QUANTA 2006

Basic Operations

MAY 2006

Copyright

(1) Copyright

(2) Trademark

(3) Restrictions on Government Use

(4) Acknowledgments, and References

To print photographs or files of computational results (figures and/or data) obtained using Accelrys software, acknowledge the source in an appropriate format. For example:

> “Computational results obtained using software programs from Accelrys Software Inc. Dynamics calculations performed with the Discover program using the CFF forcefield, ab initio calculations performed with the DMol^3 program, and graphical displays generated with the Cerius^2 molecular modeling system.”

To reference an Accelrys publication in another publication, Accelrys Software, Inc., is the author and the publisher. For example:

> Accelrys Software Inc., Cerius^2 Modeling Environment, Release 4.8, San Diego: Accelrys Software Inc., 2006.

(5) Request for Permission to Reprint and Acknowledgment

Accelrys may grant permission to republish or reprint its copyrighted materials. Requests should be submitted to Accelrys Scientific Support, either through electronic mail to support@accelrys.com, or in writing to:

> Accelrys Scientific Support
> 10188 Telesis Court, Suite 100
> San Diego, CA 92121

Please include an acknowledgement “Reprinted with permission from Accelrys Software Inc., Document name, Month Year, Accelrys Software Inc., San Diego.” For example:

> Reprinted with permission from Accelrys Software Inc., Cerius^2 User Guide, June 2006, Accelrys Software Inc.: San Diego.

Contents

Preface

Who should read this book

QUANTA Basic Operations is designed primarily for first-time users, to introduce the basic design, layout, and operating principles of QUANTA® and to explain procedures for executing basic setup tasks.

What this book contains

This book contains feature and function descriptions that give an overview of QUANTA and its capabilities. Tutorial exercises are combined with descriptions to provide a hands-on introduction to the software. Since the book is designed primarily for new users, neither the descriptions nor the tutorial exercises comprehensively document all aspects of QUANTA.

The book is written assuming that you are familiar with:

- Basic UNIX® commands and file and directory structures.
- Windowing software on your QUANTA workstation.
- The difference between a shell window and a console window on your QUANTA workstation.
- How your workstation mouse works.

Assumptions also are made that you have a home directory where you can create subdirectories and a licensed copy of QUANTA installed on your workstation.

How to use the QUANTA introductory documentation set

This book is part of a documentation set that introduces users to QUANTA. The set consists of:

- *Basic Operations.*

- *Generating and Displaying Molecules.*
- *Simulation, Search, and Analysis.*

Although each book is self-contained, you should be familiar with basic operations to successfully complete building and editing tasks; and you should be familiar with molecule building and editing before proceeding with simulation and analysis.

Within each book, exercises are step-by-step procedures that may be linked across chapters. Datafiles may be modified or reused by several exercises.

Because of the sequential nature of the documentation set and the exercises in each book, you will obtain the most coherent and complete view of the software if you go through the books in the intended order.

Documentation set roadmap

The table below lists key topics and where to find information about them in the introductory set of QUANTA books. Book names are abbreviated:

- Ops = *Basic Operations.*
- Gen = *Generating and Displaying Molecules.*
- SSA = *Simulation, Search, and Analysis.*

Since aspects of a topic may be covered in a number of places, use the indices as an important resource for locating all the information on a topic. If you are reading this as online documentation at Accelrys's website, you can use the search functionality to find information.

For information about	Look at
Understanding program layout and design	Ops Chapter 1
Using operating principles and procedures	Ops Chapter 2
Manipulating molecules and displays	Ops Chapter 2; Gen Chapters 6 and 7
Building structures	Gen Chapters 1, 2, 4, and 5
Editing structures	Gen Chapters 1, 2 and 8; SSA Chapter 4
Applying constraints for CHARMm calculations	SSA Chapter 2

For information about	Look at
Importing and exporting files	Gen Chapter 8
Executing calculations using CHARMm or external programs	Gen Chapters 2, 3, 4, 6, and 9; SSA Chapter 1
Handling data — visual displays, tables, graphs	Ops Chapter 2; Gen Chapters 6 and 7; SSA Chapters 8 and 9
Performing experimental simulations	Gen Chapter 3, SSA Chapters 1, 3, 4, 5, and 7
Analyzing data	SSA Chapter 6

Related documentation

Other books in the QUANTA documentation set include:

- *QUANTA Installation Guide.*
- *CHARMm Principles.*

Other QUANTA-related documentation is included with specific software application packages. This documentation includes:

- *Protein User's Reference.*
- *Protein Homology Modeling Tutorial.*
- *X-Ray Structure Analysis User's Guide.*

Conventions

Typographical conventions used in this book include:

This	Looks like this
File examples, command-line examples, information in the text- port or on the message line.	Fixed-width
Menu and tool names.	**Bold**
Place-holder words that you need to replace with some other word.	*Italic*

1. Using QUANTA

This chapter introduces QUANTA and its features. It describes the basic design, layout, and flow of the program and procedures for getting started. Read this chapter to learn or to review how to get into QUANTA and how to use its basic tools.

Abbreviated information on how to start and run QUANTA also is included in the front matter of the two other books in this set, *Generating and Displaying Molecules* and *Simulation, Search, and Analysis.*

Online help

The QUANTA manuals, CHARMm documents, and various tutorials are accessible from the menu function **Information/Help**. In addition, interactive help for some palette tools and menu bar items can be turned on with the **SET HELP** command or by clicking the palette or menu with the right mouse button.

By default, online help starts a browser. It is therefore necessary for the command “netscape” to exist in your path. Use the UNIX command

```
>  which netscape
```

to see if this is true. Alternatively, help text can be displayed in the textport. This can be specified using the **SET HELP** command, but it happens automatically if Netscape cannot be started. Only text is printed to the textport — HTML formatting, graphics, table layout, etc. are of course lost.

QUANTA design, layout, and basic functions

QUANTA uses standard windows and menus in its interface. Most QUANTA processes are initiated by selecting choices from the menu bar. This bar is located at the top of the Molecule window that is displayed at program startup.

In general, as you move from left to right on the menu bar, you move from housekeeping functions through structure creation and editing functions, to simulation and analysis functions. Optional application choices such as **Protein Design** are located at the lower end of the **Applications** menu.

QUANTA runs under the UNIX operating system. There are several methods to access the operating system and perform system functions without exiting from QUANTA.

QUANTA functions can be organized into four broad categories: structure import and creation, structure editing, simulation, and analysis. Each category involves inputting data, manipulating or operating on the data, and outputting new or revised data.

Structures from external databases or structures created with other applications are imported into QUANTA using the **Import** selection in the **File** menu. Structures are created in QUANTA using one of the **Builder** choices in the **Applications** menu: **2D Sketcher** (ChemNote®)**, 3D Builder** (Molecular Editor)**, Sequence Builder,** or **Nucleic Acid Builder.** Molecules also can be constructed using one of the optional builders that you may have purchased with QUANTA.

Editing involves making changes to a structure. QUANTA provides editing tools that can modify structures to fit your criteria. Capabilities of these tools include:

- Adding or deleting atoms, bonds, residues, and fragments.
- Splitting or joining structures.
- Assigning different atoms to a structure, such as replacing a carbon atom with some other atom.
- Changing bond types.

Editing mainly occurs in the Molecular Editor, which is selected from the top of the **Edit** menu. Editing also occurs as torsion angles are determined as part of a conformational search.

QUANTA also offers many display options that allow you to choose how to graphically represent a molecule. Most display options are available through the **Draw** menu. These options are not editing tools. They do not determine what portions of a structure are active during QUANTA operations. They do determine how you see a structure or portion of a structure.

The display tools within the program allow you to:

- Rotate, translate, and scale structures.
- Selectively display different portions of a structure.
- Select colors for specific elements.
- Overlay and align multiple structures.
- Create graphical objects, such as ribbon drawings.

Simulations include energy minimization calculations, dynamics and dynamics animation, hydrogen bond calculations, conformational search, solid docking, and molecular similarity processes. CHARMm®, a computational chemistry program developed at Harvard University has been incorporated into QUANTA as the main calculation engine. Thus, many QUANTA processes are launched from the **CHARMm** menu. Other simulation choices are located in the **Calculate** and **Applications** menus.

The analysis function resides in the **Applications** menu and is applied to the results of simulation processes, such as dynamic animation and conformational search.

QUANTA menus also contain housekeeping functions, including setup and display choices. The **Preferences** menu includes functions for tailoring general operations of the program, such as color definitions, output settings, and MSF saving options. Selections in the **View** menu determine general viewing parameters.

Understanding files

All files and directory structures are organized by user-assigned names. Regardless of how a structure is created, information about it is stored in a molecular structure file or MSF. Filenames are written *filename*.msf, where *filename* stands for whatever name you give any file. An MSF contains at least this basic information:

- All atoms in the structure.
- Atom locations in 3D coordinate space.
- Atom types, names, and charges.
- Residue and segment information.

Other information, such as solvent accessibility, thermal mobility, and electrostatic potential may be added to an MSF.

Naming of files converted to MSFs is controlled in the Molecular Modeling application by the MSF Automatic File Name Generation function. Options for saving files are available in the **Preferences** menu under the **MSF Settings** selection.

You will encounter other file types as you work with QUANTA. Ones that are frequently created or used include:

- .cst file — Contains all environment settings used in a QUANTA run. On initial startup or on restart, QUANTA looks for a default.cst first. Initially, this file contains only default parameters. Changes are made

by selecting various preferences and options during a session. When you have established a preference or option, it is maintained in subsequent QUANTA sessions until you change it. This eliminates the need to reestablish your preferences every time you start a session, as long as you do not remove the .cst file from your subdirectory between sessions.

The .cst file also encapsulates other files that are used during a QUANTA run. For example, a file such as the color selection file .csf is copied into the .cst file at the end of each session, and the separate file is deleted. When you start a new session, the .csf file is re-created. This reduces the number of saved file and makes it easier to transfer work to another directory.

- .mbk file — The QUANTA brick map files (.mbk) can be used to store 3D information on a grid. These files can then be used to create contoured wire-frame representations of the information, graphical objects in QUANTA, or a map within the X-Ray structure package.

  The information on this 3D grid is stored in bricks. The overall grid is divided into 6 x 6 x 6 pieces, and the data for each brick is stored in a single direct-access record. This approach increases the speed and flexibility of selecting and retrieving portions of a complete map for contouring. In practice, bricks overlap one another on one edge to ensure that a continuous surface is generated in contouring.

  The header of the brick map file contains all the information about the contents of the file. QUANTA reads brick map files generated by earlier versions of the program.

- .mol file — Assigned to structures created in the QUANTA 2D Sketcher and saved as a 3D structure. This type of file is converted to an MSF when you exit the 2D Sketcher and return to the Molecular Modeling application or when a .mol file is imported into the Molecular Modeling application. The sketcher also creates a .pct file that saves 2D graphical information for use with the 2D application.

- .rtf file — Stores information for generating a representation of a molecule. The residue topology file (RTF) is used by CHARMm to generate a principle structure file (see below). An RTF describes the molecular topology of each residue in a structure's sequence. It contains information necessary to compute the energy of the molecule, as well as for performing other calculations based on the empirical energy function in CHARMm. Atom names, types, masses, partial atomic charges, definitions of bonds, angles, torsions, and improper torsion angles (out-of-plane angles) are included in an RTF. Coordinates are not included, nor is sequence data. An RTF is automatically generated when you construct or modify a molecule in two or three dimensions.

- .psf file — The central molecular model data structure used to identify all atoms and atom combinations for CHARMm calculations. The principle structure file (PSF) defines the structure in terms of contributions to the CHARMm energy function. A PSF identifies the number and type of all atoms along with their atomic masses and charges. It lists all intramolecular combinations that contribute to the CHARMm energy function, with information on bonds, bond angles, torsion angles, and improper torsions. A PSF is constructed directly from the molecular structure or from the data contained in an RTF and the CHARMm parameter file. The PSF assembles specified residues into a structure.

Setting the QUANTA environment

You must have a licensed copy of QUANTA installed on your system before you can start the program. Several environment variables must also be defined. If you have difficulties in either of these areas, please check with your system administrator.

The first step in getting started with QUANTA is to create several subdirectories, each able to run QUANTA, within your UNIX directory. This permits the datafiles generated by QUANTA to be organized systematically. Create a new subdirectory each time you begin a new QUANTA project.

Each subdirectory should contain a separate QUANTA environment that consists of:

- Structures.
- QUANTA user preferences.
- Generated data.

Setting up to run the tutorials

To establish a working area for these tutorial exercises:

1. **Create a subdirectory.**

Move the mouse so the cursor is located in a shell window. Enter the text:

```
> mkdir ~/accelrys_quanta
```

QUANTA is started from the accelrys_quanta directory for all tutorial exercises in the QUANTA documentation set. QUANTA should not be run from a console window. The console window is reserved for system messages.

2. Move into your accelrys_quanta subdirectory.

With the mouse cursor in the shell window, enter:

```
> cd ~/accelrys_quanta
```

3. Copy a sample structure file to the accelrys_quanta subdirectory.

With the mouse cursor in the shell window, enter:

```
> cp $QNT_ROOT/userguide/PEPTIDE.msf .
```

```
> chmod u+w PEPTIDE.msf
```

If you get the message:

```
QNT ROOT: Undefined variable
```

check with your system administrator.

When this step is complete, the directory is ready for starting QUANTA.

Starting QUANTA

When QUANTA is started, the program automatically opens in Molecular Modeling mode.

To start QUANTA for the first time:

1. **Begin the program.**

With the mouse cursor in the shell window, enter:

```
> quanta
```

You see a dialog box stating that the accelrys_quanta directory does not contain the necessary startup files. You see this dialog box whenever you run QUANTA for the first time in a new directory or if you have deleted your .cst file. The startup files contain setup information.

2. **Create the necessary startup files.**

Move the mouse so that the cursor is located over the rectangle in the dialog box containing the word **Yes** (in other words, the **Yes** button). Click the left mouse button. The dialog box is removed from the screen, and the necessary startup files are created. Selecting **No** in this dialog box exits QUANTA without creating files.

QUANTA enters Molecular Modeling mode, and the QUANTA Molecule window, textport, Molecule Management table, a dial emulator, and several palettes are displayed. If the QUANTA windows are not displayed, ask your system administrator for help in defining the environment or creating a license.

When you start QUANTA from a directory that was used earlier for QUANTA, the program automatically loads the last structure used in the most recent session in that directory.

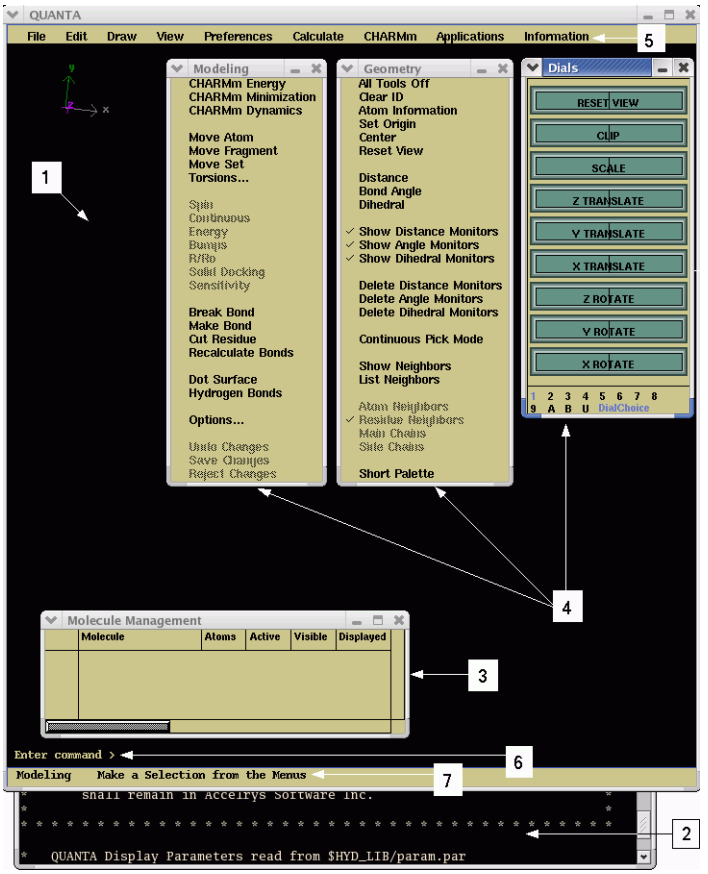
Understanding QUANTA windows

There are four types of windows in the basic QUANTA display:

1. Molecule window.

2. Textport window.
3. Molecule Management Table window.
4. Palettes and Emulator windows.

The windows are labeled numerically below.



These windows provide access to the functions and tools required to manipulate structures or provide status messages about the execution of various procedures and calculations. Some QUANTA applications use different windows to accommodate specific application-related tasks.

Tools for manipulating a QUANTA window can be accessed by placing the cursor on the button in the top-left corner of the window title bar and pressing the right mouse button to display a pulldown menu. Generally, you can

size, move, or hide a window by selecting an item from this menu. Palette windows cannot be resized.

> **Note:** Though each window can be moved, pushed, or popped in the same manner as all standard workstation windows, windows may not be closed. The X button in the upper right corner is disabled and selecting **Close** from the window tools merely hides the window and is equivalent to pushing it into the background.

When you iconify the Molecule Window in QUANTA, all displayed windows except the textport are closed. A single QUANTA icon is displayed. All the windows are redisplayed when the Molecule Window is redisplayed. Any other window can be iconified by itself.

Molecule window

The Molecule window is a resizable window that contains the following areas, in addition to the visualization pane:

5. Menu bar
6. Command line
7. Message line

Each area is labeled numerically above.

Viewing area

A molecular structure is displayed in the viewing area. After initial startup, QUANTA displays the last structure used in the previous session. In the upper-left corner of the viewing area, the x, y, and z axes show the orientation of a displayed structure in 3D coordinate space. These axes move with the motion of a displayed structure.

Menu bar

The menu bar contains nine menus from which various QUANTA functions and external programs can be accessed. Table 1 lists the menu titles and describes the functions contained in each. See Appendix A for a complete list of menu choices and function descriptions.

Command line

You can enter specific QUANTA commands on the command line located directly under the viewing area. By using the command line, you can bypass menus and palettes and send instructions directly to QUANTA or

Table 1. QUANTA menu bar

Menu	Function
File	Manipulates files creating QUANTA input and output. Terminates QUANTA and returns to operating system. Starts a new QUANTA session.
Edit	Initiates tasks used to change the contents of MSFs.
Draw	Initiates tasks used to change the display of MSFs. Creates visual guides to study MSFs.
View	Controls display of the viewing area. Saves and recalls orientations of MSFs.
Preferences	Establishes operational and window characteristics.
Calculate	Performs computational chemistry calculations within QUANTA. Provides access to third-party software programs that perform quantum mechanical or computational chemistry calculations.
CHARMm	Establishes specifications for CHARMm calculations. Starts CHARMm and performs various CHARMm calculations. Terminates CHARMm.
Applications	Initiates various QUANTA software modules used to create and analyze MSFs.
Information	Provides current program version, preference, and structure information.

the operating system. Commands cannot be entered when a dialog box is open. Refer to the *QUANTA Command Dictionary* for more information on command-driven QUANTA.

Message line

The message line is located directly under the command line. During QUANTA operations, this area displays instructions and error messages.

Textport

The textport is a scrollable, resizable window that displays QUANTA operational messages and information about displayed structures. For example, a message may list the atoms contained in the displayed structure, confirm picking of an atom in the viewing area, or transmit an error message.

The textport window can be manipulated using standard window manipulation procedures.

Molecule Management table

The Molecule Management table, which is displayed below the Molecule window, reports basic information about open MSFs. It also can be used to manipulate various aspects of a structure's display.

Table 2. Columns in the Molecule Management table

Column	Description
Molecule	Names of the MSFs loaded into QUANTA
Atoms	Number of active atoms in each MSF
Active	Whether each MSF is active, i.e., if computational operations can be performed on it
Visible	Whether the MSF is visible
Displayed	The number of displayed atoms per MSF
Residues	The number of residues per MSF
Segments	The number of segments per MSF
Width	The line width used to draw each MSF
Style	The line style used to draw each MSF
Close	Allows each MSF to be closed
Molecule	Repeats the first column

When you start QUANTA, the Molecule Management table lists the last structure and setting used in the previous QUANTA session.

The default size of the table includes the first four or five columns; to access the others, resize the window or scroll horizontally using the scroll-bar at the bottom of the table window.

As each MSF is opened, a row is created for it at the bottom of the table. MSFs are listed in the order they are opened. Rows are removed from the table when MSFs are closed. The default table size enables the first three or four rows to be viewed; resize the window or scroll down using the bar at the right of the table window to access any others.

The table can be shown or hidden with the Molecule Management table toggle in the Draw/Manage Display dialog box. It can also be manipulated using standard window manipulation procedures.

The Molecule Management table is operated by single or double clicks in the cells and column headers, providing various textual and graphical effects.

Palettes

Palettes appear automatically, usually to the right of the viewing area, when an appropriate choice from a QUANTA menu is selected. Each palette is contained in a window. Generally, a palette contains a collection of related tools in a menu format. Selecting a palette item can open additional palettes

Table 3. Effects of picks in the Molecule Management table

Column	Cell pick	Column header pick
Molecule	The corresponding MSF flashes on screen, and information about this MSF is reported to the textport.	Information about each of the MSFs is listed to the textport.
Atoms	Information about each atom in the MSF is listed to the textport.	Information about atoms in all MSFs.
Active	The activity of the MSF is toggled.	The activity of all MSFs is turned on/off together.
Visible	The visibility of the MSF is toggled.	The visibility of all MSFs is turned on/off together.
Displayed	Lists the display mask selection for all MSFs.	Lists the display mask selection for all MSFs.
Residues	Lists information about each residue in the MSF to the textport.	Lists information abut residues in all MSFs.
Segments	Lists information about each segment in the MSF to the textport.	Lists information about segments in all MSFs.
Width	Increments the line width used to draw the molecule. This continues to a value specified by the command SET LINE MAX, then goes back to 1.	Increments the line width of the first molecule and sets all others the same.
Style	Toggles through the line styles used to draw the molecule. 0 = full line 1 = dashed line 2 = dotted line	Toggles the line style of the first molecule and sets all others the same.
Close	On the first pick, the cell is primed, saying **Yes** instead of **No**; the second pick causes the MSF to be closed.	Displays a dialog box asking for confirmation whether to close all MSFs.

or dialog boxes. Manipulation and behavior of palette windows is similar to that of other windows, except that palette windows cannot be resized.

Selection of tools in palettes can make other tools unavailable or make previously unavailable tools available. A selected tool is always checked and highlighted, indicating it is active. Some tools remain active unless they are manually turned off. Other tools are automatically turned off when a process is completed.

Tools that are dimmed (gray) on a palette are either not available or are inappropriate for the work you are doing. A tool cannot be selected when dimmed. This prevents you from making an inappropriate choice. A dimmed tool may be restored to available status when it is toggled by the selection of another palette tool.

Dial emulator

The dial emulator is contained in a window in the lower right corner of the QUANTA startup screen. It is equivalent to mechanically operated buttons or dials, but operated by clicking with the mouse. The dial emulator window can be manipulated using standard window management procedures.

The dial emulator contains a family of dial sets numbered 1–9, A, B, and U. A particular dial emulator set can be selected by clicking the left mouse button when the cursor is over the appropriate number or letter at the bottom of the palette.

There are two types of dial emulators, global and task-specific. The global dials in Dial Set 1 control rotation, translation, scaling, and clipping for all structures in open MSFs. You can check the Molecule Management table to determine what MSFs are open.

Task-specific dials (Dial Sets 2–9, A, B, and U) control transformations for specific sections of structures, atom colors and labels, menu colors, dynamics, comparison flashing, and settings for various functions. Some dial sets, such as Dial Set 2, Fragment and Set Transformation, are automatically opened when an appropriate palette selection is made.

Table 4 provides a complete list of dial emulators. Many of the dials have mouse and keyboard equivalents that are also listed

Table 4. Dial emulator sets

Dial sets	Keyboard and mouse button equivalents
Set 1 Global transformations	
Reset view	Type the command Reset
Scale	Shift Key - No Button
Clip	Shift Key - Left Button
Z Rotate	Right Button
Y Rotate	Middle Button *(X,Y Rotation)*
X Rotate	Middle Button *(X,Y Rotation)*
Z Translate	Shift Key - Right Button
Y Translate	Shift Key - Middle Button *(X,Y Translation)*
X Translate	Shift Key - Middle Button *(X,Y Translation)*
Set 2 Fragment and set transformation	
Z Rotate	Ctrl Key - Right Button
Y Rotate	Ctrl Key - Middle Button *(X,Y Rotation)*
X Rotate	Ctrl Key - Middle Button *(X,Y Rotation)*

Table 4. Dial emulator sets (Continued)

Dial sets	Keyboard and mouse button equivalents
Z Translate	Ctrl Key - Left Button
Y Translate	Ctrl Key - No Button *(X,Y Translation)*
X Translate	Ctrl Key - No Button *(X,Y Translation)*
Set 3 Torsions	
Torsion 1	Alt Key - No Button*
Torsion 2	Alt Key - Left Button*
Torsion 3	Alt Key - Middle Button
Torsion 4	Alt Key - Right Button
Torsion 5	Alt Key - Left and Middle Buttons
Torsion 6	Alt Key - Left and Right Buttons
Torsion 7	Alt Key - Middle and Right Buttons
Torsion 8	Alt Key - All Buttons
Set 4 Atom translations	
Z Translate	Shift Key - Left and Right Buttons
Y Translate	Shift Key - Left and Middle Buttons *(X,Y Translation)*
X Translate	Shift Key - Left and Middle Buttons *(X,Y Translation)*
Set 5 Atom colors 1–6, atom IDs and labels, H bond colors	
ID H, S, I	--
HB H, S, I	--
Color 6 H, S, I	--
Color 5 H, S, I	--
Color 4 H, S, I	--
Color 3 H, S, I	--
Color 2 H, S, I	--
Color 1 H, S, I	--
Set 6 Atom colors 7–14	
Color 14 H, S, I	--
Color 13 H, S, I	--
Color 12 H, S, I	--
Color 11 H, S, I	--
Color 10 H, S, I	--
Color 9 H, S, I	--
Color 8 H, S, I	--
Color 7 H, S, I	--

***Note**: Use <Ctrl>-<Alt>-<Shift> for these operations if the <Alt> key actions invoke the window manager features on Linux.

Table 4. Dial emulator sets (Continued)

Dial sets	Keyboard and mouse button equivalents
Set 7 Menu colors	
Menu 5 H, S, I	--
Menu 4 H, S, I	--
Menu 3 H, S, I	--
Menu 2 H, S, I	--
Menu 1 H, S, I	--
Set 8 Dynamics	
Dynamics	Shift Key - Middle and Right Buttons
Speed	Shift Key - All Buttons
Set 9 Not currently assigned	
Set A Comparison flashing	
Speed	Middle and Right Buttons
Set B Rocking and stereo settings	
Focal Length	--
Eye Separation	--
Stereo Angle	Ctrl Key - Left and Middle Buttons
Separation	Ctrl Key - Left and Right Buttons
Rock Angle	Ctrl Key - All Buttons
Rock Speed	Ctrl Key - Middle and Right Buttons
Set U User defined dials	
User Defined 1	Shift + Ctrl Keys - No Buttons
User Defined 2	Shift + Ctrl Keys - Left Button
User Defined 3	Shift + Ctrl Keys - Middle Button
User Defined 4	Shift + Ctrl Keys - Right Button
User Defined 5	Shift + Ctrl Keys - Left and Middle Buttons
User Defined 6	Shift + Ctrl Keys - Left and Right Buttons

Using the mouse

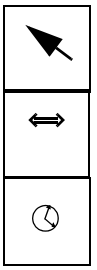
Most QUANTA operations can be performed with the mouse. The mouse is used to position the cursor at a location in the window, select items in a window, and move a displayed structure.

There are two methods of operating mouse buttons; both use the left button most of the time. The first method is a single click of the button, that is,

press and release. For example, making a selection in a dialog box requires a single click of the left mouse button.

The second method is to press and hold the button while moving (or *dragging*) the mouse to move the cursor. For example, selecting a menu function requires pressing then holding the left mouse button to display a menu, then moving the cursor down to highlight a selection. Releasing the mouse button completes the selection of the function.

The cursor changes appearance as it moves to different window locations, indicating the type of operation in progress or required.



viewing area

dial emulators

wait

Practice mouse techniques with this exercise:

1. Display different cursors.

Move the mouse so the cursor is over the viewing area, then move the cursor over the dial emulators. The appearance of the cursor changes in each area.

2. Display a hidden palette.

Move the mouse so the cursor is located over the title bar of the Geometry palette located behind the Modeling palette. Click the left mouse button. The Geometry palette pops in front of the Modeling palette.

Move the mouse so the cursor is located over the title bar of the Modeling palette. Click the left mouse button. The Modeling palette pops in front of the Geometry palette.

Understanding menus

The QUANTA interface contains a set of menus to simplify the use of QUANTA. Each menu contains a group of related functions. Selecting a function can cause a dialog box or a new palette to be displayed.

Menu types

QUANTA provides three types of menus: pulldown, pull-right, and sticky. Any menu can be used as either a pulldown or sticky menu, depending on how you display it. Pull-right menus are accessed from pulldown and sticky menus.

Some menus or menu choices are dimmed (gray) when the operations in which QUANTA is engaged make those choices inappropriate or if they access applications for which you do not have a valid license.

Complete the following three exercises to learn how to use the three menu types. The first exercise involves pulldown menus.

1. Display the File menu as a pulldown menu.

Move the mouse so the cursor is over the word **File** in the menu bar. Press and hold the left mouse button. The contents of the **File** menu are displayed as a pulldown menu.

With the mouse still over the word **File** in the menu bar, release the button. The **File** menu is closed.

2. Display other menus as pulldown menus.

Display the **File** menu again. Continue to press and hold the mouse button as you slowly move the cursor to the right over the other names in the menu bar.

The contents of each menu are displayed as the cursor is moved. As one menu opens, the previously displayed menu closes.

Release the mouse button to close the last menu.

3. Display menu function information in the message line.

Display the **File** menu again.

Continue to press and hold the mouse button and move the cursor down over the **Open** function, the first item in the **File** menu. When the cursor is on the selection, it is highlighted.

With the function name highlighted, look at the message line at the bottom of the viewing area.

The message:

```
Select MSFs to Open
```

is displayed, providing information or directions that apply to the highlighted selection. Messages are also displayed for selections from sticky and pull-right menus.

Move the cursor away from the menu so that no selections are highlighted and release the mouse button. The **File** menu is closed and no function is selected. Repeat this procedure as often as you like to get information on any menu item.

The following exercise involves pull-right menus.

1. Display a pull-right menu.

Go to the **View** menu and display it. Highlight the menu item **Window Layout.** This item has an arrow to its right. When the item is highlighted, its pull-right menu is automatically displayed.

2. Make a selection from a pull-right menu.

Still holding the mouse button down, move the cursor into the pull-right menu and select the item **Full Screen**. Release the button to complete the selection. The **Molecule** window expands to occupy the full screen.

Reduce the window to its previous size by repeating the selection process, selecting **Default Screen** from the **Window Layout** menu.

The following exercise involves sticky menus.

1. Enable sticky menus.

By default, the **Sticky Menu** preference is off. To change the preference, go to the **Preferences** menu and highlight the **Sticky Menu** selection. A pull-right menu is displayed. Select **On** in the pull-right menu and release the mouse button.

2. Display the View menu as a sticky menu.

Select the **View** menu in the menu bar by clicking once quickly with the left mouse button. A window containing the **View** menu opens and remains displayed.

3. Display a pull-right menu as a sticky menu.

From the **View** menu, select **Window Layout.** The **Window Layout** pull-right menu opens and remains displayed. **Window Layout** remains highlighted in the **View** menu.

4. Collapse a sticky menu.

In the **View** menu, select **Window Layout** again. The **Window Layout** menu is closed. Select **View** from the menu bar. The **View** menu is closed.

You can close the **View** menu, leaving only the **Window Layout** menu displayed. When the **View** menu is closed, iconify the **Window Layout** menu by selecting the icon in the upper-right corner of the menu title bar. The menu icon is displayed briefly, then removed from the screen.

Using dialog boxes

Some menu and palette selections require additional information before a requested procedure can be executed. The additional information typically is needed to define conditions for carrying out a function. QUANTA automatically asks for additional information in these cases by displaying a palette or dialog box.

When you enter the required information in a dialog box and confirm your response by selecting an action button, QUANTA completes the procedure. A **Cancel** button is provided in most dialog boxes so that you can return to the menu or palette that called the dialog box without completing the procedure.

Some functions use more than one level of dialog boxes. Successive boxes are displayed until all the information required for the initially selected function is specified or the function is cancelled.

Selecting information in a dialog box

Dialog boxes ask for information using a variety of tools including options, data entry fields, and scrolling lists. Combinations of tools may be contained in a single dialog box.

When a dialog box is displayed, one or more options may already be selected. These defaults are the choices that are used most often and are displayed with the selection filled or marked by a thicker line around the button.

Three different types of options can be present in a dialog box. Each option permits the selection of a condition or action, but each has different features and constraints.

- **Toggle Selection** — A square box followed by a short line of text describing the choice. Selecting a toggle option automatically adds that option to any previously selected toggle options, and all are applied simultaneously.
- **Choice Selection** — A circular radio button followed by a short line of text describing the choice. Only one radio button in a set can be selected at any time. Selecting one choice selection automatically turns off any that were previously selected in that set.
- **Action Button** — A word, such as **Save**, **OK**, or **Cancel**, within a rectangular box. Generally placed at the bottom of a dialog box, the button is an instruction for processing the information in the dialog box.

  Action buttons generally exist in pairs. Selecting one of the buttons results in action being taken. For example, information is saved and the dialog box is closed. Selecting the other button results in a different, often opposite, action being taken. For example, information is not saved and the dialog box is closed.

For each type of option, a default selection has been made in the illustrated dialog box.

Data-entry boxes

Data-entry boxes provide space for you to supply information. When a particular data-entry box is highlighted, information can be entered from the keyboard. Data can consist of text, numbers, or a combination of both. Several examples of data-entry fields are shown in the Modeling Options dialog box above.

Entered data must meet certain requirements to be read into QUANTA. For example:

- Numbers must be within certain limits.
- Character strings must not exceed certain lengths.
- Path or filenames must be correct.

Most text entries in QUANTA, with the exception of filenames, are case insensitive. Filenames are governed by standard UNIX conventions.

