

Tripes Mol2 File Format

A Mol2 file (.mol2) is a complete, portable representation of a SYBYL molecule. It is an ASCII file that contains all the information needed to reconstruct a SYBYL molecule. This section describes the Mol2 file format in detail so you may write your own programs to read and write Mol2 files.

Mol2 files are written in a *free format* to avoid the restrictions created by fixed format files.

Additional Information:

- Glossary and Format Restrictions on page 3
 - How to Name SYBYL Objects on page 3
- Sample Mol2 File on page 5
- SYBYL Atom Types on page 7

The following record types are currently available in Mol2 files:

- @<TRIPES>ALT_TYPE
- @<TRIPES>ANCHOR_ATOM
- @<TRIPES>ASSOCIATED_ANNOTATION
- @<TRIPES>ATOM
- @<TRIPES>BOND
- @<TRIPES>CENTER_OF_MASS
- @<TRIPES>CENTROID
- @<TRIPES>COMMENT
- @<TRIPES>CRYSIN
- @<TRIPES>DATA_FILE
- @<TRIPES>DICT
- @<TRIPES>EXTENSION_POINT
- @<TRIPES>FF_PBC
- @<TRIPES>FFCON_ANGLE
- @<TRIPES>FFCON_DIST
- @<TRIPES>FFCON_MULTI
- @<TRIPES>FFCON_RANGE
- @<TRIPES>FFCON_TORSION

- @<TRIPOS>LINE
- @<TRIPOS>LSPLANE
- @<TRIPOS>MOLECULE
- @<TRIPOS>NORMAL
- @<TRIPOS>QSAR_ALIGN_RULE
- @<TRIPOS>RING_CLOSURE
- @<TRIPOS>ROTATABLE_BOND
- @<TRIPOS>SEARCH_DIST
- @<TRIPOS>SEARCH_OPTS
- @<TRIPOS>SET
- @<TRIPOS>SUBSTRUCTURE
- @<TRIPOS>U_FEAT
- @<TRIPOS>UNITY_ATOM_ATTR
- @<TRIPOS>UNITY_BOND_ATTR

Glossary and Format Restrictions

The following terms are used throughout this document.

Record Type Indicator (RTI)	A printable ASCII string with an at sign (@) in column 1, terminated by an end of line. It is used to indicate the type of data which follows in a .mol2 file.
Data Line	A printable ascii string terminated by an end of line. Lines may be continued with a back slash (\) and may not begin with '@' or '#'.
Data Record	A data record consists of one or more data lines.
Field	An item of interest in a data record. May be a string, integer, or a real. Fields are usually separated by spaces or tabs.
Section	A section of a .mol2 file. A section begins with an RTI and is terminated by another RTI or end of file. Data records within a section all have the same format.
String	A collection of printable ASCII characters containing no white space.
Comment	A printable ascii string beginning with a hash or pound sign (#) in column 1 and terminated by end of line. Comment lines may not be continued.
White Space	Tabs, spaces, or any combination of the two.
Blank Line	Any line containing only white spaces. These are ignored by the MOL command.
****	Used to indicate that a non-optional string field is empty.

How to Name SYBYL Objects

Molecules	Names can be arbitrarily long and complex, containing any characters (alphanumeric, underscore,...). Enclose the name in double quotes (" ") if it starts with a numeric character, has special characters, or a space.
-----------	---

Atoms, Substructures, Sets, Features	<p>There is no limit on the number of characters in a name. Tripos recommends 7 characters for atoms and substructures, and 31 for sets and features. Names must start with an alphabetic character, but are case insensitive (characters are held internally in uppercase). Names may contain digits, underscores (<code>_</code>), and apostrophes (<code>'</code>) in any position after the first.</p> <ul style="list-style-type: none">• CA CA12—Valid atom names• 1C C(1)—Invalid atom names <p>Note: Only set names are required to be unique.</p>
Chains	<p>Names are restricted to 4 characters and must start with an alphanumeric character. If the name begins with an alphabetic character, the following characters can be A-Z, 0-9, <code>_</code>, <code>\$</code> or <code>'</code>. If the name begins with a numeric character, the following characters must also be numeric. They may contain underscores in any position after the first position and before the last position.</p>

Sample Mol2 File

The lines of a .mol2 file for benzene have been numbered for easy reference during the following discussion.

```
1      #      Name: benzene
2      #      Creating user name: tom
3      #      Creation time: Wed Dec 28 00:18:30 1988
4
5      #      Modifying user name: tom
6      #      Modification time: Wed Dec 28 00:18:30 1988
7
8      @<TRIPOS>MOLECULE
9      benzene
10     12 12 1 0      0
11     SMALL
12     NO_CHARGES
13
14
15     @<TRIPOS>ATOM
16     1      C1      1.207  2.091  0.000  C.ar  1      BENZENE0.000
17     2      C2      2.414  1.394  0.000  C.ar  1      BENZENE0.000
18     3      C3      2.414  0.000  0.000  C.ar  1      BENZENE0.000
19     4      C4      1.207  -0.697  0.000  C.ar  1      BENZENE0.000
20     5      C5      0.000  0.000  0.000  C.ar  1      BENZENE0.000
21     6      C6      0.000  1.394  0.000  C.ar  1      BENZENE0.000
22     7      H1      1.207  3.175  0.000  H      1      BENZENE0.000
23     8      H2      3.353  1.936  0.000  H      1      BENZENE0.000
24     9      H3      3.353  -0.542  0.000  H      1      BENZENE0.000
25     10     H4      1.207  -1.781  0.000  H      1      BENZENE0.000
26     11     H5      -0.939  -0.542  0.000  H      1      BENZENE0.000
27     12     H6      -0.939  1.936  0.000  H      1      BENZENE0.000
28     @<TRIPOS>BOND
29     1      1      2      ar
30     2      1      6      ar
31     3      2      3      ar
32     4      3      4      ar
33     5      4      5      ar
34     6      5      6      ar
35     7      1      7      1
36     8      2      8      1
37     9      3      9      1
38     10     4      10     1
39     11     5      11     1
40     12     6      12     1
41     @<TRIPOS>SUBSTRUCTURE
42     1      BENZENE1      PERM  0      ****  ****  0      ROOT
```

Comments

Lines 1,2,3,5 and 6 are comments. They are ignored by the MOL command and are provided only for your convenience. These lines are added by SYBYL when a .mol2 file is written out. They contain the molecule name and information about the time the molecule was created and last modified.

Blank Lines

Lines 4,7,13, and 14 are blank and will be ignored by the MOL command.

Record Type Indicators (RTI)

Lines 8, 15, 28, and 41 in the example are RTIs. They divide the file into 4 sections. A section begins with a RTI and ends at the next RTI or the end of the file. Each section contains a different type of information needed to reconstruct the molecule. In this example, section one (lines 8-14) contains information global to the molecule. The second section (lines 15-27) contains all the information about atoms in the molecule. Section three (lines 28-40) contains all the bond information and section four (lines 41 and 42) contains information necessary to reconstruct substructures.

Data Records

Lines 9-12, 16-27, 29-40, and 42 are all data records. The format of a data record depends on that section of the file in which it resides.

SYBYL Atom Types

The chemical atom types used in SYBYL are listed below with their mnemonic representation and a brief description.

