DRAFT CHAPTER OF THE OFFICIAL PyMOL MANUAL (For PyMOL Sponsors)

A new installment

This chapter is part of a comprehensive manual-in-progress, so you will find references to chapters that are not included here. Nonetheless, we think you may find it helpful, especially if you are a new user. Please email help@schrodinger.com if you find it in any way confusing or incomplete.

Only for sponsors

This is an incentive product for PyMOL sponsors. Please do not post it publicly or otherwise share it with the general public. Incentive products, such as this manual, are exclusively for sponsors, and sponsors are what make possible PyMOL's continued development, documentation, and support. To confirm or inquire about sponsorship, please email sales@pymol.org.

Chapter ES. Expressing Selections

***PyMOL affords the user great flexibility in applying commands by providing a wide variety of selection-expressions. By covering the range of possible selection-expressions, this chapter completes the tour of PyMOL commands begun in Chapter TC. The information here corresponds to* PyMOL version 1_2r1.**

Contents

Selecting Atoms

Single-word selectors

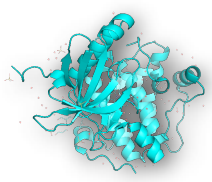
As we've seen, selection-expressions are frequently arguments to PyMOL commands. The simplest selection-expressions are single words. While many selection-expressions themselves take arguments, single-word selectors are complete without them.

Table ES.1 shows several single-word selectors that have short forms to save on typing. Some short forms must be followed by a period and a space, so that PyMOL can recognize them. Short and long forms have the same effect, so use the form you prefer.

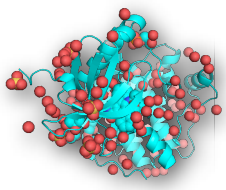
Table ES.1 Single-word selectors

Single-word Selector	Short Form	Description
`all`	`*`	All atoms currently loaded into PyMOL
`hetatm`	`het`	All atoms loaded from Protein Data Bank HETATM records
`hydro`	`h.`	All hydrogen atoms currently loaded into PyMOL
`none`		No atoms (empty selection)
`present`	`pr.`	All atoms with defined coordinates in the current state (used in creating movies)
`visible`	`v.`	All atoms in enabled objects with at least one visible representation

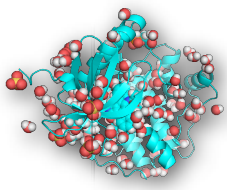
`PyMOL> color blue, all`	is equivalent to	`PyMOL> color blue, *`
`PyMOL> hide spheres, hydro`		`PyMOL> hide spheres, h.`
`PyMOL> show spheres, hetatom`		`PyMOL> show spheres, het`



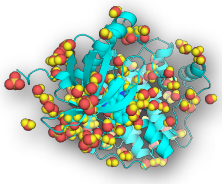
`as cartoon`



`show spheres, het`



`h_add hetatm`



`color yellow, h.`

Property selectors are also used to complete selection-expressions.

Matching property identifiers to select atoms

PyMOL reads data files written in PDB, MOL/SDF, Macromodel, ChemPy Model, and Tinker XYZ formats. Several data fields in these formats are used in selection-expressions, including property identifiers (see Figure ES.1 and Table ES.2).

For example, the selector `resi`, followed by a list of residue identifiers, is a selection-expression, as in

```
PyMOL> origin resi 1
```

Lists of residue identifiers may contain one entry (as in `resi 1`) or more. When listing more than one entry, separate them using plus signs (no spaces are allowed), as in

```
PyMOL> center resi 1+2+3
```

The list may be take the form of a range, given with a dash (again, no spaces),

```
PyMOL> orient resi 1-3
```

The list may include ranges and individual residue identifiers, as in

```
PyMOL> zoom resi 100-110+90+82
```

The identifier for a blank field in an input file is an empty pair of quotes, as in

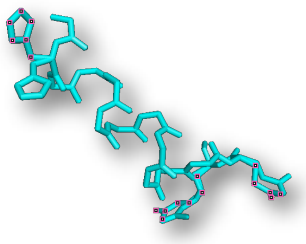
```
PyMOL> color yellow, ss ""
```

`ss ""` selects all atoms that are not assigned a secondary structure (see Table ES.2).

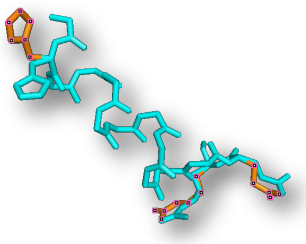
Atom selections can span multiple objects. For example, if several structure files have been fetched or loaded, a selection such as

```
PyMOL> select name ca
```

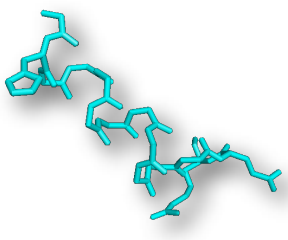
includes all the atoms named C-alpha in all of the loaded objects.



`select alts, alt a`



`color orange, alt a`



`hide sticks, alt a`

Figure ES.1 The `altloc` (`alt`, for short) data field can be used in PyMOL selection-expressions.

Table ES.2 Selectors for matching property identifiers

Selector	Short Form	Macro Form*	Function and Example
`altloc`	`alt`		Selects atoms by matching the alternative location identifiers (single letters) in the source file `PyMOL> select altconf, alt a+""`
`chain`	`c.`	`A+B//`	Selects atoms by matching the chain identifiers (single letters or numbers) in the source file `PyMOL> select firstch, chain a`
`id`	`id`		Selects atoms by matching the atom identifiers (integers) in the source file `PyMOL> select idno, id 23`
`index`	`index`		Selects atoms by matching their internal PyMOL object atom indices (integers starting with 1 for the first atom in the source file) `PyMOL> select firstreadgroup, index 1-100`
`name`	`n.`	`*/CA+CB`	Selects atoms by matching the code names (1-, 2-, 3-, or 4-letter) in the source file `PyMOL> select carbons, name ca+cb+cg+cd`

`resname` or `resn`	`r.`	`asp+glu/`	Selects atoms by matching the residue names (3-letter codes for amino acids and 1- or 2-letter codes for nucleic acids) in the source file
			`PyMOL> select aas, resn asp+glu+asn+gln`
			`PyMOL> select bases, resn a+g`

`resi`, `resid`, `residue`, or `resIdent`	`i.`	`1-10/`	Selects atoms by matching the sequence number residue identifiers (1-, 2-, 3- or 4-digit integers) in the source file, using lists,
			`PyMOL> select multis10, resi 1+10+100+1000`
			and ranges,
			`PyMOL> select nterm, i. 1-10`

`segment` or `segi`	`s.`	`lig///`	Selects atoms by matching the segment names (1-, 2-, 3-, or 4-letters) in the source file `PyMOL> select ligand, segi lig` Segment identifiers are used when multiple copies of an identical set of chains exist inside of a single molecular object. They also come in handy for complex proteins that have more chains than can represented with a single-character alphanumeric identifier.

