3-Dimensional Pharmaceuticals, Inc.

BETTER PHARMACEUTICALS BY DESIGN®

DirectedDiversity® Reference

Table of Contents

Reference One

Reaction Language

Overview

This reference contains addition information about the reaction language used for reaction scripting in the DirectedDiversity® Browser's Exploder tool.

Specifically, this Reference includes the following sections:

Exploder Reaction Scripting

Stereochemistry in Reaction Scripting

The following table identifies the appropriate section to read in order to learn more about a specific area.

To learn more about …	Read the section….	See page
Writing a reaction script	Exploder Reaction Scripting	2
Writing a reaction script that takes stereochemistry into account.	Stereochemistry in Reaction Scripting	7

Exploder Reaction Scripting

General Syntax Rules

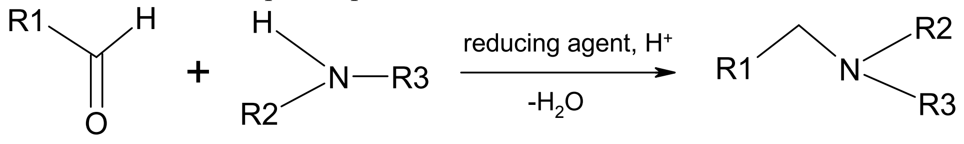
The syntax of the Exploder reaction scripting language is similar to that of the Tcl programming language (see `http://www.scriptics.com/resource/` for more information on Tcl).
Here are the general syntax rules:

- Semicolons and newlines are command separators.
- The reaction scripting language is case sensitive.
- Identifiers should not include spaces.
- String literals are enclosed in double quotes.
- Comments begin with the # symbol which is not a part of a string and extend to the end of line.

An Exploder script can contain one and only one reaction scheme definition.

Sample Script

Shown below is a sample script for the reductive amination reaction:



```
proc amination {libname libfile amines aldehydes} {
   reagent amine;
   reagent aldehyde;
   amine ignore "C(=O)[NH2]";
   amine ignore "C(=O)[NH][C,c]";
   amine define "[C,c][NH2]";
   amine define "[C,c][NH&!a]C";
   aldehyde define "[C,c][CH]=O";
   product p;
   p add amine $amines;
   p add aldehyde $aldehydes;
   p insert bond aldehyde:1 amine:1;
   p remove atom aldehyde:2;
   p explode $libname $libfile;
}
```

Reaction Scheme Definition

As it can be seen in the above example, a reaction scheme is defined using the **proc** keyword as shown below:

```
proc name {library_name library_file r1_file [r2_file ...]} {
   procedure_body
}
```

where:
`name` is the reaction script name, typically it reflect the reaction name, e.g. amination.
`library_name` is a parameter that specifies name that will be given to the generated library. The Exploder will replace `library_name` with the name specified by the user. The name of the library and will be used to construct the names of the products in that library: each product molecule will be named `name-serial_number`.
`library_file` is a parameter that specifies name of the output file for the generated library. The Exploder will replace `library_file` with the filename specified by the user.
`r1_file, r2_file, ...` are parameters that specify filenames of the reagent sources. The Exploder will replace the `r1_file, r2_file,...` arguments with the names of SDF or SMILES files specified by the user.
`procedure_body` consists of three parts, a **reagent definition**, a **product definition**, and an **execution trigger** which will be described below.

NOTE: The opening figure bracket of the script's body must be on the same line as the argument list.

Reagent Definition

The reagent definition section of a script specifies the reagents involved in the reaction and the patterns that identify the reagents:

```
reagent reagent1
reagent1 define pattern1 [pattern2 ...]
```

Patterns are specified as SMARTS strings (see Reference Chapter 5. SMARTS) enclosed in double quotes. It is possible to define several patterns for a single reagent, in which case the program will attempt to match the patterns in the order they are defined. However, the sequence number of each atom that participates in the reaction should be the same in all patterns defined for that particular reagent.

The **`ignore`** keyword is used to specify undesirable patterns:

```
reagent1 ignore ignore_pattern1 [ignore_pattern2 ...]
```

Then, if the Exploder finds that a fragment of `reagent1` matches `ignore_pattern1`, it will not use it in the reaction even if this fragment also matches `pattern1` specified with the **`define`** keyword. NOTE: patterns are only ignored if the **reactive** pattern is a **subset** of the **ignore** pattern and **not** if they just intersect. For instance, in the sample script at the beginning of the chapter, the following statements

```
amine ignore  "C(=O)[NH2]";
amine ignore  "C(=O)[NH][C,c]";
```

tell the Exploder not to use primary and secondary amides in the reaction.

Often, a pattern can be mapped onto a reagent in multiple ways. A desired mapping policy for a reagent can be explicitly specified with the **`policy`** keyword followed by one of the following keywords:
`first` - only one random mapping is used (this is the default policy),
`all` - all possible mappings are used (NOTE: Browser will show only one of the products resulted from multiple mappings. To extract all products from the combinatorial library file created by the Exploder tool, use the **zbimap** and **molextract** command-line utilities),
`single` - only those reagents that can be mapped unambiguously are used.

```
reagent1 policy { first | all | single}
```

A special type of reagent is called an agent. Agent differs from reagent it the way it is defined. Agent is defined directly in the script (there is no need to provide a source file) and it can have only one chemical structure associated with it. For example, a fixed scaffold used in the reaction can be defined as agent:

```
agent agent1
agent1 define smiles
```

In all other respects agents are identical to reagents.

Product Definition

The product definition section of a script declares the product and specifies the transformations of the reagents that lead to its formation. Note: only one product per reaction can be defined. The steps that can be involved in the product definition are described below.

Declare product *`p`* (required):

```
   product p
```

Add a molecule of *`reagent1`* from source *`r1`* (at least one molecule of a reagent should be added to form a product):

```
   p add reagent1 $r1
```

Add a molecule of *`reagent2`* from source *`r2`* to product *`p`*:

```
   p add reagent2 $r2
```

When adding an agent to the product, source should be omitted:

```
   p add agent3
```

Add an additional molecule of *`reagent1`* to product *`p`* and declare the name of that fragment as *`reagent1copy`*, which can be used to refer to that fragment later. Copying a reagent molecule is useful when reaction stoichiometry is not 1:1:

```
p copy reagent1 reagent1copy
```

Remove the atom `a1` of `reagent1`:

```
p remove atom reagent1:a1
```

An atom involved in the transformations is identified by the name of the reagent from which it comes and its sequence number in the reagent's pattern. For instance, in the sample script at the beginning of the chapter the following statement

```
p remove atom aldehyde:2
```

instructs the Exploder to remove the oxygen atom that comes from the aldehyde reagent `"C[CH]=O"`. NOTE: Oxygen is the third atom in this pattern, but the atoms are numbered starting from 0, therefore, its index is 2.

