Additional copies of this manual are available for $10 each, including postage, by writing to MIDAS Software Distribution, Computer Graphics Laboratory, School of Pharmacy, University of California, San Francisco, CA 94143-0446. Please include a check or money order payable to The Regents of the University of California with your request. Sorry, but purchase orders cannot be accepted. Orders received without payment enclosed will be returned unprocessed.

Introduction

Background

The **MidasPlus**™ molecular visualization system is a collection of programs developed by the Computer Graphics Laboratory at the University of California, San Francisco (UCSF). The major component of the MidasPlus system is an interactive graphics display program, **MIDAS** (Molecular Interactive Display and Simulation), designed for the display and manipulation of macromolecules such as proteins and nucleic acids. Many ancillary programs are also part of the system and allow for such features as computing the solvent-accessible surface of a molecule, calculating electrostatic potentials, and so on.

MidasPlus is the most recent in a series of interactive molecular graphics systems whose direct lineage extends back to the first developments in molecular graphics at Project MAC, Massachusetts Institute of Technology, in 1964.[1,2] National Institutes of Health (NIH) support began with the formation of the Computer Graphics Laboratory at Princeton University in 1969 under the leadership of Prof. Robert Langridge and resulted in a number of pioneering developments including CAAPS (Computer Aided Analysis of Protein Structure).[3] In 1976 Langridge moved this NIH research resource to UCSF. A new graphics package[4] was designed to operate under the UNIX[5] operating system and a new molecular graphics system (MMS) was also designed, initially in collaboration with the group under Prof. J. Kraut at U.C. San Diego. This evolved into a system, MIDS, which was good enough to accommodate the new developments in color graphics and made possible the display of interactive surfaces.[6] The MIDS system was used by numerous visitors to our laboratory in the late 1970's.

In 1980 we decided to redesign the system completely, making use of the lessons learned over the previous 15 years. The result, the original MIDAS system,[7,8] emphasized highly interactive display and manipulation, with a data structure designed for very fast access to large and complex molecules such as proteins and nucleic acids.[9] The system was originally developed on the UNIX operating system for use with an Evans and Sutherland Picture System 2 display; however, between 1982 and 1989 MIDAS was ''ported'' to a variety of graphics display engines (PS2, MPS, PS300 family, and Silicon Graphics IRIS family) and operating systems (BSD UNIX, System V UNIX and VMS). In addition, we began to distribute copies of this system, including *documented source code*, which has proven to be an excellent training tool as well as a starting point for other molecular modeling packages. The system has continued to evolve, often in response to new ideas received directly from MIDAS users. There are now more than 500 sites running MidasPlus in 38 states and 30 countries. Nearly 1,400 publications have resulted from work done at the UCSF Computer Graphics Laboratory using MIDAS and MidasPlus.

MidasPlus

In order to take advantage of the substantial advances in graphics display technology and workstation functionality and performance in the mid-1980's, we decided in 1989 to ''rewrite'' the MIDAS system. This work was done during the winter of 1989 and the resulting system was named MidasPlus. MidasPlus has significantly increased functionality and performance compared to its predecessor. This was accomplished while maintaining compatibility with the original MIDAS command language.

More recently, we have focused our efforts towards providing a software development framework for *others* to build on. We view the extensibility of MidasPlus as critical to its success and have concentrated on providing an open architecture and ''developer-friendly'' environment. This allows programmers to add new commands and

MidasPlus is a trademark of the UCSF Computer Graphics Laboratory.

[1] R. Langridge and A.W. MacEwan, in *Proceedings, IBM Scientific Computing Symposium on Computer Aided Experimentation* (1965).

[2] C. Levinthal, *Scientific American* **214** (6), 42-52 (1966).

[3] R. Langridge, *Federation Proceedings of the American Society of Experimental Biology* **33**, 2322-2328 (1974).

[4] T.E. Ferrin and R. Langridge, *Computer Graphics* **13**, 320-331 (1980).

[5] D.M. Ritchie and K. Thompson, *Communications of the ACM* **17**, 7 (1974). UNIX is a registered trademark of Unix Systems Laboratory.

[6] R. Langridge, T.E. Ferrin, I.D. Kuntz, M.L. Connolly, *Science* **211**, 661-666 (1981).

[7] T.E. Ferrin, C.C. Huang, L.E. Jarvis, and R. Langridge, *J. Mol. Graphics* **2**, 55 (1984).

[8] T.E. Ferrin, C.C. Huang, L.E. Jarvis, and R. Langridge, *J. Mol. Graphics* **6**, 13-27 (1988).

[9] T.E. Ferrin, C.C. Huang, L.E. Jarvis, and R. Langridge, *J. Mol. Graphics* **6**, 2-12 (1988).

functions to MidasPlus without modification of the MIDAS source code. The MidasPlus ‘‘delegate’’ facility makes it significantly easier for new or specialized features to be added, without requiring knowledge of the internals of a large and sophisticated interactive graphics program such as MIDAS. The first delegate program, *Builder*, was a module for performing site-directed drug design using a molecular lattice approach.[10] Many additional MidasPlus delegates have now also been developed (see table below). It is especially noteworthy that several of these MidasPlus delegates have been designed and implemented by graduate students and postdoctoral fellows. Appendix 5 describes the details of the MidasPlus delegate facility. Also, since the MidasPlus distribution is supplied as documented source code, users wishing to develop new delegate modules are free to use any of the existing delegates as a training tool and starting point for their own developments.

MidasPlus Delegates

Name	Author(s)[11]	Function
Builder[†]	D. Roe (Graduate student)	Assemble lead compounds
Conic	Huang, Pettersen, Couch, and Ferrin	Generate shadowed CPK-style molecular images
Density	C. Schafmeister (Graduate student)	Display X-PLOR electron density contour maps
Discern	C. Bayly (Postdoc) and C. Huang	Visualize multidimensional data
Gd	H. De Winter (REGA)	Contour GRID output / prepare GRID input
GDE[†]	M. Young (Graduate student)	Interface to Genetics Data Environment program
Ksdssp	C. Huang	Define protein secondary structure
Label3d	C. Huang and J. Newdoll	Create three-dimensional labels for images
Makems	S. Newmyer (Graduate student)	Create solvent-accessible surfaces within MIDAS
MidasMenu	Huang, Morris (Genentech) and Ferrin	Graphical user interface front-end for MidasPlus
Midasmovie	D. Konerding (Graduate student)	Animate trajectory of structures
Neon	T. Hynes (Genentech)	Generate models with solid stick bonds and shadows
Noeshow	E. Pettersen and S. Farr-Jones (Postdoc)	Display NOE constraints
Qpack[†]	L. Gregoret (Graduate student)	Protein structure analysis and modeling
Rainbow	E. Pettersen	Color protein backbones based on sequence position
Ribbonjr	C. Huang	Depict protein secondary structure
Stereoimg	E. Pettersen and C. Huang	Create space-filling stereo image pairs
Viewdock	C. Huang	Assist analysis of DOCK output

Future Development Plans

MidasPlus is a powerful tool for its target problem domain of molecular visualization. Nonetheless, as noted above, its original design dates to 1980 and significant advances in programming and visualization techniques have occurred since then. In order to take best advantage of these advances, we are designing and implementing a new molecular visualization package, code-named *Chimera*, to succeed MidasPlus. We anticipate Chimera building on the strengths of MidasPlus (*e.g.* atom selection syntax, extensibility) while incorporating new features such as:

- Integrated menu and command-line interfaces.
- A fully programmable object-oriented command language[12] with complete access to molecular data.
- Interactive ‘‘solid’’ representations of molecules.
- Advanced graphics techniques such as texture-mapping, transparency, and volume visualization.
- Support for web publishing (import/export VRML).
- Closer integration with other UCSF packages (*e.g.* Sparky, DOCK, AMBER).
- Increased portability.

[10]R.L. Lewis, D.C. Roe, C.C. Huang, T.E. Ferrin, R. Langridge and I.D. Kuntz, *J. Mol. Graphics* **10**, 66-78 (1992).

[11]Computer Graphics Laboratory staff unless otherwise noted.

[†]Not currently included in the MidasPlus package.

[12]Based on *Python*: http://www.python.org

Acknowledgements

The original version of MIDAS was written by Prof. Thomas Ferrin and Dr. Conrad Huang. Dr. Huang is also the principal author of MidasPlus, the MidasPlus delegate facility, and many of the current MidasPlus delegates. Eric Pettersen and Greg Couch have also contributed significantly to the large effort required in programming, testing and documenting the many features the MidasPlus system offers. All of these individuals are still affiliated with our laboratory. The Laboratory Director is Prof. Thomas Ferrin. Thanks are due to many of our existing MidasPlus users for countless suggestions over the years that led to improvements in the system. The development of MidasPlus is supported by the National Institutes of Health, National Center for Research Resources (NCRR), P41 RR-01081, T. Ferrin, Principal Investigator.

Because a substantial portion of the funding for our laboratory comes from a NCRR grant (P41 RR-01081) from the National Institutes of Health, it is critically important that publications resulting from work using the MidasPlus system or incorporating graphical images produced with MidasPlus acknowledge our laboratory. We ask that a statement similar to the following be used:

> **Molecular graphics images were produced using the MidasPlus package from the Computer Graphics Laboratory, University of California, San Francisco (supported by NIH P41 RR-01081).**

The article which describes the MIDAS/MidasPlus system and should be included in your references is:

> T. E. Ferrin, C. C. Huang, L. E. Jarvis and R. Langridge, ‘‘The MIDAS display system,’’ *J. Mol. Graphics* **6**, 13-27 (1988).

If you make use of **conic** images in your publication (**neon** is a preprocessor for **conic**) you should also include the following reference:

> C. C. Huang, E. F. Pettersen, T. E. Klein, T. E. Ferrin and R. Langridge, ‘‘Conic: A fast renderer for space-filling molecules with shadows,’’ *J. Mol. Graphics* **9**, 230-236 (1991).

Similarly, images of van der Waals surfaces can be referenced with:

> P. A. Bash, N. Pattabiraman, C. C. Huang, T. E. Ferrin and R. Langridge, ‘‘Van der Waals Surfaces in Molecular Modeling: Implementation with Real-Time Computer Graphics,’’ *Science* **222**, 1325-1327 (1983).

We would also appreciate receiving two reprints of any publications resulting from your work with MidasPlus.

Manual Organization

The MidasPlus User’s Manual is divided into four main sections. Part I, entitled ‘‘Getting Started,’’ is a short chapter designed to get new users going quickly and also provides pointers on where to learn more. Part II, ‘‘Command Reference Guide,’’ provides a detailed description of the MIDAS command language and the concise syntax and example usage of each of the many MIDAS commands. New users should be sure to consult the ‘‘Command Synopsis’’ subsection in Part II as it provides a list of all available MIDAS commands grouped together by function. Differences in the way MidasPlus acts on various workstations are explicitly noted throughout the manual. Part III, ‘‘Advanced Concepts,’’ discusses a variety of topics, including important details in MIDAS input file format, displaying molecular surfaces, computing electrostatic potential surfaces, and adding hydrogen atoms. Beginning users may skip Part III on first reading, but should be aware that the input data used by MIDAS is critical to a productive modeling session; hence, careful reading of this section is essential if you are using a data file that has a nonstandard format. Finally, the appendices describe additional details such as atom-naming conventions, special characters and symbols used by MIDAS, default options, aliases and device assignments, and the MidasPlus delegate mechanism. Especially noteworthy is Appendix 6, ‘‘MidasPlus Program Suite,’’ which describes many ancillary programs that are part of the overall MidasPlus package. The MidasPlus delegates listed above are described in detail in Appendix 6, as well as a wealth of ‘‘stand-alone’’ programs intended to complement the functionality provided by MIDAS. Lastly, there is a full descriptive index at the end of the manual which may be of use in locating particular topics of interest.

Recent changes to the MidasPlus User’s Manual are now indicated with a vertical bar in the right margin area of the manual, such as shown with this paragraph. Changes, additions and bug fixes are also detailed in the ‘‘Release Notes’’ document. Thus, users already familiar with MidasPlus can quickly determine what has changed in the current release.

E-mail Communications about Midas

The UCSF Computer Graphics lab maintains several e-mail addresses related to MidasPlus:

- `midas-inquiry@cgl.ucsf.edu` -- Inquiries about your current license, obtaining new licenses, or getting new MidasPlus releases.
- `midas-bugs@cgl.ucsf.edu` -- Bug reports for MidasPlus programs.
- `midas-ideas@cgl.ucsf.edu` -- Requests for enhancements or other suggestions.

Also, check ‘‘MidasPlus Web Resources’’ in Part I of this manual for information about MidasPlus-related web pages.

Part I: Getting Started

MidasPlus is capable of displaying:

(1) Molecular structures from information contained in Protein Data Bank[1] (PDB) format files,[2]

(2) Van der Waals surfaces associated with molecular structures,[3]

(3) Molecular surfaces computed by the DMS program,[4] or

(4) Graphics objects consisting of text, lines, and/or points.[5]

MidasPlus expects input PDB files to adhere fairly well to the PDB standard. However, some software packages generate ‘‘PDB files’’ with poor conformance to the PDB standard. If you have trouble displaying a file that is supposedly in PDB format, you should consult the ‘‘Protein Data Bank Format’’ section in Part III of this manual, which discusses the details of PDB file format as well as solutions to common problems found in non-conformant PDB files.

If you already have a PDB format file that is known to be correct, or you want to try out MidasPlus with one of the test files included with the software distribution, the following section shows you how to quickly display a model and then manipulate it. If MIDAS responds sluggishly on your system, consult the ‘‘Performance Issues with Midas’’ section in Part III of this manual for possible remedies.

1.1. Displaying the Model

The MidasPlus display program is used to display the PDB file. To invoke the MidasPlus display program give the command:

midas (on SGI or Digital DEC Alpha workstations)[6]

or

open /usr/local/midas/bin/Midas.app (on *NeXT* workstations)

This will bring up three windows[7]: a main modeling window, a command and reply area (below the main window), and a sideview and controls area (to the right of the main window). Depending on your system, the main window may be automatically sized and placed or you may be presented with a ‘‘rubberband box’’ for sizing and placing the main window. In either case, the secondary windows will place themselves in the correct relationship to the main window. If you move the main window, the others will follow and retain their same relative positions. This behavior can be overridden, if desired, with the command ‘‘`devopt move_windows off`.’’

After a few seconds delay, MIDAS should be ready for command line input at the ‘‘Command:’’ prompt. The *control panel* on the right side is described in detail below under **Manipulating the Model**.

[1]Bernstein, F. C. The Protein Data Bank: a Computer Archive. J Mol. Biol. 112, 535-542 (1977).

[2]Information about obtaining standard PDB files can be found in the ‘‘Obtaining Standard PDB files’’ section of Part I of this manual.

[3]As described under ‘‘VDW Surfaces’’ in Part III of this manual.

[4]As described under ‘‘Solvent-Accessible Surfaces’’ in Part III.

[5] As described under ‘‘Non-Molecular Graphics Objects’’ in Part III.

[6]Note that the directory */usr/local/midas/bin* must be on your execution path for the **midas** command to be found. See section 2.3, ‘‘Midas Start-Up’’ for further details.

[7]The interface for the OpenGL version of MIDAS is described here. On some older SGI systems, MIDAS uses the older IRIS GL graphics library for performance reasons. IRIS GL MIDAS presents a slightly different interface. All aspects of the described OpenGL interface apply except for the following. IRIS GL MIDAS uses only one window. The command and reply areas are embedded in the lower left part of the window, and the control panel is embedded in the lower right. The control panel initially shows only the side view and selection buttons. By clicking on the ‘‘Sliders’’ button, the control panel will switch to showing sliders (equivalent to the ‘‘Assignments’’ area of the OpenGL interface). Selecting ‘‘Sideview’’ will switch back. The selection buttons show only the model number, not the associated file name. If desired, it is possible to override the use of the OpenGL version of MIDAS in favor of the IRIS GL version (and vice versa). Consult the ‘‘SGI-Specific Performance Issues’’ section in Part III of this manual for details.

Once the ‘‘Command:’’ prompt appears, characters typed on the keyboard are interpreted directly as commands to MIDAS. You can type commands even if your mouse is in the main window rather than the command/reply window.

The first step in displaying a model is to ‘‘open’’ the model. If the name of the PDB file is *1gcn*, the command used to open the model is:

open 0 1gcn

The model name may also be a pathname to a model in another directory. The 0 (zero) in the above example is the optional model number. If the model number is left out, then the smallest unused (non-negative) model number is used.

1.2. Manipulating the Model

After execution of the **open** command completes, the model should appear in the main window. On a three-button mouse, the left and middle mouse buttons allow direct manipulation of the selected model(s) and the right button calls up a system menu. On a two-button mouse (like the *NeXT*’s), the original system menu functionality of the right button is lost and the right button acts just like the middle button on a three-button mouse. *On the NeXT, the right mouse button will not function unless the ‘‘Menu Button’’ has been enabled via the Preferences application.* Depressing and holding the left button controls model rotation. While the button is held down, a dashed blue circle is displayed on the screen. Moving the mouse outside the circle results in model rotation about the z axis. The area inside the circle is a ‘‘virtual trackball;’’ when the mouse is within this area it ‘‘grabs’’ the trackball and rotates it. The model rotates as if it were inside the trackball. Once comfortable with this interaction method, display of the dashed circle can be suppressed, if desired, by typing ‘‘`~set showsphere`.’’ The ‘‘icon’’ used for the mouse cursor changes when the left mouse button is depressed and also changes dependent on whether or not the mouse is inside or outside the trackball circle, regardless of whether or not the actual circle is displayed. Thus the mouse icon can be effectively used as a visual feedback clue to indicate the type of rotational motion currently in effect. Depressing and holding the middle mouse button allows dragging the model(s) left/right and up/down. In other words, the middle button controls global translation of the selected model(s). Depressing and holding the left and middle mouse buttons simultaneously results in z translation of the selected model(s).

More mouse-based interactions are possible via the *control panel*, located to the right of the main window. In the control panel appear controls for manipulating the selection state of models, the viewing parameters, and some general-purpose sliders.

The sideview shows a stylized representation of the molecule(s), as viewed from the side. Also represented are the viewer’s eyepoint (the box at the junction of the convergent lines) and the hither and yon clipping planes (the left and right vertical lines, respectively). A typical sideview area is shown in Figure 1.

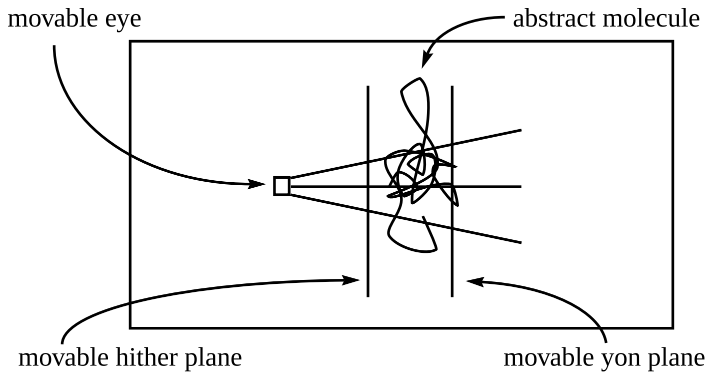


Figure 1. Sideview

The eyepoint and clipping planes can be moved by positioning the mouse over them, depressing the left button, and dragging the item. In addition, on non-*NeXT* systems, dragging a clipping plane with the left and middle mouse buttons depressed moves both planes apart or together (thickness), while dragging a plane with only the middle button depressed will move both planes in synchrony (sectioning). Moving the eyepoint changes the apparent size (scale) of the image. Holding down the Shift key will halve the rate of any motion controlled by the sideview control panel (including sliders) as well as translations (not rotations) of the models in the main MIDAS window.

The ‘‘Assignments’’ area of the control panel contains a series of pseudo-devices, referred to as sliders, for controlling various modeling interactions. Each slider is numbered, and may be assigned a function, such as model rotation or translation, clipping plane manipulation, or bond rotation adjustment. Some sliders are pre-assigned with functions, but can be reassigned as needed. See the **assign** command in Part II for further details. On non-*NeXT* systems, sliders are implemented as a series of four buttons for fast (double arrow) and slow (single arrow) motion in the left/right (or near/far) directions. Holding down the left mouse button over a slider button will cause motion appropriate to the slider label to occur continuously. On *NeXT* systems, sliders are represented with a ‘‘grabbable’’ bar that can be pulled to the left or right to control the rate of motion, and that will snap back when released.

The bottom section of the control panel contains controls related to model selection. On non-*NeXT* systems, this area contains a list of open models containing both the model number and file name from which the model was opened. Highlighted models are selected for motion and will respond to textual motion commands as well as mouse-directed movement. Un-highlighted (unselected) models will not. Clicking on a model label will toggle the selection status. When a new model is opened it is automatically selected. The pop-up list at the top of this panel section has two choices: ‘‘all’’ and ‘‘individually.’’ Choosing ‘‘all’’ will select all models; this is useful for global adjustment of the modeling session. Choosing ‘‘individually’’ will revert to the selections before ‘‘all’’ was chosen. On *NeXT* systems, the bottom control panel section simply contains an array of buttons with model numbers as labels, which can be used to toggle the selection status of the corresponding model. One button is labeled with ‘‘All’’ and can be used to toggle between selecting all models and selecting individual models, similar to the pop-up list described above. For all systems, when there are more open models than selection devices, you will need to use the **select** command instead of the mouse to select and deselect them.

On non-*NeXT* systems, each section of the control panel has a checkbox next to its title. Initially each checkbox is selected. Deselecting a checkbox will cause that section of the control panel to ‘‘collapse’’ and display only the title. Reselecting the checkbox will re-expand the corresponding control panel section.

Finally, the ‘‘Session’’ menu can be used for opening models or sessions, and saving sessions. On non-*NeXT* systems, it is located at the top of the main window. On *NeXT* systems, it is an item of the main MIDAS menu. The appropriate sub-menus are ‘‘Open...’’ (for opening sessions), ‘‘Import PDB...’’ (for opening PDB files), and ‘‘Save As..’’ (for saving sessions). Each one will bring up a panel to navigate the file system and select the file or session name of interest. If you already know the precise location of the PDB file or session, it is probably faster to use the **open**, **load**, or **save** commands (respectively).

From this point on, there are many MIDAS commands available for manipulating the model. These are described in detail in Part II, ‘‘Command Reference Guide,’’ of this document. The following categories of commands are provided to help you get started:

For information on:	The pertinent commands are:
Adding groups or new residues	**addgrp addaa delete swapaa swapna**
Coloring models	**color intensity set**
Displaying molecular surfaces	**surface vdw vdwopt**
Interactive manipulation of models	**assign select**
Labeling model components	**label rlabel**
Making videos	**rock roll set sleep source wait**
Recovering coordinates	**fix getcrd save write**
Rotating bonds	**rotat brotat reverse assign angle**
Selective display of model components	**chain display show**
Viewing in stereo	**stereo**

1.3. On-Line Help

MIDAS features an on-line help system. All commands documented in Part II of this manual are also available on-line. The command

help *command*

will produce a short synopsis of the specific command in question. The command

help

without any arguments produces a list of all available MIDAS commands. The help facility is very useful for both the novice and experienced MIDAS user and obviates constantly referring to the User's Manual for every command.

1.4. Tutorials

There are three tutorials available for MidasPlus. One of these is available from the World Wide Web, one from the MidasPlus distribution CD-ROM, and one is available from both. To determine if the CD-ROM tutorials have been installed on your system, check whether the main tutorials directory, /usr/local/midas/tutorials, exists. If it does, the tutorials have been installed. If not, you may want to have your administrator install them. Also, if you mount the distribution CD-ROM on your system, you can access the tutorials in the Midas-2.1/tutorials directory on the CD-ROM.

The main tutorials directory has two subdirectories, cgl/ and glasfeld/. The cgl/ subdirectory contains exercises developed for a MidasPlus class taught at UCSF. It is an introduction to most of the features of MidasPlus, as well as some basic UNIX commands. The README file in the cgl/ subdirectory has instructions on that tutorial's usage. The glasfeld/ subdirectory contains a hypertext tutorial developed for use in a course taught by Arthur Glasfeld at Reed College. It not only teaches many commands in MidasPlus, but also many aspects of structural biochemistry. You can use this tutorial by pointing your favorite web browser at file:/usr/local/midas/tutorials/glasfeld/intro.html. This tutorial is also available on the World Wide Web at http://www.cgl.ucsf.edu/home/glasfeld/tutorial/intro.html. The tutorial available only via the Web was developed by Dr. Bryan Jones at the Biomolecular Structure Center of the University of Washington for the benefit of users there. We have slightly modified the tutorial to make it generically applicable to any MidasPlus installation. It is a more condensed introduction to MidasPlus functionality than the Glasfeld tutorial. The URL is http://www.cgl.ucsf.edu/home/meng/tut/t.html.

1.4.1. Demonstration Images

There are demonstration images provided with the MidasPlus distribution. These images show various uses of the rendering programs available with MidasPlus. Each image is accompanied by a general description of how it was prepared.

If the demos were installed on your system (they are optional), then the sample images will be found in /usr/local/midas/demos/images. The file README.INDEX in the images directory contains information on the contents of the directory.

If the demos were not installed on your system, they can either be installed (consult the MidasPlus Installation Guide for details), or viewed directly from the MidasPlus CD-ROM, once mounted. On the CD-ROM, the images directory is Midas-21/demos/images.

1.5. MidasPlus Web Resources

The main MidasPlus web page is located at http://www.cgl.ucsf.edu/midasplus.html. Much useful information is linked to from this page, such as the on-line tutorials discussed in section 1.4. One particularly important link points to a page discussing known problems in the current MidasPlus release. This page not only details all currently known bugs, but includes workarounds and/or fixes if available.

There is also a delegate for viewing molecular dynamics trajectories available on the web. The delegate was written by David Konerding and is called Midasmovie. Information about the latest version of Midasmovie can be found at http://picasso.ucsf.edu/~dek/movie.html.

1.6. Obtaining Standard PDB Files

If your host is connected to the Internet, it is possible to obtain PDB files either via the World Wide Web or via anonymous ftp. To fetch files via the web, point your favorite web browser at http://www.pdb.bnl.gov and follow the ‘‘Searching and Browsing the PDB’’ link. To fetch files via anonymous ftp, ftp to the host ftp.pdb.bnl.gov (130.199.146.1). For detailed instructions on how to use ftp, type ‘‘man ftp’’ at your shell prompt. When you connect, use the login name ‘‘anonymous.’’ Any password will be accepted, though it is usual to give your normal Internet e-mail address as the password (*e.g.* joe@cgl.ucsf.edu). Upon logging in, you will receive a detailed informational message about usage of the server as well as available data files.

If you lack Internet access, or want a full PDB distribution but have limited bandwidth to the Internet, you can get information on obtaining copies of coordinates from the data bank by writing to Protein Data Bank, Chemistry Department – Building 555, Brookhaven National Laboratory, Upton, New York 11973 USA. Alternatively, contact can be made by e-mail to orders@pdb.pdb.bnl.gov or phone 516-344-5752 or fax 516-344-1376.

1.7. Helpful Hints

In perusing a manual of this length, it is easy to overlook some useful techniques for making common modeling tasks easier. This section presents a few techniques that seem to be missed frequently.

Shortcuts

- It is usually much easier to pick a desired atom off the display (see section 2.1.7, ‘‘Atom Picking’’) than to type in its atom specifier.
- Previously typed commands can be retrieved with Control-P, and edited if necessary (see Appendix 3, ‘‘Special Characters and Symbols Used in Midas’’).
- Frequently typed commands and command arguments can be shortened by using aliases (see the **alias** command). Also, Midas will accept abbreviated command names as described in section 2.4, ‘‘Command Synopsis.’’
- It is useful to become acquainted with the full power of atom specifiers (section 2.1); they have been extended significantly in recent releases. For example, you can color yellow all beta-sheet residues within 5 angstroms of a helix with the command ‘‘`color yellow /sheet & /helix z<5`,’’ or color green all hydrogens in model 0 within 2.5 angstroms of nitrogens in model 1 with the command ‘‘`color green #0@H= & #1@N= za<2.5`.’’
- On the SGI and DEC Alpha, you can push any window (including the MIDAS window) behind other windows by positioning the mouse cursor over the window and typing ALT-F3. Similarly, you can pull a window to

the front by putting the mouse cursor over it and typing ALT-F1. This is very convenient if you need to get to your shell windows frequently while using MIDAS.

- On the SGI only, there is program, called *cedit*, that is quite handy for determining what red-green-blue values to use to get a desired color. To start it, type `cedit` (or possibly `/usr/sbin/cedit`). Cedit displays three sliders, one each for red, green, and blue. By moving the level indicators up and down you can ‘‘mix in’’ various amounts of red, green, and blue into the composite color displayed in the rectangle on the right side of the cedit window. The number values below the slider bars vary from 0 to 255, so you need to divide by 255 to get the 0–1 range expected by MidasPlus commands or utilities that use RGB values.

Error Recovery

- If your models have been moved so that they are no longer visible on the display, the **window** command should make them visible again.
- It is prudent to save session files (see **save** command) periodically during long modeling sessions to avoid having one command inadvertently wipe out a carefully set-up view. However, if you have not saved a session, it is sometimes possible to recover by careful use of the **record** command. By default, MidasPlus records all your typed commands in a temporary file which is removed when MidasPlus is stopped. You can use this file to bring back the display that you had before the mistake (less any mouse movements of the models). To do this, run the command ‘‘`record` *filename*’’ to save the temporary file to a file called *filename*, use your favorite editor to remove the ‘‘record’’ command at the end of the file as well as any other commands you don’t want, then start a new MidasPlus session and in it run ‘‘`read` *filename*’’ to restore your work.

Miscellaneous Tips

- The *midas.tty* program can be used to run MidasPlus commands in batch mode, or to capture MidasPlus replies into a file. This can be useful if you have a repetitive non-interactive task (making the same amino acid substitution into dozens of molecules, for example) or need information presented in replies (such as RMS deviations from a match command). *Midas.tty* is fully described in Appendix 6 of this manual.

Part II: Command Reference Guide

2.1. Referencing Models, Residues and Atoms

2.1.1. Models, Residues and Atoms

MIDAS uses a hierarchical command syntax developed in 1980 by the UCSF Computer Graphics Laboratory for referencing models, residues and atoms. In each MIDAS model, molecules are made up of residues. The residues are chained together in a specific sequence to form the molecule. Each residue is made up of atoms organized according to the coordinates and connectivity information. This scheme reflects nature's organization of biomolecules: amino acid chains make up protein molecules and nucleotide chains make up DNA molecules.

MIDAS allows the user to display multiple models (molecules) simultaneously. These molecules are assigned model numbers by the user with the **open** command or a default by MIDAS. Each molecule consists of one or more residues, each of which has a unique associated residue number according to its location in the residue sequence. The atoms which make up the residue each have associated atom names which are unique within any single residue. Thus, any displayed atom may be uniquely described by its model number, residue number and atom name.

The residue and atom names are determined at the time the input file is read in, and generally match the standard Brookhaven Protein Data Bank (PDB) residue and atom names. The principal exception is that residue numbers in `HETATM` records have an asterisk (‘‘`*`’’) appended since residue numbers in `HETATM` records frequently duplicate those in `ATOM` records. The symbols for these reference levels are defined as follows[1]:

Atom Specification Symbols

Symbol	Reference Level	Definition
#	model number	a number assigned to the displayed model by the user via the **open** command *or* the file name that the model was **open**ed from.[2]
:	residue	a residue type (standard Protein Data Bank abbreviation) *or* residue sequence number *or* range of sequence numbers
@	atom	an atom name (standard Protein Data Bank abbreviation)

The following examples illustrate the use of these symbols for referencing models, residues and atoms. Note that the lack of either a residue specifier or an atom specifier or both is interpreted to mean ‘‘all’’ units of the associated reference level.

`#0`	(all atoms in all residues in model 0)
`#1:50`	(all atoms of residue 50 in model 1)
`#1:50*`	(all atoms of `HETATM` residue 50 in model 1)
`#0:12@CA`	(alpha carbon of residue 12 in model 0)

[1]Note: A summary of all special symbols described in this section appears in Appendix 3 of this document.

[2]If MIDAS had to modify the file name to locate the PDB file (see the documentation for the **set molpath** command in Part II), then the modified name must be used. The modified name can be shown with the **set filenames** command. In the OpenGL version of MIDAS, the modified name is also visible in the control panel.

Groups of atoms or residues may be specified. For example:

`#0:12@CA@N`	(alpha carbon of residue 12 in model 0 and nitrogen of residue 12 in model 0)
`#0:12@CA,N`	(alpha carbon and nitrogen of residue 12 in model 0)
`#1:LYS`	(all lysine residues in model 1)
`#3:45-83`	(range of residues 45 through 83 in model 3)

Notice in the above example that the two atoms, ‘CA’ and ‘N’, may be delimited by either a comma or the symbol ‘@’. In either case, the preceding (most recent) molecule and residue information applies to the named atoms. Using the ‘@’ notation specifies an ordering of the atoms and causes MIDAS to work on the named atoms in that order. Using commas specifies a ‘‘group’’ of atoms, in no particular order, which allows MIDAS to work on them in whatever order is most efficient. Thus, in specifications where the *order* of the atoms is significant (*e.g.* the **match** command), the ‘@’ notation should be used. For models and residues, the same conventions are followed, with ‘@’ replaced by ‘#’ or ‘:’, respectively. For example, for atoms on different residues but the same model:

`#1:12,14@CA`	(alpha carbons in residue 12 and residue 14)
`#1:12:14@CA`	(all atoms in residue 12 and alpha carbon in residue 14)
`#1:12-20@CA:14@N`	(alpha carbons in residues 12 through 20 and nitrogen in residue 14)
`:LYS@CA`	(alpha carbons in all lysine residues)

In the example above, the first statement gives two residues which make up a single residue specification. Therefore, the carbon atoms in both residues are selected. In the second example, the entire residue 12 and only the carbon in residue 14 are selected.

The residue sequence number has to be followed by the residue insertion code and the chain identifier (in that order) if they are present in the PDB file.

2.1.2. Wildcard Symbols

The global wildcard symbol ‘‘*’’ matches all atoms in a residue or all residues in a model. It stands alone as a symbol, *i.e.* it cannot be used to match parts of names or sequences, such as G* or *A.[3] To do that, use the ‘‘=’’ wildcard character. For example, **color red @c=** means to color all atoms whose names begin with the letter ‘‘c.’’ This works for residue names too (but not residue sequence numbers). The single-character wildcard symbol ‘‘?’’ is used to select atom *names* and residue *names* whose names follow patterns. ‘‘?’’ cannot be used to match sequence numbers. For example:

`#1:12@*`	(all atoms in residue 12 of model 1)
`#0,1,2:50-*@CA`	(all alpha carbon atoms in residues 50 to the end of models 0, 1 and 2)
`#2:G??`	(all three-character residue names which begin with the letter ‘G’ in model 2)
`#0:*@H@H?@H??`	(all hydrogen atoms with one, two or three letter names in model 0)

The percent symbol, ‘‘**%**,’’ may be used to specify every *n*th item where *n* is an integer. For example, #1:*%5 selects every fifth residue in model 1; #1:HIS@*%4 selects the fourth atom of each HIS residue.

[3]This is due to the fact that ‘*’ is a legal character in such names.

2.1.3. Atom Properties

Atom properties are specified with the ‘‘/’’ operator. The currently supported properties include **display**, **label**, **vdw**, **surface**, **visible**, **strand** (also **sheet**), **helix**, **mark**, and **color**. If a property name is preceded by a ‘‘**!**,’’ it means the atom must *not* have that property. The visible property is true if an atom actually appears on the screen. The strand and helix properties are true for atoms mentioned in PDB `SHEET` and `HELIX` records, respectively. The color and mark properties must be followed by an equals sign, ‘‘=,’’ and then respectively either a color designation (see the **color** command for details) or a mark name (see the **mark** command). The following specifier selects all atoms named CA which appear on the screen, are not labeled, and are not green or red:

```
@ca/visible,!label,!color=green,!color=red
```

When several properties are listed in one atom specifier, MIDAS returns only those atoms that have all the listed properties. For example,

```
@n/sheet,color=green
```

would specify green nitrogens in β sheets. To specify atoms that are green *or* that are in β sheets, a somewhat more verbose syntax is required:

```
@n/sheet @n/color=green
```

This latter technique can be employed whenever specifying atoms having one of several properties. The one exception to the multiple property behavior described above is for color properties. Since atoms cannot have multiple colors simultaneously, the default behavior for specifiers involving multiple color properties is to return atoms that have any of the specified colors. So, for example, an atom specifier of:

```
/sheet,color=red,color=green
```

would return all atoms that are red *or* green that are also in a β sheet.

2.1.4. Zone Specifiers

Zone specifiers are used to select atoms and residues that are within a given distance of the referenced atom(s). **z<** and **zr<** specify all *residues* within the given distance from the referenced atoms. **za<** specifies all *atoms* within the given distance. **z>**, **zr>**, and **za>** yield the sets complementary to their ‘‘<’’ counterparts. For example,

```
#1 za<10
```

selects all atoms within 10 angstroms of model 1.

2.1.5. Temperature Factors and Electrostatic Potentials

Atoms may be selected by temperature factor and surface points by electrostatic potential using the ‘‘<’’ and ‘‘>’’ symbols. Electrostatic potential selection requires the symbol ‘‘e>’’ or ‘‘e<’’ to select potentials above or below a specified value, respectively. Temperature factor selection requires the symbol ‘‘b>’’ or ‘‘b<’’ to select factors above or below a specified value, respectively. These symbols follow the *entire* atom specification and *must* have a preceding blank. For example,

`#1:HIS  b>25.0`	(all atoms in histidine residues in model 1 which have temperature factors exceeding 25.0)
`#1  e>10  e<20.0`	(all surface points in model 1 with potentials between 10 and 20 kcal/mole per elementary charge)

Note that electrostatic potentials only apply to surface points and are calculated and incorporated into surface files by the **esp** utility program; see Appendix 6 of this document for details.

2.1.6. Atom Intersections

Intersections of groups of atoms are handled with the ‘‘**&**’’ operator. For example, one may want all atoms in model 1 which are within 10 angstroms of model 0:

```
#1 & #0 z<10
```

2.1.7. Atom Picking

Atom picking allows a user to select atoms (one at a time) using the mouse instead of typing in the atom names on the keyboard. This is useful for identifying atoms whose names are not known and for ‘‘non-typists.’’ An atom can be picked by pointing the cursor at it and then clicking the left or middle mouse button while the ALT (SGI) or Alternate (*NeXT* or DEC Alpha) key is depressed. On non-*NeXT* systems, the cursor will change shape into a picking arrow while the picking key is being held down. When the pick is made, the corresponding atom specifier will be inserted in front of the command line cursor. When picking atoms, it may be necessary to rotate the molecule so that the desired atom is not obscured by other atoms, or so that neighbor atoms are not picked by mistake.

The command may be edited at any point using the editing commands detailed in Appendix 3. The command is executed when the user hits the RETURN key.

2.2. Command Overview

MIDAS commands allow the user a variety of modes of execution. The user may:

(1) Type in commands at the graphics system keyboard,

(2) Set up a command file named *.midasrc* which is automatically read each time MIDAS is executed,

(3) Set up a command file which can be executed via the **source** or **read** commands,

(4) Control movements via the mouse or auxiliary devices such as a joystick or ‘‘Spaceball.’’

Commands which are typed in at the graphics system keyboard are echoed in the reply area. Replies generated by MIDAS appear after the echoed command, and are prepended with a ‘>’. There may be several lines of reply messages. For example:

```
assign 0 clipping
> Clipping plane is missing
> Usage: assign [ slider_num [ function [ direction ] ] ]
```

appears in the reply area when the user inadvertently forgets to supply the required clipping plane argument to the **assign** command.

2.3. MIDAS Start-Up

The directory */usr/local/midas/bin* must be on your execution path for the *midas* command to be found. You can put the following line in your *.cshrc* file in your home directory to add the directory to your path:

```
set path = (/usr/local/midas/bin $path)
```

The above would take effect upon your next login. To have an immediate effect, just type the above line at your shell prompt.

The MIDAS commands in the start-up file *.midasrc* are executed each time MIDAS is executed. There is also a general system start-up file and host-specific start-up file executed each time MIDAS runs. These are located in /usr/local/midas/resource/midas/midas.rc and /usr/local/midas/resource/midas/rc/dev-*hostname* (respectively). Note that the ‘‘hostname’’ portion of the latter filename should be replaced with the name of the machine that the file applies to, as returned by the UNIX *hostname* command. The order of execution of start-up files proceeds as follows:

(1) The general system start-up file OR the file specified by the user’s environment variable ‘‘MIDASRC.’’ (See Appendix 4 of this document.)

(2) The *.midasrc* in the user’s home directory.

(3) The *.midasrc* in the user’s present working directory.[4]

(4) At this point any command-line session file or PDB files are opened.

(5) The host-specific start-up file.

(6) Any script file(s) specified with the **–s** command line option.

Start-up files are conveniently used for assigning sliders as well as defining aliases and setting display options (see the **assign**, **alias**, and **set** commands). Any legal MIDAS command, however, may be included in a start-up file.

MIDAS normally starts up in a window with a standard title, resize borders, *etc*. On a non-*NeXT* system, use the **–f** command line option to get a full-screen window with no border. If you always want full screen behavior when running MIDAS, you can put the following line in your *.cshrc* file:

```
alias midas midas -f
```

2.4. Command Synopsis

MIDAS commands may be grouped together on one line using the ‘‘;’’ character as a delimiter. For example:

label #1; color 32,s #1; color green,b #1

If a large system is being displayed, it might be advantageous to use such a compound command since the graphics image is drawn only after the entire command has been executed.

Each description of the MIDAS commands in this document contains a line indicating the correct usage of the command. The usage includes the command name appearing in **boldface** print followed by command line parameters in *italic* or roman print. Parameters appearing in *italics* require substitution of the appropriate name, digit, *etc*. by the user. Parameters appearing in roman print are literals and should be typed in as they appear in the usage line. Parameters which appear inside square brackets (‘‘[...]’’) are optional. All parameters not appearing inside square brackets are required for the command to execute. Keyword parameters are sometimes separated by vertical bars (‘‘|’’), which indicate that the keywords are mutually exclusive. Parameters named *atom_specification* always refer to a selection of atoms, residues and/or models as described in section 2.1. Note that omitting *atom_specification* is the same as specifying all atoms in all models.

MIDAS accepts abbreviated forms for all commands. Abbreviations are disambiguated by ranking commands by frequency of use. For example, ‘‘rec’’ may be substituted for **record**. Typing ‘‘re’’ happens to mean **reset** instead of **read**.

The commands available in MIDAS are summarized in the following tables and described individually in detail on subsequent pages:

[4] On the *NeXT* machine, note that if MIDAS is started by *open*ing a file with a *.pdb* extension or by double-clicking on the **MidasPlus** application icon, then MIDAS will be running in the user’s home directory.

MidasPlus Commands and Utilities Grouped by Function

Commands marked with an asterisk (‘‘’’) are not used directly at the MIDAS command prompt but are instead used either at the UNIX command line or via the MIDAS* delegate *facility. Their use and features are documented in Appendix 6 of this manual.*

Input/Output

bs*	convert **pdbrun**-style file to ball-and-stick printable PostScript
copy	send display image to a printer or file
load	load a saved session into MIDAS
open	open a PDB file, solvent-accessible surface, or object file for display
pdbopen*	PDB file browser
save	save a MIDAS session
write	output a model into a file

Model Movement

align	align two atoms along the z axis
assign	assign functions to sliders
cofr	change center of rotation
freeze	stop all motion
move	translate selected models
rock	rock a structure about the x, y or z axis
roll	roll a structure or bond rotation about the x, y, or z axis
select	select models for move, rock, roll, or turn commands
setcom	set molecule’s effective center of mass
speed	set the control speed of sliders and spaceball
turn	rotate a structure about the x, y, or z axis

Structure Information

angle	monitor bond or dihedral angle
discern*	visualize multidimensional data
distance	monitor atom distances
getcrd	return <x,y,z> coordinates for an atom
label	label atoms and residues
ribbonjr	ribbon representation of secondary structure
rlabel	enable residue labeling

Display Control

center	specify center of image
chain	chain specified atoms together
clip	move clipping planes
color	color bonds, labels and surfaces
display	display specified molecules, residues, atoms
intensity	set depth cue intensity at hither and yon clipping planes
label3d*	create arbitrary labels with depth when viewed in stereo
objdisplay	display specified non-molecule graphics object
push/pop	save/restore the current model orientations on/from a ‘‘stack’’
rainbow*	‘‘rainbow’’ color atom chains from red to blue
reset	reset all models to original orientations
ribbonjr	display using ribbon representation
savepos	save a model’s current orientation
scale	apply a scaling factor to all models
section	change displayed image’s cross section
set/unset	set options (especially *fullscreen*, *halfbond*, and *linewidth*)
show	display specified atoms and no others
stereo	specify whether to use stereo and in what manner
thickness	change thickness of the displayed image cross section

* Command documented in Appendix 6.

window	display the entire molecule on the screen

Molecular Surfaces

dms*	calculate a solvent-accessible molecular surface
esp*	calculate electrostatic potential
makems*	compute and display solvent-accessible surface of an atom specification
pdb2site*	convert PDB file to **dms** site file
surface	display a model's solvent-accessible surface
vdw	display van der Waals surface
vdwopt	set van der Waals surface options

Presentation Graphics

conic	display shadowed, space-filling image
ilabel*	label an SGI image with arbitrary text
itops*	convert an SGI image or TIFF file to Color PostScript
label3d*	create labels that **neon**/**ribbonjr** can render
neon	generate a molecular model with solid stick bonds and shadows
ps2illustrator*	convert **copy** output to Adobe Illustrator format
ribbonjr	display secondary structure ribbon image
stereoimg*	show rendered stereo pair
stereops2illustrator*	convert stereo **copy** output to Adobe Illustrator format

Bond Rotation

angle	monitor bond or dihedral angle
assign	assign functions to sliders
brotation	initiate a ‘‘backwards’’ bond rotation
fix	make bond rotations permanent
fixreverse	fix and reverse bond rotation
reverse	reverse the direction of a bond rotation
rotation	initiate a bond rotation
speed	set the control speed of sliders and spaceball

Structure Creation/Modification

addaa	add an amino acid to the end of a molecule
addgrp	add a new group to a residue
bond	make a bond between two atoms
delete	delete a group from a residue
gennuc*	generate DNA/RNA structure from sequence data
longbond*	break excessively long bonds
pdb2group*	create **addgrp**-style group description from PDB file
swapaa	exchange one amino acid for another
swapna	exchange one nucleotide for another

Multi-Model Positioning

match	superimpose two models
matrixcopy	copy transformation matrix from one model to another
matrixget	output a transformation matrix to a file
matrixset	set a transformation matrix from a file
set	set options (especially *independent* and *refmodel*)
watch	graphically monitor interatomic distances
watchopt	specify parameters used by **watch** command

* Command documented in Appendix 6.

Cooperation with Non-MidasPlus Packages

density*	display X-PLOR electron density maps
gd*	display energy contours generated by Peter Goodford's GRID program
midasmovie	show molecular dynamics trajectories
noeshow*	display NMR constraints on a model
viewdock*	assist viewing of sets of ligands identified by DOCK program

System Access

cd	change current working directory
midaspop	pop MIDAS window to front of other screen windows
midaspush	push MIDAS window behind other screen windows
stop	terminate the current MIDAS session
system	execute a UNIX shell command

PDB File Enhancement/Correction

dnacheck*	correct nucleotide pseudo-PDB files to conform to PDB standard
fixatname*	correct AMBER pseudo-PDB files so they are in standard PDB format
ksdssp	generate Protein Data Bank `HELIX` and `SHEET` records
uncryst*	expand `CRYST` records in PDB file to generate symmetric subunits
unmtrix*	expand `MTRIX` records in PDB coordinate file

Script Support

mrotate*	generate script that brings model(s) into specific orientation
read	read a command file
record	record all executed MidasPlus commands in a file
sleep	temporarily suspend all input processing
source	read and execute a command file
update	change coordinates of a model from a PDB file
wait	suspend input processing until model has stopped moving

Auxiliary Program Support

delegate	specify an action involving delegate program(s)
echo	display text in reply area
help	display contents of a text file on the screen
makemark	reserve name for future **mark** command
mark	associate a name with a group of atoms
matrixcopy	copy transformation matrix from one model to another
matrixget	output a transformation matrix to a file
matrixset	set a transformation matrix from a file
pdbrun	pipe PDB file describing current models to UNIX command
pickabort	abort a **pickatom** request
pickatom	used by delegates to request that the user pick an atom
run	execute a shell command and send output to MIDAS
update	change coordinates of a model from a PDB file

Video Support

expr-smpte*	generate times codes for use with *videodisk* utility
fade*	Fade from one Iris image to another
videodisk*	control V-LAN videodisk

* Command documented in Appendix 6.

Miscellaneous

alias	define or display command aliases
cd	change current working directory
colordef	define color name
colorrename	rename color name
devopt	set device-specific option
gentpl*	generate a MIDAS template from a PDB coordinate file
help	show information about MidasPlus commands
midas.tty*	terminal-based version of MIDAS display program
run2ses*	convert **pdbrun**-style file to MIDAS session file
set/unset	set options
stop	terminate the current MIDAS session
version	report MIDAS version number

* Command documented in Appendix 6.

The actions of many MIDAS commands may be reversed by preceding them with the tilde character (‘‘˜’’). This is essentially an ‘‘undo’’ for the following commands:

Reverse Command Functions

Command	Function
˜alias	delete an alias
˜angle	remove an angle monitor
˜assign	deactivate sliders
˜bond	remove a bond between two atoms
˜brotation	remove a ‘‘backwards’’ rotation
˜chain	break chaining for all atoms listed
˜clip	halt an ongoing clipping operation
˜cofr	use default center of rotation
˜display	remove atoms from the display
˜distance	remove a distance calculation
˜label	remove atom and residue labels
˜makemark	destroy mark name
˜mark	remove mark name from atoms
˜move	stop an ongoing move operation
˜objdisplay	undisplay a graphics object
˜open	close a model
˜pop	equivalent to **push**
˜push	equivalent to **pop**
˜rlabel	don’t display residue labels
˜rock	stop **rock** motion
˜roll	stop **roll** motion
˜rotation	remove a rotation
˜savepos	delete saved position
˜scale	stop ongoing scaling operation
˜section	stop an ongoing section operation
˜select	deselect a model
˜set	unset an option
˜setcom	use default center of mass
˜show	remove atoms from the display
˜stereo	equivalent to ‘‘**stereo** off’’
˜surface	undisplay a solvent-accessible surface
˜thickness	stop an ongoing thickness operation
˜turn	stop **turn** motion
˜vdw	remove a van der Waals surface display
˜watch	stop **watch** monitoring

MidasPlus Detailed Command Descriptions (alphabetical order)

2.4.1. Addaa

Usage: **addaa** *residue_type*, *residue_sequence* [, *conformation*] *residue*

Addaa adds an amino acid of type *residue_type*, with sequence number *residue_sequence*, in the specified conformation after the specified *residue*. *Conformation* may be one of:

EXT	extended (default)
ALPHA	alpha helix
PBETA	parallel beta sheet
ABETA	antiparallel beta sheet

Residue must be the last residue of a chain.

