



OpenEye
Scientific Software

MolProp TK – C++
Release 2.1.2

OpenEye Scientific Software, Inc.

January 11, 2012

CONTENTS

1	Front Matter	1
2	Filtering Theory	3
2.1	Filtering Theory	3
2.2	Filter Preprocessing	7
2.3	Molecular Properties and Predictors	8
2.4	Filter Files	12
2.5	Bibliography	68
3	Toolkit	69
3.1	Filter Object	69
4	API	75
4.1	OEMolProp Classes	75
4.2	OEMolProp Constants	79
4.3	OEMolProp Functions	80
5	Release Notes	85
5.1	MolPropTK 2.1.2	85
5.2	MolPropTK 2.1.1	85
5.3	MolPropTK 2.1.0	85
6	Indices and tables	87
	Bibliography	89
	Index	91

FRONT MATTER

Copyright 1997-2012 OpenEye Scientific Software, Santa Fe, New Mexico. All rights reserved.

All rights reserved. This material contains proprietary information of OpenEye Scientific Software. Use of copyright notice is precautionary only and does not imply publication or disclosure.

The information supplied in this document is believed to be true but no liability is assumed for its use or the infringement of the rights of others resulting from its use. Information in this document is subject to change without notice and does not represent a commitment on the part of OpenEye Scientific Software.

This package is sold/licensed/distributed subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out or otherwise circulated without OpenEye Scientific Software's prior consent, in any form of packaging or cover other than that in which it was produced. No part of this manual or accompanying documentation, may be reproduced, stored in a retrieval system on optical or magnetic disk, tape, CD, DVD or other medium, or transmitted in any form or by any means, electronic, mechanical, photocopying recording or otherwise for any purpose other than for the purchaser's personal use without a legal agreement or other written permission granted by OpenEye.

This product should not be used in the planning, construction, maintenance, operation or use of any nuclear facility nor the flight, navigation or communication of aircraft or ground support equipment. OpenEye Scientific Software, shall not be liable, in whole or in part, for any claims arising from such use, including death, bankruptcy or outbreak of war.

Windows is a registered trademark of Microsoft Corporation. Apple, OS X, and Macintosh are registered trademarks of Apple Computer, Inc. AIX and IBM are registered trademarks of International Business Machines Corporation. UNIX is a registered trademark of the Open Group. RedHat is a registered trademark of RedHat, Inc. Linux is a registered trademark of Linus Torvalds. SPARC is a registered trademark of SPARC International Inc.

SYBYL is a registered trademark of TRIPOS, Inc. MDL is a registered trademark and ISIS is a trademark of Accelrys, Inc. SMILES, SMARTS, and SMIRKS may be trademarks of Daylight Chemical Information Systems. Macromodel is a trademark of Schrodinger, Inc.

Python is a trademark of the Python Software Foundation. Java is a trademark or registered trademark of Sun Microsystems, Inc. in the U.S. and other countries.

Other products and software packages referenced in this document are trademarks and registered trademarks of their respective vendors or manufacturers.

FILTERING THEORY

2.1 Filtering Theory

2.1.1 Introduction

Filtering attempts to eliminate inappropriate or undesirable compounds from a large set before beginning to use them in modelling studies. The goal is to remove all of the compounds that should not be suggested to a medicinal chemist as a potential hit. This exercise is obviously case dependent, depending on ease of the assay, intended target, personal bias of the modeller & medicinal chemist, strengths of the company, etc.

To match this need, FILTER's default filter encapsulates many of the standard filtering principles, such as removal of unstable, reactive, and toxic moieties. In addition, MolPropTK allows the customization of the filtering criteria to fit specific needs.

The criteria for passing or failing a given molecule fall into three categories.

- Physical properties
 - Molecular weight
 - Topological polar surface area (TPSA)
 - logP
 - Bioavailability
- Atomic and functional group content
 - Absolute and relative content of heteroatoms
 - Limits on a very wide variety of functional groups
- Molecular graph topology
 - Number and size of ring systems
 - Flexibility of the molecule
 - Size and shape of non-ring chains

All of the data MolPropTK generates in filtering molecules can be written to a tab-separated file for easy import into a spreadsheet. This function allows for combining the values dynamically for a variety of purposes, including, but not limited to, determining which filter values best fit each project's needs.

History

When OpenEye's work on filtering technology began in 2000, it was designed simply to remove compounds with reactive or otherwise undesirable functional groups. Over the years, the understanding of lead-like and drug-like compound selection has advanced. In addition, with the publication of Lipinski's "Rule of 5" [Lipinski-1997], more and more pharmacokinetic properties have been pushed earlier into the virtual screening process.

In addition to providing basic functional group selection, the technology is a one-stop database preparation tool aimed at generating databases suitable for high-throughput virtual screening.

- Cheminformatics quality-control
 - Valence-state validation
 - Aromaticity perception
 - Implicit hydrogen perception
 - Bond-order perception
- Database preparation
 - Setting pKa states
 - Applying normalizations (tautomers & dative or hypervalent states)
- Compound selection
 - Physical properties (see *above*)
 - Assay counter-indicators (aggregators and dyes)
 - PK ([Martin-2005], [Veber-2002], [Egan-2000], [Lipinski-1997])

Finally, it should be pointed out that in the virtual screening world, time is of the essence. Algorithms for preliminary database preparation should not take large amounts of time. Because of this, all the calculations included in MolPropTK are 2D or graph-based algorithms. While this does occasionally limit the technology, it allows for the delivery of a product that is appropriate for the task of virtual-screening database preparation.

The Rant!

Nearly every computational tool used in early drug discovery yields statistically predictive, rather than absolutely definitive results. In nearly every case, prudence demands that one consider the causes of false-positives and false-negatives and make an attempt to optimize the area under the receiver-operator curve (ROC) for the computational tool. However, there are well known methods for improving statistical predictions of this nature that are independent of the absolute false-positive and false-negative rates. These methods include filtering the population to which a test will be applied. By applying a test to smaller populations that only contain molecules appropriate for the specific application at hand, the negative impact of the false-positive rate on the predictive results can be dramatically improved.

A familiar example from the medical world will serve to illustrate this principle. Assume we have a test for the presence of the new *foo virus* which has an exceptional ROC curve with false-positive and false-negative values (1/1,000 and 1/1,000 respectively). Let us assume that the *foo-syndrome*, caused by the *foo virus*, effects 1 person in 20,000. If we gave this test to 100,000 people from the general population, we would expect 5 to actually have the *foo syndrome*. With this test, there is only a 0.05% percent chance that any of them would not be detected (i.e. be a false-negative). However, we would expect there to be 100 false positive test results. Thus of the 105 total positive test results, only 4.8% would actually have the *foo syndrome* (positive predictive value = 4.8%).

Table 2.1: Confusion table for the unfiltered foo virus test (prevalence 1 in 20,000)

	Actual Positive	Actual Negative	
Predicted Positive	True Positives = 5	False Positives = 100	Positive Predictive Value = 4.8%
Predicted Negative	False Negatives = 0	True Negatives = 99,895	

Alternatively, we could start by using very simple screening before applying the test. We first eliminate people who do not have any risk factors for contracting the *foo virus*. Next we may eliminate people whose blood is incompatible with the test for the *foo virus*. Further, we may want to eliminate people who acknowledge that they will refuse treatment for the *foo virus* even if we determine that they do have it. By these admittedly simple screens, we apply the test for the *foo virus* to a much smaller group with a decidedly higher prevalence of the virus. For instance, after the filtering, we may be left with a group of only 1,000 people who have a 1 in 200 chance of having the syndrome. Now, we still have the same 5 people who actually have the disease, but we only expect 1 false positive test. Suddenly, there are 6 total positive tests, and 83% of them actually have the syndrome! This is reflected in a much more reasonable (83%) positive predictive value.

Table 2.2: Confusion table for the filtered foo virus test (prevalence 1 in 200)

	Actual Positive	Actual Negative	
Predicted Positive	True Positives = 5	False Positives = 1	Positive Predictive Value = 83%
Predicted Negative	False Negatives = 0	True Negatives = 994	

Bringing the discussion back to drug design. If we have a ligand-based design tool such as ROCS, we can imagine that the receiver-operator curve may have a false positive rate as low as 1 in 10,000. For this exercise, let's assume no false negatives. When using that to identify 50 inhibitors from a database of 2.5 million available compounds, we'd identify 300 potential inhibitors, and 5 out of every 6 of these would be a false positive (positive predictive value of 17%)! If we first run filter and eliminate 65% of the 2.5 million compounds, this leaves us with 875,000 compounds to push through ROCS. There will be about 88 false positives to go with the 50 true positives and the positive predictive value will increase over two-fold with relatively little work.

Table 2.3: Confusion table for the unfiltered ROCS virtual screen

	Actual Positive	Actual Negative	
Predicted Positive	True Positives = 50	False Positives = 250	Positive Predictive Value = 17%
Predicted Negative	False Negatives = 0	True Negatives = 2,499,700	

Table 2.4: Confusion table for the filtered ROCS virtual screen

	Actual Positive	Actual Negative	
Predicted Positive	True Positives = 50	False Positives = 88	Positive Predictive Value = 36%
Predicted Negative	False Negatives = 0	True Negatives = 874,862	

Filtering Principles

The same principle of increasing positive predictive value by removing obvious true negatives applies to screening for lead candidates, regardless of whether it is virtual screening or high-throughput screening. While both are reasonable screens, each can be plagued by very low positive-predictive values (despite low false-positive rates), particularly when applied to all available compounds, or large virtual libraries. Simple filtering techniques focus the set of compounds passed on to more computationally intensive screening methods.

The first approach to consider is filtering based on *functional groups*. Generally speaking, there are toxic and reactive functional groups that you simply do not want to consider (alkyl-bromides, metals etc). There are also functional groups that are not strictly forbidden, but are not desired in large quantities. For instance, parafluoro-benzene, or trifluoromethyl have specific purposes, but heavily fluorinated molecules can be eliminated.

Beyond simple functional group filtering, you can consider both simple and complex physical properties which can be used to characterize the kinds of compounds you would like to keep and those you would like to eliminate. These

properties attempt to consider “drug-likeness”, such as bio-availability, solubility, toxicity, and synthetic accessibility even before the primary high-throughput or virtual screening, which primarily are geared toward detecting potency alone. The best known of the physical property filters is *Lipinski’s “rule-of-five”*, which focuses on bioavailability [Lipinski-1997]. However, many other physical properties, such as *solubility*, *atomic content*, *ring structures*, and *surface area ratios* can also be considered. MolPropTK provides algorithms for calculating many of these properties, and applying them with filters based on literature studies.

Finally, you should eliminate the types of compounds that can be troublesome at later stages. For instance, Shoichet’s *aggregating compounds* often produce false positives that can waste enormous resources if they were identified by virtual or high-throughput screening [McGovern-2003] [Seidler-2003]. Similarly, *dyes* can appear to be inhibitors by interfering with colorimetric or fluorometric assays or binding non-specifically to the target protein.

Variations of Filters

Different types of filters are appropriate under different circumstances. Very early in a project, when little or no SAR is available, very strict drug-like filters can be applied. This prevents a project team from spending chemistry resources pursuing difficult compounds that may not be modifiable to introduce appropriate properties. However, when considering compounds for purchase for HTS, different filters can be applied. Oprea, et al, pointed out that the best molecules for initial HTS are smaller and less functionalised than drugs, but with some activity [Oprea-2000]. Therefore, strict lead-like filters can be applied to ensure that hits identified from HTS have sufficient “room” for elaboration into (usually larger and more highly functionalised) leads. However, when SAR suggests that particular compounds or series may yield valuable information, filtering criteria can be loosened, because the secondary screens (QSAR models, similarity to known actives) that are being applied are effective in detecting useful compounds. Reflecting back on the medical analogy, this is the case where an improved primary screen with a dramatically improved false-positive rate (say 1 in 100,000) can be safely applied to a larger population without terrible effects on the positive-predictive value.

MolPropTK provides the following “light” (lenient) and “heavy” (restrictive) filters:

- BlockBuster
- Lead

The BlockBuster filter is based on 141 best-selling, non-antibiotic, prescription drugs. We designed the physical property portion of the filter so that it passes all of the compounds. The physical property values in this filter are quite good. However, the functional group filters in this filter are probably too restrictive because 141 compounds is not sufficient to span all acceptable functionality.

Note: The original [Oprea-2000] Drug filter is provided as well. However, experience has shown it to be too restrictive. The BlockBuster filter was developed in response to complaints about the Drug filter being too restrictive.

Hint: We recommend the BlockBuster filter, the default, for most purposes. If your project is unusual, or you are unsatisfied with the results we recommend you review the *filter file* for your specific filtering needs.

If you decide to modify the *filter file* the depictions found in the *Functional Group Rules* section can be particularly helpful in determining what functional groups are indicated by each name in the file.

Accumulation of Rules

FILTER contains numerous rules that judge the quality of molecules on many different facets. When examined individually, each of these rules seems quite reasonable and even profitable. However, when each molecule is tested against hundreds of individual filters, the fraction of molecules that pass all the filters can be surprisingly small. Sometimes less than 50% of vendor databases pass the filters. If this is unacceptable we recommend you examine the predicted aggregator, solubility, and Veber filters. In our experience, these are the most common failures. The best method of investigating failures is looking at the filter log.

The BlockBuster filter was adjusted to demonstrate this in a tangible way. For each value, rather than spanning the entire range, its properties were set to cover from the 2.5th percentile to the 97.5th percentile. The differences between the original BlockBuster filter and the adjusted filter are both in reasonable ranges. For instance, the full range of molecular weight for the BlockBuster filter spans 130 to 781, while the 2.5th percentile is 145 and the 97.5th percentile is 570. The remarkable result is that when the reduced filter is used, only 75 of the 141 original molecules pass the filter! This demonstrates how slight changes to many filters can lead to a significant reduction in the number of compounds that pass all of the filters.

Taking the opposite approach of allowing everything to pass can be equally futile. To demonstrate this a filter was designed around the “small molecule drug” file available from the DrugBank [website](#). In order to pass every one of these molecules, including some that would not be acceptable for modern project work, many of the individual filters must be set to unreasonable values. For instance, the molecular weight range is 30 to 1500 and the hetero-atom count range is 1 to 60!

Hint: OpenEye can not magically divine the needs of every project. You should personally inspect the *filter file*. The depictions found in the *Functional Group Rules* section can be particularly helpful in discerning what functional groups are desirable and undesirable.

2.2 Filter Preprocessing

Before the applying any of the *molecular property filters* a preprocessing step occurs that can alter the molecule significantly to fit the criteria needed for most modeling applications. This filtering preprocessing step is a precisely defined series of stages that occur on the molecule in the following order:

1. *Metal Removal*
2. *Salt Removal*
3. *pKa Normalization*
4. *Normalization*
5. *Reagent Selection*
6. *Type Checking*

2.2.1 Metal Removal

Metal removal is the first stage of elemental based filtering. This stage will remove specified metal complexes from the molecule. It will not reject a molecule for having a metal complex. This allows the filter to treat atoms in the counter-ion portion of a molecule separately from the atoms in the primary portion of the molecular record.

For instance, this allows organic molecules that are complexed with silver to be eliminated based on their metal chelate even though they themselves are acceptable while at the same time eliminating a sulphate counter-ion from another molecule before it leads to elimination of the acceptable cationic molecule.

See Also:

Elemental Filters

2.2.2 Salt Removal

This step deletes all atoms that are not part of the largest connected component of a compound. This effectively eliminates all non-covalently bound portions of the compound.

2.2.3 pKa Normalization

MolPropTK has a rule-based system to set the ionization state of input molecules. If pKa normalization is turned on, the molecule is set to its most energetically favorable ionization state for $pH=7.4$. The rule-based nature of this calculation allows it to be very fast. Further, despite being rule-based, this approach takes into account many secondary charge interactions.

While more advanced levels of theory can be found for predicting ionization states, this method is very well suited to virtual-screening database preparation. However, MolPropTK may not be appropriate for hit-to-lead or lead optimization.

See Also:

2.2.4 Normalization

In addition to *pKa normalization*, MolPropTK allows any number of additional molecular normalizations. Since normalizations are usually specific to a particular company or site, MolPropTK provides the ability for users to input normalizations, such as the nitro tautomer state, but does not provide default implementations.

See Also:

2.2.5 Reagent Selection

Reagent selection for small linear library synthesis or large combinatorial library synthesis is still a necessary task at many pharmaceutical companies. For a user hoping to identify a set of acyl-halide reagents, they can specify a selection parameter to require that each compound have exactly one acyl-halide. In addition they might want to modify the filter to exclude functional groups (such as primary amines) that may be acceptable for typical lead-like molecules, but are not acceptable for the specific reagent the user has in mind.

Therefore, the selection parameter is the reverse of a filtering parameter. The molecule must *include* the given substructure in order to pass the filter.

See Also:

The *select parameter* in filter files.

2.2.6 Type Checking

This checks the valence state and formal charge of the entire molecule. The check identifies molecules that are poorly specified, or represent nonsensical chemical states, often from corrupt input data. For example, an oxygen with eight hydrogens attached or a carbon with a +9 formal charge would be rejected.

See Also:

2.3 Molecular Properties and Predictors

MolPropTK provides a range of properties to predictors to be used as molecular filters. Molecular properties are distinct physical properties that can be measured such as the following:

- *Structural and Chemical Features*
- *Functional Groups*

There have been several attempts at developing fast-approximate QSAR models for bioavailability that have been published. The first of these was *Lipinski's work* ([Lipinski-1997]) and it has been followed by work at Pharmacopia [Egan-2000], Abbott [Martin-2005], and GSK [Veber-2002]. The simplest and probably most trusted is the work of *LogP* and *PSA* used by Egan. The most recent work by Martin, the Abbott Bioavailability Score (ABS) appears to be a refinement of the first generation models and is designed specifically to categorize a molecule's probability of having a bioavailability >10% in rats.

See Also:

The *Pharmacokinetic Predictors* section in the Filter Files chapter.

2.3.1 Structural and Chemical Features

There are a number of important structural and chemical features of molecules that it is desirable to limit for virtual screening. The following simple measures are provided as filterable properties:

- Molecular weight
- Ring count
- Ring-system size
- Size of non-ring structures
- Length of unbranched chains
- Hetero-atom fraction
- Halide fraction
- Formal charges
- Rotatable bonds

It also includes slightly more complex algorithms for hydrogen-bond donors and acceptors as well as chiral centers.

See Also:

The *Basic Properties* section in the Filter Files chapter.

2.3.2 Functional Groups

Functional group removal remains at the heart of the filter algorithm. Functional groups fall into several categories including:

- Reactive or labile groups
- Undesirable groups
- Generally acceptable groups
- Protecting groups
- User-derived groups

While reasonable defaults are provided for each functional group in all of the above categories, it is unusual to find an experienced computational or medicinal chemist who agrees with all of the default values. Examining the filter file at least once is strongly encouraged. Developing a custom filter to fit a desired design is also strongly encouraged. The filter files were designed as basic guides.

See Also:

The filter files contain only functional group names. If there is any confusion regarding the definitions, please refer to the *Functional Group Rules* section for complete descriptions.

Dyes

Years ago, many high-throughput assays could be interfered with by colored molecules. While assay technology has continued to advance and often this is not a problem, it is recognized that most molecules that are dyes are not the type of molecules that are commonly carried forward in lead-development projects. Therefore, a pattern-based filter for dye molecules is included. While these patterns occasionally identify molecules that are considered acceptable by some users, in general, they identify molecules that the majority of chemists would rather not see at the top of their virtual-screening hitlists.

2.3.3 LogP

The XLogP algorithm ([Wang-1997]) is provided because its atom-type contribution allows calculation of the XLogP contribution of any fragment in a molecule and allows minimal corrections in a simple additive form to calculate the LogP of any molecule made from combinations of fragments. Further, although the method contains many many free parameters, its simple linear form allows for ready interpretation of the model and most of the parameters in the model make rational sense.

Unfortunately, the original algorithm is difficult to implement as published. First, the internal-hydrogen bond term was calculated using a single 3D conformation. It was found that this was both arbitrary and unnecessary. This arbitrary 3D calculation has been replaced with a 2D approach to recognize common internal-hydrogen bonds. In tests, this 2D method worked comparably to the published 3D algorithm. Next, the training set had a few subtle atom-type inconsistencies.

Both of these problems were corrected and refit to the original XLogP training data. This implementation gives results that are quite similar to the original XLogP algorithm, so it is called OEXLogP to distinguish it from the original method.

See Also:

- The *XLogP parameter* in the Filter Files chapter.

2.3.4 LogS

The work of [Yalkowsky-1980] at Arizona has resulted in what is now called the “generalized solvation equation.” It states that the solubility of a compound can be broken into two steps, first the melting for the pure solid to pure liquid and second, the phase transfer from pure liquid into water. For many small organic molecules, this second step is somewhat related to LogP.

Because of this relation we choose to explore the use of the XLogP atom-types in solubility prediction. The expectation was that this might provide an approximate though robust and fast method for calculating solubility. We fit the XLogP atom-types to a training set of nearly 1000 public solubilities. From this we derived a linear model for solubility. The model is extremely fast and is useful for classifying compounds as insoluble, poorly soluble, slightly soluble, moderately soluble or very soluble. The model is notable for the difficulty it has predicting solubilities for compounds with ionizable groups. Further, it is not suitable for the PK predictions that come late in a project. However, it is useful for eliminating compounds with severe solubility problems early in the virtual-screening process.

See Also:

The *solubility parameter* in the Filter Files chapter.

2.3.5 Polar Surface Area

Topological polar-surface area (TPSA) is based on the algorithm developed by Ertl *et al* [Ertl-2000]. In Ertl's publication, use of TPSA both with and without accounting for phosphorus and sulfur surface-area is reported. However, evidence shows that in most PK applications one is better off not counting the contributions of phosphorus and sulfur atoms toward the total TPSA for a molecule. This implementation of TPSA allows either inclusion or exclusion of phosphorus and sulfur surface area with the default being to not include it.

See Also:

Including phosphorus and sulfur surface area:

- The `PSA_USE_SandP` parameter in the *filter file*

Warning: TPSA values are mildly sensitive to the protonation state of a molecule. If the `pKaNorm` parameter is false, the TPSA value is calculated using the input structure and if `pKaNorm` is true, the TPSA is calculated using the pKa normalized molecular structure.

See Also:

The *pKa normalization* section of the Filter Preprocessing chapter.

2.3.6 Lipinski and Hydrogen-bonds

The work of Lipinski ([Lipinski-1997]) introduced the application of simple filter-like rules to roughly predict late-stage PK properties, in particular oral bioavailability. Unfortunately, Lipinski's "Rule-of-Five" has come into the common vernacular to such a large degree that some of the specific details are often lost in the commotion. There are two critical examples of this. First, Lipinski used "violation of 2 rules" to categorize compounds. In the subsequent analysis, a significant difference in the two populations of molecules was detected, however, little analysis of the importance of a single violation was done. However, many now consider a single violation to be bad and two violations to be worse. At this writing, there is no known evidence to support this. Second, Lipinski used well-codified and well-understood yet imprecise definitions of "hydrogen-bond donors" and "hydrogen-bond acceptors" in his classification model. While this makes the algorithm quite understandable and easy to implement, it sometimes causes confusion with those who prefer more refined definitions of hydrogen-bond donors and acceptors.

To address the first problem, users are allowed to set the number of Lipinski failures required to reject a molecule. In keeping with the original publication, the default value is 2. To address the second problem, two kinds of hydrogen-bond donors and acceptors are calculated. In the Lipinski calculation, the published definitions are used (donor count is the number nitrogen or oxygen atoms with at least one hydrogen attached and acceptors are the number of nitrogens and oxygens). For calculation of the number of donors and acceptors in the molecule for the sake of chemical properties, a more complex algorithmic approach is taken. This approach identifies the donors and acceptors outlined in the work of Mills and Dean ([MillsDean-1996]) and also in the book by Jeffrey ([Jeffrey-1997]).

See Also:

The *Lipinski violations* and *hydrogen-bond acceptors* sections in the Filter Files chapter.

2.3.7 Aggregators

The Shoichet lab ([McGovern-2003], [Seidler-2003]) has demonstrated the importance of small-molecule aggregation in medium and high-throughput assays. Because these small-molecule aggregates can sequester some proteins, they give the appearance of being active inhibitors. There are now several hundred published aggregators in addition to a published QSAR model for predicting aggregation propensity. The user has the ability to eliminate any of the known aggregators as well as the ability to eliminate compounds that are predicted to be aggregators using the QSAR model. In OpenEye's experience, the published QSAR model for predicting aggregators is quite aggressive. It occasionally identifies compounds that are known to be genuine small-molecule inhibitors of a specific protein. While in most cases

this model can be useful, until more definitive work is published, we feel you should gain some experience with the model and judge its performance for yourself.

More recent work in industry indicates that in some cases, aggregation properties are specific to the particular experimental conditions being used. Thus we recommended caution in the interpretation of these predictions. Nevertheless, aggregation remains an important issue in HTS hit follow-up and, short of experimental validation, this flag may be the best-available.

See Also:

The *aggregators parameter* in the Filter Files chapter.

2.4 Filter Files

There are two parameter files a user can provide if they would like to override or augment the default parameter sets. We recommend that you do this by modifying the provided default (*filter_example* and *newrule_example*) files rather than starting from scratch.

There are two primary files a user may choose to provide. The first is the “filter file”. It provides acceptable limits for all of the physical properties and functional groups in the default filter. The second is the “newrule file”. If you have a filter you like, but would like to augment it with a set of additional rules, these can be added with a newrule file.

There are four types of statements that can occur in a filter file:

- physical property limits
- rules
- new rules
- selections

The statements should occur one-per-line in the filter file.

Note: If the appropriate line is not in the filter file, or the value is false, the respective measure will not be used in filtering and its value will not be included in any table-based output.

2.4.1 Physical Property Limits

There are a large number of physical property limits. They occur as three fields on a line. For example:

```
MIN_HETEROATOMS 2 "Minimum number of heteroatoms"
```

The first field is the property keyword, the second field is the value assigned to that keyword, and the third field is a brief informational message. There are a fixed number of physical property keywords. No additional physical property keywords can be added by the user. The current keywords and brief definitions of each are listed below.

Hint: The values listed below are those found in the default *BlockBuster* filter.

Basic Properties

Molecular Weight

Isotopic molecular weight

```
MIN_MOLWT 130 "Minimum molecular weight"  
MAX_MOLWT 781 "Maximum molecular weight"
```

Heavy Atom Count

Number of non-hydrogen atoms

```
MIN_NUM_HVY 9 "Minimum number of heavy atoms"  
MAX_NUM_HVY 55 "Maximum number of heavy atoms"
```

Carbon Count

Number of carbons

```
MIN_CARBONS 3 "Minimum number of carbons"  
MAX_CARBONS 41 "Maximum number of carbons"
```

Hetero-Count

Number of non-carbon and non-hydrogen atoms

```
MIN_HETEROATOMS 1 "Minimum number of heteroatoms"  
MAX_HETEROATOMS 14 "Maximum number of heteroatoms"
```

Hetero-Atom to Carbon Ratio

Hetero-count/carbon-count

```
MIN_Het_C_Ratio 0.04 "Minimum heteroatom to carbon ratio"  
MAX_Het_C_Ratio 4.0 "Maximum heteroatom to carbon ratio"
```

Chiral Count

Number of chiral atoms

```
MIN_CHIRAL_CENTERS 0 "Minimum chiral centers"  
MAX_CHIRAL_CENTERS 21 "Maximum chiral centers"
```

Hydrogen-bond Acceptors

Number of atoms which match any of the following:

- degree 2, aromatic, non-positive nitrogens
- electron rich or negative, valence less than 4, non-aromatic nitrogens

- negatively charged or not electron withdrawn and neutral oxygens
- degree 1, double bonded, electron rich sulfur

```
MIN_HBOND_ACCEPTORS 0 "Minimum number of hydrogen-bond acceptors"  
MAX_HBOND_ACCEPTORS 13 "Maximum number of hydrogen-bond acceptors"
```

Hydrogen-bond Donors

Number of hydrogen atoms on nitrogen, oxygen, or sulfur atoms

```
MIN_HBOND_DONORS 0 "Minimum number of hydrogen-bond donors"  
MAX_HBOND_DONORS 9 "Maximum number of hydrogen-bond donors"
```

Lipinski Acceptors

Number of nitrogens or oxygens

```
MIN_LIPINSKI_ACCEPTORS 1 "Minimum number of oxygen & nitrogen atoms"  
MAX_LIPINSKI_ACCEPTORS 14 "Maximum number of oxygen & nitrogen atoms"
```

Lipinski Donors

Number of hydrogens attached to nitrogen or oxygen

```
MIN_LIPINSKI_DONORS 0 "Minimum number of hydrogens on O & N atoms"  
MAX_LIPINSKI_DONORS 6 "Maximum number of hydrogens on O & N atoms"
```

Halide Fraction

Percent of molecular weight from halides

```
MIN_HALIDE_FRACTION 0.0 "Minimum Halide Fraction"  
MAX_HALIDE_FRACTION 0.66 "Maximum Halide Fraction"
```

Formal Count

Number of atoms with a formal charge (excludes dative)

```
MIN_COUNT_FORMAL_CRG 0 "Minimum number formal charges"  
MAX_COUNT_FORMAL_CRG 4 "Maximum number of formal charges"
```

Formal Sum

Total formal charge

```
MIN_SUM_FORMAL_CRG -2 "Minimum sum of formal charges"  
MAX_SUM_FORMAL_CRG 2 "Maximum sum of formal charges"
```

Connected Non-Ring

Considers sets of contiguous (bonded) non-ring atoms

```
MIN_CON_NON_RING 0 "Minimum number of connected non-ring atoms"  
MAX_CON_NON_RING 19 "Maximum number of connected non-ring atoms"
```

Unbranched Chains

The size of unbranched non-ring chains

```
MIN_UNBRANCHED 1 "Minimum number of connected unbranched non-ring atoms"  
MAX_UNBRANCHED 13 "Maximum number of connected unbranched non-ring atoms"
```

Total Functional Group Count

Total number of functional groups. Does not count any ring-systems as functional groups. Degree 1 heteroatoms, particularly those with double bonds or dative bonds are considered part of ring systems and do not count as a functional group.

```
MIN_FCNGRP 0 "Minimum number of functional groups"  
MAX_FCNGRP 7 "Maximum number of functional groups"
```

Note: This is different than the functional group rules.

