JOURNAL OF APPLIED CRYSTALLOGRAPHY

ISSN 1600-5767

Specification of the Crystallographic Information File format, version 2.0

Herbert J. Bernstein,[a] John C. Bollinger,[b]* I. David Brown,[c] Saulius Gražulis,[d] James R. Hester,[e] Brian McMahon,[f] Nick Spadaccini,[g] John D. Westbrook[h] and Simon P. Westrip[i]

[a]Rochester Institute of Technology, 85 Lomb Memorial Drive, Rochester, NY 14623, USA, [b]Department of Structural Biology, St Jude Children's Research Hospital, 262 Danny Thomas Place, Memphis, Tennessee 38105, USA, [c]BIMR, McMaster University, 1280 Main Street West, Hamilton, Ontario, Canada L8S 4M1, [d]Institute of Biotechnology, Vilnius University, Graiciuno 8, Vilnius, LT-02241, Lithuania, [e]Australian Nuclear Science and Technology Organisation, New Illawarra Road, Lucas Heights, NSW 2234, Australia, [f]International Union of Crystallography, 5 Abbey Square, Chester CH1 2HU, UK, [g]The University of Western Australia, Crawley, 6009, Australia, [h]Rutgers, State University of New Jersey, Piscataway, NJ 08854, USA, and [i]The Walled Garden, Horton Green, Cheshire SY14 7EY, UK. *Correspondence e-mail: john.bollinger@stjude.org

Received 21 August 2015
Accepted 16 November 2015

Edited by A. J. Allen, National Institute of Standards and Technology, Gaithersburg, USA

Keywords: CIF; CIF 2.0.

Supporting information: this article has supporting information at journals.iucr.org/j

Version 2.0 of the CIF format incorporates novel features implemented in STAR 2.0. Among these are an expanded character repertoire, new and more flexible forms for quoted data values, and new compound data types. The CIF 2.0 format is compared with both CIF 1.1 and STAR 2.0, and a formal syntax specification is provided.

1. Introduction

The Crystallographic Information File (CIF; Hall *et al.*, 1991, 2005) is a well established format for data exchange and archiving in crystallography. Since its début, CIF and CIF applications have come to support an extensive ontology-based global framework for crystallographic data exchange and processing, sometimes called the Crystallographic Information Framework (also CIF; Hall & McMahon, 2005).

Although CIF version 1.1 (Hall *et al.*, 2005) and its parent format STAR 1.0 (Hall, 1991; Hall & Spadaccini, 1994) have broad expressive power, their designs incorporate limitations that were common at the time of their introduction. These restrict the characters and therefore languages that can be readily represented, and they make presentation of vectors, matrices and other compound data structures cumbersome. The text quoting conventions do not allow for the inclusion of all possible strings.

Since the initial STAR specification, the electronic data needs of science have grown enormously, and today's research activities require much richer metadata descriptors and more flexible approaches to data internationalization. Internet access to widely disparate and rapidly expanding information continues to be a strong driver for these requirements. These needs are addressed in STAR 2.0 (Spadaccini & Hall, 2012*a*), which takes Unicode (http://www.unicode.org/) as its character repertoire, modifies and extends the quoting rules, and provides new data types. This extended syntax provides for a higher level of data specificity, validation and automation. It is supported by a semantically rich dictionary definition language, DDLm (Spadaccini & Hall, 2012*b*), and the purpose-built language dREL for DDLm methods scripts. Using methods expressions, DDLm can define machine-parsable and executable relationships between data items, as

well as facilities for user-defined types and functions. To enable future use of these improved technologies in CIF, a new version, 2.0, of the CIF format has been developed. This format is derived from STAR 2.0 and is described in detail below.

2. Overview of CIF 1.1

CIF format version 2.0 has the same general form and high-level data model as has version 1.1, which is described in detail in Volume G of *International Tables for Crystallography*, 'ITVG' (Hall *et al.*, 2005). CIF 1.1 describes a file format for zero or more containers for data characterized as a set of discrete values ('data values') identified by distinct tags ('data names'). These data may be presented as a sequence of data names each followed by a single data value, or as a data 'loop' that presents tabular data as one or more data names followed by one or more corresponding groups of data values.

At the topmost level, CIFs are organized into data blocks, each identified by a distinguishing name (a 'block code'). Data blocks may contain save frames, each distinguished within the scope of its data block by its own name (a 'frame code'). Data names may not be repeated within the same innermost container (data block or save frame).

