key on the standard keyboard.

At the end of this section, there is a Note describing some alternative methods to handle the hyphen.

A [**-**] Token in the Top line of a rule captures the occurrence of a {hyphen} code in the input document.

A [-] Token in the Bottom line of a rule inserts the {hyphen} code into the output document.

Starts:

[TeXt + #], [FILTxT], [GRPTxT], [KEEP], [<TeXt>]

Stops:

[STab], [CF], [PF], [CR + LF], [TxT/all]

Notes:

Each word processor also has a hard, non-breaking hyphen, soft hyphens, "optional hyphen" and sometimes you can enter an "Em" or "En" dash or other kinds of dash marks.

Each kind of hyphen or dash, other than the standard keyboard "breaking hyphen", can be controlled with TagWrite. TagWrite does not provide default Tokens for these hyphens.

If you want to search for or insert any other type of hyphen in your document, you must create a new Token for it. You may build these Tokens. Refer to Appendix A and B of this manual and to the *Microsoft Word Technical Reference Manual*, Chapter 10.

Suggestions:

If you use the WordPerfect {hard hyphen} produced by pressing Home and the hyphen key, you produce a standard ASCII hyphen. You may produce the WordPerfect “hard hyphen” in a TagWrite rule by pressing the ordinary hyphen key in the Template Editor.

RTF Only: It is true also that if you are only working with RTF, you can use the keyboard hyphen in the Template Editor to represent the keyboard hyphen entered from within Microsoft Word. We recommend using the hyphen Token [-] to avoid confusion, but if you are an experienced TagWrite user, you can take advantage of this shortcut for RTF only.

Note: Keyentry Suggestion: Consider using the {hyphen} to start a paragraph if you need a way to delimit a certain category of paragraph for your keyentry standard. You would then set up a rule to look for a “[-][TeXt + #][HNL]”. Score the rule higher than any rule containing a [TeXt+#], [<TeXt>], [FILTxT] or [GRPTxT] Supertokens because the hyphen starts these Supertokens.

[^] (normal space)

Definition:

A caret in square brackets [^] represents a space Token in the Top line of a rule. It captures the occurrence of a {space} hex 20 code in the input document.

A space [^] Token in the Bottom line of a rule is treated as a normal {space} hex 20 in the output document.

Key Entry:

Spacebar

Starts:

None

Stops:

[NO], [STab], [CF], [PF], [TxT/all], [CF + PF]

Notes:

1. The caret is available in the Codes list in Token [**^**]. You do not need to use the [^] Token from the codes list. If you desire, you may generate a caret in the Template Editor by striking the Spacebar. Brackets are optional.
2. It is recommended, for clarity's sake, that you use the space Token with square brackets [^] whenever you follow it with a Frequency Indicator.
3. The occurrence of an ordinary hex 20 space is unpredictable and can occur randomly in a word processed document. For this reason, the space Token should always be set to be insignificant unless you have a specific reason and controlled application setting in which you can safely treat hex 20 space as significant.

   SUGGESTION: There are times when you want to ensure that one or more stray blank spaces are removed from the beginning of a line of text. By placing "[^]*" at the correct place in the Top rule, all spaces will be captured.
4. Scoring a rule using a [^]

Any time an initial space is required to fulfill a Rule, that Rule must be scored higher than any other rule seeking flush left text. The following example for a straight ASCII Template illustrates the point:

Current Element		**Next Element**
body		**body**
	([^][^][^][^]) [^]+[TeXt + #][CR][LF]	
30	**[TeXt + #][CR][LF]**	
	[TeXt + #][CR][LF]	
20	**<flush>[TeXt + #][CR][LF]**	

Any string of text preceded by five or more spaces will be given the tag of , <five>, while any other text will be tagged as <flush>.

[ALL]

Definition:

An [ALL] Token in the Bottom line of a rule returns everything captured in the Top line of the rule, including: Tokens, Supertokens, alphanumerics, {hard new line} and all other information captured by the Top line of a rule.

The [ALL] special code cannot be used in a Top line of a rule.

Started by:

Not applicable; [ALL] is never used in the Top line rule.

Stopped by:

Not applicable; [ALL] is never used in the Top line rule.

Notes:

Use [ALL] in the Bottom rule when you want to ensure that everything captured in the Top line is written to the new document. When [ALL] is used in the Bottom line, it may be preceded or followed by any characters, like tagnames or {hard new line}s.

[ALL] is useful when you wish to import all {tab}s and/or all word processor character format commands as well as all text into the typesetting software environment.

Do not use [ALL] for untagging templates. [ALL] will return the tagname captured in the Top line. Use [KEEP] for untagging.

[BELL]

Definition:

A [BELL] Token in the Top line of a rule captures the occurrence of the ASCII code (hex 07) for the BELL code in the input document.

A [BELL] Token in the Bottom line of a rule inserts an ASCII BELL code (hex 07) in the output document.

Starts:

None

Stops:

None

Notes:

[BELL] is commonly used as a typesetting delimiter by the U. S. Government Printing Office.

[BELL] is included in this list also to illustrate that you can create a Token out of any single ASCII character. In addition, you can create Tokens from any word processing code supported in the Setting file.

[BLD]

Definition:

A [BLD] Token in the Top line of a rule captures the occurrence of a {bold on} format command in the input document.

A [BLD] Token in the Bottom line of a rule inserts a {bold on} format command into the output document.

Starts:

[TeXt + #], [KEEP], [CF], [CF + PF]

Stops:

[STab], [FILTxT], [TxT/all], [PF]

Examples:

Example 1

You can use the [BLD] Token in the Top to strip unneeded {bold} formatting codes from your input document. In the following example, [BLD] is included in the Top rule, but omitted in the Bottom rule. This technique effectively eliminates the {bold on} code from the new document.

```
Current                                              Next
Element                                              Element
------------------------------------------------------------------
0    body                                              body
       [BLD] * [TeXt + #][HNL]
       ------------------------------------------
  00   [TeXt + #][HNL]
  ----------------------------------------------------------------
```

Example 2

In this example, the {bold} is being added in the Bottom rule. The objective is to ensure that the text appearing in the typeset version will appear in bold print. In the Bottom rule, make sure to include the [BLDOFF] Token before the [HNL] Token in such instances.

```
Current                                              Next
Element                                              Element
------------------------------------------------------------------
0    body                                              body
       [CENTER] [TeXt + #][HNL]+
       ------------------------------------------
  00   <title>[BLD][TeXt + #][BLDOFF][HNL]
  ----------------------------------------------------------------
```

Notes:

If your SGML Document Type Definition or typesetting software requires a specific embedded code to indicate bold on, then you can key enter that tagname on the Bottom line immediately preceding the text. Be sure to use the matching bold off tag name following the bolded text or subsequent text will continue to be formatted in bold.

The [CF], [CF + PF], and [TeXt + #] Supertokens are started by {bold on} and capture the {bold on} code. If the [TeXt + #] Supertoken happens to be started by a {bold on} word processing format code, and if the [TeXt + #] Supertoken is written back in the Bottom line, then the {bold on} code will be written to the output document. See the sections on [CF] and [TeXt + #] for additional information.

The [BLD] and [BLDOFF] Tokens and the [CF + PF] Supertoken may be used in conjunction with the [FILTxT] Supertoken to capture character formatting within a paragraph.

[BLDOFF]

Definition:

A [BLDOFF] Token in the Top line of a rule captures the occurrence of a {bold off} code in the input document.

A [BLDOFF] Token in the Bottom line of a rule inserts a {bold off} code into the output document.

Starts:

none

Stops:

[STab], [FILTxT], [PF]

Notes:

The [BLD] and [BLDOFF] Tokens may be used in conjunction with the [FILTxT] Supertoken character formatting within a paragraph.

The [BLDOFF] Token is preset to insignificant in TagWrite release.

Caution: Do not use [BLDOFF] or other “off” Tokens in the Bottom rule unless it is preceded somewhere in the rule by [BLD] (on). Some word processors cannot tolerate the presence of an “off” code if there is no prior “on” code.

[CENTER]

This section describes {center} for WordPerfect and RTF.

Note: WordPerfect supports two center format commands: normal {center} and a separate {center justified} format that is often used in place of the normal {center}. The TagWrite [CENTER] Token. is used to capture both WordPerfect {center justify} and normal {center}. This functionality is hard coded in TagWrite, and cannot be modified by the Template Designer.

Definition:

The [CENTER] Token is used in Microsoft Word and WordPerfect

A [CENTER] Token in the Top line of a rule captures the occurrence of centered text in the input document.

A [CENTER] Token in the Bottom line of a rule inserts a {center} format command into the output document.

Starts:

[PF], [CF + PF], [KEEP]

Stops:

[NO], [STab], [TxT/all],[CF], [FILTxT]

Examples :

Example 1

The following example shows the Top line of a rule written to capture centered chapter headings. In the Bottom rule, the center Token is dropped, the chapter heading is tagged, and the text in the heading is passed to the output document.

Current Element			Next Element
0	body		body
		[CENTER][TeXt + #][HNL]	
	00	<chaphead>[TeXt + #][HNL]	

Example 2

In the following example the Top line of the rule from Example 1 is re-written to require the presence in the text file of the centered word “Chapter”. This rule is written with the expectation that the word “chapter” will be followed by the chapter number. The word “CHAPTER” is written back in all caps in the Bottom line of the rule followed by any text or number captured by [TeXt + #]. This is a common method used to trap the chapter number and ensure that it is preceded by the word “CHAPTER”.

Current Element			Nex Element
0	body		body
		[CENTER]*((C) | (c))((hapter)|(HAPTER))[TeXt + #][HNL]	
	10	<chaptitl>e[TeXt + #][HNL]	

Notes:

Some word processors provide a {left and right indent} format code. The {center} code is not the same as the {left and right indent} code. The {left and right indent} code is insignificant in TagWrite as delivered.

[CF] (Character Format)

Definition:

A [CF] Supertoken in the Top line of a rule captures any one occurrence or many occurrences of either the {bold} or {underline} or {italics} or {strikethrough} or {small caps} format commands in the input document without having to specify which format command to capture. All format commands in any order will be captured.

A [CF] Supertoken in the Bottom line of a rule takes the {bold}, {underline}, {italics}, {strikethrough}, or {small caps} format command captured by the [CF] in the Top rule, and inserts it into the output document.

Code No.

80000040

Started by:

{Bold}, {underline}, {italics}, {small caps} {strikethrough}, {superscript}, {subscript}, {revised}, or {double underline}

Note: You may add such formats as super or subscript, double underline or other character formats. You must modify the [CF] Supertoken according to the instructions in Appendices 1 and 2 to this manual.

Stopped by:

{indent}, {center}, {flush right}, {first line indent}, {hard new line} or any alphanumeric character or punctuation.

Examples:

Example 1

In the following example, the [CF] Supertoken captures unwanted format commands in the Top line of a rule. The unwanted format commands are deleted from the tagged file by omitting [CF] (and [TAB]) from the Bottom line of the rule.

```
4   body                                            body
       [CF][TAB][TeXt+#][HNL]+

  20   <paratext>[TeXt+#][HNL][HNL]
```

Notes:

1. [CF] starts when it sees a character format code and stops when it sees text. Therefore, one, or many character format codes will be absorbed by [CF] if they occur before [CF] is stopped by the occurrence of an alphanumeric character. Thus, [CF] will capture any or all character format commands in any order.
2. A major reason to use [CF] is to eliminate tagging errors. By using [CF] in the Top rule, the heart of the rule can be fulfilled regardless of the appearance of character formatting codes at the start of the paragraph.

   You may be well advised to use an asterisk * with [CF]. Most often you use the [CF] Supertoken to remove stray character formatting or formatting that is not critical to making a tagging decision. The very notion of “stray” implies that you do not know if the character formatting is present or not. Thus, by using [CF]* with an asterisk, you can pick up the stray formatting if it is present, and if it is not present, the main part of the rule can still be fulfilled.