Ring Systems

Number of ring systems (contiguous systems of ring atoms and bonds)

```
MIN_RING_SYS 0 "Minimum number of ring systems"  
MAX_RING_SYS 5 "Maximum number of ring systems"
```

Ring Size

Maximum size of any single ring system

```
MIN_RING_SIZE 0 "Minimum atoms in any ring system"  
MAX_RING_SIZE 20 "Maximum atoms in any ring system"
```

Rotor Count

Number of rotatable bonds. Allows optional adjustment for aliphatic rings following the method of [Oprea-2000].

```
MIN_ROT_BONDS 0 "Minimum number of rotatable bonds"  
MAX_ROT_BONDS 16 "Maximum number of rotatable bonds"  
ADJUST_ROT_FOR_RING true "BOOLEAN for whether to estimate degrees of freedom in rings"
```

Rigid Count

Number of rigid bonds (non-rotatable bonds)

```
MIN_RIGID_BONDS 4 "Minimum number of rigid bonds"  
MAX_RIGID_BONDS 55 "Maximum number of rigid bonds"
```

LogP

The logP calculation is a derivative of the published XLogP algorithm [Wang-1997]., but reparameterized without the dependence on 3D coordinates or the SYBYL/Mol2 aromaticity model.

XLogP

Calculated LogP

```
MIN_XLOGP -3.0 "Minimum XLogP"  
MAX_XLOGP 6.85 "Maximum XLogP"
```

Solubility

The solubility predictions are based on using the atom-types from the XLogP algorithm, [Wang-1997], and reparameterizing them based on available solubility data. Rather than a quantitative cutoff, solubility uses categories. The 6 allowable categories are:

1. insoluble
2. poorly
3. moderately
4. soluble
5. very
6. highly

These categories are keywords used in the filter files as follows.

Solubility

Calculated solubility class

```
MIN_SOLUBILITY insoluble "Minimum solubility"
```

Pharmacokinetic Predictors

Several secondary filters that are built upon published combinations of simpler properties are available.

Note: All of these properties are used for filtering in the default filters.

Lipinski Violations

Number of allowable Lipinski violations. A single Lipinski violation is considered acceptable. The published work, [Lipinski-1997], allows compounds to pass with a single violation but not multiple violations.

```
MAX_LIPINSKI 3 "Maximum number of Lipinski violations"
```

See Also:

The *Lipinski theory* section in the Molecular Properties and Predictors chapter.

PSA

Peter Ertl's, [Ertl-2000], topological polar surface area (phosphorus and sulfur area is optional).

```
PSA_USE_SandP false "Count S and P as polar atoms"
MIN_2D_PSA 0.0 "Minimum 2-Dimensional (SMILES) Polar Surface Area"
MAX_2D_PSA 205.0 "Maximum 2-Dimensional (SMILES) Polar Surface Area"
```

See Also:

The *PSA theory* section in the Molecular Properties and Predictors chapter.

GSK/Veber

Veber's measure of bioavailability (PSA > 140 or Rotatable bonds >10). [Veber-2002].

```
GSK_VEBER false "PSA>140 or >10 rot bonds"
```

Abbott/Martin

Yvonne Martin's Abbott Bioavailability Score. This is reported as a probability that F>10% in rats. [Martin-2005]

```
MIN_ABS 0.11 "Minimum probability F>10% in rats"
```

Pharmacopia/Egan

Egan egg measure of bioavailability (LogP >5.88 or PSA > 131.6). [Egan-2000]

```
PHARMACOPIA false "LogP > 5.88 or PSA > 131.6"
```

Aggregators

Aggregators are small molecules that can interfere with assay results by sequestering protein in an aggregation of small molecules in solution. They appear to have activity in many assays, but in fact are usually not specific inhibitors of the protein in question. Includes two measures of whether a molecule is one of the aggregators defined by Shoichet et. al. [McGovern-2003] [Seidler-2003] The first measure, AGGREGATORS, is whether the molecule is an exact match to one of the approximately 400 published aggregators. The second measure, PRED_AGG, is whether the molecule hits in Shoichets QSAR model for predicting aggregators.

Aggregators

Whether a compound is known or predicted to aggregate in concentrations common in virtual screening.

```
AGGREGATORS true "Eliminate known aggregators"  
PRED_AGG false "Eliminate predicted aggregators"
```

Elemental Filters

The elemental filters are applied in this order:

1. Test for the existence of any of the metals in the ELIMINATE_METALS filter in the molecule.
2. Remove salts by stripping away all the disconnected components except for the largest.
3. Test to make sure only atoms specified in ALLOWED_ELEMENTS filter are in the molecule.

See Also:

- *Metal Removal*
- *Salt Removal*

The format of the two elemental filter fields is the keyword followed by a comma delimited list of atomic symbols.

Eliminate Metals

Any molecule with the atoms indicated in ELIMINATE_METALS fail to pass the filter.

```
ELIMINATE_METALS Sc, Ti, V, Cr, Mn, Fe, Co, Ni, Cu, Zn, Y, Zr, Nb, Mo, Tc, Ru, Rh, Pd, Ag, Cd
```

Allowed Elements

Molecules with atoms other than those specified by `ALLOWED_ELEMENTS` fail to pass the filter.

```
ALLOWED_ELEMENTS H,C,N,O,F,P,S,Cl,Br,I
```

2.4.2 Functional Group Rules

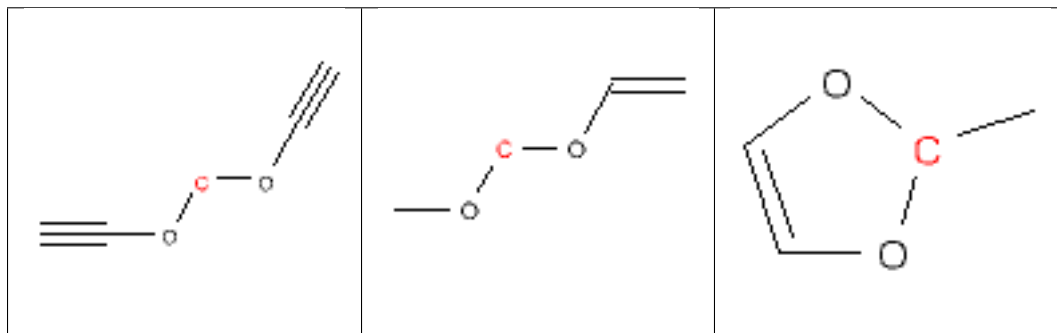
Rules statements set the limits for the maximum number of the specified type of functional group that may be allowed in the molecule.

The first field of a rule statement is the word `RULE` in all capital letters. The second field is a number indicating the maximum number of the group allowed in a molecule. The third field is the functional group keyword. Functional-group keywords are case sensitive.

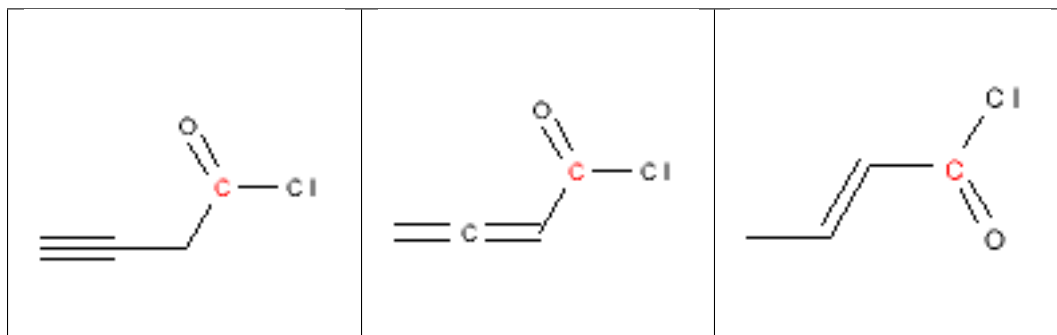
```
RULE 0 acid_halide
```

The following is a list of functional groups which filter recognizes by default. Three example matches are provided with the atoms that correspond to each other highlighted. Due to the highly complex nature of the patterns, in particular recursive SMARTS, it is not possible to fully highlight every atom that was included as part of the match.

