'\\' represents the single backslash character, *i.e.* the \ is an escape character.

'\f' represents the form-feed character, *i.e.* ASCII 12.

'\n' represents the new-line character, *i.e.* ASCII 10.

'\r' represents the carriage-return character, *i.e.* ASCII 13.

'\t' represents the tab character, *i.e.* ASCII 09.

'\v' represents the vertical-tab character, *i.e.* ASCII 11.

There are STAR specifications not definable in the EBNF. The EBNF can be used to define the tokenization of the input stream, and a STAR parser should test that the following condition is true. The number of data elements in <data_loop_values> of a <data_loop> production *must* be an integer multiple of the number of data names in the associated <data_loop_field>.

The STAR syntax specified in the EBNF follows.

A2.1.1.1. Lexical tokens

We accept a space, a horizontal and a vertical tab as <blank>.

```
<blank> ::= ' '| '\t' | '\v'  (ASCII 32 09 11)
```

The non-printing single ASCII characters 10, 12, 13 or any sequence of these are always interpreted as being a single line terminator. In this way there should not be operating-system-dependent ambiguity in those architectures that use character sequences as line terminators. This necessarily requires that these characters can only be used for line termination.

```
<terminate> ::= { '\n' | '\r' | '\f' }+
                (ASCII 10 13 12)
```

We define a 'comment' to be initiated with <blank> or <terminate> and the character #, followed by any sequence of characters (which include <blank>). The only characters not allowed are those in the production <terminate>, and hence these characters terminate a comment. Note the requirement of a leading <blank> or <terminate> is dropped if the # character is the first character in the file.

```
<comment> ::= { <blank> | <terminate> }+ '#' <char>*
```

We accept as white space *all* elements in the above three productions. White spaces are the lexemes able to delimit the lexical tokens. Note that a comment is a legitimate white space because it must end with a line terminator, and hence delimits tokens.

```
<wspace> ::= { <blank> | <terminate> | <comment> }+
```

Non-blank characters are composed of all the characters in our set, excluding <blank> and <terminate> characters.

```
<non_blank_char> ::= <ordinary_char> | '"' | '#' | '$'
                   | '\'' | ';' | '_' | '[' | ']'
```

<char> characters are composed of all the characters in our set, excluding <terminate> characters.

```
<char> ::= <blank> |  <non_blank_char>
```

We define a 'line of text' to be a line contained within a semicolon-bounded text block. Hence the first character *cannot* be a semicolon, and is followed by any number of characters from the set <char> and terminated with a line-termination character or just the termination character. This allows for 'blank' lines in the semicolon-bounded text block.

```
<line_of_text> ::=
     { <not_a_semi_colon> <char>* <terminate>
     | <terminate> }
```

Productions for specific characters.

```
<D_quote>       ::= '"'  (ASCII 34)
<S_quote>       ::= '\'' (ASCII 39)
<semi_colon>    ::= ';'  (ASCII 59)
```

All printable characters *except* the double quote.

```
<not_a_D_quote> ::= <ordinary_char> | '#' | '$' | '\''
                  | ';' | '_' | '[' | ']' | <blank>
```

All printable characters *except* the single quote.

```
<not_an_S_quote> ::= <ordinary_char> | '"' | '#' | '$'
                   | ';' | '_' | '[' | ']' | <blank>
```

All printable characters *except* the left and right square brackets.

```
<not_an_S_bracket> ::= <ordinary_char> | '"' | '#'
                     | '$' | ';' | '_'   | '\''
                     | <blank>
```

All printable characters *except* the semicolon.

```
<not_a_semi_colon> ::= <ordinary_char> | '"' | '#'
                     | '$' | '\'' | '_' | '[' | ']'
                     | <blank>
```

Ordinary characters are all those printable characters that can initiate a non-quoted text string. These exclude the special characters, ", #, $, ' and _ and in some cases ;.

```
<ordinary_char> ::=
       '!' | '%' | '&' | '(' | ')' | '*' | '+' | ','
     | '-' | '.' | '/' | '0' | '1' | '2' | '3' | '4'
     | '5' | '6' | '7' | '8' | '9' | ':' | '<' | '='
     | '>' | '?' | '@' | 'A' | 'B' | 'C' | 'D' | 'E'
     | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M'
     | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U'
     | 'V' | 'W' | 'X' | 'Y' | 'Z' | '\\' | '^' | '`'
     | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h'
     | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p'
     | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x'
     | 'y' | 'z' | '{' | '|' | '}' | '~' | <blank>
```

The keywords (in a case-insensitive form).

```
<DATA_> ::= {'d'|'D'} {'a'|'A'} {'t'|'T'} {'a'|'A'} '_'
<LOOP_> ::= {'l'|'L'} {'o'|'O'} {'o'|'O'} {'p'|'P'} '_'
<GLOBAL_> ::= {'g'|'G'} {'l'|'L'} {'o'|'O'} {'b'|'B'}
              {'a'|'A'} {'l'|'L'} '_'
<STOP_> ::= {'s'|'S'} {'t'|'T'} {'o'|'O'} {'p'|'P'} '_'
<SAVE_> ::= {'s'|'S'} {'a'|'A'} {'v'|'V'} {'e'|'E'} '_'
```

The operating-system-dependent end-of-file marker.

```
<EOF>   ::= end-of-file marker
```

A2.1.1.2. STAR grammar

A STAR File may be an empty file, or it may contain one or more data blocks or global blocks.

```
<STAR_File> ::=
     <wspace>* {<data_block> | <global_block>}*
```

There can be any amount of white spaces (remember <wspace> includes comments) before and at least one white space or an end of file (EOF) after a data or global block. This forces white space between data (and global) blocks in a single file. There must be *at least* one data item in any data or global block. This means a file consisting of just a data or global block heading is *invalid*.