C.3	sp3 carbon
C.2	sp2 carbon
C.ar	aromatic carbon
C.1	sp carbon
N.3	sp3 nitrogen
N.2	sp2 nitrogen
N.1	sp nitrogen
O.3	sp3 oxygen
O.2	sp2 oxygen
S.3	sp3 sulfur
N.ar	aromatic nitrogen
P.3	sp3 phosphorous
H	hydrogen
Br	bromine
Cl	chlorine
F	fluorine
I	iodine
S.2	sp2 sulfur
N.pl3	trigonal planar nitrogen
LP	lone pair
Na	sodium
K	potassium
Ca	calcium
Li	lithium
Al	aluminum
Du	dummy
Du.C	dummy with the weight of carbon
Si	silicon
N.am	amide nitrogen
S.o	sulfoxide sulfur
S.o2	sulfone sulfur
N.4	sp3 positively charged nitrogen
O.co2	oxygen in carboxylate or phosphate group
C.cat	carbocation, used only in a guadinium group
H.spc	hydrogen in SPC water model
O.spc	oxygen in SPC water model
H.t3p	hydrogen in TIP3P water model
O.t3p	oxygen in TIP3P water model
ANY	any atom
HEV	heavy (non H) atom
HET	heteroatom (N, O, S, P)
HAL	halogen
Mg	magnesium
Cr.oh	hydroxy chromium
Cr.th	chromium
Se	selenium
Fe	iron
Cu	copper
Zn	zinc
Sn	tin
Mo	molybdenum
Mn	manganese
Co.oh	hydroxy cobalt

@<TRIPOS>ALT_TYPE

Each data record associated with this RTI contains the information necessary to reconstruct user defined alternate atom types. The data records following this RTI consist of two lines. The first line contains the type specification and the second consists of the type set name followed by a list of atom IDs and atom type mnemonics. All fields must be separated by a single space.

Format:

`type_specification`

`type_set_name atom_id type_mnemonic atom_id
type_mnemonic ...`

- `type_specification` = `koll_uni_alt_type_set` or `koll_all_alt_type_set`.
- `type_set_name` (string) = the name of the type set: `koll_uni` or `koll_all`
- `atom_id` (integer) = the ID number of the atom belonging to the type set.
- `type_mnemonic` (string) = the atom type assigned to the atom.

Examples:

```
KOLL_UNI_ALT_TYPE_SET  
KOLL_UNI 1 O2 6 NT 2 O2 10 NT 20 NT  
KOLL_ALL_ALT_TYPE_SET  
KOLL_ALL 8 N*
```

Atoms 1 and 2 are assigned the type O2, and atoms 6, 10 and 20 are assigned the type NT. The atom types belong to the Kollman united-atom parameter set. In the second example, atom 8 is assigned the type N*.

@<TRIPLOS>ANCHOR_ATOM

The data record associated with this RTI consists of a single line which contains the atom ID of the anchor atom.

Format:

atom_id

- atom_id (integer) = the ID number of the anchor atom used in SEARCH.

Example:

5

Atom 5 is the anchor atom.

@<TRIPOS>ASSOCIATED_ANNOTATION

Each data record associated with this RTI contains a unique identifier followed by the SPL description of the annotation. The sequence of data records is repeated for additional annotation objects.

Format:

```
feature_name
object_spl
END#OF#OBJECT
[feature_name
object_spl
END#OF#OBJECT] . . .
```

- feature_name = a string identifying the feature.
- object_spl = the SPL necessary to reconstruct the annotation object
- END#OF#OBJECT = the string “END#OF#OBJECT“, used as a terminator

The SPL generated by SYBYL for annotation objects is written so that each recreated object will have a unique ID string, no matter when it is created. The first line of each object’s SPL generates a new ID and assigns it to an SPL variable specifically set aside for this use (you should not use the variable ANN_TEMP_ID for anything other than SPL to restore saved annotation objects).

The generation of unique IDs is essential to the correct behavior of the annotation objects. Conflicting ID strings can and will cause unexpected results.

Example:

```
OB00003
SETVAR ANN_TEMP_ID %ANN_GENERATE_ID(%ANN_TARGET_PLANE())
ANNOTATE ARROW CREATE $ANN_TEMP_ID \
  ANN!SYBYL!PLANE C -3.261262 -6.262136 \
  ANN!SYBYL!PLANE C 3.887888 -6.310680
ANNOTATE ARROW SET STYLE $ANN_TEMP_ID HEAD Filled
ANNOTATE ARROW SET STYLE $ANN_TEMP_ID TAIL Filled
ANNOTATE ARROW SET STYLE $ANN_TEMP_ID BODY Filled
ANNOTATE ARROW SET COLOR $ANN_TEMP_ID HEAD White
ANNOTATE ARROW SET COLOR $ANN_TEMP_ID TAIL White
```

```
ANNOTATE ARROW SET COLOR $ANN_TEMP_ID BODY White
ANNOTATE ARROW SET SIZE $ANN_TEMP_ID HEAD 0.357495
ANNOTATE ARROW SET SIZE $ANN_TEMP_ID TAIL 0.357495
ANNOTATE ARROW SET SIZE $ANN_TEMP_ID BODY 0.357495
END#OF#OBJECT
OB00016
SETVAR ANN_TEMP_ID %ANN_GENERATE_ID(%ANN_TARGET_PLANE())
ANNOTATE TEXT CREATE $ANN_TEMP_ID \
  ANN!SYBYL!PLANE C 4.276677 -6.456311
Some sample text
.
ANNOTATE TEXT SET COLOR $ANN_TEMP_ID White
ANNOTATE TEXT SET FONT_FAMILY $ANN_TEMP_ID "Helvetica"
ANNOTATE TEXT SET SIZE $ANN_TEMP_ID 12
ANNOTATE TEXT SET WEIGHT $ANN_TEMP_ID MEDIUM
ANNOTATE TEXT SET SLANT $ANN_TEMP_ID ROMAN
ANNOTATE TEXT SET ALIGNMENT $ANN_TEMP_ID LEFT
END#OF#OBJECT
```

Description of two annotation objects: an arrow and a text string.

@<TRIPOS>ATOM

Each data record associated with this RTI consists of a single data line. This data line contains all the information necessary to reconstruct one atom contained within the molecule. The atom ID numbers associated with the atoms in the molecule will be assigned sequentially when the .mol2 file is read into SYBYL.

Format:

```
atom_id atom_name x y z atom_type [subst_id  
  [subst_name [charge [status_bit]]]]
```

- atom_id (integer) = the ID number of the atom at the time the file was created. This is provided for reference only and is not used when the .mol2 file is read into SYBYL.
- atom_name (string) = the name of the atom.
- x (real) = the x coordinate of the atom.
- y (real) = the y coordinate of the atom.
- z (real) = the z coordinate of the atom.
- atom_type (string) = the SYBYL atom type for the atom.
- subst_id (integer) = the ID number of the substructure containing the atom.
- subst_name (string) = the name of the substructure containing the atom.
- charge (real) = the charge associated with the atom.
- status_bit (string) = the internal SYBYL status bits associated with the atom. ***These should never be set by the user.*** Valid status bits are DSPMOD, TYPECOL, CAP, BACKBONE, DICT, ESSENTIAL, WATER and DIRECT.

Example:

```
1 CA -0.149 0.299 0.000 C.3 1 ALA1 0.000  
BACKBONE|DICT|DIRECT  
1 CA -0.149 0.299 0.000 C.3
```

In the first example the atom has ID number 1. It is named CA and is located at (-0.149, 0.299, 0.000). Its atom type is C.3. It belongs to the substructure with ID 1 which is named ALA1. The charge associated with the atom is 0.000 and the SYBYL status bits associated with the atom are BACKBONE, DICT, and DIRECT. Example two is the minimal information necessary for the MOL command to create an atom.

@<TRIPOS>BOND

Each data record associated with this RTI consists of a single data line which contains all the information necessary to reconstruct one bond in the molecule.

Format:

```
bond_id origin_atom_id target_atom_id bond_type  
[status_bits]
```

- bond_id (integer) = the ID number of the bond at the time the file was created. This is provided for reference only and is not used when the .mol2 file is read into SYBYL.
- origin_atom_id (integer) = the ID number of the atom at one end of the bond.
- target_atom_id (integer) = the ID number of the atom at the other end of the bond.
- bond_type (string) = the SYBYL bond type (see below).
- status_bits (string) = the internal SYBYL status bits associated with the bond. *These should never be set by the user.* Valid status bit values are TYPECOL, GROUP, CAP, BACKBONE, DICT and INTERRES.

