2.2.6. CIF metadata and dictionary compliance

The development of several CIF dictionaries and fields of application has rapidly progressed beyond the specific purpose of describing a small-molecule or inorganic crystal structure for which CIF was devised. With these, a variety of application-specific metadata approaches have evolved to characterize the role of a particular CIF within a family of possible applications. These approaches use data definitions in dictionaries in which enumerated codes identify the file relationships. The mmCIF dictionary (see Chapter 3.6) allows informal identification of 'external reference files' which act as libraries of standard molecular geometry. The pdCIF dictionary (see Chapter 3.3) specifies identifiers that may be included within data blocks of external files containing calibration results. It is the responsibility of the file users to manage a lookup table or database between the referenced identifiers and the location of the files to which they pertain.

Two categories of data items currently exist in the core dictionary to allow a file to indicate its relationship to CIF dictionaries and other data files. (Equivalent categories are also present in the mmCIF dictionary.) AUDIT_CONFORM is a category of data names identifying the dictionaries that hold definitions of the data names in the current CIF. Particularly where the referenced dictionaries include any of the various public dictionaries described in Part 3 of this volume, this serves to establish the discipline within the broad fields of crystallography, structural biology and structural chemistry to which the data are most relevant.

The category AUDIT_LINK allows an informal textual description of the relationship between the data blocks within the current file. It is 'informal' in the sense that the relevant data items are free-text in nature. It would surely be useful to have a catalogue of more specific designations to allow automated software to track such relationships as the separate reference and modulated structures in an incommensurate compound, or the multiple trial refinements of a protein structure. The challenge is to determine and classify such standard relationships between data blocks.

In the future it is hoped that a common approach to metadata will be developed to enable all CIF instantiations to be uniquely identified and interrelated. Development of standard descriptions of the relationships between structural entities of this sort (reference geometries, calibration results, partial refinements, modulated superposed structures *etc.*) will be an important stage in the formalization of complete CIF metadata, and will become an important step towards categorization of data entities needed for interoperability between different file formats and across a wide range of scientific disciplines.

2.2.7. Formal specification of the Crystallographic Information File

Version 1.1 specification

BY S. R. HALL, N. SPADACCINI, I. D. BROWN, H. J. BERNSTEIN, J. D. WESTBROOK AND B. MCMAHON

This section presents the documents *File syntax* (Sections 2.2.7.1–3) and *Common semantic features* (Section 2.2.7.4) that together comprise the formal CIF specification as approved by COMCIFS.

2.2.7.1. Syntax

2.2.7.1.1. *Introduction*

(1) This document describes the full syntax of the Crystallographic Information File (CIF).

2.2.7.1.2. *Definition of terms*

(2) The following terms are used in the CIF specification documents with the specific meanings indicated here.

(2.1) A **CIF** is a file conforming to the specification herein stated, containing either information on a crystallographic experiment or its results (or similar scientific content), or descriptions of the data identifiers in such a file.

(2.2) A **data file** is understood to convey information relating to a crystallographic experiment.

(2.3) A **dictionary file** is understood to contain information about the data items in one or more data files as identified by their data names.

(2.4) A **data name** is a case-insensitive identifier (a string of characters beginning with an underscore character) of the content of an associated data value.

(2.5) A **data value** is a string of characters representing a particular item of information. It may represent a single numerical value; a letter, word or phrase; extended discursive text; or in principle any coherent unit of data such as an image, audio clip or virtual-reality object.

(2.6) A **data item** is a specific piece of information defined by a data name and an associated data value.

(2.7) A **tag** is understood in this document to be a synonym for data name.

(2.8) A **data block** is the highest-level component of a CIF, containing data items or save frames. A data block is identified by a **data-block header**, which is an isolated character string (that is, bounded by white space and not forming part of a data value) beginning with the case-insensitive reserved characters `data_`.

(2.9) A **block code** is the variable part of a data-block header, *e.g.* the string `foo` in the header `data_foo`.

