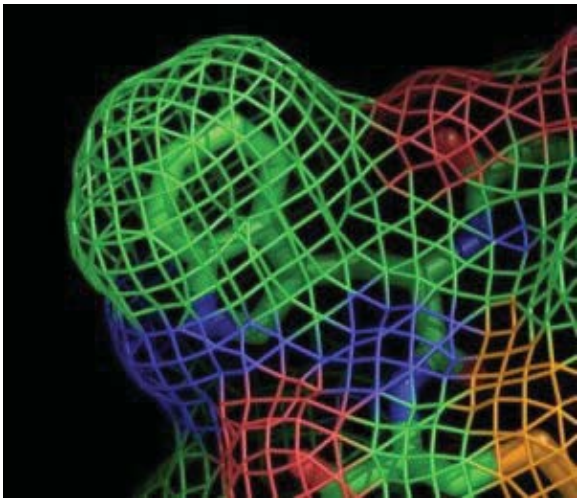
PyMOL User's Guide



written by
Warren L. DeLano, Ph.D.
with assistance from
Sarina Bromberg, Ph.D.

Table of Contents

Table of Contents

Table of Contents

Copyright Notice and Usage Terms

Copyright Notice

Terms of Usage for the PyMOL User's Manual

This manual is NOT free. It is a *PyMOL Incentive Product* created to help you use the program while also generating recurring sponsorship for the project. This manual is made available for evaluation via the "honor" system: **You may evaluate this manual for a continuous period of up to one year without obligation.** If you wish to continue using this document beyond the end of the evaluation period, then you must become a sponsor of the project by purchasing a PyMOL license and a subscription to maintenance and support from DeLano Scientific LLC (http://www.delanoscientific.com).

Of course, if you are willing to sponsor the project today, then please don't wait a full year to start. The sooner your sponsorship comes in, the sooner we can apply it to improve the software and documentation!

Existing PyMOL subscribers may use this manual for no additional cost. However, subscribers who do not renew their subscription upon expiration must discountinue use of this and all other PyMOL Incentive Products. Though we have no direct means of enforcing this, we ask, in recognition of our declared scientific mission, that you honor the trust placed in you.

PyMOL users who are unable to sponsor the project by purchasing a PyMOL license and maintenance subscription are welcome to use Open−Source versions of PyMOL and any free documentation that can be found on the internet.

Trademarks

This chapter last updated June 2004 by Warren L. DeLano, Ph.D.

Preface

Why PyMOL?

PyMOL is one lone scientist's answer to the frustration he encountered with existing visualization and modeling software as a practicing computational scientist.

Anyone who has studied the remarkable complexity of a macromolecular structure will likely agree that visualization is essential to understanding structural biology. Nevertheless, most researchers who use visualization packages ultimately run up against limitations inherent in them which make it difficult or impossible to get exactly what you need. Such limitations in a closed−source commercial software package cannot be easily surmounted, and the same is still true for free programs which aren't available in source form.

Only open−source software allows you to surmount problems by directly changing and enhancing the way software operates, and it places virtually no restrictions on your power and opportunity to innovate. For these reasons, we believe that **open−source software is an intrinsically superior *research* product** and will provide greatest benefit to computer−assisted scientific research over the long term.

Launched over Christmas break in December 1999, PyMOL was originally designed to: (1) visualize multiple conformations of a single structure [trajectories or docked ligand ensembles] (2) interface with external programs, (3) provide professional strength graphics under both Windows and Unix, (4) prepare publication quality images, and (5) fit into a tight budget. All of these goals have since been realized. Although PyMOL is far from perfect and lacks such desirable features such as a general "undo" capacity, it now has many useful capabilities for the practicing research scientist. We hope that you will find PyMOL to be a valuable tool for your work, and we encourage you to let us know what ideas you have for making it even better.

Words of Caution

About the Manual: This version of the manual has been updated for PyMOL version 0.86 (January 2003) but is still quite rough. Prepare yourself for omissions, errors, and potentially obsolete information. Make an informed decision to use the PyMOL manual at your own risk. Understand that thiss same caution applies to the program as a whole −− you shouldn't be using PyMOL if you aren't willing to troubleshoot problems and take the initiative on the mailing list in order to discover solutions.

About the program: PyMOL was created in an efficient but highly pragmatic manner, with heavy emphasis on delivering powerful features to end users. Expediency has almost always taken precedence over elegance, and adherence to established software development practices is inconsistent. PyMOL is about getting the job done now, as fast as possible, by whatever means were available. PyMOL succeeds in meeting important needs today, but we view it as merely an initial step in a promising direction.

In time, we hope that we and others will follow by creating PyMOL−like software platforms which meet the needs of users but also provides the design rigor and code quality necessary to enable broad participation of outside developers. Though PyMOL will undoubtably continue to expand and improve over the next decade, we expectd that its long term impact will primarily be to inspire other development efforts having more time and resources, and which will undoubtable achieve greater heights.

That isn't to say that you can't find good things about PyMOL's internal design. Indeed, we believe that there are many successful and instructive aspects to the program. However, we just hope to appropriately calibrate your expectations with respect to the code you will find if you with to "dive under to hood". Though the

program is Open−Source, it is best thought of as a dense, semi−opaque tool, best extended through Python rather than as a C coding environment in which to embed new technologies.

Strengths

- **Cross−Platform.** A single code base supports both Unix, Macintosh, and Windows, using OpenGL and Python and a small set of Open−source external dependencies.
- **Command−Line and GUI Control** Real world applications require both.
- **Atom Selections.** Arbitrary logical expressions facilitate focused visualization and editing.
- **Molecular Splits/Joins.** Structures can be sliced, diced, and reassembled on the fly and written out to standard files (i.e. PDB).
- **Movies.** Creating movies is as simple as loading multiple PDB files and hitting play.
- **Surfaces.** As good if not better than Grasp, and mesh surfaces are supported too.
- **Cartoon Ribbons.** PyMOL's cartoons are almost as nice as Molscript but are much easier to create and render.
- **Scripting.** The best way to control PyMOL is through reusable scripts, which can be written in the command language or in Python.
- **Rendering.** A built−in ray tracer gives you shadows and depth on any scene. You also render externally.
- **Output.** PNG files output from PyMOL can be directly imported into PowerPoint.
- **Conformational Editing.** Click and drag interface allows you to edit conformations naturally. Sculpting allows the molecule to adapt to your changes.
- **Expandability.** The PyMOL Python API provides a solid way to extend and interface.

Weaknesses

- **User Interface.** Development has been focused on capabilities, not on easy−of−use for new users.
- **Documentation.** Only recently has any documentation become available.
- **Object−Orientation.** There is a single monolithic, functional API.
- **Electrostatics.** PyMOL is not yet a replacement for Delphi/Grasp.
- **No Mechanics Engine** Although PyMOL sports potent molecular editing features, you can't yet perform any "clean−up".

This chapter last updated January 2004 by Warren L. DeLano, Ph.D.

Introduction

Welcome to PyMOL!