Bond Types:

- 1 = single
- 2 = double
- 3 = triple
- am = amide
- ar = aromatic
- du = dummy
- un = unknown (cannot be determined from the parameter tables)
- nc = not connected

Example:

```
5 4 9 am BACKBONE | DICT | INTERRES  
5 4 9 am
```

The bond in the first example has ID number 5 and connects atoms 4 and 9. It is an amide bond. The status bits indicate the bond is part of the backbone chain, joins two residues, and that a dictionary was used when creating the molecule. The second example is a minimal representation of the same bond.

@<TRIPOS>CENTER_OF_MASS

Each data record associated with this RTI consists of two data lines. The first contains the name of the center of mass and any comment which may be associated with it. The second data line contains the ID number of the dummy atom representing the center of mass and the ID numbers of the atoms used to define it.

Format:

```
center_of_mass_name [comment]
```

```
cmass_atom_id atom_set_id
```

- center_of_mass_name (string) = name of the center of mass.
- comment (strings to the end of line) = the associated comment
- cmass_atom_id(integer) = ID number of the dummy atom representing the center of mass
- atom_set_id(integer) = ID number of the set containing the atoms used to define the center of mass

Examples:

```
AL_O2C2_2 center of cyclic system  
36 6
```

The center of mass is named AL_O2C2_2 and its comment reads “center of cyclic system”. The center of mass atom has rank 36 in the atom list. The ID number of the set containing the atoms used to define the center of mass is 6.

@<TRIPOS>CENTROID

Each data record associated with this RTI consists of two data lines. The first contains the name of the centroid and any comment which may be associated with it. The second data line contains the ID number of the dummy atom representing the centroid and the ID numbers of the atoms used to define it.

Format:

centroid_name [comment]

cent_atom_id atom_set_id

- centroid_name (string) = name of the centroid.
- comment (strings to the end of line) = the comment associated with the centroid.
- cent_atom_id (integer) = ID number of the dummy atom representing the centroid.
- atom_set_id (integer) = ID number of the set containing the atoms used to define the centroid.

Example:

CENTRO pyridine centroid

48 1

The centroid atom is named CENTRO and its comment reads “pyridine centroid”. The centroid atom has rank 48 in the atom list. The ID number of the atom used to define the centroid is 1.

@<TRIPOS>COMMENT

Note: This RTI has not been implemented. Instead comments appear in the file immediately above the @<TRIPOS>ATOM line.

Format:

string

- string = a comment in free format

Examples:

A very special molecule

@<TRIPOS>CRYSIN

Each data record associated with this RTI consists of a single data line. The data line contains 6 crystallographic cell constants followed by the space group number and space group setting.

Format:

cell cell cell cell cell cell space_grp setting

- cell (real) = one of the 6 crystallographic cell constants.
- space_grp (integer) = the space group number.
- setting (integer) = defines the axial orientation with respect to the standard setting defined in the International Tables for X-Ray Crystallography. Setting numbers are documented in SYBYL's Crystal Tools Manual (Appendix: Space Group Tables).

Example:

```
12.312000 4.959000 15.876000 90.000000 99.070000 90.000000  
4 1
```

The first six real numbers are the crystallographic cell constants (a, b, c, α , β , γ). The space group number is 4 and the space group setting is 1.

@<TRIPOS>DATA_FILE

Each data record associated with this RTI consists of a single data line. It contains the file specification, the data class and data type of the file. It is your responsibility to ensure the file exists in the proper place. If the file does not exist, the current working directory is checked, and if the file is not found, a message will be printed.

Format:

file_spec data_class data_type

- file_spec (string) = the path name and the filename without extension.
- data_class (integer) = the class of data in the file referred by file_spec.
- data_type (integer) = the type of data in the file referred to by file_spec.

Example:

```
/usr/sol/tom/EXSEARCH. 1 0
```

A group of search files named EXSEARCH.

@<TRIPES>DICT

Each data record associated with this RTI consists of a single data line which contains the dictionary type and dictionary name. It is your responsibility to ascertain that the dictionary exists.

Format:

dict_type dict_name

- dict_type (string) = the dictionary type: biopolymer, protein, nucleic_acid, saccharide
- dict_name (string) = the name of the dictionary as found in the default location for dictionaries: MACROMOL, PROTEIN, BIGPRO, DNA, RNA, SUGAR

Example:

PROTEIN PROTEIN

The dictionary is a general BIOPOLYMER dictionary and it is named MACROMOL.

@<TRIPOS>EXTENSION_POINT

Each data record associated with this RTI consists of two data lines. The first contains the name of the extension point and any comment which may be associated with it. The second data line contains the ID number of the dummy atom representing the extension point and the ID numbers of the atoms used to define it.

Format:

```
extension_point [comment]
```

```
extpt_atom_id atom_set_id a1 a2 a3 dist angle torsion
```

- extension_point (string) = name of the extension point
- comment (strings to the end of line) = the associated comment
- extpt_atom_id (integer) = ID number of the dummy atom representing the extension point
- atom_set_id (integer) = ID number of the set containing the atoms used to define the extension point
- a1 (integer) = ID of atom to which extension point is attached
- a2 (integer) = ID of atom which defines angle of point-a1-a2
- a3 (integer) = ID of atom which defines torsion of point-a1-a2-a3
- dist (float) = distance from extension point to a1
- angle (float) = angle defined by extension point to a1 to a2
- torsion (float) = torsion from extension point to a1 to a2 to a3

Examples:

```
DS_O2C2_1 CO extension
```

```
34 4 11 8 7 2.900000 120.000000 0.000000
```

The extension point is named DS_O2C2_1 and its comment reads "CO extension".

The extension point atom has rank 34 in the atom list. The ID number of the set containing the atoms used to define the extension point is 4.

The extension point connects directly to atom with ID 11. The distance between the two points is 2.900000Å.

The extension point forms an angle with atom 11 and atom 8 of 120°.

The extension point is at a torsion setting corresponding to a torsion from the point to atom 11 to atom 8 to atom 7 of 0.000000°.

@<TRIPOS>FF_PBC

This keyword is followed by periodic boundary conditions information. The specific format of a substructure record is as follows. (In what follows, "PBC" stands for Periodic Boundary Conditions.)

Format:

```
format_version_number pbc_type pbc_x_coord_min \  
  pbc_y_coord_min pbc_z_coord_min pbc_x_coord_max \  
  pbc_y_coord_max pbc_z_coord_max solvent_type \  
  num_solvent_shells reorient_molecule_flag \  
  status_flag apply_pbc_flag \  
  calc_electrostatics_flag 8_atom_id_numbers
```

- format_version_number = version number of the PBC file format
- pbc_type (integer) = number representing the type of PBC (currently, rectangular)
- pbc_x_coord_min (real) = x coordinate of the lower left back corner of the box
- pbc_y_coord_min (real) = y coordinate of the lower left back corner of the box
- pbc_z_coord_min (real) = z coordinate of the lower left back corner of the box
- pbc_x_coord_max (real) = x coordinate of the upper right front corner of the box
- pbc_y_coord_max (real) = y coordinate of the upper right front corner of the box
- pbc_z_coord_max (real) = z coordinate of the upper right front corner of the box
- solvent_type (string) = name of the solvent
- num_solvent_shells (integer) = number indicating the number of solvent shells to place around the solute
- reorient_molecule_flag = orientation of the solute: align_w_prin_axes or no
- status_flag = reserved for internal use
- apply_pbc_flag = APPLY_PBCS, DEFINE_MODIFY or IGNORE_PBCS
- calc_electrostatics_flag = CALCULATE_ELECTROSTATICS
- GET_NEW_CHARGES
- IGNORE_ELECTROSTATICS

-
- MARK_CHARGES_VALID
 - 8_atom_id_numbers (integers) = atom ID numbers of the 8 dummy atoms representing the corners of the box.