If the entered data violates these requirements when an action button is selected, QUANTA reports an error in the textport and does not close the dialog box until additions, deletions, or corrections are made in the data-entry box to correct the error. Errors made during input can be erased using the <Backspace> key when the data entry box is highlighted.

Scrolling list

A scrolling list consists of a set of items, typically filenames and directories. Most scrolling lists permit the selection of only one item at a time.

To scroll down an increment, click the left mouse button anywhere in the scroll area. To scroll up, click the right mouse button anywhere in the scroll area. To scroll up or down, click above or below the scroll bar with the middle mouse button.

Using an action button

The following short procedure demonstrates the use of an action button:

1. **Select the About QUANTA function from the Information menu.**

   Display the **Information** menu and select **About QUANTA**. A dialog box describing the QUANTA release is displayed.

2. **Select the Continue button.**

   Select the **Continue** button at the bottom of the dialog box. The button is briefly highlighted, indicating it is selected. The dialog box is closed.

Using the File Librarian

The File Librarian is used for selecting files in QUANTA. It lists files in your working directory or in other directories on the network. You can select a file by typing the filename into a data entry box or by clicking the filename in the list to insert the desired name into the data entry box.

A File Librarian dialog box contains several selection and message areas. It can be used to open or save a selected file. Although there is some variation in display, all File Librarian dialog boxes work in the same way.

At the top of a File Librarian dialog box are two lines of text, one indicating the purpose of the box and the second listing the current directory's path. Just below these lines is a highlighted data entry box that permits keyboard entry of the filename you want to select. Up to 1024 characters can be entered to identify path and file. Standard UNIX wildcard characters may be used in the data entry box to specify a file or set of files, and environment variables are interpreted.

Below the data entry box is the scrolling list of files and subdirectories. Files in subdirectories of the current directory can be accessed by selecting the appropriate subdirectory from the scrolling list. The list is automatically updated with the contents of the subdirectory you select.

To the right of the scrolling list is a set of radio buttons to define how files are ordered in the scrolling list (alphabetical by filename or in chronological order). The action buttons labeled **Dir Up** and **Home Dir** facilitate moving among directories to search for files.

The Open File Librarian dialog box has a unique display for selected files. When you select a file by highlighting it, it appears both in the data entry box and in the **Selected Files** display area. A filename typed into the data entry box is added to the **Selected Files** list by clicking the **Add** button to the right of the display area. If you want to remove an entry from the **Selected Files** list, highlight the filename, then select the **Remove** button to the right of the display area.

Append and **Replace** radio buttons are located near the bottom of the dialog. They permit the addition of selected files to those already loaded into QUANTA or the replacement of all currently loaded MSF files.

At the bottom of the File Librarian dialog box are action buttons. When one of these buttons is selected, the information defined in the dialog box is implemented. These buttons typically are labeled **Open** and **Cancel**.

Use the following exercise to become familiar with basic procedures for using a File Librarian. The example uses the Open MSFs dialog box for opening an MSF.

1. Display the File Librarian.

Display the **File** pulldown and move the cursor over the **Open** function. With the function name highlighted, release the mouse button.

The Open MSF dialog box appears. The current directory is identified as accelrys_quanta in your home directory. The scrolling list contains one file entry, PEPTIDE.msf. This file was created and the MSF was copied to that directory in a previous exercise. (See “Setting up to run the tutorials” on page 9.)

2. Change directories within the File Librarian.

Select the **Dir Up** button. The **Current Directory** message changes to identify the directory path of the home directory. The scrolling list changes to display files and subdirectories located in the home directory.

Move the mouse so the cursor is located over the accelrys_quanta directory in the scrolling list. If the accelrys_quanta directory is not visible in the scrolling list, move the list up and down with the scrollbar until the directory is displayed.

Click the left mouse button. The directory name is highlighted, indicating it is selected. The **Current Directory** message changes to accelrys_quanta under the home directory. The scrolling list changes to list the files located in the accelrys_quanta directory.

3. Select the entry PEPTIDE.msf from the scrolling list

Move the mouse so the cursor is located over PEPTIDE.msf. Click the left mouse button.

The PEPTIDE.msf name is highlighted in the scrolling list, indicating it is selected. The filename is also displayed in the data entry box and in the **Selected Files** scrolling list. Several files can be added to the **Selected Files** scrolling list one at a time, by clicking additional MSFs.

Alternatively, you may add PEPTIDE.msf to the **Selected Files** list by typing its name in the data entry box and selecting the **Add** button.

4. Confirm choices.

Select the **Open** action button at the bottom of the dialog box to confirm the file choices. The dialog box is closed and the PEPTIDE.msf structure is displayed in the viewing area.

Using the Molecule Management table

The Molecule Management Table reports filename, number of atoms, activity, visibility, number of displayed atoms, number of residues, number of segments, line width, and style for each MSF that is open in QUANTA.

The **Atoms** column lists the number of active atoms in the open MSF.

The **Active** and **Visible** columns contain **Yes**/**No** toggles. All operations are allowed for active structures. For inactive structures, only operations that provide information but do not affect the integrity of the molecule are allowed. Visible structures appear in the viewing area regardless of activity status. Invisible structures do not appear in the viewing area. Both visible and invisible structures can be active.

The **Displayed** column reports the number of atoms in an MSF that are displayed in the viewing area. This number may or may not agree with the number in the **Atoms** column, since atoms may be active but not displayed.

Selecting the button in the upper-right corner of the window enlarges the window to display the rest of the columns. You can also scroll through the table columns.

Selecting a cell in any of the columns displays information in the textport. For example, if you select a cell in the **Molecule** column of a specific MSF, a summary description of the MSF is printed in the textport. It includes the structure's name, the number of selected atoms, the visibility status, the activity status, and the number of displayed atoms.

If you select **Atoms**, a list is printed in the textport including information on atom number, segment ID, residue ID, residue name, atom type, and xyz coordinates. Similar information is supplied when you select any cell in the other columns.

The following exercise demonstrates how the Molecule Management Table can be used.

1. Select Molecule cell.

Select the **Molecule** cell for PEPTIDE.msf. A summary description of PEPTIDE.msf is printed in the textport.

2. Select Atoms cell.

Select the **Atoms** cell for PEPTIDE.msf. The textport lists all atoms in the MSF. If your molecule is large, not all the information may be displayed in the textport. If this is the case, the textport listing stops with a prompt:

```
>  Press <q> to quit or any other key to continue
```

With the cursor in the window, press the letter **q** to stop the listing.

3. Select Visibility cell.

Select the **Visible** cell entry for PEPTIDE.msf. The visibility status changes from **Yes** to **No**, and the structure goes from being displayed to not being displayed in the viewing area.

Select the cell again, changing it back to **Yes**. The structure is redisplayed.

4. Select Activity cell.

Select the **Active** cell for PEPTIDE.msf. The activity status changes from **Yes** to **No**, and the visual display of bonds changes for the MSF.

Select the **Active** cell again and return the MSF to its original state. The cell changes from **No** to **Yes**, and the bonds in the MSF visibly change, becoming brighter and thicker.

5. Select Displayed, Residues, and Segments cells.

Select the **Displayed**, **Residues**, and **Segments** cells for PEPTIDE.msf. As each cell is selected, additional information about the structure is displayed in the textport.

6. Select Width cell.

Select the **Width** cell to increment the bond line width by 1. On reaching 8, the line width cycles back to 1.

7. Select Style cell.

Select the **Style** cell to change the line style among the values 0 solid, 1 dashed, and 2 dotted.

8. Select Close.

Select **Close**. You are prompted for a confirmation to close the MSF (verify by selecting the same cell again). The MSF is closed.

9. Select a row label.

This molecule is made active and visible; all other molecules have their status changed to inactive and invisible.

Displaying structures

When an MSF is opened in the QUANTA Molecular Modeling application, a graphic representation of the structure's atoms and bonds is displayed in the viewing area. QUANTA has a variety of display options that enable you to focus attention on particular parts or characteristics of a structure.

Display defaults

The default representations of bonds and atoms in QUANTA are as follows:

- Single, double, and triple bonds are represented as single lines.
- Atoms are represented as the end points of bonds or as the point where two bonds meet.
- Each half of a bond is colored according to the type of atom located at the end of the bond.
- The default colors of atoms are set as in Table 5.

Table 5 Default atom colors

Atom type	Color
Carbon	Light green
Nitrogen	Blue
Oxygen	Red
Sulfur	Yellow
Hydrogen	White
Phosphorus	Light yellow
Fluorine	Dark green
Chlorine	Dark green
Iodine	Purple
Silicon	Yellow
Metals	Gray

Additional information about editing a structure and customizing its display is available in Manipulating Molecular Displays (page 63) and in *Generating and Displaying Molecules*, Chapters 1 and 2.

Picking an atom

Many QUANTA operations require that an atom or set of atoms be picked before processing. When an atom is picked, an identification tag (ID) is displayed on the screen adjacent to the picked atom. This ID contains atom and residue information. Additional information is displayed in the textport. You can choose the information that you want to display in an atom ID. The procedure for choosing this information is described in the next exercise.

The QUANTA default for displaying an atom ID is to display the ID of the last-picked atom only. This preference can be changed by opening the **Preferences** menu and selecting **ID Mode and Style,** then making a selection from the pull-right menu.

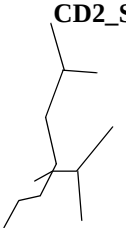
1. Display an atom ID.

Place the tip of the cursor over the end point of a bond. Click the left mouse button. The atom ID is displayed next to the atom, and information about the atom appears in the textport.

Move the mouse so the cursor is over a point where two bonds meet. Click the left mouse button. The ID for the first picked atom disappears, and an ID for your new pick is displayed. Information about this atom also appears in the textport.

Pick several other atoms in the structure. IDs are successively displayed as you move from one atom pick to another.

CD2_SEG1:19



2. Display multiple IDs

Go to the **Preferences** menu. Select **ID Mode and Style**. A pull-right menu appears.

Select **Multiple IDs**, then return to the viewing area and pick several atoms in succession. Each time you pick an atom, an ID is displayed. No IDs are removed as you continue to pick additional atoms.

Repick the first picked atom. The ID flashes briefly.

3. Delete selected IDs

Return to the **ID Mode and Style** pull-right menu. Select **Repick Deletes.** Click several IDs in the viewing area. They are removed.

4. Change the information displayed in IDs

Return to the **Preferences** menu and open the pull-right menu for **ID Mode and Style.**

Select **ID Style** from the menu.

A dialog box appears with the directive on the top line of the box:

Choose what to label with:

Select the toggle selection boxes you want, then select the **OK** action button.

Return to the viewing area and click several atoms. The IDs that are displayed contain information that reflects the choices you made in the dialog box.

5. Remove remaining atom IDs

Bring the Geometry palette in front of the Modeling palette. All the tools on the Geometry palette are now visible. From the Geometry palette, select **Clear ID.** All remaining IDs are removed from the viewing area.

Moving structures

QUANTA provides several mechanisms for moving structures or parts of structures in 3D space. Each mechanism can perform transformation operations, such as rotations or translations, as well as clipping and scaling of the displayed structure.

The available mechanisms for moving structures are:

- Dial emulators
- Mouse
- Keyboard commands

- Button box
- Graphics tablet
- Spaceball

The dial emulators and the mouse provide all necessary transformation operations. Keyboard commands can be used to provide precise manipulations.

The button box, graphics tablet, and spaceball are optional hardware devices for Silicon Graphics machines that provide additional ways of performing the same operations. For information about using these devices, see Appendix B. For information on installing these devices, see the *QUANTA Installation Guide* and your system administrator.

Using dial emulators

Dial emulators are grouped logically into sets. Each emulator set is associated with a number from 1 through 9 or the letter A, B, or U. See the complete list of emulator sets and dials and their functions in Table 4.

Dial emulators that control vector quantities like translation, rotation, and scaling are affected by the position of the cursor over the emulator dials. For example, if you click the left side of the X-rotation emulator dial using any mouse button, the structure in the viewing area rotates clockwise. Pressing and holding the button while dragging the mouse to the left results in continuous clockwise rotation that increases in speed as the cursor moves farther left.

Clicking the right side of the dial rotates the structure counterclockwise. Dragging the mouse to the right with the mouse button depressed results in continuous counterclockwise rotation that increases in speed as the cursor moves farther right.

Work through the following exercises to become familiar with dial emulator operation.

1. Rotate the displayed structure about the z axis.

Move the mouse so the cursor is over the left side of the **Z Rotate** dial. The cursor changes to a double arrow.

Press and hold the left mouse button. The structure starts rotating clockwise around the z axis. The axes displayed in the upper-left corner of the viewing area rotate with the structure.

Release the mouse button. The structure and the axes stop rotating.

2. Change direction and speed of rotation.

Move the mouse so the cursor is over the right side of the **Z Rotate** dial. Press and hold the left mouse button. The structure starts rotating counterclockwise around the z axis.

As you continue to press and hold the left mouse button, move the cursor back to the middle of the **Z Rotate** dial. The rotation slows, then stops.

As you continue to press and hold the left mouse button, move the cursor to the left of the **Z Rotate** dial. The rotation starts again slowly clockwise.

Move the cursor towards the left edge of the **Z Rotate** dial. The speed of the clockwise rotation increases.

Move the cursor completely out of the **Z Rotate** dial in a direct line to the left. The rotation continues at an even faster rate.

Release the mouse button. The structure and axes stop rotating.

3. Scale the structure

Move the cursor over the **Scale** dial. Press and hold the left mouse button as you move the cursor to the right of center. The molecule enlarges.

Move the cursor to the left of center and watch the structure become smaller.

4. Move the structure through a clip zone

Move the cursor over the **Clip** dial, a little left of center. Press and hold the mouse button as you move the cursor to the left until some of the atoms in your structure disappear. Release the mouse button.

You have narrowed the depth of the slice along the z axis within which parts of your structure are visible.

Move the cursor over the **Z Translate** dial. Using this dial, move the structure back and forth along the z axis, observing different slices of the structure in the clip zone.

5. Return the structure and axes to their original orientation

Select the **Reset View** dial. The structure and the axes return to their original orientation.

Using the mouse and keyboard

Transformations can also be performed using a combination of keyboard keys and mouse buttons. Mouse transformations move the structure at the same speed and in the same direction that the mouse is moved. Mouse and keyboard actions that give equivalent movements as a specific dial emulator are listed in Table 4.

In addition, commands such as **ROT X 90** (rotate 90 degrees around the x axis) can be used to provide precise manipulations of structures. Consult the *QUANTA Command Dictionary* for more information.

Do these exercises to practice using the mouse and keyboard to move your structure.

1. Rotate the displayed structure about the xy axes

Move the mouse so the cursor is in the viewing area. Press and hold the middle mouse button. Move the mouse so the cursor moves around the viewing area.

The structure rotates at the same speed and in the same direction as the mouse. The axes, displayed in the upper-left corner of the viewing area, rotate with the structure.

Release the mouse button. The structure and axes stop rotating.

2. Rotate the displayed structure about the z axis

Move the mouse so the cursor is in the viewing area. Press and hold the right mouse button. Move the mouse so the cursor moves to the left side of the viewing area.

The structure rotates clockwise at the same speed as the mouse. The axes, displayed in the upper-left corner of the viewing area, rotate with the structure.

While continuing to press and hold the right mouse button, move the mouse so the cursor moves to the right side of the viewing area. The structure rotates counter-clockwise at the same speed as the mouse.

Release the mouse button. The structure and axes stop rotating.

3. Translate the displayed structure on the xy axes

Move the mouse so the cursor is in the viewing area. Press the <Shift> key and hold the middle mouse button simultaneously. Move the mouse so the cursor moves around the viewing area.

The structure moves about the viewing area at the same speed and in the same direction as the mouse.

Release the <Shift> key and the mouse button. The structure stops moving.

4. Return the structure to its original orientation

Select the **Reset View** dial on the Dial Emulator. The structure returns to its original orientation. **Reset View** in the Geometry pallet and the **Reset View** dial perform the same function.

Using palettes

Palettes contain tools that perform many operations. Palettes appear on the screen when the tools they contain may be needed. For example, when QUANTA is started, the program is automatically placed in the Molecular Modeling application, and the Modeling and Geometry palettes are displayed. These palettes contain commonly used QUANTA tools. Table 6 and Table 7 list selections on these palettes and provide brief descriptions of each.

Table 6. Geometry palette

Selection	Description
All Tools Off	Turns off any Geometry selections that are active and deletes the associated monitors.
Clear ID	Removes the display of atom IDs as a result of picking atoms in the viewing area. Does not affect labels generated by the **Label Atoms** selection.
Atom Information	Prints details of subsequently picked atoms in the Textport, including atom name, atom number, residue number, segment number, molecule number, xyz coordinates, fourth parameter value, color, CHARMm atom type name and number, atomic charge, energy parameters, radius parameters, and set number and label.
Set Origin	Places the picked atom at the center of the viewing area. This atom becomes the center of rotation for subsequent operations.
Center	Calculates the geometric center of currently displayed atoms, and places this center at the center of the viewing area. This geometric center becomes the center of rotation for subsequent operations.
Reset View	Returns the structures to their original orientation.
Distance	Calculates the distance between subsequently picked pairs of atoms. Displays the results in the Textport and in the viewing area if **Show Distance Monitors** is also active. Picked atoms need not be bonded.
Bond Angle	Calculates the angle between successive triplets of picked atoms. Displays the results in the Textport and in the viewing area if **Show Angle Monitors** is also active. Picked atoms need not be bonded.

Table 6. Geometry palette (Continued)

Selection	Description
Dihedral	Calculates the dihedral angle between successive quadruplets of picked atoms. Displays the results in the Textport and in the viewing area if **Show Dihedral Monitors** is also active. Picked atoms need not be bonded.
Show Distance Monitors	Shows distance monitors as a result of picking pairs of atoms while **Distance** is selected. This distance monitor is updated during modeling operations. Distance monitors are removed from the screen when this tool is turned off, but are not deleted. Reselecting this tool will redisplay the monitors.
Show Angle Monitors	Shows angle monitors as a result of picking triplets of atoms while **Bond Angle** is selected. This angle monitor is updated during modeling operations. Angle monitors are removed from the screen when this tool is turned off, but are not deleted. Reselecting this tool will redisplay the monitors.
Show Dihedral Monitors	Shows dihedral angle monitors as a result of picking quadruplets of atoms while **Dihedral** is selected. This dihedral angle monitor is updated during modeling operations. Dihedral angle monitors are removed from the screen when this tool is turned off, but are not deleted. Reselecting this selection will redisplay the monitors.
Delete Distance Monitors	Deletes any existing distance monitors.
Delete Angle Monitors	Deletes any existing angle monitors.
Delete Dihedral Monitors	Deletes any existing dihedral monitors.
Continuous Pick Mode	Affects how picked atoms are used in calculations with **Distance, Bond Angle,** and **Dihedral**. This mode is useful for calculating all geometric values along a chain. If this tool is active, and atoms A, B, C, D, E, and F picked, bonds AB, BC, CD, DE, and EF; angles ABC, BCD, and DEF; and torsions ABCD, BCDE, and CDEF are calculated. If this tool is not active, the bonds AB, CD, and EF; angles ABC and DEF; and torsion ABCD are calculated.
Show Neighbors	Calculates and displays the nearest nonbonded neighbors to subsequently picked atoms. Making this selection activates **Atom, Residues, Main Chains,** and **Side Chains.**
List Neighbors	Calculates nearest non-bonded neighbors to subsequently picked atoms and lists them in the Textport. Making this selection activates **Atom, Residues, Main Chains,** and **Side Chains**.

Table 6. Geometry palette (Continued)

Selection	Description
Atom	Causes the **Show/List Neighbors** calculation to be done from the picked atom to all other atoms.
Residue	Causes the **Show/List Neighbors** calculation to be done from all atoms in the residue containing the picked atom to all atoms in other residues.
Main Chains	Causes the **Show/List Neighbors** calculation to be done from peptide main chain atoms (C, Cα, N, O) in the residue containing the picked atom to all other main chain atoms.
Side Chains	Causes the **Show/List Neighbors** calculation to be done from peptide side chain atoms (all but C, Cα, N, O) in the residue containing the picked atom to all other side chain atoms.
Short Palette	Allows palette to be shortened so that only the first six selections are displayed.

Table 7. Modeling palette

Selection	Description
CHARMm Energy	Runs CHARMm to calculate the total energy of a structure including all active atoms, whether or not they are currently displayed.
CHARMm Minimization	Runs a CHARMm minimization process on a structure using all active atoms, whether or not they are currently displayed.
CHARMm Dynamics	Runs CHARMm dynamics simulations using all active atoms, whether or not they are currently displayed.
Move Atom	Translates an atom in the x, y, and z directions.
Move Fragment	Rotates and translates a selected bonded fragment.
Move Set	Rotates and translates a selected set of atoms.
Torsions...	Defines torsions for bond rotations in a molecular structure.
Spin	Automatically creates new conformations by rotating about active torsions. Searches for close contacts and stops when it finds a conformation with no close contacts, or when every conformation has been searched.
Continuous	Links the **Energy, Bumps** and **R/R0** functions if the relevant selection are made. Keeps them operating during modeling operations.
Energy	Calculates a relative energy calculation based on a 6-12 van der Waals potential and a Coulombic electrostatic potential between selected atoms.
Bumps	Reports close contacts between atoms in a moving structure and those in a stationary structure.

Table 7. Modeling palette (Continued)

Selection	Description
R/R0	Calculates the ratio of the current distance between atom pairs to the radius for an optimal van der Waals interaction. When this ratio is less than one, the atoms are too close and a dashed line connecting the atoms is displayed with the value of the ratio.
Solid Docking	Guides docking of one piece of a structure or molecule against another. Available as an option when **Move Fragment** is selected. When **Solid Docking** is selected, the modeling application limits motion of a fragment so it behaves as if other molecules were solid objects.
Sensitivity	Sets values of translation, rotation, and bumps distance cutoff for the solid docking facility.
Break Bond	Removes the bond between two picked atoms.
Make Bond	Creates a temporary bond to use, for example, to simultaneously manipulate fragments.
Cut Residue	Breaks all bonds to a selected residue, creating a fragment.
Recalculate Bonds	Recalculates bonds and redraws a structure. This can eliminate erroneous bonds caused by close contacts after mutation or fragment movement.
Dot Surface	Generates a van der Waals dot surface on selected atoms, residues, and molecules.
Hydrogen Bonds	Draws the hydrogen bonds in a structure.
Options...	Sets modeling parameters for distance cutoffs, spin increment value, and torsional features.
Undo Changes	Restores the original coordinates of an MSF, maintaining **Move Atom**, **Move Fragment**, **Move Set**, and **Torsion** tools, if active.
Save Changes	Writes new coordinates to the MSF and exits any active tools.
Reject Changes	Restores the original coordinates of an MSF and exits any active tools.

QUANTA decides when and if a particular palette is displayed, depending on:

- Functions selected from the menu bar.
- Dialog box input.
- Selections made within menus or palettes.

A function can be performed by making the appropriate selection from one or more palettes. Some selections remain active until they are manually turned off. Other selections are automatically turned off when the associ-

ated functions are complete. Depending on the palette and selections that you make, more than one selection can be made from a palette.

Some tools can become dimmed or grayed when others are active. These dimmed tools represent inappropriate choices based on what is currently selected. A tool cannot be selected when it is dimmed. Dimmed tools are restored to an active status when the selection of another palette tool no longer makes the dimmed tool an inappropriate choice.

Use the following exercise as practice for selecting and using palette tools. Before you do these exercises, make sure that you have selected **Multiple IDs** from the **ID Mode and Style** pull-right menu of the **Preferences** menu. All tools used in this exercise are on the Geometry palette.

1. Calculate the distance between atoms

On the Geometry palette, select **Distance**. The tool is checked and highlighted.

Pick an atom in the PEPTIDE.msf structure. The ID for the picked atom is displayed and additional information is reported in the textport.

Pick a second atom in the structure.

Again, the atom ID is displayed and information is reported in the textport. In addition, a distance monitor is displayed as a dashed line between the two picked atoms. The distance between the atoms, measured in angstroms, is shown in the viewing area and reported in the textport.

Pick two more atoms. Another distance monitor is displayed between these atoms on the screen, and the distance is reported in the textport.

2. Calculate distances using Continuous Pick Mode.

Select **Continuous Pick Mode.** The tool is checked and highlighted.

Pick two atoms. A distance monitor is displayed between the two picked atoms in the viewing area.

Pick a third atom. A distance monitor is displayed between the second and third picked atoms.

Pick a fourth atom. A distance monitor is displayed between the third and fourth picked atoms.

3. Turn off Distance and Continuous Pick Mode.

Select **Distance** again. When you release the mouse button the tool is no longer checked and highlighted, indicating it is no longer selected. The distances between atoms and atom IDs remain displayed.

Select **Continuous Pick Mode** and turn it off.

4. Clear IDs.

Select **Clear IDs**. All IDs are cleared from the viewing area but the distance monitors are still displayed.

5. Hide and show the distance monitors.

Select **Show Distance Monitors**. This tool, which by default is selected, is now turned off. The display of the distance monitors is turned off.

Select **Show Distance Monitors** again. This tool is highlighted. The distance monitors are again displayed.

6. Delete the distance monitors.

Select **Delete Distance Monitor**. The distance monitors are deleted from the viewing area.

7. Calculate the angle between atoms.

Select **Bond Angle**. The tool is checked and highlighted.

Pick three atoms in the structure.

An angle monitor indicating the angle formed by these atoms is displayed in the viewing area. The value of the angle is also reported in the textport. If the atom IDs obscure the angle monitors, use **Clear ID** to remove the IDs.

Pick three more atoms. Another angle monitor is displayed on the screen and listed in the textport.

8. Calculate both distances and angles.

Select **Distance.** Both **Distance** and **Bond Angle** are now selected.

Pick two atoms. A distance monitor is displayed between these two atoms.

Pick a third atom for an angle calculation. An angle monitor is displayed. The third atom can also be the first atom in another distance calculation.

9. Calculate torsion angles.

Select **Show Distance Monitors** and **Show Angle Monitors** to turn them off. Select **Dihedral**.

Pick four atoms in a new area of the structure to form the first torsion angle. A torsion monitor is displayed.

Pick four more atoms, thus forming a second torsion angle. A value is displayed, which represents the angle formed by the two planes.

10. Turn off all calculation selection. Delete monitors and IDs.

Select **Dihedral** to turn it off.

Select the **Delete Distance Monitors**, **Delete Angle Monitors**, and **Delete Dihedral Monitors** to delete monitors from the viewing area.

Select **Clear ID** to delete IDs from the viewing area.

Defining menu and window colors

Colors are automatically assigned for different parts of the QUANTA interface. You can change the colors by using Dial Set 7 on the Dial Emulator or by opening the **Preferences** menu, displaying the pull-right menu for **Color Definitions,** and selecting **Menu Colors**.

Each color used in QUANTA is determined by the interaction of hue (the actual color), saturation (percentage of color), and intensity (percentage of light).

Hue values are based on a 360° color wheel, where red is assigned a value of 0° or 360°. Color changes occur in 1° increments. The six pure colors occur at 60° increments. Table 8 lists these colors and their color wheel hue value.

A mix of colors falls anywhere within the range. For example, red is defined as 0°, and yellow as 60°. Between red and yellow are 59 incremental shades of orange. The 30° position represents an orange color composed of equal amounts of red and yellow.

Saturation values, specified in decimal values from 0 to 1, determine the purity of a color by how much white it contains. For example, saturation

Table 8. Color definitions

Color	Color wheel hue value
Red	0° or 360°
Yellow	60°
Green	120°
Light Blue	180°
Blue	240°
Purple	300°

values determine if a color with a hue value of zero is red (saturation = 1.000) or pink (saturation < 1.000). All pure colors have saturation values of 1 (100%).

Intensity values, specified in decimal values from 0 to 1, determine the brightness of the color by how much light it contains. For example, intensity values determine if a color with a hue value of 240 is blue (intensity = 1.000) or dark blue (intensity < 1.000). Black has the intensity value of 0.

Hue, saturation, and intensity parameters that define each structure color are stored in the constants file *filename*.cst.

In adjusting colors, you can adjust all three parameters — hue, saturation, and intensity. These three values combine to produce the actual display color. Table 9 provides several examples.

Table 9. Values combined for selected display colors

Hue	Saturation	Intensity	Display color
0	0.000	1.000	White
0	0.500	1.000	Light pink
0	1.000	0.500	Dark red
0	1.000	0.000	Black

Table 10 lists the choices in the dialog box that appears when you choose **Menu Colors** from the **Color Definitions** pull-right menu in the **Preferences** menu. The table describes the function or the default color setting and lists the dial equivalent for each dialog box selection.

Do the following exercises to practice adjusting menu colors.

1. Display menu color dials.

Table 10. Menu selections and dials for defining menu and window colors

Menu or window area	Dial	Default color or description
Text in menus, palettes, dialog boxes	Set 7 dial 5	Black except command line, where text and background are reversed
Dial emulators	Set 7 dial 4	Light green
Selected tools and boxes	Set 7 dial 3	Light blue
Background of text areas	Set 7 dial 2	Off-white except textport
Background of viewing area	Set 7 dial 1	Black
Blackout text areas	No dial equivalent. To reset menu colors, type on the command line: **Set color menu reset**. Then type **Set dbox on**.	Sets the background and text of all text area to the background of the viewing area.
Reset to default	No dial equivalent. Type on the command line: **Set color menu reset**	Sets all menu colors back to default values.

Select Dial Set **7** in the dial emulator selection box located at the bottom of the Dial Emulator window. The menu color dials are displayed.

2. Define the hue for menu color dial 2, that defines the background color for text areas

Move the cursor to the left side of the **H box** of dial emulator 2. Click the left mouse button.

The message line displays the current values of hue, saturation, and intensity for color 2.

With the cursor over the **H**, press and hold the left mouse button.

The hue changes for both dial emulator 2 and the background for text areas of the menu bar, palettes, dialog boxes, and the message line. The text of the command line is also affected. The hue value, displayed in the message line, decreases to reflect the change in the display. When this value reaches zero, it automatically changes to 360 and begins to decrease in value again. The hue of dial emulator 2 and the hue of the

text areas continue to change relative to the hue value.

Move the cursor to the right side of the **H box.**

The hue of dial emulator 2 and the hue of the text areas continue to change. However, the hue value displayed in the message line is now increasing as the color changes.

Release the mouse button. The color stops changing.

3. Define saturation for menu color 2

Move the cursor to the left side of the **S box** of dial emulator 2. With the cursor over **S**, press and hold the left mouse button.

The saturation changes for both dial emulator 2 and the text areas, darkening as the saturation value displayed in the message line decreases. When this value reaches 0.01, it automatically stops, as does the color change.

Move the cursor to the right side of the **S box**.

The saturation of dial emulator 2 and the saturation of the text areas resumes changing. However, the color is now lightening and the saturation value is increasing. When the saturation value reaches 1.000, it automatically stops, as does the color change.

Release the mouse button.

4. Define intensity for menu color 2

Move the cursor to the left side of the **I box** of dial emulator 2. Press and hold the left mouse button.

The intensity changes for both dial emulator 2 and the text areas. The intensity value, displayed in the message line, decreases to reflect the change in color. When this value reaches 0.01, it automatically stops. The intensity of dial emulator 2 and the intensity of the text areas also stops changing.

Move the cursor to the right side of the **I box**.

The intensity of dial emulator 2 and the intensity of the text areas resumes changing. However, the value is now increasing. When the intensity value reaches 1.00, it automatically stops. The intensity of dial emulator 2 and the intensity of the text areas also stops changing.

Release the mouse button.

5. Close dial emulator set 7.

Select Dial Set **1** in the dial selection box located at the bottom of the Dial Emulator window. The menu color dials are replaced by dial set 1.

Use the next exercises to practice defining menu colors using menu selections.

1. Display the dialog box containing menu color options.

Display the **Preferences** menu. Select **Color Definitions** from the menu and open its pull-right menu. Select **Menu Colors.** A dialog box is displayed that contains options for customizing menu colors.

2. Define the hue, saturation, and intensity for the background color of the viewing area.

Select **Background of Viewing Area** then select the **OK** button.

A dialog box containing data entry fields for hue, saturation, and intensity of the background color is displayed. Each entry box contains the currently set value.

Enter the values 300 for hue, 0.75 for saturation, and 0.50 for intensity, then click the **OK** button.

The background color of the viewing area is changed to reddish purple.

The dialog box containing available color options for the QUANTA window areas is redisplayed.

3. Return menu colors to their default settings.

Select the option **Reset to Default**, then click the **OK** button.

The background color of the viewing area is returned to its original hue, saturation, and intensity. Since the function resets any menu colors that were changed during the session, the color of the text areas is also returned to its original value.

The dialog box containing available color options for the QUANTA window areas is redisplayed.

Click the **Exit** button. The dialog box is removed from the screen.

You also can reset menu colors by selecting **Reset All** in the **Color Definitions** pull-right menu. This selection also resets any other color changes, such as atom colors, you made. If you do not reset menu colors

to the default settings, the menu and window color settings you chose are retained in future sessions.

Using the keyboard

The QUANTA interface, including menus, dialog boxes, and the mouse, eliminates the need for you to remember complex commands and minimizes use of the keyboard. Some keyboard entry is required to enter data in dialog boxes.

QUANTA contains command equivalents for all Molecular Modeling functions and for some functions in other applications. These commands can be used as an alternative to making selections from menus and dialog boxes.

Using the keyboard with data entry boxes

With the QUANTA standard interface, you must use the keyboard to enter information into data entry boxes. You can also use it to accelerate the process of selecting dialog boxes and executing functions. When you type characters on the keyboard, they are automatically entered into the highlighted data entry box of the open dialog box. The cursor does not need to be over the highlighted box for you to place characters into it.

You can accelerate the process of executing a dialog box by substituting the <F1> (enter) and <F2> (escape) keys for mouse selections.

Using the <Tab> key moves you sequentially through data entry boxes. Each box is highlighted as it become active. When you reach the last box, pressing <Tab> again returns you to the first box.

Keyboard commands

You also can use the keyboard to enter commands on the command line. The cursor does not need to be over the command line to enter commands, but it does need to be in the Molecule window. As you type, commands are shown on the command line. Dialog boxes prompt for any missing information. Commands can only be entered when a dialog box is not displayed.

Using commands permits you to execute operations without displaying menus and dialog boxes. Commands can be entered using uppercase or lowercase characters, and they can be abbreviated to the first four characters of the command.