At the lexical level, all data values in a CIF are presented as strings of characters from the allowed set. They can be expressed in CIF 1.1 delimited by apostrophes (`'`), delimited by quotation marks (`"`), or delimited by semicolons (`;`) appearing as the first characters of their lines.[1] They may also be expressed literally, without any quotation or delimiter other than whitespace, provided that they do not contain whitespace, that they do not start with any of several characters reserved for this purpose, and that they do not take one of a few reserved forms, including, but not limited to, forms mimicking the beginning of a data block or save frame. Conventionally, literal values are described as 'whitespace-delimited strings'; values delimited by matching apostrophes or quotation marks are described as 'quoted strings', and values delimited by beginning-of-line semicolons are described as 'text fields'. Interpretation of a CIF data value can be sensitive to whether it is presented in whitespace-delimited form.

3. Changes in CIF 2.0

CIF 2.0 syntax is for the most part an extension of CIF 1.1, but not strictly so. An enumeration of the differences between the two versions of the format is presented in the following sections.

3.1. Character set and encoding

CIF 1.1 files are text files, in an unspecified system-dependent sense of that term, consisting of characters drawn from a 98-character subset of those representable in ASCII (ANSI, 1986). CIF 2.0 files, on the other hand, draw from nearly all Unicode characters (see §5) and are always encoded according to UTF-8 (The Unicode Consortium, 2014*a*). Almost any Unicode character may appear in CIF 2.0 block codes, frame codes, data names and data values. We hesitate to refer to CIF 2.0 files as 'Unicode text files', however, because CIF 2.0 recognizes few of Unicode's text semantics. For example, although Unicode defines a wide variety of characters that serve spacing and line termination purposes – and almost all of these are allowed in CIF 2.0 documents – only a few serve those purposes in CIF 2.0 syntax. CIF 2.0 files are permitted to begin with a Unicode byte-order mark (character `U+FEFF`). Such a code may be inserted automatically by text editors, and it can assist those and other programs in identifying the encoding of the file content. This character serves no other purpose in CIF 2.0: it is considered an artefact of the encoding, not a literal contributor to the file content, and it is disallowed elsewhere in CIF 2.0 files.

The designation of UTF-8 as the sole character encoding for CIF 2.0 constitutes less of a distinction from CIF 1.1 than it may seem to do, because many of the most common single-byte encodings, including US-ASCII, the ISO-8859 family and Windows-1252, encode CIF 1.1's allowed characters in the same way that UTF-8 does. Furthermore, UTF-8 itself is a common default encoding for modern computer systems. As a result, many existing CIFs are already encoded in UTF-8, whether by coincidence or by design.

3.2. Whitespace and line termination

Whitespace in general and line termination in particular are significant in CIF. CIF 1.1 leaves the definition of a 'line' as an aspect of the system-dependent notion of a text file, but CIF 2.0 defines the subdivision of files into lines exclusively in terms of a fixed set of character sequences that, for its purposes, belong to and terminate one line, separating it from the next. Specifically, CIF 2.0 recognizes and attributes identical meaning as line termination to three distinct character sequences: (1) a line feed (`U+000A`) not immediately preceded by a carriage return (`U+000D`), (2) a carriage return not immediately followed by a line feed, and (3) a carriage-return/line-feed pair. CIF 2.0 processors are required to behave as if each appearance of any of the three equivalent forms of CIF 2.0 line termination in their inputs had been converted to a line feed prior to analysis.

As in CIF 1.1, the horizontal tab character (`U+0009`) and the space character (`U+0020`) alone are recognized as in-line whitespace characters in CIF 2.0. CIF keywords, data block headers, save frame headers, data names and data values all must be separated from each other by whitespace (an in-line whitespace character or a line terminator, followed by an arbitrary number of CIF comments, additional in-line whitespace characters and line terminators). The line terminator immediately prior to a text-field opening delimiter serves both to separate the preceding data name or data value from the text field and to indicate the start of the text field; additional

[1] Throughout this paper, we adopt the typographic convention that a monospaced typeface indicates characters that may appear in a CIF 2.0 file (see also §5.1). The character referred to as 'apostrophe' is, strictly, the ASCII character 'single quote' (0x27), Unicode `U+0027`.

whitespace prior to that line terminator is not required. Otherwise, whitespace is optional in CIF 2.0 in these positions:

(*a*) between the enclosing square brackets ([,]) of a List value (see §3.8) and the values within, and between the brackets of an empty List;

(*b*) between the enclosing braces ({,}) of a Table value (see §3.9) and the entries within, and between the braces of an empty Table; and

(*c*) between a Table key and its associated value.