Example:

```
v1.0 1 -12.4001 -12.4001 -18.6001 12.4001 -12.4001 -18.6001 \  
none 2 0 0 0 0 2251 2252 2253 2254 2255 2256 2257 2258
```

@<TRIPOS>FFCON_ANGLE

Each data record associated with this RTI consists of a single data line. The data line contains the ID numbers of the 3 atoms which define the angle followed by the desired value for the angle (in degrees) and a constant which is used in the function used to calculate the penalty for deviation from the desired value. The three atom ID numbers must be unique.

Format:

atom1 atom2 atom3 target_value constant

- atom1 (integer) = atom ID of one atom used to define angle.
- atom2 (integer) = atom ID of one atom used to define angle.
- atom3 (integer) = atom ID of one atom used to define angle.
- target_value (real) = desired value of angle in degrees.
- constant (real) = constant used in penalty calculation.

Example:

2 4 5 9.000000e+01 1.000000e+02

The angle defined by atoms 2, 4, and 5 has a desired value of 90° and the penalty constant is 100.

@<TRIPOS>FFCON_DIST

Each data record associated with this RTI consists of a single data line which contains the atom ID numbers of the two atoms defining the distance followed by the desired distance and a penalty constant for deviation from the desired distance. The atom IDs must be unique.

Format:

atom1 atom2 target_distance penalty_constant

- atom1 (integer) = ID number of the first atom used to define distance.
- atom2 (integer) = ID number of the second atom used to define distance.
- target_value (real) = desired value of the angle in degrees.
- penalty_constant (real) = constant used in penalty calculation.

Example:

4 6 2.000000e+00 2.500000e+00

The distance between atoms 4 and 6 has an ideal value of 2Å and the penalty constant is 2.5.

@<TRIPOS>FFCON_MULTI

Each data record associated with this RTI consists of a single data line with the ID number of the atom involved in a multifit constraint followed by the penalty constant to be used in the penalty function.

Format:

atom penalty_constant

- atom (integer) = ID number of the atom involved in the multifit constraint.
- penalty_constant (real) = constant used in the penalty calculation.

Example:

2 2.000000e+00

Atom 2 is involved in a multifit constraint with a penalty constant of 2.0.

@<TRIPOS>FFCON_RANGE

Each data record associated with this RTI consists of single data line which consists of two atom ID numbers which define the distance to be constrained followed by the minimum and maximum value acceptable for the distance, a penalty constant, and a function power which determines the power of the penalty function.

Format:

atom1 atom2 min_dist max_dist penalty_constant power

- atom1 (integer) = ID number of the first atom used to define distance.
- atom2 (integer) = ID number of the second atom used to define distance.
- min_dist (real) = minimum distance between the two atoms.
- max_dist (real) = maximum distance between the two atoms.
- penalty_constant (real) = constant used in penalty function when the distance is outside the specified range.
- power (integer) = functional power of penalty function.

Example:

40 2 6.000000e+00 7.000000e+00 5.000000e+00 2

The distance between atoms 40 and 2 has a desired range of 6 to 7 Å. The penalty constant is 5 and the functional power of the penalty function is 2.

@<TRIPOS>FFCON_TORSION

Each data record associated with this RTI consists of a single data line containing the four atom ID numbers of the atoms defining the torsional angle followed by a penalty constant which is used in the penalty function and the desired value of the torsional angle.

Format:

```
atom1 atom 2 atom3 atom4 penalty_constant
target_value
```

- atom1 (integer) = ID of the first atom used to define the torsional angle.
- atom2 (integer) = ID of the second atom used to define the torsional angle.
- atom3 (integer) = ID of the third atom used to define the torsional angle.
- atom4 (integer) = ID of the fourth atom used to define the torsional angle.
- penalty_constant (real) = constant used in penalty calculation.
- target_value (real) = desired value of torsional angle in degrees.

Example:

```
5 6 7 8 2.000000e+00 1.800000e+02
```

The torsional angle defined by atoms 5, 6, 7, and 8 has a desired value of 180° and the penalty constant is 2.

@<TRIPOS>LINE

Each data record associated with this RTI consists of two data lines. The first contains the name of the line and any comment which may be associated with it. The second data line contains the ID number of the dummy atom representing the line and the ID numbers of the atoms used to define it.

Format:

line_point [comment]

line_atom_id atom_set_id a1 a2 Dist

- line_point (string) = name of the line
- comment (strings to the end of line) = the associated comment
- line_atom_id (integer) = ID number of the dummy atom representing the line
atom_set_id (integer) = ID number of the set containing the atoms used to define the line
- a1 (integer) = ID of atom which defines the origin of the axis, and to which the line point is attached
- a2 (integer) = ID of atom which defines the positive direction of the axis
- dist (float) = distance from the origin along the axis

Examples:

```
DS_N2C2_1 HBond  
38 7 6 37 3.000000
```

The line is named DS_N2C2_1 and its comment reads “HBond”.

The line’s atom has rank 38 in the atom list. The ID number of the set containing the atoms used to define the line is 7.

The origin of the line is atom ID 6. The positive direction for placing the extension point is determined by atom ID 37.

The distance from atom 6, in the direction of atom 37, is 3.000000 Å.

@<TRIPOS>LSPLANE

Each data record associated with this RTI consists of two data lines. The first contains the name of the plane and any comment which may be associated with the plane. The second data line consists of the four dummy atom ID numbers which define the corners of the plane followed by the set ID of the set which contains the ID numbers of the atoms used to define the plane and the coefficients of the equation of the plane. ($Ax + By + Cz = D$)

Format:

plane_name [comment]

atom1 atom2 atom3 atom4 set_id A B C D

- plane_name (string) = name of the plane.
- comment (strings to the end of line) = the comment associated with the plane.
- atom1 (integer) = ID number of a dummy atom at one corner of the plane.
- atom2 (integer) = ID number of a dummy atom at one corner of the plane.
- atom3 (integer) = ID number of a dummy atom at one corner of the plane.
- atom4 (integer) = ID number of a dummy atom at one corner of the plane.
- set_id (integer) = ID number of the set containing the atoms used to define the plane.
- A (real) = x coefficient of plane equation.
- B (real) = y coefficient of plane equation.
- C (real) = z coefficient of plane equation.
- D (real) = constant in plane equation.

Example:

MY_PLANE example of plane definition

```
49 50 51 52 2 4.955655e-03 6.672358e-02 9.977592e-01
1.987229e-01
```

The plane is named MY_PLANE and its comment reads “example of plane definition”. The displayed plane has 4 corners represented by dummy atoms 49, 50, 51, and 52. The ID number of the set containing the atoms used to define the plane is 2. The equation of the plane is $4.955655e-03 * X + 6.672358e-02 * Y + 9.977592e-01 * Z = 1.987229e-01$.

@<TRIPOS>MOLECULE

Each data record associated with this RTI consists of six data lines. The first data line is the name of the molecule. The second data line contains the number of atoms, bonds, substructures, features, and sets associated with the molecule. The third data line is the molecule type. The fourth data line tells the type of charges associated with the molecule. The fifth data line contains the internal SYBYL status bits associated with the molecule. The last data line contains any comment which may be associated with the molecule.

