

NAME

Bond

SYNOPSIS

```
use Bond;

use Bond qw(:all);
```

DESCRIPTION

Bond class provides the following methods:

`new`, `Copy`, `DeleteBond`, `GetAtoms`, `GetBondedAtom`, `GetCommonAtom`, `GetLargestRing`, `GetNumOfRings`, `GetNumOfRingsWithEvenSize`, `GetNumOfRingsWithOddSize`, `GetNumOfRingsWithSize`, `GetNumOfRingsWithSizeGreaterThan`, `GetNumOfRingsWithSizeLessThan`, `GetRings`, `GetRingsWithEvenSize`, `GetRingsWithOddSize`, `GetRingsWithSize`, `GetRingsWithSizeGreaterThan`, `GetRingsWithSizeLessThan`, `GetSizeOfLargestRing`, `GetSizeOfSmallestRing`, `GetSmallestRing`, `IsAromatic`, `IsInRing`, `IsInRingOfSize`, `IsNotInRing`, `IsOnlyInOneRing`, `SetAtoms`, `SetBondOrder`, `SetBondType`, `SetID`, `SetStereochemistry`, `StringifyBond`

Bond 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`

```
$NewBond = new Bond([%PropertyNameAndValues]);
```

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

```
ID = SequentialObjectID
@Atoms = ();
BondType = ""
BondOrder = ""
```

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

Examples:

```
$Bond = new Bond();
$DoubleBond = new Bond('Atoms' => [$Atom2, $Atom1],
                        'BondOrder' => 2);
```

`Copy`

```
$BondCopy = $Bond->Copy();
```

Copy *Bond* and its associated data using `Storable::dclone` and return a new *Bond* object

`DeleteBond`

```
$Bond->DeleteBond();
```

Delete *Bond* between atoms in from a molecule.

`GetAtoms`

```
@BondedAtoms = $Bond->GetAtoms();
```

Returns an array containing *Atom* objects involved in *Bond*

`GetBondedAtom`

```
$BondedAtom = $Bond->GetBondedAtom($Atom);
```

Returns *BondedAtom* object bonded to *Atom* in *Bond*

`GetCommonAtom`

```
$CommonAtom = $Bond->GetCommonAtom($OtherBond);
```

Returns *Atom* object common to bonds *Bond* and *\$OtherBond*

`GetLargestRing`

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

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

GetNumOfRings

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

Returns number of rings containing *Bond* in a molecule

GetNumOfRingsWithEvenSize

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

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

GetNumOfRingsWithOddSize

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

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

GetNumOfRingsWithSize

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

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

GetNumOfRingsWithSizeGreaterThan

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

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

GetNumOfRingsWithSizeLessThan

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

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

GetRings

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

Returns an array of references to arrays containing ring atoms corresponding to all rings containing *Bond* in a molecule

GetRingsWithEvenSize

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

Returns an array of references to arrays containing ring atoms corresponding to all rings with even size containing *Bond* in a molecule

GetRingsWithOddSize

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

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

GetRingsWithSize

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

Returns an array of references to arrays containing ring atoms corresponding to all rings with specific *RingSize* containing *Bond* in a molecule

GetRingsWithSizeGreaterThan

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

Returns an array of references to arrays containing ring atoms corresponding to all rings with size greater than specific *RingSize* containing *Bond* in a molecule

GetRingsWithSizeLessThan

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

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

GetSizeOfLargestRing

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

Returns size of the largest ring containing *Bond* in a molecule

GetSizeOfSmallestRing

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

Returns size of the smallest ring containing *Bond* in a molecule

GetSmallestRing

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

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

IsAromatic

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

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

IsInRing

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

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

IsInRingOfSize

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

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

IsNotInRing

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

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

IsOnlyInOneRing

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

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

SetAtoms

```
$Bond->SetAtoms($AtomsRef);
$Bond->SetAtoms(@Atoms);
```

Set atoms of *Bond* to atoms in *Atoms* array or in a reference to an array of atoms and return *Bond*

SetBondOrder

```
$Bond->SetBondOrder($BondOrder);
```

Sets bond order of *Bond* to specified *BondOrder* and returns *Bond*. Possible bond order values: 1 = Single, 1.5 = Aromatic, 2 = Double, 3 = Triple, 4 = Quadruple.

Notes:

- o *BondType* property is automatically assigned using default *BondType* values for specified *BondOrder*.
- o *BondType* values can also be explicit set.
- o To make bonds aromatic in a ring, explicitly set "Aromatic" property for bond/atoms and make sure appropriate *BondOrder* values are assigned.
- o Dative bond types are treated as single bond types with explicit formal charge of + and - on first and second bond atoms.

SetBondType

```
$Bond->SetBondType($BondType);
```

Sets bond type for *Bond* to specified *BondType* and returns *Bond*. Possible bond type values for different bond orders are:

```
1 : Single, SingleWedge (Up), SingleHash (Down), SingleWavy, Dative
2 : Double, DoubleCross (Cis/Trans)
3 : Triple
4 : Quadruple
1.5 : Aromatic (Single/Double), Tautomeric(Single/Double)
```

Notes:

- o *BondOrder* property is automatically assigned using default *BondOrder* values for specified *BondType*.

SetStereochemistry

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

Sets bond stereochemistry for *Bond* to specified *Stereochemistry* and returns *Bond*. Possible *Stereochemistry* values: *Z*, *cis*, *E*, *trans*

StringifyBond

```
$BondString = $Bond->StringifyBond();
```

Returns a string containing information about *Bond* object

AUTHOR

Manish Sud <msud@san.rr.com>

SEE ALSO

Atom.pm, Molecule.pm

COPYRIGHT

Copyright (C) 2004-2008 Manish Sud. All rights reserved.

This file is part of MayaChemTools.

MayaChemTools is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.