CIF - Changes to the specification

08 August 2011

This document specifies changes to the *syntax* and binary form of CIF. We refer to the current syntax specification of CIF as CIF1, and the new specification as CIF2. To date all archival CIFs are CIF1.

The changes to syntax are necessitated by the adoption of new dictionary functionalities that introduce several extensions, including new data types, and method definitions using dREL.

It is assumed the reader has a thorough understanding of the CIF1 specification.

TERMINOLOGY

Reference to **character(s)** means abstract characters assigned code points by **Unicode**. Specific **characters** are referenced according to Unicode convention, U+*xxxx*[*x*[*x*]], where *xxxx*[*x*[*x*]] is the four- to six-digit hexadecimal representation of the assigned code point. The preferred character encoding for CIF2 is UTF-8.

Reference to **ASCII** characters means characters U+0000 through U+007F, or, equivalently the first 128 characters of the **ISO-8859-1** (**LATIN-1**) character set.

Reference to **newline** or **\n** means the sequence that conventionally terminates a line record (which is environment dependent). **See Change 3**.

Reference to **whitespace** means the characters ASCII space (U +0020), ASCII horizontal tab (U+0009) and the **newline** characters. Without regard to local convention, the various other characters that Unicode classifies as whitespace (character categories Zs and Zp) do not constitute **whitespace** for the purposes of CIF2.

PREAMBLE

CIF2 significantly extends CIF1 functionality, primarily through new dictionary features. CIF2 is not fully backwards-compatible with CIF1: many files compliant with CIF1 are also compliant with CIF2, but some are not (see especially change 5, below). The CIF1 standard will continue to operate for the foreseeable future in parallel with CIF2.

CHANGE 1 – NEW (MAGIC CODE)

CIF2 file is uniquely identified by a required magic code at the beginning of its first line . The code is,

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

followed immediately by **whitespace**. The immediately following space on this line is reserved for encoding disambiguation signatures. Note that where a Unicode BOM is used, it would appear prior to the magic code in the byte stream and does not form part of the CIF text.

CHANGE 2 – NEW (CHARACTER SET)

CIF2 files are standard variable length plain text files, which for compatibility with older processing systems will have a maximum line length of 2048 characters. As discussed above and below, however, there are some restrictions on the character set for token delimiters, separators and data names. For compatibility with CIF1 behaviour, there is no formal restriction on the encoding of CIF2 files, providing they contain only code points from the ASCII range. If a CIF2 file contains characters equivalent to Unicode code points greater than U+007F (127 decimal), then the particular encoding used must either be UTF8 or algorithmically identifiable from the CIF2 file itself. Acceptable identification algorithms will be published as necessary as annexes to this standard (see description of magic code and encoding disambiguation in Change 1). Annexes notwithstanding,

> (i) a CIF2 file containing characters outside the ASCII range with no BOM and no disambiguation signature will be a UTF8 file, and
>
> (ii) a CIF2 file containing characters outside the ASCII range with a valid UTF8 or UTF16 BOM and no disambiguation signature, will be a Unicode file written in the indicated encoding.

In keeping with XML restrictions we allow the characters

```
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+10FFFD
```

In addition, character U+FEFF and characters U+xFFFE or U+xFFFF where x is any hexadecimal digit are disallowed. Unicode reserves the code points E000 – F8FF for private use. The IUCr and only the IUCr may specify what characters are assigned to these code points in the context of CIF2.

Reasoning: There is growing demand for the wider character set afforded by Unicode to be made available in applications, especially those where internationalisation is an issue.

CHANGE 3 – RESTRICTION

Treatment of Newline

CIF2 processors are required to treat <U+000A>, <U+000D> and <U+000D><U+000A> as newline characters, by normalising them to <U+000A> on read. No other characters or character sequences may represent newline. In particular, CIF2 processors should not interpret the Unicode characters U+2028 (line separator) or U+2029 (paragraph separator) as newline.

CHANGE 4 – DEFINITION

Character set for *data names*.

In CIF2 the tags referred to as data names are composed of characters from the allowed set above, excluding a whitespace, since this terminates data name string. A data name begins with an ASCII _ and may be followed by any number of characters within the 2048 character restriction.

All data names in a **valid** CIF must be defined in a CIF dictionary referenced implicitly or explicitly by the CIF, and the DDLs in which such dictionaries are written may place additional restrictions on the data names that can be defined. For example, data names defined in a DDLm dictionary must match the regular expression `_[A-Za-z0-9_.]+` (the `.` is the explicit **ASCII** period character).

Note: In CIF2, as with CIF1, there is no explicit meaning to the sequence of characters, or the placement of _ or . in the data name. The separation of the category and attribute in a data name by a period (.) is purely a convention.

CHANGE 5 – RESTRICTION

Whitespace-delimited *data values*.