Format:

`mol_name`

`num_atoms [num_bonds [num_subst [num_feat
[num_sets]]]]`

`mol_type`

`charge_type`

`[status_bits`

`[mol_comment]]`

- `mol_name` (all strings on the line) = the name of the molecule.
- `num_atoms` (integer) = the number of atoms in the molecule.
- `num_bonds` (integer) = the number of bonds in the molecule.
- `num_subst` (integer) = the number of substructures in the molecule.
- `num_feat` (integer) = the number of features in the molecule.
- `num_sets` (integer) = the number of sets in the molecule.
- `mol_type` (string) = the molecule type: `SMALL`, `BIOPOLYMER`, `PROTEIN`, `NUCLEIC_ACID`, `SACCHARIDE`
- `charge_type` (string) = the type of charges associated with the molecule: `NO_CHARGES`, `DEL_RE`, `GASTEIGER`, `GAST_HUCKEL`, `HUCKEL`, `PULLMAN`, `GAUSS80_CHARGES`, `AMPAC_CHARGES`, `MULLIKEN_CHARGES`, `DICT_CHARGES`, `MMFF94_CHARGES`, `USER_CHARGES`
- `status_bits` (string) = the internal SYBYL status bits associated with the molecule. ***These should never be set by the user.*** Valid status bits are `system`, `invalid_charges`, `analyzed`, `substituted`, `altered` or `ref_angle`.
- `mol_comment` (all strings on data line) = the comment associated with the molecule.

Example:

```
vitamin B2          vitamin B2          vitamin B2
54 56 2 34 5       54 56 2 34 5       54
SMALL              SMALL              SMALL
NO_CHARGES        NO_CHARGES        NO_CHARGES
****
This is a comment
```

In all examples the molecule is a small molecule named “vitamin B2” and there are no charges associated with the molecule.

The first example contains a comment which reads “This is a comment”. The four asterisks indicate there are no SYBYL status bits associated with the molecule. *Note:* the asterisks must be present because the comment line exists. The status bit line is optional only when the comment line does not exist.

In the second example, since there is no comment line, the optional status bit line is also absent.

The third example is the minimal representation of the data record.

In general if one optional field is not present, all the fields after it must not be present.

@<TRIPOS>NORMAL

Each data record associated with this RTI consists of two data lines. The first contains the name of the normal followed by the name of the least squares plane associated with the normal and the comment associated with the normal. The second data line contains ID numbers of the three atoms which represent the two end points and the midpoint of the normal followed by the ID number of the plane associated with the normal. The ID of the plane is the ID at the time the file was created and it may differ from the actual ID of the plane at the time the .mol2 file is read into SYBYL. The MOL command uses the plane name on the first data line to associate the normal and the plane.

Format:

```
normal_name plane_name [comment]
```

```
end_pt_1 end_pt_2 mid_pt plane_id
```

- normal_name (string) = the name of the normal.
- plane_name (string) = the name of the plane associated with the normal.
- comment (remaining strings on line) = the comment associated with the normal.
- end_pt_1 (integer) = the ID of the atom of one end of the normal.
- end_pt_2 (integer) = the ID of the atom at the other end of the normal.
- mid_pt (integer) = the ID of the atom at the midpoint of the normal.
- plane_id (integer) = the ID of the plane associated with the normal.

Example:

```
NORM_A PHENYL_A Normal to phenyl ring A  
53 54 52 2
```

Defines a normal whose end points lie at dummy atoms 53 and 54 and whose midpoint is at atom 52. The plane associated with this normal is named PHENYL_A. The normal is named NORM_A and it has a comment which reads "Normal to phenyl ring A".

@<TRIPOS>QSAR_ALIGN_RULE

The data records associated with this RTI contain the rigid body alignment rule corresponding to the alignments created via the command `QSAR COMFA ALIGNMENT DEFINE AS_IS`. Each data record consists of two lines, the first having the name of the rule and the lengthy second line having a description of the rule. This description is not available for any other use within SYBYL; it should be regarded as an internally meaningful description only.

Format:

```
alignment_name  
  
internal description of the rule
```

Example:

```
AL_RULE  
pat 763680005 32 \  
-1.139374813064933e-01 3.497718954458833e-01 \  
-8.126781080500223e-01 \  
-7.506399292149334e-01 6.606869326105553e-01 \  
-5.698574019449137e-03 -5.367933126516231e-01 \  
-6.048041186338151e-01 5.882728258011506e-01 \  
3.852176477794031e-01 4.446400287437351e-01 \  
8.086424479818470e-01 1.040096634345387e+02\  
1.773840783728993e+02 4.160000000000000e+02
```

Here the alignment rule is named `AL_RULE`.

The internal description, although not designed for external use, consists of the following elements in order:

- the username of the creator,
- an integer used as a time-date stamp for the creation of the rule,
- an integer which is the number of atoms involved,
- three floating numbers describing the translation from the centered position,
- six floating numbers representing the rotation matrix from the internally-used canonical rotation to the alignment rule rotation,
- three internal consistency checks that the molecule is unchanged since the definition of the rule: the sum of absolute values of the sum of aligned coordinates, the sum of absolute values of aligned coordinates, the sum of atomic numbers

@<TRIPOS>RING_CLOSURE

Each data record associated with this RTI consists of a single data line. It contains the bond ID of the ring closure followed by the distance, how much distance variance is permitted for the closure bond, and the variance in valence angles for the closure atoms.

Format:

bond_id dist_var ang_var

- bond_id (integer) = the ID number of the ring closure bond.
- dist_var (real) = the extent of bond length deformation allowed at the ring closure during search.
- ang_var (real) = the extent of bond angle deformation allowed at the ring closure during search.

Example:

10 0.100 5.00

Bond 10 is the ring closure bond and its closure atoms are allowed a distance change of 0.1 Å and valence angle variation of 5°.

@<TRIPOS>ROTATABLE_BOND

Each data record associated with this RTI consists of a single data line which contains the bond ID of the rotatable bond, the bond ID numbers of the two bonds forming the reference angle, the rotatable bond label, the status of the rotatable bond, the bond ID of the associated ring closure bond, and the count of angle ranges associated with this bond. If count is nonzero, the corresponding number of pairs of low and high values for the angle ranges follows the angle increment.

The position of each of these fields on the data line is crucial for accurate reading. We recommend that you use the MOL command to generate a sample file for reference before you write your own program.

Format:

```
b_id ref_1 ref_2 rot_lab status ring_id count inc
  {low high}
```

- b_id (integer) = the ID number of the bond defined as rotatable.
- ref_1 (integer) = the ID number of the first bond used to define the reference angle.
- ref_2 (integer) = the ID number of the second bond used to define the reference angle.
- rot_lab (integer) = the rotatable bond label.
- status (integer) = the active/inactive status flag.
- ring_id (integer) = ID number of the ring closure bond, 0 if the bond is not in a flexible ring.
- count (integer) = the number of angle ranges associated with this bond.
- inc (integer) = the increment value for the angle rotation.
- low (integer) = the low value of an angle range.
- high (integer) = the high value of an angle range.

Example:

```
15 13 33 1 1 0 1 30 0 359
```

Bond 15 is a rotatable bond, with bonds 13 and 33 forming the reference angle. The rotatable bond is labeled 1, it is active, and is not in a ring so the ring_id is set to zero. There is only one angle range which has an increment value of 30° and the angle may range between 0 and 359°.

@<TRIPOS>SEARCH_DIST

Each data record associated with this RTI consists of a single data line which contains the ID numbers of the two atoms whose distance is used as a constraint, followed by the minimum and maximum distance allowable for the two atoms.

Format:

atom1 atom2 minimum maximum

- atom1 (integer) = the ID number for the first atom.
- atom2 (integer) = the ID number for the second atom.
- minimum (real) = the minimum desired distance between the two atoms.
- maximum (real) = the maximum desired distance between the two atoms.

Example:

40 2 6.000000e+00 7.000000e+00

The distance between atoms 40 and 2 may range between 6.0 and 7.0 Å.

@<TRIPOS>SEARCH_OPTS

Each data record associated with this RTI consists of a single data line (use \ as the continuation character), containing the options and any distance map or coordinate map definitions used in the most recent SEARCH SETUP session executed with this molecule.

