Star_Base: Accessing STAR File Data

Nick Spadaccini*,† and Sydney R. Hall‡

Department of Computer Science and Crystallography Centre, University of Western Australia, Nedlands 6009, Australia

Received August 4, 1993⊗

Star_Base is ANSI C compliant software for extracting data from a STAR File. It employs a versatile query language to return data as STAR conformant data.

INTRODUCTION

The STAR File[1,2] is a simple universal approach to electronic data exchange and archiving. It is the basis for the Crystallographic Information File[3] (CIF) used by the *Cambridge Crystallographic Data Centre*, *International Centre for Diffraction Data* and *Protein Data Bank* databases for data entry. CIF's are also used for electronic text and data submissions to the journal *Acta Crystallographica*.

There is considerable software for generating data in CIF format but few programs for extracting CIF data. At present *QUASAR*[4] is available for data access and the *CIFtbx*[5] Fortran library is used by programmers developing CIF applications. These two programs are written in Fortran for maximum portability, but this tends to limit their suitability for some applications in modern processing environments. Neither approach provides a query language suitable for accessing databases, such as the *IUCr World Directory of Crystallographers*[6], or makes use of DDL data dictionaries[7] (*CIFtbx* does to a limited extent). However, the most serious limitation of existing access mechanisms is that none can handle the full STAR File syntax, and this precludes their use with applications such as the *Molecular Information File*[8] (MIF).

The *Star_Base* software has been developed in ANSI C to satisfy the need for a general purpose access tool for large STAR Files. It provides the database user with a rich query language for constructing data requests and a close coupling with DDL dictionaries for STAR data and file validation. *Star_Base* parses user defined queries sequentially and returns data to the output stream in the order requested. The user is protected from generating nonconformant *STAR* output files, and data redundancy or conflicts are automatically resolved using the rules of sequential precedence.

STAR_BASE QUERY LANGUAGE

The terminology used in the description of the query language is defined in Appendix I. Where possible in this description, examples are used to illustrate the query syntax. These examples are based on the two STAR data files listed in Appendix II.

The query language is based on user defined requests and scope specifications. Three levels of data requests are supported. They are, in order of increasing complexity, as follows:

1. **data** request	single request for specific data items in the file
2. **conditional** request	single request for data items in the file, conditional on item values
3. **branching** request	combinations of requests 1 and 2 in which data items are requested within defined scopes, conditional on item (or associated item) values

These request types are described separately, but may be applied in any order, number, or combination.

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

a. ***name*** *string*	e.g. _atom_site_code
b. ***block*** *string*	e.g. **_data**_quantum_solution_5
c. ***frame*** string	e.g. **save**_tyrosine

Rules for Data Requests.

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

(ii) A request for a **data block** returns all *global blocks* preceding the data block. Using input file 2 (see Appendix II), a request for "**data_2**" returns

```
global_
_atom_bond_order_convention              simple
_atom_bond_order_convention_source       IUPAC

data_2
```

contents of data block "2"

(iii) A request for a **save frame** also returns the data block encompassing the save frame. In addition, all *frame pointer codes* (see Appendix I) are resolved so that if a requested save frame contains pointer codes to other save frames, these are also returned. Using input file 2 (Appendix II), a request for "**save_ethyl**" returns

† Department of Computer Sciences.
‡ Crystallography Centre.
⊗ Abstract published in *Advance ACS Abstracts,* March 1, 1994.

0095-2338/94/1634-0509$04.50/0

```
data_3

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_identifier_symbol      $ethyl
save_

save_carboxylic_acid
   loop_ _atom_identity_symbol            $R1
save_

loop_
     _reaction_component_symbol           $carboxylic_acid
```

(iv) A request for **global_** returns all global blocks, along with all data block headers in their scope. Using input file 2 (Appendix II), a request for "**global_**" returns

```
global_
_atom_bond_order_convention                simple
_atom_bond_order_convention_source         IUPAC

data_1

data_2

global_
_atom_bond_order_convention_source         CODATA

data_3
```

(v) The request need not be specified explicitly. Two *wild card* characters are permitted. An asterisk (*) represents *any sequence* of characters and a question mark (?) represents *any single* character.

