2.2.7.3.1. *CIF grammar*

(58) A CIF may be an empty file, or it may contain only comments or white space, or it may contain one or more data blocks. Comments before the first block are acceptable, and there must be white space between blocks.

```
<CIF> ::= <Comments>? <WhiteSpace>?
          { <DataBlock>
             { <WhiteSpace> <DataBlock> }*
             { <WhiteSpace> }?
          }?
```

(59) For a data block, there must be a data heading and zero or more data items or save frames.

```
<DataBlock> ::= <DataBlockHeading>
                  { <WhiteSpace>
                    { <DataItems> | <SaveFrame> }
                  }*
```

(60) A data-block heading consists of the five characters **data_** (case-insensitive) immediately followed by at least one non-blank character selected from the set of ordinary characters or the non-quote-mark, non-blank printable characters.

```
<DataBlockHeading> ::= <DATA_> { <NonBlankChar> }+
```

(61) For a save frame, there must be a save-frame heading, some data items and then the reserved word **save_**.

```
<SaveFrame> ::= <SaveFrameHeading>
                {<WhiteSpace> <DataItems>}+
                <WhiteSpace> <SAVE_>
```

(62) A save-frame heading consists of the five characters **save_** (case-insensitive) immediately followed by at least one non-blank character selected from the set of ordinary characters or the non-quote-mark, non-blank printable characters.

```
<SaveFrameHeading> ::= <SAVE_> { <NonBlankChar> }+
```

(63) Data come in two forms:

(i) A data-name tag separated from its associated value by a `<WhiteSpace>`.

(ii) Looped data. The number of values in the body must be a multiple of the number of tags in the header.

```
<DataItems>  ::= <Tag> <WhiteSpace> <Value> |
                 <LoopHeader> <LoopBody>
<LoopHeader> ::= <LOOP_> { <WhiteSpace> <Tag> }+
<LoopBody>   ::= <Value> { <WhiteSpace> <Value> }*
```

2.2.7.4. Common semantic features

2.2.7.4.1. *Introduction*

(1) The Crystallographic Information File (CIF) standard is an extensible mechanism for the archival and interchange of information in crystallography and related structural sciences. Ultimately CIF seeks to establish an ontology for machine-readable crystallographic information – that is, a collection of statements providing the relations between concepts and the logical rules for reasoning about them.

Essential components in the development of such an ontology are:

(*a*) the basic rules of grammar and syntax, described in Sections 2.2.7.1 to 2.2.7.3;

(*b*) a vocabulary of the tags or data names specifying particular objects;

(*c*) a taxonomy, or classification scheme relating the specified objects;

(*d*) descriptions of the attributes and relationships of individual and related objects.

In the CIF framework, the objects of discourse are described in so-called data dictionary files that provide the vocabulary and taxonomic elements. The dictionaries also contain information about the relationships and attributes of data items, and thus encapsulate most of the semantic content that is accessible to software. In practice, different dictionaries exist to service different domains of crystallography and a CIF that conforms to a specific dictionary must be interpreted in terms of the semantic information conveyed in that dictionary.

However, some common semantic features apply across all CIF applications, and the current document outlines the foundations upon which other dictionaries may build more elaborate taxonomies or informational models.

2.2.7.4.2. *Definition of terms*

(2) The definitions of Section 2.2.7.1.2 also hold for this part of the specification.

2.2.7.4.3. *Semantics of data items*

(3) While the STAR File syntax allows the identification and extraction of tags and associated values, the interpretation of the data thus extracted is application-dependent. In CIF applications, formal catalogues of standard data names and their associated attributes are maintained as external reference files called data dictionaries. These dictionary files share the same structure and syntax rules as data CIFs.

(4) At the current revision, two conventions (known as dictionary definition languages or DDLs) are supported for detailing the meaning and associated attributes of data names. These are known as DDL1 (Hall & Cook, 1995) and DDL2 (Westbrook & Hall, 1995), and they differ in the amount of detail they carry about data types, the relationships between specific data items and the large-scale classification of data items.

(5) While it may be formally possible to define the semantics of the data items in a given data file in both DDL1 and DDL2 data dictionaries, in practice different dictionaries are constructed to define the data names appropriate for particular crystallographic applications, and each such dictionary is written in DDL1 or DDL2 formalism according to which appears better able to describe the data model employed. There is thus in practice a bifurcation of CIF into two dialects according to the DDL used in composing the relevant dictionary file. However, the use of aliases may permit applications tuned to one dialect to import data constructed according to the other.