Over the years, PyMOL has become a capable molecular viewer with support for animations, high−quality rendering, crystallography, and other common molecular graphics activities. It has been be adopted by many hundreds (perhaps even thousands) of scientists spread over thirty countries. However, PyMOL is still very much a work in progress, with development expected to continue for years to come.

Is PyMOL Free Software?

Yes, but...

PyMOL is Copyrighted software that is Free for all parties to use, modify, and redistribute. Because of PyMOL's unusual status, you can be confident that the time you invest today in learning the package will provide you with long term utility no matter where your career happens to takes you. You will never be required to pay software license fees in order to use Open−Source PyMOL or to share it with others who might find it useful.

Nevertheless, PyMOL is not free to develop, document, maintain, and support. If you decide to adopt the package, then you are asked and expected to contribute to the project in some manner. Although such contributions may take a variety of forms, most PyMOL supporters choose to sponsor the project by purchasing (usually through their school or employer) a license and a renewable subscription to maintenance and support. All such contributions are entirely volutary since we have intentionally abandoned the usual means of compelling compliance. Instead, we depend on your *free will* to provide vital funding for development and to cover other necessary expenses. In return, we provide specific incentives (called *Incentive Products*) as a reward for helping to fund the project. Example incentive products include this manual, extra features, enhanced platform−specific binary versions, and various other conveniences

Please take this request seriously. If you value PyMOL, then it is clearly in your interest to sponsor it. To find out how to donate or to purchase a license, visit the PyMOL web site at **http://www.pymol.org**

The DeLano Scientific Mission

DeLano Scientific LLC is a private vision−centered software company which owns, develops, and supports the PyMOL package. Our mission as a commercial entity is to create highly effective tools for scientific research and to distribute them as broadly as possible while still succeeding as a healthy business. As a "boot−strapped" company, DeLano Scientific is not beholden to any outside investors who would insist upon maximum returns on investment. Thus, we have the rare privilege of being able to place Scientific and Medical Progress ahead of Profit in our hierarchy of values.

We have chosen a free and open−source approach for PyMOL because we believe this strategy will have the greatest positive impact on humanity. Visualization is key part of understanding the nature of life at the molecular level, and powerful visualization tools need to be universally available to all students and scientists if we are to make rapid progress in biomedical research. If PyMOL is successful, then we hope to expand the scope of our endeavors to meet other critical research needs in related areas.

Growth of the DeLano Scientific will depend entirely on the willingness of PyMOL users to adopt, nuture, and advocate for our volitional approach to software funding. Eventually, we hope to evolve into a major provider of scientific software for biomedical research and be distinguished by the quality, openness, and accessibility of our products, the trusting and nonexploitive relationships we form with our customers, and our willingness to work with all parties in advancing scientific software technologies.

This chapter last updated June 2004 by Warren L. DeLano, Ph.D.

Installation

Windows

Recommendations

- Windows 2000 or XP.
- A late−model 3D OpenGL compatible graphics accelerator card from nVidia, ATI, 3Dlabs or similar.
- 512 MB RAM (768 MB or 1 GB preferred).
- 3 Ghz Pentium 4 processor or similar.

Minimal System Requirements

- Windows 98 and ME, or later. PyMOL will not run on Windows 95 and NT.
- 3D OpenGL compatible graphics accelerator card.
- 256 MB RAM.
- 500 Mhz Pentium 3 processor.

Unless you have prior experience with Python, we recommend installing a version of PyMOL which does not require an external Python interpreter. Avoid versions of which contain "−py21", "−py22", "−py23" or similar in the filename.

Python−Free Installation

1. Download the ".zip" format archive. For example,

   ```
   pymol-0_90-bin-win32.zip
   ```
2. Extract the .zip file using WinZip (Windows XP can open .zip files directly).
3. Double click on the "Setup" or "Setup.exe" icon in the folder.
4. Answer the questions which follow.

You can now launch PyMOL from the Start menu.

Python−Dependent Installation

If you already have Python installed and wish to use PyMOL with that interpreter, the process is virtually identical. The only difference is that you need to download a version of PyMOL which matches your desired Python version in the filename. For example:

`pymol-0_90-bin-win32-py22.zip`	would work with
`Python-2.2.2.exe`	available from http://www.python.org.

MacOS X

Recommendations

- Mac OS 10.2.x or 10.3.x.
- Dual 2.0+ Ghz G5 system.
- GeForce4 or Radeon 9x00 OpenGL accelerator.
- 1 GB of RAM.

Minimial Requirements

- Mac OS X 10.2.x
- Single 833 Mhz G4 system (will run on less, but performance is poor).
- 3D OpenGL graphics acceleration.
- 512 MB of RAM (1 GB recommended).

If you use Fink

PyMOL is part of the Fink ports collection.

```
sudo -s
apt-get pymol install
```

should be sufficient to get a functioning instance on your system. However, it may not be the most recent version. We also highly recommend installation of Apple's X Server, which enables PyMOL to access your accelerated graphics hardware.

If you do not use Fink

Option 1: MacPyMOL

At the request of various Macintosh users, as well as Apple itself, we have created **MacPyMOL**, a special native Aqua version of PyMOL for the Macintosh. The latest version of MacPyMOL can be downloaded from http://delsci.com/macpymol.

However, note that this version is an Incentive Product only available to PyMOL sponsors (but students and teachers are exempt from this requirement). For more information on MacPyMOL, contact support@delanoscientific.com.

Option 2: PyMOL X11 Hybrid

This version of PyMOL includes a native Aqua−based OpenGL window and an X11−based Tcl/Tk external GUI (graphical user interface). Before launching the PyMOL X11 Hybrid, you must have Apple's X11 server installed and launched. The advantage of using this version is that it is fully compatible with Open−Source PyMOL, and does not require Fink. However, unlike MacPyMOL, this version does not support direct export of QuickTime movies. This binary build is free, but not Open−Source.

1. Download the "pymol−0_XX−bin−osx−x11−hybrid.dmg.gz" compressed disk image.
2. Extract the archive and mount the disk image.
3. Copy the "PyMOLX11Hybrid" folder to the main Applications folder on your hard disk.

You can then launch PyMOLX11Hybrid by double−clicking on the PyMOLX11Hybrid icon.

Linux and Unix

System Requirements

- 3D OpenGL graphics acceleration.

There are a several different ways to install PyMOL on Linux. Please consult the PyMOL Web Site for additional information.

Dependency−Free Approaches

These do not require installing any other packages in a privileged location on your system. All you need to do is download a "tar"−ball appropriate for your system, such as the following:

- pymol−0_93−bin−linux−libc6−i386.tgz (for Linux)
- pymol−0_93−bin−irix65−r10k.tgz (for SGI)
- pymol−0_93−bin−solaris8−sun4u.tgz (for Solaris)

issue the following commands

```
gunzip < pymol-...-bin-...-.tgz | tar -xvf -
./setup.sh
```

which will install the program, and then

```
./pymol.com
```

will launch PyMOL. You may then want to make a symbol link for this file to **~/bin/pymol** for easy launching.

```
ln -s $PWD/pymol.com ~/bin/pymol
```

Install a minimal dependency binary build.