3.3. Version code

The content of a well formed CIF 2.0 file begins with a structured comment identifying the file's format. Such a comment is recommended for CIF 1.1 files but is required for CIF 2.0 files. The format for CIF 2.0 is

```
#\#CIF_2.0
```

where the '0' (zero) is immediately followed by whitespace. This comment serves as a 'magic number' or 'magic code' by which human beings and computer programs alike can recognize the file type. It is essential for correct interpretation of the file, because CIF 2.0 text can otherwise be difficult to distinguish from CIF 1.1 text, but CIF 2.0 is not a strict superset of CIF 1.1.

3.4. Data names, block codes and frame codes

CIF 1.1 limits the lengths of data names, block codes and frame codes to 75 characters. CIF 2.0, on the other hand, places no limit on these beyond that implicit in the overall 2048-character line-length limit it shares with CIF 1.1. Furthermore, the expanded character repertoire of CIF 2.0 applies to these CIF elements. They may contain any of the (Unicode) characters allowed in CIF 2.0, excepting only those that CIF recognizes as whitespace. It should be noted, however, that, although long names are permitted in CIF 2.0, existing CIF-based data dictionary formalisms cannot express definitions for data names that are at or very near the line-length limit.

As in CIF 1.1, CIF 2.0 block codes are required to be unique within their data files, frame codes are required to be unique within their data blocks, and data names are required to be unique within their directly enclosing data blocks and save frames, all in a case-insensitive manner. Both uniqueness and case (in)sensitivity are more complicated in the Unicode context of CIF 2.0 than in the essentially ASCII context of CIF 1.1, however. Unicode pre-composed characters *versus* their component characters, canonical character equivalence and the relative order of combining characters, among other considerations, all contribute to the result that two character sequences may be distinct on a character-by-character basis, yet be attributed identical significance by Unicode. For many of the same reasons, although Unicode defines case mappings, naïve application of case mapping rules does not provide a consistent basis for case-insensitive comparison. Therefore, CIF 2.0 defines data name, block code and frame code uniqueness in terms of the Unicode canonical caseless matching algorithm (The Unicode Consortium, 2014*b*): no two data blocks in any CIF may have names that are canonical caseless matches of each other, no two save frames in any data block may have names that are canonical caseless matches of each other, and no two data names belonging directly to the same block or frame may be canonical caseless matches of each other.

3.5. Quoted and whitespace-delimited strings

CIF 2.0 revises the syntax for quoted and whitespace-delimited string data values. In CIF 1.1, quoted data values may include their delimiter (apostrophe or quotation mark), provided that it is not followed by whitespace. CIF 2.0, on the other hand, does not permit quoted data values to embed their delimiter under any circumstance. Furthermore, whereas CIF 1.1 permits whitespace-delimited data values to contain opening and closing square bracket characters, except as the first character, and to contain opening and closing braces anywhere, CIF 2.0 excludes these four characters from appearing anywhere in whitespace-delimited data values. This CIF 2.0 restriction avoids any ambiguity with respect to values of the new List and Table data types. The characters explicitly forbidden from starting CIF 1.1 whitespace-delimited data values are also forbidden from starting CIF 2.0 whitespace-delimited data values, thereby avoiding other ambiguities, and of course whitespace-delimited values cannot contain whitespace. Otherwise, CIF 2.0 permits quoted and whitespace-delimited data values to contain any Unicode character from its character set that it does not recognize as a line terminator.

3.6. Triple-quoted strings

CIF 2.0 provides a new way to express single- and multi-line data values: triple-quoted strings. A triple-quoted string begins with a delimiter consisting of three apostrophes (''') or three quotation marks (""""), and ends with the next subsequent appearance of its opening delimiter. They cannot embed their delimiter, but they can embed the opposite delimiter, individual or pairs of apostrophes or quotation marks, and the text-field delimiter. Unlike text fields, triple-quoted strings may start anywhere that a value may start, and may end anywhere on the same or any subsequent physical line. The characters of the value have no special significance to the CIF format itself (for instance, there is no mechanism for eliding characters), but of course an application consuming them will attribute whatever significance it chooses to the value. In particular, a backslash does not protect the apostrophes or quotation marks composing the delimiters from interpretation as delimiters.

