2.1. Specification of the STAR File

By S. R. Hall and N. Spadaccini

2.1.1. Introduction

A human language, in all its forms, is spoken, written and comprehended according to grammatical principles that evolve continuously with the need to express efficiently new ideas and experiences. In this chapter we will describe how similar principles have been developed to construct, describe and understand scientific data, such as numbers and codified text. The efficient and flexible expression of data may be achieved using grammatical rules similar to those of spoken languages, though the precise and unambiguous rendition and communication of data must preclude those nuances, subtleties and individual interpretation so important to the spoken and graphical world of poetry, literature and art. It is important to record these aspects of human endeavour, but they are distinctly different from the objectives of science, where information must be expressed with maximum precision and efficiency.

Understanding any language involves two fundamental steps – identification of the individual elements of a language sequence, and comprehension of the sequence structure. In a spoken language these steps normally comprise the simultaneous recognition of individual words and the understanding of their grammatical context. Such a simple decoding process belies the enormous potential for complexity in spoken languages. Nevertheless, most ‘word sequences’ are understood in this way and a similar approach may be used with non-textual data.

As discussed in Section 1.1.3, until quite recently most approaches to storing scientific data electronically were based on fixed-format structures. These are simple to construct and easy to comprehend provided the data layout is fixed and widely understood. That is, items, as well as lists of items, are written in a fixed sequential order that is mutually agreed on by those writing and reading the data. However, the preordained nature of a fixed format, which intentionally prevents changes to the data structure in a file, also poses a serious limitation for many scientific applications. This is because the nature of data used in scientific disciplines, such as in crystallography, evolves continuously and requires recording processes that are extensible and adaptable to change.

A lack of ready extensibility in the expression of data is particularly problematical for long-term archiving. For example, the recovery of information stored in a fixed format may be impossible if the layout details are lost or altered with time. Less rigid fixed-format variants, using keywords to identify groups of data, can improve the flexibility of data recording but they often preclude the introduction of new kinds of data.

In the 1980s it was widely recognized across the sciences that more general, extensible and expressive approaches were needed for recording, transmitting and archiving electronic data. This led to the development of free-format file structures. These file structures are the basis for universal file formats intended to: (*a*) store all kinds of data; (*b*) be independent of computer hardware (*i.e.* portable); (*c*) be both machine-parsable and human-understandable; (*d*) adapt to future data evolution (*i.e.* be extensible and robust); and (*e*) facilitate data structures of any complexity (*i.e.* have rich syntax capabilities).

The extent to which these objectives can be met determines the universality of a data-storage approach. Several of these properties are difficult to achieve simultaneously, and the compromises that have been adopted have largely determined the success of universal data languages in different fields. The STAR File is a universal data language applicable to all scientific disciplines, and the Crystallographic Information File described in Chapter 2.2 of this volume is a specific instance of the STAR format that has been adopted by the crystallographic community.

2.1.2. Universal data language concepts

As discussed above, the basic precepts of a universal data language parallel those of spoken languages in that a syntax and a grammar are used to describe and arrange (*i.e.* present) information, and hence define its comprehension. In particular, because data *values* (as numbers, symbols or text) are not often self-descriptive (*e.g.* a temperature value as a number alone cannot be distinguished from a number representing an atomic weight) interpretation depends critically on appropriate cues to the meaning and organization of data. A cue, in this context, has traditionally been prior knowledge of a fixed format, but can be explicit descriptions of data items, or even embedded codes identifying the relationship between data items. Cues represent the contextual thread of a data language, in the same way that grammatical rules provide words in natural-language sentences with meaning, and convert computer languages into executable code. This section will consider the concepts of data and languages that are important in understanding the syntax and purpose of a STAR File.

The first requirement of a language for electronic data transmission is that it be computer-interpretable, *i.e.* parsable. One should be able to read, interpret, manipulate and validate data entirely by machine. The other essential attributes of a universal file, as listed in Section 2.1.1, are flexibility, extensibility and applicability to all kinds of data. In modern-day science, data items change and expand continually and it is therefore paramount that a data language can accommodate new kinds of data seamlessly. It is the inflexibility and restrictive scope of fixed formats that limit their usefulness to crystallography for purposes other than local and temporary data retention.

2.1.2.1. Data models

Various approaches exist for meeting the objectives for a universal file listed in Section 2.1.1. The task is complicated by the fact that properties such as generality and simplicity, or accessibility and flexibility, are in many respects technically incongruent, and, depending on the nature of the data to be represented, particular compromises may be taken for efficiency or expediency. In other words, data languages, like spoken languages, are not equally efficient at expressing concepts, and the syntax of a language may need to be tailored to the target applications.

Affiliations: Sydney R. Hall, School of Biomedical and Chemical Sciences, University of Western Australia, Crawley, Perth, WA 6009, Australia; Nick Spadaccini, School of Computer Science and Software Engineering, University of Western Australia, 35 Stirling Highway, Crawley, Perth, WA 6009, Australia.