Compile PyMOL from source along with the "ext" dependencies distribution.

Because the installation process is often subject to change, please see the INSTALL file from the current distribution for detailed instructions. In summary,

1. Download, extract, configure, and compile the external dependencies.
2. Download and extract the current PyMOL source distribution.
3. Create a symbolic link from the external dependencies to "ext" in the PyMOL directory.
4. Configure compilation by copying and modifying a "Rules.make" from the setup directory to reflect your system.
5. Run "make" to build pymol.
6. Create a pymol.com specific to your installation location.

You should be able to launch PyMOL by running pymol.com. I usually symbolic link this file into my "bin" directory as "pymol".

Dependency−Based Approaches

You must install the following packages on your system

```
python (2.x), tcl (8.x), tk (8.x), libpng (1.x), zlib (1.x),
glut (3.x), glut-devel (3.x), pmw*, and numeric* (numpy)
```

(* = not required for RPM packages.)

You then have several choices:

Using RedHat binary packages (RPMs).

```
rpm -i pymol-0.90-1.rh73.py22.i386.rpm
```

Using Python's distutils to compile and install PyMOL as a standard Python module.

```
python setup.py build      (as a user)
python setup.py install    (as root)
python setup2.py install   (as root)
```

You can now run PyMOL with "./pymol.com".

Using Makefiles with preinstalled system dependencies.

Because the installation process is often subject to change, please see the INSTALL file from the current distribution for detailed instructions. In summary,

1. Download and extract the current PyMOL source distribution.
2. Configure PyMOL by copying and modifying a "Rules.make" from the "setup" directory to reflect your system.
3. Run "make" to build pymol.
4. Create a pymol.com specific to your installation location.

You should be able to launch PyMOL by running pymol.com, and it may be convenient to add a symbolic link from this file into your "bin" directory as "pymol".

This chapter last updated June 2004 by Warren L. DeLano, Ph.D.

Getting Started with Mouse Controls

Launching

Using the Mouse

On Windows:

Click on the **Start** menu, follow it to **Programs** (or **All Programs** on Windows XP), and then release the mouse on **PyMOL**.

On Mac OSX (native version)

Double−click on the PyMOL icon in the **Applications** folder on your main hard drive

Using a Command Line

Various command line options can be included under both Windows and Unix to automatically open files and launch scripts. See "launching" in the reference manual for more information on these options.

On Windows:

At the command prompt, issue:

```
c:\program files\delano scientific\pymol\pymolwin.exe
```

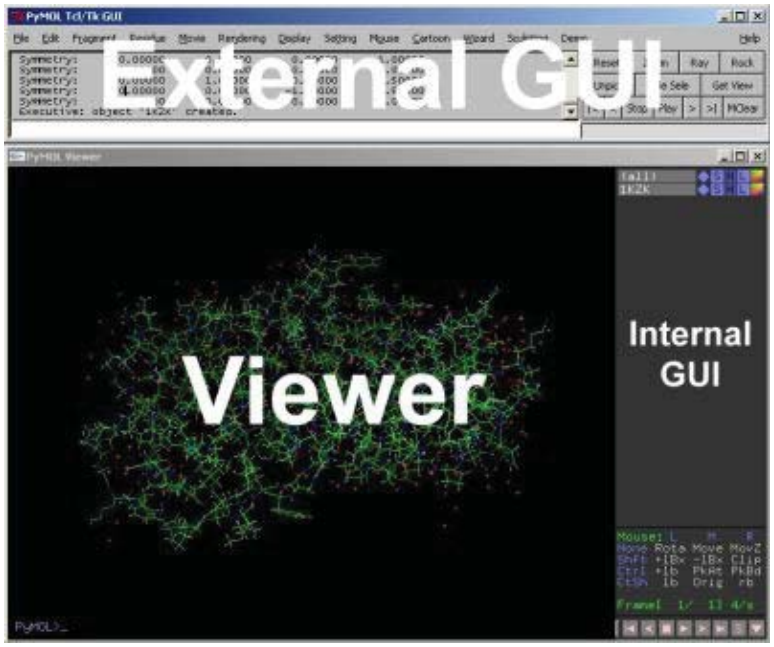
If PyMOL in installed somewhere nonstandard, then use the correct drive letter and path.

On Unix, Linux, and MacOS X (Fink version)

If you installed using using a package such as an RPM, then there is a good chance that "**pymol**" is already in your path. If not, then edit **pymol.com** in the PyMOL distribution and make sure **PYMOL_PATH** points to the actual location of the distribution. Enter **./pymol.com** to start pymol. You will probably want to create a link "**pymol**" from this file in to a "bin" directory in your path so that you can launch the program anywhere by simply entering "**pymol**".

PyMOL's Windows

PyMOL normally starts with two windows: The Viewer Window and the External (Tcl/Tk) GUI Window.

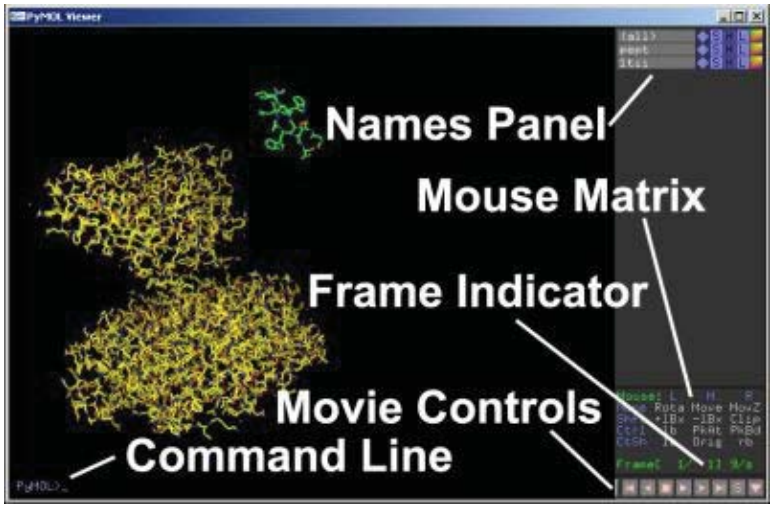


PyMOL's two windows.

GUI is an abbreviation for Graphical User Interface, which usually consists of menus, buttons, text boxes, and other familiar gadgets. By default, PyMOL actually has two GUI's: (1) an "Internal" GUI which appears inside the Viewer Window, and (2) an "External" GUI which appears inside of its own window. The reasons for this are boring and technical, but know that both GUI's will eventually be unified into a single interface in the future.

The Viewer Window

The PyMOL Viewer represents the heart of the PyMOL system. This is a single OpenGL window where all 3D graphics are displayed and where all direct user interaction with 3D models takes place.



