2.2.4.3. Line lengths

The STAR File does not restrict the lengths of text lines. The original CIF specification introduced an 80-character limit to facilitate programming in Fortran and transmission of CIFs by email. Contemporary practices have enabled the line-length limit in the version 1.1 specification to be extended to 2048 characters. While the new limit in effect mandates the use of a 2048-character input buffer in compliant CIF-reading software, there is no obligation on CIF writers to generate output lines of this length.

2.2.4.4. Lengths of data names and block codes

CIF imposes another length restriction that is not integral to the STAR File syntax: data names, data-block codes and frame codes may not exceed 75 characters in length. This is an increase over the 32-character limit of the original specification, although this extension had already been approved by COMCIFS to coincide with the release of version 2 of the core dictionary (see Chapter 3.2).

There is no fundamental technical reason underlying this restriction; it permits assignment of a fixed-length buffer for recording such data names within a software application, but perhaps more significantly, it encourages a measure of conciseness in the creation of data names based on hierarchical component terms.

2.2.4.5. Case sensitivity

Following the general STAR File approach, the special tokens `loop_`, `save_`, the reserved word `global_`, data-block codes and save-frame codes, and data names are all case-insensitive. The case of any characters within data values must, however, be respected.

2.2.5. Common semantic features

As mentioned in Section 2.2.1, the STAR File structure allows retrieval of indexed data values without prior knowledge of the location of a data item within the file. Consequently there is no significance to the order of data items within a data block. However, molecular-structure applications will typically need to do more than retrieve an arbitrary string value. There is a need to identify the nature of individual data items (achieved portably through the definitions of standard data names in dictionary files), but also to be able to process the extracted data according to whether it is numerical or textual in nature, and possibly also to parse and extract more granular information from the entire data field that has been retrieved.

Different CIF applications have a measure of freedom to define many of the details of the content of data fields and the ways in which they may be processed – in effect, to define their semantic content. However, there are a number of conventions that are common to all CIF applications and this should be recognized in software applicable to a range of dictionaries.

These are discussed in some detail in the formal specification document in Section 2.2.7.4; in this section some introductory comments and additional explanations are given.

2.2.5.1. Data-name semantics

It is a fundamental principle of the STAR File approach that a data name is simply an arbitrary string acting as an index to a required value or set of values. It is equally legitimate to store the value of a crystal cell volume either as

```
_bdouiGFG78=z   1085.3(3)
```

or as

```
_cell_volume    1085.3(3)
```

provided that the users of the file have some way of discovering that the cell volume is indeed indexed by the tag `_bdouiGFG78=z` or `_cell_volume`, as appropriate.

However, it is conventional in CIF applications to define (in public dictionary files) data names that imply by their construction the meaning of the data that they index. Chapter 3.1 discusses the principles that are recommended for constructing data names and defining them in public dictionaries, and for utilizing private data names that will not conflict with those in the public domain.

Careful construction of data names according to the principles of Chapter 3.1 results in a text file that is intelligible to a scientist browsing it in a text editor without access to the associated dictionary definition files. In many ways this is useful; it allows the CIF to be viewed and understood without specialized software tools, and it safeguards some understanding of the content if the associated dictionaries cannot be found. On the other hand, there is a danger that well intentioned users may gratuitously invent data names that are similar to those in public use. It is therefore important for determining the correct semantic content of the values tagged by individual data names to make maximum possible disciplined use of the registry of public dictionaries, the registry of private dataname prefixes, and the facilities for constructing and disseminating private dictionaries discussed in Chapter 3.1.

2.2.5.2. Data typing

In the STAR File grammar, all data values are represented as character strings. CIF applications may define data types, and in the macromolecular (mmCIF) dictionary (see Chapter 3.6) a range of types has been assigned corresponding to certain contemporary computer data-storage practices (*e.g.* single characters, case-insensitive single characters, integers, floating-point numbers and even dates). This dynamic type assignment is supported by the relational dictionary definition language (DDL2; see Chapter 2.6) used for the mmCIF dictionary and is not available for all CIF applications.

However, a more restricted set of four primary or base data types is common to all CIF applications.

The type **numb** encompasses all data values that are interpretable as numeric values. It includes without distinction integers and non-integer reals, and the values may be expressed if desired in scientific notation. At this revision of the specification it does not include imaginary numbers. All numeric representations are understood to be in the number base 10.

It is, however, a complex type in that the standard uncertainty in a measured physical value may be carried along as part of the value. This is denoted by a trailing integer in parentheses, representing the integer multiple of the uncertainty in the last place of decimals in the numeric representation. That is, a value of '1085.3(3)' corresponds to a measurement of 1085.3 with a standard uncertainty of 0.3. Likewise, the value 34.5(12) indicates a standard uncertainty of 1.2 in the measured value.

Care should be taken in the placement of the parentheses when a number is expressed in scientific notation. The second example above may also be presented as 3.45E1(12); that is, the standard uncertainty is applied to the mantissa and not the exponent of the value.

Note that existing DDL2 applications itemize standard uncertainties as separate data items. Nevertheless, since the DDL2 dictionary includes the attribute `_item_type_conditions.code` with an allowed value of 'esd', future conformant DDL2 parsers might be expected to handle the parenthesized standard uncertainty representation.

