IEEE P1003.2 Draft 11.2 – September 1991

(7) If the current character is an unquoted `<newline>`, the current token shall be delimited.

(8) If the current character is an unquoted `<blank>`, any token containing the previous character is delimited and the current character is discarded.

(9) If the previous character was part of a word, the current character is appended to that word.

(10) If the current character is a #, it and all subsequent characters up to, but excluding, the next `<newline>` are discarded as a comment. The `<newline>` that ends the line is not considered part of the comment.

(11) The current character is used as the start of a new word.

Once a token is delimited, it shall be categorized as required by the grammar in 3.10.

3.3.1 Token Recognition Rationale. *(This subclause is not a part of P1003.2)*

The (3) rule about combining characters to form operators is not meant to pre- 1
clude systems from extending the shell language when characters are combined in 1
otherwise invalid ways. Portable applications cannot use invalid combinations 1
and test suites should not penalize systems that take advantage of this fact. For 1
example, the unquoted combination |& is not valid in a POSIX.2 script, but has a 1
specific KornShell meaning. 1

The (10) rule about # as the current character is the first in the sequence in which a new token is being assembled. The # starts a comment only when it is at the beginning of a token. This rule is also written to indicate that the search for the end-of-comment does not consider escaped `<newline>` specially, so that a comment cannot be continued to the next line.

3.4 Reserved Words

Reserved words are words that have special meaning to the shell. (See 3.9.) The following words shall be recognized as reserved words:

```
!       elif    fi      in      while
case    else    for     then    {
do      esac    if      until   }
done
```

[4] (after `{`)

---

4) In some historical systems, the curly braces are treated as control operators. To assist in future standardization activities, portable applications should avoid using unquoted braces to represent the characters themselves. It is possible that a future version of POSIX.2 may require this, although probably not for the often-used `find {}` construct.

This recognition shall occur only when none of the characters are quoted and when the word is used as:

(1) The first word of a command

(2) The first word following one of the reserved words other than `case`, `for`, or `in`

(3) The third word in a `case` or `for` command (only `in` is valid in this case)

See the grammar in 3.10.

The following words may be recognized as reserved words on some systems (when none of the characters are quoted), causing unspecified results:

`function` `select` `[[` `]]` 2

Words that are the concatenation of a name and a colon (`:`) are reserved; their use produces unspecified results.

3.4.1 Reserved Words Rationale. *(This subclause is not a part of P1003.2)*

All reserved words are recognized syntactically as such in the contexts described. However, it is useful to point out that `in` is the only meaningful reserved word after a `case` or `for`; similarly, `in` is not meaningful as the first word of a simple command.

Reserved words are recognized only when they are delimited (i.e., meet the definition of *word* ; see 3.1.16), whereas operators are themselves delimiters. For instance, `(` and `)` are control operators, so that no `<space>` is needed in `(list)`. However, `{` and `}` are reserved words in `{ list;}`, so that in this case the leading `<space>` and semicolon are required.

The list of unspecified reserved words is from the KornShell, so portable applications cannot use them in places a reserved word would be recognized. This list 2
contained `time` in earlier drafts, but it was removed when the `time` utility was 2
selected for the UPE. 2

There was a strong argument for promoting braces to operators (instead of reserved words), so they would be syntactically equivalent to subshell operators. Concerns about compatibility outweighed the advantages of this approach. Nevertheless, portable applications should consider quoting `{` and `}` when they represent themselves.

The restriction on ending a name with a colon is to allow future implementations that support named labels for flow control. See the rationale for `break` (3.14.1.1).

3.5 Parameters and Variables

A parameter can be denoted by a name, a number, or one of the special characters listed in 3.5.2. A variable is a parameter denoted by a name.

A parameter is set if it has an assigned value (null is a valid value). Once a variable is set, it can only be unset by using the `unset` special built-in command.

3.5.1 Positional Parameters

A positional parameter is a parameter denoted by the decimal value represented by one or more digits, other than the single digit 0. When a positional parameter with more than one digit is specified, the application shall enclose the digits in braces (see 3.6.2). Positional parameters are initially assigned when the shell is invoked (see `sh` in 4.56), temporarily replaced when a shell function is invoked (see 3.9.5), and can be reassigned with the `set` special built-in command.

3.5.1.1 Positional Parameters Rationale. *(This subclause is not a part of P1003.2)*

The digits denoting the positional parameters are always interpreted as a decimal value, even if there is a leading zero.

3.5.2 Special Parameters

Listed below are the special parameters and the values to which they shall expand. Only the values of the special parameters are listed; see 3.6 for a detailed summary of all the stages involved in expanding words.

* Expands to the positional parameters, starting from one. When the expansion occurs within a double-quoted string (see 3.2.3), it expands to a single field with the value of each parameter separated by the first character of the **IFS** variable, or by a `<space>` if **IFS** is unset.

@ Expands to the positional parameters, starting from one. When the expansion occurs within double-quotes, each positional parameter expands as a separate field, with the provision that the expansion of the first parameter is still joined with the beginning part of the original word (assuming that the expanded parameter was embedded within a word), and the expansion of the last parameter is still joined with the last part of the original word. If there are no positional parameters, the expansion of @ shall generate zero fields, even when @ is double-quoted.

Expands to the decimal number of positional parameters.

? Expands to the decimal exit status of the most recent pipeline (see 3.9.2).

– (Hyphen) Expands to the current option flags (the single-letter option names concatenated into a string) as specified on invocation, by the `set` special built-in command, or implicitly by the shell.