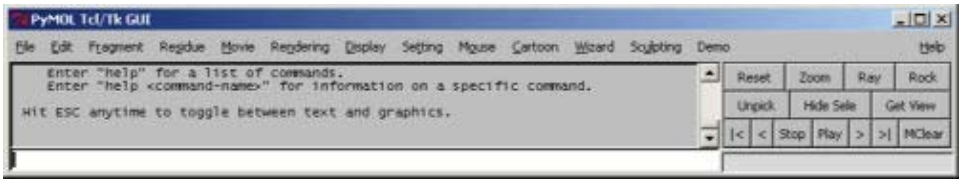
PyMOL Viewer window with Internal GUI enabled (Default).

The Internal GUI contained within this window (right) allows you to perform actions on specific objects and specific atom selections. From top to bottom, it contains an object list, a mouse button configuration matrix, a frame indicator, and a set of "VCR"−like controls for working with movies.

The Viewer also contains a command line (bottom) which can be used to enter PyMOL commands. It is also possible to view PyMOL text output in the Viewer window. you can **hit the ESC key anytime to toggle between text and graphics mode inside the Viewer window.**

The PyMOL Viewer can be run all by itself, and it provides the complete capability of the PyMOL core system. If desired, the Command line and Internal GUI can be disabled. Many tasks can be made easier and more efficient through use of standard menus and controls. For the most part, such gadgets are currently found in an External GUI window.

The External GUI Window



The default Tcl/Tk External GUI included with PyMOL.

By default, PyMOL comes with a single external GUI window which provides a standard menu bar, an output region, a command input field, and a series of buttons. One important advantage of the external GUI window is that **standard "cut and paste" functions for text will only work within the External GUI**, and not within in the PyMOL Viewer. Furthermore, **you must use Ctrl−X, Ctrl−C, and Ctrl−V to cut, copy, and paste** because a standard Edit menu has not yet been implemented.

Notes For Developers: External GUIs are the foundation for modularity and customizability in the PyMOL system. These windows constitute independent processes (or threads) which can control the behavior of PyMOL, and potentially interact with other programs. They are completely customizable at the Python scripting level, and mutiple external GUIs can exist at once (within the restrictions of Tkinter and wxPython).

External GUIs communicate with PyMOL through the Python API (Application Programming Interface). Those of you who want to link up you own programs with PyMOL should generally use a separate external GUI window to control the interaction, rather than changing internal PyMOL code. That way the programs will continue to work together even while development on each program proceeds independently. The internal GUI and all external GUI windows can be enabled and disabled using simple command line options (see reference for "launching").

Loading PDB Files

Using the External GUI Menu

The default external GUI provides a standard **Open...** item in the **File** menu which you can use to select the file you wish to open.

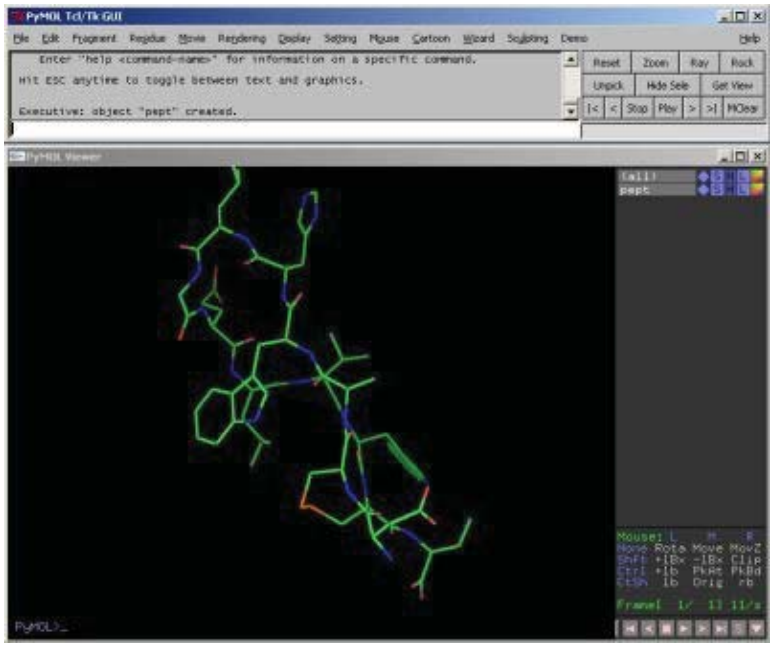
Using Commands

```
SYNTAX

   load <filename>

EXAMPLE

   load test/dat/pept.pdb
```



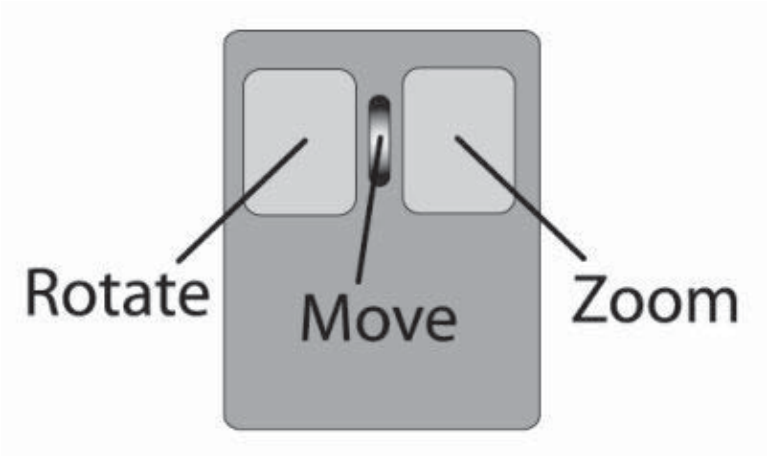
PyMOL after loading a PDB file.

Manipulating the View

In PyMOL, the mouse is the primary control device, and keyboard modifier keys (SHIFT, CTRL, SHIFT+CTRL) are used in order to modulate button behavior. **A three button mouse is required for effective use of PyMOL**, but common mice such as the Microsoft Intellimouse and Microsoft Wheel Mouse will work just fine under Windows.

Basic Mouse Control

On mice with a scroll wheel, you can push down on the wheel in order to use it as a middle button.



Here is a table of the basic mouse button/keyboard combinations for view manipulation:

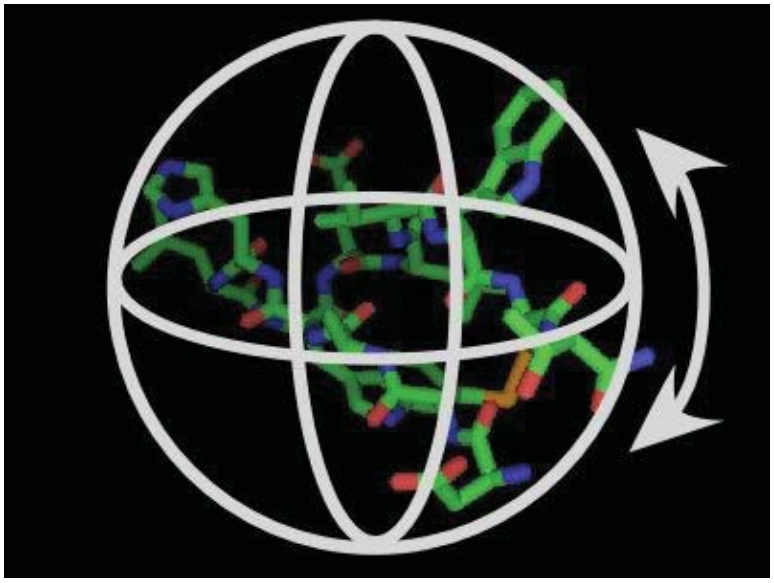
Keyboard Modifier	Left Button	Middle Button	Right Button
(none)	Rotate Camera (Virtual Trackball)	Move Camera in XY (In Plane of Screen)	Move Camera in Z (Scale)
Shift Key			Move Clipping Planes
Control Keys			
Control and Shift Keys		Set Origin of Rotation	