The temperature factor for the new residue is set to the highest currently found in the model.

Example:

addaa tyr,30 #0:29	Add **tyr** as residue **30** after residue **#0:29**

2.4.2. Addgrp

Usage: **addgrp** *group*, *bond_length*, *bond_angle* [, *dihedral_angle* [, *new_residue_name*]] *atom1 atom2 atom3*

Addgrp adds a new chemical group whose position is determined by the three specified atoms. The parameters required are the group name (which corresponds to a file in a directory, see below) the *bond_length* from *atom1* to the first atom of the added group, and the *bond_angle* formed by the group being added, *atom1* and *atom2*. *Dihedral_angle* (formed by the added group, *atom1*, *atom2*, and *atom3*) must be a positive value between 0 and 360, inclusive. It defaults to 0. *New_residue_name* defaults to the old residue name.

Group files are searched for in ${HOME}/groups/ [user's groups][5] and /usr/local/midas/resource/midas/groups/ [system groups], in that order. The user should check the system groups directory to determine if a description of the group he/she wishes to add exists, and if so what the file name is. If there is no system file for the group, the user may create one and place it in his/her personal groups directory. The easiest way to create a group file is to use the **pdb2group***(1)* utility (see Appendix 6), provided that the user has some PDB file containing the chemical group. Otherwise, the group description will have to be created by hand. The procedure for doing so is described in ‘‘Chemical Group Description Files’’ in Part III of this manual. A group name must contain only alphanumeric characters.

Note that adding a group creates a new residue which is colored white and has no labels. The temperature factor for the new residue is set to the highest currently found in the model.

See also: **delete**

[5]The user's groups directory location can be overridden by setting the GROUPS environment variable to a directory or a colon-separated list of directories. If the GROUPS environment variable is not set, and ${HOME}/groups/ does not exist, then the current directory will be searched instead.

2.4.3. Alias

Usage: **alias** [[ˆ]*name* [*wordlist...*]]

Usage: ˜**alias** [ˆ]*name*

Alias assigns to *name* the specified *wordlist.* All subsequent appearances of the space-delimited *name* will be substituted with the *wordlist.* If the *name* is preceded with a ‘ˆ’, then the substitution will only occur at the beginning of a command. This is useful for aliasing a long command to a short name without worrying about that same short name inadvertently being used (and expanded) in the middle of other commands. The *wordlist* may contain multiple commands separated by semicolons, in which case the *wordlist* must be embedded in double quotes, like this:

```
alias name "command1 ; command2"
```

The **alias** command without any arguments reports all current aliases. The **alias** command with *name* only reports the alias for that *name*. **˜alias** *name* deletes the alias for that *name*.

2.4.4. Align

Usage: **align** *atom1 atom2*

Align positions the model(s) containing the specified atoms such that the two atoms lie along the z axis at the center of the screen. *Atom2* is positioned in the front and *atom1* is in the back.

See also: **reset**, **push/pop**, **savepos**, **window**

2.4.5. Angle

Usage: **angle** [*angle_number*] *atom1 atom2 atom3* [*atom4*]

Usage: ˜**angle** *angle_number*

Angle monitors the angle in degrees between the three specified atoms. If four atoms are specified, the dihedral angle is monitored. The atoms need not be connected and no diagnostic is given if the atoms are not connected. Up to 16 angles (0–15) may be monitored simultaneously. If the angle number is not specified, MIDAS will assign one for you.

˜angle *angle_number* will remove the indicated angle monitor.

2.4.6. Assign

Usage: **assign** [*slider_number function* [*direction*]]

Usage: ˜**assign** *slider_number*

Assign is used to activate control panel sliders. Each slider’s number is displayed just to the left or right of the slider itself on the control panel. Usually only one slider may be assigned to any one function and reassignment of a function cancels the previous assignment. This default action of canceling a previous assignment can be disabled via the **unset reassign** command (see **set/unset**).

Function may be any of the following key words or the first two letters of the appropriate key word:

Slider Functions

Keyword	Function
translation	model translation
rotation	model and bond rotation
clipping	clipping planes
scaling	changes size of selected models
section	moves hither and yon planes in the same direction at the same rate
thickness	moves hither and yon planes in the opposing directions at the same rate
nothing	for canceling slider assignment

The *direction*, if applicable, may be any of:

Slider Directions

Direction	Designation	Applicability
x	the x axis	translation, rotation
y	the y axis	translation, rotation
z	the z axis	translation, rotation
h	hither plane	clipping, section, thickness
y	yon plane	clipping, section, thickness
0–15	rotation number	rotation

Note that if *direction* is an integer, the corresponding intramolecular bond rotation is assigned to the slider. *Direction* is required for the functions **translation**, **rotation**, and **clipping**.

The default device assignments are located in /usr/local/midas/resource/midas/midas.rc and are listed in Appendix 4 of this document. The user may find it convenient to make additional automatic assignments by constructing a ‘‘.midasrc’’ file in his or her home directory and/or present working directory. Each time MIDAS is executed, any ‘‘.midasrc’’ files in the user’s home directory *and* the current working directory are executed before user commands are processed.

The **assign** command without any arguments reports all current assignments. **˜assign** removes an assignment.

Example: **assign** 0 clipping h

See also: **brotation**, **select**, **rotation**

2.4.7. Bond

Usage: **bond** *atom1 atom2*

Usage: **˜bond** *atom1 atom2*

The **bond** command tells MIDAS that *atom1* and *atom2* are bonded. *Atom1* and *atom2* must be in the same model.

˜bond breaks an existing bond between two atoms.

2.4.8. Brotation

Usage: **brotation** [*rotation_number*] *atom1 atom2* [*atom3 atom4*]

Usage: ˜**brotation** *rotation_number*

Brotation produces a ‘‘backwards’’ rotation, *i.e.* the portion of the molecule which remains fixed in a **rotation** command is rotated in the **brotation** command and *vice versa*. Aside from that, the **brotation** command is identical to the **rotation** command. See **rotation** for an explanation of the syntax and behavior of **brotation**.

See also: **rotation**, **reverse**, **fix**, **fixreverse**

2.4.9. Bs

Usage: pdbrun conect **bs** [*options*]

Bs generates a ball-and-stick PostScript description of the scene currently displayed by MIDAS. The generated PostScript can be saved into a file for later use, or sent directly to a printer. **Bs** PostScript can be saved to a file with a command such as:

`pdbrun conect bs` [options] > *savefile*

Bs PostScript can be sent to a printer with a command such as:

`pdbrun conect bs` [options] `| lpr -P`*printername*

A more detailed description of **bs**, along with a list of available options, can be found in Appendix 6 of this document.

See also: **copy**

2.4.10. Cd

Usage: **cd** *path_name*

Cd changes the current working directory to *path_name*. If you are not familiar with the concept of directories and path names, see ‘‘UNIX for Beginners’’ (Kernighan) and the description of the ‘‘cd’’ command in section 1 of the UNIX User’s Manual. Note that all subsequent commands that have filename arguments are executed in the new working directory, for example, **pdbrun**, **record**, **save**, **source**, **write**, *etc*.

Example: **cd** ../crodna

2.4.11. Center

Usage: **center** *atom_specification*

Center places the center of the atoms in the *atom_specification* at the center of the screen.

See also: **window**

2.4.12. Chain

Usage: **chain** *atom_specification*

Usage: ˜**chain** *atom_specification*

The **chain** command draws pseudobonds between the specified atoms, undisplaying all others. This is particularly useful for displaying the backbone atoms of a protein. ˜**chain** is similar to ˜**display** except that only bonds/chains are undisplayed (labels, *etc*. remain).

Example: **chain** @ca chain all alpha carbons

See also: **display**, **show**

2.4.13. Clip

Usage: **clip** *plane* [*units* [*frames* [*wait_frames*]]]

Usage: ˜**clip** *plane*

The clipping planes may be moved relative to their current position by a specified number of *units*. *Units* is a positive or negative distance in angstroms. A positive number moves the plane towards the user. A negative number moves it away from the user. *Plane* may be either **hither** or **yon**.

Frames moves the clipping plane in the specified manner for the specified number of image update frames. *Wait_frames*, if specified, indicates the number of frames to wait before beginning the move. ˜**clip** will halt an ongoing clip. *Frames* and *wait_frames* default to 1 and 0, respectively. These parameters are useful for controlling the rate of clipping and are helpful when constructing MIDAS command scripts and making videos. MIDAS continues to process commands while the requested motion is in progress. To have command processing pause until the motion is finished, use the **wait** command.

Clip can be invoked without the *units* parameter, in which case it will report the distance to the clipping plane (from the ‘‘eye point’’).

See also: **intensity**, **section**, **thickness**, **wait**

2.4.14. Cofr

Usage: **cofr** [*atom_specification* | view]

Usage: ˜**cofr**

When invoked without any arguments, the **cofr** command reports the current center of rotation.

If given an *atom_specification*, the **cofr** command sets the center of the bounding box of the given atom(s) as the center of subsequent rotations.

If the **view** keyword is given instead, the center of rotation is set to be the center of the current view.

˜**cofr** causes MIDAS to use its default center of rotation (which is the center of mass of the selected models), instead of any previous user-specified one.

See also: **setcom**

2.4.15. Color

Usage: **color** *color_name*[[–*color_name*][,s][,l][,b][,v][,c]] *atom_specification*

The **color** command allows the user to selectively color bonds, labels and surfaces by model, residue, and atom. Other graphics objects (see ‘‘Non-Molecular Graphics Objects’’ in Part III) have colors embedded in their definitions and are unaffected by the **color** command. *Color_name* is a name that has been previously defined by the **colordef** command. A special *color_name* is **byatom**, which is a simple attempt at determining color from atom name. Atom name prefixes and their corresponding colors are as follows:

CL/BR	magenta
FE	gray
C	gray
O	red
N	light blue
H/D/digit	white
S	yellow
I/F	magenta
B	gray
others	blue

These colors are coded into MIDAS and cannot be easily changed. Other built-in *color_name*s are described in the **colordef** command.

Optionally, the user may specify one of the designations **s**, **l**, **b** or **v** to color surfaces, labels, bonds, and van der Waals surfaces, respectively. In the absence of these specifiers, MIDAS colors everything, *i.e.* bonds, surfaces, labels and van der Waals surfaces of all specified atoms.

Note that the solvent-accessible surface and the van der Waals surface may be displayed simultaneously, and thus colored separately. See the **surface** and **vdw** commands in this document.

The *atom_specification* uses the standard MIDAS syntax. Note that the **˜color** command will not work, *i.e.* models may not be ‘‘uncolored.’’

If no *atom_specification* is given, all open models are colored.

Examples:

color green #2:HIS	color all histidine residues in model 2 green.
color 20,s #3	color model 3’s surface color 20 (midway between blue and magenta).
color red,b #0:4@*	color all bonds in the fourth residue of model 0 red.
color blue,s #1 e<5	color all atom surfaces on model 1 with electrostatic potential less than 5 kcal/mole per elementary charge the color blue.
color red /color=blue	color red every atom that is blue.

Ranges of colors may be mapped onto models based on either surface electrostatic potential or atom temperature factor. A range of color is specified by either keywords or numbers, such as *blue-red* or *16-32*. By default, color hues are interpolated counterclockwise on a standard color wheel; the **c** designation changes the interpolation to be in the clockwise direction. For example, blue to red interpolated counterclockwise goes through magenta, whereas clockwise interpolation goes through green. To color models by electrostatic potential use the surface designation, **s**, with the color range. The ends of the color range are mapped to the lowest and highest potential values *of all open models*. The color of a surface point is the color in the color range that corresponds to its electrostatic potential in the potential range. If boundary conditions are given, *i.e.* **e>**, **e<**, the colors are mapped within the boundaries. Similarly, for temperature factors, use the **l**, **b**, or **v** designation and specify boundary conditions using **b<** and **b>**. The default designation maps bond color by temperature factor and surface color by electrostatic potential.

For finer distribution of potentials and temperature factors, broader color ranges should be used. For example, *blue-red* maps only 120 degrees of the color wheel, whereas *cyan-yellow* maps 240 degrees. Alternatively, the interpolation direction may be changed by specifying the **c** designator.

Examples:

color blue-red,s #0	color lowest potential surface blue and highest red. Intermediate potentials are mapped to shades of magenta.
color red-blue,b #0	color the bonds of the lowest temperature factor atoms red and highest blue.
color 8–32,s #0 e>–20 e<20	color *ms* surfaces with potential –20 kcal/mole per elementary charge cyan (color 8) and with potential +20 kcal/mole per elementary charge orange (color 32). Intermediate potentials are mapped to shades of blue, magenta, and red.

Note that displaying many colors, especially color ranges, has the potential of slowing down interactive response time significantly.

See also: **colordef**, **colorrename**

2.4.16. Colordef

Usage: **colordef** *color_name red green blue*

Usage: **colordef** *color_name existing_color_name*

colordef adds the given *color_name* to the list of color names that MIDAS recognizes. A color name is an arbitrary sequence of letters, digits, and underscores. A color name may be redefined at will. The *red*, *green*, and *blue* arguments must be floating point numbers between zero and one, inclusive.

By convention, *red*, *green*, *blue*, *cyan*, *yellow*, *magenta*, *white*, *gray*, *grey*, and *black* are predefined. In OpenGL versions of MIDAS, all X window system color names are also predefined. They can be listed by running the program /usr/bin/X11/showrgb. Also, for backwards compatibility, the integers from 0 to 65 are in fact color names that are predefined and may not be redefined.

Numerical Color Mapping

Color name	Equivalent color number	colordef value		
		red	green	blue
green	1	0.0	1.0	0.0
cyan	8	0.0	1.0	1.0
blue	16	0.0	0.0	1.0
magenta	24	1.0	0.0	1.0
red	32	1.0	0.0	0.0
yellow	48	1.0	1.0	0.0
white	0	1.0	1.0	1.0
black	64	0.0	0.0	0.0
gray/grey	65	0.5	0.5	0.5

Examples:

colordef activesite red	create a special color for the active site

On the *NeXT*, there is a color (called *colorpanel*) that is constantly redefined to be the same as the current color in the color panel when the Color Panel is active. The *colorpanel* color can be used in commands just like any normal color name.

See also: **color**, **colorrename**

2.4.17. Colorrename

Usage: **colorrename** *color_name new_color_name*

colorrename replaces the given *color_name* with the *new_color_name*. The new color name is an arbitrary sequence of letters, digits, and underscores. This command is most useful on the *NeXT*, where one might color the active site with the **colorpanel** color, and then rename it to a different name to save it.

See also: **color**, **colordef**

2.4.18. Conic

Usage: **conic** *options*

The **conic** command produces a space-filling rendering of the displayed molecule(s). The current orientation and coloring is retained. Each atom is depicted as a sphere of the appropriate radius with realistic highlighting and shadowing effects. This image is not interactive and takes anywhere from a few seconds to a few minutes to produce. Clicking the left button returns to MIDAS.

There are many options, which are detailed fully in the *conic* manual page included in Appendix 6 of this document. (The **conic** command is actually an alias that executes the **pdbrun** command and sends data to the **conic** program.)

See also: **ribbon**, **pdbrun**

2.4.19. Copy

Usage: **copy** [date] [box] [flat] [bg|background *color*] [intensity *0-1*] [printer *printer*] [file *file*] [title *title* ...]

The **copy** command sends a PostScript description of the current picture to a printer or disk file.

The optional keyword **date** puts today's date in the lower left hand corner of the copy. The optional keyword **box** draws a heavy border around the copy. The **flat** option forces all vectors (line segments) to be the same thickness, thereby disabling the pseudo-depthcuing effect created with variable width lines.

The **background** option sets the background copy color without changing what is shown on the screen. Similarly, the **intensity** option controls the color intensity of the copy background without affecting the screen background.

If the **printer** keyword is given, the copy is sent to *printer*. If the **file** keyword is given, the Postscript is saved in *file*. If neither is given, the copy is sent to the default system printer.

If the **title** keyword is given, the entire rest of the line is taken to be the title for the picture. Therefore, if **title** is specified, it must be the last thing in the **copy** command. If no title is given, you will be prompted for the title. If no title is desired, simply hit RETURN at the prompt. The title is centered at the bottom of the picture.

See also: **bs**

2.4.20. Delegate

Usage: **delegate** list

Usage: **delegate** start *name command* [*command_options*]

Usage: **delegate** stop *name*

The **list** subcommand lists all the active delegates.

The **start** and **stop** subcommands start and terminate a delegate of the given *name*, respectively.

The delegate mechanism allows other programs to extend the interactive capabilities of MidasPlus. The delegate mechanism is discussed in detail in Appendix 5 of this manual. Note that the *discern*(1) utility described in Appendix 6 of this document is a MIDAS delegate. For a detailed description of an application developed using delegates, see ''Automated Site-Directed Drug Design Using Molecular Lattices'' in the *Journal of Molecular Graphics*, volume 10, number 2, June 1992 [R. A. Lewis, *et al.*].

Note that MIDAS first searches the directory */usr/local/midas/lib/midas* for *command* before examining the rest of the user's execution path. This is useful to avoid name conflicts between MIDAS commands using the **run**/**pdbrun**/**delegate** mechanism and UNIX commands of the same name.

Finally, it should also be noted that a delegate's state is not saved in MidasPlus session files. If a session is restored, no delegates will be active.

See also: **pdbrun**, **pickatom**, **run**, **system**

2.4.21. Delete

Usage: **delete** *new_residue_type atom_specification*

Delete removes a branch of atoms from a residue. All atoms from *atom_specification* to the end of the side chain are deleted from the residue. Deleting from a main chain atom produces unpredictable results. Because the structure of the residue changes with deletion, a *new_residue_type* must be specified to differentiate between the original and new residue types.

See also: **addgrp**, **bond**

2.4.22. Density

Usage: delegate start *dens* **density** *mapfile1* [*mapfile2* ...]

The **density** delegate shows electron density contours at selected density levels from X-PLOR map files. Details of this delegate's usage can be found in Appendix 6 of this document.

2.4.23. Devopt

Usage: **devopt** [*option* [*argument*]]

Devopt sets device-specific options.

Issuing a **devopt** command with no arguments will list the options available on your machine as well as each option's current setting.

The *NeXT* version does not have any device options. The SGI Iris GL graphics library version (older SGI hardware) has a few options — see the last section below. All of the X/Motif/OpenGL versions (SGI, DEC UNIX, IBM AIX, PC Linux, others) support the rest.

X windows-related:

xlocations
: Report locations of graphics window and auxiliary windows in a form suitable for a .Xdefaults file.

move_windows on **|** off
: Turn on and off the moving of auxiliary windows (command panel and control panel) when the graphics window is moved.

win_placement shell **|** frame
: The **move_windows** code needs to know how the window manager places windows. One of these will be better than the other. You need to move the graphics windows after changing this option to see what effect it has.

spaceball_axes world **|** mouse
: 6D input devices (3D rotation and 3D translation), such as the Logitech Magellan or the Spacetec IMC SpaceBall, can have their axes interpreted like the mouse (motion perpendicular to screen) or like the real world (up is up) — assuming that the computer monitor screen is perpendicular to the desktop and that the input device is parallel to the desktop.

Stereo-related:

See the MidasPlus Installation Guide for a more involved discussion of these choices. MIDAS only supports hardware stereo on DEC's and SGI's.

switch_stereo on **|** off
: Control whether will try to switch the video to a stereo format when going into stereo (and back when leaving stereo). If your monitor has to be switched by hand to show stereo, then you may want to turn this option off. On SGI's, if the default visual is a stereo visual, then you will want to turn this off.

stereo_monitor *format*
: SGI only. Choose the setmon(1) format (*e.g.* `1280x1025_96s`) for stereo. Low-end systems (interlaced stereo) can only pick `str_rect` or `str_bot`. High-end systems may have multiple choices.

normal_monitor *format*
: SGI only. Choose the setmon(1) format (*e.g.* `72`) to return to after leaving stereo. This is initialized from the value of ''/sbin/nvram monitor.'' The default is 60, which matches SGI's default.

stereo_pointer_adjust on **|** off
: DEC only. On older DEC systems (*e.g.* ZLX series), when the monitor goes into interlaced stereo, the mouse does not. Turn on to compensate.

OpenGL-related:

These options trade off visual aesthetics for speed — when they are turned off, the graphics is faster. They are all on by default.

smooth on **|** off
: Turns antialiasing of lines and points on and off. Turning this option off can speed up line drawing by a factor of 3. DEC Alphas with the old PXG graphics are unusable unless **smooth** is off (and not much better anyway).

antialias_points on **|** off
: Controls whether points are antialiased — must be set before **smooth** is turned on.

dither on **|** off
: On systems with a low number of graphics bitplanes ($\leq$ 24), dithering makes the images look better. The effect is more dramatic when the linewidth is greater than 1.

zbuffer on **|** off
: Use zbuffer to resolve occlusion. If you turn this off and leave **smooth** and **antialias_points** on, MIDAS will act like a vector graphics display.

Miscellaneous:

memlock on **|** off
: If MIDAS is setuid root, then it can lock all of its data into memory so it won't be paged out by competing processes. This significantly enhances MIDAS' performance on systems where large memory background processes are running. Also, to the best of our knowledge, there are no security holes opened by enabling this option. Since MIDAS must be setuid root for this option to work, your system administrator has to explicitly enable this during MidasPlus installion.

GL-related:

smooth on **|** off
Same as OpenGL version.

antialias_points on **|** off
Same as OpenGL version.

colormap on **|** off
Use a colormap for antialiasing lines. Limits the number of colors you can have (22 maximum), but is much faster on ancient SGI's.

ucsf_stereo on **|** off
Setting this option on indicates that the computer monitor needs to be switched into stereo video mode by hand.

See also: **set**

2.4.24. Discern

Usage: delegate start *disc* **discern**

The **discern** delegate allows the visualization of arbitrary multidimensional data by using shape and color in additional to the Cartesian coordinate dimensions. Details of this delegate's usage can be found in Appendix 6 of this document.

2.4.25. Display

Usage: **display** *atom_specification*

Usage: **˜display** *atom_specification*

Display selectively displays the atoms of a model. The *atom_specification* may be any combination of molecules, residues and atoms for the currently open models. To display only selected portions of a model, use **˜display** to remove the unwanted atoms and labels. The **display**, **label**, **vdw** and **surface** commands together allow any combination of bonds, labels and atom surfaces to be displayed.

See also: **chain**, **show**, **label**, **surface**, **vdw**

2.4.26. Distance

Usage: **distance** [*distance_number*] *atom1 atom2*

Usage: **˜distance** *distance_number*

Distance dynamically calculates and displays distances in angstroms between specified atoms. A dotted line is drawn between each pair of atoms for which a distance calculation is active. The user may assign a *distance_number* between 0 and 15, inclusive, to each active distance calculation. If *distance_number* is not given, one will be assigned for you. The distance calculation may be deactivated using **˜distance** *distance_number*. *Atom1* and *atom2* may be any two atoms in the displayed models, specified in standard MIDAS atom selection syntax.

Examples:

distance 1 #1:12@CA #0:47@CA	assign distance 1
˜**distance** 0,1	remove distances 0 and 1

See also: **watch**

2.4.27. Echo

Usage: **echo** *text*

Echo places all of the *text* argument into the MIDAS reply buffer visible at the bottom of the graphics window. **Echo** may be used by MIDAS scripts to send messages back to the user.

2.4.28. Fix

Usage: **fix** *rotation_number*

The **fix** command removes a previously activated torsional bond rotation and leaves the structure as currently displayed. It is necessary to **fix** a rotation before giving a **save** command if the current position of the rotation is to be reflected in the **save**d PDB file of the model with the active rotation.

2.4.29. Fixreverse

Usage: **fixreverse** *rotation_number*

Fixreverse fixes the named bond rotation and activates the reverse bond rotation using the same *rotation_number*. This command insures that the model will not move relative to other displayed models as is often the case when the **reverse** command is used.

Warning: In the case of multiple bond rotations, *if* the set of atoms rotated by a given bond rotation includes the pivotal atom of another bond rotation and *if* the set of atoms rotated by this second rotation includes the pivotal atom of the first, *then* MIDAS will not allow such rotations. In executing **fixreverse**, reversing the bond rotation may cause such a conflict and be disallowed. Reversing bond rotations must be done in an appropriate order such that intermediate combinations are legal (see **rotation**).

2.4.30. Freeze

Usage: **freeze**

Freeze stops all motion on the screen.

See also: **rock**, **roll**

2.4.31. Gd

Usage: delegate start *g* **gd** [–r *filename*]

The **gd** delegate allows the visualization of three-dimensional energy maps as output by Peter Goodford's GRID program. Details of this delegate's usage can be found in Appendix 6 of this document.

2.4.32. Getcrd

Usage: **getcrd** *atom_specification*

The **getcrd** command returns the untransformed x, y and z coordinates for the atom specified. The *atom_specification* must select one only atom. The coordinates are returned in the command reply area at the bottom of the MIDAS window.

2.4.33. Help

Usage: **help** [*topic* | *filename*]

The **help** command displays information on the selected topic. If no *topic* is specified, MIDAS displays a list of all available topics. If, instead of a topic, a full pathname to a file (*i.e.* with a leading '/') is given, the contents of that file are displayed. This is primarily useful for delegate programs that wish to display help specific to the delegate.

2.4.34. Intensity

Usage: **intensity** *hither_value* [*yon_value*]

Intensity changes the value of the intensity at the hither and yon clipping planes. The values range from 0 to 1. Default values are 1 for hither (maximum intensity) and 0.2 for yon (minimum intensity). Note that the location of the hither and yon clipping planes may be changed with the **clip** command or by manipulating the sideview or sliders in the control panel. For backwards compatibility, it is legal to use intensity values in the range 0–255 (which are then mapped to 0–1).

See also: **clip**

2.4.35. Ksdssp

Usage: **ksdssp** *options*

The **ksdssp** (*K*absch and *S*ander *d*efine *s*econdary *s*tructure of *p*roteins) command computes the regions of secondary structure in the displayed models. Many PDB files have their secondary structure information supplied in `HELIX` and `SHEET` records, but some do not and **ksdssp** is useful in these cases. The MIDAS **ribbonjr** command requires secondary structure information in order to make an accurate rendering of displayed models.

To do the secondary structure calculation, the **ksdssp** command invokes a separate program, also called *ksdssp*, which is described in Appendix 6 of this manual. Options given to the **ksdssp** command are passed on to the *ksdssp* program, so consult Appendix 6 for a description of the options. Typically, running **ksdssp** without *any* options does a reasonable job of computing secondary structure; nonetheless, in some cases options are necessary. Note that the ‘‘PDB file’’ and ‘‘output file’’ arguments mentioned in the *ksdssp* manual page should never be supplied to the **ksdssp** command within MIDAS; they should only be given when the *ksdssp* program is run outside of MIDAS.

See also: **ribbonjr**

2.4.36. Label

Usage: **label** *atom_specification*

Usage: **˜label** *atom_specification*

Atoms and residues are labeled appropriately. *Atom_specification* may include any atoms, residues or models. The residue sequence number and type appears after the first labeled atom in the residue.

Examples:

label #3	label everything in model 3
label #2:HIS	label all histidine residues in model 2
˜label #2:40	unlabel the 40th residue in model 2

See also: **rlabel**, **label3d**

2.4.37. Label3d

Usage: delegate start *ll* **label3d**

The **label3d** delegate allows the placement of text labels at arbitrary three-dimensional coordinates in the modeling scene. Such labels can be imaged by **neon** and **ribbonjr**. Details of this delegate’s usage can be found in Appendix 6 of this document.

See also: **label**

2.4.38. Link

Usage: **link** *residue*

Usage: **˜link** *residue*

This command is obsolete and is provided for backwards compatibility with old MIDAS command scripts. Use **bond** instead.

See also: **bond**

2.4.39. Load

Usage: **load** *session_name*

Load restores the state of a saved modeling session (see the **save** command). The current modeling session will be replaced with the saved session. However, since delegate states are not saved in session files, any active delegates will remain active.

See also: **open**, **save**

2.4.40. Longbond

Usage: pdbrun conect longbond [–**l** *bondlength*] [–**r** *bond_ratio*]

The **longbond** command is used to break excessively long bonds. Such bonds frequently result when opening PDB files that lack proper TER or HETATM records. Details of this command's usage can be found in Appendix 6 of this document.

2.4.41. Makemark

Usage: **makemark** [*name*]

The **makemark** command reserves *name* for use in future **mark** commands. The **makemark** command with no argument returns a list of currently reserved mark names. A reserved mark name can be freed for reuse with **˜makemark**.

See also: **mark**

2.4.42. Makems

Usage: delegate start *ms* **makems**

The **makems** delegate computes and displays the solvent-accessible molecular surface of a MIDAS model. Details of this delegate's usage can be found in Appendix 6 of this document.

See also: **open**, **surface**, **vdw**

2.4.43. Mark

Usage: **mark** *name atom_specification*

The **mark** command associates the given *name* with the set of atoms specified. The mark can be used in later atom specifications with the syntax

/mark=*name*

which will intersect the set of marked atoms with any other parts of the atom specification. This is explained in detail in ''Referencing Models, Residues, and Atoms'' in Part II of this manual. The mark name can also be used in a **pdbrun** command to limit the atoms involved to those in the mark.

Note that the **makemark** command must be used to reserve the mark name before the **mark** command can be used.

Using **˜mark** will remove the specified atoms from the named mark. Also note that the **mark** command marks *additional* atoms with the given name. Atoms marked with that name from previous **mark** commands are still so marked. To mark *just* the atoms given in the **mark** command, it would be necessary to first clear out the mark with **˜mark**.

See also: **makemark**, **pdbrun**

2.4.44. Match

Usage: **match** [selected] *atom_specification*

The **match** command uses the least squares fit method to superimpose two models. The *atom_specification* should contain an equal number of atoms from two different models.

The atoms are matched according to the order in which they are specified, *i.e.* the first atom of the first model is matched to the first atom of the second model, second atom to second atom, *etc*. The MIDAS command syntax allows much flexibility in specification, *i.e.* atom specifications can use all the shorthands available for related atoms. For example, the user might match models 1 and 2 thus:

match #1:3@C1@C2@P@O2 #2:3@C1@C2@P@O2

MIDAS will transform the first model so that its atoms overlay those of the second model. Specifying the **selected** option will make **match** transform not only the first model but all selected models as well, using the same transformation that was applied to the first model.

The user should be aware that the order in which atoms are specified in a list does not necessarily force the order in the match. For example,

match #1:3@C1,C2,P,O2 #2:3@C1,C2,P,O2

is not specific as to the order of the atoms C1, C2, P and O2. In this case, MIDAS will order the atoms as they occur in the residue connectivity for residue 3. If residue 3 of model 1 and residue 3 of model 2 are the same, this is not a concern. If they are different, however, then the order is not specific and the models may not be superimposed in the way the user would expect. Ordering may be forced by using the @ designation:

match #1:3@C1@C2@P@O2 #2:3@C1@C2@P@O2

The RMS error value from the least squares fit is returned in the command reply area at the bottom of the MIDAS window. The RMS error is only for those atoms named in the **match** command line (since the models being matched may or may not have identical numbers of atoms).

2.4.45. Matrixcopy/Matrixget/Matrixset

Usage: **matrixcopy** *from_model to_model*

Usage: **matrixget** *filename*

Usage: **matrixset** *filename*

Matrixcopy makes the 4x4 transformation matrix of model *to_model* the same as that of model *from_model*.

Matrixget prints the current 4x4 transformation matrices to a file named *filename*.

Matrixset reads matrices from the file named *filename* and sets the current transformation matrices (using the same file format as **matrixget**).

See also: **cofr**, **getcrd**

2.4.46. Midasmovie

Midasmovie is a delegate for displaying molecular dynamics trajectories, written by David Konerding. **Midasmovie** is not provided with the MidasPlus distribution, but is available free via the web. You can find information about the installation and use of **midasmovie** by pointing your favorite web browser at http://picasso.ucsf.edu/~dek/movie.html.

2.4.47. Midaspush/Midaspop

Usage: **midaspush**

Usage: **midaspop**

Midaspush pushes the MIDAS display window behind all other screen windows and icons. These windows and icons can then be used normally. **Midaspop** brings MIDAS back to the top. Make sure that the mouse cursor is over the MIDAS window when **midaspop** is typed because MIDAS will not receive keystrokes typed with the mouse cursor positioned over other screen windows. If you accidentally type with the mouse incorrectly positioned, erase your input in the other window, then move the mouse over the MIDAS window and type **midaspop**.

Note that if MIDAS is not using the full screen, and therefore has a window frame around it, it is possible to use the standard mouse techniques for pushing and popping windows. Also, on non-*NeXT* systems, clicking the right-mouse button while holding down the ALT key will bring up a window-manipulation menu when done over the MIDAS window.

2.4.48. Move

Usage: **move** *axis units* [*frames* [*wait_frames*]]

Usage: **~move** *axis*

The **move** command translates the selected molecule(s) along the specified *axis*. *Axis* may be *x, y* or *z*. *Units* is a floating point number in angstroms. A positive value for *units* indicates translation to the right, up, or toward the user for the x, y, and z axes, respectively.

Frames moves the models in the specified manner for the specified number of image update frames. *Wait_frames,* if specified, indicates the number of frames to wait before beginning the move. **~move** will halt an ongoing move. *Frames* and *wait_frames* default to 1 and 0, respectively. These parameters are useful for controlling the rate of motion and are helpful when constructing MIDAS command scripts and making videos. MIDAS continues to process commands while the requested motion is in progress. To have command processing pause until the motion is finished, use the **wait** command.

If the **move** command does not work, it is likely you have failed to **select** the target models.

See also: **select**, **wait**

2.4.49. Mrotate

Usage: delegate start *mr* **mrotate**

The **mrotate** delegate is used to interpolate smoothly between different model positionings. The interpolation can be shown directly in MIDAS or saved as MIDAS commands to a text file. Details of this delegate's usage can be found in Appendix 6 of this document.

2.4.50. Neon

Usage: **neon** *options*

The **neon** command produces a three-dimensional representation of the displayed molecule(s) with appropriate shadows. The molecule(s) can be depicted as space-filling spheres, balls and sticks, sticks, or tubes. The current orientation and coloring is retained. This image is not interactive and takes anywhere from a minute to several minutes to produce. After the image is displayed on an SGI, clicking the left mouse button returns to MidasPlus.

There are many options, which are detailed fully in the *neon* manual page included in Appendix 6 of this document. (The **neon** command is actually an alias that executes the **pdbrun** command and sends data to the **neon** and **conic** programs.)

See also: **conic**, **ribbonjr**, **pdbrun**

2.4.51. Noeshow

Usage: **noeshow** [*options*] *constraint_file* [*torsion_file*]

The **noeshow** command allows the display of NMR-derived constraints on a model structure in MIDAS. Full details of the use of this command can be found in Appendix 6 of this document.

2.4.52. Objdisplay

Usage: **objdisplay** *atom_specification*

Usage: **˜objdisplay** *atom_specification*

Usage: **objdisplay** *model_number ...* **|** *model_range*

Usage: **˜objdisplay** *model_number ...* **|** *model_range*

Objdisplay displays one or more entire non-molecule graphics objects (see **open** for a description of graphics objects).

If the argument to **objdisplay** is an atom specification, then the object(s) with model numbers corresponding to the specified atoms will be displayed.

Otherwise the argument(s) to **objdisplay** should be one or more model numbers or ranges (of the form *#–#*), separated by spaces. Note that the *model_number* should be an integer without a preceding **#** symbol.

Examples:

objdisplay 1	displays the object associated with model 1
objdisplay 1 5–8	displays the objects associated with models 1 and 5 through 8
˜**objdisplay** /sheet	undisplay all objects associated with models with beta sheets

See also: **display**, **open**

2.4.53. Open

Usage: **open** [original] [model | surface | ms | object] [*model_number*] *filename* [*surface_filename*]

Usage: [˜**open** | **close**] [model | surface | ms | object] *model_number*

Open causes the contents of the file *filename* to be read and shown as model *model_number* (if *model_number* is omitted, the lowest available model number is used). The model number is used to uniquely reference the model in subsequent commands and therefore should be remembered.

The optional **original** keyword is used to prevent MIDAS from transforming the coordinates of the model or object that is being opened (**original** has no effect on surfaces). Normally, if the display option *refmodel*'s value is **set** to a model number, newly opened objects and models are transformed by the same rotations and translations in effect for that model number (see **set/unset**). The use of the **original** keyword prevents the application of these transformations to the newly opened model. This keyword is normally only of interest to implementors of MidasPlus delegates.

How MIDAS interprets *filename* is controlled by the optional keyword specifier. If no keyword is given, or the keyword **model** is specified, MIDAS will try to open the *filename* as a PDB file. Note that MIDAS only recognizes PDB version 1 records. If your file contains PDB version 2 records, MIDAS will note them as unknown records in the reply area and otherwise ignore them.

The following keywords can be shortened to as few letters as necessary to disambiguate them from other keywords. Note that the **original** keyword above cannot be shortened.

The **ms** or **surface** keywords indicate that *filename* is the output of the **ms** or **dms** utility, used to generate solvent-accessible surfaces, and that the surface should be associated with the indicated *model_number* (*model_number* cannot be omitted in this case). Optionally, a surface file, *surface_filename*, may be specified as a fourth argument (in which case the **ms** or **surface** keyword would be omitted). The opened surface is associated with the open model. An opened surface file is not initially displayed, since it may be quite large. See the **surface** command for displaying all or part of an open surface.

MIDAS can read **ms** and PDB files that have been compressed using the UNIX ''compress'' command. Compressed **ms** and PDB files use substantially less disk space than regular files. There is no need to first uncompress the files or to specify any special keyword on the **open** command line.

If **open** is used with the **object** keyword, the file is assumed to specify a non-molecule graphic object, and can be opened as a new model, or associated with an existing model. Each line in the object file is a command or text. If it is text, then it is displayed in the current color and font at the current position. All of the commands start with a period and are as follows:

.comment *text*	comments
.c *text*	comments
.font *name size*	set font and size
.color *color_designation*	set color (see below)
.cmov *x y* [*z*]	set character position
.dot *x y* [*z*]	show dot at position
.marker *x y* [*z*]	show marker at position
.m *x y* [*z*]	move to location
.move *x y* [*z*]	move to location
.d *x y* [*z*]	draw line from last location
.draw *x y* [*z*]	draw line from last location

The *color_designation* parameter to the **.color** directive can be simply a single MIDAS color, including user-defined colors (see *colordef*), or two colors and a mixture fraction, separated by commas. The mixture fraction is a number between zero and one that indicates the relative amounts of the two colors to use to get the actual color. Zero indicates use exclusively the first color, while one indicates use only the second color.

If the **autocolor** option has been set (using the command **set autocolor**), then each newly opened model without color definitions will be given a unique color so that different models can be easily distinguished.

To close a model, use **˜open** followed by the model number.

If a model is closed and another model opened with the same model number, then none of the transformations applied to the previous model are applied to the newly opened model. Note that this is in direct contrast to some previous versions of MIDAS where all the transformations were applied, in order to facilitate docking. If docking is necessary, one model's transformation matrix can be applied to another model with the **matrixget**/**matrixset** commands.

Filename may be a pathname to a file or the name of a file in the current working directory. To change directories, use the **cd** command.

See also: **cd**, **makems**, **match**, **write**, **set**, **surface**

2.4.54. Pdb2site

Usage: pdbrun **pdb2site** –o *site_file*

Pdb2site is a simple way to generate a site file for use in conjunction with the *dms*(1) command for computing molecular surfaces. Details of **pdb2site**'s usage can be found in Appendix 6 of this document.

2.4.55. Pdbopen

Usage: **pdbopen** [*options*]

The **pdbopen** command launches a pair of windows for browsing a PDB distribution hierarchy for PDB files of interest. Naturally, this is only useful if a PDB distribution is available on your system. Full details of the use of this command can be found in Appendix 6 of this document.

2.4.56. Pdbrun

Usage: **pdbrun** [all] [conect] [nouser | noobj | surface] [mark=*name* ...] *command* [*cmd_args...*]

Pdbrun causes *command* and *cmd_args* to be passed to the user's preferred UNIX shell for execution. A set of concatenated PDB files describing the current models (hereinafter referred to as a PDBRUN file) is also passed as standard input to *command*. Normal shell metacharacters (notably output redirection) can be used. Errors are ignored and any shell output is interpreted as MIDAS commands and executed. For instance,

```
    pdbrun cat > file
```

would result in a PDBRUN file describing the current models to be saved to a file named *file* (see also the **save** and **write** commands). MIDAS has no way of directly restoring state from a saved PDBRUN file (but see the *run2ses*(1) command in Appendix 6).

Note that MIDAS first searches the directory */usr/local/midas/lib/midas* for *command* before examining the rest of the user's execution path. This is useful to avoid name conflicts between MIDAS commands using the **run**/**pdbrun**/**delegate** mechanism and UNIX commands of the same name.

The **all** option specifies that all atoms, not just those that are displayed, should be sent to the given *command*.

The **conect** [*sic*] option specifies that PDB-standard `CONECT` records should be generated for all residues, even if they have standard connectivity.

MIDAS places `USER` records in the PDBRUN file to indicate display information such as atom colors, view direction, clipping planes, *etc*. The **nouser** option specifies that no `USER` records should appear in the file. The **noobj** keyword causes just `GFX`-type `USER` records to be omitted from the file. The **surface** keyword causes `USER GFX VERTEX` records to be placed in the file for displayed VDW or solvent-accessible surfaces.

The **mark**=*name* option specifies that only those atoms referenced by the named mark should be sent to *command*. Note that unless the **all** option is given as well, only those marked atoms that are also displayed, labeled, vdw'ed, or surfaced will be sent. Specifying more than one **mark**=*name* option results in the union of the given marks being sent.

Any output from *command* is interpreted as MIDAS commands and executed. Control returns to MIDAS when the **pdbrun** command terminates. The **pdbrun** command facilitates extensions to the normal MIDAS command set. In particular, both the **conic** and the **ribbonjr** commands are implemented using **pdbrun.**

As mentioned above, a PDBRUN file is passed as input to *command*. This PDBRUN file describes not only current model coordinates, but also other aspects of the modeling environment, such as clipping plane positioning, *etc*. Modeling environment annotations are supplied in various types of `USER` records, described in detail in Appendix 1. The PDBRUN file has three principal sections.

The first section (*Annotation Headers*) has global modeling information. The second section (*Per Molecule Annotations*) has molecular data for each model. Each model's data starts with a `USER FILE` record and is terminated with an `END` record. The serial numbers in `ATOM`, `HETATM`, and `TER` records increment continuously from one model to the next. Due to field widths, this imposes a limit of 99999 atoms in a scene. Any PDB records present in the model's original source file (and not normally interpreted by MIDAS, *e.g.* `REMARK`s) are preserved and placed just after the `USER FILE` record. The data for non-molecular graphics objects are also in this section, bounded by `USER OBJECT` and `USER ENDOBJ` records. The third section (*Annotation Trailers*) contains records specifying the angles and distances being explicitly monitored in the current view. Throughout, fields in PDB records that are not wide enough to hold their associated values are filled with asterisks instead.

Details of the meaning and format of the `USER` records employed by **pdbrun** can be found in Appendix 1.

A discussion of the motivation for and design criteria of the PDBRUN format can be found in:

G. S. Couch, E. F. Pettersen, C. C. Huang and T. E. Ferrin ‘‘Annotating PDB files with scene information’’ *J. Mol. Graphics* **13**, 153-158 (1995).

See also: **delegate**, **run**, **save**, **system**, **write**, **vdwopt**

2.4.57. Pickatom/Pickabort

Usage: **pickatom**

Usage: **pickabort**

These commands are used in conjunction with the MIDAS delegate facility (see Appendix 5). A delegate program can request that the user select a particular atom on the display by first sending a **pickatom** command to MIDAS, verifying that the command has been accepted by MIDAS, then waiting for the user to select an atom and for MIDAS to send the indication of the selected atom back to the delegate program.

When MIDAS accepts a **pickatom** command from a delegate, it will reply with

```
Waiting for pick
```

If another delegate program is already waiting for an atom to be selected, MIDAS will reply with

```
Already picking for delegate "XXX"
```

After MIDAS accepts a **pickatom** command and the user selects an atom, the command that MIDAS sends to the delegate is of the form

pickatom #*model_number*:*residue_sequence@atom_name*

If, instead of selecting an atom, the user types **pickabort**, MIDAS will send the command

```
pickabort
```

to the requesting delegate.

The commands and replies described above are primarily for use by delegate application programmers. The only important point that a user needs to remember is that typing **pickabort** will terminate a delegate picking request without actually selecting an atom. The **pickatom** command is only valid when sent by a delegate and has no effect when typed at the keyboard.

See also: **delegate**, **picking**

2.4.58. Push/Pop

Usage: **push**

Usage: **pop**

Push saves the current orientation of all open models on an image stack.[6] **Pop** retrieves the last ‘‘pushed’’ orientation. Any number of model orientations may be saved on the stack, memory allowing, and retrieved on a last-in first-out basis.

See also: **align**, **reset**, **savepos**, **window**

[6]For those unfamiliar with the concept of a ‘‘stack,’’ think of a pile of pictures which is created by ‘‘pushing’’ pictures one at a time onto the *top* of the pile and in which pictures are retrieved one at a time by taking the *top* picture off the pile.

2.4.59. Rainbow

Usage: **rainbow** [*model_number* ...]

The **rainbow** command colors model chains from end to end continuously from red to blue. This can sometimes be of assistance in visually following chain connectivity in a complicated structure. Full details of the use of this command can be found in Appendix 6 of this document.

2.4.60. Read

Usage: **read** *filename* [*filename*...]

Read executes the contents of the named file(s) as a list of commands. **Read** differs from **source** in that **source** updates the display after each command while **read** only updates the display after all commands are done.

See also: **record**, **source**, **run**, **pdbrun**

2.4.61. Record

Usage: **record** *filename*

Record saves all the commands remembered by the **set record** command in the file *filename*. Saved commands can later be re-executed with a **read** or **source** command, although the command file should first be edited since the **record** command itself will be in the file. **Record** is very useful for creating demonstration scripts for later playback. Note that there is normally a **set record** in the system startup file distributed with MIDAS so that, by default, all typed user input is remembered. Thus all commands used since MIDAS startup can be saved easily with **record**. This is useful for reproducing results or applying commands used on one set of models to a different set of models.

See also: **read**, **set**, **source**

2.4.62. Redraw

Usage: **Redraw** *atom_specification*

This command no longer does anything. It is provided for backwards compatibility with old MIDAS command scripts.

2.4.63. Reset

Usage: **reset** [list | *view_name*]

Reset returns models back to saved orientations. In each MIDAS session, the original orientation of the *first* model(s) opened is saved as *view_name* ‘‘default.’’ This orientation may be retrieved by the command **reset** or the command **reset default**. Other orientations may be saved using the **savepos** command:

savepos *view_name*

and retrieved using the **reset** command:

reset *view_name*

For a list of existing *view_name*s, give the command:

reset list

See also: **align**, **push/pop**, **savepos**, **window**

2.4.64. Reverse

Usage: **reverse** *rotation_number*

Reverse reverses the direction of a bond rotation. If the direction is reversed, then the portion of the molecule which rotated previously (as delineated in the **rotation** command description) remains fixed and *vice versa* in subsequent rotations. Also, the rotation angle is reset to zero degrees.

Warning: In assigning multiple bond rotations to a model, two rotations cannot affect a common set of atoms unless one affected set is a complete subset of the other. That is, rotations must be properly nested. In executing **reverse**, reversing the bond rotation may cause such a conflict and be disallowed. Reversing bond rotations must be done in an appropriate order such that all intermediate combinations are legal (see **rotation**).

See also: **brotation**, **rotation**, **fixreverse**

2.4.65. Ribbonjr

Usage: **ribbonjr** *options*

The **ribbonjr** command produces an aesthetic representation of the secondary structure of the displayed molecule(s), in a manner reminiscent of a ‘‘Jane Richardson drawing.’’ The current model orientation and coloring is retained. There are many options, which are detailed fully in the **ribbonjr** manual page included in Appendix 6 of this document. In addition to ribbon depictions of protein structures, **ribbonjr** can display ribbon representations of nucleotide structures, optionally using stylized representations of nucleotide sugars and bases (with the **–P** flag).

Ribbonjr has three principal display modes (selected with the **–f** flag): **screen**, **inventor**, and **midas**.

In **screen** mode (the default), **ribbonjr** displays a static ribbon drawing in a separate window. Clicking the left mouse button in the window will dismiss it.

In **inventor** mode (SGI only), **ribbonjr** will invoke the *ivview*(1) program (part of the IRIS Inventor program suite) to display a ribbon representation of the molecule. This representation can be rotated by the user in real time. This window can be dismissed by selecting ‘‘Quit’’ from the ‘‘File’’ pulldown menu at the upper left of the ivview window. Note that IRIS Inventor must be installed on your system for this mode to be useful (if you are uncertain if Inventor is installed, check to see if /usr/sbin/ivview exists).

In **midas** mode, a wireframe representation is generated and then displayed in MIDAS itself as a graphics object in the lowest available model number. The representation can be removed with the ‘‘`~open object` *model_num*’’ command. Consult ‘‘Non-Molecule Graphics Objects’’ in Part III of this manual for further information on MIDAS graphics objects.

If a protein is displayed, then in order for the **ribbonjr** command to show the correct secondary structure, the model must have been opened from a PDB file that contained correct `HELIX` and `SHEET` records. If such records were not present, it is possible to have MIDAS compute the secondary structure information with the **ksdssp** command. **Ribbonjr** needs no additional information to correctly display a nucleotide structure.

By default, **ribbonjr** shows all displayed atoms, not just the ribbon. The **–a** option will limit display to the ribbon only. If it is desirable to show a few key sidechains, but not all sidechains, display can be restricted to backbone atoms in MIDAS with the command ‘‘`chain mainchain`’’ and then the desired sidechains can be shown with the **display** command before running **ribbonjr**.

When **ribbonjr** is invoked from MIDAS, MIDAS actually executes a front-end program that in turn executes the actual *ribbonjr* program. This makes it easier to use the **inventor** and **midas** formats of *ribbonjr* since the front-end captures the *ribbonjr* output in a temporary file and displays the file in *ivview* or MIDAS itself, as appropriate. If it is necessary to invoke *ribbonjr* directly (for instance if you want to capture the output into a permanent file), then one must circumvent the front-end and invoke the *ribbonjr* program itself. This can be done with the command ‘‘`pdbrun surface /usr/local/midas/bin/ribbonjr`.’’

See also: **conic**, **ksdssp**

2.4.66. Rlabel

Usage: **rlabel** *atom_specification*

Usage: **˜rlabel** *atom_specification*

Rlabel enables residue labeling of the first displayed atom of each residue in *atom_specification*. Such labeling is enabled by default in MIDAS. **˜rlabel** may be used to turn off display of residue labels.

