CIF - Changes to the specification

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.

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. 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. UTF-8 directly supports an extensive range of printable objects that are not accessible through ASCII. Currently a very limited range of special characters is supported through the IUCr specific escape-sequencing mechanism.

CHANGE 3 – RESTRICTION

Character set for *data names*.

In CIF2 the tags referred to as data names are comprised of characters only from the ASCII set, and restricted to those in the regular expression `[A-Za-z0-9_.]` (the `.` is the explicit ASCII period character).

Reasoning : Data names can appear in any dREL scripts and operate at that level as programming identifiers. Characters such as `[ ] + - /` *in a data name make the parsing of a dREL script at best ambiguous, but often syntactically incorrect.*

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

CHANGE 4 – RESTRICTION

Non-delimited *data values*.

A data value in CIF2 may be a non-delimited string of UTF-8 characters, but excluding the ASCII characters, `: { } [ ]`.

As with CIF1, the first character of a non-delimited string cannot be any of the ASCII characters, `“ ‘ _ $`, since these have special meaning. A non-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) non-delimited strings will be allowed values. Exclusion of the above 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, form-feed and vertical tab characters, and is terminated by the first subsequent ASCII single-quote. **The string cannot contain any 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, form-feed and vertical tab characters, and is terminated by the first subsequent ASCII double-quote. **The string cannot contain any 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 UTF-8 characters, and is terminated by the first subsequent 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 newlines 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 ‘’). 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 also 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. The elements of a List can be any ASCII space separated data values and hence it is a recursive data type.

This syntax can represent a List as an ordered set of values. For example

```
[1.2 “a” ‘b’ [1.2 “a” ‘b’]]
```

In the context of being outside of data tokens a list is initiated by an ASCII left square bracket (`[`) and terminated by the pair-matching ASCII right square bracket (`]`). Expressions in square brackets can be split over more than one physical line. This implies implicit line joining, and there is no newline token between implicit continuation lines. For example

```
[1.2 “a” ‘b’
[1.2 “a” ‘b’]]
```

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. The elements of a Table can be any ASCII space separated data values defined in STAR, hence it is a recursive data type. The index label must be a single or double quoted string, separated by an ASCII colon (`:`) from the 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”””}
```

In the context of being outside of data tokens a list is initiated by an ASCII left curly brace (`{`) and terminated by the pair-matching ASCII right curly brace (`}`). Expressions in curly braces brackets can be split over more than one physical line. This implies implicit line joining, and there is no newline token between implicit continuation lines (as in the previous example).

CHANGE 9 – REFINEMENT to CIF1

Separating tokens.

For delimited values, the first subsequent instance of the terminating character sequence terminates that token. Whitespace characters (ASCII space, ASCII tab or ASCII newline) are used to separate tokens. That is, whitespace separation is required **between the end of one 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 requirement of whitespace is explicitly between the end of one token and the beginning of the next token. That does NOT require that whitespace is necessary between the beginning of one token and the beginning of the next token (similarly for the ends of tokens). Hence in the List example above there is no requirement for whitespace between `[[` and `]]`.

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