An abbreviated version of this table, the Mouse Matrix, is always displayed in the Internal GUI, in order to help you remember which key and mouse button performs which action:

	L	M	R
None	Rota	Move	MovZ
Shft			Clip
Ctrl			
CtSh		Orig	

When using PyMOL on a laptop, it may be necessary to attach an external mouse or reassign the particular mouse controls you plan to use onto the reduced set of buttons that you have available internally (see reference on the "button" command).

Virtual Trackball Rotation

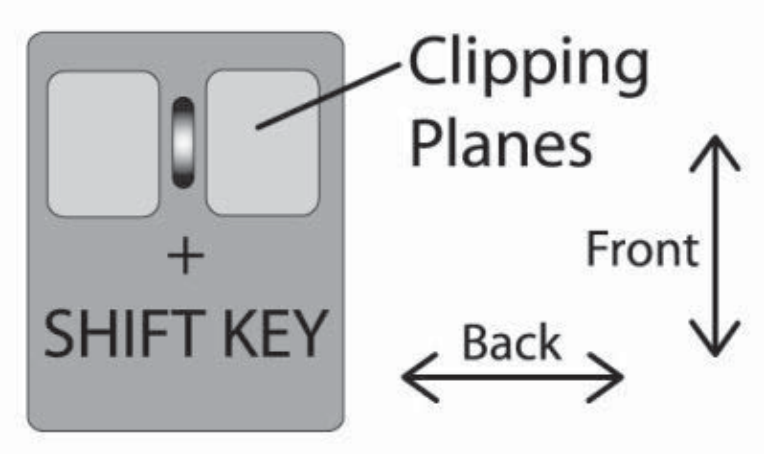


PyMOL's Virtual Trackball.

Virtual trackball rotation works as if there is an invisible ball in the center of the scene. When you click and drag on the screen, it is as if you put your finger on the sphere and rotated it in approximately the same manner. If you click outside the sphere, then you get rotation about the Z−axis only. Generally, the view will be easiest to control by either clicking in the center of the scene and moving outwards (mostly XY−rotation), or by clicking and draging around the edge of the screen and moving in a circular fashion (Z−rotation).

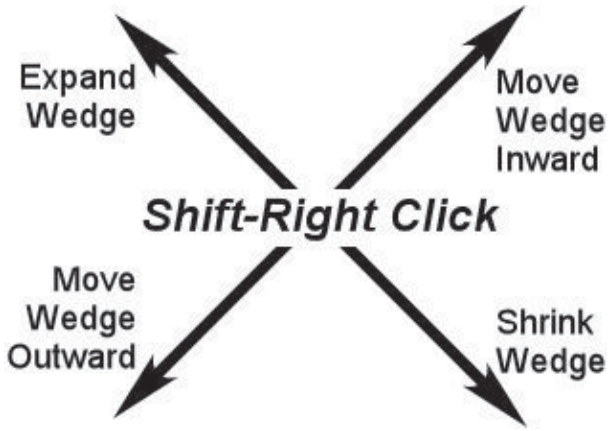
Moving Clipping Planes

PyMOL's clipping plane control is somewhat unusual and may take a few minutes to get used to. Instead of having separate controls for the front and back clipping planes, controls are combined into a single mode where **up−down mouse motion moves the front** (near) clipping plane and **left−right mouse motion controls the back** (far) clipping plane.



Control of clipping planes.

The advantage of the PyMOL clipping plane control is that tedious tandem manipulations of the clipping planes now becomes easy through the diagonal motions shown below.

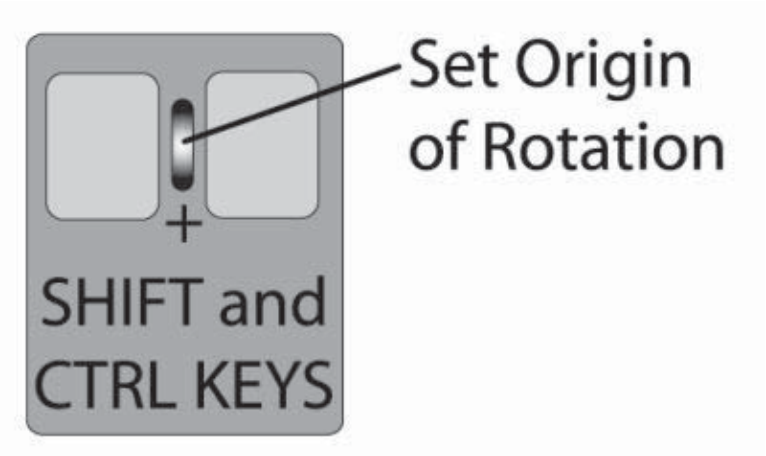


Changing the visible "wedge" by moving clipping planes in tandem.

You can also use the "clip" command to control the clipping planes.

Changing the Origin of Rotation

When visualizing molecules, it is frequently necessary to change the origin of rotation so that you can inspect a particular region of the molecule. The fastest way to do this in PyMOL is to Control−Shift−Middle−Click on a visible atom in the scene.



Getting Comfortable

At this point, we recommend that you spend five or ten minutes getting comfortable with the controls described in this chapter. Specifically, you should be able to accomplish the following tasks:

1. Load a PDB file into PyMOL.
2. Rotate, translate, and zoom the camera.
3. Adjust the front and back clipping planes to clearly view a slice of the molecule.
4. Change the origin of rotation about any particular atom of interest.

Getting Started with Commands

This section steps through a typical PyMOL session, introducing typed commands and describing how PyMOL responds to them. The details of command syntax are in the section titled "PyMOL Command Language."

The PyMOL language is case−sensitive, but upper case is not used in the current package. So just remember to type all your commands in lower case.

Recording Your Work (Optional)

While you are learning PyMOL or doing complex projects, you may want to keep a record of all the commands you give in a plain text log−file that you can read and edit. To open a log−file, type the command **log_open** followed by a file−name:

```
SYNTAX

   log_open log-file-name

EXAMPLE

   PyMOL> log_open log1.pml
```

All your commands, typed or clicked, will be recorded in the log−file. You should give your *log−file−name* the extension ".pml" so you can load the file as a script, to repeat your commands in a new session (see the subsection titled "Sessions and Scripts" below).

To stop recording your commands, type **log_close**. If you don't type **log_close** before you exit PyMOL, your log−file will still be saved to disk.

If you just want to save the current state of your PyMOL work without concern for the steps you took and the commands you gave, you can create a session−file (see "Sessions and Scripts").