`symbol` or `element`	`e.`		Selects atoms by matching the atomic element symbols (1- or 2-letter chemical symbols from the periodic table) in the source file
			`PyMOL> select polar, symbol o+n`
			`PyMOL> select sulfurs, element s`

*see Atom Selection Macros

Note that the PyMOL selector `name` refers to the atom names provided in the name field of input files: `ca`, `cb`, etc. We use the word "name" differently in the next section. Details of the atom and residue formats can be found in the official guide to PDB file formats, http://www.rcsb.org/pdb/docs/format/pdbguide2.2/guide2.2_frame.html.

Figure CS.2 is a useful reference for atom and bond names in proteins.

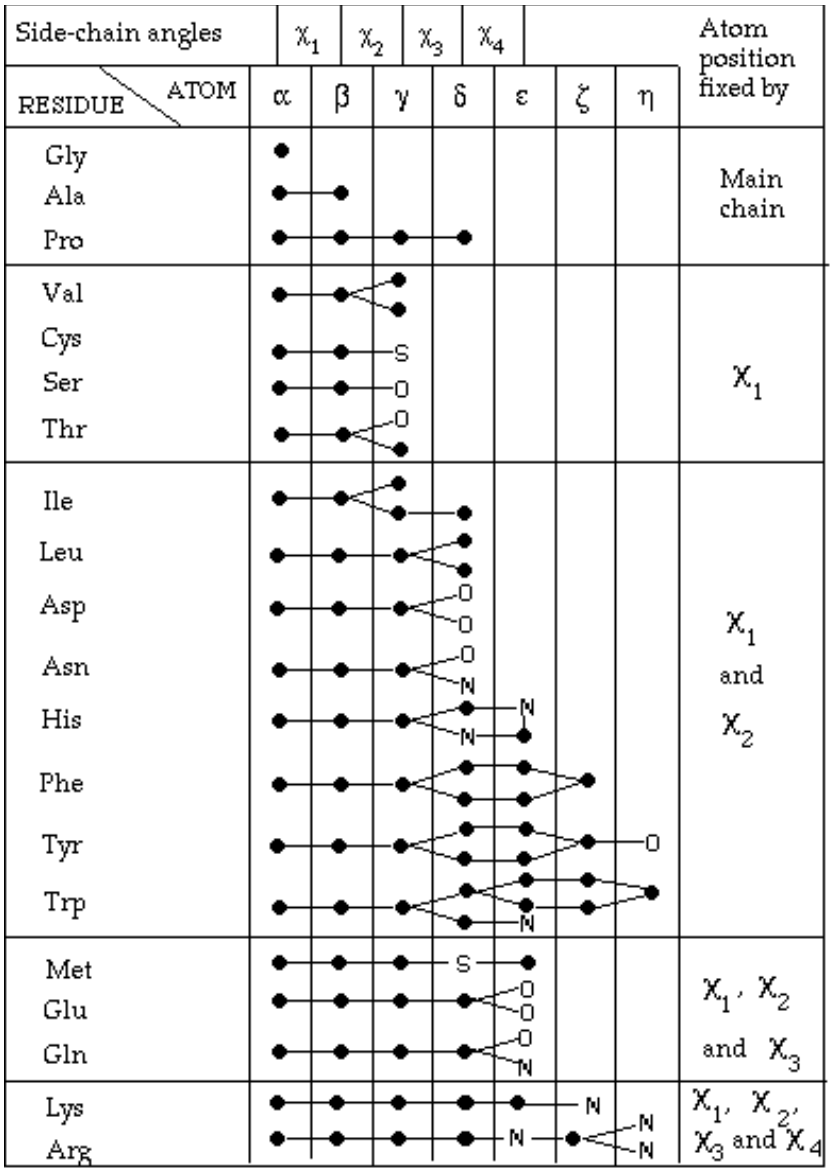


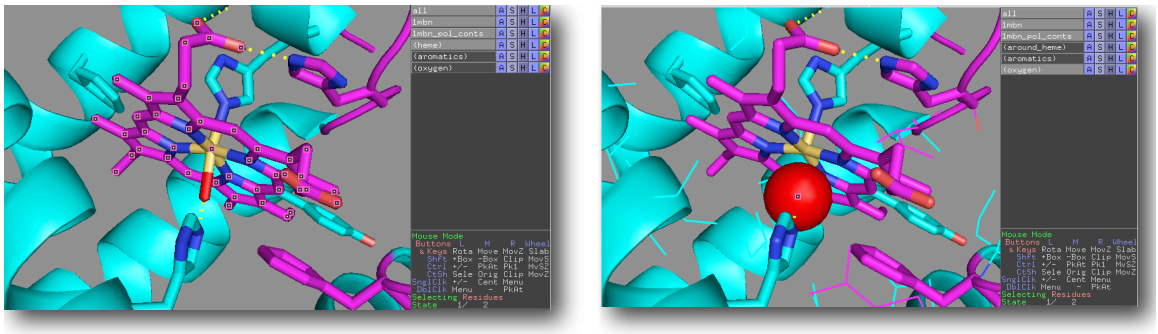
Figure ES.2 Key to protein sidechain atom and bond names
(Ponder and Richards, *Internal packing and protein structural classes*, Cold Spring Harb Symp Quant Bio. 1987).

Named atom selections

You can name selections to facilitate their re-use, or you can specify them anonymously (without names) as we did in the last section. To give names to selections, you can use the syntax given here, or you can use the renaming facility of the objects & selections menus. Object and selection names may include the characters A/a to Z/z, numerals 0 to 9, the dash (-), and the underscore character (_). Characters to avoid include: ! @ # $ % ^ &* ( ) ' " [ ] { } \ | ~ ` <> . ? /

SYNTAX:	`select selection-name, selection-expression`
	The selection-name and selection-expression are both arguments to select, so they are separated by a comma.
EXAMPLE:	`PyMOL> select met118, resi 118`
	The selection is created and appears in the objects & selections list with the name **(met118)**.