Insert a bond of the order `bond_order` (**`single`**, **`double`** or **`triple`**; **`single`** is the default) between atom `a1` of `reagent1` and atom `a2` of `reagent2`.

```
p insert bond reagent1:a1 reagent2:a2 [bond_order]
```

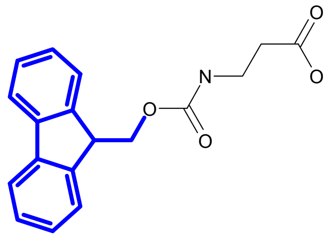
Remove a bond between atoms `a1` and `a2` of `reagent1`.

```
p remove bond reagent1:a1 reagent1:a2
```

Remove attachments to a specified atom.

```
p remove attachment reagent1:a1
p remove fragment reagent1:a1 reagent2:a2
```

NOTE: The **`remove attachment`** statement removes all substituents attached to the atom and not mapped by the reagent's pattern. For example, if the reagent `r1` is defined as `OC(=O)N` and mapped onto `O=C(O)CCNC(=O)OCC1c2ccccc2c3ccccc13`) (FMOC-β-ALA), then **`remove attachment`** `r1:0` would remove the highlighted part of the FMOC substructure attached to the oxygen atom.



The **`remove fragment`** `reagent1:a1 reagent2:a2` statement is similar to **`remove attachment`**. It removes the second of two connected atoms and everything else attached to that atom, whether the removed atoms overlap with the reagent's pattern or not.

Set an atom's property:

```
   p set atom reagent1:a1 property value
```

The following atom properties can be set: **`symbol`**, **`atomicNo`**, **`radical`**, **`charge`**, **`mass`**, and **`chirality`**. Chiral centers can be described as **`R`**, **`S`**, **`th`** (tetrahedral) or **`al`** (allenyl).

Set a bond's property:

```
   p set bond reagent1:a1 reagent1:a2 order/wedge
```

The bond order can be set to **`single`**, **`double`**, or **`triple`**; the wedge property can be set to **`up`** or **`down`**.

Conditional handling of substructures which may or may not be present (e.g. leaving protecting groups) can be achieved using the **`if`** statement. Note that the mapped leaving group is referred to as **`pattern`**. All conditional operations must be placed after the general reaction definition. For example:

```
if {[reagent1 contains "*C(c1ccccc1)(c2ccccc2)c3ccccc3"]} {
   product1 remove fragment pattern:0 pattern:1;
};
```

Execution Trigger

To initiate exploding of the library the following statement must be added at the end of the script:

```
   p explode $library
```

The **`explode`** statement should be the last statement the script. None of the instructions after **explode** will be executed.

Stereochemistry in Reaction Scripting

An important feature of the stereochemical reactions is that a combination of reagents normally leads to multiple products that are in stereoisomeric relationships with each other. Specification of the stereochemically preferred product or products (e.g. due to a stereospecificity or stereoselectivity of the reaction) in the reaction script can be done in a number of ways.

Foremost, the stereochemical configuration of an atom can be explicitly specified as **R**, **S**, **`unspecified`**, **`racemic`** or **`inverse`**. For example:

`product` **`set atom`** *`reagent`*`:`*`n`* **`configuration R`**

Configurations **`R`**, **`S`**, or **`unspecified`** indicate a single product of the specified or unspecified chirality. The **`racemic`** configuration indicates a mixture of both R and S stereoisomers. The **`inverse`** specification can be applied to a chiral atom, which exchanges one of its substituents during the reaction. The correct R/S configuration of the product will be determined based on the inversion of the original configuration.

Alternatively, the stereochemical configuration of an atom can be specified by listing the atom's substituents in a clockwise order and designating last substituent as an **`up`** or **`down`** wedge. For example:

`product` **`set atom`** *`reagentA`*`:`*`n`* **`configuration`** *`reagentA`*`:`*`n1`*
`reagentA``:`*`n2`* *`reagentB`*`:`*`n3`* `H` **`down`**

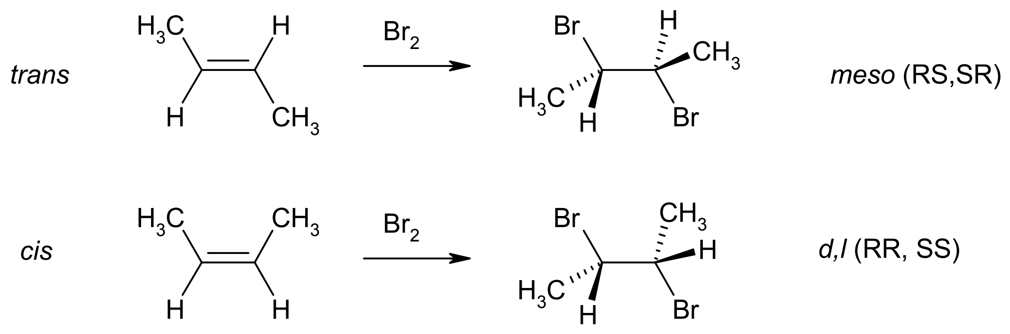
The stereochemical configuration of a bond can be also specified explicitly as **`cis`** or **`trans`** or as **`E`** or **`Z`**. In addition, because often bonds are formed and modified during the reactions according to a certain mechanism, the stereochemical configuration of a bond can be specified as a **`syn_product`** or an **`anti_product`** of the reaction transformations. Note that it is not always possible to specify a single product by using the **`syn_product`** and **`anti_product`** keywords.

The examples below discuss various types of stereochemical reactions and demonstrate how a desired stereochemical product can be specified in the reaction script.

Addition to a double bond

Use **syn_product** or **anti_product** to specify the correct mechanism. For example, bromination of olefins is an *anti*-addition (i.e. first bromine atom attacks from one side of the double bond's plane, whereas the second atom attacks from the other side). Hence, for each of the two possible configurations of the double bond (*cis* or *trans*), the formation of two stereoisomers is possible.