This was one of the issues confronting the Working Party on Crystallographic Information formed by the IUCr in 1988 (Section 1.1.6) to decide on the most appropriate universal file data language for crystallography from those under development (McCarthy, 1990). It is interesting historically to note that one of these was HGML – not the web markup language we know today, but the *Human Genome Mapping Library* language. Another language considered by the working party was ASN.1 (ISO, 2002) used by the National Institute of Standards and Technology and several US Government departments. It is an accepted ANSI and ISO standard for data communication and is supported by software, such as NIST's OSI Toolkit. ASN.1 possesses a rich set of language constructs suited to representing complex data, but suffers from data identifiers that are encoded and not human-readable, and a syntax that is verbose (particularly for repetitive data items such as those common in crystallography). These characteristics mean that a typical Protein Data Bank (PDB) file expressed in ASN.1 notation increases in size by up to a factor of 5. This was hardly an attraction in the 1980s when storage media were very expensive. Moreover, the ASN.1 syntax is not particularly intuitive, and is difficult to read and to construct. In contrast, the STAR File proposed at the first WPCI meeting had a relatively simple syntax, was human-readable and provided a concise structure for repetitive data. It also proved suitable for constructing electronic dictionaries, as will be discussed in later chapters. However, its serious and well recognized weakness in 1988 was that any recording approach using a simple syntax to encode complex data must involve sophisticated parsing software, and at that time only the prototype software (*QUASAR*; Hall & Sievers, 1990) was available. It was therefore not a straightforward decision for the WPCI to decide to recommend the STAR File syntax as a more appropriate data language for crystallographic applications. It was this decision that led to the development of the CIF approaches described in this volume.

2.1.3. The syntax of the STAR File

The syntax of the STAR File (Hall, 1991; Hall & Spadaccini, 1994) has been used to develop a number of discipline-specific exchange and archival approaches, including the Crystallographic Information File (CIF) (Hall *et al.*, 1991), the Molecular Information File (MIF) (Allen *et al.*, 1995), the dictionary definition language (DDL1) (Hall & Cook, 1995), the macromolecular dictionary definition language (DDL2) (Westbrook & Hall, 1995) and the STAR dictionary definition language (StarDDL) (Spadaccini *et al.*, 2000). The details of the CIF, MIF, DDL1 and DDL2 approaches are given in Chapters 2.2, 2.4, 2.5 and 2.6, respectively.

A STAR File is a sequential file containing lines of standard ASCII characters. A file may be divided into any number of discrete sets of unique data items. Sets may be in the form of data blocks, global blocks or save frames. The syntax rules for these sets are given below in descriptive form. A more rigorous description of the STAR File syntax is given in Appendix 2.1.1 in extended Backus–Naur form (McLennon, 1983).

The STAR File is a free-form language in which spaces (ASCII 32), vertical tabs (ASCII 11) and horizontal tabs (ASCII 9) are collectively referred to as `<blank>`, and newlines (ASCII 10), form feeds (ASCII 12) and carriage returns (ASCII 13) are collectively referred to as `<terminate>`. White spaces `<wspace>`, used to separate lexical tokens within the file, are all characters in the joined set of `<blank>` and `<terminate>`.

2.1.3.1. Text string

A text string is defined as any of the following.

(*a*) A sequence of non-white-space characters on a single line excluding a leading underscore `<_>` (ASCII 95).

Examples:

```
5.324
light-blue
```

(*b*) A sequence of characters on a single line containing the leading digraph `<wspace><'>` and the trailing digraph `<'><wspace>`. `<'>` is a single-quote character (ASCII 39) and `<wspace>` is white space.

Examples:

```
'light blue'
'classed as "unknown"'
'Patrick O'Connor'
```

Note that the use of the `<'>` character in the text string that is bounded by a `<'>` character is not precluded unless it is immediately followed by `<wspace>`. The leading and trailing digraphs serve to delimit the string and do not form part of the data. In the above example the value associated with the text field `'light blue'` is **light blue**.

(*c*) A sequence of characters on a single line containing the leading digraph `<wspace><">` and the trailing digraph `<"><wspace>`. `<">` is a double-quote (ASCII 34) character and `<wspace>` is white space.

Examples:

```
"low melting point"
"Patrick O'Connor"
"Doug Collins' crystal"
"classed as "unknown""
```

The use of the `<">` character in the text string that is bounded by a `<">` character is not precluded unless it is immediately followed by `<wspace>`. The leading and trailing digraphs serve to delimit the string and do not form part of the data.

The text strings of type (*a*), (*b*) and (*c*) cannot contain line-breaking characters, and therefore cannot span multiple lines. There can be more than one text string per line if each value is preceded by a data name, or the values are part of a looped list (see Section 2.1.3.5).

(*d*) A sequence of lines starting with `<terminate><;>` and finishing with `<terminate><;>`, where `<;>` is the semicolon character (ASCII 59).

Example:

```
; School of CSSE
  UWA
;
```

The requirement that the `<;>` character be the first character on the line does not preclude the presence of other characters on the same line, in as much as it does not violate the STAR File structure.

The leading and trailing digraphs delimit the text field and do not form part of the data. The character sequence between the digraphs, including any line-breaking characters, constitutes the value of the text field. The value associated with the above example is `<blank>`**School**`<blank>`**of**`<blank>`**CSSE** `<terminate><blank><blank>`**UWA** (note in particular that the `<terminate>` character preceding the final `;` delimiter is *not* part of the value).

references