(vi) A request for looped data will return these items *in the order requested* along with the encompassing loop structure. *Note the difference between the following two requests.* Using input file 1 (Appendix II), a request for

"**_basis_set_atomic_name**
_basis_set_atomic_symbol
_basis_set_contraction_scheme"

returns

```
data_Gaussian

loop_
     _basis_set_atomic_name
     _basis_set_atomic_symbol
     loop_
          _basis_set_contraction_scheme
     stop_

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

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

whereas a request for (*note the change of order*)

"**_basis_set_atomic_name**
_basis_set_contraction_scheme
_basis_set_atomic_symbol"

returns

```
data_Gaussian
loop_
     _basis_set_atomic_name
     loop_
          _basis_set_contraction_scheme
     stop_
     _basis_set_atomic_symbol

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

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

(vii) A request for data in a **save frame** returns those items plus any structural information. Using input file 2, a request for "**_atom_identity_node_**" returns

```
data_1
loop_
     loop_
          _atom_identity_node       A1 A2 A3 A4

data_2
loop_     _atom_identity_node       1 2 3 4

data_3
save_methyl
   loop_ _atom_identity_node        1 2
save_

save_ethyl
   loop_ _atom_identity_node        1 2 3
save_

save_R1
   loop_ _variable_identifier_symbol      $methyl      $ethyl
save_

save_carboxylic_acid
   loop_
        _atom_identity_node         1  2  3  4
   loop_

        _atom_identity_symbol       $R1
save_

loop_
     _reaction_component_symbol $carboxylic_acid
```

(viii) If the requested data item includes a save frame pointer, the referenced save frame is returned *intact*. All other pointers contained within the returned data are resolved. Using input file 2, a request for "**_atom_identity_symbol**" returns

```
data_1
loop_
        _atom_identity_symbol     B1    B2    B3    B4

data_2
loop_
        _atom_identity_symbol     C  C  N  O,S

data_3
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
```

(ix) A request for a data item in a global data block will also return the data block headers within the scope of the global block. Using input file 2, a request for

"**_atom_bond_order_convention**"

returns

```
global_
_atom_bond_order_convention        simple

data_1

data_2
_atom_bond_order_convention        RPN

data_3
```

Chart 1

```
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_comments_index
          _basis_set_atomic_energy
     stop_

     hydrogen    H          1      1.0079
               (2)->[2]     {1:}   PKC1.1.1    R44    .      .          -0.485813
               (3)->[2]     {2:1}  PKC1.23.1   R75    R75    C13,C19    -0.496979
          stop_

          lithium    Li     3      6.94
               (4)->[4]     {1:}             PKC3.1.1    R44    .    .      -7.376895
               (9,4)->[3,2] {7:2:1,3:1}      PKC3.9.1    R2     R98  C77    -7.431735
stop_
```

Chart 2

```
if_ data_1
   scope_data_block_
       if_ _audit_source_creation_date ~< 90     # if before 1990 return all atom info
       _atom*                                    # default scope inherited from above
       endif_
   endscope_
endif_
```

```
global_
_atom_bond_order_convention        simple
_atom_bond_order_convention_source IUPAC

data_1
loop_
     _atom_identity_node
     _atom_identity_symbol
     loop_
          _atom_bond_node_1
          _atom_bond_node_2
          _atom_bond_order
     stop_

     A1         B1     1      2      sin    stop_
     A2         B2     1      6      dou    30    40    trip    stop_
     A3         B3     1      7      sin    stop_
     A4         B4     2      3      dou    stop_
loop_
     _atom_attributes_node
     _atom_attributes_charge
     _atom_attributes_isotope      1  +1  12    6  0  15

global_
_atom_bond_order_convention_source       CODATA
```

(ix) The default *scope* of a *data request* is the entire input file. Control of the search scope is only possible with *branching requests*.