2.2.7.4.4. *Data-name semantics*

(6) Strictly, data names should be considered as void of semantic content – they are tags for locating associated values, and all information concerning the meaning of that value should be sought in an associated dictionary.

(7) However, it is customary to construct data names as a sequence of components elaborating the classification of the item within the logical structure of its associated dictionary. Hence a data name such as **_atom_site_fract_x** displays a hierarchical arrangement of components corresponding to membership of nested groupings of data elements. The choice of components readily indicates to a human reader that this data item refers to the fractional *x* coordinate of an atomic site within a crystal unit

cell, but it should be emphasized from a computer-programming viewpoint that this is coincidental; the attributes that constrain the value of this data item (and its relationship to others such as `_atom_site_fract_y` and `_atom_site_fract_z`) must be obtained from the dictionary and not otherwise inferred.

(8) *Comment:* In practice data names described in a DDL2 dictionary are constructed with a period character separating their specific function from the name of the category to which they have been assigned. In the absence of a dictionary file, this convention permits the inference that the data item with name `_atom_site.fract_x` will appear in the same looped list as other items with names beginning `_atom_site.`, and that all such items belong to the same category.

2.2.7.4.5. *Name space*

(9) The intention of the maintainers of public CIF dictionaries is to formulate a single authoritative set of data names for each CIF dialect (*i.e.* DDL1 and DDL2), thus facilitating the reliable archive and interchange of crystallographic data. However, it is also permissible for users to introduce local data names into a CIF. Two mechanisms exist to reduce the danger of collision of data names that are not incorporated into public dictionaries.

(10) The character string `[local]` (including the literal bracket characters) is *reserved* for local use. That is, no public dictionary will define a data name that includes this string. This allows experimentation with data items in a strictly local context, *i.e.* in cases where the CIF is not intended for interchange with any other user.

(11) Where CIFs including local data items are expected to enjoy a public circulation, authors may register a *reserved prefix* for their sole use. The registry is available on the web at http://www.iucr.org/iucr-top/cif/spec/reserved.html.

A reserved prefix, *e.g. foo*, must be used in the following ways:

(i) If the data file contains items defined in a DDL1 dictionary, the local data names assigned under the reserved prefix must contain it as their first component, *e.g.* `_foo_atom_site_my_item`.

(ii) If the data file contains items defined in a DDL2 dictionary, then the reserved prefix must be:

(*a*) the first component of data names in a category defined for local use, *e.g.* `_foo_my_category.my_item`.

(*b*) the first component following the period character in a data name describing a new item in a category already defined in a public dictionary, *e.g.* `_atom_site.foo_my_item`.

(12) There is no syntactic property identifying such a reserved prefix, so that software validating or otherwise handling such local data names must scan the entire registry and match registered prefixes against the indicated components of data names. Note that reserved prefixes may not themselves contain underscore characters.

2.2.7.4.6. *Note on handling of units*

(13) The published specification for CIF version 1.0 permitted data values expressed in different units to be tagged by variant data names (Hall *et al.*, 1991, p. 657):

> ... Many numeric fields contain data for which the units must be known. Each CIF data item has a default units code which is stated in the CIF Dictionary. If a data item is not stored in the default units, the units code is appended to the data name. For example, the default units for a crystal cell dimension are ångströms. If it is necessary to include this data item in a CIF with the units of picometres, the data name of `_cell_length_a` is replaced by `_cell_length_a_pm`. Only those units defined in the CIF Dictionary are acceptable. The default units, except for the ångström, conform to the SI Standard adopted by the IUCr.

This approach is deprecated and has not been supported by any official CIF dictionary published subsequent to version 1.0 of the core. All data values must be expressed in the single unit assigned in the associated dictionary.

