Daylight Chemical Information Systems, Inc.
120 Vantis - Suite 550 - Aliso Viejo, CA 92656
tel +1 949-831-9990 - fax +1 949-831-9902 - www.daylight.com

Dayight MCL Manual

Table of Contents

MCL Manual

Daylight Version 4.9
Release Date 08/01/11

1. Introduction

MCL is an English language interface to Merlin. In general:

- MCL is intended to be readable and writable by both machines and humans.
- MCL statements consist of reserved words and other text, mainly user-defined names for things such as columns and hitlists.
- MCL statements resemble English sentences: the first word starts with a capital letter; statements end with a period; everything is case sensitive. Correct MCL grammar is also correct English grammar.
- Whitespace outside quoted text is ignored. Line numbers are reported for errors; line-breaks (newlines) are otherwise ignored.
- Non-reserved text needs to be quoted only if it collides with reserved words or contains characters other than letters or digits. For clarity, such text is uniformly quoted in this document.

2. Conventions Used in this Document

The following section includes prototypes and brief explanations for all MCL statements, using these conventions.

** **	***"text"*** user text prototype; quotes are not needed if text doesn't contain reserved words and contains only letters and digits.
	word words not in quotes are case-specific reserved words
	[A] optional specification, i.e., A may be omitted
	[A|B] optional choice, i.e., accepts A or B or neither
	[A ...] repeating specification, i.e., A may be repeated
	<word> reserved word required, typically a flag

The following prototypes are used in this document:

** **	**column** name of column (as per previous Create column...)
	database database name in the form: base@host:service:user

field field name (e.g., "pH") or field number (e.g., "#5")

fontname short font name as per option, or "default"

hitlist hitlist name

hitlist2 name of a different hitlist than "hitlist"

integer *or* n an integer, e.g., just digits

lpr lines per row, an integer in range 1-8

max maximum value, form is context dependent

min minimum value, form is context dependent

name user supplied name, max length 128 chars

regexp regular expression

smarts a molecular pattern in the SMARTS language

smiles a molecule in the SMILES language

string a string of any character except NUL (\" means ").

tag a datatype tag

tan a real (tanimoto) value in the range 0.0 - 1.0

User entry conventions:

this entry	is the same as
`"column"`	`[ column ] "column"`
`["hitlist"]`	`[ [ hitlist ] "hitlist" ]`
`alpha`	`alpha "alpha"`
`beta`	`beta "beta"`

Assume this has been done: **`Set default hitlist "defhits".`** Then:

statement	**equivalent full statement**
`Remove "CLUSTER" repeats.`	`Remove column "CLUSTER" repeats.`
`Reset "defhits".`	`Reset hitlist "defhits".`
`Reset.`	`Reset hitlist "defhits".`
`Move to row 10 of "defhits".`	`Move to row 10 of hitlist "defhits".`
`Move to row 10.`	`Move to row 10 of "defhits".`
`... Tversky 0.9 0.1 ...`	`... Tversky alpha 0.9 beta 0.1`

3. Reserved Words

The following (case-sensitive) words are reserved in MCL:

```
above       default    least       Print       structure(s)
Add         depiction  less        Put
alpha       Display    lines       range       substring
ares        down       list        Read        substructure(s)
```

```
as          entitled   matching     regexp
at          Exchange   missing      Remove      superstructure(s)
available   field      repeats      table
below       file       molecule(s)  Reset       tautomer(s)
beta        font       Move         Reverse     text
by          Free       named        row         than
Clear       from       nativeorder  Select      to
column      function   next         Set         Tversky
containing  graph(s)   non          similar     tversky
Copy        hitlist    not          similarity  up
Create      in         of           Sort        value(s)
database    into       pattern      status      with
datatype    Invert     per          string(s)   Write
```

The following words (sort flags) are also reserved in MCL:

```
/AAZ   /ANC  /ANCP  /ANCPW  /ANCW  /ANP  /ANPW
/ANW   /ASC  /CAS   /LEN    /MF    /NAB  /NUM
```

The following pairs of reserved words are synonyms:

```
graph ........... graphs
molecule ........ molecules
string .......... strings
structure ....... structures
substructure .... substructures
superstructure .. superstructures
tautomer ........ tautomers
value ........... values
```

Additionally, the symbols $1 - $9 are reserved as formal parameters.