   If you did not use the asterisk, then the [CF] Supertoken would have to be fulfilled in order for the Rule itself to be fulfilled. This is rarely the intended outcome.

   Caution: If you use the asterisk in the Top line of a rule with the [CF] Supertoken or any Supertoken, then you cannot place the [CF] Supertoken in the Bottom line of that rule. Remember that the asterisk stands for “zero or more occurrences.” This means that [CF]* might not be fulfilled. If a Supertoken is not fulfilled in the Top, you cannot write it back in the Bottom. TagWrite will malfunction if you try to write an empty (zero occurrence) Supertoken to the output document.

[CF + PF]

Definition:

A Supertoken that is started by all significant word processing character format codes and all significant paragraph format codes.

[CF + PF]* cleans out the paragraph and character formatting where you do not want significant formats to influence fulfillment of the Rule.

Normally used with an asterisk.

Used primarily in the Cleanup Escape Rule and in the Top line following a Style Token to clean out paragraph and character formatting between the style Token and the beginning of text. If your version of TagWrite supports styles, see “Preparing a Styles Application”.

Code No.:

80000004

Started by:

{Bold}, {underline}, {italics}, {small caps}, {strikethrough}, {tab}, {superscript}, {subscript}, {revised}, {double underline}, {indent}, {center}, {first line indent}, {flush right}, {hanging indent}, {right indent}, {justification}

Stopped by:

{Space}, {hard space}, a style name embedded in the word processor file, or any alphanumeric character or punctuation.

Examples

Example 1: Styles

Current Element		Next Element
12	body	body
	[&style name][CF + PF]*[TeXt+#][HNL]+	
00	<tagname>[TeXt + #][HNL]	

The Top line of the rule will identify a word processor style name and then the [CF + PF]* Supertoken (with asterisk) absorbs any of the paragraph or character formats which are present. Because of the asterisk, if no character or paragraph format codes are present in the word processed file, the search for the Style Token will still be fulfilled.

If a character or paragraph format code is present, [CF + PF] starts and continues until it meets the first text character. This rule works only if your version of TagWrite supports the Style Module.

Example 2: Cleanup Escape Rule

[CF + PF] is used in a clean-up Escape Rule as in the example below:

```
    body                                        body
       [CF + PF][HNL]+
  99
```

The Top line is looking for any occurrence of a stray character or paragraph format and a {hard new line}. The Bottom line throws it away. This kind of Clean-up Escape Rule restrains your application from entering bogus tags where “phantom” paragraphs composed of stray formatting occur.

Scored higher than all other rules with which it might compete. In the default version of TagWrite, only rules using [TeXt+#] and [FILTxT] can compete.

Notes:

1. Since [CF + PF] does **not** stop when it meets a {tab}, you cannot use [CF + PF] to clean out unwanted formatting in rules that depend on a [TAB] Token as part of the Top Line.
2. Used in place of the separate ([CF] | [PF]) expression.

COUNTER TOKENS

[COLn], [COUNT^n], [C^n], [C^nA], [C^na]

Definition:

This section describes the second Counter Token also known as the “Column” or “Count” Token.

The second Counter Token is never used in the Top line of a rule.

A second Counter Token located in the Bottom line of a rule writes the current count to the output document, and then increments the Counter by the increment value defined in the COUNTER file. Access to the Counter file is described in Chapter 10 of the Template Designer’s Guide.

The Counter is reset to its starting value defined in the COUNTER file whenever a rule containing an End Counter or Reset Counter Token in the Bottom line is fulfilled.

Examples:

Example 1

In the following example, a Column Token is used to number text in a list or table.

Current Element		Next Element
0	body	body
	[TAB][TeXt + #][HNL]	
00	<list>[COL1].[TAB][TeXt + #][HNL]	

Example 2

In the following example, the Counter is used to number text in a list. When an “ ” string and one or more {hard new line}s are encountered in the input document, the column Counter is reset by [END/1] and two {hard new line} codes are written to the output document.

Current Element		Next Element
	body	body
	[TAB][TeXt + #][HNL]	
10	<list>[COUNT^1].[TAB][TeXt + #][HNL]	

body		body
	(<END>)[HNL]+	
10	[END/1][HNL][HNL]	

Note: <END> in the Top line is seeking to match actual ASCII text in the document indicating that the end of a list of some sort has been reached. We use this example because it is brief. In fact, the end of a structure in a document can be recognized in many ways. The most likely way would be to have the numerical counting rules in a separate Element. The Element would have a blank Top line Escape Rule. When the Escape Rule is fulfilled, the Bottom would return the [END/1] Counter Token, thus resetting the [C^1] counter. In complex counting rules, one category of rules may be reset by the occurrence of a second category of rules. When the first instance of the second category of rules is fulfilled, the Bottom rule of the second category may contain an [END/1] Token to reset the first group of Counters.

Counters can be used creatively to tag and untag any item that requires sequential ordering.

Notes:

The starting count and increment values for these Counters are defined in COUNTER file. See Chapter 10 of the Template Designer's Guide for more information on defining and operating Counters.

The default starting and increment values for the Counters are:

- [COLn], [COUNT^n], [C^n]: Start with 1 and increment by 1 .
- [C^nA]: Start with "A" and increment alphabetically.
- [C^na]: Start with "a" and increment alphabetically.

The Counters are reset to their starting values by [/TABLEn], [END/n], [/n], [/nA], or [/na], [/ROWn], [/RESET^n], [/Rn], [R^A], and [/R^a]. If a Counter reset is not encountered, the Counter continues incrementing until the end of the file.

COUNTER TOKENS (continued)

[ROWn], [HOLD^n], [H^n], [H^nA], [H^na]

Definition:

The row or hold Counter Token is never used in the Top line of a rule. The Template Editor “beeps” when attempting to put a Counter in the Top line.

A row or hold Counter Token in a Bottom line rule writes the current count into the output document.

The Counter is incremented (by the value defined in COUNTER file) whenever a [/ROWn], [R^n], [R^nA], [R^na], or [RESET^n] is encountered in the Bottom line of a rule that is fulfilled.

The Counter is reset to the starting value defined in the COUNTER file whenever its corresponding [/TABLEn], [/n], [/nA], [/na], or [END/n] is encountered in the Bottom line of a rule that is fulfilled.

See Chapter 10 of the Template Designer’s Guide for a complete discussion of counters.

Examples:

Example 1: In this example, hold and count Counters are used to assign numbers to text in a column. When the text string “ LEVEL” is encountered, the hold Counter is incremented by one, and the count Counter is reset to its starting value. When the text string “ ” is encountered, both Counters are reset to their starting values and two hard new line codes are written into the output document.

	Current Element		Next Element
	body	[TeXt + #][HNL]	body
10		<list>[HOLD^1](.)[COUNT^1][^][TeXt + #][HNL]	
	body	(<NEW^LEVEL>)[HNL]+	body
10		[RESET^1]	
	body	(<END>)[HNL]+	body
10		[END/1][HNL][HNL]	

Notes:

The starting count and increment values for the Counter are defined in COUNTER file. See Chapter 10 of the Template Designer’s Guide for a complete discussion of counters.

COUNTER TOKENS (continued)

[/ROWn], [RESET^n], [R^n], [R^nA], [R^na]

Definition:

The end of row or reset Counter Token is never used in the Top line of a rule.

An end of row or reset Counter Token located on the Bottom line of a rule increments the [ROWn], [HOLD^n] [H^n], [H^nA], or [H^na] Counter by the increment value specified in the COUNTER file and resets the [COLn], [COUNT^n], [C^n], [C^nA], or [C^na] Counter to the Counter's starting value as defined in the COUNTER file.

Example:

In the following example, hold and count Counters are used to assign numbers to text in a column. When " LEVEL" is encountered, the hold Counter is incremented by one and the count Counter is reset to its starting value.

	Current Element		Next Element
0	body		body
		[TeXt + #][HNL]	
10		<list>[HOLD^1](.)[COUNT^1][^][TeXt + #][HNL]	
0	body		body
		(<NEW LEVEL>)[HNL]+	
10		[RESET^1]	

Notes:

The starting count and increment values for the Counter are defined in the COUNTER file.

COUNTER TOKENS (continued)

[/TABLEn], [END/n], [/n] [/nA], [/na]

Definition:

The end of table, or end count, Counter Token is never used in the Top line of a rule.

An end of table, or end count, Counter Token located in the Bottom line of a rule resets the related Counter to its starting value as defined in COUNTER file.

Example:

In the following example, hold and count Counters are used to assign numbers to text in a column. When the text string " LEVEL" is encountered, the hold Counter is incremented by one and the count Counter is reset to its starting value. When the text string " " is encountered, both Counters are reset to their starting values and two {hard new line} codes are written to the output document.

Current Element			Next Element
0	body		body
		[TeXt + #][HNL]	
	10	<list>[HOLD^1](.)[COUNT^1][^][TeXt + #][HNL]	
0	body		body
		(<NEW LEVEL>)[HNL]+	
	10	[RESET^1]	
0	body		body
		(<END>)[HNL]	
		[END/1][HNL][HNL]	

Notes:

The starting count and increment values for each Counter are defined in COUNTER file. See Chapter 10 of the Template Designer's Guide for more information on defining and operating Counters.

[CR][LF]

Definition:

Used ONLY with straight ASCII files. Do not use in the Bottom line if you are writing a file from ASCII to WordPerfect or RTF.

The combination of the [CR] and [LF] Tokens in the Top line of a rule captures the occurrence of the {carriage return} {line feed} codes in the input document.

The combination of the [CR] and [LF] Tokens in the Bottom line of a rule inserts the {carriage return} {line feed} codes into the output document.

Caution: The [CR][LF] combination is not the same as a [HNL] Token.

WordPerfect and Microsoft RTF each use their own, single code in place of the {carriage return}{line feed} combination. This single code is represented by the TagWrite [HNL] Token.

When working on an ASCII to ASCII TagWrite application, you use [CR][LF]. When working ASCII to WordPerfect or RTF, or WordPerfect or RTF to ASCII, you use [HNL]. TagWrite knows how to prepare the text file for movement between ASCII and WordPerfect or RTF.

Starts:

None

Stops:

All Supertokens

Notes:

Please make a note of these four crucial points:

1. In most ASCII to ASCII applications, a {carriage return} code is almost always used in combination with a {line feed} code. These two codes in combination are known as a Carriage Return/Line Feed combination, or 0D 0A combination. The {carriage return} code is represented by the Token [CR]. The {line feed} code is represented by the Token [LF].
2. In instances where you wish to use frequency indicators with the [CR][LF] tokens to capture multiple occurrences of the {carriage return}{line feed} combination, you must place the [CR][LF] tokens in parentheses

   ([CR][LF] **)+**
3. Even though the [CR][LF] tokens appear on one line in the CODES column of the Template Editor, they are individual Tokens and must always be treated as two individual Tokens. The appearance of [CR][LF] on one line in the CODES column of the Template Editor is made available for your convenience because [CR] and [LF] tend to be used together in ASCII to ASCII applications.
4. One or more {carriage return}{line feed} combinations stop every Supertoken in the release version of TagWrite 3.0. It is very important to understand that although a {carriage return}{line feed} combination stops the Supertoken, the {carriage return}{line feed} combination itself is not

captured by the Supertoken. This means that after the Supertoken stops, you must intentionally capture {carriage return}{line feed} codes with the [CR][LF] Tokens in combination.

Examples:

Example 1

In the following example, one or more {carriage return}{line feed} combinations between paragraphs in the input document are captured in the Top line of the rule by the [CR][LF] tokens, and are replaced by a single {carriage return}{line feed} combination in the bottom line of a rule. The Top line of this rule is fulfilled by any paragraph which begins with an alphanumeric or punctuation character and ends with one or more {carriage return}{line feed} combinations. The Bottom line writes back a tagged paragraph and only one [CR][LF] combination to the new file. All extra {carriage return}{line feed} combinations from the original file are filtered out.