Here is a reaction script example:

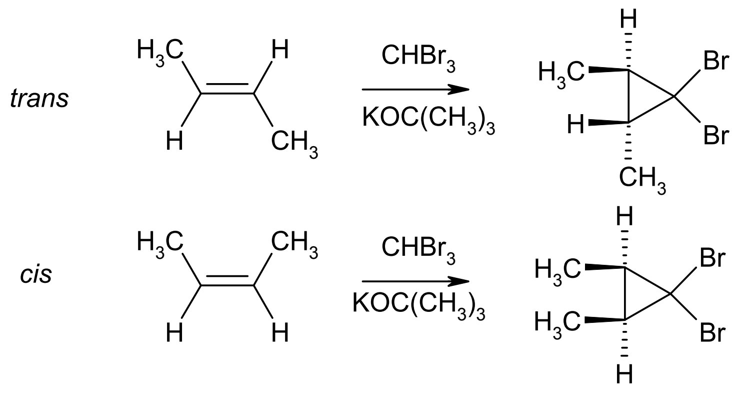


```
proc anti_addition {name lib olefins brominating_agents} {
   reagent olefin;
   reagent br2;
   olefin define "C=C";
   br2 define "[Br][Br]";
   product p;
   p add olefin $olefins;
   p add br2 $brominating_agents;
   p insert bond olefin:0 br2:0;
   p insert bond olefin:1 br2:1;
   p remove bond br2:0 br2:1;
   p set bond olefin:0 olefin:1 single anti_product;
   p explode $name $lib;
}
```

Cyclo-addition to a double bond

Cyclo-addition to a double bond is simply a *syn*-addition because of the ring constrain. Therefore, **`syn_product`** would specify the correct mechanism.

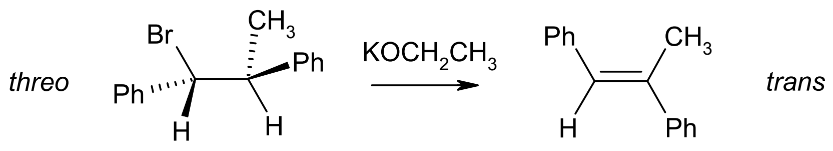
Here is a reaction script example:



```
proc cyclo_addition {name lib olefins brominating_agents} {
   reagent olefin;
   reagent bragent;
   olefin define "C=C";
   bragent define "C([Br])([Br])[Br]";
   product p;
   p add olefin $olefins;
   p add bragent $brominating_agents;
   p insert bond olefin:0 bragent:0;
   p insert bond olefin:1 bragent:0;
   p remove atom bragent:3;
   p set bond olefin:0 olefin:1 single syn_product;
   p explode $name $lib;
};
```

Elimination

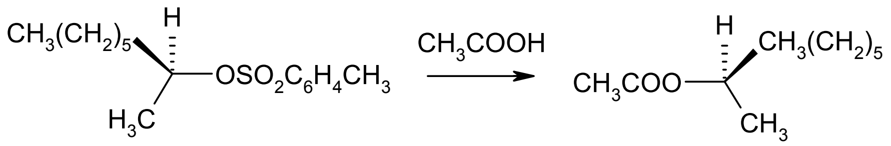
Elimination leading to the formation of a double bond can be either a *syn*-elimination, or *anti*-elimination. In the case of *syn*-elimination, the broken covalent bonds are on the same face of the formed double bond. In the case of *anti*-elimination, the two removed substituents are on the opposite faces of the formed double bond. To specify the desired reaction mechanism, the **syn_product** or **anti_product** keywords are used. For instance, dehydrohalogenation is an example of an *anti*-elimination reaction:



```
proc anti_elimination {name lib r1s} {
   reagent r1;
   r1 define "[CH]C[Br]";
   product p;
   p add r1 $r1s;
   p remove atom r1:2;
   p set bond r1:0 r1:1 double anti_product;
   p explode $name $lib;
};
```

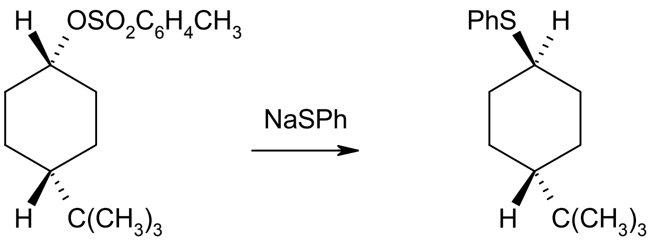
Nucleophilic substitution

Nucleophilic substitution can lead to inversion of the configuration of a chiral center. To indicate the inversion in the script, the **inverse** keyword is used. For example:



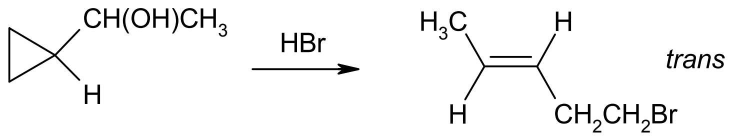
```
proc Sn_inversion {name lib r1s r2s} {
   reagent r1;
   reagent r2;
   r1 define "[C&X4]O";
   r2 define "C(=O)O";
   product p;
   p add r1 $r1s;
   p add r2 $r2s;
   p remove fragment r1:0 r1:1;
   p insert bond r1:0 r2:2 single;
   p set atom r1:0 configuration inverse;
   p explode $name $lib;
};
```

The **inverse** keyword is also applicable in the situation where the atom inverting its configuration is not chiral by itself, but forms a *stereo pair* with another atom. *Stereo pair* means that the substituents of the paired atoms can be in a *cis* or *trans* (or *E* or *Z*) configuration in respect to the plane of the paired atoms. For example, the reaction script above would be suitable for the following reaction as well after the redefinition of the second reagent as `r2 define "ccS"`.