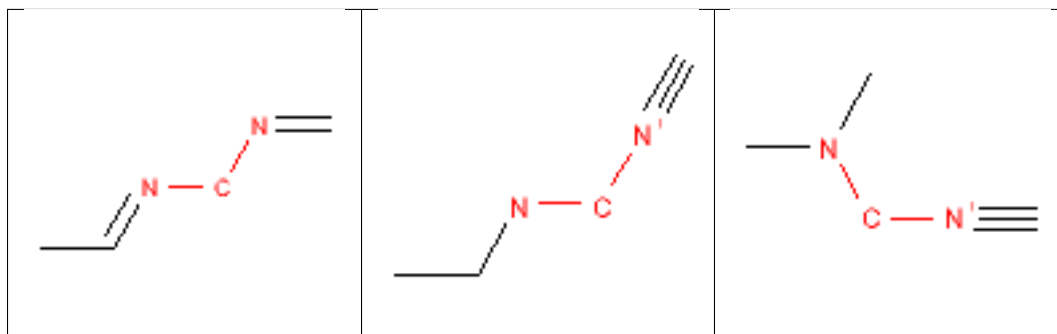
acetal



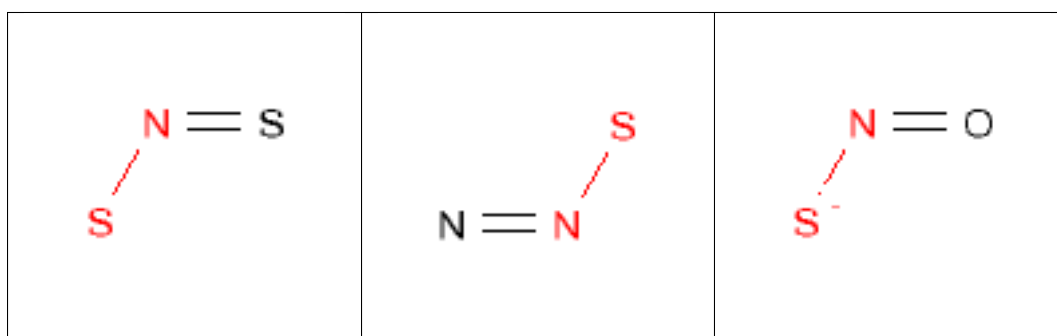
acid_halide



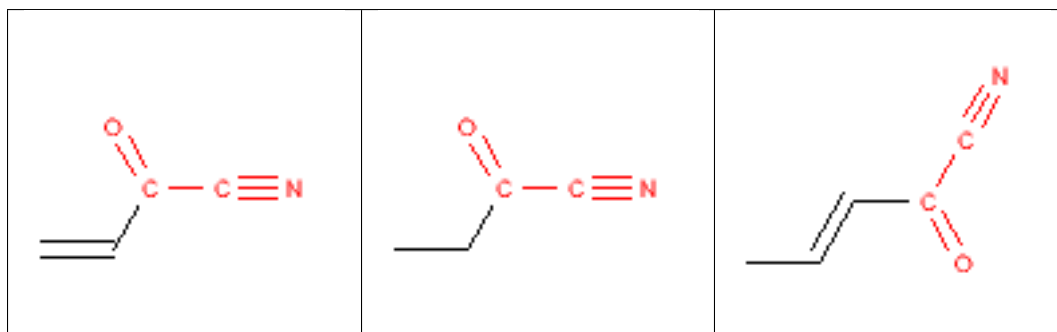
acyclic_NCN



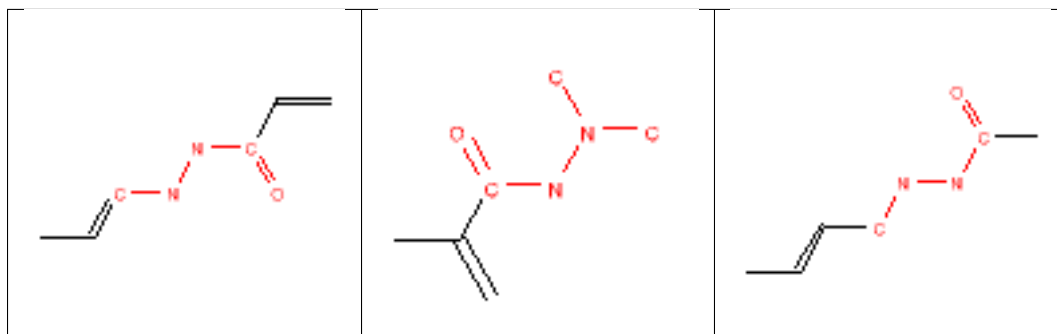
acyclic_NS



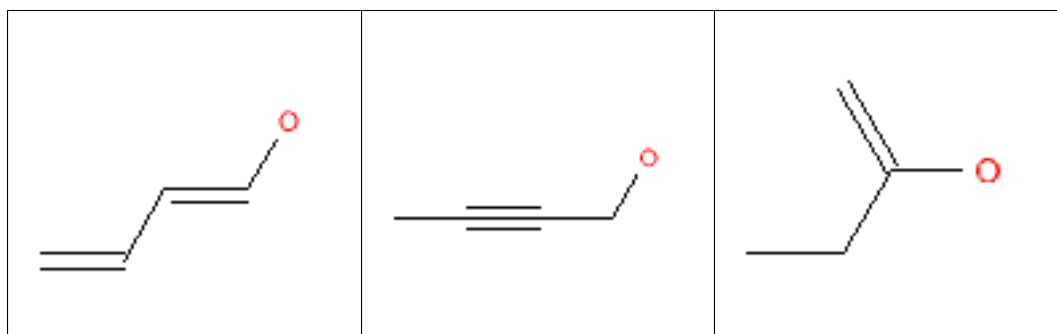
acyl_cyanides



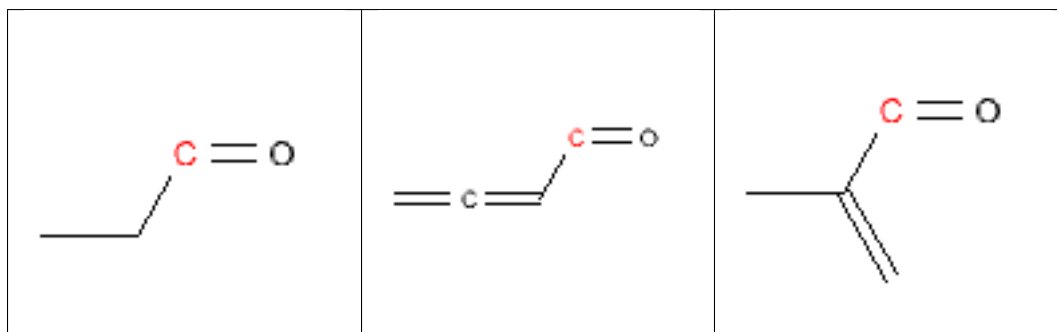
acylhydrazide



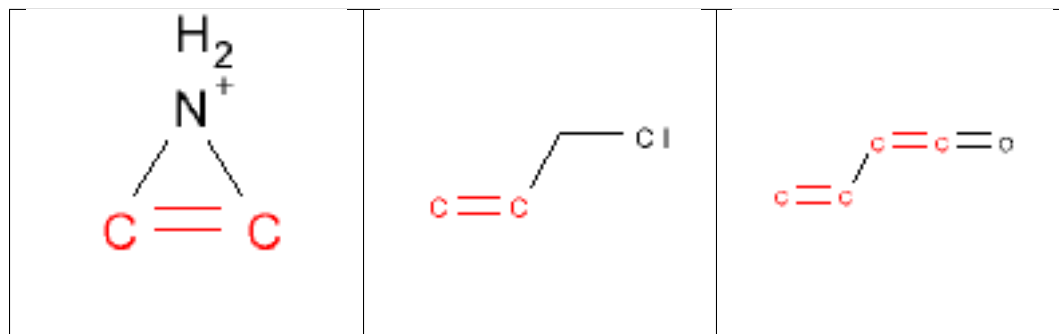
alcohol



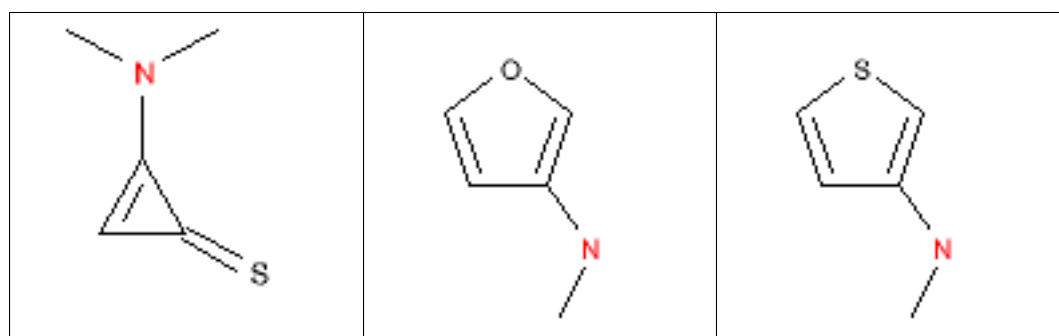
aldehyde



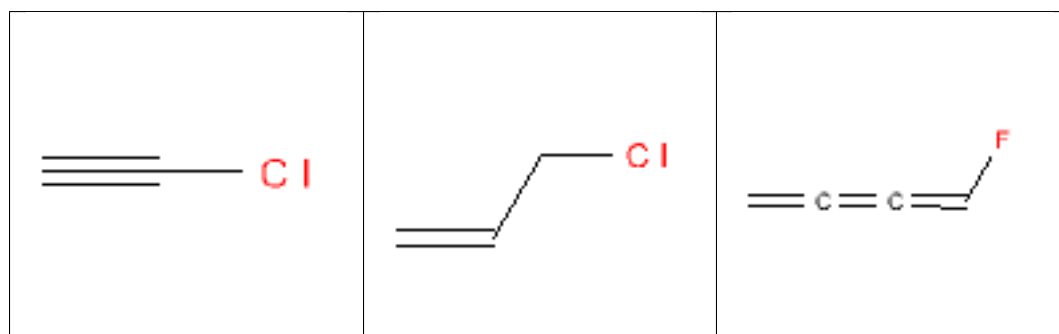
alkene



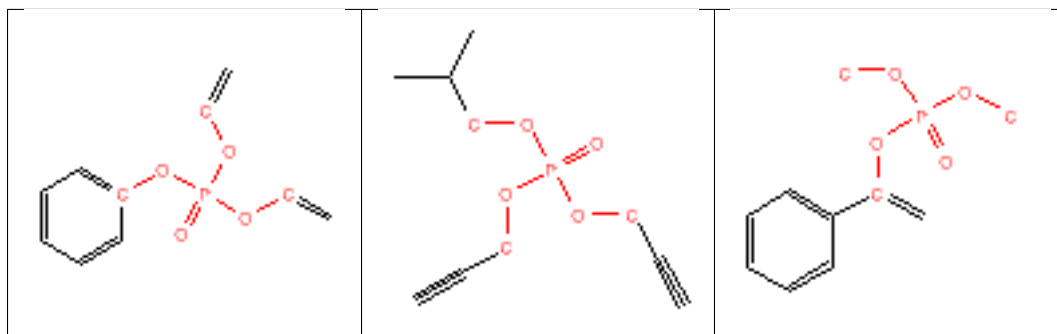
alkylaniline



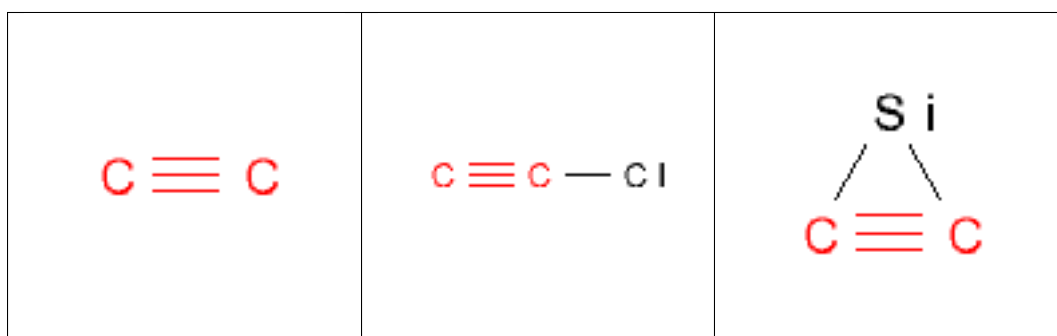
alkyl_halide



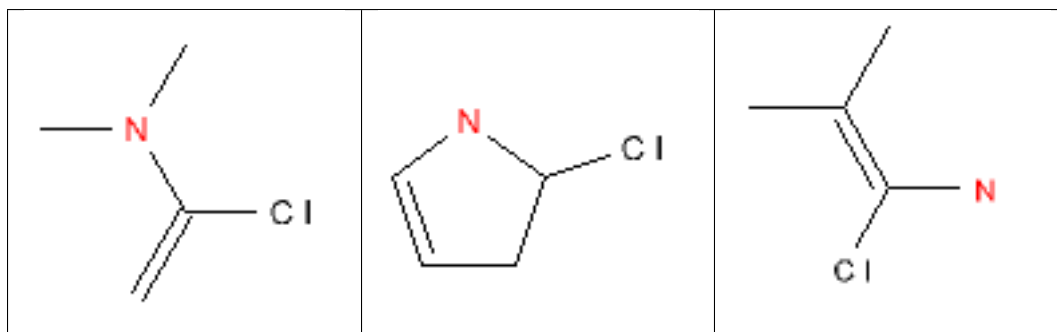
alkyl_phosphate



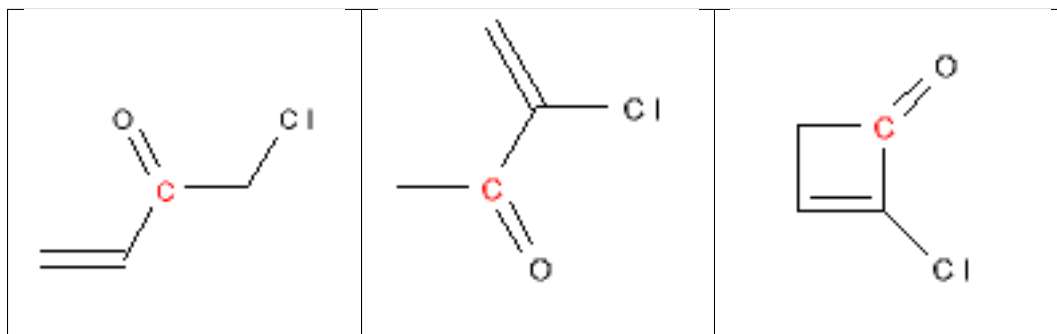
alkyne



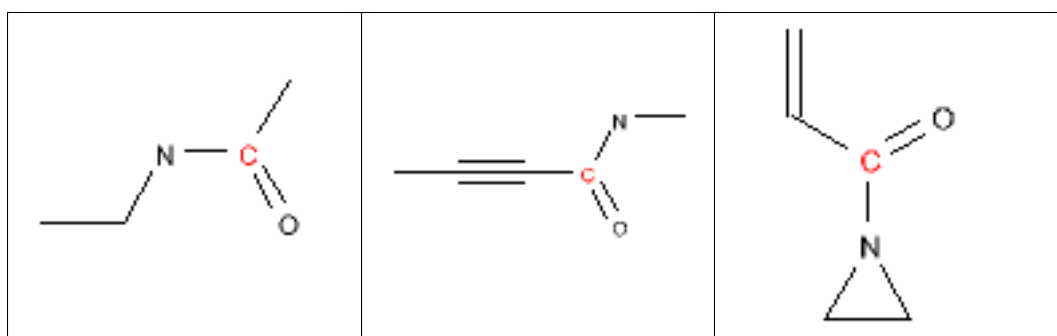
alphahalo_amine



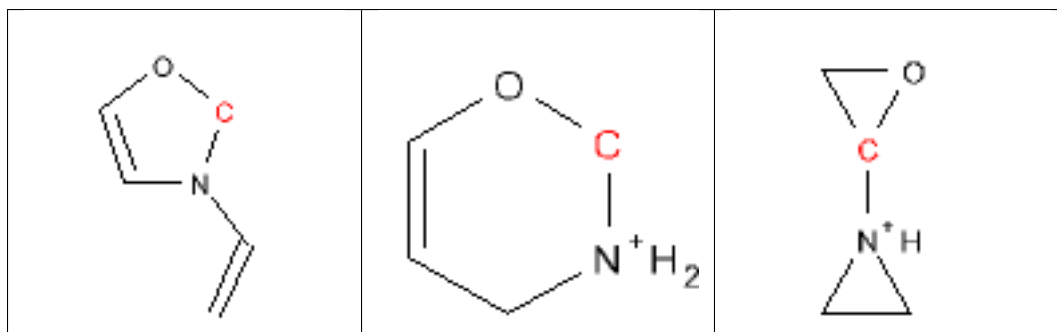
alphahalo_ketone



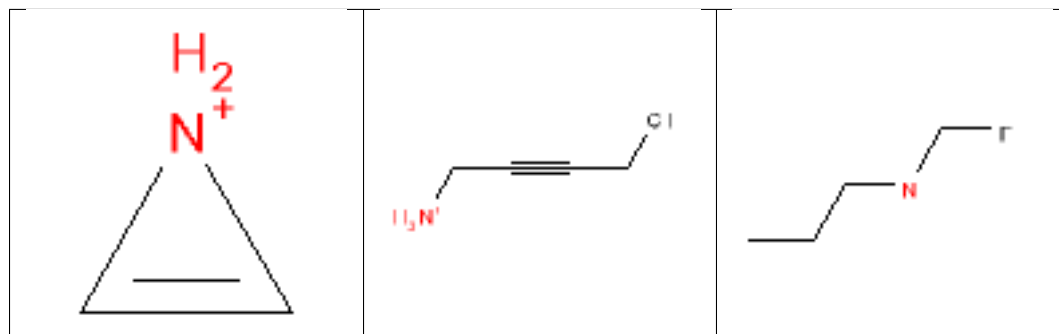
amide



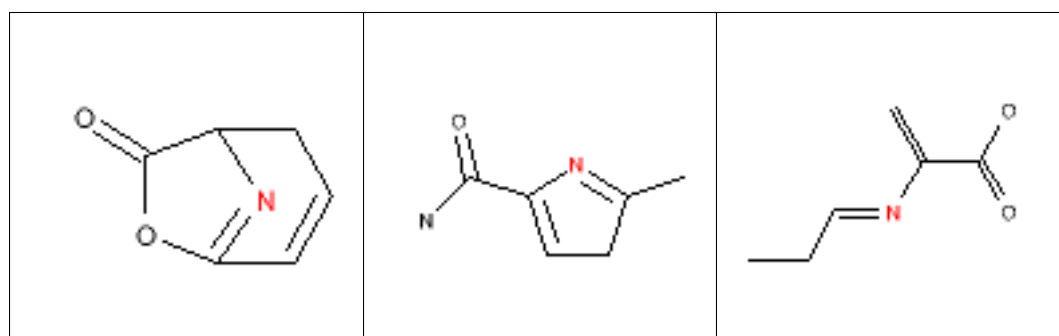
aminal



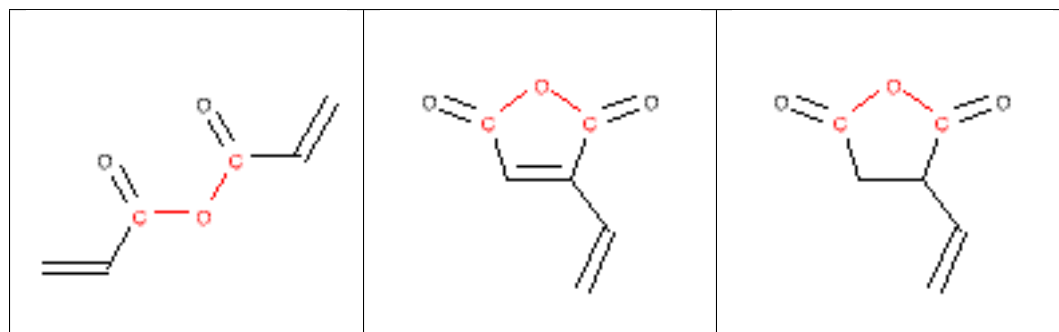
amine



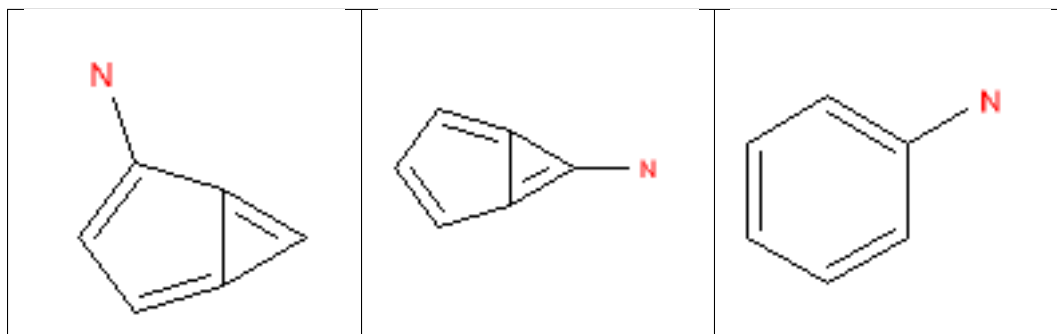
amino_acid



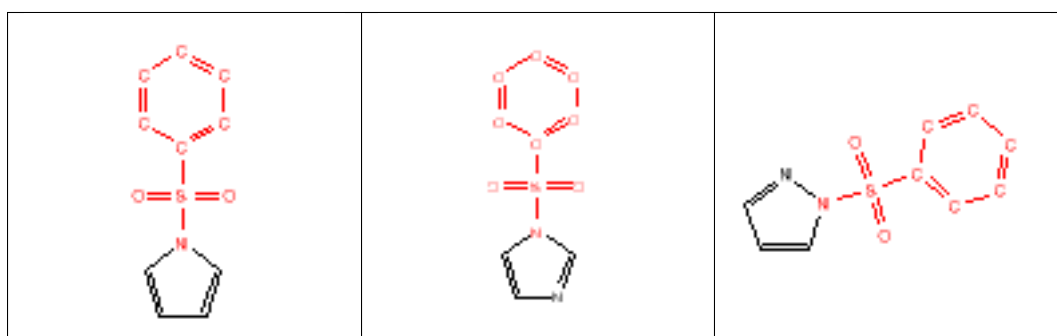
anhydride



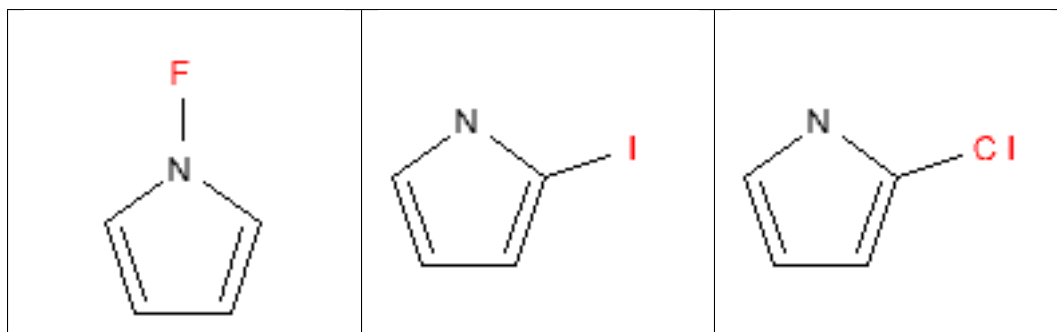
aniline



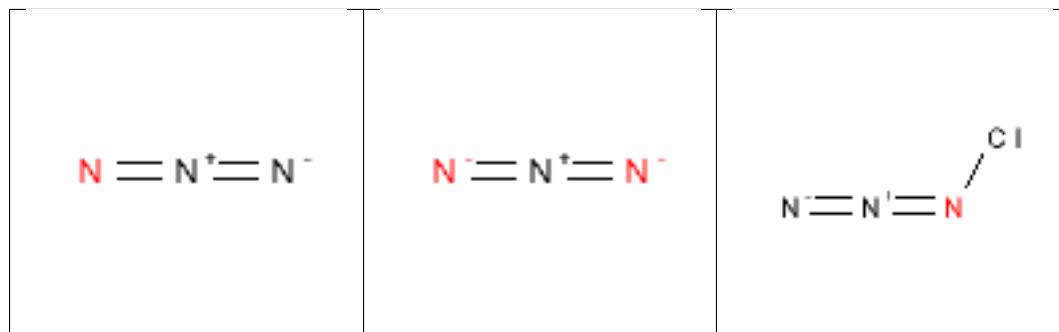
arenesulfonyl



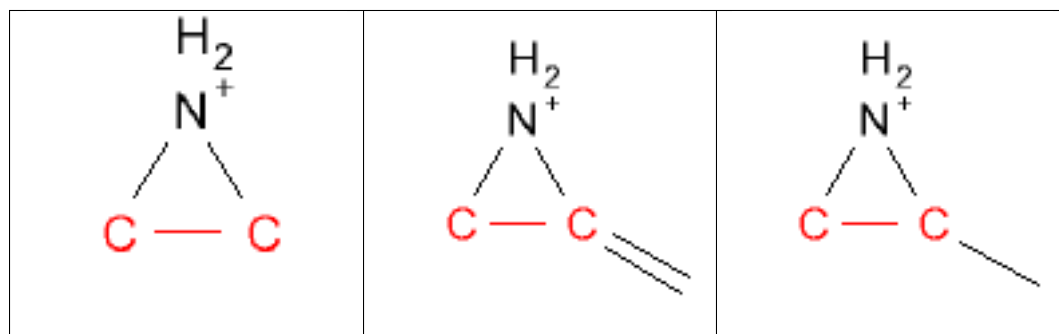
aryl_halide



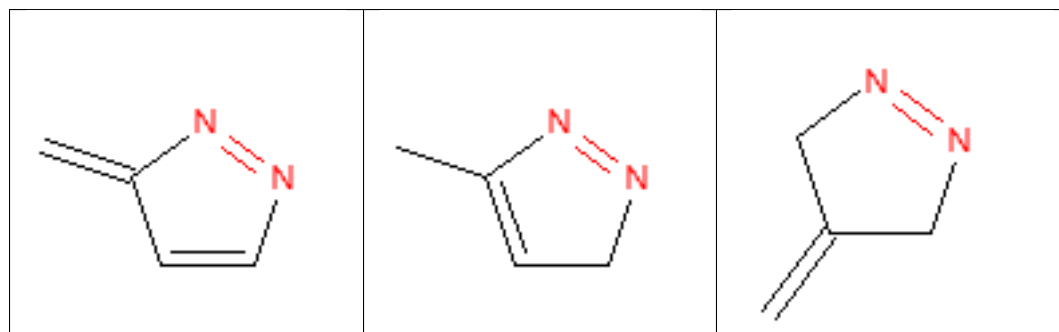
azide



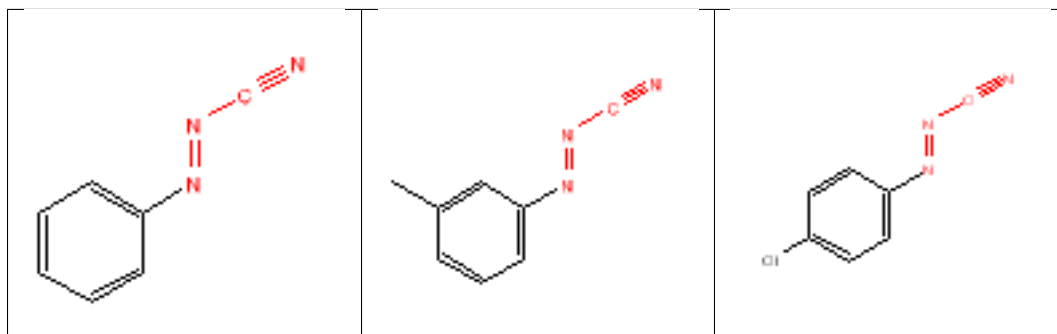
aziridine



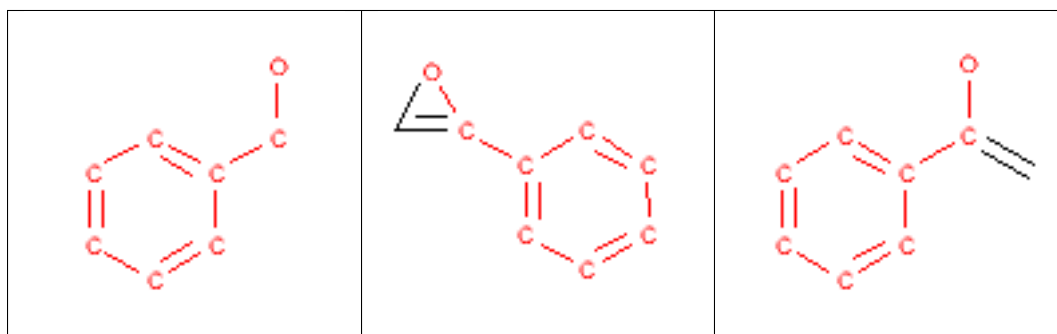
azo



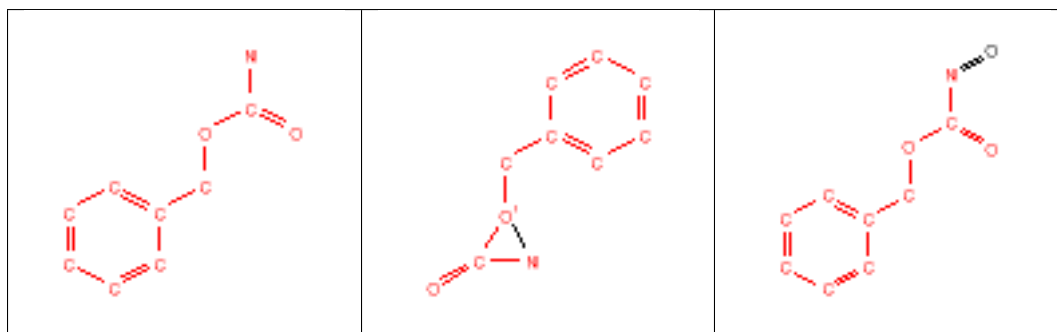
azocyanamides



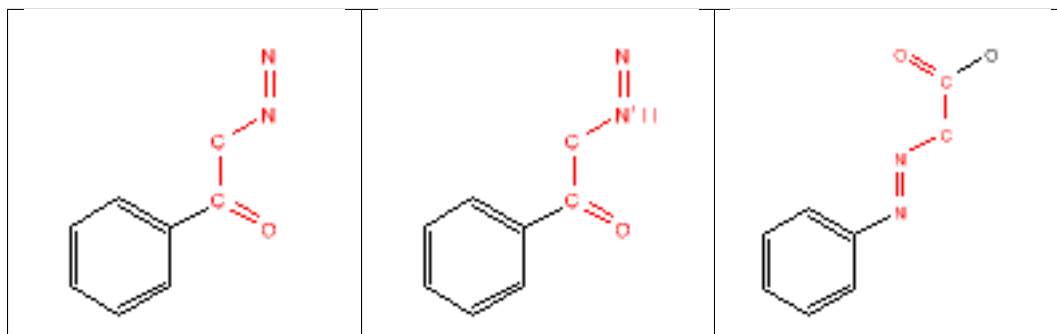
benzyl_ether



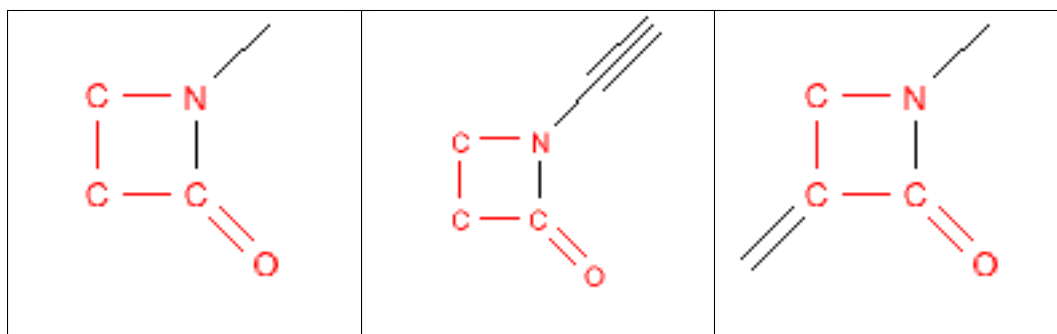
benzyloxycarbonyl_CBZ



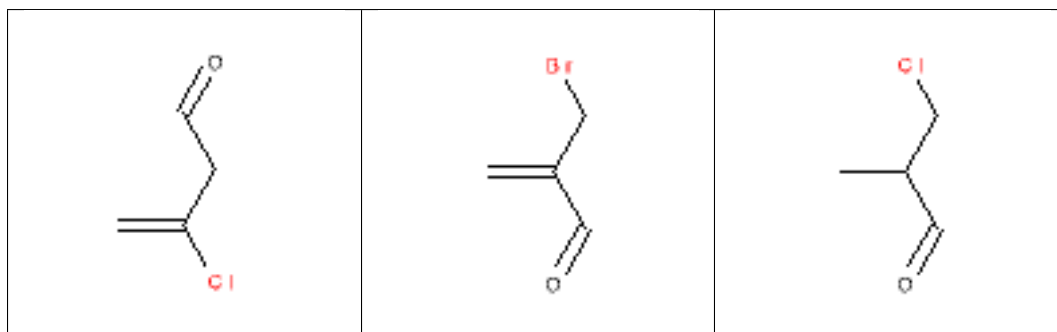
beta_azo_carbonyl



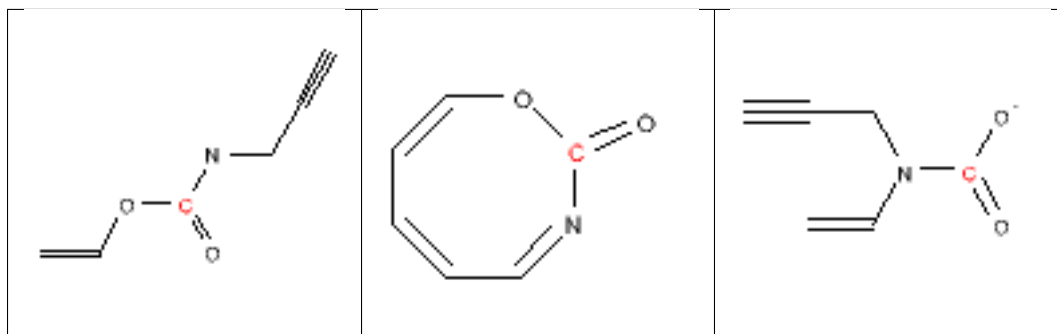
beta_carbonyl_quat_nitrogen



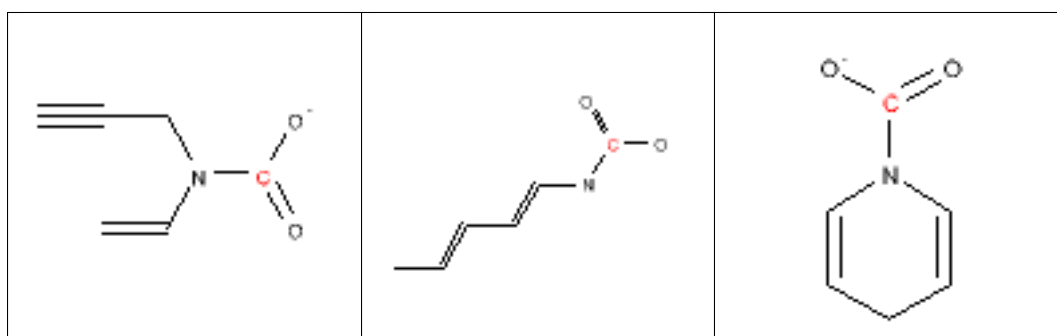
beta_halo_carbonyl



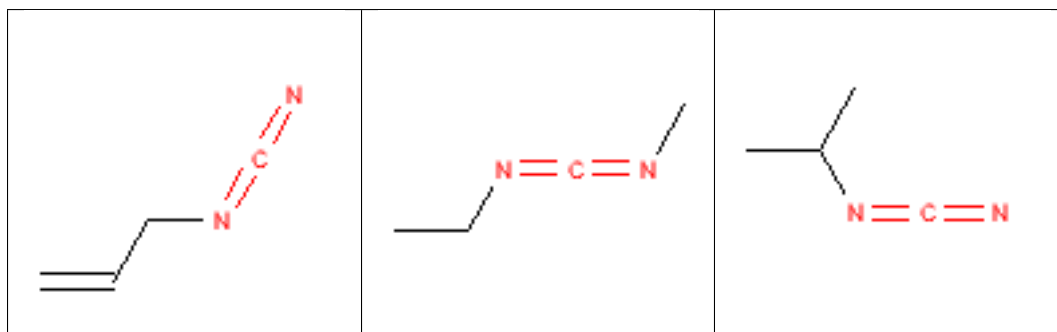
carbamate



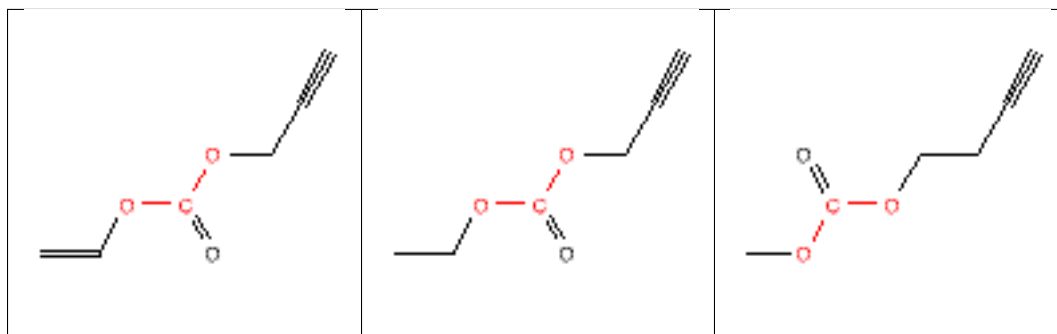
carbamic_acid



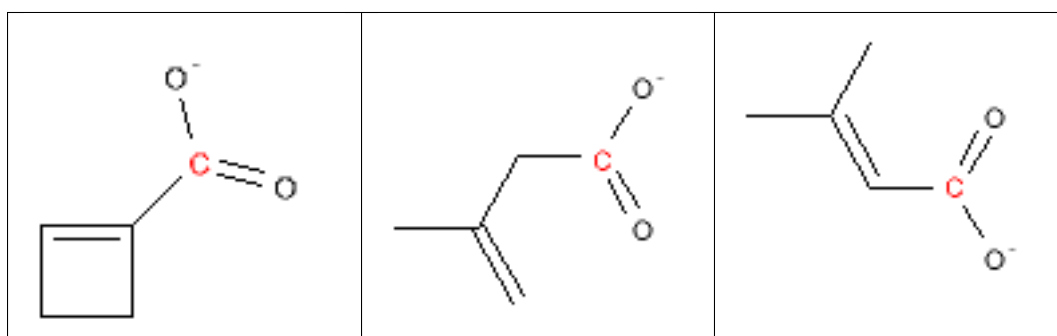
carbodiimide



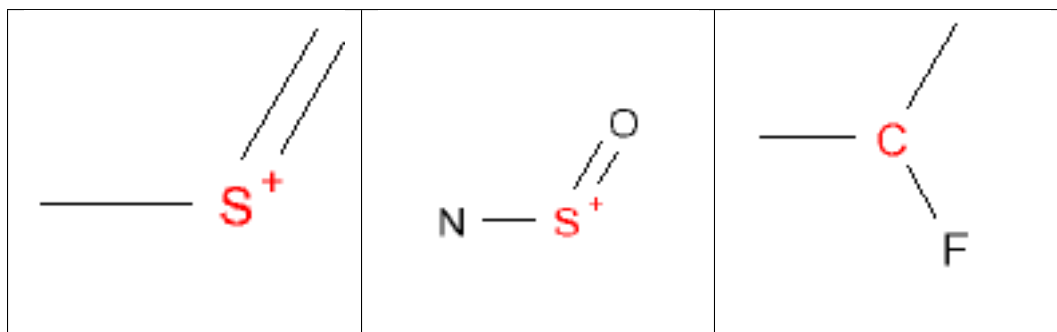
carbonate



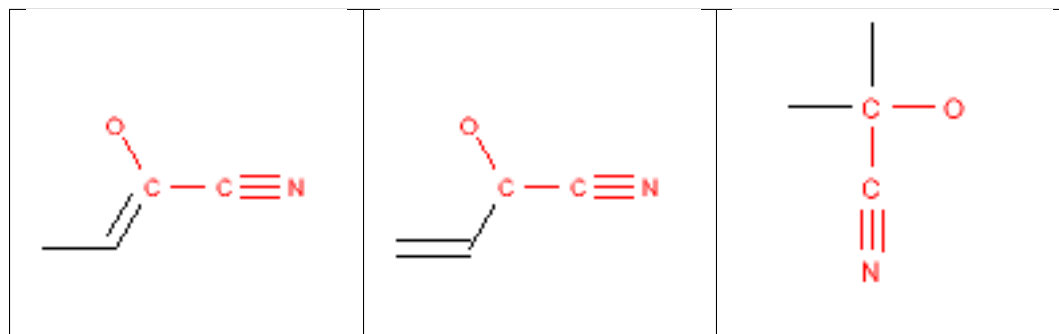
carboxylic_acid



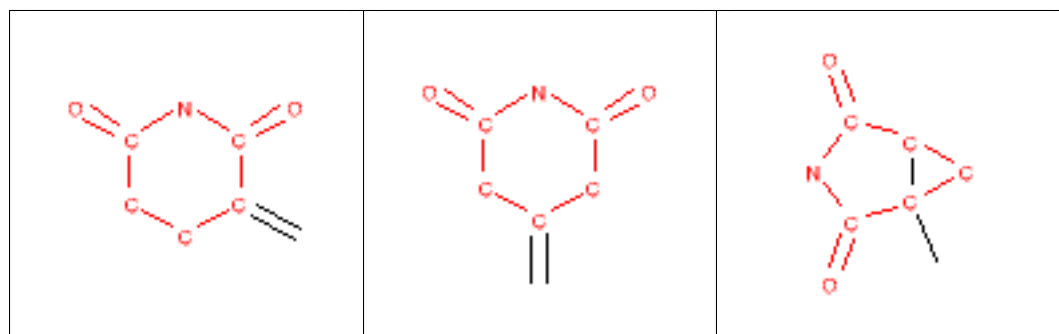