Command-driven QUANTA

Command-driven QUANTA provides a secondary command input mode that displays command information in the textport instead of in dialog boxes. This mode is entirely non-graphical.

Commands can be executed in command-driven mode. However, commands are organized differently from menu selections. To enter command-driven mode, type:

```
>  set dbox off
```

To exit this mode, type:

```
>  set dbox on
```

See the *QUANTA Command Dictionary* for more information about using commands and for a complete list of QUANTA commands.

Complete the following exercises to practice using keyboard commands.

1. Display the About QUANTA dialog box with a keyboard command.

Place the cursor anywhere in the Modeling window and type the command:

```
>  about
```

The characters appear on the command line as they are typed.

Press the <Enter> key to activate the command.

The characters are erased from the command line, and the command is sent to QUANTA. The About QUANTA dialog box is displayed.

2. Use keyboard commands to remove the About QUANTA dialog box.

Press the <F1> key to select the **Continue** button in the dialog box. The dialog box is removed from the screen.

3. Change the QUANTA display with keyboard commands

QUANTA provides a function that changes the color of the contents of the dials, palettes, menu bar, structure name, command line, message line, and textport to the current background color. Adjustment of these colors is useful in generating presentation slides or printouts of structures. This function can be accessed using keyboard commands.

Press <F4>. The viewing area is enlarged to full-screen size and all other windows are hidden. Press <F4> again. The viewing area returns to normal. <F4> is an on-off toggle for **Photo Mode**. This function is similar but not identical to **Blackout Text Area** on the **Menu Colors** dialog box.

Binding commands to function keys

QUANTA commands can be bound to function keys on your workstation keyboard. Function keys <F3> through <F12> can be redefined. Either a single command or a pair of commands can be bound. For a single command, every time the function key is pressed, the command you associated with the key is sent to QUANTA. For two commands, the commands are sent alternatively to allow toggling.

By default, function key bindings are defined by the file $HYD LIB/function key.dat. To view the default settings, open the **Information** menu in the menu bar. Select **Current Session** to open a pull-right menu and select **Function Keys**. The function key bindings are listed in the textport.

The $HYD_LIB/funkey.ps PostScript file can be printed out and taped to your keyboard above the function keys to show the default bindings.

Complete the following exercise to become familiar with the procedure for assigning your own key bindings.

1. Define function key <F5>.

Open the **Preferences** menu in the **menu bar.** Select **Device Settings** to open the pull-right menu.

Select **Function Keys** from the menu. A dialog box is displayed with a default number 3 in the data entry box.

Enter 5 in the box to select function key 5 as the key to be defined.

Click **OK.**

Another dialog box is displayed asking:

How should FK5 behave?

Select the radio button for the choice:

The Key acts the same each hit.

Click **OK.**

A third dialog box is displayed, requesting:

Enter the command this key will send.

In the data entry box, type the command:

List atom all in the data entry field.

Click **OK.** You have defined key F5.

2. Define function key <F6>.

When you complete the last step in defining <F5>, the dialog box for specifying the function key number to be defined is redisplayed with the number 5 as the default.

Enter 6 in the data entry box to select the function key to be defined.

Click **OK.**

The next dialog box is displayed, asking:

How should FK6 behave?

Select the radio button for the choice:

The Key acts the same each hit.

Click **OK.**

A third dialog box is displayed, requesting:

Enter the command this key will send.

In the data entry box, type the command:

%Clear ID

The % denotes that the command is a palette selection.

Click the **OK** button. You have defined key <F6>.

3. Define function <F7>

The dialog box for specifying the function key number to be defined again is displayed, this time with the number 6 as the default.

Enter 7 in the data entry box to select the function key to be defined.

Click **OK.**

The next dialog box is displayed, asking:

How should FK7 behave?

This time, select the radio button for the choice:

The Key acts as a toggle.

Click **OK.**

The third dialog box is displayed, requesting:

Enter the command this key will send.

In the data field, type the entry:

VIEW STEREO

Click **Return**.

Another data entry dialog box is displayed.

Enter the alternate command:

VIEW MONO

Click **OK.**

4. Save your changes to a local directory

Finish the definition process by clicking **Cancel** in the last dialog box for <F7>. A final dialog box is displayed with saving selections.

Select **OK** to accept the default choice:

Save to Local Directory.

Executing system commands

QUANTA provides two methods of accessing your workstation's operating system when you are running QUANTA. The first is to open a standard workstation window. The second is to enter single operating system commands on the Command Line. This makes it possible to perform many operating system functions without exiting QUANTA. If you want to use any UNIX commands that require input, they must first be added to the STRINGTABLE in the $QNT_ROOT/resources/core_quanta.res file by your system administrator.

Window access to the operating system

To execute operating system commands on a workstation, you must create a standard workstation window. This window is manipulated in the same manner as any standard workstation window.

Use the following exercise to familiarize yourself with the procedure for reaching the operating system from QUANTA.

1. Create a system window

Display the **File** menu and select **Open System Window**.

A standard system window is displayed on the screen. The current directory in this window is the same as the directory from which QUANTA was started, for example, accelrys_quanta. All environment variables set in the startup window are also the same.

Move the mouse so the cursor is in the system window. You can now enter system commands in the system window.

2. Return to QUANTA

Iconify the system window by moving the cursor to the upper- right corner of the window and selecting the iconify button or close the window by typing **exit** within it. The system window is cleared from the screen to an icon at the top of the QUANTA screen or is removed completely.

Command line access to the operating system

A single specific operating system command can be entered on the QUANTA command line. QUANTA passes the command to the workstation operating system. When the operating system has completed executing the command, the results are passed back to QUANTA and reported in the textport.

To pass a command to the operating system, the operating system command must be preceded with the QUANTA command:

system or **sys**

If you enter this command, or if you use the **vi**, **view**, or **emacs** commands without adding an operating system command, QUANTA displays a system window.

QUANTA commands are case insensitive, but system commands must be entered using proper case.

Use this short exercise to practice command-line access to the operating system.

Pass the pwd command to the operating system.

Move the mouse so the cursor is in the Molecule window. Enter the command:

```
>  system pwd
```

The message:

```
UNIX>pwd
```

is displayed in the textport, indicating that the print working directory command has been sent to the operating system.

When the system finishes processing the **pwd** command, your current directory path is displayed in the textport.

Exiting QUANTA

Your work and current preferences are automatically saved when you exit QUANTA. The .cst file and currently open MSFs are saved into a common file when you select **Save Status** from the **File** menu.

To exit QUANTA, display the **File** menu and select **Exit QUANTA**. QUANTA windows are removed from the screen and you are returned to the system.

Summary

QUANTA is a windows-based application that supports importing, creating, and editing molecular structures and computer-based experimental simulation and analysis. The power of QUANTA is its ability to handle extensive and complex data about structures and to translate that data into graphic representations.

As a starting point for learning QUANTA, this chapter covered its basic layout and design and the fundamental procedures for getting started. In

particular, the chapter described how to use the QUANTA interface — windows, menus, palettes, dialog boxes, dial emulators, the keyboard, the mouse, and the Molecule Management table — to display and move structures and obtain information about them.

Table 11 generalizes and summarizes information in this chapter on making selections in QUANTA.

Table 11. Making selections in QUANTA

Selection	Instructions
Atom	Move the cursor over the end point of a bond or over the point where two bonds meet. Click the left mouse button to select the atom.
Button	Move the cursor over the desired button in the dialog box. Click the left mouse button to select the action described in the button.
Choice selection	Move the cursor over the circular button next to the desired option in the dialog box. Click the left mouse button to display a filled circular button and to select the option.
Data entry box	Move the cursor over the data entry box in the dialog box. Click the left mouse button to select the box. Use the keyboard to enter data in it.
Dial emulator	Move the cursor over the desired dial emulator. Press and hold the left mouse button to start the dial. While pressing the left mouse button, move the cursor to the left or right to change the speed and direction of the dial. Release the mouse button to stop the dial.
Menu selection	Display a pulldown. While pressing the left mouse button, move the cursor over the selection. When the desired selection is highlighted, release the left mouse button to select it. Display a sticky menu. Move the cursor over the desired selection. Click the left mouse button to select it.
Palette selection	Move the cursor over the desired selection on the palette. Click the left mouse button to select it. If the selection does not automatically turn off when the operation is complete, click the left mouse button again over the selection.
Pulldown menu	Move the cursor over the menu title to be displayed. Press and hold the left mouse button to display the menu. Release the mouse button to close the menu.
Scrolling list	To display a new section of a scrolling list in a dialog box, move the cursor over the up arrow or down arrow located to the right of the scrolling list. Click the left mouse button to select the scroll direction. To select an option from a scrolling list, display the section of the list that contains the desired option. Move the cursor over the desired option. Click the left mouse button to select the option in the scrolling list.
Sticky menu	Move the cursor over the name of the menu to be displayed. Click the left mouse button to display the menu in a window. Move the cursor back over the name of the menu and click the left mouse button to close the menu.
Textport	Move the cursor to the command line. Click the left mouse button to hide or display the textport.
Toggle selection	Move the cursor over the box next to the option in the dialog box. Click the left mouse button to put an X in the box and select the option. Click the left mouse button again in the box to remove the X and to turn off selection of the option.

2. Manipulating Molecular Displays

Most of this chapter focuses on how to manipulate molecule displays in the viewing area. The chapter describes display choices, including how to display and label complete or partial images of a structure and how to customize atom, label, and bond colors. It also includes how to select subsets of the atoms in your structure for calculations.

No manipulations described in the display sections of this chapter affect basic structural information used in modeling operations or calculations. Display manipulations *do not* affect this information in an active molecular structure or in an MSF on the disk.

In contrast, the *Selecting active atoms* section of this chapter describes methods for selecting some or all parts of a structure for modeling operations and calculations. These procedures *do* affect structural information used in modeling operations and calculations. They operate on the set of atoms contained in an active molecular structure, but not on the information contained in the disk-stored MSF. If you want to store a new information set you create through the active atoms selection process, you should save it to a new MSF.

Displaying a structure with selected active atoms may appear similar to displaying a structure where some atoms have been masked for visual purposes, but these two functions are actually very different. Do not confuse them.

Starting your QUANTA session

This chapter assumes you have already run QUANTA. If you have not, refer to Starting QUANTA in Chapter 1 for information about initial start-up.

To continue work on the exercises included in these QUANTA introductory books after subsequent start-ups, enter the command:

```
> cd ~/accelrys_quanta
```

Then enter the command:

```
> quanta
```

QUANTA opens by displaying the structure PEPTIDE.msf in the viewing area. The Molecular Management table displays the structure's name and status.

If you are opening QUANTA for any other purpose, do so in another subdirectory using the same procedure. The interface will show the last structure displayed in the previous session in that subdirectory.

Displaying structures and substructures

You can display some or all atoms of a molecular structure in the viewing area. By using masking to filter atoms, QUANTA helps you visually focus on one area or aspect of a structure. Meanwhile, any calculations that are performed include all atoms in the structure.

Display Atoms in the **Draw** menu is used to determine which atoms are selected for viewing. Atoms in a structure can be displayed using the predefined choices in the pull-right menu, or by choosing **Selection Tools** in the **Display Atoms** menu to bring up the Display Atoms and Display Utilities palettes. Using palettes takes more steps but gives you more flexibility in tailoring your display.

Selections in the **Display Atoms** pull-right menu are those used frequently. See Table 12 for a list and description of these selections, which display only the atoms matching the specified category. For example, you can select **Protein Backbone** and have only the atoms in the backbone displayed.

Table 12. Display Atoms menu

Selection	Description
All Atoms	Displays all active atoms in the structure.
All Except Solvent	Displays all atoms in the structure except residues of type WAT, HOH, TIP, TIP3, SOL, SOLV, ST2, NON, H2O, TIP4.
Non-Hydrogen Atoms	Displays all but hydrogen atoms in the structure.
Protein Backbone	Displays only protein backbone atoms.
C-Alpha trace	Displays a tracer line that connects the alpha carbons in a protein.
CA Trace+NonProtein	Displays protein as Cα trace plus all atoms in other molecules.
Selection Tools	Brings up the Display Atoms and Display Utilities palettes.

The Display Atoms palette includes the same display selections as the pull-right menu plus additional choices. Some selections result in automatic dis-

plays. Some require you to make additional selections. Table 13 lists all palette selections and provides a short functional description of each.

Table 13. Display Atoms palette

Selection	Description
All MSFs	Display selections affect all MSFs listed in the Molecular Management table.
One MSF...	Display selections affect specified atoms contained in the specified MSF.
Apply to All MSFs	Copies selection commands to all active structures.
Include	Includes all subsequent selections in the display.
Exclude	Excludes all subsequent selections from the display.
All Atoms	Selects all atoms in the structure.
All Non-hydrogen Atoms	Selects only the non-hydrogen atoms in the structure.
Solvent	Selects all residues of type WAT, HOH, TIP, TIP3, SOL, SOLV, ST2, NON, H2O, TIP4.
Alpha-Carbon Atoms	Selects only protein alpha-carbon atoms.
Protein Backbone	Selects only protein backbone atoms.
Nucleic Acid Backbone	Selects only atoms that compose the backbone of a nucleic acid molecule.
Residue Range...	Select residue range for display by completing information in a dialog box. The selection process may be repeated.
Residue Types...	Select residue types for display by completing information in a dialog box. The selection process may be repeated.
Atom Types	Select atom types for display by completing information in a dialog box. The selection process may be repeated.
Pick Molecule	One molecular structure is picked for display using the mouse
Pick Segment	A segment in a molecular structure is picked using the mouse. All atoms in the segment are selected for display. The process may be repeated.
Pick Residue	A residue in a molecular structure is picked using the mouse. The atoms are selected for display. The selection process may be repeated.
Pick Residue Range	A residue range is picked using the mouse. All atoms in the range are selected for display.
Pick Atom	An atom in a molecular structure is picked using the mouse. The selection process may be repeated.
Proximity Tools	Brings up the Proximity Tools palette containing multiple selections for display of nearby portions of a structure.
Type in a Selection	Selection commands for picking portions of a structure can be typed into a dialog box.
Undo	Returns display selection to the state before the previous selection.
Clear	Deletes all selections. Resets to default (Include).
Revert	Returns display selection to the state when the function was first entered.
Quit	Returns the display to the atoms contained in the original selection, and exits the palette without saving.
Finish	Displays the atoms contained in the new selection and exits the palette.

When you make a selection from the Display Atoms palette, the resulting display is in two colors. Default colors are blue for unselected atoms and red for selected atoms. When you first open the palette, all atoms are in blue (unselected), regardless of previous selections. You can revert to your previous selection by using **Revert** on the Atoms Display palette.

To help verify a selection, check the number of atoms listed in the **Displayed** column of the Molecule Management table. All atoms in a structure, whether selected or unselected for display, are included in calculations for the structure. Display selections are stored in a .dsf file.

After you have selected the portion of a molecule you want to display, make choices from the Display Utilities palette to view different aspects of the selected portion of the molecule.

See Table 14 for a complete list and description of the Display Utilities palette selections. If you have more than one MSF displayed, you can define separate criteria for displaying the atoms or structure within each MSF. For example, you can choose to display all the atoms in one structure and the backbone of another structure.

Table 14. Display Utilities palette

Selection	Description
Display options	
Mark Selected Atoms	Marks each selected atom with a purple asterisk.
Mark Unselected Atoms	Marks each unselected atom with a purple asterisk.
Color Selected Atoms	Displays selected atoms in color 3 (default is red). Displays unselected atoms in color 2 (default is blue).
Display Selected Atoms	Displays only selected atoms in the viewing area.
File and edit options	
Read Selection Commands	Reads commands from *file*.dsf and updates the viewing area to reflect the new set of commands.
Append Selection Commands	Appends to the current file, .dsf, with a set of commands from another file, *file*.dsf, and updates the viewing area to reflect the new combined set of commands.
Save Selection Commands	Saves the current set of commands to a new .dsf. Only available when **One MSF** is selected in the Display Atoms palette. Selections can only be saved for one MSF at a time.
List Selection Commands	Displays the current selection commands in the Textport.
Edit File Manually	Allows a .dsf file to be manually edited. Opens a new window, which uses the vi editor by default for editing the file[a].
Help on Commands	Provides a list in the textport of display selection commands that can be used from the keyboard. Each entry includes a short example.

[a]It is possible to change the default editor by using the **setenv** command before QUANTA is started. The syntax is: **setenv** ***EDITOR*** (name of system editor to be used).

Complete the following exercises to become familiar with the automatic display options.

1. Open the Display Atoms menu and make a selection.

In the **Draw menu**, select **Display Atoms** and open the pull-right menu.

Select **C-Alpha Trace**. The display of your structure becomes a light green trace line connecting the alpha carbons of the peptide.

Now select **Protein Backbone** from the pull-right menu. The C-alpha trace is replaced by a multi-colored display of the atoms that make up the protein backbone of PEPTIDE.msf. The textport and the **Displayed** column of the Molecular Management table report that 103 atoms are displayed.

2. Open the Display Atoms and Display Utilities palettes.

Go again to the **Display Atoms** menu. Select **Selections Tools**.

The molecule display reverts to all atoms. The Display Utilities and Display Atoms palettes appear to the right with their default settings checked and highlighted. The molecule changes color so that all atoms are blue, the default color for other unselected atoms.

In the upper-right corner of the viewing area, the words Selected and Unselected appear in blue and red respectively to remind you of the colors associated with display selections.

The textport message reads:

```
If you wish to start from your previous selection, please
use the Revert tool.
```

The **Displayed** column of the Molecule Management table indicates that 0 atoms are displayed, and the **Active** column indicates that all atoms (201) are active.

3. Redisplay the peptide backbone

Select **Revert** from the Display Atoms palette. The protein backbone atoms of the peptide are selected and displayed in red. Other atoms remain blue.

4. Further customize your display

Select **Display Selected Only** from the Display Utilities palette. The display changes so that only the atoms in the backbone appear in the same colors as the protein backbone displayed after it was selected from the pull-right menu.

Now select **Mark Selected Atoms** from the Display Utilities palette. The complete peptide molecule is displayed, but the atoms contained in the backbone of the structure are now marked with purple asterisks.

Select **Mark Unselected Atoms** from the palette. The complete molecule is displayed, but this time, atoms not contained in the backbone of the structure are marked with purple asterisks.

Select **Color Selected Atoms.** The display returns to a red and blue display.

5. Exit without saving your selection

Select **Quit** from the Display Atoms palette. The screen returns to standard viewing mode. The protein backbone of the structure is displayed as it was prior to using the palette tools.

The Display Atoms and Display Utilities palettes also provide selections that establish specifications based on designated atoms contained in the displayed structure. Using these tools is a three-step process.

a. Specify if one or more atoms are to be included or excluded in a display by selecting either **Include** or **Exclude** from the Display Atoms palette. **Include** is the default.

b. Define the structural category represented by selected atoms by selecting a choice from the Display Atoms palette, for example, **Pick Residue** or **Pick Segment.**

c. Using the mouse, pick an atom in the displayed structure to designate the portion of the molecule to be affected by the selected structural category from Step b.

The practice examples below illustrate this process.

1. Exclude a range of residues from a selection

Display the **Draw** menu. In the **Display Atoms** pull-right menu**,** select **Selection Tools**. The Display Utilities and Display Atoms palettes appear to the right with their default settings checked and highlighted.

The peptide molecule is displayed in blue (default color 2) indicating that all atoms are unselected. The words Selected and Unselected appear in the upper-right corner.

Select **Include** and **All Atoms** from the Display Atoms palette. The entire structure is selected and displayed in red.

Select **Exclude** and **Pick Residue Range...** from the Display Atoms palette. Using the mouse, select a range of residues to exclude by picking any two atoms in the structure.

As you pick, each atom ID is displayed. All atoms contained in the residue range that you identify are now unselected and highlighted in blue. Information about the picked atoms and the number of atoms in the selection is displayed in the textport in a message such as:

```
   Picked Atom: HZ1_SEG1:16 PEPTIDE.msf Residue Type: LYS
Picked Atom: CB_ SEG1:18 PEPTIDE.msf Residue Type: VAL
171 Atoms to be displayed
```

Choose **List Selection** from the Display Utilities palette. Information similar to the following is displayed in the textport:

```
The Current Display Selection is:
DATA PEPTIDE.msf
ALL
EXCLUDE
ZONE   SEG1:16   TO   SEG1:18
```

Select **Pick Residue Range** again to turn it off. Select **Clear ID** from the Geometry palette to remove IDs.

2. Examine residue types

Select **Exclude** and **All Atoms** from the Display Atoms palette. The molecule is displayed in blue (unselected). **Include** is automatically selected.

Select **Residue Type** from the palette. A dialog box is displayed with a list of residues in the molecule.

Select **Ser** from the list, then click **OK**.

The dialog box closes and all serine residues in the molecule are displayed in red. A message in the textport indicates the number of selected atoms in the display.

Repeat the process, this time selecting **Arg** in the dialog box. All arginines in the molecule are displayed in red along with the serine residues.

Select **Include** and **All Atoms** from the palette. The structure returns to an all-blue display of unselected atoms.

At any point during a selection process, it is possible to list the selection commands you have made thus far by choosing **List Commands** from the Display Utilities palette. The list is displayed in the textport.

Using proximity tools

If you select **Proximity Tools** on the Display Atoms palette, the Proximity Selections palette is opened. Selections on this palette allow you to display and study specific portions of the molecule and areas nearby.

When this palette is open, proximity dimensions are listed in the lower-right corner of the viewing area. For example, when you open the palette for the first time, information in the lower right corner of the viewing area reads:

```
Radius = 5.000 Å
```

This is the default radius set for all proximity selections.

With the opening of the Proximity Selections palette, Dial Set U opens in the dial emulator. The radius dial is displayed for controlling the radius of a sphere, the default for proximity selections. This and other dials displayed when proximity selections are activated work in the same manner as other translation and transformation dials.

Table 15 lists palette selections and provides a brief functional description of each. As in the Display Atoms palette, some proximity selections are predefined and some are interactive, allowing you to customize your selection. Interactive selections are multiple-step processes.

Table 15. Proximity Selections palette

Selection	Description
All MSFs	Display selections affect all MSFs listed in the Molecule Management table.
One MSF	Display selections affect specified atoms contained in the specified MSF.
Include	Inclusion tool for display choices. All selections are displayed.
Exclude	Exclusion tool for display choices. All selections are not displayed.
Atoms in Same Ring	Selects all atoms in same ring as subsequently picked atoms.
Connected Atoms	Displays a dialog box controlling extent of selection of atoms connected to subsequently picked atoms.
Around Atom	Display selections are made in proximity to the designated atom.
Around Residue	Display selections are made in proximity to the designated residue.
Around Segment	Display selections are made in proximity to the designated segment.
Around Molecule	Display selections are made in proximity to the designated molecule.
Graphical Sphere	Places a graphical sphere with a default radius of 5 Å in the viewing area. The sphere is placed initially over the center of the structure.
Graphical Cylinder	Places a graphical cylinder with both a default radius and length of 5 Å in the viewing area. Initially, the cylinder is placed over the center of the structure.

Table 15. Proximity Selections palette (Continued)

Selection	Description
Graphical Slab	Places a graphical slab with a default length, width, and depth of 5 Å in the viewing area. Initially, the slab is placed over the center of the structure.
Object Style	Allows the graphical selectors to be displayed as translucent or solid objects, or as an outline.
Whole Residues	Modifies the behavior of the previous seven selections. With this selection on, selection of any atom in a residue selects the whole residue.
Set Object Origin	Positions a graphical sphere, cylinder, or slab over a newly picked atom.
Origin at Coord	Positions the graphical selector at a specified coordinate.
Set Radius/Length	Allows user-selected graphical object parameters.
Apply Selection	Completes the atom selection process using graphical objects.
Undo	Returns the viewing area to the state before the previous selection.
Clear	Deletes all selections; resets to Include.
Revert	Returns the viewing area to the state when the Display Atoms palette was first entered.
Exit Proximity	Returns to the Display Atoms palette. Selected atoms are still displayed; graphical objects are not.

Complete the following exercises to familiarize yourself with the Proximity Selections palette selections.

1. Open the Display Atoms and Display Utilities palettes

Display the **Draw** menu. In the **Display Atoms** pull-right menu**,** select **Selection Tools**. The Display Utilities and Display Atoms palettes appear to the right with their default settings checked and highlighted.

2. Clear all previous display selections

With the Display Atoms palette open, select **Exclude** and **All Atoms**. The structure is displayed in blue.

Select **Proximity Tools** from the Display Atoms palette. The Proximity Selections palette opens. Proximity parameter information is displayed in the lower-right corner of the viewing area.

3. Make proximate atoms selections

Select **Around Atom** and use the mouse to select the atom of your choice.

The picked atom ID is displayed. All atoms within 5.0 Å of the picked atom are selected and displayed in red. The textport displays information about the picked atom and reports the number of selected atoms.

Return to the palette and select **Around Residue.** Using the mouse, pick a new atom to select a central residue.

The new atom ID replaces the previous selection, and a new set of proximate atoms is selected and displayed in red. The textport lists information about the new picked atom and reports the total number of selected atoms in the structure.

Choose **Clear** from the palette. All selected atoms are removed and the entire structure is displayed in blue. Select **Clear ID** from the Geometry palette to clear the viewing area of IDs.

4. Use a graphical object to select proximate atoms

Select **Graphical Cylinder** from the Proximity Selections palette.

A cylinder appears in the viewing area. In the lower-right corner, the default radius (5 Å) and length (5 Å) of the cylinder are displayed.

Select **Set Object Origin** from the palette, then use the mouse to pick an atom to be the origin around which the object is centered.

An ID is displayed and the object becomes centered around the atom you picked. The textport lists information about the atom, and the message line reports the coordinates of the cylinder's origin.

5. Manipulate the graphical object and structure to examine the selected atoms

Use the Rotate X, Rotate Y, and Rotate Z dials to adjust the position of the cylinder around its center.

Choose **Apply Selection** from the palette. Atoms within the cylinder are selected and displayed in red. The textport reports the number of atoms selected.

Select Dial Set 1 at the bottom of the dial emulator. Use the dials on this set or use your mouse and keyboard to move your molecule and the cylinder so that you can see selected atoms within the cylinder from a variety of views.

6. Remove the graphical object

Click **Graphical Cylinder** again to deselect it.

The cylinder is removed from the viewing area, but the atoms remain selected and displayed in red.

Select **Clear ID** from the Geometry palette to clear the atom ID from the viewing area.

7. Exit the proximity function

Return to the Proximity Selections palette and select **Exit Proximity**. The palette is closed and proximity parameter information is removed from the viewing area. Selected atoms remain selected and displayed in red.

8. Return to molecular modeling mode

Select **Include** and **All Atoms** from the Display Atoms palette. The entire molecule is displayed in red.

Select **Finish**. The Display palettes close and the peptide is displayed in standard molecular modeling mode.

Customizing display colors

Visual feedback in a molecular study is often enhanced by highlighting specific atoms with color. QUANTA provides a variety of options for assigning color to displayed structures and to specific areas within a structure.

Atoms in an MSF, as well as hydrogen bonds, IDs, and labels, are assigned color numbers and default colors. Any default color assignment associated with a particular color number or with a particular type of atom, bond, or label can be reassigned. You also can change the color of a predefined set of atoms or any specific selection of atoms in a structure, and you can highlight different areas or characteristics of that structure.

If more than one MSF is displayed, you can define criteria for coloring the atoms within each structure differently or define criteria for coloring some portions of all structures the same way.

Note: If something goes wrong with your colors, you can reset colors by entering the command **set colo reset**.

Customizing atom colors

Atoms contained in a structure can be displayed in as many as 14 colors. In addition, one color is assigned to IDs and labels, and one is assigned to hydrogen bonds. These colors are defined using the Dial Emulator, Dial Sets 5 and 6, or by a variety of choices provided by the **Color Definitions** selection in the **Preferences** menu.

Table 16 lists the dials in sets 5 and 6, their assignments, and their default colors.

Table 16. Dials for defining MSF colors

Dial set/dial	Color number	Default color
Set 5		
Dial 1	Atom color 1	Green
Dial 2	Atom color 2	Blue
Dial 3	Atom color 3	Red
Dial 4	Atom color 4	Yellow
Dial 5	Atom color 5	White
Dial 6	Atom color 6	Light yellow
Dial ID	Label and ID color	Bright green
Dial HB	Hydrogen bond color	White
Set 6		
Dial 7	Atom color 7	Bright green
Dial 8	Atom color 8	Purple
Dial 9	Atom color 9	Light grey
Dial 10	Atom color 10	Beige
Dial 11	Atom color 11	Dark grey
Dial 12	Atom color 12	Light blue
Dial 13	Atom color 13	Orange
Dial 14	Atom color 14	Pink

See Dial emulator for a basic description of dial emulators and their manipulation.

For the color dial sets, each dial has four parts:

- Label box indicating a color represented by the dial.
- H box controlling hue values.
- S box controlling saturation values.
- I box controlling intensity values.

The following exercise illustrates manipulation of the color dial emulators to customize the colors you use in your MSF displays:.

1. Display the first set of color dials

Move the cursor over the **5** in the **Dial Emulator Selection** box located at the bottom of the dial emulator. Click the left mouse button.

The first six atom color dials, and the color dials for labels and IDs (**ID**) and hydrogen bonds (**HB**), are displayed. By default, the color of dial 1 (light green) is assigned to the carbon atoms in the displayed structure.

2. Define the hue for color 1

Move the cursor to the left side of the **H** box of dial emulator 1 and click the left mouse button. The message line displays the current values of hue, saturation, and intensity for color 1.

With the cursor over **H**, press and hold the left mouse button.

The hue changes for both dial emulator 1 and the carbon atoms in the displayed peptide structure. The hue value, displayed in the message line, decreases to reflect the change in the display. When this value reaches zero, it automatically changes to 360 and decreases in value again. The hue of dial emulator 1 and the hue of the carbon atoms in the displayed structure continue to change relative to these values.

As you continue to hold the left mouse button down, move the cursor to the right side of the **H** box.

The hue of dial emulator 1 and the hue of the carbon atoms in the displayed structure continue to change. However, the hue value displayed in the message line is now increasing.

Release the mouse button. The hue stops changing.

3. Define the saturation for color 1

Move the cursor to the left side of the **S** box of dial emulator 1. With the cursor over the **S**, press and hold the left mouse button.

The saturation changes for both dial emulator 1 and the carbon atoms. The saturation value, displayed in the message line, decreases to reflect the change in color. When this value reaches 0.01, it automatically

stops. The saturation of dial emulator 1 and the saturation of the carbon atoms also stop changing.

As you continue to hold the left mouse button down, move the cursor to the right side of the **S** box.

The saturation of dial emulator 1 and the saturation of the carbon atoms resume changing. However, the value is now increasing. When the saturation value reaches 1.00, it automatically stops. The saturation of dial emulator 1 and the saturation of the carbon atoms also stop changing.

Release the mouse button. The saturation stops changing.

4. Define the intensity for color 1

Move the cursor to the left side of the **I** box of dial emulator 1. Press and hold the left mouse button.

The intensity changes for both dial emulator 1 and the carbon atoms.

The intensity value, displayed in the message line, decreases to reflect the change in color. When this value reaches 0.01, it automatically stops. The intensity of dial emulator 1 and the intensity of the carbon atoms also stop changing.

As you continue to press and hold the left mouse button, move the cursor to the right side of the **I** box.

The intensity of dial emulator 1 and the intensity of the carbon atoms resumes changing, however, the value is now increasing. When the intensity value reaches 1.00, it automatically stops. The intensity of dial emulator 1 and the intensity of the carbon atoms also stop changing.

Release the mouse button. The intensity stops changing

Using menu functions to customize atom colors

Each atom display color also can be individually defined using menu selections. Selecting **Color Definitions** in the **Preferences** menu opens a pull-right menu that contains the selections listed and described in Table 17. In most cases, selections are predefined and automatically applied.

Table 17. Color Definitions pull-right menu

Selection	Description
All Definitions	Provides a dialog box in which all options for color customizing are presented for selection.
Blue to Red	Establishes a range from blue to red for all 14 atom colors. Used when coloring structures by a property, such as charge.
Red to Blue	Establishes a range from red to blue for all 14 atom colors. Used when coloring structures by a property, such as charge.
Black and White	Sets all MSF display colors to black, white, and greys in a white viewing area.
Reset Atoms	Resets atom colors to default values.
Reset All	Resets all atom and menu color displays to default values.
Menu colors	Establishes colors for various QUANTA menu areas.

The **All Definitions** selection in the **Color Definitions** pull-right menu opens a Customize Color Settings dialog box with the selections listed and described in Table 18. Some of the selections in this box are identical to those on the pull-right menu.

Table 18. Customize Color Settings dialog box

Selection	Description
Display Colors 1 to 14	Provides options to define the hue, saturation, and intensity of any of the 14 atom colors.
Reset Atom Coloring to Default	Sets all atom colors back to their default values.
Black & White	Sets all MSF display colors to black, white, and greys in a white viewing area.
Greyscale Range for Hardcopy	Displays options for setting a black to white range for the 14 atom colors.
CPK Drawings	Sets colors that are recognized conventions for CPK space-filling models.
Solid Models	Sets colors suitable for displaying solid models on a white background.
Spectral Ramp	Displays a dialog box for setting up a customized range of fully saturated colors.

Table 18. Customize Color Settings dialog box (Continued)

Selection	Description
Electrostatic Ramp	Displays a dialog box for setting up a customized range of varying saturations.
User-Defined Smooth Range	Based on user-specified criteria, establishes a range for two or more display colors.
Blue to Red Smooth Range	Establishes a range from blue to red for all 14 atom colors. Used when coloring structures by a property, such as charge.
Red to Blue Smooth Range	Establishes a range from red to blue for all 14 atom colors. Used when coloring structures by a property, such as charge.
ID/Labels	Establishes color of labels and IDs.
Hydrogen Bonds	Establishes color of displayed hydrogen bonds.
Close Contacts	Establishes color for displayed lines and distances when the **Bumps** tool is selected.
Neighbors	Establishes color for displayed lines and distances when the **Neighbors** tool is selected.
Menu Colors	Establishes colors for various QUANTA menu areas.
Save Current Settings	Saves settings you have selected in the current session. You can save these settings as personal defaults in the default file default.grb.
Use Saved Settings	Recalls and uses saved settings.
Reset All to Default	Sets all colors back to their default values.

Use the following exercise to become familiar with menu functions that define MSF display colors:.

1. Make a selection from the Color Definitions menu

Display the **Preferences** menu and select **Color Definitions**. A pull-right menu opens. Select **Black and White** from the pull-right menu. The MSF in the viewing area is displayed in black, white, and grey colors against a white background.