3.7. Text fields

Text fields (described above) are CIF 1.1's provision for multi-line data values. Because CIF 1.1 has no other mechanism for expressing multi-line strings and no mechanism for embedding the text-field delimiter in a text field, it cannot express data values that contain that delimiter. Additionally, the line-length limit prevents the expression of values having

any physical line exceeding that limit. ITVG documents a widely used semantic convention for CIF 1.1 line folding, by which long logical lines can be expressed in text fields *via* shorter physical lines, but it is not part of the formal syntax.

CIF 2.0 partially addresses these issues by the addition of triple-quoted strings, but ultimately addresses all cases by two related means: (1) by adopting a text prefixing protocol (see §5.2), and (2) by incorporating a version of the CIF 1.1 line-folding protocol for text fields into the CIF 2.0 specification proper (see §5.3). Text prefixing is especially targeted at permitting text-field delimiters to be expressed in text-field values, though it is more general than that. It serves its purpose by physically separating semicolons within text fields from the beginnings of their lines, and it can be employed either with or without line folding.

3.8. List data type

The new 'List' data type provided by CIF 2.0 represents, as a single (compound) value, an ordered sequence of values of any type or types. Syntactically, a List value takes the form of a whitespace-separated sequence of data values enclosed in square brackets. For example,

```
loop_
 _colour_name
 _colour_value_rgb
 red [1 0 0]
 green [0 1 0]
```

or

```
_refln.hklFoFc [[1 3 -4] 23.32(9) 22.97(11)]
```

As shown, Lists can contain other Lists, to any level of nesting. Similarly, they may contain values of the Table data type discussed next.

3.9. Table data type

The new 'Table' data type provided by CIF 2.0 represents, as a single (compound) value, an unordered collection of entries representing associations between string keys and data values of any type. This sort of data structure is also known variously as a 'map', 'dictionary' or 'associative array', among other names. Syntactically, a Table value takes the form of a whitespace-separated sequence of key–value pairs, enclosed in braces. The values may be any CIF data value. The keys take the form of quoted or triple-quoted strings as described above, with a colon appended immediately after the closing delimiter. Keys may be separated from their values by an arbitrary amount of whitespace, including none. For example,

```
{"symm":"P 4n 2 3 -1n"
'avec':[10.3 0.0 0.0]
'bvec':[0.0 10.3 0.0]
'cvec':[0.0 0.0 10.3]
"description":
"""Cubic space group
and metric cell vectors"""}
```

Like those in Lists, the values in a Table may be Lists or other Tables, nested to any depth.

4. Comparison with STAR 2.0

CIF 2.0 is for the most part a restricted profile of STAR 2.0, but there are two incompatibilities:

(1) In STAR 2.0, list elements and table entries are separated from each other by commas (and optional whitespace), whereas in CIF these elements are separated by mandatory whitespace alone.

(2) STAR 2.0 requires files to contain at least one data block, whereas STAR 1.0 and all versions of CIF to date permit files to contain no data blocks at all (and therefore, for CIF, no data).

These incompatibilities are superficial and can readily be overcome by automated translation in either direction.

Additionally, CIF 2.0 documents are subject to several restrictions relative to STAR 2.0 documents:

(*a*) Save frames may not be nested in CIF.

(*b*) CIF does not permit nested loops and therefore does not use the `stop_` keyword (but does reserve it).

(*c*) CIF documents may not contain `global_` sections (and the `global_` keyword is reserved).

(*d*) CIF requires that data names, data block codes and save frame codes be unique within their scopes in a case-insensitive sense, whereas STAR requires only exact uniqueness.

(*e*) CIF does not recognize STAR 2.0's mechanism for embedding string delimiters, nor is the escape character on which it is based (`U+0007`) in CIF's allowed set.

(*f*) CIF does not recognize STAR save frame references, and it reserves whitespace-delimited values having the form of STAR frame references.

(*g*) CIF does not recognize or allow STAR 2.0 ref-tables.

(*h*) CIF 2.0 does not allow whitespace between the quoted string and its immediately following colon in Table keys.

(*i*) The character set supported by CIF 2.0 is a slight restriction of the one supported by STAR 2.0.

(*j*) CIF imposes a 2048-character limit on line lengths.

(*k*) CIF 2.0 requires files to start with a CIF version comment.

Well formed CIF 2.0 files translated as suggested above can be interpreted as STAR 2.0, and, having been successfully parsed, CIF data can be processed *via* libraries and utilities targeting STAR 2.0 data. There is one caveat, however: CIF 2.0's line-folding and prefixing protocols for text fields. Text fields that employ these mechanisms are valid in STAR, but STAR processors will not automatically interpret their values the same way that CIF 2.0 processors do.