4. MCL Environment

MCL is designed to run in various environments: internally in an interactive program such as xvmerlin, as a CGI program run by a server such as httpd, and as a non-interactive "batch" process. Actually, in all cases MCL runs in a non-interactive way, i.e., a whole program (or at least a whole statement) is expected to be delivered to the MCL processor. MCL features which are environment-specific are described here.

Some MCL output control statements are provided for use in an xvmerlin-like environment. These statments are ignored in an ASCII- oriented batch environment. Examples of such commands are:

```
Set font "Times-14".
Display smiles as depiction.
```

In an interactive environment like xvmerlin, user input is typically transcribed to the MCL program which is then run. Although this is possible in a batch environment, it is often more convenient to use a fixed MCL program with data supplied externally, e.g., via command line arguments in the `mcl' program. The symbols **$1** - **$9** are used to refer to formal parameters to the MCL program. Each symbol represents an externally supplied string which is used as an MCL language token. If a parameter is referenced in MCL but not supplied by the environment, the MCL program generates an error and quits.

A few (very few) other MCL statements operate differently in different environments. The most important example is `"Select database..."`. In an interactive environment such as xvmerlin, the database to be

selected is expected to be already open; if not, the statement fails and a warning is issued. In a batch environment, "`Select database...`" connects to servers and opens specified database(s); the statement only fails if a database fails to open for some reason.

5. MCL Objects

The most fundamental MCL object is the database, selected with the "`Select database...`" statement. All MCL statements apply to the currently selected database.

To be useful, each database must have one or more named columns, which represent a kind of data (actually, one field of a datatype) and a function (e.g., the `FIRST, LONGEST, AVERAGE, COUNT,` etc. of that kind of data).

To be useful, each database must also have one or more named hitlists, which represent an ordered set of entries in the database. Positions in a hitlist are referred to as "rows". On initial creation, hitlists represent all entries in the databases, i.e., there are the same number of rows in a hitlist as there are entries in its database, and the initial order is defined as the "native order" of the database.

Hitlists have a "current position" property. Although this is referred to by its numeric (1-origin) position in the hitlist, the "current position" is defined by the entry in that position, and is stable with respect to operations which change the hitlist order. In other words, if the entry at the "current position" before an operation is present after the operation, it will still be current.

6. Defaults and Style

To produce readable MCL code, it is wise to select names carefully. In our examples we give columns short names consisting capital letters (which tends to make them stand out) and hitlist names that end with the word "hits", e.g., "hits" or "curhits" or "savehits".

A number of conventions are used in our examples for clarity: hitlists are usually specified explicity, user-supplied names are shown quoted, and each statement is shown on its own line, e.g.:

```
Reset "hits".
Sort "hits" by "COST".
Sort "hits" by "CLUSTER".
Remove "CLUSTER" repeats in "hits".
```

In general, columns and hitlists may be specified using only their names. If it seems clearer to do so, the keywords "column" or "hitlist" may precede column and hitlist names, although it is never required. The following MCL code produces the same program as that above:

```
Reset hitlist "hits".
Sort hitlist "hits" by column "COST".
Sort hitlist "hits" by column "CLUSTER".
Remove column "CLUSTER" repeats in hitlist "hits".
```

However, it is perfectly acceptable to omit quotes and references to the default hitlist as long as the meaning remains clear, e.g., the following MCL code also means the same thing as that shown above:

```
Reset.  Sort by COST.  Sort by CLUSTER.  Remove CLUSTER repeats.
```

Tautomer and graph searches are unusual because they are done by the Thor server directly from SMILES (all other searches are done by the Merlin server on data in columns). In these searches (only) the search is always done with data from a SMILES ($SMI) column; MCL syntax gives you no opportunity to move to the next tautomer in an existing hitlist or specify which column is to be searched, e.g.:

```
Put tautomers of "O=c1[nH]c2cncnc2[nH]1" into "hits".
```

