CIF - Changes to the specification

18 February 2010

This document specifies changes to the *syntax* 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 **ASCII** characters means those characters in **UNICODE Code Page 0** or, equivalently the first 127 characters of the **LATIN-1** character set. The lexical characters of CIF2 are restricted to the 7 bit ASCII range because this enables faster parsing since one can defer UTF-8 decoding to later.

Reference to **newline** or **\n** means the sequence that terminates the line record (which are architecture dependent). The most common **newline** sequences are `#x0A` (ASCII line feed) for *nix and Mac OSX, and `#x0D#x0A` (ASCII carriage return and line feed) for Windows. CIF applications also recognise `#x0D` as a **newline**. This is also consistent with most text based Internet protocols.

The UNICODE characters `#x2028` (line separator) and `#x2029` (paragraph separator) are not syntactically significant in CIF2, and are treated as any other character.

*Reasoning: A small subset of **ASCII** is specified to have syntactic significance in CIF2, and only these characters. In particular `#x2028` and `#x2029` are rendered inconsistently across applications. Likewise alternative UNICODE renderings of* `" ' : ; { } [ ]` *have no syntactic significance in CIF2. However we do allow the wider use of UNICODE as data.*

Reference to **whitespace** means the characters ASCII space (`#x20`), ASCII horizontal tab (`#x09`) and the **newline** characters. In the same vein as above, the additional 20 UNICODE characters that constitute **whitespace** are not syntactically significant in CIF2.

PREAMBLE

CIF2 significantly extends CIF1 functionality, primarily through new dictionary features. The CIF1 standard will continue to operate for the foreseeable future in parallel with CIF2. Applications built on the CIF2 standard will be able to process CIF1 data files.

CHANGE 1 – NEW (MAGIC CODE)

A CIF2 file is uniquely identified by a magic code on its first line. The code is,

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

CHANGE 2 – NEW (ENCODING)

CIF2 files are standard variable length binary files, but for historical reasons will have a maximum record length of 2048 bytes. In a general sense the contents of the file are characters that are encoded in UTF-8, however there are some restrictions on the character set for token delimiters, separators and for data names.

In keeping with XML restrictions we allow the UTF-8 characters

```
#x9  #xA  #xD
#x20 – #xD7FF
#xE000 – #xFFFD
#x10000 – #x10FFF
```

The characters `#xE000–#xF8FF` are reserved for private use, and the IUCr can specify what these characters must be.

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 – DEFINITION

Character set for *data names*.

In CIF2 the tags referred to as data names are comprised of characters from the allowed UTF-8 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 byte restriction.

Important restriction: In the case where the contents of a CIF2 data file are defined in the new DDLm dictionary there is an imposed restriction on the character set of a data name. CIF2 data names defined in a dictionary are comprised of characters only from the **ASCII** set, beginning with an **ASCII** _ and may be followed by any number of **ASCII** characters in the regular expression `[A–Za–z0–9_.]` (the `.` is the explicit **ASCII** period character). Any data item defined in the dictionary as a compound data type can have its individual elements in a CIF2 data file, represented as a *qualified* data name which allows for the additional ASCII characters `[ ] " '` to be used. The *qualified* data name is the same string as the dictionary defined data name with the addition of square bracket enclosed indices, either numerical or one of the CIF2 quote strings. The qualified data name cannot appear as a dictionary definition, only the parent data name can be defined.

The BNF below defines the qualified data name entity.

```
<QualifiedDataname>    = <Dataname> <IndexExpression>+

<IndexExpression>      = '[' <digits> ']' |
```

```
                          '[' <quote> <chars>+ <quote> ']'
<quote>                 = '\'' | '"' | '"""'
```

Data names can appear in any dREL scripts and operate at that level as programming identifiers. Characters such as **[ ] + - /** *in a data name cannot be supported in the dictionary since they make the parsing of a dREL script at best ambiguous, but often syntactically incorrect. The presence of the* [ ] *characters in a qualified data name implies these individual vales are to be used to populate the parent compound data item.*

This restriction is similar to those found in most programming languages, except that the period character is explicitly allowed in the tag.

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 4 – RESTRICTION

Whitespace-delimited *data values*.

A data value in CIF2 may be a whitespace-delimited string of allowed UTF-8 characters.

The first character of a whitespace-delimited string cannot be any of the ASCII characters, `" ' _ $ [ {`, and the terminal character cannot be `]` or `}`, since these have special meaning. A whitespace-delimited string cannot exactly match any STAR keyword, `loop_ global_ save_* stop_ data_*`, where * refers to zero or more characters.

Reasoning : Within compound data types (lists and tables) whitespace-delimited strings will be allowed values. Exclusion of the above initial and terminal characters ensures unambiguous parsing of the compound data types.