Ring opening

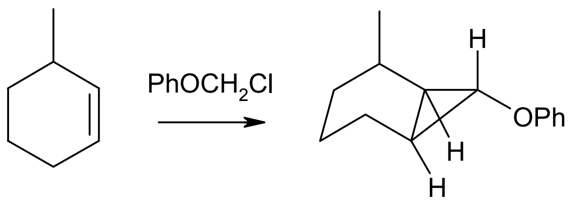
Acid-catalyzed ring-opening of cyclopropylcarbinols leads to the formation of a *trans* double bond, which is specified with the **trans** keyword in the sample script below:



```
proc ring_opening {name lib carbinols bragents} {
   reagent carbinol;
   reagent bragent;
   carbinol define "C1CC1C(O)";
   bragent define "[Br]";
   product p;
   p add carbinol $carbinols;
   p add bragent $bragents;
   p remove atom carbinol:4;
   p insert bond carbinol:0 bragent:0;
   p remove bond carbinol:0 carbinol:2;
   p set bond carbinol:2 carbinol:3 double trans;
   p explode $name $lib;
};
```

Fused ring formation

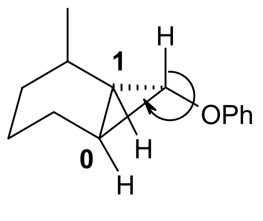
Addition of phenoxycarbene to cyclohexene is an example of how a *syn*-addition to an alkene creates fuzed rings and three chiral centers:



In the example below the addition to the double bond is specified to be *syn*-addition, and the configuration of the chiral center formed from the phenoxycarbene's carbon atom is explicitly defined:

```
proc alkene_addition {name lib r1s r2s} {
   reagent chexene;
   reagent carbene;
   chexene define "C=CC(C)C";
   carbene define "[Cl]CO";
   product p;
   p add chexene $r1s;
   p add carbene $r2s;
   p remove atom carbene:0;
   p insert bond chexene:0 carbene:1;
   p insert bond chexene:1 carbene:1;
   p set bond chexene:0 chexene:1 single syn_product;
   p set atom carbene:1 configuration H carbene:2 chexene:0
     chexene:1 down;
   p explode $name $lib;
};
```

Note that configuration of the phenoxycarben's chiral atom is defined by listing substituents in the *clockwise* order and specifying whether the last substituent in the list is directed *up* or *down* in respect to the plane of drawing:



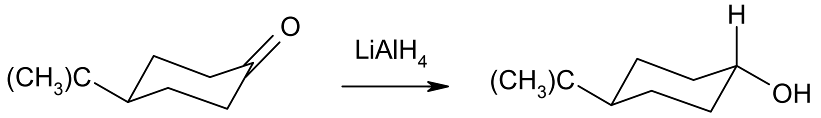
In fact, the script above is not unambiguous, because cyclohexene's carbon atoms could be in either R,S or S,R configurations. In order to specify a single product, the script line with the **syn_product** keyword should to be replaced with the following code:

```
   p set bond chexene:0 chexene:1 single;
   p set atom chexene:0 configuration S;
```

```
    p set atom chexene:1 configuration R;
```

Addition to Carbonyl Group

Addition to a carbonyl's double bond can create an asymmetric carbon atom, preferred configuration of which can often be specified with the **`set atom ... configuration ...`** statement. However, in the case of an exocyclic carbonyl, addition to a carbonyl's double bond can create a "stereo pair" of atoms:

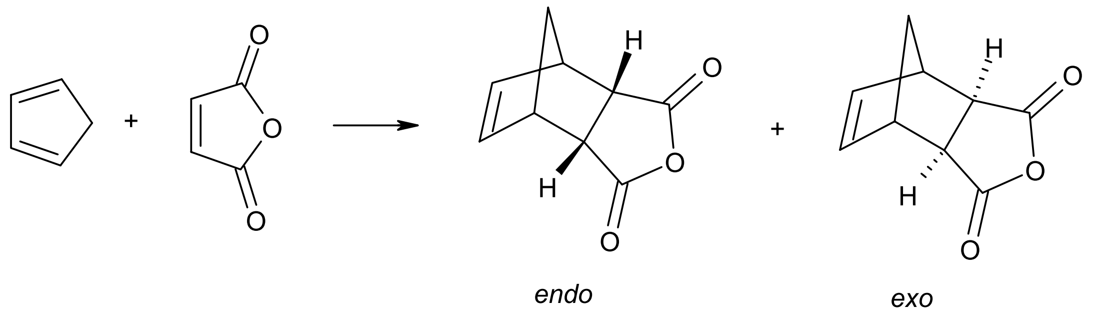


Cis or *trans* (or *E* or *Z*) configuration of the "stereo pair" can be specified by the special statement **`set pair ...`** . Here is an example of the reaction script using the **`set pair`** statement:

```
proc carbonyl_addition {name lib chexanones} {
   reagent chexanone;
   chexanone define "C1CCC(=O)CC1";
   product p;
   p add chexanone $chexanones;
   p set bond chexanone:3 chexanone:4 single;
   p set pair chexanone:0 chexanone:3 trans;
   p explode $name $lib;
};
```

Diels-Alder Reaction

Arguably the most important cycloaddition reaction, the Diels-Alder reaction, is stereospecific and is a *syn*-addition with respect to both the alkene and the diene:



Here is an example of the corresponding reaction script:

```
proc diels_alder {name lib r1s r2s} {
   reagent diene;
   reagent dienophile;
   diene define "C1C=CC=C1";
```

```
    dienophile define "*C=C*";
    product p;
    p add diene $r1s;
    p add dienophile $r2s;
    p insert bond diene:1 dienophile:1;
    p insert bond diene:4 dienophile:2;
    p set bond diene:1 diene:2 single;
    p set bond diene:3 diene:4 single;
    p set bond diene:2 diene:3 double;
    p set bond dienophile:1 dienophile:2 single syn_product;
    p explode $name $lib;
};
```

However, the empirical Alder rule says that if two isomeric adducts are possible, the one with an unsaturated substituent(s) on the alkene oriented toward the newly formed cyclohexene double bond is the preferred product. In the above example the addition of dienophiles to cyclopentadiene usually favors the *endo* stereoisomer. In order to specify that *endo* isomer is the main product, the script line with the **syn_product** keyword should to be replaced with the following code:

```
    p set bond dienophile:1 dienophile:2 single;
    p set atom dienophile:1 configuration dienophile:0
      dienophile:2 diene:1 H up;
    p set atom dienophile:2 configuration diene:4 dienophile:1
      dienophile:3 H up;
```

In addition, an upward direction of the norbornene bridge should be specified explicitly:

```
    p set atom diene:1 configuration diene:2 diene:0
      dienophile:1 H down;
    p set atom diene:4 configuration dienophile:2 diene:0
      diene:3 H down;
```

Reference Two

Command Utilities

Overview

Documentation for the command-line utilities is available as part of the DirectedDiversity® Browser installation in the form of an indexed, searchable help file.