Current Element			Next Element
1	body		body
		[TeXt + #]([CR][LF])+	
	10	<para>[TeXt + #][CR][LF]	

Example 2

To capture a paragraph ending with a specific number of {carriage return}{line feed} combinations, place the desired number of [CR][LF] Tokens, within parentheses, at the end of the Top rule. The following rule looks for a text paragraph followed by two or more {carriage return}{line feed} combinations, and writes back a paragraph tagged as "agname" and one [CR][LF] combination.

Current Element			Next Element
1	body		body
		[TeXt + #] ([CR][LF])([CR][LF])+	
	10	<tagname>[TeXt + #][CR][LF]	

Example 3

To write a specific number of {carriage return}{line feed} combinations to the output document, place the desired number of [CR][LF] tokens at the end of the Bottom line. In the following example, all {carriage return}{line feed} combinations at the end of the text paragraph are captured, and three {carriage return}{line feed} combinations are written to the output document. This example also illustrates that you can search for the tagname "agname" in the Top line followed by a text paragraph and one or more {carriage return}{line feed} combinations, and write back the text paragraph followed by three {carriage return}{line feed} combinations.

Current Element			Next Element
1	body		body
		<tagname>[TeXt + #] ([CR][LF])([CR][LF])+	
	10	[TeXt + #][CR][LF][CR][LF][CR][LF]	

Example 4

To eliminate extra {carriage return}{line feed} combinations that were accidentally keyentered into the input document by the typist, use the following rule with a blank Bottom line:

Current Element			Next Element
1	body		body
		([CR][LF])+	
	10		

Reminders:

1. When you move from ASCII to WordPerfect or RTF format, treat your Template as if it were built for WordPerfect or RTF as appropriate and use [HNL].

   When going from ASCII to WordPerfect or RTF, TagWrite, through the “Options” and “With header” menu, automatically handles {carriage return} or {line feed} codes to prepare the ASCII file for importation into either WordPerfect or RTF.

2. Likewise, when going from RTF or WordPerfect to ASCII using the TagWrite “Save File In ASCII” option, use [HNL]

   Do **not** use [CR][LF] in the Template when going from WordPerfect or RTF to ASCII. TagWrite automatically saves your file in correct ASCII format.

3. Control of [CR][LF] combinations can be critical for using TagWrite to go from an ASCII file to an ASCII file.

ASCII to ASCII For Typesetting Software

In ASCII applications, most typesetting software recognizes the end of a paragraph by the presence of either one or no more than two {carriage return}{line feed} combinations. Spacing above and below a paragraph usually is determined in the typesetting software as a property of the tag name.

Most typesetting software, therefore, requires either one or no more than two {carriage return}{line feed} combinations at the end of a paragraph. Most importantly, the presence of additional or unexpected {carriage return}{line feed} combinations can sometimes cause inter-paragraph spacing errors during typesetting.

Most TagWrite tagging rules, therefore, will be written to return, in the Bottom Line, the number of {carriage return}{line feed} combinations specified by the typesetting software.

Untagging in ASCII to ASCII format

Untagging presents the reverse situation. Most ASCII documents in their draft form use two {carriage return}{line feed} combinations at the end of a paragraph. TagWrite untagging rules that reformat a tagged file into an untagged, ASCII file will usually include two {carriage return}{line feed} combinations in the Bottom rule.

Using [CR] and [LF] individually

There may be times when you receive non-standard ASCII files (such as files from mainframes) which do not contain the {carriage return}{line feed} combination, but only individual occurrences of either the {carriage return} or the {line feed}. The [CR] and [LF] tokens appear individually in the CODES column of the TagWrite Template Editor for instances when you need to use them separately.

[DULIN], [DULINOFF] (double underline)

Definition

Double underline is a character format supported in WordPerfect and Microsoft Word (RTF).

> **Caution:** Because {double underline} is not often used, it is **Insignificant** in TagWrite. Significance and Insignificance are technical terms in TagWrite explained in detail in Chapter 6 of the Template Designer's Guide. It is subject to insignificant character drop-off under certain circumstances. This can be avoided with Top line rules or by making [DULIN] significant in the Settings file.

A [DULIN] Token in the Top line of a rule captures the occurrence of a {double underline on} code in the input document.

A [DULIN] Token in the Bottom line of a rule inserts a {double underline on} code into the output document.

A [DULINOFF] Token in the Top line of a rule captures the occurrence of a {double underline off} code in the input document.

A [DULINOFF] Token in the Bottom line of a rule adds a {double underline off} code to the output document.

Starts:

[TeXt + #], [KEEP], [CF], [CF + PF]

Stops:

[STab], [TxT/all], [FILTxT], [PF]

Examples:

Example 1

You can use the [DULIN] Token to strip unneeded double underline formatting codes from the document. In the following example, [DULIN] is included in the Top rule, but omitted from the Bottom rule. This technique effectively drops the double underline from the file created by TagWrite.

Current Element		Next Element
body		body
	[DULIN]*[TeXt + #][HNL]	
10	[TeXt + #][HNL]	

Example 2

You can use the [CF]* Supertoken in the Top rule to strip formatting codes from the input document. In the Bottom rule, you can write back {double underline}, the desired text, and {double underline off}. This technique allows selective output of double underline in the file created by TagWrite.

Current Element		Next Element
body		body
	[CF]*[TeXt + #][HNL]	
10	[DULIN][TeXt + #][DULINOFF][HNL]	

Notes:

Since [TeXt + #] is started by the {double underline} on and off code, a {double underline} code at the beginning of a paragraph will trigger it. In fact, the {double underline on} code will be the first character captured by [TeXt + #]. If the [TeXt + #] Supertoken is then written back in the Bottom line, the {underline on} code will be retained and written to the new document. If you want to trap and remove the {double underline on} code, you must use the [DULIN] Token (or the [CF] Supertoken) before [TeXt + #] in the Top line of the rule. The Top rule of Examples 1 and 2 illustrate the trapping technique.

[ENDNOTE] (WordPerfect only)

Starts:

None

Stops:

[FILTxT], [CF], [PF], [CF+PF]

A “endnote”, for the purposes of this discussion, is text entered using the endnote utility in WordPerfect.

> **Note**: Internally, Microsoft Word uses endnote and footnote interchangeably with display determined by menu choice. The [ENDNOTE] Token is not used with Microsoft Word (RTF) applications.

In WordPerfect, Endnote text is not an actual, integrated part of the text paragraph to which the endnote is attached. The text of the endnote is contained within special format codes within WordPerfect. TagWrite uses special Template techniques featuring the [ENDNOTE] Token to capture and write endnote text.

Use of the [ENDNOTE] Token is explained in detail in the WordPerfect section of Chapter 8 of of the Template Designer’s Guide.

Note:

[FILTxT] is stopped by the WordPerfect {endnote] format. You can, therefore, add [ENDNOTE] to your “filtxt” intraparagraph tagging Element. The endnote rule in the “filtxt” Element acts the same as rules for bold, italic, underline, index and others that might be handled as intraparagraph tagging.

For a more complete example of the use of [FILTxT], see the first section of the chapter titled “Advanced Template Techniques”

FIELDS (RTF only) [FLD], [FLDINST], [FLDRSLT]

RTF only

Definition:

A field is a unique kind of RTF known group. Fields are special formatting instructions particular to Microsoft RTF files. Fields allow the insertion of variable information, such as date, time, and author, into a file. Their uses can be extremely complex; for example, if your RTF word processor supports Windows Dynamic Data Exchange, the DDE information which points to a linked file will be stored as a field in the RTF source file.

A field is composed of three separate known groups: Field, Field Instruction, and Field Result. The field group is the main group which contains the field instruction and the field result. The field instruction group contains the instruction or command which controls the word processor. The field result group contains the text and/or formatting information which is the result of the field. In some instances (auto numbering in Word for Windows; certain kinds of DDE links) the field result will be blank.

Examples:

The RTF “date" field inserts the current date into the file. This field group would appear in the RTF file as follows:

```
{\field{\*\fldinst date }{\fldrslt 11/20/91}}
```

The following rule would serve to tag the date text:

```
5     body
          [FLD][FLDINST](date)[/GRP][FLDRSLT][GRPTxT]
          [/GRP][/GRP]
   20     <date>[GRPTxT]</date>
```

Notice the multiple use of the [/GRP] Token. The first instance closes the field instruction group; the second instance closes the field result group; and the final instance closes the entire field group. It is very critical to match each group with its end group Token.

Also notice that the [GRPTxT] Supertoken is used to pick up the text of the field result, while a parenthetical search expression is used to pick up the field instruction. Remember that a Supertoken can not be used twice in the top line of a rule if it will be written back in the bottom line. If your field instructions are too complex to retrieve with parenthetical text string searches, it may be useful to create a second Supertoken for group text, perhaps named [GRPTxT2].

The [GRPTxT] Supertoken is described in detail on page of this "Command Reference." The [/GRP] Token is described in detail on page .

This final example demonstrates a search for legal-style autonumbering at the opening of a paragraph in a Microsoft Word for Windows RTF file. The field group appears in the RTF file as:

```
{\field{\*\fldinst autonumlgl }{\fldrslt }}
```

And the Template rule:

```
5    body
        [FLD][FLDINST](autonumlgl)[/GRP][FLDRSLT]
        [/GRP][/GRP][TeXt+#][HNL]+

  20    <legalnum>[TeXt+#][HNL]
```

If you wish to write Template rules that work with fields, it is strongly recommended that you acquire a copy of the Microsoft Word *Technical Reference*. That book provides a chapter which explains the function of each field instruction used in Microsoft Products. It also explains in detail the syntax of RTF files.

[FILTxT]

Definition:

“FILTxT” stands for filter text from character formatting.

[FILTxT] is always used in a looping Element arrangement – either an Element that loops within itself (with appropriate Escape Rules) or Elements that loop between each other.

[FILTxT] is used to isolate and identify intra-paragraph character formatting (bold, italic and so forth) of text that may start or be within a paragraph. It also can be configured to identify {index} codes within paragraphs.

[FILTxT] also is used in the table tagging rules illustrated in Chapter 9 of the Template Designer’s Guide.

A [FILTxT] Supertoken in the Top line of a rule captures text until the Supertoken encounters a character format listed below.

A [FILTxT] Supertoken in the Bottom rule writes back the text that has been captured, but does not write back the word processing character format code that stops the Supertoken.

Started by:

Any alphanumeric character and all punctuation.

Stopped by:

Stopped by virtually all character format codes on and off; table tokens; some paragraph format tokens, DDE or Link; RTF fields: Any of these codes: {bold on}, {underline on}, {italics on}, {strikethrough on}, {small caps on}, {bold off}, {underline off}, {italics off}, {small caps off}, {strikethrough off}, {tab}, {1st line indent}, {center}, {carriage return/linefeed}, {field instruction}, {field result}, {field start}, {end group}, {flushright}, {footnote start}, {hanging indent}, {hidden text on}, {hidden text off}, {index}, {indent}, {justification on},{rtf reset}, {rtf plain}, {revision on}, {revision off}, {right indent}, {tab}, {table of contents}, {uppercase on}, {uppercase off}, {WP justification}, {table start, table row and table cell}, {hard new line}.

Code No.:

80000200

Example:

In Chapter 9 of the Template Designer’s Guide, “Advanced Template Techniques”, the first section on intraparagraph character tagging carefully describes the use of [FILTxT]. The example in that chapter is too detailed to restate here.

The second section of Chapter 9 carefully describes the use of [FILTxT] for tagging cells and rows of tables.