cation_C_Cl_I_P_or_S



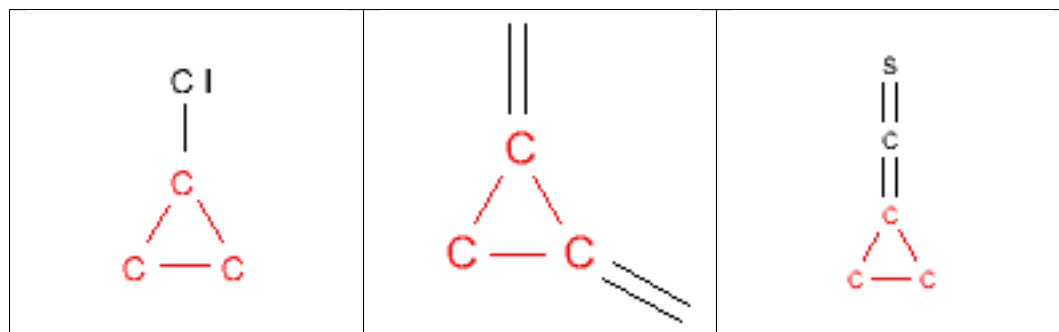
cyanohydrins



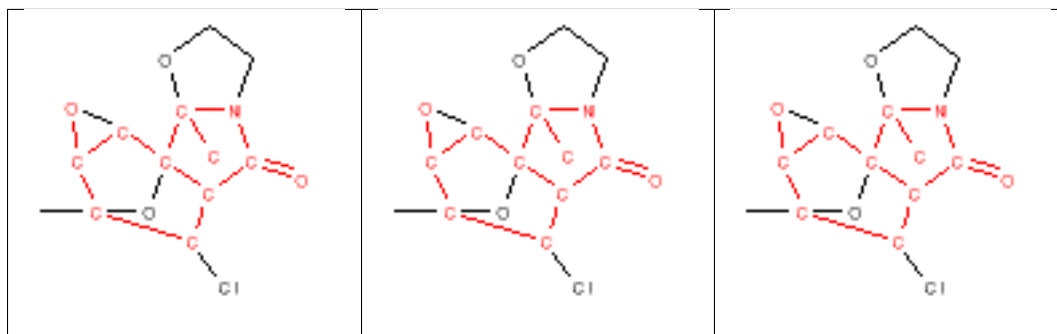
cycloheximide_derivatives



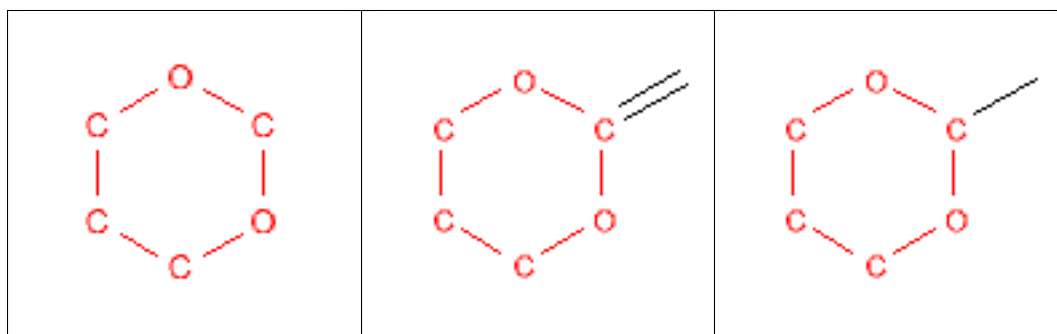
cyclopropyl



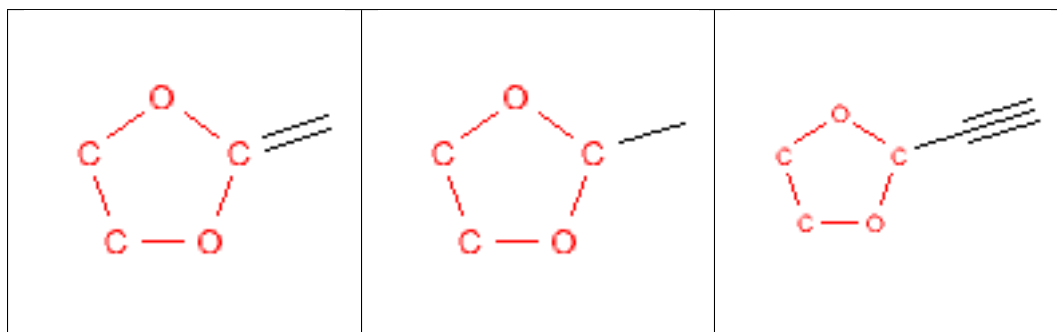
cytochalasin_derivatives



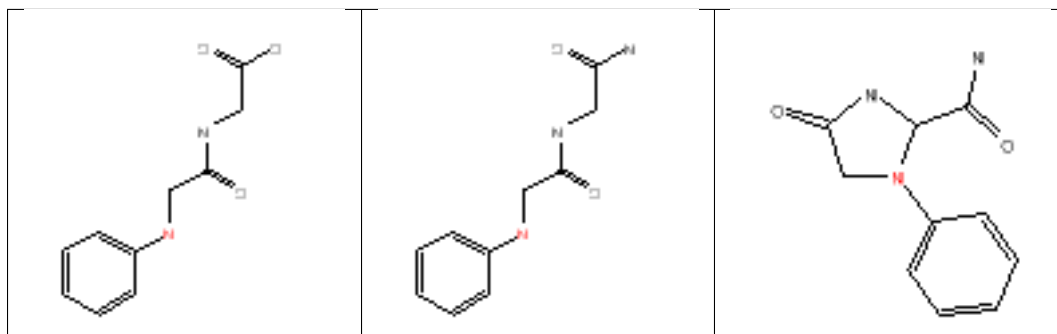
dioxane_6MR



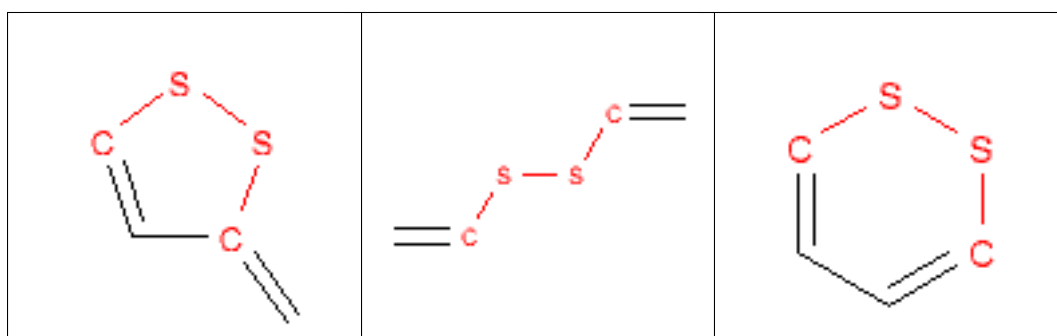
dioxolane_5MR



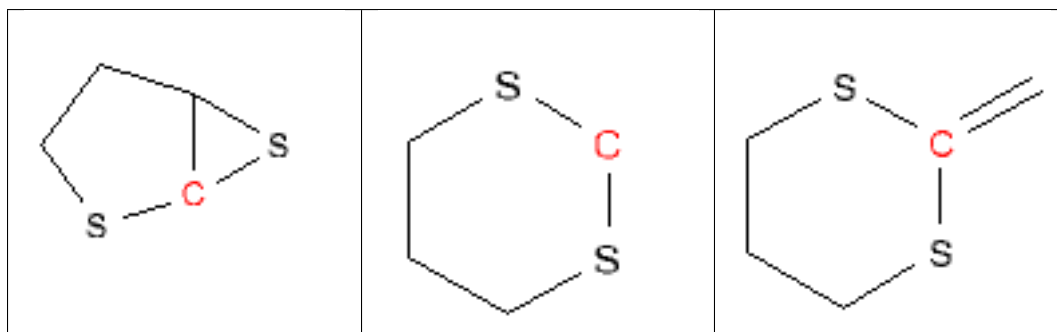
di_peptide



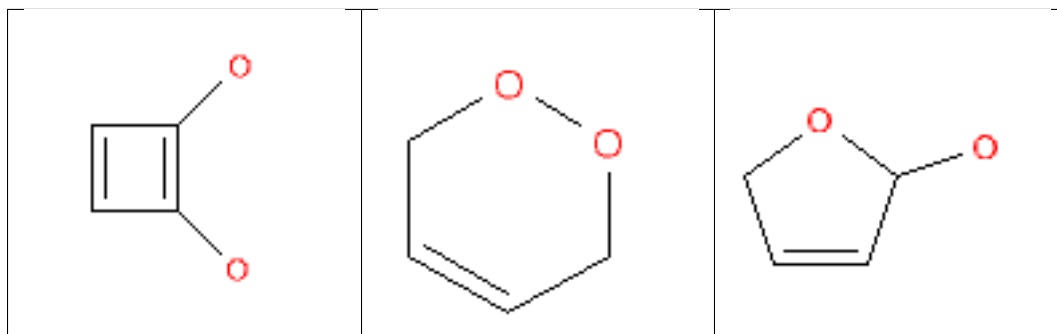
disulfide



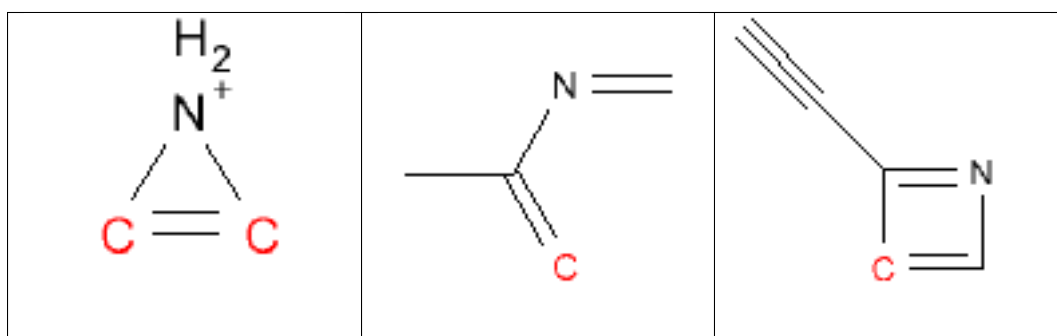
dithioacetal



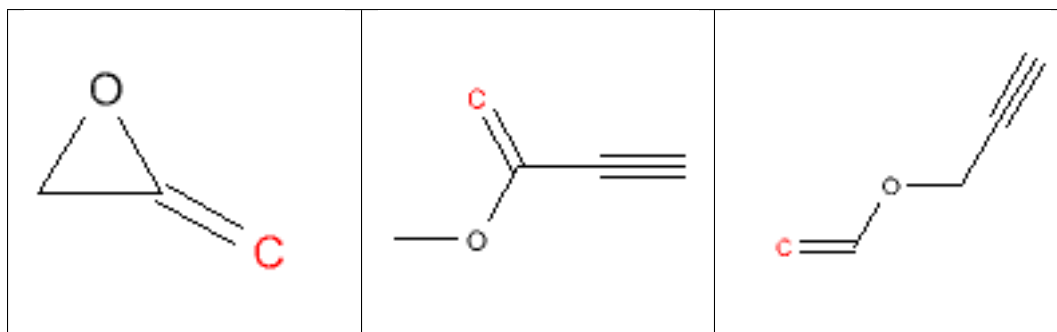
dye



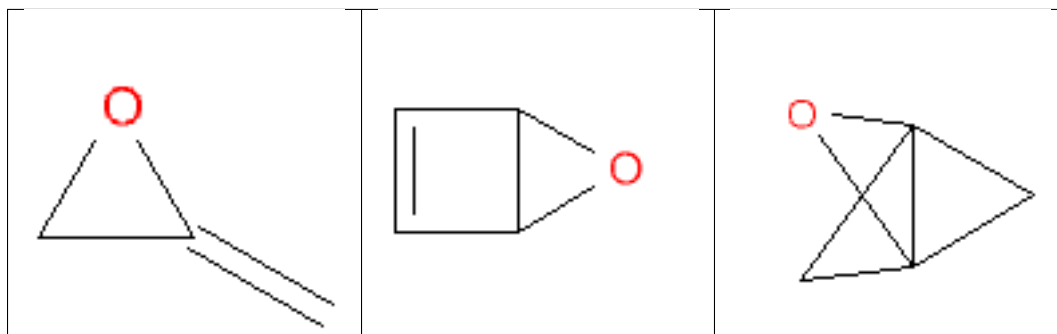
enamine



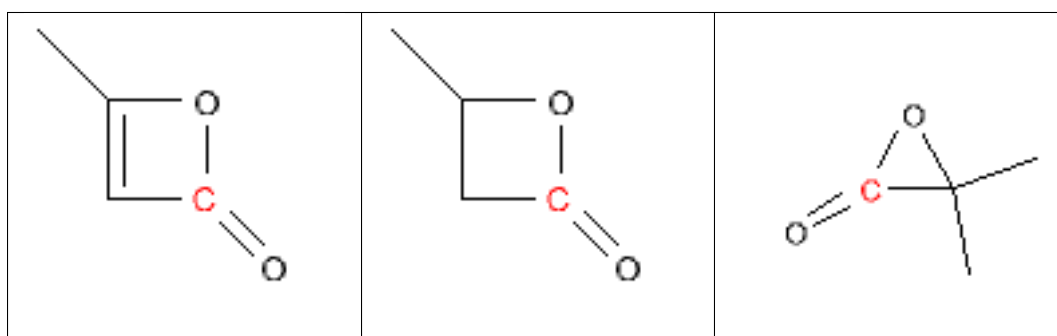
enol_ether



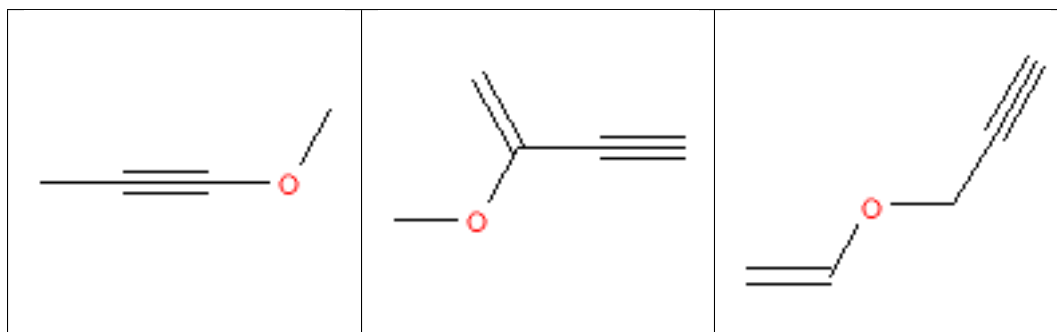
epoxide



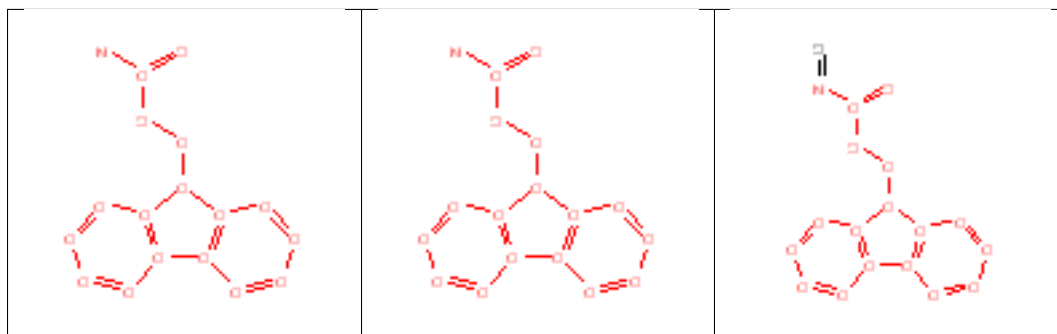
ester



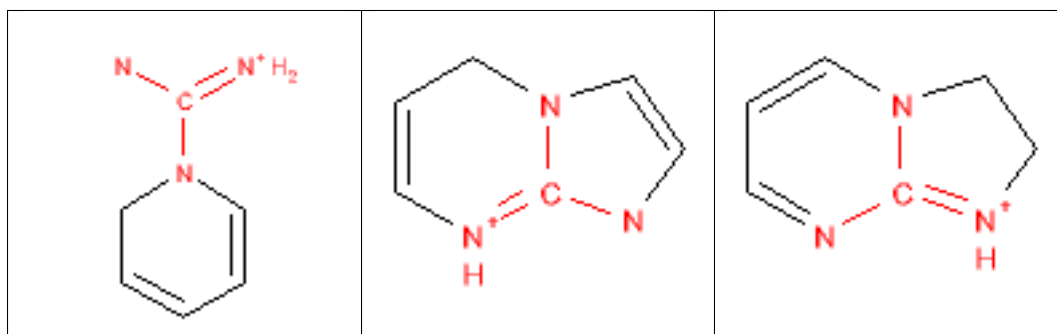
ether



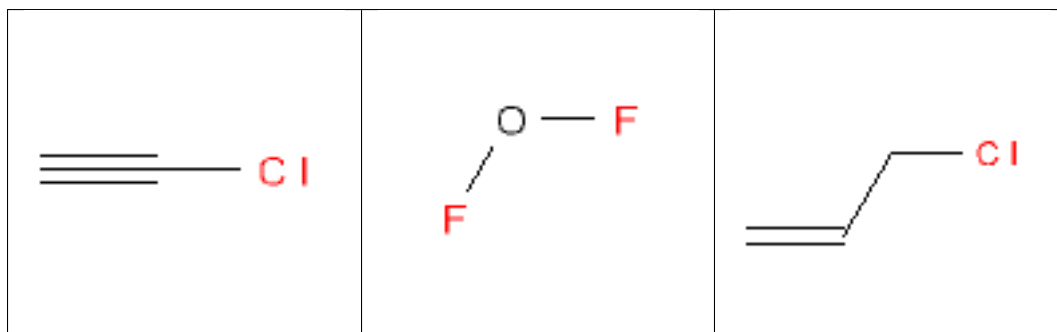
fluorenylmethoxycarbonyl_Fmoc



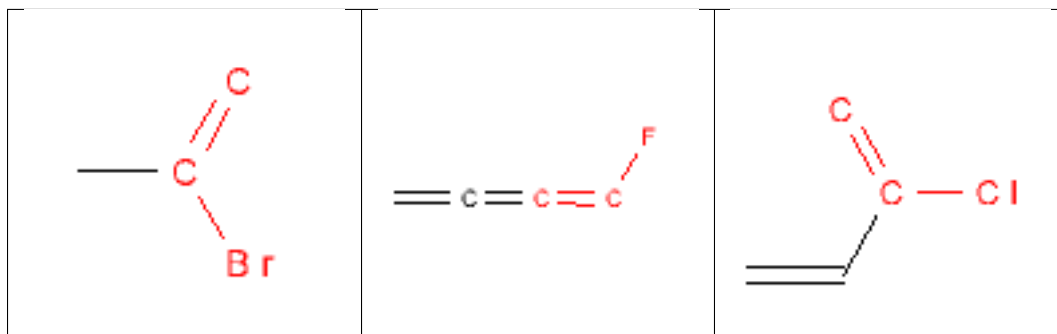
guanidine



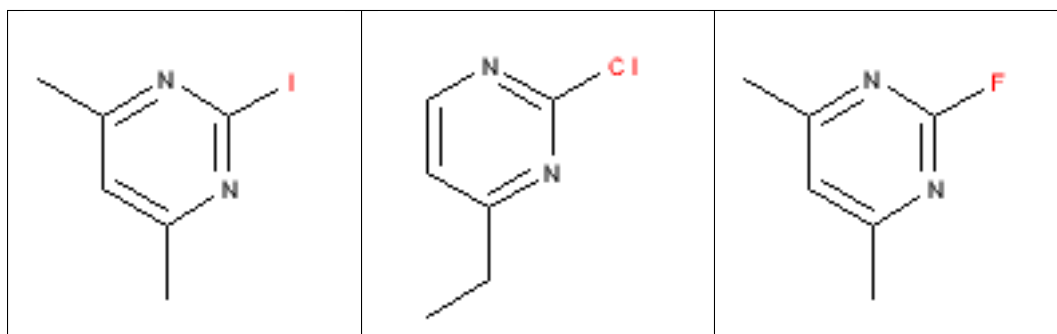
halide



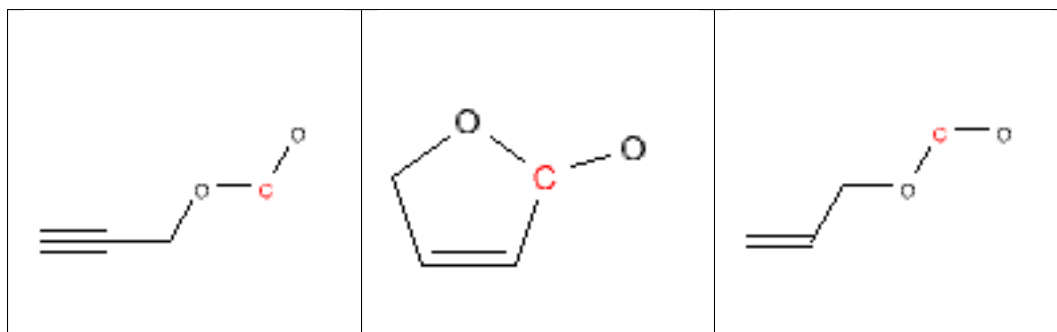
halo_alkene



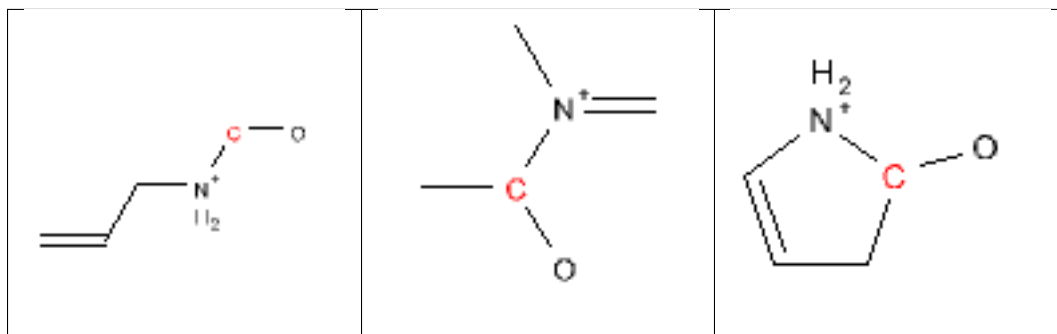
halopyrimidine



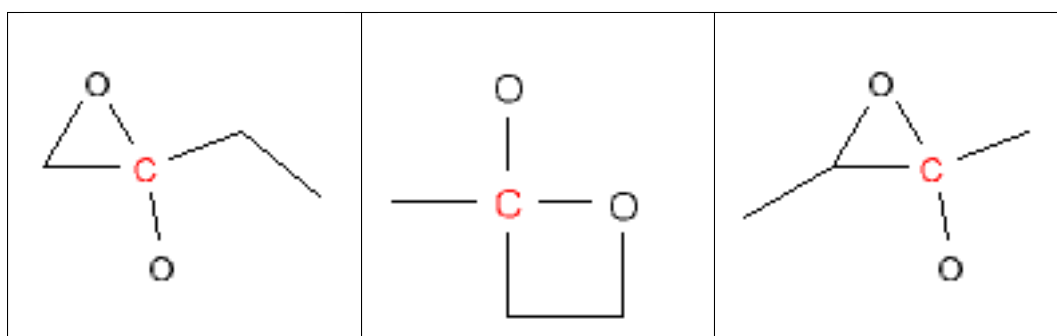
hemiacetal



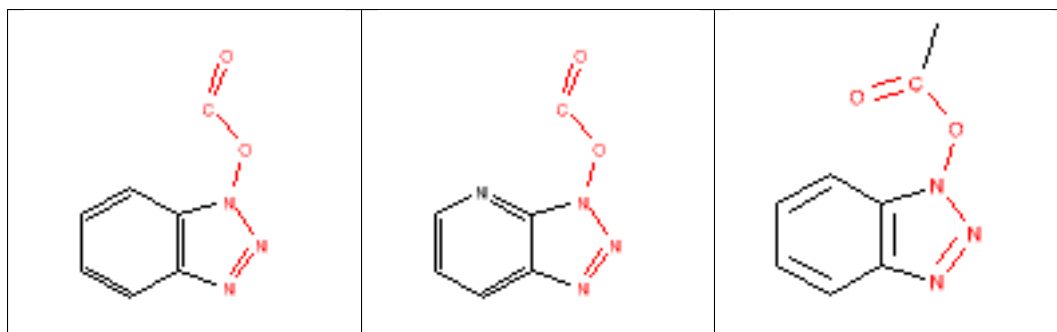
hemiaminal



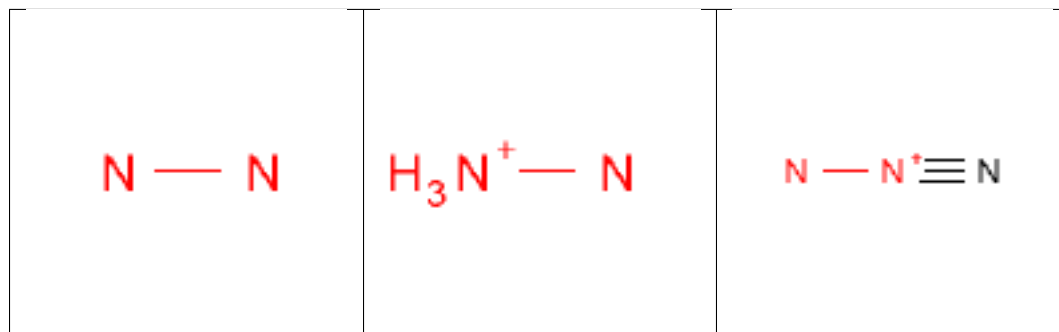
hemiketal



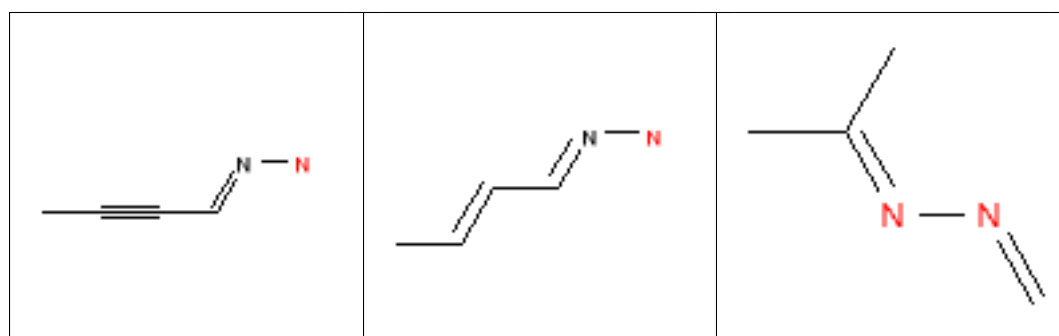
HOBT_esters



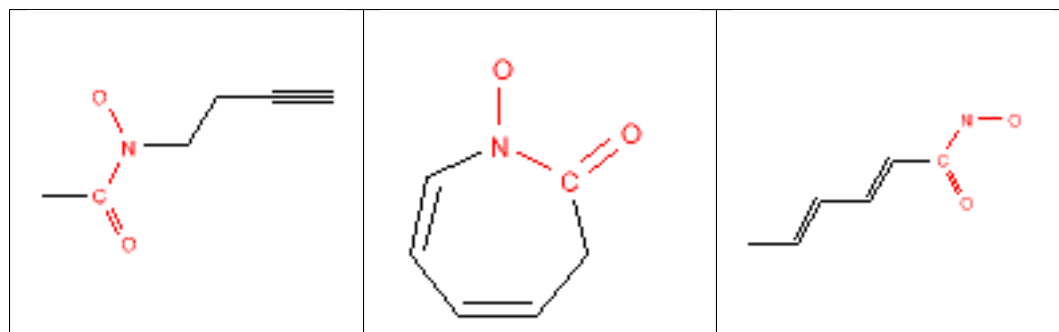
hydrazine



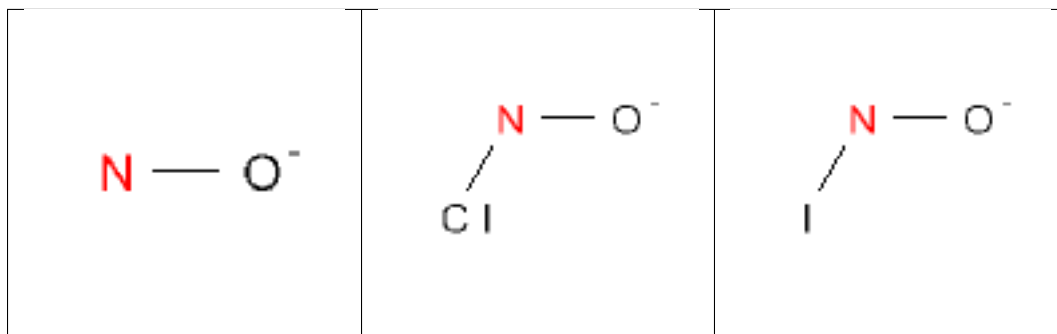
hydrazone



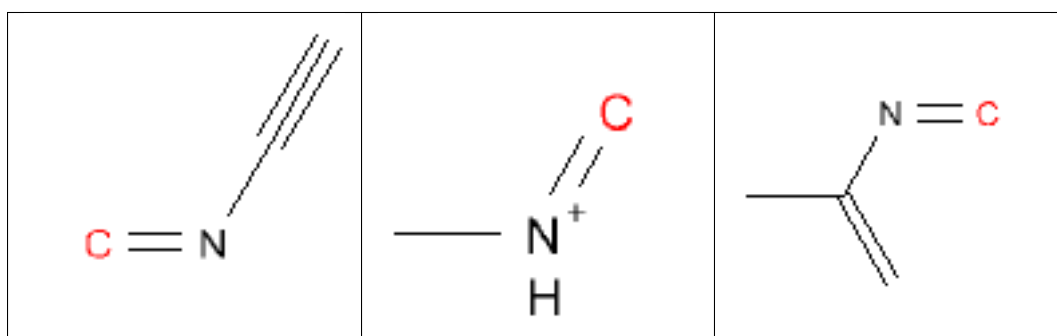
hydroxamic_acid



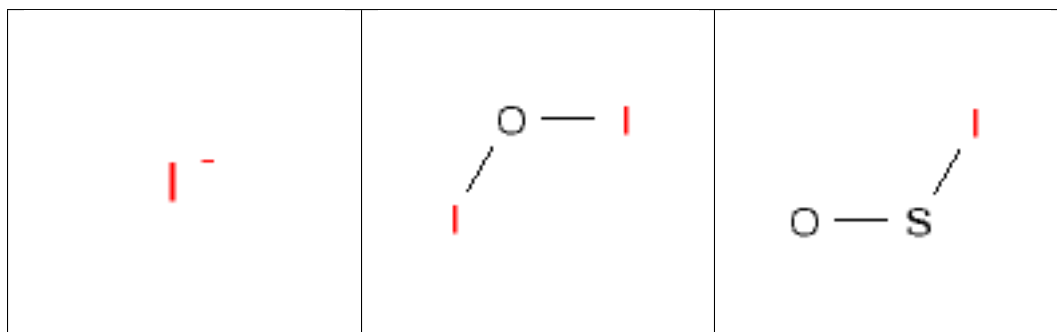
hydroxylamine



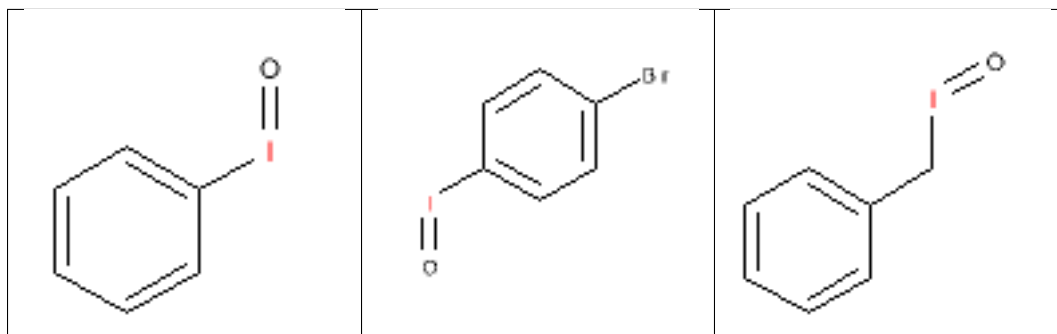
imine



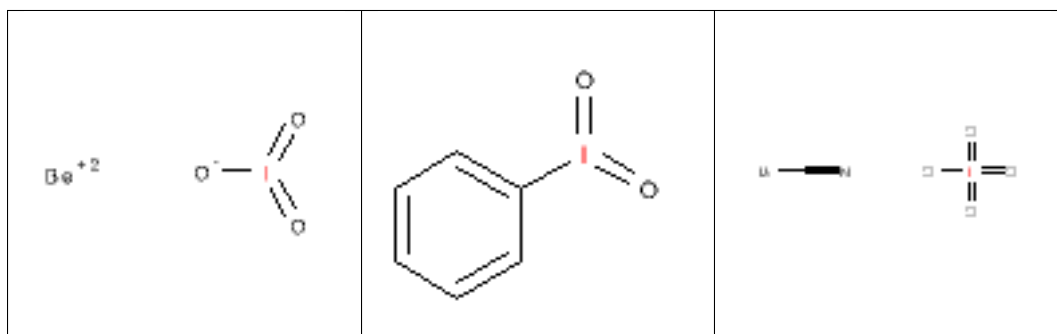
iodine



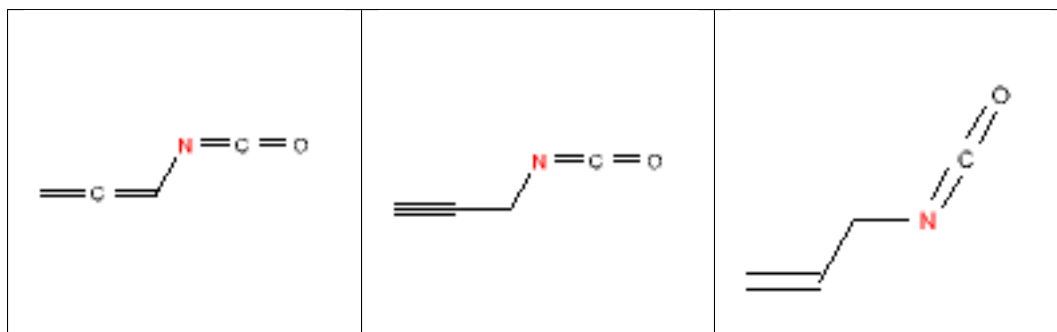
iodoso



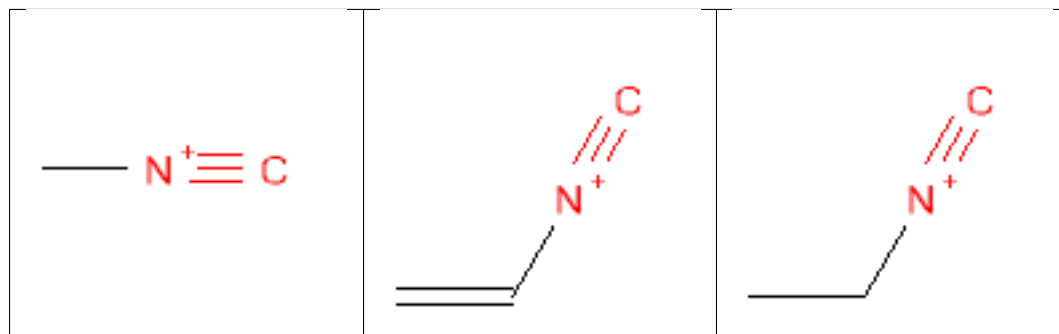
iodoxy



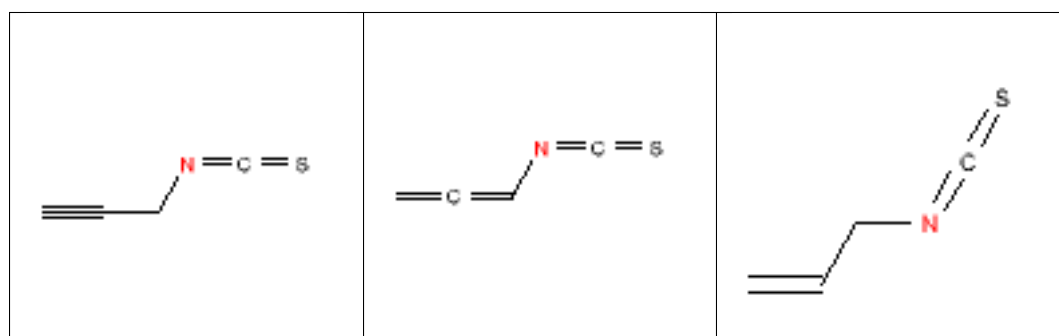
isocyanate



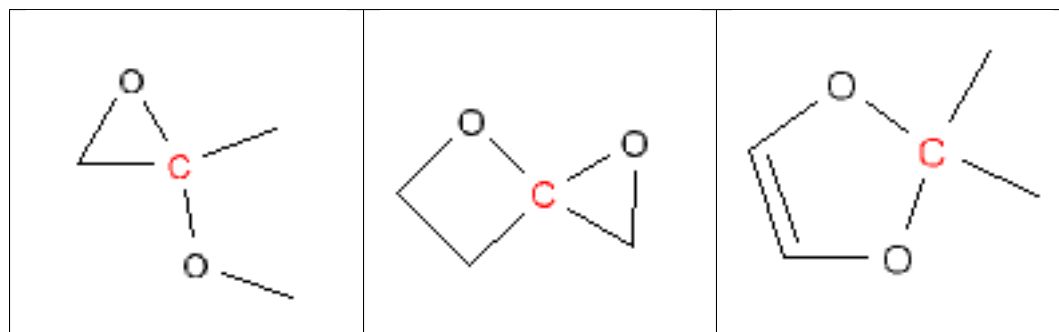
isonitrile



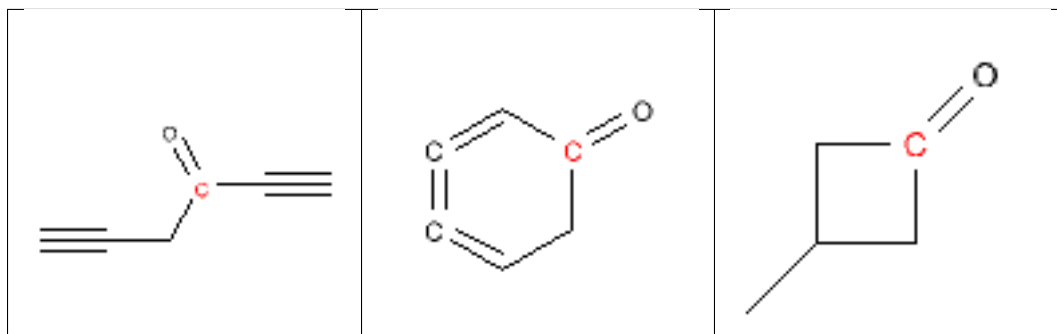
isothiocyanate



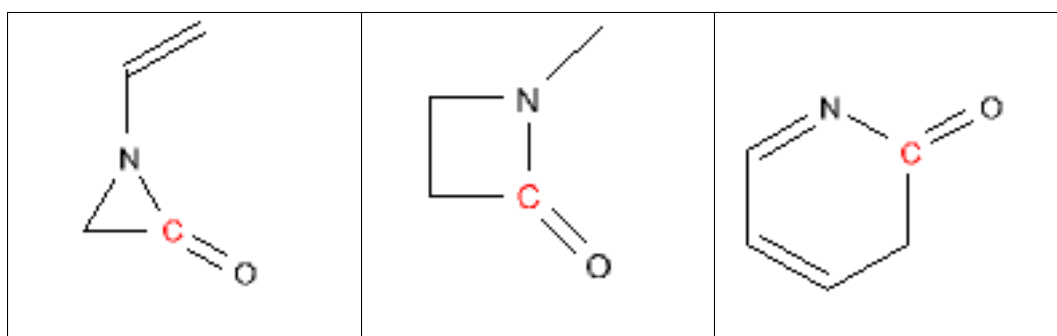
ketal



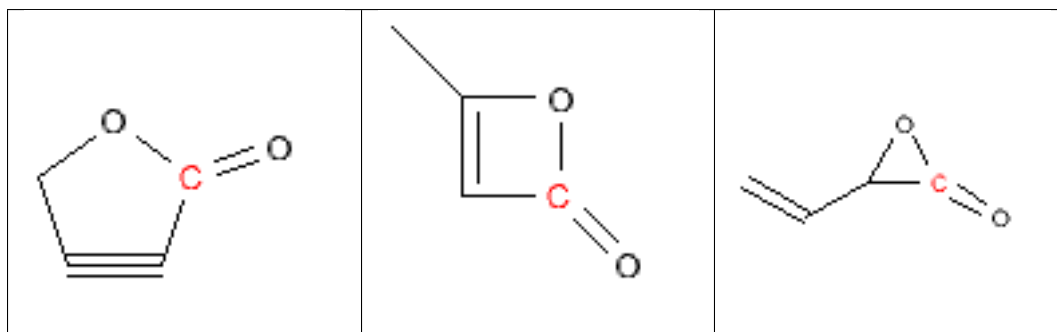
ketone



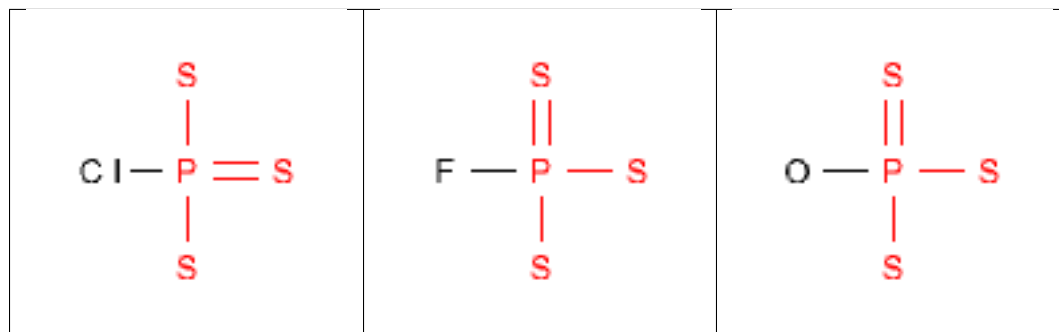
lactam