Opening the Command-Line Utilities Documentation

Step	**Action**
1	Click on the Start menu button
2	Click on DirectedDiversity 3.5
3	Click on Documentation
4	Click on Command line tools ref

Reference Three

DirectedDiversity® File Formats

Overview

This reference contains additional information about the file formats used by the DirectedDiversity® tools.

Specifically, this reference includes the following sections:

- *Indexed Archives Format*
- *Matrix Files Format*
- *Structure Files Format*
- *ZBI Files Format*

The following table identifies the appropriate section to read in order to learn more about a specific area.

To learn more about …	Read the section….	See page
Indexed archives formatting	Indexed Archives Format	17
ASCII matrix file formatting	Matrix Files	18
Binary matrix file formatting		18
Interconverting ASCII and binary files		18
Numerical types for matrix elements		18
SDF files formatting	Structure Files Format	20
SMILES files formatting		20
CLB files formatting		20
ZBI files formatting	ZBI Files Format	21

Indexed Archives Format

Description

An indexed archive is a pair of two files, one containing a list of records in ASCII format, and another containing a list of indices in binary format. The first file can be any ASCII file that consists of records of the same type (albeit not necessarily the same length), such as an SDF or SMILES file. The second file, called an index file, is a binary file containing byte offsets of each record in the original ASCII file. This information allows fast direct access to any record in the original file, regardless of the size of the file. An indexed archive is created by indexing an ASCII file, i.e., by parsing it and generating the corresponding binary index file. This process does not modify the original ASCII file. An index file is placed either in the directory with the corresponding ASCII file, or in the system's temporary directory (see below). Its name is generated by adding the `.i` extension to the indexed ASCII file's name. Index files can be created either explicitly (e.g. by running the molindex utility), or implicitly by applications such as molconvert, the Browser and others.

Before an application uses an index file, it verifies that the indices are in sync with the corresponding ASCII file. If this is not the case, or if the index file cannot be found, the application re-creates the index file in the directory where the ASCII file to be indexed resides. If the user does not have permission to write in that directory, a temporary index file is created in the system's scratch directory (usually, C:\TEMP). Temporary index files exist only while the application that creates them is running, and are deleted when that application exits. NOTE: Deleting the `.i` index files is not harmful, since they will be automatically regenerated when needed (which will cause some delay in the program execution).

See Also

Structure File Utilities, Structure Files

Matrix Files Format

ASCII Matrix Files

ASCII matrix files are formatted as follows:

nrows`<ws>`*ncols*`<ws>`d_{11}`<ws>`d_{12}`<ws>. . .`$d_{1\ ncols}$`<ws>`d_{21}`<ws>`d_{22}`<ws>. . .`$d_{2\ ncols}$`<ws>. . .`$d_{nrows\ 1}$`<ws>`$d_{nrows\ 2}$`<ws>. . .`$d_{nrows\ ncols}$

where *nrows* and *ncols* are the number of rows and columns, respectively, <ws> is one or more whitespace characters (the space character (0x20) or any other character with the ASCII code in the range from 0x09 to 0x0D) . d_{ij} are string representations of the matrix elements formatted as follows:

[*sign*] [*digits*] [**.***digits*] [{**d** | **D** | **e** | **E**}[*sign*]*digits*]

Sign is either plus (+) or minus (–); and *digits* are one or more decimal digits. If no digits appear before the radix character (.), at least one must appear after the radix character. The decimal digits can be followed by an exponent, which consists of an introductory letter (**d**, **D**, **e**, or **E**) and an optionally signed integer. If neither an exponent part nor a radix character appears, a radix character is assumed to follow the last digit in the string. Here is an example of a matrix with 2 rows and 3 columns as it is written to an ASCII file:

```
2 3
1.0        2e-2 3.0
-3.1415926 4.8  7.235
```

Binary Matrix Files

NOTE: To ensure portability across various computer platforms, all DirectedDiversity® binary files, including matrix files, use the **big-endian** byte ordering. That is, the bytes with the lower addresses are more significant than the bytes with the higher addresses. **It is the OPPOSITE of the ordering used in the Intel Architecture**. If you need to transfer matrix data between DirectedDiversity® software and any other application, it is recommended that you use the ASCII format described above. The DirectedDiversity toolset contains the `mconvert` utility that can convert an ASCII matrix file to a binary matrix file and vice versa.

A binary matrix file contains a **header** followed by the **matrix elements** in the same order as described above for the ASCII matrix files. The header contains the number of rows and columns in the matrix represented as a pair of 32-bit integer numbers. The matrix elements can be of any one of the following numerical types:

Numerical Type	Description
float	The IEEE single-precision (32-bit) real
double	The IEEE double-precision (64-bit) real
long	32-bit signed two's complement integer with the sign bit located in bit 31
int	32-bit signed two's complement integer with the sign bit located in bit 31
short	16-bit signed two's complement integer with the sign bit located in bit 15
ulong	32-bit unsigned integer

uint	32-bit unsigned integer
ushort	16-bit unsigned integer

By default, all DirectedDiversity® software uses matrix files in the binary float format.

Binary Distance Matrix Format

Description

Binary distance matrices are real symmetric matrices containing all pairwise distances (or similarities) for a given set of objects. Distance matrices store only the lower triangle in a row-major order and are sometimes referred to as matrices stored in the symmetric matrix (packed) format. The diagonal elements may be different. A binary distance matrix file contains a **header** followed by the **matrix elements**, specified in binary format. The header contains the number of rows (and columns) in the matrix represented as a 32-bit integer number. The matrix elements can be of any one of the following numerical types:

`float`

The IEEE single-precision (32-bit) real

`double`

The IEEE double-precision (64-bit) real

NOTE: To ensure portability across various computer platforms, all DirectedDiversity® binary files, including distance matrix files, use the **big-endian** byte ordering. That is, the bytes with the lower addresses are more significant than the bytes with the higher addresses. **It is the opposite of the ordering used in the Intel® Architecture.**

Structure Files Format

SDF

SDF are MDL® Information Systems, Inc. **S**tructure-**D**ata **F**iles: An SD file contains structures and data for any number of molecules. The documentation on the SDF format can be downloaded from the MDL® Information Systems website (follow this link, proceed to the **Download Center**->**Free Products and Information** and download the **MDL File Formats** document).