7. Search Logic

Merlin provides 10 different types of searches:

- find superstructure of a given molecule (replacing H's)
- find molecules which match a given SMARTS pattern
- find substructures with a given embedded molecule
- find tautomers of a given molecule
- find molecules with same oxdation-suppresed graph as given molecule
- find structures which are similar to a given molecule
- find strings which are contain an given embedded substring
- find strings which match a regular expression exactly
- find strings which match a regular expression approximately
- find values in a given range

Each type of search may be invoked to produce in seven different actions:

- create a new hitlist of matching entries
- add matching entries to a hitlist
- add non-matching entries to a hitlist
- delete matching entries from a hitlist
- delete non-matching entries from a hitlist
- find matching entry in a hitlist
- find non-matching entry in a hitlist

The English-like MCL language allows these (70) different searches to be specified quite naturally, e.g., to create a hitlist of dopamine superstructures:

```
Put SMILES superstructures of "NCCc1ccc(O)c(O)c1" into "hitlist".
```

Note that there is no way to create a hitlist of non-matches using just one statement. Use two statements to do this, e.g.,

```
Put SMILES superstructures of "NCCc1ccc(O)c(O)c1" into "hitlist".
Invert "hitlist".
```

or use the reverse logic:

```
Clear "hitlist".
Add SMILES non superstructures of "NCCc1ccc(O)c(O)c1" to "hitlist".
```

In general, "non" or "not" is used as a word to reverse the meaning of a match. The one exception is similarity search where the following two statements refer to complementary sets of structures:

```
Remove SMILES structures at least  0.9 similar to "NCCc1cc(O)c(O)c1".
Remove SMILES structures less than 0.9 similar to "NCCc1cc(O)c(O)c1".
```

In an attempt to clarify the effect of a search operation, MCL syntax requires that the preposition must match the verb, e.g.:

```
 Put  ...... into "hitlist".
 Add  ...... to   "hitlist".
 Remove .... from "hitlist".
 Move to ... in   "hitlist".
```

"Move to ..." searches cause the current position to be moved to the next entry which meets the given requirements, i.e., the next position in the current hitlist, starting with the current position. For example, this MCL code produces a hitlist of entries with known pKa's, sorted by pKa:

```
Reset "hits".
Remove missing "pKa" from "hits".
Sort "hits" by "pKa".
```

If we move to the first entry in this list which meets another criterion, e.g., a given pharmacological activity, we are assured that we are pointing to the one which has the lowest-valued pKa:

```
Move to row 1.
Move to "ACTIVITY" string containing "NARCOTIC" in "hits".
```

We can remove entries with lower pKa values (row 0 is current position):

```
Remove above row 0 of "hits".
```

And then do the reverse:

```
Reverse "hits".
Move to row 1.
Move to "ACTIVITY" string containing "NARCOTIC" in "hits".
Remove above row 0 of "hits".
```

The resulting hitlist now contains "all entries with pKa's in the range of pKa's of known narcotics". One might repeat this type of search for other properties (LogP, CMR, etc.) to select structures which meet an observed physiochemical profile (e.g., of known narcotics).

All Merlin (and thus MCL) sorts are stable, i.e., after a sort, the order of entries with equal value is unchanged. For example, the following MCL code produces a hitlist containing only the lowest-cost member of each cluster.

```
Reset "hits".
Sort "hits" by "COST".
Sort "hits" by "CLUSTER".
Remove "CLUSTER" repeats in "hits".
```

8. MCL Statements

Select database "database" ["thorbase"].

Selects a database by name in the form: `base@host:service:user.` "database" is used for Merlin access; "thorbase" for Thor access. If "thorbase" is not specified, "database" is used with the service name "thor". Only the "base" part of the name is needed, e.g. "wdi93". Default values are the local machine name for "host"; "merlin" or "thor" for "service"; and the user's login name for "user".