CHANGE 5 – 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 UTF-8 characters excluding the newline, and is terminated by the first subsequent ASCII single-quote. **Clearly, the string within cannot contain ASCII single-quote characters.**

At a lexical level the contents of the string are treated as raw. For example

```
loop_ _author.family_name   'Harris' 'Gr\"uber'
```

The lexer returns the string value as `Harris` and `Gr\"uber`, leaving the handling of any elide characters to the calling application.

(2) A string delimited by ASCII double-quotes (`"`). The string is initiated by an ASCII double-quote, can consist of UTF-8 characters excluding the newline, and is terminated by the first

subsequent ASCII double-quote. **Clearly, the string within cannot contain ASCII double-quote characters.**

At a lexical level the contents of the string are treated as raw. For example

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

The lexer returns the string value as `He said, 'We're going in circles'`.

(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 UTF-8 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.**

At a lexical level the contents of the string are treated as raw. For example

```
_recipe.ingredients

;Sugar
Flour
Butter
;
```

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

CHANGE 6 – 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 `'''`). At a lexical level the contents of the string are treated as raw. 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 lexer returns the string values, `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 7 – NEW

List data type.

The ASCII square bracket (`[ ]`) is accepted in STAR for delimiting the List compound data type. A List is an ordered set. 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 values are whitespace separated. 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 values are whitespace separated they can be split over more than one physical line. There is implicit line joining, and the newline has no meaning with regard to the List values. For example

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

is identical to the previous example list.

CHANGE 8 – 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 set that is indexed by a string label. 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 whitespace separated, and consist of; an index label as a single- or double-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 they can be split over more than one physical line. There is implicit line joining, and the newline has no meaning with regard to the Table values. **The index of a Table element must be one of the CIF2 quote delimited strings**.

CHANGE 9 – REFINEMENT to CIF1

Separating tokens.

For delimited values, the first subsequent instance of the terminating character sequence terminates that token. **Whitespace** characters are used to separate tokens. That is, whitespace separation is required ***between the end of a token and the beginning of the next token***. For example

```
"123" abc' [[1 2 3] [4 5 6]]

    {"first":Bolt "second":Johnson "third":Borzov}
```

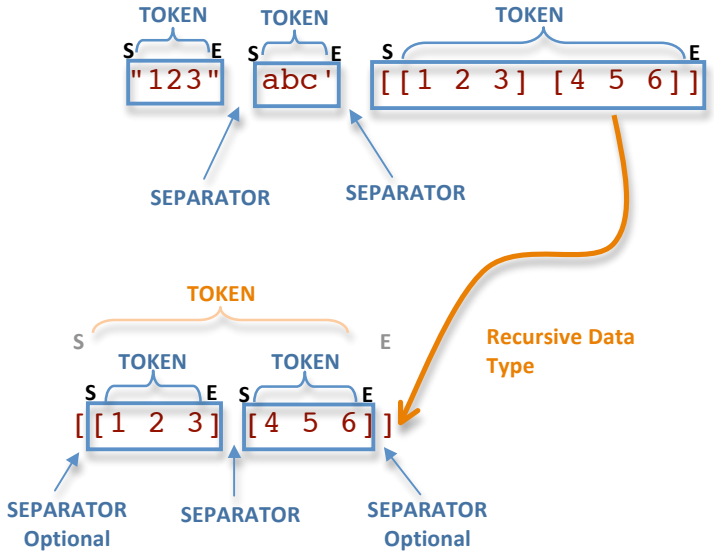
Note: the **whitespace** separator is required between the end of a token and the beginning of the next token. That does NOT imply **whitespace** is necessary between the beginning of a token and the beginning of the next token (the same holds for the ends of tokens). Hence in the List example above there is no requirement for **whitespace** between `[[` or `]]`.

In a Table the internal token is of the form `"String":Value`, rather than two tokens separated by a `:`.

Reasoning: In CIF1 quoted strings are delimited by a **whitespace + quote character** *and* **quote character + whitespace** *pair. The whitespace at beginning and end are explicitly part of the quoted string token. In CIF2 we adopt a simpler, more common approach with quoted strings delimited by a* **quote character** *pair. In CIF2 a* **square bracket** *pair delimits a List, and a* **curly brace** *pair delimits a Table. The only meaningful use of* **whitespace** *is to separate data tokens. One can recover from missing* **whitespace** *but we formally define missing* **whitespace** *as an error in this specification. The same production (grammar) rules for data tokens can be used recursively for the compound data types.*

EXAMPLES

Interpretation of data tokens and separators in a CIF data file.



The values returned by the parser of these CIF2 data could be represented internally (***in a general sense***) as follows,

```
string(123) string(abc') list(list(1,2,3),list(4,5,6))
```

Note the delimiters are not part of the values.