Figure ES.3 demonstrates the use of named-selections in typed commands to create Figure X.17 from figure X.16.



ES.3 Figure X.17 (right) is created from Figure X.16 (left) using three typed commands and named-selections.

```
PyMOL> select around_heme, heme expand 6.0
PyMOL> show lines, around_heme
PyMOL> as spheres, oxygen
```

Named-selections will continue to work after you make changes to the object they were selected from. For example, suppose you fetch a file containing the structure of a peptide, and type

```
PyMOL> select bb, name c+o+n+ca
```

```
PyMOL> count_atoms bb
```

Now PyMOL counts the atoms in the named-selection **(bb)** and returns, let's say, 52 atoms in the command history window. Now type

```
PyMOL> remove resi 5
```

The typed command `remove` works on objects, just as the menu choice **A** Action / **remove atoms** does. The atoms of `resi 5` are removed from PyMOL's memory. If you recount the atoms in the selection **(bb)**,

```
PyMOL> count_atoms bb
```

PyMOL now returns a smaller number of atoms.

While atoms can be `removed` from selections, atoms subsequently added to a structure after a selection is made will not be included in the pre-existing selection. (In programming terms, this is to say that selections are *static*.) Only atoms that exist *at the time the selection is defined* (and that still exist in PyMOL's memory) are included in the selection. For example, make the following selection with any structure enabled in a PyMOL session.

```
PyMOL> select static_demo, all
```

Count the atoms in the selection

```
PyMOL> count_atoms static_demo
```

This time, PyMOL counts the atoms in the named-selection **(static_demo)** and returns, let's say, 502 atoms in the command history window. Now type

```
PyMOL> h_add
```

PyMol adds hydrogens in the appropriate places. The command `h_add` takes **all** as its default argument. Compare the number of atoms in the selection **(static_demo)** with the number of atoms in **all**.

```
PyMOL> count_atoms static_demo

PyMOL> count_atoms all
```

The selection **(static_demo)** doesn't contain the added hydrogens, but the object **all** does.

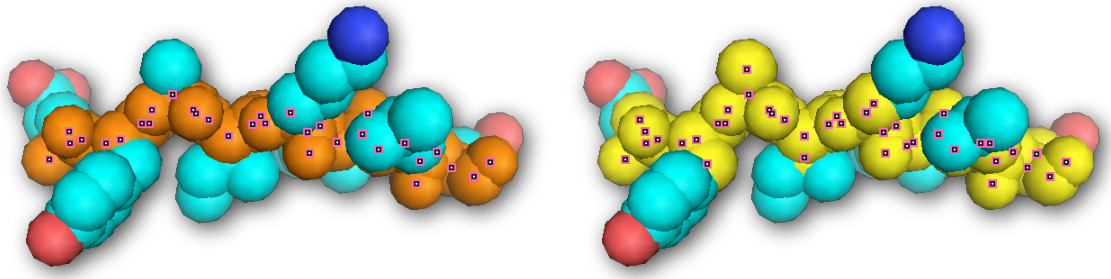
Named-selections can be used as arguments to subsequent atom selections. By using a named-selection as an argument to a subsequent selection, you can add atoms to selections using typed commands. Let's start again by making the named-selection **(bb)**,

```
PyMOL> select bb, name c+o+n+ca
```

Now use `bb` as an argument to a subsequent selection

```
PyMOL> select bigbb, bb or name cb
```

As shown in Figure ES.4, all atoms named C, O, N, C-alpha or C-beta are now selected. Note that the keyword `or` is used to select all atoms in the two groups, `bb` and `name cb`. Joining `bb` and `name cb` with the logical operator `and` would have selected no atoms because logical operators are interpreted in their boolean sense, not in their natural language sense. See the subsection on Selection Algebra below.



```
select bb, name c+o+n+ca
color orange, bb
```

```
select bigbb, bb or name cb
color yellow, bigbb
```

Figure ES.4 The list of atoms in the selection made and colored on the left is expanded, renamed, and recolored by the typed commands on the right.

Named-selections created by typing perform identically to named-selections created by clicking. You can add to them by toggling them to the active state (clicking on their names in the objects & selections list) and then left clicking on atoms or residues (etc) in the Display Area.

Matching input property values to select atoms

Source data files (PDB, MOL/SDF, Macromodel, ChemPy Model, and Tinker XYZ) report atom properties, in addition to conferring atom identifiers. PyMOL's typed commands allow you to select lists of atoms according to their numeric property values, which are read in from the data input (see Table ES.3).

Table ES.3 Selectors for matching numeric property identifiers

Numeric Range Selector	Short Form	Function and `Example`*

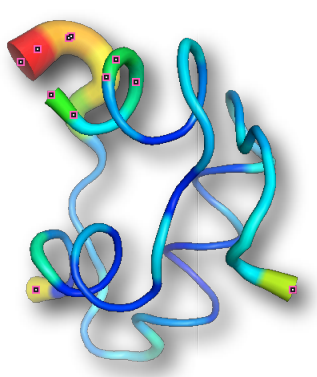
`b`		Selects atoms based on the temperature factors (real numbers) in the source file
		`PyMOL> select fuzzy, b > 30`
		`PyMOL> select crisp, b < 10`
		`PyMOL> select defined, (b = 0)`

`formal_charge`	`fc.`	Selects atoms based on formal charges (integers) in the source file
		`PyMOL> select fposq, formal_charge > 0`
		`PyMOL> select fnegq, fc. < 0`
		`PyMOL> select fung, (fc. = 1)`

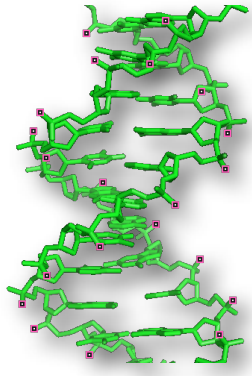
`numeric_type`	`nt.`	Selects atoms based their numeric type-numbers (integers) in MOL2 and similar source files** `PyMOL> select numtype5, numeric_type 5`

`partial_charge`	`pc.`	Selects atoms based on assigned partial charges in MOL2, pqr and similar source files**
		`PyMOL> select posq, partial_charge > 0.0`
		`PyMOL> select negq, pc. < -0.25`
		`PyMOL> select ung, (pc. = 0)`