5. CIF 2.0 syntax

The specifications for the CIF syntax, version 2.0, comprise a formal grammar as well as detailed specifications for the line-folding and text prefixing protocols. These are presented in the next sections.

Table 1
CIF 2.0 formal syntax and grammar.

Symbol	Production
CIF2-file	(file-heading, [line-term, [wspace-any, data-block, {wspace, data-block}], [wspace], [comment]]) - ({allchars}, 2049 * char, {allchars})
file-heading	[?U+FEFF?], magic code, {inline-wspace}
magic code	'#\#CIF_2.0'
data-block	data-heading, {block-content}
data-heading	data-token, container-code
block-content	wspace, (data | save-frame)
save-frame	save-heading, {frame-content}, wspace, save-token
save-heading	save-token, container-code
frame-content	wspace, data
container-code	non-blank-char, {non-blank-char}
data	(data-name, wspace-data-value) | data-loop
data-loop	loop-token, wspace, data-name, {wspace, data-name}, wspace-data-value, {wspace-data-value}
data-name	'_', non-blank-char, {non-blank-char}
list	'[', [list-values-start, {wspace-data-value}], [wspace], ']'
list-values-start	(wspace-any, nospace-value) | (wspace-any, [comment], text-field) | ([{wspace-to-eol}, inline-wspace, {inline-wspace}], wsdelim-string) | (wspace-to-eol, {wspace-to-eol}, wsdelim-string-sol)
table	'{', [wspace-any, table-entry, {wspace, table-entry}], [wspace], '}'
table-entry	(quoted-string | triple-quoted-string), ':', (nospace-value | wsdelim-string | wspace-data-value)
wspace-data-value	(wspace, nospace-value) | ([wspace-lines], inline-wspace, {inline-wspace}, wsdelim-string) | (wspace-lines, wsdelim-string-sol) | ([wspace], [comment], text-field)
nospace-value	quoted-string | triple-quoted-string | list | table
wsdelim-string-sol	wsdelim-string - (';', {non-blank-char})
wsdelim-string	(lead-char, {restrict-char}) - (((data-token | save-token), {non-blank-char}) | loop-token | global-token | stop-token)
lead-char	restrict-char - ('"' | '#' | '$' | "'" | '_')
restrict-char	non-blank-char - ('[' | ']' | '{' | '}')
quoted-string	(quote-delim, quote-content, quote-delim) | (apostrophe-delim, apostrophe-content, apostrophe-delim)
quote-content	{char - quote-delim}
quote-delim	'"'
apostrophe-content	{char - apostrophe-delim}
apostrophe-delim	"'"
triple-quoted-string	(quote3-delim, quote3-content, quote3-delim) | (apostrophe3-delim, apostrophe3-content, apostrophe3-delim)
quote3-delim	'"""'
quote3-content	{ ['"', ['"']], not-quote, {not-quote} }
not-quote	allchars - '"'
apostrophe3-delim	"'''"
apostrophe3-content	{ ["'", ["'"]], not-apostrophe, {not-apostrophe} }
not-apostrophe	allchars - "'"
text-field	text-delim, text-content, text-delim
text-delim	line-term, ';'
text-content	{allchars} - ({allchars}, text-delim, {allchars})
data-token	('D' | 'd'), ('A' | 'a'), ('T' | 't'), ('A' | 'a'), '_'
save-token	('S' | 's'), ('A' | 'a'), ('V' | 'v'), ('E' | 'e'), '_'
loop-token	('L' | 'l'), ('O' | 'o'), ('O' | 'o'), ('P' | 'p'), '_'
global-token	('G' | 'g'), ('L' | 'l'), ('O' | 'o'), ('B' | 'b'), ('A' | 'a'), ('L' | 'l'), '_'
stop-token	('S' | 's'), ('T' | 't'), ('O' | 'o'), ('P' | 'p'), '_'
non-blank-char	char - inline-wspace
wspace	(inline-wspace | line-term), wspace-any
wspace-lines	[inline-wspace, {inline-wspace}, [comment]], line-term, {wspace-to-eol}
wspace-any	{wspace-to-eol}, {inline-wspace}
wspace-to-eol	{inline-wspace}, [comment], line-term
comment	'#', {char}
char	allchars - line-term
inline-wspace	?U+0020? | ?U+0009?
line-term	(?U+000D?, [?U+000A?]) | ?U+000A?
allchars	?U+0009? | ?U+000A? | ?U+000D? | ?U+0020 - U+007E? | ?U+00A0 - U+D7FF? | ?U+E000 - U+FDCF? | ?U+FDF0 - U+FFFD? | ?U+10000 - U+1FFFD? | ?U+20000 - U+2FFFD? | ?U+30000 - U+3FFFD? | ?U+40000 - U+4FFFD? | ?U+50000 - U+5FFFD? | ?U+60000 - U+6FFFD? | ?U+70000 - U+7FFFD? | ?U+80000 - U+8FFFD? | ?U+90000 - U+9FFFD? | ?U+A0000 - U+AFFFD? | ?U+B0000 - U+BFFFD? | ?U+C0000 - U+CFFFD? | ?U+D0000 - U+DFFFD? | ?U+E0000 - U+EFFFD? | ?U+F0000 - U+FFFFD? | ?U+100000 - U+10FFFD?