Loading Data

Next you need to input your data from a file, say atomic coordinates in PDB format:

```
SYNTAX

   load data-file-name

EXAMPLE

   PyMOL> load $PYMOL_PATH/test/dat/pept.pdb
```

Given this command, PyMOL will open and read the file "pept.pdb," create and name a corresponding object, display a representation of the object in the viewer, and add the object's name to the control panel.

By default, PyMOL names the object after the file it read. You can assign a different name to the object by typing the name in the command line:

```
SYNTAX
```

```
   load data-file-name, object-name

EXAMPLES

   PyMOL> load $PYMOL_PATH/test/dat/pept.pdb        # The object is named "pept".
                                                    # PyMOL doesn't use
                                                    # the file-name extension
                                                    # ".pdb" in the object-name.

   PyMOL> load $PYMOL_PATH/test/dat/pept.pdb, test  # The object is named "test".
```

(Note that "#" is a comment character, so anything you type to the right of "#" in a command line is not interpreted by PyMOL.)

The command for loading a file follows typical PyMOL syntax. **load** is a *keyword*; it calls PyMOL to perform a certain function. *data−file−name* and *object−name* are arguments to **load**. These arguments tell PyMOL what file to load and what to name it, but in general, arguments to keywords just supply information that PyMOL needs to carry out commands.

Manipulating Objects

After PyMOL creates an object, you can manipulate it in the view window and control panel with your mouse, and also by typing commands. For example, you can change from the default representation, called **lines**, to the more hefty **sticks**. First get rid of the **lines** and then show the **sticks**:

```
SYNTAX

   hide representation

   show representation

EXAMPLES

   PyMOL> hide lines       # The object shown in lines disappears from view.

   PyMOL> show sticks      # The object is represented as sticks in the viewer.
```

Other representations are **cartoon**s, **ribbon**s, **dots**, **spheres**, **surface**s, and **mesh**es (See the Section titled "Representations").

Atom Selections

If you want to manipulate a subset of the atoms and bonds in a molecule, you can use *atom selections.* PyMOL is pretty sophisticated when it comes to selecting, grouping and naming groups of atoms. For example, you can select particular residues or atoms in a binding pocket, or a hydrophobic patch, or all the alanines in a helix, and so on. The Section titled "PyMOL Command Language" gives the details for selecting interesting groups of atoms.

You can use a selection just once, or you can name it to make it easier to use again later. For example, you can zoom in on a selection "on the fly:"

```
SYNTAX

   zoom selection-expression    # Select the atoms just for zooming.
```

```
EXAMPLE

   PyMOL> zoom resi 1-10          # The selector resi
                                  # chooses amino acid residues
                                  # given by the PDB sequence number
                                  # identifier "1-10."
```

*Selection−expression*s range from single words to long complicated expressions. An *object−name* may be a *selection−expression*. The default *selection−expression* is **all**, which refers to all the atoms that are currently loaded. If a *selection−expression* is missing, PyMOL will apply the command to **all**. We'll keep our *selection−expression*s short in this section.

If you name the selection, you will be able to manipulate it any number of times. Object and selection names may include the upper or lower case characters (A/a to Z/z), numerals (0 to 9), and the underscore character (_). Characters to avoid include:

```
! @ # $ % ^ &* ( ) ' " [ ] { } \ | ~ ` <> . ? /
```

First, name the selection:

```
SYNTAX

   select selection-name, selection-expression

EXAMPLES

   PyMOL> select akeeper, resi 1-10    # Select the residues and name them "akeeper."
```

Then use it:

```
SYNTAX

   zoom selection-name

   hide representation, selection-name

   show representation, selection-name

EXAMPLES

   PyMOL> zoom akeeper                 # Zoom in on them in the viewer.

   PyMOL> hide everything, akeeper     # Hide their representation in the viewer.

   PyMOL> show spheres, akeeper        # Show them in a different representation,
                                       # spheres, this time.
```

When you create a *selection−name,* PyMOL puts it in the control panel so you can apply control panel functions to the selection using your mouse (See the section titled "PyMOL Command Language").

Named−selections such as "akeeper" are manipulated like PyMOL objects, but objects and named−selections are fundamentally different. PyMOL creates an *object−name* to locate data when you load a data file. Making selections is a way of pointing to a subset of that data. To distinguish *selection−name*s from *object−name*s, *selection−name*s are shown inside parentheses in the control panel.

When you delete a *selection−name*, the data are still found under the *object−name*, but the data are no longer organized as the selection. In contrast, after you delete an object, you must reload the data to have access to it again.

```
SYNTAX

   delete selection-name

   delete object-name

EXAMPLES

   PyMOL> delete akeeper              # "akeeper" is gone, but the object remains.

   PyMOL> delete pept                 # The atoms and bonds in "pept" are gone.
```

Coloring Objects and Selections

You can apply various colors to selections and objects using typed commands. Predefined *color−name*s are listed under the settings/colors pull−down menu. Many of them can be chosen from the control panel. See the section titled "Settings" to find out how to define more colors.

```
SYNTAX

   color color-name                    # All the representations of
                                       # loaded objects are colored.

   color color-name, selection-expression       # The representation of
                                                # the selection is colored.

EXAMPLES

   PyMOL> color white                  # Everything turns white.
                                       # This looks fine on the
                                       # default black background,
                                       # but causes disappearance
                                       # if you've changed the background to white.

   PyMOL> color orange, pept           # Remember that "pept" is our object-name,
                                       # so the entire object is colored orange.

   PyMOL> color green, resi 50+54+58   # The representation of
                                       # residues numbered 50, 54 and 58
                                       # is colored green.

   PyMOL> color yellow, resi 60-90     # The representation of
                                       # residues numbered 60 through 90
                                       # is colored yellow.

   PyMOL> color blue, akeeper          # Residues numbered 1-10,
                                       # which were collected in
                                       # the named selection "akeeper,"
                                       # are colored blue.

   PyMOL> color red, ss h              # The representations of
                                       # helical residues
                                       # are colored red.

   PyMOL> color yellow, ss s           # The representations of
```

```
                                    # beta sheet residues
                                    # are colored yellow.

   PyMOL> color green, ss l+""      # The representations of
                                    # loop and unassigned residues
                                    # are colored green.
```

In the last three examples, the *selector* **ss** chooses secondary structures specified by **h** for helix, **s** for beta sheet strand and **l+""** for loops and unspecified structures.