A small number of archived CIFs exist with variant data names as permitted by the above clause. If it is necessary to validate them against versions of the core dictionary subsequent to version 1.0, the formal compatibility dictionary cif_compat.dic (ftp://ftp.iucr.org/cifdics/cif_compat.dic) may be used for the purpose. *No other use should be made of this dictionary.*

2.2.7.4.7. *Data-value semantics*

(14) The STAR syntax permits retrieval of data by simply requesting a specific data name within a specific data block. Prior knowledge about data type (*e.g.* text or numbers), whether the item is looped or whether the item exists in the file at all is unnecessary. However, applications in general need to know data type, valid ranges of values and relationships between data items, and a program designer needs to know the purpose of the data item (*i.e.* what physical quantity or internal book-keeping function it represents). While such semantic information may be defined informally for local data items (ones not intended for exchange between different users or software applications), formal descriptions of the semantics associated with data values are catalogued in data dictionary files. Currently two formalisms (dictionary definition languages) for describing data-value attributes are supported; full specifications of these formalisms (known as DDL1 and DDL2) are provided in Chapters 2.5 and 2.6.

2.2.7.4.7.1. *Data typing*

(15) Four base data types are supported in CIF. These are:

(i) **numb**: a value interpretable as a decimal base number and supplied as an integer, a floating-point number or in scientific notation;

(ii) **char**: a value to be interpreted as character or text data (where the value contains white-space characters, it must be quoted);

(iii) **uchar**: a value to be interpreted as character or text data but in a case-insensitive manner (*i.e.* the values `FOO` and `foo` are to be taken as identical);

(iv) **null**: a special data type associated with items for which no definite value may be stored in computer memory. It is the type associated with the special character literal values `?` (query mark) and `.` (full point), which may appear as values for any data item within a data file (see Section 2.2.7.4.8 below). It is also the type assigned to items defined in dictionary files that may not occur in data files.

(16) *Comment:* Many applications distinguish between multi-line text fields and character-string values that fit within a single line of text. While this is a convenient practical distinction for coding purposes, formally both manifestations should be regarded as having the same base type, which might be 'char' or 'uchar'. Applications are at liberty to choose whether to define specific multi-line text subtypes, and whether to permit casting between subtypes of a base type. The examples of character-string delimiters in Section 2.2.7.1.4(20) are predicated on an approach that handles all subtypes of character or text data equivalently.

(17) Where the attributes of a data value are not available in a dictionary listing, it may be assumed that a character string inter-

pretable as a number should be taken to represent an item of type 'numb'. However, an explicit dictionary declaration of type will override such an assumption.

2.2.7.4.7.2. *Subtyping*

(18) The base data types detailed in the previous section are very general and need to be refined for practical application. Refinement of types is to some extent application-dependent, and different subtypes are supported for data items defined by DDL1 and DDL2 dictionary files. The following notes indicate some considerations, but the relevant dictionary files and documentation should be consulted in each case.

(19) *DDL1 dictionaries*. Values of type 'numb' may include a standard uncertainty in the final digit(s) of the number where the associated item definition includes the attribute

```
_type_conditions    esd
```

(or `_type_conditions su`, a synonym introduced to DDL1 in 2005). For example, a value of 34.5(12) means 34.5 with a standard uncertainty of 1.2; it may also be expressed in scientific notation as 3.45E1(12).

(20) *DDL2 dictionaries*. DDL2 provides a number of tags that may be used in a dictionary file to specify subtypes for data items defined by that dictionary alone. Examples of the subtypes specified for the macromolecular CIF dictionary are:

code	identifying code strings or single words
ucode	identifying code strings or single words (case-insensitive)
uchar1	single-character codes (case-insensitive)
uchar3	three-character codes (case-insensitive)
line	character strings forming a single line of text
uline	character strings forming a single line of text (case-insensitive)
text	multi-line text
int	integers
float	floating-point real numbers
yyyy-mm-dd	dates
symop	symmetry operations
any	any type permitted

2.2.7.4.8. *Special generic values*

(21) The unquoted character literals `?` (query mark) and `.` (full point) are special and are valid expressions for any data type.

(22) The value `?` means that the actual value of a requested data item is *unknown*.

(23) The value `.` means that the actual value of a requested data item is *inapplicable*. This is most commonly used in a looped list where a data value is required for syntactic integrity.

2.2.7.4.9. *Embedded data semantics*

(24) The attributes of data items defined in CIF dictionaries serve to direct crystallographic applications in the retrieval, storage and validation of relevant data. In principle, a CIF might include as data items suitably encoded fields representing data suitable for manipulation by text processing, image, spreadsheet, database or other applications. It would be useful to have a formal mechanism allowing a CIF to invoke appropriate content handlers for such data fields; this is under investigation for the next CIF version specification.

