NAME

Atom

SYNOPSIS

use Atom;

DESCRIPTION

Atom class provides the following methods:

new, AddHydrogens, Copy, DeleteAtom, DeleteHydrogens, DoesAtomNeighborhoodMatch, GetAtomicInvariantValue, GetAtomicWeight, GetBondToAtom, GetBonds, GetBondsToHeavyAtoms, GetBondsToHydrogenAtoms, GetBondsToNonHydrogenAtoms, GetExactMass, GetExplicitHydrogens, GetFormalCharge, GetFreeRadicalElectrons, GetGroupNumber, GetHeavyAtomNeighbors, GetHeavyAtomNeighborsAtomInformation, GetHeavyAtomNeighborsBondformation, GetHighestCommonValence, GetHydrogenAtomNeighbors, GetHydrogens, GetImplicitHydrogens, GetLargestBondOrder, GetLargestBondOrderToHeavyAtoms, GetLargestBondOrderToNonHydrogenAtoms, GetLargestRing, GetLowestCommonValence, GetMassNumber, GetMissingHydrogens, GetNeighbors, GetNeighborsUsingAtomSpecification, GetNonHydrogenAtomNeighbors, GetNonHydrogenAtomNeighborsAtomInformation, GetNonHydrogenAtomNeighborsBondInformation, GetNonHydrogenNeighborOfHydrogenAtom, GetNumOfAromaticBondsToHeavyAtoms, GetNumOfAromaticBondsToNonHydrogenAtoms, GetNumOfBondTypesToHeavyAtoms, GetNumOfBondTypesToNonHydrogenAtoms, GetNumOfBonds, GetNumOfBondsToHeavyAtoms, GetNumOfBondsToHydrogenAtoms, GetNumOfBondsToNonHydrogenAtoms, GetNumOfDoubleBondsToHeavyAtoms, GetNumOfBondsAvailableForHeavyAtoms, GetNumOfBondsAvailableForNonHydrogenAtoms, GetNumOfDoubleBondsToNonHydrogenAtoms, GetNumOfExplicitHydrogens, GetNumOfHeavyAtomNeighbors, GetNumOfHydrogenAtomNeighbors, GetNumOfHydrogens, GetNumOfImplicitHydrogens, GetNumOfMissingHydrogens, GetNumOfNeighbors, GetNumOfNonHydrogenAtomNeighbors, GetNumOfRings, GetNumOfRingsWithEvenSize, GetNumOfRingsWithOddSize, GetNumOfRingsWithSize, GetNumOfRingsWithSizeGreaterThan, GetNumOfRingsWithSizeLessThan, GetNumOfSigmaAndPiBondsToHeavyAtoms, GetNumOfSigmaAndPiBondsToNonHydrogenAtoms, GetNumOfSingleBondsToHeavyAtoms, GetNumOfSingleBondsToNonHydrogenAtoms, GetNumOfTripleBondsToHeavyAtoms, GetNumOfTripleBondsToNonHydrogenAtoms, GetPeriodNumber, GetPotentialTotalCommonValence, GetRings, GetRingsWithEvenSize, GetRingsWithOddSize, GetRingsWithSize, GetRingsWithSizeGreaterThan, GetRingsWithSizeLessThan, GetSizeOfLargestRing, GetSizeOfSmallestRing, GetSmallestRing, GetSpinMultiplicity, GetSumOfBondOrders, GetSumOfBondOrdersToHeavyAtoms, GetSumOfBondOrdersToHydrogenAtoms, GetSumOfBondOrdersToNonHydrogenAtoms, GetValence, GetValenceElectrons, GetValenceFreeElectrons, GetX, GetXYZ, GetXYZVector, GetY, GetZ, IsAmideCarbon, IsAmideNitrogen, IsAromatic, IsArsenic, IsBondedToAtom, IsBromine, IsCarbon, IsCarboxylCarbon, IsCarboxylOxygen, IsCarboxylateCarbon, IsCarboxylateOxygen, IsChlorine, IsFluorine, IsFunctionalClassType, IsGuadiniumCarbon, IsGuadiniumNitrogen, IsHBondAcceptor, IsHBondDonor, IsHalogen, IsHeteroAtom, IsHydrogen, IsHydrogenBondAcceptor, IsHydrogenBondDonor, IsHydrophobic, IsInRing, IsInRingOfSize, IsIodine, IsIsotope, IsLipophilic, IsMetallic, IsNegativelyIonizable, IsNitrogen, IsNonCarbonOrHydrogen, IsNotInRing, IsOnlyInOneRing, IsOxygen, IsPhosphateOxygen, IsPhosphatePhosphorus, IsPhosphorus, IsPolarAtom, IsPolarHydrogen, IsPositivelyIonizable, IsSaturated, IsSelenium, IsSilicon, IsStereoCenter, IsSulfur, IsSulphur, IsTellurium, IsTerminal, IsTopologicalPharmacophoreType, IsUnsaturated, SetAtomSymbol, SetAtomicNumber, SetExplicitHydrogens, SetMassNumber, SetStereoCenter, SetStereochemistry, SetX, SetXYZ, SetY, SetZ, StringifyAtom

Atom class is derived from ObjectProperty base class which provides methods not explicitly defined in Atom or ObjectProperty class using Perl's AUTOLOAD functionality. These methods are generated on-the-fly for a specified object property:

```
Set<PropertyName>(<PropertyValue>);
$PropertyValue = Get<PropertyName>();
Delete<PropertyName>();
```

METHODS

new

```
$NewAtom = new Atom([%PropertyNameAndValues]);
```

Using specified *Atom* property names and values hash, new method creates a new object and returns a reference to newly created Atom object. By default, following properties are initialized:

```
ID = SequentialObjectID
Name = "Atom <SequentialObjectID>"
AtomSymbol = ""
```

```
AtomicNumber = 0
XYZ = ZeroVector
```

Except for *ID* property, all other default properties and other additional properties can be set during invocation of this method.

Examples:

```
$Atom = new Atom();
$CarbonAtom = new Atom('AtomSymbol' => 'C', 'XYZ' => (0.0, 1.0,
              0.0));
$OxygenAtom = new Atom('AtomName' => 'Oxygen', AtomSymbol' => 'O',
              'XYZ' => (1.0, 1.0, 1.0));
```

AddHydrogens

```
$NumOfHydrogensAdded = $Atom->AddHydrogens();
```

Adds hydrogens to an Atom present in a Molecule object and returns the number of added hydrogens. The current release of MayaChemTools doesn't assign hydrogen positions.

Copy

```
$AtomCopy = $Atom->Copy();
```

Copy *Atom* and its associated data using Storable::dclone and return a new Atom object.

DeleteAtom

```
$Atom->DeleteAtom();
```

Delete *Atom* from a molecule.

DoesAtomNeighborhoodMatch

```
$Status = $Atom->DoesAtomNeighborhoodMatch($CentralAtomSpec);
$Status = $Atom->DoesAtomNeighborhoodMatch($CentralAtomSpec,
                           $NbrAtomSpecsRef);
$Status = $Atom->DoesAtomNeighborhoodMatch($CentralAtomSpec,
                           $NbrAtomSpecsRef, $AllowedNbrBondSpecsRef);
$Status = $Atom->DoesAtomNeighborhoodMatch($CentralAtomSpec,
                           $NbrAtomSpecsRef, $NbrBondSpecsRef,
                           $AllowedNbrOfNbrAtomSpecsRef);
```

Returns 1 or 0 based on whether atom matches central atom and its neighborhood using specified atom and bonds specifications. Neighborhood atom and bond specifications are specified as array references containing neighbor atom and bond specifications.

