

2.1. SPECIFICATION OF THE STAR FILE

2.1.3.2. Data name

A data name (or tag) is the identifier of a data value (see Section 2.1.3.3) and is a sequence of non-white-space characters starting with an underscore character `<_>` (ASCII 95).

Example:

```
_publication_author_address
```

2.1.3.3. Data value

A data value is a text string preceded by its identifying data name. Privileged keywords, such as described in Sections 2.1.3.5 to 2.1.3.8, are excluded from this definition.

2.1.3.4. Data item

A data item is a data value and its associated data name. Each data item stored in a STAR File is specified with this combination.

2.1.3.5. Data loop list

A looped list consists of the keyword `loop_` followed by

(a) a sequence of data names (possibly with nested `loop_` constructs); and

(b) a sequence of loop packets, each containing data values which are identified in the same order as the data names in (a).

A looped list specifies a table of data in which the data names represent the ‘header descriptors’ for columns of data and the packets represent the rows in the table. Looping lists may be nested to any level. Each loop level is initialized with the `loop_` keyword and is followed by the names of data items in this level. Data values that follow the nested data declarations must be in exact multiples of the number of data names. Each loop level must be terminated with a `stop_`, except the outermost (level 1) which is terminated by either a new data item or the privileged strings indicating a save frame (Section 2.1.3.6), a data block (Section 2.1.3.7), a global block (Section 2.1.3.8) or an end of file.

An example of a simple one-level loop structure is:

```
loop_
  _atom_identity_number
  _atom_type_symbol      1 C   2 C   3 O
```

Nested (multi-level) looped lists contain matching data packets [as per (b) above] and an additional `stop_` to terminate each level of data. Here is a simple example of a two-level nested list.

```
loop_
  _atom_id_number
  _atom_type_symbol
  loop_
    _atom_bond_id_1
    _atom_bond_id_2
    _atom_bond_order
      1 C   1 2 single  1 3 double stop_
      2 C   2 1 single                stop_
      3 O   3 1 double                stop_
```

The matching of data names to value packets is applied at each loop level. Initially the data values are matched to the data names listed in the outermost level loop. This process is iterated to successively inner levels. At the innermost loop level, data matching is maintained until a `stop_` is encountered. This returns the matching process to the next outer level. The matching process is recursive until the loop structure is depleted. Here is an example of a three-level loop structure.

```
loop_
  _atomic_name
  loop_
    _level_scheme
    _level_energy
```

```

loop_
  _function_exponent
  _function_coefficient
hydrogen
(2)->[2]      -0.485813
1.3324838E+01  1.0
2.0152720E-01  1.0 stop_
(2)->[2]      -0.485813
1.3326990E+01  1.0
2.0154600E-01  1.0 stop_
(2)->[1]      -0.485813
1.3324800E-01  2.7440850E-01
2.0152870E-01  8.2122540E-01 stop_
(3)->[2]      -0.496979
4.5018000E+00  1.5628500E-01
6.8144400E-01  9.0469100E-01
1.5139800E-01  1.0000000E+01 stop_ stop_
```

2.1.3.6. Save frame

A save frame is a set of unique data items wholly contained within a data block. The frame starts with a `save_framecode` statement, where the `framecode` is a unique identifying code within the data block. Each frame is closed with a `save_` statement.

Example:

```
data_example
save_phenyl
  _object_class      molecular_fragment
  loop_
    _atom_identity_node
    _atom_identity_symbol 1 C 2 C 3 C 4 C 5 C 6 C
save_
loop_ _molecular_fragments $ethyl $phenyl $methyl
```

A save frame has the following attributes:

(a) A save frame may contain data items and loop structures but not other save frames [see (f)].

(b) The scope of the data specified in a save frame is the save frame in which it is specified.

(c) Data values in a save frame are distinct from any identical items in the parent data block.

(d) A save frame may be referenced within the data block in which it is specified using a data item with a value of `$framecode`.

Example:

```
loop_ _amino_acid_seq
  _amino_acid_data  1 $tyr 2 $arg 3 $arg 4 $leu
```

where ‘arg’, ‘tyr’ and ‘leu’ are frame codes identifying three save frames of data.

(e) A frame code must be unique within a data block.

(f) A save frame may not contain another save frame, but it may contain references to other save frames in the same data block using frame codes.

2.1.3.7. Data block

A data block is a set of data containing any number of unique items and save frames. A data block begins with a `data_blockcode` statement, where `blockcode` is a unique identifying name within a file. A data block is closed by another `data_blockcode` statement, a `global_` statement or an end of file.