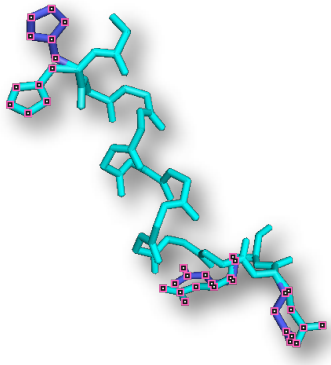
`q`		Selects atoms based on the occupancy values (real numbers) in the source file
		`PyMOL> select ghostly, q > 0.5`
		`PyMOL> select ambly, q < 1.0`
		`PyMOL> select definitely, (q = 1)`



`select b > 30`



`select fc. < 0`



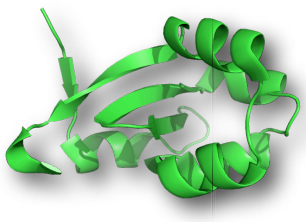
`select q < 1.0`

*Parentheses around expressions containing equal signs keeps PyMOL from assigning the values to the variables.
**PDB files do not include these property identifiers

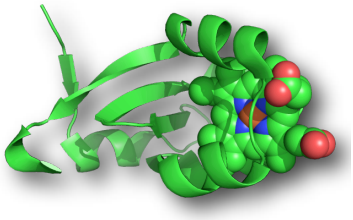
Not all input property values are numeric. Table ES.4 describes selections you can type using non-numeric input property values.

Table ES.4 Non-numeric input property selectors

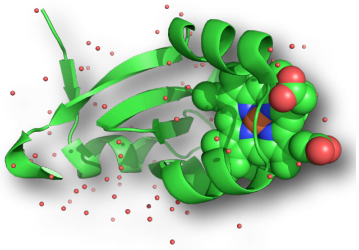
Selector	Function and `Example`
`bonded`	Selects atoms based on their participation in one or more bonds (This selector is useful for hiding atomic ions and single-atom solvent molecules.) `PyMOL> select singletons, not bonded`
`hetatm`	Selects atoms based on whether they are classified as PDB HETATMs (This selector is useful for locating nonstandard residues, ligands, solvent, salt, or modified amino acids and nucleic acids.) `PyMOL> select notable, hetatm`
`ss`	Selects atoms based on their assigned secondary structures (Helical regions are assigned `h`, beta strands are assigned `s`, and loops are assigned `l`, or they are blank, `""`.) `PyMOL> select allss, ss h+s+l+""`
`text_type`	Selects atoms based their textual types in MOL2 and similar input files (PDB files do not include this information) `PyMOL> select CTtype, text_type CT`



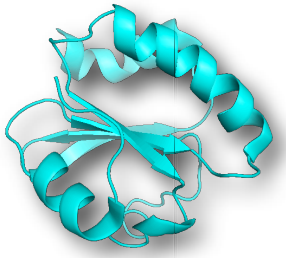
`as cartoon 1m2I`



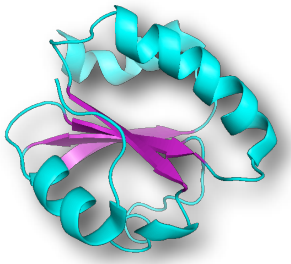
`show spheres, hetatm & bonded`



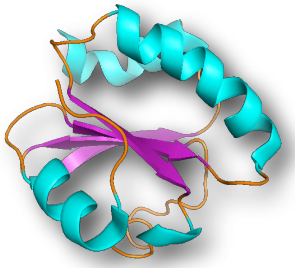
`show nb_spheres, hetatm & not bonded`



`as cartoon 1rqm`



`color purple, ss s`



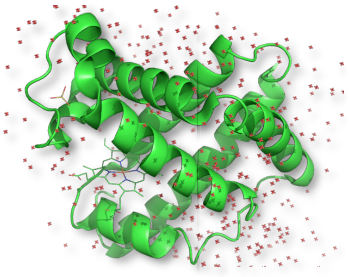
`color orange, ss ""`

Matching PyMOL properties to select atoms

In addition to the properties read in from input files, PyMOL assigns properties to atoms according their chemical categories, shown in Table ES.5, and according to to how they are currently represented, shown in Table ES.6.

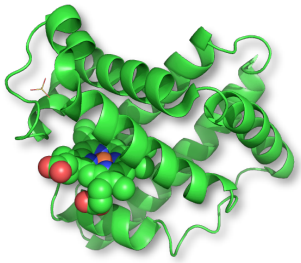
Table ES.5 Chemical categories

Selector	Function and `Example`
`polymer`	selects input atoms in polypeptide and poly-DNA residues `PyMOL> select protein, polymer`
`solvent`	selects input atoms in HOH, WAT, H20, TIP, and SOL "residues" `PyMOL> select waters, solvent`
`organic`	selects atoms in carbon-containing molecules other than recognized* polymers `PyMOL> select lig, organic`
`inorganic`	selects atoms in non-solvent molecules that do not contain carbon `PyMOL> select ions, inorganic`
`hydrogens`	selects hydrogens `PyMOL> select h-placed, hydrogens`
`donors`	selects hydrogen bond donors if molecular valences are specified in the input file `PyMOL> select may_share_h, donors`
`acceptors`	selects hydrogen bond acceptors if molecular valences are specified in the input file `PyMOL> select may_accept_h, acceptors`

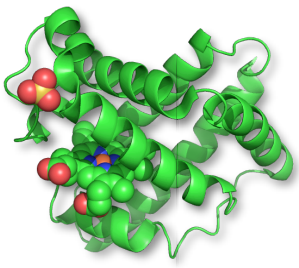


`as cartoon, polymer`

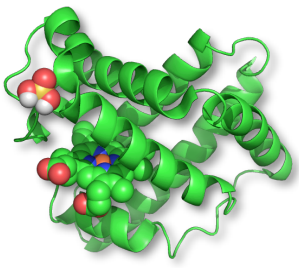
`remove solvent`



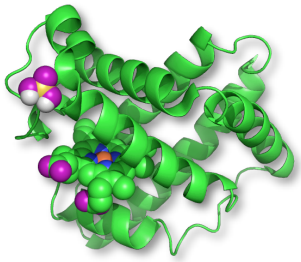
`show spheres, organic`



`show spheres, inorganic`



`h_add donors`



`color purpleblue, acceptors`

*PyMOL recognizes only polypeptides and polynucleotides as polymers

Table ES.6 Representation property selectors

Selector	Short Form	Function and Example
color		Selects atoms based on their current colors PyMOL> select carbons, color green
enabled		Selects atoms based on their inclusion in currently enabled objects PyMOL> select mygroup, enabled
object or model	o. or m.	Selects atoms based on their inclusion in a given object, PyMOL> select myobj, m. 1dn2
present		Selects atoms based on their existence with defined coordinates in the current state PyMOL> select defined, present
state		Selects atoms based on their existence with defined coordinates in the indicated state. PyMOL> select my2, state 2
visible		Selects atoms that are shown in the Display Area PyMOL> select shown, visible PyMOL> select hidden, not visible