(2.10) A **save frame** is a partitioned collection of data items within a data block, started by a **save-frame header**, which is an isolated character string beginning with the case-insensitive reserved characters `save_`, and terminated with an isolated character string containing only the case-insensitive reserved characters `save_`.

(2.11) A **frame code** is the variable part of a save-frame header, *e.g.* the string `foo` in the header `save_foo`.

2.2.7.1.3. *File syntax*

(3) The syntax of CIF is a proper subset of the syntax of STAR Files as described by Hall (1991) and Hall & Spadaccini (1994). The general structure is described below in Section 2.2.7.1.4 and a number of subsections list specific restrictions to the STAR syntax that are in force within CIF. A formal language grammar using computer-science notation is included as Section 2.2.7.2.

2.2.7.1.4. *General features*

(4) A CIF consists of **data names** (tags) and associated values organized into **data blocks**. A data block may contain **data items** (associated data names and data values) and/or it may contain **save frames**.

(5) **Save frames** may only be used in dictionary files.

Implementation note: At a purely syntactic level there is no way to distinguish between dictionary and data files. (It is also to be noted that not all dictionary files contain save frames.) A fully validating parser must therefore be able to detect the start and termination of save frames, the uniqueness of the frame code within a data block and the uniqueness of data names within a frame code. It is, however, legitimate for an application-based parser designed to handle only the contents of data files to consider the presence of a save frame as an error.

(6) A **data block** begins with the reserved case-insensitive string `data_` followed immediately by the name of the data block, forming a **data-block header**. A **save frame** has a similar structure to a data block, but may not itself contain further save frames. A save frame begins with the reserved case-insensitive string `save_` followed immediately by the name of the save frame, forming a **save-frame header**. Unlike a data block, a save frame also has a marker for the end of the frame in the form of a repetition of the reserved case-insensitive word `save_`, this time without the name of the frame. Save frames may not nest. Within a single CIF, no two data blocks may have the same name; within a single data block no two save frames may have the same name, although a save frame may have the same name as a data block in the same CIF.

(7) A given **data name** (tag) [see (2.4) and (2.7)] may appear no more than once in a given data block or save frame. A tag may be followed by a single value, or a list of one or more tags may be marked by the preceding reserved case-insensitive word `loop_` as the headings of the columns of a table of values. White space is used to separate a data-block or save-frame header from the contents of the data block or save frame, and to separate tags, values and the reserved word `loop_`. Data items (tags along with their associated values) that are not presented in a table of values may be relocated along with their values within the same data block or save frame without changing the meaning of the data block or save frame. Complete tables of values (the table column headings along with all columns of data) may be relocated within the same data block or save frame without changing the meaning of the data block or save frame. Within a table of values, each tag may be relocated along with its associated column of values within the same table of values without changing the meaning of the table of values. In general, each row of a table of values may also be relocated within the same table of values without changing the meaning of the table of values. Combining tables of values or breaking up tables of values would change the meanings, and is likely to violate the rules for constructing such tables of values.

(8) The case-insensitive word `global_`, used in STAR Files to introduce a group of data values with a scope extending to the end of the file, is an additional reserved word in CIF (that is, it may not be used as the unquoted value of any data item).

(9) If a **data value** (2.5) contains white space or *begins* with a character string reserved for a special purpose, it *must* be delimited by one of several sets of special character strings (the choice of which is constrained if the data value contains characters interpretable as marking a new line of text according to the discussion in the following paragraphs). Such a data value will be indicated by the term *non-simple data value*.

(10) A *simple* data value (*i.e.* one which does not contain white space or begin with a special character string) may optionally be delimited by any of the same set of delimiting character strings, *except* for data values that are to be interpreted as numbers.

(11) The special character strings in this context are listed in the following table. The term 'non-simple data values' in this table refers to data values beginning with these special character strings.