Let:

```
AS = Atom symbol corresponding to element symbol, atomic number (#n)
     or any atom (A)

X<n>  = Number of non-hydrogen atom neighbors or heavy atoms
        attached to atom
T<n>  = Total number of atom neighbors including implicit and explicit
        hydrogens
BO<n> = Sum of bond orders to non-hydrogen atom neighbors or heavy
        atoms attached to atom
LBO<n> = Largest bond order of non-hydrogen atom neighbors or heavy
         atoms attached to atom
SB<n> = Number of single bonds to non-hydrogen atom neighbors or
        heavy atoms attached to atom
TSB<n> = Total number of single bonds to atom neighbors including implicit
         and explicit hydrogens
DB<n> = Number of double bonds to non-hydrogen atom neighbors or
        heavy atoms attached to atom
TB<n> = Number of triple bonds to non-hydrogen atom neighbors or
        heavy atoms attached to atom
AB<n> = Number of aromatic bonds to non-hydrogen atom neighbors or
        heavy atoms attached to atom
H<n>   = Number of implicit and explicit hydrogens for atom
Ar     = Aromatic annotation indicating whether atom is aromatic
RA or RA<n>  = Ring atom annotation indicating whether atom
```

```
                is a ring
    TR<n>  = Total number of rings containing atom
    FC<+n/-n> = Formal charge assigned to atom
    MN<n> = Mass number indicating isotope other than most abundant isotope
    SM<n> = Spin multiplicity of atom. Possible values: 1 (singlet),
            2 (doublet) or 3 (triplet)
```

Then, atom specification corresponds to:

```
    AS.X<n>.T<n>.BO<n>.LBO<n>.<SB><n>.TSB<n>.<DB><n>.<TB><n>.AB<n>.H<n>.Ar.
    RA<n>.TR<n>FC<+n/-n>.MN<n>.SM<n>
```

Except for AS which is a required atomic invariant in atom specification, all other atomic invariants are optional. For an atom specification to match an atom, the values of all specified atomic invariants must match. Exclamation in from of atomic invariant can be used to negate its effect during the match.

For *FC* value matching, the following value operators are also supported:

```
    o +* : Any positive value
    o -* : Any negative value
    o > ValidNumber or >= ValidNumber
    o < ValidNumber or <= ValidNumber
```

A comma delimited atom specification string is used to match any one of the specified atom specification.

Notes:

```
    o During atom specification match to an atom, the first atomic invariant is always
assumed to
      atom symbol.
```

Examples:

```
    o ('N', 'N', 'N')
    o ('N.FC0', 'N.FC0', 'N,N.FC+1.H1')
    o ('N.H2', 'N.H2', 'N.H1')
    o ('C,N', '!N', '!H')
    o ('C,N', 'N.Ar', 'N.R5')
```

Let:

```
    -|1|s|Single = Single bond
    =|2|d|Double = Double bond
    #|3|t|Triple  = Triple bond
    :|1.5|a|Ar|Aromatic = Aromatic bond

    @|RB|Ring = Ring bond
    ~|*|Any = Any bond
```

Then, bond specification corresponds to:

```
    -.:
    =.@
    Double.Aromatic
```

For a bond specification to match bond between two atoms, the values of all specified bond symbols must match. Exclamation in from of bond symbol can be used to negate its effect during the match.

A comma delimited bond specification string is used to match any one of the specified atom specification.

Notes:

```
    o During atom neighborhood match for central atom neighborhood atom and bond
specifications,
      implicit or missing hydrogens are automatically checked for any matches to
unmatched
      specifications.
```

Examples:

```
    Aromatic carbon in a 5 membered ring:
                              $Atom->DoesAtomNeighborhoodMatch('C.Ar.RA5');

    AcetylenicCarbon: $Atom->DoesAtomNeighborhoodMatch('C.T2.TB1'); or
```

```
                        $Atom->DoesAtomNeighborhoodMatch('C.T2.TB1',
                                ['*', '*'], ['#', '-']);

        GuadiniumCarbon: $Atom->DoesAtomNeighborhoodMatch('C.X3.BO4',
                                ['N.FC0', 'N.FC0', 'N.FC0,N.FC+1'],
                                ['-', '-', '='],
                                ['C,H', 'C,H', 'C,H']);

        AmideCarbon: $Atom->DoesAtomNeighborhoodMatch('C.X3.BO4,C.X2.BO3',
                                ['C,H', 'O', 'N'],
                                ['-', '=', '-'],
                                ['C,H', 'C', 'C,H,N,O,S,P,F,Cl,Br,I']);

        CarboxylCarbon: $Atom->DoesAtomNeighborhoodMatch('C.X3.BO4,C.X2.BO3',
                                ['C,H', 'O', 'O.X1.FC0'],
                                ['-', '=', '-'],
                                ['C,H', 'C', 'C']);

        CarboxylateCarbon: $Atom->DoesAtomNeighborhoodMatch('C.X3.BO4,C.X2.BO3',
                                ['C,H', 'O', 'O.X1.FC-1'],
                                ['-', '=', '-'],
                                ['C,H', 'C', 'C']);
```

DeleteHydrogens

```
$NumOfHydrogensDeleted = $Atom->AddHydrogens();
```

Delete hydrogens from an Atom present in a Molecule object and returns the number of deleted hydrogens.

GetAtomicInvariantValue

```
$Value = $Atom->GetAtomicInvariantValue($AtomicInvariant);
```

Returns atomic invariant value for a specified *AtomicInvariant*. The current release of MayaChemTools supports following abbreviations and descriptive names for *AtomicInvarints*:

```
AS : Atom or element symbol
X : NumOfNonHydrogenAtomNeighbors or NumOfHeavyAtomNeighbors
T : TotalNumOfAtomNeighbors
BO : SumOfBondOrdersToNonHydrogenAtoms or SumOfBondOrdersToHeavyAtoms
LBO : LargestBondOrderToNonHydrogenAtoms or LargestBondOrderToHeavyAtoms
SB :  NumOfSingleBondsToNonHydrogenAtoms or NumOfSingleBondsToHeavyAtoms
TSB :  TotalNumOfSingleBonds
DB : NumOfDoubleBondsToNonHydrogenAtoms or NumOfDoubleBondsToHeavyAtoms
TB : NumOfTripleBondsToNonHydrogenAtoms or NumOfTripleBondsToHeavyAtoms
AB : NumOfAromaticBondsToNonHydrogenAtoms or NumOfAromaticBondsToHeavyAtoms
H :  NumOfImplicitAndExplicitHydrogens
Ar : Aromatic
Str : Stereochemistry
RA : RingAtom
FC : FormalCharge
AN : AtomicNumber
AM : AtomicMass
MN : MassNumber
SM : SpinMultiplicity
```

GetAtomicWeight

```
$Value = $Aom->GetAtomicWeight();
```

Returns atomic weight of an Atom which corresponds to either explicity set *AtomicWeight* atom property or atomic weight of the corresponding element in the periodic table available by PeriodicTable module.