The preferred behaviour of a CIF application is to determine the type of a data value by looking up the corresponding dictionary definition. However, some CIF-reading software may not be designed with the ability to parse dictionaries; and indeed any CIF reader may encounter data names that are not defined in a public or accompanying dictionary. It is therefore appropriate to adopt a strategy of interpreting as a number any data value that looks like one, *i.e.* adopts any of the permitted ways to represent a numeric value. Therefore, in the absence of a specific counter-indication (from a dictionary definition), the data value in the following example may be taken as the numeric (integer) value 1:

```
_unknown_data_name   1
```

On the other hand, if `_unknown_data_name` were explicitly defined in a dictionary with a data type of 'char', then the value should be stored as the literal character `1`.

This is a subtle point, perhaps of interest only to software authors. Nevertheless, the consistent behaviour of CIF applications will depend on correct implementation of this behaviour.

The data type **char** covers single characters or extended character strings. Since CIF tokens are separated by white space, any character string that includes white-space characters (including line-terminating characters) must be delimited by one or other of a set of special characters used for this purpose. The detailed rules for quoting such strings are given in Section 2.2.7.1.4 and comprise the standard CIF syntax rules for this case. No semantic distinction is made in general between short character strings and text strings that extend over several lines, described in the specification document as 'text fields', although again particular CIF applications may choose to impose distinctions. Note that numbers within a quoted string or a text block (bounded by semicolons in column 1) are not interpreted as type 'numb' but as type 'char'.

The data type **uchar** was introduced explicitly at revision 1.1 of the CIF specification, and is intended to formalize the description and automated handling of certain strings in CIFs that are case-insensitive (such as data names and data-block headers).

The data type **null** is a special type that has two uses. It is applied to items for which no definite value may be stored in computer memory. As such it is a formal device for allowing the introduction of data names into dictionary files that do not represent data values permissible within a data file instance. The usual example is that of the special data names introduced in DDL1 dictionaries (such as the core dictionary) to discuss categories.

The more important use of the null data type is its application to the meta characters '`?`' (query) and '`.`' (full point) that may occur as values associated with any data name and therefore have no specific type. (Arguably, for this case 'any' might be a better type descriptor than 'null'.)

The substitution of the query character '`?`' in place of a data value is an explicit signal that an expected value is missing from a CIF. This 'missing-value signal' may be used instead of omitting an item (*i.e.* its tag and value) entirely from the file, and serves as a reminder that the item would normally be present.

The substitution of the full-point character '`.`' in place of a CIF data value serves two similar, but not identical, purposes. If it is used in looped lists of data it is normally a signal that a value in a particular packet (*i.e.* a value in the row of the table) is 'inapplicable' or 'inappropriate'. In some CIF applications involving access to a data dictionary it is used to signal that the default value of the item is defined in its definition in the dictionary. Consequently, the interpretation of this signal is an application-specific matter and its use must be determined according to the application. For example, in a CIF submitted for publication in *Acta Crystallographica* the presence of a '`.`' value for the item `_geom_bond_site_symmetry_1` is predetermined as the default value 1_555 (as per the dictionary definition). Note that, in this instance, it is also equivalent to 'no additional symmetry' or 'inapplicable'.

2.2.5.3. Extended data typing: content type and encoding

The initial implementation of CIF assumed that most character strings would represent identifiers or terse descriptions or comments, and that the correct behaviour of the majority of CIF applications would be simply to store these in computer memory or retrieve them verbatim. Only a few data values were foreseen as having extended content that might need special handling. For example, the complete text of a manuscript was envisaged as being included in the field `_publ_manuscript_processed`. The handling of this field (its extraction and typesetting) would be left to unspecified external agents, although some clue as to the provenance of the contents of that field (and thus their appropriate handling) would be given by `_publ_manuscript_creation`.

However, the evolution of CIF applications has required that some element of typographic markup be permitted in a growing number of data values, and future applications may be envisaged in which graphical images, virtual-reality models, spreadsheet tables or other complex objects are embedded as the values of specific data items. Since it will not be possible to write general-purpose CIF applications capable of handling all such embedded content, techniques will need to be developed for transferring each such field to a specialized but separate content handler. In the meantime, the rather *ad hoc* conventions for introducing typographic markup available at present are described in Sections 2.2.7.4.13–17. It is hoped that in the future different types of such markup may be permitted so long as the data values affected can be tagged with an indication of their content type that allows the appropriate content handlers to be invoked.

It has also been necessary to allow native binary objects to be incorporated as CIF data values. This was done to support the storage of the large arrays of image data obtained from area detectors. Since the CIF character set is based on printable ASCII characters only, encodings including compression have been developed to permit interconversion between ASCII and binary representations of such data (see Chapter 2.3).

Nowadays, arbitrary embedded objects may be transported in web pages *via* the http protocol (Fielding *et al.*, 1999) or as attachments to email messages structured according to the MIME protocols (*e.g.* Freed & Borenstein, 1996). Identification of encoding techniques and hooks to invoke suitable handlers are carried in the relevant Content-Type and Content-Encoding http or MIME headers. It is suggested that this may form the basis of suitable tagging of content types and encoding for future CIF development.

A candidate for a CIF-specific encoding protocol is the special convention introduced with CIF version 1.1 to interconvert long lines of text between the new and old length limits (Section 2.2.7.4.11). This is an encoding in the sense that it is a device designed to retain any semantic content implicit in textual layout, while conforming to slightly different rules of syntax. It is designed to enable CIFs written to the longer line-length specification to be transformed so that they can still be handled by older software. Since the object of the exercise is to manage legacy applications, it is likely that the interconversion will be done through external applications, or filters, designed specifically for the purpose. Such a conversion filter is conceptually the same as a filter to convert a binary file into an ASCII base-64 encoding, for example.