See also: **label**

2.4.67. Rock

Usage: **rock** [*axis* [*angle* [*frames* [*wait_frames*]]]]

Usage: **˜rock** [*axis*]

Rock will rotate the selected structures back and forth about the ***x***, ***y*** or ***z*** *axis*. The *angle* indicates the number of degrees the structure rotates at the fastest point in the sinusoidal period (an *angle* value of 9 corresponds approximately to a 90 degree arc).

The cycle time for **rock** is approximately 2.4 seconds. This corresponds to 18 frames forward and 18 frames back (15 frames per second).

If arguments are omitted, defaults will be used. These defaults are: 0 *wait_frames*, infinite *frames*, *angle* of 3, and y *axis*. MIDAS continues to process commands while the requested motion is in progress. To have command processing pause until the motion is finished, use the **wait** command.

If **rock** does not work, it is likely you have failed to **select** the target models.

See also: **freeze**, **select**, **wait**

2.4.68. Roll

Usage: **roll** [*axis* [*angle* [*frames* [*wait_frames*]]]]

Usage: ˜**roll** [*axis*]

Roll will rotate the selected structures about the ***x***, ***y***, or ***z*** *axis* continuously. If *axis* is an integer, it refers to the corresponding bond rotation.

If *angle* is specified, the structure is rolled through that angle (given in degrees) for each frame. If *angle* has a negative value the direction of the rotation is reversed. *Frames*, if specified, indicates the number of image update frames over which the **roll** operation is carried out. If *frames* is not specified, the structure continues to roll until explicitly turned off with the command ˜**roll** *axis*. Note that two or three **roll**s may be active at the same time; that is, the user can roll the structure around two or three axes at the same time. Each must be turned off explicitly.

Wait_frames, if specified, indicates the number of frames to wait before beginning the roll. This is useful for making videos. For example,

roll x 2 90 30

rolls the model 180 degrees over 90 image updates (*i.e.* 2 degrees on each frame update) after waiting 30 image update cycles before beginning. MIDAS continues to process commands while the requested motion is in progress. To have command processing pause until the motion is finished, use the **wait** command.

The default values for *axis* and *angle*, if omitted, are *y* and *3*, respectively.

If **roll** does not work, it is likely you have failed to **select** the target models.

See also: **freeze**, **select**, **turn**, **wait**

2.4.69. Rotation

Usage: **rotation** [*rotation_number*] *atom1 atom2* [*atom3 atom4*]

Usage: ˜**rotation** *rotation_number*

Rotation activates a bond rotation. All rotations are assigned a *rotation_number* between 0 and 15 inclusive, either by the user or automatically by MIDAS. Once a bond rotation is activated, the rotation may be manipulated and controlled by assigning the rotation to a slider via the **assign** command. The MIDAS system startup command file pre-assigns bond rotations 0 through 4 to sliders (they can be de-assigned if necessary). Alternatively, the **turn** command can be used to set the angle of the bond rotation to an exact value.

The rotation number and current bond angle are reported on the top of the display screen. If only *atom1* and *atom2* are specified, the angle reported is relative to the beginning position, *i.e.* the position when the assignment was made. If four atoms are specified, then the torsional angle is reported. The ˜**rotation** command removes the rotation and returns the bond to its original conformation (use the **fix** command to preserve the new conformation).

When a bond is rotated, part of the model remains fixed and the remainder rotates. For a rotation in a side chain, the ‘‘backbone side’’ will remain fixed and the side chain will rotate. For a rotation in the backbone, the beginning residues of the structure will remain fixed and the terminal residues will rotate. The rotation, looking along the bond from the fixed to the rotating section, is clockwise. If the **brotation** command (which see) is used instead of **rotation**, then the rotating and fixed portions as described above are interchanged, and the rotation (again along the bond from the fixed to the rotating section) is *counter*-clockwise.

In assigning multiple bond rotations to a model, two rotations cannot affect a common set of atoms unless one affected set is a complete subset of the other. That is, rotations must be properly nested.

Examples:

rotation 1 #1:1@c8,c9	assign rotation 1 to the bond between c8 and c9 in the first residue of model 1
rotation 3 #1:2@C:3@N	assign rotation 3 to the bond between atom C in residue 2 and atom N in residue 3

See also: **brotation**, **reverse**, **assign**, **fix**, **fixreverse**, **roll**, **turn**

2.4.70. Run

Usage: **run** *cmd* [*cmd_args*...]

Run passes its arguments *cmd* and *cmd_args* to the shell for execution and takes the output from these as a series of MIDAS commands.

The user may interrupt the execution of the MIDAS commands by pressing the **ESC** key. Execution of the current command is completed before processing the interrupt.

Note that MIDAS first searches the directory */usr/local/midas/lib/midas* for *cmd* before examining the rest of the user's execution path. This is useful to avoid name conflicts between MIDAS commands using the **run**/**pdbrun**/**delegate** mechanism and UNIX commands of the same name.

See also: **pdbrun**, **system**

2.4.71. Save

Usage: **save** *session_name*

Save stores the model orientations, bond rotations, distance calculations, slider assignments and user options for the current session in a MIDAS session file. The model coordinates are saved in PDB format. Since the model orientations are stored in the session file and not the data files, PDB files produced with **save** do not have their coordinates transformed. To get transformed coordinates, use the **write** or **pdbrun** command.

To restart a saved session, from the UNIX command line use the command:

midas *session_name*

Also, the **load** command can be used to restore a saved session into MIDAS (replacing the ongoing modeling session).

Note that if bond rotations have been made and the user wishes to have the saved PDB files reflect the new conformation, the **fix** command must be invoked before the session is saved.

It should also be noted that **save** only saves the state of MIDAS itself and not the state of associated MIDAS delegates. Delegates active when the session is saved will not be active when the session is restored.

See also: **fix**, **load**, **pdbrun**, **stop**, **write**

2.4.72. Savepos

Usage: **savepos** [*view_name*]

Usage: ˜**savepos** [*view_name*]

Savepos saves the current model orientations and associates *view_name* with it. If *view_name* is missing, the name ‘‘default’’ is used. The view may be retrieved using the **reset** command.

A view may be ‘‘forgotten’’ using the command ˜**savepos** *view_name*. This saves space in session files.

For a list of all existing *view_name*s, give the command:

savepos list

See also: **align**, **reset**, **push/pop**, **window**

2.4.73. Scale

Usage: **scale** *factor* [*frames* [*wait_frames*]]

Usage: ˜**scale**

Scale increases the size of the displayed view by the specified scaling *factor*. The scaling factor must be positive — a scaling factor less than one will decrease the size of the displayed view. Note that this command does not modify the model coordinates.

Frames scales the models in the specified manner for the specified number of image update frames. *Wait_frames*, if specified, indicates the number of frames to wait before beginning the scaling. ˜**scale** will halt an ongoing scale. *Frames* and *wait_frames* default to 1 and 0, respectively. These parameters are useful for controlling the rate of scaling and are helpful when constructing MIDAS command scripts and making videos. MIDAS continues to process commands while the requested motion is in progress. To have command processing pause until the motion is finished, use the **wait** command.

See also: **wait**

2.4.74. Section

Usage: **section** *units* [*frames* [*wait_frames*]]

Usage: ˜**section**

Section moves the hither and yon clipping planes the specified number of angstrom *units*. This has the effect of displaying a different cross section of the displayed model(s). A positive number of units displays a cross section closer to the user, whereas a negative number displays a cross section farther away from the user.

Frames moves the clipping planes in the specified manner for the specified number of image update frames. *Wait_frames*, if specified, indicates the number of frames to wait before beginning the clip. ˜**section** will halt an ongoing section. *Frames* and *wait_frames* default to 1 and 0, respectively. These parameters are useful for controlling the rate of clipping and are helpful when constructing MIDAS command scripts and making videos. MIDAS continues to process commands while the requested motion is in progress. To have command processing pause until the motion is finished, use the **wait** command.

See also: **thickness**, **clip**, **wait**

2.4.75. Select

Usage: **select** *atom_specification*

Usage: **select** *model_number ...* **|** *model_range*

Usage: **select** all

For each Usage *above:*

Usage: **~select** ...

Select selects a model or models for subsequent **move**, **rock**, **roll** and **turn** commands as well as interactive mouse manipulations.

If the argument to **select** is an atom specification, then the model(s) containing those atoms will be selected.

If **select** is used with the keyword ‘‘all,’’ then all models will be selected. **~select all** will then revert to the previous selection state. This feature is convenient for switching back and forth between moving models relative to one another and global motion of all models.

Otherwise the argument(s) to **select** should be one or more model numbers or ranges (of the form *#-#*), separated by spaces. Note that *model_number* cannot include a **#** symbol.

The selection status of models can also be controlled via the control panel. See ‘‘Manipulating the Model’’ in Part I of this manual for further details.

Examples:

select 1	selects model 1
select 1 5–8	selects models 1 and 5 through 8

See also: **assign**

2.4.76. Set/Unset

Usage: **set** *keyword* [*value*]

Usage: **unset** *keyword*

There are two types of display options in MIDAS, those that act as off/on toggles and those that vary over a range of values. For the toggle type of option, **set** with the appropriate *keyword* enables the option and **~set** *keyword* or **unset** *keyword* disables the option. For value type options, **set** *keyword value* sets the value while **set** *keyword* displays the current value.

Although initially all toggle options are disabled, MIDAS sets certain options on at the beginning of each session by reading the initialization file */usr/local/midas/resource/midas/midas.rc*. See Appendix 4 of this document for a list of the display options enabled during initialization.

See also: **devopt**

The available MIDAS display options are:

(see next page)

Set/Unset Toggle Options

Keyword	Function
autocolor	Gives each newly opened model a different color.
cofg	Puts a ‘+’ at the center of rotation for the selected models. The default center of rotation is at the center of mass of the selected models. The center of rotation can be changed with the **cofr** command, and the center of mass of a model can be set with the **setcom** command. If the *independent* toggle option is on, a ‘+’ will be displayed at the center of rotation of each individual model.
control	Displays the control panel.
filenames	Makes MIDAS display the filenames of the open models in the top left corner of the window above any bond rotation monitors.
fullscreen	Makes MIDAS resize itself to use the full screen. Useful if MIDAS is started without a desired **–f** option. If then unset, MIDAS will revert to its original size.
halfbond	Atoms are colored by halfbond connections to other atoms instead of each atom having one whole bond associated with it. Note that using halfbond mode may degrade response time.
independent	If set, models rotate about their own centers of mass, otherwise models rotate about the combined center of mass.
labels	Turns on distance, rotation, and angle monitoring labels.
ortho	Uses orthographic instead of perspective projection.
reassign	If set, multiple assignments to a single slider (see **assign**) will cause reassignment, *i.e.* only the last assignment will apply. If unset, then multiple assignments will accumulate, *i.e.* all will apply.
record	Initiates remembering of subsequent typed user commands. Note that commands that implicitly generate additional commands (*e.g.* **read**, **source**, **run**, **pdbrun**) are remembered but not expanded. Issuing a new **set record** when record mode is already set resets remembering from scratch. **˜set record** clears the command memory and prevents subsequent commands from being remembered. This will save some time and disk space since the commands are remembered in a temporary disk file. To save the remembered commands to a permanent file, use the **record** command.
showsphere	Controls whether a circle defining the transition between ***x,y*** versus ***z*** rotation is shown when MidasPlus is in ‘‘virtual trackball’’ manipulation mode.
text	Displays the `Command` and `Reply` text lines on the bottom of the display screen. This option can be turned off using **unset text** when taking photographs.
verbose	MIDAS prints confirmation messages after each successful command. If verbose is **unset** these messages will not appear.

Set/Unset Value Options

Keyword	Function
bg_color	Sets the MIDAS background color. *Value* can be any color name as described in detail under **colordef**.
eyesep	The separation between the centers of the viewer's eyes, in inches. This information is necessary to compute the projections for stereo viewing. It is rarely necessary to change the default setting for most adults, but it might be necessary for children viewing stereo.
font	Allows atom labels to be in any font style. The *value* is a font name, *e.g.*, **Helvetica**. The default font on the SGI is **raster**. A list of acceptable font names can be found in /usr/local/midas/resource/midas/xpsfontmap (the first column).
fontsize	Allows atom labels to be any point size. The default font size is 12. The default font on the SGI is **raster**, which is not scaleable, so setting a font size for it has no effect.
linewidth	Controls the thickness with which bonds are drawn. The default is 1, and larger values produce thicker lines (and slower interaction). On some systems, if anti-aliasing is turned on, lines cannot be wider than 1. **devopt smooth off** will turn off anti-aliasing.
molpath	A colon-separated list of directories that will be searched for PDB files by the **open** command. Tildes (˜) can be used to refer to user's home directories, à la *csh*(1). If the PDB file is not found in any of the molpath directories (or the current directory), then MIDAS will attempt to be ''smart'' and treat the molpath directories as PDB distribution-style directories and look for the file again. This only occurs if the given file name is 4 characters long (*i.e.* it seems to be a standard 4-character PDB identification code). If so, then the molpath directories will be searched for a subdirectory with a 2-character name the same as the middle 2 characters of the PDB ID code. If such a subdirectory exists, then it will be searched for a file by the name of ''pdb*idcode*.ent.'' For example, if attempting to open ''1gcn,'' the molpath directories will be searched for ''gc/pdb1gcn.ent.''
nameplate	Controls the placement of the MidasPlus logo on the bottom of the screen. A *value* of 0.0 puts it to the extreme left; 1.0 puts it to the right.

Set/Unset Value Options (continued)

Keyword	Function
refmodel	Specify the model number to use as a reference coordinate system for newly opened models. For example, if **refmodel** is set to **0**, then the coordinates of a newly opened model will be subjected to the same scaling, rotation and translation that model 0 is currently undergoing. The models can be individually manipulated thereafter. If **refmodel** is unset, then newly opened models will not be subjected to any transformations. If **refmodel** is set to –1, then the lowest model number in use will be used as the basis for transforming newly opened models.
viewdist	The distance from the viewer to the screen, in inches. This information is needed to correctly compute stereo projections. Its default setting is appropriate for most modeling work, but the value may have to be increased if a demonstration is being given where many people are further from the screen than normal.
vpsep	Amount of vertical separation between left and right eye images in stereo mode, in scan lines. This option should only have to be set once for each machine with stereo. It controls the vertical convergence of the left and right images when the stereo system is turned on. Once the correct *value* is determined (empirically), it should be put in /usr/local/midas/resource/midas/midas.rc. The easiest way to determine the correct value is to enter stereo mode *without putting on the stereo glasses*. There should be two overlapped images on the screen, displaced slightly horizontally. If the value of **vpsep** is correct, the images should be aligned vertically. If they are not aligned, adjust the value of **vpsep** until they are (you can type ‘‘**set vpsep**’’ to determine the current vpsep value).
walleye_scale	Scales the size of walleye-type stereo image pairs (see **stereo** command). The default size is correct when using opticomechanical stereo viewers, while a larger scale is useful for taking pictures for publications.

2.4.77. Setcom

Usage: **setcom** *model x_coord y_coord z_coord* [*radius* [*natoms*]]

Usage: **˜setcom** *model*

When MIDAS is asked to rotate one or more models, it needs to know the center of mass of the model(s). **Setcom** is used to change the center of mass parameters in the rare case where the ones automatically computed by MIDAS are unacceptable.

X_coord, *y_coord*, and *z_coord* specify the new center of mass for *model*. If *radius* is given, it should be the shortest radius (in angstroms) from the new center of mass that encloses the model. This helps MIDAS do a better job of framing the models in the graphics window. Specifying *natoms* essentially tells MIDAS how much weight this model has when computing a combined center of mass for multiple models.

˜setcom will cause MIDAS to revert to the default center of mass that it normally computes.

See also: **cofr**

2.4.78. Show

Usage: **show** *atom_specification*

Usage: **˜show** *atom_specification*

Show displays the specified atoms and removes all others in those models from the display. Models not involved in *atom_specification* will be unaffected. Note that this differs from the **display** command in that the **show** command displays *only* those atoms specified (and will undisplay all others). **˜Show** removes the specified atoms completely from the display.

See also: **chain**, **display**

2.4.79. Sleep

Usage: **sleep** *number_of_seconds*

Sleep causes MIDAS to pause for *number_of_seconds* seconds and then resume operation. This command is useful in command scripts where a break in the action is required.

See also: **wait**

2.4.80. Source

Usage: **source** *filename* [*filename*...]

Source reads one or more command files of MIDAS commands. **Source** differs from **read** in that **source** will display the results of each command as it is executed while **read** only updates the display after all commands have completed.

The user may interrupt the execution of the *source* file by pressing the **ESC** key. MIDAS completes execution of the current command before processing the interrupt.

See also: **read**, **run**, **pdbrun**

2.4.81. Speed

Usage: **speed** *value*

Speed changes the speed of functions activated by the sliders or ‘‘spaceball.’’ Thus, the sensitivity of the devices are altered for scaling and rotation functions. *Value* is a positive or negative integer which reflects the *relative* change in speed. The absolute range is 2 to 14 with default value of 10.

Note that in the OpenGL version of MIDAS, holding down the Shift key will halve the rate of speed of all motions controlled by the sideview control panel as well as translations (but not rotations) of models in the main window.

2.4.82. Stereo

Usage: **stereo** off **|** sequential **|** walleye **|** crosseye **|** lefteye **|** righteye

Stereo specifies how images are displayed. **Off** indicates a monocular image. **Sequential** indicates a time-sequential stereographic image of the type that is utilized by ‘‘Crystal Eyes’’ stereo systems. Note that **sequential** requires special hardware to work properly and that not all workstations may support this hardware (only SGI and DEC Alpha workstations support hardware stereo). **Walleye** indicates side-by-side stereo pairs having positive horizontal parallax. That is, the left-eye image is shown on the left-hand side of the display window and the right-eye image is shown on the right-hand side of the window. This is the classic method of generating stereo image pairs. The sizes of the images are constrained to avoid overlap and can be changed with the *walleye_scale* variable (see the **set** command). **Crosseye** indicates side-by-side stereo pairs having negative horizontal parallax. That is, the left-eye image is shown on the right-hand side of the display window and the right-eye image is shown on the left-hand side of the window. The user must look ‘‘crosseyed’’ at the images to perceive the stereopsis effect. Image size may fill the entire window in this mode. **Lefteye** and **righteye** show the left and right view of a stereo image, respectively. This is useful for taking stereo slides since you can photograph each eye view individually.

If stereo convergence in **sequential** mode is poor, try using the **set** command to adjust values for *vpsep*, *viewdist*, and *eyesep* (especially *vpsep*).

The command **˜stereo** is equivalent to **stereo off**.

See also: **set**

2.4.83. Stereoimg

Usage: **stereoimg** –p *renderer* [*options*]

The **stereoimg** command generates stereo pairs using MIDAS-callable rendering programs (*e.g. ribbonjr*, *neon*, *etc.*). Full details of the use of this command can be found in Appendix 6 of this document.

2.4.84. Stop

Usage: **stop**

Stop terminates the current MIDAS session without saving any of the currently displayed models.

See also: **save**

2.4.85. Surface

Usage: **surface** *atom_specification*

Usage: **˜surface** *atom_specification*

Surface selectively displays solvent-accessible surface points for models. The syntax is the same as for the **display** command, except that it applies to surface points rather than to bonds.

To prepare a solvent-accessible surface for display see ''Solvent-Accessible Surfaces'' in Part III of this manual. The **open** command is used to open prepared MIDAS solvent-accessible surface files followed by the **surface** command to display the surface.

Alternatively, the **vdw** command may be used to display the model's van der Waals surface. Display of this surface requires no advance calculation and is more convenient unless the solvent-accessible surface is specifically desired.

Examples:

surface #0	display the surface for model 0
surface #1:5	display the surface for model 1, residue 5
˜surface #1:5,32,64	undisplay surface for model 1, residues 5, 32 and 64

See also: **makems**, **vdw**

2.4.86. Swapaa

Usage: **swapaa** *new_residue_type* [,preserve] *residue*

Swapaa replaces *residue* with a *new_residue_type* residue. Both the old and new residue must be standard amino acids. *New_residue_type* is a three-letter residue name; *residue* is a single residue specified by number (*e.g.* `:21`) or (unique) residue name (*e.g.* `:LYS`). The side chain of the new residue will be in standard conformation unless the **preserve** keyword is given, in which case as much of the existing conformation as possible is saved. Preserving conformation can cause rings to become nonplanar (*e.g.* swapping in a PHE for a LYS while preserving conformation).

The temperature factor for the new residue is set to the highest currently found in the model.

2.4.87. Swapna

Usage: **swapna** *new_residue_type* [,preserve] *residue*

Swapna replaces *residue* with a *new_residue_type* residue. Both the old and new residue must be standard nucleotides: A, T, G, C, or U. *New_residue_type* is a one-letter nucleotide code; *residue* is a single residue indicated by (unique) name (*e.g.* `:G`) or number (*e.g.* `:24`).

The temperature factor for the new residue is set to the highest currently found in the model.

The keyword **preserve** is recognized, although it currently does not affect **swapna** behavior in any way.

2.4.88. System

Usage: **system** *command*

System executes the UNIX *command* under the user's preferred shell. *Command* may not be an interactive program. The output from *command* will appear in the MIDAS reply area. In the IRIS GL version of MIDAS, the reply area isn't scrollable, so it may be desirable to show long replies in the shell window instead of the reply area. To do so, the user should give the command:

!*command*

(note ''!'' mark) instead of ''**system** *command*.''

As an alternative to the **system** command, the **midaspush** command may be used to access other screen windows or icons (*i.e.* go off and do something else for awhile and then return to MIDAS). If MIDAS was started in its own window, the standard window manipulation menu may also be use for this purpose. Lastly, one could also use the **save** command to retain orientations, rotation and slider assignments, *etc.* and then quit MIDAS entirely and return later for another modeling session.

See also: **run**, **pdbrun**, **midaspush**, **save**

2.4.89. Thickness

Usage: **thickness** [*units* [*frames* [*wait_frames*]]]

Usage: **˜thickness**

The **thickness** command changes the distance between the hither and yon clipping planes by the specified number of angstrom *units*. A positive *unit* value increases the distance between the clipping planes, whereas a negative value decreases it. This results in displaying an increased or decreased cross section of the current model(s).

Frames moves the clipping planes in the specified manner for the specified number of image update frames. *Wait_frames*, if specified, indicates the number of frames to wait before beginning the move. **˜thickness** will halt an ongoing thickness operation. *Frames* and *wait_frames* default to 1 and 0, respectively. These parameters are useful for controlling the rate of clipping and are helpful when constructing MIDAS command scripts and making videos. MIDAS continues to process commands while the requested motion is in progress. To have command processing pause until the motion is finished, use the **wait** command.

Thickness without the *units* parameter will report the distance in angstroms between the clipping planes.

See also: **section**, **clip**, **wait**

2.4.90. Turn

Usage: **turn** [*axis* [*angle* [*frames* [*wait_frames*]]]]

Usage: **turn** [*axis*]

Turn functions the same as **roll** except that the default values for the *angle* and number of *frames* are 3 degrees and 1 frame, respectively. Thus, the command **turn** *y* will generate an approximate left-eye stereo image, although **stereo lefteye** and **stereo righteye** are preferable since these commands generate a more technically accurate stereo image pair.

See also: **roll**, **stereo**, **wait**

2.4.91. Update

Usage: **Update** transformed **|** original *filename*

Update changes the coordinates of a subset of atoms in a model. The user must supply either the **transformed** or **original** keywords and a PDB filename containing atom records with new atom coordinates. If PDB `USER FILE` record(s) are present in the file (see Appendix 1), then the indicated models are updated, otherwise the lowest numbered model is updated. If PDB `HELIX` or `SHEET` records are present in the file, they will take precedence over such records originally supplied with the model(s).

If the keyword is **transformed**, then the new coordinates are considered as having already been transformed by the current rotation and translation matrices. If the keyword is **original**, then the new coordinates are treated as untransformed coordinates and the current rotation and translation matrices are applied to the new coordinates before they are integrated with the rest of the model.

2.4.92. Vdw

Usage: **vdw** *atom_specification*

Usage: **˜vdw** *atom_specification*

Vdw displays the van der Waals surface for the selected atoms. The syntax is the same as for the **display** command, except that it applies to surface points instead of bonds.

The **vdw** command may be interrupted by the ESC key. Atoms whose vdw surfaces were already computed when the interrupt occurred will have their surfaces displayed. Since computing the vdw surface is much faster than displaying it, pressing the ESC key might not help.

Note that the default vdw radii used by MIDAS assume that no explicit hydrogens are present in the model(s). This behavior can be changed with the **vdwopt** command.

Details of the **vdw** algorithm can be found in the Bash *et al.* reference given in full on page 3 of this manual.

See also: **makems**, **surface**, **vdwopt**

2.4.93. Vdwopt

Usage: **vdwopt** [density *value*] [radii *file* | hydrogen | default] [extend *len*] [define *atom_type radius*]

Vdwopt sets user options for displaying van der Waals surfaces. The options are as follows:

density *value*	The user may change the dot density of the displayed surface relative to the initial value of 1, which corresponds to 5 dots per square angstrom. The density of dots varies linearly with the *value* provided. Thus, a density *value* of 2 gives 10 dots per square angstrom.
radii *file*	Indicates a file containing alternate van der Waals radii. The alternate file must contain a complete set of radii for all atoms in the model. The file contains a series of records consisting of an atom name (character string) followed by a space and the atom radius in angstroms (floating point number). This is the same format as used by the **ms** program (see Appendix 6). Optionally, a third field can specify the residue type. The VDW radii data files that MIDAS uses for models with and without explicit hydrogens are, respectively, /usr/local/midas/resource/midas/vdw.hydrogen and /usr/local/midas/resource/midas/vdw.default. Use the **default** or **hydrogen** options (below) to return to default values.
hydrogen	Indicates that hydrogen atoms are included in the model and van der Waals radii should not compensate for missing hydrogens. The result is smaller atom radii and distinct hydrogen atoms.
default	Indicates that van der Waals radii should compensate for missing hydrogen atoms. **Vdwopt default** undoes the effect of **vdwopt hydrogen**.
extend *len*	Increases all van der Waals radii by the constant *len* angstroms. Multiple **extend**s are not additive – instead, the last **extend** has precedence.
define *atom_type radius*	Indicates that all atoms with an atomic symbol of *atom_type* should be assigned a van der Waals radius of *radius* angstroms.

See also: **vdw**

2.4.94. Version

Usage: **version**

Version reports information about which version of MIDAS is currently being executed and is displayed in the reply area of the MIDAS window. It is useful to supply this information when reporting MIDAS bugs, so that it is possible to determine if the problem has already been solved in a more recent version of the program.

2.4.95. Wait

Usage: **wait** [*wait_frames*]

Wait suspends command processing for the given number of *wait_frames* or until all model movement has ceased. In the latter case, if a command such as **roll** has been activated for a given number of frames, that motion is completed before the next command is executed.

The **wait** may be interrupted by pressing the **ESC** key. This breaks out of **wait** but does not freeze the screen (*i.e.* any active motion continues until completion).

See also: **sleep**

2.4.96. Watch/Watchopt

Usage: **watch** *atom_specification*

Usage: **˜watch**

Usage: **watchopt** [distance *distance*]

Watch monitors interatomic distances. By default, it checks for distances less than the sum of the van der Waals radii of two atoms. Atoms in the *atom_specification* are checked against all other atoms to whom their distance can potentially vary (*i.e.* to atoms on other models or atoms on the same model across from active bond rotations) Close contacts are displayed as yellow dotted lines, in the same manner as distance monitors. One can specify a fixed distance by using the **distance** option to the **watchopt** command, where *distance* is in angstroms. A *distance* of zero means to use the default vdw radii for comparison. Only distances that potentially vary are checked, *i.e.*, atoms that are fixed relative to each other are not checked.

˜watch will terminate **watch** monitoring.

See also: **distance**

2.4.97. Window

Usage: **window** [*atom_specification*]

With no arguments, **window** puts all displayed models on the screen by changing the scale and position of the view. The orientation of the models is not changed. If the image has drifted off the screen, this is an excellent way of making it visible again.

If given an *atom_specification* as an argument, **window** will recalculate the view to enclose just those atoms instead of all of the models.

See also: **align**, **center**, **push/pop**, **reset**, **savepos**

2.4.98. Write

Usage: **write** [surface] [relative *n*] *model_number* [*filename*] [relative *n*]

This command causes the specified *model* to be written out as a PDB file. Current bond rotations must be **fix**ed before the **write** if they are to be reflected in the written file.

The **relative** option specifies that the atomic coordinates written out are relative to the *untransformed* atomic coordinates of model *n*; otherwise, the coordinates are written as currently transformed. This option may be specified in either of two places in the command line (see usage above).

If a *filename* is not given, then the file that the model was opened from will be overwritten.

If the **surface** option is given, then the model's surface is written instead. The surface is written in the format used by the *dms* program (see Appendix 6). Note that the **relative** option is not supported for surfaces.

See also: **fix**, **pdbrun**, **save**

[This page intentionally left blank]

Part III: Advanced Concepts

This section describes some additional features of MIDAS and some ancillary programs which may be used in conjunction with MIDAS for specific modeling problems. Beginning users may wish to skip this section on first reading. Included are descriptions of:

- Connectivity and templates
- Calculation and display of a molecular surface
- Calculation of electrostatic potential for a surface
- Details of Protein Data Bank format
- Saving/printing screen images
- Photographing screen images
- Displaying non-molecular objects
- Using Midas in conjunction with a web browser
- Performance issues with Midas

Note: The descriptions include sample command lines. In these examples the model name used is *dfr*, an acronym for dihydrofolate reductase, a system which has been extensively modeled. The user's own file names should be substituted for ‘‘dfr’’ when running commands.

3.1. Connectivity and Templates

Many macromolecules in nature, such as proteins and DNA, are long chains of component molecules (amino acids and nucleic acids). MIDAS uses the same approach in building molecular models. Small component molecules are chained together to build complex models. In MIDAS, these building blocks are called *residues*.

Each residue is made up of *atoms* which are connected in a specific pattern. Each atom in the residue has a unique name, usually combining a key letter, such as C for carbon, N for nitrogen, *etc.* and additional letters to uniquely identify it. For example, atoms may be named CA, C1, C2, N1, or O2. Thus, a residue contains a fixed number of atoms with specific names.

The pattern of bonding between the atoms is termed *connectivity*. The connectivity within a residue must form a connected graph of uniquely named atoms. Since the atoms are uniquely named, the bonding can be defined in an unambiguous manner. The definition of a MIDAS residue includes both connectivity and atom-naming information. If you use a PDB file with non-unique atom names in some residues, MIDAS will still display it; however, you will not be able to address atoms uniquely.

When residues are used to build models, the connectivity of atoms is the same for each occurrence of any given residue in the model. For example, if we build a residue named ‘‘gly,’’ then each time a ‘‘gly’’ residue appears in the model protein, it will have the same basic pattern of connectivity of atoms and the same atom names.

MIDAS uses files called *templates* to name each residue's atoms and the connectivity of those atoms. Each template consists of a map which describes how the atoms are connected and which atoms link the residue to other residues. If a template doesn't exist for a particular residue, then MIDAS will use atomic radii and distances to connect atoms within the residue.

In most cases, you can safely forget about the existence of template files, partially because MidasPlus usually gets the connectivity right, and because you can fix the connectivity with MidasPlus commands once the model is displayed (see the **bond** command). Templates are discussed in more detail in ‘‘More on Connectivity and Templates’’ (section 3.5).

3.2. Molecular Surfaces

MIDAS represents molecules by drawing three-dimensional ‘‘wire frame’’ stick figures on the screen. Users often want to characterize the surfaces of these molecules as well. This can be accomplished in several ways within MIDAS: the two basic types are space-filling ‘‘solid’’ models and surfaces represented by a densely spaced dot surface. Surfaces represented by dots offer greater interactivity, since they can be manipulated in real time in the same way as wire frame models. Space-filling models, on the other hand, offer shading and shadows and produce photographic quality images; these models are generated using the **conic** command (see Appendix 6 for full details). Note that although MIDAS renders dots surfaces that are manipulable interactively, it is possible to generate more aesthetic, but non-interactive, dot surfaces with the **neon** or **ribbonjr** commands (also detailed in Appendix 6).

There are two different types of dot surfaces available:

(1) A *vdw* or van der Waals surface uses the van der Waals radii of the atoms in the model. MIDAS assigns a van der Waals radius to each atom according to its atom type as determined by the Protein Data Bank naming conventions (*e.g.* if the first letter encountered in atom name is C, then the atom is carbon, and so forth for N (nitrogen), O (oxygen), *etc.*). MIDAS creates a surface around each atom, clipping off areas of overlap to create a Corey-Pauling-Koltun (CPK) type model. The radii and density of dots used can be changed with the **vdwopt** command in MIDAS.

(2) A *solvent-accessible* surface may be calculated for any molecule. This surface is defined by rolling a theoretical water ‘‘probe’’ around the van der Waals surface of the molecule and using the contact and reentrant points to determine the surface. This type of surface is smooth, free of ‘‘seams’’ between atoms, since the surface is determined by a probe water molecule.

These two types of molecular surface representation differ significantly in their speed of generation and utility. Vdw surfaces are generated quickly and do not break or ‘‘tear’’ with bond rotations. Solvent-accessible surfaces take approximately 100 times longer to compute, and the surface tears with bond rotations.

3.2.1. VDW Surfaces

The **vdw** command in MIDAS will display the van der Waals surface for all atoms of a MIDAS model or a subset of atoms as described in the ‘‘Command Reference Guide’’ section of this document. The user selects those atoms, residues, and models for which the surface is displayed.

Often the user does not want the surfaces of all atoms both interior and exterior displayed, but rather only the surfaces of those atoms which comprise the surface of the molecule or the surface of an active site. It can be a tedious process in MIDAS to select each of these surface atoms individually or even residue by residue. The zone specifiers in atom specifications (section 2.1.4) provide a convenient way to select specific sites of interest.

Example of Site Selection

Suppose the user wants to display the surfaces of only those atoms within a certain distance of a particular set of atoms or coordinate positions, as in determining an active site of a molecule. If the receptor’s PDB file were opened as model zero and the ligand’s PDB file were opened as model one, the following command

```
vdw #1 za<8
```

will show the vdw surface of only those atoms within 8 angstroms of any ligand atom (coordinate value). If the user wishes to change the test radius from 8 angstroms to 12 angstroms, for example, the command line is:

```
vdw #1 za<12
```

See section 2.1 of this document for additional details on how to specify atoms and zones around atoms.

3.2.2. Solvent-Accessible Surfaces

There are two approaches to creating solvent-accessible surfaces for use with MIDAS:

The first, easier approach is to use the *makems* delegate while in MIDAS itself to interactively specify what atoms or models to surface. *Makems* will then invoke the **dms** program appropriately to compute the solvent-accessible surface, and display the resultant surface in MIDAS. While simple, this approach has some limitations that may make it inappropriate if:

- A large surface is to be computed — which may take several minutes or hours.
- **Dms** must be run with non-default settings (*e.g.* dot density, probe radius, *etc.*)

- **Dms** non-surface output is of interest (*e.g.* total surface area information)
- The generated surface file must be retained for later use (*makems* automatically deletes it).

The second approach involves invoking **dms** from the UNIX command line with the appropriate PDB and site-specification files, and taking the resultant surface, **open**ing it and its associated PDB file in MIDAS, and then displaying the surface with the **surface** command.

The procedure for running *makems* is covered in detail in the *makems* manual page, in Appendix 6 of this manual. The procedure for running **dms** from the UNIX command line is discussed below.

3.2.2.1. Running Dms Directly

(1) Begin with a PDB file which has been displayed and is known to be correct.

(2) The surface may be calculated for the entire model or specific sections. To select specific sections, a file must be prepared containing the residues and/or atoms of interest. Either the file may be prepared by hand (if the atoms/residues of interest are already known) or the *pdb2site* utility, run from within MIDAS, can be used to generate the file. Both approaches are described below.

Preparing a Site File by Hand

The format for the site file is a series of lines containing the residue name, residue number and atom specification separated by spaces. Chain identifiers (if present in the PDB file) should be appended to the residue number. Residue insertion codes (if any) should be placed between the residue number and any chain identifier. Residues in `HETATM` records should have a ‘‘*’’ appended to them. An atom specification can be an atom name, ‘‘*’’ (any atom in the named residue), or ‘‘FRM’’ followed on the next line by ‘‘TO’’ (used in pairs to denote residue ranges). Note that atom names should correspond exactly to the name given to the same atom by MIDAS.

As an example, a file appearing as:

```
ASN   15  FRM
LYS   21  TO
HIS  123  *
GLU  141  CB
GLU  141  CG
```

selects residues 15 through 21, inclusive, residue 123 (the ‘‘*’’ selects all the atoms of the residue), and atoms ‘CB’ and ‘CG’ of residue 141. The surface is calculated only for these specified atoms and residues.

Using pdb2site

Usually the selected portion of the model is an active site. The user may determine which atoms are of interest by one of two methods:

(2a) display the model and choose the atoms visually.

(2b) display the model and choose the atoms within a given radius of a ligand (use the zone specifiers in the atom specification syntax).

Once you have displayed only those atoms you want surfaced, you could then use the *pdb2site* utility to generate a site file, via the MIDAS command ‘‘`pdbrun pdb2site -o` *site*.’’

(3) Use the **dms** program to generate the surface. An example command for creating a surface is:

dms *dfr* **–a** **–d** 0.5 **–g** *logfile* **–i** *site* **–o** *dfr.dms*

where:

dfr	is the name of the PDB file.
–a	indicates that all atoms are to be included in the calculation. If this flag is missing only amino acids are included in the calculation.
–d 0.5	indicates the density of points on the surface.
–g *logfile*	directs informative messages from the program to the file *logfile*.
–i *site*	directs **dms** to calculate the surface for the entire molecule but report the surface for only those atoms and residues selected in the file *site*.
–o *dfr.dms*	directs the output from the program (*i.e.* the surface) to the file *dfr.dms.*

At UCSF, the **dms** program should always be run in the background using the **submit** command. This prevents long-running jobs from adversely affecting interactive users. Thus, the command line for the above example becomes:

submit dms *dfr* **–a** **–d** 0.5 **–g** *logfile* **–i** *site* **–o** *dfr.dms*

(4) The progress of the **dms** program may be monitored in two ways:

(4a) Use the **ps** command to see if the program is running.

(4b) Read the logfile (*i.e.* the file name given with the **–g** flag) and the *submit.out* file created by **submit**. If the program is left to run overnight, it is a good idea to check these two files before leaving the laboratory to make sure the program didn't run into immediate difficulty and stop.

The completed **dms** output file can be quite large, so be sure adequate disk space is available.

(5) To display the surface, invoke the **midas** display program. Open the PDB file first using the **open** command as described in the ''Command Reference Guide'' section of this document. Then using the same model number, open the surface file. For example:

```
open 1 dfr
open surface 1 dfr.dms
```

opens PDB file *dfr* as model 1 and associates the surface *dfr.dms* with it. This may be shortened to:

```
open 1 dfr dfr.dms
```

To display the surface, use the **surface** command:

```
surface #1
```

3.2.3. Electrostatic Potential Molecular Surfaces

Users who wish to display electrostatic potential molecular surfaces should run the **dms** program as described above with an additional flag, **–n**. This flag generates surface normals in the **dms** output file. The **dms** output file may be used to generate the electrostatic potential of the surface using the program **esp**.

Esp requires a **dms** file and a PDB file as input. Assuming the **dms** file name is *dfr.dms* and the PDB file name is *dfr*, the command to calculate electrostatic potential is:

esp **–i** *dfr.dms* **–o** *dfr.esp* **–a** *dfr*

The program generates an annotated **dms** surface file, which in this example is named *dfr.esp*.

The atomic charges for the various residue types are held in the system file /usr/local/midas/resource/charges.esp. The user may substitute his or her own file of charges if desired. Use the system file as a format guide to generate the appropriate file. Include this file in the command line:

esp **–i** *dfr.dms* **–o** *dfr.esp* **–q** *chrgs* **–a** *dfr* where *chrgs* is the name of the charges file. The **−q** flag and charges file must precede the name of the PDB file in the command line. Note that the electrostatic surface potential computed by **esp** depends crucially on the charge data in the charges file, so users should verify the data given in this file is correct for their particular application.

The **esp** program has several other options which are described in the **esp** manual page; see Appendix 6 of this document.

3.3. Protein Data Bank Format

MidasPlus uses the Protein Data Bank (PDB) format as a standard for all coordinate text files. The Protein Data Bank at Brookhaven National Laboratory (BNL) is a clearing house for macromolecular coordinate data, and distribution CD-ROMs are written in a standardized format specified by BNL. A complete and concise description of the format is given in the document ‘‘Protein Data Bank Atomic Coordinate Entry Format Description,’’ which is published and distributed by Brookhaven National Laboratory. A newsletter is also published on a periodic basis by BNL. MidasPlus supports version 1 of the PDB format. Version 2 records and fields are ignored by MidasPlus programs, but this should not impact modeling with MIDAS since records/fields added in version 2 do not contain any information needed by MIDAS. In this section, a shortened description of the PDB format is presented that is sufficient for use in creating MIDAS input.

The complete PDB file structure contains a wealth of information including source, journal citations, and identification of substructures such as disulfide bonds, helices, beta sheets, and active sites. Since the entire Protein Data Bank entry contains much more information on a macromolecule than is needed for model building, knowledge of a subset of the format is sufficient for MIDAS users. Users should bear in mind, however, that adhering strictly to the format specifications is key to successful model building. The modeling programs are unforgiving about incorrect input formats, and much time and frustration can be saved by diligence in data preparation.

3.3.1. Description

Protein Data Bank format is a character-oriented format which consists of lines of information in a file. One file generally contains enough information to characterize a single molecule or model. Each line of information in the file is called a *record*. There are usually several different types of records present in the same file, such as ATOM records, which contain coordinate values for atoms, SSBOND records, which contain disulfide linkage information, and TER records which signal the end of a chain of residues. These records are arranged in a specific sequence to characterize the molecule.

Protein Data Bank Record Types
Recognized by MIDAS

Record Type	Data Provided by Record
ATOM	atomic coordinate record containing the x,y,z orthogonal angstrom coordinates for the given atom
HETATM	atomic coordinate record containing x,y,z coordinates for atoms of nonstandard residues. These record types are used by BNL to distinguish standard residues, such a amino acids and nucleic acids, from nonstandard groups, such as inhibitors, substrates, and saccharides. The only functional difference with ATOM records is that HETATM residues are by default not connected to other residues. Note that water residues should be in HETATM records.
SSBOND	defines disulfide bond linkages between amino acid residues. Notice that the templates as described thus far allow only for linkage to the next residue in the chain and to the previous residue in the chain. SSBOND records are special-case links which are handled separately in MIDAS, and bond rotations about these links are not allowed.
TER	indicates the end of a chain of residues. For example, a hemoglobin molecule consists of four subunit chains which are not connected. TER indicates the end of a chain and prevents a connection (line) to the next chain. This record type is also used to prevent connection of substrates to other displayed parts of the model.
CONECT	gives explicit connectivity of atoms.
HELIX	indicates the location and type (right-handed alpha, *etc.*) of helices. One record per helix.
SHEET	indicates the location, sense (anti-parallel, *etc.*) and registration with respect to the previous strand in the sheet (if any) of each strand in the model. One record per strand.

The following table describes the format for each record type. The *record type* appears in columns 1 to 6 of each line of a PDB file. This *record type* determines the kind and format of information on the remainder of that line. Note that the data appears in specific *columns*. This refers to the spaces on the line in which the data appears. For example, in an ATOM record, the first four places contain the record type, ‘‘ATOM.’’ The next two places are blank. The 7th through 11th places contain the atom serial number. The serial number is right-justified, so if the serial number is ‘‘1,’’ for example, the digit 1 will appear in the 11th place and the other places will be blank. For the atom with serial number ‘‘100’’ the number will appear in places 9 through 11, leaving places 7 and 8 blank. It is necessary to reproduce this format *exactly* in order for MIDAS to interpret the data properly. Any deviation is likely to cause errors preventing the successful construction of a model.

Note that HELIX and SHEET records are only required if you intend to use the midas **ribbonjr** command to render a ‘‘Jane Richardson’’ type ribbon depiction of the model. The **ribbonjr** command uses information contained in these records to determine what parts of the backbone to render as helices and sheets. The MIDAS **ksdssp** command can be used to generate HELIX and SHEET records if they are absent.

Protein Data Bank Format

Record Type	Columns	Data	Justification	Data Type
ATOM	1-4	‘‘ATOM’’	left	character
	7-11	Atom serial number	right	integer
	13-16	Atom name	left*	character
	17	Alternate location indicator		character
	18-20	Residue name	right	character
	22	Chain identifier		character
	23-26	Residue sequence number	right	integer
	27	Code for insertions of residues		character
	31-38	X orthogonal Å coordinate	right	floating
	39-46	Y orthogonal Å coordinate	right	floating
	47-54	Z orthogonal Å coordinate	right	floating
	55-60	Occupancy	right	floating
	61-66	Temperature factor	right	floating
TER	1-3	‘‘TER’’		character
	7-11	Serial number (optional)	right	integer
	18-20	Residue name (optional)	right	character
	22	Chain identifier (optional)		character
	23-26	Residue sequence number (optional)	right	integer
	27	Code for insertions of residues (optional)		character
HETATM	1-6	‘‘HETATM’’		
	7-66	same as ATOM records		
SSBOND	1-6	‘‘SSBOND’’		character
	8-10	Sequence number (optional)	right	integer
	12-14	Residue name (CYS)	right	character
	16	Chain identifier		character
	18-21	Residue sequence number	right	integer
	26-28	Residue name (CYS)	right	character
	30	Chain identifier		character
	32-35	Residue sequence number	right	integer
CONECT	1-6	‘‘CONECT’’		
	7-11	Atom serial number	right	integer
	12-16	Atom serial number (covalent bond)	right	integer
	17-21	Atom serial number (covalent bond)	right	integer
	22-26	Atom serial number (covalent bond)	right	integer
	27-31	Atom serial number (covalent bond)	right	integer
	32-36	Atom serial number (hydrogen bond)	right	integer
	37-41	Atom serial number (hydrogen bond)	right	integer
	42-46	Atom serial number (salt bridge)	right	integer
	47-51	Atom serial number (hydrogen bond)	right	integer
	52-56	Atom serial number (hydrogen bond)	right	integer
	57-61	Atom serial number (salt bridge)	right	integer

*Atoms whose chemical symbols (as distinct from remoteness indicator) are one letter long are left-justified in columns 14-16. Those which are two characters long (*e.g.* zinc symbol ‘‘ZN’’) are left-justified starting in column 13. Refer to the Brookhaven document ‘‘Protein Data Bank File Record Formats’’ for details.

Protein Data Bank Format
(*continued*)

Record Type	Columns	Data	Justification	Data Type
HELIX	1-5	‘‘HELIX’’	left	character
	8-10	Helix serial number	right	integer
	12-14	Helix identifier	right	character
	16-18	Initial residue name	right	character
	20	Chain identifier		character
	22-25	Residue sequence number	right	integer
	26	Code for insertions of residues		character
	28-30	Terminal residue name	right	character
	32	Chain identifier		character
	34-37	Residue sequence number	right	integer
	38	Code for insertions of residues		character
	39-40	Type of helix[†]	right	integer
	41-70	Comment	left	character
SHEET	1-5	‘‘SHEET’’		character
	8-10	Strand number (in current sheet)	right	integer
	12-14	Sheet identifier	right	character
	15-16	Number of strands (in current sheet)	right	integer
	18-20	Initial residue name	right	character
	22	Chain identifier		character
	23-26	Residue sequence number	right	integer
	27	Code for insertions of residues		character
	29-31	Terminal residue name	right	character
	33	Chain identifier		character
	34-37	Residue sequence number	right	integer
	38	Code for insertions of residues		character
	39-40	Strand sense with respect to previous[‡]	right	integer
		The following fields identify two atoms, the first in the current strand and the second in the previous strand, which are hydrogen bonded to each other. These fields should be blank for strand 1.		
	42-45	Atom name (as per ATOM record)	left	character
	46-48	Residue name	right	character
	50	Chain identifier		character
	51-54	Residue sequence number	right	integer
	55	Code for insertions of residues		character
	57-60	Atom name (as per ATOM record)	left	character
	61-63	Residue name	right	character
	65	Chain identifier		character
	66-69	Residue sequence number	right	integer
	70	Code for insertions of residues		character

[†]Helix types are:

1	Right-handed alpha (default)	6	Left-handed alpha
2	Right-handed omega	7	Left-handed omega
3	Right-handed pi	8	Left-handed gamma
4	Right-handed gamma	9	2/7 ribbon/helix
5	Right-handed 3/10	10	Polyproline

[‡]Parallel is indicated with ‘‘1,’’ anti-parallel with ‘‘−1.’’ Strand 1 has sense indicator ‘‘0.’’

For those who are familiar with the FORTRAN programming language, the following format descriptions will be meaningful. For those users unfamiliar with FORTRAN, ignore this gibberish:

ATOM **HETATM**	Format (A6,I5,1X,A4,A1,A3,1X,A1,I4,A1,3X,3F8.3,2F6.2)
SSBOND	Format (A6,1X,I3,1X,A3,1X,A1,1X,I4,4X,A3,1X,A1,1X,I4)
CONECT	Format (A6,11I5)
HELIX	Format (A6,1X,I3,1X,A3,2(1X,A3,1X,A1,1X,I4,A1),I2,A30)
SHEET	Format (A6,1X,I3,1X,A3,I2,2(1X,A3,1X,A1,I4,A1),I2,2(1X,A4,A3,1X,A1,I4,A1))

3.3.2. Examples of PDB Format

Glucagon is a small protein of 29 amino acids in a single chain. The first residue is the amino-terminal amino acid, histidine, which is followed by a serine residue and then a glutamine. The beginning portion of the PDB file appears thus:

```
ATOM      1  N   HIS     1      49.668  24.248  10.436  1.00 25.00
ATOM      2  CA  HIS     1      50.197  25.578  10.784  1.00 16.00
ATOM      3  C   HIS     1      49.169  26.701  10.917  1.00 16.00
ATOM      4  O   HIS     1      48.241  26.524  11.749  1.00 16.00
ATOM      5  CB  HIS     1      51.312  26.048   9.843  1.00 16.00
ATOM      6  CG  HIS     1      50.958  26.068   8.340  1.00 16.00
ATOM      7  ND1 HIS     1      49.636  26.144   7.860  1.00 16.00
ATOM      8  CD2 HIS     1      51.797  26.043   7.286  1.00 16.00
ATOM      9  CE1 HIS     1      49.691  26.152   6.454  1.00 17.00
ATOM     10  NE2 HIS     1      51.046  26.090   6.098  1.00 17.00
ATOM     11  N   SER     2      49.788  27.850  10.784  1.00 16.00
ATOM     12  CA  SER     2      49.138  29.147  10.620  1.00 15.00
ATOM     13  C   SER     2      47.713  29.006  10.110  1.00 15.00
ATOM     14  O   SER     2      46.740  29.251  10.864  1.00 15.00
ATOM     15  CB  SER     2      49.875  29.930   9.569  1.00 16.00
ATOM     16  OG  SER     2      49.145  31.057   9.176  1.00 19.00
ATOM     17  N   GLN     3      47.620  28.367   8.973  1.00 15.00
ATOM     18  CA  GLN     3      46.287  28.193   8.308  1.00 14.00
ATOM     19  C   GLN     3      45.406  27.172   8.963  1.00 14.00
```

Notice that each line or *record* begins with the record type, ATOM. The atom serial number is the next item in each record. Although each atom in the file is given a unique serial number, this information is not required by MIDAS.