Since MCL is designed to run in a non-interactive environment, one may specify user and database read passwords ("userpw" and "basepw" respectively) in database names, i.e.,

```
base%basepw@host:service:user%userpw
```

Beware: it is tempting to write passwords directly into MCL programs and scripts -- if you are maintaining a secure database, having such files around will undermine your security. Assume you have an MCL program ("myprog.mcl") which accesses a database ("hotdrugs") with a read password ("passone") served by the thor services on a remote host ("computer") with thor user password ("passtwo"). What you want to avoid is writing database names into the MCL program, e.g.,

```
Select database "hotdrugs%passone@computer:thor:thor%passtwo".
```

Instead, program MCL database selection using parameter(s):

```
Select database $1.
```

and call the MCL program with a database parameter containing environment variables (which can be in a script):

```
$ mcl -i myprog "hotdrugs%$MY_HOTPASS@computer:thor:thor%$MY_PASS"
```

The MCL program (and the script calling it) will work if you have set the environment variables MY_HOTPASS and MY_PASS correctly, which you can do interactively if you desire. This approach isn't perfect and it's a bit of a hassle (universal aspects of security), but is well-suited to many of our customer's environments.

Create column of datatype "tag" [field "field"] [function <func>] named "name".

Creates a named column. "tag" is an internal tag, e.g. "$CAS".

"field" may be either a field name (e.g., "Reference") or the 0-origin index of the field, preceded by '#' (e.g., **"#0"** refers to the first field). If unspecified, the first field (**"#0"**) is used.

If specified, <func> must be one of the following:

```
ALL         AVERAGE     COUNT       FIRST       LAST
LONGEST     MAXIMUM     MINIMUM     SHORTEST    STDDEV
```

If unspecified, FIRST (the first dataitem) is used.

Two kinds of columns are special in that they are required for certain kinds of searches: a column of type "$SMI" (SMILES) is required for tautomer and graph searches and a column of type "SIMILARITY" is required for sorting or searching by structural similarity.

By convention, columns are named after the type of data and are composed of capital letters. However, any string can be used (though you might have to quote it), e.g.,

```
Create column of datatype P function AVERAGE named "Mean(LogP(o/w))".
```

Create hitlist "hitlist".

Creates an empty, named hitlist. Hitlists are relatively expensive, so it's a good idea to reuse them. By convention, hitlists names end with "hits", e.g., "hits", "curhits" and "bkphits". However, any string may be used (though you might have put quotes around it).

Free column "column".

Free hitlist "hitlist".

Free database "database".

Free object by name in current context (database). Freeing a database automatically frees its columns and hitlists.

Set default hitlist "hitlist".

Make named hitlist default -- if the hitlist is not specified in MCL statements where it is optional to do so, this hitlist will be used. The first hitlist created for each database is normally the default and need not be set explicitly. It is important to set the default hitlist only if you want to use a different one or if you deallocate an existing default hitlist (with Free hitlist...).

Reset ["hitlist"].

Reset the named (or default) hitlist such that all rows are hit, i.e., same as xvmerlin's "Set all hit" menu item.

Note: this doesn't quite mean "reset to initial state", because the current hit (if any) remains current. You can truly reset to the initial state by:

```
Reset "hitlist".  Move to row 1 of "hitlist".
```

Clear ["hitlist"].

Clear the named (or default) hitlist such that no rows are hit, i.e. same as the sequence:

```
Reset "hitlist".  Invert "hitlist".
```

Invert ["hitlist"].

Invert the named (or default) hitlist, i.e., make non-hits hits and vise-versa.

Reverse ["hitlist"].

Reverse the order of the named (or default) hitlist.

Copy "hitlist" to "hitlist2".

Copy the contents of one extant hitlist to another. This can be used to backup a hitlist (e.g., `Copy curhits to bkphits.`) for later operations such as restoration (e.g., `Copy bkphits to curhits.`)

Add "hitlist" to "hitlist2".

Add hits in the first hitlist to those in the second. E.g., the statement

```
Add bkphits to curhits.
```

replaces curhits with the union of `curhits` and `bkphits`.