2.2.7.4.10. *CIF conventions for special characters in text*

(25) The one existing example of embedded semantics is the text character markup introduced in the CIF version 1.0 specification and summarized in paragraphs (30)–(37) below. The specification is silent on which fields should be interpreted according to these markup conventions, but the published examples suggest that they may be used in any character field in a CIF data file except as prohibited by a dictionary directive. It is intended that the next CIF version specification shall formally declare where such markup may be used.

2.2.7.4.11. *Handling of long lines*

(26) The restriction in line length within CIF requires techniques to handle without semantic loss the content of lines of text exceeding the limit (2048 characters in this revision, 80 characters in the initial CIF specification). The line-folding protocol defined here provides a general mechanism for wrapping lines of text within CIFs to any extent within the overall line-length limit. A specific application where this would be useful is the conversion of lines longer than 80 characters to the CIF version 1.0 limit. This 80-character limit is used in the examples below for illustrative purposes.

These techniques are applied only to the contents of text fields and to comments.

In order to permit such folding, a special semantics is defined for use of the backslash. It is important to understand that this does not change the syntax of CIF version 1.0. All existing CIFs conforming to the CIF version 1.0 specification can be viewed as having exactly the same semantics as they now have. Use of these transformational semantics is optional, but recommended.

In order to avoid confusion between CIFs that have undergone these transformations and those that have not, the special comment beginning with a hash mark immediately followed by a backslash (`#\`) as the last non-blank characters on a line is reserved to mark the beginning of comments created by folding long-line comments, and the special text field beginning with the sequence line termination, semicolon, backslash (`<eol>;\`) as the only non-blank characters on a line is reserved to mark the beginning of text fields created by folding long-line text fields.

The backslash character is used to fold long lines in character strings and comments. Consider a comment which extends beyond column 80. In order to provide a comment with the same meaning which can be fitted into 80-character lines, prefix the comment with the special comment consisting of a hash mark followed by a backslash (`#\`) and the line terminator. Then on new lines take appropriate fragments of the original comment, beginning each fragment with a hash mark and ending all but the last fragment with a backslash. In doing this conversion, check for an original line that ends with a backslash followed only by blanks or tabs. To preserve that backslash in the conversion, add another backslash after it. If the next lexical token (not counting blanks or tabs) is another comment, to avoid fusing this comment with the next comment, be sure to insert a line with just a hash mark.

Similarly, for a character string that extends beyond column 80,

(i) first convert it to be a text field delimited by line termination–semicolon (`<eol>;`) sequences,

(ii) then change the initial line termination–semicolon (`<eol>;`) sequence to line termination–semicolon–backslash–line termination (`<eol>;\<eol>`),

(iii) and break all subsequent lines that do not fit within 80 columns with a trailing backslash. In the course of doing the translation,

(*a*) check for any original text lines that end with a backslash followed only by blanks or tabs;

(*b*) to preserve that backslash in the conversion, add another backslash after it, and then an empty line.

(More formally, the line folding should be done separately and directly on single-line non-semicolon-delimited character strings

to allow for recognition of the fact that no terminal line termination is intended – see below.)

In order to understand this scheme, suppose the CIF fragment (1) below were considered to have long lines. They could be transformed into (2) as follows:

(1) *Initial CIF*

```
#########################################################
### CIF submission form for Rietveld refinements
###
###                          Version 14 December 1998
###
#########################################################
data_znvodata
_chemical_name_systematic
; zinc dihydroxide divanadate dihydrate
;

_chemical_formula_moiety      'H2 O9 V2 Zn3, 2(H2 O)'
_chemical_formula_sum         'H6 O11 V2 Zn3'
_chemical_formula_weight      480.05
```

(2) *Transformed CIF*

```
#\
############################\
############################
### CIF submission form for Rietveld refinements
###
###                          Version 14 December 1998
###
#########################################################
data_znvodata
_chemical_name_systematic
;\
 zinc dihydroxide divan\
adate dihydrate
;

_chemical_formula_moiety
;\
H2 O9 V2 Zn3, 2(H2 O)\
;
_chemical_formula_sum         'H6 O11 V2 Zn3'
_chemical_formula_weight      480.05
```