2. Conditional Request. A *conditional request* is a *data request* involving one or more conditions. Only data items satisfying the condition are returned. The state of a conditional request is TRUE if the condition is satisfied, and FALSE if the condition is not satisfied *or* if the data name is not present in the input file. The scope of a conditional request is the input file (only the scope of a *branching request* may be specified).

A ***conditional request*** may have the following constructions.

<data request>
<data request> <operator> <text string>
<conditional request> **&** *<conditional request>*
<conditional request> | *<conditional request>*
! *<conditional request>*

<data request> <operator> <text string>. This request returns data which have values satisfying the condition *<operator><text string>*. A *<text string>* is any sequence of the printable characters (ASCII 32-126). If this string contains blank characters (ASCII 32), it must be contained in single quotes (ASCII 39) or double quotes (ASCII 34). If the condition is TRUE, the *<data request>* items are returned.

The standard ***operator*** characters =, !, <, and > and the digraphs !=, <=, and >=, are used to test *numerical* relationships. For these operators the *<text string>* is interpreted as a number.

Two additional operator charcters are used to test **text data** items. They are the tilde (ASCII 126) to signal *string equality* and the question mark (ASCII 63) to signal *substring containment*. The diagraphs ~=, ?=, ~<, and ~> are used to test if the value of the data item(s) identified in the *<data request>* is identical to, contains, is less than, or is greater than the *<text string>*. The trigraphs ~!=, ?!=, ~<=, and ~>= are used to test if text data are not identical to, not

Chart 3

```
if_ data_1
scope_data_block_
 if_ _atom_identity_node ~= A2
 scope_loop_packet_
       if_ _atom_bond_order ~= dou
       scope_loop_structure_
       _*
       endscope_
       endif_
 endscope_
 endif_
endscope_
endif_
```

```
data_1
loop_
     _atom_identity_node
     _atom_identity_symbol
     loop_
          _atom_bond_node_1
          _atom_bond_node_2
          _atom_bond_order
     stop_

     A1          B1      1      2      sin     stop_
     A2          B2      1      6      dou     30     40     trip     stop_
     A3          B3      1      7      sin     stop_
     A4          B4      2      3      dou     stop_
```

Chart 4

```
loop_
    _atom_identity_node
    _atom_identity_symbol
         loop_
             _atom_bond_node_1
             _atom_bond_node_2
              atom_bond_order
                              A1 B1  1  2 sin                  stop
                              A2 B2  1  6 dou     30 40 trip   stop
                              A3 B3  1  7 sin                  stop_
```

containing, are not less than or equal, and are not greater than or equal.

Using input file 2 (Appendix II), a request for "**_*_order ~= trip**" returns

```
data_1
loop_
     loop_
          _atom_bond_order stop_  trip  stop_
```

Using input file 1, a request for "**_basis_set_function_exponent < 1.0e-01**" returns

```
data_Gaussian

loop_
     loop_
          loop_
               _basis_set_function_exponent   stop_    stop_

               4.7192775E-02       stop_
               0.076663
               0.028643
               0.07201
               0.02370             stop_  stop_
```

<conditional request> & <conditional request>. This construction allows for the conjunction of conditionals. The **&** operator requires the intersection of the sets of data which individually satisfy the conditions to be a non-empty set. Using input file 1, a request for "**_basis_set_function_exponent > 1.0 & _basis_set_function_exponent < 1.5**" returns

```
data_Gaussian
loop_
     loop_
          loop_
               _basis_set_function_exponent  stop_  stop_

               1.0514394E+00       stop_
               1.15685
               1.488                     stop_   stop_
```

The conjunction of conditionals based on different data names is the empty set. The request "**_atom_bond_node = 1 & _atom_bond_symbol = C**" returns no data!