5.1. Formal grammar for CIF 2.0

This section presents a formal syntax and grammar for CIF 2.0, in a format based on ISO 14977 Extended Backus–Naur Form (EBNF) (International Standards Organization, 1996; see Table 1). It describes in symbolic form how terminal symbols – sequences of literal characters – can be assembled into aggregates represented by non-terminal symbols, the latter into larger aggregates, and so forth, ultimately to achieve an aggregate that corresponds to an entire CIF.

The terminal symbols of the grammar are represented by character sequences enclosed in apostrophes or quotation marks, such as "'" and '#\#CIF_2.0', and by EBNF special sequences, which are delimited by pairs of question marks. An apostrophe- or quotation-mark-enclosed character sequence corresponds to the characters so enclosed. Special sequences in this grammar represent single Unicode characters: specific characters are designated *via* the two characters 'U+' followed by the 4–6 hexadecimal digits of the character's Unicode code point value (*e.g.* ?U+0041? corresponds to the letter 'A'). A special sequence consisting of a pair of character designators with a hyphen between corresponds to any single character whose Unicode code point value is in the range bounded by the two designated values, inclusive. Whitespace within these special sequences is not significant.

Non-terminal symbols of the grammar are defined in terms of patterns of terminal and non-terminal symbols ('productions'). In these patterns, square brackets enclose sequences of symbols whose appearance is optional; braces enclose sequences of symbols that may be repeated any number of

times, including none; and parentheses group symbols. The comma (,) expresses concatenation – text matching the symbol or group to its left, followed by text matching the one to its right. The vertical line (|) expresses alternation – either text matching the symbol or group to its left, or text matching the one to its right. The hyphen (-) represents exception – text that matches the symbol or group to its left but not the one to its right. The asterisk (*) expresses enumerated repetition – the symbol or group to its right, repeated exactly the number of times designated by the number to its left.

A file is a well formed CIF 2.0 file if and only if the following criteria are met: its contents consist of well formed UTF-8 code sequences; the complete UTF-8-decoded contents can be exactly matched to the CIF 2.0 symbol in this grammar according to the rules presented therein; all data names, block codes and frame codes expressed in it are unique within their respective scopes, in the sense described in §3.4; and the data values in each loop construct can be evenly divided among the data names in that loop. An annotated EBNF representation of CIF 2.0 grammar and syntax, corresponding to Table 1, is available in the supporting information.

5.2. Text prefix protocol

The text prefix protocol encodes the logical content of a CIF text field by prepending a prefix to each line in a manner that can be recognized and accurately reversed. Its main purpose in CIF 2.0 is to allow the text-field delimiter to appear in the logical content of a text field, and it accomplishes that by allowing a prefix to be inserted before the semicolon of the delimiter, so that it does not appear at the beginning of its physical line. The remainder of this section describes the text prefix protocol in terms of interpreting physical text fields to evaluate their logical content.

A 'prefix' consists of a sequence of one or more characters that are permitted in a text field, except for backslash (\) or a CIF line terminator, and it does not begin with a semicolon. The text prefix protocol applies to text fields whose physical content begins with a prefix, followed by either one or two backslashes, any number of in-line whitespace characters (including none), and a line terminator or the end of the field, and whose subsequent lines each begin with the same prefix. The line containing the terminating semicolon is not accounted part of the content for this purpose. Such a text field is called a 'prefixed text field', and the logical ('un-prefixed') content of such a field is derived from its physical content by the following procedure:

(1) Remove the prefix from each line, including the first.

(2) If the remaining part of the first line starts with two backslashes then remove the first of them; otherwise remove the whole first line.

