Table 2.2.7.1. *(cont.)*

(*f*) White space and comments.

Syntactic unit	Syntax	Case sensitive?
`<WhiteSpace>`	`{<SP>|<HT>|<eol>|<TokenizedComments>}+`	yes
`<Comments>`	`{ '#' {<AnyPrintChar>}* <eol>}+`	yes
`<TokenizedComments>`	`{<SP>|<HT>|<eol>|}+ <Comments>`	yes

(*g*) Character sets.

Syntactic unit	Syntax	Case sensitive?
`<OrdinaryChar>`	`{ '!'|'%'|'&'|'('|')'|'*'|'+'|','|'-'|'.'|'/'|'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'|'{'|'|'|'}'|'~' }`	yes
`<NonBlankChar>`	`<OrdinaryChar>|<double_quote>|'#'|'$'|<single_quote>|'_' |';'|'['|']'`	yes
`<TextLeadChar>`	`<OrdinaryChar>|<double_quote>|'#'|'$'|<single_quote>|'_'|<SP>|<HT>|'['|']'`	yes
`<AnyPrintChar>`	`<OrdinaryChar>|<double_quote>|'#'|'$'|<single_quote>|'_'|<SP>|<HT>|';'|'['|']'`	yes

2.2.7.1.10. *Version identification*

(34) As an archival file format, the CIF specification is expected to change infrequently. Revised specifications will be issued to accompany each substantial modification. A CIF may be considered compliant against the most recent version for which in practice it satisfies all syntactic and content rules as detailed in the formal specification document. However, to signal the version against which compliance was claimed at the time of creation, or to signal the file type and version to applications (such as operating-system utilities), it is *recommended* that a CIF begin with a structured comment that identifies the version of CIF used. For CIFs compliant with the current specification, the first 11 bytes of the file should be the string

```
#\#CIF_1.1
```

immediately followed by one of the white-space characters permitted in paragraph (22).

2.2.7.2. A formal grammar for CIF

2.2.7.2.1. *Summary*

(35) The rows of Table 2.2.7.1 are called 'productions'. Productions are rules for constructing sentences in a language. They are written in terms of 'terminal symbols' and 'non-terminal symbols'. 'Terminal symbols' are what actually appear in a language. For example, `'poodle'` might be given as a string of terminal symbols in some language discussing dogs. Non-terminal symbols are the higher-level constructs of the language, *e.g.* sentences, clauses, *etc.* For example `<DOG>` might be given as a non-terminal symbol in some language discussing dogs. Productions may be used to infer rules for parsing the language. For example,

```
<DOG> ::= { 'poodle'|'terrier'|'bulldog'|'greyhound' }
```

might be given as a rule telling us what names of types of dogs we are allowed to write in this language. In this table, terminal symbols (*i.e.* terminal character strings) are enclosed in single quotes. To avoid confusion, the terminal symbol consisting of a single quote (*i.e.* an apostrophe) is indicated by `<single_quote>` and the terminal symbol consisting of a double quote is indicated by `<double_quote>`. The printable space character is indicated by `<SP>`, the horizontal tab character by `<HT>` and the end of a line by `<eol>`. To allow for the occurrence of a semicolon as the initial character of an unquoted character string, provided it is not the first character in a line of text, the special symbol `<noteol>` is used below to indicate any character that is not interpretable as a line terminator. The cases of context sensitivity involving the beginning of text fields and the ends of quoted strings are discussed below, but they are most commonly resolved in a lexical scan.

(36) Productions can be used to produce documents, or equivalently to check a document to see if it is valid in this grammar. The angle brackets delimit names for the syntactic units (the 'non-terminal symbols') being defined. The curly braces enclose alternatives separated by vertical bars and/or followed by a plus sign for 'one or more', an asterisk for 'zero or more' or a question mark for 'zero or one'.

(37) In most cases, each production has a single non-terminal symbol in the syntactic unit being defined. However, in some cases, both the syntactic unit and the syntax begin or end with some common symbol. This indicates that a specific context is required in order for the rule to be applied. This is done because the initial semicolon of a semicolon-delimited text field only has meaning at the beginning of a line, and quoted strings may contain their initial quoting character provided the embedded quoting character is not immediately followed by white space. This 'context-sensitive' notation is unusual in defining computer languages (although very common in the full specifications of many computer and non-computer languages). This context-sensitive notation greatly simplifies the definitions and is simple to implement. The formal definitions are elaborated below.

(38) In the present revision, the production for `<TextField>` is a trivial equivalence to `<SemiColonTextField>`. The redundancy is retained to permit possible future extensions to text fields, in particular the possible introduction of a bracket-delimited text value.

2.2.7.2.2. *Explanation of the formal syntax*

Comment: Readers not familiar with the conventions used in describing language grammars may wish to consult various lecture notes on the subject available on the web, *e.g.* Bernstein (2002).

(39) In creating a parser for CIF, the normal process is to first perform a 'lexical scan' to identify 'tokens' in the CIF. A 'token' is a grammatical unit, such as a special character, or a tag or a value, or some major grammatical subunit. In the course of a lexical scan, the input stream is reduced to manageable pieces, so that

the rest of the parsing may be done more efficiently. The convention followed in this document is to mark the 'non-terminal' tokens that are built up out of actual strings of characters or which do not have an immediate representation as printable characters by angle brackets, <>, and to indicate the tokens that are actual strings of characters as quoted strings of characters.