GetBondToAtom

```
$Bond = $Atom->GetBondToAtom($OtherAtom);
```

Returns a Bond object corresponding to bond between *Atom* and *OtherAtom* in a molecule.

GetBonds

```
@Bonds = $Aoto->GetBonds();
```

Returns an array of Bond objects corresponding to all bonds from *Atom* to other bonded atoms in a molecule.

GetBondsToHeavyAtoms

```
@Bonds = $Atom->GetBondsToHeavyAtoms();
```

Returns an array of Bond objects corresponding to bonds from *Atom* to other bonded non-hydrogen atoms in a molecule.

GetBondsToHydrogenAtoms

```
@Bonds = $Atom->GetBondsToHydrogenAtoms();
```

Returns an array of Bond objects corresponding to bonds from *Atom* to any other hydrogen atom in a molecule.

GetBondsToNonHydrogenAtoms

```
@Bonds = $Atom->GetBondsToNonHydrogenAtoms();
```

Returns an array of Bond objects corresponding to bonds from *Atom* to other bonded non-hydrogen atoms in a molecule.

GetExactMass

```
$ExactMass = $Atom->GetExactMass();
```

Returns exact mass of an *Atom* which correspond to one of these three values: explicity set *ExactMass* property; mass of natural isotope for an explicty set value of *MassNumber*; most abundant natural isotope mass for *Atom* with valid atomic number value available by PerodicTable module.

GetExplicitHydrogens

```
$NumOfExplicitHydrogens = $Atom->GetExplicitHydrogens();
```

Returns number of hydrogens explicity bonded to an *Atom* in a molecule.

GetFormalCharge

```
$FormalCharge = $Atom->GetFormalCharge();
```

Returns formal charge of an *Atom* in a molecule.

GetFreeRadicalElectrons

```
$FreeRadicalElectrons = $Atom->GetFreeRadicalElectrons();
```

Returns number of free radical electrons corresponding to to one of these three values: *FreeRadicalElectrons* property; *SpinMultiplicity* property; value of 0.

For atoms with explicit assignment of *SpinMultiplicity* atom property values,

```
Singlet  - two unparied electrons corresponding to one spin state
Doublet - free radical; an unpaired electron corresponding to two
          spin states
Triplet - two unparied electrons corresponding to three spin states
          (divalent carbon atoms: carbenes)
```

FreeRadicalElectrons are calculated as follows:

```
Doublet: 1 (one valence electron not available for bonding)
Singlet: 2 (two valence electrons not available for bonding)
Triplet: 2 (two valence electrons not available for bonding)
```

GetGroupNumber

```
$GroupNumber = $Atom->GetGroupNumber();
```

Returns group number of an *Atom* in a molecule with a valid atomic number.

GetHeavyAtomNeighbors

```
$NumOfHeavyAtoms = $Atom->GetHeavyAtomNeighbors();
```

```
@HeavyAtoms = $Atom->GetHeavyAtomNeighbors();
```

Return number of heavy atoms or an array of Atom objects corresponding to heavy atoms bonded to an *Atom* in a molecule.

GetHeavyAtomNeighborsAtomInformation

```
($NumOfAtomNeighbors, $AtomNeighborsRef,
 $NumOfAtomNeighborsType, $AtomNeighborsTypeMapRef) = $Atom->
                          GetHeavyAtomNeighborsAtomInformation();
```

Returns atoms information for all non-hydrogen atoms attached to an *Atom* in a molecule.

The following values are returned:

- Number of non-hydrogen atom neighbors
- A reference to an array containing atom objects corresponding to non-hydrogen atom neighbors
- Number of different types of non-hydrogen atom neighbors
- A reference to a hash containing atom symbol as key with value corresponding to its count for non-hydrogen atom neighbors

GetHeavyAtomNeighborsBondformation

```
($NumOfBonds, $BondTypeCountMapRef,
$AtomsBondTypesCountMapRef,
$AtomsBondTypeAtomsMap) = $Atom->
                          GetHeavyAtomNeighborsBondformation();
```

Returns bonds information for all non-hydrogen atoms attached to an *Atom* in a molecule.

The following values are returned:

- Number of bonds to non-hydrogen atom neighbors
- A reference to an array containing bond objects corresponding to non-hydrogen atom neighbors
- A reference to a hash containing bond type as key with value corresponding to its count for non-hydrogen atom neighbors. Bond types are: Single, Double or Triple
- A reference to a hash containing atom symbol as key pointing to bond type as second key with values corresponding to count of bond types for atom symbol for non-hydrogen atom neighbors
- A reference to a hash containing atom symbol as key pointing to bond type as second key with values corresponding to atom objects array involved in corresponding bond type for atom symbol for non-hydrogen atom neighbors

GetHighestCommonValence

```
$HighestCommonValence = $Atom->GetHighestCommonValence();
```

Returns highest common valence of an *Atom* which corresponds to either explicity set *HighestCommonValence* atom property or highest common valence of the corresponding element in the periodic table available by PerodicTable module.

GetHydrogens

```
$NumOfHydrogens = $Atom->GetHydrogens();
```

Returns total number of hydrogens for an *Atom* in a molecule including both hydrogen atom neighbors and implicit hydrogens.

GetHydrogenAtomNeighbors

```
$NumOfHydrogenAtomNeighbors = $Atom->GetHydrogenAtomNeighbors();
@HydrogenAtomNeighbors = $Atom->GetHydrogenAtomNeighbors();
```

Return number of hydrogen atoms or an array of *Atom* objects corresponding to hydrogen atoms bonded to an *Atom* in a molecule.

GetImplicitHydrogens

```
$NumOfImplicitHydrogens = $Atom->GetImplicitHydrogens();
```

Returns number of implicit hydrogens for an *Atom* in a molecule. This value either corresponds to explicitly set *ImplicitHydrogens* atom property or calculated as the difference between the value of

potential total valence and sum of bond orders to both hydrogen and non-hydrogen atom neighbors.

GetPotentialTotalCommonValence

```
$PotentialTotalValence = $Atom->GetPotentialTotalCommonValence();
```

Returns potential total common valence of an *Atom* in a molecule corresponding to a specific valence model set for the molecule using its SetValenceModel method or default internal valence model. It is used during the calculation of missing or implicit hydrogens.

The current release of MayaChemTools supports three valence models: *MDLValenceModel, DaylightValenceModel, InternalValenceModel or MayaChemToolsValenceModel*.

For *MDLValenceModel* and *DaylightValenceModel*, the following data files, distributed with the package, are used to calculate potential total valence:

```
lib/data/MDLValenceModelData.csv
lib/data/DaylightValenceModelData.csv
```

The calculation of potential total common valence for these two models is performed as follows: Calculate current effective total valence of the *Atom* by adding up the bond order of its neighbors and number of free radical electrons; Find available common valence for the *Atom*, corresponding to any specified formal charge, higher than the effective total valence, and return it as *PotentialTotalValence*.

The calculation of potential total common valence For *InternalValenceModel* or *MayaChenToolsValenceModel* doesn't uses PeriodicTable module to retrieve values for common valence, which in turn reads in PeriodicTableElements.csv file distributed with the package.

For elements with one one common valence, potential total common valence corresponds to:

```
CommonValence + FormalCharge - FreeRadicalElectrons
```

For elements with multiple common valences, each common valence is used to calculate total potential common valence as shown above, and the first total potential common valence greater than the sum of bond orders to all neighbors is selected as the final total common valence.

