

# Introduction to PyMOL

© by Kristian Rother ([www.rubor.de](http://www.rubor.de)) (10/2005)

## License

This document is covered by the GFDL (Gnu Free Documentation License). It may be copied, distributed and modified freely, as long as the license is not altered and the original authors contribution is mentioned in the document.

## Audience

This document is intended for scientists that are new to the PyMOL molecular graphics system but have already used other software to display or manipulate 3D structures of molecules. It is assumed that you have installed PyMOL successfully.

## What is PyMOL?

PyMOL is an open-source tool to visualize molecules available from ([www.pymol.org](http://www.pymol.org)). It runs on Windows, Linux and MacOS equally well. PyMOL has excellent capabilities in creating high-quality images from 3D structures, it has well-developed functions for manipulating structures and some basic functions to analyze their chemical properties. The possibilities to write scripts and plugins as well as to incorporate PyMOL in custom software are vast and superior to most other programs.

PyMOL has been written mostly in the Python language ([www.python.org](http://www.python.org)), while the time-critical parts of the system have been coded in C. This way, Python programs interact most easily with the PyMOL GUI.

## The graphical user interface (GUI)

The PyMOL user environment consists of two windows, whose most important functions are listed below:

1. **3D screen (center of big window)** - the molecules appear here. Alternatively, a text console can be toggled on and off using [ESC].
2. **Object list (top right of big window)** - loaded molecules and user-defined selections appear here. Each of them can be toggled on and off by clicking its name. Also, individual display modes can be selected using the [S]how, [H]ide, [L]abel and [C]olor buttons.
3. **Command line (bottom of big and medium windows)** - Here, text commands can be entered. Uses [TAB] expansion.
4. **Mouse controls (bottom right of big window)** - For modeling purposes, functions of the mouse buttons in combination with the keyboard are summarized here. A click into this control box toggles between two modes for selecting and modeling atoms.
5. **Rendering menu (top of medium window)** - tweaks the quality of the current scene.
6. **Settings menu (top of medium window)** - colors and details of the display modes can be adjusted here.
7. **Wizards & plugins menu (top of medium window)** - contains advanced tools and some demos.

In the following, only the command line syntax will be discussed further.

## First steps

### Loading a molecule:

PyMOL can read the following file formats: ".ent", ".pdb", ".mol", ".xplor", ".mmod", ".ccp4", ".r3d", ".trj" and ".pse". Practically, this is done using one of the commands:

```
load example_1.pdb
load example_2.pdb, my_protein
```

Each new molecule will appear as a separate entry in the object list in the upper right corner. Therefore, they will be termed objects in the following. Note that you can navigate directories by `cd`, `dir`, `ls` and `pwd` just as in Unix.

### Setting display modes:

There is a number of commands for adjusting display and color of objects which are quite self-explanatory. They all work quite similarly, so the list of examples is not exhaustive:

```
show cartoon
hide lines, my_protein
show sticks
show spheres, my_protein
show surface, example_1
hide nonbonded
color teal
color red, my_protein
```

A list of colors is available from the 'Settings' menu or the [C] button in the object list. Custom colors can be defined using the `set_color` command.

### Resetting the view:

If the molecules have slipped out-of-sight or the rotation centre is weird, everything (or a single molecule) can be centered again:

```
orient
orient my_protein
```

### Saving scenes, molecules and images:

Saving the whole PyMOL environment is very straight:

```
save my_session.pse
```

Storing molecules is almost as easy. When no object name is given, PyMOL will store everything:

```
save everything.pdb
save my_molecule.pdb, my_protein
```

Images can be written in ".png" format:

```
png first_picture.png
```

### Getting help:

PyMOL has an excellent built-in documentation. This gives both a list of all available commands and descriptions of all commands with syntax and available options. Additionally, a few specific help topics are available.

```
help
help orient
help show
```

## Creating publication-quality images

### Setting the camera position:

Often, it is desirable to view the molecules from a certain angle. After the right angle has been found by mouse, the orientation can be transferred to a script:

```
get_view
```

Prints a corresponding `set_view` command containing a vector with the exact camera position. It can be copied from the text field in the menu window box to any `".pml"` file directly.

### Background color:

For printed journals, white background is often more appropriate (and cheaper when printing the manuscript). This can be adjusted with:

```
bg_color white
```

### Color spaces:

There are two kinds of colors: RGB (red-green-blue) for screen display, and CMYK (cyan-magenta-yellow-black) for printers. As some RGB colors look terrible when printed, the built-in CMYK translation should be enabled when creating print images. It is found in the 'Display' -> 'Color Space' menu. Please note that the actual look of CMYK colors is to some extent device-dependent. Most of the default colors of PyMOL are CMYK-safe.

### Labels:

PyMOL can display labels for atoms, residues etc., but they are almost worthless for publications. My recommendation is to add them on the final image using MS Powerpoint or any painting software.