Logical selection operators

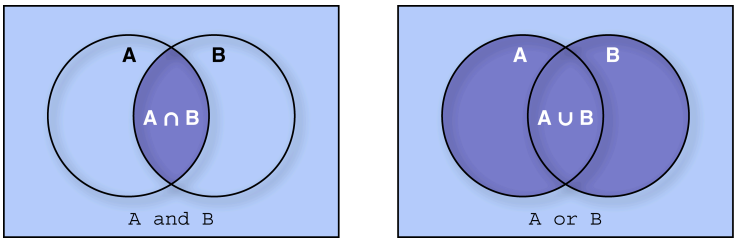
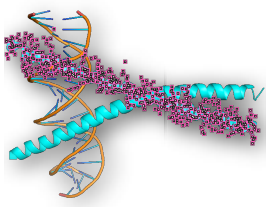
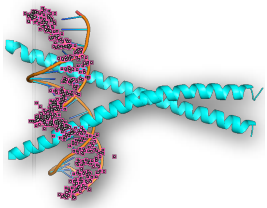
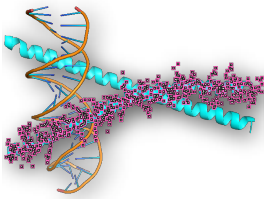
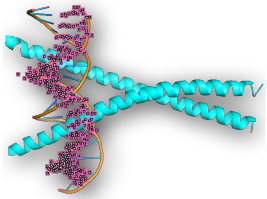
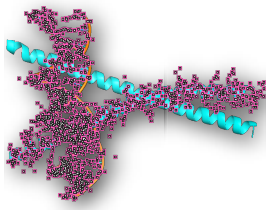
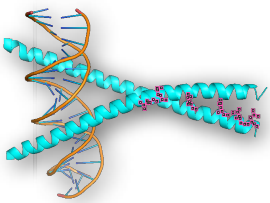
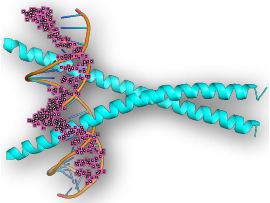
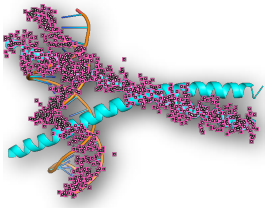


Figure ES.5 The logical operator AND (left) selects only items that belong to both A and B. The logical operator OR (right) selects items that belong to either A or B.

Selections can combined with the boolean logical operators AND, OR and NOT. The boolean AND selects only those items that have all of the named properties, and the boolean OR selects items that have any of them. Venn diagrams in Figure ES.5 show that AND selects the areas of overlap, while OR selects both areas. Table CS.7 shows PyMOL's logical and similarity selection operators.

Table ES.7 Logical and similarity selection operators

Operator*	Short form*	Function and `Example`
`not s1`	`! s1`	Selects atoms that are not included in `s1` `PyMOL> select sidechains, ! bb`
`s1 and s2`	`s1 & s2`	Selects atoms included in both `s1` and `s2` `PyMOL> select my-cas, resi 10-20 and name ca`
`s1 in s2`	`s1 in s2`	Selects atoms in `s1` whose identifiers `name`, `resi`, `resn`, `chain` and `segi` all match atoms in `s2`. This is useful in applying a selection from one object to a separate (partial or complete) copy of that object `PyMOL> select same_atms, pept in prot`
`s1 like s2`	`s1 l. s2`	Selects atoms in `s1` whose identifiers `name` and `resi` match atoms in `s2` `PyMOL> select similar_atms, pept like prot`
`s1 or s2`	`s1 | s2`	Selects atoms included in either `s1` or `s2` `PyMOL> select polars, elem N or elem O`
 `chain a`		 `chain b`
 `chain c`		 `chain d`
 `!chain a`		 `resn leu & chain a`
 `chain b like chain d`		 `chain a or chain b`

*`s1` and `s2` are dummy variables that stand for selection-expressions such as `chain a` or `hydro`.

Logical selections can be combined. For example, you can select atoms that are included in chain a, but are not included in residue 125, by typing

```
PyMOL> select chain a and (not resi 125)
```

Typing a selector with a list of arguments joined by plus (+) signs, as in

```
PyMOL> select name cb+cg1+cg2 and chain A
```

is equivalent to repeating the selector with each argument in the list, and joining the expressions with the `or` operator, as in

```
PyMOL> select (name cb or name cg1 or name cg2) and chain A
```

Both commands select the C-beta, C-gamma-1 and C-gamma-2 atoms in chain A.

Like arithmetic operators, logical operators create results that can depend on which operation is performed first. They have an order of precedence. To ensure that the operations are performed in the order you have in mind, use parentheses. PyMOL will expand its logical selection out from the innermost parentheses. For example

```
PyMOL> select (chain a or (chain b and (not resi 31-45)))
```

selects first among atoms that are not in residue number `31-45`. Of those atoms, it chooses those which are also in `chain b`. Next it adds all atoms in `chain a` (see Figure ES.6).