Turning Objects and Selections On and Off

PyMOL can hold several objects in memory at the same time. The commands **disable** and **enable** allow you to eliminate representations of objects from the viewer while still controlling their properties with commands.

```
SYNTAX

   enable object-name

EXAMPLE

   PyMOL> load $PYMOL_PATH/test/dat/fc.pdb
   PyMOL> load $PYMOL_PATH/test/dat/pept.pdb

   PyMOL> disable pept                          # All representations of "pept"
                                                # are removed from view.

   PyMOL> color yellow, name c+o+n+ca           # Backbone atoms in both "fc"
                                                # and "pept" are colored yellow,
                                                # but "pept" atoms
                                                # are still not visible.

   PyMOL> enable pept                           # "pept" atoms are visible and
                                                # its backbone atoms are yellow.
```

You can also use the **disable** command to get rid of the pink dots that identify the last−named selection in the viewer:

```
SYNTAX

   enable selection-name


EXAMPLE

   PyMOL> select bb, name c+o+n+ca     # Atoms included in the
                                       # named-selection are indicated
                                       # with pink dots in the viewer.

   PyMOL> disable bb                   # The pink dots disappear,
                                       # but the named selection "bb"
                                       # is still visible.

   PyMOL> color red, bb                # You can still manipulate "bb."
```

You can still operate on the representations of objects that are disabled, even with the commands **show** and **hide**. The results will be apparent when you subsequently **enable** the object.

Changing Your Point of View

Dragging on a molecule with the mouse is often the easiest way to manouver, but typed commands such as **zoom** and **orient** give you a different form of control, allowing computations to direct the view. **zoom**, as the name suggests, brings an object or selection close up in the center of the field of view. If the object or selection doesn't fit in the current view, the view opens out to include it. If it is just a small part of the current view, the view closes in to fill more of the screen with it.

```
SYNTAX

   zoom selection-expression        # The "camera" moves close
                                    # to the selection so it fills the viewer,
                                    # or moves further away to include
                                    # all of the selection in the viewer
```

orient is a useful command when you want a fresh start in viewing the molecule. It aligns the object or selection so its largest dimension is shown horizontally, and its second largest dimension is shown vertically.

```
SYNTAX

   orient selection-expression      # The selection is aligned
                                    # for maximum visibility in the viewer.
```

You can store orientations and recall them later in your PyMOL session using the command **view.** Storing a view only saves the viewpoint on the objects in the viewer. It does not save their representation. To store a view for later in the session, you need to "key" it, that is, to give a name or number as an argument to the command **view**. A second argument tells PyMOL whether you want it to store the view or recall it.

```
SYNTAX

   view key, action           # The possible actions are store and recall.

EXAMPLES

   PyMOL> view v1, store      # The current view is named "v1" and stored.

   PyMOL> view v1, recall     # The view keyed "v1" is restored.

   PyMOL> view v1             # recall is the default argument to view,
                              # so this also recalls "v1."
```

The keyword **view** only stores an orientation for the duration of the current PyMOL session. The next section gives the recipe for saving and restoring views in different PyMOL sessions.

Saving Your Work

PyMOL saves your work in f kinds of processes: (1) Before you give a series of commands, you can initiate the process of logging your commands into plain text log−files that can later be used as scripts. (2) At any point in a PyMOL session you can save the memory state of the program by creating a session file that can later be loaded to restore that memory state. (3) You can write a graphics file to store the image you have created in the viewer for sharing or publication.

Scripts and Log Files

A PyMOL script is just a text file, such as a log−file, containing typed PyMOL commands separated by carriage returns. When a script is loaded into PyMOL the commands it contains are executed. PyMOL expects scripts to have ".pml" file−name extensions (this is not strictly required, but it is good practice).

You can use log−files as scripts, and you can create scripts in a text editor such as **emacs**, **jot**, or **notepad**. It's often useful to keep a text editor open in a separate window while using PyMOL. Commands can then be cut and pasted between the two programs.

You can open a new log−file by typing **log_open** *log−file−name*, or by clicking on "log" under the "File" menu and naming the log−file in the dialog box. You can also append commands to an existing log−file by choosing "append" or "resume" in the "File" menu. When you "resume" rather than "append," the existing log−file is first loaded as a script, and then subsequent commands are written to it.

Once you have opened a log−file in any of these ways, PyMOL will write and save all your commands, whether they are typed or given by clicking on the buttons in the GUI.

However, to store the orientation of a molecule into a log−file, you need to give the command **get_view** (type it or use the GUI button). You may find it convenient to **get_view** several times in a PyMOL session, and then edit the log−file to select the most useful views.

Scripts can be executed in several ways. Under Windows, scripts can be run in a new PyMOL session by double clicking on the script's icon. Alternatively, you can run a script using the "File" menu's "Run" option. PyMOL also understands "**@**" as the typed command that loads a script:

```
SYNTAX

   @script-file-name

EXAMPLE

   PyMOL> @my_script.pml
```

You can also include the *script−file−name* when launching PyMOL from a command line:

```
SYNTAX

   pymol script-file-name

EXAMPLE (Windows)

   C:\> pymol.exe my_script.pml

EXAMPLE (Unix)

   unix> ./pymol.com my_script.pml
```

png Files

Once you are satisfied with the representation and orientation of your molecule, you may want to save the image in a graphics file. Before you do that, you can improve the quality of the graphics by switching from PyMOL's fast default graphics engine, OpenGL, to its ray tracer. The ray tracer is slower, but produces higher

quality renderings for display and publication. Ray tracing shows how light is reflected and how shadows fall in a three−dimensional world. Ray tracing may take some minutes for a large complex object. The keyword **ray** calls PyMOL's raytracer to redraw and display the image in the view window (See the section titled "Ray Tracing" for more details).

To save an image to a file, raytraced or not, use the "Save Image" option in the "File" menu or type the **png** command:

```
SYNTAX

   png file-name

EXAMPLE

   PyMOL>  png $PYMOL_PATH/pep     # The file-name extension ".jpeg" is
                                   # added. The image file "pep.jpeg" is stored
                                   # in a path below the PyMOL installation.
```

The PNG file format is directly readable by PowerPoint. It can be converted into other formats using a package like ImageMagick.

Session Files

If you want to be able to return to the current state of PyMOL, then you can create a session−file (Choose "Save Session" in the "File" menu and respond to the dialog box by naming the file with a ".pse" file−name extension). This utility works like the "Save" utility in a word processing program. A PyMOL session−file is a symbolic record of the state of PyMOL's memory, including the the objects that have been loaded or created, the named−selections that have been created, and the display in the viewer.

When you open the saved session−file, PyMOL's memory returns to the state that was saved. Because a session−file represents a PyMOL memory state, opening one means that you are eliminating everything that you currently have in PyMOL's memory, and replacing it with the memory state from the session−file.

A session−file differs from a log−file or a script in a number of ways. You have to open a log−file before you give the commands you want to save, but a session−file can be created at any point. A session−file is invoked by choosing "open" under the file menu, while a log−file is "run" as a script. Also, you can't write or edit session−files, as you can log−files and scripts.

It's a good idea to create session−files at strategic points in PyMOL sessions, for example, when you decide to explore one of several options. In this way, session−files can be used to replace an "undo" utility, which PyMOL lacks. You can store any number of PyMOL states in successive session−files, and revert to them, effectively "undo"−ing the work you did since creating the session−file.

Command−Line Shortcuts

Since almost nobody likes to type, PyMOL's command−line interface includes several "shortcut" features designed to reduce typing. If you are a unix user, you will recognize the similarity with features found in tcsh or bash.