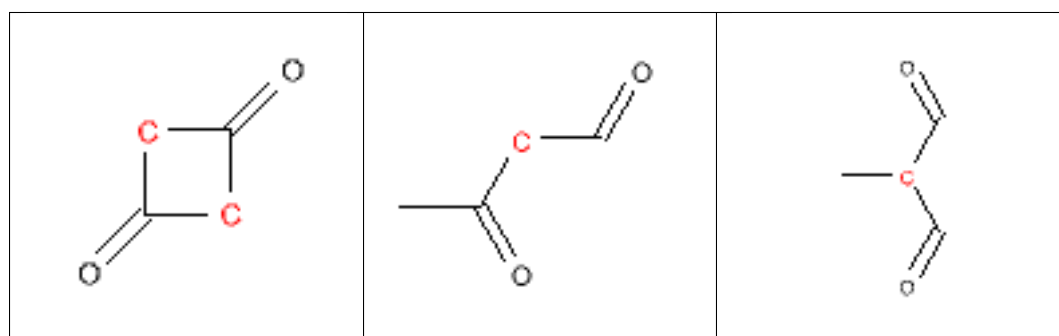
lactone



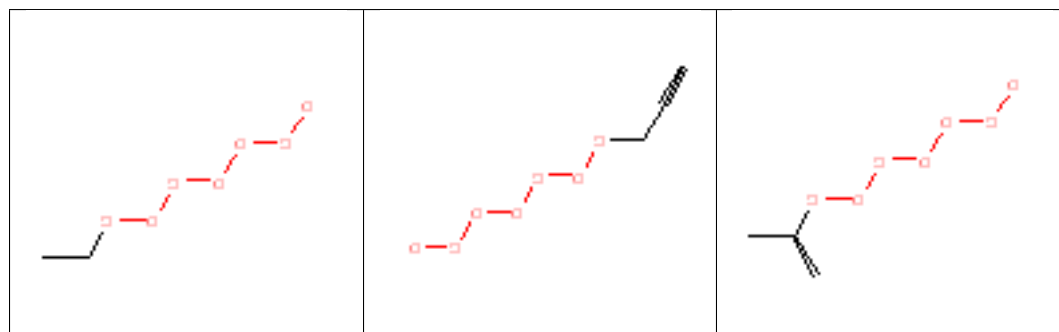
lawesson_s_reagent



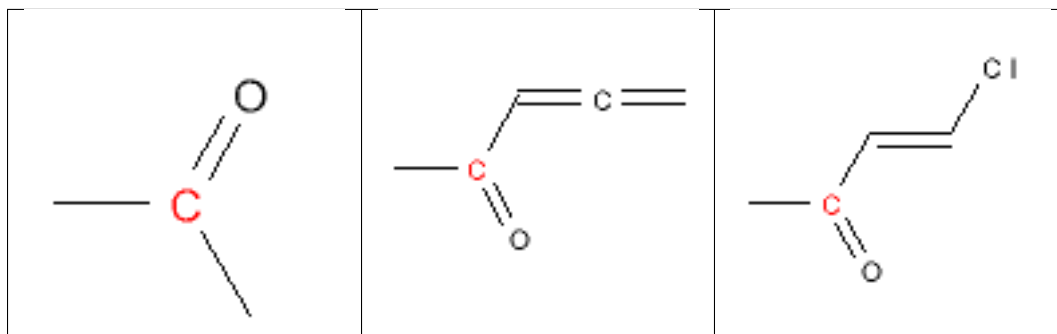
malonic



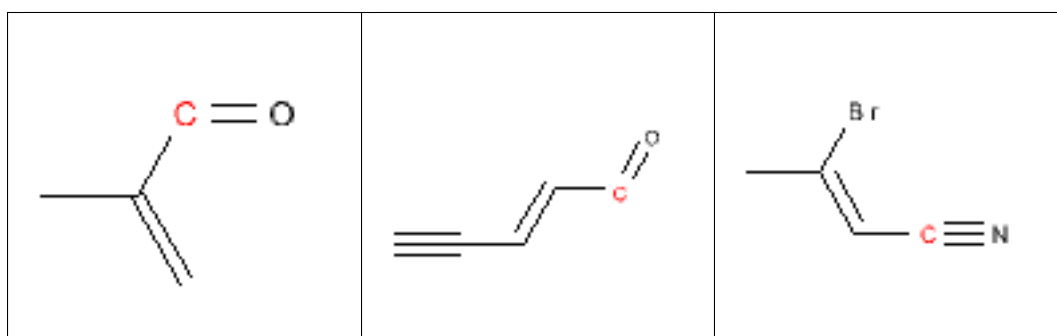
methoxyethoxymethyl_MEM



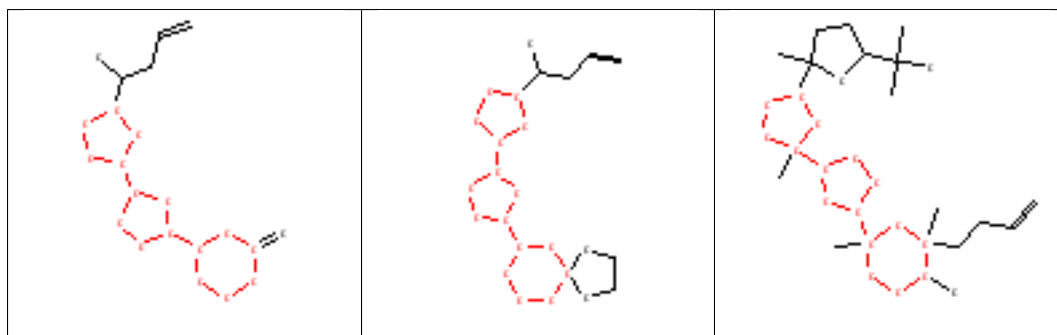
methyl_ketone



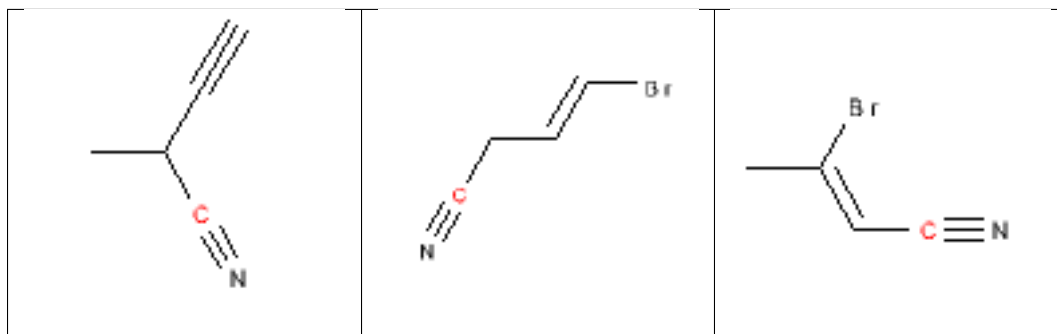
michael_acceptor



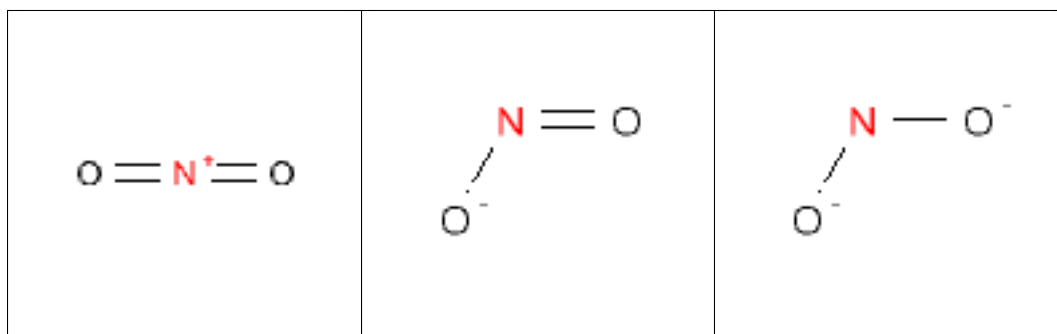
monensin_derivatives



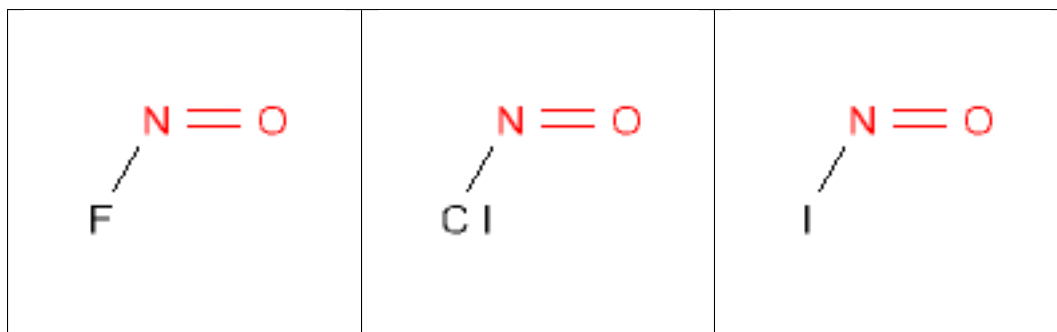
nitrile



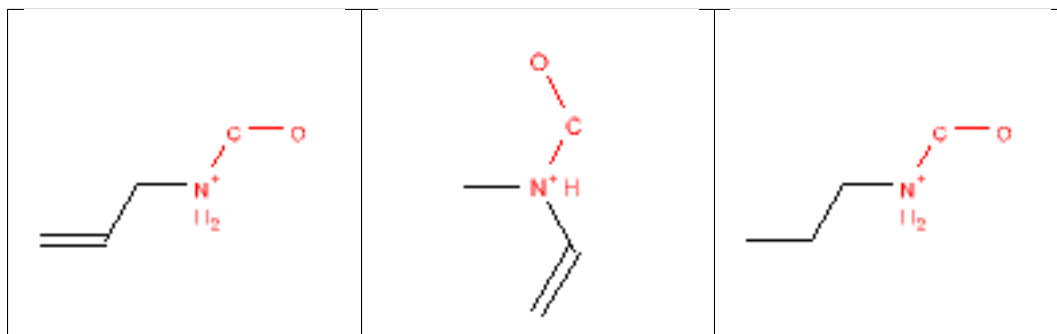
nitro



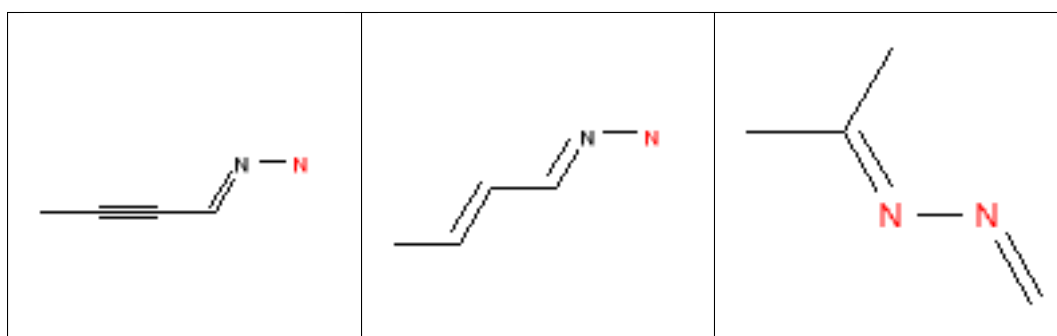
nitroso



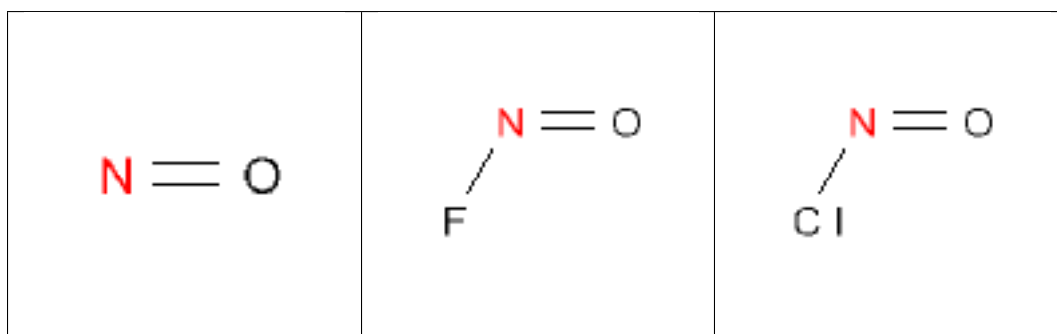
N_methoyl



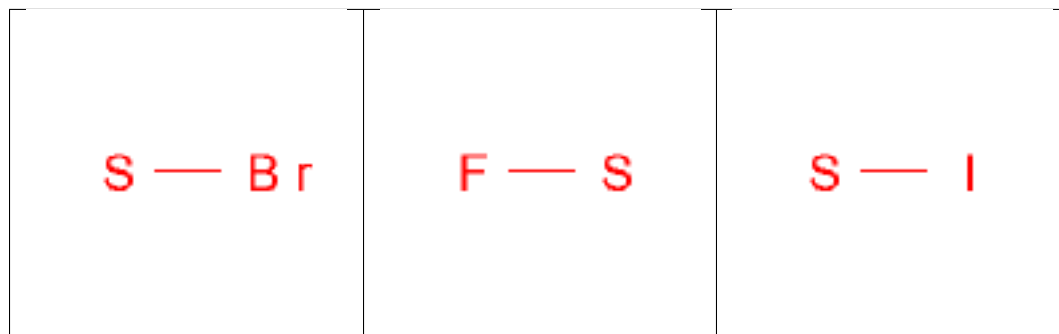
nonacylhydrazone



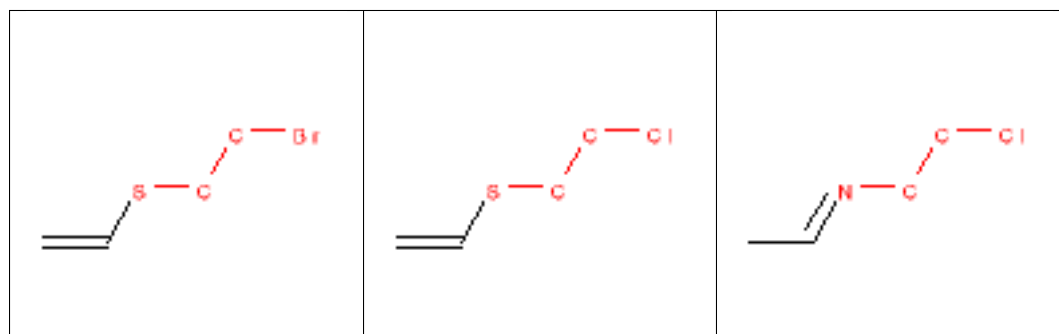
noxide



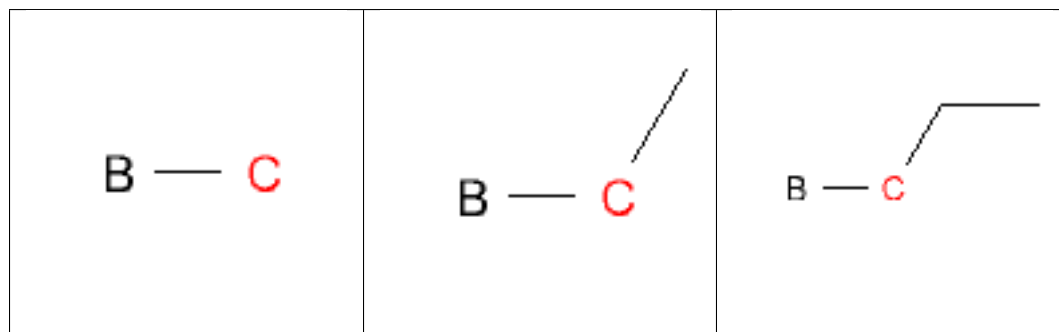
N_P_S_Halides



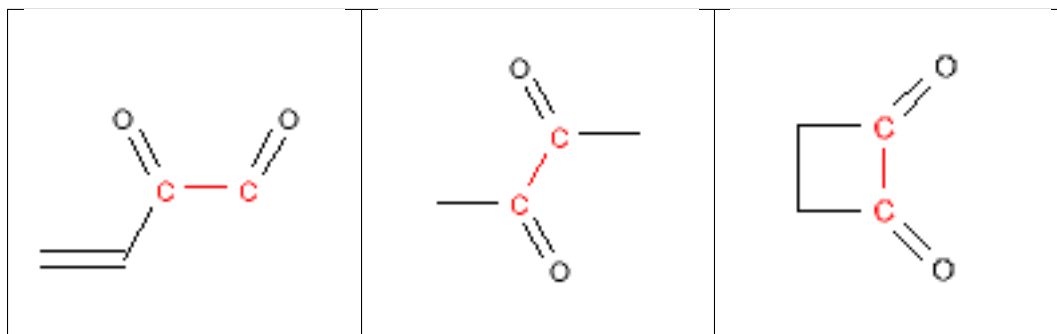
NS_beta_halothyl



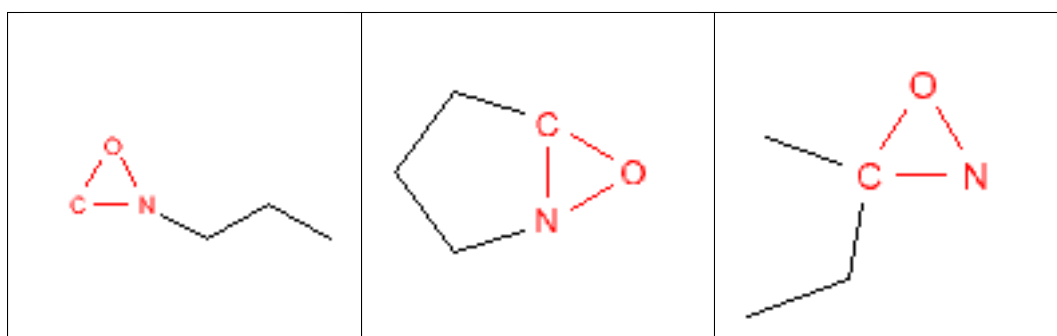
organometallic



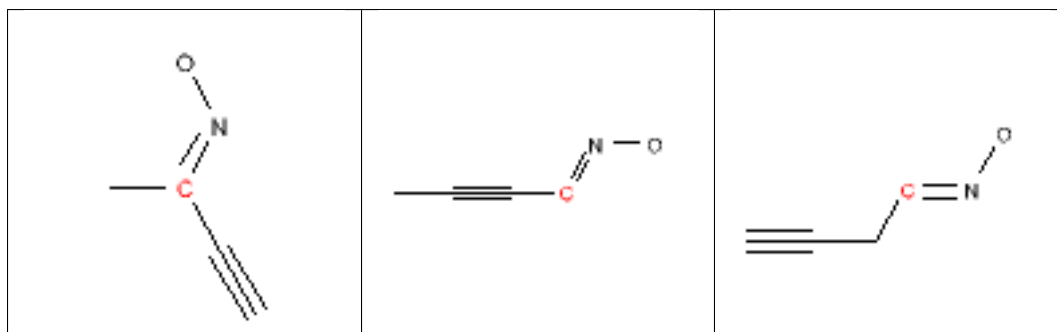
oxalyl



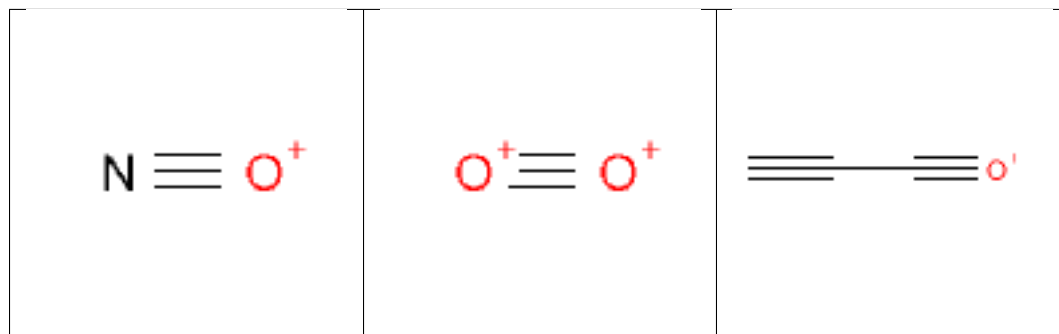
oxaziridine



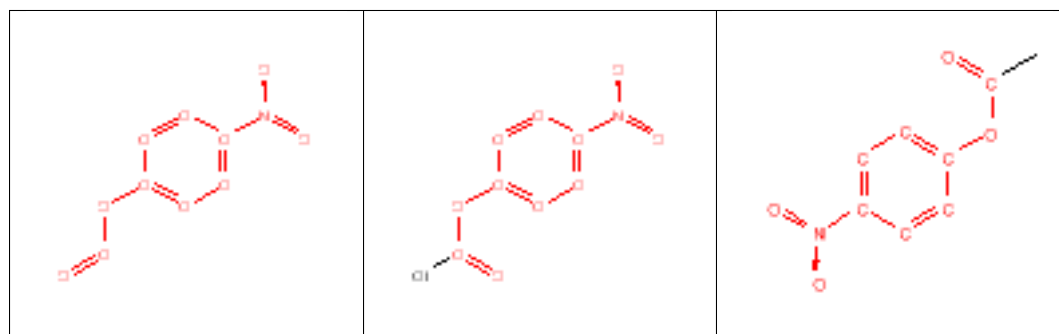
oxime



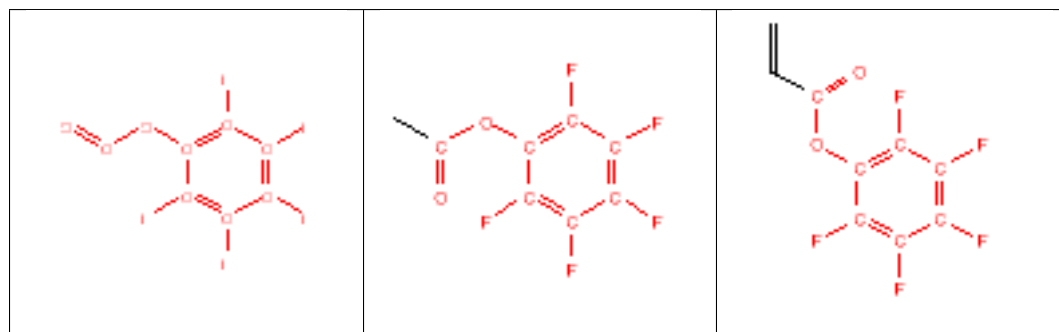
oxygen_cation



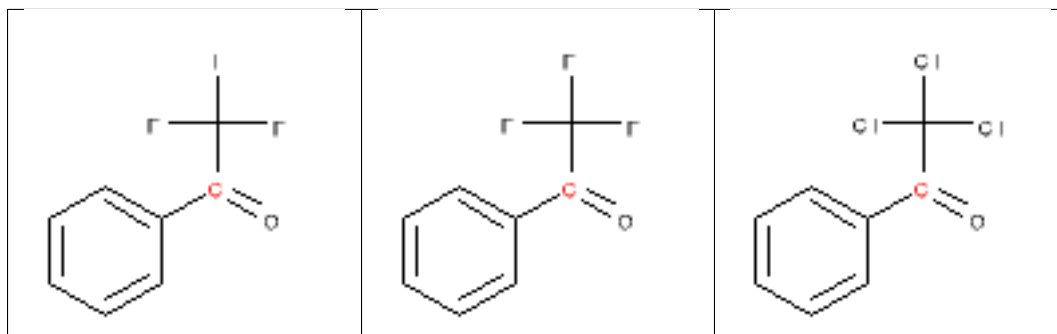
paranitrophenyl_esters



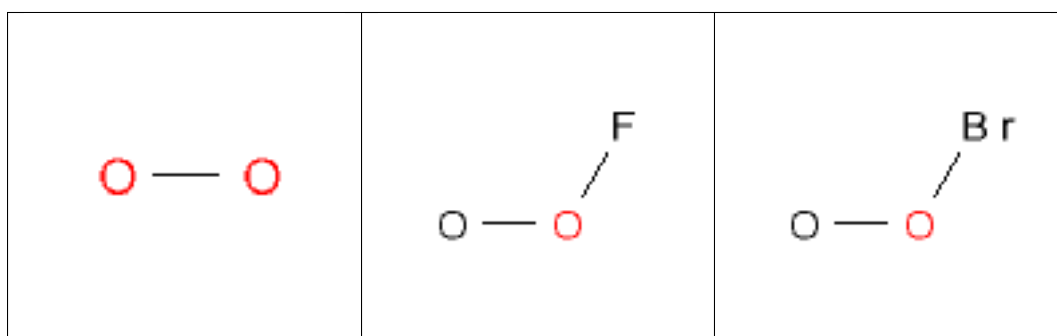
pentafluorophenyl_esters



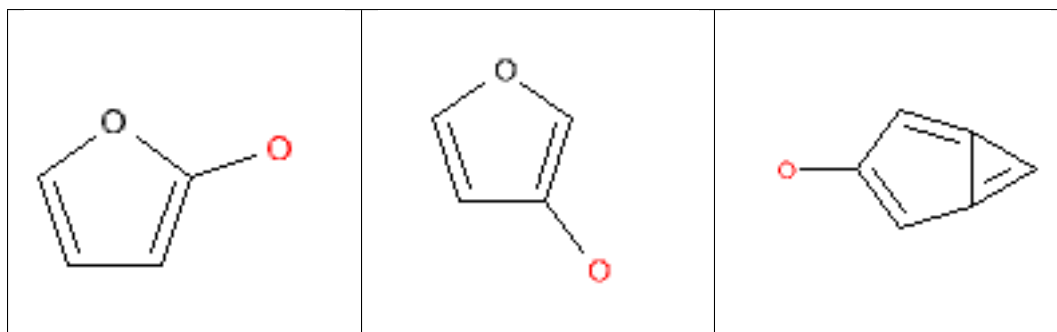
perhalo_ketone



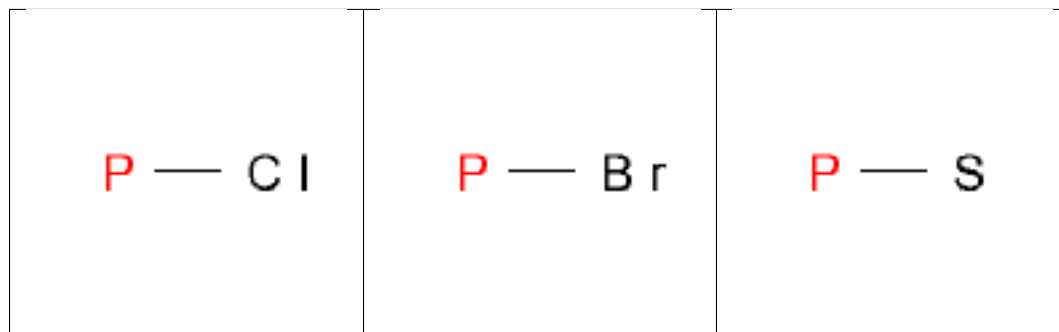
peroxide



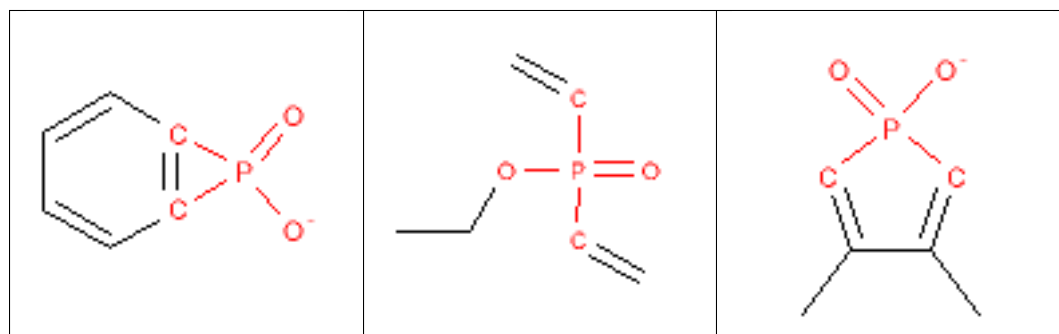
phenol



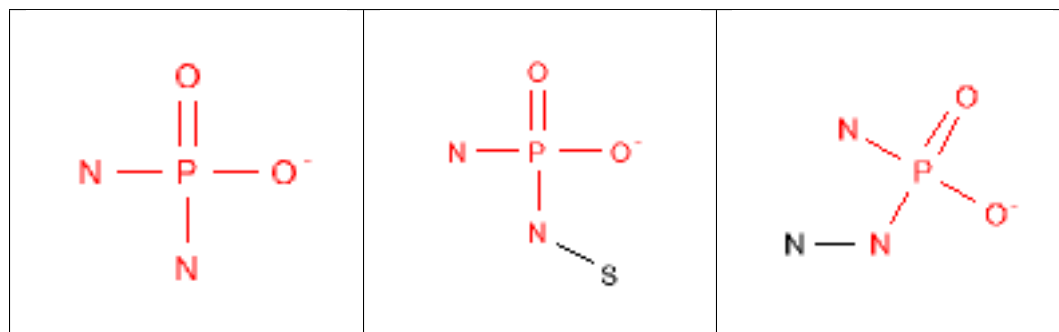
phosphanes



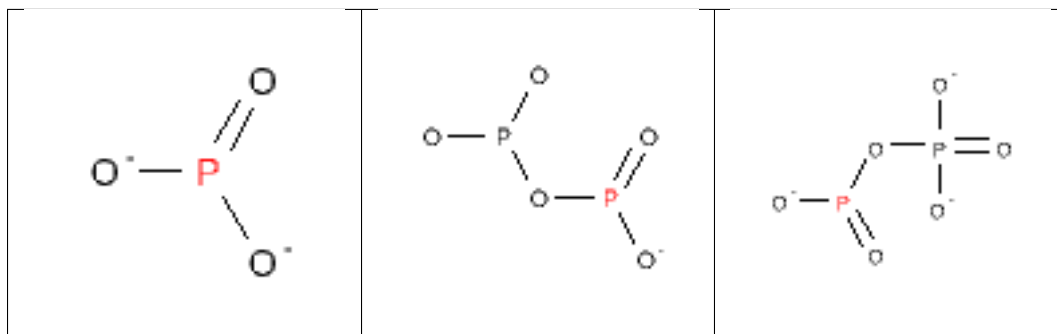
phosphinic_acid



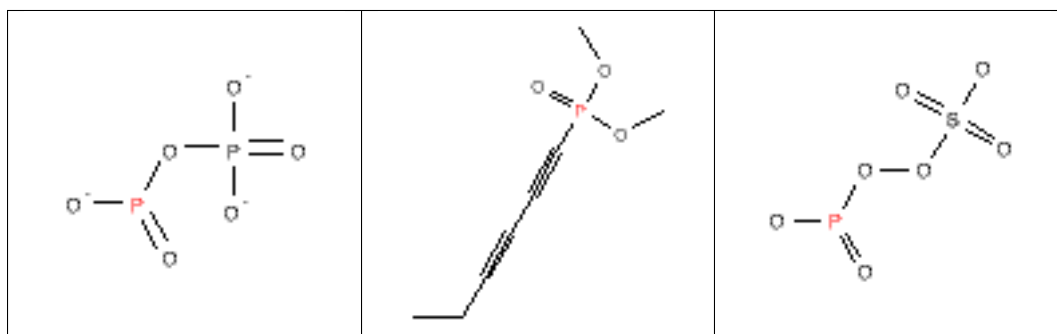
phosphonamide



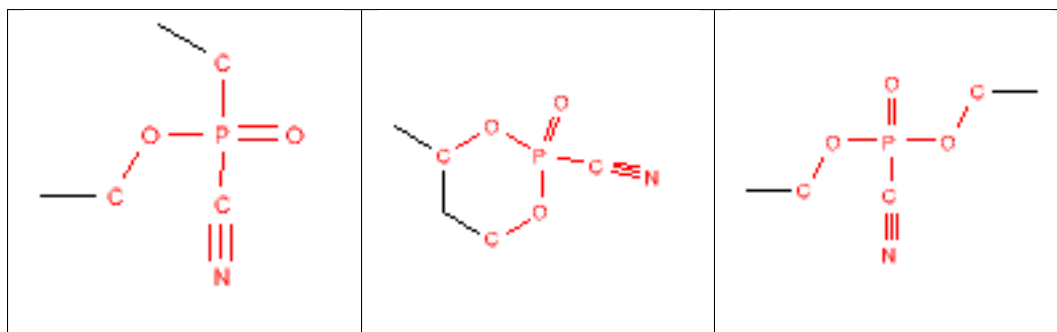
phosphonic_acid



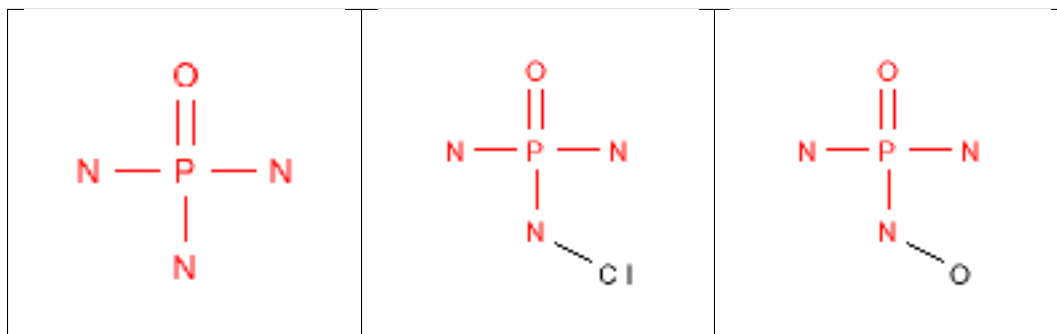
phosphonic_ester



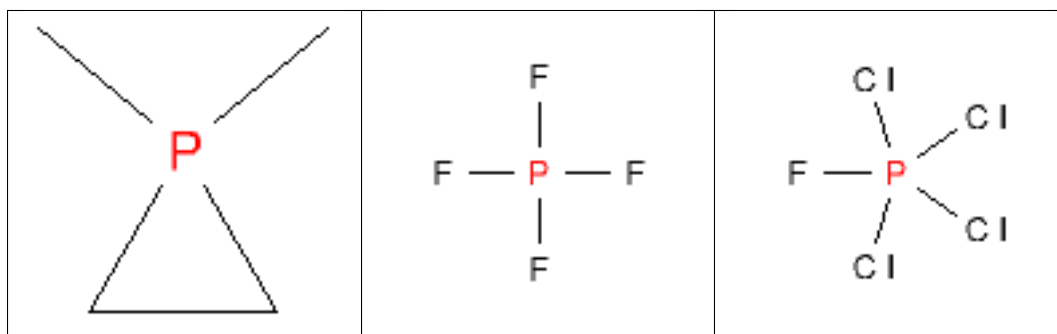
phosphonylnitrile



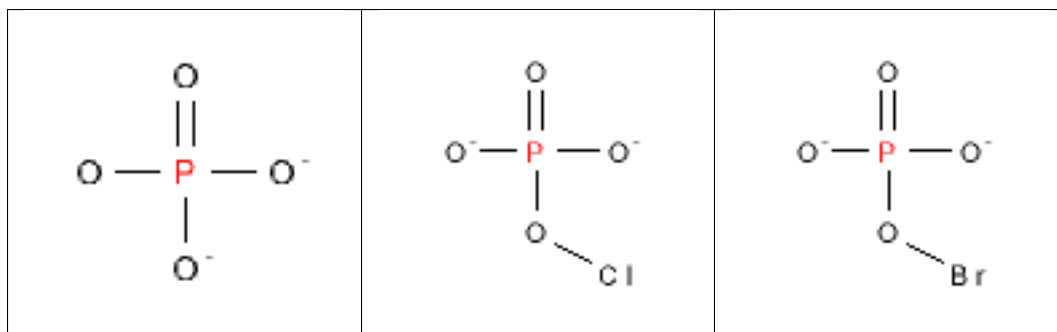
phosphoramides



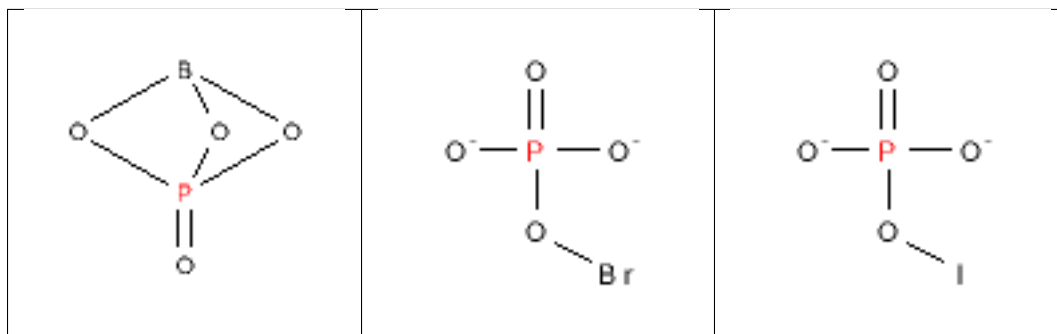
phosphoranes



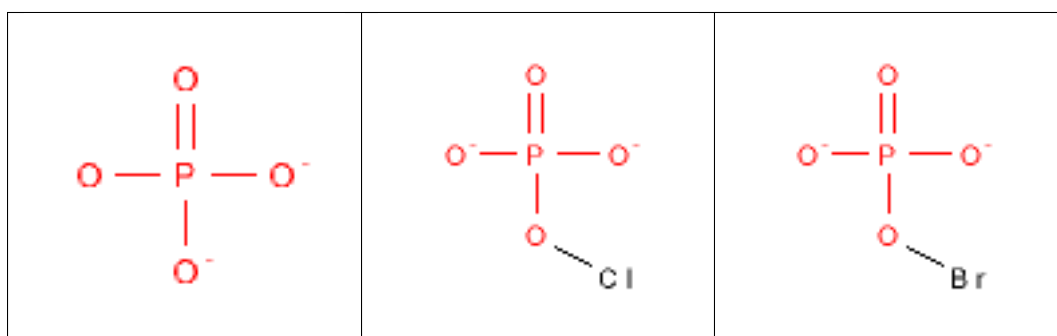
phosphoric_acid



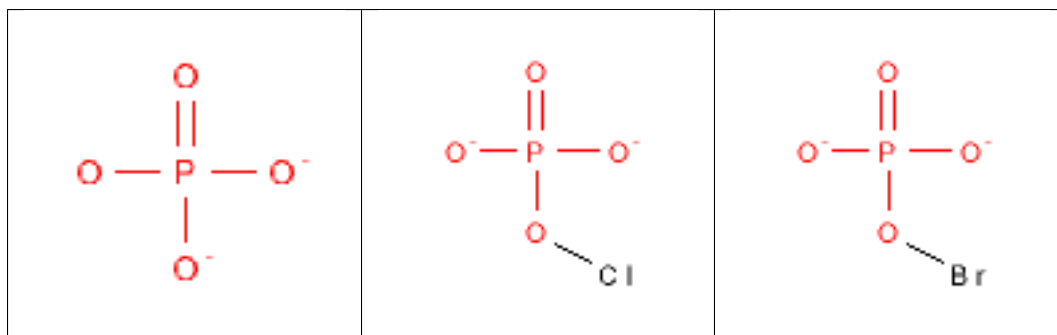
phosphoric_ester



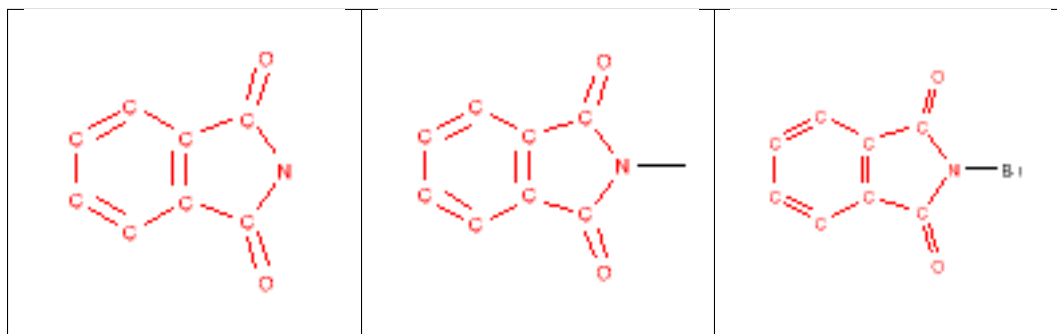
phosphoryl



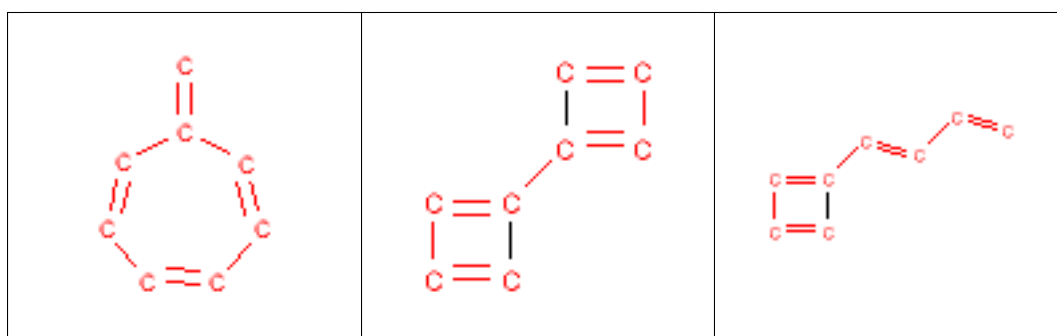
phosphoryl