A data value in CIF2 may be a whitespace delimited string of allowed characters.

A whitespace delimited string cannot contain any of the ASCII characters `[ { ] }`. Furthermore, the first character of a whitespace delimited string cannot be any of the ASCII characters `" ' _ $`. STAR keywords may not appear as whitespace delimited strings: `loop_ global_ save_* stop_ data_*` (case insensitive), where * refers to zero or more characters.

Reasoning: The above exclusions are required for CIF2 syntax to be unambiguous.

CHANGE 6 – RESTRICTION

Delimited strings.

The delimited strings accepted in CIF2 are,

(1) A string delimited by ASCII single-quotes(`'`). The string is initiated by an ASCII single-quote, can consist of allowed characters excluding the newline, and is terminated by the first subsequent ASCII single-quote. **Clearly, the string within cannot contain ASCII single-quote characters.**

CIF2 does not specify any interpretation of the contents of the string. For example

```
loop_ _author.family _name 'Harris' 'Gr\"uber'
```

As far as CIF2 is concerned, the string values are `Harris` and `Gr\"uber`; handling of any elide characters is left to the calling application.

(2) A string delimited by ASCII double quotes (`"`). The string is initiated by an ASCII double-quote, can consist of allowed characters excluding the newline, and is terminated by the first subsequent ASCII double-quote. **Clearly, the string within cannot contain ASCII double-quote characters.**

CIF2 does not specify any interpretation of the contents of the string. For example

```
_quote.literal "He said, 'We're going in circles'"
```

The string value is `He said, 'We're going in circles'`, including the embedded single-quotes.

(3) A string delimited by ASCII newline semi-colon (`\n;`). The string is initiated by an ASCII newline semi-colon sequence, consists of any of the allowed characters, and is terminated by the first subsequent ASCII newline semi-colon sequence. **Clearly, the strings within cannot contain an ASCII newline semi-colon sequence.**

The string is interpreted according to the CIF Line-Folding Protocol (Change 11) when its signature is present, and according to the CIF Text Prefix Protocol (Change 12) when its signature is present. Otherwise, CIF2 does not specify any interpretation of the contents of the string. For example

```
_recipe.ingredients
;Sugar
Flour
Butter
;
```

The string value is `Sugar\nFlour\nButter`, where `\n` is the literal newline.

When the Line-Folding Protocol and Text Prefix Protocol are applied to the same value, the value shall appear in CIF as if line-folding had been performed first, followed by prefixing (see example 3 in Change 12).

With respect to changes 8, 9, and 10, the newline in the opening delimiter is considered to separate a delimited string of this kind from the preceding syntax element.

CHANGE 7 – NEW

Triple-quote delimited strings.

The ASCII """ sequence (alternatively ASCII ''') delimits the beginning of a string that may contain any printable character and whitespace and is terminated by the first subsequent """ sequence (alternatively '''). CIF2 does not specify any interpretation of the contents of the string. The string can contain separable " and "" characters, (alternatively ' and ''). **Clearly, the string within cannot contain an ASCII """ (or alternatively ASCII ''')**.

For example

```
"""He said "His name is O'Hearly"."""

'''In {\bf \TeX} the accents are \' and \".'''
```

The string values are, `He said "His name is O'Hearly".` and `In {\bf \TeX} the accents are \' and \".`. No interpretation of any elides is undertaken; this is the responsibility of the calling application. The triple quote string supports embedded **newlines**, which are considered part of the string.

CHANGE 8 – NEW

List data type.

The ASCII square bracket ([ ]) is accepted in STAR for delimiting the List compound data type. A List is an ordered sequence of values. A data value of type List is initiated by an ASCII left square bracket([) and terminated by the pair-matching ASCII right square bracket (]). The List elements are separated from each other by whitespace. For example

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

The elements of a List can be any CIF2 data values, and hence it is a recursive data type. For example

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

Since List elements are whitespace separated (and may include \n;-delimited strings), a List can span more than one physical line. For example

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

is identical to the previous example list.

Whitespace is allowed, but not required, between the [ ] delimiters and the values within, and between the opening and closing delimiters of an empty List.

CHANGE 9 – NEW

Table data type.

The ASCII curly brace ({}) is accepted in STAR for delimiting the Table (*Associative Array*) compound data type. A Table is an unordered mapping from string labels (“indices”) to CIF2 data values. Labels must be unique within each Table (excluding Tables nested within it).