Command Completion using TAB

If you type the first few characters of a command and then hit TAB, PyMOL will either complete the command or print out a list of which commands match the command.

```
EXAMPLE

   PyMOL> sel

   # hitting TAB will produce

   PyMOL> select
```

If you hit the TAB key on a blank command line, PyMOL will output a list of its commands.

Filename Completion using TAB

Some of the files you need to load into PyMOL may have long paths and filenames. PyMOL makes it easier to load such files by automatically completing unambiguous paths and filenames when you hit the TAB key. For instance,

```
EXAMPLE

   PyMOL> load cry

   # If "crystal.pdb" exists in the current directory, hitting TAB will generate

   PyMOL> load cystal.pdb
```

If there is some ambiguity in the filename, PyMOL will complete the name up to the point of ambiguity and then print out the matching files in the directory.

Automatic Inferences

There are a small number of "fixed string" arguments to PyMOL commands. For example, in

```
   PyMOL> show sticks
```

"sticks" is a fixed string argument to **show**. Because there is only a small set of such arguments to **show**, PyMOL will infer your meaning even if you only provide it with a few letters. For example

```
   PyMOL> show st
```

works just as well.

Keywords are also inferred in this manner, so

```
   PyMOL> sh st
```

works too, as long as **show** is the PyMOL only command starting with "sh".

NOTE: PyMOL's command language continues to grow and develop, so it is important to use full−length commands and string arguments in scripts. Otherwise, you could not be sure that a later command or argument would not cause your abbreviation to become ambiguous. For example, "sh st" would no longer work if a **shutoff** command were added to the PyMOL language.

Other Typed Commands and Help

This "Getting Started" section used the most frequent PyMOL commands in very brief examples. The section titled "Simple Examples" shows other commands that combine representations, selections and property changes. More complicated examples appear in the section titled "Cookbook and Complex Examples," and a comprehensive listing of typed commands appears in the section titled "Command and API Reference."

To see a list of the keyword commands that are available in PyMOL on your computer screen, type **help** and "enter" (Typing TAB and "enter" will work too). Add the keyword if you want help on a particular command:

```
SYNTAX

   help keyword

EXAMPLE

   PyMOL> help load
```

PyMOL responds by displaying the manual page that discribes the command in the PyMOL viewer. Command line completion works inside of help, so if you don't remember the full keyword, type **help**, the first character or so of the keyword, and hit TAB. Python will display a list of possible help topics.

Click inside the viewer and hit escape to toggle back and forth between the display of the manual page or the list of commands and the molecules you have loaded in PyMOL.

All the keywords that PyMOL understands are listed alphabetically and described in the "Reference" section. PyMOL commands run on top of the Python programming language and may contain Python statements. After you type in a command and hit return, PyMOL will check whether the first word is one of its keywords (or if it can be extended into a keyword). If not, PyMOL will pass the command on to the Python interpreter. PyMOL will return a Python error message if neither a PyMOL nor a Python keyword is recognized.

Command Syntax and Atom Selections

Syntax

A typed PyMOL command always starts with a keyword that calls PyMOL to execute an action. It ends with a carriage return ("enter" on your keyboard).

The simplest commands consist of a keyword alone. For example, typing **quit** will end your PyMOL session. The **quit** command never takes an argument.

Many commands have default arguments, so you can type the keyword alone and PyMOL will supply the rest. For example, the default argument to **zoom** is the *selection−expression* **all**:

```
EXAMPLE

   PyMOL> zoom                    # All visible representations
                                  # are included in the view.
```

For some keywords, certain arguments are required and others are supplied by default. For example, the keyword **color** requires one argument, the *color−name*. As for **zoom**, the default *selection−expression* is **all**:

```
SYNTAX

   color color-name

   color color-name, selection-expression

EXAMPLES

   PyMOL> color red               # All the representations
                                  # are colored red.

   PyMOL> color red, name ca      # Only the representations of
                                  # atoms named c-alpha are colored red.
```

When you type a command that has more than one argument, **color** *color−name, selection−expression* in this case, a comma must separate the arguments.

*Selection−expression*s are an essential type of keyword argument. They can be simple or complex, with several different kinds of syntax.

Selection−expressions

*Selection−expression*s stand for lists of atoms in arguments that are subject to PyMOL commands. You can name the selections to facilitate their re−use, or you can specify them anonymously (without names). Object and selection names may include the upper or lower case characters A/a to Z/z, numerals 0 to 9, and the underscore character (_). Characters to avoid include:

```
! @ # $ % ^ &* ( ) ' " [ ] { } \ | ~ ` <> . ? /
```

*Selection−expression*s describe the class of atoms you are referencing. Most of them require identifiers to complete the specification. For example, the selector **resi** references biopolymer residues by sequence

number, and the identifier gives the number. The selector **name** references atoms according to the names described in the PDB, and the identifier gives the name (**ca** for alpha carbons, **cb** for beta carbons, etc). A handful of *selection−expression*s don't require identifiers, but most do.

PyMOL uses several logical operators to increase the generality or specificity of *selection−expression*s. Logical combinations of selectors can get complex, so PyMOL accepts short forms and macros that express them with a minimum of keystrokes. This section describes named−selections, and then gives the syntax for making selections in a progression from simple one−word selectors to complex combinations of selectors, using macros and short forms.

Named Atom Selections

Atom selections can be named for repeated use by using the **select** command:

```
SYNTAX

   select  selection-name, selection-expression
                                        # The  selection-name and
                                        # the  selection-expression
                                        # are both arguments to select
                                        # so they are separated by a comma.

EXAMPLE

   PyMOL> select bb, name c+o+n+ca   # Create an atom selection named "bb"
                                     # including all atoms named
                                     # "C","O","N", or "CA";
   PyMOL> color red, bb              # color the selection red,
   PyMOL> hide lines, bb             # hide the line representation,
   PyMOL> show sticks, bb            # show it using the stick representation,
   PyMOL> zoom bb                    # and zoom in on it.
```

In this case, the *selection−expression* is the property selector **name**, which takes the list of identifiers **ca+c+n+o** to complete the specification. Property selectors and their identifiers are discussed below.

Named atom selections appear in the PyMOL names list in the control panel. They are distinguished from objects by a surrounding set of parentheses. The control panel options available under the diamond menu differ between objects and atom−selections, because objects and named selections play slightly different roles in PyMOL. Named selections are pointers to subsets of data that are found under an object name. After an object is deleted, the data are no longer available, unless you reload the object. Any named selections that refer to atoms in that object will no longer work. But when named selections are deleted, the data are still available under the object name. Disabling objects eliminates them from the viewer, but disabling named−selections just turns off the pink dots that highlight them in the viewer.

Atom−selections, named or not, can span multiple objects:

```
EXAMPLE

   PyMOL> load $PYMOL_PATH/test/dat/fc.pdb
   PyMOL> load $PYMOL_PATH/test/dat/pept.pdb

   PyMOL> select alpha_c, name ca      # The named selection "alpha_c"
                                         # is created -- it includes atoms
                                         # in both "fc" and "pept" objects.
   PyMOL> color red, name ca             # "CA" atoms in both objects
```