International Tables for Crystallography (2006). Vol. G, Section 5.2.2.3, pp. 489–491.


```
data_Gaussian

loop_
     _basis_set_atomic_name
     _basis_set_atomic_symbol
     _basis_set_atomic_number
     _basis_set_atomic_mass
    loop_
     _basis_set_contraction_scheme
     _basis_set_funct_per_contraction
     _basis_set_primary_reference
     _basis_set_source_exponent
     _basis_set_source_coefficient
     _basis_set_atomic_energy
       loop_
     _basis_set_function_exponent
     _basis_set_function_coefficient

      hydrogen    H    1    1.0079
#     --------
 (2)->[2]   1:    PKC1.1.1   R44 .       -0.485813
      1.3324838E+01    1.0
      2.0152720E-01    1.0 stop_
 (2)->[2]   1:    PKC1.2.1   R33 .       -0.485813
      1.3326990E+01    1.0
      2.0154600E-01    1.0 stop_
 (2)->[1]   2     PKC1.14.1  R24 R24     -0.485813
      1.3324800E-01    2.7440850E-01
      2.0152870E-01    8.2122540E-01 stop_
 (3)->[2]   2:1   PKC1.23.1  R75 R75     -0.496979
      4.5018000E+00    1.5628500E-01
      6.8144400E-01    9.0469100E-01
      1.5139800E-01    1.0000000E+01 stop_ stop_

     lithium    Li   3     6.94
#    -------
 (4)->[4]   1:    PKC3.1.1   R44 .       -7.376895
       3.4856175E+01     1.0
       5.1764114E+00     1.0
       1.0514394E+00     1.0
       4.7192775E-02     1.0 stop_
 (9,4)->[3,2]  7:2:1,3:1
                  PKC3.9.1   R2  R98     -7.431735
       921.271    0.001367       138.730    0.010425
       31.9415    0.049859       9.35329    0.160701
       3.15789    0.344604       1.15685    0.425197
       0.44462    0.169468       0.44462   -0.222311
       0.076663   1.116477       0.028643   1.0
       1.488      0.038770       0.2667     0.236257
       0.07201    0.830448       0.02370    1.0      stop_
 (4,3)->[3,2]   4:2:1,2:1
                  PKC3.30.1  R77 R77     -7.419509
       1.09353E+02    1.90277E-02 1.64228E+01    1.30276E-01
       3.59415E+00    4.39082E-01 9.05297E-01    5.57314E-01

       5.40205E-01   -2.63127E-01 1.02255E-01    1.14339E+00
       2.85645E-02    1.00000E+00

       5.40205E-01    1.61546E-01 1.02255E-01    9.15663E-01
       2.85645E-02    1.00000E+00  stop_ stop_
```

Fig. 5.2.2.1. Example quantum chemistry basis set functions in STAR File format.

coefficients). At the innermost loop level, a loop packet is simply a row within a table of exponents and coefficients of the basis set function.

If one were to treat this example file as a database of indeterminate structure and query the values associated with one of the data names, for example, **`_basis_set_function_exponent`**, one would retrieve a series of strings `1.3324838E+01`, `2.0152720E-01` *etc.* However, the value strings in themselves are insufficient to allow the reconstitution of any data structure in the file. One also needs an expression of the levels within the nested loop structure at which the values were located, and an indication that they were associated with different packets of information at those various levels. This additional information about the context of each value is sufficient to determine its position within the data structure without any other *a priori* information regarding the data model. The context is most easily expressed by listing the output values in STAR File format.

```
data_Gaussian
loop_
    loop_
        loop_
     _basis_set_function_exponent
        stop_
    stop_

        1.3324838E+01 2.0152720E-01 stop_
        1.3326990E+01 2.0154600E-01 stop_
        1.3324800E-01 2.0152870E-01 stop_
        4.5018000E+00 6.8144400E-01 1.5139800E-01 stop_
    stop_

        3.4856175E+01 5.1764114E+00 1.0514394E+00
        4.7192775E-02 stop_

        921.271 138.730 31.9415 9.35329 3.15789 1.15685
        0.44462 0.44462 0.076663 0.028643 1.488 0.2667
        0.07201 0.02370 stop_

        1.09353E+02 1.64228E+01 3.59415E+00 9.05297E-01
        5.40205E-01 1.02255E-01 2.85645E-02 5.40205E-01
        1.02255E-01 2.85645E-02 stop_
    stop_
```

Fig. 5.2.2.2. Retrieval from the example file in Fig. 5.2.2.1 of the value of **`_basis_set_function_exponent`** with associated context.

Fig. 5.2.2.2 is an output listing of the requested values for this example, where the context is expressed as the innermost of three nested loop levels and distinct packets at this level are indicated. It will be seen also that by tracing the disposition of **`stop_`** words the embedding within higher-level loop packets can also be inferred.

5.2.2.3. Context in data sets

Another indicator of context in the previous example is the data-block header, which was reproduced in the output of Fig. 5.2.2.2.

The STAR File allows data instances in three types of location: in a data block, in a save frame or in a global block.