The [FILTxT] Supertoken generally is used under circumstances where on and off codes delimit text that must be captured and surrounded by tags or other data. The “filtxt” Element using [FILTxT] captures normal text; stops when a [FILTxT] stopper word processing code (like {bold}) is present within a paragraph; moves on to capture the next block of text; stops when the end delimiter (like {end bold}) is encountered; writes back the text preceded by a tagname or other information; captures the closing word processing code; and writes back a closing tagname or other information.

Properly written, the “filtxt” Element continues to loop and continues to look for text and character formatting until the paragraph is ended with a {hard new line} code. The loop will then end allowing TagWrite to look for the next task posed by the text document.

See first and second sections of Chapter 9 in the Template Designer’s Guide. Also see the “Preparing a Styles Application” portion of the user manual supplied with the TagWrite Styles Modules.

[Flush^Rt]

Definition:

A [Flush^Rt] Token in the Top line of a rule captures the occurrence of a {flush right} command in the input document.

The {flush right} format command in Microsoft Word is sometimes called “Alignment Right”. In WordPerfect it is sometimes called “Justification Right”.

A [Flush^Rt] Token in the Bottom line of a rule inserts the {flush right} command into the output document.

Starts:

[PF], [CF + PF]

Stops:

[CF], [NO], [STab], [TxT\all], [FILTxT]

Example:

In the following example, the date in a memorandum is identified by flush right text.

Current Element			Next Element
0	**body**		**body**
		[Flush^Rt][TeXt + #][HNL]+	
00		**<date>[TeXt + #][HNL]**	

[FOOTNOTE]

Starts:

None

Stops:

[FILTxT], [CF], [PF], [CF+PF]

A “footnote”, for the purposes of this discussion, is text entered using the footnote utility in WordPerfect or Microsoft Word.

Footnote text is not an actual, integrated part of the text paragraph to which the footnote is attached. The text of the footnote is contained within special format codes within WordPerfect and Microsoft Word. TagWrite uses special Template techniques featuring the [FOOTNOTE] Token to capture and write footnote text.

TagWrite requires slightly different techniques when using the [FOOTNOTE] Token for WordPerfect or for Microsoft Word.

Use of the [FOOTNOTE] Token is explained in detail in Chapter 8 of the Template Designer’s Guide. There are separate sections for WordPerfect and Microsoft Word.

Note:

[FILTxT] is stopped by the WordPerfect {footnote] format. You can, therefore, add [FOOTNOTE] to your “filtxt” intraparagraph tagging Element. The footnote rule in the “filtxt” Element acts the same as rules for bold, italic, underline, index and others that might be handled as intraparagraph tagging.

For a more complete example of the use of [FILTxT], see the first section of Chapter 9 in the Template Designer’s Guide, titled “Advanced Template Techniques”

[GRPTxT]

RTF Only

The [GRPTxT] Supertoken is for use with RTF only.

Definition

The [GRPTxT] (for “group text”) Supertoken is used in RTF templates in conjunction with the [ENDNOTE], [INDEX], [TOC], [FLDINST], [FLDRSLT] and [/GRP] tokens to retrieve text that is inside known groups in RTF files. The [GRPTxT] Supertoken is for use with RTF only.

The Syntax and Structure of Known Groups

The most common known groups that require the use of the [GRPTxT] Supertoken are the footnote and index groups. Known groups in RTF always follow the following syntax:

Group Start Token–Group Text–Group End Token

Where the group start Token is either [ENDNOTE], [INDEX], [TOC], [FLDINST], or [FLDRSLT]; the group text is captured by the [GRPTxT] Supertoken; and the group end Token is represented by [/GRP].

Code Number:

80000008

Started by:

All alphanumeric and punctuation characters.

Stopped by:

The [/GRP] Token.

Example:

The following example demonstrates a search within a paragraph for text which has been formatted for automatic generation of footnotes, indexing and table of contents. After the initial text is found in the body element, the Template changes to the group element in which rules search for footnote, index and table of contents groups, as well as bold or italicized text. When the end of the paragraph is reached, the Template returns to the body element.

	Current Element		Next Element
1	body		group
		[TAB][FILTxT]	
20		<paratext>[FILTxT]	
2	group		group
		[ENDNOTE][CF+PF]*[GRPTxT][/GRP]	
40		<footnote>[GRPTxT]</footnote>	
2	group		group
		[INDEX][CF+PF]*[GRPTxT][/GRP]	
40		<index>[GRPTxT]</index>	
2	group		group
		[TOC][CF+PF]*[GRPTxT][/GRP]	
40		<toc>[GRPTxT]</toc>	
2	group		group
		[BLD]	
40		<bold>	
2	group		group
		[ITAL]	
40		<italic>	
2	group		group
		([BLDOFF]|[ITALOFF])	
40		<normal>	
3	group		group
		[FILTxT]	
30		[FILTxT]	
4	group		body
		[HNL]+	
20		[HNL]	

[/GRP]

RTF only.

The [/GRP] token is for use with RTF only.

Definition:

The {end group} Token is used in RTF templates only to indicate the end of an RTF known group. It is used in conjunction with the [END-NOTE], [INDEX], [TOC], [FLDINST],and [FLDRSLT] tokens and the [GRPTxT] Supertoken to tag text which has been formatted for automatic generation of footnotes, indexes, and tables of contents in RTF word processors. [/GRP] is the only Token which ends the [GRPTxT] Supertoken.

Starts:

None

Stops:

[GRPTxT]

Example:

For complete examples on the use of [/GRP] see the examples given above under [GRPTxT] and [ENDNOTE].

[HIDE]

Definition:

Hidden text is indicated in RTF by the control word \v.

In RTF, hidden text is contained in the RTF source file but the hidden text neither displays on screen nor prints.

In TagWrite, hidden text is considered to be a Character Format and, with one exception, it is handled the same as [BOLD].

The exception is that hidden text is **Insignificant**.

> **Caution:** [INDEX] captures index markers in the document. Index markers are hidden text. Hidden text must be insignificant in order for [INDEX] to be captured by the Template.

[HNL]

Definition:

The hard new line Token is used to represent the word processor {hard new line} which is entered into a word processed file by pressing the <Enter> key.

A [HNL] Token in the Top line of a rule captures the occurrence of a {hard new line} code in the input document.

A [HNL] Token in the Bottom line of a rule inserts the {hard new line} code into the output document.

Starts:

None

Stops:

All TagWrite pre-configured Supertokens are stopped by [HNL].

> **Note:** You can make a Supertoken that is not stopped by [HNL]. There is no requirement to stop a Supertoken with [HNL].

Notes:

One or more {hard new line} codes stop every Supertoken in the release version of TagWrite 3.0. It is very important to understand that although a {hard new line} code stops the Supertoken, the {hard new line} code itself is not captured by the Supertoken. This means that after the Supertoken stops, you must intentionally capture {hard new line} codes with a [HNL] Token.

Examples:

Example 1

In the following example, one or more {hard new line} codes between paragraphs in the input document are captured in the Top line of the rule by the [HNL]+ codes, and are replaced by a single {hard new line} code in the Bottom line of a rule. The Top line of this rule is fulfilled by any paragraph which begins with a text or number character and ends with one or more {hard new line} codes. The Bottom line writes the paragraph and only one [HNL] to the new file. All extra {hard new line} codes from the original file are filtered out.

	Current Element		Next Element
	body		body
		[TeXt + #][HNL]+	
10		[TeXt + #][HNL]	

Example 2

To capture a paragraph ending with a specific number of {hard new line} codes, place the desired number of [HNL] Tokens at the end of the Top rule.

Current Element		Next Element
body		body
	[TeXt + #][HNL][HNL]	
10	[TeXt + #][HNL]	

Example 3

To write a specific number of {hard new line} codes to the output document, place the desired number of [HNL] Tokens at the end of the Bottom line. In the following example, all {hard new line}s at the end of the text paragraph are captured, and three {hard new line} codes are written to the output document by placing three [HNL] Tokens at the end of the Bottom line of the rule.

Current Element		Next Element
body		body
	[TeXt + #][HNL]+	
10	[TeXt + #][HNL][HNL][HNL]	

Example 4

To eliminate extra {hard new line} codes that were accidentally keyentered into the input document by the typist, use the following rule with a blank Bottom line:

Current Element		Next Element
body		body
	[HNL]+	
1		

Most robust Templates will contain some form of [HNL] cleanout rule in each Element.

See also the discussion on the “Cleanup Escape Rule” in Chapter 10 of the Template Designer’s Guide.

Notes:

Control of {hard new line} can be critical for tagging and untagging:

1. Generally, SGML applications do not take account of hard new lines. However, for the sake of clarity if the SGML file is to be read and for the sake of certainty if the SGML file is to be untagged and reformatted, it is generally sound policy to separate paragraphs by hard new lines. This means that if the original document is prepared in WordPerfect or Microsoft Word and processed in TagWrite, the TagWrite [HNL] Token will give proper paragraph separation, and TagWrite “Save File As ASCII” will transform the [HNL] into [CR][LF] combinations.
2. Tagging for typesetting. Most typesetting software recognizes the end of a “paragraph” by the presence of a {hard new line}. Generally, the paragraph is tagged and treated as a single typographical entity. Spacing above and below a paragraph usually is determined in the typesetting software as a property of the tag name.

   Most typesetting software, therefore, requires only one {hard new line} at the end of a paragraph. Indeed, the presence of more than one {hard new line} can sometimes cause inter-paragraph spacing errors during typesetting. Most TagWrite tagging rules, therefore, are written to return one [HNL] in the Bottom line. Although there will be occasions where more than one [HNL] will be written to a Bottom rule, generally, it is best to control inter-paragraph spacing using the tools provided by the typesetting software. It is generally not a good policy to try to control inter-paragraph spacing with {hard new line} codes.
3. Untagging presents the reverse situation. Most word processed documents use two {hard new lines} at the end of a paragraph. TagWrite untagging rules that reformat a tagged file into an untagged, word processed format will usually include two {hard new lines} in the Bottom rule.

ASCII And The Difference Between [HNL] and [CR][LF]

If you are working with ASCII files, it is very important to understand the difference between [HNL] and [CR][LF]. The difference is discussed fully in the discussion on [CR][LF] earlier in this Chapter.

Difference Between “line break” and [HNL].

In Microsoft Word, a {hard new line} is differentiated from a {line break}. A {line break} code terminates text on a line before the right margin and breaks the line, but does not terminate the paragraph or the paragraph formatting. The {line break} code, therefore, is not the same as the {hard new line} code which creates a new paragraph. The {line break} code is not supported by a preset Token or Supertoken.

If you want to support {line break} with a Supertoken, you should treat it as an intraparagraph format as explained in the section on [FILTxT].

[HPg]

Definition:

A [HPg] Token in the Top line of a rule captures the occurrence of a {hard page break} code in the input document.

A [HPg] Token in the Bottom line of a rule inserts the {hard page break} code into the output document.

Starts:

None

Stops:

None

Example:

{hard page break} codes are inserted into documents to signify major changes – such as a new chapter, to control strict paging of the text, or for other reasons. For example, {hard page break} is often used as a field delimiter for data base structures.

The following example illustrates a fragment of a Template where {hard page break} causes the end of a chapter in a word processed file.

The "body" Element's Top line rule searches for and captures a {hard page break} code. When the Top rule is fulfilled, the Bottom returns a hard page break tag and a [HNL]. TagWrite then branches to the "chap" Element. Its Top line rule then searches for a centered chapter heading.

Current Element		Next Element
body		chap
	[HPg]	
	<hard^page>[HNL]	
chap		chap
	[CENTER][TeXt + #][HNL]+	
	<chaptitle>[TeXt + #][HNL]	

[H^]

Definition:

A [H^] Token in the Top line of a rule captures the occurrence of a hard space in the input document.

A [H^] Token in the Bottom line of a rule inserts a hard space into the output document.