The atom name is the third item in the record. Notice that the first one or two characters of the atom name consists of the chemical symbol for the atom type. All the atom names beginning with ‘‘C’’ are carbon atoms; ‘‘N’’ indicates a nitrogen and ‘‘O’’ indicates oxygen. The next character is the remoteness indicator code, which is transliterated according to:

α	A
β	B
γ	G
δ	D
ε	E
ζ	Z
η	H

The last character of the atom name is a branch indicator, if required.

The next data field is the residue type. Notice that *each* record contains the residue type. In this example, the first residue in the chain is HIS (histidine) and the second residue is a SER (serine).

The next data field contains the residue sequence number. Notice that as the residue changes from histidine to serine, the residue number changes from ‘‘1’’ to ‘‘2.’’ Two like residues may be adjacent to one another, so the residue number is very important for distinguishing between them.

The next three data fields contain the X, Y, and Z coordinate values, respectively. The next data field is the occupancy and is not used by MIDAS. The final data item is an optional atom temperature factor.

The glucagon data file continues in this manner until the final residue is reached:

```
ATOM    239  N   THR    29       3.391  19.940  12.762  1.00 21.00
ATOM    240  CA  THR    29       2.014  19.761  13.283  1.00 21.00
ATOM    241  C   THR    29        .826  19.943  12.332  1.00 23.00
ATOM    242  O   THR    29        .932  19.600  11.133  1.00 30.00
ATOM    243  CB  THR    29       1.845  20.667  14.505  1.00 21.00
ATOM    244  OG1 THR    29       1.214  21.893  14.153  1.00 21.00
ATOM    245  CG2 THR    29       3.180  20.968  15.185  1.00 21.00
ATOM    246  OXT THR    29       -.317  20.109  12.824  1.00 25.00
TER     247      THR    29
```

Note that this residue includes the extra oxygen atom, OXT, on the terminal carboxyl. The ‘‘TER’’ record terminates the amino acid chain.

A more complicated protein, fetal hemoglobin, consists of two amino acid chains (alpha and gamma) and two heme groups. The first ten lines of the PDB file for this molecule appear as:

```
ATOM      1  N   VAL A   1       6.280  17.225   4.929  1.00  0.00
ATOM      2  CA  VAL A   1       6.948  18.508   4.671  1.00  0.00
ATOM      3  C   VAL A   1       8.436  18.338   4.977  1.00  0.00
ATOM      4  O   VAL A   1       8.813  17.657   5.941  1.00  0.00
ATOM      5  CB  VAL A   1       6.317  19.598   5.527  1.00  0.00
ATOM      6  CG1 VAL A   1       6.959  20.999   5.376  1.00  0.00
ATOM      7  CG2 VAL A   1       4.819  19.636   5.383  1.00  0.00
ATOM      8  N   LEU A   2       9.259  18.958   4.152  1.00  0.00
ATOM      9  CA  LEU A   2      10.715  18.872   4.330  1.00  0.00
ATOM     10  C   LEU A   2      11.156  20.058   5.187  1.00  0.00
```

This data file appears initially much the same as the file for glucagon with the exception that the fifth data field now contains the single-character chain indicator. In this case, the chain indicator is ‘‘A,’’ indicating the alpha chain of the hemoglobin molecule. This field was simply blank in the glucagon example. At the end of chain A, the heme group records appear:

```
ATOM   1058  N   ARG A 141      -6.576  12.834 -10.275  1.00  0.00
ATOM   1059  CA  ARG A 141      -8.044  12.831 -10.214  1.00  0.00
ATOM   1060  C   ARG A 141      -8.186  14.096  -9.365  1.00  0.00
ATOM   1061  O   ARG A 141      -7.591  15.139  -9.671  1.00  0.00
ATOM   1062  CB  ARG A 141      -8.579  11.531  -9.580  1.00  0.00
ATOM   1063  CG  ARG A 141      -8.386  11.441  -8.054  1.00  0.00
ATOM   1064  CD  ARG A 141      -8.727  10.045  -7.568  1.00  0.00
ATOM   1065  NE  ARG A 141      -9.095  10.056  -6.143  1.00  0.00
ATOM   1066  CZ  ARG A 141      -9.268   8.931  -5.414  1.00  0.00
ATOM   1067  NH1 ARG A 141      -8.602   8.795  -4.282  1.00  0.00
ATOM   1068  NH2 ARG A 141     -10.097   7.962  -5.830  1.00  0.00
ATOM   1069  OXT ARG A 141      -8.973  13.984  -8.310  1.00  0.00
TER    1070      ARG A 141
HETATM 1071 FE   HEM A   1       8.133   8.321 -15.014  1.00  0.00
HETATM 1072  CHA HEM A   1       8.863   8.752 -18.417  1.00  0.00
HETATM 1073  CHB HEM A   1      10.362  10.946 -14.389  1.00  0.00
HETATM 1074  CHC HEM A   1       8.482   7.374 -11.743  1.00  0.00
HETATM 1075  CHD HEM A   1       6.982   5.180 -15.773  1.00  0.00
HETATM 1076  N A HEM A   1       9.452   9.545 -16.178  1.00  0.00
```

The last residue in the alpha chain is an ‘‘ARG’’ (arginine). Again, the extra oxygen atom ‘‘OXT’’ appears in the terminal carboxyl. The ‘‘TER’’ record indicates the end of the peptide chain. It is important to have ‘‘TER’’ records at the end of peptide chains so that MIDAS does not attempt to connect the end of one chain to the start of another. The atom number, residue type, chain indicator and residue number in the ‘‘TER’’ record are optional.

In the example above, the ‘‘TER’’ record is correct and should be present, but MIDAS would have broken the molecule chain at that point even without a ‘‘TER’’ record because ‘‘HETATM’’ residues are not connected to other residues or to each other. The heme group is a single residue made up of ‘‘HETATM’’ records. Note that the residue numbering begins again with ‘‘1’’ as the new chain begins.

At the end of the heme group associated with the alpha chain, the gamma chain begins:

```
HETATM 1109  CAD HEM A   1       7.582   6.731 -20.480  1.00  0.00
HETATM 1110  CBD HEM A   1       8.992   6.848 -20.968  1.00  0.00
HETATM 1111  CGD HEM A   1       8.998   6.529 -22.465  1.00  0.00
HETATM 1112  O1D HEM A   1       9.693   5.683 -22.895  1.00  0.00
HETATM 1113  O2D HEM A   1       8.276   7.153 -23.229  1.00  0.00
ATOM   1114  C   ACE G   0       7.896 -18.462  -1.908  1.00  0.00
ATOM   1115  O   ACE G   0       7.246 -18.839   -.922  1.00  0.00
ATOM   1116  CH3 ACE G   0       9.415 -18.301  -1.832  1.00  0.00
ATOM   1117  N   GLY G   1       7.354 -18.174  -3.077  1.00  0.00
ATOM   1118  CA  GLY G   1       5.904 -18.282  -3.283  1.00  0.00
ATOM   1119  C   GLY G   1       7.139 -19.112  -2.930  1.00  0.00
ATOM   1120  O   GLY G   1       7.026 -20.248  -2.448  1.00  0.00
ATOM   1121  N   HIS G   2       8.300 -18.533  -3.176  1.00  0.00
ATOM   1122  CA  HIS G   2       9.565 -19.224  -2.889  1.00  0.00
```

Here the ‘‘TER’’ card is implicit in the start of a new chain. The new chain identifier is ‘‘G.’’ The file continues in the same pattern as before until the entire gamma chain and its associated heme group have been specified.

Remember that the spacing of the data fields is crucial. Refer to the table given previously to determine the precise columns in which data *must* appear. If a data field does not apply, it should be left blank. For example, a protein which consists of a single amino acid chain has no chain identifier and thus column 22 is blank.

From this example, it is apparent that Protein Data Bank format relies on the concept of *residues* much in the same way as MIDAS uses templates. The same rules apply for PDB residues and template residues. These can be summarized as:

(1) All atoms within a single residue must have unique names. For example, residue ‘‘VAL’’ may have only one atom named ‘‘CA.’’ Other residues may also have a ‘‘CA’’ atom but not more than one ‘‘CA’’ may appear in ‘‘VAL.’’

(2) Residue names are a maximum of three characters long and uniquely identify the residue type. Thus, all residues of a given name in a file will be the same type of residue and have the same structure. For example, residue ‘‘SER’’ has a certain connectivity specified in the template for ‘‘SER.’’ Each occurrence of a serine residue in the Protein Data Bank file will conform to this pattern of connectivity. If there is a mismatch between the atoms in a PDB file residue and that residue’s template, then atoms found in the PDB file and not the template will be connected using simple distance-based criteria, whereas atoms in the template but not the PDB file will simply be omitted.

3.3.3. Common Errors in PDB Format Files

If a data file fails to produce a model or produces an incorrect model in MIDAS, it is sometimes difficult to determine where in the hundreds of lines of data the mistake occurred. This section enumerates some of the most common errors found in PDB files.

3.3.4. Program-Generated PDB Files

Spurious Long Bonds

A couple of common errors in program-generated PDB files result in MIDAS displaying very long bonds between residues that should be disconnected. To address this problem, the *longbond* program has been developed. *Longbond* can be run from within MIDAS to break bonds of excessive length. Consult the *longbond* manual page in Appendix 6 for details.

If you are writing a program that generates PDB files and long bonds appear, you may be interested in the common errors that cause them so that your program can be corrected. One such error is the omission of `TER` cards at the end of molecule chains. According to the PDB standard, `TER` cards mark the end of molecule chains and MIDAS depends on them in order to determine when to connect adjacent residues. You should insert them in your PDB file if they are missing. Consult section 3.4.1 for details of `TER` card format. Alternatively, you could mark all chains with differing chain IDs. MIDAS will not connect adjacent residues if they have differing chain IDs.

A second common cause of long bonds is improper use of `ATOM` records when `HETATM` records should be used. `HETATM` records should be employed for isolated compounds that do not form chains, such as H_2O or hemoglobin. MIDAS then knows not to connect adjacent ‘‘residue’’ numbers. Many programs generate files that fail to employ `HETATM` records appropriately. Make sure that the first six columns of the `ATOM` record are changed to `HETATM` so that the remaining columns stay aligned correctly.

Incorrect Secondary Structure Display – Proteins

For some PDB files, **ribbonjr** will display the entire structure as random coil with no sheets or helices even though such structure motifs clearly exist. There are two common causes for this phenomenon. The first is that there are no `HELIX` or `SHEET` records in the PDB file. MIDAS and **ribbonjr** both depend on these records to demarcate regions of secondary structure. If they are missing, these records need to be generated and inserted into the PDB file. The MidasPlus utility *ksdssp* can be used to generate `HELIX` and `SHEET` records. *Ksdssp* is fully described in Appendix 6 of this document. Also, the MIDAS **ksdssp** command will invoke the *ksdssp* utility from within MIDAS and apply the generated secondary structure records to the displayed structures.

The second common cause of missing secondary structure is incorrectly aligned atom names in PDB records. As described in section 3.4.1 of this document, atom names are composed of an atomic symbol (such as ‘‘C’’), *right* justified in columns 13-14 of `ATOM` and `HETATM` records, and trailing identifying characters (such as ‘‘A’’) *left* justified in columns 15-16. Many programs simply left-justify the entire atom name starting in column 13. The difference can be seen clearly in a short segment of hemoglobin:

correct

```
HETATM  976 FE   HEM     1      12.763  34.157   9.102  1.00  0.00
HETATM  977  CHA HEM     1      16.124  33.461  10.405  1.00  0.00
HETATM  978  CHB HEM     1      11.350  32.580  12.046  1.00  0.00
HETATM  979  CHC HEM     1       9.326  34.709   7.887  1.00  0.00
HETATM  980  CHD HEM     1      14.138  35.379   6.119  1.00  0.00
```

incorrect

```
HETATM  976 FE   HEM     1      12.763  34.157   9.102  1.00  0.00
HETATM  977 CHA  HEM     1      16.124  33.461  10.405  1.00  0.00
HETATM  978 CHB  HEM     1      11.350  32.580  12.046  1.00  0.00
HETATM  979 CHC  HEM     1       9.326  34.709   7.887  1.00  0.00
HETATM  980 CHD  HEM     1      14.138  35.379   6.119  1.00  0.00
```

A MidasPlus utility, **fixatname**, easily corrects the above misalignment. It is fully documented in Appendix 6 of this manual.

Incorrect Secondary Structure Display – Nucleotides

There are several possible causes of problems when attempting to show nucleotide structures with **ribbonjr**, including variant atom name nomenclature, reversed order of residues (standard is 5’ to 3’), and separate sugar and phosphate residues (they should be combined). A MidasPlus utility, **dnacheck**, corrects all of these faults as well as a few other minor problems. **Dnacheck** is fully described in Appendix 6 of this document.

3.3.5. Hand-Edited PDB Files

Duplicate Atom Names

One possible editing mistake is the failure to uniquely name all atoms within a given residue. Notice in the following example that two atoms in residue VAL are named CA.

```
ATOM      1  N   VAL A   1       6.280  17.225   4.929  1.00  0.00
ATOM      2  CA  VAL A   1       6.948  18.508   4.671  1.00  0.00
ATOM      3  C   VAL A   1       8.436  18.338   4.977  1.00  0.00
ATOM      4  O   VAL A   1       8.813  17.657   5.941  1.00  0.00
ATOM      5  CA  VAL A   1       6.317  19.598   5.527  1.00  0.00
ATOM      6  CG1 VAL A   1       6.959  20.999   5.376  1.00  0.00
ATOM      7  CG2 VAL A   1       4.819  19.636   5.383  1.00  0.00
ATOM      8  N   LEU A   2       9.259  18.958   4.152  1.00  0.00
ATOM      9  CA  LEU A   2      10.715  18.872   4.330  1.00  0.00
ATOM     10  C   LEU A   2      11.156  20.058   5.187  1.00  0.00
```

This error often does not become apparent until the residue is labeled and the resulting model is found to be missing a ‘‘CB’’ atom. If the MidasPlus program is used to (implicitly) construct the model, then it will draw a bond to the closest atom and silently ignore the other set of coordinates. This is one of the reasons it is sometimes advisable to construct a template using **gentpl**, since this program is more rigorous in checking for input errors; see section 3.6 for more details.

Residues Out of Sequence

In the following example, notice that the second residue (SER) appearing in the file is erroneously numbered residue 5. MIDAS will accept this input without complaint. The resulting model will have residue 5 connected to residue 1 and residue 3. This is all well and good, but only if it is what was originally intended. If, however, residue number 5 was to appear between residue 4 and residue 6, then it should have appeared in that order in the PDB file. Thus, if one finds that residues are connected in an incorrect order, the ordering and not the numbering in the data file should be changed.

```
ATOM      1  C   HIS     1      49.169  26.701  10.917  1.00 16.00
ATOM      2  CA  HIS     1      50.197  25.578  10.784  1.00 16.00
ATOM      3  CB  HIS     1      51.312  26.048   9.843  1.00 16.00
ATOM      4  CD2 HIS     1      51.797  26.043   7.286  1.00 16.00
ATOM      5  CE1 HIS     1      49.691  26.152   6.454  1.00 17.00
ATOM      6  CG  HIS     1      50.958  26.068   8.340  1.00 16.00
ATOM      7  N   HIS     1      49.668  24.248  10.436  1.00 25.00
ATOM      8  ND1 HIS     1      49.636  26.144   7.860  1.00 16.00
ATOM      9  NE2 HIS     1      51.046  26.090   6.098  1.00 17.00
ATOM     10  O   HIS     1      48.241  26.524  11.749  1.00 16.00
ATOM     11  C   SER     5      47.713  29.006  10.110  1.00 15.00
ATOM     12  CA  SER     5      49.138  29.147  10.620  1.00 15.00
ATOM     13  CB  SER     5      49.875  29.930   9.569  1.00 16.00
ATOM     14  N   SER     5      49.788  27.850  10.784  1.00 16.00
ATOM     15  O   SER     5      46.740  29.251  10.864  1.00 15.00
ATOM     16  OG  SER     5      49.145  31.057   9.176  1.00 19.00
ATOM     17  C   GLN     3      45.406  27.172   8.963  1.00 14.00
ATOM     18  CA  GLN     3      46.287  28.193   8.308  1.00 14.00
```

Common Typos

Another common error often arises in the data entry process. Sometimes the letter ‘‘l’’ may be erroneously substituted for the number ‘‘1.’’ This error has different repercussions depending upon what data field the error occurs in. If the letter ‘‘l’’ appears in the residue number, MIDAS does not complain, but then the letter ‘‘l’’ (rather than the number ‘‘1’’) must be used in all subsequent references to that residue. This can be very confusing, especially when all the other residues in the model are numbered. If the letter ‘‘l’’ appears in place of a ‘‘1’’ in the coordinate values, MIDAS accepts the input and sets those coordinate values equal to 0.000. Thus, if the user finds some atoms grossly misplaced in the MIDAS model, the corresponding error in the PDB file may be the use of ‘‘l’’ instead of ‘‘1.’’ Such errors may be readily located if the text of the data file appears in uppercase, since a text editor may be invoked to search for all occurrences of the lowercase letter ‘‘l.’’

3.4. Modeling Hydrogen Atoms

3.4.1. Hydrogens in Protein Data Bank Files

Users who have coordinate values for hydrogen atoms may include those values in the PDB data file using the Brookhaven Protein Data Bank convention for hydrogen atoms. The conventions for hydrogen atoms in PDB files are as follows:

(1) Hydrogen atoms appear as ATOM records following the ATOM records of all other atoms of a particular residue.

(2) The name of each hydrogen atom is determined by the name of the atom to which it is connected:

The first space of the name (column 13) is an optional digit to be used if two or more hydrogens are attached to the same atom.

The second column, 14, is used for the chemical symbol, ‘‘H.’’

The next two columns contain the remoteness indicator (one or two characters) of the atom to which the hydrogen is attached.

For example,

```
ATOM      1  N   VAL     1       0.330  15.770  15.090  3.30  1.46
ATOM      2  CA  VAL     1       1.650  16.390  15.360  0.96  1.50
ATOM      3  C   VAL     1       2.670  15.670  16.230  3.18  1.43
ATOM      4  O   VAL     1       3.170  16.250  17.200 -0.72  1.48
ATOM      5  CB  VAL     1       1.760  17.680  16.180  1.77  1.47
ATOM      6  CG1 VAL     1       3.120  18.310  15.900  4.31  1.50
ATOM      7  CG2 VAL     1       0.630  18.680  15.930  2.42  1.51
ATOM      8  D   VAL     1      -0.250  15.310  14.410  4.96  1.46
ATOM      9  HA  VAL     1       2.200  16.520  14.420 -2.30  1.50
ATOM     10  HB  VAL     1       1.750  17.420  17.260 -2.08  1.62
ATOM     11 1HG1 VAL     1       3.210  18.510  14.820 -0.21  1.60
ATOM     12 2HG1 VAL     1       3.230  19.250  16.460 -0.92  1.57
ATOM     13 3HG1 VAL     1       3.910  17.600  16.200 -1.79  1.56
ATOM     14 1HG2 VAL     1      -0.270  18.120  15.640 -3.19  1.55
ATOM     15 2HG2 VAL     1       0.440  19.240  16.860 -0.54  1.56
ATOM     16 3HG2 VAL     1       0.940  19.370  15.130  2.66  1.57
```

Note in this example that:

- All hydrogens appear after the other atoms of the residue.
- Atom 9, ‘‘HA’’ is attached to atom 2, ‘‘CA.’’ The remoteness indicator ‘‘A’’ is the same for these atoms.
- There are three hydrogen atoms connected to ‘‘CG1.’’ These three all have the same remoteness indicator, but contain a distinguishing digit in column 13. Thus, each has a unique name.
- It is not necessary to use a digit as a prefix to the atom name when only one hydrogen is attached to a given atom.

3.5. More on Connectivity and Templates

MIDAS has an algorithm for constructing atom connectivity based on interatomic distances. With reasonable coordinate data, this algorithm generally produces the correct connectivity. If, however, it consistently produces incorrect connectivity for a particular residue, the user may override the algorithm with a template.

MIDAS uses files called *templates* to form the connectivity of atoms. Each template consists of a map which describes how the atoms are connected and which atoms link the residue to other residues. This includes the following information:

(1) the residue name,

(2) the starting atom of the residue (important for connecting this residue to the previous one),

(3) the ending atom of the residue (important for connecting this residue to the next residue), and

(4) connections between atoms in the residue.

Template file names must be the same as the corresponding residue name and have a *.ins* extension. It is the template file that is used to determine the connectivity and atom-naming information when reading in PDB files. The following example is a template instruction file for glycine:

```
RESIDUE       GLY
START  N
DRAW   CA
DRAW   C
DRAW   O
MOVE   C
DRAW   OXT
END    C
```

Notice that:

(1) The residue has a name, GLY, a starting atom, N, and an ending atom C. Thus, if GLY were found in a chain of amino acids, this residue would be connected to the previous residue in the chain via atom N and to the next residue in the chain via atom C.

(2) If you follow the DRAW and MOVE instructions on paper with a pencil (starting at N, draw a bond to CA, draw a bond from CA to C, draw a bond from C to O, lift up the pencil and move it back to C, draw a bond to OXT), you will find a pattern of connectivity for glycine such as can be found in any standard biochemistry text.

(3) Templates specify connectivity only, and do not give any information about bond angle or bond length. In MIDAS, connectivity is determined by the MOVE and DRAW instructions in the template files, not by the distance between atoms (unless no template exists). If the user provides coordinates for glycine in which the distance from the C to the O is 10 angstroms, MIDAS is constrained to draw the long bond because of the connectivity specification in the glycine template. It is the user who defines the templates and ultimately controls which atoms are connected.

(4) The observant reader may have noticed that the carboxyl acid group contains two oxygen atoms, OXT and O. If this GLY residue is the terminal amino acid in a peptide chain, then both oxygen atoms are appropriate to the model. If, however, the residue appears within a sequence of amino acids, then only one oxygen is appropriate and the bond to the atom named OXT is not drawn but instead a bond is drawn between the ‘‘END’’ atom (in this example a carbon) and the ‘‘START’’ atom of the next residue (typically a nitrogen in the case of amino acids). MIDAS decides which bonds not to draw in this case by ignoring atoms which appear in the template but for which the user provides no coordinates. Thus, only if the user provides a coordinate value for the atom OXT will MIDAS draw OXT. If OXT does not appear in the template, however, providing coordinates for OXT is *not* sufficient for adding the additional atom since the associated connectivity information must also be provided. In other words, an atom must be included in the template in order to appear in the MIDAS model.

(5) Note that none of the hydrogen atoms contained within a real glycine amino acid are included in the above template specification. Historically, hydrogen atom coordinates have not been included in protein and nucleic acid structures because of the limited resolution provided by x-ray crystallography techniques. For structures where hydrogen atom information is available, appropriately named atoms (*e.g.* HA, 1HG1...) can be added to the residue definitions and thus incorporated into the template file in the same manner as other atoms in the template.

Most commonly used templates have already been constructed and reside in a library accessible to MIDAS (/usr/local/midas/resource/midas/models). The naming conventions used in these templates are those specified by the Brookhaven Protein Data Bank and include amino acids, nucleic acids, and many prosthetic groups. Thus, for the most part the residues needed to build a molecular model are already available. For any exceptions, however, the user must either rely on the distance-based connectivity algorithm or construct his or her own set of MIDAS templates and use these alone or in combination with the existing MIDAS library of templates. A program exists to automatically construct new template files when needed; see **gentpl** in Appendix 6 for details.

3.5.1. Building and Modifying MIDAS Templates

Occasionally it is necessary to modify existing templates. For example, **gentpl** sometimes generates templates with incorrect START and END atoms. This results in the use of the wrong atoms to connect the current residue to the previous and next residues in the chain. To correct this problem, the user may edit the template directly. To do so, invoke the standard system text editor on the appropriate *.ins* file, which should be found in the user’s private ‘‘models’’ directory. The default private models directory is ${HOME}/models/. This can be overridden by setting the ‘‘MODELS’’ environment variable. If that directory doesn’t exist nor does the default models directory, then the template will be found in the current directory. Since *.ins* suffix files contain only simple character text, they can be edited directly. The name of the START atom and the END atom can be changed as needed; see previous section for additional information on the format of template files. For example, suppose **gentpl** generated the following template:

```
RESIDUE      FEA
START FE1
DRAW  O
DRAW  FE2
DRAW  NA
DRAW  NB
DRAW  NC
```

```
END     NC
```

The start atom, FE1, is correct but the connection to the following residue should be made via FE2 instead of INC. The file can be edited to read:

```
RESIDUE        FEA
START  FE1
DRAW   O
DRAW   FE2
DRAW   NA
DRAW   NB
DRAW   NC
END    FE2
```

Sometimes **gentpl** is unable to generate a template from the PDB file because the atoms in the file are too far apart to determine the correct connectivity. An error message appears indicating ‘‘unreachable atoms.’’ **gentpl** uses a standard list of atomic radii for determining connectivity. This list is located in /usr/local/midas/resource/connect.tpl. Atoms falling within the standard distances in the list are joined. Any atoms which are not joined to the main body of the residue are reported as ‘‘unreachable atoms.’’

One approach to solving the ‘‘unreachable atoms’’ problem is changing the standard radii to larger values so that atoms which failed to bond before can be appropriately connected. To do this one can:

(1) Make a copy of the atomic radii file in your own private directory using the command:

cp /usr/local/midas/resource/connect.tpl *connect*

(2) Edit *connect* to increase the atom radii so that the appropriate connections can be made.

(3) Use the **gentpl** program to create the new template. For example, if the residue for which we want to build a template is VLX and this residue occurs in the PDB file *glx.pdb*, the command used is:

gentpl –r *VLX* **–i** *glx.pdb* **–c** *connect*

where *connect* is the name of the atomic radii file just edited. **Gentpl** searches through the PDB file for the specified residue and uses the atomic radii in the *connect* file to generate the connectivity of atoms and the corresponding MIDAS template.

If this method fails to produce an accurate template, the user must resort to creating the template instruction file with a text editor. Refer to the previous section for a description and example of the connectivity instruction file. Look at some of the *.ins* files in the template library (/usr/local/midas/resource/midas/models) to use as examples. Do not worry about the order in which the DRAW and MOVE commands appear in your instruction file as long as they correctly specify the connectivity and there is a path from the START atom to all other atoms in the residue.

3.6. Capturing Screen Images

It is frequently desirable to use MidasPlus images in contexts other than an interactive modeling session, such as in publications, web pages, or presentations. This section discusses saving screen images to disk in various formats, some format conversions for saved files, and how to print saved files, as well as some tips for screen photography.

3.6.1. Saving Screen Images to Disk

The ability to save screen images to disk is important for several reasons:

- It is the first step in printing images.
- Saved images can be incorporated easily into web pages or otherwise electronically disseminated.
- Saved images can be annotated or otherwise enhanced in programs such as Adobe Photoshop or SGI Showcase.

On SGI workstations, images saved in SGI image format can be manipulated with standard IRIX image tools to achieve a variety of effects. If these tools were installed on your system, then ‘‘man imagetools’’ will describe them. If they were not installed (*i.e.* ‘‘man imagetools’’ responds with ‘‘No manual entry found for imagetools'') and you wish to install them, consult the next section (''Converting Image Formats'') for installation information. Also, the *ilabel*(1) program, documented in Appendix 6, can be used to add text labels to SGI image files.

In generating an image you intend to save, it is frequently desirable to obtain the highest quality image possible, at the expense of increased rendering time. You should check the documentation for the rendering command you are using for options that can be used for increased quality. For example, the **ribbonjr** ''–r'' option and the **conic** (and **neon**) ''–a'' option will produce higher-quality images than standard.

We discuss in turn how to save images from MIDAS, **ribbonjr**, **conic**, and **neon**. Methods for converting saved images to other formats are covered in the ''Converting Image Formats'' section which follows.

MIDAS

The **copy file** command will save an Encapsulated PostScript file of the current modeling display. In some situations, it may be desirable to save a bitmap image of the MIDAS screen to a file (*e.g.* for image labeling). We are aware of methods for bitmap grabbing only on SGI and *NeXT* systems. On an SGI system, the *snapshot* program can be used to capture all or part of the screen into an SGI image file. Consult the *snapshot* manual page (''man snapshot'') for details. *Snapshot* can be invoked from the UNIX command line or from within MIDAS with **system snapshot**. On a *NeXT* system, *Grab.app* can be used to capture the contents of the MIDAS window into a TIFF file. If you do not want the control panel in your MIDAS image, the command **~set control** will cause it to be undisplayed. Similarly, the command **~set text** will cause the MIDAS command line and reply area to not be shown.

Ribbonjr

Ribbonjr is capable of saving to a wide variety of formats. Many of these formats (*e.g. inventor*, *pov*, *rayshade*) are simply input files for other renderers. The only format which is really an image is *tiff*. In order to save to TIFF, you must give the command-line option ''–f tiff'' and append the name of a save file to the end of the **ribbonjr** command. An additional complication is that when invoked from MIDAS, the **ribbonjr** command actually runs a front-end script that in turn runs the *ribbonjr* program. This is so that the output of the *midas* and *inventor* formats can be captured and immediately displayed. In order to use *tiff* format, you must circumvent the front end:

pdbrun surface /usr/local/midas/bin/ribbonjr –f tiff [other options] – *savefile*

Note that the final dash in the above command (before *savefile*) is space-separated from both the preceding options and from *savefile*. Also note that **ribbonjr** will still render the image to the screen because it uses the graphics subsystem for computing the image.

Conic/neon

Neon is a preprocessor which uses *conic* as a back end for final rendering. **Neon** passes any command-line arguments it gets to *conic*; therefore, the method for saving image files is identical for **neon** and **conic**. The remainder of this description will refer to **conic**, but is applicable to both. **Conic** can save to Encapsulated PostScript, SGI image file (on SGIs only!), and TIFF formats, by specifying the ''–f'' command-line flag along with the argument *ps*, *sgi*, or *tiff*, respectively. The command flag ''–o *savefile*'' must also be specified. If you use the *ps* format, you may also want to supply the ''–H'' flag (hex-encode data instead of using binary data) since many printers don't accept binary PostScript. Note that, unlike **ribbonjr**, if **conic** is given the ''–o *savefile*'' option, it won't also display the image unless the ''–s'' flag is given as well.

3.6.2. Converting Image Files to Other Formats

The *itops*(1) utility, provided with MidasPlus and documented in Appendix 6, can convert both TIFF and SGI image files to (optionally Encapsulated) PostScript. Encapsulated PostScript can be included in other documents, whereas standard PostScript can be printed directly. In fact, such *itops* output can be printed by piping directly to *lpr*(1) or *lp*(1), obviating an intermediate disk file.

On SGI systems, the *imgcopy* program can convert SGI image files to a variety of formats. Consult the *imgcopy* manual page for details of usage. A listing of the formats that *imgcopy* can convert to can be obtained by running the *imgformats* program.

On *NeXT* systems, the (free) *OmniImage.app* program can convert between a wide variety of image file formats. If you don't already have it, point your favorite web browser at http://omnigroup.com/Software/OmniImage/.

If you're an adventurous sort with a C compiler, the freely available *netpbm* suite of image tools can perform a wide variety of image format conversions. *Netpbm* is available for anonymous FTP from the sites listed in the table below.

Site	Directory	Geographic Locale
wuarchive.wustl.edu	/graphics/graphics/packages/NetPBM	USA
ikaros.fysik4.kth.se	/pub/netpbm	Sweden
ftp.informatik.uni-oldenburg.de	/pub/netpbm	Germany
peipa.essex.ac.uk	/ipa/src/manip	England
ftp.rahul.net	/pub/davidsen/source	USA

3.6.3. Printing Screen Images

The MIDAS display program's **copy** command will print the current MIDAS modeling session view. To print images from ancillary rendering programs such as *ribbonjr*, *conic*, and *neon*, one must first capture image files in TIFF or SGI image format as described in the preceding two sections. Such files can then be converted to printable PostScript files with the *itops*(1) utility, described in Appendix 6. *Itops* produces PostScript on standard output, which can be redirected to a file or piped directly to a print spooler (either *lpr* or *lp*, depending on your system's setup). A typical command (issued from the UNIX command line) to produce a printout from an image file would be:

itops −a −r *saved_image* **| lpr -P***printername*

The ''−a'' flag allows auto-scaling to best fit the image on the printed page, and the ''−r'' flag allows the image to be optionally rotated 90 degrees for best fit. Consult the *itops* manual page for other options.

Note that despite the fact that *conic* can produce PostScript with the ''−f ps'' flag, the product is Encapsulated PostScript that, while suitable for inclusion in other documents, cannot be printed directly. To produce directly printable *conic* PostScript, the procedure above of producing a TIFF or SGI image file and converting it must be followed.

3.6.4. Inclusion of Images in Documents

The preparation involved in including an image in a document largely depends on the type of document itself, which can be anything from a web page to a journal publication.

Most popular web browsers can directly image TIFF files and all (graphical) browsers can image GIF files, so images saved in those formats are typically used in web pages. The ''Saving Screen Images'' and ''Converting Image Files to Other Formats'' sections above detail how to obtain TIFF and (via conversion) GIF files from MidasPlus screen images.

Journals usually accept photographs, and the ''Screen Photography'' section (later in Part III) offers advice for photographing MidasPlus screen images.

Many types of documents, including journal articles, are prepared with word processing programs. Word processors that accept images typically allow the inclusion of TIFF format files. At times, it may be desirable to use an Encapsulated PostScript image. MIDAS (with the **copy file** command) and *conic* or *neon* (with the ''−f ps'' flag) can all produce Encapsulated PostScript files directly. TIFF and SGI image files can be converted to Encapsulated Postscript via the *itops* utility with the ''−E'' command-line flag. A typical command (issued from the UNIX command line) to convert an image file to Encapsulated PostScript would be:

itops −E *saved_image* > *EPS_file*

3.7. Making Videos

Many MIDAS commands are useful for producing video tapes and movies, and commands such as **move**, **rock**, **roll** and **turn** have optional parameters for the number of image update frames over which to carry out the command execution and the number of frames to wait before beginning execution. There is some limited support for stop motion ‘‘frame-by-frame’’ video creation. Consult the *videodisk*(1) manual page in Appendix 6 for further details.

MIDAS videos are produced from ‘‘scripts’’ or command files which can be created by the user with a text editor or recorded during a MIDAS session using the **record** command. Using command scripts minimizes the disk space required for files. To review a script, **save** a session, run the script, and do a **load** to return to the saved session’s state. To stop MIDAS in the middle of a script you may have to press the ESC key multiple times.

3.8. Screen Photography

The following table is useful as a guideline for taking photographs of Midas screen images.

Exposure Times in Seconds Using ASA 200 Slide Film for Midas Pictures

	Fstop = 2.8 *a				Fstop = 4.0 *b				Fstop = 5.6 *c				Fstop = 8.0 *d			
	1	1/2	1/4	1/8	1	1/2	1/4	1/8	1	1/2	1/4	1/8	1	1/2	1/4	1/8
ribbon (black bkg)				X	X	X	X	X								
ribbon + titles (black bkg)			X	X	X	X	X		X	X			X			
ribbon (color bkg)						X	X		X	X			X			
spacefill (black bkg)				X		X	X	X	X	X			X			
spacefill (color bkg)								X		X	X		X	X		
spacefill + titles (color bkg)							X	X	X	X			X			
sticks linewidth = 3 (color bkg)		X	X	X	X	X										
dot surface/sticks	X	X	X		X	X										
dot surf/sticks/titles	X	X	X		X	X										
most stick pictures	X	X	X		X	X										

Please note the following:

1) Most of the above pictures have five possible settings. For the safest results, at least four settings should be used. If your picture is on the bright side, you can leave out the left-most setting, while if it is dark, you can leave off the right-most setting if you are running out of or conserving film. Film is usually cheaper than the time it takes to redo pictures, however.

2) For those pictures with many possible settings, such as space-filling with a black background, it is not necessary to try each setting. It really depends on the brightness of your image. The middle settings, five of your choice, will do. The brighter images will not need the left-most settings, while the darker images will not need the right-most settings.

3) Solid-image pictures, such as space-filling pictures, ribbons, and slides with backgrounds, may show some of the screen refresh lines. These pictures need all five exposures for this reason.

4) When using the labeling program *ilabel* (see Appendix 6), it is usually best not to use solid white labels. Change the color of your labels to be, for example, slightly blue. White labels will usually be too bright to photograph well. Try to make your labels no brighter than the brightest thing in your picture.

5) To use ASA 100 film, double exposure time at the appropriate fstop, **or** use the same times listed, but one smaller fstop:
e.g.: instead of using fstop 4.0 for 1/2 second,
- use fstop 4.0 for 1 second **or**
- fstop 2.8 for 1/2 second.
- (see *a - *d, below)

*a = when using ASA 100 film, Fstop = 2.8, but times must be doubled

*b = when using ASA 100 film, Fstop = 2.8
*c = when using ASA 100 film, Fstop = 4.0
*d = when using ASA 100 film, Fstop = 5.6

3.9. Non-Molecular Graphics Objects

In addition to displaying molecular structures from PDB file information, MIDAS can display graphics objects of arbitrary connectivity from instructions in a simple command file. Graphics objects can be colored on a line-by-line basis, and given textual labels in a variety of fonts and colors. Graphics objects are frequently useful for writers of delegate programs that need to represent graphical information not directly supported by MIDAS. User manipulation of graphics objects can be made to occur in synchrony with a particular molecular model, or independently.

A graphics object command file consists of one or more lines of the following:

Explanatory comments (not displayed)	
.comment *text*	comments
.c *text*	comments
Drawing-related commands	
.color *color_designation*	set color (see below)
.m *x y* [*z*]	move to location
.move *x y* [*z*]	move to location
.d *x y* [*z*]	draw line from last location
.draw *x y* [*z*]	draw line from last location
.dot *x y* [*z*]	show dot at position
.marker *x y* [*z*]	show marker at position
Labeling-related commands	
.color *color_designation*	set color (see below)
.font *name size*	set font and size
.cmov *x y* [*z*]	set character position
label_text	label (note no initial '.')

The *color_designation* parameter to the **.color** directive can be simply a single MIDAS color, including user-defined colors (see the **colordef** command), or two colors and a mixture fraction, separated by commas. The mixture fraction is a number between zero and one that indicates the interpolation fraction along a color wheel between the two colors to use to get the actual color. Zero corresponds to the first color, while one corresponds to the second color.

In MIDAS, the ''**open obj** *modelnum*'' command is used to open object files. If a model number corresponding to an open molecule is used, then the object will move in synchrony with that molecule. Display and undisplay of an object is controlled with the **objdisplay** command, and is independent of the display status of the corresponding molecule (if any).

3.10. Chemical Group Description Files

The MIDAS **addgrp** command is used to add a new chemical group to a displayed molecular structure. Many common chemical groups are supplied in the MIDAS groups directory (see **addgrp**), but it may be necessary to create a group file to describe an uncommon chemical group. Such a file would be placed in the user's personal group directory (see **addgrp**) for use.

If the user has some PDB file that contains the chemical group, by far the easiest method to get the corresponding group file is to use the *pdb2group*(1) utility provided with the MidasPlus distribution (see Appendix 6). Otherwise, the group description file must be constructed by hand. Information for doing so is provided below.

The format of a group file is most easily understood by providing an example. This is the contents of the group file for S-phenyl (file name: S-Ph):

```
 (1)    S-phenyl
 (2)    new S  0.000    0.000   0.000
 (3)    new C  0.000    0.000   0.000
 (4)    new C  0.000    0.000   0.000
 (5)    new C  0.000    0.000   0.000
 (6)    new C  0.000    0.000   0.000
 (7)    new C  0.000    0.000   0.000
 (8)    new C  0.000    0.000   0.000
 (9)    read internal coordinates for new group
(10)    + 1 0 n3 n2  1.75  105.0   60.0
(11)    + 2 1  0 n3  1.4   120.0   60.0
(12)    + 3 2  1  0  1.4   120.0  180.0
(13)    + 4 3  2  1  1.4   120.0    0.0
(14)    + 5 4  3  2  1.4   120.0    0.0
(15)    + 6 5  4  3  1.4   120.0    0.0
(16)    = 6 1  2  3  1.4   120.0    0.0
```

Line *(1)* is the descriptive name of the chemical group. Lines *(2)* through *(8)* identify the atoms that compose the group. There is one line per atom in the group. The only significant field on these lines is the second field, which is the name of an atom in the group. Note that to ensure uniqueness **addgrp** will append a number to the atom name, so in MIDAS the final names of atoms in this group would be `S1`, `C2`, `C3`, *etc.* The other fields must be present but are otherwise ignored. Line *(9)* marks the end of the lines describing group atoms. It should be present *verbatim*.

The lines that follow give information as to how each atom is positioned relative to other atoms in the group, and possibly also with respect to atoms mentioned on the **addgrp** command line that invoked this group file. The atoms in the group are implicitly numbered in ascending order starting from zero. So, the atom mentioned in line *(2)* would be numbered 0, the atom in line *(3)* would be 1, *etc*. The three atoms given on the **addgrp** command line are referred to as n3, n2, and n1 (from left to right on the **addgrp** command line).

The information provided on the **addgrp** command line completely determines the position of the first atom of the group (atom 0, line *(2)*), so there is no corresponding position information line for it in the group file. The first position information line, line *(10),* provides information to position atom 1 (line *(3)*).

The fields of a position information line, reading from left to right, are: a '+' or '=' character, four atom designators, a bond length, a bond angle, and a dihedral angle. The '+' or '=' character is required to be present for historical reasons, but is otherwise ignored. The four atom designators that follow indicate, from left to right: the atom being positioned, an atom to which the first is connected, an atom that forms a bond angle with the first two, and an atom that forms a dihedral with the first three. The trailing three fields provide the actual values of the bond length, bond angle, and dihedral.

Each positioning line causes MIDAS to draw a corresponding bond. If there are more bonds in the group than there are atoms (if a ring is present, for example), then additional positioning lines need to be provided for the additional bonds. Line *(16)* is such a line. These lines are typically introduced with a '=' instead of a '+,' though '+' will still work.

3.11. Web-Browsing PDB Files with Midas

If you browse web sites that offer PDB files, you may want to have MIDAS invoked to view the PDB files of interest. The procedure for doing so is fairly simple. In your home directory, edit your ''.mailcap'' file (or create one if none exists). Add the following line to the file:

```
chemical/x-pdb; midas -nofork %s
```

This will also cause MIME-capable mail-reading programs such as SGI's *MediaMail*(1) to use MIDAS to display PDB files.

If you make PDB files available via the web, you have to ensure that your web server marks such files with the ''chemical/x-pdb'' MIME content type. You may need to contact your webmaster to set this up. Conversely, if a site you are browsing fails to mark its PDB files with the ''chemical/x-pdb'' content type, then despite the .mailcap setup above, MIDAS will not be invoked to display the PDB file.

3.12. Performance Issues with Midas

By default, MIDAS is configured to use the best combination of aesthetic and performance options for common modeling situations. On platforms with certain graphics hardware, or for certain modeling sessions (*e.g.* very large systems), you may wish to vary from these default options to obtain improved performance. This section details various performance options you may want to change at times.

3.12.1. Issues Common to All Platforms

There are two aesthetic options normally enabled in MIDAS that you may want to disable for increased performance: anti-aliasing (smoothing of line segments) and half-bond coloring.

Disabling anti-aliasing will result in lines with considerably more noticeable "stair-stepping" artifacts, but can significantly increase performance on some platforms − particularly those with no hardware anti-aliasing support. Anti-aliasing is turned off with the command **devopt smooth off**.

Disabling half-bond coloring means that bonds drawn between atoms of different colors will not be half one color and half the other. Instead, the entire bond will be the color of one of the two endpoint atoms. This makes little visual difference for uniformly colored models or models colored on a per-residue basis,[1] but may be unsatisfactory for models colored on a per-atom basis. But, since MIDAS only has to draw one line for each bond instead of two, disabling half-bond mode can result in a two-fold speedup on some platforms. Half-bond mode is disabled with the command **˜set halfbond**.

It should be noted that if you use thicker lines than the default (via the **set linewidth** command), this has much the same impact on performance as anti-aliasing, and in fact combines with anti-aliasing. If you wish to use increased line widths but experience sluggish performance, you should investigate disabling anti-aliasing as per the above.

3.12.1.1. Competing with Background Jobs

If your MIDAS workstation is also used for long-running background computation jobs, you may experience degraded performance due to the fact that memory areas containing MIDAS code and data may be swapped out to disk in favor of background jobs when the MIDAS user is not actively manipulating the model or typing commands.

If it is desirable to have MIDAS ‘‘wire down’’ its memory areas and prevent them from being swapped out, you can do so with the **devopt memlock on** command. If you get an error message when executing this command, then your system administrator did not enable this option when installing MIDAS. The MidasPlus Installation Guide describes how your system administrator can enable the option if it is desired to do so.

3.12.2. SGI-Specific Performance Issues

In the MidasPlus 2.0 release, the MIDAS binary provided for SGI systems was written using the Iris GL graphics library. Since that time, SGI has transitioned to the OpenGL graphics library and new SGI systems support OpenGL natively (*i.e.* their graphics hardware is designed for OpenGL) and support for Iris GL programs is provided through compatibility libraries. Conversely, older systems with native Iris GL hardware support OpenGL through compatibility libraries. This raises a performance issue in that some graphics boards exhibit markedly degraded performance when used with the non-native type of library.

As a consequence, on SGI systems the **midas** executable is a front-end program that first queries the system to determine the type of graphics hardware present, and then executes either **midas.irisgl** or **midas.opengl** depending on which should provide better performance. The heuristics used by the **midas** front-end program were determined empirically from the various graphics hardware types available to us at UCSF, and may not cover every system type correctly − particularly more esoteric hardware options. Nonetheless, we believe them to be correct for 95%+ of SGI workstations. If you believe you have a system where the incorrect back-end program is selected, you can test this out by executing the alternate back-end directly, as described below. To have **midas** choose a different back-end for your workstation permanently,

[1] It also makes no performance difference for these color schemes since MIDAS optimizes half-bonds of the same color into a single draw.

edit the file /usr/local/midas/resource/glgraphics as per the comments contained in that file.

To determine which back-end program **midas** is selecting, execute **midas –noexec**, which will print out the name of the back-end program (with ''–noexec'' appended, which should be ignored).

In the simple case where you want to run the back-end with no arguments or with just a session file argument, you can do so with no problem. If you want to use command-line options, however, you need to run **midas –noexec** with your options appended so that you can see what those options correspond to when used with the back-end program (again ignoring the appended ''–noexec''). The options differ between the two back ends due to the way the X window system (used in conjunction with OpenGL) handles command-line options. Of course, this only tells you what options to use with the *normally selected* back end. If you actually want to execute the other back end, the **midas –noexecother** will show how to run the other back-end program with your desired command-line arguments.

Executing the back-end program directly also allows you to control the user interface used, since the interfaces differ in some regards between the two versions.

Appendix 1: Pdbrun-Format USER Records

A PDBRUN file has three principal sections. The first section (*Annotation Headers*) has global modeling information. The second section (*Per-Molecule Annotations*) has molecular data for each model. Each model's data starts with a `USER FILE` record and is terminated with an `END` record. The serial numbers in `ATOM`, `HETATM`, and `TER` records are incremented continuously from one model to the next. Due to field widths, this imposes a limit of 99999 atoms in a scene. Any PDB records present in the model's original source file (and not normally interpreted by MIDAS, *e.g.* `REMARK`s) are preserved and placed just after the `USER FILE` record. The data for non-molecular graphics objects are also in this section, bounded by `USER OBJECT` and `USER ENDOBJ` records. The third section (*Annotation Trailers*) contains records specifying the angles and distances being explicitly monitored in the current view. Throughout, fields in PDB records that are not wide enough to hold their associated values are filled with asterisks instead.

Below, the `USER` records found in each section are explained in detail. The formats for these records in both the C and FORTRAN programming languages are presented at the end of this appendix.

Annotation Headers

`USER   PDBRUN` *version*
: Specifies the pdbrun format version used to annotate the file. MidasPlus 2.1 uses version 6. MidasPlus 2.0 used version 5. There are utility programs for converting between these two versions if necessary. See *pdbrun5to6*(1) and *pdbrun6to5*(1) in Appendix 6 of this manual.

`USER   EYEPOS` *x y z*
: The viewer's position (*x*,*y*,*z*), for purposes of calculating what to display.

`USER   ATPOS` *x y z*
: The location the viewer is assumed to be looking towards for determining line of sight.

`USER   WINDOW` x_{left}, x_{right}, y_{bottom}, y_{top}, z_{hither}, z_{yon}
: View volume display bounds, relative to the line of sight. Except in stereo views, the MIDAS view volume is symmetric about the line of sight; x_{left} and y_{bottom} are both negative and equal in magnitude respectively to x_{right} and y_{top}, which are both positive. z_{hither} and z_{yon} are positive distances from the viewer to the *hither* and *yon* clipping planes, respectively.

`USER   FOCUS` *focal length*
: Distance from viewer to focal plane, along line of sight. Used primarily when making stereo projections. The focal length is typically between the hither and yon distances. If the focal length is zero, then an orthographic projection is used rather than a perspective projection.

`USER   VIEWPORT` x_{min} x_{max} y_{min} y_{max}
: Extent of the MIDAS window, in screen coordinates.

`USER   BGCOLOR` *red green blue*
: Color of the MIDAS background, given as amounts of red, green, and blue, each in the range 0 to 1.

Per-Molecule Annotations

`USER   FILE` *model filename*
: Each embedded PDB file or graphics object file is identified by the original *filename* that the molecular or graphics data was fetched from, and by its *model* number.

`USER   MARKNAME` *name*
: This record notes a mark name in use in the MIDAS session (see the **makemark** command). That mark name may be then used in subsequent `USER MARK` records.

`USER   MARK` *name*
: This annotation marks the following `ATOM` or `HETATM` record with the given *name*. Unlike the `USER COLOR` and `USER RADIUS` records (which see), marks are not inherited by subsequent records. A single atom may have multiple marks.

`USER  CNAME` *red green blue color-name*

This annotation defines a named color and its composition. *Red*, *green*, and *blue* are floating point values between 0 and 1, inclusive.