Select "hitlist" in "hitlist2".

Remove hits in the second hitlist which are not in the first. E.g.,

```
Select bkphits in curhits.
```

replaces curhits with the intersection of `curhits` and `bkphits`.

Exchange "hitlist" with "hitlist2".

Exchange the contents of the two hitlists, e.g., the implementation of the "undo" facility in xvmerlin is equivalent to:

```
Exchange undohits with curhits.
```

Move to row "integer" [of "hitlist"].

Set the current hitlist position, where "integer" is interpreted as:

```
unsigned number .... absolute row number (1 is first row)
zero ............... current row
signed number ...... number relative to current row
```

Moves to approprate extreme value (first or last row) if out-of-range, e.g.,

```
Move to row 9999999.
```

will move to the last row of most databases.

Remove row "integer" [of "hitlist"].
Remove above row "integer" [of "hitlist"].
Remove below row "integer" [of "hitlist"].
Removes row(s) from hitlist.

Remove "column" repeats [in "hitlist"].
Removes rows from hitlist with values in given column which are identical to the value in the previous row. Typically used to select the first row of each value in a sorted list.

Remove missing "column" [in "hitlist"].
Removes rows from hitlist for which data is not available in given column.

Put "column" superstructures of "smiles" [into "hitlist"].
Add "column" [non] superstructures of "smiles" [to "hitlist"].
Remove "column" [non] superstructures of "smiles" [from "hitlist"].
Move to "column" [non] superstructures of "smiles" [in "hitlist"].
The classic "substructure" search. The "Put" form replaces the given hitlist; "Add" and "Remove" modify it; "Move" sets the current position but leaves the hitlist unchanged. $SMI or ISM columns are typically used.

Put "column" structures matching "smarts" [into "hitlist"].
Add "column" structures [not] matching "smarts" [to "hitlist"].
Remove "column" structures [not] matching "smarts" [from "hitlist"].
Move to "column" structure [not] matching "smarts" [in "hitlist"].
Search for a SMARTS pattern. "Put" form replaces the given hitlist; "Add" and "Remove" modify it; "Move" resets the current position but leaves the hitlist unchanged. $SMI or ISM columns are typically used.

Put "column" substructures of "smiles" [into "hitlist"].
Add "column" [non] substructures of "smiles" [to "hitlist"].
Remove "column" [non] substructures of "smiles" [from "hitlist"].
Move to "column" [non] substructure of "smiles" [in "hitlist"].
Converse of the classic "substructure" search, this one looks for structures embedded in the given molecule. The "Put" form replaces the given hitlist; "Add" and "Remove" modify it; "Move" resets the current position but leaves the hitlist unchanged. $SMI or ISM columns are typically used for "column".

Put "column" structures <op> "tan" similar to "smiles" [into "hitlist"].
Add "column" structures <op> "tan" similar to "smiles" [to "hitlist"].
Remove "column" structures <op> "tan" similar to "smiles" [from "hitlist"].
Move to "column" structure <op> "tan" similar to "smiles" [in "hitlist"].
... where <op> is <**at least | less than**>.
Select (or find) structures based on similarity to a given molecule. "tan" is a number indicating Tanimoto similarity for qualification where 1.0 is perfect similarity (identity). These values are typically used:

0.90 ..... very high
0.75 ..... high
0.60 ..... moderate
0.50 ..... rough

$SMI or ISM columns are typically used for "column", e.g.,

```
Add SMILES structures at least 0.9 similar to "NCCc1ccc(O)c(O)c1".
```

Note: a column of type SIMILARITY must exist to do these searches.

Beginning with 4.9, the input 'smiles' argument may be an ASCII fingerprint string. Merlinserver checks the input string and if it detects a fingerprint it uses the fingerprint as the query directly. Note that the size of the fingerprint must be at least as large as the largest fingerprint in the pool being searched, otherwise an error is returned (merlinserver can fold the query down as needed but can't "unfold" one to match a larger database fingerprint).