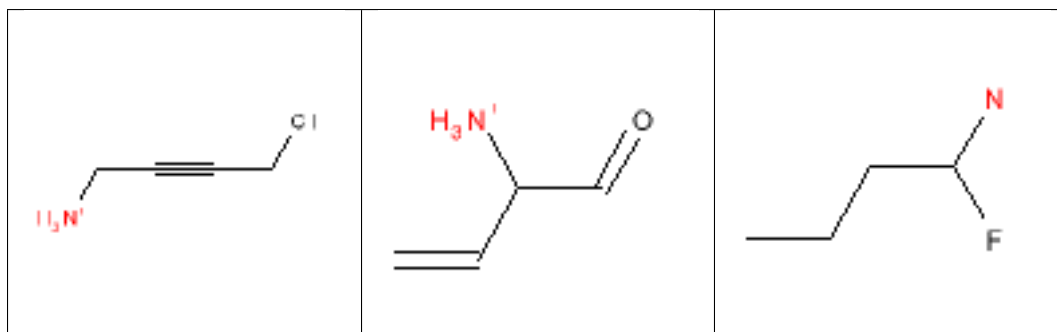
phthalimides_PHT



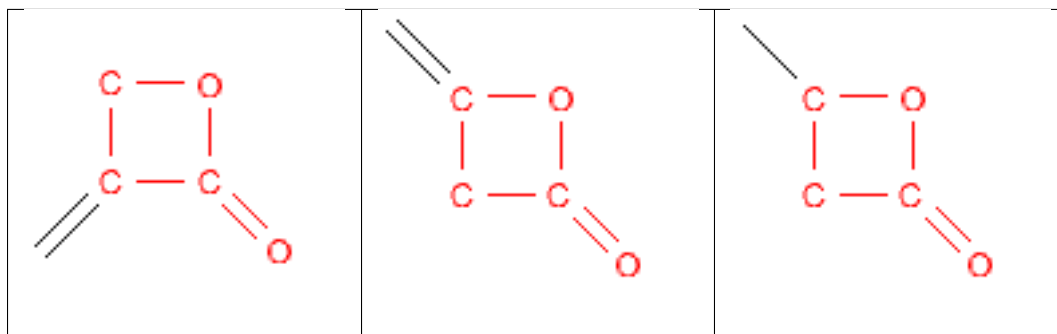
polyenes



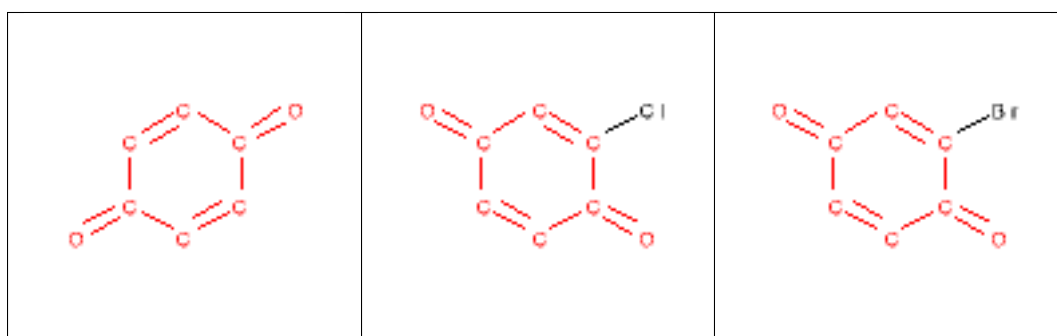
primary_amine



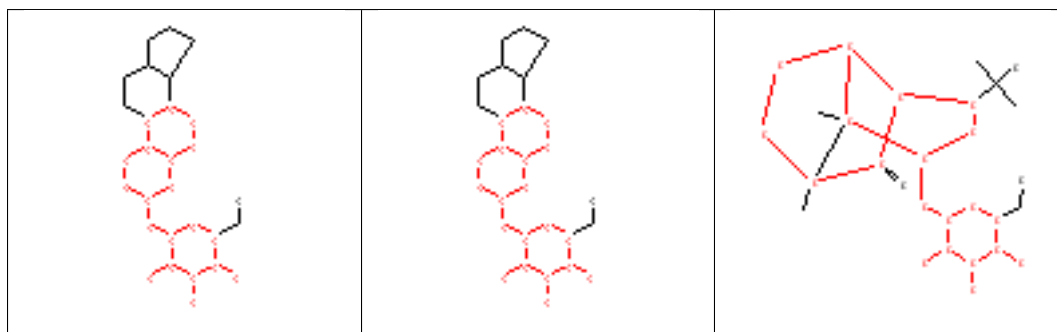
propiolactones



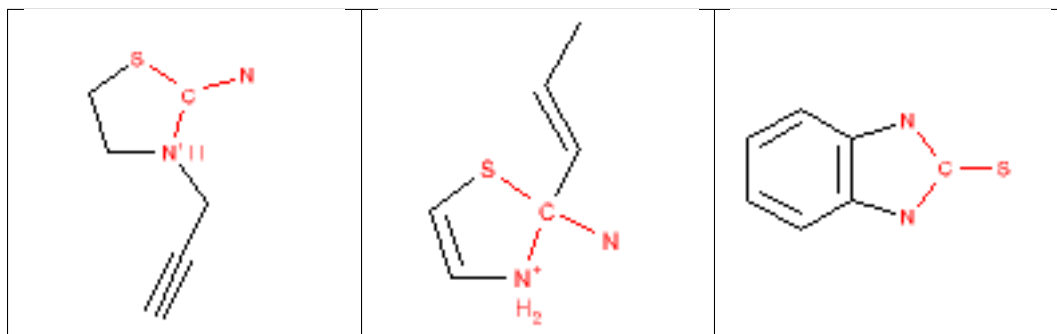
quinone



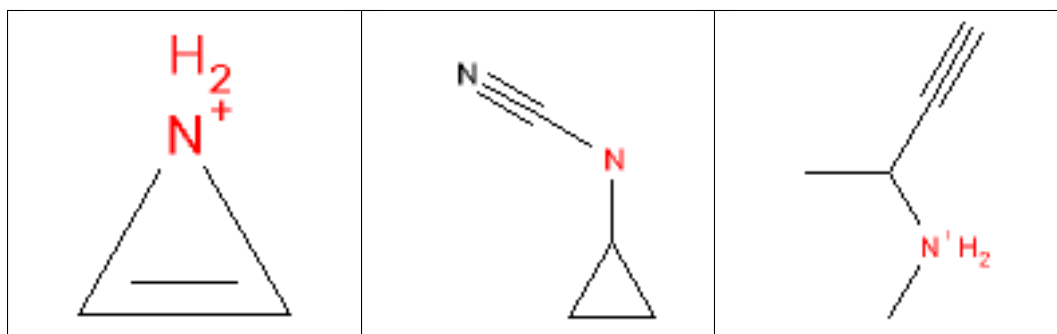
saponin_derivatives



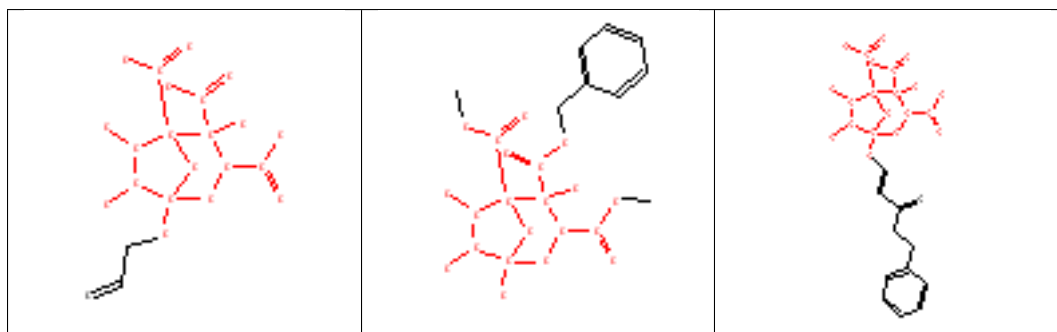
SCN2



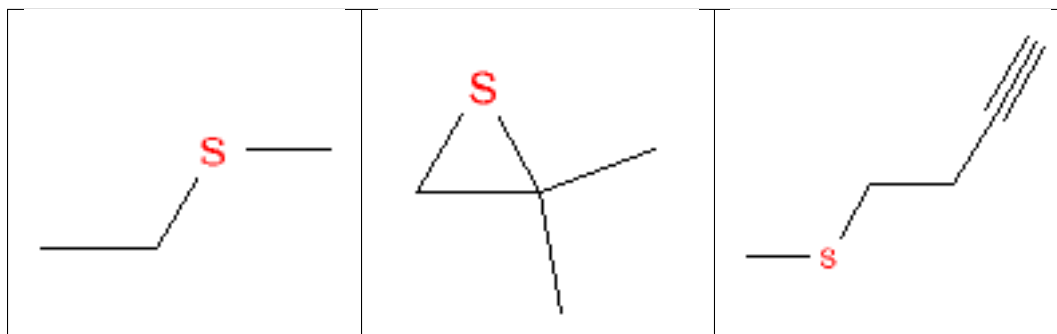
secondary_amine



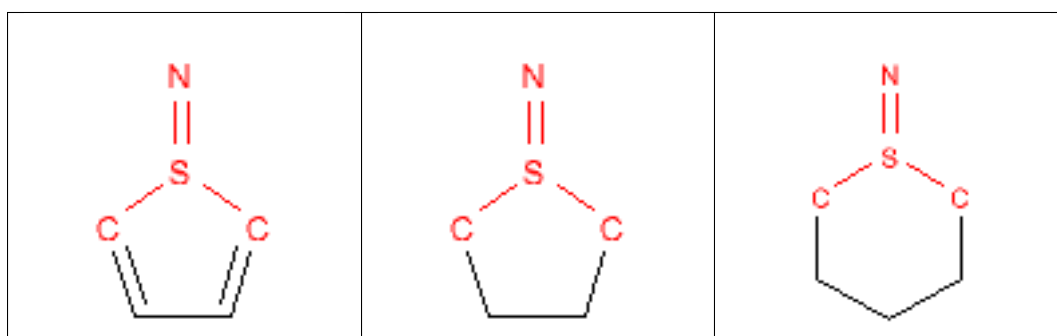
squalestatin_derivatives



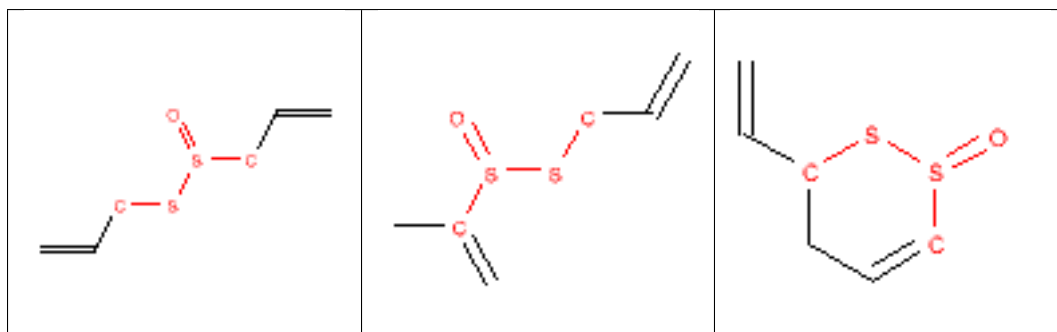
sulfide



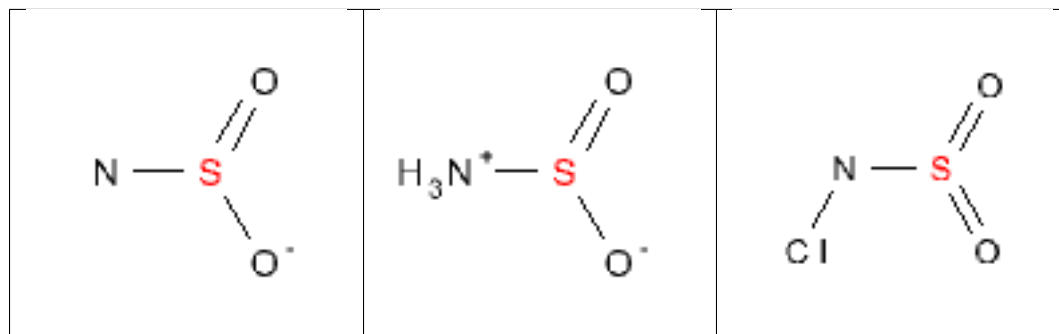
sulfinimine



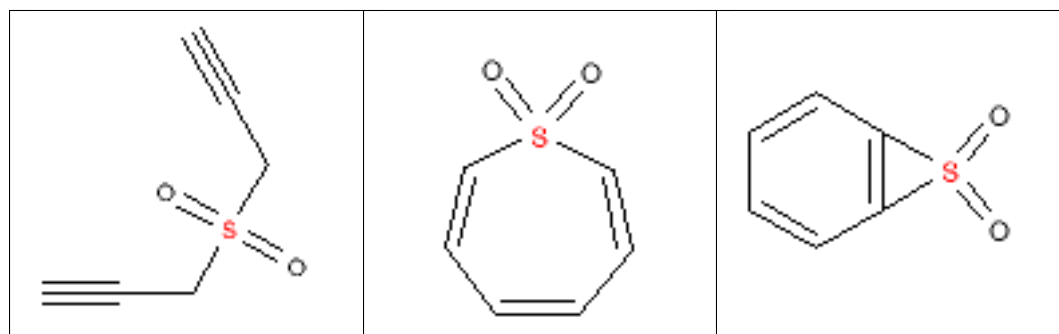
sulfinylthio



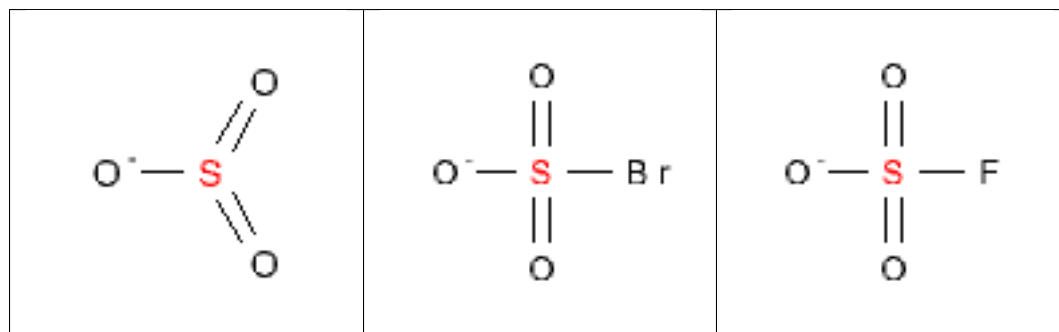
sulfonamide



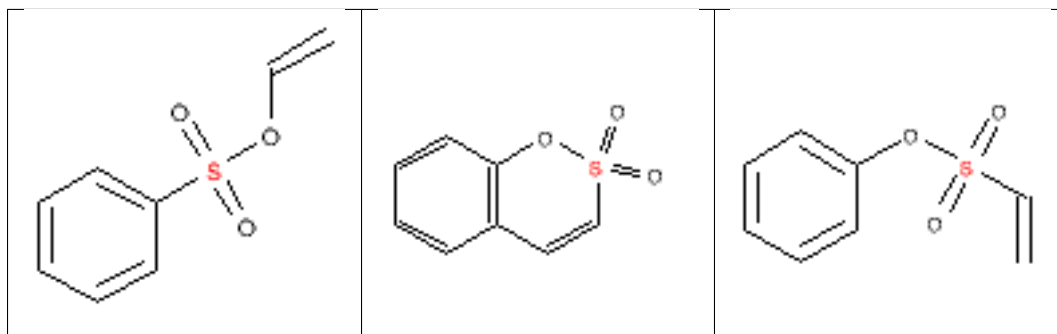
sulfone



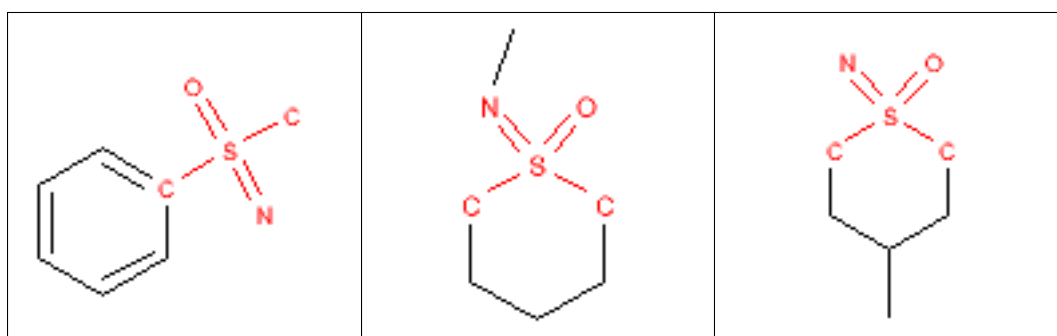
sulfonic_acid



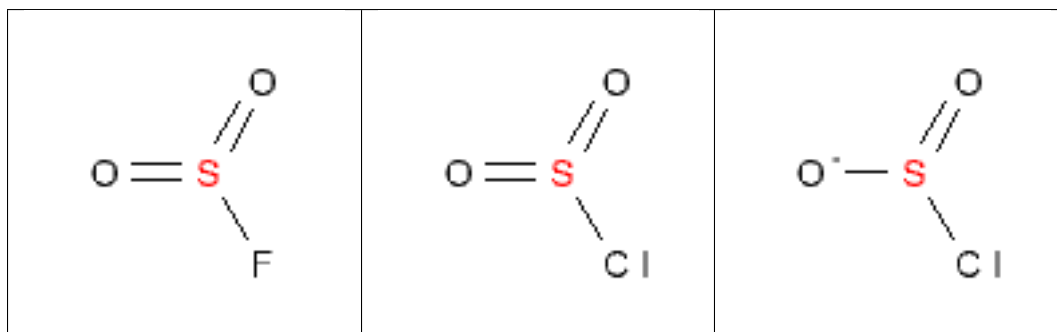
sulfonic_ester



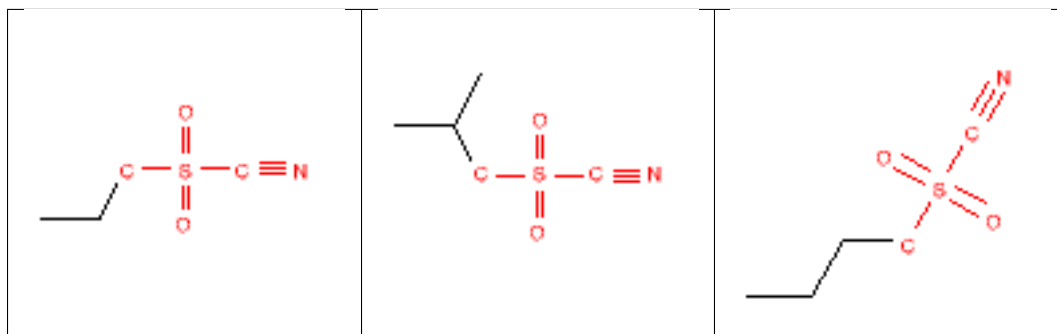
sulfonimine



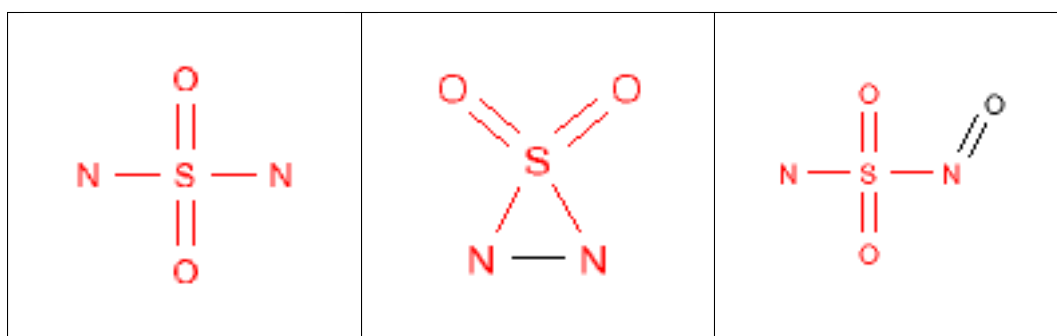
sulfonyl_halide



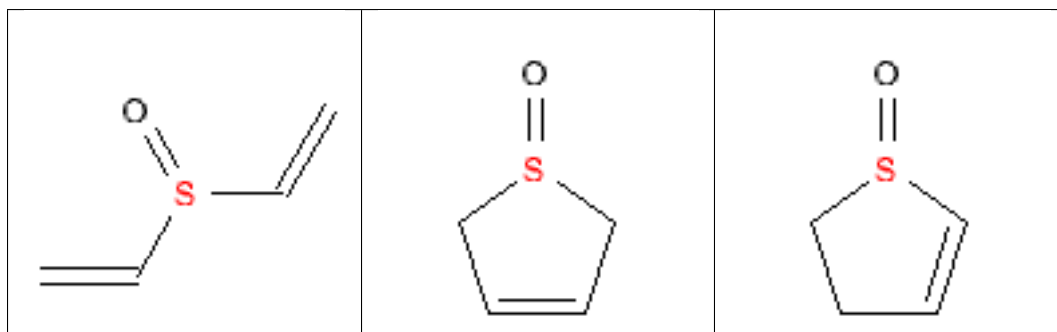
sulfonylnitrile



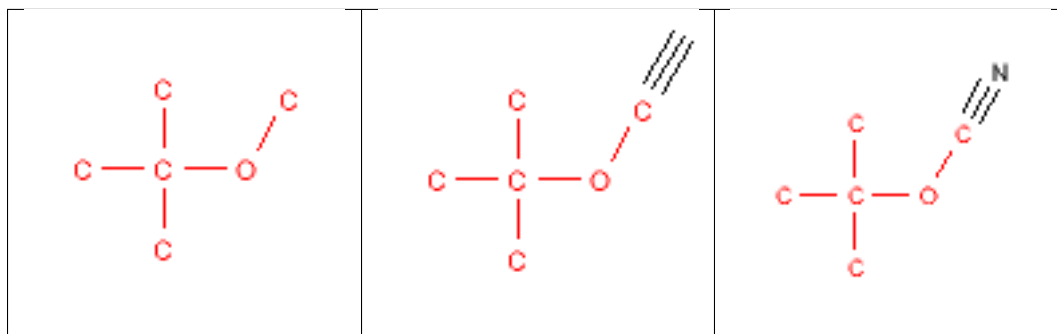
sulfonylurea



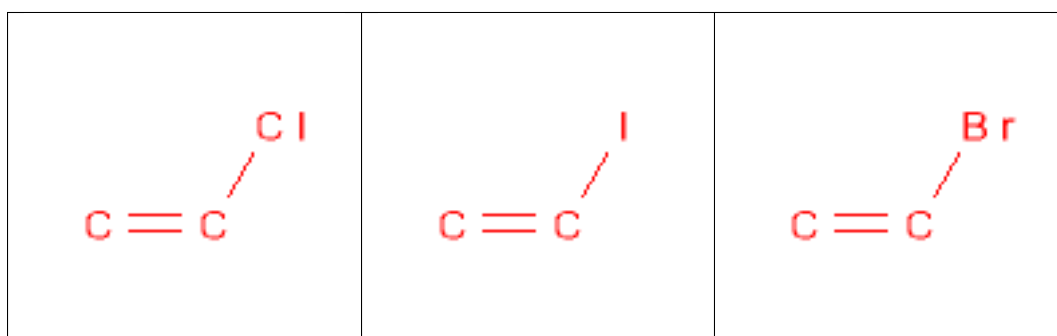
sulfoxide



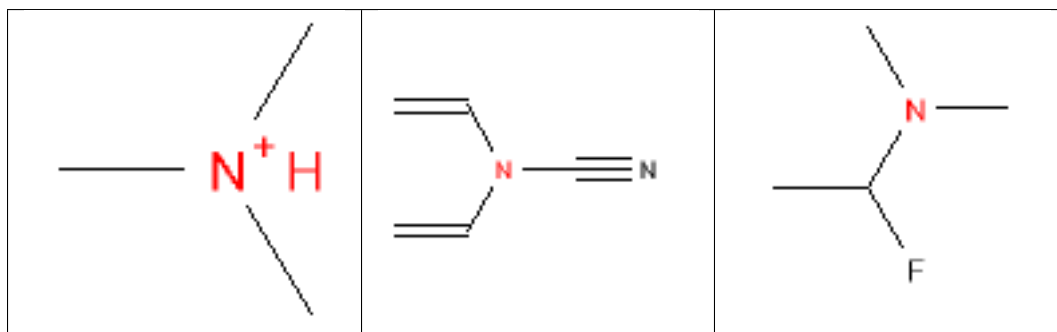
t_butyl_ether



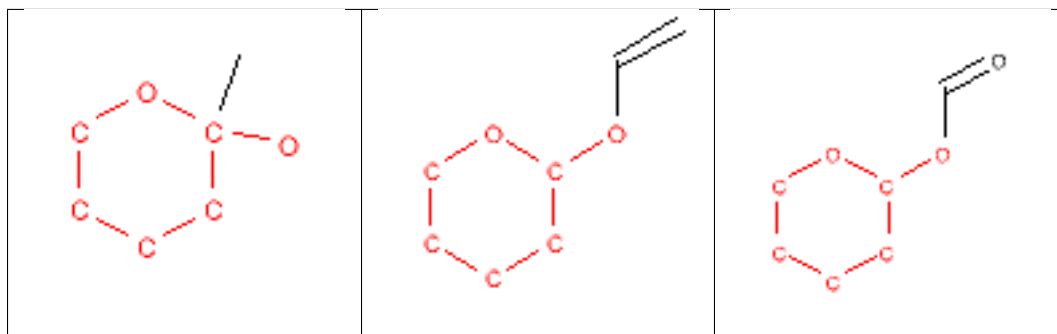
terminal_vinyl



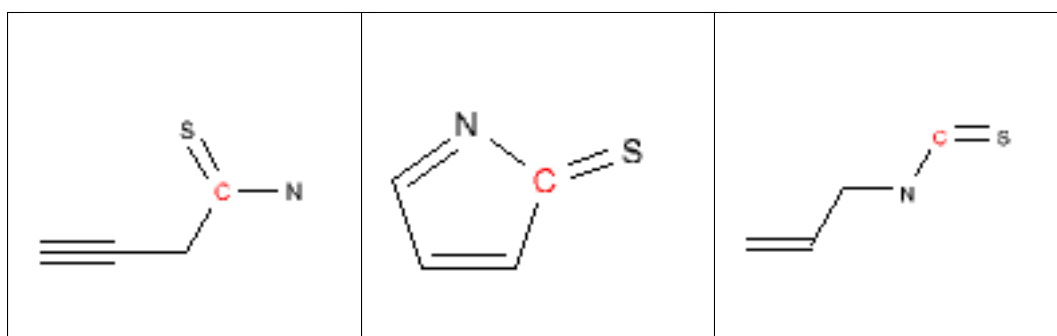
tertiary_amine



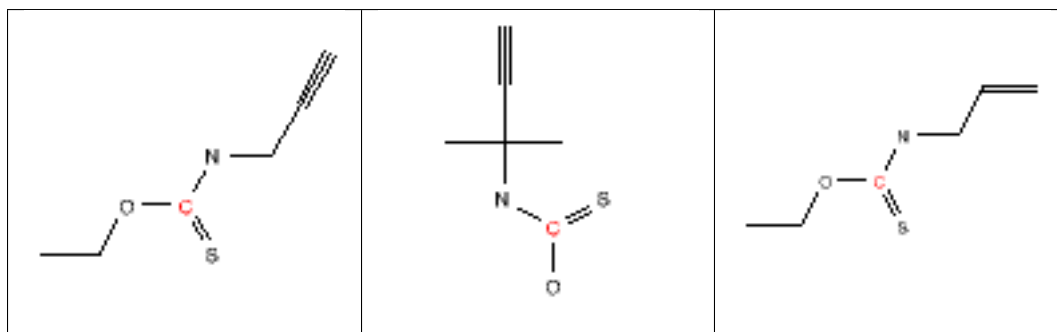
tetrahydropyran_THP



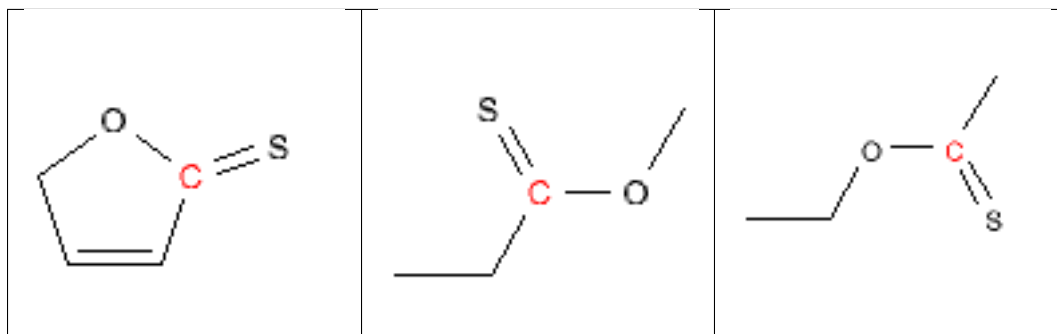
thioamide



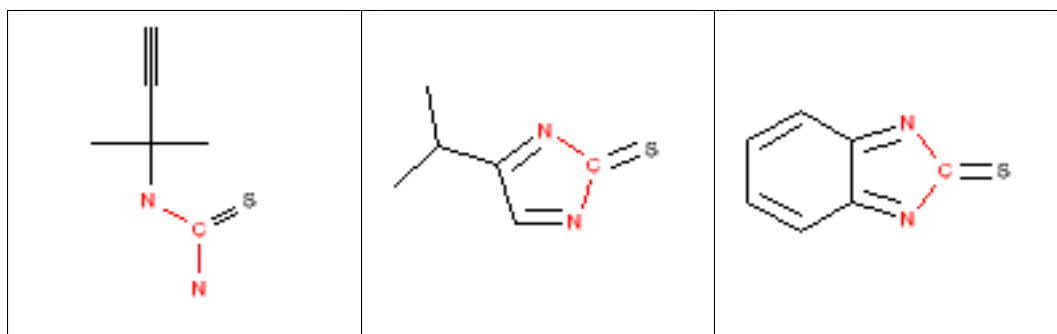
thiocarbamate



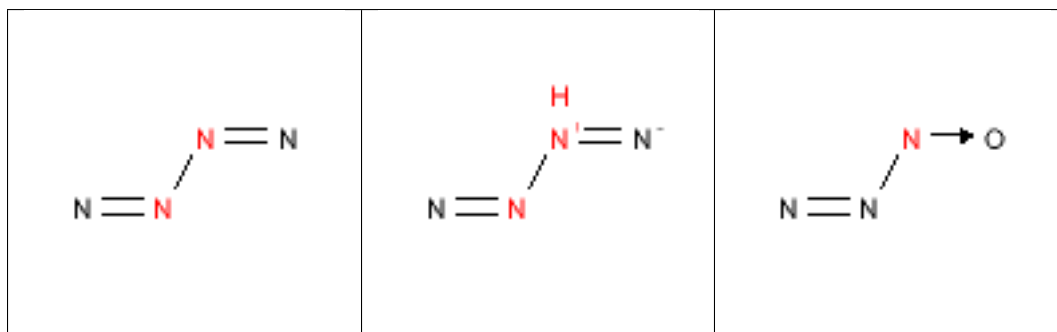
thioester



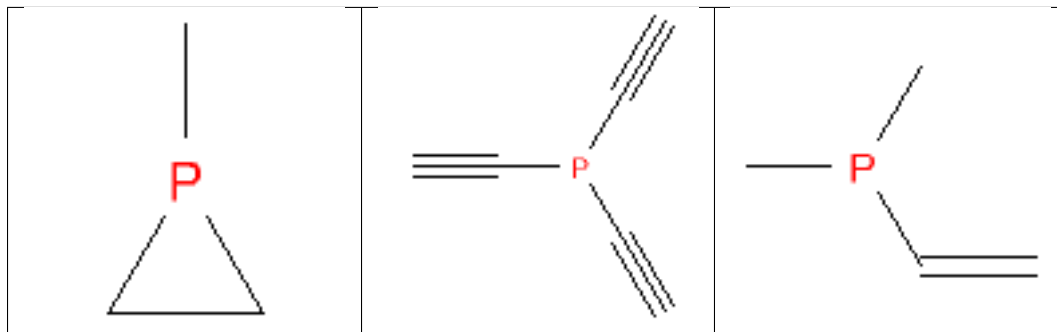
thiourea



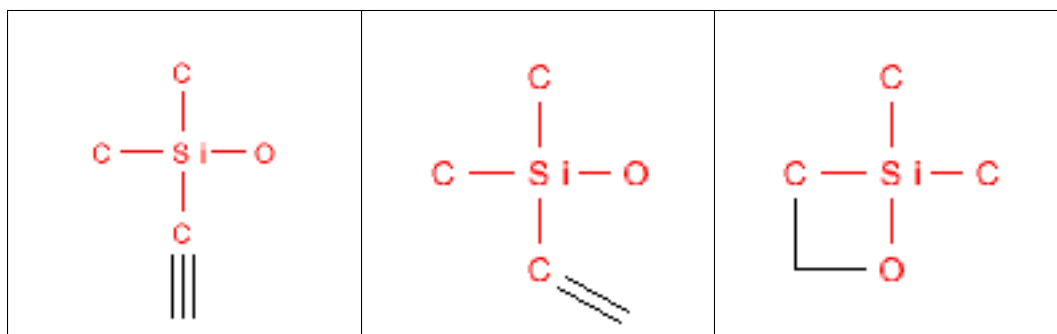
triazine



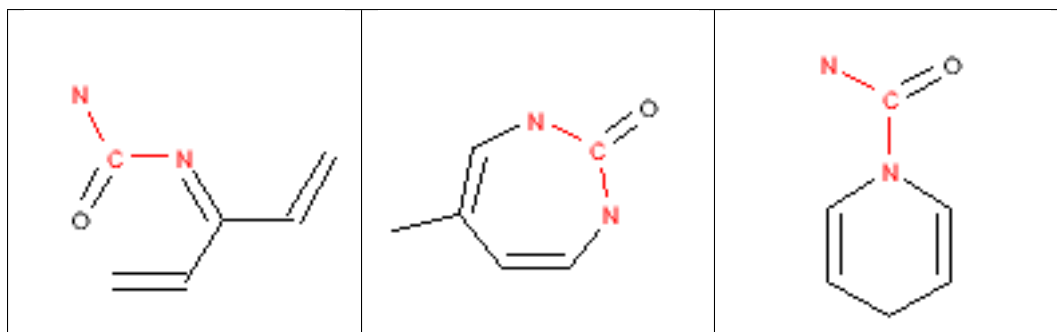
tricarbo_phosphene



trimethylsilyl_TMS



urea



2.4.3 New Rules

New rules specify additional functional groups or substructures that may be used. They must specify a substructure definition in the form of a SMARTS in addition to the substructure name and maximum limit. For example:

```
NEWRULE norborane 1 C1CC2CCC1C2
```