`USER  COLOR` *red green blue color-specification*

This annotation sets the color of the following `ATOM` or `HETATM` record. If a color is not given for an atom, then it inherits the value of the previous `USER COLOR` record. The *color-specification* is either a single color name (see `USER CNAME` record above) or a comma-separated triplet of [color-name$_0$, color-name$_1$, interpolation-factor], where interpolation-factor is a floating-point value between −1 and 1, inclusive, and the absolute value indicates the proportion of hue, lightness, and saturation (HLS) components of the two named colors.[†] The [*red*, *green*, *blue*] triplet is redundant (because the values are already implied by *color-specification*), but are provided as a convenience so that programs reading **pdbrun** files need not provide code to interpret the *color-specification* themselves.

`USER  RADIUS` *radius*

This annotation sets the van der Waals radius in angstroms of the following `ATOM` or `HETATM` record. If a radius is not given for an atom, then it inherits the value of the previous `USER RADIUS` record.

`USER  OBJECT`
`USER  ENDOBJ`

These annotations denote the start and end of a user-defined graphics object specification (see *3D Graphics Annotations*, below).

`USER  CHAIN` *atom$_0$ atom$_1$*

This annotation indicates a pseudo-bond. A `USER CHAIN` annotation is used to denote that two atoms should be depicted as if they were bonded even though they are not — such as in a backbone trace where α-carbons of a protein are shown as connected. *Atom$_0$* and *atom$_1$* are atom serial numbers.

Annotation Trailers

`USER  ANGLE` *atom$_0$ atom$_1$ atom$_2$ atom$_3$ value*

A `USER ANGLE` annotation will be issued for each angle being interactively monitored in MIDAS. *Atom$_0$*, *atom$_1$*, ... are atom serial numbers. If *atom$_3$* is non-zero, it is a dihedral angle, otherwise a simple bond angle. The *value* is the angle in degrees.

`USER  DISTANCE` *atom$_0$ atom$_1$ value*

A `USER DISTANCE` annotation will be issued for each distance being interactively monitored in MIDAS. *Atom$_0$* and *atom$_1$* are atom serial numbers. The *value* is the distance in angstroms.

3D Graphics Annotations

These annotations appear between `USER OBJECT` and `USER ENDOBJ` records.

`USER  GFX BEGIN` *type*
`USER  GFX END`

The graphics `BEGIN` and `END` bound a graphics primitive. Within a graphics primitive, there can be any number of vertices (see below) and optional color changes. *Type* is one of the following:

`POINTS`

Each vertex represents a point.

`MARKERS`

Markers are like points, but they are represented differently. Markers are used for singleton atoms, whereas

[†]Color interpolation is done using the HLS color model because it provides superior interpolation results compared to an RGB color model. HLS color interpolation is as follows: $H = H_0 + |f| * (H_1 - H_0)$, $L = L_0 + |f| * (L_1 - L_0)$, $S = S_0 + |f| * (S_1 - S_0)$ — where the hue is interpolated in a counterclockwise direction when the interpolation factor, *f*, is positive, and clockwise when it is negative.

points are used for a molecular surface.

`LINES`
Every two vertices describe a line.

`LINE-STRIP`
Each vertex is connected to the previous vertex in a line.

`LINE-LOOP`
A `LINE-LOOP` is like a `LINE-STRIP`, but the last vertex is connected to the first.

`TRIANGLES`
Every three vertices describe a triangle.

`TRIANGLE-STRIP`
The first three vertices are a triangle, and each subsequent vertex forms a connected triangle with the previous two.

`TRIANGLE-FAN`
The first vertex is the center of a fan and the subsequent vertices form triangles around the center.

`QUADS`
Every four vertices describe a quadrilateral.

`QUAD-STRIP`
The first four vertices are a quadrilateral, and each subsequent pair of vertices forms a quadrilateral with the previous two.

`POLYGON`
The vertices form a polygon.

It should be noted that since MIDAS cannot display solid surfaces, it will never use some of the above types (*e.g.* `TRIANGLE-STRIP`). However, *ribbonjr* can in fact accept *all* the above types. Thus, if one developed a program that employed one of the non-MIDAS types, or wrote a filter that added such types to a MIDAS-generated PDBRUN file, *ribbonjr* could be used to display the result.

Further 3D graphics annotations are:

`USER  GFX COLOR` *red green blue color-specification*
User-defined graphics object color; same format as the `USER COLOR` record above. Color records may appear outside or within graphics primitives and apply to the following primitive or vertex.

`USER  GFX NORMAL` $n_x\ n_y\ n_z$
A normal vector for the next vertex.

`USER  GFX VERTEX` *x y z*
A vertex with given coordinates.

`USER  GFX FONT` *font-size font-name*
This annotation switches subsequent label text to the given font name and font size.

`USER  GFX TEXTPOS` *x y z*
This annotation sets the position of the next label (see below).

`USER  GFX LABEL` *text*
This annotation places a label starting at the current text position. The label text must be between double quotes. The quoted text follows the ANSI C language rules for string constants, that is, a backslash character starts an escape sequence, a backslash double quote character sequence indicates a double quote, and so on.

Annotation Record Formats

For the C format specification, the leading `USER` is left out, and the format given below starts in column 7. The Fortran formats include reading the word `USER`. Annotations marked with a dagger (†) are required.

Annotation	C format	Fortran format
	Annotation Headers	
PDBRUN†	`PDBRUN %2d`	6A1,6A1,X,I2
EYEPOS†	`EYEPOS %9.3f %9.3f %9.3f`	6A1,6A1,3(X,F9.3)
ATPOS†	`ATPOS  %9.3f %9.3f %9.3f`	6A1,5A1,X,3(X,F9.3)
WINDOW†	`WINDOW %9.3f %9.3f %9.3f %9.3f %9.3f %9.3f`	6A1,6A1,6(X,F9.3)
FOCUS†	`FOCUS  %9.3f`	6A1,5A1,2X,F9.3
VIEWPORT	`VIEWPORT %9.3f %9.3f %9.3f %9.3f`	6A1,8A1,3(X,F9.3)
BGCOLOR	`BGCOLOR %5.3f %5.3f %5.3f`	6A1,7A1,3(X,F5.3)
	Per-Molecule Annotations	
FILE†	`FILE %4d %-.56s`	6A1,4A1,X,I4,X,56A1
MARKNAME	`MARKNAME %-.57s`	6A1,8A1,X,57A1
MARK	`MARK %-.57s`	6A1,4A1,X,57A1
CNAME	`CNAME %5.3f %5.3f %5.3f %-.38s`	6A1,5A1,3(X,F5.3),X,38A1
COLOR	`COLOR %5.3f %5.3f %5.3f %-.38s`	6A1,5A1,3(X,F5.3),X,38A1
RADIUS	`RADIUS %7.3f`	6A1,6A1,X,F7.3
OBJECT	`OBJECT`	6A1,6A1
ENDOBJ	`ENDOBJ`	6A1,6A1
CHAIN	`CHAIN %6d %6d`	6A1,5A1,2(X,I6)
	Annotation Trailers	
ANGLE	`ANGLE %6d %6d %6d %6d %9.3f`	6A1,5A1,4(X,I6),X,F9.3
DISTANCE	`DISTANCE %6d %6d %9.3f`	6A1,8A1,2(X,I6),X,F9.3
	3D Graphics Annotations	
GFX BEGIN	`GFX BEGIN %-.32s`	6A1,9A1,32A1
GFX END	`GFX END`	6A1,7A1
GFX COLOR	`GFX COLOR %5.3f %5.3f %5.3f %-.38s`	6A1,9A1,3(X,F5.3),X,38A1
GFX NORMAL	`GFX NORMAL %9.3f %9.3f %9.3f`	6A1,10A1,3(X,F9.3)
GFX VERTEX	`GFX VERTEX %9.3f %9.3f %9.3f`	6A1,10A1,3(X,F9.3)
GFX FONT	`GFX FONT %3d %-.53s`	6A1,8A1,X,I3,X,53A1
GFX TEXTPOS	`GFX TEXTPOS %9.3f %9.3f %9.3f`	6A1,9A1,3(X,F9.3)
GFX LABEL	`GFX LABEL "%-.56s"`	6A1,9A1,X,56A1

Summary of Changes from PDBRUN Version 5 to Version 6

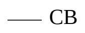
Record Type	Version 5	Version 6
USER PDBRUN	5	6
MASTER	One record for entire file	One record per embedded file
USER COLOR USER RADIUS	Follows ATOM/HETATM record. Applies only to that record.	Precedes ATOM/HETATM record. Applies to that record and any subsequent records until next USER COLOR/RADIUS.
USER COLOR USER CNAME	RGB triplet follows *color-specification*.	RGB triplet precedes *color-specification*.

MODEL
ENDMDL
USER FILE
END
	Models demarcated with MODEL/ENDMDL pairs.	Models demarcated with USER FILE/END pairs.

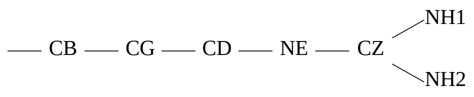
USER OBJECT USER ENDOBJ	Model number indicated in record.	Model number indicated by surrounding USER FILE/END pair.
USER FONT	Font size follows font name.	Font size precedes font name.
USER GFX TEXTPOS	Non-existent	Indicates position of ensuing text label.
USER GFX LABEL	Includes text label position.	Text label position indicated by preceding USER GFX TEXTPOS record. Multiple consecutive USER GFX LABELs indicate concatenation.
GFX primitives	Quads, triangles, polygons, and normals not supported.	Quads, triangles, polygons, and normals supported by *ribbonjr*.

Appendix 2: PDB Atom-Naming Conventions for Amino Acids

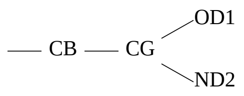
Alanine



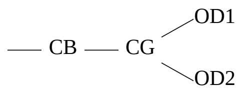
Arginine



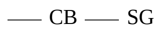
Asparagine



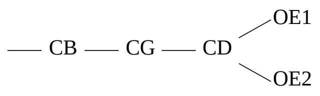
Aspartic acid



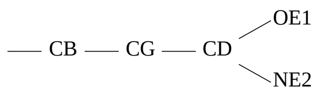
Cysteine



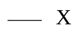
Glutamic acid



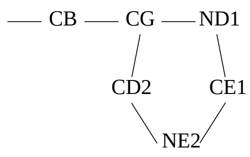
Glutamine



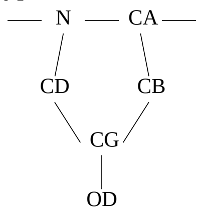
Glycine



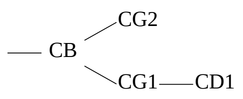
Histidine



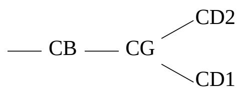
Hydroxyproline



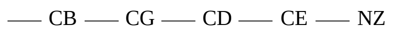
Isoleucine



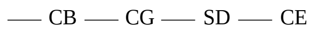
Leucine



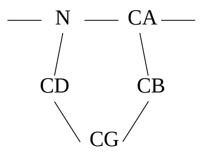
Lysine



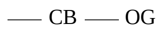
Methionine



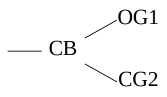
Proline



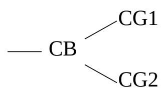
Serine



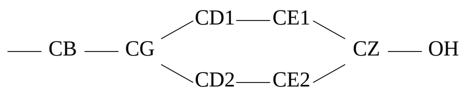
Threonine



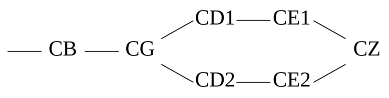
Valine



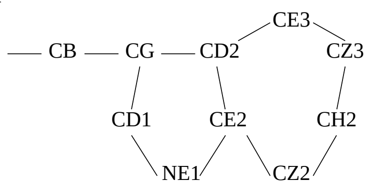
Tyrosine



Phenylalanine



Tryptophan



Appendix 3: Special Characters and Symbols Used in MIDAS

The following table describes symbols which have special meaning to MIDAS. In addition to these symbols, several special characters are available for use in command-line editing (users of the EMACS text editor will readily recognize most of these special editing characters).

Symbol	Function	Usage
#	model number	**#** *model_number* where *model_number* is an integer.
:	residue	**:** *residue* where *residue* is a residue name, residue sequence number, or range of residues.
@	atom name	**@** *atom_name* where *atom_name* is an atom name or range of atoms.
–	range	specifies a range of atoms such as @CB–* (beta carbon to the last atom in residue branch), a range of residues such as :35-66 (residues 35 through 66) or a range of colors such as red-blue (shades of red, magenta, and blue).
,	name separator	separates names or ranges in an atom specifier (but does not indicate a specific order), *e.g.* #1@CA,CB,CG or :21-30,45.
*****	whole wildcard match	matches whole atom or residue names. For example, #0:*****@CA selects the alpha carbon atoms of all residues.
=	partial wildcard match	matches partial atom or residue names, *e.g.* #0:*@C= matches all atoms whose names begin with C.
?	single character wildcard	used for atom and residue *names* only. For example :G**??** selects all three letter residue names beginning with G.
%	every *nth* residue or atom	For example, :***%**5 selects every fifth residue in the sequence.
z>	zone specifier	**z<***zone* and **zr<***zone* select all residues within *zone* angstroms of the indicated atoms. **za<***zone* selects all atoms (not residues) within *zone* angstroms. Using > instead of **<** results in the complementary set of atoms.
b>	temperature factor	**b>** *temp_factor* selects all atoms with temperature factors greater than *temp_factor*. **b<** *temp_factor* selects all atoms with temperature factors less than *temp_factor*. For example, **b**>20 **b**<25 selects all atoms with temperature factors greater than 20 and less than 25.

Symbol	Function	Usage
e>	electrostatic potential	**e>** *potential* selects all atoms with electrostatic potentials greater than *potential*. **e<** *potential* selects all atoms with electrostatic potentials less than *potential*. For example, **e>**10 **e<**20 selects all atoms with electrostatic potentials between 10 and 20 kcal/mole per elementary charge.
&	intersection	Finds the set of atoms that appear in both atom specifiers on left and right. *e.g.* #1 & #2:1 zr<10 gives all residues in model one that are within 10 angstroms of residue one in model two.
;	command separator	separates multiple commands on a single line.
RETURN	return	accept the line.
LINEFEED	linefeed	accept the line.
RUBOUT	backspace	erase the character before the cursor.
CTRL-H	backspace	erase the character before the cursor.
CTRL-U	line kill	erase the whole line.
CTRL-W	word kill	erase the word before the cursor.
CTRL-D	delete	erase the character under the cursor.
CTRL-K		erase to end of line.
CTRL-P	history	retrieve previous command.
CTRL-N	history	retrieve following command.
CTRL-A		go to beginning of line.
CTRL-E		go to the end of line.
CTRL-B		move back a single character.
CTRL-F		move forward a single character.
CTRL-L		move cursor one word left.
CTRL-R		move cursor one word right.
CTRL-G		insert next character without interpretation.
ESC	break	break after the completion of the current command. (see **source** command.)

Note: Control characters are typed by holding down the **CTRL** key on the keyboard and typing the corresponding alphabetic character. For example, to type **CTRL**-H hold down the key marked ‘‘**CTRL**’’ while at the same time striking the ‘‘H’’ key.

Appendix 4: Default Options, Aliases and Device Assignments

The following is a list of the default options, aliases and device assignments made by MidasPlus at the start of each MidasPlus session. These default assignments are stored in */usr/local/midas/resource/midas/midas.rc*. The user may replace these defaults with an alternate file defined by the UNIX environment variable MIDASRC. If this variable is set to a legal source file name, it is executed *instead of* the MidasPlus default file. This is especially useful for making video tapes, as the user may not want extraneous text appearing on the screen at the beginning of the session.

The default assignments are:

assign 0 section	*assign useful functions to the first few sliders*
assign 1 thickness	
assign 2 scaling	
# slider 3 left open for user assignment	
~assign 3	
assign 4 rot 0	*pre-assign some bond rotations*
assign 5 rot 1	
assign 6 rot 2	
assign 7 rot 3	
colordef white 1 1 1	*define standard color names*
colordef green 0 1 0	
colordef cyan 0 1 1	
colordef blue 0 0 1	
colordef magenta 1 0 1	
colordef red 1 0 0	
colordef yellow 1 1 0	
colordef black 0 0 0	
colordef gray .5 .5 .5	
colordef grey gray	
alias model #	*a few useful aliases*
alias molecule #	
alias residue :	
alias atom @	
alias sidechain @ cb - *	
alias mainchain @ n,ca,c,o	
alias ^close ~open	*aliases that start with ^ only expand at beginning of commands*
alias ^conic pdbrun /usr/local/midas/bin/conic	*several "commands" are actually* ***pdbrun*** *aliases*
alias ^ksdssp pdbrun ksdssp	
alias ^mardishow pdbrun noobj mardishow	
alias ^neon "preneon | /usr/local/midas/bin/conic"	
alias ^noeshow pdbrun noobj noeshow	
alias ^pdbopen delegate start pdbopen pdbopen	
alias ^preneon "pdbrun conect surface /usr/local/midas/bin/neon"	***preneon*** *is useful if several* ***neon*** *outputs must be combined before* ***conic*** *processing*
alias ^quit stop	*help those new users!*
alias ^rainbow pdbrun all nouser rainbow	
alias ^ribbonjr pdbrun surface ribbonjr	
alias ^stereoimg run stereoimg	
devopt move_windows on	*by default, keep the three* MIDAS *windows in same relative positions*
set record	*start remembering commands for later* ***record****ing*
set text	*show command line and reply area*
set labels	*show distance, angle, and rotation monitors*
set control	*show control panel*

set verbose autocolor halfbond	
set refmodel -1	*newly opened models oriented same as lowest open model*
set reassign	*new slider assignments override old*
set molpath .:~/pdb:/mol/pdb:/usr/local/midas/resource/pdb	*directories searched to locate PDB files*

Appendix 5: The MidasPlus Delegate Mechanism

What are delegates?

MIDAS has a rich command language, and users can specify nearly all operations by typing them in on the keyboard. Occasionally, however, it is useful to have other computer programs compose the commands. This functionality is partially provided by the **pdbrun** command, which sends the current transformed coordinates of molecules to a user-specified program, and treats the output of that program as MIDAS commands. **pdbrun** is most useful for ‘‘one-shot’’ type computations, such as alternative rendering (*conic*) or coloring atoms according to sequence position (*rainbow*). For computations that require information at several different times during a single MIDAS session, however, **pdbrun** is too inefficient.

The solution is a mechanism that allows designated programs, called *delegates*, to communicate with MIDAS while executing in parallel. For example, there is a delegate called *mrotate* supplied with the MidasPlus distribution that supports two commands: **snapshot** and **interpolate**. Users can use the **snapshot** command to save molecule positions at selected points during a session, and the **interpolate** command to smoothly interpolate between two saved positions. While it would be possible to add the **snapshot** and **interpolate** commands directly to MIDAS, this approach requires one to have both an understanding of the internal program structure and permission to change source files. Writing the rotation delegate required no modifications to MIDAS, and once the matrix arithmetic was solved, took less that a day to implement. Less than 300 lines of new code (including comments) were written.

The delegate facility provides an alternative to modifying MIDAS every time additional functionality is desired. Users who are willing to program can easily implement their own flavors of delegates and need not wait for MIDAS developers to implement desired new features. In addition, delegates can use information sources other than MIDAS. Thus, they can potentially inject some much-needed chemistry information into MIDAS.

User Perspective

At MIDAS start-up, there are no active delegate programs. To make use of a delegate program during a MIDAS modeling session, the user issues a **delegate start** command to request that MIDAS start the delegate program. One of the parameters of the **delegate start** command is an arbitrary string used as a name to refer to that delegate during the remainder of the MIDAS session. MIDAS command lines that start with that string will be given to the delegate program for processing, less the initial delegate-name string.

Delegates are controlled from MIDAS using the *delegate* command, which supports three operations: **start**, **stop**, and **list**. New delegates are invoked with the command

delegate start *delegate_name command* [*options...*]

where *delegate_name* is the name that the user will use to refer to the started delegate process, and *command* is the Unix command to execute the delegate program. Associated command-line options, if any, can be given as *options*. Once a delegate is running, the user can send commands to it by prefixing the command with the name of the delegate, *i.e.*,

delegate_name delegate_command [*arguments...*]

To terminate an active delegate, the user can issue the command

delegate stop *delegate_name*

Finally, the command

delegate list

lists the names of all active delegates.

Implementor Perspective

Delegate programs interact with MIDAS by issuing valid MIDAS commands and receiving as feedback the replies to those commands. The delegate also receives any commands directed to it by the user.

When a delegate is executed, its standard input (C standard I/O library **stdin** or Unix file descriptor 0) and standard output (**stdout** or descriptor 1) are connected to MIDAS. Data sent to **stdout** will be interpreted by MIDAS. Data received on **stdin** are either commands from the user (via the *delegate_name* mechanism), or replies from MIDAS commands (which are normally displayed to the user).

The communications protocol between the delegate and MIDAS is very simple. An event diagram of the protocol is shown in Figure 1. On start-up, the delegate sends lines of commands to MIDAS for execution. For each line of command, MIDAS sends back to the delegate all the reply lines, followed by a line containing only the word **SYNC**. When the delegate is finished with its initialization, it informs MIDAS by sending a line containing only the word **SYNC**. At this point, the delegate should wait for user commands, which will arrive on **stdin** via MIDAS.

When the delegate receives a user command, it, once again, sends lines of commands to MIDAS, using exactly the same synchronization as on start up. Between the time it forwards a user command to the delegate and the time it receives the terminating **SYNC** message from the delegate, MIDAS will not accept any input from the user. Thus, a buggy delegate can cause MIDAS to appear to hang until the delegate exits. When the delegate receives an end-of-file indication on **stdin**, it should terminate gracefully without trying to communicate with MIDAS, as **stdout** may already been closed. Conversely, MIDAS detects when a delegate closes **stdout** and removes the delegate from the list of active delegates. This might be necessary if the delegate detects an error and wishes to exit or if the delegate wants to perform a long calculation and needs no futher interaction with the user.

While waiting for a user command, a delegate may spontaneously generate MIDAS commands. This is useful for programs that need to run for a substantial period of time before yielding results and can let the user manipulate the model while the computation occurs. These delegates can send the **SYNC** message before entering background

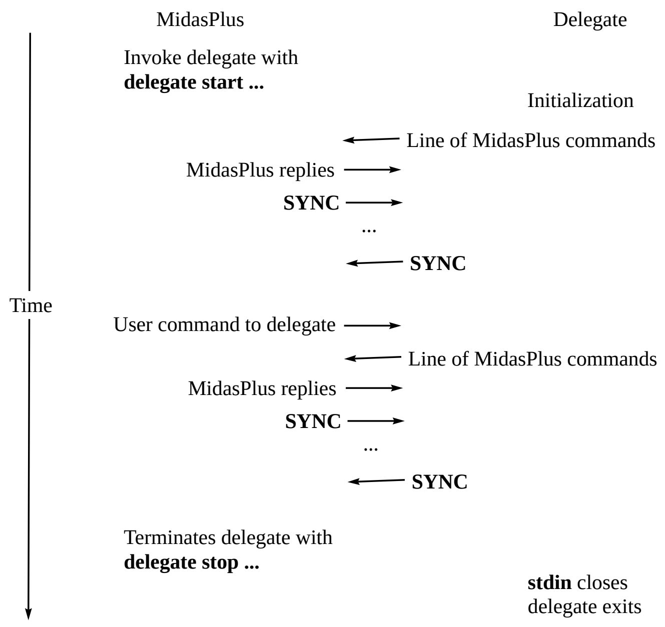


Figure 1 – Event diagram of delegate communications protocol

computation mode, and then spontaneously generate MIDAS commands once the calculation completes. The only drawback to using background computation is that the user may try to send more commands to the delegate, resulting in a race condition, where the user command may be interpreted as a reply to a MIDAS command. Implementors of this type of delegate should clearly document which user commands use background computation modes, and possibly warn the user at run-time that further input is not advisable.

A delegate can request that the user pick an atom by sending the **pickatom** command to MIDAS. The delegate should be prepared to handle the situation where the user aborts the pick with the **pickabort** command. Both commands are documented fully in Part II of this manual. Note that the *discern*(1) utility (described in Appendix 6) is a MidasPlus delegate and its source code is supplied with the MidasPlus source distributions.

Translating MIDAS atom specifiers into corresponding PDB records can be very difficult in some cases. To assist delegates in expanding atom specifiers, MIDAS has a **mark** command in order to associate a name with atoms corresponding to an atom specifier. The delegate can first **mark** the user-specified atoms and then use the **pdbrun** command with the *mark=name* option to obtain PDB records corresponding to the mark. See the **mark** command documentation for further details. For maximum efficiency, a delegate should prevent MIDAS from transmitting **pdbrun** information that is not of interest. The **pdbrun** keywords *nouser* (do not include USER records) and *noobj* (do not include non-molecular graphics object descriptions) are used for this purpose.

It is also possible to write delegates that start MIDAS instead of the other way around. If *midas* is started with the ‘‘**–d** *delegate_name*’’ option, then *midas* will treat its standard input and output as a delegate of the given name, which make it possible for *midas* to be started as a subprocess from a delegate program.

Other Resources and Information

As a delegate implementor, you may be able to reduce your work considerably by being aware of the following:

- MIDAS can display non-molecular graphics objects composed of lines, dots, and/or text. This maybe of use if your delegate needs to graphically annotate the displayed molecules. See section 3.10, ‘‘Non-Molecular Graphics Objects,’’ for more details.
- The MidasPlus distribution provides C and C++ libraries to simplify PDB-format input and output. There are man pages for ‘‘pdb’’ and ‘‘pdb++’’ that describe the routines provided and how to access them.
- Source code for some MidasPlus delegates is provided in the directory /usr/local/midas/resource/examples/delegates. Examining the source code may help clarify some aspects of delegate design, and some subroutines might be useful to you as is. Source code for the entire MidasPlus distribution, including all delegates, is provided on the installation CD. If you need to access that source code, you will have to ask your system administrator if he/she has installed the MidasPlus sources, and if so, in what location.
- There have been several additions to the MIDAS command language specifically to support delegate operations. A list of commands related to delegate support can be found in the table of commands grouped by function, near the beginning of section 2.4. Specifically, the groupings ‘‘Auxiliary Program Support’’ and ‘‘Script Support’’ are of interest.

An example of a delegate that simply echoes user input follows.

```
#include <stdio.h>
#include <stdarg.h>

#ifndef TRUE
#define TRUE    1
#define FALSE   0
#endif

FILE    *record_fp;

/*
 * Sample MidasPlus delegate
 */
main(int ac, char **av)
{
        register int    c;
        int             verbose = FALSE;
        char            buf[BUFSIZ];
        extern int      optind;
```

```
     extern char       *optarg;
        static int     send_command(char *, ...);

        /*
         * Process command line arguments
         */
        while ((c = getopt(ac, av, "vr:")) != EOF)
                switch (c) {
                  case ’v’:
                        verbose = TRUE;
                        break;
                  case ’r’:
                        record_fp = fopen(optarg, "w");
                        if (record_fp != NULL)
                                setbuf(record_fp, (char *) NULL);
                        break;
                }

        /*
         * First we sync with MIDAS, which is expecting us to notify
         * it of proper start up.
         */
        (void) printf("SYNC\n");
        (void) fflush(stdout);

        /*
         * We simply echo back any commands to MIDAS.  If we are
         * in verbose mode, we notify the user before and after
         * each command
         */
        while (fgets(buf, sizeof buf, stdin) != NULL) {
                if (verbose)
                        send_command("echo Delegate executing %s", buf);
                send_command(buf);
                if (verbose)
                        send_command("echo Delegate done\n");
                (void) printf("SYNC\n");
                (void) fflush(stdout);
        }

        exit(0);
}

/*
 * send_command:
 *      Send a command to MIDAS and wait until the
 *      synchronizing string comes back
 */
static
int
send_command(char *fmt, ...)
{
        va_list args;
        char    buf[BUFSIZ];

        va_start(args, fmt);
        (void) vfprintf(stdout, fmt, args);
        if (record_fp != NULL) {
                fputs("> ", record_fp);
                vfprintf(record_fp, fmt, args);
        }
        va_end(args);
        (void) fflush(stdout);        /* Make sure buffered data get sent */

        /*
         * Keep reading until we see a line containing only SYNC,
         * which denotes end of MIDAS replies
         */
        while (fgets(buf, sizeof buf, stdin) != NULL) {
```

```
        if (record_fp != NULL)
                fprintf(record_fp, "< %s", buf);
            if (strcmp(buf, "SYNC\n") == 0)
                break;
        }

        return;
}
```

Appendix 6: MidasPlus Program Suite

MidasPlus has several utilities for preparing surface files, building templates and correcting common PDB file problems. These utilities are typically run directly from the UNIX shell and *not* from the MIDAS command prompt. In addition, the recent addition of the MIDAS commands **run**, **pdbrun**, and **delegate** have made it relatively easy to extend MIDAS functionality with auxiliary programs. Several such programs have been contributed to the MidasPlus distribution and are described in this section. These support programs are typically invoked from within MIDAS. The manual pages for MIDAS and all associated utility and support programs are included here and the table below summarizes the function of each program.

MidasPlus Program Suite

Program Name	Function
bs*	create printable PostScript ball-and-stick representation of molecule
conic	generate CPK-style molecular models with shadows
discern*	visualize multidimensional data
density*	display X-PLOR electron density maps
dnacheck*	correct nonstandard PDB files of nucleotide structures
dms	calculate a solvent-accessible molecular surface
esp	calculate electrostatic potential
fixatname	correct AMBER pseudo-PDB files so they are in standard PDB format
gd*	display GRID-generated energy contours
gennuc*	generate ideal DNA/RNA structures from sequence data
gentpl	generate a MIDAS template from a Protein Data Bank coordinate file
ilabel	label an SGI image with arbitrary text
itops	convert SGI image or TIFF to color PostScript
ksdssp	generate HELIX and SHEET records for PDB files lacking them
label3d*	create labels with correct depth for stereo or *neon*
longbond*	break excessively long bonds
midas	MidasPlus molecular interactive display program
midas.tty	terminal-based version of MidasPlus display program
mrotate*	generate script to bring model(s) into specific orientation
neon*	generate a molecular model with solid stick bonds and shadows
noeshow*	display NMR-derived distance and torsion constraints
pdb2group*	convert PDB file to **addgrp**-style group description file
pdb2site*	convert PDB file to **dms** site file
pdbopen*	browse and/or open available PDB files
rainbow*	smoothly color molecule chain from red to blue
ribbonjr	generate ribbon representation of molecules
run2ses*	convert PDBRUN-format file to MIDAS session file
stereoimg*	create *conic*, *neon*, or *ribbon* images in side-by-side stereo
uncryst*	generate crystallographic symmetry units from CRYST records
unmtrix	expand MTRIX records in Protein Data Bank coordinate file
viewdock*	facilitate use of DOCK-generated bound ligand structures

* user-contributed software, see below

Each program description contains a synopsis line indicating the correct usage of the command. The usage includes the command name in **boldface** type followed by command line parameters. Command line parameters enclosed in brackets are optional, and the brackets should be omitted if you use such a parameter. These command line parameters appear in:

boldface print — indicating a flag. The flag is usually a single character and is preceded by a ‘‘–’’ character and is typed as is.

italic print	indicating a parameter for which the user substitutes the appropriate name, digit, *etc*.

User-Contributed Software

Programs marked with asterisks in the preceding table are user-contributed software which has been incorporated into the MidasPlus distribution. Such programs are ‘‘unsupported’’ in the sense that bug reports and suggested enhancements for these programs will not receive the same priority as for core parts of the MidasPlus distribution. This is not to say that bug reports or enhancement requests for user-contributed software will be ignored, just that since the source code may not have been written at UCSF, the time required to determine how to change the code may make such a change impractical.

Nonetheless, user-contributed software has extended MidasPlus functionality in many significant ways, and we encourage you to contribute any software developed at your site. Contributions for MidasPlus distribution must include:

- All source code necessary to compile your program along with instructions as to how to perform the compilation (for example, a ‘‘README’’ and ‘‘Makefile’’).
- Full documentation of the use of the program, preferably in Unix manual page format (the source for the **discern** manual page is in /usr/local/midas/src/contrib/discern/discern.1 as an example).
- A demo that can be run to verify that the software is working correctly when we receive it.

To notify us of your desire to contribute software, either send electronic mail to **midas-ideas@cgl.ucsf.edu** or regular mail to:

MidasPlus Distribution Coordinator
Computer Graphics Laboratory
University of California at San Francisco
San Francisco, California 94143-0446

MidasPlus User's Manual

Table of Contents

NAME

bs – generate ball-and-stick style diagram in PostScript

SYNOPSIS

bs [**–b** *ball_fraction*] [**–e**] [**–f**] [**–m** *margin*] [**–s** *stick_fraction*] [**–F** *fill_type*] [*PDBRUN-file*]

DESCRIPTION

Bs takes a PDB file as output by the *midas* **pdbrun** command (with the *conect* keyword argument), and generates a ball-and-stick image of the atoms and bonds using PostScript. The output of *bs* may be sent directly to a PostScript printer. Alternatively, *bs* can generate Encapsulated PostScript files that may be incorporated into documents.

Bs interprets not only the standard PDB format records, but also ''USER'' records of the appropriate format. Any USER record that begins with the words EYEPOS, WINDOW, CHAIN, COLOR, or RADIUS is interpreted specially. For details of this format, see the **pdbrun** format description in the Midas User's Manual.

COMMAND LINE FLAGS

The command line flags interpreted by *bs* are:

–b *ball_fraction*
Set the radius of the rendered circle to be *ball_fraction* of the radius of the atom. The default value is 0.25.

–e Generate Encapsulated PostScript. Normally the image is automatically scaled to fit an 8.5x11 page with one-inch borders. When this flag is given, the image is scaled to fit in a 3-inch square.

–f Put a frame around the displayed image.

–m *margin*
Set the margin between the image and its bounding box. This margin is only present when the **–e** flag is absent. The margin is given in PostScript units (*i.e.*, 72 units per inch), and is 2 by default.

–s *stick_fraction*
Set the radius of the rendered stick to be *stick_fraction* of the radius of the rendered circles. The default value is 0.5. *Stick_fraction* must be between 0.0 and 1.0 (exclusive).

–F *fill_type*
Normally, the circles and sticks drawn by *bs* are figure outlines. If the *fill_type* string contains an 'a' or 'A,' the atoms will be drawn as filled circles. If the *fill_type* string contains a 'b' or 'B', the bonds will be drawn as filled sticks.

EXAMPLES

From MIDAS, the command to print a *bs* figure with a surrounding frame would be:

pdbrun conect bs –f **|** lpr -P*printername*

The command to save an Encapsulated PostScript *bs* image to a file would be:

pdbrun conect bs –e > *savefilename*

BUGS

Bs requires that USER records specifying EYEPOS and WINDOW be present in the input file. These records are automatically supplied by *midas*(1), but are not simple to compute manually.

SEE ALSO

midas(1)

AUTHORS

Conrad Huang
UCSF Computer Graphics Laboratory

NAME

conic – generate CPK-style molecular models with shadows

SYNOPSIS

conic [**–p**] [**–f** *output-format*] [**–s**] [**–a** *mode*] [**–o** *output-file*] [**–x** *pixels-wide*] [**–y** *pixels-high*] [**–c** *config-file*] [**–e** *shell-command*] [**–t**] [**–v**] [**–A**] [**–C**] [**–F**] [**–S** *scale_factor*] [**–W**] [*PDB -file*]

DESCRIPTION

Conic reads a Protein Data Bank file and generates a Corey-Pauling-Koltun style image of the molecule. If no PDB file is specified, standard input is used. There can be an arbitrary number of light sources. Specular highlights, diffuse reflections, and shadows are all computed properly.

‘‘Capturing Screen Images’’ in Part III of the MidasPlus manual discusses saving, converting, and printing *conic* images.

COMMAND-LINE FLAGS

The command-line flags interpreted by *conic* are:

–p Use preview mode. Set the image size to 645x484 and antialias mode to **none** (see below).

–s Invoke *imgview*(1) on the computed image file (SGI or TIFF image formats). This flag is only meaningful when used in conjunction with the **–o** flag or the **output** configuration file option.

–a *mode*
Set the antialias mode. *Mode* is the same as the argument to the **antialias** option in the configuration file (see below).

–o *file* Store the computed image in *file* in the selected image format (see the **–f** flag). The default save format is TIFF. The image is not displayed unless the **–s** flag is also specified.

–x *size* Set the horizontal image size to *size* pixels.

–y *size* Set the vertical image size to *size* pixels.

–e *shell-command*
Execute the shell command when the image has finished drawing and exit when the command is done.

–c *file* Use *file* as the *conic* configuration file.

–f *output-format*
Use the given output format if possible. The supported output formats are: **screen**, **sgi**, **ps** and **tiff**, which are: on the screen, SGI image file, Encapsulated PostScript file and TIFF file formats respectively. The default format is **tiff** if the **–o** flag (see above) is specified, and **screen** otherwise. Also, if you use the **ps** format, see the **–H** option.

–t Make the background color transparent. This only works when writing output to SGI and TIFF image files, and it adds an alpha channel to the file so that the resulting image can be composited onto other backgrounds.

–v Print progress messages.

–A Ignore `USER COLOR`, `USER RADIUS`, and `USER MATPROP` records present in the input file. Since the MIDAS **pdbrun** command provides `USER COLOR` and `USER RADIUS` records for each atom, this flag must be used if atomic information is coming from MIDAS but a color scheme specified in an atom information file is desired (see the COLORING THE MOLECULE section of this manual page).

–C Force *conic* to display full spheres at the near clipping plane. This option affects those spheres whose centers lie across the near clipping plane from the viewer, but whose nearest extent crosses the clipping plane (*conic* always discards spheres whose centers are closer than the clipping plane). By default, *conic* shows the sphere with a portion ‘‘cut away.’’ With this option, the entire sphere is shown. It should be noted that the ‘‘cut away’’ depiction is inaccurate in two regards: the partially cut spheres still cast full shadows, and no part of a sphere is shown if its center is in front of the clipping plane, even though it may extend through the plane. Typically these inaccuracies are not noticeable, but in some situations (such as using point light sources) they may produce odd-looking results.

–F Set the image size to be the full screen.

–H If the output format is **ps**, causes binary data to be hex-encoded. Though raw binary format is more space-efficient, many printers cannot print binary data unless it is hex-encoded. Note that future versions of *conic* may make hex-encoding the default (and the **–H** flag would turn it off).

–S *scale_factor*
Zoom in on the center of the image by the specified factor. The size of the window remains unchanged.

–W Force MIDAS to wait until *conic* has exited before continuing.

NeXT DIFFERENCES

The **–s** option is not supported. Only the TIFF and Encapsulated PostScript file formats are supported. You should place the output in a file whose name ends with **.tiff** or **.eps**, so the Workspace may open it correctly. MIDAS simulates the effects of the **–s** option.

CONFIGURATION FILE

The scene computed by *conic* is described by a list of options in a configuration file. If the configuration file is absent, or the option is omitted, then a default value will be used. Lines beginning with '**#**' are comments and are ignored. All other lines are options, which begin with a keyword and are followed by space-separated values. The available options are listed below.

ambient *r g b*
Set the ambient light to the given RGB value, which is three floating-point intensities ranging from 0 to 1. The default ambient lighting is (0.2 0.2 0.2).

antialias *mode*
Set the antialiasing algorithm. *Mode* may be **none**, for no antialiasing; **3/2**, for mapping 3x3 calculation pixels onto 2x2 image pixels; or **2x2**, for mapping 2x2 calculation pixels onto single image pixels. Antialiasing improves the picture quality at the expense of computation time. The time increase is proportional to the number of pixels computed modulo the startup time. Thus, for small molecules, which have low startup times, going from mode **none** to **2x2** will increase the computation time four-fold. The relative increase is less for large molecules since the startup time for large molecules is a significant fraction of total computation time. The default antialias mode is **none**.

atinfo *file*
Use the given file as the *atom information* file, which contains default information on how each type of atom should be colored. Coloring the molecule is described in greater detail below. This option has no effect if *conic* is invoked from within MIDAS, as MIDAS fully specifies atom colors and radii.

background *r g b* [*r g b* [*r g b*]]
Set the background color for the image. If only one RGB value is given, then the entire background is set in that color. If two RGB values are given, then the background is interpolated between the two colors from bottom to top. If three RGB values are specified, then the background is smoothly interpolated from the first color at the bottom of the image to the second color in the middle to the third color at the top. The default background color is (0 0 0). NOTE: if this option is given in the configuration file, it will override any color specified in the input PDB file.

cone *x y z r g b dx dy dz angle*
Define a cone light. The absolute Cartesian coordinates of the light source are (*x y z*). The color of the light is given by (*r g b*). The Cartesian direction of the cone light is given by (*dx dy dz*), and the half-angle of the cone is *angle* degrees.

eye *r g b*

Conic places an additional point light source which coincides with the eye position. The purpose of this light source is to weakly illuminate shadowed areas so that they have discernible features rather than a uniform color. The **eye** option sets the color of the point light source. The default value is (0.3 0.3 0.3).

format *image-format*
Use the given image format if possible (see the **–f** option above).

fov *angle*
Sets the field-of-view half-angle, in degrees. The default value is 15 degrees.

input *file*
Use *file* as the Protein Data Bank file.

light *x y z r g b*
Add an infinite light source to the scene being computed. The direction of the light source is specified by (*x y z*). The color of the light source is specified by (*r g b*). By default, *conic* defines a light source with direction (1 1 1) and color (1 1 1). The default light source is removed if other sources are specified via the **light** option.

location *x y*
Sets the image location on the screen. This parameter is only meaningful if the output is to the screen, *i.e.*, for output format **screen**.

ls_flags *flags*
Change the default flags of subsequently specified light sources. By default, a light source only shines on a point if there are no intervening spheres. If the **noshadow** flag is specified, however, all points are considered to be lit. The **shadow** flag will undo the effects of a **noshadow** flag for subsequent light sources. The **noshadow** flag is generally used if the scene is very complex, and having shadows makes the resulting image difficult to interpret. This problem may also be mitigated by using multiple light sources.

matprop *kd ks power*
Define default material properties. *Kd* is the diffuse reflection coefficient. *Ks* is the specular reflection coefficient. *Power* controls how sharply defined a specular light is, and must be a positive even integer. The higher the value of *power*, the smaller the specular reflection area. The default values are 0.5, 0.25, and 8, respectively. *Kd* and *ks* must be in the range 0–1, and *power* must be 2 or higher.

output *file*
Store the computed image in *file* in the selected image format (see the **–f** and **–o** options above).

point *x y z r g b*
Define a point light source. The arguments are the same as those for the **light** option, except that (*x y z*) defines the light position rather than direction.

quad *x1 y1 z1 x2 y2 z2 x3 y3 z3*
Define a quadrilateral (actually a parallelogram) in the image. Since the quad is a parallelogram, only three vertices are necessary to define it. Quads are ‘‘second-class’’ objects: They can be in shadow from first-class objects (spheres), but cannot cast shadows themselves. In fact, they cannot even block first-class objects; first-class objects show through. Quads are typically used to construct large background areas that show the shadow of the scene as a whole. For best results, there are two important things to note. First, the default field-of-view for *conic* is quite narrow, and if you do not see an expected shadow it may be because it is falling outside the field of view. In such a case, you may want to expand the field of view half-angle (using the **fov** keyword, see above) from the default 15 degrees to 25 or 30 degrees. The second thing to note is that, because of ambient light, shadows will not be black. If you desire black shadows, turn off ambient lighting (using the **ambient** keyword, above).

quad_color *r1 g1 b1* [*r2 g2 b2 r3 g3 b3 r4 g4 b4*]

Define the vertex colors of following quads. The interior color of the quad will be smoothly interpolated between the vertex colors. If only one RGB triple is specified, all vertices will have that color.

rcone *x y z r g b dx dy dz angle*
Rcone is to **cone** as **rpoint** is to **point.**

rpoint *x y z r g b*
Define a point light source relative to the scene, similar to **point.** The (*x y z*) coordinate is relative to the center of the scene, with lengths normalized such that the distance from the eye to the center of the scene is 1. Thus, the option

rpoint 0 0 1 1 1 1

would define a point light source that coincided with the eye, whereas

rpoint 0 2 0 1 1 1

would define a point light source directly above the center of the scene, twice as far above the scene as the distance from the center of the scene to the eye.

rquad *x1 y1 z1 x2 y2 z2 x3 y3 z3*
Rquad is to **quad** as **rpoint** is to **point.**

rspot *x y z r g b dx dy dz power*
Rspot is to **spot** as **rpoint** is to **point.**

size *x y* Sets the image size. The default image size is 1280x1024. If *conic* is invoked from within MIDAS, then the default image size will be the same as the MIDAS window size.

spot *x y z r g b dx dy dz power*
Define a spotlight. The absolute Cartesian coordinates of the light source are (*x y z*). The color of the light is given by (*r g b*). The Cartesian direction of the spotlight is given by (*dx dy dz*). The intensity of the spotlight drops off as the angle between the spotlight direction and the pixel direction; the rate of decrease is the cosine of the angle raised to the *power*th power. *Power* must be an even integer; odd integers will be incremented silently.

COLORING THE MOLECULE

Conic uses two sources of atom radius and coloring information. If neither source of information yields a radius and color for an atom, then the atom is ignored.

The first source is embedded in the input to *conic*, which is an extended Protein Data Bank format. The format is identical to standard PDB format except that `ATOM` and `HETATM` records may be preceded by `USER` records, whose text field contains a keyword and some values. (The **pdbrun** command of MIDAS generates output of this format.) The keywords that *conic* uses are `COLOR`, `RADIUS`, and `MATPROP`. `COLOR` is followed by three floating-point RGB intensities and a color specification. `RADIUS` is followed by a floating-point number representing the atom radius in angstroms. `MATPROP` is followed by the three parameters to the **matprop** option in the configuration file. Once a `COLOR`, `RADIUS`, or `MATPROP` is given, it applies to all of the succeeding atoms in the file. An example of the extended format follows.

```
USER  COLOR 0.000 1.000 0.000 green
USER  RADIUS   1.800
USER  MATPROP   0.500   0.250  16.000
ATOM      1  C   HIS     1      49.168  26.701  10.916  1.00 16.00
```

If the input fails to specify the color, radius, or material properties of an atom, *conic* uses an *atom information* file to supply missing values. The file contains comment lines, which begin with '#', and information lines, which have either five or eight fields: atom type, radius, an RGB triple, and optionally three material property values (see the **matprop** keyword in the CONFIGURATION FILE section for the meaning of material property fields and their default values). The atom type is either one or two characters and is used to match the atom type in the PDB input. The atom type '*' is a special case and matches any atom which does not match any other information lines. Using an *atom information* file, simple color-by-type images may be generated from raw PDB files.

The default *atom information* file contains the following lines:

```
C     1.8   0.5   0.5   0.5
N     1.8   0     0     1
O     1.5   1     0     0
S     1.85  1     1     0
H     1.0   1     1     1
P     1.9   1     0.5   0
F     1.35  0     1     0
CL    1.8   0     1     0
BR    1.95  0     1     0
I     2.15  0     1     0
B     1.8   0.5   0     0
FE    0.64  0.5   0     0
CU    1.28  0.5   0     0
ZN    1.38  0.5   0     0
```

EXAMPLES

The demonstration images included on the MidasPlus distribution CD show how to achieve a variety of striking effects and give detailed instructions on how each image was made. If these demonstration images have been installed on your system, they will be found in /usr/local/midas/demos/images. The ‘‘README.index’’ file there has further information. If the demonstration images have not been installed on your system, you need to mount the distribution CD-ROM, and you will find the images in the CD-ROM directory Midas-2.1/demos/images.

BUGS

Light intensity does not attenuate with distance.

SEE ALSO

imgview(1), midas(1)

FILES

/usr/local/midas/resource/conic.atinfo − default atom information file

AUTHORS

Eric F. Pettersen, Conrad Huang, Gregory S. Couch
UCSF Computer Graphics Laboratory

NAME

density – MidasPlus delegate for displaying formatted X-PLOR electron density data

MIDAS COMMAND SYNTAX

`Command:` **delegate start dens density** *mapfile1* [*mapfile2* ...]

DESCRIPTION

Density is a MidasPlus delegate for displaying electron density contour maps generated from X-PLOR *.map* files. The *.map* files must be of the formatted type. When requested, *density* will create a MIDAS object representing the electron density at one or more contour levels, and open it for display as a user-specified model number. *Density* assumes that the structure associated with the electron density is open as model 0, and therefore rotates and translates the density object it creates to align it with model 0. The user can specify the size of the grid to be created and where it is centered.

Density is a program designed to run in conjunction with MIDAS using the MidasPlus **delegate** mechanism. The standard way that *density* is invoked is for the user to start a MIDAS session, and issue the MIDAS command

delegate start dens density *mapfile1* [*mapfile2* ...]

which runs *density* with the specified formatted X-PLOR *.map* files and tells MIDAS to pass commands beginning with **dens** through to *density* for further processing. The commands that *density* can handle are described below.

COMMANDS

When *density* is acting as a delegate for MIDAS (as in the example above), the user can give it commands by typing

dens *density_command*

in the MIDAS command window. The list of commands is given below.

size *grid_units*
: Specifies the size of the grid that is to be contoured. The default size is 20 units. The size of an individual grid unit is determined by X-PLOR at the time the *.map* file is created. It is generally a few percent of the unit cell size.

center *atom*
: Defines the center of the contoured region. The easiest way to specify the central atom is by using the MIDAS pick mechanism described under ‘‘Atom Picking’’ in Part II of the MidasPlus manual.

display *mapfile_prefix model_number contour_level contour_color* [*contour_level contour_color* ...]
: The **display** command generates the electron density contours at the specified contour levels in the specified colors and opens them as model *model_number*. Up to four contour level/color pairs can be specified. *Mapfile_prefix* is one of the X-PLOR map file names specified on the **delegate start** command line, with the *.map* extension removed. Contour maps can be closed using the **˜open** *model_number* command, and can be turned off and on with the [**˜**]**objdisplay** *model_number* command.

Note that *density* only examines the first character of any command keyword, so for instance the command **display** could be shortened to **disp** or just **d** if desired.

EXAMPLES

The following commands would be used to open contour maps at levels ±2.0, colored blue and red, around the iron atom in residue HEM from the X-PLOR file *dens.map* as model 1 in MIDAS.

delegate start dens density dens.map
: Read the data from the X-PLOR map file *dens.map*.

dens center #0:HEM@FE
: Center the contours about the iron atom in the HEM residue.

dens display dens 1 –2.0 blue 2.0 red
: Display contours at levels ±2.0, colored blue and red, in model 1. Since the grid size was

not explicitly specified, it defaults to 20 units.

CAVEATS

The contour maps should always be rotated and translated in synchrony with model 0 (the structure). Should they ever be mistakenly manipulated out of alignment, they can be realigned by using the MIDAS **matrixcopy** command.

Different areas of the protein can be contoured at the same time by specifying a new center and opening additional contour maps into other model numbers.