Select **Reset All** from the pull-right menu. The display returns to default colors.

2. Set hue, saturation, and intensity for an atom display color

Select **All Definitions** from the **Color Definitions** menu.

The Customize Color Settings dialog box is displayed.

Select **Display Colors 1 to 14,** then click **OK.**

A dialog box is displayed and asks for the color number to be changed.

Enter **2** and click **OK**.

A dialog box containing data entry fields for hue, saturation, and intensity of color 2 is displayed. Each field contains the current value. By default, color 2 is blue and is assigned to nitrogen atoms in the displayed structure.

Enter the values 300 for hue, 1.0 for saturation, and 1.0 for intensity. Click **OK**.

The color of the nitrogen atoms in the displayed structure changes to purple. Dial emulator 2, used to define color 2, also changes to purple.

The dialog box containing available color options for structures is redisplayed.

Click **Exit** to close the box.

3. Define a color for a non-atom portion of an MSF display.

Pick an atom in the displayed structure.

The atom should be located to the far right of the viewing area, so that it is not obscured by dialog boxes. The atom ID is displayed in the default green color.

Reopen the Customize Color Settings dialog box by displaying the **Preferences** menu, opening the pull-right **Color Definitions** menu, and selecting **All Definitions**.

Select **ID/Labels** from the dialog box and click **OK**. A dialog box containing data entry fields for hue, saturation, and intensity of ID/Labels is displayed. Enter the values 60, 1.00, and 1.00 in each successive data entry box. Click **OK**.

The color of the displayed ID changes to yellow. The dialog box containing available options for display colors is reopened.

4. Define a set of structure colors.

Select **CPK Drawings** from the dialog box and click **OK**.

Atom and viewing area colors are changed to the CPK convention.

The dialog box containing available color options for structures is redisplayed.

Click **Exit** to remove the dialog box and view the entire structure.

5. Reset all colors to their defaults.

Display the **Preferences** menu and open the **Color Definitions** pull-right menu. Select the option **Reset All**. The viewing area and all structure colors return to their default values.

6. Close Dial Set 5.

Move the cursor over the 1 in the dial selection box at the bottom of the dial emulator. Click the left mouse button. The color dials set is replaced by Dial Set 1.

Assigning structure colors

The two types of color assignment options are automatic and user-defined. Automatic color assignment requires atoms to be specified by category, while user-defined options allow you to specify exactly the atoms or areas of a structure that you want to color according to various criteria.

QUANTA assigns color numbers based on your selections. The color number corresponds to one of the 14 colors for atom display. The actual color this number represents is defined by default or by adjustments you make using dial emulators or the **Color Definitions** function in the **Preferences** menu. For information on color definition, see “Customizing display colors” on page 75.

You make color assignments by selecting **Color Atoms** from the **Draw** menu. A pull-right menu contains several commonly used selections. These are listed along with brief functional descriptions in Table 19.

Table 19. Color Atoms pull-right menu

Selection	Description
By element	Colors display atoms according to their elemental makeup. Default colors apply unless redefined.
By segment	Colors display atoms by segment. First segment color is color 14 (default = pink), second segment color is color 13 (default = orange), and so forth.
By molecule	Colors entire molecule a single color. Default color for first molecule is color 1 (light green).
Selection tools	Opens Color Atoms and Color Schemes and Utilities palettes and Color Dials Set 1.

If you select **Selections Tools** from the pull-right menu, the Color Atoms and Color Schemes and Utilities palettes are opened and displayed to the right of the viewing area. In addition, Color Dials Set 1 comes up in the Dial Emulator window.

The legend ***Color 2*** is displayed in the top right-corner of the viewing area, indicating the color that will be applied to subsequent selections. The pull-right menu is dimmed and cannot be accessed again until the palettes are closed.

The Color Schemes and Utilities palette includes tools that assign subsequently selected atoms to be colored according to chemical or physical properties or according to atom location in the MSF, or color all structures according to molecule, segment, or bond order.

The palette also includes tools that set the display color values (default color, blue-to-red smooth range, red-to-blue smooth range, and user-defined smooth range). These selections are also available through **Color Definitions** in the **Preferences** menu.

A third category of tools on this palette is editing and filing. Table 20 lists the palette selections along with brief functional descriptions.

Table 20. Color Schemes and Utilities palette

Selection	Description
Color schemes	
Next Color Number	Causes the Color By Number scheme to use the next of the 14 colors.
Choose Color Number	Allows any of the 14 colors to be used with the Color by Number tool.
Color by Number	Subsequently selected atoms will be colored using the color number defined using the first two tools.
Element Type	Colors subsequently selected atoms by element.
Atomic Charge	Colors subsequently selected atoms in ascending order of charge value using the 14 atom color numbers. The display colors should be set to a smooth or tonal range when using this scheme.
Fourth Parameter	Displays dialog box requiring user-defined input of a fourth parameter for coloring the subsequently selected atoms. The display colors should be set to a smooth or tonal range when using this scheme.
Property Value	Colors subsequently selected atoms using the current property definition. (See “Atom property coloring” on page 89.).
Amino Acids by Polarity	Colors subsequently selected amino acid residues according to their polarity. Each residue is assigned a color number predefined in QUANTA. These assignments are: TYPE LYS ARG HIS = COL 2 TYPE ASP GLU = COL 3 TYPE SER THR CYS TYR ASN GLN = COL 4 TYPE ALA VAL LEU ILE PRO PHE TRP MET GLY = COL 5
Secondary Structure	Colors each subsequently selected residue by its secondary structure type as defined in the Protein Design application.

Table 20. Color Schemes and Utilities palette (Continued)

Selection	Description
Each Molecule a Different Color	Colors each MSF with a different color.
Each Segment a Different Color	Colors all atoms by segment. First segment color is Color 14 (default = pink), second segment color is Color 13 (default = orange), and so forth.
Color by Bond Type	Colors bond types with different color numbers: Single bond = Color 1 Double bond = Color 2 Triple bond = Color 3 Aromatic bond = Color 7 Resonant bond = Color 12 Undefined bond = Color 14 This selection overrides all other color assignments but is only available if bond orders are stored in the MSF.
Color definitions	
Default Color	Resets atom colors to the QUANTA default values.
Blue-to-Red Smooth Range	Changes the 14 atom colors to range from blue to red.
Red-to-Blue Smooth Range	Changes atom colors to range from red to blue.
User-Defined Smooth Range	Displays a dialog box that allows you to set up a customized spectral color ramp (varying fully saturated hues).
Blue to Red Tonal Range	Changes atom colors to ramp from blue to red through white.
User-Defined Tonal Range	Displays a dialog box that allows you to set up a customized “electrostatic” color ramp (varying saturations).
Edit and File tools	
Read Selection Commands	Reads selection commands from a selection file, overwriting current selection commands and updating the viewing area.
Append Selection Commands	Appends entries in a selection file to the current command list and updates the viewing area.
Save Selection Commands	Saves selections to a file. Available only if the **One MSF** selection is checked on the Color Atoms palette. Selections can only be saved one MSF at a time.
List Selection Commands	Displays current selection commands in textport.
Edit File Manually	Allows a .csf file to be manually edited. Opens a new window, which by default uses the vi editor for editing the file.
Help on Commands	Provides a list in the textport of color selection commands that can be entered from the keyboard. Each entry includes a short example.

The Color Atoms palette contains most of the selections available for user-defined assignment of colors. Table 21 lists the selections in this palette and provides a brief functional description of each. Unlike the other palettes that affect what is displayed in the viewing area, the Color Atoms palette does not include the selections **Include** and **Exclude**. Color selections are not binary (on or off), but display and label selections are.

Table 21. Color Atoms palette

Selection	Description
All MSFs	Color assignments affect all MSFs listed in the Molecule Management Table.
One MSF	Color assignments affect atoms selected from the specified MSF.
Apply to All MSFs	Applies color assignments to all active MSFs.
All Atoms	The current color scheme is applied to all atoms.
All Non-Hydrogen Atoms	The current color scheme is applied to all atoms except hydrogen.
Solvent	Displays solvent residues in the current color scheme.
Alpha-Carbon Atoms	Displays all alpha carbon atoms in the current color scheme.
Protein Backbone	Displays a protein backbone in the current scheme.
Protein Sidechain	Displays protein sidechains in the current color scheme.
Nucleic Acid Backbone	Displays the backbone atoms of a nucleic acid in the current color scheme.
Residue Range	Displays atoms contained within the specified range of residues in the current color scheme.
Residue Type	Displays atoms of picked residue type in the current color scheme.
Atom Type	Displays atoms of a picked atom type in the current color scheme.
Pick Molecule	Displays all atoms of a picked molecule in the current color scheme.
Pick Segment	Displays all atoms of a picked segment in the current color scheme.
Pick Residue	Displays all atoms of a picked residue in the current color scheme.
Pick Residue Range	Displays all atoms of a specified residue range in the current color scheme.
Pick Atom	Displays picked atom in the current color scheme.
Proximity Tools	Opens the Proximity Selections palette.
Type in a Selection	Displays dialog box requiring user-defined input of specific selection and color commands.
Undo	Returns viewing area to the state before the previous command.
Clear	Clears all color selections leaving all atoms displayed in color 1.
Revert	Returns viewing area to the state when **Color Atoms** was selected.
Quit	Quits without saving changes, reverting to the original file.
Finish	Saves the new color assignments and exits the palette.

The **Proximity Tools** selection in the Color Atoms palette opens the Proximity Selections palette for assigning colors to atoms close to a portion of the structure you have selected. Any selections you make using this palette displays all proximate atoms using the current color scheme.

The selections on this palette are the same as those on the palette described in “Using proximity tools” on page 71, except that the **Include** and **Exclude** tools are not required.

Use the following exercises to become familiar with color assignment functions:

1. Make a color assignment using the pull-right menu.

Open the **Draw** menu and select **Color Atoms** to open the pull-right menu. Select **By Segment**. The entire peptide turns pink (color 14).

2. Make color assignments using the Color Atoms and Color Schemes and Utilities palettes.

Display the **Draw** menu again and select **Color Atoms** to open the pull-right menu. Select **Selection Tools.**

The Color Atoms, Color Schemes and Utilities, and Color Dials Set 1 palettes are displayed to the right of the viewing area with their default settings. The legend ***Color 2*** is displayed in the top- right corner of the viewing area, indicating that selected atoms will be displayed in color 2. The structure is still displayed in color 14 (pink).

3. Color the Protein backbone in PEPTIDE.msf.

Select **Protein Backbone** from the Color Atoms palette.

The backbone is displayed in color 2 (default = blue), and the rest of the molecule continues to be displayed in pink.

Select **List Selection-Commands** from the Color Schemes and Utilities palette.

The textport reads:

```
The current color selection is:
DATA PEPTIDE.msf
Segm Seg1 = COL 14
PICK C N CA HA HA1 HA2 O H HN HT? OCT? OXT? CT? = COL 2
```

4. Color leucine residues.

Select **Next Color Number** from the Color Atoms palette.

The information in red in the upper-right corner of the viewing area reads:

```
*Color 3*
```

Select **Residue Type** from the Color Atoms palette. A dialog box opens. From the scrolling list of residues, pick Leu**.** All leucine residues in the structure are displayed in red.

Select **List Selection-Commands** from the Color Schemes and Utilities palette.

The textport reads:

```
The current color selection is:
DATA PEPTIDE.msf
Segm Seg1 = COL 14
PICK C N CA HA HA1 HA2 O H HN HT? OCT? OXT? CT? = COL 2
TYPE LEU = COL 3
```

5. Changing the color scheme.

Select **Color by Atomic Charge** from the Color Schemes and Utilities palette.

A textport message reads:

```
   Note - you should set the bond colors to a smooth range
when using this mode
```

Select **Blue-to-Red Smooth Range** from the Color Schemes and Utilities palette.

The molecule's colors and the color dials change to reflect this choice. Color 1 is blue and color 14 is red. Color numbers and charge increase in tandem.

Now select the **All Atoms** tool to color the whole molecule by charge.

6. Return to Molecular Modeling mode.

Select **Default Colors** from the Color Schemes and Utilities palette and then select **Quit** in the Color Atoms palette.

The application returns to Molecular Modeling mode. The peptide is displayed in pink.

From the **Draw** menu, reopen the **Color Atoms** pull-right menu and select **By Element.** The peptide colors return to standard element default colors.

Atom property coloring

The **Property Colors** command on the **Draw** menu gives access to functions that facilitate coloring atoms or residues according to various properties.

Define properties

This opens a dialog box that sets up property coloring.

Property

This radio button group specifies which property is to be used. The choices are:

- **Atomic Charge**
- **Atomic Mass**

- **VDW Radius**
- **Fourth Parameter**
- **Position of Secondary Structure Element**
- **Residue Position**

There is a toggle box, **Calculate over Residue**, which can be applied to any of the first four properties. This allows the property to be computed over each residue as the **Average** (optionally with main/side chain discrimination), **Sum**, **Minimum** or **Maximum**.

If the **Fourth Parameter** is chosen, then there is also the choice of which item of extra information to load. This can be per-atom or per-residue information of type REAL, INT4, INT2, or ASTR.

The **Position of Secondary Structure** property assigns the atoms in each sheet or helical secondary structure element to consecutive color numbers, leaving other parts white. The colors should be set to a spectral ramp.

The sixth property, **Residue Position**, can be defined relative to each segment, each MSF or within all atoms of all loaded MSFs. Finally, you specify which of the 14 available colors to use for the property coloring and can have these automatically set up to be a **Spectral** or **Electrostatic** ramping or leave the color map as before.

Normalization

Allows the property values to be normalized. The choices are:

Do Not Normalize

Values are left as they are.

Normalize

Normalizes the values within the range 0–1, using the formula:

$$normalized\_prop = \frac{property\_value - minimum}{maximum - minimum}$$

Zero Point Normalization

Normalizes the values in the range -1 to 1:

- For *property_value* < *central_value*:

$$normalized\_prop = \frac{property\_value - central\_value}{center - minimum}$$

- For *property_value* > *central_value*:

$$normalized\_prop = \frac{property\_value - central\_value}{maximum - central\_value}$$

The normalization can be performed relative to all MSFs or within each MSF.

Spectral colors

Displays a dialog box that allows the creation and adjustment of a spectral color ramp. This is a gradation of the colors through different hues, all with full saturation and intensity. The molecule is scaled down and moved so that it can still be seen to judge the effect during the adjustments, and the color legend is displayed.

The dialog box allows control over which of the 14 colors to use for the ramp. The starting and ending hues can be specified, in terms of a color wheel number or symbolic name:

RED=0 (360)
YELLOW=60
GREEN=120
CYAN=160
BLUE=240
PURPLE=270
MAGENTA=320

Intermediate hues can be specified numerically or by “thumbwheel” adjustments. You can go around the color wheel clockwise or counterclockwise: if the starting hue is red and the final color is blue, then clockwise goes through green and counterclockwise goes through purple.

Electrostatic colors

Similar to **Spectral Colors**, but ramps from a fully saturated color down to white. There is a choice for whether the ramp should be white at the minimum color number, the maximum, or the midpoint. In the last case, two hues are specified. The typical electrostatic color scheme goes from blue to red via white.

Toggle Legend

Toggles the display of the color legend object.

Toggle blended bonds

By default, QUANTA uses half-bond vectors, i.e., if two atoms of different colors are bonded then the bond is drawn half one color and half the other. In some cases, in particular when coloring by a smoothly varying property, it is more visually appealing to blend the colors along the bond. This option toggles between half-bond and blended-bond style. This setting also applies to solid representations such as licorice.

In addition to the visual representation of properties as color, the values are also included in the Atom Property table.

Status of properties

The command PROP STAT prints information about the current property calculation to the textport. This is of the following form:

```
Fourth Parameter: BVALUE for each atom
No normalization performed
Minimum:       0.000
Maximum       49.398
Mean          10.821
Sum        18861.395
SD:            7.628
      Low         High     Number
      0.0000       3.5285  222
      3.5285       7.0569  392
      7.0569      10.5854  385
      10.5854      14.1138  305
      14.1138      17.6423  173
      17.6423      21.1708  102
      21.1708      24.6992   52
      24.6992      28.2277   40
      28.2277      31.7561   33
      31.7561      35.2846   16
      35.2846      38.8131   13
      38.8131      42.3415    5
      42.3415      45.8700    3
      45.8700      49.3984    2
```

Type of property calculation

Whether data is normalized

Minimum value

Maximum value

Mean

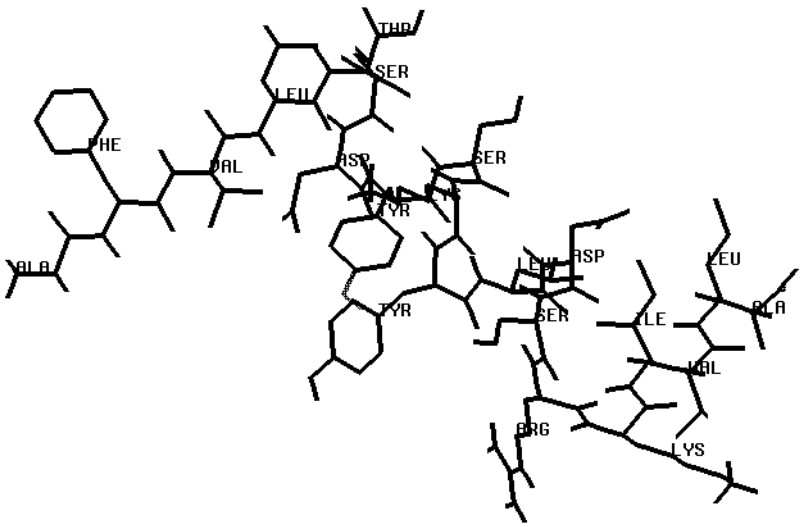
Sum

Standard deviation

The range of values in each category (color) of data, and the number of atoms for which the property falls into each category. The number of categories is determined by the number of colors used to display the property.

Labeling atoms

Atoms or groups of atoms can be selected and labeled. Labels include information such as atom name, residue ID, and atomic charge. Each label can be customized to contain different pieces of information. As label selections are made, labels are displayed next to the appropriate atoms. Labels can be assigned to atoms whether or not they are displayed. You can produce a labeled structure with one or more types of labels, each associated with one or more selected atoms or atom groups.



Label Atoms in the **Draw** menu is used to determine which atoms have labels. Labels can be displayed using the predefined choices in the **Label Atoms** pull-right menu or by choosing **Selection Tools** to open the Label Atoms and the Label Components and Utilities palettes. Using palettes requires more steps but gives you more flexibility in tailoring labels.

Selections in the pull-right menu are those used most commonly for labeling atoms and residues. See Table 22 for a list and description of the selections in this menu. Display of labels selected from the pull-right menu can be turned on and off from this menu.

The Label Atoms and Label Components and Utilities palettes together include the same display selections as the pull-right menu, plus numerous additional choices. The Label Atoms palette selections determine which atoms or group of atoms are labeled. Table 23 lists all palette selections and

Table 22. Label Atoms pull-right menu selections

Selection	Description
Off	Removes all labels from viewing area.
Atom Name	Labels each atom with atom name.
Atom Type	Labels each atom with atom type.
Untyped Atoms	Labels any untyped atoms as ‘”Untyped”.
Atom Charge	Labels each atom with atomic charge.
Residue ID	Labels each residue with residue ID.
Residue Name	Labels each residue with residue name.
Termini	Labels protein N and C termini
On	Displays previously defined labels in viewing area.
Selection Tools	Opens Label Atoms and Label Components and Utilities palettes.

provides a short functional description of each. Most of the selections in this palette work in conjunction with **Include** and **Exclude** to generate specified labels.

Table 23. Label Atoms palette

Selection	Description
All MSFs	Label selections affect all MSFs listed in the Molecule Management Table.
One MSF	Label selections affect all atoms contained in the specified MSF.
Apply to All MSFs	Applies label selections to all active MSFs.
Include	Inclusion tool for label choices. All selections will be displayed.
Exclude	Exclusion tool for label choices. Selections will not be displayed.
All Atoms	Adds labels on all atoms in MSF using currently selected label components.
All Non-Hydrogen Atoms	No hydrogen atoms are labeled.
First Atom of Residues	Labels are only displayed on first atoms of residues.
Central Atom of Residues	Labels are displayed only on central atoms of residues.
Alpha Carbon Atoms	Labels are displayed only on alpha carbon atoms.
Protein Backbone	Labels are displayed only on protein backbone atoms.
Nucleic Acid Backbone	Labels are displayed only on nucleic acid backbone atoms.
Residue...	A dialog box queries for residues to label in a specified range.
Residue Type...	A dialog box gives a scrolling list to select a specific residue type to be labeled.
Atom Type...	A dialog box gives a scrolling list to select a specific atom type to be labeled.
Pick Molecule	Selection is made from the active MSFs on the screen using the mouse.

Table 23. Label Atoms palette (Continued)

Selection	Description
Pick Segment	Labels are applied to atoms in a segment. Selection is made on the screen using the mouse.
Pick Residue	Labels are applied to atoms in a residue. Selection is made on the screen using the mouse.
Pick Residue Range	Labels are applied to atoms in a residue range. Selection is made on the screen using the mouse.
Pick Atom	A label is applied to a chosen atom. Selection is made on the screen using the mouse.
Type in a Selection	A dialog box is used to enter selection commands.
Undo	Returns viewing area to the state before the previous command.
Clear	Removes the display of all labels.
Revert	Returns viewing area to the state when **Label Atoms** was selected.
Quit	Quits without saving changes, reverting to the original file.
Finish	Saves the new label selections and exits the palette.

The Label Components and Utilities palette selections determine the information displayed in labels. Table 24 lists all palette selections and provides a short functional description of each.

Table 24. Label Components and Utilities palette

Selection	Description
Selection Tools	
Atom Name	Labels include atom names.
Segment Name	Labels include associated segment names.
Residue ID	Labels include residue ID names.
MSF Name	Labels include MSF name.
Residue Name	Labels include residue names.
Atomic Charge	Labels include atomic charges.
Atom Type	Labels include atom types.
Fourth Parameter	Labels include fourth parameter values.
Molecule Activity	Labels include molecule activity values.
Atom Number	Labels include atom numbers.
User defined...	Dialog is displayed from which a user label can be input.

Table 24. Label Components and Utilities palette (Continued)

Selection	Description
Residue Name Style	Opens a dialog box allowing you to control whether the residue type of standard groups should be displayed as the normal 3-character name (e.g., ALA), or the 1-letter code (e.g., A) or the full name (e.g., Alanine). This setting is used for atom labels and IDs.
Show Labels	Toggles labels on and off.
File and Edit Options	
Read Selection-Commands	Reads command from *file*.lab and updates the viewing area to reflect the new set of commands.
Append Selection-Commands	Appends the current *file*.lab with a set of commands from another *file2*.lab, and updates the viewing area to reflect the new combined set of commands.
Save Selection-Commands	Saves the current set of commands to a new file.
List Selection-Commands	Lists the current set of commands in the textport.
Edit File Manually	Allows a .lab file to be manually edited. Opens a new window which, by default, uses the vi editor for editing the file.
Help on Commands	Provides a list in the textport of display selection commands that can be used from the keyboard. Each entry includes a short example.

Complete the following procedures for practice with generating labels.

1. Choose label information from the pull-right menu.

Display the **Draw** menu and select **Label Atoms** to open the pull-right menu. Labels are automatically turned on when you select **On**.

Select **Atom Name** from the pull-right menu. All atoms are labeled by name in bright green.

Display the **Draw** menu again and select **Label Atoms** to open the pull-right menu. Select **Residue Name** from the pull-right menu. The **Atom Name** labels are replaced by a single label per residue.

Return once more to the **Draw** menu and select **Label Atoms.** Select **Off** in the pull-right menu to remove all labels.

2. Choose label information from palettes.

Display the **Draw** menu and select **Label Atoms** to open the pull-right menu. Select **Selection Tools** from the pull-right menu to open the Label Atoms and Label Atoms Components and Utilities palettes.

Select **Atom Name, Segment Name**, and **Show Labels** from the Label Atoms Components and Utilities palette.

From the Label Atoms palette, select **Include** and **First Atom of Residues.** A dialog box asks you to choose the interval between labels on residues. Click **OK** to accept the default value. Labels in the format *atomname_segmentname* are displayed next to the first atom in each residue.

Select **List Selection-Commands** from the Label Components and Utilities palette.

The textport reads:

```
   The current label selection is:
DATA PEPTIDE.msf
FIRST =ATNAM SEGNAM
```

3. Changing the label format.

From the Label Components and Utilities palette, select **Residue ID, Residue Name**, and **Show Labels.**

From the Label Atoms palette, select **Include** and **Alpha-Carbon Atoms.** In addition to the labels on the first atom in each residue, new labels in the format *residueID_residuename* are displayed next to each alpha-carbon atom.

Select **List Section-Commands.**

The textport reads:

```
The current label selection is:
```

```
   DATA PEPTIDE.msf
FIRST =ATNAM SEGNAM
PICK CA = RESID RESNAM
```

From the Label Components and Utilities palette deselect the tool **Show Labels.**

The selection is no longer checked and highlighted, and labels are no longer displayed. The textport message remains unchanged.

From the Label Atoms palette select **Clear.** All labels are removed from the screen. Select **List Selections Commands.**

The textport reads:

```
   The current label selection is:
DATA PEPTIDE.msf
```

4. Exit from the Label function.

Select **Quit** from the Label Atoms palette. The system returns to Molecular Modeling mode without saving or displaying the labels selected from the palettes.

5. Reselect labels from the pull-right menu.

From the **Draw** menu, select **Label Atoms** and open the pull-right menu. Select **On**. The residue labels from the first part of this exercise are redisplayed.

Selecting active atoms

QUANTA uses the list of atoms contained in an MSF for all modeling operations and calculations. All the atoms in an MSF or any portion of an MSF can be selected for any particular operation or calculation and then saved to a new .msf file. Atoms that are not selected are not saved in a newly created file and are not available for computations.

In general, this facility should be used sparingly, only when it is necessary to restrict the number of atoms in memory. Use **Display** selections, not **Active Atom** selections for determining what is viewed on the screen.

The **Active Atoms** function provides several mechanisms, both menu- and command-driven, for specifying which atoms, groups of atoms, segments, and/or residues in active MSFs are to be used for current QUANTA modeling operations and calculations.

The **Active Atoms** function is located in the **Edit** menu. When you select **Active Atoms**, a pull-right menu is displayed that contains two common predefined choices for atom display: **All Atoms** and **All Except Solvent**. To select other active atom clusters, choose **Selection Tools**. The Active Atoms and Active Atoms Utilities palettes open on the right of the screen.

Table 25 lists the selections in these palettes and provides brief functional descriptions of each selection.

Table 25. Active Atoms palette

Selection	Description
All MSFs	Selects all MSFs that have been chosen on the Molecule Management table if each MSF has <500 atoms. This option is turned off if one MSF has >500 atoms.
One MSF...	Selects all the atoms contained in the specified MSF and marks only those atoms. Only the selected MSF is displayed.
Apply to All MSFs	Applies specified selections of one MSF to all active MSFs.
Include	Includes all selections in the display. Default selection.
Exclude	Excludes the display of all selections.
All Atoms	Selects all atoms in the structure. Can be used to restore structure after using other atom selection options.
All Non-Hydrogen Atoms	Selects all atoms with the exception of hydrogen atoms.
Solvent	Selects all residues of type WAR, HOH, TIP, TIP3, SOL, SOLV, ST2, NON, H2O, TIP4.
Alpha-Carbon	Selects only protein alpha-carbon atoms.
Protein Backbone	Selects only protein backbone atoms.
Nucleic Acid Backbone	Selects only nucleic acid backbone atoms.

Table 25. Active Atoms palette (Continued)

Selection	Description
Residue Range...	Active residue range selected by completing information in a dialog box. The process can be repeated.
Residue Type...	Active residue type selected by completing information in a dialog box. The process can be repeated.
Atom Type...	Active atom type selected by completing information in a dialog box. The process can be repeated.
Pick Molecule	Using the mouse, selects a molecule from the active MSFs on the screen.
Pick Segment	Using the mouse, selects a segment by picking an atom in the segment from the active MSFs on the screen.
Pick Residue	Using the mouse, selects a residue by picking an atom in the residue from the active MSFs on the screen.
Pick Residue Range	Using the mouse, selection of a residue range is made by picking two atoms from the active MSFs on the screen.
Pick Atom	Using the mouse, selects an atom by picking it from the active MSFs on the screen.
Proximity Tools	Opens the Proximity Tools palette containing multiple choices for selecting a central atom and nearby portions of a molecule.
Type in a Selection	A dialog box is used to enter selection commands.
Undo	Returns the viewing area to the state before the previous command.
Clear	Deletes all selections. Resets the defaults.
Revert	Returns the viewing area to the initial state when **Active Atoms** was selected.
Quit	Exits Active Atoms without saving changes and reverts to the original .asf file.
Finish	Exits Active Atoms and writes the changes to a temporary .asf file. At least one atom of the active molecule must be selected to use this option.

Specifying and saving active atom selections

When you use the Active Atoms and Active Atoms Utilities palettes, you can make two types of atom selections: categorical and user-defined. Categorical (or automatic) atom selections specify atoms by category. User-defined (or interactive) atom selections establish specifications based on picked atoms in the display structure. You can also make a selection that allows you to enter selection commands from the keyboard.

When you have completed your selection of active atoms, *unselected atoms are no longer used for any calculations.* However, these atoms still exist in the MSF that you have been using, so it is possible to recover unselected atoms by selecting **Include** and **All Atoms** from the Active Atoms palette.

If you want to permanently modify the MSF, save it in a new file by using the **Save As** selection in the **File** menu. Give the new file a different name, since only selected atoms are included in the new MSF. All other atoms are deleted permanently and cannot be used again.

Use the following practice exercises to become familiar with the Active Atoms function and with the selection-command mode of making selections.

Selection commands can be entered from the command line, from the Active Atoms palette using **Type in a Selection**, or from the Active Atoms Utilities palette using **Edit Selection Manually**, which opens a shell window for editing. See Using Selection Commands on page 97 for more information about specific commands.

1. Make active atom selections using pull-right menu and palette choices.

Display the **Edit** menu. Select **Active Atoms** and open the pull-right menu. Select **Selection Tools.**

The Active Atoms and Active Atoms Utilities palettes open. The peptide molecule changes to blue indicating that no atoms are selected. As indicated in the message in the upper-right corner of the viewing area, selected atoms are displayed in red. Red and blue are the default colors for display colors 3 and 2, respectively.

Select **Residue Range** from the Active Atoms palette. When the dialog box appears, fill in the data entry boxes with the residue range 1 to 15 and click **OK.** A portion of the molecule turns red, indicating that these atoms are selected.

The textport message indicates that 155 atoms are selected. However, the **Active** column of the Molecule Management table indicates that the molecule still contains 201 active atoms.

Select **Finish** from the Active Atoms palette.

The palettes close and the molecule display changes to standard element colors. Only residues 1 through 15 are displayed. The textport message

indicates that 155 atoms are selected, and the **Active** column in the Molecule Management table indicates that those 155 atoms are the ones that are active in the molecule. QUANTA no longer recognizes the additional atoms that were part of the original peptide.

Display the **Edit** menu and reselect **Active Atoms** to open the pull-right menu. Select **All Atoms.** All 201 atoms of the original peptide are restored to active status and displayed in the viewing area.

2. Make active atom selections using selection commands.

Display the **Edit** menu. Choose **Active Atoms** then **Selection Tools** from the pull-right menu. The Active Atoms and Active Atom Selection Utilities palettes appear to the right with their default settings.

Select **Type in a Selection** from the Active Atoms palette. Use the dialog box to enter commands sequentially, using the tab key or the mouse to move from line to line.

Type the following commands:

```
> incl and all
```

These commands select all atoms as active.

Type:

```
> excl Zone 1 to 4
```

This selects the atoms (in residues 1 through 4) to be excluded from calculations

Click the **DONE** button.

Your input is accepted and the first four residues are now displayed in color 2 (blue, unselected), with the rest of the molecule displayed in color 3 (red, selected).

This message is displayed in the textport as the commands are entered:

```
>  164 atoms will be selected as active.
```


Select **Finish** from the **Active Atoms** palette. The molecular display returns to standard mode, and the modified structure becomes a new .asf file that will be incorporated into the .cst file to be used for calculations.

3. Save the modified structure in a new MSF

Display the **File** menu and select **Save As**. A dialog box containing options for saving a structure is displayed.

Select the options:

Write out current connectivity

Write out current bondtypes

Apply rotational transformations

Use the new MSF now

Update extra information

The File Librarian is displayed, prompting for the new file name. Enter the name newpeptide.msf and select the **SAVE** button.

The number of atoms contained in the structure remains at 164. The following message is displayed in the textport:

```
164 atoms selected for molecular structure newpeptide.msf
164 atoms will be displayed
```

Since the changed MSF is saved in a new file, the content of the original MSF, PEPTIDE.msf, is not changed.

Using selection commands

QUANTA generates selection commands every time you change any aspect of the way a molecular structure is displayed or used in calculations. You can enter selection commands directly from the keyboard as an alternative to making selections from menus or palettes or in combination with such selections. When you become familiar with selection commands, keyboard entry can provide a quicker, more accurate method for making complicated selections.

Atom specifications used in selection-commands take the basic form: *atomname segmentname: resid filename.msf*.

Selection-commands can be combined within a command line using the logical operators .and., .or., .not., and .xor. For example:

```
>  type try phe .and. round cb 38
```

selects the atoms contained in any aromatic residue around the beta carbon in residue number 38.

Wildcards can also be used with appropriate selection-commands, including type, pick, atom, and at.Table 26 lists the wildcards that are available and describes the way each operates.

Table 26. Selection specification wildcards

Wildcard	Description
*	Matches any string.
%	Matches a single character.
#	Matches any number.
+	Matches any digit.
?	Matches any single character including a trailing space.

For example, to pick all protons attached to the beta carbon in a protein, type the selection-command:

```
>  Pick HB+
```

To match all name carbons with three characters in their names, type the selection-command:

```
>  Pick C%%
```

The expressions:

- **GT** (greater than)
- **GE** (greater than or equal to)
- **LT** (less than)
- **LE** (less than or equal to)
- **EQ** (equal to)
- **NE** (not equal to)

can be used with the commands x/y/zcor, fourth, and charge to specify ranges of values. For example:

```
>  four gt 10
```

selects all atoms for which the currently loaded fourth parameter value is greater than 10.