The first field is the NEWRULE keyword. The second field defines the name associated with the substructure (primarily for logging purposes). The third field indicates the maximum number of the substructure that can be allowed. The fourth field is the SMARTS string for the substructure, norborane in this case. This example rule would indicate that

molecules with a single norborane substructure would be allowable, but that those with 2 or more norboranes would be eliminated.

New rules that have a name that is identical with one of the original rules take precedence over the original rule.

2.4.4 Selection Statements

The select statement allows a filter file to specify the required number of substructures in order to be able to pass the filter. These statements are similar to new rules except that they list a required range for passing the filter rather than the range for failing to pass the filter. For example:

```
SELECT amine 1 1 [N;!$(*-*[!#6;!#1]);!$(*-a);!$(*,#*)]
```

The first field is the `SELECT` keyword. The second field indicates the name for the selection (again for logging purposes). The third field is the minimum number of substructures required to be in the molecule. The fourth field is the maximum number of substructures allowed in the molecule. The fifth field is the substructure defined by a SMARTS pattern. The example requires that molecules contain exactly one amine. Currently, only a single `SELECT` statement is allowed in the filter file. Any complex boolean substructure statements can be incorporated directly into the SMARTS. If multiple `SELECT` statements occur in a filter file, only the final one will be applied.

2.5 Bibliography

TOOLKIT

3.1 Filter Object

All filtering operations are controlled via the `OEFilter` object. The `OEFilter` object is typically configured with a specified filter and then applied iteratively over a molecule file. Listing 1 demonstrates configuring the `OEFilter` object with the default *lead-like filter* and then writes out the molecules that pass the filter. The `OEFilter` object's `OEFilter::operator()` method is used to test whether the molecule passes the filter.