Character or string	Role
`_`	identifies data name
`#`	identifies comment
`$`	identifies save-frame pointer
`'`	delimits non-simple data values
`"`	delimits non-simple data values
`[`	reserved opening delimiter for non-simple data values [see (19)]
`]`	reserved closing delimiter for non-simple data values [see (19)]
`;` (at the beginning of a line of text)	delimits non-simple data values
`data_`	identifies data-block header
`save_`	identifies save-frame header or terminator

In addition, the following case-insensitive *reserved words* may not occur as unquoted data values.

Reserved word	Role
`loop_`	identifies looped list of data
`stop_`	reserved STAR word terminating nested loops or loop headers
`global_`	reserved as a STAR global-block header

(12) The complete syntactic description of a numeric data value is included in Section 2.2.7.3(57) under the production (*i.e.* rule for constructing a part of the language) `<Numeric>`.

(13) *Comment:* The base CIF specification distinguishes between character and numeric values [see Section 2.2.7.4(15)]. Particular CIF applications may make more finely grained distinctions within these types. The paragraphs immediately above have the corollary that a data value such as `12` that appears within a CIF may be quoted (*e.g.* `'12'`) *if and only if* it is to be interpreted and stored in computer memory as a character string and not a numeric value. For example `'12'` might legitimately appear as a label for an atomic site, where another alphabetic or alphanumeric string such as `'C12'` is also acceptable; but it may *not* legitimately be used to represent an integer quantity twelve.

(14) Matching single- or double-quote characters (`'` or `"`) may be used to bound a string representing a non-simple data value *provided* the string does not extend over more than one line.

(15) *Comment:* Because data values are invariably separated from other tokens in the file by white space, such a quote-delimited character string may contain instances of the character used to delimit the string *provided* they are not followed by white space. For example, the data item

```
_example  'a dog's life'
```

is legal; the data value is `a dog's life`.

(16) *Comment:* Note that constructs such as

```
'an embedded \' quote'
```

do *not* behave as in the case of many current programming languages, *i.e.* the backslash character in this context does not escape the special meaning of the delimiter character. A backslash preceding the apostrophe or double-quote characters does, however, have special meaning in the context of accented characters (Section 2.2.7.4.15) provided there is no white space immediately following the apostrophe or double-quote character.

(17) The special sequence of end of line followed immediately by a semicolon in column one (denoted '`<eol>;`') may also be used as a delimiter at the beginning and end of a character string comprising a data value. The complete bounded string is called a **text field** and may be used to convey multi-line values. The end of line associated with the closing semicolon does *not* form part of the data value. Within a multi-line text field, leading white space within text lines must be retained as part of the data value; trailing white space on a line may however be elided.

(18) *Comment:* A text field delimited by the `<eol>;` digraph *may not* include a semicolon at the start of a line of text as part of its value.

(19) Matching square-bracket characters, '[' and ']', are *reserved* for possible future introduction as delimiters of multi-line data values. At this revision of the CIF specification, a data value may not begin with an unquoted left square-bracket character '['. (While not strictly necessary, the right square-bracket character ']' is restricted in the same way in recognition of its reserved use as a closing delimiter.)

(20) *Comment:* For example, the data value `foo` may be expressed equivalently as an unquoted string `foo`, as a quoted string `'foo'` or as a text field

```
;foo
;
```

By contrast, the value of the text field

```
; foo
  bar
;
```

is

```
 foo<eol>  bar
```

(where `<eol>` represents an end of line); the embedded space characters are significant.

(21) A comment in a CIF begins with an unquoted character '#' and extends to the end of the current line.

2.2.7.1.5. *Character set*

(22) Characters within a CIF are restricted to certain printable or white-space characters. Specifically, these are the ones located in the ASCII character set at decimal positions 09 (HT or horizontal tab), 10 (LF or line feed), 13 (CR or carriage return) and the letters, numerals and punctuation marks at positions 32–126.