Table 27 lists commands that can be used interactively with the **Display Atoms, Color Atoms, Label Atoms,** and **Active Atoms** menu selections. These commands can be entered from the command line, from a dialog box that opens when you choose **Type in a Selection**, or from a shell window that opens after you select **Edit Selection Manually** in the appropriate palettes. The commands are listed alphabetically.

Table 27. Selection commands

Selection	Command form	Description
All	all	Selects all atoms.
Atom	atom *atnam resnm resid*	Selects atoms matching specified atom name (*atnam*), residue name (*resnm*), and residue number (*resid*). Any specification can be replaced with an asterisk (*), acting as a wildcard.
Atom Type	atype *atnum*	Selects atoms matching specified atom type number.
	atype *atype*	Selects atoms matching specified atom type name.
Backbone	back	Selects all protein backbone atoms.
Charge	charge *ex value*	Selects atoms matching the expression of the specified atomic charge value.

Table 27. Selection commands (Continued)

Selection	Command form	Description
Centroid	cent all	Select the atom that lies nearest to the geometric center of all atoms.
	cent mole	Select the atom that lies nearest to the geometric center of each molecule.
	cent resi	Select the atom that lies nearest to the geometric center of each residue.
Conn	conn all	Selects all atoms in the same fragments as the specified atom. Cannot be used in active atom selection.
	conn short	Selects the atoms in the shortest end fragment from the specified atom Cannot be used in active atom selection.
	conn *n*	Selects the atoms within *n* bonds of the specified atom. Cannot be used in active atom selection.
Coordinate	xcor *ex value*	Selects atoms matching the expression of the specified value for the x coordinate.
	ycor *ex value*	Selects atoms matching the expression of the specified value for the y coordinate.
	zcor *ex value*	Selects atoms matching the expression of the specified value for the z coordinate.
Cylinder	cyli {byres} *radius atom atspec1 atom atspec2* cyli {byres} *radius x1y1z1 x2y2z2*	Selects a cylinder of specified radius with an axis between two specified atoms or coordinates. Using byres, a whole residue is selected if any atom within is selected.
Data	data *filename.msf*	Delimits atom selections for different MSFs.
Exclude	excl	Excludes all the objects following this statement up to the next INCL statement.
First	first	Selects the first atom in a residue. Can take an integer to select the first atom in every n^{th} residue. Except for Active atom selections, will select the first displayed atom.
Fourth	four *ex value*	Selects atoms matching the expression of the specified fourth parameter value. The Load command (load *fourth*, where *fourth* = parameter value), must be used before the Fourth command.
Include	incl	Includes all the objects following this statement up to the next EXCL statement.
Load	load = *label*	Load the data in the MSF extra information field *label* as the Fourth parameter.
Neighbor	nayb *dis resid*	Selects atoms that fall within the distance of the specified residue.
	nayb byres *dis resid*	Selects all atoms of the residue if any of its atoms fall within the specified distance of the specified residue.
Non-hydrogen	nonh	Selects all nonhydrogen atoms.

Table 27. Selection commands (Continued)

Selection	Command form	Description
Number	natom *n*	Selects atom number *n* in MSF.
	nres *n*	Selects residue number *n* in MSF.
	nseg *n*	Selects segment number *n* in MSF.
Pick	pick *atnam1 atnam2*	Selects atoms matching specified atom names.
Protein	prot	Selects all standard amino acids.
Proximity	prox *dis segnm*	Selects atoms that fall within the distance of the specified segment.
	prox byres *dis segnm*	Selects all atoms of the segment if any of its atoms fall within the specified distance of the specified residue.
Residue Type	type *resnm1 resnm2*	Selects specified residues by name.
Ring	ring *atspec {hydr}*	Selects any atom in the same ring or rings as the specified atom. The modifier hydr will include any hydrogens attached to the ring.
Round	roun *dis atspec*	Selects the entire residue if any of its atoms fall within the distance of the atom.
	roun *dis x y z*	Selects the entire residue if any of its atoms fall within the distance of the specified coordinates.
Segment	segm *segnm*	Selects a segment by name.
Slab	slab{byres} *atom at spec* slab {byres} *x-width, y-high z-thick center*	Selects a slab of the given dimensions centered on a point specified by an atom specification or a coordinate.
Solvent	solv	Selects all atoms in solvent residues.
Sphere	sphe *dis atspec*	Selects atoms that fall within the distance of the atom.
	sphe *dis x y z*	Selects atoms that fall within the distance of the specified coordinates.
Zone	zone *n1 to n2*	Selects a range of residues by identifier.
	zone *n1 n2 n3 n*	Selects individual residues by identifier.

Customizing defaults

You can customize the default display and coloring regime to be used when an MSF is first created or opened. This can be done by creating files named default.dsd and default.csd, respectively, containing the desired selection commands. For display selections, this can be done using the function **Default Display Mask** on the **Preferences** menu. The definition of solvent can be overridden by a file, solvent.asd, containing a suitable selection command. The definitions of what constitute protein or nucleic acid main and sidechains can be redefined by creating a set of files named backbone_prot.asd, backbone_nucl.asd, sidechain_prot.asd, and sidechain_nucl.asd.

These should contain just atom names, with wildcards if desired, since they will be assumed to be PICK selections.

The above files are all optional. They are looked for first in the current directory, and then in a directory pointed to by the environment variable $QNT_USR if you have set this up. If the file does not exist then the normal defaults are used.

Summary

In QUANTA, you can customize many aspects of MSF display as well as select and modify the atoms involved in modeling operations and calculations. This chapter deals with both types of manipulation.

Operations for customizing displays are carried out by the **Display Atoms, Color Atoms,** and **Label Atoms** functions in the **Draw** menu using pull-right menus and palette selections. Keyboard commands can be used interactively with these functions. Customization of atom display, colors, and labels is superficial, never affecting basic structural information used in modeling operations and calculations. The power of the display manipulations lies in their ability to focus attention on certain aspects or areas of a single molecule or several molecules.

The **Active Atoms** function in the **Edit** menu affects the atoms of an MSF used in modeling operations and calculations. All or any portion of the atoms in an MSF can be selected for any particular operation or calculation and then saved to a new .msf file. Atoms that are not selected are not saved in the newly created file.

Although the selection mechanisms for the **Display** function are similar to those of the **Active Atoms** function and MSF displays may look the same, the **Active Atoms** function is fundamentally different from the display function. Don’t confuse the two functions. Using them interchangeably may lead to a loss of data.

3. QUANTA Scripting Language

Contents

Introduction

QUANTA is designed such that every mouse click corresponds to a typed (“keyboard”) command. The QUANTA “record” mode saves all commands, whether from the keyboard or the mouse, into a single file. This consistency lets you record a movie-like script, illustrating and highlighting key points of a structure or interaction for a later presentation, with a minimum of effort.

The commands can be removed later if rotations and translations are not required for end use. The Record functionality may also be used for precisely recording a sequence of manipulations for an archive, or to generate commands to be used as the core of a more complex script (a “macro”).

To generalize a script and to increase its power, you may use variable substitution and add flow-control commands. QUANTA supports a subset of the Tool Command Language (TCL) for this purpose. See Introduction to TCL Scripting Control on page 115. The addition of TCL commands and QUANTA command modifications must be done with a text editor.

Command Logging

Creating a QUANTA log file

Creating a QUANTA log file (record script) is straightforward.

1. To begin recording commands, select **File/Record Session/Start** from the QUANTA interface menu.
2. In the window, enter the file name for the script (“script.rec” for example), or type “record start script.rec”.
3. When you wish to stop, pause or resume recording, select the menu pulldown at **File/Record Session** or type in “record <keyword>” where <keyword> is “stop”, “pause” or “resume.”

> **Note:** Scripts allow exact duplication of a QUANTA run, including translations and rotations initiated by the mouse. Unedited scripts can quickly become very large. See Script Modification on page 110 for information on trimming the size of log files.

Playing back a script

Depending on the script, you may need to reset QUANTA by closing files. QUANTA record scripts can be started in two ways:

1. Use the interface menu (or its keyboard equivalent); for example **File/Replay Session/Start** >.
2. Type:

   ```
   > @<script name>
   ```

   Assuming a script name of *script.rec*, correct commands would be: `@script.rec` or `@script`.

Script Modification

Script modification is necessary when you want something other than an exact logging of a QUANTA session. QUANTA provides no tools for adding TCL commands or editing record files. Scripts are ASCII files; they can be modified using any text editor. Common modifications are script simplification (removing unnecessary commands), changes of the target of the

scripts, or introduction of flow (looping and branch logic). Flow control is discussed in Introduction to TCL Scripting Control on page 113.

Script simplification

If a given script is not meant for visual illustration, the first step to simplify it is generally to remove some or all mouse-introduced translation and rotation and other related commands from the file.

QUANTA logs mouse-based rotation, translation, zoom, and z-clipping adjustment commands as relative motion commands. All begin with %MOUSE. Similarly, commands issued using the pseudo-dials box in QUANTA are logged as %TRANS.

QUANTA does record some movement-related commands. A record file can be significantly reduced in size by removing “redundant” %mouse commands. The following awk script does this:

```
awk 'BEGIN {old = " "}; \
   $1 == "%MOUSE" || $1 == "%TRANS" \
     {if ($0 != old){print $0; old=$0} }; \
   $1 != "%MOUSE" && $1 != "%TRANS" \
     {print $0; old=" "}' \
  input.rec > output.rec
```

If a script requires no structure movement, then all movement commands can be removed.

In UNIX a simple way to remove all %MOUSE commands is to filter them out using the “grep” command:

```
>  grep -v '%MOUSE' script.rec > short_script.rec
```

Table 28 displays a summary of the %MOUSE command parameters.

Table 28. MOUSE Command Parameters

%MOUSE *nn xxxx yyyy mp*

nn	**Function**
2	xy rotation
4	z rotation
16	scale
18	xy translation
20	z translation

Table 28. MOUSE Command Parameters

%MOUSE *nn xxxx yyyy mp*	
nn	**Function**
33	z clip width

Note the number (*nn*) is derived from a binary code:

- 1 = left mouse button
- 2 = middle mouse button
- 4 = right mouse button
- 8 = <Ctrl> key
- 16 = <Shift> key
- 32 = <Alt> key

xxxx and *yyyy* are the xy position of the mouse on the screen (screen size dependent).

mp = 1 when the mouse button is depressed. It signals the start position. *mp* = 0 for subsequent contiguous mouse commands.

Adding new commands to a script with a text editor

While creating a script starting with a record file, you may find that some commands that are absent from the initial script are needed, or that some commands should be changed. You could have QUANTA generate a new record file and generate a new script. Another option, which is perhaps the best way to learn the QUANTA syntax, is to use the QUANTA “command echo” option. When you use this feature, QUANTA reports in the text window the syntax of each command as it is executed.

To turn on command echoing, type:

```
> set echo on
```

For some commands, you may prefer to use keyboard commands that are never generated by the interface. For example, you can replace mouse movement commands with explicit movement commands such as ROT x 90 and TRAN y –10. For a summary of useful keyboard commands, see the Appendix on page 115.

Introduction to TCL Scripting Control

The real power of QUANTA scripts comes from TCL additions to QUANTA commands. These additions include variables, branching, lists, and subroutines.

All TCL commands in QUANTA must be preceded with the “!” character. All comments must contain an “*” in the first non-blank column.

> **Note:** Keep in mind that QUANTA implements only a subset of TCL.

Please refer to Tool Command Language (TCL) (page 189), for more information about TCL.

Variable definitions and usage.

```
!set protein 1crn.msf
!set str_len 3
!set aaname “”
```

Variables must be preceded by the "$" character when used (as opposed to being defined) in an expression:

```
clip $width
mole open $protein, behavior replace ok
```

TCL commands that return values are placed in square brackets:

```
!set str_res [!expr $colon_index+1]
!set resid [!string range $string $str_res $str_len]
```

Branching and looping

> **Note:** Curly brackets create a statement block. Many TCL commands operate on lists. Lists will contain the characters “_list” in the variable's name for clarity. See Lists creation and modification on page 114.

FOREACH command:

```
!set count 0
!foreach value $value_list {
         . . .
            !incr count
        }
```

FOR command:

```
!for {!set count 0} {$count < $max} {!incr count} {
        . . .
      }
```

IF/ELSE command:

```
!if {$value < 0} then {
    . . .
 } else {
    . . .
}
```

WHILE command:

```
!while {$start < [!expr $len-1]} {
   . . .
 }
```

Below is a full TCL script that rotates a pre-saved scene (save the scene using **File/Save Status**):

```
# This script a requires a QUANTA status file called "scene1"
#
#  Reinstates the molecule, scene and settings
stat rest scene1
# Begin controlled transformations
!for {!set frame 1} {$frame<=4} {!incr frame} {
  scale 1.08
}
!for {!set frame 1} {$frame<=90} {!incr frame} {
  rota y 1.0
}
!for {!set frame 1} {$frame<=60} {!incr frame} {
  rota x 0.666667
}
```

Lists creation and modification

`!list $val1 $val2 $val3`	#returns a list
`!glob -nocomplain  *.msf`	#returns list of file names matching filter
`!lappend shrt_list $item`	#extend list short_list

List extraction

`!lsearch $abig_list $str`	#returns position in list of $str
`!lindex  $my_list $index`	#returns the specified entry in the list (from 0)

Operations

!expr $index+($val/2.0)	returns value+1
!incr $variable<optional inc. value>	increments variable

!string range $string $f $l	returns substring
!string tolower $str	converts to lowercase
!scan $string {%I %I} $val1 $val2	

Input / Output

!echo "there are $numfiles files loaded"	
!set $fileid [!open myfile.txt <r|w|a>]	

** note "fileid"contains a code returned by the open command that must be used in puts and read expressions.

!puts $fileid "There are [!llength $result] pairs."	#write a line to file
!set string [!gets $fileid]	#return next line from file to $string

Appendix

Commonly used QUANTA keyboard commands:

> **Note:** QUANTA command keywords are not case sensitive. Uppercase signifies that it is required, but more characters are allowed.

SET ECHO ON	Turn on textport command echo
SET ECHO OFF	Turn off textport command echo
SCALE <scale>	Zoom in or out
ROTate <x|y|z> <angstrom value>	Rotate view angle
TRANslate <x|y|z> <angstrom value>	Translate view

`CLIP <angstrom value>`	set clipping plane width
`MAP TAB SHOW`	Show the map table
`MAP TAB HIDE`	Hide the map table
`MAP ON`	Display active map
`MAP OFF`	Hide active map

TCL documentation

Please refer to Tool Command Language (TCL) (page 189), for detailed information about TCL.

Sample TCL/rec scripts

See QUANTA Script Examples (page 236).

Binding commands to function keys

QUANTA commands can be bound to function keys on your workstation keyboard. Function keys F3 through F12 may be redefined. Either a single command or a pair of commands can be bound. In the former case, every time the function key is pressed, the command you have associated with the key is sent to the QUANTA command interpreter. In the latter case, the commands are sent alternately to allow toggling.

By default, function key bindings are defined by the file *$HYD LIB/function key.dat*. To view the default settings, open the **Information** menu in the menu bar. Select **Current Session** to open a pull-right menu and select **Function Keys**. The list of function key bindings is listed in the Textport.

You may print the PostScript file *$HYD_LIB*/funkey.ps to create a strip showing the default bindings to place above the function keys on your keyboard.

Complete the following exercise to familiarize yourself with the procedure for assigning customized key bindings.

1. Define function key F5.

Open the **Preferences** menu in the **menu bar.** Select **Device Settings** to open the pull-right menu. Select **Function Keys** from the menu. A dialog box is displayed with a default number **3** in the data entry field.

Enter **5** in the field to select function key 5 as the key to be defined.

Click **OK.**

Another dialog box is displayed asking:

How should FK5 behave?

Select **The Key acts the same each hit**.
Click **OK.**

A third dialog box is displayed, requesting:

Enter the command this key will send.

In the **data entry** field, type:

```
> List atom all in the data entry field.
```

Click **OK.**

You have defined key F5.

2. Define function key F6.

When you complete the last step in defining F5, the dialog box for specifying the function key number to be defined is redisplayed with the number 5 as the default.

Enter **6** in the **data entry** field to select the function key to be defined.

Click **OK.**

The next dialog box is displayed, asking:

How should FK6 behave?

Select **The Key acts the same each hit**.

Click **OK.**

A third dialog box is displayed, requesting:

Enter the command this key will send.

In the **data entry** field, type:

```
>  %Clear ID
```

The % denotes that the command is a palette selection.

Click **OK**.

You have defined key F6.

3. Define function F7.

The dialog box for specifying the function key number to be defined is displayed, this time with the number **6** as the default.

Enter **7** in the **data entry** field to select the function key to be defined.

Click **OK.**

The next dialog box is displayed, asking:

How should FK7 behave?

Select **The Key acts as a toggle**.

Click **OK.**

A third dialog box is displayed, requesting:

Enter the command this key will send.

In the **data entry** field, type:

```
> VIEW STEREO
```

Click **Return**.

Another data entry dialog box is displayed.

Enter the alternate command:

```
> VIEW MONO
```

Click **OK.**

4. Save your changes to a local directory.

Finish the definition process.

Another dialog box is displayed.

Click **Cancel**.

A final dialog box is displayed with choices for saving the changes.

Click **OK** to accept the default: **Save to Local Directory**.

A. Main Menus

The following tables list all selections in menus accessed from the QUANTA main menu bar. Each table summarizes the menu for one of the nine selections on the main menu bar. A -> following a menu selection indicates that the selection provides access to a pull-right menu. Pull-right menu selections are listed in the second column of each table.

Table 29. File menu functions and commands

Selection	Pull-right selection	Generated commands	Description
Open		mole open	Opens MSF files list for selection.
Close		mole close	Closes current MSF.
Save As		msf create	Saves current MSF.
Revert		msf undo	Reverts to previous MSF.
Copy MSF		msf clone	Makes a copy of current MSF.
Import		data import	Brings external files into QUANTA.
Import Multiple PDBs		data pdbs	Imports more than one PDB file at a time.
Export		data export	Sends QUANTA files to external applications.
Sequences ->	Read Sequence/Alignment File	dseq read	Options for manipulating sequence data. Please see the *Protein User's Reference Guide* for further details.
	Read Sequence Data File	dseq data	
	Write Alignment File	dseq write	
	Plot Sequence Viewer	dseq plot	
	Remove Sequence(s)	dseq remove	
Save Status		status archive	Saves MSFs and environment (*filename*.cst) to .tar file. Compresses file to save disk space.
Restore Status		status dearchive	Restores MSFs and previous display conditions.
QUANTA Aliases ->	Define	alias defi	Allows user-defined single command alias to replace a series of complex commands.
	List	alias list	
	Delete	alias dele	
Record Session ->	Start	record on	Records all actions during a QUANTA session. **Record Session** must be off for replay.
	Stop	record off	
	Suspend	record suspend	
	Resume	record resume	

Table 29. File menu functions and commands (Continued)

Selection	Pull-right selection	Generated commands	Description
Replay Session ->	Start	replay on	Replays recorded QUANTA session. **Record Session** must be off for replay.
	Stop	replay off	
	Suspend	replay suspend	
	Resume	replay resume	
Log Output ->	Start	log on	Copies all information in the Textport to a log file.
	Stop	log off	
	Suspend	log suspend	
	Resume	log resume	
Open System Window		system	Opens system window from within QUANTA.
Plot Molecules ->	Generate	plot	Generates, previews, and outputs to a file various styles of plot.
	Preview	preview	
	Plot Solid	plot solid	
X,Y Graphs		xypl	Starts the graph application in a new window.
Restart QUANTA		restart	Closes open MSFs. Copies default *.cst* file and returns QUANTA to startup conditions.
Exit QUANTA		end	Saves MSFs as necessary and shuts down QUANTA.

Table 30. Edit menu functions and commands

Selection	Pull-right selections	Generated commands	Description
Molecular Editor		3ded	Launches application for building and editing 3D molecules.
Active Atoms ->	All Atoms	sele chan all all solv	Specifies which atoms in an MSF are to be used for modeling operations and calculations.
	Remove Solvent	sel chan all all noso	
	Selection Tools	select palette	
Select Sets		setd	Changes the current set selection.
Define Dummy Atoms		dummy	Creates and manages dummy atoms to be used in calculations and analysis.
Atom Property Editor		msf table	Edits MSF to change atom names, types, and charges, segment and residue names, and fourth parameter data.

Table 30. Edit menu functions and commands (Continued)

Selection	Pull-right selections	Generated commands	Description
Split and Clean		clean	Allows imported structures to be split according to their molecular type and then have hydrogens added and atom types assigned using appropriate methods.
Apply Dictionary		retype	Reassigns types and charges from a dictionary in QUANTA.
Symmetry ->	Brief Listing	data symmetry card	Defines and manages symmetry. Provides brief and full symmetry information.
	Full Listing	data symmetry print	
	Define	data symmetry write	
	Apply	syms	
Crystal Disorder ->	List	diso list	Changes the current crystal disorder selections with select, delete, exclude, color, label, deassign, and list options.
	Delete	diso dele	
	Excluded	diso excl	
	Color	diso colo	
	Select	diso sele	
	Label	diso labe	
	Deassign	diso deas	
Incorporate Data ->	Fourth Parameter	data incorporate four	Reads external data into MSF.
	Vectors	data incorporate vect	
	Charges	data incorporate char	
	Show	data incorporate show	
	Delete	data incorporate dele	
Incorporate Coordinates		msf update	Reads external coordinates into MSF.
Fourth Parameter ->	Edit BValue Occupancy	edit	Selects and edits fourth parameter values including BValue and Occupancy displays.
	Select Interpretation	set fourth	
Atom Data ->	Charge	set charge	Edits MSF for atom charge, type, coordinates, BValue, or other atom parameters.
	Parameters	set param	
	Type	set type	
	Coordinates, BValue	edit atom	
Bond Options		bond algo	Selects bond algorithm and bond display mode for open MSFs.

Table 31. Draw menu functions and commands

Selection	Pull-right selections	Generated commands	Description
Display Atoms ->	All Atoms	disp chan all all	Defines and manages display options for atoms in open MSFs.
	All Except Solvent	disp chan all noso	
	Non-Hydrogen Atoms	disp chan all nonh	
	Protein Backbone	disp chan all back	
	C-Alpha Trace	disp chan all ca	
	CA Trace+NonProtein	disp chan all capl	
	Selection Tools	display palette	
Color Atoms ->	By Element	colo chan all atom	Defines and manages color options for atoms in open MSFs.
	By Segment	colo change all segm colo diff	
	By Molecule	color palette	
	Selections Tools		
Property Colors ->	Define Properties	prop col	Options for coloring structures according to various properties such as thermal parameters or residue position.
	Normalization	prop norm	
	Spectral Colors	set col spec	
	Electrostatic Colors	set col elec	
	Toggle Legend	prop lege	
	Toggle Blended Bonds	prop blen	
Label Atoms ->	Off	label off	Defines and manages label information for atoms within open MSFs.
	Atom Names	labe init sel: all end with atnam	
	Atom Type	all end with type	
	Atomic Charge	all end with char	
	Residue IDs	cent resi end with seg resi	
	Residue Name	cent resi end with resn	
	On	label on	
	Selection Tools	label palette	
Bond Style ->	Half-Bond Vectors	bond half	Selects bonding display style.
	Full-Bond Vectors	bond full	
	Color by Bond Type	bond btyp	
	Single Vector Bonds	bond sing	
	Multi-Vector Bonds	bond mult	
Line Style ->	Antialiasing On	anti on	Selects line style including anti-aliasing option for bonds.
	Antialiasing Off	anti off	
	Settings	set style	
Manage Display		show	Toggles display lists off and on.
Create Objects		obje create	Creates and manages graphical objects from list of available options.

Table 31. Draw menu functions and commands (Continued)

Selection	Pull-right selections	Generated commands	Description
Object Parameters ->	Mesh Grid Size	set grid	Adjusts graphical object parameters such as dimensions and colors for mesh grids, ribbons, and brick maps.
	Ribbon Parameters	set ribbon	
	Brick Map Radius	set mapr	
Ruler		obje ruler	Displays a ruler in the viewing area.
Protractor		obje protractor	Displays a protractor in the viewing area.
Solid Models ->	Selection Tools	solid pal	Creates and displays a solid model of a molecular structure as a graphical object or in an external file, depending on system capability.
	Protein Cartoon	solid chan all secstr	
	van der Waals	solid chan all VDW	
	Ball and Stick	solid chan all ball	
	Liquorice	solid chan all liqu	
	Redraw Model	solid redo	
	Settings	solid para	
Ray Trace ->	Original Raytrace	rayt	Generates renderings of a displayed structure with multiple light sources, shadows, and reflections.
	Rayshade Solid	rays	
Raster Models ->	Z-buffered Spheres	raster zbuf	Generates a static space-filling representation of a displayed structure using one of a set of raster image choices.
	Flat Disk	raster disk	
	Outlined Disk	raster outl	
	Smooth Spheres	raster smoo	
	Metallic Spheres	raster meta	
	Interpenetrating Spheres	raster inte	
Dot Surfaces		surf dot	Calculates, displays, and deletes VDW dot surfaces. Not a graphical object.
Captions and Arrows ->	Place Arrow	arrow place	Adds and removes annotations to a molecular display in the viewing area.
	Move Arrow	arrow move	
	Delete Arrow	arrow remo	
	Delete All Arrows	arrow dele	
	Place Caption	caption place	
	Move Caption	caption move	
	Delete Caption	caption remo	
	Delete All Captions	caption dele	

Table 32. View menu functions and commands

Selection	Pull-right selections	Generated Commands	Description
Mono[a]		view mono	Displays monocular view of molecular structures. Default setting. Toggles with Stereo and Stamp.
Stereo ->a	Crystal Eyes	view sgraph	Presents stereo view of molecular structures that may be viewed with or without an external device. Toggles with Mono and Stamp. Stereo in a window requires some special setups. See the patches page at the Accelrys website.
	Internal Stereo	view istr	
	Side by Side	view stereo	
	Left Eye View	view left	
	Right Eye View	view right	
	Leeds Viewer	view leeds	
	Stereo in a Window	view swin	
Stamp	a	view stamp	Presents all open MSF displays in left-to-right sequence. Does not alter coordinates. Toggles with Mono and Stereo.
Rock ->	On	view rock	Rocks molecular structures to either side of preselected axis (X,Y,Z) to a predetermined angle limit. Toggles with Roll.
	Off	view static	
Roll ->	On	view roll	Rolls molecular structures 360 degrees around predetermined axis (X,Y,Z). Toggles with Rock.
	Off	view static	
Flash ->	Clock	view flash	Flashes display of molecular structure on and off at predetermined rate.
	Cycle	view flac	
	Off	view stab	
Settings	aasdfa	set vpar	Sets parameters for Rock, Roll, Stereo, and Flash.

Table 32. View menu functions and commands (Continued)

Selection	Pull-right selections	Generated Commands	Description
Manage View ->	Orient	orient	Fixes precise orientations along X, Y, or Z axis. Saves specific transformations and restores previously saved view.
	Save	view save	
	Read	view read	
	Change File	view file	
Show Windows ->	All	view pale on, view dial on, show man otab	Displays one or more standard QUANTA windows.
	Palettes	view pal on	
	Dials	view dial on	
	Molecule Table	show man	
	Object Table	show otab	
Hide Windows ->	All	view pale off, view dial off, hide man otab	Hides one or more standard QUANTA windows.
	Palettes	view pal off	
	Dials	view dial off	
	Molecule Table	hide man	
	Object Table	hide otab	
Window Layout ->	Save Current	wind save	Adjusts and saves layout of standard QUANTA windows.
	Full Screen	wind full	
	Default Screen	wind defa	
	Saved Screen	wind reve	
Windows ->	QUANTA	TYPE=WINLIST	Allows user to select and bring forward specific window. Window list is context- sensitive, varying with application or function in use.
	Dials		
	Geometry		
	Modeling		
	Molecule Management		

[a] Perspective can be added and removed to stereo viewing using the commands: view pers on/off and can be added to the mono view with the command: view perm and removed with the command: view mono The perspective parameters (eye separation and focal length) can be adjusted using the Dials Emulator Set B.

Table 33. Preference menu functions and commands

Selection	Pull-right selections	Generated commands	Description
Color Definitions ->	All Definitions	set color	Defines colors applied to atoms and menus.
	Blue to Red	set colo smoo	
	Red to Blue	set colo red	
	Black and White	set colo black	
	Reset Atom	set colo atom	
	Reset All	set colo reset	
	Menu Colors	set colo menu	
Bonding Algorithm		bond defa	Defines which bonding algorithm to apply by default to subsequently opened MSFs.
Default Atom Display		set disp	Defines display selection to be applied when MSFs are opened.
Suppress Objects ->	On	view suppress on	Suppresses object displays until any molecular movement is complete to allow faster manipulations.
	Off	view suppress off	
ID Mode and Style ->	One ID Only	set id last	Selects information included in ID and determines how IDs are displayed.
	Repick Flashes	set id flas	
	Repick Deletes	set id dele	
	ID Style	set idst	
Output Settings ->	Verbose Messages	set mess verb	Regulates how much information is supplied to the Textport. The default setting is Normal Messages.
	Normal Messages	set mess norm	
	Terse Messages	set mess terse	
	Textport Paging	set page	
	Echo Commands On	set echo on	
	Echo Commands Off	set echo off	
	Confirm Dialogs On	set conf on	
	Confirm Dialogs Off	set conf off	
Atom Symbol		set symb	Defines the display of single, non-bonded atoms.
PDB Import/Export		set rename	Options used when PDB files are imported/exported.

Table 33. Preference menu functions and commands (Continued)

Selection	Pull-right selections	Generated commands	Description
MSF Add/Replace ->	Prompt for Options Add to Current Use New MSF only		Controls behavior when a new MSF has been created.
Set MSF Saving Options		set save	Sets MSF saving preferences.
Device Settings ->	Dial Emulators Real Dials Button Box Function Keys Z-Buffer	set dial set knob set butt set func set zbuf	Defines settings for external devices or for keyboard function keys.
Sticky Menus ->	On Off	set sticky on set sticky off	Defines whether menus are allowed to be sticky or not.
All Settings		set	Presents all Preference options including those not listed in the Preference menu.

Table 34. Calculate menu functions and commands

Selection	Pull-right selections	Generated commands	Description
Select Host ->	Solvent Surfaces Interaction Maps CINDO MOPAC UHBD	surf solv host calc prob host exec cindo host exec mopac host uhbd host	Selects host for running any of the calculations listed in the pull-right menu. Allows calculations to be off-loaded to other CPUs.
Retrieve Results		exec retrieve	Examines and retrieves correct results files from a remote host. Automatically copies results files into current local working directory.
Solvent Surfaces ->	Calculate Use Input File Display List Dots	surf solv calc surf solv use surf solv disp surf solv list	Calculates and displays Connolly solvent surfaces.
Solid Surfaces ->	Calculate Calculate Options Re-Display Grid Map	surf soli calc surf soli options surf soli disp surf solid prop	Calculates and displays solid surfaces.
Interaction Maps ->	Probe Molecule Display Map	calc prob calc prom calc prob disp	Calculates and displays empirical energy maps using a probe or a molecule to sample the environment of the displayed structure.
Electrostatic ->	Potential Field	exec vscalc exec field	Calculates electrostatic potential or electrostatic field.
CINDO ->	Calculate Use Input File Read/Display Results	exec cindo calc exec cindo use exec cindo post	Sets up and manages CINDO calculations.
MOPAC ->	Calculate Use Input File Read/Display Results Restart Shutdown Cleanup Files	exec mopac calc exec mopac use exec mopac post exec mopac rest exec mopac shut exec mopac clean	Sets up and manages MOPAC calculations.
UHBD		uhbd	Sets up and manages UHBD calculations.
Hydrogen Bonds		hbond	Sets up and manages calculations and displays of hydrogen bonds.
Dipole Moment		calc dipole	Calculates and displays a dipole moment vector.

Table 34. Calculate menu functions and commands (Continued)

Selection	Pull-right selections	Generated commands	Description
Close Contacts		calc contact	Calculates and displays close contacts.
Geometry Monitors		moni	Options for calculating, displaying and listing geometry monitors.
Internal Coordinates - >	Bonds Angles Dihedrals	calc bond length interactive calc bond angle interactive calc bond dihed interactive	Calculates bond lengths, bond angles, and dihedral angles using selected atoms. Reports results to the Textport.
Vectors and Planes ->	Lines Planes Distances Angles Dihedrals	line plane calc geometry dist calc geometry angle calc geometry dihedral	Calculates vectors and planes between selected atoms.
Coordinate Statistics		calc stat	Calculates minimum, maximum, mean, and standard deviation of molecular coordinates.
Show Status		inqu status	Displays the status of background calculations started by QUANTA.