### Raytracing:

PyMOL contains an efficient raytracer for high-quality rendering. By default, it calculates an image the size of the 3D window, but the size in pixels can also be given explicitly:

```
ray
ray 1200,1200
```

Note that the latter would correspond to a 4\*4 inch image with 300dpi. In the 'Rendering' menu, options such as picture quality, antialiasing, fog and shadows can be adjusted. The calculated image should be stored using the `png` command immediately, as it is lost as soon as the scenery is changed. Alternatively to the `ray` command, input files for PovRay can be written, but using the built-in raytracer is much easier.

## Using input scripts

The commands that the PyMOL command line understands, can also be stored in script files. Such PyMOL scripts should always have the ending `".pml"`. They are plain ASCII files with one command in each line. A script can be invoked by the command

```
@filename.pml
```

Do not confuse the `".pml"` scripts with PyMOL sessions `".pse"` (store everything the GUI contains). Sessions are very useful, but they tend to fail on older PyMOL versions. Also, scripts can be adopted for other tasks more easily. The `".pml"` scripts should also not be confused with Python scripts `".py"`, that PyMOL can also read. Those are executed with

```
run filename.py
```

## Manipulating molecules

### Selecting parts of molecules:

The most important command in PyMOL is

```
select <selection>,<expression>
```

It creates a named subset of one or several molecules that can be manipulated as if it were a molecule object. The command has an easy and a difficult part: <selection> is the name for the subset, <expression> can be almost anything:

```
# select parts of molecules (chains, residues, amino acids, atoms, elements)
select sel01, chain A
select sel02, resi 1-100
select sel03, resn trp
select sel04, name ca
select sel05, elem fe
```

```
# select secondary structures (helices and sheets):
select sel06, ss h
select sel07, ss s
```

```
# adress specific objects and selections
select sel08, my_protein
select sel09, sel08
```

```
# combine selections
select <name>, <expression> and <expression>
select <name>, <expression> or <expression>
select <name>, not <expression>
select <name>, (<expression>) and <expression>
```

```
# select all oxygens from chain a but no waters.
select sel10, elem O and chain A and not resn HOH
```

```
# access the molecule hierarchy directly
select sel11, my_protein//A/1-100/*
```

```
# select binding pocket of a ligand in 5.0 angstroms diameter
select sel12, resn HEM around 5.0
```

It is recommended that you spend much of your time learning PyMOL around the `select` command. There is an overview of the selection algebra available by `help selections`.

### Selections and objects:

These two concepts need further clarification: An **object** is the primary representation of a molecule with all atoms, bonds, colors and display modes in PyMOL's memory. A **selection** is a pointer to a defined set of atoms in one or several objects. It follows that each atom can belong to more than one selection but only one object. In the PyMOL object list (upper right corner of the window), the names of selections are indicated by brackets. Practically, the biggest difference between both is, that clicking an objects' name will hide/show it completely!

There is a default selection, named "(all)", the function of which should be obvious.

The difference between selections and objects is probably best demonstrated on an example:

```
load lysozyme.ent, lysozyme
select calpha_sel, lysozyme and name ca
create calpha_obj, lysozyme and name ca
```

The first command creates an object named "lysozyme", as it loads new atoms to the memory. The second defines a selection "calpha\_sel" which contains **pointers** to all alpha-carbons of "lysozyme". The third command creates an object named "calpha\_obj" which contains the **coordinates** of the alpha-carbons. Effectively, these atoms are

duplicated in memory by the create command. When e.g. the color of both "lysozyme" and "alpha\_sel" are altered, the changes will override each other. "alpha\_obj" remains unaffected, as it has its own atoms. When "lysozyme" is removed, the selection will also disappear, but the duplicate object remains.

### **Settings:**

In the Settings->Edit All menu, a long list of parameters specifies how PyMOL draws molecules. There are some of them that can be tweaked easily to improve pictures. Two examples can be given below.

### **Ball-and-stick-mode:**

PyMOL has no native ball-and-stick representation. However, it can be simulated by combining spheres and sticks:

```
hide all
show spheres
show sticks
set sphere_scale, 0.3
set stick_radius, 0.2
```

Alternatively, the settings can be altered in the dialog. By default, the sticks and balls have the same color. If different colors are desired for them, this can be achieved by creating separate objects for both.

### **Transparent surfaces:**

Often, it looks very nice to have a part of the molecule invisible. PyMOL can make surfaces partially transparent:

```
hide all
show surface
show cartoon
set transparency, 0.5
```

Again, you may want to shade the internal part differently than the surface, or have separate surfaces for a protein and a ligand. This issue can be solved by creating separate objects for all parts.

### **Modeling:**

The basic modeling functions of PyMol consist of the following:

- Mouse commands for picking, moving and rotating atoms or bonds. Click on the bottom right field with the mouse button commands to activate the modeling functions, and try combinations of keys and mouse buttons (PkAt - pick atom, mvFrag - move fragment, rotFrag - rotate fragment etc.). Note that when moving parts it depends on where you grab a molecule what is actually moved. An Undo function is available via Ctrl-Z.
- You can add whole amino acid residues or other chemical groups via the Edit->Fragment menu. Newer PyMOL versions have a comfortable click-box, where entire amino acid chains can be designed that way. (The 'Builder' button in the menu window)
- Existing amino acids can be altered using the Wizards->Mutagenesis tool. A number of side chain rotamers is available.