Put "column" structures <op> "value" Tversky alpha beta to "smiles" [into "hitlist"].
Add "column" structures <op> "value" Tversky alpha beta to "smiles" [to "hitlist"].
Remove "column" structures <op> "value" Tversky alpha beta to "smiles" [from "hitlist"].
Move to "column" structure <op> "value" Tversky alpha beta to "smiles" [in "hitlist"].
... where <op> is <at least | less than>

Select (or find) structures based on Tversky similarity to a given molecule using given alpha/beta parameters. Alpha and beta are typically in the range 0.0 - 1.0. "value" is a number indicating Tversky similarity; the meaning depends on specific alpha/beta settings but higher values are typically used than with Tanimoto similarity, e.g.,

0.95 ..... very highly similar
0.90 ..... highly similar
0.85 ..... moderately similar
0.80 ..... roughly similar
0.00 ..... select all structures

$SMI or ISM columns are typically used for "column", e.g.,

```
Add structures at least 0.9 Tversky alpha
0.9 beta 0.1 to "NCCc1ccc(O)c(O)c1".
```

Note: a column of type SIMILARITY must exist to do these searches.

Beginning with 4.9, the input 'smiles' argument may be an ASCII fingerprint string. Merlinserver checks the input string and if it detects a fingerprint it uses the fingerprint as the query directly. Note that the size of the fingerprint must be at least as large as the largest fingerprint in the pool being searched, otherwise an error is returned (merlinserver can fold the query down as needed but can't "unfold" one to match a larger database fingerprint).

Put <tautomers | graphs> of "smiles" [into "hitlist"].
Add [non] <tautomers | graphs> of "smiles" [to "hitlist"].
Remove [non] <tautomers | graphs> of "smiles" [from "hitlist"].

Select tautomers or graphs of a given molecule. A "graph" match discounts oxidation state completely, i.e., the molecules' heavy atoms are connected in the same way (ignoring hydrogens, bond orders, and charges). A "tautomer" match requires a graph match and the net charge and hydrogen count much match, i.e., molecules which differ only in positions of H's (protons) and

electrons. (Labile hydrogens are not distinguished from non-labile ones.)

Note: to do these searches, a column of type $SMI (SMILES) must exist and the Thor database must have a $SMI datatype with AUTOGRAPH normalization (this is true for default Thor databases).

Put "column" strings containing[/cmp] "string" [into "hitlist"].
Add "column" strings [not] containing[/cmp] "string" [to "hitlist"].
Remove "column" strings [not] containing[/cmp] "string" [from "hitlist"].
Move to "column" string [not] containing[/cmp] "string" [in "hitlist"].

where cmp is one of:

```
/ASC ... ASCII (default)       /ANCW ... ignore case and w/s
/ANC ... ignore case           /ANCP ... ignand case or punct
/ANW ... ignore whitespace     /ANPW ... ignore punct and w/s
/ANP ... ignore punctuation    /ANCPW .. ignore case, punct, and w/s
```

Select (or find) strings containing a given substring. Various character classes can be ignored by appending an option to the keyword "containing". For instance, the statement:

```
Put "NAME" strings containing/ANCPW "METHYLPYRROLE" into "curhits".
```

does comparisons ignoring case, punctuation, and whitespace (spaces, tabs, and newlines), i.e., makes a hitlist of entries with names containing the substrings "Methyl Pyrrole", "methyl-pyrrole", "methylpyrrole", etc.

Put "column" strings matching "regexp" [into "hitlist"].
Add "column" strings [not] matching "regexp" [to "hitlist"].
Remove "column" strings [not] matching "regexp" [from "hitlist"].
Move to "column" string [not] matching "regexp" [in "hitlist"].

Select (or find) strings matching a regular expression.

Put "column" values in range "min" to "max" [into "hitlist"].
Add "column" values [not] in range "min" to "max" [to "hitlist"].
Remove "column" values [not] in range "min" to "max" [from "hitlist"].
Move to "column" value [not] in range "min" to "max" [in "hitlist"].

Select (or find) values by range specification.