For example, given

```
_example
;CIF>\
CIF>data_example
CIF>_text
CIF>;This is an embedded text field
CIF>;
; # here the field terminates.
```

the corresponding un-prefixed value of item `_example` is

```
data_example
_text
;This is an embedded text field
;
```

The cases where the initial prefix is followed by two backslashes are exactly those in which the text prefix protocol and the line-folding protocol both apply to the same text field. In that case, one of the effects of removing prefixes according to the above procedure is to yield content in a form on which line unfolding can be performed (see next section).

5.3. Line-folding protocol

The line-folding protocol encodes the logical content of a CIF text field by splitting some or all of the logical lines into shorter physical lines, in a manner that can be recognized and reversed. The remainder of this section describes the line-folding protocol in terms of interpreting physical text fields to evaluate their logical content.

The line-folding protocol applies to text fields whose content (after decoding the text prefix protocol, if applicable) begins with a 'fold separator' consisting of a backslash, followed by any number of in-line whitespace characters, followed by a line terminator or the end of the text field. No whitespace precedes the initial fold separator. When one is present, the line terminator at the end of a fold separator is included as part of that separator.

Given un-prefixed (see §5.2) text-field content to which the line-folding protocol applies, the logical text it represents is derived from it by removing each fold separator, including the initial one. Different lines may have different amounts of whitespace in their fold separators, and the field may contain both folded and unfolded lines. This example combines text prefixing with line folding:

```
_example.long_line
;prefix:\
prefix:data_example
prefix:_text
prefix:;This line was\
prefix: folded.
prefix:;
; # here the field terminates.
```

The corresponding un-prefixed unfolded value of item `_example.long_line` is

```
data_example
_text
;This line was folded.
;
```

Note that the line-folding protocol cannot elide text-field delimiters because the line terminator belonging to that deli-

miter is not accounted part of the field content. It follows from the protocol specification, however, that if the physical content of a line-folded text field ends with a fold separator then that separator will not appear in the unfolded value.

6. Community adoption

The purpose of this paper is simply to describe the new format specification. We do not suggest strategies or time scales for its adoption. In order to accommodate new features, CIF 2.0 is not upwards compatible from CIF 1.1. This has implications for archiving and interoperability, both key design features of the CIF standard. However, any valid CIF 1.1 file can be readily up-converted to be a valid CIF 2.0 file. With careful management and the development of suitable conversion tools, CIF 2.0 can be introduced into different parts of the crystallographic information ecosystem (journal publication, data deposition and archiving, software processing) at the pace with which the community is most comfortable. That process can be followed and contributed to on an external web site maintained by the IUCr Committee for the Maintenance of the CIF Standard: http://cif2.iucr.org/.

We emphasize that the extent to which CIF 2.0's new features are employed is at the discretion of user communities. For example, CIF dictionary designers and maintainers will decide whether and to what extent data names containing non-ASCII characters will be defined.

An important enabler of format adoption will be the availability of basic tools: we note that a C library for reading and writing CIF 2.0 files is now freely available (Bollinger, 2016).

Appendix *A* discusses aspects of format conversion relevant to any strategy of gradual adoption of the new format.

7. Summary and conclusions

CIF 2.0 introduces an extended character set, a more conventional string quoting mechanism, and new List and Table data types to meet the evolving needs for multilingual publication and more complex data. Many existing CIF 1.1 files are already CIF 2.0 compliant after the addition of a version header, and any CIF 1.1 file can be converted with minor effort. The major burden will be on software developers as the number of CIF 2.0 files increases and changes to legacy applications or file down-conversions are required.

APPENDIX *A*
Conversion issues

A1. Adapting CIF 1.1 files to CIF 2.0 applications

While it is likely that some CIF 2.0 applications will be designed to also handle legacy CIF 1.1 files, it would be prudent to be aware of the steps needed to make a valid CIF 1.1 file readable according to CIF 2.0 syntax. The steps are as follows:

(i) Prepend the CIF 2.0 version code and convert to UTF-8 (see §§3.3 and 3.1) with one of the standard line endings. Note that for most files the character set and line ending will already be conformant to CIF 2.0 and no conversion will be required.

(ii) Re-quote character strings that contain embedded string terminators (see §3.5). Triple quotes will normally be an effective solution and may be stylistically preferred. Alternatively, converting the value to a text field will always be effective and does not interfere with or change the file's interpretation as CIF 1.1.

(iii) Quote whitespace-delimited data values that anywhere contain a left or right square bracket character or a left or right brace character.