FormalCharge sign is reversed for electropositive elements with positive formal charge during common valence calculations. Electropositive elements, metals and transition elements, have usually plus formal charge and it leads to decrease in common valence; the negative formal charge should result in the decrease of common valence.

The sign of formal charge is adjusted as follows.

Group numbers > 14 - Group numbers 15 (N), 16 (O), 17 (F), 18 (He):

Formal charge sign is not adjusted. Positive and negative values result in the increase and decrease of valence.

Group 14 containing C, Si, Ge, Sn, Pb...:

Formal charge sign is reversed for positive values. Both positive and negative values result in the decrease of valence.

Group 13 containing B, Al, Ga, In, Tl...:

Formal charge sign is always reversed. Positive and negative values result in the decrease and increase of valence.

Groups 1 (H) through 12 (Zn)...:

Formal charge sign is reversed for positive values. Both positive and negative values result in the decrease of valence.

Lanthanides and actinides:

Formal charge sign is reversed for positive values. Both positive and negative values result in the decrease of valence.

GetLargestBondOrder

```
$LargestBO =$Atom->GetLargestBondOrder();
```

Returns largest bond order for an *Atom* among the bonds to other atoms in a molecule.

GetLargestBondOrderToHeavyAtoms

```
$LargestBO =$Atom->GetLargestBondOrderToHeavyAtoms();
```

Returns largest bond order for an *Atom* among the bonds to other heavy atoms in a molecule.

GetLargestBondOrderToNonHydrogenAtoms

```
$LargestBO =$Atom->GetLargestBondOrderToNonHydrogenAtoms();
```

Returns largest bond order for an *Atom* among the bonds to other non-hydrogen atoms in a molecule.

GetLargestRing

```
@RingAtoms = $Atom->GetLargestRing();
```

Returns an array of ring *Atom* objects corresponding to the largest ring containing *Atom* in a molecule.

GetLowestCommonValence

```
$LowestCommonValence = $Atom->GetLowestCommonValence();
```

Returns lowest common valence of an *Atom* which corresponds to either explicity set *LowestCommonValence* atom property or highest common valence of the corresponding element in the periodic table available by PerodicTable module.

GetMassNumber

```
$MassNumber = $Aom->GetMassNumber();
```

Returns atomic weight of an Atom which corresponds to either explicity set *MassNumber* atom property or mass number of the most abundant natural isotope of the corresponding element in the periodic table available by PeriodicTable module.

GetMissingHydrogens

```
$NumOfMissingHydrogens = $Atom->GetMissingHydrogens();
```

Returns number of missing hydrogens for an *Atom* in a molecule. This value either corresponds to explicitly set *ImplicitHydrogens* atom property or calculated as the difference between the value of potential total valence and sum of bond orders to both hydrogen and non-hydrogen atom neighbors.

GetNeighbors

```
$NumOfNeighbors = $Atom->GetNeighbors();
@Neighbors = $Atom->GetNeighbors();
```

Returns number of neighbor atoms or an array of *Atom* objects corresponding to all atoms bonded to an *Atom* in a molecule.

GetNeighborsUsingAtomSpecification

```
@AtomNeighbors = $Atom->GetNeighborsUsingAtomSpecification($AtomSpec);
$NumOfNeighbors = $Atom->GetNeighborsUsingAtomSpecification($AtomSpec);

@AtomNeighbors = $Atom->GetNeighborsUsingAtomSpecification($AtomSpec,
                 @ExcludeNeighbors);
```

Returns number of neighbor atoms or an array of *Atom* objects matching atom specification corresponding to atom neighbors of an *Atom* in a molecule. Optionally, *Atom* neighbors can be excluded from the neighbors list using *ExcludeNeighbors*.

Notes:

```
o AtomSpecification correspond to any valid AtomicInvariant based atomic
specifications
  as supported by DoesAtomNeighborhoodMatch method
o Multiple atom specifications can be used in a string delimited by comma
```

GetNonHydrogenAtomNeighbors

```
$NumOfNeighbors = $Atom->GetNonHydrogenAtomNeighbors();
@Neighbors = $Atom->GetNonHydrogenAtomNeighbors();
```

Returns number of non-hydrogen atoms or an array of Atom objects corresponding to non-hydrogen atoms bonded to an *Atom* in a molecule.

GetNonHydrogenAtomNeighborsAtomInformation

```
($NumOfAtomNeighbors, $AtomNeighborsRef,
 $NumOfAtomNeighborsType, $AtomNeighborsTypeMapRef) = $Atom->
                          GetNonHydrogenAtomNeighborsAtomInformation();
```

Returns atoms information for all non-hydrogen atoms attached to an *Atom* in a molecule.

The following values are returned:

```
    o Number of non-hydrogen atom neighbors
    o A reference to an array containing atom objects corresponding to
      non-hydrogen atom neighbors
    o Number of different types of non-hydrogen atom neighbors
    o A reference to a hash containing atom symbol as key with value
      corresponding to its count for non-hydrogen atom neighbors
```

GetNonHydrogenAtomNeighborsBondInformation

```
($NumOfBonds, $BondTypeCountMapRef,
$AtomsBondTypesCountMapRef,
$AtomsBondTypeAtomsMap) = $Atom->
                          GetNonHydrogenAtomNeighborsBondInformation();
```

Returns bonds information for all non-hydrogen atoms attached to an *Atom* in a molecule.

The following values are returned:

```
    o Number of bonds to non-hydrogen atom neighbors
    o A reference to an array containing bond objects corresponding to
      non-hydrogen atom neighbors
    o A reference to a hash containing bond type as key with value
      corresponding to its count for non-hydrogen atom neighbors. Bond
      types are: Single, Double or Triple
    o A reference to a hash containing atom symbol as key pointing to bond
      type as second key with values corresponding to count of bond types for atom
      symbol for non-hydrogen atom neighbors
    o A reference to a hash containing atom symbol as key pointing to bond
      type as second key with values corresponding to atom objects array involved
      in corresponding bond type for atom symbol for non-hydrogen atom neighbors
```

GetNonHydrogenNeighborOfHydrogenAtom

```
$Atom = $Atom->GetNonHydrogenNeighborOfHydrogenAtom();
```

Returns non-hydrogen or heavy atom neighbor of a hydrogen atom in a molecule..

GetNumOfAromaticBondsToHeavyAtoms

```
$NumOfBonds = $Atom->GetNumOfAromaticBondsToHeavyAtoms();
```

Returns number of aromatic bonds from an *Atom* to other non-hydrogen or heavy atoms in a molecule.

GetNumOfAromaticBondsToNonHydrogenAtoms

```
$NumOfBonds = $Atom->GetNumOfAromaticBondsToNonHydrogenAtoms();
```

Returns number of aromatic bonds from an *Atom* to other non-hydrogen or heavy atoms in a molecule.

GetNumOfBonds

```
$NumOfBonds = $Atom->GetNumOfBonds();
```

Returns number of bonds from an *Atom* to other atoms in a molecule.

GetNumOfBondsAvailableForHeavyAtoms

```
$NumOfBonds = $Atom->GetNumOfBondsAvailableForHeavyAtoms();
```

Get number of bonds available to form additional bonds with heavy atoms, excluding any implicit bonds to hydrogens set using *ImplicitHydrogens* property.