Format:

```
version ref_conformation angles \  
  energies energymax energycharges \  
  vdwfactor hybondfac vdw14fac distdims \  
  [distout distin [constraint_name supercn] \  
  [{atom1 atom2 mindist maxdist grid}]] \  
  coordims [coorout coorin [constraint_name] \  
  [{atom accuracy}]]
```

- version (integer) = an internal code indicating the SYBYL version number.
- ref_conformation (integer) = zeroed (0) or current (1) torsion angles used.
- angles (integer) = conformational data is recorded (1) or suppressed (0).
- energies (integer) = energies are calculated (1) or not (0).
- energymax (real) = the maximum relative energy value to retain.
- energycharges (integer) = electrostatics are included (1) or not (0).
- vdwfactor (real) = the general vdw factor to be applied.
- hybondfac (real) = the additional vdw factor to be applied to hydrogen bonded atoms.
- vdw14fac (real) = the vdw factor to be applied to atoms 1-4 to each other.
- distdims (integer) = the number of distance map dimensions.
- distout (integer) = output distance map is to be created (1) or not (0).
- distin (integer) = distance map is used to constrain (1) or not (0).
- constraint_name (string) = the name of the constraint distance map.
- supercn (integer) = constraining distance map is superconstraining (1) or not (0)
- For each distance map dimension:
 - atom1 (integer) = the ID number for the first atom in this dimension.
 - atom2 (integer) = the ID number for the second atom in this dimension.

-
- mindist (real) = the minimum desired distance between the two atoms.
 - maxdist (real) = the maximum desired distance between the two atoms.
 - grid (real) = the grid size for this dimension.
 - coordims (integer) = the number of coordinate map dimensions.
 - coorout (integer) = output coordinate map is to be created (1) or not (0).
 - coorin (integer) = coordinate map is used to constrain (1) or not (0).
 - constraint_name (string) = the name of the constraint coordinate map.
 - For each coordinate map dimension:
 - atom (integer) = the ID number for the atom in this dimension.
 - accuracy (real) = the accuracy with which to match coordinates for this dimension.

Example:

```
0 0 1 1 1.000000e+02 1 9.000000e-01 6.500000e-01 8.700000 \
e-01 2 1 0 16 20 0.000000e+00 1.000000e+03 2.000000 \
e-01 17 20 0.000000e+00 1.000000e+03 2.000000e-01 0
```

This is a sample entry from a search where energies were calculated with electrostatics, retaining those conformers within a 100 kcal/mole window of the minimum conformation; angles recorded; a zeroed reference conformation was used; and a two dimensional distance map was created; but no coordinate maps were involved.

@<TRIPOS>SET

Each data record associated with this RTI consists of two data lines. The first data line contains the set name, type, object class, sub type, SYBYL status and comment. If the set type is `STATIC`, the second data line contains the number of elements in the set followed by a list of the ID numbers of the element members. If the set type is `DYNAMIC` the second data line contains the rule used to define the set. If you need to break up a long data line, use a backslash (\) as the continuation character.

Colored atoms and bonds are stored in named sets. This preserves the color scheme in the `.mol2` file.

STATIC set format:

```
set_name set_type obj_type [sub_type [status  
  [comment]]]  
  
num_members member member member ...
```

DYNAMIC set format:

```
set_name set_type obj_type [sub_type [status \  
  [comment]]]  
  
rule
```

Fields:

- `set_name` (string) = the name of the set.
- `set_type` (string) = the set type: static or dynamic
- `obj_type` (string) = the type of objects in the set: `ATOMS`, `BONDS` or `SUBSTS`
- `sub_type` (string) = the subtype of the set: `<user>`, `AGGREGATE`, `COLOR-GROUP`, `DISPGROUP`, `LSPLANE`, `DICT`, `AMSOM`, `CENTROID`, `NORMAL`, `SUBSTITUTION` or `FILL`.
- `status` (string) = the SYBYL status bits. *These should never be set by the user.* Valid status bits are `RELEVAL`, `SYSTEM`, `GLOBAL`, `ACTIVE_AGG`, `DELETE_EMPTY`.
- `comment` (remaining strings on data line) = the comment for the set.
- `num_members` (integer) = the number of members in a `STATIC` set.
- `member` (integer) = the ID of a member in a `STATIC` set.
- `rule` (all strings on data line) = the rule defining the `DYNAMIC` set.

Example:

```
CARBONS STATIC ATOMS <user> **** All carbons in molecule
18 2 4 5 7 8 9 10 11 12 14 17 18 19 20 21 22 23 48
```

The set named “CARBONS” is static and has been defined by the user. It does not contain any SYBYL status bits, hence the four asterisks before the comment: “All carbons in molecule”. The set consists of 18 carbon atoms, numbers 2, 4, 5, 7, 8, 9, 10, 11, 12, 14, 17, 18, 19, 20, 21, 22, 23, and 48

```
SITE DYNAMIC ATOMS <user> **** Sphere of 6Å around SER195
{sphere(SER195.*,6)}
```

The set named “SITE” is dynamic, contains atoms and was defined by the user. It does not contain any SYBYL status bits, hence the four asterisks before the comment: “Sphere of 6 ang. around SER195”. The rule used to create the set was {sphere(SER195.*,6)}.

@<TRIPOS>SUBSTRUCTURE

Each data record associated with this RTI consists of a single data line. The data line contains the substructure ID, name, root atom of the substructure, substructure type, dictionary type, chain type, subtype, number of inter substructure bonds, SYBYL status bits, and user defined comment.

Format:

```
subst_id subst_name root_atom [subst_type [dict_type  
[chain [sub_type [inter_bonds [status  
[comment]]]]]]]
```

- subst_id (integer) = the ID number of the substructure. Provided for reference only, but not used by the MOL command when reading the file.
- subst_name (string) = the name of the substructure.
- root_atom (integer) = the ID number of the root atom of the substructure.
- subst_type - string) = the substructure type: temp, perm, residue, group or domain.
- dict_type (integer) = the type of dictionary associated with the substructure.
- chain (string) = the chain to which the substructure belongs (= < 4 chars).
- sub_type (string) = the subtype of the chain.
- inter_bonds (integer) = the number of inter substructure bonds.
- status (string) = the internal SYBYL status bits. *These should never be set by the user.* Valid status bit values are LEAF, ROOT, TYPECOL, DICT, BACKWARD and BLOCK.
- comment (remaining strings on data line) = the comment for the substructure.

Example:

```
1 ALA1 1 RESIDUE 1 A ALA 1 ROOT|DICT Comment here
```

The substructure has 1 as ID, ALA1 as name and atom 1 as root atom. It is of type RESIDUE and the associated dictionary type is 1 (protein). It is part of the A chain in the molecule and it is an ALanine. There is only one inter substructure bonds. The SYBYL status bits indicate it is the ROOT substructure of the chain and it came from a dictionary. The comment reads "Comment here".

```
1 ALA1 1
```

Minimal representation of a substructure.

@<TRIPOS>U_FEAT

The data records associated with this RTI contain the information necessary to reconstruct UNITY features and constraints defined in the molecule. Each data record following this RTI consists of one line representing either a UNITY feature or a UNITY constraint.

Header

The first three fields in most features' records are `class`, `type`, and `name`. In the description of the features which follow, the shorthand `<header>` represents these three fields.

Classes:

- 1 Feature
- 2 Constraint
- 3 Atom
- 4 Macro (Donor atom, hydrophobic, etc.)
- 5 Spatial constraint

Types:

- | | | | |
|----|------------------------------|----|------------------------------------|
| 0 | Centroid | 14 | Spatial Point |
| 1 | Plane | 15 | Spatial Torus |
| 2 | Line | 16 | Tetrahedral |
| 3 | (not used) | 17 | 4-Point (Torsion) Angle Constraint |
| 4 | Extension Point | 18 | Partial Match Directive |
| 5 | Normal Point | 19 | Spatial Line |
| 6 | Distance Constraint | 20 | Spatial Plane |
| 7 | Angle Constraint | 21 | Multi-Sphere Excluded Volume |
| 8 | Excluded Volume Constraint | 22 | Multi-Sphere Containing Volume |
| 9 | (not used) | 23 | Fragment |
| 10 | Containing Volume Constraint | 24 | Spatial Cap |
| 11 | Line/Plane Angle Constraint | 25 | Markush |
| 12 | Receptor Site | 26 | Surface Volume Constraint |
| 13 | Macro Reference | 27 | Bond Path Constraint |

Properties and Features

Many features also contain lists of defining properties and features. The format for this is:

```
<# of properties> <property id # list> <# of
  features> <feature names or id # list>
```

Some features may be defined using only properties, and some using only features, but both lists are usually included for ease of parsing. The lists for a feature with no properties and two centroid features looks similar to:

```
0 2 CENTROID1 CENTROID2
```

In the description of the features which follow, the shorthand <props and feats> is used to signify these lists.