SMILES Files

A SMILES file is an ASCII file that consists of one or more lines (terminated with the newline character with the ASCII code of 10) formatted as follows:

```
SMILES_STRING
[NAME][<TAG_NAME1>TAG1<TAG_NAME2>TAG2...<TAG_NAME_n>TAG_n]
```

The number of lines in the file is equal to the number of molecules it contains.

I. ***SMILES_STRING*** is a string representation of the molecule in the **S**implified **M**olecular **I**nput **L**ine **E**ntry **S**pecification format.

II. ***NAME*** is a name of the molecule. It can contain any number of any printable characters other than the "less than" < character (ASCII 32 – 59, 61-126); the trailing spaces are insignificant and are removed when a SMILES file is read by the DirectedDiversity® structure file utilities.

III. ***TAG_NAME*** is a name of the tag, which should be enclosed in <>, and ***TAG*** is the tag itself. ***TAG_NAME*** can contain any printable characters other than the "less than" < or "greater than" > character (ASCII 32 – 59, 61, 63-126). ***TAG*** can contain any printable character other than the "less than" < character (ASCII 32 – 59, 61-126); the trailing spaces are insignificant and are removed when a SMILES file is read by the DirectedDiversity® structure file utilities.

Here is an example of a SMILES file containing two molecules:

```
C=O formaldehyde<ID>1540<quantity>50g
ClC(Cl)(Cl)Cl carbon tetrachloride<ID>1541<quantity>100ml
```

CLB Files

The CLB abbreviation stands for **C**ombinatorial **LiB**rary format. The CLB format is a proprietary format developed by 3-Dimensional Pharmaceuticals, Inc. A CLB file contains complete information about the reagents and reaction scheme and thus stores the products of the encoded reaction in an implicit form. CLB files are created/read by structure file utilities and the DirectedDiversity® Browser.

ZBI Files Format

Description

A ZBI (**Z**ero-**B**ased **I**ndex) file is an ASCII files that contain a list of non-negative integers separated by one or more white space characters. These integers typically represent indices that identify a subset of elements in a set, such as a subset of structures in a structure file, a subset of rows or columns in a matrix file, etc. The first integer represents the size of the subset, i.e. the number of indices that follow, as shown below:

```
n
i1 i2 i3  ... in
```

ZBI files are used extensively in many DirectedDiversity® applications.

See Also

zbi, boolean, Structure Files, Matrix Files, Structutre File Utilities, Matrix Utilities

Reference Four

SMILES

Overview

This reference contains information about the SMILES (**S**implified **M**olecular **I**nput **L**ine **E**ntry **S**pecification) general-purpose chemical nomenclature and data exchange format. SMILES specifically represents a valence model of a molecule.

This reference includes the following sections:

- *SMILES Notation*
- *For More Information*

The following table identifies the appropriate section to read in order to learn more about a specific area.

To learn more about …	**Read the section….**	**See page**
Writing SMILES strings	SMILES Notation	24
Getting SMILES information from text and literature	For More Information	25
Getting SMILES information from web sites		25

SMILES Notation

SMILES is a linear notation for chemical graphs. In the SMILES notation, a chemical structure is represented by a line of ASCII characters not containing whitespaces.

Atoms

Atoms are defined inside square brackets:

```
[ <mass> symbol <chiral> <hydrogen_count> <sign<charge>> ]
```

For example: **[OH]**, **[13C]**, **[NH3+]**, **[Fe+2]**. The atoms of the "organic subset" can be represented by their atomic symbols (**B, C, N, O, F, P, S, Cl, Br,** and **I.) without** brackets if the number of attached hydrogens conforms to the lowest normal valence consistent with explicit bonds. Hydrogen atoms are normally omitted or specified as hydrogen counts within square brackets. (For example, methane can be encoded as either **C**, or **[CH4]**, but the methyl radical can only be encoded as **[CH3]**). The second letter of an atomic symbol (if any) should always be lowercase, e.g., Br but not BR. The first letter must be uppercase, unless an atom is aromatic, in which case either uppercase, or lowercase symbols can be used. For example, **O** represents the oxygen in methanol, while **o** represents the oxygen in furan.

Bonds

Single bonds are represented by `**-**‘ (or nothing, this is the default bond order), double bonds are represented by `**=**’, triple bonds are represented by `**#**’, and aromatic bonds are represented by `**:**’. For example: **CC** or **C-C**, **C=O**, **C#N**, **cc** or **C:C**.

Branching

Branches are specified by enclosing them in parentheses, which may be nested or stacked. For example:

C(=O)O - carboxylic acid
S(=O)(=O)Cl - sulfonyl chloride
C(C(=O)O)C(=O)O - malonic acid

Rings

Ring closure bonds are specified by appending matching digits to the specifications of the joined atoms, with the bond symbol preceding the digit (if needed). For example:

C1CCCCC1 - cyclohexane
c1cc2ccccc2cc1 - naphthalene
C12C3C4C1C5C4C3C25 – cubane

Note that there can be more than one digit (ring closures) appended to an atom.

For more information

Text and Literature

See the *Daylight CIS Theory Manual* for authoritative information.
See also *"SMILES 1. Introduction and Encoding Rules"*, Weininger, D., *J. Chem. Inf. Comput. Sci.*, **1988**, 28, 31.

Web Site(s)

Refer to the following web sites for additional information about SMILES.

SMILES Specifications
(**http://www.daylight.com/dayhtml/doc/theory/theory.smiles.html)**

SMILES Tutorial
(**http://www.daylight.com/dayhtml/smiles/smiles-intro.html)**

Reference Five

SMARTS

Overview

This reference contains information about the SMARTS format. SMARTS is an extension of SMILES to express molecular structure patterns. It is designed for specifying substructures to facilitate finding a particular pattern in a molecule. SMARTS uses special atom and bond symbols as well as logical operators to denote more general patterns.

Specifically, this reference includes the following sections:

- *Atomic Primitives*
- *Bond Primitives*
- *Logical Operators*
- *Recursive SMARTS*
- *For More Information*

The following table identifies the appropriate section to read in order to learn more about a specific area.