Example:

```
data_rhinovirus
all information relevant to rhinovirus included here

data_influenza
all information relevant to influenza virus
included here
```

A data block has the following attributes:

(a) A block code must be unique within the file containing the data block.

(b) Data blocks may not be referenced from within a file [in contrast to save frames – see Section 2.1.3.6(d)].

(c) The scope of data specified in a data block is the data block. The value of a data item is always associated with the data block in which it is specified.

(d) Data specifications in a data block are unique, except they may be repeated within a save frame. Data specifications in a save frame are independent of the parent data block specifications.

(e) If a data item is not specified in a given data block, the global value is assumed. If a global value is not specified, the value is unknown.

2.1.3.8. Global block

A global block is a set of data items which are implied to be present in all data blocks which follow in a file, unless specified explicitly within a data block. A global block starts with a `global_` keyword and is closed by a `data_blockcode` statement or an end of file.

Example:

```
global_
information that is default within subsequent
data blocks
data_influenza
```

A global block has the following attributes:

(a) The scope of global data is from the point of declaration to the end of file.

(b) A global block may contain data items, loop structures and save frames.

(c) Multiple global blocks are concatenated to form a single block in which the last item specification has precedence.

(d) A data item specified within a data block has precedence over a data item specified in a prior global block.

2.1.3.9. Data sets and scopes

A data set is the generic term for a unique set of data. A STAR File may contain three types of data sets: global blocks, data blocks and save frames. The attributes of data sets are as follows.

(a) A file may contain any number of data sets.

(b) The data names defined within a data set must be unique to that set. That is, all `data_blockcode` names must be unique within the file, all data names must be unique within a `global_` block, all data names and `save_framecodes` must be unique within a data block, and all data names must be unique within a save frame.

(c) The scope of data sets is hierarchical (Fig. 2.1.3.1). Global blocks encompass all following data blocks; data blocks scope all contained save frames.

(d) The scope of a save frame is all data items contained within the frame.

(e) The scope of a data block is the boundaries of the data block, *i.e.* the end of the file or the start of the next data block, including any contained save frames. The same data item may be defined within a save frame and within the parent data block. All specifications of this item will be recognized when accessing the data block.

(f) The scope of a global block is the file, from the point of invocation to the end of file or the start of the next global block. It encompasses all contained global data items, data blocks and save frames. Globally specified data are active provided identical items are not specified in subsequent data sets.



Fig. 2.1.3.1. Nested scopes of STAR data sets.

2.1.3.10. Privileged constructs

The following constructs are privileged.

(a) Text strings starting with the character sequences `data_`, `loop_`, `global_`, `save_` or `stop_` are privileged words (keywords) and may not be used as values in text strings of the type defined in Section 2.1.3.1(a).

(b) A sharp character `<#>` (ASCII 35) is an explicit end-of-line signal provided it is not contained within a text string of the types defined in Section 2.1.3.1(b), (c) or (d). Characters on the same line and following an active sharp character are considered as comment text.

2.1.3.11. Using `stop_` in looped lists

In Section 2.1.3.5 we discuss how `stop_` is used to terminate a loop of data values and to return the looped list to the next outer nesting level. This same construction applies in the looped list of data names. The following, although not particularly intuitive, is a valid construction.

```
loop_
  _atom_id_number
  loop_
    _atom_bond_id_1
    _atom_bond_id_2
    _atom_bond_order stop_
  _atom_type_symbol
    1 1 2 single 1 3 double stop_ C
    2 2 1 single stop_ C
    3 3 1 double stop_ O
```

This is equivalent to the loop definition given in Section 2.1.3.5. One can use the `stop_` in name definitions to inhibit the nesting of loops in the definitions.

Appendix 2.1.1

Backus–Naur form of the STAR syntax and grammar

This description of the STAR syntax and grammar is annotated to clarify issues that cannot be represented in a pure extended Backus–Naur form (EBNF) definition.

The allowed character set in STAR is restricted to ASCII 09–13, 32–126. Other characters from the ASCII set are illegal. If such characters are present in a file the error state is well defined, but the functionality of the error handler is not specified. For instance, one may choose to return an illegal file exception and terminate the application or equally one may choose to ignore and skip over the illegal characters.

The concept of white space `<wspace>` includes a comment, since these only serve (in a parser sense) to delimit tokens anyway. We adopt the convention here of enclosing terminal symbols in single forward quotes. There are necessary provisos to this, and for representing formatting characters they are:

‘\’ represents the single-quote character, *i.e.* the \ is an escape character.