Starts:

[TeXt + #], [FILTxT], [KEEP], [<TeXt>]

Stops:

[NO], [STab], [CF], [TxT/all], [PF], [CF + PF]

Notes:

The hard space code is not standard ASCII and, therefore, is a unique word processing format code. You must use the [H^] Token in the Template Editor to control a {hard space} code in the Top or Bottom line of a rule.

[H^] is not a trivial Token. There are a number of clever ways to use it. For example, in TagWrite version 3.0, the {hard space} code is set to start the Supertoken [TeXt + #].

There is a good use for this when using TagWrite to auto tag the data collected using “fill in the blank” word processing macros (data entry word processing applications). If a {hard space} code is present as the first character in the answer form, it will be invisible on screen, but digitally present. If the “fill in the blank” line is not filled in by the user, the [TeXt + #] Supertoken will still be fulfilled. This will allow the “blank” line to be tagged correctly because even if the user did not fill in the blank, it may be important for that data structure to be tagged.

Specifically, if the macro asks the user to fill in a telephone number, but the telephone number is omitted, the Top rule still will be fulfilled when it encounters the {hard space} code. The Bottom rule will return the word “Telephone:”, and the [TeXt + #] Supertoken which contains the {hard space} code. The typeset copy will show the word “Telephone:” and blank space where the telephone number should be.

[INDEX]

RTF Only

See the note at the end of this section on capturing index using Word-Perfect.

Definition:

The [INDEX] Token is used in RTF templates only to search for text inserted within an RTF index group for purposes of auto-indexing. The [INDEX] Token, when used correctly, enables you to capture text which has been marked for indexing and insert that text within descriptive tags suitable for your document publishing system or SGML. For information about indexing text with your RTF word processing software, consult your word processor's technical documentation.

Starts:

None

Stops:

[FILTxT], [CF], [PF], [CF+PF]

Examples:

The [INDEX] Token represents the start of the RTF index group, and must be used in conjunction with the Supertoken [GRPTxT] and the Token [/GRP]. In the following example, [FILTxT] is used in the body element. When an [INDEX] Token is encountered, the Template changes to the "index" element, where it searches for the text which has been marked for indexing within a paragraph. When the end of the paragraph is reached, the Template returns to the body element:

Current Element			Next Element
1	**body**		**filtxt**
		[FILTxT]	
	20	**<para>[FILTxT]**	
2	**filtxt**		**filtxt**
		[INDEX][CF+PF]*[GRPTxT][/GRP]+	
	40	**<indextag>[GRPTxT]</indextag>**	
3	**filtxt**		**filtxt**
		[FILTxT]	
	30	**[FILTxT]**	
4	**filtxt**		**body**
		[HNL]+	
	20	**[HNL]**	

Note On Index and WordPerfect:

TagWrite does not capture WordPerfect index delimiters or its text. When a WordPerfect file is saved as ASCII using either the TagWrite or WordPerfect “Save As ASCII” utility, the index information is lost.

You can use a word processing code like {underline} or in place of the word processor’s index code if you do not need to generate an actual index in the word processor but do need to tag the text file with index codes. Underlining text that is to be indexed is much easier than trying manually to insert specific start and stop index codes.

To use {underline} as the index marker, set-up the rule as follows: In the filtxt Element, add a TOP rule that looks for the {underline} word processor code. By picking up underline, you can return the index tag.

```
9    filtxt                                      filtxt
       [ULIN]
  4    <index_start>

10   filtxt                                      filtxt
       [ULINOFF]
  3    <index_end>
```

[INDT]

Definition:

An [INDT] Token in the Top line of a rule captures the occurrence of a {left indent} command in the input document.

An [INDT] Token in the Bottom line of a rule inserts the {left indent} code into the output document.

Starts:

[PF], [CF + PF], [KEEP]

Stops:

[CF], [NO], [STab], [TxT/all], [FILTxT]

Examples:

Example 1

The {indent} word processing format code is often used to indicate a "stepped" or "indented" paragraph. An indented paragraph is one where the first line and all subsequent lines are indented. "Steps" are commonly used in technical manuals and other publications to form paragraph lists. A first level step is often indicated by one word processor indent; a second level step is indicated by two indents, and so forth. The following example, demonstrates rules for three levels of steps.

```
1    body                                                body
        [CF]*[INDT][TeXt +#] [HNL]+
  10    <step1>[TeXt +#] [HNL]

2    body                                                body
        [CF]*[INDT] [INDT][TeXt + #][HNL]+
  9     <step2>[TeXt +#] [HNL]

0    body
        [CF]*[INDT] [INDT][INDT][TeXt + #][HNL]+
  8     <step3>[TeXt + #][HNL]
```

Example 2

For bulleted lists, we suggest using an {indent} code followed by an asterisk in the word processed file to indicate that you want to tag the paragraph as a bulleted list. See the following Template rules.

```
1   body                                              body
      [CF]*[INDT] ('*') [TeXt +#] [HNL]+
  10  <bullet>[TeXt +#] [HNL]
```

Note the use of single quotes and parentheses. In the example, the asterisk is enclosed in single quotes because it is text and not a Frequency Indicator.

Example 3

A more robust version of this rule could be written as follows:

```
1   body                                              body
      [CF]*[INDT] ( '*' )+[TeXt +#] [HNL]+
  10  <bullet>[TeXt +#] [HNL]
```

By enclosing the asterisk in single quotes, we treat the asterisk as text. By enclosing the entire quote mark and asterisk expression in parentheses like this (' * ') we can use a frequency indicator after the entire expression. In the Template example above, the plus sign Frequency Indicator causes the Top rule to search for one or more occurrences of the asterisk as a text character. Although the key entry convention may call for a bulleted list to be entered with one asterisk, we are accounting for the possibility that the typist has made an error and used two asterisks.

Notes:

The [INDT] Token represents a {left indent} code only. TagWrite treats {left and right indent} codes as insignificant. You can build a Token for {left and right indent} although it is rather exotic. As always, it would be better to keep your applications as simple as possible.

[ITAL]

Definition:

An [ITAL] Token in the Top line of a rule captures the occurrence of an {italics on} code in the input document.

An [ITAL] Token in the Bottom line of a rule inserts an {italics on} code into the output document.

Starts:

[TeXt + #], [KEEP], [CF]. [CF + PF]

Stops:

[FILTxT], [TxT/all], [STab], [PF]

In general you use the [ITAL] (italics on) Token in the same manner as [BLD] (bold on).

Example:

You can use the [ITAL] Token to strip unneeded italics formatting codes from the document. In the following example, [ITAL] is included in the Top rule, but omitted in the Bottom rule. This technique effectively drops the {italics on} code from the new document:

	Current Element		**Next Element**
0	**body**		**body**
		[ITAL] * [TeXt + #][HNL]	
10		**[TeXt + #][HNL]**	

Since [TeXt + #] is started by the {italics on} code, an italics format code at the beginning of a paragraph will trigger it. In fact, the {italics on} code will be the first character captured by [TeXt + #]. If the [TeXt + #] Supertoken is then written back in the Bottom line, the {italics on} code will be retained and written to the new document. If you want to trap and remove the {italics on} code, then, as in the example above, you must use the [ITAL] Token (or the [CF] Supertoken) before [TeXt + #] in the Top line of the rule.

The [ITAL] and [ITALOFF] Tokens may be used in conjunction with the [FILTxT] Supertoken to capture character formatting within a paragraph. [FILTxT] is described at the beginning of Chapter 10 of the Template Designer's Guide.

[ITALOFF]

Definition:

An [ITALOFF] Token in the Top line of a rule captures the occurrence of an {italics off} code in the input document.

An [ITALOFF] Token in the Bottom line of a rule inserts an {italics off} code into the output document.

Starts:

None

Stops:

[FILTxT], [STab], [TxT/off], [PF]

Notes:

> **Caution:** Do not use [ITALOFF] in the Bottom rule unless it is preceded somewhere in the rule by [ITAL] (on). Some word processors balk at the presence of an "off" code without a prior "on" code.

The [ITAL] and [ITALOFF] Tokens may be combined with the [FILTxT] Supertoken to capture paragraph or character formatting within a paragraph.

[JUST] (RTF Only)

RTF allows a justify format command. Justification means that text aligns evenly at the left and right margins of the page.

The TagWrite [JUST] Token captures the occurrence of {justification on} at the start of a paragraph.

Starts:

[PF], [PF+CF]

Stops:

[FILTxT]

Notes:

[JUST] is set to be insignificant, thus, within the Element, it must be scored higher than any rule started by a Significant Token or by a Supertoken.

The [JUST] Token is supported for Microsoft Word (RTF) only. WordPerfect justification {full justify} is considered to be a default by TagWrite. The [JUST] Token does not capture WordPerfect {full justify}

[KEEP]

Definition:

A [KEEP] Supertoken in the Top line of a rule captures everything in the input document except a [HNL], which stops [KEEP].

A [KEEP] Supertoken in the Bottom line of a rule inserts everything captured by the [KEEP] in the Top line of the rule into the output document.

Hex No.:

0x80000010

Started by:

{Tab}, {indent}, {center}, {flush right}, {first line indent}, {bold}, {underline}, {italics}, {small caps}, {strikethrough}, and all alphanumeric characters and punctuation.

Stopped by:

{Hard new line} code.

Examples:

Example 1

For untagging, the following example shows how to write a rule to untag a file. The Top line of the rule captures a tag name followed by [KEEP] and at least one [HNL]. The Bottom line of the rule does not return the tagname. The Bottom does pass everything captured by the Top line's [KEEP] and adds two {hard new line} codes to reproduce the original word processed document. This removes the tagname, but retains all text and format commands contained in the original document. Done carefully, you can restore a word processing file to its exact, original form.

Current Element		**Next Element**
body		**body**
	<para^I>[KEEP][HNL]+	
10	**[KEEP][HNL][HNL]**	

Note: Do not use [ALL] to untag a file because [ALL] returns everything in the Top line including the tag name. As shown above, you can first capture the specific tag name in a parenthetical, alphanumeric, expression and then, with [KEEP], capture the rest of the text up to the {hard new line}. Writing [KEEP] back in the Bottom line allows you to drop the tag name and insert any additional word processing format information you desire.

Example 2

In the following example, the [KEEP] Supertoken creates an Escape Rule for your main “body” Element group. This allows the Template to preserve text when no valid Template tagging rule applies.

Current Element		Next Element
body		body
	[KEEP][HNL]+	
0	[KEEP][HNL]	

If no rule in the Template is fulfilled, this [KEEP] Escape Rule captures all characters and commands, and then writes them to the output document. The errant paragraph is not tagged, but it is retained. The Escape Rule prevents TagWrite from slowing down when it finds a paragraph for which there is no Template rule.

Note the 0 zero score assigned to the escape rule and read about its importance below.

> **Caution:** The [KEEP] Escape Rule is designed only for the main or “body” Element of your Template. Do not use the [KEEP] Escape Rule in sub-Elements of your Template because you will create a rule that loops to the end of file or which exits the sub-Element prematurely.
>
> The [KEEP] Escape Rule is discussed thoroughly in “Advanced Template Techniques”.

Notes:

The [KEEP] Escape Rule must be scored zero (00). If [KEEP] is scored higher than any other rule, [KEEP] will always be fulfilled. In that case, the file will not be tagged. [KEEP] will simply capture each paragraph and write it back to disk.

TagWrite will always terminate a tagging operation when it encounters an end-of-file. That means that there is an overriding escape rule which will work even when there is no terminating [HNL] (as required in the example above.) The Bottom line of the active rule will not be executed and thus the final paragraph will not be tagged if the text does not end with an appropriate “Token-stopping” character such as an [HNL].

If the last paragraph of a document will fulfill neither a tagging rule nor a [KEEP] rule, check to make sure that the last paragraph is ended by a {hard new line}. Most rules are written to terminate with a [HNL] Token, but sometimes the typist simply ends the paragraph without a {hard new line}. In the worst case, you should simply receive a tagging error, but not experience data loss.