To learn more about …	**Read the section….**	**See page**
Using Atomic Primitives	Atomic Primitives	28
Using Bond Primitives	Bond Primitives	30
Using Logical Operators	Logical Operators	31
Using Recursive SMARTS	Recursive SMARTS	32
Getting SMARTS information from web sites	For More Information	32

Atomic Primitives

SMARTS provides a number of primitive symbols describing atomic properties beyond those used in SMILES (atomic symbol, charge, and isotopic specifications). The following table lists the atomic primitives used in SMARTS (all SMILES atomic symbols are also legal). In this table <n> stands for a digit, and <c> for a chiral class.

Symbol	Symbol name	Atomic property requirements	Default
*****	wildcard	any atom	(no default)
a	aromatic	aromatic	(no default)
A	aliphatic	aliphatic	(no default)
D<n>	degree	<n> explicit connections	(no default)
H<n>	total-H-count	<n> attached hydrogens	exactly one
h<n>	implicit-H-count	<n> implicit hydrogens	exactly one
R<n>	ring membership	in <n> SSSR rings	any ring atom
r<n>	ring size	in smallest SSSR ring of size <n>	any ring atom
v<n>	valence	total bond order <n>	(no default)
X<n>	connectivity	<n> total connections	(no default)
-<n>	negative charge	-<n> charge	-1 charge (-- is -2, etc)
+<n>	positive charge	+<n> formal charge	+1 charge (++ is +2, etc)
#n	atomic number	atomic number <n>	(no default)
@	chirality	anticlockwise	anticlockwise, default class
@@	chirality	clockwise	clockwise, default class
@<c><n>	chirality	chiral class <c> chirality <n>	(no default)
@<c><n>?	chiral or unspec	chirality <c><n> or unspecified	(no default)
@U	chirality	unspecified chirality	(no default)
<n>	atomic mass	explicit atomic mass	unspecified mass

Examples:

C	aliphatic carbon atom
c	aromatic carbon atom
a	any aromatic atom
[#6]	carbon atom
[Ca]	calcium atom
[++]	atom with a +2 charge
[R]	atom in any ring
[D3]	atom with 3 explicit bonds (implicit H's don't count)
[X3]	atom with 3 total bonds (includes implicit H's)
[v3]	atom with bond orders totaling 3 (includes implicit H's)
C[C@H](F)O	match chirality (H-F-O anticlockwise viewed from C)
C[C@?H](F)O	matches if chirality is as specified or is not specified

Bond Primitives

Various bond symbols are available to match connections between atoms. A missing bond symbol is interpreted as "single or aromatic".

Symbol	**Atomic property requirements**
-	single bond (aliphatic)
/	directional single bond "up"
\	directional single bond "down"
/?	directional bond "up or unspecified"
\?	directional bond "down or unspecified"
=	double bond
#	triple bond
:	aromatic bond
~	any bond (wildcard)
@	any ring bond

Examples:

C	any aliphatic carbon
cc	any pair of attached aromatic carbons
c:c	aromatic carbons joined by an aromatic bond
c-c	aromatic carbons joined by a single bond (e.g. biphenyl)

Logical Operators

Atom and bond primitive specifications may be combined to form expressions by using logical operators. In the following table, "e" is a SMARTS expression (which may be a primitive). The logical operators are listed in order of decreasing precedence (high precedence operators are evaluated first).

Symbol	Expression	Meaning
exclamation	**!**e1	not e1
ampersand	e1**&** e2	a1 and e2 (high precedence)
comma	e1**,**e2	e1 or e2
semicolon	e1**;**e2	a1 and e2 (low precedence)

All atomic expressions that are not simple primitives must be enclosed in brackets. The default operation is `&' (high precedence "and"), i.e., two adjacent primitives without an intervening logical operator must both be true for the expression (or subexpression) to be true. The ability to form expressions gives the SMARTS user the power to specify exactly what is desired. Two forms of the AND operator are used in SMARTS instead of grouping operators. Examples:

[CH2]	aliphatic carbon with two hydrogens (methylene carbon)
[!C;R]	(NOT aliphatic carbon) AND in ring
[!C;!R0]	same as above ("!R0" means not in zero rings)
[n;H]	H-pyrrole nitrogen
[n&H]	same as above
[nH]	same as above
[c,n&H1]	any aromatic carbon OR H-pyrrole nitrogen
[c,n;H1]	(aromatic carbon OR aromatic nitrogen) and exactly one H
[Cl]	any chlorine atom
[35*]	any atom of mass 35
[35Cl]	chlorine atom of mass 35
[F,Cl,Br,I]	the 1st four halogens.

Recursive SMARTS

Any SMARTS expression may be used to define an atomic environment by writing a SMARTS starting with the atom of interest in this form:
$(SMARTS)
Such definitions may be considered atomic properties. These expressions can be used in the same manner as other atomic primitives (also, they can be nested). Examples:

*C	atom connected to methyl (or methylene) carbon
*CC	atom connected to ethyl carbon
[$(*C);$(*CC)]	atom in both above environments (matches CCC)

The additional power of such expressions is illustrated by the following example, which derives an expression for methyl carbons that are ortho to oxygen and meta to a nitrogen on an aromatic ring.

CaaO	C ortho to O
CaaaN	C meta to N
Caa(O)aN	C ortho to O and meta to N (but 2O,3N only)
Ca(aO)aaN	C ortho to O and meta to N (but 2O,5N only)
C[$(aaO);$(aaaN)]	C ortho to O and meta to N (all cases)

For More Information

Web Site(s)

Refer to the following web site(s) for additional information on SMARTS:
SMARTS Specifications
(http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html)

Reference Six

Perl

Overview

Perl is a high-level programming language written by Larry Wall and others. It derives from the C programming language and to a lesser degree from sed, awk, the Unix shell, and at least a dozen other sources. Perl's process, file, and text manipulation facilities make it particularly well suited for tasks involving quick prototyping, system utilities, software tools, system management tasks, database access, graphical programming, networking, and World Wide Web programming. Perl is distributed free and supported by its users. The DirectedDiversity® Setup program will install Perl 5.0 if desired.

Web Site

Additional information about Perl can be obtained at the following web sites:

```
http://language.perl.com
http://www.perl.com
http://www.perl.org
```

Download Perl for Windows from

`http://www.activestate.com/ActivePerl/` (recommended)

or another source, see

`http://www.perl.com/CPAN-local/ports/#win32`

Reference Seven

Publications

Overview

This reference lists publications relevant to the DirectedDiversity® Browser and command-line utilities algorithms.