SEE ALSO

MidasPlus User's Manual

AUTHOR

Christian Schafmeister, UCSF

NAME

discern – MidasPlus delegate for visualizing multidimensional data

MIDAS COMMAND SYNTAX

`Command:` **delegate start disc discern**

DESCRIPTION

Discern is a MidasPlus delegate for qualitatively visualizing three-dimensional points with additional associated values. *Discern* generates MIDAS object files that represent data points as ellipsoids. The center of the ellipsoid represents the Cartesian coordinates of a data point. The color, size, and lengths of the major and minor axes of the ellipsoids represent the additional values. The ellipsoids are displayed using longitudinal and latitudinal lines. By varying the number of these lines, one can get different representations. For example, using 3 longitudinal and 1 latitudinal lines yields trigonal bipyramids, while 5 and 1 yields pentagonal bipyramids. This is most useful for distinguishing data points from different data files (all data points from a given data file are displayed using the same ellipsoid representation).

The first application of *discern* was to display a set of points with both a calculated and an estimated potential. The ellipsoids were colored from blue to red corresponding to the calculated potential values. The sizes and axes lengths of the ellipsoids were proportional to the difference squared and ratio of the calculated and estimated potential values, respectively. This display clearly showed the potential form, and highlighted the areas where the estimated and calculated values differed significantly. In addition, it also showed whether the estimated value was too low or too high. The *discern* commands for setting up this display are given in the EXAMPLES section below.

Discern is a program designed to run in conjunction with MIDAS using the MidasPlus **delegate** mechanism. The standard way that *discern* is invoked is for the user to start a MIDAS session, and issue the MIDAS command

delegate start disc discern

which runs *discern* and tells MIDAS to pass commands beginning with **disc** through to *discern* for further processing. The commands that *discern* can handle are described below.

COMMANDS

When *discern* is acting as a delegate for MIDAS (as above), the user can give it commands by typing

disc *discern_command*

in the MIDAS command window. The list of commands is given below. Other than **open** and **use**, all *discern* commands are applied to a single selected model (there can only be one selected model at any given time).

open [*model_number*] *datafile_name*

Open the file *datafile_name* and display it in MIDAS as model *model_number* . If *model_number* is absent, use the lowest available model number. This command also makes the newly opened model the selected model.

use *model_number*

Make model *model_number* the selected model.

symbol [**nside** *ns*] [**nlayer** *nl*] [**aspect** *na*]

Redisplay the selected model using ellipsoids with *ns* longitudinal lines, *nl* latitudinal lines, with the ellipsoid compressed length-to-width by a factor of *na*. Any of the three specifications may be omitted, in which case the value will remain unchanged. For a newly opened object, the default values for *ns*, *nl*, and *na* are 6, 1 and 1.0 respectively.

property *name* [*formula*]

Define or list a property for use with the *size*, *major*, *minor*, and *color* commands. If *formula* is omitted, the formula for computing property *name* is listed; otherwise, the formula will be used for computing property *name* (any previously defined formula is discarded). The formula consists of data columns, the usual arithmetic operators, and simple function calls. Data columns are introduced by the ‘‘$’’ character, followed by the column number (columns are numbered starting from one, not zero). Arithmetic operators include ‘‘+,’’ ‘‘-,’’ ‘‘/,’’ ‘‘*,’’ and ‘‘^’’ which correspond to addition, subtraction, division, multiplication, and exponentiation respectively. The precedence of the operators is the normal arithmetic one, and parentheses may be used to override. The list of supported functions is: **sqrt** (square root), **sin** (sine), **cos** (cosine), **acos** (arccosine), **exp** (exponential), **log** (natural logarithm), and **abs** (absolute value). An example of a **property** command is

property diffsq ($4 – $5) ^ 2

Properties with numeric names are predefined to correspond to data columns (*e.g.*, ‘‘1’’ corresponds to data column 1).

size *keyword-value-pair ...*
major *keyword-value-pair ...*
minor *keyword-value-pair ...*
color *keyword-value-pair ...*

These commands redraw the selected model using different display parameters based on the *keyword-value-pair* arguments. **Major** and **minor** are the axes lengths of the ellipsoid; **size** is a scale factor applied to both axes; **color** is the color of the ellipsoid. The supported keywords are listed below.

property *name*
Specifies that the display parameter will be interpolated using values computed with property *name*. In the special case where *name* is **none**, the default display parameter will be used.

bmin *value*
Specifies the lower bound for the data values. If this keyword is not used, the minimum value for the specified property is used. Data points whose value is below the given lower bound are not displayed.

bmax *value*
Specifies the upper bound for the data values. If this keyword is not used, the maximum value for the specified property is used. Data points whose value is above the given upper bound are not displayed.

min *param*
Specifies the display parameter associated with the lower bound of the data values. For the **color** command, the value is any string corresponding to a MIDAS color; for other commands, the value is a floating point number.

max *param*
Specifies the display parameter associated with the upper bound of the data values. For the **color** command, the value is any string corresponding to a MIDAS color; for other commands, the value is a floating point number.

state Report the current state associated with the display parameter. If no property has been associated with the parameter, the **property** field is reported as **none**.

The display parameter associated with a data point is linearly interpolated using the following formulae:

fraction = (data_value – lower_bound) / (upper_bound – lower_bound)
parameter = min_param + fraction ∗ (max_param – min_param)

Note that *min_param* may be larger than *max_param*, in which case the display parameter is inversely proportional to the data value. The ellipsoid axes’ lengths are computed in angstroms; size is dimensionless; and color is interpolated in hue-saturation-value (HSV) space by MIDAS. For a newly opened object, the display parameters are 1 (for **size**, **major**, and **minor**) and white (for **color**), unless overridden in the data file.

axes [**x** *xname*] [**y** *yname*] [**z** *zname*]
Define the property values to use for Cartesian coordinates. Any of the axis

specifications may be omitted, in which case the previously defined property will be used. For a newly opened object, the *xname*, *yname*, and *zname* properties are ‘‘1,’’ ‘‘2,’’ and ‘‘3’’ respectively (*i.e.*, the first three columns are the Cartesian coordinates).

direction [**x** *xname*] [**y** *yname*] [**z** *zname*]
Define the property values to use for the direction of ellipsoid major axes. For a newly opened object, if the data file contains the **directional** property (see below), the *xname*, *yname*, and *zname* properties are ‘‘4,’’ ‘‘5,’’ and ‘‘6’’ respectively (*i.e.*, the three columns after the Cartesian coordinates are the direction vector). If the data file does not contain the **directional** property, then all ellipsoids will be oriented with their major axes parallel to the z axis. Any of the axis specifications may be omitted, in which case the previously defined property is used. If no property has been defined for any of the three axes, the ellipsoids will remain parallel to the z axis.

FILE FORMAT

The format of the data file that *discern* will read consists of a series of property descriptions followed by the data values. The following property descriptions are mandatory:

field *number*
Each data point has *number* values associated with it. This includes the three Cartesian coordinates, so the minimum value for *number* is 3.

data This must be the last property description. Subsequent lines in the file are treated as data points.

The following property descriptions are optional:

count *number*
There are approximately *number* data points. This value is only advisory, and *discern* will only use it to guide memory allocation strategy.

nside *number*
Ellipsoids will be drawn with *number* longitudinal lines. The default value of this property description is 5. It must be at least 3.

nlayer *number*
Ellipsoids will be drawn with *number* latitudinal lines. The default value of this property description is 1. It must be at least 1 and odd.

size *number*
If the user does not issue a **size** command, ellipsoid axes lengths will be scaled by a factor of *number*. The default value of this property description is 1.

directional
Normally, ellipsoids are drawn with the major axis parallel to the z axis. This property description specifies that for each data point, the three columns following the initial Cartesian coordinates form a direction vector for the major axis of the ellipsoid.

Each line following the property descriptions is treated as a single data point. Each data point is represented as a list of floating point numbers. The first three items in the list form the initial Cartesian coordinates for the data point (this may be changed using the **axes** command [see above]). There must be the same number of elements in the list as the number specified by the **field** property description.

EXAMPLES

The following is a sample data file:

```
field 7
size  0.2
nlayer      3
directional
data
```

```
0 0 0 1.0 2.0 1.0  2.0
1 0 0 5.0 4.0 1.0  0.8
0 1 0 2.5 3.0 0.25 1.2
```

The following are MIDAS commands that may be used to display the data file as described in the DESCRIPTION section above.

disc open discern.data
: Open the data file and display it.

disc color property 4 **min** blue **max** red
: Color the data points based on column 4, and make the minimum value correspond to blue and maximum to red. Intermediate values will map to colors ranging from blue to cyan through magenta to red.

disc size property 5 **min** 0.1 **max** 0.5
: Make the ellipsoid size correspond to column 5, and map the minimum value to 0.1 angstrom and maximum to 0.5 angstrom.

BUGS

There is no way to change the direction vector from columns 4, 5, and 6.

SEE ALSO

MidasPlus User’s Manual

AUTHOR

Conrad Huang, Computer Graphics Laboratory, UCSF
Chris Bayly, UCSF

NAME

dms – calculate a solvent-accessible molecular surface

SYNOPSIS

dms *file* [**–a**] [**–d** *density*] [**–g** *file*] [**–i** *file*] [**–n**] [**–w** *radius*] [**–v**] **–o** *file*

DESCRIPTION

Dms calculates the molecular surface of a molecule. The molecular surface resembles the van der Waals surface of a molecule, except that crevices between atoms are smoothed over and interstices too small to accommodate the probe are eliminated. The surface includes cavities in the interior of the molecule, even if they are not accessible to a solvent molecule coming from the outside.

The molecular surface calculated is that defined by F. M. Richards (1977, *Ann. Rev. Biophys. Bioeng.*). In particular, the calculated molecular surface is that traced out by the *surface* of the probe sphere rather that the probe sphere's *center*. According to Richards' definition the molecular surface consists of two parts: *contact surface* and *reentrant surface.* The contact surface is made up of ''those parts of the molecular van der Waals surface that can actually be in contact with the surface of the probe.'' The reentrant surface is defined by ''the interior-facing part of the probe when it is simultaneously in contact with more than one atom.'' *Dms* reports the amounts of contact and reentrant surface area, and the combined total surface area on the standard error output (see the **–g** flag below).

File is an input file of coordinates. The input file must be in the Protein Data Bank format. The first letter or first two letters of the atom name is used to determine the element type. By default, implicit hydrogens are included for carbon, nitrogen and oxygen atoms, thus aromatic carbons and nitrogens will have van der Waals radii that are somewhat too big. Note that only amino acid and nucleic acid residues will be included unless **–a** is also specified.

Dms can be set up to run on multiple machines simultaneously for increased performance. By default, it only runs on the local host. The UCSF MidasPlus Installation Guide that came with the MidasPlus CD-ROM contains instructions on how to configure *dms* to use multiple machines.

If it is desired to simply visualize a small molecular surface from within MIDAS, it may be easier to use the *makems*(1) delegate, rather than run *dms* directly. Consult the *makems* manual page for further details.

OPTIONS

The flags may be in any order. The meanings of the flags are described below:

–a Include all atoms, not just those in amino acid and nucleic acid residues.

–d Change the density of points on the surface. *Density* is a factor affecting the density of points on the surface; the default of 1.0 produces about 5 points per square angstrom. Only values between 0.1 and 10.0 are permitted. For large proteins, a density of 0.5 is recommended.

–g Write all the informative messages to *file,* instead of the standard error output. Genuine errors still go to the standard error output. This file is not rewound at any time, so messages from several runs may be accumulated.

–i Calculate the molecular surface only for those residues and atoms specified in *file,* but keeping the rest of the molecule for collision checks. The file consists of a series of lines such as the following:

```
ASP  205 CA
TYR   13 *
GLY  116 FRM
HIS  178 TO
```

The asterisk means all atoms of the residue and the ''FRM'' and ''TO'' mean all residues from 116 to 178 inclusive. The sequence number may contain letters, and if the PDB input file contains chain identifiers, then those should be appended on the right of the sequence number. Residue insertion codes (if any) should be placed between the sequence number and any chain identifier. Residues contained in `HETATM` records should have an asterisk appended to the end of the residue identifier. The surface generated using the **–i** flag is not always the same as the surface generated by running the entire molecule and afterwards selecting out the desired atoms. The first surface will not include reentrant surface lying between an atom in the **–i** file and atoms not in the file. The *pdb2site*(1) utility may be useful for generating site files. Consult the *pdb2site* manual page for further details.

–n Include the unit normals to the surface with each surface point record.

–v Produce more verbose output. *Dms* will announce each computation phase as it is entered as well as a count of the atom types in the molecule and the number of computation requests handled by each host that participated in the *dms* calculation.

–o The output is written to *file.* This flag is not optional.

–w Change the water probe radius from the default radius of 1.4 angstroms. This parameter must be between 1.0 and 201.0.

The output consists of a series of atom and surface point records, with the same format for the first six fields. Each atom is followed by the surface points (if any) which belong to it. These first six fields are in the following format: residue name, sequence number, atom name, x coordinate, y coordinate, z coordinate. For an atom record, the seventh field is ‘‘A.’’ For a surface point record, the seventh field begins with an ‘‘S,’’ followed by a ‘‘C,’’ ‘‘R,’’ or ‘‘S’’ according to whether the point is part of contact, reentrant, or ‘‘saddle’’ surface (‘‘saddle’’ is a type of reentrant surface where the probe is in contact with exactly two atoms). This is followed a digit used for depicting different density levels. The eighth field is the molecular surface area associated with the point in square angstroms. If the **–n** flag is specified, the next three fields are the unit normal vector pointing outward from the surface. Informative messages and errors are written to the standard error output unless a **–g** file is specified.

The chemical elements and radii that the program handles are detailed in the table below. The program gets these values from the file */usr/local/midas/resource/dms/radii*. If there is a file in the current directory called *radii*, then *dms* will use that file instead. So in order to add uncommon elements or use different radii, one should copy the default file and modify it. The file format is documented in the file itself.

Element	Radius
H	1.20
C	1.90
N	1.50
O	1.40
F	1.35
P	1.90
S	1.85
Cl	1.8
Fe	0.64
Cu	1.28
Zn	1.38
Br	1.95
I	2.15
Other	1.90

SEE ALSO

pdb2site(1), The UCSF MidasPlus Installation Guide

AUTHOR

Conrad Huang
University of California, San Francisco

FILES

/usr/local/midas/resource/dms/radii   default atomic radii

DIAGNOSTICS

Many and varied. Be sure to examine the **–g** file before you leave a background job running overnight.

NAME

dnacheck – regenerate DNA files to match Protein Data Bank specifications

SYNOPSIS

dnacheck [–c *file*] [**–h**] [**–r**] [**–A**] [**–C** *directory*] [**–D**] [**–C**] [**–I**] [*PDB_file* [*output_file*]]

DESCRIPTION

Dnacheck reads a Protein Data Bank (PDB) file containing DNA atomic coordinates and creates a file that matches the Protein Data Bank format for nucleotides. *Dnacheck* corrects three types of problems:

(1) Residue and atom name mapping. PDB specifies a set of residue and atom names for nucleotides. Frequently, data files not from PDB use slightly different naming conventions. *Dnacheck* knows about some of these conventions and will convert names to the PDB standard set.

(2) AMBER output. AMBER's older force field (Weiner *et al.*) models nucleotides as separate phosphate and sugar-base residues. *Dnacheck* recombines the sugar-base residues with the phosphate residues to form complete PDB nucleotide residues.

(3) Reversed direction. PDB format requires that DNA strands be specified starting from the 5' end. Some files have strands that start from the 3' end. *Dnacheck* reverses the order of the residues in these strands.

Dnacheck does *not*, unfortunately, correct RNA file deficiencies.

CONFIGURATION FILES AND BLUEPRINTS

Dnacheck works in four steps:

(A) reads a configuration file describing a list of blueprint files and some naming conventions associated with the blueprints, and then reads the blueprints;

(B) applies the blueprints to the input file to take care of type (1) problems as described in the DESCRIPTION section above;

(C) applies simple heuristics to take care of type (2) problems;

(D) takes care of type (3) problems by applying simple heuristics to the first two residues of a chain to determine whether strand-reversal is necessary.

Blueprints are simply PDB-format files each containing the ATOM or HETATM records of a single residue, followed by CONECT records. The CONECT records must be present, and may be generated using the **pdbrun** command in MIDAS.

The format of the configuration file is most easily explained by example. The following is part of the default configuration file:

```
blueprint T
      synonym THY THE
      alias C5M C7 C5A
      alias O4* O1*
      alias O1P OA
      alias O2P OB
```

The first line states that there is a blueprint that should be applied to residues named ''T.'' (By convention, the file name of the blueprint is the same as the name of the residue to which it applies.) The second line states that this blueprint should also be applied to residues named ''THY'' or ''THE.'' The third line states that atoms named ''C7'' or ''C5A'' should be renamed ''C5M.'' The fourth through sixth lines add similar name translations.

The configuration file consists of a series of these blueprint descriptions. Residues in the input file matching a blueprint are altered to have the same residue type as the blueprint name, atoms in the residues are translated if they match one of the aliases, and atom record types (ATOM *vs.* HETATM) are modified to match those in the blueprint. If there are residues in the input file that do not match any blueprints, they are left unmodified.

COMMAND OPTIONS

–c *file* Specify the name of the configuration file. The default configuration file that *dnacheck* uses is *config*. The configuration file must be in either the current directory or the default directory (see **–C** below).

–h Convert any residue which is not connected to other residues to be a ‘‘hetero-residue’’ (*i.e.*, the PDB records for the atoms of the residue are of type HETATM instead of ATOM). Normally, *dnacheck* retains the record type of atoms from the input file. This option is most useful when there are many unconnected residues, such as waters, which are of type ATOM in the input file, but should actually be of type HETATM.

–r Renumber the sequence number of residues. Normally, *dnacheck* retains the sequence number, insertion code, and chain identifier of residues from the input file. This option makes *dnacheck* renumber the residues, making all insertion codes the space character. Hetero-residues are given consecutive sequence numbers starting from 1, with chain identifiers set to the space character. Non-hetero-residues are split into chains, with residues in each chain given consecutive sequence numbers starting from 1. If there is only one chain in the file, the chain identifier is set to the space character; otherwise, the chain identifiers are set to consecutive alphabetic characters starting with ‘‘A.’’

–A Do not fix type (2) problems described in the DESCRIPTION section above (*i.e.*, skip step (C)).

–C *directory*
Set the default directory where *dnacheck* searches for configuration files and blueprints. Normally, *dnacheck* looks for configurations files and blueprints first in the current directory, then in the default directory */usr/local/midas/resource/dnacheck*. Even with this option set, *dnacheck* will still search the current directory first.

–D Do not fix type (3) problems described in the DESCRIPTION section above (*i.e.*, skip step (D)).

–I Do not fix type (1) problems described in the DESCRIPTION section above (*i.e.*, skip step (B)).

PDB_file
The input Protein Data Bank (PDB) file may contain any legal PDB records. Only ATOM records will be used. All others are silently discarded. If no *PDB_file* argument is given or is ‘‘–,’’ the data is read from standard input.

output_file
The output of *dnacheck* is a set of PDB format records. If no *output_file* argument is given or is ‘‘–,’’ the records are written to standard output.

LIMITATIONS

Only DNA structures are handled; RNA structures are not. If someone were to develop RNA blueprints (and test them with the **–C** option), we would be willing to redistribute them on the MidasPlus web site.

SEE ALSO

UCSF MidasPlus User’s Manual

AUTHORS

Conrad Huang
UCSF Computer Graphics Laboratory

NAME

esp – calculate electrostatic potential

SYNOPSIS

esp –i *dms_file* [**–o** *esp_file*] [**–q** *file*] **–a** *pdb_file* [**–r**] [**–n**] [**–c** *cutoff*] [**–e** *epsilon*] [**–p** *len*] [**–v**] [**–w**]

DESCRIPTION

Esp calculates the electrostatic potential of a solvent-accessible surface and stores it in an annotated *ms* surface file. This information can then be used by MIDAS to selectively color molecular surfaces based on electrostatic potential. *Esp* prints out a summary of the conditions used to calculate the potential.

–i *dms_file*	specifies a *dms* surface input file. The electrostatic potential is calculated for all points of this surface. Use the *dms(1)* program with the **–n** flag (to calculate normals) to generate *dms_file*. If the **–p** flag (see below) is given a value of 0, the normals need not be calculated by *dms*(1).
–o *esp_file*	indicates the annotated *dms* surface file to which the calculated electrostatic surface is output.
–q *charge_file*	The option **–q** *charge_file* supplies an alternate charge file for residue types. The default file used is */usr/local/midas/resource/charges.esp*. Instructions for constructing alternate charge values are contained in the default file. The **–q** flag must *precede* the name of the Protein Data Bank format file (**–a** flag) to which the alternate charge file is applied. Thus, a series of command line parameters, **–q** *file1* **–a** *db1* **–q** *file2* **–a** *db2* may be used to associate alternate charge files with specific models.
–a *pdb_file*	is the name of the Protein Data Bank format file containing the coordinates for the associated *dms* surface file and any other atoms which should be included in the electrostatic calculation. The potential is calculated *only* for those surface points in *dms_file*, but *all* atoms in *pdb_file* are used in the calculation.
–r	indicates that the dielectric constant is dependent on the distance from the atom or charge to each surface point.
–n	specifies neutral spheres. Charges are summed within the *cutoff* radius defining the sphere, and an equal and opposite charge is spread uniformly across the sphere surface.
–c *cutoff*	indicates the cutoff radius in angstroms. The default value is 10.0.
–e *epsilon*	*Epsilon* is the dielectric constant, or if the **–r** flag is specified, *epsilon* is the factor that is multiplied by the distance in angstroms to give the effective dielectric. The default value is 1.0.
–p *len*	calculates the potential at a distance *len* angstroms from the surface. Positive values of *len* lie outside of the surface, while negative values lie within the surface. The default value of *len* is 1.4 angstroms.
–w	indicates that the electrostatic potential of each point should be appended to the corresponding line in the input *dms_file*.
–v	indicates verbose mode. Auxiliary information (*e.g.*, the number of points found for each atom) is reported.

SEE ALSO

dms(1), midas(1) MidasPlus User's Manual

AUTHOR

Conrad Huang
UCSF Computer Graphics Laboratory

The idea of neutral spheres came from Paul Weiner while he was a graduate student in the Department of Pharmaceutical Chemistry, UCSF.

NAME

expr-smpte — evaluate SMPTE expressions

SYNOPSIS

expr-smpte [**−d**] *time-code1 operation time-code2*

DESCRIPTION

Expr-smpte computes the result of the given SMPTE time code binary expression. SMPTE time codes are given in a HH:MM:SS:FF format, where HH is the number of hours, MM is the number of minutes (00−59), SS is the number of seconds (00−59), and FF is the frame number (00−29). Leading zeros after a colon are required. The time codes may be abbreviated by omitting earlier parts, *i.e.*, **12** would be frame 12, and **2:05** would be frame 5 in the third second. Relative time codes may have a negative sign prefix.

The **−d** option turns on support for the Drop Frame Time Code. The SMPTE standard time code drops frames numbered :00 and :01 each minute, except for every 10th minute, to compensate for NTSC variance from real-clock time. This option is not implemented yet.

SEE ALSO

videodisk(1)

AUTHOR

Greg Couch

NAME

fade – fade from one SGI image to another

SYNOPSIS

fade [–**l** *loop*] [–**s** *steps*] [–**p** *pause*] [–**e** *shell_command*] *from_image to_image*

DESCRIPTION

Fade reads two image files, displays the first image, and fades to the second image. Both image files must be in SGI Image Library RGB format, and must be identical in size. If the –**l** argument is given, *fade* will alternately fade from one image to the other for *loop* times. The default value of *loop* is one. The –**s** argument specified the total number of frames displayed during the fade, including the starting and ending frames. As *steps* increases, the fade becomes smoother, but takes longer. The default value of *steps* is 40. The –**p** flag specifies the number of clock ticks to pause after a fade has completed, before proceeding to the next fade or exiting. The number of clock ticks per second is defined by the variable **CLK_TCK** in file <limits.h>. The default value of *pause* is zero. The –**e** argument specifies a shell command that is executed at the end of each frame. This is useful if one wished to record individual frame images (*e.g.*, saving to videodisk with the *videodisk*(1) program).

SEE ALSO

videodisk(1)

AUTHOR

Conrad Huang
Computer Graphics Laboratory
University of California, San Francisco

NAME

fixatname – correct AMBER pseudo-PDB files so they are in standard PDB format

SYNOPSIS

fixatname [*file1* ...]

DESCRIPTION

Versions of AMBER prior to version 3.0 revision A produced PDB files that had atom names aligned in the wrong columns (see MidasPlus User's Manual, Part III, *Protein Data Bank Format* for details of PDB format). *Fixatname* corrects this misalignment. AMBER PDB files have their atom names left-adjusted in columns 13-16 of each line. *Fixatname* left-adjusts the atom name in columns 14-16, with the former contents of column 16 being placed in column 13. This results in atom names that conform to the PDB standard with the rather infrequent exception of atom names with a two-character atomic symbol (such as iron, *FE*). Such cases can be corrected by hand after processing by *fixatname*.

Fixatname reads from standard input if no file names are specified. *Fixatname* prints the corrected PDB file on standard output.

BUGS

Atoms with two-character atomic symbols are misaligned, as mentioned above.

SEE ALSO

MidasPlus User's Guide

AUTHOR

Conrad Huang
UCSF Computer Graphics Laboratory

NAME

fixses – update pre-MidasPlus 2.0 session files

SYNOPSIS

fixses *sessionname.ses*

DESCRIPTION

Sessions saved with version of MidasPlus prior to release 2.0 require minor modifications to work correctly with versions 2.0 and later. *Fixses* performs these modifications.

AUTHOR

Greg Couch
UCSF Computer Graphics Laboratory

NAME

gd – a MidasPlus delegate which serves as an interface between MIDAS and Peter Goodford's GRID program [1].

MIDAS COMMAND SYNTAX

`Command:` **delegate start g gd** [**–r** *file_name*]

If the **–r** flag is used, then all communication between *gd* and MIDAS is recorded in file *file_name*.

DESCRIPTION

Gd is a MidasPlus delegate to allow the visualization of three-dimensional energy maps output by Peter Goodford's GRID program [1]. *Gd* contours these maps at user-defined energy levels, which are then displayed by MIDAS along with the molecules under study. *Gd* can also produce GRID input files with correct BOT(XYZ)...TOP(XYZ) coordinates for the molecules (or parts of molecules) of interest.

When *gd* is acting as a delegate for MIDAS, the user can supply commands by typing:

g gd_command [*parameters*]

at the MIDAS command prompt. Many commands need additional parameters, and all parameter values remain set until the user wishes to change them by supplying other values.

COMMANDS

Basically, the commands of *gd* can be classified into two categories: commands used in conjunction with the contouring and display of three-dimensional energy maps output by GRID, and commands used to generate GRID input files after the user has defined which atoms need to be included in the GRID box. The list of commands is given below:

''GRID-CONTOURING'' COMMANDS

These commands all have to do with the contouring and visualization of three-dimensional energy maps output by GRID:

color [*color_spec*]

Color_spec can be simply a single MIDAS color, including user-defined colors (see the **colordef** command in the MidasPlus manual), or two colors and a mixture fraction (separated by commas with no blanks). The mixture fraction is a number between 0 and 1, and indicates the relative amounts of the two colors to use. Zero indicates ''use exclusively the first color,'' while 1 indicates ''use only the second color.'' The default value for *color_spec* is ''gray.'' The color command can be used for coloring the GRID box boundaries (see the **show_box** command), **min** and **max** labels, and energy contours (see the **contour** command).

level [*contour_level*]

Contour_level is the level at which contouring takes place. If the level is outside the range of values found in the GRID file, no contouring takes place. The user can prompt for these maximum and minimum values with the **max** and **min** commands (see below). The default value for *contour_level* is –5.

max [*label* [*model_number*]]

The **max** command returns the maximum energy value found in the GRID file. When a *label* parameter is provided, this label will be displayed as model *model_number* at the position of maximum energy. When no *model_number* is provided, the lowest available model number will be used. No default value is provided for *label*, so plotting will not occur by default.

min [*label* [*model_number*]]

The **min** command returns the minimum energy value found in the GRID file. When a *label* parameter is provided, this label will be displayed as model *model_number* at the position of minimum energy. When no *model_number* is provided, the lowest available model number will be used. No default value is provided for *label*, so plotting will not occur by default.

show_box [*model_number*]

This command displays the boundaries of the GRID box as defined in the GRID file. If no *model_number* is defined, the lowest available model number will be used.

gridkont [*file_name*]
File_name is the name of the unformatted three-dimensional energy file produced by GRID. The default value for *file_name* is ‘‘gridkont.dat.’’

status The **status** command prompts the program for the current values of *color_spec*, *contour_level*, and *file_name*.

contour [*model_number*] [at *level*] [color *color_spec*]
Starts the contouring procedure and displays the computed contours as model *model_number* at contour level *level* using color *color_spec*. If *model_number* is absent, the lowest available model number will be used. If the specified *model_number* already exists, no contours will be drawn. *Level* and *color_spec* can also be specified using the **level** and **color** commands (see above).

‘‘GRIDIN-CREATION’’ COMMANDS

These commands deal with the creation of a GRID input file:

define_grid [*extent* [show [*model_number*]]]
Define_grid calculates the size of a box encompassing all atoms displayed in the viewing window. If the additional parameter *extent* is given, then the box size is increased along all sides by 2∗*extent* angstroms. The default value for *extent* is 0. If the optional keywords **show** [*model_number*] are given, then the grid boundaries will be displayed as model *model_number*. If *model_number* is not supplied in conjunction with the keyword **show**, the lowest available model number will be used to display the box boundaries.

makegrid [*file_name*]
This command writes a template input file for GRID, called *file_name*. If no explicit *file_name* is provided by the user it defaults to ‘‘grid.in.’’ **Makegrid** can only be executed after having defined a grid with **define_grid**.

The following commands set specificd GRID variables to user-defined values, and are used by **makegrid** to generate the GRID input file:

npla [*number_of_planes_per_Angstrom*]
Sets the GRID variable NPLA. Defaults to 1.

probe [*probe_type*]
Sets the type of the probe. Defaults to the water probe ‘‘oh2.’’ See the GRID manual for other possibilities.

KNOWN BUGS

The color of an object (*i.e.* energy contour, min/max label, and GRID box boundaries) has to be defined before the actual object is drawn. Once displayed, there is no way to change its color except by deleting the object (**˜open** *model_number*), redefining its color with **color** *color_spec*, and regenerating the object using the appropriate command.

Dashed lines cannot be produced.

If *gd* is used to show energy contours, it must be running on a machine with the same integer and floating-point binary format as the machine that produced the GRID energy contour data file. Naturally, if the GRID computation was performed on the same machine that *gd* is running on, contouring will work.

AUTHOR

Hans De Winter
Laboratory of Medicinal Chemistry
REGA-Institute of Medical Research
Minderbroedersstraat 10
B-3000 Leuven
Belgium

REFERENCE

[1] Goodford, P.J. ‘‘A computational procedure for determining energetically favorable binding sites on biologically important molecules.’’ (1985) *J. Med. Chem.* **28**, 849-857.

The GRID program can be obtained from Molecular Discovery Ltd, West Way House, Elms Parade, Oxford OX2 9LL, England, or by contacting Peter Goodford himself at `peter@biop.ox.ac.uk`.

NAME

gennuc – Generate double-helical DNA or RNA structures

SYNOPSIS

gennuc –t *structure-type* [*sequence-file*]

DESCRIPTION

Gennuc generates a Protein Data Bank (PDB) formatted file of coordinates on standard output representing double-helical DNA or RNA structures built from standard base-pair coordinates.

Valid *structure-type* code s are:

brl	right-handed B-DNA in Langridge configuration.
bra	right-handed B-DNA in Arnott configuration.
bls	left-handed B-DNA in Sasisekharan configuration.
ara	right-handed A-DNA in Arnott configuration.
arn	right-handed A-RNA in Arnott configuration.
apr	right-handed A'-RNA in Arnott configuration.

The base-pair input sequence is read from the named file or standard input if no *sequence-file* is given. Valid codes for DNA are A, T, C, and G (for adenine, thymine, cytosine, and guanine). Valid codes for RNA are A, U (uracil), C, and G. Either uppercase or lowercase characters may be used. Invalid codes are noted, but otherwise ignored.

SEE ALSO

Protein Data Bank, ''Atomic Coordinate and Bibliographic Entry Format Description.''

BUGS

Doesn't know about Z-DNA.

AUTHORS

T.E. Ferrin and N. Pattabiraman
Computer Graphics Laboratory
University of California, San Francisco

NAME

gentpl – generate a MIDAS template from a Protein Data Bank coordinate file

SYNOPSIS

gentpl –r *residue* [**–i** *infile*] [**–c** *radiifile*]

DESCRIPTION

Gentpl is a utility program for generating a MIDAS template from a Protein Data Bank coordinate file. Standard input is assumed unless otherwise specified. One file is produced: the ASCII instruction file, *residue*.ins, where ‘‘*residue*’’ is the residue name specified on the command line. These files are placed in the directory defined by the MODELS variable in the user’s program environment (see ‘‘Building and Modifying MIDAS Templates’’ in Part III of the MidasPlus User’s Manual).

–r *residue*	specifies the *residue* name. This name must correspond to the residue name as it appears in the Protein Data Bank input file.
–i *infile*	specifies an input file. The input *must* be in standard Brookhaven Protein Data Bank format. The file may contain data other than the coordinate data for the specified residue, but these extraneous records are ignored.
–c *radiifile*	specifies a file containing the radii of the atoms used to calculate the connectivity. If no file specified, the program uses as default */usr/local/midas/resource/connect.tpl*. The format of the radii file is a series of records containing the atom name followed by the atom radius in angstroms. At least one space must appear between the atom name and the radius.
–v	indicates verbose mode and is useful for tracking down errors.

BUGS

The radii file must be ordered such that in the case of overlapping atom names, longest names appear before shorter ones. For example, if the file contains the radius for both B and BR, BR must appear before B in the file.

SEE ALSO

midas(1)
Protein Data Bank File Record Formats

AUTHOR

Laurie Jarvis
UCSF Computer Graphics Laboratory

NAME

ilabel – label an Silicon Graphics image with arbitrary text

SYNOPSIS

ilabel [**–f**] [**–i** *labels_in_file*] [**–o** *labels_out_file*] *image_file*

DESCRIPTION

ilabel displays the given Silicon Graphics *image_file* and lets the user put labels over the image. The Silicon Graphics image file may be generated by *scrsave*(1), *imgsnap*(1), or some other program, such as *conic*(1L), that uses the Silicon Graphics Image Library. The labels are drawn using the Silicon Graphics GL Font Manager, which supports many fonts in arbitrary sizes. Labels may be saved to a file for reuse via the **–o** flag. The stored attributes of labels include color, vertical and horizontal justification, font, size, and position relative to the image. The saved label files may be displayed again using the **–i** flag. There may be several **–i** files but at most one **–o** file. The output file is created when the user selects **Exit** from the pop-up menu. If the file already exists, the user is asked whether the file should be overwritten (unless the **–f** flag was specified, in which case no question is asked). You can also save labels before exiting by choosing **Save Labels** off the pop-up menu. If you do this, you will be prompted for a file name whether or not you specified **–o** on the command line. You can save a new image with the labels embedded in it by choosing **Save Image** from the pop-up menu. You will be prompted for a file name. Note that if you intend to use the *itops*(1) utility to print the image, it is better to save the labels in a separate file rather than embedded in the image, since the quality of the labels will then be limited by the resolution of the printer rather than the resolution of the image. Labeled images can be converted to printable PostScript files with the *itops*(1) utility provided with the MidasPlus distribution. It is necessary to use the **–L** flag of *itops* in order to have the labels placed on the image.

To add a new label, simply click the left mouse button, which should make a triangular cursor appear, and type in the label. Labels containing multiple lines may be created by typing either **RETURN** or **LINEFEED** at the appropriate place. The left button is also used to select other labels so that they may be edited. The middle button is used to select and move labels. The right button displays a menu which contains options to show and hide the defaults panel (see below), redraw the images and labels, save the current labels in a file, save the current image, turn the mouse cursor on and off, and quit.

The default display attributes for labels are as follows:

font	Times Roman
size	14
color	white
justification	bottom left

All these values may be changed via the defaults panel, which is shown when the user selects the **Show Defaults** option from the right-button menu. When the mouse cursor is over the defaults panel, the left mouse button is used to select the justification mode and label color; the font size may be changed by typing the size and then **RETURN** (no cursor is shown); and new fonts may be selected from the right-button menu. Also, colors may be selected from anywhere on the screen. If the mouse button is depressed over the color selection area, the mouse may be moved anywhere and a color will not be selected until the button is *released.*

SEE ALSO

conic, itops, label3d, snapshot

BUGS

Cannot display arrows.

AUTHOR

Conrad Huang
UCSF Computer Graphics Laboratory

NAME

itops – convert a SGI or TIFF image file to Color PostScript

SYNOPSIS

itops [–**a**] [–**b** *bits-per-color*] [–**g** *gamma-factor*] [–**i**] [–**l**] [–**p** *page-type*] [–**q**] [–**r**] [–**s** *scale-factor*] [–**z**] [–**8**] [–**E**] [–**S**] [–**L** *label-file*] *image-file*

DESCRIPTION

itops converts an SGI RGB format or TIFF format image file to Color PostScript. The output appears on standard output and would normally be piped directly to the *lpr*(1) or *lp*(1) command.

OPTIONS

The –**l** option lists all known page types.

The –**p** option sets the page type; the default is ‘‘Letter.’’

The –**r** option rotates the image so it is in landscape mode instead of portrait mode.

The –**g** option gamma-corrects the image data by the given amount. Depending on the printer, this value should range from from 0.9 to 2.2; the default is the same value as is used for the screen.

The –**s** option scales the image by the given amount.

The –**a** option overrides the –**s** option and automatically scales the image to fit the page type. If the –**r** option is given as well, then the image is optionally rotated for the best fit.

The –**i** option makes one pixel match one square PostScript unit. Otherwise, a pixel is normally shrunk to give the PostScript image the same size as the screen image.

The –**b** option sets the number of bits per color to download. 8 is equivalent to 24-bit RGB; other possible values are 1, 2, or 4. For printers with small color gamuts, you can speed up printing by using a value smaller than the default of 8.

The –**q** option suppresses all informational and error messages (quiet).

The –**z** option will remove (zap) *image-file* after conversion.

The –**8** option tells itops to generate binary (8-bit) PostScript. The resulting output cannot be sent to most PostScript printers but is approximately half the size.

The –**E** option changes the output to be an Encapsulated PostScript file that is meant to be included in other documents. It will not print.

The –**S** option sets all of the above options that are appropriate for slides: a page type of 35mm, 24-bit RGB, binary PostScript, gamma of 1.0, and automatic scaling and rotation.

The –**L** option lets one specify an *ilabel*(1) *label-file* that will be scaled and rotated to match the image.

SEE ALSO

ilabel(1), gamma(6), snapshot(1), ‘‘Graphics Library User’s Guide,’’ ˜4Dgifts/iristools

DIAGNOSTICS

The program refuses to generate output if the image won’t fit on the page.

BUGS

The –**a** and –**r** options are almost always desired. The options are separate because some older color PostScript printers are much slower when images are rotated.

The image libraries do not allow piping images from one program to another.

Does not do any color correction.

Autoscaling fits the page size, rather than the imageable area.

AUTHOR

Greg Couch, UCSF Computer Graphics Laboratory

NAME

ksdssp – generate **HELIX** and **SHEET** records from protein coordinates

SYNOPSIS

ksdssp [–c *cutoff*] [**–h** *length*] [**–s** *length*] [**–S** *file*] [*PDB_file* [*output_file*]]

DESCRIPTION

Ksdssp is an implementation of the Kabsch and Sander algorithm for defining secondary structure of proteins. *Ksdssp* reads a Protein Data Bank format file containing coordinates of the backbone atoms (N, CA, C, O, and optionally H) of a protein and generates **HELIX** and **SHEET** records. If an amide hydrogen is missing, it is placed 1.01 angstroms from N along the bisector of (1) the vector opposite the bisector of C-N-CA, and (2) the vector opposite the C-O vector from the previous amino acid.

COMMAND ARGUMENTS

–c *energy_cutoff*

The default energy cutoff for defining hydrogen bonds as recommended by Kabsch and Sander is –0.5 kcal/mol (‘‘A good H-bond has about –3 kcal/mol binding energy’’). This option allows the user to change the cutoff.

–h *minimum_helix_length*

Normally, **HELIX** records for helices of length three residues or greater are generated. This option allows the user to change the minimum helix length.

–s *minimum_strand_length*

Normally, **SHEET** records for strands of length three residues or greater are generated. This option allows the user to change the minimum strand length. Reducing the minimum strand length to 1 is not recommended, as there are bridges in many structures that confuse the algorithm for defining sheets.

–S *summary_file*

Normally, *ksdssp* silently discards all the hydrogen-bonding information after generating the **HELIX** and **SHEET** records. This option makes *ksdssp* print the information to a file. The notation is similar to that used by Kabsch and Sander, but is in a vertical instead of horizontal format.

PDB_file

The input Protein Data Bank (PDB) file may contain any legal PDB records. Only **ATOM** records will be used. All others are silently discarded. If no *PDB_file* argument is given, the data is read from standard input.

output_file

The output of *ksdssp* is a set of PDB **HELIX** and **SHEET** records. If no *output_file* argument is given, the records are written to standard output.

SEE ALSO

Wolfgang Kabsch and Christian Sander, ‘‘Dictionary of Protein Secondary Structure: Pattern Recognition of Hydrogen-Bonded and Geometrical Features,’’ *Biopolymers*, **22**, 2577-2637 (1983).

AUTHORS

Conrad Huang
UCSF Computer Graphics Laboratory

NAME

label3d – Midas delegate for displaying 3-D labels

MIDAS COMMAND SYNTAX

`Command:` **delegate start ll label3d**

DESCRIPTION

Label3d is a MidasPlus delegate that generates MIDAS object files which contain character labels composed of move and draw commands. The built-in MIDAS atom labels are fixed-size characters which appear left-to-right across the display. Because the character labels that *label3d* generates are MIDAS objects, they may be translated, rotated and scaled in three dimensions in the same manner as molecules. An additional benefit of being MIDAS objects is that these character labels may be shown by *ribbonjr*(1) and *neon*(1).

Label3d is a program designed to run in conjunction with MIDAS using the MidasPlus **delegate** mechanism. The standard way that *label3d* is invoked is for the user to start a MIDAS session and issue the MIDAS command

delegate start ll label3d

which runs *label3d* and tells MIDAS to pass commands beginning with **ll** through to *label3d* for further processing. The complete set of commands that *label3d* can handle are described in the COMMANDS section of this manual page.

EXAMPLE

A few typical uses of *label3d* are demonstrated in this section. It is assumed throughout this section that the standard PDB file *1alc* (α-lactalbumin) has been opened for display in MIDAS and is the molecule of interest for labeling purposes. For purposes of clarity, *label3d* command names will be shown in full though shortened forms of the command names are accepted by *label3d*. The acceptable short form of the command name will be emboldened for emphasis.

Setting up for labeling

The first thing is to start the *label3d* delegate with the command:

```
delegate start ll label3d
```

You should receive a reply indicating the delegate started normally. After issuing the above command, all commands prefaced with `ll` will be handled by the *label3d* delegate, while other commands will be processed by MIDAS. If the reply states that the delegate failed to start properly, it may be that *label3d* had been invoked earlier in the session but not stopped with the *label3d* command **stop** (see *finishing labeling*, below). Typing

```
~makemark label
```

will allow the delegate to be started normally.

Next, the fonts that will be used must be opened. This example will be using two fonts, which would be opened with the commands:

```
ll open r roman-c
ll open g greek-c
```

The available fonts are described fully in the FONTS section of this document. Opening a font makes it the current font, so after the above commands are executed the current font is greek.

Making a label

To make an example label, which is associated with a specific atom, use the command:

```
ll label #0:41@CD1 Label text
```

This would display the string *Label text* (in greek!) near atom CD1 of residue 41 of model 0.

Changing label attributes

As mentioned above, the current font is greek, so to change the label text to roman, issue the command:

```
ll use r
```

The text initially starts with its lower left corner near the anchor atom. To change the justification so that the center of the right edge of the label is near the atom, use the command:

```
ll justify right center
```

If the label were too close to the structure, it could be moved away with the command:

```
ll offset -1 0
```

which would move the label 1 angstrom to the left (positive x moves right, positive y moves up, positive z [optional] moves closer).

The label could then be made a little smaller with the command:

```
ll scale 0.9
```

If, instead, you wanted to make *all* labels that use the current font to be smaller, you would use the command:

```
ll scalefont 0.9
```

Finally, to color the label light blue, the following two commands could be used (the first of which is a standard MIDAS command):

```
colordef lblue 0.0 0.7 1.0
ll color lblue
```

Making a multi-font label

To make the label ‘‘β sheet,’’ start with the command:

```
ll label #0:43@N b 
```

Note that there is a trailing space at the end of the above command, so that the text that will be appended later (‘‘sheet’’) will have a space separating it from ‘‘β.’’

The attributes of the new label could now be set with:

```
ll justify right bottom
ll use g
colordef dgreen 0.0 0.6 0.0
ll color dgreen
```

The remainder of the label would be appended with:

```
ll append sheet
```

The appended part of the label would be set to roman font and the entire label given additional space from the structure with the commands:

```
ll use r
ll offset -2 0
```

Making a title

Since titles are not associated with any particular atom, one creates a title at the origin with a command such as:

```
ll label Structure
```

and use **scale** and **position** to size and position it:

```
ll scale 2
ll position 14 -6 35
```

The above **position** command places the title in the lower right area of the screen and (due to the positive z) in front of the molecule. This z positioning is useful in stereo views.

Finishing labeling

It is rarely desirable to stop the *label3d* delegate since the label positions then can no longer be updated to compensate for model rotations. However, occasionally the user may wish to remove all current labels and start over. In this case, *do not* use the MIDAS **delegate stop** command. Instead, use the *label3d* command **stop**. This allows the *label3d* delegate to issue some cleanup commands to MIDAS before stopping. Unless these cleanup commands are executed, a new *label3d* delegate cannot be started. Note that the **stop** command will remove all current labels.

MIDAS cannot restore delegates to the state they were in when a MIDAS session is saved. If it is necessary to save a session involving a *label3d* delegate, the delegate’s state will have to be recorded with the *label3d* **save** command, and restored in the restarted session with the **source** command. Both these commands are described in detail in the COMMANDS section of this manual page.

COMMANDS

When *label3d* is acting as a delegate for MIDAS (as shown in the DESCRIPTION section), the user can give it commands by typing

ll *label_command*

in the MIDAS command window. The list of commands is given below in alphabetical order. Note that the command names can be shortened to the minimum number of characters need to distinguish the command from other *label3d* commands. For example, **append** could be shortened to **a** or **scalefont** could be shortened to **scalef**.

append *value*

Append the string *value* to the current label. This creates a multi-font label, and makes the newly appended part the current label. Subsequent commands such as **color** and **use** will only affect the new addition. To make some intermediate part of a multi-font label become the current label, use the **label** command described below.

close *font_name*

Close the font named *font_name*. If any label is currently displayed using the specified font, *label3d* reports the error and the font is not closed.

color *midas_color_name*

Make the current label be drawn using color *midas_color_name*, which must already be defined in MIDAS.

justify (**left | center | right**) [(**top | center | bottom**)]

Set the justification mode of the current label. If the vertical justification is not specified, the current vertical justification is retained. Labels are initially justified left/bottom.

label *label_ref* [*display_value*]

Make the label whose reference name is *label_ref* the current label. If no such label exists, create such a label. If *display_value* is supplied, make that the value shown for the label. If not, and the label did not previously exist, make *label_ref* the display value.

The label reference name is a any sequence of characters that does not include '[' or ']'. The brackets are reserved for referencing different components of a multi-font label. For example, **name[1]** refers to the second component of the multi-font label **name** (the index is zero-based). To create a multi-font label, see the **append** command above.

A newly created label is displayed in the current color (or white, if no color has been set) using the current font. To change these defaults, use the **color** and **use** commands.

The lower left-hand corner of the label is considered its starting point. Labels whose reference name matches a MIDAS atom specifier have starting coordinates that are the same as its atom; other labels have starting coordinates at the origin by default, but may be changed using the **position** command. **The starting coordinates of all labels with reference names matching MIDAS atom specifiers are updated from MIDAS every time a command is processed by** *label3d*.

The MIDAS object move/draw commands for a label are constructed by adding the label starting coordinates and offset to the coordinates found in the character descriptions (see **open**). Consecutive characters in a string are offset in the x dimension by the width of the left character plus a small inter-character spacing; no offsets are added in either the y or z dimension. The move/draw commands for all labels are recomputed every time a command is processed by *label3d*.

list (**label | font**)

List all labels or fonts.

offset *x y* [*z*]

Change the starting position of the current label by adding an offset of *(x,y,z)* angstroms. If *z* is not provided, zero is used in its place. This command is useful for repositioning the label when it hides or is hidden behind important atoms. Repeated use of this command is not cumulative; *i.e.*, the new offset overrides the old offset rather than being added to it.

open *font_name directory*

Read character descriptions from directory *directory* and place them in a new font named *font_name*. The newly opened font becomes the current font. If *directory* is not an absolute path (*i.e.* it does not start with a '/') then the system font directory will be searched for *directory*. Fonts available in the system font directory are described in the FONTS section of this document.

The character descriptions are stored in files whose names are either single-lettered, or begin with '0' and consist of all numeric digits. Files with single-letter names contain descriptions for the corresponding letters; files with names composed of all numeric digits contain descriptions for the characters with the corresponding octal ASCII values. Character descriptions are simply move-and-draw commands in MIDAS object file format. For example:

```
.m 0 0 0
.d 0 1 0
```

position *x y* [*z*]

Set the starting coordinates of the current label to be *(x,y,z)* in angstroms. If *z* is not provided, zero is used in its place. Note that this command has no effect if the current label's reference name matches a MIDAS atom specifier, as the label coordinates will be updated automatically.

save *filename*

Write out a series of commands to file *filename*. When this file is processed by the **source** command (see below), the same fonts and labels that existed when the **save** command was executed will be created.

scale *sx* [*sy*]

Scale the current label by *sx* in the x dimension, and *sy* in the y dimension. If *sy* is missing, it is set equal to *sx*.

scalefont *sx* [*sy*]

Scale the current font by *sx* in the x dimension, and *sy* in the y dimension. If *sy* is missing, it is set equal to *sx*.

source *filename*

Read commands from file *filename* and execute them.

stop

Terminate *label3d*. Note that this command should be used instead of the MIDAS **delegate stop** command so that *label3d* can issue some necessary cleanup commands to MIDAS before exiting. Also note that this command will *remove all labels*!

unlabel [*label_ref*]

Destroy the label named *label_ref*. If *label_ref* is omitted, the current label is destroyed.

update

Update is a do-nothing command that may be used when model positions have changed and the position or orientation of labels need to be updated.

use *font_name*

Make the current label be displayed using font *font_name*.

FONTS

The font data provided with *label3d* was derived from the public domain distribution of the Hershey Fonts.

A variety of font faces, font sizes, and levels of detail are provided.