Centroid

```
<header> <props and feats>
```

There are no features, and one property that contains the atoms which define the centroid.

The properties are stored under @<TRIPOS>SET. If the first property is:

```
ucent$CENT1    STATIC    ATOMS    UNITY
SYSTEM|DELETE_EMPTY
6 11 12 13 14 15 16
```

then the centroid feature looks like:

```
1 0 CENT1 1 1 0
```

This defines a centroid named CENT1 defined on the six atoms 11, 12, 13, 14, 15, and 16.

Additional information: Centroid feature in the SLN Manual.

Plane

```
<header> <props and feats> rms
```

There are no features, and one property that contains the atoms which define the plane. The rms value is never used, but is stored as -1.0 for backward compatibility.

A plane feature definition looks like:

```
1 1 LSPLANE1 1 2 0 -1.000000
```

where the 2 refers to property 2, which looks like:

```
ulsp$LSPLANE1    STATIC    ATOMS    UNITY
SYSTEM|DELETE_EMPTY
6 1 2 3 4 5 6
```

Additional information: Plane feature in the SLN Manual.

Line

```
<header> -2 <props and feats> <start point class>
  <start point index> <end point class>
  <end point index>
```

The -2 after the header signifies that this is a line feature for SYBYL 6.4 or later. Earlier versions without the -2 are not documented here.

The point classes are either 1 for feature, or 3 for atom. The atoms are stored in properties. Each point index is an index into the feature list (if the class is 1) or the property list (if the class is 3). So a line feature which is defined from an atom to a centroid feature looks similar to the following (the letters “a-l” refer to the guide below; they do not appear in the .mol2 file):

```
1 2 LINE1 -2 1 1 1 CENT1 3 0 1 0
a b c      d e f g h      i j k l
```

- a class (= feature)
- b type (= line)
- c name
- d -2 specifies SYBYL 6.4 version of a line
- e number of properties
- f property ID number
- g number of features
- h feature name
- i starting point is an atom
- j the atom is in property #0 in the property list
- k ending point is a feature
- l the feature is feature #0 in the feature list

The properties are stored under @<TRIPOS>SET. The first property looks like:

```
uline1$LINE1    STATIC    ATOMS    UNITY
SYSTEM|DELETE_EMPTY
1 5
```

This specifies a set containing 1 atom, namely atom 5. So the example describes a line from atom 5 to feature CENT1. If the line was from CENT1 to atom 5, the “3 0 1 0” at the end would be “1 0 3 0”. If the line was from CENT1 to CENT2, it would look like:

```
1 2 LINE1 -2 0 2 CENT1 CENT2 1 0 1 1
```

A line between two atoms would look like:

```
1 2 LINE1 -2 2 1 2 0 3 0 3 1
```

Additional information: Line feature in the SLN Manual.

Extension Point

The extension point definition has the name at the end, rather than following the class and type.

```
<class> <type> <property id> <distance> <angle>  
  <dihedral> <1st atom> <2nd atom> <3rd atom>  
  <name>
```

There is always one property containing the 3 atoms used in the extension point. Since the order of the atoms is important, the atom numbers are also included in the extension point definition itself. So the definition of an extension point looks like:

```
1 4 7 3.00 90.00 180.00 42 54 49 EXTPT1
```

which defines an extension point named EXTPT1 on atoms 42, 54, and 49, with a distance of 3 Å, an angle of 90°, and a dihedral angle of 180°. Property number 7 looks like:

```
uxp$EXTPT1    STATIC    ATOMS    UNITY  
SYSTEM|DELETE_EMPTY  
3 42 49 54
```

Additional information: Extension point feature in the SLN Manual.

Normal Point

```
<header> <distance> <props and feats> <selected  
  point>
```

There are no features, and one property that contains the atoms which define the normal. The selected point is either 0 or 1, specifying one of the two possible normal points.

A normal point feature named NORMPT1 defined on the atoms in property 4 with distance 1.5 Å looks like:

```
1 5 NORMPT1 1.500000 1 4 0 1
```

Additional information: Normal feature in the SLN Manual.

Distance Constraint

```
<header> <distance> <tolerance> <props and feats>
```

A distance constraint has either two properties, two features, or one of each. Each property contains exactly one atom. The features may be point-class features (centroids, extension points, etc.), lines, or planes.

A distance constraint between an atom in property 7 and a centroid feature, with a distance of 5 Å and a tolerance of 0.2 Å is defined:

```
2 6 DIST1 5.000000 0.200000 1 7 1 CENT1
```

Additional information: Distance constraint in the SLN Manual.

Angle Constraint

```
<header> <angle> <tolerance> <class 1> <index 1>
  <class 2> <index 2> <class 3> <index 3>
  <props and feats>
```

The class-index pairs are similar to those used for lines. If the class is 1, the index is for the feature list. If the class is 3, the index is for the property list. An angle constraint of 60° with a tolerance of 5° between atom 5, feature CENT1, and atom 10 is defined:

```
@<TRIPOS>SET
uang1$ANG1    STATIC    ATOMS    UNITY
SYSTEM|DELETE_EMPTY
1 5
uang2$ANG1    STATIC    ATOMS    UNITY
SYSTEM|DELETE_EMPTY
1 10
@<TRIPOS>U_FEAT
2 7 ANG1 60.00 5.00 3 0 1 0 3 1 2 1 2 1 CENT1
a b c    d    e    f g h i j k l m n o p
```

- a class (= constraint)
- b type (= angle constraint)
- c name
- d angle value
- e tolerance
- f first item is an atom
- g the atom is in property #0 in the property list
- h second item is a feature
- i the feature is feature #0 in the feature list
- j third item is an atom
- k the atom is in property #1 in the property list
- l there are two properties
- m first property ID number (points to uang1\$ANG1)
- n second property ID number (points to uang2\$ANG1)
- o there is one feature
- p feature name

Additional information: Angle constraint in the SLN Manual.

Excluded and Containing Volume Constraints

The single-sphere volume constraints are no longer used, but are still supported for backward compatibility. All volumes are now stored as multi-sphere constraints.

Additional information: Excluded volume constraint and Containing volume constraint in the SLN Manual.

Line/Plane Angle Constraint

```
<header> <angle> <tolerance> 0 0 0 0 <props and  
feats>
```

Older versions of this feature had class-index pairs which are no longer used, so the four 0's are written for backward compatibility. The ordering of the line or plane features does not matter. The number of properties is always 0, and the number of features is always 2. An angle constraint between LINE1 and PLANE1 looks like:

```
2 11 LPANG1 90.00 10.00 0 0 0 0 0 2 LINE1 PLANE1
```

Additional information: Line/plane angle constraint in the SLN Manual.

Receptor Site

The receptor site is defined in the same way as a multi-sphere excluded volume constraint, except that the type is 12 rather than 21.

Additional information: Receptor site constraint in the SLN Manual.