List of Publications

1. D. K. Agrafiotis, “On the use of information theory for assessing molecular diversity”, *J. Chem. Info. Comp. Sci.*, 1997, *37(3)*, 576.
2. D. K. Agrafiotis, “Stochastic algorithms for maximizing molecular diversity”, *J. Chem. Info. Comp. Sci.*, 1997, *37(5)*, 841.
3. K. Agrafiotis, J. P. Myslik, and F. R. Salemme, “Advances in diversity profiling and combinatorial series design”, *Mol. Diversity*, 1999, *4*, 1-22.
4. K. Agrafiotis and V. S. Lobanov, “An efficient implementation of distance-based diversity metrics based on k-d trees”, *J. Chem. Info. Comp. Sci.*, 1999, *39(1)*, 51-58.
5. V. S. Lobanov and D. K. Agrafiotis, “Stochastic similarity selections from large combinatorial libraries”, *J. Chem. Info. Comput. Sci.*, 2000, *40*, 460-470.
6. D. K. Agrafiotis and V. S. Lobanov, “Ultrafast algorithm for designing focused combinatorial arrays”, *J. Chem. Info. Comput. Sci.*, 2000, *40*, 1030-1038.
7. D. N. Rassokhin and D. K. Agrafiotis*, “Kolmogorov-Smirnov statistic and its applications in library design”, *J. Mol. Graphics Modell.*, 2000, *18(4-5)*, 370-384.
8. D. K. Agrafiotis and V. S. Lobanov, “Nonlinear mapping networks”, *J. Chem. Info. Comput. Sci.*, 2000, *40*, 1356-1362.
9. D. K. Agrafiotis, “A constant time algorithm for estimating the diversity of large chemical libraries”, *J. Chem. Info. Comput. Sci.*, 2001, *41(1)*, 159-167.
10. S. Izrailev and D. K. Agrafiotis, “A new method for building regression tree models for QSAR based on artificial ant colony systems”, *J. Chem. Info. Comput. Sci.*, 2001, *41(1)*, 176-180.
11. D. N. Rassokhin, V. S. Lobanov, and D. K. Agrafiotis, “Nonlinear mapping of massive data sets by fuzzy clustering and neural networks”, *J. Comput. Chem.*, 2001, *22(4)*, 373-386.
12. D. K. Agrafiotis, D. N. Rassokhin, and V. S. Lobanov, “Multidimensional scaling and visualization of large molecular similarity tables”, *J. Comput. Chem.*, 2001, *22(5)*, 488-500.

13. D. K. Agrafiotis and D. N. Rassokhin, "Design and prioritization of plates for high-throughput screening", *J. Chem. Info. Comput. Sci.*, 2001, *41(3)*, 798-805.
14. D. K. Agrafiotis, “Multiobjective optimization of combinatorial libraries”, *IBM J. Res. Develop.*, 2001, *45(3/4)*, 545-566. (Special Issue on Deep Computing in the Life Sciences).
15. V. S. Lobanov and D. K. Agrafiotis, “Combinatorial networks”, *J. Mol. Graphics Modell.*, 2001, *19(6)*, 571-578.
16. D. K. Agrafiotis and V. S. Lobanov, "Multidimensional scaling of combinatorial libraries without explicit enumeration", *J. Comput. Chem.*, 2001, *22(14)*, 1712-1722.
17. S. Izrailev and D. K. Agrafiotis, "Variable selection for QSAR by artificial ant colony systems", *SAR and QSAR in Environ. Res.*, 2002, *13*, 417-423.
18. D. K. Agrafiotis and D. N. Rassokhin, "A fractal approach for selecting an appropriate bin size for cell-based diversity estimation", *J. Chem. Info. Comput. Sci.*, 2002, *42*, 117-122.
19. V. S. Lobanov and D. K. Agrafiotis, “Scalable methods for the construction and analysis of virtual combinatorial libraries”, *Combin. Chem. and High-Throughput Screen.*, 2002, *5*, 167-178.
20. D. K. Agrafiotis and W. Cedeño, "Feature selection for structure-activity correlation using binary particle swarms", *J. Med. Chem.*, 2002, *45*, 1098-1107.
21. D. K. Agrafiotis, V. S. Lobanov, and F. R. Salemme, "Combinatorial informatics in the post-genomics era, *Nature Rev. Drug Discov.*, 2002, *1*, 337-346.
22. D. K. Agrafiotis, W. Cedeño, and V. S. Lobanov, "On the use of neural network ensembles in QSAR and QSPR", *J. Chem. Info. Comput. Sci.*, 2002, *42*, 903-911.
23. H. Xu and D. K. Agrafiotis, "Retrospect and prospect of virtual screening in drug lead discovery", *Curr. Topics Med. Chem.*, 2002, *2*, 1305-1320.
24. D. K. Agrafiotis, “Multiobjective optimization of combinatorial libraries”, *J. Comput. Aid. Mol. Des.*, 2002, *16*, 335-356.
25. W. Cedeño and D. K. Agrafiotis, "Combining particle swarms and k-nearest neighbors for the development of quantitative structure-activity relationships", *Int. J. Comput. Res.*, 2002 (in press).
26. W. Cedeño and D. K. Agrafiotis, "Application of niching particle swarms to QSAR and QSPR", *Proceedings of the 14-th European Symposium on QSAR*, Bournemouth, UK, Sep. 8-13, 2002.
27. D. K. Agrafiotis and H. Xu, "A self-organizing principle for learning nonlinear manifolds", *Proc. Natl. Acad. Sci. USA*, 2002, *99*, 15869-15872.
28. D. K. Agrafiotis, "Stochastic proximity embedding", *J. Comput. Chem.*, 2003, (in press).
29. D. K. Agrafiotis and H. Xu, "A geodesic framework for analyzing molecular similarities", *J. Chem. Info. Comput. Sci.*, 2003 (in press).
30. W. Cedeño and D. K. Agrafiotis, "Using particle swarms for the development of QSAR models based on k-nearest neighbor and kernel regression", *J. Comput. Aid. Mol. Des.*, 2003 (in press).
31. S. Izrailev and D. K. Agrafiotis, "A method for quantifying and visualizing the diversity of QSAR models", submitted
32. D. N. Rassokhin and D. K. Agrafiotis, "A modified update rule for stochastic proximity embedding", submitted.