Comment: The ASCII characters at decimal positions 11 (VT or vertical tab) and 12 (FF or form feed), often included in library implementations as white-space characters, are explicitly excluded from the CIF character set at this revision.

(23) *Comment:* The reference to the ASCII character set is specifically to identify characters in an established and widely available standard. It is understood that CIFs may be constructed and maintained on computer platforms that implement other character-set encodings. However, for maximum portability only the characters identified in the section above may be used. Other printable characters, even if available in an accessible character set such as Unicode, must be indicated by some encoding mechanism using only the permitted characters. At this revision, only the encoding convention detailed in Section 2.2.7.4(30)–(37) is recognized for this purpose.

2.2.7.1.6. *White space*

(24) Any of the white-space characters listed in paragraph (22) (*i.e.* HT, LF, CR) and the visible space character SP (position number 32 in the ASCII encoding) may be used interchangeably to separate tokens, with the exception that the semicolon characters delimiting multi-line text fields must be preceded by the white-space character or characters understood as indicating an end of line (see next paragraph).

2.2.7.1.7. *End-of-line conventions*

(25) The way in which a line is terminated is operating-system dependent. The STAR File specification does not address different operating-system conventions for encoding the end of a line of text in a text file. For a file generated and read in the same machine environment, this is rarely a problem, but increasingly applications on a network host may access files on different hosts through protocols designed to present a unified view of a file system. In practice, for current common operating systems many applications may regard the ASCII characters LF or CR or the sequence CR LF as signalling an end of line, inasmuch as these represent the end-of-line conventions supported under the common operating systems Unix, MacOS or DOS/Windows. On platforms with record-oriented operating systems, applications must understand and implement the appropriate end-of-line convention. Care must be taken when transferring such files to other operating systems to insert the appropriate end-of-line characters for the target operating system. A more complete discussion is given in (42) below.

2.2.7.1.8. *Case sensitivity*

(26) Data names, block and frame codes, and reserved words are case-insensitive. The case of any characters within data values must be respected.

2.2.7.1.9. *Implementation restrictions*

(27) Certain allowed features of STAR File syntax have been expressly excluded or restricted from the CIF implementation.

2.2.7.1.9.1. *Maximum line length and character set*

(28) Lines of text may not exceed 2048 characters in length. This count excludes the character or characters used by the operating system to mark the line termination.

The ASCII characters decimal 11 (VT) and 12 (FF) are excluded from the allowed character set [see paragraph (22)].

2.2.7.1.9.2. *Maximum data-name, block-code and frame-code lengths*

(29) Data names may not exceed 75 characters in length.

(30) Data-block codes and save-frame codes may not exceed 75 characters in length (and therefore data-block headers and save-frame headers may not exceed 80 characters in length).

2.2.7.1.9.3. *Single-level loop constructs*

(31) Only a single level of looping is permitted.

2.2.7.1.9.4. *Non-expansion of save-frame references*

(32) Save frames are permitted in CIFs, but expressly for the purpose of encapsulating data-name definitions within data dictionaries. No reference to these save frames is envisaged, and the save-frame reference code permitted in STAR is not used. This means that unquoted character strings commencing with the `$` character may not be interpreted as save-frame codes in CIF. Use of such unquoted character strings is *reserved* to guard against subsequent relaxation of this constraint.

2.2.7.1.9.5. *Exclusion of global_ blocks*

(33) In the full STAR specification, blocks of data headed by the special case-insensitive word `global_` are permitted before normal data blocks. They contain data names and associated values which are inherited in subsequent data blocks; the scope of a value extends from its point of declaration in a global block to the end of the file. Because rearrangements of the order of data blocks and concatenation of data blocks from different files are commonplace operations in many CIF applications, and because of the difficulty in properly tracking and implementing values implied by global blocks, use of the `global_` feature of STAR is expressly *forbidden* at this revision. To guard against its future introduction, the special case-insensitive word `global_` remains *reserved* in CIF.

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