A data value of type Table is initiated by an ASCII left curly brace ({) and terminated by the pair-matching ASCII right curly brace (}). The Table elements are separated from each other by **whitespace**, and consist of: an index label as a single- or double-quote delimited string, or as a triple-quoted string; an ASCII colon(:); and a CIF2 data value. 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"""}
```

A Table is a recursive data type. Since Table elements are whitespace separated (and also since values may be `\n;`-delimited strings), a Table may span more than one physical line.

Whitespace is allowed, but not required, between the `{}` delimiters and the elements within, and between the opening and closing delimiters of an empty Table.

CHANGE 10 – REFINEMENT to CIF1

Separating syntax elements.

CIF2 keywords, data block headers, save frame headers, data names, and data values must all be separated from each other by **whitespace**. Whitespace not otherwise part of a CIF2 syntax element is significant only for this purpose.

Reasoning: The CIF1 specification relies implicitly on the syntactic structure of the language to require whitespace separators between syntax elements. The CIF2 syntax no longer implicitly provides whitespace separators in some cases (notably, after most types of data values), therefore the requirement is now made explicit.

CHANGE 11 – REFINEMENT to CIF1

Text Field Line-Folding Protocol.

The CIF text field line-folding protocol is a mechanism for splitting logical lines of text across two or more physical lines of a CIF semicolon-delimited data value ('text field'). A version of this protocol appears among the "Common Semantic Features" of CIF 1.1 and is in wide use in that context; in CIF 2.0, text field line-folding is part of the CIF syntax, as described below. These syntax specifications do not address use in CIF 2.0 documents of CIF 1.1's analogous line-folding semantics for comments.

The protocol applies to text fields whose contents (after interpretation of the text prefix protocol, if applicable) begin with a backslash, followed by any number of whitespace characters other than newline, followed by newline or the end of the text field; that sequence is designated `\<ws>*\n` below. There must not be any whitespace preceding the initial backslash.

Given un-prefixed (Change 12) text field contents to which the line-folding protocol applies, the logical text it represents is derived from it by removing each occurrence of `\<ws>*\n`, including the initial one. Different lines may have different amounts of whitespace between the trailing backslash and newline. For example,

```
_recipe.ingredients
;\
Wheat \
Flour
Butter \
;
```

The string value is `Wheat Flour\nButter`, where `\n` is the literal newline.

Note that the line-folding protocol cannot elide the terminating `\n;` of a text field because the `\n` of that delimiter is not accounted part of the field contents. It follows from the

definition of \<ws>*\n, however, that if the field contents end with \<ws>* then that will not appear in the unfolded value, as exhibited by the example above.

Furthermore, it follows from the fact that the line-folding protocol applies to the *contents* of text fields that it does not affect the recognition of text field *delimiters*. In particular, it is always a syntax error (just as in CIF 1.1) for the closing delimiter of a text field to be followed by a backslash without intervening whitespace. For historical reasons, some CIF 1.1 software attempts to recover from that specific syntax error at the end of line-folded text fields by reinterpreting the closing delimiter as an embedded newline/semicolon sequence. These specifications neither forbid nor require such behavior.

CHANGE 12 – NEW

Text Prefix Protocol.

The CIF text-prefix protocol is a mechanism for formatting the logical content of a CIF semicolon-delimited value ("text field") so as to allow embedding literal \n; sequences. It may also be useful for improving human readability of some CIFs.

The protocol applies to text fields whose physical contents begin with a prefix (see below), followed by one or two backslashes (\), optionally followed by any amount of whitespace other than a newline, followed by a newline. A prefix consists of a sequence of one or more characters that are permitted in a text field, except for backslash or newline, and does not begin with a semicolon.

The second and all subsequent physical lines of the contents of a prefixed text field must begin with the designated prefix for that field. The line containing the terminating semicolon is not part of the contents for this purpose. The logical (*i.e.* 'un-prefixed') contents of the field can be derived from the physical contents 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 line.

Example 1:

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

The string value is

```
data_example\n_text\n;This is an embedded multiline value\n;
```

, where \n is the literal newline.

Example 2: Whitespace prefixes

The specifications permit a prefix to contain or even to consist entirely of whitespace characters:

```
data_providing_example
_example
; \
 data_example
 _text
 ;This is an embedded multiline value
 ;
; # here the field terminates.
```

The string value is the same as in example 1 above. Note that the prefix here is a specific sequence of whitespace characters, as given at the beginning of the field, not an indentation level nor any other sequence of whitespace except the particular one given.

Example 3: Combining prefixing and line folding

As already specified, text prefixing and line folding may be combined, in which case the CIF representation of the value is as if it had first been folded and then the result prefixed:

```
_embedded.cif
;CIF>\\
CIF>data_embedded \
CIF>_recipe.ingredients
CIF>;\
CIF>Wheat \
CIF>Flour
CIF>Butter
CIF>;
;
```

The string value is

```
data_embedded _recipe.ingredients\n;Wheat Flour\nButter\n;
```

, where again `\n` is the literal newline.

Reasoning: together with the text field line-folding protocol, the text prefix protocol provides a mechanism for representing arbitrary strings as CIF2 data values.