Table 35. CHARMm menu functions and commands

Selection	Pull-right selections	Generated commands	Description
Select CHARMm Host ->	Interactive	char mach	Presents options for running CHARMm calculations.
	Batch	exec batch host	
	NQS	exec nqs host	
Initialization Options		char set init	Defines topology and parameter files to be included in charmm.cis. Defines CHARMm output file.
CHARMm Process ->	Initialize Interactive	char init	Determines and manages how and where CHARMm runs.
	Terminate Interactive	char kill	
	Interactive Status	inqu stat char	
	Start Batch Job	exec batch use	
	Batch Status	exec batch stat	
	Start NQS Job	exec nqs use	
	NQS Status	exec NQS stat	
Energy Terms		char set ener	Specifies what energy terms are included in a CHARMm calculation.
Update Parameters		char set upda	Specifies parameters that generally control nonbond terms in CHARMm energy calculations.
Minimization Options		char set mini	Specifies method and parameters to apply in a minimization calculation.
Dynamics Options		char set dyna	Specifies parameters that control a dynamics calculation.
Constraints Options ->	Sel Atom Constraints	char cons atom edit	Specifies and manages atom, distance, dihedral, and stochastic boundary constraints during calculations.
	Atom Constraints On	char cons atom on	
	Atom Constraints Off	char cons atom off	
	Dihedral/Distance	charm cons dihe create	
	Dihedral On	charm cons dihe on	
	Dihedral Off	charm cons dihe off	
	Distance On	charm cons dist on	
	Distance Off	charm cons dist off	
	Stochastic Bdry Settings	charm cons sbound set	
	Stochastic Bdry On	charm cons sbound on	
	Stochastic Bdry Off	charm cons sbound off	
SHAKE Options		char set shak	Allows removal of very high frequency vibrations from consideration in dynamics simulations.
Parameters ->	Specify Files	char read	Reads, sets, and saves parameter options.
	Save Parameters	charmm para	
	Set Options	char set para	

Table 35. CHARMm menu functions and commands (Continued)

Selection	Pull-right selections	Generated commands	Description
CHARMm Mode ->	PSF	charmm mode psf	Sets PSF or RTF mode and sets specifications for each mode.
	RTF	charmm mode rtf	
	MMFF	charmm mode mmff	
	MMFFS	charmm mode mmffs	
	PSF Terms	char set psf	
	RTF Options	char set rtf	
Get Fourth Parameter ->	Forces	charmm get forces	Retrieves force and scalar data from CHARMm PSF to MSF.
	Scalars	charmm get scalars	
Optimize Hydrogens		char do hbuild	Performs a CHARMm calculation to optimize hydrogen bonds.
Solvate Structure ->	15 Å Radius Sphere	char do solv 15SP	Adds solvent within specified distances.
	15 Å Length Box	char do solv 15bo	
	30 Å Length Box	char do solv	
	Solvate Residues 8 Å	char do solv 8sol	
	Solvate Residues 10 Å	char do solv 10sol	
	15 Å Methanol Sphere	char do solv spmt	
	Water Image 15 Å	char do solv 15on	
	Water Image 30 Å	char do solv 30on	
	Turn Off Water Image	char do solv off	
Periodic Boundaries		char pbou	Defines a shape matrix for constant pressure dynamics.
Send CHARMm Command		char do comm	Sends CHARMm commands to CHARMm from QUANTA. Initializes QUANTA if it is not already running.
Stream CHARMm File		char do stream	Sends a command directly to CHARMm from QUANTA. Initializes CHARMm if it is not already running.
Capture Commands ->	On	char capt on	Captures CHARMm commands sent from QUANTA in a script file that can be reused.
	Off	char capt off	
	Suspend	char capt susp	
	Restart	char capt rest	
Settings ->	Show	char inqu	Shows current CHARMm setup. Saves or replaces a CHARMm setup file.
	Save	char file save	
	Restore	char file chan	

Table 36. Applications menu functions and commands

Selection	Pull-right selections	Generated commands	Description
Builders ->	2-D Sketcher	chemnote	Launches the 2D or 3D sketchers (ChemNote or the Molecular Editor), the Sequence Builder, or the Nucleic Acid Builder.
	3-D Builder	3ded	
	Sequence Builder	sequence	
	Nucleic Acid Builder	build	
	Solvent/Amorphous Builder	amgr	
Conformational Search		csea	Launches a conformational search.
Molecular Similarity		flexcomp	Launches the molecular similarity application to compare molecular structures.
Analysis		cana	Launches the analysis application.
Dynamics Animation		anim	Launches the dynamics animation application.
Protein Design		prot	Launches the protein design application.
Protein Modeler		pmod	Interface to the MODELER program.
Protein Health		health	Launches the protein health application. Works in conjunction with protein design.
Protein Profile Analysis		prof	Launches the protein profile analysis application. Works in conjunction with protein design.
X-Autofit		xfit	Enters the map skeletonization and chain tracing application.
X-Build		xfit	Enters the X-Ray model building application.
X-Solvate		xsol	Enters the solvent peak search and placement application.
X-Ligand		xlig	Enters the ligand search and placement application.
CNX		xplor	Enters the CNX refinement interface.
Crystal Modeling		xtal	Launches the Crystal Modeling application.

Table 37. Information menu functions and commands

Selection	Pull-right selections	Generated commands	Description
About QUANTA		about	Provides information about the QUANTA software.
Help...		help help	Starts QUANTA online help.

Table 37. Information menu functions and commands (Continued)

Selection	Pull-right selections	Generated commands	Description
List Molecule ->	Atoms	list atom	Lists what is active in open MSFs.
	Residues	list group	
	Bonds	list bond	
	Segments	list segment	
	Structures	list molecule	
	Extra Information	inquire extra	
	MSF Titles	inq titl	
Current Selections ->	MSF	mole list	Lists current selections in the textport.
	Atom	select list	
	Display	display list	
	Color	color list	
	Set	setd list	
	Label	label list	
QUANTA Parameters		list param	Lists atom default parameters in the Textport.
Current Session ->	Mouse Movement	inqu mouse	Provides information in the textport on preferences, selections, and operations in current session.
	Dynamics Parameters	inqu dyna	
	Drawing Modes	inqu mode	
	Calculations Constants	inq cons	
	External Jobs	inq stat	
	Function Keys Setup	inqu func	
	Button Box Setup	inq butt	
	Time and Date	inq time	
	User Name	inq user	
Command History		inquire history	Lists commands used in current session in textport.

B. External Devices for Silicon Graphics Machines

This appendix describes various external hardware devices for SGI that can be used with QUANTA for issuing commands and moving structures. These devices duplicate operations that are performed using the dial emulators, mouse, and function keys that are integral to QUANTA.

Each device can be used to perform operations such as transformations, clipping, and scaling of a displayed structure. Using any of these devices, you also can execute commands that correspond to menu or palette selections.

Compatible external devices include:

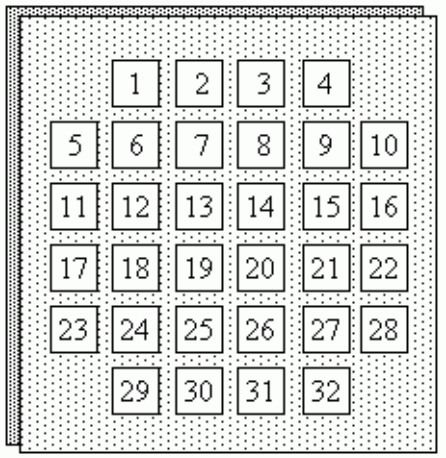
- Button box
- Dial box
- Graphics tablet
- Spaceball

Configuration information for these devices is included in the *QUANTA Installation Guide.*

Using a button box

A button box is used as an interface to specify palette selections and menu functions or issue QUANTA commands normally entered on the command line. The device is active only in the Molecular Modeling application. It has 32 function buttons that are numbered by row, from left to right.

Pressing a button sends a command to QUANTA. Depending on the command, a prompt may appear on the message line informing you of an appropriate action to take. The cursor must be in the modeling window in order for the buttons to work.



Default settings

By default, the buttons are assigned specific commands. However, these assignments can be overridden and the button box customized for your situation. Table 38 summarizes button functions.

Table 38. Button box default commands

Button	Function	Description
1	**Set Stereo**	Toggles the display between one and two images for stereo viewing.
2 - 3	**Not Used**	
4	**Reset View**	Resets global transformations applied to molecular structures.
5	**All Tools Off**	Resets the display, clearing any display of distances, neighbors, and bumps. Selected palette tools are turned off.
6	**Clear ID**	Removes the display of atom IDs and distances from the molecular structure. This does not affect labels generated by the Label Atoms function.
7	**Set Origin**	Prompts for an atom pick and places that atom at screen center. The atom becomes the center of rotation for subsequent operations.
8	**Center**	Calculates the geometric center of picked atoms and places it at screen center. The geometric center becomes the center of rotation for subsequent operations.
9	**Neighbors**	Calculates and displays the nearest neighbors to subsequently picked atoms.
10	**Geometry**	Toggles off and on the palette tools that calculate the distance between pairs of picked atoms, the angle between successive triplets of picked atoms, and the dihedral angle between successive quadruplets of picked atoms. Results are displayed in the textport.

Table 38. Button box default commands (Continued)

Button	Function	Description
11	**Move Fragment**	Prompts for an atom pick to specify a molecular fragment to transform.
12	**Move Set**	Prompts for an atom pick to specify a set to transform independently.
13	**Move Atom**	Prompts for an atom pick to specify an atom to translate in the x, y, and z directions.
14	**Torsions**	Displays the torsions palette and prompts for an atom selection to define torsions for bond rotations.
15	**Spin**	Automatically creates new conformations by rotating about active torsions. Searches for close contacts and stops when it finds a conformation with no close contacts or after searching every conformation.
16	**Cut Residue**	Prompts for a residue selection and creates a molecular fragment by breaking all existing bonds between the selected residue and the molecular structure.
17	**Continuous**	Sets and unsets interactive updates for Bump, R/Ro, and Energy calculations. When set, results of the calculation are updated and displayed interactively. When not set, a calculation is performed only when requested.
18	**Energy**	Provides a quick relative energy calculation between static and dynamic parts of a molecular structure. The energy calculation is a 6–12 van der Waals potential and a Coulombic electrostatic potential between nonbonded atom pairs. One atom in the pair belongs to the static portion of the molecular structure; the other atom belongs to the dynamic portion of the molecular structure. Parameters used in this calculation are those assigned to atoms in the parameter file.
19	**Bump**	Calculates and displays close contacts between atoms. Close contacts are defined as distances between two atoms within a bump cut-off value.
20	**R/Ro**	Calculates the ratio of the current distance between atom pairs to the radius for an optimal van der Waals interaction. When this ratio is less than one, the atoms are too close, and a dashed line connects the atoms with the value of the ratio. When the ratio is equal to one, the atoms are at an optimal distance; when the ratio is greater than one, the atoms are beyond the optimal van der Waals separation.
21	**Bond**	Creates a temporary bond (for example, to simultaneously manipulate fragments).
22	**Bond Break**	Prompts for two atoms to be picked, then removes the bond between the two selected atoms.
23	**Save**	Updates the conformation of the molecular structure, incorporating any torsional, fragment, or atom transformations.
24	**Reject**	Removes all conformational changes since the MSF was last saved, and reconstructs the picture from the MSF saved to disk.
25	**Zero**	Removes all conformational changes since the structure was last saved, removing the most recent modeling transformations.

Table 38. Button box default commands (Continued)

Button	Function	Description
26 -28	**Not Used**	
29	**Global Dials**	Activates Dial Set 1 to perform global transformations.
30	**Fragment Dials**	Activates Dial Set 2 (Fragment Motion).
31	**Torsion Dials**	Activates Dial Set 3 (Torsions).
32	**Display Dials**	Activates Dial Set B to modify the stereo focal length, eye separation, viewing angle and separation, as well as the rocking angle and speed.

Customizing the button box

Button box functions can be customized to meet specific requirements. Open the **Preferences** menu and select **Device Settings** to display a pull-right menu. Select **Button Box** from the pull-right menu and a series of dialog boxes is displayed. The first dialog box (1 in the figure below) asks you to specify which button to assign to a QUANTA command.

A button can act as a toggle or perform the same operation each time it is pressed. After the button number is specified, a dialog box is displayed (2 in the figure above) to specify the behavior of the button.

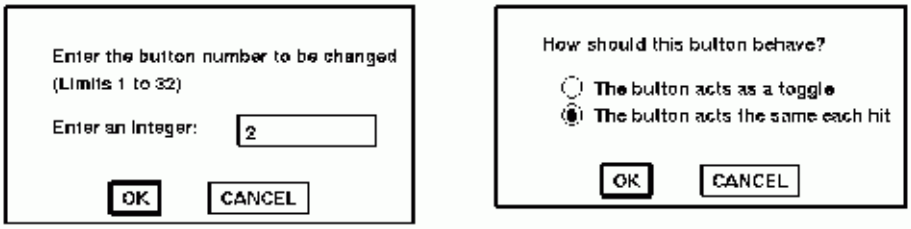
A third dialog box (3 in the figure) asks you to enter the command to send when the button is pressed. If the button is a toggle, another dialog box (4 in the figure) is displayed to enter the alternative command.

This process continues until **Cancel** is chosen from the Button Number dialog box. A correlation between buttons and QUANTA commands is established, and a dialog box requests whether or not to save the new button definitions to a file.

When the definitions are saved, the new button definitions override all default settings, and the file button.dat is created in your current directory. Subsequent QUANTA sessions initialize the button box according to this file. When the choice:

Do Not Save; Lose After this Session

is selected, new button definitions are not written to a file for use in subsequent sessions.



Button number dialog box

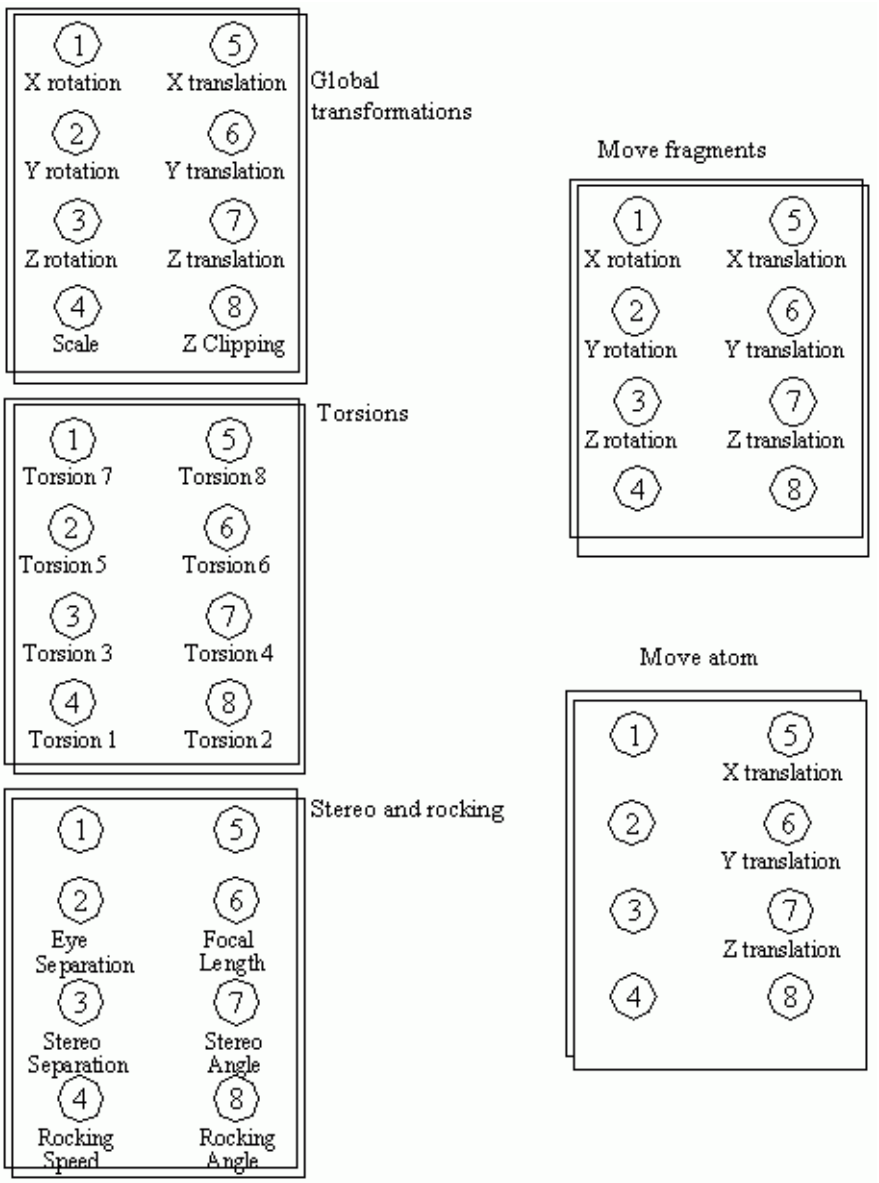
Button type dialog box

Button command dialog box

Save button definitions dialog box

Using a dial box

A dial box is a set of eight dials that perform operations such as translation, rotation, scaling, and clipping. This device is used as an alternative to the dial emulators for global transformations, rocking and stereo variables, fragment movement, atom movement, and torsion transformations. The dials perform specific operations according to the current Dial Set selection. Table 39 summarizes Dial Set functions. The dial box figures summarize default dial assignments. The layout of the Global Transformations dial set can be customized using the **Device Settings/Real Dials** function on the **Preferences** menu.



Using a graphics tablet

A graphics tablet can be used to select items from the QUANTA palette, menus, and display structure. It provides the same functionality as a mouse. This device consists of a four-button puck, a surface where the puck is

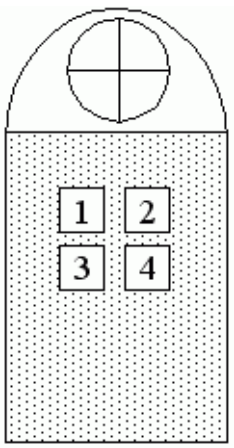
Table 39. Dial box sets

Dial set	Function
1	Global transformations.
2	Move fragment.
3	Torsion rotations.
4	Move Atom. Dials 5, 6, and 7 translate an atom respectively in the x, y, and z directions.
B	Stereo and rocking variables. For use with a stereo viewing device. The separation of the two images and the angle of the image with respect to the viewpoint are controlled by dials 3 and 7. The speed and angle of the rocking function are controlled by dials 4 and 8. Eye separation and focal length are controlled by dials 2 and 6.

placed, and a cursor. The cursor is drawn on the display screen and represents the puck's location on the tablet surface.

The responsive area of the tablet surface has a one-to-one correspondence to the display screen. The upper-left corner of the responsive tablet area maps directly to the upper-left corner of the display; the center of the tablet maps to the center of the display; and so on.

Three of the four puck buttons correspond to the three mouse buttons. The upper-left, upper-right, and lower-left puck buttons have the same functionality as the left, middle, and right mouse buttons, respectively. The lower right button on the puck is not used. The buttons are numbered by row from left to right



Button 1 is the selection button. Clicking this button while pointing to an object such as an atom, menu function, or palette tool sends the appropriate selection information to QUANTA for processing.

A graphics tablet can also control the orientation of molecular structures and viewing transformations. This is useful for quick manipulation of the image in the viewing area without having to select a new dial set from the

Dial Emulator window. Transformations performed with a graphics tablet are based on the speed and direction the puck moves on the tablet surface.

Global transformations

The graphics tablet performs global transformations of molecular structures in 3D space, duplicating the functionality provided by the first set of dial emulators. A molecular structure is manipulated by moving the puck on the tablet surface while simultaneously pressing a button or the <Shift> key. Table 40 summarizes actions you take to perform various transformations.

Table 40. Graphics tablet global transformations

Transformation	Key / button combination
X-Y Rotation	Press and hold button 2 while moving the puck. Rotation continues until the button is released.
Z Rotation	Press and hold button 3 while moving the puck. Rotation continues until the button is released.
X-Y Translation	Simultaneously press and hold button 2 and the <Shift> key while moving the puck. Translation continues until the button or the <Shift> key is released.
Z Translation	Simultaneously press and hold button 3 and the <Shift> key while moving the puck. Translation continues until the button or the <Shift> key is released.
Scaling	Press and hold the <Shift> key while moving the puck, without pressing a button. Scaling continues until the <Shift> key is released.
Z Clipping	Simultaneously press and hold button 1 and the <Shift> key while moving the puck. Z clipping continues until the button or the <Shift> key is released.

Fragment and atom movement

The graphics tablet can transform fragments and atoms in the viewing area, duplicating the functionality of Dial Emulator Sets 2 and 4. The transformation rate is determined by the speed and direction the puck moves across the tablet surface. The transformation type is determined by a keystroke/button combination. Table 41 summarizes actions to take for atom and fragment transformations.

Rotating torsions

Table 41. Graphics tablet atom/move fragment transformations

Transformation	Key / button combination
Fragment X-Y Rotation	Simultaneously press and hold button 2 and the <Ctrl> key while moving the puck. Rotation continues until the button or the <Ctrl> key is released.
Fragment Z Rotation	Simultaneously press and hold button 3 and the <Ctrl> key while moving the puck. Rotation continues until the button or the <Ctrl> key is released.
Fragment X-Y Translation	Press and hold the <Ctrl> key while moving the puck. Translation continues until the <Ctrl> key is released.
Fragment Z Translation	Simultaneously press and hold button 1 and the <Ctrl> key while moving the puck. Translation continues until the button or the <Ctrl> key is released.
Atom X-Y Translation	Simultaneously press and hold button 1, button 2, and the <Shift> key while moving the puck. Translation continues until a button or the <Shift> key is released.
Atom Z Translation	Simultaneously press and hold button 1, button 3, and the <Shift> key while moving the puck. Translation continues until a button or the <Shift> key is released.

The graphics tablet can also rotate torsions. The speed and direction of puck movement controls the torsional rotation. To rotate torsion angles, the <Alt> key is pressed and held in conjunction with none, one, or more buttons. While the <Alt> key is pressed, moving the cursor in the viewing area rotates a torsion angle.

The <Alt> key/button combination determines which torsion angle is rotated. These are listed in Table 42. Rotation continues until the <Alt> key is released.

Table 42. Graphics tablet torsion rotations

Torsion angle number	Key / button combination
Torsion Angle 1	<Alt> key only.*
Torsion Angle 2	<Alt> key and button 1.*
Torsion Angle 3	<Alt> key and button 2.
Torsion Angle 4	<Alt> key and button 3.

Table 42. Graphics tablet torsion rotations

Torsion angle number	Key / button combination
Torsion Angle 5	<Alt> key, button 1, and button 2.
Torsion Angle 6	<Alt> key, button 1, and button 3.
Torsion Angle 7	<Alt> key, button 2, and button 3.
Torsion Angle 8	<Alt> key, button 1, button 2, and button 3.

***Note**: Use <Ctrl>-<Alt>-<Shift> for these operations if the <Alt> key actions invoke the window manager features on Linux.

Miscellaneous transformations

Several transformations are provided to support QUANTA applications and functions. You can perform these using the graphics tablet, moving the cursor in the viewing area while simultaneously pressing a button and, depending on the transformation, the <Shift> or <Ctrl> key. Table 43 summarizes the specific actions to take for each transformation.

Table 43. Graphics tablet miscellaneous transformations

Transformation	Key/button combination
Comparison Flash Speed	Simultaneously press and hold button 2 and button 3 while moving the puck. The speed increases/decreases until a button is released.
Dynamics Frame	Simultaneously press and hold the <Shift> key, button 2 and button 3 while moving the puck. The dynamics frame continues to change until a button or the <Shift> key is released.

Table 43. Graphics tablet miscellaneous transformations (Continued)

Transformation	Key/button combination
Dynamics Speed	Simultaneously press and hold the <Shift> key, button 1, button 2, and button 3 while moving the puck. The dynamics speed continues to change until a button or the <Shift> key is released.
Stereo Angle	Simultaneously press and hold the <Ctrl> key, button 1, and button 2 while moving the puck. The angle of the structure with respect to the viewpoint increases/decreases until a button or the <Ctrl> key is released.
Stereo Separation	Simultaneously press and hold the <Ctrl> key, button 1, and button 3 while moving the puck. The angle of separation between the stereo images increases/decreases until a button or the <Ctrl> key is released.
Rock Speed	Simultaneously press and hold the <Ctrl> key, button 2, and button 3 while moving the puck. The speed of the rocking motion increases/decreases until a button or the <Ctrl> key is released.
Rock Angle	Simultaneously press and hold the <Ctrl> key, button 1, button 2, and button 3 while moving the puck. The angle of rotation of the rocking motion continues to change until a button or the <Ctrl> key is released.

User-defined transformations

The QUANTA Open Interface allows you to supply routines to define a dial set. User-defined dials can use the graphics tablet to adjust parameters specified in the definition. The input rate to these routines is determined by the speed and direction the puck moves across the tablet surface. The dial that this affects is determined by a keystroke/button combination. These are summarized in Table 44.

Using a spaceball

A spaceball for SGI machines is an interactive device with six degrees of freedom. It is used to transform molecular structures in the viewing area. Transformations applied to a molecular structure correspond to the forces exerted on the ball. For example, lifting the ball translates a structure in the positive y direction; pressing down on the ball translates the structure in the negative y direction; pushing the ball left moves the structure to the left. When the ball is twisted, the structure rotates in the direction of the twist.

The spaceball has eight panel buttons programmed to control transformations, and a soft button located on the back of the ball. The buttons specify

Table 44. Graphics tablet user-defined dial transformations

Transformation	Key/button combination
User-Defined 1	Simultaneously press the <Shift> and <Ctrl> key, while moving the puck. The transformation continues until one of the keys is released.
User-Defined 2	Simultaneously press the <Shift> key, <Ctrl> key, and button 1 while moving the puck. The transformation continues until one of the keys or the button is released.
User-Defined 3	Simultaneously press the <Shift> key, <Ctrl> key, and button 2 while moving the puck. The transformation continues until one of the keys or the button is released.
User-Defined 4	Simultaneously press the <Shift> key, <Ctrl> key, and button 3 while moving the puck. The transformation continues until one of the keys or the button is released.
User-Defined 5	Simultaneously press the <Shift> key, <Ctrl> key, button 1, and button 2 while moving the puck. The transformation continues until one of the keys or buttons is released.
User-Defined 6	Simultaneously press the <Shift> key, <Ctrl> key, button 1, and button 3 while moving the puck. The transformation continues until one of the keys or buttons is released.

the type of transformations that are performed, reset the spaceball hardware, and initialize the orientation of the molecular structure. Button assignments are summarized in Table 45.

Table 45 . Spaceball button assignments

Button/function	Description
1/Translations	Controls the translations in effect by cycling through three states: translations on, translations off, translations only about the axis with the greatest force exerted.
2/Rotations	Controls rotations by cycling through three states: rotations on, rotations off, rotations only about the axis with the greatest force exerted.
3/Global Motion	Sets the spaceball to perform global transformations.
4/Global Scaling	Sets the spaceball to perform global scaling. Only the z direction affects scaling (that is, pushing the ball forward reduces the scale factor; pulling the ball back increases the scale factor).
5/Move Fragment	Sets the spaceball to transform the current fragment selection.
6/Move Atom	Sets the spaceball to translate the atom selected by the Move Atom tool. Rotations are ignored.
7/Not Used	
8/Initialize	Resets the spaceball to prevent drift introduced by the hardware.
Soft/Reset	Resets the orientation of the molecular structure.

C. File Formats

This appendix describes the format of several files that are used with QUANTA, including molecular structure files (.msf), residue topology files (.rtf), display parameters files, external files, template files, template files for hydrogen atom addition, dummy atom files (.dum), brick map files (.mbk), quanta plot files (.qpt), ChemNote data files, and atom type files for chemnote and the Molecular Editor.

Molecular structure files

Molecular structure files (MSFs) contain data and information about a molecule. An MSF contains three levels of information:

- Segment data that applies to regions of the structure
- Group data for individual groups or residues within the molecule
- Atom data for individual atoms

In addition, extra information such as solvent accessibility, thermal mobility, and electrostatic potential can be included in the MSF. Extra information is incorporated as a number associated with each atom and can be retrieved through a label. This label enables the selection and coloring of a molecule based on one of these parameters. The extra information can also hold pointers to surface files, symmetry information, or vectors for each atom. In this way, virtually any information about a molecule can be held in the MSF.

The MSF is a sequential binary file. Word length is assumed to be at least 4 bytes in QUANTA unless specified otherwise. The records in the file are as follows.

1. nseg, ngroup, natom, version, header

This record is three integers, the dataset version (character * 10), and a header (character* 200) that contains various flags. The integers represent the number of segments in the file, the number of residues, and the number of atoms.

For files created in QUANTA98 and beyond, the version number is QUANTAR98. QUANTA 2006 reads earlier versions (e.g., QUANTAR3.3) and automatically updates them to QUANTA 2006 for-

mat. The utility $HYD_UTL/.msfi2i4 allows conversion between the old and new versions, if that is required.

2. **TITLE lines**

   Character*80 records, containing the title for the file. Last line is END.

3. **Segment Data consists of three segment records each of length nseg. They are:**

   a. Segment names (character * 4)

   b. Residue pointers (integer*4) that point to the first residue in the segment

   c. Number of residues in the segment (integer*4)

4. **Residue Data contains nine records each of length nresidues. They are:**

   a. Residue identifiers (character*6)

   b. Residue names (character*4)

   c. Atom pointers (integer*4) point to the first atom in the residue in the atom list

   d. Number of atoms in the residue (integer*4)

   e. Segment numbers (integer*4)

   f. X Coordinate of the center of the residue (real)

   g. Y Coordinate of the center of the residue (real)

   h. Z Coordinate of the center of the residue (real)

   i. Radius of the residue from the center (real)

5. **Atom Data consists of seven records each of length natom:**

   a. X coordinate (real)

   b. Y coordinate (real)

   c. Z coordinate (real)

   d. Atom names (character*6)

   e. Residue numbers (integer*4)

   f. Atom types (integer*4)

g. Atom charges (real)

The residue number is a pointer back into the residue data so that the appropriate residue information is available for each atom. The number for each atom type points to parameters held in the parameter file.

6. Extra Data

The QUANTA MSF format allows the storage of extra per-atom information. An MSF created by QUANTA will by default contain a set of extra information that is loaded into the fourth parameter array. This set is the atomic temperature factors labeled BVALUE.

Eight types of data can be stored: real, integer, integer*4, real vector, symmetry, connectivity, bond orders, and atom constraints. Each extra piece of information requires two records (four in the case of a real vector).

a. The first record is a header record: label type nitems file

where:

Label	is a character*10 unique label used within QUANTA to reference the information.
Type	is character*4 defining the type of data as REAL, ASTR, INTG, INT2, or REA3.
Nitems	says how many items are in each record
File	(*100) is the name of a file. The filename is used when the information is a pointer to items in a separate file (for example, in a surface file).

b. The next record(s) contains the nitem pieces of data.

c. For REAL, INTG, ASTR, and INT2, there is just one record. (Atom constraints (ASTR) can only be generated by the program.) For REA3 there are three records.

d. Connectivity information and bond order information can only be generated by the program:

Connectivity	Label	CONNECT
	Type	BOND
	Nitems	natom

A record containing the connectivity information would be: (nbc (i), (ibc (j,i,),j = 1, ncb (i)), i = 1, nitems)

where:

nbc	is an integer*4 array containing the number of bonds to each atom.
bc	is an integer*4 array containing the atom numbers of the connected atoms.

Bond orders	Label	ORDER
	Type	BOND
	Nitems	number of bonds

A record, containing a list of the bond types for each bond would be: (ibf (i), (ib(j,i), j = 1,2), i = 1, nitems)

where:

ibf	is an integer*4 array containing bond type for each bond (single = 1, double = 2, triple = 3, aromatic = 7, non-ring resonant = 12).
ib	is an integer*4 array containing the atom numbers of the atoms forming this bond.

e. There is a special format for symmetry information. The records are character*80 records, which begin with a five- character special label. These are CELL, CRYS, SYMM, SYMMC and END. The CELL line contains a, b, c (angstroms), alpha, beta, gamma (degrees), and the space group name. The CRYS line contains the cell type (TRICLINIC, MONOCLINIC).

   The SYMM line contains symmetry operations as defined in the International tables (this should only include the unique symmetry operations); lattice translations are defined by the lattice type.

   SYMMC lines contain matrices defining non-crystallographic symmetry.

   END defines the end of the symmetry information.

Display parameter file

The display parameter file (param.par) contains the names, bonding, van der Waals radii, and energy parameters of the atoms recognized by QUANTA. This file is read every time QUANTA is started up. This data structure and the QUANTA dictionaries assign graphical display parameters to atoms. For molecules read into QUANTA from an external coordinate file, atom types are assigned solely on the basis of the atom name. This assignment may not be the same as the types determined in ChemNote or the Molecular Editor applications. The parameters in this file are consistent with those used by CHARMm. However, CHARMm accesses a different data structure (PARM.PRM) to obtain parameters for calculation. If a display parameter file is not specified, then all atoms are set to a predetermined default value.

See the *QUANTA Parameter Handbook* for a complete listing of the atom types used in QUANTA.

This format example is part of $HYD_LIB/param.par, the parameter file for proteins.

```
no  bndrad vdwrad plurad  global emin    rmin  patom  hbond atype atmass
1   0.4000 0.9000 0.1000  F      -0.0498 0.800 0.044  D     H     1.00800
2   0.4000 0.9000 0.1000  F      -0.0498 0.600 0.044  D     HC    1.00800
3   0.4000 1.0000 0.1000  F      -0.0420 1.330 0.1    N     HA    1.00800
4   0.4000 0.9000 0.1000  F      -0.0498 0.920 0.044  D     HT    1.00800
```

The following values are applied to the atom type number listed in the first column:

- Bonding radius (bndrad)
- van der Waals spheres radius (vdwrad)
- Sphere radius (plurad) in plots
- Value (global) used in the global search for bonds
- Value (emin and rmin) used in the calculation of van der Waals and electrostatic energy.
- The atom is either a hydrogen bond acceptor (A), hydrogen bond donor (D or E), or not hydrogen bonded (N). The atom is “atype” CHARMm type. The CHARMm type is included for reference only. The energy parameters are taken from the CHARMm topology and parameter files for CHARMm version 22.
- Atomic mass

- If the hydrogen atoms are not defined explicitly, then the parameters for the atom to which they are bonded are modified to take some account of the hydrogen atoms.
- The modified atom types are called extended atom types. These CHARMm atomtype names end with the letter E preceded by the number of undefined hydrogen atoms allowed for in parameterization (for example, CH2E is the atom type for an aliphatic carbon atom with two undefined attached hydrogen atoms).

If you do not enter a parameter for a particular atom type, the atom type uses the default of:

```
no  bndrad vdwrad plurad  global emin    rmin  patom  hbond atype atmass
4990.9    1.4    0.2     F      0.0     0.0   0.0    N     DEFA
```

Atoms with *Mxx* are metals; atoms with *Xxx* are halogens.

If you want to extend the parameter file to include your own atom types, it is recommended that you use atom numbers between 300 and 400.

External file formats

A variety of external file formats are used in QUANTA for input and output of atomic data. This section defines what these formats mean to QUANTA.

Terms used are:

x y z - orthogonal angstrom coordinates of each atom
resid - residue identifier (residue number)
resnam - residue name, such as TRP GLU
atnam - atom name, such as CE2 CA N
bvalue - bvalue or some fourth parameter

Protein Data Bank format

QUANTA reads a standard PDB data file. Atomic coordinates are taken from both ATOM (“standard” groups) and HETATM (“non-standard” groups) records.

Chain identifiers (if present) are used to define segments. If none are present, a segment name is created. HETATMs are placed in separate segments.

The second character of two-symbol element names is lowercase on input. Lowercase characters are specified in QUANTA by preceding the character

with the escape character, usually, but this can be altered using SET ESCAPE. The atom names are all left-justified. This process is reversed on output, to maintain the correct PDB convention.