The usual way to partition a STAR File is by data blocks; each such block represents a data set in which a data name (associated with a single or multiple values) may be declared once only.

Data blocks may include save frames. A save frame is an encapsulated subsidiary data set, effectively insulated from the contents of the surrounding data block, in which data items may occur that have the same names as items in the parent data block. Indeed, 'parent' is potentially a misleading term, since no relationship is implied between the data within a save frame and those in the data block in which the save frame occurs. A reference to a save frame may, however, occur as a data *value* within the data block where the save frame is specified. Recall from Section 2.1.3.6 that save frames within a data block are uniquely identified by the *framecode* header.

Global blocks may also occur in a STAR File, preceding or interspersed between data blocks. For each data item defined within a global block, that definition is inherited by each succeeding data block that does not contain an internal definition of a data item with the same name. If there is a definition of a data item with the same name within a data block, that internal definition overrides the global definition within that data block. The situation is then re-evaluated in the next data block. If that data block does not contain an internal definition, the global definition holds.

```
data_reaction

save_methyl
  loop_
     _atom_identity_node
     _atom_identity_symbol     1  C  2  C
  loop_
     _attached_hydrogen_node
     _attached_hydrogen_count  1  3
save_

save_ethyl
  loop_
     _atom_identity_node
     _atom_identity_symbol     1  C  2  C  3  C
  loop_
     _attached_hydrogen_node
     _attached_hydrogen_count  1  3  2  3
save_

save_R1
  loop_
     _variable_alternative_number
     _variable_identifier_symbol
     _variable_node    1  $methyl  1   2  $ethyl  1
save_

save_carboxylic_acid
  loop_
     _atom_identity_node
     _atom_identity_symbol     1 $R1  2  C  3  O  4
O
  loop_
     _attached_hydrogen_node
     _attached_hydrogen_count  2  0  3  0  4  1
save_

loop_
     _reaction_component_number
     _reaction_component_symbol
     _reaction_component_type
            1  $carboxylic_acid  reactant
```

Fig. 5.2.2.3. Example STAR data structure where save frames encapsulate related data sets. See text for details.

The scope of data values is well defined (see Section 2.1.3.9). Only data expressed in a global block have values that are inherited in later portions of the STAR File. Data values in data blocks or save frames are restricted in scope to the current data block or save frame, respectively.

A consequence of these rules of scope and encapsulation is that a full description of the context of a STAR data value must also reflect any values carried through as global data or by dereferencing associated save frames. The results are not always intuitive.

Consider Fig. 5.2.2.3, which represents a partial description of a chemical reaction where one of the reactants is expressed as a generic structure described by the save frame **save_R1**. However, the generic structure in this case is restricted to a small number of alkyl groups, each described in its own save frame. Setting aside this prior knowledge, we see that a request for **_atom_identity_symbol** must return not only the data values in their embedded save frames, but also the save frames in their entirety and the higher-order data values that reference the matching save frames. It is only in this way that we can guarantee that the value can be used by any application. Fig. 5.2.2.4 demonstrates the full context of the returned requested data values.

Notice that the requested item occurs (among other places) in the save frame **save_carboxylic_acid** and this instance of the

```
data_reaction

save_methyl
loop_
     _atom_identity_node
     _atom_identity_symbol    1 C 2 C
loop_
     _attached_hydrogen_node
     _attached_hydrogen_count  1 3
save_

save_ethyl
loop_
     _atom_identity_node
     _atom_identity_symbol    1 C 2 C 3 C
loop_
     _attached_hydrogen_node
     _attached_hydrogen_count  1 3 2 3
save_

save_R1
loop_
     _variable_alternative_number
     _variable_identifier_symbol
     _variable_node    1  $methyl  1  2  $ethyl  1
save_

save_carboxylic_acid
loop_
     _atom_identity_symbol $R1 C O O
save_

loop_
     _reaction_component_symbol $carboxylic_acid
```

Fig. 5.2.2.4. Context for the requested values of **_atom_identity_symbol** in the preceding example. See text for details.

item is presented solely in the context of the save header and closure strings (it is shown in italics in Fig. 5.2.2.4). However, one of the values extracted from this location is the save-frame reference pointer $R1 that identifies **save_R1**, and the complete contents of this save frame are presented (because the data structure represented by the save frame *is* itself one of the values of the requested data item). Further de-referencing of the save-frame pointers within **save_R1** results in the extraction also of the complete save frames **save_methyl** and **save_ethyl**. In this example it is coincidental that there are instances of the requested data item (**_atom_identity_symbol**) within these returned save frames as well.

Notice, however, that establishing the full context of the returned data demands also that data values referencing the **save_carboxylic_acid** frame be presented. In this example, the value of **_reaction_component_symbol** at the outermost level of the data block is returned, a result that may at first seem surprising. It is only in this way that one can be sure that an arbitrary application will have access to the full semantic information carried by the data item.