<conditional request> | <conditional request>. This construction allows for the disjunction of conditionals. The **|** operator results in the union of the data sets which individually satisfy the conditions. Using input file 2, a request for

"**_reaction_pathway_reactant > 1.1 |**
_audit source_creation_method ?= man"

returns

```
data_3
loop_
     _reaction_pathway_reactant  1.2

data_1
_audit_source_creation_method    'manual entry'
```

Using input file 1, a request for "**_basis*name ?= ydro | _basis*symbol ~= H**" returns

```
data_Gaussian
loop_
     _basis_set_atomic_name
     _basis_set_atomic_symbol
                         hydrogen     H
```

! <conditional request>. This construction returns the set complement. All data *except* those which satisfy the condition are returned. The universal set is defined as the input file. Using input file 1, a request for "**! _basis_set_function_***" returns that given in Chart 1.

3. Branching Request. A *branching request* is the most powerful query sequence for extracting data from a STAR

Chart 5

```
loop_
    _molecular_fragment
    loop_
        _atom_identity_node
        _atom_identity_symbol
                A5    1    C     3     C     4     O     stop_
                B5    1    C     2     N     3     C     4     N     stop_

loop_
    _atom_identity_node
    _atom_identity_symbol
    loop_
        _atom_bond_node_1
        _atom_bond_node_2
        _atom_bond_order
                        1 B1  1 2 sin                    stop
                        2 B2  1 6 dou      3 4 trip      stop_
                        3 B3  1 7 sin                    stop_
```

File. It can include *conditional* statements, *scope* settings, *data request* statements, and some other commands.

The two request types described earlier, *data request* and *conditional request*, represent a query shorthand in that their scope must extend over the entire input file. In a *branching request* the scope of the search can be specified according to the result of a *conditional* statement. The simpler *data* and *conditional requests* are appropriate for many data searches, but *branching requests* will be needed for more selective and complex query situations.

A ***branching request*** has the following basic sequence: A

```
if_          <condition>    <branch request>
else_                       <branch request>
unknown_                    <branch request>
endif_
```

condition is identical to a *conditional request except* that the satisfied data items are *not* returned. The *condition* statement is used only to establish the validity, or otherwise, of executing the *branch request*. If the *condition* is TRUE the **if_** *branch request* is executed; if it is FALSE the **else_** *branch request* is executed; or if it cannot be tested because a data name is UNKNOWN (i.e. not declared in the current scope), the **unknown_** *branch request* is executed. The **unknown_** statement provides for the case when a *condition* cannot be determined as either TRUE or FALSE. If the **unknown_** branch is not included in a *branching request*, then the default truth value is set as FALSE (and the **else_** branch is executed).

The default truth value can be made to be to TRUE by use of the **assume_true_** operator. This operator forces a *conditional* to be TRUE when the status is UNKNOWN (the operator is ignored if the status is FALSE). It provides a short hand when the same *branch request* applies whether the *conditional* is TRUE or UNKNOWN. The assume_true_ operator has the syntax

_assume_true_ (<*condition*>)

A *data request* will return the complete contents of save frames whose frame pointers are referenced in the requested data items. A *condition* does NOT force this expansion and operates only on the data in the current scope. The *condition* can be made to operate over expanded data if directed to by a **scope_save_frame_** sequence (see below).

A ***branch request*** has three possible forms: a *conditional request*; another *branching request*; **scope_**<*scope setting*> <*branch request*> **endscope_**.

scope_<*scope setting*> specifies the range of data to be searched in the input file. The effect of this setting is closed with the **endscope_** statement.