In making the transformation from the backslash-folded form to long lines, it is very important to strip trailing blanks before attempting to recognize a backslash as the last character. When reassembling text-field lines, no reassembly should be done except in text fields that begin with the special sequence described above, line termination–semicolon–backslash–line termination, (`<eol>;\<eol>`), so that text fields that happen to contain backslashes but which were not created by folding long lines are not changed. It is also important to remove the trailing backslashes when reassembling long lines. The final line termination–semicolon sequence of a text field takes priority over the reassembly process and ends it, but a trailing backslash on the last line of a text field very nicely conveys the information that no trailing line termination is intended to be included within the character string.

Similarly, when reassembling long-line comments, the reassembly begins with a comment of the form hash–backslash–line termination. The initial hash mark is retained and then a forward scan is made through line terminations and blanks for the next comment, from which the initial hash mark is stripped and then the contents of the comment are appended. If that comment ends with a backslash, the trailing backslash is stripped and the process repeats. Note that the process will be ended by intervening tags, values, data blocks or other non-white-space information, and that the process will not start at all without the special hash–backslash–line termination comment.

Since there are very few, if any, CIFs that contain text fields and comments beginning this way, in most cases it is reasonable to adopt the policy of doing this processing unless it is disabled.

Here is another example of folding. The following three text fields would be equivalent:

```
;C:\foldername\filename
;

;\
C:\foldername\filename
;
```

and

```
;\
C:\foldername\file\
name
;
```

but the following example would be a two-line value where the first line had the value `C:\foldername\file\` and the second had the value `name`:

```
;
C:\foldername\file\
name
;
```

Note that backslashes should not be used to fold lines outside of comments and text fields. That would introduce extraneous characters into the CIF and violate the basic syntax rules. In any case, such action is not necessary.

2.2.7.4.12. *Dictionary compliance*

(27) Dictionary files containing the definitions and attribute sets for the data items contained in a CIF should be identified within the CIF by some or all of the data items

```
_audit_conform_dict_name
_audit_conform_dict_version
_audit_conform_dict_location
```

corresponding to DDL1 dictionaries or

```
_audit_conform.dict_name
_audit_conform.dict_version
_audit_conform.dict_location
```

for DDL2 dictionaries. Where no such information is provided, it may be assumed that the file should conform against the core CIF dictionary.

(28) The `_audit_conform` data items may be looped in cases where more than one dictionary is used to define the items in a CIF and they may include dictionaries of local data items provided such dictionary files have been prepared in accordance with the rules of the appropriate DDL.

(29) A detailed protocol exists for locating, merging and overlaying multiple dictionary files (McMahon *et al.*, 2000) (see Section 3.1.9).

2.2.7.4.13. *CIF markup conventions*

(30) If permitted by the relevant dictionary and if no other indication is present, the contents of a text or character field are assumed to be interpretable as text in English or some other human language. Certain special codes are used to indicate special characters or accented letters not available in the ASCII character set, as listed below.

2.2.7.4.14. *Greek letters*

(31) In general, the corresponding letter of the Latin alphabet, prefixed by a backslash character. The complete set is:

α	A	\a	\A	alpha	ν	N	\n	\N	nu
β	B	\b	\B	beta	o	O	\o	\O	omicron
χ	X	\c	\C	chi	π	Π	\p	\P	pi
δ	Δ	\d	\D	delta	θ	Θ	\q	\Q	theta
ε	E	\e	\E	epsilon	ρ	R	\r	\R	rho
φ	Φ	\f	\F	phi	σ	Σ	\s	\S	sigma
γ	Γ	\g	\G	gamma	τ	T	\t	\T	tau
η	H	\h	\H	eta	υ	U	\u	\U	upsilon
ι	I	\i	\I	iota	ω	Ω	\w	\W	omega
κ	K	\k	\K	kappa	ξ	Ξ	\x	\X	xi
λ	Λ	\l	\L	lambda	ψ	Ψ	\y	\Y	psi
μ	M	\m	\M	mu	ζ	Z	\z	\Z	zeta

2.2.7.4.15. *Accented letters*

(32) Accents should be indicated by using the following codes before the letter to be modified (*i.e.* use `\'e` for an acute e):

`\'`	acute (é)	`\"`	umlaut (ü)
`\=`	overbar or macron (ā)	`\``	grave (à)
`\~`	tilde (ñ)	`\.`	overdot (ȯ)
`\^`	circumflex (â)	`\;`	ogonek (ų)
`\<`	hacek or caron (ǒ)	`\,`	cedilla (ç)
`\>`	Hungarian umlaut or double accented (ő)	`\(`	breve (ŏ)

These codes will always be followed by an alphabetic character.