[NO]

Definition:

A [NO] Supertoken in the Top line of a rule captures numbers in the input document. It is especially useful for tagging numbered paragraphs.

A [NO] Supertoken in the Bottom line of a rule inserts the number captured by the last [NO] in the Top line into the output document.

Code No.:

80000002

Started by:

Numerals 0 through 9

Stopped by:

Any alphabetic character, a period, {space}, {hard space}, {indent}, {flush right}, {first line indent}, {tab}, {center}, and all other punctuation – except the hyphen and the comma.

What Is A “Number”?

A “number” in TagWrite is defined as a single digit like 1 or 2 or 3. A number may also be several digits, such as 123 or 10988. As long as the number is not broken by a word processing character that ends the [NO] Token, the size of the number is not relevant. The number 1 is as valid as the number 198,000,008.

Number And The Hyphen

Hyphens neither stop nor start [NO] Supertokens.

You can use this to advantage if you wish to track chapter number and the paragraph number while typing. Chapter 23, Paragraph 405, for instance, could be written 23-405, and the [NO] Supertoken would consider it just another single number.

Examples:

The following example, illustrates how to differentiate and tag text paragraphs that are numbered in the following way:

1 First paragraph to be tagged <PARA1>

1.1 Second paragraph to be tagged <PARA2>

1.1.1 Third paragraph to be tagged <PARA3>

	Current Element		Next Element
0	START		body
00			
1	body		body
		[CF+PF]*[NO](.)*[NO](.)*[NO](.)*([^]|[PF])*[TeXt + #][HNL]+	
20		<para3>[TeXt + #][HNL]	
2	body		body
		[CF+PF]* [NO](.)* [NO](.)* ([^] | [PF])* [TeXt + #] [HNL]+	
19		<para2>[TeXt + #][HNL]	
3	body		body
		[CF+PF]* [NO](.)* ([^] | [PF])* [TeXt + #] [HNL]+	
18		<para1>[TeXt + #][HNL]	

Notes:

Notice that the [NO] Supertoken can be used more than one time in a rule. This allows you to capture as many separate numbers as needed.

Most typesetting software and SGML require that paragraph numbering in the original document be erased. In SGML and most typesetting, paragraphs are categorized by level and the level is denoted by the Tagname.

If the level is known to the typesetting software, then auto-numbering utilities based on tag names can be used to number the paragraphs.

By capturing the number with one or more [NO] Tokens, you can choose not to write back the numbers in the Bottom line. Replace the captured number with the correct tagname.

The example above also demonstrates how to use a period with [NO] to distinguish between different levels of numbered paragraphs.

As with all Supertokens, you can not place more than one [NO] Supertoken in the Bottom line of a rule. The number that will be written back on the Bottom line will be the one captured by the last [NO] Supertoken on your Top line. This is true because the contents of the [NO] on the Top are replaced every time the Supertoken's requirements are met. Thus, it would not be helpful to use the [NO] Supertoken on the Bottom line of the first two rules in the above example.

The following example assumes that you are not going to use the auto-numbering capabilities of your typesetting software. To return to the actual numbers keyentered by the typist, you would need to use [ALL] on the Bottom line of every rule. You would also need to eliminate the plus (**+**) sign after the [HNL] in the Top line of every rule.

For example:

```
   body                                                        body
       [CF+PF]*[NO](.)*[NO](.)*[NO](.)*([^]|[PF])*[TeXt + #][HNL]
20     <para3>[ALL]
```

[PF] Paragraph Format

[PF]* is used by itself when the requirement is to capture paragraph format codes but not character formatting.

Definition:

A [PF] Supertoken in the Top line of a rule starts with the occurrence of either the {tab}, {indent}, {center}, {first line indent}, or {flush right} word processor format codes in the input document without having to specify which format command to capture.

The [PF] Supertoken continues absorbing all paragraph format information until stopped by the Stop tokens listed below.

A [PF] Supertoken in the Bottom line of a rule inserts the format captured in the Top Rule into the output document.

Code No.:

80000400

Started by:

{tab}, {indent}, {center}, {first line indent}, {flush right} {hanging indent}, {right indent}, {justification}.

Stopped by:

{Space}, {hard space}, {bold}, {underline}, {italic}, {small caps}, {strikethrough}, {hard new line} or any alphanumeric character or punctuation.

Examples:

Example 1: In the following example, the paragraph format Supertoken captures unwanted format commands in the Top line of a rule. The unwanted format commands are deleted from the tagged file by omitting [PF] from the Bottom line of the rule.

Current Element			Next Element
12	body		body
		[PF]* [TeXt + #][HNL]	
00		**[TeXt + #][HNL]**	

Notes:

1 By using [PF] in the Top rule, the heart of the rule can be fulfilled regardless of the occurrence of significant paragraph formatting codes at the start of the paragraph.

2. It is difficult to predict the placement, kind, and combination of paragraph format commands in a word processing file. [PF] will capture any or all paragraph format commands in any order.

   [PF]* is almost always used with an asterisk. Most often you use the [PF]* Supertoken to remove stray paragraph formatting or formatting that is not critical to making a tagging decision. The very notion of “stray” implies that you do not know if the paragraph formatting is present or not. Thus, by using [PF]* with an asterisk, you can pick up the stray formatting if it is present, and if it is not present, the main part of the rule can still be fulfilled.

   If you did not use the asterisk, then the [PF] Supertoken would have to be fulfilled in order for the Rule itself to be fulfilled. This is rarely the intended outcome.

   > **Caution:** If you use the asterisk on the Top line of a rule with the [PF]* Supertoken, then you cannot place the [PF] Supertoken on the Bottom line of that rule. Remember that the asterisk stands for “zero or more occurrences.” This means that [PF]* might not be fulfilled. If a Supertoken is not fulfilled in the Top, you cannot write it back in the Bottom. TagWrite will malfunction if you try to write an empty (zero occurrence) Supertoken to the output document.

3. The Supertoken [CF + PF] is separate from [PF] and performs a combined function of the separate [CF] and [PF] Supertokens. [CF + PF] is often used in Top line rules rather than [CF] or [PF] separately, especially with rules searching for word processor Styles.

[PLAIN]

Used with RTF only.

Definition

[PLAIN] clears all prior **character** formatting like {bold}, {italic}, {superscript} and all others considered by RTF to be a character format. It is a generic shut off for character formatting that can be used anywhere in a document to terminate the current character format and return formatting to "Normal".

[PLAIN] restores text to the "Normal" or default character format that has been established in the Microsoft Word document.

Used only with templates designed for Microsoft RTF. The TagWrite [PLAIN] Token is equivalent to the RTF code "\plain".

[PLAIN] generally is used in **each** Bottom rule in Templates designed for RTF.

Explanation:

A Microsoft Word document saved as RTF (or other word processors that follow the Microsoft standards for writing RTF) is designed so that every new paragraph either:

- carries the character formatting from the previous paragraph or carries character formatting from a prior piece of text unless that formatting has been shut off.

  OR

- clears into "plain" formatting ("Normal") the character display of each paragraph.

In TagWrite operations, you always have control over the formatting of each new paragraph. **In templates designed for use with RTF, we strongly recommend that you write Bottom rules that first clear the pre-existing character formatting.** You may then control and correctly declare the paragraph formatting of your output document.

The character formatting for the current paragraph is declared by:

- Placing a tagname in the Bottom rule. The tagname defines the character format that should be attached to the text of the paragraph. (See Example 1 below)

  OR

- Placing the appropriate Token in the Bottom line of the Rule like [BLD] or [ITAL] .
- Placing a stylename Token in the Bottom rule if your version of TagWrite supports the Style module and your application is a styled application. The stylename Token defines the character formatting for the paragraph.

  Caution: If you do not use [PLAIN], then the formatting of the previous paragraph may automatically be attached to the current paragraph when the output document is saved in RTF.

Starts

none

Stops

none

Examples:

Example 1

The following example has a Top rule that simply traps any specific character formatting, and picks up the text and {hard new lines} associated with the paragraph.

The Bottom rule uses [RESET] first to clear any prior paragraph formatting that may have existed in the input document. The Bottom then clears any character formatting using [PLAIN]; inserts a tagname and returns the text of the paragraph and a {hard new line}.

This is an elementary and fundamental example for writing rules for RTF templates.

Current Element			Next Element
0	body		body
		[CF]*[TeXt + #][HNL]	
00		[RESET][PLAIN]<tagname>[TeXt + #][HNL]	

Example 2

Example 2 is almost the same as example 1 except rather than write back a tagname in the Bottom line, example 2 writes back the character format {bold}.

Current Element			Next Element
0	body		body
		[CF]*[TeXt + #][HNL]	
00		[RESET][PLAIN][BLD][TeXt + #][BLDOFF]	

Notes:

In Templates designed for RTF, [RESET] followed by [PLAIN] is almost always used in the Bottom line of a Rule.

[PLAIN] can effectively be used by itself where specific character formatting needs to be shut off inside a paragraph or other structure in the document. [PLAIN] shuts off any character format; therefore, it is a generic shut off that can be used in RTF Template applications in place of the specific character format end Token.

[RESET]

Used only with RTF applications.

Definition

Resets paragraph formatting to the default, "normal' settings.

Used **only** in the Bottom line rule.

> **Caution:** Do not use [RESET] on Top. The rule will never be fulfilled.

Used in all paragraph level RTF rules, but **not** used as part of intraparagraph, “filtxt” style rules.

> **Caution:** [RESET] appearing anywhere in a paragraph will cause the reset of the entire paragraph to the default, “normal” setting. If you use [RESET] on the Bottom line of intraparagraph tagging, then the entire paragraph will be reset to the default “normal”.

The TagWrite [RESET] Token is equivalent to the RTF code “\pard”.

[RESET] generally is used in **each** Bottom rule in Templates designed for RTF.

Explanation:

A Microsoft Word document saved as RTF (or other word processors that follow the Microsoft standards for writing RTF) is designed so that every new paragraph either:

- carries the paragraph formatting from the previous paragraph. (We call this “subsequent paragraph formatting”)

  OR

- “resets” the paragraph formatting of each paragraph before adding new or default paragraph formatting.

In TagWrite operations, you always have control over the formatting of each new paragraph. In RTF applications, for each new paragraph, you first clear the previous formatting with [RESET]. This follows the logic of native RTF.

In Templates designed for use with RTF, we strongly recommend that you write Bottom rules that first “resets” the paragraph formatting. You may then control and correctly declare the paragraph formatting of your output document.

The paragraph formatting is declared by:

- Placing a tagname in the Bottom rule. The tagname defines the structure of the paragraph. (See Example 1 below)

  OR

- Placing the appropriate Token in the Bottom line of the Rule like [CENTER] or [INDENT] or [TAB]. (See Example 2 below)

  OR

- Placing a stylename Token in the Bottom rule if your version of TagWrite supports the Style module and your application is a styled application.

If no special formatting is required, that is, if you want the paragraph to carry the default, “Normal” paragraph level formatting characteristics, then use [RESET] with no additional format information.

> **Caution:** If you do not [RESET] the paragraph formatting, then the formatting of the previous paragraph may automatically be attached to the current paragraph when the output document is saved in RTF.

Starts

none

Stops

none

Examples:

Example 1

The following example has a Top rule that simply traps any character formatting, and picks up the text and {hard new lines} associated with the paragraph.

The Bottom rule uses [RESET] first to clear any prior paragraph formatting that may have existed in the input document. The Bottom then clears any character formatting (using [PLAIN]); inserts a tagname and returns the text of the paragraph and a {hard new line}.

This is an elementary and fundamental example of writing rules for RTF templates.