Note: The molecule will also be altered by all the specified *Filter Preprocessing* steps.

Listing 1: Basic filtering for lead-like molecules

```
/*  
Copyright (C) 2009 OpenEye Scientific Software, Inc.  
*****  
Filter a molecule file for "Lead-like" molecules  
*****  
*/  
#include "openeye.h"  
#include "oesystem.h"  
#include "oechem.h"  
#include "oemolprop.h"  
  
using namespace OEMolProp;  
using namespace OESystem;  
using namespace OEChem;  
  
int main(int argc, char *argv[])  
{  
    if (argc != 3)  
        OThrow.Usage("%s <input> <output>", argv[0]);  
  
    oemolistream ifs;  
    if (!ifs.open(argv[1]))  
        OThrow.Fatal("Unable to open %s", argv[1]);  
  
    oemolostream ofs;  
    if (!ofs.open(argv[2]))  
        OThrow.Fatal("Unable to create %s", argv[2]);  
  
    OEFilter filter(OEFilterType::Lead);
```

```

OEGraphMol mol;
while (OEReadMolecule(ifs, mol))
    if (filter(mol))
        OEWwriteMolecule(ofs, mol);

return 0;
}

```

By default the `OEFilter::operator()` method will print information to `OETHrow` about every molecule passed to it. The following is the format for what is printed.

```
[Isomeric SMILES]\t[Title],[Pass|Reason for failure]
```

The following is an example of this output:

```

CC1=CC(=O)C=CC1=O          NSC 1,Minimum atom count(10) not reached: 9
c1ccc2c(c1)nc(s2)SSc3nc4ccccc4s3      NSC 2,Maximum disulfide(0) exceeded: 1
c1c(cc(c(c1[N+](=O)[O-])[O-])Cl)[N+](=O)[O-]  NSC 3,Maximum heteroatom to carbon ratio(1.100000) ex
c1c(sc(n1)N)[N+](=O)[O-]          NSC 4,Minimum atom count(10) not reached: 9
c1ccc2c(c1)C(=O)c3ccc(cc3C2=O)N NSC 5,Maximum dye(0) exceeded: 2
c1ccc(c(c1)c2c3ccc(c(c3oc-4c(c(=O)ccc24)Br)Br)O)C(=O)[O-] NSC 6,Maximum atom count(25) exceeded
C[NH+](C)C1=C(C(=O)c2ccccc2C1=O)C1      NSC 7,Maximum alkyl_halide(0) exceeded: 1
Cc1ccc2c(c1[N+](=O)[O-])C(=O)c3ccccc3C2=O NSC 8,Maximum nitro(0) exceeded: 1
CC(C)(C)c1cc(c(cc1O)C(C)(C)C)O NSC 11,Pass
CC1=NN(C(=O)C1)c2ccccc2 NSC 12,Pass

```

3.1.1 Quiet Filtering

The above log output can be superfluous when using the `OEFilter` object in a more complex program. Since all the log output is written to the `OETHrow` object the verbosity level can be lowered by using the `OETHrow.SetLevel` method. Listing 2 demonstrates setting the `OETHrow` error level to `OErrorLevel::Warning`. This will only allow messages at the level of `Warning` or above through, thereby silencing the `OEFilter` object's logging output.

Listing 2: Silencing the OEFilter logging messages

```

/*****
Copyright (C) 2009 OpenEye Scientific Software, Inc.
*****/
Quietly filter a molecule file for "Lead-like" molecules
*****/
#include "openeye.h"
#include "oesystem.h"
#include "oechem.h"
#include "oemolprop.h"

using namespace OEMolProp;
using namespace OESystem;
using namespace OEChem;

int main(int argc, char *argv[])
{
    if (argc != 3)

```

```

    OThrow.Usage("%s <input> <output>", argv[0]);

    oemolistream ifs;
    if (!ifs.open(argv[1]))
        OThrow.Fatal("Unable to open %s", argv[1]);

    oemolostream ofs;
    if (!ofs.open(argv[2]))
        OThrow.Fatal("Unable to create %s", argv[2]);

    OEFilter filter(OEFilterType::Lead);

    OThrow.SetLevel(OEErrorLevel::Warning);

    OEGraphMol mol;
    while (OEReadMolecule(ifs, mol))
        if (filter(mol))
            OWriteMolecule(ofs, mol);

    return 0;
}

```

3.1.2 Molecular Property Table

The `OEFilter` object allows for the calculation of all the molecular properties it uses during the filtering process without actually applying the filter. You may find this useful for caching the `OEFilter` object results into a database. The `OEFilter::SetTable` method can be used to specify where to write a tab-delimited table of every property in the associated *filter file*. Listing 3 demonstrates how to write the tabular output to standard out.

Listing 3: Generating tabular output of all molecular properties

```

/*****
Copyright (C) 2009 OpenEye Scientific Software, Inc.
*****/
Generate a tabular output of molecular properties
*****/
#include "openeye.h"
#include "oesystem.h"
#include "oechem.h"
#include "oemolprop.h"

using namespace OEMolProp;
using namespace OESystem;
using namespace OEChem;

int main(int argc, char *argv[])
{
    if (argc != 2)
        OThrow.Usage("%s <input>", argv[0]);

    oemolistream ifs;
    if (!ifs.open(argv[1]))
        OThrow.Fatal("Unable to open %s", argv[1]);

```

```

OEFilter filter(OEFilterType::Lead);
OEThrow.SetLevel(OEErrorLevel::Warning);

bool pwned = false;
filter.SetTable(&OEPlatform::oeout, pwned);

OEGraphMol mol;
while (OEReadMolecule(ifs, mol))
    filter(mol);

return 0;
}

```

Note: A tab-delimited file format was chosen to intergrate better with unix commandline utilities. For example, this allows for quick and easy filter experimentation with the `awk` command line utility. The following snippet will print the number of molecules (plus one for the header) with a molecular weight greater than 200.

```

> molproptable drugs.sdf > drugs.txt
> cat drugs.txt | awk -F\t '{if ($12 > 200) { print $12 }}' | wc -l

```

Furthermore, all database programs have utilities for importing tab-delimited files. Loading the filter results into a third-party database would provide very speedy filter experimentation since the properties would only have to be calculated once and then cached in the database.

3.1.3 Specific Molecular Properties

The `OEFilter` object table can also be used to retrieve specific molecular properties or filtering terms for which a free function does not exist. The number of molecular properties MolPropTK calculates can be staggering. Furthermore, some properties are more experimental than others. For these reasons *free functions* are not provided for everything MolPropTK can calculate. Listing 4 demonstrates how to extract the number of Lipinski violations from the `OEFilter` object molecular property table by writing the table to a `oesstream`.

Warning: The exact number of fields the `OEFilter` object outputs depends on the *filter file* being used. The field will only be output if the filter rule is turned on. That is why the example in Listing 4 will search the header line for the position of the “Lipinski violations” field. This key-value lookup is recommended for extracting specific fields from the filter table.

Listing 4: Extract the number of Lipinski violations from the table output

```

/*****
Copyright (C) 2009 OpenEye Scientific Software, Inc.
*****/
Extract the number of Lipinski violations from the table output
*****/
#include "openeye.h"
#include "oesystem.h"
#include "oechem.h"
#include "oemolprop.h"

using namespace OEPlatform;
using namespace OEMolProp;
using namespace OESystem;

```

```

using namespace OEChem;

int main(int argc, char *argv[])
{
    if (argc != 2)
        OThrow.Usage("%s <input>", argv[0]);

    oemolistream ifs;
    if (!ifs.open(argv[1]))
        OThrow.Fatal("Unable to open %s", argv[1]);

    OEFilter filter(OEFilterType::Lead);
    OThrow.SetLevel(OEErrorLevel::Warning);

    oeosstream ostr;
    bool pwned = false;
    filter.SetTable(&ostr, pwned);

    std::vector<std::string> fields;
    OESTringTokenize(fields, ostr.str(), std::string("\t"));
    ostr.clear(); // remove the header row from the stream

    size_t fieldidx = find(fields.begin(), fields.end(),
                          std::string("Lipinski violations")) - fields.begin();

    OEGraphMol mol;
    while (OEReadMolecule(ifs, mol))
    {
        filter(mol);

        OESTringTokenize(fields, ostr.str());
        ostr.clear(); // remove this row from the stream

        std::cout << mol.GetTitle() << " " << fields[fieldidx] << std::endl;
    }

    return 0;
}

```

3.1.4 SD Data Molecular Properties

The molecular properties `OEFilter` calculates can be attached as SD data to the molecule. Listing 5 demonstrates how to use the `OEFilter::SetSDTag` method to attach the calculated molecular properties to molecules passed to the filter object.

It is very important to remember that SD tag data is highly correlated with the *tabular data* functionality. The SD data tag names are exactly the same as the name found in the table header. The SD data values are exactly what is output in the data table. The only exceptions are the “SMILES”, “Name”, and “Filter” columns since this information can be obtained other ways.

Warning: Only the properties up to the failing property will be attached by default. Use `OEFilter::SetTable` to force every property to be attached as SD data. The same rule applies to SD data as *tabular output data*, only the properties specified in the *filter file* will be attached as SD data. Listing 5 demonstrates how to set the table output to the `oenul` output stream to allow all properties to be attached as SD data, but not actually write the tabular data anywhere.

Listing 5: Attach molecular properties as SD data

```
/*  
Copyright (C) 2009 OpenEye Scientific Software, Inc.  
*****  
Attach molecular properties as SD data  
*****  
*/  
#include "openeye.h"  
#include "oesystem.h"  
#include "oechem.h"  
#include "oemolprop.h"  
  
using namespace OEMolProp;  
using namespace OESystem;  
using namespace OEChem;  
  
int main(int argc, char *argv[])  
{  
    if (argc != 3)  
        OEThrow.Usage("%s <input> <output>", argv[0]);  
  
    oemolistream ifs;  
    if (!ifs.open(argv[1]))  
        OEThrow.Fatal("Unable to open %s", argv[1]);  
  
    oemolostream ofs;  
    if (!ofs.open(argv[2]))  
        OEThrow.Fatal("Unable to create %s", argv[2]);  
  
    unsigned int fmt = ofs.GetFormat();  
    if (fmt != OEFormat::SDF && fmt != OEFormat::OEB)  
        OEThrow.Fatal("Only SD and OEB formats preserve SD data");  
  
    OEFilter filter(OEFilterType::Lead);  
    OEThrow.SetLevel(OEErrorLevel::Warning);  
  
    bool pwned = false;  
    filter.SetTable(&OEPlatform::oenul, pwned);  
    filter.SetSDTag(true);  
  
    OEGraphMol mol;  
    while (OEReadMolecule(ifs, mol))  
    {  
        filter(mol);  
        OEWwriteMolecule(ofs, mol);  
    }  
  
    return 0;  
}
```

4.1 OEMolProp Classes

4.1.1 OEFilter

`class OEFilter`

This class represents a set of filters as initialized from a *filter file*. The primary use of this class is `OEFilter::operator()` method which will return whether the molecule passes or fails the filter.

Constructors

```
OEFilter(unsigned int type=OEFilterType::Default)
```

The `OEFilter` object is usually constructed from an unsigned integer from the `OEFilterType` constant namespace. The default constructor will initialize the object to the `OEFilterType::Default` filter. The default filter is an alias for the `OEFilterType::BlockBuster` filter.

See Also:

The *Variations of Filters* section.

```
OEFilter(OEPlatform::oeistream &filterstr)
```

Fine grained filter control is provided by allowing a custom filter file to be passed to the `OEFilter` object as the `filterstr` argument.

```
OEFilter(const OEFilter &filter)
```

The `OEFilter` object can be copy constructed from another `OEFilter` object.

operator=

```
OEFilter &operator=(const OEFilter &filter)
```

Assignment operator to allow copying a `OEFilter` object.

operator()

```
bool operator() (OEChem::OEMolBase &mol)
```

Returns whether the molecule `mol` passes the `OEFilter` object's set of filter rules. This function has been optimized by applying the filter rules in order of how expensive they are to calculate. For example, molecular weight is much easier to calculate than any of the functional group rules.

However, if an output stream has been specified using the `OEFilter::SetTable` method all the filter rules will be checked. This is the only way to ensure that all the fields are filled in the tabular data output to the stream specified in `SetTable` method.

Note: The molecule may be altered by the steps described in the *Filter Preprocessing* chapter.

See Also:

This method will write logging data to `OEThrow` as described in this *section*.

operator bool

```
operator bool() const
```

Returns whether the `OEFilter` object is in a valid state ready to accept molecules to its `OEFilter::operator()` method.

Note: It is nearly impossible for an end user to create an invalid filter. This method is here for future API design to allow for greater programmatic control of the `OEFilter` object.

AddNormalizationRule

```
bool AddNormalizationRule(const OEChem::OEUniMolecularRxn &rxn)
```

Add a `OEUniMolecularRxn` to apply to the molecule. Special note should be taken about what preprocessing has already occurred on the molecule before generic normalizations are applied as described by the *Filter Preprocessing* chapter.

See Also:

The *OEChem* manual has a chapter devoted to reactions.

ClearNormalizationRules

```
void ClearNormalizationRules()
```

Remove all normalizations previously added with `OEFilter::AddNormalizationRule`.

GetFlagTableFailures

```
bool GetFlagTableFailures() const
```

Return whether values in *tabular data output* are flagged with an asterisk marking them as outside the valid range for that filter rule. This is useful for figuring out why particular molecules have failed a given filter.

See Also:`OEFilter::SetFlagTableFailures`**GetSDTag**`bool GetSDTag() const`

Return whether to attach every molecular property checked by the filter as SD data to the molecule passed to the `OEFilter::operator()` method. By default the the `OEFilter` object will not attach SD data.

See Also:`OEFilter::SetSDTag`**GetTable**`OEPlatform::oeostream *GetTable() const`

Return the `oeostream` object that tabular data is sent to. Will return a zero-pointer, the default, if no output stream has been specified.

See Also:

- *Molecular Property Table*
- `OEFilter::SetTable`

GetTypeCheck`bool GetTypeCheck() const`

Return whether molecules sent to `OEFilter::operator()` will be *type checked* before any other filtering criteria are applied. The default is `false`.

See Also:

- *Type Checking*
- `OEFilter::SetTypeCheck`

GetpKaNormalize`bool GetpKaNormalize() const`

Return whether the pKa state of molecules sent to `OEFilter::operator()` will be normalized to a neutral pH. The default is `false`.

See Also:

- *pKa Normalization*
- `OEFilter::SetpKaNormalize`

ParseNewRules

```
bool ParseNewRules(OEPlatform::oeistream &rulefile)
```

Parse the data from the rulefile `oeistream` as a NEWRULE file. These rules will override any rules specified by the *filter file* given to the `OEFILTER` constructors.

See Also:

New Rules

PrintConfig

```
void PrintConfig(OEPlatform::oeostream &log)
```

Print a human readable configuration of the object to the `oeostream log`.

SetFlagTableFailures

```
void SetFlagTableFailures(bool b)
```

Specify whether the *tabular output data* should be marked with an asterisk if that value exceeds the limits specified by the filter.

See Also:

`OEFILTER::GetFlagTableFailures`

SetSDTag

```
void SetSDTag(bool b)
```

Specify whether to attach every molecular property checked by the filter as SD data to the molecule passed to the `OEFILTER::operator()` method.

Warning: Only the properties up to the failing property will be attached by default. Use `OEFILTER::SetTable` to force every property to be attached as SD data. The same rule applies to SD data as *tabular output data*, only the properties specified in the *filter file* will be attached as SD data.

See Also:

- *SD Data Molecular Properties*
- `OEFILTER::GetSDTag`

SetTable

```
void SetTable(OEPlatform::oeostream *table, bool owned)
```

Specify the `oeostream` object that tabular data is sent to. The `owned` parameter specifies whether the `OEFilter` object should take ownership of the `table` pointer. If `owned` is `true` the `OEFilter` destructor will delete the `table` pointer, otherwise, it is the user's responsibility.

Specifying a zero-pointer will effectively turn off *tabular output data*.

See Also:

- *Molecular Property Table*
- `OEFilter::GetTable`

SetTypeCheck

```
void SetTypeCheck (bool b)
```

Specify whether molecules sent to `OEFilter::operator()` will be *type checked* before any other filtering criteria are applied.

See Also:

- *Type Checking*
- `OEFilter::GetTypeCheck`

SetpKaNormalize

```
void SetpKaNormalize (bool b)
```

Specify whether the pKa state of molecules sent to `OEFilter::operator()` will be normalized to a neutral pH.

See Also:

- *pKa Normalization*
- `OEFilter::GetpKaNormalize`

4.2 OEMolProp Constants

4.2.1 OEFilterType

This namespace controls some default built-in filters that can be passed to the `OEFilter` constructor.

Warning: Go look at the *Filter Files* before using any of the following default filters.

See Also:

Variations of Filters

BlockBuster

This is the recommended default filter. We encourage you to use it unless you are familiar with filtering or not satisfied with your initial results.

See Also:

The development of this filter is described in the *Variations of Filters* section.

Default

An alias for the current default value of `OEFilterType::BlockBuster`.

Drug

Corresponds to the [Oprea-2000] drug-like filters useful for later stage drug development. This is provided as a backwards compatible option. Experience has shown it to be too restrictive, the `OEFilterType::BlockBuster` filter is recommended.

Fragment

Filter for extracting small molecules from a database.

Lead

Corresponds to the [Oprea-2000] lead-like filters useful for preparing HTS databases.

Max

Current maximum number of provided filters.

Unknown

Currently unused for anything.

4.3 OEMolProp Functions

4.3.1 OEGet2dPSA

```
bool OEGet2dPSA(const OEChem::OEMolBase &mol, float &psa, float *atomPSA=0,
                bool SandP=false)
```

Returns the topological polar surface area for a given molecule as described in the *Polar Surface Area* section. The `SandP` parameter controls whether sulfur and phosphorus should be counted towards the total surface area.

Warning: TPSA values are mildly sensitive to the protonation state of a molecule.

The `atomPSA` parameter can be used to retrieve the contribution of each atom to the total polar surface area as shown in Listing 1.

Listing 1: Example of retrieving individual atom contributions to PSA

```
// declare an array for atom values
float * atomPSA = new float[mol.GetMaxAtomIdx()];
float psa;
OEGet2dPSA(mol, psa, atomPSA);

std::cout << "PSA = " << psa << std::endl;
for (OEIter<OEAtomBase> atom=mol.GetAtoms(); atom;++atom)
{
    unsigned int idx = atom->GetIdx();
    std::cout << idx << " " << atomPSA[idx] << std::endl;
}

// clean up
delete [] atomPSA;
```

See Also:

Polar Surface Area

4.3.2 OEGetAromaticRingCount

```
unsigned OEGetAromaticRingCount(const OEChem::OEMolBase &mol)
```

Returns the number of aromatic rings in a molecule as defined in [Ritchie-2009]. The article describes the method as the following:

The terminology 'number of aromatic rings' (or aromatic ring count) is used generically and encompasses both benzenoid aromatic rings and heteroaromatics (including, e.g. pyridine and imidazole). ... Each ring in a fused system is counted individually; thus, indole and naphthalene are each defined as having two aromatic rings.

4.3.3 OEGetFractionCsp3

```
float OEGetFractionCsp3(const OEChem::OEMolBase &mol)
```

Returns the number of sp^3 carbons divided by the total number of carbons as described in [Lovering-2009].

4.3.4 OEGetXLogP

```
bool OEGetXLogP(const OEChem::OEMolBase &mol, float &xlogp, float *atomxlogps=0)
```

Returns the XLogP for a given molecule as described in the *LogP* section.

The *atomxlogps* parameter can be used to retrieve the contribution of each atom to the total XLogP as shown in Listing 2.

Listing 2: Example of retrieving individual atom contributions to XLogP

```
// declare an array for atom values
float * atomXLogP = new float[mol.GetMaxAtomIdx()];
float xlogp;
OEGetXLogP(mol, xlogp, atomXLogP);

std::cout << "XLogP = " << xlogp << std::endl;
for (OEIter<OEAtomBase> atom=mol.GetAtoms(); atom;++atom)
{
    unsigned int idx = atom->GetIdx();
    std::cout << idx << " " << atomXLogP[idx] << std::endl;
}

// clean up
delete [] atomXLogP;
```

See Also:

LogP

4.3.5 OEMolPropGetArch

```
const char *OEMolPropGetArch()
```

4.3.6 OEMolPropGetLicensee

```
bool OEMolPropGetLicensee(std::string &licensee)
```

4.3.7 OEMolPropGetPlatform

```
const char *OEMolPropGetPlatform()
```

4.3.8 OEMolPropGetRelease

```
const char *OEMolPropGetRelease()
```

4.3.9 OEMolPropGetSite

```
bool OEMolPropGetSite(std::string &site)
```

4.3.10 OEMolPropGetVersion

```
unsigned int OEMolPropGetVersion()
```

4.3.11 OEMolPropIsLicensed

```
bool OEMolPropIsLicensed(const char *feature=0, unsigned int *expdate=0)
```


RELEASE NOTES

5.1 MolPropTK 2.1.2

- Minor documentation fixes.

5.2 MolPropTK 2.1.1

5.2.1 New features

- Added several halide that are more strict to the fragment filter.
- Added several rules to remove bad ring systems to the fragment filter.

Note: The proper way to override a `RULE` line is to use a `NEWRULE` line.

5.2.2 Minor bug fixes

- Removing duplicate nitroso and phosphoryl filter file `RULES` from the default filters.

5.3 MolPropTK 2.1.0

This is the first release of the OEMolProp toolkit. Its version number corresponds to version number of the Filter application.

INDICES AND TABLES

- *Index*
- *Search Page*

BIBLIOGRAPHY

- [Lipinski-1997] Lipinski, C.A., Lombardo, F., Dominy, B.W. and Feeney, P.J., **Experimental and Computational Approaches to Estimate Solubility and Permeability in Drug Discovery and Development Settings**, *Advanced Drug Delivery Reviews*, Vol. 23, pp. 3-25, **1997**
- [McGovern-2003] McGovern S.L., Helfand, B.T., Feng, B. and Shoichet, B.K., **A Specific Mechanism of Nonspecific Inhibition**, *Journal of Medicinal Chemistry*, Vol. 46, pp. 4265-4272, **2003**
- [Seidler-2003] Seidler J., McGovern, S.L., Doman, T.N. and Shoichet, B.K., **Identification and Prediction of Promiscuous Aggregating Inhibitors Among Known Drugs**, *Journal of Medicinal Chemistry*, Vol. 46, pp. 4477-4486, **2003**
- [Oprea-2000] Oprea, T., **Property Distribution of Drug-Related Chemical Databases**, *Journal of Computer-Aided Molecular Design*, Vol. 14, pp. 251-264, **2000**
- [Wang-1997] Wang, R., Ying, F., and Lai, L., **A New Atom-Additive Method for Calculating Partition Coefficients**, *Journal of Chemical Information and Computer Science*, Vol. 37, pp. 615-621, **1997**
- [Ertl-2000] Ertl, P., Rohde, B., and Selzer, P., **Fast Calculation of Molecular Polar Surface Area as a Sum of Fragment-Based Contributions and its Application to the Prediction of Drug Transport Properties**, *Journal of Medicinal Chemistry*, Vol. 43, pp. 3714-3717, **2000**
- [Clark-1999] Clark, D.E., **Rapid Calculation of Polar Molecular Surface Area and its Application to the Prediction of Transport Phenomena. 1. Prediction of Intestinal Absorption**, *Journal of Pharmaceutical Sciences*, Vol. 88, pp. 807-814, **1999**
- [Martin-2005] Martin, Y.C., **A Bioavailability Score**, *Journal of Medicinal Chemistry*, Vol. 48, pp. 3164-3170, **2005**
- [Veber-2002] Veber, D.F., Johnson, S.R., Cheng, H.Y., Smith, B.R., Ward, K.W. and Kopple, K.D., **Molecular Properties that Influence the Oral Bioavailability of Drug Candidates**, *Journal of Medicinal Chemistry*, Vol. 45, pp. 2615-2623, **2002**
- [Egan-2000] Egan, W.J., Merz, K.M. and Baldwin, J.J., **Prediction of Drug Absorption Using Multivariate Statistics**, *Journal of Medicinal Chemistry*, Vol. 43, pp. 3867-3877, **2000**
- [Rishton-2003] Rishton, G.M., **Nonleadlikeness and Leadlikeness in Biochemical Screening**, *Drug Discovery Today*, Vol. 8, pp. 86-96, **2003**
- [Yalkowsky-1980] Yalkowsky, S.H. and Valvani, S.C., **Solubility and Partitioning 1: Solubility of Nonelectrolytes in Water**, *Journal of Pharmaceutical Sciences*, Vol. 69, pp. 912-922, **1980**
- [MillsDean-1996] Mills, J.E.J and Dean, P.M., **Three-Dimensional Hydrogen-Bond Geometry and Probability Information from a Crystal Survey**, *Journal of Computer-Aided Molecular Design*, Vol. 10, pp. 607-622, **1996**
- [Jeffrey-1997] George A. Jeffrey, **An Introduction to Hydrogen Bonding**, *Oxford University Press*, **1997**

- [Wishart-2006] Wishart, D.S., **DrugBank: A Comprehensive Resource For In Silico Drug Discovery and Exploration**, *Nucleic Acids Research*, Vol. 34, pp. D668-672, **2006**
- [Lovering-2009] Frank Lovering, Jack Bikker, and Christine Humblet, **Escape from Flatland: Increasing Saturation as an Approach to Improving Clinical Success**, *Journal of Medicinal Chemistry*, 52 (21):6752-6756
- [Ritchie-2009] Timothy J. Ritchie and Simon J.F. Macdonald, **The impact of aromatic ring count on compound developability - are too many aromatic rings a liability in drug design?**, *Drug Discovery Today*, 14(21):1011

INDEX

A

AddNormalizationRule
 OEMolProp::OEFilter, 76

C

ClearNormalizationRules
 OEMolProp::OEFilter, 76
Constructors
 OEMolProp::OEFilter, 75

E

Example Code
 leadfilter.cpp, 69
 molpropsddata.cpp, 73
 molproptable.cpp, 71
 quietfilter.cpp, 70
 specificmolprop.cpp, 72

G

GetFlagTableFailures
 OEMolProp::OEFilter, 76
GetpKaNormalize
 OEMolProp::OEFilter, 77
GetSDTag
 OEMolProp::OEFilter, 77
GetTable
 OEMolProp::OEFilter, 77
GetTypeCheck
 OEMolProp::OEFilter, 77

L

leadfilter.cpp
 Example Code, 69

M

molpropsddata.cpp
 Example Code, 73
molproptable.cpp
 Example Code, 71

O

OEMolProp::OEFilter, 75
 AddNormalizationRule, 76
 ClearNormalizationRules, 76
 Constructors, 75
 GetFlagTableFailures, 76
 GetpKaNormalize, 77
 GetSDTag, 77
 GetTable, 77
 GetTypeCheck, 77
 operator bool, 76
 operator(), 76
 operator=, 75
 ParseNewRules, 78
 PrintConfig, 78
 SetFlagTableFailures, 78
 SetpKaNormalize, 79
 SetSDTag, 78
 SetTable, 78
 SetTypeCheck, 79
OEMolProp::OEFilterType, 79
OEMolProp::OEFilterType::BlockBuster, 79
OEMolProp::OEFilterType::Default, 80
OEMolProp::OEFilterType::Drug, 80
OEMolProp::OEFilterType::Fragment, 80
OEMolProp::OEFilterType::Lead, 80
OEMolProp::OEFilterType::Max, 80
OEMolProp::OEFilterType::Unknown, 80
OEMolProp::OEGet2dPSA, 80
OEMolProp::OEGetAromaticRingCount, 81
OEMolProp::OEGetFractionCsp3, 81
OEMolProp::OEGetXLogP, 81
OEMolProp::OEMolPropGetArch, 82
OEMolProp::OEMolPropGetLicensee, 82
OEMolProp::OEMolPropGetPlatform, 82
OEMolProp::OEMolPropGetRelease, 82
OEMolProp::OEMolPropGetSite, 82
OEMolProp::OEMolPropGetVersion, 82
OEMolProp::OEMolPropIsLicensed, 83
operator bool
 OEMolProp::OEFilter, 76

operator()
 OEMolProp::OEFilter, 76
operator=
 OEMolProp::OEFilter, 75

P

ParseNewRules
 OEMolProp::OEFilter, 78
PrintConfig
 OEMolProp::OEFilter, 78

Q

quietfilter.cpp
 Example Code, 70

S

SetFlagTableFailures
 OEMolProp::OEFilter, 78
SetpKaNormalize
 OEMolProp::OEFilter, 79
SetSDTag
 OEMolProp::OEFilter, 78
SetTable
 OEMolProp::OEFilter, 78
SetTypeCheck
 OEMolProp::OEFilter, 79
specificmolprop.cpp
 Example Code, 72