2.2.7.4.16. *Other characters*

(33) Other special alphabetic characters should be indicated as follows:

`\%a`	a-ring (å)	`\?i`	dotless i (ı)	`\&s`	German 'ss' (ß)
`\/o`	o-slash (ø)	`\/l`	Polish l (ł)	`\/d`	barred d (đ)

Capital letters may also be used in these codes, so an ångström symbol (Å) may be given as `\%A`.

(34) Superscripts and subscripts should be indicated by bracketing relevant characters with circumflex or tilde characters, thus:

superscripts	`Csp^3^`	for	Csp^3
subscripts	`U~eq~`	for	U_{eq}

The closing symbol is essential to return to normal text.

(35) Some other codes are accepted by convention. These are:

`\%`	degree (°)	`\\times`	×
`--`	dash	`+-`	±
`---`	single bond	`-+`	∓
`\\db`	double bond	`\\square`	□
`\\tb`	triple bond	`\\neq`	≠
`\\ddb`	delocalized double bond	`\\rangle`	>
`\\sim`	∼	`\\langle`	<
(*Note:* `~` is the code for subscript)		`\\rightarrow`	→
`\\simeq`	≃	`\\leftarrow`	←
`\\infty`	∞		

Note that `\\db`, `\\tb` and `\\ddb` should always be followed by a space, *e.g.* C=C is denoted by `C\\db C`.

2.2.7.4.17. *Typographic style codes*

(36) The codes indicated above are designed to refer to special characters not expressible within the CIF character set, and the initial specification did not permit markup for typographic style such as italic or bold-face type. However, in some cases the ability to indicate type style is useful, and in addition to the codes above HTML-like conventions are allowed of surrounding text by `<i> </i>` to indicate the beginning and end of italic, and by `<b> </b>` to indicate the beginning and end of bold-face type.

(37) If it is necessary to convey more complex typographic information than is permitted by these special character codes and conventions, the entire text field should be of a richer content type allowing detailed typographic markup.

References

Bernstein, H. J. (2002). *Some comments on parsing for computer programming languages.* http://www.bernstein-plus-sons.com/TMM/Parsing.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. (1999). *Hypertext Transfer Protocol – HTTP/1.1.* RFC 2616. Network Working Group. http://www.w3.org/Protocols/rfc2616/rfc2616.html.

Freed, N. & Borenstein, N. (1996). *Multipurpose Internet Mail Extensions (MIME) Part two: media types.* RFC 2046. Network Working Group. http://www.ietf.org/rfc/rfc2046.txt.

Hall, S. R. (1991). *The STAR File: a new format for electronic data transfer and archiving. J. Chem. Inf. Comput. Sci.* **31**, 326–333.

Hall, S. R., Allen, F. H. & Brown, I. D. (1991). *The Crystallographic Information File (CIF): a new standard archive file for crystallography. Acta Cryst.* A**47**, 655–685.

Hall, S. R. & Cook, A. P. F. (1995). *STAR dictionary definition language: initial specification. J. Chem. Inf. Comput. Sci.* **35**, 819–825.

Hall, S. R. & Spadaccini, N. (1994). *The STAR File: detailed specifications. J. Chem. Inf. Comput. Sci.* **34**, 505–508.

McMahon, B., Westbrook, J. D. & Bernstein, H. J. (2000). *Report of the COMCIFS Working Group on Dictionary Maintenance.* http://www.iucr.org/iucr-top/cif/spec/dictionaries/maintenance.html.

Ulrich, E. L. *et al.* (1998). XVIIth Intl Conf. Magn. Res. Biol. Systems. Tokyo, Japan.

Westbrook, J. D. & Hall, S. R. (1995). *A dictionary description language for macromolecular structure.* http://ndbserver.rutgers.edu/mmcif/ddl/.