The font faces are:

Name	**Description**
roman	Roman
greek	Greek
italic	Italic
script	Script (cursive)
cyril	Cyrillic
gothgr	Gothic German
gothgb	Gothic English
gothit	Gothic Italian

The font sizes and level of detail are:

Name	**Description**
p	Plain (very small, no lower case letters)
s	Simplex (plain, normal size, no serifs)
d	Duplex (normal size, no serifs, doubled lines)
c	Complex (normal size, serifs, doubled lines)
t	Triplex (normal size, serifs, tripled lines)
cs	Complex small (as Complex, but smaller than normal size)

Not all font faces are available in all the types. The cross-table of availability is:

	p	s	d	c	t	cs
roman	●	●	●	●	●	●
greek	●	●		●		●
italic				●	●	●
script		●		●		
cyril				●		
all gothic					●	

The name used to open the font with the *label3d* **open** command is made by composing the font face name with the type name, separated by a hyphen. So the roman font face in the complex type would be referred to as ‘‘roman-c.’’

Typically one would use the the most detailed type available for the font face unless a very large number of labels are going to be used, when perhaps a simpler type might be employed.

BUGS

Good vector fonts are hard to come by.

FILES

/usr/local/midas/resource/label3d/fonts	system font directory

SEE ALSO

MidasPlus User’s Manual

AUTHOR

Conrad Huang, Computer Graphics Laboratory, UCSF
Eric Pettersen, Computer Graphics Laboratory, UCSF (Hershey font conversion)

ACKNOWLEDGEMENTS

The Hershey Fonts were originally created by Dr. A. V. Hershey while working at the U. S. National Bureau of Standards. The format of the Font data distribution (*not* the format used by *label3d*) was originally created by:

James Hurt
Cognition, Inc.
900 Technology Park Drive
Billerica, MA 01821

NAME

longbond – remove long bonds from structure

MIDAS COMMAND SYNTAX

`Command:` **pdbrun conect longbond** [**–l** *bond_length*] [**–r** *bond_ratio*]

DESCRIPTION

Longbond examines each bond MIDAS lists in CONECT records and, if the bond length exceeds the user-specified threshold, issues a MIDAS **~bond** command to remove the bond. This command is useful for cleaning up structures for which MIDAS generates unreasonable bonds.

By default *longbond* remove each bond whose length exceeds 1.5 times its typical length, derived from the types of the bonded atoms. The ratio of 1.5 may be overridden using the **–r** flag. If the **–l** flag is given, then the bond length is compared against the absolute distance *bond_length* rather than the type-dependent length.

BUGS

Some bond-removal commands issued by *longbond* may be ignored by MIDAS because a residue would then be broken into two disconnected pieces. In such cases, the user needs to create the proper connecting bonds and then reinvoke *longbond*.

AUTHOR

Conrad Huang
UCSF Computer Graphics Laboratory

NAME

makems – MidasPlus delegate for convenient generation of molecular surfaces

MIDAS COMMAND SYNTAX

`Command:` **delegate start ms makems**

DESCRIPTION

Makems is a MidasPlus delegate that takes an atom specification and then generates and displays the corresponding solvent-accessible molecular surface. *Makems* only works correctly when surfacing a single model; it cannot correctly surface multiple models. If multiple models are present when *makems* is asked to surface a particular model, the requested model will be surfaced as if the other models did not exist.

The standard way that *makems* is invoked is for the user to start a MIDAS session, and issue the MIDAS command

delegate start ms makems

which runs *makems* and tells MIDAS to pass commands beginning with **ms** through to *makems* for further processing.

Any text sent to *makems* is interpreted as an atom specification to be surfaced. *Makems* will compute the surface by invoking *dms*(1) (with default *dms* settings) and then display the computed surface in MIDAS. Any surface previously computed by *makems* will be closed first.

EXAMPLES

Some simple examples (the examples assume that *makems* was invoked as shown in the SYNTAX section above):

`ms :CYS`

Surface all cysteine residues.

`ms`

Surface the entire molecule.

`ms /color=green`

Surface all atoms currently colored green.

BUGS

As mentioned in the DESCRIPTION section, *makems* only surfaces one model. If the atom specification given to *makems* spans multiple models, the lowest-numbered one will be surfaced.

AUTHOR

Dr. Sherri Newmyer
The Scripps Research Institute
10550 North Torrey Pines Road
La Jolla, California 92037
newmyer@scripps.edu

NAME

midas – MidasPlus molecular interactive display program

SYNOPSIS

midas [*X-options*] [**–fullscreen**] [**–nofork**] [**–nostereo**] [**–script** *script*] [**–delegate** *delegate_name*] [*session_file*|*model_file(s)*]

DESCRIPTION

Midas is the interactive 3D display program for the MidasPlus molecular modeling system. The program is designed to generate easily manipulated views of large molecules, primarily proteins and nucleic acids. *Midas* normally starts up in a window, but if the **–fullscreen** flag is used, then *midas* will use the entire graphics display on start-up. It can be abbreviated **–full**.

The **–nofork** flag tells *midas* to not go into the background. See the MIME section below for the most common usage.

If stereo will not be used, then it may be advantageous to give the **–nostereo** option. This tells *midas* to look for a double-buffered instead of a quad-buffered graphics window, thus saving screen memory for other applications. This only affects systems that support (nondistorted) stereo in a window.

The **–script** *script* option lets you specify a command script to be sourced at start-up time.

The **–delegate** *delegate_name* option treats standard input and output as a delegate with the given name.

If there is more than one filename specified on the UNIX command line, *midas* opens the first file as model 0, the second as model 1, *etc.* (see **open** in the Command Reference Guide section of the MidasPlus User's Manual). If only one file is given on the UNIX command line, then *midas* will try to read it as a session file first, and then open it as a model if it is not a session.

Midas also understands the various standard X-application options. See the OPTIONS section of the *X*(1) manual page for a list of possible options.

Details of *midas* commands and their syntax can be found in part II of the MidasPlus User's Manual.

MIME

Midas can be used as a MIME viewer for web browsers by adding the line ‘‘chemical/x-pdb; midas –nofork %s’’ to your $HOME/.mailcap. See *mailcap*(4). This will also work with MIME-capable mail programs, such as *MediaMail*(1).

ENVIRONMENT

MIDASRC
: Give an alternate location for the startup file. The default location is /usr/local/midas/resource/midas/midas.rc.

MIDAS_COPY
: Use an alternate program instead of **lpr –h** to output to a printer from the *midas* **copy** command.

MODELS
: Give an additional directory in which to search for residue templates (these define the connectivity of residues in PDB files).

GROUPS
: Give a list (colon-separated) of directories in which to search for chemical groups (user's groups) in addition to the system groups. See the **addgrp** command for more information.

SHELL
: The UNIX shell *midas* will use to run programs specified with the **delegate**, **pdbrun**, **run**, and **system** commands.

SEE ALSO

MidasPlus User's Manual, netscape(1), MediaMail(1), mailcap(4), X(1)

AUTHORS

Conrad Huang, Greg Couch, and Thomas Ferrin, UCSF Computer Graphics Laboratory
Paul Bash contributed vdw-style surfaces and the **addaa**, **addgrp**, **swapaa** and **swapna** commands.

Countless other contributions from a cast of hundreds.

NAME

midas.tty – terminal-based version of MidasPlus display program

SYNOPSIS

midas.tty [**–b**] [**–s** *script*] [**–R** *num_rows*] [**–C** *num_columns*] [*session_file|model_file(s)*] [*file...*]

DESCRIPTION

Midas.tty is a version of MIDAS that runs on any ASCII terminal. *Midas.tty* does everything that the MIDAS interactive display program does except generate a three-dimensional image of the molecular model. In particular, it accepts all normal MIDAS commands (*e.g.* **open**, **chain**, **label**, **distance**, **copy**, **save**). This is useful for editing an annotated PDB file or setting up session files without accessing the graphics display on a workstation.

If either standard input or standard output is redirected (*i.e.* is not a terminal), then *midas.tty* runs in a line-oriented batch mode. Thus, it can be called from within a shell script for mass processing of MIDAS sessions.

OPTIONS

The **–b** option puts *midas.tty* into a batch mode regardless of whether standard input or output is redirected.

The **–s** *script* option lets you specify a command script to be sourced at start-up time.

The **–R** and **–C** options allow you to change the aspect ratio of the *midas.tty* ‘‘window’’ by changing the number of rows and columns. See the BUGS section, below.

SEE ALSO

midas(1), MidasPlus User’s Manual

AUTHOR

Greg Couch, UCSF Computer Graphics Laboratory

BUGS

Interactive distances/angles are not shown in batch mode.

The aspect ratio of the terminal emulation window affects the size and orientation of hardcopy images generated with the **copy** command. Use the **–R** *num_rows* and **–C** *num_columns* command-line options to change the aspect ratio.

NAME

mrotate – MidasPlus delegate for interpolating between orientations

MIDAS COMMAND SYNTAX

`Command:` **delegate start mr mrotate**

DESCRIPTION

Mrotate is a MidasPlus delegate that generates MIDAS commands that transform models from one orientation to another. The standard way that *mrotate* is invoked is for the user to start a MIDAS session, and issue the MIDAS command

delegate start mr mrotate

which runs *mrotate* and tells MIDAS to pass commands beginning with **mr** through to *mrotate* for further processing. The complete set of commands that *mrotate* can handle are described in the COMMANDS section of this manual page.

EXAMPLE

A few typical uses of *mrotate* are demonstrated in this section. It is assumed throughout this section that the standard PDB file *1alc* (α-lactalbumin) has been opened for display in MIDAS as model 0 and is the molecule of interest for transformation purposes.

Setting up

The first thing is to start the *mrotate* delegate with the command:

```
delegate start mr mrotate
```

You should receive a reply indicating the delegate started normally. After issuing the above command, all commands prefaced with `mr` will be handled by the *mrotate* delegate, while other commands will be processed by MIDAS.

Taking snapshots of orientations

To save an orientation as a potential endpoint of interpolation later, use the command:

```
mr snapshot s1
```

The **s1** in the above command is the name that is assigned to the orientation; the name will be used in interpolation commands later. To save another orientation, move the model around using the mouse and issue the command:

```
mr snapshot s2
```

Interpolating between orientations

To interpolate from orientation **s1** to **s2**, use the commands:

```
reset s1
mr interp s1 s2 0
```

The *reset* command puts the model back into orientation **s1**. The *interp* command makes *mrotate* generate a series of MIDAS commands which will transform model 0 from orientation **s1** to orientation **s2**. Note that model 0 must be the only model selected, and that it must already be in orientation **s1** when the command is given. To save the generated MIDAS commands in file `com`, use the command:

```
mr store com s1 s2 0
```

Finishing up

To terminate the *mrotate* delegate, use the command:

```
mr stop
```

All saved snapshots will, of course, be lost.

COMMANDS

When *mrotate* is acting as a delegate for MIDAS (as shown in the DESCRIPTION section), the user can give it commands by typing

mr *mrotate_command*

in the MIDAS command window. The list of commands is given below in alphabetical order.

interp *from to n*

Generate the command to interpolate model *n* from orientation *from* to orientation *to*. This command assumes that model *n* is already in orientation *from*. If models other than *n* are selected, they will also be affected by the rotations and translations that *mrotate* generates.

snapshot *name*

Take a snapshot of the current orientation, and associate *name* with it. The standard MIDAS command:

savepos *name*

is also executed automatically so the orientation may be recovered later in MIDAS using the command:

reset *name*

Note that only the orientation (*i.e.*, the atomic positions) are saved; other attributes, such as colors and whether atoms are displayed are ignored. *Name* may be used in later *interp* and *store* commands as either the *from* or *to* orientation.

stop Terminate the delegate.

store *filename from to n*

This command does exactly the same thing as *interp* except the MIDAS commands are appended to file *filename* instead of being sent to MIDAS.

BUGS

Makes too many assumptions about where models are in MIDAS.

SEE ALSO

MidasPlus User's Manual

NAME

ms – calculate a solvent-accessible molecular surface

SYNOPSIS

ms *file* [**–a**] [**–d** *density*] [**–e** *file*] [**–g** *file*] [**–i** *file*] [**–s** *file*] [**–n**] [**–r** *b–e*] [**–w** *radius*] **–o** *file*

NOTE

The **ms** *command has been obsoleted by the* **dms** *command. It is recommended that* **dms***(1L) be used instead of* **ms***.* **Dms** *alleviates many of the shortcomings of* **ms** *such as handling of chain identifiers, recognition of uncommon atom types, etc. See the* **dms***(1L) manual page for further details.*

DESCRIPTION

Ms calculates the molecular surface of a molecule. The molecular surface resembles the van der Waals surface of a molecule, except that crevices between atoms are smoothed over and interstices too small to accommodate the probe are eliminated. The surface includes cavities in the interior of the molecule, even if they are not accessible to a solvent molecule coming from the outside.

The molecular surface calculated is that defined by F. M. Richards (1977, *Ann. Rev. Biophys. Bioeng.*). According to Richards' definition the molecular surface consists of two parts: *contact surface* and *reentrant surface.* The contact surface is made up of ''those parts of the molecular van der Waals surface that can actually be in contact with the surface of the probe.'' The reentrant surface is defined by ''the interior-facing part of the probe when it is simultaneously in contact with more than one atom.''

File is an input file of coordinates. The input file must be in the Protein Data Bank format. The first letter or first two letters of the atom name is used to determine the element type. By default, implicit hydrogens are included for carbon, nitrogen and oxygen atoms, thus aromatic carbons and nitrogens will have van der Waals radii that are somewhat too big. Note that only amino acid residues will be included unless **–a** is also specified. Because coordinates are multiplied by 100 and stored as integers, coordinates must have absolute values smaller than 327.67.

The flags may be in any order. The meanings of the flags are described below:

–a Include all atoms, not just those in amino acid residues.

–d Change the density of points on the surface. *Density* is a factor affecting the density of points on the surface: the default of 1.0 produces about 5 points per square angstrom. Only values between 0.1 and 1.0 are permitted. For large proteins, a density of 0.5 is recommended.

–e Calculate only the surface lying within the ellipsoid specified in *file. File* consists of 5 lines which define an ellipsoid. The five lines are: the ellipsoid center (line 1), an orthogonal matrix representing the orientation of the ellipsoid (lines 2-4), and the lengths of the three semiaxes (line 5). The normalized vectors of the semiaxes form the columns of the orthogonal matrix. It is recommended that you use a sphere so the matrix will be a unit matrix and all three lengths will just be the sphere radius.

–g Write all the informative messages to *file,* instead of the standard error output. Genuine errors still go to the standard error output. This file is not rewound at any time, so messages from several runs may be accumulated.

–i Calculate the molecular surface only for those residues and atoms specified in *file,* but keeping the rest of the molecule for collision checks. The file consists of a series of lines such as the following:

```
ASP  205 CA
TYR   13 *
GLY  116 FRM
HIS  178 TO
```

The asterisk means all atoms of the residue and the ''FRM'' and ''TO'' mean all residues from 116 to 178 inclusive. These records are read with a FORTRAN ''a3,1x,a4,1x,a3'' format statement and the residue name, sequence number, and atom name (if specified) must match those of the input file exactly. The residue name must be left-justified, and the sequence number must be right-justified. The sequence number may contain letters. Up to one hundred such records may be used. The **–e** and **–i** flags are compatible. The surface generated using the **–i** flag is not always the same as the surface generated by running the entire molecule and afterwards selecting out the desired atoms. The first surface will not include reentrant surface lying between an atom in the **–i** file and atoms not in the file. (The QCPE version of *ms* does not have this bug.)

–s Use the supplied file argument to specify the atomic radii. The format of the file is an atomic symbol followed by its radius, one per line. */usr/local/midas/resource/connect.tpl* uses this format.

–n Include the unit normals to the surface with each surface point record.

–o The output is written to *file.* This flag is not optional.

–r Only residues numbered *b* through *e* inclusive are used in the calculation. This is quite different from the **–i** flag. Residue sequence numbers involving letters may cause problems.

–w Change the water probe radius from the default radius of 1.4 angstroms. This parameter must be between 1.0 and 2.0.

The output consists of a series of atom and surface point records, with the same format for the first six fields. Each atom is followed by the surface points (if any) which belong to it. These first six fields are in the following format: residue name, sequence number, atom name, x coordinate, y coordinate, z coordinate. For an atom record, the seventh field is ‘‘A.’’ For a surface point record, the seventh field begins with an ‘‘S,’’ followed by a ‘‘C’’ or ‘‘R’’ according to whether the point is part of contact or reentrant surface. This is followed a digit used for depicting different density levels. The eighth field is the molecular surface area associated with the point in square angstroms. If the **–n** flag is specified, the next three fields are the unit normal vector pointing outward from the surface. Informative messages and errors are written to the standard error output unless a **–g** file is specified. The calculation takes about 5 seconds per atom for molecules of fewer than 1000 atoms and 7 seconds per atom for larger molecules (timings are for a VAX780).

The chemical elements that the program can currently handle are those which the author has found to occur in molecules of interest *and* whose van der Waals radii could be located in the literature. The atoms currently recognized are:

Element	Radius
H	1.20
C	1.90
N	1.50
O	1.40
F	1.35
P	1.90
S	1.85
Cl	1.8
Fe	0.64
Cu	1.28
Zn	1.38
Br	1.95
I	2.15

BUGS

Atoms must be consecutively numbered in the PDB file for correct results.

AUTHOR

Michael Connolly
University of California, San Francisco

LIMITATIONS

6000 atoms. Also, there cannot be more than 8000 waters involved in overlapping reentrant surface removal. This is generally not a problem, unless you have a large protein with many internal cavities and you have not used the **–d** flag to reduce the density.

FILES

r??????	intermediate file of reentrant surface points - binary
a??????	intermediate file of reentrant surface points - ASCII
s??????	intermediate file of reentrant surface points - sorted
c??????	intermediate file of contact surface points - ASCII
o??????	intermediate file of all surface points
k??????	intermediate file from getcoord
e??????	intermediate file of atoms inside ellipsoid
b??????	binary file of coordinates for buffering

DIAGNOSTICS

Many and varied. Be sure to examine the **–g** file and ‘‘submit.out’’ before you leave a background job running overnight.

NAME

neon – generate a molecular model with solid stick bonds and shadows

MIDAS COMMAND SYNTAX

`Command:` **neon** [*conic options*]
`Command:` **preneon** > *output_file*

DESCRIPTION

Neon works with the MidasPlus package to create solid stick or ball-and-stick representations of molecular models. The currently displayed atoms, their orientation, position and colors are taken from the interactive display of *midas*(1) using the **pdbrun** Midas command and are sent to *neon*. *Neon* processes the information under the control of parameters set in the *neon.dat* file (in the current directory) and the output is sent to the Midas utility *conic*(1) to create the final image. See *conic*(1) for a detailed description of its command-line options. *Neon* has three parameters for the depiction of a simplified backbone: a 'smooth' tube connecting α-carbons, an intermediate 'bent' tube, and a 'straight' tube with straight segments connecting α-carbons. *Neon* has two parameters which create a smooth gradient of color over the the model based on the temperature factors of the atoms. Dashed or solid lines can be drawn between atoms joined by the **distance** command in *midas* to illustrate hydrogen bonds and other interactions. *Neon* handles Midas objects, allowing arrows and lines to be displayed.

It is also possible to combine *neon* and *conic* rendering styles, or differing types of *neon* rendering styles. See the PRENEON section and the EXAMPLES.

''Capturing Screen Images'' in Part III of the MidasPlus manual discusses saving, converting, and printing *neon* images.

EXAMPLE OF THE NEON.DAT CONTROL FILE

This file, *neon.dat*, is read from the current working directory of *midas*. The example below lists the default control parameters used when the *neon.dat* file is missing. Input is free format, one control parameter per line.

```
0           |Tubetype           |0=all atoms, 1=smooth, 2=bent, 3=straight|
3           |Sphere density     |1=rough, 5=high resolution               |
1           |Dash    flag       |0=no dashes,  1=draw dashes              |
1           |Object  flag       |0=no objects, 1=draw objects             |
0           |CPK     flag       |0=normal Render output, 1=CPK output     |
0.25        |Stick   thickness  |Angstroms                                |
0.50        |Tube    thickness  |Angstroms                                |
0.10        |Dash    thickness  |Angstroms                                |
0.10        |Object  thickness  |Angstroms                                |
0.04        |Density thickness  |Angstroms                                |
0           |Atom rendering     |0=stick, 1=ball & stick                  |
0.25        |Atom size          |Fraction of vdw radius (ball&stick mode) |
0 0.50      |Depthcue,Percent   |0=n, 1=y; fraction aft intensity         |
0 0.0 0.0   |B-factor,Range     |0=n, 1=shade to color, 2=rainbow; min-max|
1.0 1.0 1.0 |B-factor color     |RGB (red, green, blue)                   |
5           |Number of dashes   |Number drawn between each pair of atoms  |
1.0 1.0 0.0 |Dash color         |RGB (red, green, blue)                   |
0.30        |Dash offset        |Dist. from atom to first dash, Angstroms |
0.40        |Dash/space ratio   |0.0=dots, 0.4=dashes, 1.0=solid line     |
4.00        |CA conect distance |Angstroms                                |
```

CONTROL CARDS

Tubetype

Backbone atom representation: all atoms, smooth tube, bent tube, or straight tube. The color of the tube representation corresponds to the color of the α-carbon.

Sphere density

Density of spheres drawn to create sticks, tubes, dashes and objects. A value of 3 is good for most work, 4 is recommended for final full-screen images so that individual spheres are not resolved, and 5 is best for closeups of a small number of residues. Increasing the number slows the calculation. Though the setting 5 is sufficient for almost all situations, occasionally when working on a highly scaled-up view or with very thin cylinders a higher setting is required. *Neon* permits settings up to 11. Note that each setting above 5 uses double the number of spheres of the next lower setting. Therefore, to minimize wait time, always use the lowest setting that provides a satisfactory image.

Dash flag

Set to 1 to draw lines between between atoms connected by the distance command in Midas. The type of line is under control of dash parameters listed below.

Object flag

Set to 1 to include Midas graphics objects. See ‘‘Non-Molecular Graphics Objects’’ in Part III of the MidasPlus User’s Manual for details on graphics objects.

CPK Flag

Set to 1 to get the same space-filling representation of atoms obtained with *conic* with the addition of depthcuing, coloring by temperature factor and the inclusion of graphics objects when the corresponding parameters are set.

Stick thickness

Radius in angstroms of sticks connecting atoms.

Tube thickness

Radius in angstroms of tubes connecting α-carbons.

Dash thickness

Radius in angstroms of dashed lines.

Object thickness

Radius in angstroms of lines and dots in graphics objects.

Density thickness

Radius in angstroms of electron density lines (the density itself is not changed, of course!).

Atom rendering

Controls whether atoms are represented as sticks or balls and sticks.

Atom size

Radius of balls in ball-and-stick mode as a fraction of the van der Waals radius specified by *midas*.

Depthcue, fraction

The first value turns depthcuing on or off. The second value is the fractional intensity of atoms at the back of the image.

B-factor, range

Control of coloring by temperature factor. The first value controls the coloring type. Type 0 turns the option off, so the model will be colored as displayed. Type 1 creates a gradient of color starting with the B-factor color specified below for atoms with lowest temperature factor to the displayed color for atoms with highest temperature factor. Type 2 creates a gradient of color (blue, magenta, red, orange, yellow) from lowest to highest temperature factor value. The second and third values specify the minimum and maximum temperature factors over which the gradient of color is calculated. Atoms with temperature factors above and below the values are colored yellow and blue respectively. If both values are 0.0, the minimum and maximum values are determined from the displayed atoms.

B-factor color

The RGB (red, green, blue) color of atoms with lowest temperature factor. Used with B-factor coloring type 1.

Number of dashes
: Controls the number of dashes drawn between pairs of atoms.

Dash color
: The RGB (red, green, blue) color of dashed lines.

Dash offset
: Controls the distance in angstroms between the center of atoms and the ends of the dashed lines so that dashes do not bump into the model.

Dash/space ratio
: Controls the relative proportions of dash and intervening space. A value of 0.0 gives dots, 0.4 is good for dashed lines, and 1.0 gives a solid line.

CA connect distance
: Cutoff distance in angstroms for determining if the backbone tube will be drawn between subsequent α-carbons of each model. For example, if residues 1-10 and 30-40 of a model were displayed, the cutoff would be used to find the break between residues 10 and 30. If unwanted breaks occur, this value should be increased. Note that if tubetype is 0, this parameter is ignored and CONECT records are used to establish connectivity.

RUNNING NEON

A copy of the file /usr/local/midas/resource/neon.dat should be present in your current working directory. Default values will be used if the file is not found.

To create an image, set the desired parameters in *neon.dat*. In the interactive window of *midas*, display, orient and color the model or models of interest. Enter the command **neon –p** to preview the image. After a short pause a small *conic* window will open displaying the image. Option flags following the **neon** command correspond to options of *conic*, which control the appearance of the image and whether the image is saved in a file for later use.

PRENEON

To create a complex image, multiple outputs of *neon* can be saved, combined, and then sent to *conic* to draw the final image. The Midas command **preneon** runs only *neon*, and the output must be directed to a file. If the output of **preneon** is not directed to a file it is returned by default to *midas*, which will have to work its way through many lines of invalid commands. The **preneon** command does not accept any command line flags.

EXAMPLES

1) To have a quick look at an image
Command: **neon –p**

2) To save a full screen image in the file **figure.i**
Command: **neon –F –o figure.i**

3) To create a complex image with a ligand shown as sticks and the binding site shown as space-filling atoms, set *neon.dat* for sticks, display just the ligand and save the output of *neon* in the file **site**.
Command: **preneon > site**

   Turn on the CPK flag in *neon.dat* and display just the binding-site atoms. Append the output of *neon* to the file **site**.
Command: **preneon >> site**

   Send the combined information in the file **site** to *conic* using Conic parameters from the file **param**, and save the full-screen image in the file **figure.i**.
Command: **conic –c param –F –o figure.i site**

FURTHER EXAMPLES

The demonstration images included on the MidasPlus distribution CD show how to achieve a variety of striking effects and give detailed instructions on how each image was made. If these demonstration images have been installed on your system, they will be found in /usr/local/midas/demos/images. The README.index file there has further information. If the demonstration images have not been installed on your system, you need to mount the distribution CD-ROM, and you will find the images in the CD-ROM directory Midas-2.1/demos/images.

SEE ALSO

conic(1), midas(1)
MidasPlus User’s Manual

FILES

/usr/local/midas/resource/neon.dat — default *neon* parameter file

AUTHOR

Thomas R. Hynes
Protein Engineering Department, Genentech Inc., and
Department of Pharmaceutical Chemistry, UCSF

NAME

noeshow – show AMBER INTERFACE format NMR-derived constraints
mardishow – show MARDIGRAS format NMR-derived constraints

SUPPORT PROGRAMS

sander2amber – convert AMBER SANDER namelist-style distance constraints to AMBER INTERFACE format

xplor2amber – convert X-PLOR distance constraints to AMBER INTERFACE format

xplor2tors – convert X-PLOR torsion angle information to AMBER style

diana2amber – convert DIANA distance constraints to AMBER INTERFACE format

MIDAS COMMAND SYNTAX

`Command:` **noeshow** [*options*] *constraint_file* [*torsion_file*]
or
`Command:` **mardishow** [*options*] *constraint_file*

SHELL COMMAND SYNTAX

sander2amber pdb_file sander_input_constraints > reformatted_constraints
diana2amber [–v *viol_add*] pdb_file diana_low_bounds diana_high_bounds > reformatted_constraints
xplor2amber [–c *chain_ID*] [–v *viol_add*] pdb_file xplor_constraints > reformatted_constraints
xplor2tors [*options*] xplor_torsion_output > reformatted_torsion_file

DESCRIPTION

Noeshow and *mardishow* are programs for displaying NMR-derived constraints on model structures in MIDAS. *Noeshow* can display distance constraints in AMBER INTERFACE format and, via format conversion programs, constraints in X-PLOR, DIANA, MARDIGRAS (output), or AMBER SANDER (namelist) formats. *Mardishow* can display constraints in MARDIGRAS input or output formats. *Noeshow*'s functionality is generally a superset of that provided with *mardishow*, and *mardishow* is provided principally for its ability to display MARDIGRAS input format constraints. If a conversion program for MARDIGRAS input format is developed, *mardishow* may be withdrawn.

Noeshow can also show torsion angle constraints if given the restraint analysis output from the SANDER AMBER module. The conversion program **xplor2tors** can be used to convert X-PLOR torsion angle information to a form usable by *noeshow*.

Finally, *noeshow* can be used to indicate all hydrogens within a given cutoff distance of one or more designated hydrogens. This capability can be used with or without an accompanying set of distance constraints.

USAGE

Distance constraint preparation

If your distance constraints are in AMBER INTERFACE format, they need no further preparation to be used by *noeshow*. If you have constraints in MARDIGRAS input format, they are ready for use in *mardishow* (see the ‘‘Mardishow’’ section, below). Constraints in other formats will need to be converted. If you are not certain what format your constraints are in, examples of each are provided in the ‘‘Constraint formats’’ section below.

Constraints in AMBER INTERFACE format have an ‘‘energy well’’ for each constraint. There are two upper-bound values and two lower-bound values. The lower of the two upper-bound values is where (if the distance exceeds the bound) an energy penalty begins to be applied (in the AMBER simulation) and the higher upper-bound value is where the penalty reaches a maximum. The lower bounds are similar, but apply to short distances. Some constraint formats (*e.g.* X-PLOR) describe only one upper and one lower bound. How this is mapped to the AMBER INTERFACE-type constraint style depends on the conversion procedure. The conversion procedure for the various formats are:

X-PLOR Running the *xplor2amber* conversion script (as shown in the SHELL COMMAND SYNTAX section, above) will produce a file of equivalent constraints in AMBER INTERFACE format. The bounds given in the X-PLOR constraint file are taken as describing the ‘‘floor’’ of the energy well required for the AMBER INTERFACE format description, *i.e.* the distances where an energy penalty would begin to be applied. The distances where the energy penalty reaches a maximum is taken to be 0.5 angstroms beyond the given bounds on either side. The default value of 0.5 can be changed by giving the **–v** option flag to *xplor2amber*, followed by the desired distance. As described later, the bounds of the energy well determine whether the constraint will be shown as satisfied or violated (see the ‘‘Basic usage’’ section).

Xplor2amber will normally interpret segids in the X-PLOR constraint file as chain IDs in the converted file. The ‘‘**–c** *chain_ID*’’ option to *xplor2amber* will cause it to give all atoms the specified chain ID. Note that specifying *chain_ID* as " " (a quoted space) will give the atoms *no* chain ID, even if segids are present in the X-PLOR constraint file.

DIANA DIANA format constraints are handled in almost the same manner as the just-described X-PLOR format constraints. The only differences are that the conversion script is *diana2amber* instead of *xplor2amber*, and that instead of specifying a single bounds file on the command line, separate lower- and upper-bounds files (in that order) are specified. Also, there is no **–c** option for *diana2amber*.

AMBER SANDER (namelist)

Running the *sander2amber* conversion script (as shown in the SHELL COMMAND SYNTAX section, above) will produce a file of equivalent constraints in AMBER INTERFACE format. Since AMBER SANDER constraints already describe an energy well, the mapping to an AMBER INTERFACE energy well description is straightforward.

MARDIGRAS (output)

Although *mardishow* is capable of displaying MARDIGRAS (output) format constraints without conversion, it may nonetheless be desirable in certain circumstances to convert the constraints for use with *noeshow* in order to make use of features only present in the latter program. Such a conversion is a two-step process. First, the constraints are run through the *mardi2amber* program to convert them to AMBER SANDER (namelist) format. The *mardi2sander* program is provided with the MARDIGRAS package, and is documented there. The second step is to convert the AMBER SANDER format constraints to AMBER INTERFACE format, as detailed in the paragraph above.

Torsion angle constraint preparation

You need not have torsion angle constraint information in order to use *noeshow* or *mardishow*; such information is optional. In fact, *mardishow* cannot display torsion angle constraint information, only *noeshow* can.

Noeshow uses the *output* torsion angle restraint information from the SANDER AMBER module, rather than the input torsion angle info. This is because it uses the energy penalty information provided by AMBER to color the torsion constraints. A sample of the kind of information it expects can be found in the ‘‘Constraint formats’’ section of this document.

Torsion angle information from X-PLOR can be converted for use with *noeshow*. Again, this is output information and, again, an example is provided in the ‘‘Constraint formats’’ section. The X-PLOR output information can be converted to the AMBER format with the *xplor2tors* program, the procedure for which is shown in the SHELL COMMAND SYNTAX section, above. X-PLOR torsion constraints are expressed as a desired angle and an allowable range about that angle. *Xplor2tors* has two options to control the transition from satisfied (green) to slightly violated (yellow) to badly violated (red). The ‘‘**–a** *degrees*’’ option controls how close to the edge of the range an angle can be before it is shown as slightly violated (default: 5 degrees). The ‘‘**–A** *degrees*’’ option controls how far beyond the edge of the range an angle can be before it is shown as badly violated (default: 10 degrees).

Basic usage

This section will discuss how to use *noeshow* to display NMR constraints in MIDAS. *Mardishow* is used in a very similar manner, and any differences will be discussed in the ‘‘Mardishow’’ section, below.

Noeshow is run from within MIDAS, so the first step is to display the model structure (or structures) in MIDAS. Once you’ve done that, invoke *noeshow* as shown in the MIDAS COMMAND SYNTAX section above. The parts of the command enclosed in brackets ([]) are optional and can be omitted.

Noeshow will read the file(s) you specify and generate and display a graphics object depicting the distance constraints and torsion angle constraints (if any). For the curious, graphics objects are discussed in the ‘‘Non-Molecular Graphics Objects’’ section of the MidasPlus manual, though it is not necessary to read that section to use *noeshow* effectively. The graphics object will be opened in the lowest unused model number.

Distance constraints will be depicted as lines connecting the atoms involved in the constraint. If one of the ‘‘atoms’’ is actually a pseudo-atom composed of several atoms, then the constraint will be drawn to the nearest of the atoms. The distance constraints will be colored according to how well they are satisfied:

Color	**Meaning**
red	distance exceeds both upper bounds
yellow	distance between the upper bounds
green	distance satisfies constraint
cyan	distance between lower bounds
blue	distance less than both lower bounds

Note that by default *noeshow* does not display the satisfied constraints (the **–a** flag causes them to be displayed).

Torsion angle constraints (if any) will be displayed as ‘‘cages’’ surrounding the central bond of the dihedral. The coloring of the cage is as follows:

Color	**Meaning (AMBER)**	**Meaning (X-PLOR)**
red	energy penalty > 4.0	violation > 10 degrees
orange	energy penalty > 1.0	violation < 10 degrees
yellow	energy penalty > 0.0	satisfied, but within 5 degrees of violated
green	energy penalty = 0.0	satisfied and not within 5 degrees of violated

Note that X-PLOR torsion coloring can be adjusted with options to the *xplor2tors* conversion script.

By default, *noeshow* will not display the satisfied torsion angle constraints. The **–A** option flag will cause them to be displayed as well.

Restricting display

By default, *noeshow* displays all the violated constraints present in the constraint file. At times, this may be more information than is desired. There are several methods of restricting the displayed constraints to those of interest, detailed below.

In the early stages of structure refinement, it may be desirable to show only the badly violated constraints (if you are uncertain about all the cross-peak assignments, for example). Giving *noeshow* the **–v** option will restrict the display to only badly violated constraints (*i.e.* those colored red or blue).

The simplest way to restrict constraint display to specific areas of the structure is to undisplay the parts of the structure where you don’t want constraints shown. *Noeshow* will not show constraints where one or both ends would be on an undisplayed part of the structure. To get this effect, you have to undisplay the undesired regions *before* running *noeshow*. You could then redisplay the whole structure after having generated the constraints of interest. Undisplaying parts of the structure after running *noeshow* has no effect on displayed constraints.

There is a somewhat more complicated method for restricting constraint display that offers finer-grain control than the above method. It involves using the relatively new **mark** and **makemark** commands of MIDAS. You may want to read the documentation for the above two commands in the MidasPlus manual if you are unfamiliar with them. These commands allow you to mark a set of atoms with a name. If you mark a set of atoms with the name ‘‘noemark1,’’ then *noeshow* will only show constraints where at least one end involves an atom in the marked set. If you also mark some atoms with the name ‘‘noemark2,’’ then *noeshow* will only show constraints where one end is in the first set and the other end is in the second set. For example:

> **makemark** noemark1

would create the mark name ‘‘noemark1’’ for use, and:

> **mark** noemark1 :12-14

would mark all atoms in residues 12 through 14 with the name ‘‘noemark1.’’ Running *noeshow* at this point would show only those constraints that had at least one end in residues 12 to 14.

Note that running the **mark** command with the same name a second time will *add* to the set of marked atoms, not replace the set with a new set. To replace, you would have to clear the mark with ‘‘**~mark** *name*’’ and then **mark** with the new set.

Structure Ensembles

To determine constraint satisfaction across an ensemble of structures displayed in MIDAS, *noeshow* uses exponential averaging (‘‘R to the minus sixth’’). The color-coded results are displayed on the highest-numbered open model. The same flags used to control pseudo-atom averaging (**–D** and **–x**) also control the ensemble averaging (see Options section). The **–c** flag, however, is ignored for ensemble averaging. If you want simple numerical averaging for the constraints, use ‘‘**–x** 1’’ instead of **–c**.

Normally, *noeshow* will give equal weight to each member of an ensemble. It is possible, however, to give the members unequal weightings. This could be useful, for example, if the ensemble had been generated by a program such as PARSE [1], which assigns a probability to each member of the ensemble. To indicate the weights to *noeshow*, there must be a line in each PDB file, as follows:

`USER  STRUCTURE WEIGHT` *weight*

where *weight* is any non-negative number. Note that there are *two* spaces after the `USER`. Structures lacking a line such as the above will be given a weight of 1.0 (and therefore, if no structures have the above line, all will have equal weight).

Showing possible H-H interactions

Noeshow has one capability that is designed to be used primarily without a set of distance constraints: it can show all hydrogens within a given cutoff distance of one or more designated hydrogens. The general procedure to do this is to use the marking mechanisms discussed in the ‘‘Restricting display’’ section to designate both the hydrogens of interest (marked with `noemark1`) as well as the neighboring hydrogens that should be considered for the cutoff (marked with `noemark2`). Then *noeshow* would be run with the ‘‘**–h** *cutoff*’’ option to specify the cutoff distance. *Noeshow* will draw a pink line between each pair of hydrogens satisfying the cutoff criterion and will show the corresponding distance at the midway point of the line.

Since *noeshow* is implemented as a *perl* script, it is not very fast at numerical calculations. It is therefore best if you limit the marked sets of atoms to the smallest sets that still have all the atoms of interest. For example, let’s say you have marked one or more hydrogens with `noemark1`, and you intend to run *noeshow* with a cutoff distance of 5 angstroms. You *could* simply not mark *any* atoms with `noemark2`, in which case *noeshow* would have to calculate the distance from every hydrogen marked with `noemark1` to *every other* hydrogen in the molecule. It would speed things up considerably to use `noemark2` to mark only those atoms that could possibly satisfy the cutoff, with a command such as:

mark noemark2 /mark=noemark1 za<5.1

which marks all atoms within 5.1 angstoms of the atoms in `noemark1` with the mark `noemark2` (this is one of the more advanced uses of atom specifiers; you may want to look at ‘‘Referencing Models, Residues, and Atoms’’ in the MidasPlus manual for further information, particularly the section ‘‘Atom Properties’’).

You need not mark one very small set of atoms and one larger set; you could mark approximately equally sized sets of atoms as long as the set size isn’t too large (perhaps a few dozen atoms each). For example, if you were working with a nucleic acid structure and marked all H1∗’s with both `noemark1` and `noemark2`, and then used a cutoff of about 5 angstroms, the resulting helical ‘‘chain’’ of H-H lines would show where the structure had fairly even distances as well as where there were ‘‘breaks’’ in the structure.

The −**h** option was designed to be used with no distance constraint file, and is the only option that allows the distance constraint file to be omitted from the command line. Nonetheless, you *can* specify a distance constraint file and it will be shown normally. Note that since the marking mechanism is normally used to restrict constraint display, any marks used to delimit H-H interactions will also restrict displayed constraints.

Tips on usage

Displaying multiple constraint sets

There are times that it is desirable to display several different constraint files separately. For example, if you’ve divided your constraints into positive constraints (distances determined from a cross-peak: the atoms must be at most a certain distance apart) and negative constraints (distances determined from the absence of a cross-peak: the atoms must be at least a certain distance apart), then you might like to show these sets separately at first and then together. To do this, you would run *noeshow* once for each constraint set you want displayed. *Noeshow* will open each successive graphics object in a different model number (the lowest available at the time). You can then undisplay specific graphics objects with the ‘‘**˜objdisplay** *modelnum*’’ command. ‘‘**objdisplay** *modelnum*’’ will, of course, redisplay them while ‘‘**close** *modelnum*’’ will dispose of them permanently.

Showing only torsion restraints

Showing torsion constraints with no distance constraints may seem problematic at first, since *noeshow* requires an argument specifying a file of distance constraints. However, this is simple to get around by specifying a distance constraint file that is empty. On UNIX systems (such as the SGI) there is a special system file that is always guaranteed to be empty: `/dev/null`. Therefore, invoking *noeshow* as:

`noeshow` [*options*] `/dev/null` *torsion_file*

would display only the torsion constraints in *torsion_file*.

Options

Many default behaviors of *noeshow* can be modified by command-line option flags. The options supported by *noeshow* are:

−**a** Show all constraints, including satisfied constraints.

−**A** Show all torsions, including satisfied torsions.

−**c** For multi-atom pseudo-atoms, measure constraint to geometric centers of atoms rather than doing exponential averaging. See also the −**D** and −**x** flags.

−**D** Don’t divide by number of atoms when doing exponential averaging of pseudo-atoms. See the −**x** flag for more info.

−**e** *lowbound*

Change the lower energy penalty threshold (where torsion colors change from green to yellow) to *lowbound*. Default is 1.0.

–E *hibound*
: Change the upper energy penalty threshold (where torsion colors change from yellow to red) to *hibound*. Default is 4.0.

–f *file*
: Store violated constraints in a file named *file*, sorted by magnitude of violation. See also the **–n** flag.

–F
: Don’t quit on encountering the first error; continue to report errors. This is useful when debugging problems using *noeshow*, which is discussed in the next section.

–h *cutoff*
: Show hydrogen-hydrogen interactions that are no more than *cutoff* angstroms apart. See the ‘‘Showing possible H-H interactions’’ section for a description of this option. If this option is specified, the constraint file command-line argument can be omitted.

–l
: Label atoms involved in unsatisfied constraints. For pseudo-atoms, only the atom that the constraint is drawn to (the closest) is labeled.

–L
: Expect LEaP nomenclature for atom names. LEaP is a module provided with AMBER. If you don’t know what LEaP is, you don’t care about this flag!

–n
: Used in conjunction with the **–f** flag; show signs as well as magnitudes of violated constraints.

–v
: Show only badly violated distance constraints.

–V
: Show only badly violated torsion constraints.

–x *exponent*
: Control the exponent used in exponential averaging of distances involving pseudo-atoms. Unless given the **–c** flag, *noeshow* does exponentially-weighted averaging of distances involving pseudo-atoms. If a constraint involves *n* atoms on one end and *m* atoms on its other end, then the weighted average distance is:

$$\left[\frac{\sum_n\sum_m |\vec{c}_n - \vec{c}_m|^{exponent}}{n \cdot m}\right]^{\frac{1}{exponent}}$$

where c_i is the coordinate position of the *i*th atom. The default exponent is –6. Use of the **–D** flag prevents the division by $n \cdot m$.

Errors

Noeshow displays error messages in the MIDAS reply area. Errors typically occur when there are problems matching atom names in the constraint file with those in the PDB file. Frequently in such cases many error messages are generated. In the Iris GL version of MIDAS, only the last few of these can be seen in the reply area.

If it is necessary to see all the error messages at once, this can be done by running *noeshow* outside of MIDAS . The first step is to get a PDB file from MIDAS to use as input to *noeshow*. In MIDAS, issue the command:

pdbrun cat > *inputfile*

This will save an annotated PDB file describing the current MIDAS display into a file called *inputfile*. Next, from a shell window, run:

/usr/local/midas/lib/midas/noeshow *options_used_in_Midas* < *inputfile*

The output from the above command will be a series of MIDAS commands. The output commands that start with **echo** are the ones that would display messages in the MIDAS reply area. Note that one of the output commands starts with **!rm**. This is to get MIDAS to remove a temporary file that *noeshow* produces.

You may wish to remove this temporary file yourself by typing in the output ‘‘rm’’ command to the shell window, less the leading ‘‘!.’’

Mardishow

Mardishow is used to display constraints that are in MARDIGRAS input format and can be used to display constraints in MARDIGRAS output format, though *noeshow* can be used to display MARDIGRAS output also, as outlined in the ‘‘Distance constraint preparation’’ section, above. This section will outline the salient differences between using *noeshow* and *mardishow*.

Mardishow displays constraints in the two MARDIGRAS formats directly, *i.e.* the constraints do not need to be converted to an intermediate format, unlike the procedure for *noeshow*.

Since neither MARDIGRAS input nor MARDIGRAS output format constraints have an ‘‘energy well’’ associated with them, displayed constraint violations are color-coded differently than in *noeshow*. When showing MARDIGRAS output format constraints, *mardishow* constructs a ‘‘fake energy well’’ by using the given upper and lower bounds to define the ‘‘floor’’ of the energy well. The ‘‘edges’’ of the well are placed 0.5 angstroms beyond the given bounds (this margin can be changed with the **–d** option). Coloring is then as in *noeshow*. For MARDIGRAS input format constraints, the coloring scheme is simple: if the distance is less than 5 angstroms then the constraint is colored green, if between 5 and 6 angstroms yellow, and greater than 6 angstroms red.

Mardishow does not support restricting constraint display, either through the ‘‘marking’’ mechanism of *noeshow* or by undisplaying parts of the molecule. Bad things will happen if you try.

Pseudo-atom constraints are measured to the heavy atom connected to the hydrogens, rather than involving any kind of averaging.

Mardishow is designed to work with a single model structure rather than an ensemble. The only way to use *mardishow* in conjunction with an ensemble is to display one structure of the ensemble at a time.

Mardishow supports a subset of the flags of *noeshow*, namely: **–a**, **–f**, **–F**, **–l**, and **–n**. In addition, it has the following flags:

–d *amount*
When displaying MARDIGRAS output format constraints, make the outside bounds of the ‘‘energy well’’ *amount* angstroms from the inner bounds (which are given in the constraint file).

–s Enable ‘‘stereospecific’’ constraints. If a constraint names a hydrogen involved in a rotamer and this option is given, then the constraint will be measured and drawn to that hydrogen. Otherwise, the constraint will go to the connected heavy atom.

FORMAT EXAMPLES

AMBER INTERFACE format

```
restraint/ at1=2:HA /at2=2:HN \
/r1=2.664 /r2=3.164 /r3=3.252 /r4=3.752 /k2=1.000 /k3=1.000 \
```

and/or

```
restraint / at1=1:H23 /at2 =1:H25 \
 /r1 = 0.0/r2=1.8000/r3=2.5980/r4=4.5980/k2=10.0/k3=10.0 \
 /grpat1 =1:H23,1:H24
```

and/or

```
restraint / at1=275 /at2 =290 \
 /r1 = 0.0/r2=1.8000/r3=2.5980/r4=4.5980/k2=10.0/k3=10.0 \
 /grpat1 =275, 276
```

AMBER SANDER namelist format

```
 &rst  iat = 37, 35, iresid = 1, atnam(1)='H8   ',atnam(2)='H2''2',
       r1 = 0.000, r2 = 2.895, r3 = 2.945, r4 = 4.945,
       rk2 =10.000, rk3 =10.000, &end
```

and/or

```
 &rst  iat = 108, -1,
       igr2 = 143,144,145,0,
       r1 = 0.000, r2 = 3.315, r3 = 3.395, r4 = 5.395,
         rk2 =10.000, rk3 =10.000, &end
```

X-PLOR distance constraint format

```
    assign (resid    6 and  name HB   )(resid    7 and  name HN   ) 4.0 2.2 1.0
    assign (resid   34 and  name HB#  )(resid   39 and  name HN   ) 4.0 3.7 2.5 !
    assign (resid   37 and  name HB1  )(resid   39 and  name HN   ) 4.0 2.2 1.0
    assign (resid   23 and  name HB#  )(resid   26 and  name HB#  ) 4.0 5.2 4.0 ! !#
    assign (resid   22 and  name HB2  )(resid   24 and  name HN   ) 4.0 2.2 1.0
    assign (resid   20 and  name HB#  )(resid   22 and  name HN   ) 3.0 2.7 2.0 !
```

DIANA low and high bound constraint format

```
     14 TYR  HN     10 VAL  O       2.00  1.00E+00 # H-Bond
     13 ARG  HN     14 TYR  HN      2.90  1.00E+00 # N(i)-N(i+1)
     11 VAL  HA     14 TYR  HN      5.00  1.00E+00 # ha(i)-hn(i+3)
     14 TYR  HA     17 ALA  QB      3.80  1.00E+00 # ha(i)-hb(i+3)
     14 TYR  HA     29 LEU  QD1     3.80  1.00E+00 # helix1-helix2
     14 TYR  QR     15 VAL  QG2     7.80  1.00E+00 # sequential
     18 LEU  QD1    14 TYR  QR      7.80  1.00E+00 #
     24 ASP  O      28 ALA  HN      1.80  1.00E+00 # hbond
     25 GLY  HA1    28 ALA  HN      5.00  1.00E+00 # aH(i)-HN(i+3)
```

AMBER torsion information format

```
     ------------------------------------------------------------------------------


     Final Restraint Analysis for coords: min8.rst


     Restraints, deviations, and energy contributions:    pencut =     .00

     ------------------------------------------------------------------------------
         First atom        Last atom      curr. value target   deviation  penalty
     ------------------------------------------------------------------------------
                   .
                   .
                   .
      CA   CYX   16 --  CB   CYX   16:    212.978   210.000     2.978      .863
      CA   SER   17 --  CB   SER   17:     29.656    30.000      .344      .011
      N    SER   19 --  CA   SER   19:   -158.920   -60.000      .000      .000
      CA   SER   19 --  CB   SER   19:     41.668    90.000      .000      .000
      CA   CYX   20 --  CB   CYX   20:    188.804   210.000      .000      .000
      N    ARG   23 --  CA   ARG   23:    -59.704   -60.000      .296      .009
```

X-PLOR torsion information format

```
    ---------------------------------------------------
    Number of dihedral angle constraints=   130
      overall scale =  200.0000
     ========================================
          3    GLN  C
          4    LYS  N
          4    LYS  CA
          4    LYS  C
     Dihedral= -126.639  Energy=    0.000 C=    1.000 Equil= -157.500 Delta=   -0.86
    1
     Range=  30.000 Exponent=  2
     ========================================
          4    LYS  C
          5    THR  N
          5    THR  CA
          5    THR  C
     Dihedral= -111.196  Energy=    0.000 C=    1.000 Equil= -120.000 Delta=    0.00
    0
     Range=  30.000 Exponent=  2
     ========================================
```

FEEDBACK

Feel free to mail suggestions or questions to pett@cgl.ucsf.edu .