It's different from number of implicit or missing hydrogens, both of which are equivalent.

For example, in a SMILES string, [nH] ring atom corresponds to an aromatic nitrogen. Although the hydrogen specified for n is treated internally as implicit hydrogen and shows up in missing hydrogen count, it's not available to participate in double bonds to additional heavy atoms.

GetNumOfBondsAvailableForNonHydrogenAtoms

```
$NumOfBonds = $Atom->GetNumOfBondsAvailableForNonHydrogenAtoms();
```

Get number of bonds available to form additional bonds with heavy atoms, excluding any implicit bonds to hydrogens set using ImplicitHydrogens property.

GetNumOfBondsToHeavyAtoms

```
$NumOfBondsToHeavyAtoms = $Atom->GetNumOfBondsToHeavyAtoms();
```

Returns number of bonds from an *Atom* to other heavy atoms in a molecule.

GetNumOfBondsToHydrogenAtoms

```
$NumOfBonds = $Atom->GetNumOfBondsToHydrogenAtoms();
```

Returns number of bonds from an *Atom* to other hydrogen atoms in a molecule.

GetNumOfBondsToNonHydrogenAtoms

```
$NumOfBonds = $Atom->GetNumOfBondsToNonHydrogenAtoms();
```

Returns number of bonds from an *Atom* to other non-hydrogen atoms in a molecule.

GetNumOfBondTypesToHeavyAtoms

```
($NumOfSingleBonds, $NumOfDoubleBonds,
 $NumOfTripleBonds, $NumOfAromaticBonds) = $Atom->
              GetNumOfBondTypesToHeavyAtoms($CountAromaticBonds);
```

Get number of single, double, triple, and aromatic bonds from an *Atom* to all other non-hydrogen atoms in a molecule.

Value of *CountAtomaticBonds* parameter controls whether number of aromatic bonds is returned; default is not to count aromatic bonds. During counting of aromatic bonds, the bond marked aromatic is not included in the count of other bond types.

GetNumOfBondTypesToNonHydrogenAtoms

```
($NumOfSingleBonds, $NumOfDoubleBonds,
 $NumOfTripleBonds, $NumOfAromaticBonds) = $Atom->
        GetNumOfBondTypesToNonHydrogenAtoms($CountAromaticBonds);
```

Get number of single, double, triple, and aromatic bonds from an *Atom* to all other non-hydrogen atoms in a molecule.

Value of *CountAtomaticBonds* parameter controls whether number of aromatic bonds is returned; default is not to count aromatic bonds. During counting of aromatic bonds, the bond marked aromatic is not included in the count of other bond types.

GetNumOfDoubleBondsToHeavyAtoms

```
$NumOfDoubleBonds = $Atom->GetNumOfDoubleBondsToHeavyAtoms();
```

Returns number of double bonds from an *Atom* to other heavy atoms or non-hydrogen atoms in a molecule.

GetNumOfDoubleBondsToNonHydrogenAtoms

```
$NumOfDoubleBonds =$Atom->GetNumOfDoubleBondsToNonHydrogenAtoms();
```

Returns number of double bonds from an *Atom* to other heavy atoms or non-hydrogen atoms in a molecule.

GetNumOfHeavyAtomNeighbors

```
$NumOfNeighbors = $Atom->GetNumOfHeavyAtomNeighbors();
```

Returns number heavy atom neighbors for an *Atom* in a molecule.

GetNumOfHydrogenAtomNeighbors

```
$NumOfNeighbors = $Atom->GetNumOfHydrogenAtomNeighbors();
```

Returns number hydrogens atom neighbors for an *Atom* in a molecule.

GetNumOfMissingHydrogens

```
$NumOfMissingHydrogens = $Atom->GetNumOfMissingHydrogens();
```

Returns number of implicit hydrogens for an *Atom* in a molecule. This value either corresponds to explicitly set *ImplicitHydrogens* atom property or calculated as the difference between the value of potential total valence and sum of bond orders to both hydrogen and non-hydrogen atom neighbors.

GetNumOfExplicitHydrogens

```
$NumOfExplicitHydrogens = $Atom->GetNumOfExplicitHydrogens();
```

Returns number hydrogens atom neighbors for an *Atom* in a molecule.

GetNumOfHydrogens

```
$NumOfHydrogens = $Atom->GetNumOfHydrogens();
```

Returns total number of hydrogens for an *Atom* in a molecule including both hydrogen atom neighbors and implicit hydrogens.

GetNumOfImplicitHydrogens

```
$NumOfImplicitHydrogens = $Atom->GetNumOfImplicitHydrogens();
```

Returns number of implicit hydrogens for an *Atom* in a molecule. This value either corresponds to explicitly set *ImplicitHydrogens* atom property or calculated as the difference between the value of potential total valence and sum of bond orders to both hydrogen and non-hydrogen atom neighbors.

GetNumOfNeighbors

```
$NumOfNeighbors = $Atom->GetNumOfNeighbors();
```

Returns number atom neighbors for an *Atom* in a molecule.

GetNumOfNonHydrogenAtomNeighbors

```
$NumNeighbors = $This->GetNumOfNonHydrogenAtomNeighbors();
```

Returns number non-hydrogens atom neighbors for an *Atom* in a molecule.

GetNumOfRings

```
$NumOfRings = $Atom->GetNumOfRings();
```

Returns number of rings containing *Atom* in a molecule.

GetNumOfRingsWithEvenSize

```
$NumOfRings = $Atom->GetNumOfRingsWithEvenSize();
```

Returns number of rings with even size containing *Atom* in a molecule.

GetNumOfRingsWithOddSize

```
$NumOfRings = $Atom->GetNumOfRingsWithOddSize();
```

Returns number of rings with odd size containing *Atom* in a molecule.

GetNumOfRingsWithSize

```
$NumOfRings = $Atom->GetNumOfRingsWithSize($RingSize);
```

Returns number of rings with specific *RingSize* containing *Atom* in a molecule.

GetNumOfRingsWithSizeGreaterThan

```
$NumOfRings = $Atom->GetNumOfRingsWithSizeGreaterThan($RingSize);
```

Returns number of rings with size greater than specific *RingSize* containing *Atom* in a molecule.

GetNumOfRingsWithSizeLessThan

```
$NumOfRings = $Atom->GetNumOfRingsWithSizeLessThan($RingSize);
```

Returns number of rings with size less than specific *RingSize* containing *Atom* in a molecule.

GetNumOfSigmaAndPiBondsToHeavyAtoms

```
($NumOfSigmaBonds, $NumOfPiBonds) = $Atom->
                            GetNumOfSigmaAndPiBondsToHeavyAtoms();
```

Get number of sigma and pi bonds from an *Atom* to all other non-hydrogen atoms in a molecule.

Sigma and pi bonds are counted using the following methodology: a single bond correspond to one sigma bond; a double bond contributes one to sigma bond count and one to pi bond count; a triple bond contributes one to sigma bond count and two to pi bond count.

GetNumOfSigmaAndPiBondsToNonHydrogenAtoms

```
($NumOfSigmaBonds, $NumOfPiBonds) = $Atom->
                            GetNumOfSigmaAndPiBondsToNonHydrogenAtoms();
```

Get number of sigma and pi bonds from an *Atom* to all other non-hydrogen atoms in a molecule.