CHARMm/CNX/X-PLOR PDB variant

The CHARMm/CNX/X-PLOR PDB must be used if a file is to be produced or read by CHARMm, CNX, or X-PLOR. The PDB differs from the standard Brookhaven PDB format in the following respects:

1. The segment name is a four-character string in columns 73 through 76.
2. On export, any * characters in nucleic acid names are converted to ‘ (the * is the CHARMm/CNX/X-PLOR wildcard character).
3. On export, amino/nucleic acids are not reordered to the Brookhaven conventional order.
4. Atom names are read or written straight into the atom name field (the Brookhaven convention is right-justified within the first two characters of the atom name field).
5. CNX and X-PLOR expect the residue ID to be left-justified.

CHARMm ASCII format

QUANTA recognizes both the standard CHARMm and Brunger CHARMm formats. Output is only in standard format. The CHARMm format (.crd) is as follows:

1. TITLE lines (character*80) begin with a *. Last title line is * followed by at least seven blanks. Natom --- defined as i5
2. ATOM lines: atom# resid1 resnam atnam X Y Z segid resid2 bvalue
3. format: I5,I5,1x,a4,1x,a4,3f10.5,1x,a4,1x,a4,f10.5

You are given the option of using resid1 or resid2 as the residue identifier.

Konnert format (command mode only)

resnam resid atnam X Y Z bvalue 2x,a4,1x,a4,a4,4f10.5

Diamond format (command mode only)

X Y Z bvalue resid resnam atnam 4f10.5,6x,a4,15x,a3,7x,a4

CHARMm binary format

The CHARMm binary format (.dcd) is used to hold many sets of coordinates, including the results of a dynamics run at various time steps. The format is as follows:

- HDR,ICNTRL

  character*4 HDR, integer icntrl(20)
  real*4 X(NATOM), Y(NATOM), Z(NATOM)
  real*8 XTLABC(6)
  logical QCRYS

  HDR - not used in QUANTA
  ICNTRL - contains information about the datasets held in file
  QCRYS=ICNTRL(11).EQ.1

  (1) - number of datasets in the file. Not necessarily correct.
  (2) - time of the first dataset - usually in femtoseconds
  (3) - time step between datasets
  (9) - number of fixed atoms (NFIXED)
  (11) - 1 for crystal/constant pressure calculation, 0 otherwise.
  (20) - version number (22 for CHARMm 22, 0 for previous version)

- TITLE

  ntitl,(title(i),i=1,ntitl)
  charager*80 title(32)

- NATOM

  NFREAT = NATOM-NFIXED

- If the number of fixed atoms (NFIXED) is not 0, then the next record is:

  IFREAT(I),i=1,NFREAT) integer ifreat(*) points to the free atoms in the whole list of atoms.

- IF(QCRYS) XTLABC

  (X(I),I = 1,NATOM)
  (Y(I),I = 1,NATOM)
  (Z(I),I = 1,NATOM)

  The coordinates for the first dataset in the file. Time icntrl(2). If NFIXED is 0, this first dataset is the complete set of data giving positions for both the free and fixed atoms.

  Note: If this is a file from a Crystal/Constant Pressure calculation, i.e. ICNTRL(11)=1 and QCRYS=TRUE, then there will be an extra record

containing symmetric shape index data, XTLABC. XTLABC is a symmetric shape matrix, only lower triangle is used.

- To the end of file

  IF(QCRYS) XTLABC
  (X1(I),I = 1,NFREAT)
  (Y1(I),I = 1,NFREAT)
  (Z1(I),I = 1,NFREAT)

  Coordinates for the next dataset. If NFIXED is not 0, these coordinates are used as:

  do 10 i = 1,NFREAT
  10 X(IFREAT(I)) = X1(I)

CHARMm binary property files

The CHARMm binary property files are similar to the binary coordinate files except there is only one entry for each dataset (in contrast to the three - x y z in a coordinate file).

Fractional coordinates (old Cambridge database format)

The Cambridge database file has a four-line header. In the following example, required spaces are represented by explanatory notes enclosed in square brackets [], indicating the number of spaces.

Line 1	’Reference_Structure_=_’,I5,’[3 spaces]A,B,C=’,3F8.3
Line 2	[2spaces]ALPHA,BETA,GAMMA=’,3F8.3,4X,’SPGR_=’,I3, 1X, A6
Line 3	I3,3X,’0 CODON=_’,9X,’SYMOPS=’,15
Line 4	2X,’0’,5X,’RFAC=_’,F3.1,’_ERRFLAG=0_(C-C)ESD=0’

where the *Reference Structure* is the Cambridge Data Bank code number for the structure. The cell parameters given as A, B, C and ALPHA, BETA, GAMMA, and the space group code are the only header information used and stored by QUANTA. If you are writing out a file in C. D. B. format, you must provide the other information or accept meaningless defaults.

The coordinates (in cell fractional coordinates) then follow, using the form atom number, atom name, fractional coordinates, and atoms bonded to this atom (e.g., I4, 1X, A4, 1X, 3F10.5, 1X, 6I4). All atoms bonded to each atom should be listed so each bond is in effect defined twice.

Cambridge database FDAT format

Due to licensing considerations, for more information regarding this format, please contact:

Crystallographic Data Centre
12 Union Road
Cambridge
CB2 1EZ
UK
+44 1223 336408

Gromos atom format

- TITLE LINE
- Number of atoms I5
- Coordinates
- Residue number, residue type, atom type, atom number, orthogonal coordinates (e.g., I5, 2A5, I5, 3F8.3)

Orthogonal coordinates may be in angstroms or nanometers. The program tests the coordinates and suggest which units are being used, but you can override this decision.

The Gromos program expects the atoms of a residue to be given in a specified order. When outputting Gromos files, QUANTA attempts to reorder atoms correctly. The file $HYD_LIB/gromos.ord contains a list of all the amino acid atoms in required order. You can edit this file if necessary. The required ordering for non-amino acid residues is not included in the file. The routine expects atom names to follow IUPAC-IUB conventions. If they do not, atoms are liable not to be recognized and placed at the end of the residue. You are informed when this is the case.

Converting Gromos Trajectory Files. The program called, GROCH converts Gromos format trajectory files to CHARMm format trajectory files which can be read by QUANTA. When using this program, you should be prepared to provide the following information on the Gromos format file:

- Coordinates in angstroms or nanometers
- Number of atoms in dataset
- Number of datasets

Accelrys provides the GROCH program as source code in the QUANTA Utility directory.

QM coordinate file format

Coordinate files which written as a result of quantum mechanics calculations are identified in QUANTA by the extension .qmc.

Dictionary files

Nohpro.dic file

The nohpro.dic file provides a reasonable assignment of charges in proteins when there are no hydrogens in the structure. The file contains documentation describing the strategy, which is essentially:

1. If the amino acid residue is charged, then the sidechain total charge adds up to +1 or -1.
2. If it is not charged, then:
   a. If it is a donor, the total charge is slightly positive.
   b. If it is an acceptor, the total change is slightly negative.
   c. If it is both, the total charge is null.

Generic.dic file

The generic.dic file adds some additional charges to atoms, reduces the default charges on oxygens, and adds a default charge on nitrogen.

Template files

Template files contain ideal coordinates for a residue along with other information required to perform mutations.

The first record consists of:

resnam natom npos ntor

where:

resnam =	residue name;
natom =	number of atoms in the residue, number of atom records in the file;
npos =	number of records defining bonding used to fit template mutations;
ntor =	number of torsion angles to be set up after mutation.

1. N Atom Records - The format for each atom record is:

ATNAM TYPE X Y Z

where:

ATNAM =	atom name
TYPE =	atom type
X Y Z =	atomic coordinates

These files are adapted from the Brookhaven Protein Data Bank (PDB) format so there are extra fields in this record which are not used by QUANTA.

2. N Pos Records - The format for each pos record is:

AT1 AT2 AT3 AT4

This specifies that the position of AT1 is defined in terms of the of AT2, AT3 and AT4.

3. N Tor Records - The format for each tor record is:

   At1 At2 At3 At4

   This specifies that, if AUTO TOR is turned on, the torsion angle At1-At2-At3-At4 should be set up after the mutation.

The example shown here is the template file for Valine, $HYD_LIB/tmplatnoh/val.pdb.

```
VAL      7 3 1
ATOM     151      N        VAL      19105    52.833   36.648   0.000    -0.35    -10.00
ATOM     152      CA       VAL      19 11    53.367   37.966   0.000      0.10   -10.00
ATOM     153      C        VAL      19 14    54.887   37.966   0.000      0.55   -10.00
ATOM     154      O        VAL      19 40    55.529   36.947   0.000    -0.55    -10.00
ATOM     155      CB       VAL      19 11    52.723   38.811   1.142      0.00   -10.00
ATOM     156      CG1      VAL      19 13    53.562   38.898   2.418      0.00   -10.00
```

```
ATOM     157      CG2      VAL      19 13    52.366   40.201   0.587      0.00   -10.00
5 2 1 3
6 5 2 3
7 5 2 3
1 2 5 6
```

The template files are in subdirectories of the library directory and references to them are kept in template library files in the library directory. There are three sets of templates:

- tmplatnoh.tlf - Proteins with no hydrogen atoms defined
- tmplatpol.tlf - Proteins with polar hydrogen atoms defined
- tmplatall.tlf - Proteins with all hydrogen atoms defined

QUANTA determines the correct files to use automatically. The default template library is tmplatnoh.tlf for proteins with no hydrogen atoms defined.

The following example, the protein_polarhydrogen template library file, $HYD_LIB/tmplatpol.tlf, shows the template library format.

```
*N CA C
tmplatpol/ala.pdb Alanine ala A
tmplatpol/arg.pdb Arginine arg R
tmplatpol/asn.pdb Asparagine asn N
tmplatpol/asp.pdb Aspartic_acid asp D
tmplatpol/cys.pdb Cysteine cys C
```

The first record is a * followed by the names of three backbone atoms, that is atoms whose position is invariant when residues are mutated).

If these three atoms are not found in the residue to be mutated, the program issues the error message: `Join atoms not properly defined`. If this error message appears, you should not continue with the mutation. Old versions of the template library files do not contain this first line and the program assumes that the molecule is a protein with main chain atoms N, Cα, C.

The file then contains one record for each residue, listing the name of the template file, the name of the residue, a three letter code for the residue, and a one letter code for the residue. The residue name must be a single word. If necessary, words can be connected with underscores, for example, aspartic_acid).

During interactive mutation, QUANTA looks at the atoms present in the residue to be mutated and attempts to assign the correct template library file.

Hydrogen atom addition template file for Protein Design

In the Protein Design application, QUANTA contains a simple algorithm to add hydrogen atoms to a protein. This algorithm uses a template to fit hydrogen atoms, but it does not perform energy minimization so the structure geometry may not be ideal. However, a molecular mechanics program such as CHARMm can be used to improve the structure geometry. The hydrogen bond addition routine assumes atoms have the atom type numbers defined in the param.par parameter file. If errors occur when you use this routine, it is probably due to bad atom type assignments.

The hydrogen addition algorithm superimposes a template over the existing atoms in the structure and then takes hydrogen atom coordinates from the template and adds them to the structure. In order to fit the template unambiguously, the template must contain the coordinates of three atoms which already exist in the structure. These three atoms are usually the atom to which hydrogen atoms will be attached and the two first neighbors. For example, in adding two hydrogen atoms to a tetrahedral carbon atom which is already bonded to two non-hydrogen atoms, the required template must contain coordinates for the atoms.

For some structures, the atom to which hydrogen atoms are to be added may only have one first neighbor. In this case, the third guiding atom in the template must be a second neighbor. For example, in adding three hydrogen atoms to a methyl carbon atom, the template file contains coordinates for the atoms.

The coordinates for all the templates are contained in a single file hydtpl.dat which may be found in the Library Directory.

The following format is the template for methyl C:

```
* add 3 hydrogens to tetrahedral C
4  1  1  3  2
13
10
2       3.980    -1.526    -2.575
0       3.494    -0.781    -1.318
0       1.956    -0.745    -1.257
3       5.069    -1.537    -2.593
3       3.607    -2.550    -2.558
3       3.607    -1.019    -3.465
```

Lines beginning with a * are treated as comments and are not read by the program. Each template begins with one or more comment lines.

The first line read by the program contains the following parameters:

Template number
Number of first neighbors
Number of second neighbors
Number of hydrogen atoms
Polar/non polar code

The template number must correspond to the position of the template in the template file. The polar/nonpolar code has a value of 1 for polar hydrogen atoms and 2 for nonpolar hydrogen atoms.

The next line lists the atom types for which the template is applicable. This is followed on the next line by the new atom type codes for the atoms once the hydrogens have been added.

The atom coordinates that follow are in the order:

atom to which hydrogen atoms will be attached
first neighbors
second neighbors
hydrogen atoms

The first column of the coordinate data is the atom type code. It is redundant except for hydrogen atoms. This is the atom type code that is given to added hydrogen atoms. Table 46 lists the atom type codes.

Table 46. Atom type codes

Atom number	CHARMm atom code	Number of hydrogens added	Geometry	New atom type
11	CH1E	1	tetrahedral	20
12	CH2E	2	tetrahedral	10
13	CH3E	3	tetrahedral	10
23	C5RE	1	trigonal	21
24	C6RE	1	trigonal	22
71	SH1E	1	trigonal	70
201	CU1E	1	trigonal	16
202	CU2E	2	trigonal	16
203	CW1E	1	trigonal	17
204	CW2E	2	trigonal	17
205	NP1E	1	trigonal	32
206	N5RE	1	trigonal	34
207	NT1E	1	tetrahedral	36
208	NT2E	2	tetrahedral	36
209	NT3E	3	tetrahedral	36

Table 46. Atom type codes (Continued)

Atom number	CHARMm atom code	Number of hydrogens added	Geometry	New atom type
210	NC2E	2	trigonal	37
211	OI1E	1	tetrahedral	44
212	OT1E	1	tetrahedral	45
220	N6RE	1	trigonal	35
221	NP2E	2	trigonal	32

Dummy atom file

Dummy atom files (.dum) have the following format:

NAME TYPE nat other

where:

- NAME is the dummy atom name
- TYPE is the type MIDPoint, VECTor etc.
- nat is the number of atoms defining this dummy, these follow on the
- nat subsequent records.
- other depends on type. For example there may be 3 coordinates for COOR, a distance for VECT and PERP and so on.

The following example is a file for a static dummy which was defined as a midpoint of 3 atoms. The commented lines (starting with #) form the midpoint definition.

```
DUM1      COOR 0           .103      .332         .418
#DUM1     MIDP 3
#ATOM C2 RESI      BENZ:1    MOLE benz.msf
#ATOM C4 RESI      BENZ:1    MOLE benz.msf
#ATOM C6 RESI      BENZ:1    MOLE benz.msf
```

The following file is used for a dynamic dummy:

```
DUM1      MIDP 3
ATOM   C2   RESI      BENZ:1    MOLE benz.msf
ATOM   C4   RESI      BENZ:1    MOLE benz.msf
ATOM   C6   RESI      BENZ:1    MOLE benz.msf
```

Brick map file

The QUANTA brick map files (.mbk) can be used to store 3D information on a grid. These files can then be used to create contoured wire-frame representations of the information, graphical objects in QUANTA, or a map within the X-Ray structure package.

The information on this 3D grid is stored in bricks. The overall grid is divided into 6 x 6 x 6 pieces, and the data for each brick is stored in a single direct-access record. This approach increases the speed and flexibility of selecting and retrieving portions of a complete map for contouring. In practice, bricks overlap one another on one edge to ensure that a continuous surface is generated in contouring.

The QUANTA brick map file can be used to store single values at each point on a 3D grid, a vector at each grid point, or both a number of vectors and single points. In addition, each vector or single grid point can be a single byte or a integer*4 value. How you choose which sized value to use depends on the balance between dynamic range and disk space. Within a file, the type of data used must be the same. QUANTA contains facilities to recognize and work with any of these combinations of data types.

The header of the brick map file contains all the information about the contents of the file. QUANTA reads brick map files generated by earlier versions of the program.

Brick map files are direct-access files with record lengths of 54 words for byte-type files and 216 words for integer*4-type files. The format is:

line 1: version, ntitle, filetype

- version is a character*7 variable. For earlier version of QUANTA, this should be mbk_1.0; for QUANTA 2000 and above it should be mbk_2.0.
- ntitle is the number of title records to follow (integer*4)
- filetype (integer*4) is the type of file: 1 - byte 2 - integer*4

lines 2 to ntitle+1: (title(i),i=1,ntitle)

- title lines of char*100 length

There is a limit of 50 title lines. The title contains not only textual information about the file, but also some HEADER records that detail the type of information (single point or vector) held in the file. If no HEADER records are included, QUANTA assumes this is a file containing just a single grid of scalar values.

The following additional HEADER records are only necessary if a vector field display is to be generated. The order of the HEADER records reflects the order of the data in the file. A “V” at position 27 in the HEADER record indicates that the file contains three sets of grid data corresponding to the x, y and z of a vector for the grid points. An “S” at position 27 in the HEADER record indicates a scalar set of grid data.

For example, a brick map file containing the magnitude and direction of an electrostatic field as byte values around a molecule would have the following header information:

```
mbk_1.0, 3,1
```

Gives the version number and the number of title lines (3) and indicates a byte map.

```
HEADERFIELD S1
```

Indicates that the first grid of data will be a single byte scalar value of the electrostatic field. The scale and offset from the main header apply to this grid of data

```
HEADER VECTOR ORIENTATION V1 1.000000E+01 0.000000E+00 –
```

Indicates that the next three grids of data in the file will form a vector of bytes with a scale and offset taken from the values given.

```
Electrostatic Filed map generated on ........
```

A title line

The final record before the grid data is the main header block of information specifying the various parameters that define the position, scale, grid, and so on of the map as indicated below.

```
card ntitle+2: nsec, mxyz,nbxyz,nw1,nu1,nu2,nv1,nv2,
cell,rhrms,offset,scale,lenbrk,ncode,rhmin,rhmax
(int4 or real4 as per first character of name).
```

where:

- nsec - number of sections of density stored in map
- mxyz(3) - grid points per unit cell edge in a,b,c
- nbxyz(3) - number of bricks in a,b and c in the file
- nu1,nv1,nw1 - starting grid point in a,b and c in file
- nu2 nv2 - finishing grid point in a and b in file
- cell(6) - cell constants
- ncode - orthogonalisation code
- rhmin,rhmax,rhrms - minimum, maximum and mean value

- offset, scale - the offset and scale to convert from the value in the bricked map to the value in the original map.
- lenbrk - length of the bricks in the map (usually 6)

```
card ntitle+3 - end
bricks of density written as
for brick i,j,k
write(n,rec=irecno)brick written along a fast, b medium, c low
where:
irecno = (k-1)*nbxyz(2)*nbxyz(1)+(j-1)*nbxyz(2)*nbxyz(1)+ 2+ntitle
```

The cell constants can be interpreted in one of two ways, depending on the value of the orthogonalization code, ncode. If ncode is between 1 and 6, then the grid is in a standard crystallographic fractional coordinate system with the cell constants, specified as a,b,c, alpha, beta, gamma, defining the transformation of orthogonal angstroms. This requires that one of the grid points falls on the origin. If the ncode is 0, then the cell constants define the origin and extent of the grid of points, specified as origin(x), origin(y), origin(z), extent(x), extent(y), extent(z), in orthogonal angstroms. This allows a grid that does not fall on the origin to be stored.

Special value

If a grid point in an integer*4 brick map has a value of 32766 then it will be ignored in the contouring within QUANTA. This is useful for masking certain parts of a grid, without getting a contour at this boundary.

QUANTA plot file

The plot file (.qpt) in QUANTA is a binary file with each record written as a, x, y, z (i.e., 4 real numbers). The command number is represented by “a;” x, y, and z are parameters for the command:

a = 1 change to color (or pen) x, line width y
a = 2 move to x y z
a = 3 line to x y z
a = 4 dot at x y z
a = 5 draw x characters followed by a record of string (1: x)
a = 6 character scale x y z (interpretation of this in the program

a = 7 define patterned line where:

z is length of pattern
y is space length
x is dash lengths

a = 8 new frame
a = 9 rotate everything by x degrees
a = 10 set the units for the plot
a = 11plot a symbol as char(x) at the current point
a = 12 delete this
a = 13 set the plotter limits to x,y in physical device coordinates
a = 14 set the plotting window to x,y
a = 15 flag to specify stereo plotting (x is the stereo angle to use) where the stereo is created by the plotting program
a = 16 two records to specify an rgb value for a color number the first record specifies the color number (x) the second record specifies r g b as x, y, z
a = 17 two records to specify a filled rectangle the first record contains the bottom left corner the second record contains the top right corner

Not all the commands are currently used by QUANTA and the interpretation of many of them depends on your plotter and the program you use to drive it.

ChemNote data files

The following data files are used by ChemNote, the Sequence Builder, and the Molecular Editor applications.

1. $QNT_CHEM/quanta.tpl

This is a template file for the ChemNote to CHARMm conversion mode.

2. $QNT_CHEM/chrmtype.typ and $QNT_CHEM/chrmpost.typ

These files define the CHARMm atom types.

3. $QNT_SEQ/peptide.bck

This file contains values for backbone structures of polypeptides. This file is read by the Sequence Builder. Changes are made by editing the file directory.

Conformation of a molecule is specified by indicating values for dihedral angles. Backbone structure often extends over several residues. Commonly used structures are defined in this file.

The format of the file, and instructions for modifying the information contained in the file are given in the file.

4. $QNT_SEQ/peptide.nom

This file contains the shorthand names for dihedral angles and is read by the Sequence Builder. Changes are made by editing the file directly.

Conformation of a molecule is specified by indicating values for dihedral angles. Dihedral angles are identified by the four atoms involved in the angle. However, there are many shorthand names for the important angles that make dihedral identification easier. Phi, psi, and omega are some of these shorthand names in a polypeptide backbone. Because there is more than one convention for naming dihedral angles and the shorthand names may change from residue to residue, the Sequence Builder must have a flexible way of giving dihedral angles shorthand names.

The format of the file, and instructions for modifying the information contained in the file are given in the file.

5. $QNT_SEQ/sequence.tpl

This template is used by the Sequence Builder to create a command input file for residue sequences.

6. $QNT_SEQ/seq_menu.aud

This file contains the menu and dialog box information for the Sequence Builder.

Atom type files for ChemNote and the Molecule Editor

The files chrmtype.typ and chrmpost.typ contain the definitions used to assign atom types to atoms in ChemNote and the QUANTA Molecule Editor. The files contain several rules, each associating a pattern with an atom type. If the atoms and bonds around a specific atom match the pattern in a rule, then the atom is assigned the rule's atom type.

The format of both files is the same, However, the files vary in usage. The file chrmtype.typ is applied first to obtain most of the basic typing. Highly

complicated systems such as some heterocycles and conjugated systems must have their typing refined. This is the function of the file chrmpost.typ. Both files are applied to all molecules but molecules with simple typing are not affected by the rules contained in chrmpost.typ.

In the atom type files, lines starting with '*' must appear exactly as illustrated. Lines starting with '!' are comments which are ignored by the program. The file is divided into the following sections:

1. **The standard file header**

   The program checks the format version number against the expected number and issues an error message if the numbers do not match and the file is not read. The file update version number, which represents the last date that the file was changed by Accelrys, should not be altered.

2. **The number of rules in the file**

   The format of this line is "P [#]", where [#] represents the number of rules defined in the file. In the example, the full file must define 298 rules. If you add or remove rules, adjust this number accordingly. There is no predefined array limit on the number of type rules that can be added, but each rule takes up some memory. If too many rules are added, typing may become slower, or in extreme cases even generate "out of memory" errors.

3. **The rules**

   There must be exactly as many rules as indicated by the number on the 'P' line mentioned above. It is a good idea to try to illustrate the pattern being defined in the rule using comment lines before each rule, as shown in the example. Most rules have such illustrations, and you are encouraged to keep the pictures up-to-date if you change or add rules.

   A rule begins with a line containing "T [#]", where [#] states the number of subsequent lines which make up the rule. Each line of the rule after the "T" defines an atom in the pattern, so [#] also represents the number of atoms that the pattern matches. In the first rule in the example, the line is "T 4", and the rule contains four subsequent lines.

   The first atom in the rule is special, since it is the atom whose type will be assigned when the pattern is found to match a part of the structure. The next few lines describe the atoms directly connected to the first atom, subsequent lines define atoms connected to these atoms, and so on. There is no real limit on how far a rule can extend, but on a practical basis rules rarely travel more than three atoms out.

The format of an atom line is: a b c element

The first field, a, specifies the line number relative to the current atom line where definitions for attached atom begin. For example, in the first atom line in the first rule in the example, a is 1. This means that the next line in the rule begins the definition of atoms attached to the central atom of the rule, in this case a hydrogen. A will always be 1 in the first line of a rule.

The second field, b, specifies the number of atoms connected to the current atom. All connected atoms must be defined in contiguous lines in the rule, starting with the line specified by the first field as described above. If b is positive, then exactly that many atoms must be attached to the atom for it to match the pattern. If b is negative, then there must be at least |b| atoms attached (|b| is the absolute value of b); if there are more the atom will still fit the pattern. If b is zero then the rule does not continue beyond this atom; it doesn't matter how many atoms are connected to it.

The third field, c, has a different meaning for the first atom than it does for subsequent atoms. For the first atom, it specifies the numeric atom type that will be assigned to the atom if the rule matches the atom and its surroundings. This number is associated with the atom type names in the file MASSES.RTF. For all lines after the first, this number represents the bond order for the bond between this atom and the subsequently-defined atom it is connected to. In the second line of the rule in the example, this number is 1, which means there must be a single bond between the N and H. A '?' may be used as a wildcard, in which case it's only important that a bond exists, not what sort of bond. Allowable bond orders are 1, 2, 3, 7, and 12, where 7 and 12 may be used interchangeably to indicate a resonant or aromatic bond.

The fourth field, element, specifies the element each atom must be to match the rule. In all lines after the first, a '?' may be used as a wildcard to match any element. So in the example rule, the first line specifies that the rule matches a hydrogen; the second line matches a nitrogen.

Looking at the first rule, we see that the first line specifies a hydrogen atom; it must be attached to only one atom, whose definition is on the next line. If the rule matches, the hydrogen will be given the atom type 2 (HC). The next line specifies that the attached atom must be a nitrogen, connected to the hydrogen by a single bond; furthermore it must be attached to exactly two other atoms, whose definitions follow. The following lines specify that it doesn't matter what element those two atoms are, just that one of the bonds must be resonant, and one single.

4. Special ring type rules

The above rules do not deal with cyclic systems; some atom types, however, are specific to ring systems. The last section of the file contains rules which assign ring-specific atom types.

The first line of this section has the format “R [#]”, where [#] is the number of ring rules that follow. Since each ring rule is a single line, [#] also specifies the number of lines that follow before the end of the file (excluding blank or comment lines).

Each line has the following format:

```
type size new_type ring1 ring2 ring3
```

These rules are applied after the initial typing rules are finished, so they can depend on the atom types the initial rules have assigned.

The first field, type, specifies the numeric atom type an atom must have for QUANTA to attempt to apply the rule to the atom.

The second field, size, specifies that the atom must be a member of a ring of the specified size. If size is negative, it means that the ring must be aromatic as well as having |size| number of atoms. If size is -1, however, it means the rule applies to atoms in conjugated ring systems, and then the fourth, fifth, and optionally sixth fields are used to specify the sizes of the two or three rings the atom must be a member of. If size is zero, it means the atom must be a member of at least one ring, but the number of rings and the ring size is unimportant.

The third field, new_type, specifies the atom type to assign to any atom that matches the pattern described by the current rule.

So the ring rule:

```
33 0 32
```

means that any atom of type NX (33) that appears in a ring should be changed to an NP (32). The more specific rule:

```
22 -5 21
```

means that any C6R atom (22) that is in a 5-member aromatic ring should be changed to C5R (21). Finally, the conjugated ring rule:

```
27 -1 26 6 5 0
```

means that any atom of type CR66 that is a member of both a 5- and 6-membered ring should become CR56.

It is possible to have up to three conjugated rings, so ring sizes should be delimited. If fewer than three rings are used, fill the remaining fields with 0.

5. End of file

The file is terminated by a line containing “* End of File”.

Atom typing rules example

The following example represents a portion of the atom typing rule file.

```
* Polygen Corporation: ChemNote atomtype rules file
* File format version number
86.1124
* File update version number
91.0621
*
! Total number of patterns in the data file.
P 298
! H2-N- H on a charged group - HC
! |r
!

T 4
 1 1 2 H
 1 2 1 N
 0 0 1 ?
 0 0 12 ?
!
! HC-NC- H on a uncharged guanidinium group - HC
!
.
.
.
.
T 1
 1 0 176 Re
T 1
 1 0 6 MBe
T 1
 1 0 7 B
!
! Ring cycles
!
.
.
.
.
R 21
 33 0 32
 22 -5 21
```

```
 27 5 25
 28 6 29
 28 0 14
 30 6 39
 23 6 24
 34 6 35
 72 6 73
 52 6 53
 182 6 181
 10 3 191
 10 4 193
 14 3 190
 14 4 192
 32 4 33
 27 -1 26 6 5 0
 182 -1 180 5 6 0
 25 -1 26 5 6 0
 181 -1 180 6 5 0
 195 -1 26 6 5 0
* End of File
```

D. Residue Topology Files

The following tables identify residues and patch residues contained in the residue topology files distributed by Accelrys.

Amino acid residue topology files

Table 47 lists amino acid residues and patch residues in alphabetical order by name. Residues are listed first, patch residues follow. Water model residues are listed separately at the end of the residue listing.

Table 47. Amino acid residue topology files (AMINO.RTF and AMINOH.RTF)

Name	Type	Description
ABU	Residue	Gamma amino butyric acid
ACE	Residue	Adds acetamide group to N-terminus
ALA	Residue	L-alanine
ANAP	Residue	Alpha naphthyl alpha phenylalanine
ARG	Residue	L-arginine (charged)
ARGN	Residue	L-arginine (neutral)
ASN	Residue	L-asparagine
ASP	Residue	L-aspartic acid
ASPH	Residue	L-aspartic acid (protonated)
ATP	Residue	Adenosine triphosphate
BAA	Residue	Beta alanine (3-amino propionic acid)
BEN	Residue	Benzamidine inhibitor
BNAP	Residue	Beta naphthyl alpha phenylalanine (polar H only)
CHXA	Residue	Cyclohexylalanine
CIT	Residue	Citrulline
CYS	Residue	L-cysteine
DAA	Residue	Delta amino pentanoic acid (5-amino pentanoic acid)
DPRO	Residue	D-proline
EAA	Residue	6-amino hexanoic acid
FMN	Residue	Flavin mononucleotide
GAA	Residue	Gamma amino butyric acid (4-amino butyric acid)
GDP	Residue	Guanosine diphosphate
GGLU	Residue	Gamma glutamic acid

Table 47. Amino acid residue topology files (AMINO.RTF and AMINOH.RTF) (Continued)

Name	Type	Description
GLA	Residue	Gamma carboxy glutamic acid
GLN	Residue	L-glutamine
GLU	Residue	L-glutamic acid
GLUH	Residue	L-glutamic acid (protonated)
GLY	Residue	Glycine (uses patch residue GLYP as its default N-terminal patch in AMINO.RTF)
HCYS	Residue	Homocysteine
HIS	Residue	L-histidine (uncharged with a proton on ND1)
HLYS	Residue	Hydroxylysine
HPHE	Residue	Homophenylalanine
HPRO	Residue	Hydroxyproline
HSC	Residue	L-histidine (protonated)
HSD	Residue	L-histidine (uncharged with a proton on NE2; an isomer of HIS)
HSER	Residue	Homoserine
ILE	Residue	L-isoleucine
LEU	Residue	L-leucine
LYS	Residue	L-lysine (charged)
LYSN	Residue	L-lysine (neutral)
MET	Residue	L-methionine
MP3	Residue	3-phosphoglycerate
MPD	Residue	2-methyl-2,4-pentanediol
MTX	Residue	Methotrexate
ORN	Residue	Ornithine
PC	Residue	Phosphocholine, deprotonated re 2mcp of PDB
PCA	Residue	
PCLP	Residue	Para chlorophenylalanine
PEN	Residue	Penicillamine
PFP	Residue	Para fluorophenylalanine
PGLY	Residue	Phenylglycine
PHE	Residue	L-phenylalanine
PRO	Residue	L-proline (uses patch residue PROP as its default N-terminal patch)
PSER	Residue	Phosphoserine
PTYR	Residue	Phosphotyrosine
SER	Residue	L-serine
STAO	Residue	Statone
STAI	Residue	Statine
STYR	Residue	Sulfurated tyrosine
THR	Residue	L-threonine
TRP	Residue	L-tryptophan
TYR	Residue	L-tyrosine

Table 47. Amino acid residue topology files (AMINO.RTF and AMINOH.RTF) (Continued)

Name	Type	Description
UALA	Residue	Dehydroalanine
UPHE	Residue	Z-dehydrophenylalanine
VAL	Residue	L-valine
ZAA	Residue	7-amino heptanoic acid
OH2	Residue	TIPS3P water model (with no bond between the hydrogens)
TIP3	Residue	TIPS3P water model (including bond between the hydrogens for SHAKE)
HOH	Residue	Identical to TIP3
DOD	Residue	Identical to TIP3 (treats D_2O as H_2O)
ST2	Residue	ST2 water model (including two lone pairs on the oxygen)
WAT	Residue	TIP3P Water model for PDB systole waters
ACCP	Patch residue	Default N-terminal patch for ACC
CALD	Patch residue	Replaces the C-terminus with an aldehyde group
CALC	Patch residue	Reduces the C-terminus carboxyl group to an alcohol
CAMD	Patch residue	Converts the C-terminus to an amide
CAME	Patch residue	Adds a methyl group to the alpha carbon; may not be used with Gly
CBZS	Patch residue	Replaces the C-terminus with a benzyl ester group
CEOH	Patch residue	Converts the C-terminus to an amide of ethanol amine
CETS	Patch residue	Replaces the C-terminus with an ethyl ester group
CGLY	Patch residue	Provides a charged COO^- group to Gly at the C-terminus
CH2	Patch residue	Replaces the carbonyl group with CH_2
CIS	Patch residue	Converts *trans* peptides to *cis*
CMAM	Patch residue	Converts the C-terminus to an N-methyl amide
CMES	Patch residue	Replaces the C-terminus with a methyl ester group
COOH	Patch residue	Adds a proton to the default C-terminal patch
CPHS	Patch residue	Replaces the C-terminus with a phenyl ester group
CTBS	Patch residue	Replaces the C-terminus with a tert-butyl group
CTER	Patch residue	Default C-terminal patch for all standard residues
CTRA	Patch residue	C-terminal patch for all standard residues; may only be applied after the PSF has been generated
CTRU	Patch residue	Provides a charged COO^- group to unsaturated amino acids
DCOH	Patch residue	Removes COOH from C-terminus
DEAM	Patch residue	Removes NH_3 from N-terminus; should not be used in the Sequence Builder
DISU	Patch residue	Constructs a disulfide bond between two Cys residues
DLNK	Patch residue	Joins the carboxyl group of Asp with the N-terminus to form a cyclic peptide (AMINOH.RTF only)
GLYP	Patch residue	Default N-terminal patch for Gly (AMINO.RTF only)
HPRP	Patch residue	Default N-terminal patch for HPro

Table 47. Amino acid residue topology files (AMINO.RTF and AMINOH.RTF) (Continued)

Name	Type	Description
ILNK	Patch residue	Joins two segments to form a peptide bond
KLNK	Patch residue	Joins the amino group of Lys with the C-terminus to form a cyclic peptide
LINK	Patch residue	Joins two segments to form a peptide bond; links N to C
LIMG	Patch residue	Same as LNK2 but used to patch IMAGES. In AMINOH.RFT only
LNK2	Patch residue	Same as LINK but links C to N
LNP1	Patch residue	Link patch for proline on the N-terminus (links N to C)
LNP2	Patch residue	Same as above (applied C to N)
LTOD	Patch residue	Converts an L-amino acid to a D-amino acid; may not be used on Pro
NACP	Patch residue	N-terminal acetamide for Pro or HPro
NACT	Patch residue	Replaces the N-terminus with an acetamide group; may not be used with Gly (AMINO.RTF), Pro, or HPro as the first residue (see NACP)
NBOC	Patch residue	Replaces the N-terminus with a tertbutoxylcarbonyl group; may not be used with Gly (AMINO.rtf), Pro, or HPro as the first residue
NCBZ	Patch residue	Replaces the N-terminus with a carbobenzoxyl group; may not be used with Gly (AMINO.rtf), Pro, or HPro as the first residue
NDME	Patch residue	Adds a N,N-dimethyl group to the N-terminus
NETC	Patch residue	Replaces the N-terminus with an ethylcarbonyl group; may not be used with Gly (AMINO.rtf), Pro, or HPro as the first residue
NFA	Patch residue	Replaces N-terminus with $C_{13}H_{27}$ fatty acid amide
NFAG	Patch residue	Same as NFA but for glycine as the N-terminus
NFRM	Patch residue	Replaces the N-terminus with a formamide group; may not be used with Gly (AMINO.RTF), Pro, or HPro as the first residue
NGUA	Patch residue	Adds a guandino group to the N-terminus; may not be used for Pro
NH2	Patch residue	Deprotonates the charged N-terminus
NMEC	Patch residue	Replaces the N-terminus with a methylcarbonyl group; may not be used with Gly (AMINO.rtf), Pro, or HPro as the first residue
NPHC	Patch residue	Replaces the N-terminus with a phenylcarbonyl group; may not be used with Gly (AMINO.rtf), Pro, or HPro as the first residue
NPME	Patch residue	Replaces the peptide nitrogen hydrogen with a methyl group; may not be applied to the first residue
NPOH	Patch residue	Replaces the peptide nitrogen hydrogen with a hydroxyl group; may not be used on an end residue
NPYG	Patch residue	Converts an N-terminal glutamic acid to pyroglutamic acid
NTER	Patch residue	Default N-terminal patch for all standard residues except Gly and Pro (AMINO.rtf)
NTME	Patch residue	Adds a N-methyl group to the N-terminus
NTOO	Patch residue	Replaces the peptide nitrogen with an ester oxygen; may not be applied with Gly as the ester residue; may only be used on non-terminal residues
NTRA	Patch residue	N-terminal patch for all standard residues except Gly (AMINO.rtf) and Pro; may only be applied after the PSF has been generated

Table 47. Amino acid residue topology files (AMINO.RTF and AMINOH.RTF) (Continued)

Name	Type	Description
NTRU	Patch residue	Default N-terminal patch for UALA and UPHE
N3ME	Patch residue	Adds a N,N,N trimethyl group to the N-terminus
PROP	Patch residue	Default N-terminal patch for Pro
SING	Patch residue	Used to chanrge the Cα of a single residue after the NTER and CTER patches are applied
ZTOE	Patch residue	Converts Z-dehydrophenylalanine to E-dehydrophenylalanine

Nucleic Acid Residue Topology File

Table 48 lists residues and patch residues for nucleic acids.