The permitted values of *scope setting* are

data_item_	restricts the *branch request* to the data items in the *condition*
loop_packet_	restricts the *branch request* to the contents of the loop packet in which data match the *condition*
loop_structure_	restricts the *branch request* to the loop structure in which data match the *condition*
save_frame_	restricts the *branch request* to the contents of the save frame in which data match the *condition*
data_block_	restricts the *branch request* to the contents of the data block in which data match the *condition*
file_	the *branch request* applies to the contents of the file containing data matching the *condition* (this is the default setting)

The default scope is invoked when a **scope_**<*scope setting*> is **not** specified. In such a case the scope of the *branch request* is the same as that of the *conditional*. In example 1 below the "_atom*" *branch request* is within the same scope as the *conditional* which precedes it (i.e. the *data block*).

Examples of *branching requests.* Here are some examples which illustrate the application of *branching request* options. An input request sequence is given, followed by the output data. Example 3 uses input file 1; all others use input file 2 (Appendix II).

Example 1: The **scope_data_block_** statement forces the scope of the second condition to be applied only to those data items block **data_1** (see Chart 2).

Example 2: This example is a variant on example 1 in which the scope is reduced to only those data items satisfying the second condition.

```
if_ data_1
scope_data_block_
     if_ _audit_source_creation_date ~< 90
     scope_data_item_
     _*
     endscope_
     endif_
endscope_
endif_
```

```
data_1
_audit_source_creation_date          89-11-22
```

Example 3: In this example the conditional data item **_basis_set_contraction_Gaussian** is unknown. Note how the **unknown_** control is used to output related data items.

Chart 6

Input File 1

```
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_comments_index
        _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_

   (3)->[2]   {2:1}             PKC1.23.1  R75 R75 C13,C19     -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 C77          -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_
```

Input File 2

```
global_
   _atom_bond_order_convention           simple
   _atom_bond_order_convention_source    IUPAC

data_1
   _audit_source_creation_method         'manual entry'
   _audit_source_creation_date           89-11-22

   loop_
         _atom_identity_node
         _atom_identity_symbol
         loop_
               _atom_bond_node_1
               _atom_bond_node_2
               _atom_bond_order
         A1 B1         1  2  sin                  stop_
         A2 B2         1  6  dou    30 40 trip    stop_
         A3 B3         1  7  sin                  stop_
         A4 B4         2  3  dou                  stop_
   loop_
         _attached_hydrogen_node
         _attached_hydrogen_count
                       1 0    2 1    3 0    4 1    5 1    6 1    7 2    8 2
   loop_
         _atom_attributes_node
         _atom_attributes_charge
         _atom_attributes_isotope     1 +1 12    6 0  15
```

Chart 6 (Continued)

```
data_2
   _atom_bond_order_convention          RPN

   loop_
         _atom_identity_node
         _atom_identity_symbol   1 C   2 C   3 N   4 O,S
   loop_
         _atom_bond_node_1
         _atom_bond_node_2
         _atom_bond_order         1 2 sin   2 3 sin   2 4 sin,dou

global_
   _atom_bond_order_convention_source          CODATA

data_3
   _table_of_contents
;
   This example illustrates the description of a simple chemical reaction
   in which one of the reactants and the product are expressed as generic
   structures.
;
   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
   loop_
         _reaction_pathway_reactant
         _reaction_pathway_product
                                   1.1        3.1
                                   1.2        3.2
```

```
if_ basis_set_atomic_name ~= hydrogen
  scope_loop_packet_
  if_ _basis_set_contraction_Gaussian ?= (3)
     scope_loop_packet_
        _*
     endscope_
  unknown_
     _*contraction*
  endif_
  endscope_
endif_
```

```
data_Gaussian
loop_
     loop_
          _basis_set_contraction_scheme
          _basis_set_funct_per_contraction
     stop_
               (2)->[2] {1:}
               (3)->[2] {2:1}
     stop_
```

Example 4: Scopes can be contracted and expanded. This is a request for the data block **data_1** containing an item **_atom_identity_node** with a value of "**A2**"; to then reduce the search scope to the loop packet containing the compliant data item; and to then test that the item **_atom_bond_order** in this packet is "**dou**". If it does, then return all items and packets in the loop structure (see Chart 3).