(40) The precise division between a lexical scan and a full parse is a matter of convenience. A suggested division is presented. Before getting to that point, however, there are some highly machine-dependent matters that need to be resolved. There must be a clear understanding of the character set to be used, and of how files and lines begin and end. The character set will be specified in terms of printable characters and a few control characters from the 7-bit ASCII character set. In addition, we will need some means of specifying the end of a line.

(41) The character set in CIF is restricted to the ASCII control characters `<HT>` (horizontal tab, position 09 in the ASCII character set), `<NL>` (newline, position 10 in the ASCII character set, also named `<LF>`) and `<CR>` (carriage return, position 13 in the ASCII character set), and the printable characters in positions 32–126 of the ASCII character set. These are the characters permitted by STAR with the exception of `VT` (vertical tab, position 11 in the ASCII character set) and `FF` (form feed, position 12 in the ASCII character set). In general it is poor practice to use characters that are not common to all national variants of the ISO character set. On systems or in programming languages that do not 'work in ASCII', the characters themselves may have different numeric values and in some cases there is no access to all the control characters.

(42) The `<eol>` token stands for the system-dependent end of line.

Implementation note: CIF implementations may follow common HTML and XML practice in handling `<eol>`:

> '[On many modern systems,] lines are typically separated by some combination of the characters carriage-return (#xD) and line-feed (#xA). To simplify the tasks of applications, the characters passed to an application . . . must be as if the . . . [parser] normalized all line breaks in external parsed entities . . . on input, before parsing, [*e.g.*] by translating both the two-character sequence #xD #xA and any #xD that is not followed by #xA to a single #xA character.'

(From the XML specification http://www.w3.org/TR/2000/REC-xml-20001006.)

Because Unix systems use `\n` (the ASCII LF control character, or #xA), MS Windows systems use `\r\n` (the ASCII CR control character, or #xD, followed by the ASCII LF control character, or #xA) and classic MacOS systems use `\r`, a parser which covers a wide range of systems in a reasonable manner could be constructed using a pseudo-production for `<eol>` such as

```
<eol> ::= { <LF> | <CR><LF> | <CR> }
```

provided the supporting infrastructure (such as the lexer) deals with the necessary minor adjustment to ensure that each end of line is recognized and that all end-of-line control characters are filtered out from the portions of the text stream that are to be processed by other productions. One case to handle with care is the end-of-document case. It is not uncommon to encounter a last line in a document that is not terminated by any of the above-mentioned control characters. Instead, it may be terminated by the end of the character stream or by a special end-of-text-document control character [*e.g.* #x4 (control-D) or #x1A (control-Z)]. A CIF parser should normalize such unterminated terminal lines to appear to an application as if they had been properly terminated. On the other hand, care should also be taken so that in multiple generations of CIF processing such processing does not result in an ever-growing 'tail' of empty lines at the end of a CIF document.

This discussion is *not* meant to imply that a parser for a system that uses one of these line-termination conventions must recognize a CIF written using another of these line-termination conventions.

This discussion is *not* meant to imply that parsers on systems that use other line-termination conventions and/or non-ASCII character sets need to handle these ASCII control characters.

In processing a valid CIF document, it is always sufficient that a parser be able to recognize the line-termination conventions of text files local to its system environment, and that it be able to recognize the local translations of `<SP><HT>` and the printable characters used to construct a CIF.

However, when circumstances permit, if a parser is able to recognize 'alien' line terminations, it is permissible for the parser to accept and process the CIF in that form without treating it as an error.

In writing CIF documents, the software that emits lines should follow the text-file line-termination conventions of the target system for which it is writing the CIF documents, and not mix conventions from multiple systems. In transmitting a CIF document from system to system, software should be used that causes the document to conform to the line-termination conventions of the target system. In most cases this objective can best be achieved by using 'text' or 'ascii' transmission modes, rather than 'binary' or 'image' transmission modes.

(43) In order to write the grammar, we need a way to refer to the single-quote characters which we use both to quote within the syntax and to quote within a CIF. To avoid system-dependent confusion, we define the following special tokens:

Token	Meaning
`<SP>`	' ', the printable space character
`<HT>`	the horizontal-tab character on the system
`<eol>`	the machine-dependent end of line
`<noteol>`	the complement of the above; any character that does not indicate the machine-dependent end of line
`<single_quote>`	the apostrophe, `'`
`<double_quote>`	the double-quote character, `"`

(44) There are CIF specifications not definable directly in a context-free Backus–Naur form (BNF). Restrictions in record and data-name lengths, and the parsing of text fields and quoted character strings are best handled in the initial lexical scan. A pure BNF can then be used to parse the tokenized input stream.

2.2.7.3. Lexical tokens

(45) We define a 'comment' to be initiated with the character `#`. This can be followed by any sequence of characters (which include `<SP>` or `<HT>`). The only characters not allowed are those in the production `<eol>`, which `<eol>` terminates a comment. A comment is recognized only at the beginning of a line or after blanks, *i.e.* only after space, tab or `<eol>`. For this reason we define both comments and 'tokenized comments'. No portion of the essential machine-readable content within a CIF is conveyed by the comments. Comments are for the convenience of human readers of CIFs and may be freely introduced or removed. Note however the optional structured comment sanctioned in paragraph (34) above, which has the purpose of indicating the file type and revision level to general-purpose file-handling software.

```
<Comments>          ::= { '#' {<AnyPrintChar>}* <eol>}+
<TokenizedComments> ::= { <SP>|<HT>|<eol> }+ <Comments>
```