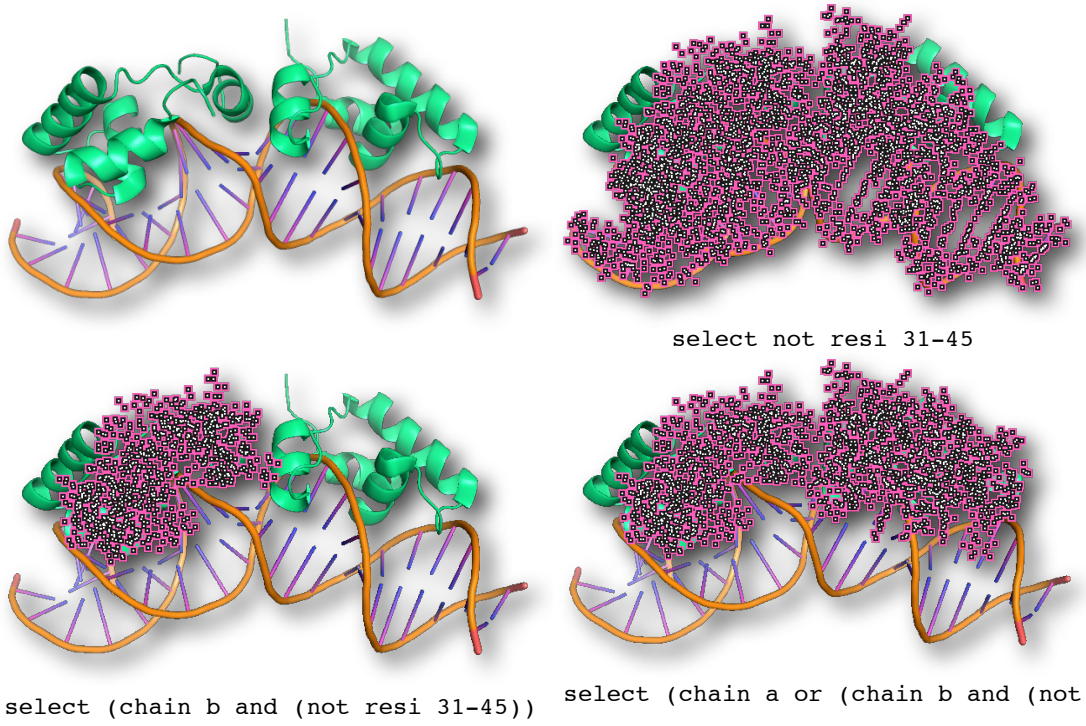
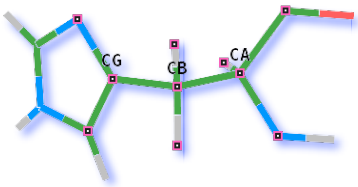
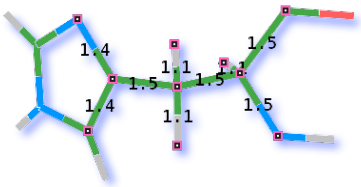
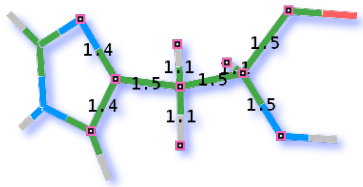
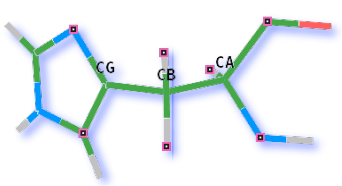
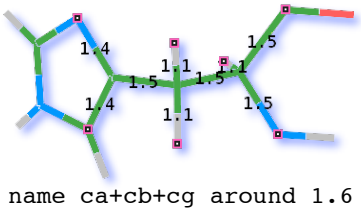
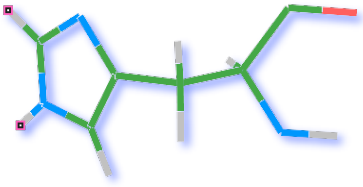


Figure ES.6

Proximity operators

To help you examine atoms according to their bonding distances or distances in space, PyMOL provides the proximity operators given in Table ES.8.

Table ES.8 Geometric and chemical proximity operators

Operator	Short form	Function and `Example`
`bound_to s1`	`bto. s1`	Selects atoms within `s1` and outside of `s1` that are bound to atoms within `s1` `PyMOL> select disulfides, CYS/SG and bound_to CYS/SG`
`s1 around X`	`s1 a. X`	Selects atoms outside of `s1` with centers within `X` Å of the center of any atom within `s1` `PyMOL> select near_ten, resi 10 around 5`
`s1 beyond X of s2`		Selects atoms in `s1` with centers beyond `X` Å from centers of atoms in `s2` `PyMOL> select c. B beyond 5.0 of c. A`
`s1 expand X`	`s1 e. X`	Expands `s1` to select `s1` and atoms with centers within `X` Å of the center of any atom in `s1` `PyMOL> select near_ten_x, near10 expand 3`
`s1 extend X`		Enlarges `s1` to select `s1` and atoms within `X` bonds of any atom in `s1` `PyMOL> select morethanresi12, (resi 12 extend 3)`
`neighbor s1`	`nbr. s1`	Selects atoms directly bonded to `s1` `PyMOL> select vicinos, neighbor resi 10`
`s1 gap X`		Selects atoms whose van der Waals radii are separated from the van der Waals radii of `s1` by a minimum distance of `X` Å. `PyMOL> select farfrm_ten, resi 10 gap 5`
`s1 near_to X of s2`		Selects atoms in `s1` (and not in `s2`) with centers that are within `X` Å of the centers of atoms in `s2` `PyMOL> select nrmol, polymer near_to 5.0 of chain a`
`s1 within X of s2`	`s1 w. X of s2`	Selects atoms in `s1` with centers that are within `X` Å `of` the centers of atoms in `s2` `PyMOL> select bbnearten, bb w. 4 of resi 10`
 `bto. name ca+cb+cg`		
 `name ca+cb+cg expand 1.6`		
 `name ca+cb+cg around 1.6`		
 `nbr. name ca+cb+cg`		
 `name ca+cb+cg around 1.6`		
 `name cb gap 1.6`		

The proximity operators vary in the details of their selections: `bound_to` selects atoms within and outside of `s1`, while `neighbor` does not select atoms within `s1`. Likewise, `expand` includes atoms inside `s1` in its selection, while `around` selects only atoms that are outside of `s1`. `Around` and all of the other distance-based selectors use the centers of atoms to define distances, *except* `gap`, which measures between van der Waals radii.

Group completion operators

PyMOL provides group completion operators to increase efficiency, especially in sessions with several chains, molecules, and objects. These operators allow you to create new selections based on selections that already exist (see Table ES.9).

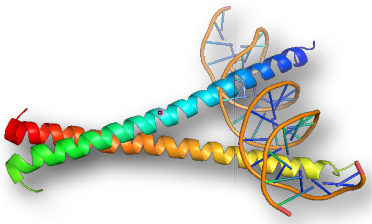
The `byfragment` operator refers to `fragments` of molecules created by the user using advanced mouse commands or the Build menu (see Chapters tocome and tocome).