Current Element			Next Element
0	body		body
		[CF]*[TeXt + #][HNL]	
	00	[RESET][PLAIN]<tagname>][TeXt + #][HNL]	

Example 2

Example 2 is almost the same as example 1 except rather than write back a tagname in the Bottom line, Example 2 writes back a specific paragraph format (e.g. {indent})

Current Element			Next Element
0	body		body
		[CF]*[TeXt + #][HNL]	
	00	[RESET][PLAIN][INDT][TeXt + #][HNL]	

Notes:

[RESET] is almost always followed by [PLAIN]. While [RESET] clears all prior **paragraph** formatting, [PLAIN] clears all prior **characte**r formatting. See the discussion of the [PLAIN] Token in this chapter.

[REVISED], [REVOFF]

Definition

Refers to {revised text} lodged in the input document. Does not refer to the document revision number. Revised text is text that has been marked as changed since the last publication. The tokens for Revised text are for use only in RTF file; WordPerfect does not have a Revised text token.

The traditional Revision Mark appears as a vertical line in the margin next to the text that has been revised.

An SGML Document Type Definition or other document standard may require that you specifically tag Revised text.

The TagWrite Template can be configured to recognize and isolate the revision code. The TagWrite Template technique will vary according to the document structure and your tagging needs. You may need to establish the revision code as part of [FILTxT]. You will also have to establish a search pattern using [REVISED]* (with an asterisk) as part of the beginning of a Top line rule.

We do not offer an example here because the application is specific and will require that you understand your document structure and your application.

Some Microsoft Word Products allow you to designate that text has been revised. [REVISED] and [REVOFF] are not for use in templates for WordPerfect.

Starts:

none in the release version of TagWrite 3.0

Stops:

none in the release version of TagWrite 3.0

Notes:

To use the revision Tokens to start and stop Supertokens, you must configure the Settings file for your application. See the Appendix B which describes how to alter the Settings file.

[SmCAP]

Definition:

A [SmCAP] Token in the Top line of a rule captures the occurrence of a {small caps on} code in the input document.

A [SmCAP] Token in the Bottom line of a rule inserts a {small caps on} code into the output document.

Starts:

[TeXt + #], [KEEP], [CF],[CF + PF]

Stops:

[STab], [CF], [FILTxT], [TxT/all], [PF]

Examples:

Example 1

You can use the [SmCAP] Token to capture text beginning with {small caps on} or to strip unneeded {small caps on} codes from the original document. In the following example, [SmCAP] is included in the Top rule, but omitted in the Bottom rule. This technique allows you to use the {small caps on} code as a special text identifier and drops the code from the output document:

Current Element		Next Element
body		body
	[SmCAP]*[TeXt + #][HNL]+	
10	[TeXt + #][HNL]	

Example 2

The following example illustrates that you can capture plain text in the Top rule and write it back in small caps. Note that the [SmCAP] Token is balanced by a [SmCAP/](small caps off) Token in the Bottom rule. If you do not insert the [SmCAP/] off Token, then the [SmCAP] (on) Token will have no intentional ending, and may serve to format a longer string of text than you intend.

Current Element		Next Element
body		body
	[TeXt + #][HNL]+	
10	[SmCAP][TeXt + #][SmCAP/][HNL]	

Notes:

The [CF] and [TeXt + #] Supertokens start with the code for {small caps on}.

The [SmCAP] and [SmCAP/] Tokens may be used in conjunction with the [FILTxT] Supertoken to capture paragraph leaders or character formatting within a paragraph. See Example under [FILTxT] in this chapter for further information.

[SmCAP/]

Definition:

A [SmCap/] Token in the Top line of a rule captures the occurrence of a {small caps off} code in the input document.

A [SmCap/] Token in the Bottom line of a rule inserts a {small caps off} code into the output document.

Starts:

None

Stops:

[STab], [FILTxT], [PF]

Notes:

> **Caution:** Do not use [SmCAP/] in the Bottom rule unless it is preceded somewhere in the rule by [SmCAP] (on). Some word processors cannot tolerate the presence of an “off” code if there is no prior “on” code.

The [SmCAP] and [SmCAP/] Tokens may be used in conjunction with the [FILTxT] Supertoken to capture character formatting at the start of a paragraph or character formatting within a paragraph. See the Example under [FILTxT] in this chapter for further information.

START

Definition:

START is not a Token or Supertoken. Notice that START is not contained in square brackets ([ ]). START is a required file marker, entered as the Current Element in Rule 0 of every Template.

START is hard coded into the Current Element of Rule Zero and cannot be edited.

Notes:

1. Normally, the START Element consists of a blank Top and Bottom line. It is not advised to use the Top line of the START Rule. The Bottom line can be used to place information at the beginning of the text file.
2. Not appended. Note that Rule 0 is not appended using the TagWrite Append utility. If there is information in the Bottom line of Rule 0, it will be lost in the append process.
3. A Next Element name is **required.**

   In most instances, the first information in the START rule is the Next Element which directs the Template to the opening Current Element for the particular application.

[STRKOFF]

Definition:

A [STRKOFF] Token in the Top line of a rule captures the occurrence of a {strikethrough off} code in the input document.

A [STRKOFF] Token in the Bottom line of a rule inserts a {strikethrough off} code into the output document.

Starts:

None

Stops:

[FILTxT], [STab], [PF]

Notes:

The [STRKTHRU] and [STRKOFF] Tokens can be used in conjunction with the [FILTxT] Supertoken to capture {strikethrough} codes that begin paragraphs, or to capture character formatting within a paragraph. See the Example under [FILTxT] in this chapter for further information.

> **Caution:** Do not use [STRKOFF] in the Bottom rule unless it is preceded somewhere in the rule by [STRKTHRU] (on). Some word processors cannot tolerate the presence of an "off" code if there is no prior "on" code.

[STRKTHRU]

Definition:

A [STRKTHRU] Token in the Top line of a rule captures the occurrence of a {strikethrough on} code in the input document.

A [STRKTHRU] Token in the Bottom line of a rule inserts a {strikethrough on} code into the output document.

Starts:

[TeXt + #], [KEEP], [CF], [CF + PF]

Stops:

[FILTxT], [STab], [TxT/all], [PF]

Example:

You can use the [STRKTHRU] Token to strip unneeded {strikethrough} codes from the input document. In the following example, [STRKTHRU] is included in the Top line of the rule, but omitted in the Bottom line. This technique effectively drops the {strikethrough on} code from the output document.

	Current Element		Next Element
0	body		body
		[STRKTHRU]*[TeXt + #][HNL]+	
00		**[TeXt + #][HNL]**	

Notes:

The [CF] and [TeXt + #] Supertokens capture the code for {strikethrough} (on). The code for {strikethrough} (on) will thus be written out to the new text file if [TeXt + #] appears in the Bottom line. The solution is to capture the {strikethrough} word processing code with [STRKTHRU] in the Top line rule. In the example above, we did capture the {strikethrough} code in the Top rule, and chose not to write it back in the Bottom.

The [STRKTHRU] and [STRKOFF] Tokens may be used in conjunction with the [FILTxT] Supertoken to capture a {strikethrough} code that begins a paragraph, or to capture {strikethrough} character formatting within a paragraph. See Example 1 under [FILTxT] in this chapter for further information or see the opening examples in Chapter 10 in the Template Designer's Guide.

[SUPER], [SUB] (Super and Subscript)

Definition:

A [SUPER] Token in the Top line of a rule captures the occurrence of a {superscript on} code in the input document. The Token in the Bottom line of a rule inserts a {superscript on} code into the output document.

A [SUB] Token in the Top line of a rule captures the occurrence of a {subscript on} code in the input document. The Token in the Bottom line of a rule inserts a {subscript on} code into the output document.

A [SUPEROFF] Token in the Top line of a rule captures the occurrence of a {superscript off} code in the input document. The Token in the Bottom line of a rule inserts a {superscript off} code into the output document.

A [SUBOFF] Token in the Top line of a rule captures the occurrence of a {subscript off} code in the input document. The Token in the Bottom line of a rule inserts a {subscript off} code into the output document.

Starts:

None

Stops:

[STab], [FILTxT]

Notes:

Do not use [SUPEROFF] or [SUBOFF] in the Bottom rule unless it is preceded somewhere in the rule by [SUPER] or [SUB]. Some word processors cannot tolerate the presence of an “off” code if there is no prior “on” code.

The superscript and subscript TagWrite Tokens may be used in conjunction with the [FILTxT] Supertoken to capture character formatting at the start of a paragraph or character formatting within a paragraph. See the first part of Chapter 10 in the Template Designer’s Guide for an explanation of how the “filtxt” Element works.

[TAB]

Definition:

A [TAB] Token in the Top line of a rule captures the occurrence of a {tab} code in the input document.

A [TAB] Token in the Bottom line of a rule inserts a {tab} code into the output document.

Starts:

[KEEP], [STab], [PF], [CF + PF]

Stops:

[FILTxT], [STab], [NO], [CF], [TxT/all]

Example:

In the following example, the [TAB] Token is used to identify paragraph levels. The number of {tab} codes entered before a paragraph allows the Template to determine the paragraph type or level.

```
      body                                                  body
          [CF]*[Tab][TeXt +#] [HNL]+
          ____________________________________________________
   10     <para1>[TeXt +#] [HNL]
   ____________________________________________________________________
0     body
          [CF]*[Tab][CF]*[Tab][TeXt +#] [HNL]+
          ____________________________________________________
          <para2>[TeXt +#][HNL]
   ____________________________________________________________________
0     body
          [CF]*[Tab][CF]*[Tab][CF]*[Tab][TeXt +#] [HNL]+
          ____________________________________________________
          <para3>[TeXt +#][HNL]
   ____________________________________________________________________
```

Notes:

Do not confuse a {tab} code with an {indent} code. They are different word processing commands. Also, do not confuse a {true tab} code with a Microsoft Word {first line indent} code.

[TAB] and [STab] are different. [STab] only serves as a substitute for [TAB] when the Token is used in a rule to pick up a {first line indent} which may be a true {tab} or Microsoft Word {first line indent}. See the discussion on this topic under [STab] in this chapter for further information.

A {tab} code, when entered at the beginning of a paragraph, causes the first line of text to be moved in from the left margin. The remaining lines are flush left to the margin.

{Tab} codes are often used between the columns of tables. Since a {tab} code stops [FILTxT], you can create Template routines that will tag virtually any tabbed table entry routine. If your tables use tabs to separate columns and you want to create special Template tagging routines, you can create new Supertokens for the purpose.

Table Tokens

Tagwrite supports tagging of cells and rows of tables generated from within Microsoft Word (RTF) and WordPerfect. A different set of table Tokens is used for each word processor.

RTF Table Tokens:

RTF Control	Meaning	TagWrite Token
\trowd	reset to table row defaults	**[TBLONRTF]**
\cell	end of cell	**[/CELLRTF]**
\row	end of row	**[/ROWRTF]**
\intbl	new row within table	**[ROWnuRTF]**

WordPerfect Table Tokens:

Meaning	TagWrite Token
Table on/table definition	**[TBLONWP]**
Beginning of a cell	**[CELLWP]**
Beginning of a row at end of line	**[ROWWP1]**
Beginning of row at end of page	**[ROWWP2]**

Note:

The use of these Tokens is rigid and well defined. Their correct use is explained and illustrated in the appropriate sections of Chapter 8 in the Template Designer's Guide.

[TOC]

RTF Only

Definition:

The [TOC] Token is used in RTF templates only to search for text which has been inserted within an RTF table of contents group for purposes of automatic table of contents generation. The [TOC] Token, when used correctly, enables you to capture text which has been marked for table of contents generation and insert that text within descriptive tags suitable for your document publishing system. For information about marking text for inclusion within a table of contents with your RTF word processing software, consult your word processor's technical documentation.