To adapt a CIF for use with any specific application – especially one that is not dictionary aware – it also may be necessary to address semantic issues with the file such as the data names used and the form of the associated values. For example, if the CIF 2.0 application expects matrix- or list-valued data names and does not recognize the CIF 1.1 equivalent data names in which single matrix elements are stored, then the matrix- or list-valued data item will need to be constructed and inserted. This task can be automated with the help of a CIF dictionary that defines the data names involved and their relationships.

A2. Tailoring CIF 2.0 files for legacy applications

Owing to CIF 2.0's richer character set and its provision for compound data structures, there is no unique recipe for adapting all CIF 2.0 files for legacy applications that expect CIF 1.1 files. The steps suggested below will produce a syntactically correct CIF 1.1 file while potentially changing the contents of data items that use non-ASCII characters or compound data values. Insofar as legacy CIF 1.1 applications were written before data names that use these CIF 2.0 features existed, legacy software will not use these data names, and any changes to their values will not affect program operation. The removal of the version code will ensure that the altered file is rejected if inadvertently passed to a CIF 2.0 application. Where the legacy application requires access to information encoded using CIF 2.0 features (for example, U_{ij} matrix elements), a DDLm data dictionary may be used to drive precise semantic conversion. With the above caveats, the following steps can be performed to start and, in some cases, complete the conversion process:

(*a*) Re-quote character strings to use only CIF 1.1 style apostrophes, quotation marks or text fields.

(*b*) Convert Unicode characters that are not in CIF 1.1's allowed set to printable ASCII characters, including those appearing in data names and container names. The CIF markup convention of §2.2.7.4.13 of ITVG provides commonly used alternatives for many of the characters that will need to be converted.

(*c*) Replace lists and tables either with text fields or with individual values.

(*d*) Remove the version code, or replace it with the optional version code `#\#CIF_1.1` if the resultant file is fully compliant with the CIF 1.1 specification.

A3. ImgCIF, CBF and CIF 2.0

Many raw crystallographic diffraction images are written in a CIF 1.1-based DDL2-style CBF/imgCIF (Crystallographic Binary File, image CIF) format, as described in ch. 2.3 of ITVG. The introduction of CIF 2.0 does not directly affect CBF/imgCIF. The maintainers of this sub-format are considering a revision supporting the new CIF 2.0 constructs, but at this time they are not supported, and as yet there is no time line for such a revision. At present, therefore, software producing CBF/imgCIF should not assume that CBF/imgCIF consumers will accept or recognize any of CIF 2.0's changes or additions to CIF 1.1.

Acknowledgements

The authors would like to thank all those who contributed to the discussions that led to this new CIF specification.

References

American National Standards Institute (1986). *ANSI X3.4-1986 – American National Standard for Information Systems – Coded Character Sets – 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII)*. American National Standards Institute, Washington, DC, USA.

Bollinger, J. C. (2016). *J. Appl. Cryst.* **49**, 285–291.

Hall, S. R. (1991). *J. Chem. Inf. Model.* **31**, 326–333.

Hall, S. R., Allen, F. H. & Brown, I. D. (1991). *Acta Cryst.* A**47**, 655–685.

Hall, S. R. & McMahon, B. (2005). Editors. *International Tables for Crystallography*, Vol. G, *Definition and Exchange of Crystallographic Data*. Dordrecht: Springer.

Hall, S. R. & Spadaccini, N. (1994). *J. Chem. Inf. Model.* **34**, 505–508.

Hall, S. R., Westbrook, J. D., Spadaccini, N., Brown, I. D., Bernstein, H. J. & McMahon, B. (2005). *International Tables for Crystallography*, Vol. G, *Definition and Exchange of Crystallographic Data*, edited by S. R. Hall & B. McMahon, pp. 25–36. Dordrecht: Springer.

International Standards Organization (1996). *ISO/IEC 14977:1996 – Information Technology – Syntactic Metalanguage – Extended BNF*. International Standards Organization, Geneva, Switzerland.

Spadaccini, N. & Hall, S. R. (2012*a*). *J. Chem. Inf. Model.* **52**, 1901–1906.

Spadaccini, N. & Hall, S. R. (2012*b*). *J. Chem. Inf. Model.* **52**, 1907–1916.

The Unicode Consortium (2014*a*). *The Unicode Standard, Version 7.0.0*, ch. 3, §3.9. Mountain View: The Unicode Consortium. http://www.unicode.org/versions/Unicode7.0.0/.

The Unicode Consortium (2014*b*). *The Unicode Standard, Version 7.0.0*, ch. 3, §3.13. Mountain View: The Unicode Consortium. http://www.unicode.org/versions/Unicode7.0.0/.