Data values declared in global blocks should be presented in the same spirit of supplying the complete context in which the value was instantiated, and not simply the value in isolation. For example, given the trivial STAR File

```
global_
     _example   foo
data_1
data_2
     _example   bar
```

a request for **_example** should return the identical file, *not* the interpolated result

```
data_1
    _example   foo
data_2
    _example   bar
```

despite the latter's equivalence purely in terms of the non-contextual values returned.

5.2.3. *Star_Base*: a general-purpose data extractor for STAR Files

The stand-alone application *Star_Base* (Spadaccini & Hall, 1994) provides a facility for performing database-style queries on arbitrary STAR Files. It is generic in nature and makes no assumptions about the nature or organization of the data in a STAR File. It may indeed be used as an application-specific database tool if the user has prior knowledge of the relationships between included data items. However, by faithfully returning context as well as value in the way outlined in Section 5.2.2, it can be applied to any STAR File even without such prior knowledge.

5.2.3.1. Program features

Star_Base is a fully functional STAR File parser and may be used to test the syntactic validity of an input STAR File. It may be used to write an input STAR File directly to the output stream, while validating the structural integrity as the contents are parsed. The input format and comments are discarded on output.

Given a valid input file, *Star_Base* guarantees to write output in fully compliant STAR format.

If a data name is supplied as a request item, *Star_Base* will return the single or multiple values associated with that data name and their associated context according to the principles of Section 5.2.2, *i.e.* all loop structures, data-block headers and global headers will be returned, and save frames will be expanded as required to accommodate de-referencing of frame codes as returned values.

Where multiple data items are requested, *Star_Base* will write their occurrences to its output stream in the order they were requested, *not* in the order of appearance in the input file. This may disturb data relationships that are implicit in the ordering or association of values in the input file, but it is the responsibility of the user to track and retain such associations where they are an essential part of an application-level data model. As emphasized before, a generic STAR tool will make no assumptions about data models and will simply return values and contexts as requested.

To illustrate the effect of this, consider a request for the following data items from the example file of Fig. 5.2.2.1:

```
_basis_set_atomic_name
_basis_set_atomic_symbol
_basis_set_contraction_scheme
```

Star_Base will return the following result:

```
data_Gaussian
loop_
    _basis_set_atomic_name
    _basis_set_atomic_symbol
    loop_
     _basis_set_contraction_scheme
    stop_

    hydrogen H
        (2)->[2] (2)->[2] (2)->[1] (3)->[2] stop_

    lithium Li
        (4)->[4] (9,4)->[3,2] (4,3)->[3,2] stop_
```

However, if the same items are requested in a different order,

```
_basis_set_atomic_name
_basis_set_contraction_scheme
_basis_set_atomic_symbol
```

the result is structured differently:

```
data_Gaussian
loop_
    _basis_set_atomic_name
    loop_
     _basis_set_contraction_scheme
    stop_
     _basis_set_atomic_symbol

    hydrogen
        (2)->[2] (2)->[2] (2)->[1] (3)->[2] stop_
    H
    lithium
        (4)->[4] (9,4)->[3,2] (4,3)->[3,2] stop_
    Li
```

In the examples so far, one or more data items have been requested by name. *Star_Base* extends the type of requests that can be made through its own query language. This gives it much of the power of a database query language such as SQL. Three types of query are supported, known as *data*, *conditional* and *branching* requests.

A *data request* is a straightforward generalization of the request by data name. Individual data items may be requested by name, as may individual data blocks or save frames. Wild carding is permitted to generalize the requests. More details are given in Section 5.2.3.2.

A *conditional request* involves one or more conditions; only data items satisfying the conditions are returned. More details are given in Section 5.2.3.3.

A *branching request* applies similar conditions to establish the context in which matching data items occur within the file, but may also apply scoping rules to select among the available contexts. Only data items matching both the conditions imposed on their values *and* the requested scope are returned. It is the existence of such branching conditions that gives *Star_Base* the ability to select data matching the specific requirements of overlying data models. Again, however, it is emphasized that the program itself operates without any semantic awareness of the significance of the data that is implied within the overlaid data model. More details on branching requests are given in Section 5.2.3.4.

5.2.3.2. The *Star_Base* data request

A *data request* is the simplest type of query used to extract single items from a file. It may be formed from any of the following string types:

(i) a **name** string, *e.g.* `_atom_identity_symbol`;

(ii) a **block** string, *e.g.* `data_Gaussian`;

(iii) a **frame** string, *e.g.* `save_methyl`.

In accordance with the principles set out earlier in this chapter, data requests satisfy the following rules:

(i) Requested data items are returned with their associated context (*i.e.* including the headers of any containing data blocks, save frames and loop structures).

(ii) A request for a data block returns all preceding global blocks (since the data block will contain by inheritance all values in the global blocks).

(iii) A request for a save frame also returns the header of the data block encompassing the save frame. All frame-pointer codes are resolved so that if a requested save frame contains pointer codes to other save frames, these are also returned.

(iv) A request for `global_` returns all global blocks, together with all data-block headers in their scope.

(v) The request need not be specified explicitly. Two wild-card characters are permitted. An asterisk (*) represents *any sequence*

references