Sigma and pi bonds are counted using the following methodology: a single bond correspond to one sigma bond; a double bond contributes one to sigma bond count and one to pi bond count; a triple bond contributes one to sigma bond count and two to pi bond count.

GetNumOfSingleBondsToNonHydrogenAtoms

```
$NumOfSingleBonds =$Atom->GetNumOfSingleBondsToNonHydrogenAtoms();
```

Returns number of single bonds from an *Atom* to other heavy atoms or non-hydrogen atoms in a molecule.

GetNumOfSingleBondsToHeavyAtoms

```
$NumOfSingleBonds = $Atom->GetNumOfSingleBondsToHeavyAtoms();
```

Returns number of single bonds from an *Atom* to other heavy atoms or non-hydrogen atoms in a molecule.

GetNumOfTripleBondsToNonHydrogenAtoms

```
$NumOfTripleBonds =$Atom->GetNumOfTripleBondsToNonHydrogenAtoms();
```

Returns number of triple bonds from an *Atom* to other heavy atoms or non-hydrogen atoms in a molecule.

GetNumOfTripleBondsToHeavyAtoms

```
$NumOfTripleBonds = $Atom->GetNumOfTripleBondsToHeavyAtoms();
```

Returns number of triple bonds from an *Atom* to other heavy atoms or non-hydrogen atoms in a molecule.

GetPeriodNumber

```
$PeriodNumber = $Atom->GetPeriodNumber();
```

Returns periodic table period number for an *Atom* in a molecule with a valid atomic number .

GetRings

```
@Rings = $Aotm->GetRings();
```

Returns an array of references to arrays containing ring atoms corressponding to all rings containing *Atom* in a molecule.

GetRingsWithEvenSize

```
@Rings = $Aotm->GetRingsWithEvenSize();
```

Returns an array of references to arrays containing ring atoms corressponding to all rings with even size containing *Atom* in a molecule.

GetRingsWithOddSize

```
@Rings = $Aotm->GetRingsWithOddSize();
```

Returns an array of references to arrays containing ring atoms corressponding to all rings with odd size containing *Atom* in a molecule.

GetRingsWithSize

```
@Rings = $Aotm->GetRingsWithSize($RingSize);
```

Returns an array of references to arrays containing ring atoms corressponding to all rings with specific *RingSize* containing *Atom* in a molecule.

GetRingsWithSizeGreaterThan

```
@Rings = $Aotm->GetRingsWithSizeGreaterThan($RingSize);
```

Returns an array of references to arrays containing ring atoms corressponding to all rings with size

greater than specific *RingSize* containing *Atom* in a molecule.

GetRingsWithSizeLessThan

```
@Rings = $Aotm->GetRingsWithSizeLessThan($RingSize);
```

Returns an array of references to arrays containing ring atoms corressponding to all rings with size less than specific *RingSize* containing *Atom* in a molecule.

GetSizeOfLargestRing

```
$Size = $Atom->GetSizeOfLargestRing();
```

Returns size of the largest ring containing *Atom* in a molecule.

GetSizeOfSmallestRing

```
$Size = $Atom->GetSizeOfSmallestRing();
```

Returns size of the smallest ring containing *Atom* in a molecule.

GetSmallestRing

```
@RingAtoms = $Atom->GetSmallestRing();
```

Returns an array of ring *Atom* objects corresponding to the largest ring containing *Atom* in a molecule.

GetSpinMultiplicity

```
$SpinMultiplicity = $Atom->GetSpinMultiplicity();
```

Returns spin multiplicity of an *Atom* corresponding to one of these three values: explicitly set SpinMultiplicity property value; calculated from FreeRadicalElectrons property; value of 0.

The SpinMultiplicity is calculate from *FreeRadicalElectrons* property as follows:

```
FreeRadicalElectrons: 1; SpinMultiplicity: 2
FreeRadicalElectrons: 2; SpinMultiplicity: 1
FreeRadicalElectrons: other; SpinMultiplicity: 0
```

GetSumOfBondOrders

```
$SumBondOrders = $Atom->GetSumOfBondOrders();
```

Returns sum of bond orders corresponding to all atoms bonded to an *Atom* in a molecule.

GetSumOfBondOrdersToHeavyAtoms

```
$SumBondOrders = $Atom->GetSumOfBondOrdersToHeavyAtoms();
```

Returns sum of bond orders corresponding to all heavy atoms bonded to an *Atom* in a molecule.

GetSumOfBondOrdersToHydrogenAtoms

```
$SumBondOrders = $Atom->GetSumOfBondOrdersToHydrogenAtoms();
```

Returns sum of bond orders corresponding to all hydrogen atoms bonded to an *Atom* in a molecule.

GetSumOfBondOrdersToNonHydrogenAtoms

```
$SumBondOrders = $Atom->GetSumOfBondOrdersToNonHydrogenAtoms();
```

Returns sum of bond orders corresponding to all non-hydrogen atoms bonded to an *Atom* in a molecule.

GetValence

```
$Valence = $Atom->GetValence();
```

Returns valence of an *Atom* in a molecule. Valence corresponds to number of electrons used by an atom in bonding:

```
Valence = ValenceElectrons - ValenceFreeElectrons = BondingElectrons
```

Single, double and triple bonds with bond orders of 1, 2, and 3 correspond to contribution of 1, 2, and 3 bonding electrons. So:

```
Valence = SumOfBondOrders + NumOfMissingHydrogens + FormalCharge
```

where positive and negative values of FormalCharge increase and decrease the number of bonding electrons, respectively.

The current release of MayaChemTools supports the following three valence models, which are used during calculation of implicit hydrogens: MDLValenceModel, DaylightValenceModel, InternalValenceModel or MayaChemToolsValenceModel.

Notes:

```
. Missing hydrogens are included in the valence.
. For neutral molecules, valence and sum of bond orders are equal.
. For molecules containing only single bonds, SumOfBondOrders and
  NumOfBonds are equal.
. Free radical electrons lead to the decrease in valence. For atoms with
  explicit assignment of SpinMultiplicity property values corresponding to
  Singlet (two unparied electrons corresponding to one spin state), Doublet
  (free radical; an unpaired electron corresponding to two spin states),
  and Triplet (two unparied electrons corresponding to three spin states;
  divalent carbon atoms (carbenes)), FreeRadicalElectrons are calculated as
follows:

    SpinMultiplicity: Doublet(2); FreeRadicalElectrons: 1 (one valence
        electron not available for bonding)
    SpinMultiplicity: Singlet(1)/Triplet(3); FreeRadicalElectrons: 2 (two
        valence electrons not available for bonding)
```

GetValenceElectrons

```
$ValenceElectrons = $Atom->GetValenceElectrons();
```

Returns valence electrons for an Atom which corresponds to either explicity set *ValenceElectrons* atom property or valence electrons for the corresponding element in the periodic table available by PeriodicTable module.

GetValenceFreeElectrons

```
$ValenceFreeElectrons = $Atom->GetValenceFreeElectrons();
$ValenceFreeElectrons = $Atom->GetValenceFreeElectrons(
                        $ExcludeFreeRadicalElectrons);
```

Returns valence frees electrons for an Atom in a molecule. It corresponds to:

```
ValenceElectrons - Valence
or
ValenceElectrons - NumOfMissingHydrogens - SumOfBondOrders - FormalCharge
```

Free radical electrons are included in the valence free electrons count by default.