Macro Reference

```
<header> <macro name> <target coordinate>  
<props and feats> <center coordinate>  
<vector 1> <vector 2> <color>
```

The number of properties is always 0. The number of features is 1 if there is a connection attribute (the feature is the other macro reference that this macro reference is connected to). The center coordinate and the vectors are for internal SYBYL use when adding a spatial constraint to the macro reference.

```
4 13 ACCEPTOR_ATOM1 ACCEPTOR_ATOM 1.0 2.0 3.0 0 0  
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  
4 13 DONOR_SITE1 DONOR_SITE 4.0 5.0 6.0 0 1  
ACCEPTOR_ATOM1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

This describes an acceptor atom at coordinates (1.0, 2.0, 3.0), and a donor site at coordinates (4.0, 5.0, 6.0) which is connected to the acceptor atom.

Additional information: Macro atoms in the SLN Manual.

Spatial Point

<header> <tolerance> <target coordinate> <props and feats> <color>

There is either one feature or one property. The property contains one atom.

A spatial point constraint on a centroid with a tolerance of 0.5 Å and a target coordinate (1.4388, -4.727, 0.8463) is defined:

```
5 14 SPATIALPT1 0.500000 1.438800 -4.727000 0.846300 \
0 1 CENT1 GREEN
```

Additional information: Spatial point constraint in the SLN Manual.

Spatial Torus

**<header> <radius> <center coordinate> <tolerance>
<normal vector> <props and feats> <color>**

There is either one feature or one property. The property contains one atom.

A spatial torus constraint on an atom defined in property 4 with a radius of 1.2 Å, a tolerance of 0.3 Å, a center coordinate (1.4388, -4.727, 0.8463), and a normal vector (-0.988, -0.016, 0.156) looks like:

```
5 15 TORUS1 1.2000 1.4388 -4.7270 0.8463 0.3000 \
-0.9880 -0.0160 0.1560 1 4 0 YELLOW
```

Additional information: Torus constraint in the SLN Manual.

Tetrahedral

<header> <central atom id> <distance> <property id>

The property contains 4 atoms, but the central atom is the atom to which the tetrahedral is attached.

If property 6 is:

```
utet$TETRA1    STATIC    ATOMS    UNITY
SYSTEM|DELETE_EMPTY
4 5 9 11 22
```

then a tetrahedral feature on atoms 5, 9, 11, and 22, with the tetrahedral attached to atom 9 at a distance of 3 Å, looks like:

```
1 16 TETRA1 9 3.000000 6
```

Additional information: Tetrahedral feature in the SLN Manual.

4-Point (Torsion) Angle Constraint

This is identical to the Angle Constraint, except that there are four class-index pairs rather than three, and the type is 17 rather than 11.

Additional information: Dihedral constraint in the SLN Manual.

Partial Match Directive

<header> <min> <max> <color> <props and feats>

There are no properties. The features are those which are included in the partial match.

A partial match directive on 5 features with minimum 3 and maximum 4 looks like:

```
2 18 PARTIAL1 3 4 RED 0 5 CENT1 CENT2 DONOR1 HYD1
LSPLANE2
```

Additional information: Partial match directive in the SLN Manual.

Spatial Line

**<header> <angle> <tolerance> <starting coordinate>
<vector> <props and feats> <color>**

There are no properties, and one line feature.

A spatial line constraint with an angle of 0 degrees, a tolerance of 10 Å, a starting coordinate (-0.0035, 1.1756, 1.5260), and a vector (-0.0025, 0.7322, -0.6810) is defined:

```
5 19 SPATLINE1 0.00 10.00 -0.0035 1.1756 1.5260 \  
-0.0025 0.7322 -0.6810 0 1 LINE1 BLUE
```

Additional information: Spatial line constraint in the SLN Manual.

Spatial Plane

**<header> <angle> <tolerance> <starting coordinate>
<vector> <props and feats> <color>**

There are no properties and one plane feature.

A spatial plane constraint with a tolerance of 10 Å, a starting coordinate (-0.0035, 1.1756, 1.5260), and a vector (-0.0025, 0.7322, -0.6810) looks like:

```
5 20 SPATPLANE1 10.00 -0.0035 1.1756 1.5260 -0.0025 \  
0.7322 -0.6810 0 1 LSPLANE1 ORANGE
```

Additional information: Spatial plane constraint in the SLN Manual.

Multi-Sphere Excluded and Containing Volume Constraints

**<header> <vdw ratio> <sphere count> \
<radius 1> <coordinate 1> \
<radius 2> <coordinate 2> \
. . .
<radius n> <coordinate n>**

An excluded volume constraint with a vdw ratio of 0.9 and 3 spheres is defined:

```
2 21 EXCVOL1 0.900000 3 \  
.
```

```

3.500000 1.000000 2.000000 3.000000 \
2.000000 4.000000 5.000000 6.000000 \
2.800000 7.000000 8.000000 9.000000

```

A containing volume looks the same, except that the type is 22 instead of 21.

Fragment

Identical in structure to a Centroid, except the type is 23 instead of 0.

Additional information: Fragment feature in the SLN Manual.

Spatial Cap

```

<header> <point> <center coordinate> <tolerance>
  <bend angle> <twist angle> <vector 1> <vector 2>
  <rotatable> <props and feats> <color>

```

There are no properties and one feature. <rotatable> is 1 if the cap is rotatable, 0 otherwise.

A rotatable spatial cap on a donor atom with a tolerance of 0.5 Å, a bend angle of 70° and a twist angle of 30° looks like:

```

5 24 SPATIAL_CAP1 4.947179 -3.548140 -11.860902 \
  6.604500 -2.884300 -9.575600 0.500000 70.000000 \
  30.000000 0.575350 -0.253583 -0.777604 0.575350 \
  -0.253583 -0.777604 1 0 1 DONOR1 MAGENTA

```

Additional information: Spatial cap constraint in the SLN Manual.

Markush

```

<header> <markush definition>

```

A feature for a Markush with name NORC and definition N|C is defined:

```

1 25 U_MARKUSH_NORC N|C

```

Additional information: Markush atoms in the SLN Manual.

Surface Volume Constraint

```

<header> <usurf file name> <vdw ratio>

```

A surface volume constraint which has a vdw ratio of 0.8 and is defined in a file named SURFACE1.usurf looks like:

```

2 26 SURFACE1 /home/user/SURFACE1.usurf 0.800000

```

Additional information: Surface volume constraint in the SLN Manual.

Bond Path Constraint

```

<header> <min> <max> 0 2 <feature 1> <feature 2>

```

A bond path constraint of 1 to 10 bonds between two features looks like:

```
2 27 BP_HYDRO1_ACCEPT_AT1_1 1 10 0 2 HYDRO1 ACCEPT_AT1
```

Additional information: Bond path constraint in the SLN Manual.

@<TRIPOS>UNITY_ATOM_ATTR

The data records associated with this RTI contain the information necessary to reconstruct UNITY atom attributes defined in the molecule. UNITY atom attributes are used to define charge, chirality, etc.

<atom number> <number of attributes>

<attribute 1 name> <attribute 1 value>

<attribute 2 name> <attribute 2 value>

.

.

.

<attribute n name> <attribute n value>

If atom 5 has a charge of -3, chirality N and isotope 1, and atom 9 has a user-defined attribute A with value XYZ and a user-defined boolean attribute B, the records are:

```
5 3
charge -3
S N
I 1
9 2
A XYZ
B
```

@<TRIPOS>UNITY_BOND_ATTR

The data records associated with this RTI contain the information necessary to reconstruct UNITY bond attributes defined in the molecule. UNITY bond attributes are used to define stereochemistry, fragmentation bond for Topomer Search, etc.

```
<bond number> <number of attributes>
<attribute 1 name> <attribute 1 value>
<attribute 2 name> <attribute 2 value>
.
.
.
<attribute n name> <attribute n value>
```

If bond 6 has stereochemistry I and a user-defined attribute B with value XYZ, the records are:

```
6 2
S I
B XYZ
```

Example of topomeric bonds in a scaffold query to a Topomer Search:

```
4 1
CB
10 1
CB
```