Table 2.2.7.1. *A formal grammar for CIF*

(*a*) Basic structure of a CIF.

Syntactic unit	Syntax	Case sensitive?
<CIF>	<Comments>? <WhiteSpace>? {<DataBlock> {<WhiteSpace> <DataBlock> }* {<WhiteSpace> }? }?	yes
<DataBlock>	<DataBlockHeading> {<WhiteSpace> { <DataItems> | <SaveFrame>} }*	yes
<DataBlockHeading>	<DATA_> {<NonBlankChar> }+	no
<SaveFrame>	<SaveFrameHeading> {<WhiteSpace> <DataItems> }+ <WhiteSpace> <SAVE_>	yes
<SaveFrameHeading>	<SAVE_> {<NonBlankChar> }+	no
<DataItems>	<Tag> <WhiteSpace> <Value> | <LoopHeader> <LoopBody>	yes
<LoopHeader>	<LOOP_> {<WhiteSpace> <Tag>}+	no
<LoopBody>	<Value> {<WhiteSpace> <Value> }*	yes

(*b*) Reserved words.

Syntactic unit	Syntax	Case sensitive?
<DATA_>	{'D'|'d'} {'A'|'a'} {'T'|'t'} {'A'|'a'} '_'	no
<LOOP_>	{'L'|'l'} {'O'|'o'} {'O'|'o'} {'P'|'p'} '_'	no
<GLOBAL_>	{'G'|'g'} {'L'|'l'} {'O'|'o'} {'B'|'b'} {'A'|'a'} {'L'|'l'} '_'	no
<SAVE_>	{'S'|'s'} {'A'|'a'} {'V'|'v'} {'E'|'e'} '_'	no
<STOP_>	{'S'|'s'} {'T'|'t'} {'O'|'o'} {'P'|'p'} '_'	no

(*c*) Tags and values.

Syntactic unit	Syntax	Case sensitive?
<Tag>	'_'{<NonBlankChar>}+	no
<Value>	{'.' | '?' | <Numeric> | <CharString> | <TextField> }	yes

(*d*) Numeric values.

Syntactic unit	Syntax	Case sensitive?
<Numeric>	{<Number> | <Number> '(' <UnsignedInteger> ')' }	no
<Number>	{<Integer> | <Float> }	no
<Integer>	{{ '+' | '-' }? <UnsignedInteger>	no
<Float>	{ <Integer><Exponent> | { {'+'|'-'} ? { {<Digit>} * '.' <UnsignedInteger> } | { <Digit>} + '.' } } {<Exponent>} ? } }	no
<Exponent>	{ {'e' | 'E' } | {'e' | 'E' } { '+' | '-' } } <UnsignedInteger>	no
<UnsignedInteger>	{<Digit> }+	no
<Digit>	{ '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' }	no

(*e*) Character strings and text fields.

Syntactic unit	Syntax	Case sensitive?
<CharString>	<UnquotedString> | <SingleQuotedString> | <DoubleQuotedString>	yes
<eol><UnquotedString>	<eol><OrdinaryChar> {<NonBlankChar>}*	yes
<noteol><UnquotedString>	<noteol>{<OrdinaryChar>|';'} {<NonBlankChar>}*	yes
<SingleQuotedString><WhiteSpace>	<single_quote>{<AnyPrintChar>}* <single_quote> <WhiteSpace>	yes
<DoubleQuotedString><WhiteSpace>	<double_quote> {<AnyPrintChar>}* <double_quote> <WhiteSpace>	yes
<TextField>	{<SemiColonTextField> }	yes
<eol><SemiColonTextField>	<eol>';' { {<AnyPrintChar>}* <eol> {{<TextLeadChar> {<AnyPrintChar>}*}? <eol>}* } ';'	yes

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