Examples:

```
NH3: ValenceFreeElectrons = 5 - 3 = 5 - 3 - 0 - 0 = 2
NH2: ValenceFreeElectrons = 5 - 3 = 5 - 2 - 1 - 0 = 2
NH4+; ValenceFreeElectrons = 5 - 5 = 5 - 4 - 0 - 1 = 0
NH3+; ValenceFreeElectrons = 5 - 5 = 5 - 3 - 1 - 1 = 0
C(=O)O- : ValenceFreeElectrons on O- = 6 - 0 = 6 - 1 - 0 - (-1) = 6
C(=O)O- : ValenceFreeElectrons on =O = 6 - 2 = 6 - 2 - 0 - 0 = 4
```

GetX

```
$X = $Atom->GetX();
```

Returns value of X-coordinate for an *Atom*.

GetXYZ

```
@XYZ = $Atom->GetXYZ();
$XYZRef = $Atom->GetXYZ();
```

Returns an array or a reference to an array containing values for *Atom* coordinates.

GetXYZVector

```
$XYZVector = $Atom->GetXYZVector();
```

Returns a *Vector* object containing values for *Atom* coordinates

GetY

```
$Y = $Atom->GetY();
```

Returns value of Y-coordinate for an *Atom*.

GetZ

```
$Z = $Atom->GetZ();
```

Returns value of Z-coordinate for an *Atom*.

IsAmideCarbon

```
$Status = $Atom->IsAmideCarbon();
```

Returns 1 or 0 based on whether it's amide carbon *Atom*.

An amide group is defineds as:

```
R-C(=O)-N(-R')-R''
```

where:

```
o R = Hydrogen or groups of atoms attached through carbon
o R' = Hydrogens or groups of atoms attached through carbon or
       hetro atoms
o R'' = Hydrogens or groups of atoms attached through carbon or
       hetro atoms
```

IsAmideNitrogen

```
$Status = $Atom->IsAmideNitrogen();
```

Returns 1 or 0 based on whether it's amide nitrogen *Atom*.

IsAromatic

```
$Status = $Atom->IsAromatic();
```

Returns 1 or 0 based on whether it's an aromatic *Atom*.

IsArsenic

```
$Status = $Atom->IsArsenic();
```

Returns 1 or 0 based on whether it's an arsenic *Atom*.

IsBondedToAtom

```
$Status = $Atom->IsBondedToAtom($OtherAtom);
```

Returns 1 or 0 based on whether *Atom* is bonded to *OtherAtom*.

IsBromine

```
$Status = $Atom->IsBromine();
```

Returns 1 or 0 based on whether it's a bromine *Atom*.

IsCarbon

```
$Status = $Atom->IsCarbon();
```

Returns 1 or 0 based on whether it's a carbon *Atom*.

IsCarboxylCarbon

```
$Status = $Atom->IsCarboxylCarbon();
```

Returns 1 or 0 based on whether it's a carboxyl carbon atom in carboxyl group: R-C(=O)-OH.

IsCarboxylOxygen

```
$Status = $Atom->IsCarboxylOxygen();
```

Returns 1 or 0 based on whether it's a carboxyl oxygen atom in carboxyl group: R-C(=O)-OH.

IsCarboxylateCarbon

```
$Status = $Atom->IsCarboxylateCarbon();
```

Returns 1 or 0 based on whether it's a carboxylate carbon atom in carboxyl group: R-C(=O)-O-.

IsCarboxylateOxygen

```
$Status = $Atom->IsCarboxylateOxygen();
```

Returns 1 or 0 based on whether it's a carboxylate oxygen atom in carboxyl group: R-C(=O)-O-.

IsChlorine

```
$Status = $Atom->IsChlorine();
```

Returns 1 or 0 based on whether it's a chlorine *Atom*.

IsFluorine

```
$Status = $Atom->IsFluorine();
```

Returns 1 or 0 based on whether it's a fluorine *Atom*.

IsFunctionalClassType

```
$Status =$Atom->IsFunctionalClassType($Type);
```

Returns 1 or 0 based on whether it's a specified functional class *Type*.

The current release of MayaChemTools supports following abbreviations and descriptive names for *FunctionalClassType*:

```
HBD: HydrogenBondDonor
HBA: HydrogenBondAcceptor
PI :  PositivelyIonizable
NI : NegativelyIonizable
Ar : Aromatic
Hal : Halogen
H : Hydrophobic
RA : RingAtom
CA : ChainAtom
```

The following definitions are used to determine functional class types: [Ref 60-61, Ref 65-66]:

```
HydrogenBondDonor: NH, NH2, OH
HydrogenBondAcceptor: N[!H], O
PositivelyIonizable: +, NH2
NegativelyIonizable: -, C(=O)OH, S(=O)OH, P(=O)OH
```

IsGuadiniumCarbon

```
$Status = $Atom->IsGuadiniumCarbon();
```

Returns 1 or 0 based on whether it's a guadinium carbon in guadinium group by checking its neighbors for a nitrogen in guadinium group.

IsGuadiniumNitrogen

```
$Status = $Atom->IsGuadiniumNitrogen();
```

Returns 1 or 0 based on whether it's a guadinium nitrogen in guadinium group.

A guadinium group is defined as:

```
R2N-C(=NR)-(NR2) or R2N-C(=NR2+)-(NR2)
```

where:

```
o R = Hydrogens or group of atoms attached through carbon
o Only one of the three nitrogens has a double bond to carbon
  and has optional formal charge allowing it to be neutral or charged state
```

IsHBondAcceptor

```
$Status =$Atom->IsHBondAcceptor();
$Status =$Atom->IsHBondAcceptor($HydrogenBondsType);
```

Returns 1 or 0 based on whether it's a hydrogen bond acceptor *Atom*.

IsHBondDonor

```
$Status =$Atom->IsHBondDonor();
$Status =$Atom->IsHBondDonor($HydrogenBondsType);
```

Returns 1 or 0 based on whether it's a hydrogen bond donor *Atom*.

IsHydrogenBondAcceptor

```
$Status =$Atom->IsHydrogenBondAcceptor();
$Status =$Atom->IsHydrogenBondAcceptor($HydrogenBondsType);
```

Returns 1 or 0 based on whether it's a hydrogen bond acceptor *Atom*.

IsHydrogenBondDonor

```
$Status =$Atom->IsHydrogenBondDonor();
$Status =$Atom->IsHydrogenBondDonor($HydrogenBondsType);
```

Returns 1 or 0 based on whether it's a hydrogen bond donor *Atom*.

The current release of MayaChemTools supports identification of two types of hydrogen bond donor and acceptor atoms with these names:

```
HBondsType1 or HydrogenBondsType1
HBondsType2 or HydrogenBondsType2
```

The names of these hydrogen bond types are rather arbitrary. However, their definitions have specific meaning and are as follows:

```
HydrogenBondsType1 [ Ref 60-61, Ref 65-66 ]:

    Donor: NH, NH2, OH - Any N and O with available H
    Acceptor: N[!H], O - Any N without available H and any O

HydrogenBondsType2 [ Ref 91 ]:

    Donor: NH, NH2, OH - N and O with available H
    Acceptor: N, O - And N and O
```

By default, *HydrogenBondsType1* is used to calculate number hydrogen bond donor and acceptor atoms. *HydrogenBondsType2* corresponds to RuleOf5 definition of hydrogen bond donors and acceptors.