Sort[/cmp] "hitlist" by "column".

where cmp is one of:

```
/ASC ..... ASCII                 /ANCPW ... ignore case, punct and w/s
/ANC ..... ignore case           /AAZ ..... letters only, ignore case
/NUM ..... numeric               /ANP ..... ignore punctuation
/NAB ..... absolute numeric      /ANW ..... ignore whitespace
/CAS ..... CAS Number compare    /ANCP .... ignore case and punct
/ANCW .... ignore case and w/s   /MF ...... molecular formula
/ANPW .... ignore punct and w/s  /LEN ..... length of string
```

Sort hitlist by value of given column. If cmp is not specified, the "normal" comparison type for the column's datatype will be used.

Sort ["hitlist"] by nativeorder.

Rearrange the given hitlist to original pool order.

Sort ["hitlist"] by similarity ["column"] [to "smiles"].

Sort hitlist by Tanimoto similarity to given "smiles", saving similarity values in similarity column "column", if specified. If "column" is omitted, the default similarity column will be used. If "smiles" is omitted, existing values in the column will be used.

This produces a descending sort in similarity, i.e., the most similar structures (high similarity values) sort to the top of the list.

Set font "fontname".

Set lines per row to "lpr".

Display smiles as <text | depiction>.

Special functions for interactive environments such as xvmerlin. "fontname" must as specified in a FONT_ options (or "default"). "lpr" (lines per row) must be in the range 1-8.

Note: These statements are disabled in non-interactive environments such as the `mcl' program (they do nothing, successfully).

Print status.

Print the status of the current environment: name the database, name all columns and show their datatypes, name all hitlists and show their length, indicate which hitlist is default and report the current hitlist position.

Print "string" ["string" ...]

Print string literally, followed by newline. If more than one string is supplied they are concatenated then output. Special characters such as newline and bell are copied to output literally. There is no way to print the NULL character to output.

Print <list | table> [of "hitlist"] from row "integer" to row "integer"
**   [containing "column" ["column" ...]]   [entitled "string" ["string" ...]]**

where integer row values are interpreted as:

```
unsigned number .... absolute row number (1 is first row)
zero ............... current row
signed number ...... number relative to current row
```

If the `containing ...' phrase is specified, only those columns will be printed; if omitted, all columns will be printed. If the `entitled ...' phrase is specified, the following strings will be printed as a title; if omitted, no title will be printed. For example, the statement: Print table of curhits from row 1 to row 100 containing "SMILES" "Name" entitled "Table 1. The first 100 entries in this hitlist.". produces a 2-column table of the first 100 rows in the hitlist "curhits". If the hitlist is shorter than 100, the table will be truncated. Print list from row -10 to row +10. prints data from all columns in the default hitlist for the current position and 10 rows of context above and below (truncated as needed).

Write ["hitlist"] to file "filename".

Save the hitlist to the given file as root ID's in TDT format. Such files are typically given the suffix ".hits" and used for later restoration with the MCL "Read" statement.

If the given file already exists and is writable, it is overwritten. If the given file is not writable, an error message is generated and MCL execution continues.

If the filename is "STDOUT", standard output is using instead of a file.

Read ["hitlist"] from file "filename".

Restore the hitlist from the given file, typically one previously generated by an MCL "Write" statement.

Root ID's in the input file must match those in the database exactly. This is the case if the file was generated via the "Write" statement. If generated from another source, one must insure that all ID's are normalized, e.g., case must match, SMILES must be uniquified, etc.

If the given file does not exist or is not readable, an error message is generated and MCL execution continues.

If the filename is "STDIN", standard input is used instead of a file.

9. Changes in this Release

Changes for Release 4.95:

- Abillity to take fingerprint instead of smiles for Tanimoto and Tversky searches.

Changes for Release 4.51:

- Tversky similarity added.

Changes for Release 4.42:

- Implemented hitlist save ("Write") and restore ("Read").
- Read, Write, and file become reserved keywords.
- HTML output option added to MCL processor (mcl -h).
- Password specifications are allowed in database names.

10. Index