## **rTools**

The rtools plugin (<http://www.rubor.de/bioinf>) extends PyMOL's capabilities. It needs to be installed separately. The three 'most wanted' features are described in a short below.

### **PDB access:**

Rtools provides a command to retrieve protein structures directly from the PDB server:

```
pdb lcse
```

### **Movies:**

PyMOL's built-in functions for creating animations are a hell to use. They have been wrapped by more comfortable commands that let an user define very precisely what is to happen. Any animation in PyMOL consists of a series of pictures ("frames"). The movie commands let things happen in a certain interval of frames. Look at the example:

```
# first, any previous data is wiped out
mv_clear

# in the first 100 frames, the object "prot1" shall turn around by a total of
# 270 degrees.
mv_rot 1-100, x, 270, prot1

# in the next 50 frames, nothing shall happen
# after that, another object shall move by 20 angstroms.
mv_trans 150-250, x, -20, prot2

# after that, everything shall turn smoothly
mv_sinrot, 300-400, y, 90

# it is also possible to insert movement overlapping with previous ones
mv_set 130-170,transparency,1.0,0.0

# a single command will stop the animation after it has run once
mv_cmd 450,mstop

# finally, the movie is calculated and starts automatically
movie
```

Please refer to the rtools manual for further information about how to install and use rtools.

### **Script Box:**

A floating box with buttons that each run a *.pml* script is created. The good thing is that the buttons are easy to customize: By default, all buttons are listed in the file `PYMOL_HOME/modules/pmg_tk/startup/rtools/scripts.lst`. It might be a good idea to make this file writable for all, so users can enter their favorite scripts there. Alternatively, you can add additional buttons at run-time:

```
add_button <label>,<command>
```

*(When you try this during startup via the `.pymolrc` file, the `.pymolrc` file could be executed faster than rtools awakens. In that case, you will need to issue some kind of "5 sec delay" in `.pymolrc`).*

## **Topics not covered by this Introduction: Python scripting**

PyMOL's true strength is that its engine can be tweaked through the glove box while driving. The Python internals are coded with many access points for users wanting to interact/manipulate/extend the environment. The best way is to write plugins as Python scripts (although there is at least one Java application that is known to interact with PyMOL well: <http://bionf.charite.de/strap>). The possibilities of Python start with using PyMOL commands from a sophisticated program, continue with extending the PyMOL menus and commands, starting PyMOL from other programs, and i don't know where it ends.

One frequent use of Python scripts is to make new commands available. A short example shall be given below:

```
#
# USAGE
# rb_cartoon <selection>
#
# Displays the specified selection as cartoons and colors them red.
#
from pymol import cmd

def rb_cartoon(selection):
    cmd.hide("everything", selection)
    cmd.show("cartoons", selection)
    cmd.do("color red,%s"%(selection))
    # cmd.color("red",selection) would do the same thing.

# tell PyMOL that there is a new shell command available.
cmd.extend(rb_cartoon,"rb_cartoon")
```

There is an optional .pymolrc file that can be used to issue script commands that shall be executed automatically when PyMOL starts. PyMOL has a default directory where plugins can be placed (PYMOL\_HOME/modules/pmg\_tk/startup). Otherwise, Python scripts need to be started using the run <script.py> command.

## Further reading

There is a thorough load of documentation on PyMOL available on the web:

- <http://www.pymol.org> - The PyMOL manual, which is pretty extensive (150+ pages). It consists of two main parts: First, a detailed explanation of the main modes to work with PyMOL is given (mouse navigation, menus, scripting, rendering etc.), and second, all the script commands are listed in the command reference. The latter part should be identical to the reference texts obtained by the `help <command>` in the PyMOL command line.
- <http://www.pymolwiki.org> - The PyMOL wiki serves as a main gathering point for user-contributed documentation. The Wiki covers mostly intermediate to advanced topics in a high quality and has the advantage that quite often, example scripts are given. Unfortunately there are many areas still uncovered by the Wiki. The main index is reached via the '**Top level of contents**' point on the main page.
- Another tutorial that explains many things not explained here and vice versa has been written by Gareth Stockwell (<http://www.ebi.ac.uk/~gareth/pymol/>)
- PyMOL has a very active **user community** on a mailing list on <http://pymol.sourceforge.net>. The traffic is moderate (2-3 messages per day), it has a friendly climate and new PyMOL users can find advice on both straight and complicated questions there quickly.
- Some experienced PyMOL users have written plugins for PyMOL that are available on their personal websites (if not already on the Wiki). First to mention are Robert Campbell's crystallography tools (<http://adelie.biochem.queensu.ca/~rlc/work/pymol/>), the scripts provided by Michael George Lerner (<http://www-personal.umich.edu/~mlerner/PyMOL/>) and, of course ☺, the rtools written by Kristian Rother (<http://www.rubor.de/bioinf>). Links can be found on the PyMOL homepage. This list may continue.