NOTE

Noeshow and *mardishow* are *perl* scripts and, as such, require a *perl* command interpreter in order to work. Most systems already have a *perl* interpreter installed. If yours does not, one is provided in the MidasPlus distribution. Consult the MidasPlus Installation Guide for further details.

REFERENCES

[1] Ulyanov NB; Schmitz U; Kumar A; James TL.
Probability assessment of conformational ensembles: sugar repuckering in a DNA duplex in solution. *Biophysical Journal*, 1995 Jan, **68**(1):13-24.

AUTHOR

Eric Pettersen
UCSF Computer Graphics Laboratory

ACKNOWLEDGEMENTS

Noeshow and *mardishow* could not have been written without the assistance and encouragement of Dr. Shauna Farr-Jones.

Xplor2amber was written with the assistance of Dr. Bryan Finn and Dr. David Schweisguth, and *xplor2tors* with the assistance of Dr. Brian Jones. *Diana2amber* was written with the assistance of Dr. Michael Zawrotny.

NAME

pdb2group – generate Midas group file from Protein Data Bank (PDB) file

SYNOPSIS

pdb2group –a *anchor_atom* **–3** *n3_atom* **–2** *n2_atom* [**–d** *group_description*] [*PDB_file*]

DESCRIPTION

Pdb2group takes a PDB file with explicit CONECT records (even for standard residues such as amino acids) and generates a ‘‘group’’ file suitable for use with *midas*(1). The group file is used by MIDAS commands such as **swapaa**, **swapna**, and **addgrp**, and defines the connectivity and relative position of a group of atoms. The generated group file is written to standard output.

FILE FORMAT

A group file consists of a series of text lines, and can be divided into four parts. The first part is the title and is a single-line description of the group, *e.g.*, 4-OH-phenyl. The second part is the list of atoms in the group and is a series of lines of the form

```
new atom_name x y z
```

The atom names should be in uppercase letters and need not be unique. The Cartesian coordinates specified by these lines are generally ignored by most programs, and may be set to (0, 0, 0) if they are not conveniently available. The third part of file group file is a separator and is a single line:

```
read internal coordinates for new group
```

The fourth and final part of the group file is the connectivity and relative position description and is a series of lines of the form

```
mode atom1 atom2 atom3 atom4 bond_length bond_angle dihedral_angle
```

The mode field of the line is a single character, either ‘+’ or ‘=,’ and is generally ignored. Each of the four atom fields may be either a number, or `n3`, `n2`, or `n1`. If it is a number, it refers to an atom in the group. The atoms listed in the group are numbered sequentially, with the first atom being 0. If the atom field is one of the special strings, it refers to an atom in the molecule to which the group will be attached. Atom 0 is always attached to atom `n3`. The bond length is the distance between `atom1` and `atom2`. The bond angle is the angle formed by `atom1`, `atom2`, and `atom3`, with `atom2` as the vertex. The dihedral angle is the angle formed by all four atoms, with `atom2` and `atom3` as the internal vertices.

An example of a group file, for 4-hydroxy-phenol, follows:

```
4-OH-phenyl
new C  0.000    0.000   0.000
new C  0.000    0.000   0.000
new C  0.000    0.000   0.000
new C  0.000    0.000   0.000
new O  0.000    0.000   0.000
new C  0.000    0.000   0.000
new C  0.000    0.000   0.000
read internal coordinates for new group
+ 1 0 n3 n2  1.4   120.0   60.0
+ 2 1  0 n3  1.4   120.0  180.0
+ 3 2  1  0  1.4   120.0    0.0
= 4 3  2  1  1.37  120.0  180.0
+ 5 3  2  1  1.4   120.0    0.0
+ 6 5  3  2  1.4   120.0    0.0
+ 6 0  1  2  1.4   120.0    0.0
```

Group file format is also described in detail in ‘‘Chemical Group Description Files’’ in part 3 of the MidasPlus manual.

COMMAND ARGUMENTS

The required command-line arguments to *pdb2group* are the names of *anchor_atom*, *n3_atom*, and *n2_atom*. The *anchor_atom* is the first atom to be listed in the computed group file. *N3_atom* and *n2_atom* refer to atoms in *PDB_file* and are used to compute bond and dihedral angles between the group and the rest of the molecule; they do not appear in the group file. Atoms in the PDB file which are connected to *anchor_atom* only through *n3_atom* will not appear in the group file. The PDB file must contain only one residue.

The title of the group file is the *group_description*, if given, or the PDB file name otherwise. If the PDB file is read from standard input, the title is ‘‘−.’’

EXAMPLE

Although lysine is provided as a standard group, for purposes of illustration the derivation of a group file describing a lysine side-chain is shown here.

Creating the PDB input file

Creating the PDB input file by hand is a tedious procedure, but may be necessary if you have no PDB file containing the desired group, and no model-building tools are available. If so, refer to ‘‘Protein Data Bank Format’’ in part 3 of the MidasPlus User’s Manual for the exact format description. You would need to create a file containing a single residue containing the group of interest, with explicit CONECT records for all atoms in the file.

A much more pleasant alternative, if you have a PDB file on hand with the group in it, or can make a basic PDB file containing the group that MIDAS can display, is to use MIDAS to generate the final PDB input file. The procedure would be to open the starting PDB file in MIDAS, limit the display to the residue of interest, and then run the command:

pdbrun conect nouser cat > *final_PDB_input_file*

To continue the example of generating a lysine side-chain group description file, if the standard PDB entry 1gcn were open in MIDAS as model 0, then the following commands would be used:

show #0:12	*display only lysine residue*
pdbrun conect nouser cat > lysgroup.in	*save PDB file with CONECT records into ‘‘lysgroup.in’’*

Running pdb2group

Referring to the SYNOPSIS section above, the *anchor_atom* specified on the command line should be the first added atom of the new group when the group is added to an existing structure. *N3_atom* and *n2_atom* are atoms that are not in the new group, but that the new group connects to. *N3_atom* connects directly to the *anchor_atom*, and *n2_atom* connects to *n3_atom*.

So in the lysine group example, the appropriate *pdb2group* command would be:

```
pdb2group −a CB −3 CA −2 C −d Lysine lysgroup.in > lysgroup.out
```

SEE ALSO

MidasPlus User’s Manual

AUTHORS

Conrad Huang, UCSF Computer Graphics Laboratory

NAME

pdb2site – convert a PDB file into a DMS site file

SYNOPSIS

pdb2site [–**i** *PDB_file*] [–**o** *site_file*]

DESCRIPTION

Pdb2site reads a Protein Data Bank file and converts it into a site file, suitable for use with the –**i** flag of *dms*(1). The –**i** argument specifies the input PDB file. If no input file is given, *pdb2site* will read from standard input. The –**o** argument specifies the output site file. If no output file is given, *pdb2site* will write to standard output.

Pdb2site is frequently used in conjunction with MIDAS to generate a site file for use with *dms*(1). The procedure would be to display the atoms of interest in MIDAS, and then type the command:

pdbrun pdb2site –o *site_file*

This would generate a file called *site_file* which is in the format appropriate for a *dms* site file.

SEE ALSO

dms(1), MidasPlus User's Manual
Protein Data Bank, Atomic Coordinate and Bibliographic Entry Format Description.

AUTHOR

Conrad Huang
Computer Graphics Laboratory
University of California, San Francisco

NAME

pdbopen – Midas delegate for browsing and opening PDB entries

MIDAS COMMAND SYNTAX

`Command:` **pdbopen** [–**directory** *dir*] [–**index** *file*]

DESCRIPTION

Pdbopen is a MidasPlus delegate that presents an X/Motif user interface that allows users to browse through an index of the Protein Data Bank (PDB) coordinate entries, examine information associated with each entry, and open entries in MIDAS. *Pdbopen* may be invoked with the following flags:

–directory
Set the directory to use as the PDB repository. The default directory is */mol/pdb*.

–index Set the file to use as the PDB entry index. The default index file is found in *index/entries.idx* in the PDB repository directory.

Pdbopen reads the PDB index file and then displays two windows: an index browser and an entry inspector. Each PDB entry consists of several fields, including its PDB ID code, compound information, and ascession [sic] date. The browser lists short forms of PDB entries of interest while the inspector displays all the fields of a selected PDB entry.

The index browser consists of a text list, a menu bar, and two buttons. The text list consists of PDB entries, one per line, displayed in a scrollable region. At most one entry in the text list may be selected using the mouse, and additional information about that entry is displayed in the entry inspector. The menu bar has three entries: **Sort**, **Show** and **Hide**. The **Sort** menu will sort the entries displayed in the text list according to the chosen field; the **Show** and **Hide** menus will show and hide the chosen fields in the short form used to display entries in the text list. The **Open** button sends a command to MIDAS to open the selected entry. The PDB entries displayed in the text list may be pruned using the entry inspector (see below); the **Display All** button adds all entries from the index file back into the text list so that another search for ‘‘interesting’’ entries may be done.

The entry inspector consists of a number of text fields and several buttons. Each text field corresponding to a field in a PDB entry and, when an entry is selected in the index browser, displays the value of the field. The **Open** button sends a command to MIDAS to open the entry with displayed ID code. The **Find**, **Undo** and **Clear** buttons are used to prune entries from the text list in the index browser. The **Find** button searches through the list of entries in the text list and removes any whose fields do not match the values given in the inspector text fields. For the **Resolution** and **Zero** fields, the values are compared numerically; a relational operator may precede the numeric value ‘‘<= 2’’ means less than or equal to two) and a missing operator implies equality. For all other fields, the values are treated as case-independent regular expressions; a field is defined to match the value if any part of the field string matches the regular expression (see Regular Expression Examples below). The **Undo** button will undo the effect of the previous **Find**. Two **Undo**’s in a row is the same as doing nothing. The **Clear** button will clear all the text fields. Because an empty value in a text field matches anything, **Clear** is useful if the subsequent **Find** only searches on one or two fields. The **Help** button brings up a window displaying these instructions.

REGULAR EXPRESSION EXAMPLES

Pdbopen uses case-independent *egrep*(1)-style regular expressions. The simplest regular expression is a string with no magic characters. For example, a value of ‘‘serine’’ will match the field ‘‘HYDROLASE(SERINE PROTEINASE),’’ because the field contains the string ‘‘serine.’’ Magic characters (*i.e.*, ^$.∗+()[]|\) are used to construct more complex regular expressions, such as ‘‘[rd]na’’ which will match all fields containing either the string ‘‘rna’’ or ‘‘dna.’’ Finally, the following expression matches A-T and T-A transitions (in the Compound field for DNA): ‘‘[(CATGUP]AP∗T[)CATGUP]|[(CATGUP]TP∗A[)CATGUP].’’

SEE ALSO

egrep(1)

AUTHOR

Conrad Huang, Computer Graphics Laboratory, UCSF

NAME

pdbrun5to6 – convert PDBRUN version 5 files to version 6

SYNOPSIS

pdbrun5to6 [**–o** *output-filename*] [*input-filename*]

DESCRIPTION

Pdbrun5to6 converts PDBRUN version 5 (or less) annotated PDB files to version 6. Version 5 files were generated by version 2.0 and earlier of MIDAS's **pdbrun** command.

Pdbrun5to6's primary use is to convert saved PDBRUN files for renderers that require PDBRUN version 6.

Another use is to get **pdbrun**-invoked programs that output PDBRUN version 5 files to work with MidasPlus 2.1. Most **pdbrun**-invoked programs do *not* output PDBRUN files. Instead, they either output a series of MIDAS commands (*e.g. rainbow*) or have no output (*e.g. conic*). A few, however, do output PDBRUN files, which are then piped to other programs as input. *Neon*, for example, outputs a PDBRUN file which is then piped to *conic* as input. To make an ''old'' PDBRUN-producing program work with MidasPlus 2.1, its alias should be changed from:

alias ˆ*old-program* **pdbrun** *old-program*

to:

alias ˆ*old-program* **pdbrun pdbrun6to5 |** *old-program* **| pdbrun5to6**

Programs that do not produce PDBRUN output need only employ *pdbrun6to5* to work with MidasPlus 2.1. Consult the *pdbrun6to5* manual page for details.

SEE ALSO

pdbrun6to5(1), midas(1), MidasPlus User's Guide

AUTHOR

Greg Couch
UCSF Computer Graphics Laboratory

NAME

pdbrun6to5 – convert PDBRUN version 6 files to version 5

SYNOPSIS

pdbrun6to5 [**–o** *output-filename*] [*input-filename*]

DESCRIPTION

Pdbrun6to5 converts PDBRUN-annotated PDB files, as generated by MIDAS's **pdbrun** command, from version 6 to version 5. Its primary use is to make version 2.1 of MIDAS work with older **pdbrun**-invoked programs.

The MIDAS **pdbrun** command executes a program and provides a PDBRUN-format file (see Appendix 1) as standard input to that program. Any standard output of the program is interpreted as MIDAS commands and executed. Such a program that used PDBRUN version 5 can be made to work with MidasPlus 2.1 by changing its invocation alias from:

alias ˆ*old-program* **pdbrun** *old-program*

to:

alias ˆ*old-program* **pdbrun pdbrun6to5** *old-program*

A few **pdbrun** programs produce modified PDBRUN files as their output. Their output is then piped to another program as input. An example of such a program is *neon*, which pipes its output to *conic* for rendering. Such programs using PDBRUN version 5 must employ *pdbrun5to6* in their updated aliases to work with MidasPlus 2.1. Consult the *pdbrun5to6* manual page for details.

SEE ALSO

pdbrun5to6(1), midas(1), MidasPlus User's Guide

AUTHOR

Greg Couch
UCSF Computer Graphics Laboratory

NAME

ps2illustrator – convert Midas copy file (PostScript) to Adobe Illustrator format

SYNOPSIS

ps2illustrator < *copy-file* > *Illustrator-file*

DESCRIPTION

Ps2illustrator reads in a PostScript file generated by the MIDAS **copy** command and converts it to an Adobe Illustrator format file for inclusion in desktop publishing documents.

SEE ALSO

stereops2illustrator(1), midas(1)

BUGS

Depends on the *csplit* and *nawk* programs that might not be available on all platforms.

AUTHOR

Scott Dixon
SmithKline Beecham Pharmaceuticals

NAME

rainbow – ‘‘rainbow’’-color molecule chains

MIDAS COMMAND SYNTAX

`Command:` **rainbow** [*modelnumber ...*]

DESCRIPTION

Rainbow colors each chain in the specified model(s) (all models if none specified) from red to blue (transiting through yellow, green, and cyan in the process). This may be of some assistance in tracing tangled molecule chains. Since standard PDB files are ordered from N- to C-terminus, this means that *rainbow* colors the N-terminus red and the C-terminus blue.

By default *rainbow* projects its color scheme across the entire model(s) requested, even if parts of those models are not displayed. This is because the **rainbow** command in MIDAS is an alias that expands to ‘‘**pdbrun** all nouser rainbow,’’ which causes all model information to be sent to *rainbow*. To restrict the coloring to just the displayed regions, you would have to type ‘‘**pdbrun** nouser rainbow’’ at the MIDAS command prompt.

LIMITATIONS

Rainbow will frequently not work correctly if the device option *colormap* is ‘‘on’’ (from the MIDAS command **devopt colormap on**). In colormap mode there is only a fixed number of colors available, frequently less than *rainbow* needs to make a spectrum.

NOTE

Rainbow is a *perl* script, and as such needs a copy of the perl interpreter in the directory */usr/local/bin*. *Perl* is supplied with all supported platforms of the MidasPlus distribution. However, if you wish to obtain the most recent *perl* version, the perl distribution can be anonymously ftp’ed from jpl-devvax.jpl.nasa.gov (137.79.113.100), and most likely will be found in the pub/perl.4.0 subdirectory there.

BUGS

The model numbers specified on the *rainbow* command line cannot be preceded by hash marks (#) as they are in most other MIDAS commands, because the *rainbow* command is passed through a shell interpreter; the standard shell treats hash marks and anything following hash marks as comments and strips them.

AUTHOR

Eric Pettersen
UCSF Computer Graphics Laboratory

NAME

ribbonjr – generate ribbon representation of proteins or nucleic acids

SYNOPSIS

ribbonjr [*options*] [*PDB-file* [*output-file*]]

DESCRIPTION

Ribbonjr reads a Protein Data Bank file and generates a ribbon representation of the molecule. The ribbon position and orientation are controlled by two atoms: the guide atom and the twist atom. For amino acids, the guide atom is the α-carbon and the twist atom is the carbonyl oxygen. This choice of atoms makes the ribbon run approximately parallel to the peptide plane. For nucleotides, the guide atom is C5∗ and the twist atom is C1∗. This choice makes DNA ribbons approximately perpendicular to the helical axis.

The PDB file may carry extra atom information such as color and radius in the same fashion described in the *conic*(1) manual page (under section ‘‘Coloring the Molecule’’). The color of the ribbon is the same as the color of the guide atoms. If no PDB file is specified or ‘‘–’’ is provided as the PDB file name, then standard input is used.

PDB files generated by some programs may not conform to the PDB standard. The utility *dnacheck*(1), provided with the MidasPlus distribution, corrects many of the common problems found in such files. Consult the *dnacheck* manual page for further details. Also, for proteins, *ribbonjr* requires the information contained in HELIX and SHEET PDB records in order to correctly display secondary structure. If a PDB file lacks such records, the *ksdssp*(1) utility can be used to generate the records. The *ksdssp* manual page contains further details.

Unless their display is explicitly suppressed with the **–a** command line option, non-mainchain atoms and bonds are shown as balls and sticks. Mainchain atoms are not individually depicted unless they are connected to a displayed non-mainchain atom. The mainchain atoms for amino acids are N, CA, C and O. The mainchain atoms for nucleotides are P, O1P, O2P, O3P, C5∗, O5∗, O3∗ and C3∗. The bonds are derived either from CONECT records if they exist in the PDB file, or from drawing templates found in MIDAS template directories.

The ribbon representation from *ribbonjr* may be in one of several formats: **midas**, **inventor**, **rayshade**, **pov**, **screen**, and **tiff**. MIDAS object format is described in detail under ‘‘Non-Molecule Graphics Objects’’ in Part III of the MidasPlus manual. **Inventor** format is the standard Silicon Graphics format; **inventor** output may be viewed using *ivview*(1), *SceneViewer*(1), or *showcase*(1). **Rayshade** and **pov** output are for use with raytracers of the same name. The format description may be found with in the raytracer documentation. **Screen** is not actually a file format; instead, the ribbon representation is displayed directly on screen using the OpenGL or Silicon Graphics GL library: there is no ‘‘output file.’’ **Tiff** output requires that an output file be specified (though it needs to use the screen to calculate the image). The default output format is **screen**, even if an output file is specified (except on NEXTSTEP systems, where neither **screen** nor **tiff** formats are supported). To use other formats, use the **–f** command line flag (see below).

‘‘Capturing Screen Images’’ in Part III of the MidasPlus manual discusses saving, converting, and printing *ribbonjr* images.

COMMAND-LINE FLAGS

The command-line flags interpreted by *ribbonjr* are:

–a Do not display any atoms using balls and sticks. By default, all non-mainchain atoms and bonds are displayed.

–b *r,g,b*[*,r,g,b*[*,r,g,b*]]
Set the background color in RGB format (each component ranges between 0 and 1). If one set of RGB values is given, then the entire background is set to that color. If two sets are given, the background color is interpolated from the first to the second color starting at the top going downwards. If three sets are specified, the background color interpolates from the first color at the top of the image to the second color in the middle, to the third color at the bottom. This option is only meaningful when the output format is **screen**. The default background color is black (0,0,0).

−c *color-scheme*
: Select the color scheme to use for the ribbon. The supported color schemes are **residue**, **structure**, and **xsection**. In the **residue** scheme, the ribbon corresponding to a residue will have the same color as the residue's guide atom. In the **structure** scheme, the color is determined by the secondary structure type (helix, strand or turn); the actual colors are specified using the **−H**, **−S**, and **−T** flags (see below). In the **xsection** scheme, the color is determined by the residue cross-section type (see **−x** flag below). The default color scheme is **residue**.

−e *shell-command*
: Execute the shell command when the image has finished drawing. *Ribbonjr* will not exit after executing the shell command (see the **−p** flag below).

−f *output-format*
: Select the output format. The supported formats are **midas**, **inventor**, **rayshade**, **pov**, **screen**, and **tiff**. The default output format is **screen** (or **midas** if drawing to the screen is not supported on the system, *e.g.*, NEXTSTEP systems). The **rayshade** and **pov** formats are input to widely available raytracers (see also **−x** option). Unfortunately, due to the large triangles used to render the bicubic patches, shadows generated by these raytracers look wrong in certain places. Due to this limitation, the **pov** output actually turns off shadow computation.

−g Display guide atoms at their true Cartesian coordinates. Normally, when atoms are displayed, guide atoms are displayed at interpolated locations on the ribbon rather than at their Cartesian coordinates, which may not be on the ribbon since the ribbon is not guaranteed to pass through the guide atom. Displaying guide atoms on the ribbon typically makes the image look better because side chains are attached to the ribbon itself rather than hanging in space.

−h Turn off half-bond mode. Normally, bonds are drawn as two cylinders, each with the color of the closer atom. Turning off half-bond mode makes *ribbonjr* use half as many cylinders, making interactive performance considerably better.

−l *ambient,diffuse,specular,shininess,*
: Set the lighting parameters of ribbons, balls, and sticks. The values are the ambient, diffuse and specular reflectance, and shininess of the material. All values should be between 0 and 1. The default values are 0.5, 0.5, 0.5 and 0.5 respectively.

−m *draw-mode*
: Select the ribbon representation. The supported modes are **3D** and **flat**. The **3D** mode produces Jane Richardson-type ribbon representation of the molecule. The **flat** mode produces two curves corresponding to the edges of a fixed-width flat ribbon, and regularly spaced line segments connecting the curves. The default representation is **3D**.

−n Do not draw a border around the image.

−o Do not display any MIDAS graphics objects present in the input. By default, MIDAS objects are displayed. The lines composing the object are rendered in *ribbonjr* as cylinders with half-spheres capping the ends. By default, these cylinders have an extremely small radius, so that they look like lines. The cylinder radius can be controlled with the **−O** flag.

−p When output is in **screen** or **tiff** format, do not wait for the user to click the mouse button. By default, the user must click the left mouse button before the *ribbonjr* window is closed.

−r *level* Sets the refinement level to *level* . The refinement level affects the output quality from *ribbonjr*. The level ranges from 0 to 10, with the default being 0. The higher the level number, the better the quality. Levels greater than zero increase the sphere subdivision and levels greater than 5 cause the image to be antialiased using the accumulation buffer as well. In the process of using the accumulation buffer, the image will be drawn to the screen several times before the final image is displayed.

−s *count*
: Select the number of segments used to represent one residue. By default, the ribbon for a residue

is divided into 10 segments. For interactive use in **Inventor** format, this number should probably be set lower.

–t Make the background color transparent. This works with the **tiff** format to add an alpha channel to the file, so the resulting image can be composited onto other backgrounds. *This option depends on OpenGL support for ''destination alpha'' and consequently does not work on all systems.*

–x *filename*
Read cross-section information from *filename* in addition to the default system file. Cross-section specifications in file *filename* will override those from the system file. If there is no file named *filename* in the current directory, *ribbonjr* will search for the file in the system directory. The cross-section file format is described in the CROSS-SECTION FILE section, below.

Several cross-section files are provided in the *ribbonjr* system directory. They are:

xs.default – the default when no cross-section file is specified; it produces a ribbon with an elongated-ellipsoid cross-section that is smooth along the ribbon's width and length.

xs.rayshade – a cross-section file appropriate for use when generating output that will then be used in a ray-tracer program such as **pov** or **rayshade**. The polygonal sections used to render the ribbon are much more finely subdivided than normal.

xs.rect – a rectangular cross-section with sharp edges. Produces a ribbon with flat faces and sides.

xs.ribbed – a ''novelty'' cross-section file. Uses an elongated-ellipsoid cross-section that instead of being smooth is composed of a series of short flat segments. The ribbon appears similar to the default ribbon but with ''ribs'' running down the ribbon's length.

–A *scale-factor*
Specify the scale factor between the radius of an atom and the radius of the sphere that represents the atom in ball-and-stick mode. The default scale factor is 0.2. For some images, you may also want to use the **–r** option to increase the sphere subdivision and thereby get better-looking ''balls.''

–B *scale-factor*
Specify the scale factor between the radius of a bond's atoms and the radius of the cylinder that represents the bond in ball-and-stick mode. The default scale factor is 0.2. For some images, you may also want to use the **–r** option to increase the sphere subdivision and thereby get better-looking ''balls.''

–D *width,height*
Specify the window dimensions when output format is **screen** or **tiff**. The default dimensions are obtained from USER records in the PDB input file. If no dimensions are specified in the PDB file, they are set to 645x484.

–E *helix-extension*
When *ribbonjr* constructs the ribbon representation, it uses the guide atom coordinates as the basis of its control points. When using B-splines, the ribbon corresponding to a helix tends to be very slender, because the spline does not interpolate through the control points. To produce helices of a reasonable radius, the coordinates of helical guide atoms are translated by a short distance away from the helical axis. When using other types of splines, the helices are not as compressed as using B-splines. The default extension depends on the type of spline used to produce the ribbon (see **–R** flag below). The default extension is 1.5 angstrom for B-splines and 0.5 angstroms for all other types.

–F Use full screen mode. Set the image size to use the entire screen.

–G *guide-fraction*
When *ribbonjr* constructs the ribbon representation, it uses the guide atom coordinates as the basis of its control points, one per residue. For each residue, the control point is between its guide atom

and the guide atom of the following residue. Normally, the control point is exactly halfway between the two guide atoms (*guide-fraction* = 0.5). This option is most useful when the ribbon **must** go through the guide atoms. In this case, the guide fraction should be set to zero, the helix extension (see **–E** flag above) should be set to zero, and an interpolating spline type (see **–R** flag below) should be selected.

–H *red,green,blue*
Set the helix color in color scheme **structure**. The default helix color is (1,0,0). Use of this flag implicitly sets *color-scheme* to **structure** (see the **–c** flag).

–J *scale_factor*
When using the **na_sugar** option (see **–P** below), the oxygen atom in the sugar ring is artificially enlarged by *scale_factor*. The default scale factor is 1.5.

–L *x,y* Specify the window location when output format is **screen** or **tiff**. 0,0 is the lower left corner. If no location is specified, the user gets to choose the screen location using the mouse when the window is created.

–N Show normals if output is in **midas** format.

–O *radius*
Specify the radius of cylinders, in angstroms, used to render MIDAS graphics objects (see the **–o** flag).

–P *polygon_option*
Display special polygons in addition to atoms, bonds, and ribbons. Currently two types of special polygons are supported: **na_base** and **na_sugar**. The **na_base** polygons are polygons drawn above and below the rings of the bases in nucleotides. When this option is selected, the atoms in the rings are automatically constrained to lie in a plane, in order to avoid nonplanar ring surfaces. The colors of the polygons are the same as the colors of the N1 and N9 atoms. The **na_sugar** polygons are polygons drawn around the sugar rings in nucleotides. Because the five-membered sugar rings are nonplanar, each ring is covered by three triangles on either side. All three triangles have as one vertex the oxygen atom in the sugar. The oxygen can be highlighted further by artificially increasing its radius (see **–J** above). The polygon color is the same as that of atom C1∗.

–R *spline-type*
Ribbonjr constructs ribbons using splines that pass near or through control points, whose coordinates are based on the positions of guide atoms. Different types of splines may be used. The supported types are **bspline**, **hermite**, **bezier**, and **cardinal**. The default spline type is **bspline**. See the **–E** flag above for more information about helical representation.

–S *red,green,blue*
Set the strand color in color scheme **structure**. The default strand color is (0,1,0). Use of this flag implicitly sets *color-scheme* to **structure** (see the **–c** flag).

–T *red,green,blue*
Set the turn color in color scheme **structure**. The default turn color is (0,0,1). Use of this flag implicitly sets *color-scheme* to **structure** (see the **–c** flag).

–W Force MIDAS to wait until *ribbonjr* has exited before continuing.

–Z *debug-level*
Specify the debug level. The output is probably somewhat cryptic.

CROSS-SECTION FILE

The Jane Richardson-type ribbon representation uses different shapes for different secondary structure types. For example, turns are commonly represented as thin tubes, and helices and strands are represented as wide ribbons. In this case, the cross-section of a turn is a small circle, and those of helices and strands are elongated ovals. *Ribbonjr* reads the standard cross-section descriptions from a default file, but the user may override them by supplying his own cross-section file (see the **–x** command-line flag).

The grammar for cross-section and interface descriptions is shown below (optional elements are surrounded by square brackets and user-selected values are in italics).

```
[ faceted | edged | segmented ] path name = {
        x1, y1 [ (r1, g1, b1) ]
        x2, y2 [ (r2, g2, b2) ]
        …
        xN, yN [ (rN, gN, bN) ]
}

[ faceted | edged | segmented ] spline kind(count) name = {
        x1, y1 [ (r1, g1, b1) ]
        x2, y2 [ (r2, g2, b2) ]
        …
        xN, yN [ (rN, gN, bN) ]
}

interpolate name1 name2
point name1 name2
```

The first form describes a cross-section using a connected, closed path. The name of the cross-section type is *name*, and the list of coordinates are supplied in *x* and *y*. *Name* must start with an alphabetic character and can only contain alphanumeric characters, ‘‘_’’ (underscore), and ‘‘.’’ (period). Each coordinate also optionally has a color, whose value is specified by *r*, *g*, and *b*, associated with it (for use with **xsection** color scheme − see the **−c** flag). The last coordinate of a cross-section is automatically connected to the first coordinate by *ribbonjr*. The second form describes a cross-section using a series of spline control points. The type of spline is given by *kind*, which may be one of **bspline**, **bezier**, **hermite**, or **cardinal**. The number of intermediate points generated on the spline is given by *count*. The name, coordinates, and colors are the same as in the path specification.

Both cross-section specifications may be prefixed by one of three keywords: **faceted**, **edged**, or **segmented**. A **faceted** ribbon will not be smooth along its length. As it curves, there will be edges across its width to accommodate the curvature. An **edged** ribbon will not be smooth around its circumference, there will be edges along its length. A **segmented** ribbon will combine the features of a **faceted** and **edged** ribbon. It would seem more descriptively natural to interchange the the **faceted** and **segmented** keywords, but unfortunately (due to backwards compatibility issues) the keywords are as described.

Three types of cross-sections must be defined in a cross-section file: **helix**, **strand**, and **turn**. *Ribbonjr* also uses the cross-section type **arrow** for creating arrows at the end of strands if it is defined. When *ribbonjr* reads in the PDB file, it automatically assigns residue cross-section types based on secondary structure. The user may override these cross-section type assignments by supplying XSECTION USER records in the PDB file (see below).

The third form in the grammar specifies how cross-section type changes from *name1* to *name2* should take place. Normally, when the cross-section type changes, the ribbon abruptly changes from one type to the other. If **interpolate** is specified, however, the cross-section linearly changes from *name1* to *name2*. This is useful for presenting a smooth transition between different cross-section types, such as from **strand** to **turn**. Note that *name1* and *name2* cross-section types must have the same number of coordinates in their specifications. If **point** is specified, the cross-section is linearly interpolated from type *name1* to type **turn**, and then abruptly changes to type *name2*. This transition method is usually used with *name1* being **arrow** and *name2* being **strand** or **helix**.

PDB USER RECORDS

In addition to the USER records that MIDAS uses, *ribbonjr* also interprets several of its own: FLIP, XSECTION and NORIBBON. The FLIP record specifies a residue whose orientation should not be computed in the standard way. *Ribbonjr* orients a residue by defining both a direction and a normal vector. The direction is defined by the vector from the guide atom of the previous residue to the guide atom of the next residue. The normal vector is defined as perpendicular to the direction and in the peptide plane defined by the previous residue and the current one; the normal vector is also constrained to be less than 90 degrees from the normal vector of the previous residue. If the current residue is FLIPped, however, its normal vector is replaced by its opposite vector. The XSECTION record specifies the cross-section type for a range of residues. For further information about cross section types, refer to the preceding description of the **–x** command line flag, and the CROSS-SECTION FILE section of this manual page. The NORIBBON record specifies a range of residues which should not have a ribbon representation (*i.e.*, they will be drawn as balls and sticks).

The alignment of the residue information in these USER records are exactly the same as those on a HELIX record. Examples of these USER records are below, with a standard HELIX record present as a reference record.

```
HELIX    1   A PHE      6  LEU     26  1
USER  FLIP     PHE      6
USER  XSECTION PHE      6  LEU     26  xsection-type
USER  NORIBBON PHE      6  LEU     26
```

EXAMPLES

The demonstration images included on the MidasPlus distribution CD show how to achieve a variety of striking effects and give detailed instructions on how each image was made. If these demonstration images have been installed on your system, they will be found in /usr/local/midas/demos/images. The `README.index` file there has further information. If the demonstration images have not been installed on your system, you need to mount the distribution CD-ROM, and you will find the images in the CD-ROM directory Midas-2.1/demos/images.

SEE ALSO

Carson, M. and Bugg, C.E., **Algorithm for ribbon models of proteins**, *Journal of Molecular Graphics*, Vol 4 (1986) pp 121-122.
conic(1), dnacheck(1), ksdssp(1), midas(1), showcase(1).

BUGS

Residues with ring structures on the mainchain may not be depicted correctly.

FILES

/usr/local/midas/resource/ribbonjr/xs.default – default cross-section file
/usr/local/midas/resource/midas/models/∗.ins – default residue drawing templates

AUTHORS

Conrad Huang
UCSF Computer Graphics Laboratory

NAME

run2ses – convert a PDBRUN file into a Midas session

SYNOPSIS

run2ses [**–i** *PDBRUN_file*] **–o** *session*

DESCRIPTION

Run2ses reads a PDBRUN file (a Protein Data Bank file with additional USER records, normally generated by the **pdbrun** command in MIDAS), and converts it into a MIDAS session. The **–i** argument specifies the input PDBRUN file. If no input file is given, *run2ses* will read from standard input. The **–o** argument specifies the output session name.

SEE ALSO

Midas User's Manual
Protein Data Bank, Atomic Coordinate and Bibliographic Entry Format Description.

FILES

session.*	Component files of the MIDAS session

BUGS

Silently ignores PDBRUN graphics types that MIDAS doesn't understand.

AUTHOR

Conrad Huang
Computer Graphics Laboratory
University of California, San Francisco

NAME

stereoimg – produce stereo pair of rendered molecule

MIDAS COMMAND SYNTAX

`Command:` **stereoimg –p** *renderer* [**–c**] [**–t**] [**–s**] [**–v**] [**–o** *outfile*] [**–C** *renderer_options*]

DESCRIPTION

Stereoimg takes the molecule(s) being displayed in MIDAS and creates a walleye stereo image pair using the specified *renderer*, which must be either *ribbonjr*, *conic* or *neon*. By default, the image is stored in a file called *stereoimg.tiff* and can be displayed with the program *imgview*.

The options flags have the following meanings:

–c Generate a crosseye stereo pair instead of walleye.

–t Normally *stereoimg* fits each half of the stereo pair into a half-screen by expanding one dimension of the original MIDAS window until it is the correct aspect ratio. If the **–t** option is given, the MIDAS window will be brought into the correct aspect ratio by trimming one dimension. This may result in part of the image being clipped, but will also result in less blank screen area. For optimal screen filling, reshape the MIDAS window to about the same aspect ratio as half the screen and get the image you want before running *stereoimg*.

–s Show the image immediately with *imgview*(1). Without this flag, the image is deposited in the output file and not displayed.

–o Deposit the image into the named output file, instead of the default *stereoimg.tiff*.

–p Use the named rendering program to generate the pair image. Legal values for the renderer flag are *conic*, *ribbonjr* and *neon*. This flag is mandatory.

–v Make the pair in a fashion appropriate for use with a mechanical stereo viewer, typically used with journal publications. Each eye image will have a square aspect ratio. If you convert the image to PostScript with the *itops*(1) program, giving a scale factor (with the **–s** option to *itops*) of 0.375 will produce a final image where each eye image is 2.5 inches square (a typical size used in journals). Other scale factors can be used for larger or smaller images.

–C This flag indicates the end of arguments for *stereoimg*. Any arguments which follow the **–C** flag will be passed on to the rendering program.

NeXT DIFFERENCES

On NEXTSTEP, *ribbonjr* does not work as a rendering program.

BUGS

Stereoimg assumes that the size of the screen is 1280 by 1024.

AUTHORS

Conrad Huang
Eric Pettersen
UCSF Computer Graphics Laboratory

NAME

stereops2illustrator – convert a stereo Midas copy file (PostScript) to Adobe Illustrator format

SYNOPSIS

stereops2illustrator < *stereo-copy-file* > *Illustrator-file*

DESCRIPTION

Stereops2illustrator reads in a PostScript file generated by the MIDAS **copy** command when displaying an image in stereo and converts it to an Adobe Illustrator format file for inclusion in desktop publishing documents.

SEE ALSO

ps2illustrator(1), midas(1)

BUGS

Depends on the *csplit* and *nawk* programs that might not be available on all platforms.

AUTHOR

Scott Dixon
SmithKline Beecham Pharmaceuticals

NAME

tiffpair – assemble a pair of TIFF images side by side

SYNOPSIS

tiffpair *outimage leftimage rightimage*

DESCRIPTION

Tiffpair reads a pair of TIFF image files and places them side by side. The height of the two images must be the same. *Tiffpair* is used by *stereoimg*(1) to compose stereo TIFFs from left- and right-eye images.

SEE ALSO

stereoimg(1), assemble(1SGI)

NAME

uncryst – generate atomic coordinates from crystallographic symmetry

SYNOPSIS

uncryst [–**s** *symmetry_description_file*] [–**l** *level*] [–**M**] [*input_PDB_file* [*output_PDB_file*]]

DESCRIPTION

Uncryst reads a Protein Data Bank file and generates coordinates for subunits based on the crystallographic symmetry information obtained from the CRYST and SCALE records. *Uncryst* assumes that the input PDB file contains atoms from a single subunit, and creates a PDB file which contains multiple copies of the input atoms, with each group transformed by a symmetry operator defined by the space group definition from the CRYST record.

The –**s** argument specifies a file that contains the symmetry descriptions of crystallographic space groups. The format of the file is given below. If the –**s** option is not specified, then a default file (see FILES section) will be consulted. The –**l** argument specifies the output level (see below for explanation). The default output level is zero. The –**M** argument specifies that each matrix used to transform the coordinates be printed. In addition, the determinants of the rows and columns of the rotational part of the matrix are also printed. These numbers should all be unity; deviation from unity results in skewed coordinates for the generated subunits.

If the output PDB file argument is omitted (or given as ‘‘–’’), *uncryst* will write to standard output. If the input PDB file argument is omitted (or given as ‘‘–’’), *uncryst* will read from standard input.

SYMMETRY DESCRIPTION

The symmetry description file contains a list of space group definitions. Each space group definition consists of a set of symmetry operators, divided into levels. Each symmetry operator describes how to transform atomic coordinates in fractional crystallographic coordinates. A symmetry operator is only applied if its level is less than or equal to the output level. The following is a single space group definition:

```
(1)     spacegroup "R 3"
(2)           level 0
(3)                           x,              y,         z
(4)                          -y,          x - y,         z
(5)                      y - x,              -x,         z
(6)           level 1
(7)                     x + 1/3,        y + 2/3,  z + 2/3
(8)                    -y + 1/3,  x - y + 2/3,  z + 2/3
(9)                 y - x + 1/3,       -x + 2/3,  z + 2/3
(10)                    x + 2/3,        y + 1/3,  z + 1/3
(11)                   -y + 2/3,  x - y + 1/3,  z + 1/3
(12)                y - x + 2/3,       -x + 1/3,  z + 1/3
```

Line 1 indicates that this is the definition for the space group named ‘‘R 3.’’ The name is used to compare against the space group symbol found in columns 56 through 66 of a PDB CRYST record. Line 2 specifies that the subsequent symmetry operators are assigned a level of zero. Lines 3, 4, and 5 are symmetry operators. Line 3 defines a symmetry operator that regenerates the input atomic coordinates (this operator is typically present). Lines 4 and 5 define two symmetry operators that generate new subunits. Line 6 specifies that subsequent symmetry operators are assigned a level of one. Lines 7 through 12 define six symmetry operators that generate new subunits.

Each symmetry operator is defined by three formulae which transform the x, y, and z coordinates respectively in the crystallographic fractional unit cell. Each formula is a simple arithmetic expression consisting of numbers, ‘‘x,’’ ‘‘y,’’ and ‘‘z.’’ Only the simple arithmetic operators are permitted. Grouping using parentheses is not supported. An additional restriction on the formulae is that they may only be linear combinations of ‘‘x,’’ ‘‘y,’’ and ‘‘z.’’ Formulae that violate this restriction are silently ignored. There can be no blank lines in a space group. Lines beginning with a ‘‘#’’ can be used to insert comments.

SEE ALSO

Protein Data Bank, Atomic Coordinate and Bibliographic Entry Format Description.
International Tables for X-Ray Crystallography, Vol. 1, *Symmetry Groups*.

FILES

/usr/local/midas/resource/uncryst.sdf	Default space group file

AUTHOR

Conrad Huang and Eric Pettersen
Computer Graphics Laboratory
University of California, San Francisco

NAME

unmtrix – expand MTRIX records in PDB files

SYNOPSIS

unmtrix [–**i** *original_PDB_file*] [–**o** *new_PDB_file*] [–**m** *MTRIX_serial_id*]

DESCRIPTION

Unmtrix reads a Protein Data Bank file and generates coordinates for subunits specified by MTRIX records. The –**i** argument specifies the input PDB file. If no input file is given, *unmtrix* will read from standard input. The –**o** argument specified the output PDB file. If no output file is given, *unmtrix* will write to standard output. Normally, *unmtrix* will generate coordinates for all MTRIX records which do not have atoms associated with them. When given the –**m** argument, *unmtrix* will generate coordinates only for the MTRIX records that match the given *MTRIX_serial_id*.

EXAMPLES

Since MTRIX records contain a 3x3 rotation matrix as well as a 3x1 translation vector, the records are always given in groups of three, like this:

```
MTRIX1   1 -0.725530  0.071920 -0.684420       68.86822    1            1RSL 138
MTRIX2   1 -0.000100 -0.994540 -0.104400       81.59939    1            1RSL 139
MTRIX3   1 -0.688190 -0.075680  0.721570       32.52771    1            1RSL 140
```

The second column is a serial number for the three-record group. If additional MTRIX records were necessary for additional subunits, each triplet would have a unique serial number. The third through fifth columns contain the rotation matrix. The sixth column is the translation vector. The seventh column indicates whether the coordinates generatable with these records are already present in the PDB file. If there is a ‘‘1’’ in the seventh column, the coordinates are present in the entry. If the column is blank, the coordinates are absent. The trailing columns are record serial number information, found on all PDB records.

The above MTRIX records would be for a subunit whose coordinates are already present in the PDB entry, indicated by the ‘‘1’’ in the seventh column. *Unmtrix* will always ignore such records. *Unmtrix* will only operate on MTRIX records with a blank seventh column, an example set of which is shown below.

```
MTRIX1   1   -.91000   -.41511    .00700        -.29750                1REI 128
MTRIX2   1   -.41396    .91000    .02598        -.29878                1REI 129
MTRIX3   1   -.01700    .02136  -1.00000       33.14600                1REI 130
```

SEE ALSO

Protein Data Bank, Atomic Coordinate and Bibliographic Entry Format Description.

AUTHOR

Conrad Huang
Computer Graphics Laboratory
University of California, San Francisco

NAME

videodisk — control a V-LAN videodisk

SYNOPSIS

videodisk [**–u** *unit*] [**–v** *vcopts*] *command* [*args*]

DESCRIPTION

Videodisk is a C-shell script that controls a V-LAN-interfaced videodisk. As distributed, it is dependent on the VideoCreator hardware and software from SGI, although the V-LAN commands used should work with any videodisk recorder. The **–u** option sets the V-LAN device unit number (defaults to unit 1). The **–v** option sets command-line flags for the various VideoCreator programs used. The VideoCreator command-line flags can also be set with the VCOPTS environment variable. Frames arguments to commands are given in the SMTPE timecode format HH:MM:SS:FF (see **expr-smpte**(1) for details on SMPTE time codes). The commands are:

recordsetup
: Set the videodisk up for recording. Turn off the screen saver. Initialize log file (see below).

recordscreen
: Place a SGI screen image into the VideoCreator framebuffer and record it on the videodisk.

recordimage *image-file*
: Place a SGI image file into the VideoCreator framebuffer and record it on the videodisk.

recordframe [*duration*]
: Record the contents of the VideoCreator framebuffer. A *duration* of more than one frame may be given as a SMPTE time code.

stoprecord
: Leave record mode. Turn the screen saver back on.

play *start-frame* [*stop-frame*]
: Play back the contents of the videodisk starting at the given *start-frame* and ending at the optional *stop-frame*.

stopplay
: Stop the playback.

location
: Report the current frame location on the videodisk in SMPTE timecode format.

loop *start-frame stop-frame*
: Continuously play back the frames between the given *start-frame* and *stop-frame.*

stoploop
: Stop the looping.

RECORD LOGGING

The file /tmp/vlan*unit*.log, where *unit* is the unit number argument to **videodisk**, contains one line for each completed **record***xxx* command. Each line is composed of the date, the location on the videodisk in SMPTE format, and the record command that was completed (setup, screen, image, frame). The log file is overwritten by the next **recordsetup**.

ENVIRONMENT

VCOPTS — command-line flags for VideoCreator programs

SEE ALSO

expr-smpte(1), vcvlancmd(1), vcpaste(1), vcsetmode(1)

BUGS

Depends on the VideoCreator. Last time we checked, when **fieldmerge**(1) was used to build frames with different images in each field, the field order was *field1 field0* instead of the order listed on the fieldmerge manual page.

AUTHOR

Greg Couch

NAME

viewdock – Midas delegate for browsing DOCK output

USAGE

viewdock [**–m**] *PDB_file*

DESCRIPTION

One of the problems of dealing with DOCK output is that there is a large number of compounds to examine, and MidasPlus does not offer a simple interface for browsing through these compounds. Some compounds may be discarded immediately upon visual examination, since they obviously do not fit well; others may need to be examined further, depending on their scores.

Viewdock is a MidasPlus delegate that reads a list of compounds from a Protein Data Bank (PDB) format (format described below) file named on the command line, starts MIDAS to display the compounds, and presents a panel for examining, manipulating and annotating compound information. If a file named *PDB_file*.info is present, *viewdock* assumes that it is from a previous session and will use the additional compound information from that file. If *viewdock* is invoked with the **–m** flag, MIDAS will not be invoked; this option may be used to scan through the PDB file without having to look at all the atomic coordinates.

The PDB file that *viewdock* reads should be an edited version of the ‘‘extended’’ PDB file produced by DOCK. The file contains a list of compounds, each of which consists of a set of REMARK records, followed by the atomic coordinates in ATOM records, and terminated with a TER record. The first REMARK records must contain the compound name, *e.g.*,

```
REMARK RHOIFOLIN
```

The name of the compound, `RHOIFOLIN` in this example, must be present for *viewdock* to work properly; subsequent REMARK records are optional and have no format restrictions. The ATOM records for the compound follow the REMARK records. The output file produced by DOCK almost follows the PDB format; however, the last few fields in ATOM records are placed such that they cross PDB field boundaries. MIDAS follows the PDB format strictly and will reject the erroneous records, so the ATOM records need to be edited to remove the offending fields. Finally, the TER record ends a compound entry.

For each compound, *viewdock* stores its name, its residue sequence number, the optional information from REMARK records, and its *discard state*: one of **deleted** (entries which were discarded in a prior sessions), **marked** (entries which were discarded in the current session), or **undeleted** (undiscarded entries). All this information is presented in an interface panel which consists of several parts: a compound browser, a MIDAS residue-number field, an information window, a **Discard/Restore** button, and a menu bar.

The name of a compound, its MIDAS residue number and its discard state are shown in the compound browser; names of **marked** compounds are noted by surrounding them with short dashes (‘‘- *compound name (sequence)* -’’) whereas names of **deleted** compounds are surrounded with long dashes (‘‘— *compound name (sequence)* —’’). At most one compound may be selected, using either the mouse or up and down arrow keys. The MIDAS residue sequence number and the optional information of the selected compound is displayed below the browser in the sequence field and information window respectively.

Viewdock operates in one of three modes: **examine**, **prune**, and **identify**. In **examine** mode, when the user selects an entry in the browser, the compound is automatically displayed in MIDAS and any previously displayed compounds are undisplayed; picking in MIDAS has no effect. In **prune** mode, when the user picks a compound in MIDAS, the compound is discarded; selecting an entry in the browser does not affect what is displayed in MIDAS. In **identify** mode, when the user picks a compound in MIDAS, the compound is selected in the browser; selecting an entry in the browser does not affect what is displayed in MIDAS.

When the selected entry in the browser is **undeleted**, the button below the information window is labeled as **Discard**; clicking on the button will make the selected entry **marked**. When the selected entry is **marked** or **deleted** the button is labeled as **Restore**; clicking on the button will make the selected entry **undeleted**.

The menu bar items include:

Exit
: Terminate *viewdock*. *Viewdock* can save compound state in a file called *PDB_file*.info, which may be used by future *viewdock* sessions. The **Exit** menu allows the user to choose whether compound state is saved and whether to terminate the associated MIDAS session.

List
: Select which compounds are displayed in the compound browser based on their discard states.

Mode
: Select the operating mode.

Midas
: Select what is displayed in MIDAS. **Show Undeleted** will show all undeleted compounds, while **Show Displayed** will show all compounds that appear in the browser.

Rewrite
: Create a PDB file that only contains undeleted compounds. The new file is created by removing lines from the input PDB file (*i.e.*, not DOCK's ‘‘extended’’ PDB file).

MIDAS INTERACTIONS

Viewdock uses the following command to invoke MIDAS:

```
/usr/local/midas/bin/midas -d dock
```

This default behavior may be overridden by setting the environment variable VIEWDOCK_MIDAS to the appropriate MIDAS invocation command.

Viewdock may also be invoked as a delegate from a running MIDAS session. When *viewdock* detects that neither its standard input nor its standard output are files, it assumes that it was invoked through the MidasPlus delegate mechanism and acts accordingly. So the command:

```
delegate start viewdock /usr/local/midas/bin/viewdock PDB_file
```

will start an instance of *viewdock* for *PDB_file*. Multiple instances of *viewdock* may run simultaneously.

IMPLEMENTATION NOTES

Viewdock is implemented as a Python script, so the Python interpreter (with Tkinter, tk and tcl included) and libraries must be installed. Point your favorite web browser at http://www.python.org for information about obtaining the latest release of Python.

SEE ALSO

midas(1), python(1)

AUTHOR

Conrad Huang, Computer Graphics Laboratory, UCSF