Example 5: This search returns data containing save frame pointer codes. The request is that the data block **data_3** contains an item **_reaction_component_number** equal to 1; reduce the search scope to the loop packet containing the compliant data item and return only the **_attached_hydrogen_count** item from that packet. This is satisfied in **save_methyl**, **save_ethyl**, and **save_carboxylic_acid**. In order that these save frames may be referenced in the returned STAR

File, the additional data are output as follows: **save_R1** because it contains the pointers **$methyl** and **$ethyl**; **_atom_identity_symbol** which is set to **$R1**; and **_reaction_component_symbol** which is set to **$carboxylic_acid**.

```
if_ data_3
scope_data_block_
 if_ _reaction_component_number = 1
 scope_loop_packet_
      _attached_hydrogen_count
 endscope_
 endif_
endscope_
endif_
```

```
data_3

save_methyl
loop_ _attached_hydrogen_count     3
save_

save_ethyl
loop_ _attached_hydrogen_count     3       3
save_

save_R1
loop_ _variable_identifier_symbol  $methyl  $ethyl
save_

save_carboxylic_acid
loop_ _atom_identity_symbol         $R1
loop_ _attached_hydrogen_count      0       0       1
save_

loop_ _reaction_component_symbol    $carboxylic_acid
```

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of the Australian Research Council for 1992 Grant 336 583. We wish to acknowledge the efforts of Andrew Hall who converted our concepts into ANSI C software. His diligence and patience enabled all of the ideas and objectives to converge into a practical and robust program. We also thank Mark Favas, Brian McMahon, and all those who tested the *Star_Base* software.

APPENDIX I: GLOSSARY OF SB TERMS

Scope is a contiguous region of data over which an query operator is valid. Six possible scopes are used in sb: a file; a data block; a save frame (and associated pointers); a loop structure; a loop packet; a data item.

Loop structure is the data contained within a single data loop or a discrete set of nested loops. For example the shaded box in Chart 4 highlights data in a loop structure.

Loop packet is the data for a single set of looped data names. A loop structure is made up of one or more packets of data. The data within a loop packet returned by **sb** will depend on the loop level of the requested data item. Two examples in Chart 5 below illustrate this. In each case the highlighted loop packet contains the data item **_atom_identity_node** with a requested value of **"2"**.

APPENDIX II: EXAMPLE INPUT FILES

The examples given in the body of the paper are based on the two input sample files given in Chart 6.

REFERENCES AND NOTES

(1) Hall, S. R. The STAR File: A New Format for Electronic Data Transfer and Archiving. *J. Chem. Inf. Comput. Sci.* **1991**, *31*, 326–333.
(2) Hall, S. R.; Spadaccini, N. The STAR File: Detailed Specifications. *J. Chem. Inf. Comput. Sci.*, previous paper in this issue.
(3) Hall, S. R.; Allen, F. H.; Brown, I. D. The Crystallographic Information File (CIF): A New Standard Archive File for Crystallography. *Acta Crystallogr.* **1991**, *A47*, 655–685.
(4) Hall, S. R.; Sievers, R. CIF Appl. I. *QUASAR*: for extracting CIF data. *J. Appl. Crystallogr.*, **1993** *26*, 469–473.
(5) Hall, S. R. CIF Applications. IV. *CIFtbx* a toolbox for Manipulating CIF's. *J. Appl. Crystallogr.* **1993**, *26*, 482–494.
(6) Epelboin, Y. Keywords for the Database of Crystallographers and the World Directory. *Acta Crystallogr.* **1992**, *A48*, 949–954.
(7) Hall, S. R.; Cook, A. P. F. STAR Dictionary Definition Language: Initial Specifications. *J. Chem. Inf. Comput. Sci.*, to be submitted.
(8) Allen, F. H.; Barnard, J. M.; Cook, A. F. P.; Hall, S. R. The Molecular Information File (MIF): Initial Specifications. *J. Chem. Inf. Comput. Sci.*, to be submitted.