Table ES.9 Group completion operators

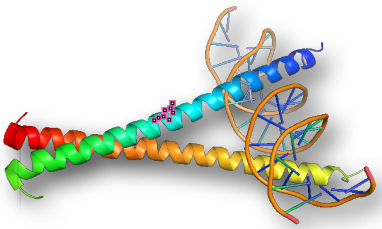
Group completion operator	Short form	Completes the selection of...
`bycalpha s1`	`bca. s1`	all alpha carbons in all residues in `s1` `PyMOL> select Acas, bycalpha chain a`
`bychain s1`	`b. s1`	all atoms in all chains in `s1` `PyMOL> select wholechains, bychain resi 31`
`byfragment s1`	`byfrag s1` `bf.`	all atoms in all fragments in `s1` `PyMOL> select myfrag, bf. sele`
`bymolecule s1`	`bymol s1` `bm. s1`	all atoms in all molecules in `s1` `PyMOL> select allmols, bymol elem S`
`byobject s1`		all atoms in all objects in `s1` `PyMOL> select allobject, byobject chain a`

`byresidue s1`	`byresi s1`
	`byres s1`
	`br. s1`
		all atoms in all residues in `s1`
		`PyMOL> select args, byres name NE`

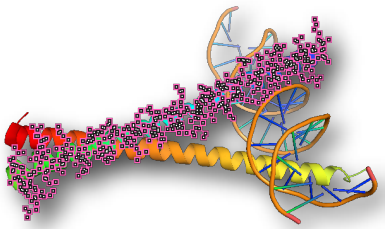
`bysegment s1`	`bysegi s1`
	`byseg s1`
	`bs. s1`
		all atoms in all segments in `s1`
		`PyMOL> select segX, byseg chain a`



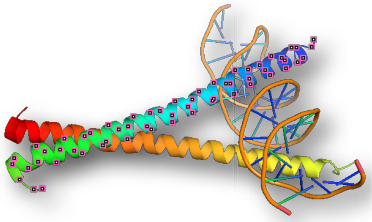
`select one_atom, ///c/255/ca`



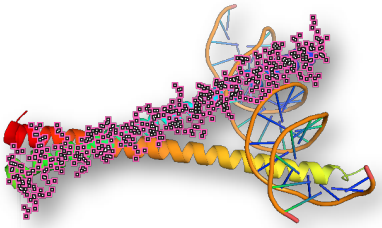
`select one_res, byres one_atom`



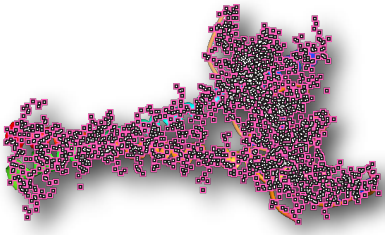
`select chain_c, bychain one_atom`



`select c_alphas_in_c, bycalpha chain_c`



`select c_molecule, bymolecle one_res`



`select the_object, byobject one_atom`

Atom Selection Macros

Atom selection macros enable you to represent long atom selection phrases such as

```
PyMOL> select 1uwh and chain b and resi 142 and name ca
```

in compact form, such as

```
PyMOL> select /1uwh//b/142/ca
```

Slashes delimit five fields in the selection macro, in a hierarchy that runs from left to right. The left-most field, after the first slash, specifies an `entity`, that is, an object or selection. The second field, after the second slash, specifies segments, if required. The third field, more frequently used, specifies chains. The fourth specifies nucleotide or amino acid residues, and the fifth specifies atoms. That is, the slashes define the structure

```
/entity/segment/chain/residue/atom
```

Because the fields are recognized in this order, you only need type identifiers between the slashes, as in

```
/entity-name/segment-identifier/chain-identifier/residue-identifier/atom-identifier
```

or

```
/entity-name/segment-identifier/chain-identifier/residue-name/atom-identifier
```

omitting the selectors `segi`, `chain`, `resi` or `resn`, and `name`. If you type a number in the residue field, as in

```
PyMOL> select /1uwh//b/142/ca
```

PyMOL will recognize it as a residue identifier. If you type only alphabetical characters in the residue field, as in

```
PyMOL> select /1uwh//b/asp/ca
```

PyMOL will recognize it as a residue name code. In this case PyMOL selects the C-alphas in all aspartates in chain b of 1uwh.

The macro selects atoms using the boolean `AND`. That is, the selected atoms must match all of the given identifiers.

PyMOL has to be able to recognize the macro as one word, so no spaces are allowed in it. Also, a macro must contain at least one slash for PyMOL to distinguish it as a macro and not some other type of selection-expression.

Macros come in two flavors: those that begin with a slash and those that don't. If the macro begins with a slash, PyMOL expects to find the fields starting with with the left-most `entity` field. A macro that starts with a slash may have any of the following field formats:

```
/entity/segment/chain/residue/atom
```

```
/entity/segment/chain/residue
```

```
/entity/segment/chain
```

```
/entity/segment
```

```
/entity
```

For example,

```
PyMOL> select /1uwh//b/142
```

selects all atoms in `1uwh and chain b and resi 142`. And

```
PyMOL> select /1uwh//b
```

selects all atoms in `1uwh and chain b`

You may want to select all C-alphas, or all chain A's, regardless of the objects or segments to which they belong. Often, the segment is not even identified in the source file. Sometimes you want to select within the one object you have loaded. In such cases, when you don't need to specify a given field, just leave it blank. For example,

```
PyMOL> select /1uwh///142
```

selects all atoms in `resi 142 and 1uwh`. Other examples of macros starting with slashes are shown in Figure ES.7.

If the macro does not begin with a slash, it is interpreted differently. In this case, PyMOL expects to find the fields ending with the atom field or with the slash after the residue field. Macros that don't start with a slash may take the following field formats:

```
residue/
```

```
residue/atom

chain/residue/atom

segment/chain/residue/atom

entity/segment/chain/residue/atom
```

For example,

```
PyMOL> select 32/
```

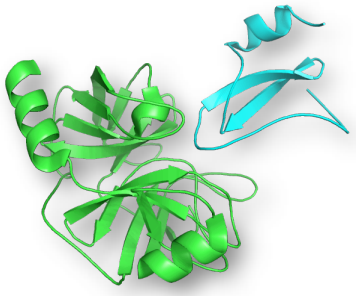
selects all atoms with residue identifier 32, and

```
PyMOL> select 10/cb
```

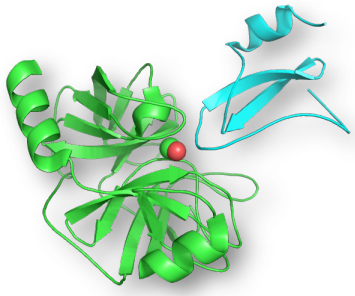
selects the C-beta of residue 10. Also,

```
PyMOL> select a/10-12/c+o
```

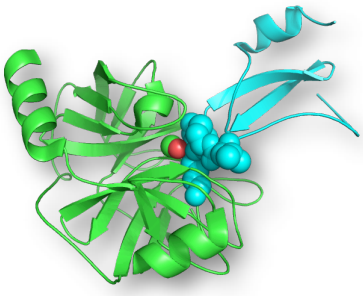
selects carbon and oxygen atoms in residues 10 through 12 of chain a. Other examples are shown in Figure ES.7.