IsHalogen

```
$Status =$Atom->IsHalogen();
```

Returns 1 or 0 based on whether it's a halogen *Atom*.

IsHeteroAtom

```
$Status = $Atom->IsHeteroAtom();
```

Returns 0 or 1 based on whether it's a hetro *Atom*. Following atoms are considered hetro atoms: N, O, F, P, S, Cl, Br, I.

IsHydrogen

```
$Status = $Atom->IsHydrogen();
```

Returns 1 or 0 based on whether it's a hydrogen *Atom*.

IsHydrophobic

```
$Status =$Atom->IsHydrophobic();
```

Returns 1 or 0 based on whether it's a hydrophobic *Atom*.

IsInRing

```
$Status = $Atom->IsInRing();
```

Returns 1 or 0 based on whether *Atom* is present in a ring.

IsInRingOfSize

```
$Status = $Atom->IsInRingOfSize($Size);
```

Returns 1 or 0 based on whether *Atom* is present in a ring of specific *Size*.

IsIodine

```
$Status = $Atom->IsIodine();
```

Returns 1 or 0 based on whether it's an iodine *Atom*.

IsIsotope

```
$Status =$Atom->IsIsotope();
```

Returns 1 or 0 based on whether it's an isotope *Atom*.

IsLipophilic

```
$Status =$Atom->IsLipophilic();
```

Returns 1 or 0 based on whether it's a lipophilic *Atom*.

IsMetallic

```
$Status = $Atom->IsMetallic();
```

Returns 1 or 0 based on whether it's a metallic *Atom*.

IsNegativelyIonizable

```
$Status =$Atom->IsNegativelyIonizable();
```

Returns 1 or 0 based on whether it's a negatively ionizable atom *Atom*.

IsNitrogen

```
$Status = $Atom->IsNitrogen();
```

Returns 1 or 0 based on whether it's a nitrogen *Atom*.

IsNonCarbonOrHydrogen

```
$Status =$Atom->IsNonCarbonOrHydrogen();
```

Returns 1 or 0 based on whether it's not a carbon or hydrogen *Atom*.

IsNotInRing

```
$Status = $Atom->IsNotInRing();
```

Returns 1 or 0 based on whether *Atom* is not present in a ring.

IsOnlyInOneRing

```
$Status = $Atom->IsOnlyInOneRing();
```

Returns 1 or 0 based on whether *Atom* is only present in one ring.

IsOxygen

```
$Status = $Atom->IsOxygen();
```

Returns 0 or 1 based on whether it's an oxygen *Atom*.

IsPhosphorus

```
$Status = $Atom->IsPhosphorus();
```

Returns 0 or 1 based on whether it's a phosphorus *Atom*.

IsPhosphateOxygen

```
$Status = $Atom->IsPhosphateOxygen();
```

Returns 1 or 0 based on whether it's a phosphate oxygen in phosphate group.

A phosphate group is defined as:

```
AO-(O=)P(-OA)-OA
```

Where:

```
        A - Any group of atoms including hydrogens
```

IsPhosphatePhosphorus

```
        $Status = $Atom->IsPhosphatePhosphorus();
```

Returns 1 or 0 based on whether it's a phosphate phosphorus in phosphate group.

IsPolarAtom

```
        $Status = $Atom->IsPolarAtom();
```

Returns 0 or 1 based on whether it's a polar *Atom*. Following atoms are considered polar atoms: N, O, P, S.

IsPolarHydrogen

```
        $Status = $Atom->IsPolarHydrogen();
```

Returns 0 or 1 based on whether it's a hydrogen *Atom* bonded to a polar atom.

IsPositivelyIonizable

```
        $Status =$Atom->IsPositivelyIonizable();
```

Returns 1 or 0 based on whether it's a positively ionizable *Atom*.

IsSaturated

```
        $Status = $Atom->IsSaturated();
```

Returns 1 or 0 based on whether it's a saturated *Atom*. An atom attached to other atoms with only single bonds is considered a saturated atom.

IsSelenium

```
        $Status = $Atom->IsSelenium();
```

Returns 0 or 1 based on whether it's a selenium *Atom*.

IsStereoCenter

```
        $Status = $Atom->IsStereoCenter();
```

Returns 0 or 1 based on whether it's marked as a stero center *Atom* by explicit setting of *StereoCenter* atom propert to value of *1*.

IsSilicon

```
        $Status = $Atom->IsSilicon();
```

Returns 0 or 1 based on whether it's a silicon *Atom*.

IsSulfur

```
        $Status = $Atom->IsSulfur();
```

Returns 0 or 1 based on whether it's a sulfur *Atom*.

IsSulphur

```
        $Status = $Atom->IsSulphur();
```

Returns 0 or 1 based on whether it's a sulfur *Atom*.

IsTellurium

```
        $Status = $Atom->IsTellurium();
```

Returns 0 or 1 based on whether it's a tellurium *Atom*.

IsTerminal

```
        $Status = $Atom->IsTerminal();
```

Returns 0 or 1 based on whether it's a terminal *Atom* attached to no more than one non-hydrogen atom.

IsUnsaturated

```
$Status = $Atom->IsUnsaturated();
```

Returns 1 or 0 based on whether it's as unsaturated *Atom*. An atom attached to other atoms with at least one non-single bond is considered an unsaturated atom.

IsTopologicalPharmacophoreType

```
$Status =$Atom->IsTopologicalPharmacophoreType();
```

Returns 1 or 0 based on whether it's any of the supportyed topological pharmacophore *Atom* type. See *IsFunctionalClassType* for a list of supported types.

SetAtomSymbol

```
$Atom->SetAtomSymbol($AtomicSymbol);
```

Sets atom symbol for *Atom* and returns *Atom* object. The appropriate atomic number is also set automatically.

SetAtomicNumber

```
$Atom->SetAtomicNumber($AtomicNumber);
```

Sets atomic number for *Atom* and returns *Atom* object. The appropriate atom symbol is also set automatically.

SetMassNumber

```
$Atom->SetMassNumber($MassNumber);
```

Sets mass number for *Atom* and returns *Atom* object.

SetStereoCenter

```
$Atom->SetStereoCenter($StereoCenter);
```

Sets stereo center for *Atom* and returns *Atom* object.

SetStereochemistry

```
$Atom->SetStereochemistry($Stereochemistry);
```

Sets stereo chemistry for *Atom* and returns *Atom* object.

SetX

```
$Atom->SetX($Value);
```

Sets X-coordinate value for *Atom* and returns *Atom* object.

SetXYZ

```
$Atom->SetXYZ(@XYZValues);
$Atom->SetXYZ($XYZValuesRef);
$Atom->SetXYZ($XYZVector);
```

Sets *Atom* coordinates using an array, reference to an array or a *Vector* object and returns *Atom* object.

SetY

```
$Atom->SetY($Value);
```

Sets Y-coordinate value for *Atom* and returns *Atom* object.

SetZ

```
$Atom->SetZ($Value);
```

Sets Z-coordinate value for *Atom* and returns *Atom* object.

StringifyAtom

```
$AtomString = $Atom->StringifyAtom();
```

Returns a string containing information about *Atom* object.

AUTHOR

Manish Sud <msud@san.rr.com>

SEE ALSO

Bond.pm, Molecule.pm, MoleculeFileIO.pm

COPYRIGHT