Starts:

None

Stops:

[FILTxT], [CF], [PF], [CF+PF]

Examples:

The [TOC] Token represents the start of the RTF table of contents group, and must be used in conjunction with the Supertoken [GRPTxT] and the Token [/GRP]. In the following example, [TOC] is used in the body element. When a [TOC] Token is encountered, the Template changes to the "toc" element, where it searches for the text which has been marked for table of contents within a paragraph. When the end of the paragraph is reached, the Template returns to the body element:

	Current Element		Next Element
1	**body**		**toc**
		[FILTxT]	
20		**<para>[FILTxT]**	
2	**toc**		**toc**
		[TOC][CF+PF]*[GRPTxT][/GRP]	
40		**<toc>[GRPTxT]</toc>**	
3	**toc**		**toc**
		[FILTxT]	
30		**[FILTxT]**	
4	**toc**		**body**
		[HNL]+	
20		**[HNL]**	

[TeXt + #]

Definition:

A [TeXt + #] Supertoken in the Top line of a rule is started by any alphanumeric character, by punctuation, or by certain character format commands noted below. [TeXt + #] captures all text and formatting commands of any length until a {hard new line} code is encountered. The {hard new line} stops [TeXt + #], but the {hard new line} is not captured by [TeXt + #].

A [TeXt + #] Supertoken in the Bottom line of a rule inserts the string captured by [TeXt + #] in the Top line of the rule.

Code No.:

80000001

Started by:

All numbers, letters, and punctuation as well as these codes: {bold on}, {underline on}, {italics on}, {small caps on} and {strikethrough on}. [TeXt + #] is not started by paragraph format codes such as: {tab}, {center}, {flush right}, {indent}, or {first line indent}.

Stopped by:

[HNL]

Notes:

Notice that the codes: {bold on}, {underline on}, {italics on}, {small caps on} and {strikethrough on} all start [TeXt + #]. This means that if you want to specifically trap these character format codes, you must place the desired character format Token or [CF] before [TeXt + #] on the Top line.

You cannot place more than one [TeXt + #] Supertoken in the Bottom line of a rule.

If you place more than one [TeXt + #] Supertoken in the Top line, only the contents of the last [TeXt + #] Token in the Top line will be available. This is true because the contents of [TeXt + #] in the Top line will be replaced every time the Supertoken's requirements are met.

[<TeXt>]

Definition:

A [<TeXt>] Supertoken in the Top line of a rule captures alphanumeric characters, punctuation and format commands delimited by angle brackets < >. The left angle bracket delimiter **<** does not start [<TeXt>] and the right angle bracket > stops, but is not captured by, [<TeXt>].

A [<TeXt>] Supertoken in the Bottom line of a rule takes the characters captured by the [<TeXt>] Supertoken in the Top line of the rule and places them into the output document. Note that [<TeXt>] does not capture the angle bracket itself, thus the angle bracket ASCII characters are not transferred to the Bottom line by [<TeXt>].

Hex No.:

0x80000020

Started by:

Any alphanumeric character and all punctuation except the left and right angle brackets < >.

Stopped by:

A left or a right angle bracket < or > or the [HNL] code.

Notes:

The purpose of the [<TeXt>] Supertoken is not to capture the angle brackets, but rather to capture the text embedded between angle brackets.

Once the text embedded in angle brackets is captured in the Top, you can write it back in any format in the Bottom or throw it away.

See Untagging in Chapter 9 of the Template Designer's Guide and Flat ASCII in Chapter 8 of that manual for examples.

[TxT/all]

Definition:

A [TxT/all] Supertoken in the Top line rule captures one and only one upper or lower case alphabetic character.

A [TxT/all] Supertoken in the Bottom line of a rule takes the character captured by the [TxT/all] Supertoken in the Top line and places it into the output document. If you don't use the [TxT/all] Supertoken in the Bottom rule, the character is discarded.

Code No.

80000100

Started by:

Any upper or lower case alphabetic character.

Stopped by:

All punctuation, any alphanumeric character, and these codes: {tab}, {indent}, {center}, {flush right}, {first line indent}, {space}, {hard space}, {bold}, {italics}, {small caps}, {strikethrough}, {underline}, and {hard new line}.

Example:

The [TxT/all] Supertoken can be used to remove a letter (a, b, c, A, B, C, etc.) that serves as an ordering system for lists or paragraphs in the input document.

Sample text would look like this:

a. Text of the paragraph

    (a) Text tabbed one level

        a. Text tabbed two levels with "a" underlined

The example Template rules that follow search for:

Rule 1 – a text paragraph preceded by a letter and a period.

Rule 2 – a text paragraph preceded by a {tab} code and a letter in actual parentheses.

Rule 3 – a text paragraph preceded by two {tab} codes, an underlined letter and a period.

Current Element			Next Element
1	body		body
		[TxT/all](.)[TeXt + #][HNL]+	
	10	<para1>[TeXt + #][HNL]	
2	body		body
		[TAB](’(’)[TxT/all](’)’)[TeXt + #][HNL]+	
	11	<para2>[TeXt + #][HNL]	
3	body		body
		[TAB][TAB][ULIN][TxT/all](.)[TeXt + #][HNL]+	
	12	<para3>[TeXt + #][HNL]	

The single quotes around the left and right parens in the second example rule are necessary because we are searching for actual parentheses. Left and right parens are reserved characters in the TagWrite Template Editor, so they must be enclosed in single quotes if they are to be treated as actual text.

Notes:

Use [TxT/all] when the input document contains alphabetized paragraph ordering that must be identified and tagged.

Use [TxT/all], as in this example, when the input document contains alphabetized paragraph ordering and the output document requires that paragraph numbering (alphabetizing) be replaced by tagnames that identify the level of the paragraph. The letter is captured in the Top line of the rule but is not written back in the Bottom line. Instead, it is replaced by a tagname.

Virtually all SGML Document Type Definitions and typesetting systems have this requirement. For example, [TxT/all] is also used for SGML applications where paragraph numbers are replaced by tag names.

To capture numbers, substitute the [NO] Supertoken for the [TxT/all] Supertoken in the Top rule line.

A more robust version of the examples above would search for stray spaces or tabs before the [TeXt + #] Supertoken to ensure that the spacing between the text and the typeset auto numbering is consistent.

[ULIN]

Definition:

An [ULIN] Token in the Top line of a rule captures the occurrence of an {underline on} code in the input document.

An [ULIN] Token in the Bottom line of a rule inserts an {underline on} code into the output document.

Starts:

[TeXt + #], [KEEP], [CF], [CF + PF]

Stops:

[STab], [FILTxT], [TxT/all], [PF]

Examples:

Example 1

You can use the [ULIN] Token to strip unneeded underline formatting codes from the document. In the following example, [ULIN] is included in the Top rule, but omitted from the Bottom rule. This technique effectively drops the underline from the file created by TagWrite.

Current Element		**Next Element**
body		**body**
	[ULIN]*[TeXt + #][HNL]	
10	**[TeXt + #][HNL]**	

Example 2

You can use the [CF]* Supertoken in the Top line to strip formatting codes from the input document. In the Bottom rule, you can write back {underline}, the desired text, and {underline off}. This technique allows selective output of underline in the file created by TagWrite.

Current Element		**Next Element**
body		**body**
	[CF]*[TeXt + #][HNL]	
10	**[ULIN][TeXt + #][ULINOFF][HNL]**	

Notes:

Since [TeXt + #] is started by the {underline on} code, an {underline} code at the beginning of a paragraph will trigger it. In fact, the {underline on} code will be the first character captured by [TeXt + #]. If the [TeXt + #] Supertoken is then written back in the Bottom line, the {underline on} code will be retained and written to the new document. If you want to trap and remove the {underline on} code, you must use the [ULIN] Token (or the [CF] Supertoken) before [TeXt + #] in the Top line of the rule. The Top rule of Examples 1 and 2 illustrate the trapping technique.

The [ULIN] and [ULINOFF] Tokens may be used in conjunction with the [FILTxT] Supertoken to capture character formatting within a paragraph. See the first part of Chapter 10 in the Template Designer's Guidefor an example of how the "filtxt" Element is constructed.

[ULINOFF]

Definition:

An [ULINOFF] Token in the Top line of a rule captures the occurrence of an {underline off} code in the input document.

An [ULINOFF] Token in the Bottom line of a rule adds an {underline off} code to the output document.

Starts:

none

Stops:

[FILTxT], [STab], [PF]

Notes:

See Example 2 in the [ULIN] (on) section of this chapter for an illustration of use in the Bottom line of [ULINOFF] when writing back underlined text.

> **Caution:** Do not use [ULINOFF] in the Bottom rule unless it is preceded somewhere in the rule by [ULIN] (on). Some word processors cannot tolerate an “off” code without a prior “on” code.

The [ULIN] and [ULINOFF] Tokens may be used in conjunction with the [FILTxT] Supertoken to capture character formatting within a paragraph. See the example under [FILTxT] in this chapter and the first section of Chapter 10 of the Template Designer’s Guide.

WordPerfect Justification Group

Center justify:

WordPerfect 5.1 including WordPerfect For Windows allow text to be centered normally or centered with justification. Normal center and justified center are two different format commands.

Normal WordPerfect {center} centers only one line of text. It does not wrap.

Center **justify** permits multiple, wrapping lines centered one over the other.

Handling of normal {center} and {center justified} is described in the section on the [CENTER] Token, page of this chapter.

Right justify:

Text in WordPerfect formatted as {right justify} causes text to align evenly on the right margin of the page with a ragged left margin.

WordPerfect {right justify} is supported by the [FLUSH^RT] TagWrite Token described on page of this chapter.

Full justified text

WordPerfect 5.1 and higher supports {full justify}. Full justification causes text to be aligned evenly at the left and right margins of a page.

WordPerfect {full justify} is considered a default by TagWrite. That is, TagWrite does not consider {full justify} to be a discrete format command in WordPerfect. The TagWrite [JUST] Token does **not** capture WordPerfect {full justify}.

TagWrite supports {full justify} with the [JUST] Token for Microsoft Word (RTF) described on page .

Left justified text

WordPerfect {left justify} is considered a default by TagWrite. TagWrite does not support a Token to capture {left justify}.

[/WPDATA]

Definition:

The [/WPDATA] Token is used only in WordPerfect templates that support Styles applications.

[/WPDATA] represents the WordPerfect code that signals the end of the formatting data associated with a WordPerfect style. The [/WPDATA] Token is required on the Bottom lines of rules that write back WordPerfect styles; in such instances it immediately follows the style Token.

The [/WPDATA] Token must not be used with RTF style applications.

Correct use of [/WPDATA] is explained in the manual titled “Preparing a Styles Application” which is part of the TagWrite Styles Module.

Starts:

None

Stops:

[CF], [PF], [CF+PF], [STab]

[WPLINK]

Definition:

"Link" is the WordPerfect term for Dynamic Data Exchange (DDE) in the Microsoft Windows environment.

Within the native WordPerfect file, the DDE link information is contained in a WordPerfect, link information group. This information group is not an actual part of the text file since it is stored in a WordPerfect binary format.

The [WPLINK] TagWrite Token captures the entire WordPerfect link group. The information captured in [WPLINK] must be written back on the bottom line using [ALL].

	Element	**Element**
1	**filtxt**	**filtxt**
	[WPLINK]	
20	**<link path> [ALL]**	

Starts:

None

Stops:

[FILTxT]

On the Bottom line you may write any ASCII text information before and after the [ALL] Token. With this technique, the path name can be surrounded with any information, like a tag or other information, that is required for your system.

> **Caution:** Do not use **[WPLINK]** Token on the Bottom line! To write back the link information, you must use the TagWrite **[ALL]** Token.

Save Link As ASCII

When the WordPerfect file is Saved As ASCII through the TagWrite menu, the link path information is preserved as ASCII text information.

Untagging and Writing WordPerfect Links

> **Caution:** TagWrite does **not** allow untagging a document and writing back WordPerfect links to create a valid WordPerfect file. The [WPLINK] Token cannot be used on the Bottom line of a rule.