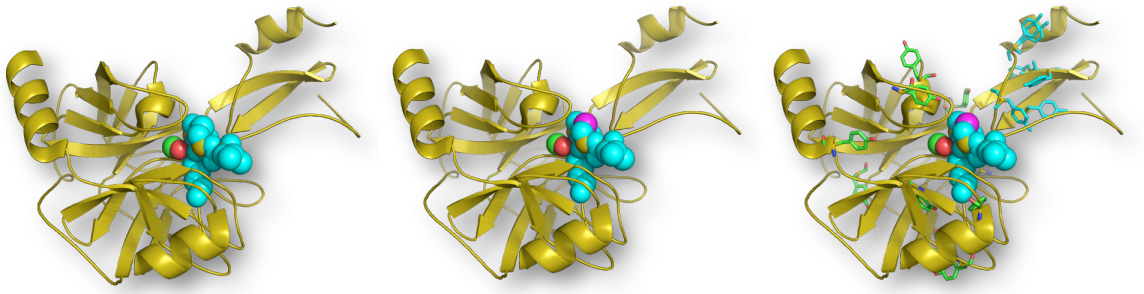
```
color cyan, ///b
or
color cyan, b//
```



```
show spheres, ///a/197/
cb+og
or
show spheres, a/197/cb+og
```



```
show spheres, ///b/37-40
or
show spheres, b/37-40/
```



`color olive, /////ca` or `color olive */ca`	`color magenta, ///b/cys/sg` or `color magenta, b/cys/sg`	`show sticks, ////tyr` or `show sticks, tyr/`

Figure ES.7 Here only one object (2uuy) is loaded (and then shown `as cartoon`). Because there is only one object and there is no segment information in the source file, the `entity` and `segment` fields are omitted in all of the selection macros. Both selection macro flavors are shown for each selection; the first alternatives start with a slash and the second ones don't.

Two optional macro fields are defined by the back-apostrophe (`), so the complete array of fields is

```
/entity/segment/chain/resName`resIdent/atom`altLoc
```

The back-apostrophe is useful in a few specific cases. First, it is useful when you need to specify both a `resName` and a `resIdent`, that is a residue name code and a residue number. For data files in which solvent molecules have some of the same residue numbers as polymer residues, the ambiguity is resolved by including both a `resName` and a `resIdent` in the residue field. For example,

```
PyMOL> select phe`42/
```

selects all atoms with `resi 42 and resn phe`, that is, residue identifier 42 and residue name phe (see Figure ES.8). When you use the back apostrophe in the residue field, PyMol always expects the text to the left of it to be a `resName`, and the the text to the right to be a `resIdent`.

The second use of the back apostrophe is for resolving selections with alternate locations, using the `altLoc` identifier. Just as you can use either a resName or a resIdent in the residue field, you can use either an altLoc identifier or an atom name in the atom field. For example,

```
PyMOL> select 63/`b
```

selects all atoms matching location b in residues numbered 63. Typing

```
PyMOL> select 63/CA
```

selects all C-alphas in residues numbered 63. And

```
PyMOL> select 63/CA`b
```

selects all C-alphas in residues numbered 63 with alternate location identifier b. When you include a back-apostrophe in the atom field, text to the left of it will be considered an atom `name`, and text to the right will be considered an alternate location identifier.

An asterisk (*), or wildcard in programming terms, can be used in any field to match all identifiers. Figure ES.8 demonstrates the use of the asterisk and back apostrophe and shows other examples of selection macros that don't begin with a slash.

Macros are converted into long form before being submitted to the selection engine. Remembering this can help in the interpretation of error messages.

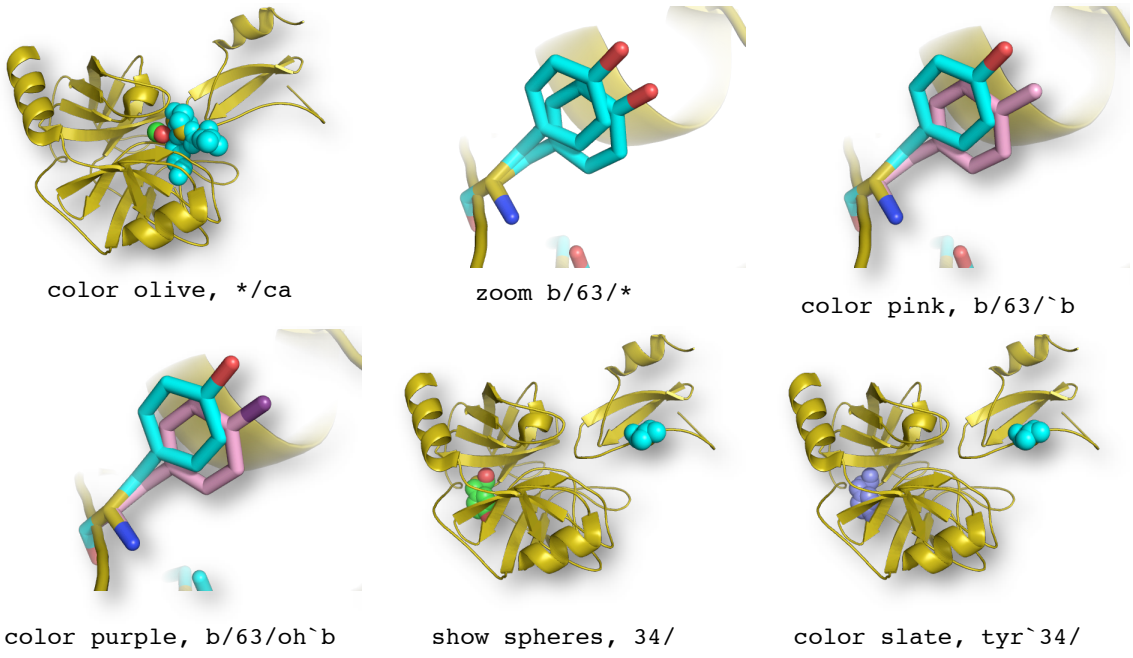


Figure ES.8