Table 48. Nucleic acid residue topology files (DNA.RTF, DNAH.RTF, RNA.RTF, RNAH.RTF)

Name	Type	Description
G	Residue	Deoxyguanosine (DNA.RTF and DNAH.RTF)
G	Residue	Guanosine (RNA.RTF and RNAH.RTF)
A	Residue	Deoxyadenosine (DNA.RTF and DNAH.RTF)
A	Residue	Adenosine (RNA.RTF and RNAH.RTF)
C	Residue	Deoxycytidine (DNA.RTF and DNAH.RTF)
C	Residue	Cytidine (RNA.RTF and RNAH.RTF)
T	Residue	Deoxythymidine (DNA.RTF and DNAH.RTF)
T	Residue	Thymidine (RNA.RTF and RNAH.RTF)
U	Residue	Uridine (RNA.RTF and RNAH.RTF)
CAMP	Residue	Cyclic AMP (RNA.RTF and RNAH.RTF)
2END	Patch residue	Modifies ribose to a 2’ endo conformation
DEOX	Patch residue	Converts ribose to deoxyribose
FC	Patch residue	Fully charges the phosphate group
5PHO	Patch residue	Places a phosphate group at the 5’ end
3PRI	Patch residue	Places an OH at the 3’ end after PSF generation
5PRI	Patch residue	Places a phosphate at the 5’ end after PSF generation
PUR	Patch residue	Patch for purine bases similar to PYR
PYR	Patch residue	Patch to add H1 to pyrimidines when using base only
5TER	Patch residue	Places a hydroxyl group at the 5’ end
3TER	Patch residue	Places a hydroxyl group at the 3’ end

Carbohydrate residue topology file

Table 49 lists carbohydrate residues and patch residues in alphabetical order. The **Link Polysaccharide Monomers** selection can be accessed from the **Edit** menu in the **Sequence Builder**. Residues are *not* automatically linked together. Links must be explicitly defined.

Table 49. Carbohydrate residue topology file (POLYSACCHARIDEH.RTF)

Name	Type	Description
FRUC	Residue	beta D fructose
FUC	Residue	beta L fucose
GAL	Residue	beta D galactose
GCU	Residue	beta D glucuronic acid
GLU	Residue	beta D glucose
GLUC	Residue	beta D glucose
MAN	Residue	beta D mannose
NAG	Residue	beta D N-acetyl glucosamine
NAM	Residue	beta D N-acetylmuramic acid
RIBO	Residue	beta D ribose
SIQ1	Residue	alpha D-sialic acid (Brady+CHARMm charges)
SIA2	Residue	alpha D-sialic acid (CINDO charges)
AL11	Patch residue	Constructs an aldose link 1-1 (alpha)
AL12	Patch residue	Constructs an aldose link 1-2 (alpha)
AL13	Patch residue	Constructs an aldose link 1-3 (alpha)
AL14	Patch residue	Constructs an aldose link 1-4 (alpha)
AL16	Patch residue	Constructs an aldose link 1-6 (alpha)
ALLO	Patch residue	Converts glucose to allose
ALPH	Patch residue	Converts beta-pyranoses to alpha-pyranoses
ALTR	Patch residue	Converts glucose to altrose
36AN	Patch residue	Creates a 3-6 anhydro galactose
1ASN	Patch residue	Links a pyranose to asparagine ND2
BE11	Patch residue	Constructs an aldose link 1-1 (beta)
BE12	Patch residue	Constructs an aldose link 1-2 (beta)
BE13	Patch residue	Constructs an aldose link 1-3 (beta)
BE14	Patch residue	Constructs an aldose link 1-4(beta)
BE16	Patch residue	Constructs an aldose link 1-6 (beta)
COH	Patch residue	Generates uronic acids from an aldose
GALA	Patch residue	Converts glucose to galactose

Table 49. Carbohydrate residue topology file (POLYSACCHARIDEH.RTF) (Continued)

Name	Type	Description
GULO	Patch residue	Converts glucose to gulose
IDO	Patch residue	Converts glucose to idose
LHEX	Patch residue	Converts D-pyranoses to L-pyranoses
MANN	Patch residue	Converts glucose to mannose
NAG2	Patch residue	Adds an N-acetyl group at the 2-carbon
2NH3	Patch residue	Converts glucose to glucoseamine (2-carbon)
OME	Patch residue	Joins beta O_ME group at C1
2PO4	Patch residue	Constructs a phosphate ester to the hydroxyl group at the 2-carbon
3PO4	Patch residue	Constructs a phosphate ester to the hydroxyl group at the 3-carbon
4PO4	Patch residue	Constructs a phosphate ester to the hydroxyl group at the 4-carbon
6PO4	Patch residue	Constructs a phosphate ester to the hydroxyl group at the 6-carbon
PROT	Patch residue	Links a pyranose to a serine hydroxyl group
SA23	Patch residue	Joins O2 of sialic acid to O3 of Gal
SER	Patch residue	Links a pyranose to a serine hydroxyl group; uses Brady-CHARMm charges
SG23	Patch residue	Links a pyranose to a serine hydroxyl group; uses CINDO charges
2SO4	Patch residue	Constructs a sulfate ester to the hydroxyl group at the 2-carbon
3SO4	Patch residue	Constructs a sulfate ester to the hydroxyl group at the 3-carbon
4SO4	Patch residue	Constructs a sulfate ester to the hydroxyl group at the 4-carbon
6SO4	Patch residue	Constructs a sulfate ester to the hydroxyl group at the 6-carbon
TALO	Patch residue	Converts glucose to talose
1THR	Patch residue	Links a pyranose to Thr hydroxyl
1TYR	Patch residue	Links a pyranose to Tyr hydroxyl
XYLO	Patch residue	Converts D-glucose to D-xylose

Metal ions topology file

The metal ions topology file contains residue definitions for a variety of metal ions. Table 50 lists the residues listed in this file.

Porphyrin topology file

Table 51 lists the residues and patch residues found in this file in alphabetical order.

Table 50. Assorted metal ions topology file (IONS.RTF)

Name	Type	Description
AG	Residue	Silver ion
AL	Residue	Aluminum ion
AU	Residue	Gold ion
BA	Residue	Barium ion
BE	Residue	Beryllium ion
BR	Residue	Bromine ion
CA	Residue	Calcium ion
CD	Residue	Cadmium ion
CL	Residue	Chlorine ion
CO	Residue	Cobalt
CS	Residue	Cesium ion
CU	Residue	Copper ion +1
F	Residue	Fluorine ion
FE	Residue	Iron +2
GA	Residue	Gallium
HG	Residue	Mercury
I	Residue	Iodine
IN	Residue	Indium
K	Residue	Potassium ion
LI	Residue	Lithium ion
MG	Residue	Magnesium ion
MN	Residue	Manganese +2
MO	Residue	Molybdenum
NA	Residue	Sodium ion
NI	Residue	Nickel
PB	Residue	Lead +2
PD	Residue	Palladium
PO_4	Residue	Phosphate group
PT	Residue	Platinum
RB	Residue	Rubidium
SN	Residue	Tin
SO_4	Residue	Sulfate group
SR	Residue	Strontium
TI	Residue	Titanium
V	Residue	Vanadium
ZN	Residue	Zinc ion
ZR	Residue	Zirconium

Table 51. Porphyrin topology file (PORPHYRINH.RTF)

Name	Type	Description
BCL	Residue	Bacteriochlorophyll a (minus the Mg)
BCMG	Residue	Bacteriochlorophyll with Mg^{2+}
CHPA	Residue	Chlorophyll a with phytyl sidechain
CHPB	Residue	Chlorophyll b with phytyl sidechain
CO	Residue	Carbon monoxide
HEM	Residue	Protoporphryn IX
HEMO	Residue	Porphryn IX with Fe and O_2 with no Fe–O bonds
O2	Residue	Diatomic oxygen
PCL	Residue	Phytochlorophyll without Mg^{2+}
PPX	Residue	Protoporphryn IX with no Fe
PRIX	Residue	Protoprophyrin IX
PFE	Patch residue	Adds Fe to protoporphryn IX to make heme
PLO2	Patch residue	Attaches O_2 and HIS to HEM
PLIG	Patch residue	Attaches CO and HIS to HEM
PMG	Patch residue	Adds Mg to bacteriochlorophyll

Protein Data Bank topology file

Table 52 lists the residues and patch residues in the file. The PDBAMINO.RTF files contain polar hydrogen definitions for all amino acids, substrates, and co-factors found in the Protein Data Base. The atom naming convention used in that data base is adopted in this QUANTA file. These files should not be used in combination with AMINO.RTF or AMINOH.RTF.

Table 52. Protein Data Bank topology file (PDBAMINO.RFT)

Name	Type	Description
ABU	Residue	Gamma amino butyric acid
ACE	Residue	Adds acetamide group to N-terminus
ALA	Residue	Alanine
ALB	Residue	Beta alanine
ARG	Residue	Arginine
ARGN	Residue	Neutral arginine
ASN	Residue	Asparagine
ASP	Residue	Aspartic acid

Table 52. Protein Data Bank topology file (PDBAMINO.RFT) (Continued)

Name	Type	Description
ASPH	Residue	Protonated aspartic acid
ATP	Residue	Adenosine triphosphate
BEN	Residue	Benzamidine inhibitor
CYS	Residue	Cysteine
FMN	Residue	Flavin mononucleotide
GDP	Residue	Guanosine diphosphate
GLN	Residue	Glutamine
GLU	Residue	Glutamic acid
GLUH	Residue	Protonated glutamic acid
GLY	Residue	Glycine
HIS	Residue	Histidine (uncharged, proton on ND1)
HSC	Residue	Protonated histidine
HSD	Residue	Histidine isomer (proton on NE2)
HSE	Residue	Homoserine
HYL	Residue	Hydroxyproline
ILE	Residue	Isoleucine
LEU	Residue	Leucine
LYS	Residue	Lysine
LYSN	Residue	Neutral lysine
MET	Residue	Methionine
MP3	Residue	3-phosphogycerate
MPD	Residue	2-methyl-s, 4-petanediol
MTX	Residue	Methotrexate
ORN	Residue	Ornithine
PC	Residue	Phosphocholine, deprotonated
PCA	Residue	
PEN	Residue	Penicillamine
PHE	Residue	Phenylalanine
PRO	Residue	Proline
SER	Residue	Serine
ST2	Residue	
THR	Residue	Threonine
TRP	Residue	Tryptophan
TYR	Residue	Tyrosine
VAL	Residue	Valine
HOH	Residue	TIP3P water model for PDB crystal waters
OH2	Residue	Water
TIP3	Residue	Water model
WAT	Residue	TIP3P water model for crystal waters

Table 52. Protein Data Bank topology file (PDBAMINO.RFT) (Continued)

Name	Type	Description
ACCP	Patch residue	N-terminal patch for ACC.
CALC	Patch residue	Reduces C-terminal carboxyl to an alcohol
CALD	Patch residue	Replaces C-terminus with an aldehyde group (Apply this patch only after the CTER patch has been applied.)
CAMD	Patch residue	Converts the C-terminus to an amide
CAME	Patch residue	Alpha methyl patch for all residues except glycine
CBZS	Patch residue	Replaces C-terminus with a benzyl ester group (Apply this patch only after the CTER patch has been applied.)
CETS	Patch residue	Replaces the C-terminus with a ethyl ester group (Apply this patch only after the CTER patch has been applied.)
CGYL	Patch residue	Provides a charged COO^- group to Gly C-terminal
CH2	Patch residue	Replaces the carbonyl group in the backbone by CH_2
CIS	Patch residue	Converts *trans* peptide to *cis*
CMAM	Patch residue	Converts the C-terminus to an N methyl-amide
CMES	Patch residue	Replaces the C-terminus with a methyl ester group (Apply this patch only after the CTER patch has been applied.)
COOH	Patch residue	Protonated carboxyl C-terminus
CPHS	Patch residue	Replaces the C-terminus with a phenyl group (Apply this patch only after the CTER patch has been applied.)
CTBS	Patch residue	Replaces the C-terminus with a t-bytyl ester group (Apply this patch only after the CTER patch has been applied.)
CTER	Patch residue	Provides a charged COO^- group for the C-terminus
CTRA	Patch residue	Provides a charged COO^- group for the C-terminus
DCOH	Patch residue	Removes COOH from the C-terminus (Not recommended for Pro or Gly.)
DEAM	Patch residue	Removes NH_3 from the N-terminus (Doesn't work in Sequence Builder.)
DISU	Patch residue	Patch for disulfides (Requires two Cys.)
DLNK	Patch residue	Connects the carboxyl group of Asp with the N-terminus of a peptide to form a cyclic peptide (Apply this patch as RES (Asp) - RES (N-term).)
GLYP	Patch residue	N-terminal patch for glycine
HPRP	Patch residue	N-terminal patch for Hyp
ILNK	Patch residue	
KLNK	Patch residue	Connects the amino group of Lys with the C-terminus (Apply this patch as RES (Lys) - RES (N-term).)
LIMG	Patch residue	Creates a peptide bond between the C-terminus of the primary segment and the N-terminus of its image
LINK	Patch residue	Joins two segments and forms a peptide bond or can be used to form a cyclic peptide (Apply this patch as RES(N-term) - RES (C-term). Do not apply the patch to Gly residues or any residue without an alpha-carbon atom type CH1E.)

Table 52. Protein Data Bank topology file (PDBAMINO.RFT) (Continued)

Name	Type	Description
LNK2	Patch residue	Joins two segments and forms a peptide bond or can be used to form a cyclic peptide. (Reverse version of LINK) (Apply this patch as RES(N-term) - RES (C-term). Do not apply the patch to Gly residues or any residue without an alpha-carbon atom type CH1E.)
LNP1	Patch residue	Same as LINK except Pro at the N-terminus
LNP2	Patch residue	Same as LNK2 except Pro at the N-terminus
LNPT	Patch residue	Creates a peptide bond between the C-terminus of the primary segment and the N-terminus of its image
LTOD	Patch residue	Converts an L-residue to a D-residue
NACP	Patch residue	Replaces the N-terminal group with a acetamide group
NACT	Patch residue	Replaces the N-terminal group with a acetamide group (This patch does not work with Pro or Gly as the first residue. Apply this patch only after the NTER patch PROP has been applied.)
NBOC	Patch residue	Replaces the N-terminal group with a tert-bul ester group; provides a charged NH_3^+ group for the N-terminus
NCBZ	Patch residue	Replaces the N-terminus with a carbobenzoxly group (This patch does not work with Pro or Gly as the first residue. Apply this patch only after the NTER patch PROP has been applied.)
NDME	Patch residue	N,N -dimethyl terminal group
NETC	Patch residue	Replaces the N-terminus with an ethyl ester (This patch does not work with Pro or Gly as the first residue. Apply this patch only after the NTER patch PROP has been applied.)
NFA	Patch residue	Replaces the N-terminus with a fatty acid amide $C_{13}H_{27}$ (This patch does not work with Pro or Gly as the first residue. Apply this patch only after the NTER patch PROP has been applied.)
NFAG	Patch residue	Replaces the N-terminus with a fatty acid amide $C_{13}H_{27}$ (This patch is specific for Gly at the N-terminus. Apply this patch only after the NTER patch has been applied.)
NFRM	Patch residue	Replaces the N-terminus with a formamide group
NGUA	Patch residue	Replaces the N-terminus with a guanidino group (Do *not* use for Pro. Change atom type for Cα for Gly and edit dihedrals.)
NH_2	Patch residue	Deprotonated N-terminus
NMEC	Patch residue	Replaces the N-terminus with a methylester (This patch does not work with Pro or Gly as the first residue. Apply this patch only after the NTER patch PROP has been applied. Change atom type for Cα for Gly.)
NPHC	Patch residue	Replaces the N-terminus with a phenylester group (This patch does not work with Pro or Gly as the first residue. Apply patch only after the NTER patch has been applied.)
NPME	Patch residue	N-methyl (for peptides) (Not to be applied to first residue.)
NPYG	Patch residue	Converts N-terminal glutamic acid to pyroglutamic acid
NTER	Patch residue	Provides a charged NH_3^+ group for the N-terminus (This patch does not work with Gly or Pro as the first residue. Use GLYP or PROP. This patch does not work with dehydro amino acids as the first residue.)
N3ME	Patch residue	N,N,N-trimethyl terminal group

Table 52. Protein Data Bank topology file (PDBAMINO.RFT) (Continued)

Name	Type	Description
NTME	Patch residue	N-methyl amino terminal group
NTOO	Patch residue	Replaces the peptide nitrogen in a residue with an ester oxygen (This patch requires a sequence of three residues where the second residue is the ester residue.)
NTRA	Patch residue	Provides a charged NH_3^+ group for the N-terminus with angle definitions (This patch does not work with Pro or Gly or with dehydro amino acids as the first residue.)
NTRU	Patch residue	Provides a charged NH_3^+ group for the N-terminus.
PROP	Patch residue	N-terminal patch for Pro
ZTOE	Patch residue	Converts Z-dehydroPhe to E-dehydroPhe

E. Tool Command Language (TCL)

Overview

Built into QUANTA is a version of the TCL command language, which is similar to UNIX shell languages and LISP. It supports variable expansion, provides constructs for looping and conditional execution, and contains commands for basic system access. It is also possible to combine complex sequences of TCL commands into new commands using TCL's procedure definition mechanism.

QUANTA can execute scripts containing a mixture of QUANTA and TCL commands. To execute such a script, use QUANTA's REPLAY command. Scripts can be used to automate common QUANTA tasks; for example, to perform a particular task on every molecule file in a directory. QUANTA passes all commands, including normal QUANTA commands, through TCL's command processor, allowing the use of TCL variables as arguments to QUANTA commands.

TCL stands for "Tool Command Language" and is pronounced "tickle." It is comprised of two elements:

- A language
- A library

As a simple textual language, it is intended primarily for issuing commands to interactive programs such as text editors, debuggers, illustrators, and shells. It has a simple syntax and is also programmable, so TCL users can write command procedures to provide more powerful commands than those in the built-in set.

Its library package can be embedded in application programs. The TCL library consists of a parser for the language, routines to implement the built-in commands, and procedures that allow each application to extend TCL with additional commands specific to that application. The application program generates TCL commands and passes them to the TCL parser for execution. Commands may be generated by reading characters from an input source, or by associating command strings with elements of the application's user interface, such as menu entries, buttons, or keystrokes.

When the TCL library receives commands it parses them into component fields and executes built-in commands directly. For commands implemented by the application, TCL calls back to the application to execute the

commands. In many cases commands will invoke recursive invocations of the TCL interpreter by passing in additional strings to execute (procedures, looping commands, and conditional commands all work in this way).

An application program gains three advantages by using TCL for its command language.

1. TCL provides a standard syntax. Once users know TCL, they will be able to issue commands easily to any TCL-based application.
2. TCL provides programmability All a TCL application needs to do is to implement a few application-specific low-level commands. TCL provides many utility commands plus a general programming interface for building up complex command procedures. By using TCL, applications need not re-implement these features.

TCL can be used as a common language for communicating between applications. Inter-application communication is not built into the TCL core described here, but various add-on libraries, such as the Tk toolkit, allow applications to issue commands to each other. This makes it possible for applications to work together in much more powerful ways than was previously possible.

Interpreters

The central data structure in TCL is an interpreter (C type “Tcl_Interp”). An Interpreter consists of a set of command bindings, a set of variable values, and a few other miscellaneous pieces of state. Each TCL command is interpreted in the context of a particular interpreter. Some TCL-based applications will maintain multiple interpreters simultaneously, each associated with a different widget or portion of the application. Interpreters are relatively lightweight structures.They can be created and deleted quickly, so application programmers should feel free to use multiple interpreters if that simplifies the application. Eventually TCL will provide a mechanism for sending TCL commands and results back and forth between interpreters, even if the interpreters are managed by different processes.

Data Types

TCL supports only one type of data: strings. All commands, all arguments to commands, all command results, and all variable values are strings. Where commands require numeric arguments or return numeric results, the arguments and results are passed as strings. Many commands expect their

string arguments to have certain formats, but this interpretation is up to the individual commands. For example, arguments often contain TCL command strings, which may get executed as part of the commands. The easiest way to understand the TCL interpreter is to remember that everything is just an operation on astring. In many cases TCL constructs will look similar to more structured constructs from other languages. However, the TCL constructs are not structured at all; they are just strings of characters, and this gives them a different behavior than the structures they may look like.

Although the exact interpretation of a TCL string depends on who is doing the interpretation, there are three common forms that strings take: commands, expressions, and lists. The following sections discuss these three forms in more detail.

Basic Command Syntax

The TCL language has syntactical similarities to both UNIX shells and LISP. However, the interpretation of TCL commands is different than in either UNIX or LISP. A TCL command string consists of one or more commands separated by newline characters or semi-colons. Each command consists of a collection of fields separated by white space (spaces or tabs). The first field must be the name of a command, and the additional fields, if any, are arguments that will be passed to that command. For example, the command

```
> set a 22
```

has three fields: the first, set, is the name of a TCL command, and the last two, a and 22, will be passed as arguments to the set command. The command name may refer either to a built-in TCL command, an application-specific command bound in with the library procedure Tcl-CreateCommand, or a command procedure defined with the proc built-in command. Arguments are passed literally as text strings. Individual commands may interpret those strings in any fashion they wish. The set command, for example, will treat its first argument as the name of a variable and its second argument as a string value to assign to that variable. For other commands arguments may be interpreted as integers, lists, file names, or TCL commands.

Command names should normally be typed completely (i.e., no abbreviations). However, if the TCL interpreter cannot locate a command it invokes a special command named unknown which attempts to find or create the command. For example, at many sites unknown will search through library directories for the desired command and create it as a TCL procedure if it is found. The unknown command often provides automatic completion of

abbreviated commands, but usually only for commands that were typed interactively. It's probably a bad idea to use abbreviations in command scripts and other forms that will be re-used over time; changes to the command set may cause abbreviations to become ambiguous, resulting in scripts that no longer work.

Comments

If the first non-blank character in a command is #, then everything from the # up through the next newline character is treated as a comment and ignored. When comments are embedded inside nested commands (e.g., fields enclosed in braces) they must have properly-matched braces (this is necessary because when TCL parses the top-level command it doesn't yet know that the nested field will be used as a command, so it cannot process the nested comment character as a comment).

Grouping arguments with double-quotes

Normally each argument field ends at the next white space, but double-quotes may be used to create arguments with embedded space. If an argument field begins with a double-quote, then the argument isn't terminated by white space (including newlines) or a semi-colon (see below for information on semi-colons); instead, it ends at the next double-quote character. The double-quotes are not included in the resulting argument. For example, the command

```
>  set a "This is a single argument"
```

will pass two arguments to set: a and This is a single argument. Within double-quotes, command substitutions, variable substitutions, and backslash substitutions still occur, as described below. If the first character of a command field is not a quote, then quotes receive no special interpretation in the parsing of that field.

Grouping arguments with braces

Curly braces may also be used for grouping arguments. They are similar to quotes except for two differences. First, they nest; this makes them easier to use for complicated arguments like nested TCL command strings. Second, the substitutions described below for commands, variables, and backslashes do not occur in arguments enclosed in braces, so braces can be used to prevent substitutions where they are undesirable. If an argument field

begins with a left brace, then the argument ends at the matching right brace. TCL will strip off the outermost layer of braces and pass the information between the braces to the command without any further modification. For example, in the command

```
> set a {xyz a {b c d}}
```

the set command will receive two arguments: a and xyz a{b c d}.

When braces or quotes are in effect, the matching brace or quote need not be on the same line as the starting quote or brace; in this case the newline will be included in the argument field along with any other characters up to the matching brace or quote. For example, the eval command takes one argument, which is a command string; eval invokes the TCL interpreter to execute the command string. The command

```
> eval ( set a 22 set b 33 }
```

will assign the value 22 to a and 33 to b.

If the first character of a command field is not a left brace, then neither left nor right braces in the field will be treated specially (except as part of variable substitution; see below).

Command substitution with brackets

If an open bracket occurs in a field of a command, then command substitution occurs (except for fields~enclosed in braces). All of the text up to the matching close bracket is treated as a TCL command and executed immediately. Then the result of that command.is substituted for the bracketed text. For example, consider the command

```
> set a [set b]
```

When the set command has only a single argument, it is the name of a variable and set returns the contents of that variable. In this case, if variable b has the value foo, then the command above is equivalent to the command

```
> set a foo
```

Brackets can be used in more complex ways. For example, if the variable b has the value foo and the variable c has the value gorp, then the command

```
> set a xyz [set b].[set c]
```

is equivalent to the command

```
> set a xyzfoo.gorp
```

A bracketed command may contain multiple commands separated by newlines or semi-colons in the usual fashion. In this case the value of the last command is used for substitution. For example, the command

```
> set a x[set b 22 expr $b+2]x
```

is equivalent to the command

```
> set a x24x
```

If a field is enclosed in braces then the brackets and the characters between them are not interpreted specially; they are passed through to the argument verbatim.

Variable substitution with $

The dollar sign ($) may be used as a special shorthand form for substituting variable values. If $ appears in an argument that isn't enclosed in braces then variable substitution will occur. The characters after the $, up to the first character that isn't a number, letter, or underscore, are taken as a variable name and the string value of that variable is substituted for the name. For example, if variable foo has the value test, then the command

```
> set a $foo.c
```

is equivalent to the command

```
> set a test.c
```

There are two special forms for variable substitution. If the next character after the name of the variable is an open parenthesis, then the variable is assumed to be an array name and all of the characters between the open parenthesis and the next close parenthesis are taken as an index into the array. Command substitutions and variable substitutions performed on the information between the parentheses before it is used as an index. For example, if the variable **x** is an array with one element named **first** and value **87** and another element named **14** and value **more**, then the command

```
> set a xyz$x(first)zyx
```

is equivalent to the command

```
> set a xyz87zyx
```

If the variable **index** has the value **14**, then the command

```
> set a xyz$x($index)zyx
```

is equivalent to the command

```
> set a xyzmorezyx
```

The second special form for variables occurs when the dollar sign is followed by an open curly brace. In this case the variable name consists of all the characters up to the next curly brace. Array references are not possible in this form: the name between braces is assumed to refer to a scalar variable. For example, if variable **foo** has the value **test**, then the command

```
> set a abc$(foo)bar
```

is equivalent to the command

```
> set a abictestbar
```

Variable substitution does not occur in arguments that are enclosed in braces; the dollar sign and variable name are passed through to the argument verbatim.

The dollar sign abbreviation is simply a shorthand form. **$a** is completely equivalent to **[set a]**; it is provided as a convenience to reduce typing.

For more information on arrays, see section on variables and arrays.

Separating commands with semi-colons

Normally, each command occupies one line (the command is terminated by a newline character). However, semi-colon (“**;**”) is treated as a command separator character; multiple commands may be placed on one line by separating them with a semi-colon. Semicolons are not treated as command separators if they appear within curly braces or double-quotes.

Backslash substitution

Backslashes may be used to insert non-printing characters into command fields and also to insert special characters like braces and brackets into fields without them being interpreted specially. The backslash sequences understood by the TCL interpreter are listed below. In each case, the backslash sequence is replaced by the given character.

For example, in the command

```
> set a \(x\[\Yz\141
```

the second argument to **set** will be “**{x[yza**”.

Table 53Backslash Sequences Understood by the TCL Interpreter

Sequence	Description
\b	Backspace (Ox8)
\f	Form feed (Oxc)
\n	Newline (Oxa)
\r	Carriage-return (Oxd)
\t	Tab (Ox9)
\v	Vertical tab (Oxb)
\{	Left brace (“{“)
\}	Right brace (“}“)
\[	Open bracket (“[“)
\]	Close bracket (“]“)
\$	Dollar sign (“$”)
\<space>	Space (“v”); doesn’t terminate argument
\;	Semi-colon; doesn’t terminate command
\”	Double-quote
\<newline>	Nothing: this joins two lines together into a single line. This backslash feature is unique in that it will be applied even when the sequence occurs within braces
\\	Backslash (“\”)
\ddd	The digits *ddd* (one, two, or three of them) give the octal value of the character. Null characters may not be embedded in command fields; if *ddd* is zero then the backslash sequence is ignored (i.e., it maps to an empty string).

If a backslash is followed by something other than one of the options described above, then the backslash is transmitted to the argument field without any special processing, and the TCL scanner continues normal processing with the next character. For example, in the command

```
> set \*a \\\(foo
```

The first argument to **set** will be ***a** and the second argument will be **\foo**. If an argument is enclosed in braces, the backslash sequence is passed through to the argument as is, without making any special interpretation of the characters in the backslash sequence. In particular, backslashed braces are not counted in locating the matching right brace that terminates the argument. For example, in the command

```
> set a {\abc}
```

the second argument to **set** will be **\{abc**.

This backslash mechanism is not sufficient to generate absolutely any argument structure; it only covers the most common cases. To produce particu-

larly complicated arguments it is probably easiest to use the **format** command along with command substitution.

Command summary

1. A command is just a string.
2. Within a string commands are separated by newlines or semi-colons (unless the newline or semi-colon is within braces or brackets or is backslashed).
3. A command consists of fields. The first field is the name of the command, and may be abbreviated. The other fields are strings that are passed to that command as arguments.
4. Fields are normally separated by white space.
5. Double-quotes allow white space and semi-colons to appear within a single argument. Command substitution, variable substitution, and backslash substitution still occur inside quotes.
6. Braces defer interpretation of special characters. If a field begins with a left brace, then it consists of everything between the left brace and the matching right brace. The braces themselves are not included in the argument. No further processing is done on the information between the braces except that backslash-newline sequences are eliminated.
7. If a field doesn't begin with a brace, then backslash, variable, and command substitution are done on the field. Only a single level of processing is done: the results of one substitution are not scanned again for further substitutions or any other special treatment. Substitution can occur on any field of a command, including the command name as well as the arguments.
8. If the first non-blank character of a command is a #, everything from the # up through the next newline is treated as a comment and ignored.

Expressions

The second major interpretation applied to strings in TCL is as expressions. Several commands, such as expr, for, and if treat one or more of their arguments as expressions and call the TCL expression processor to evaluate them. The operators permitted in TCL expressions are a subset of the operators permitted in C expressions, and they have the same meaning and precedence as the corresponding C operators. Expressions almost always yield

numeric results (integer or floating-point values). For example, the expression

```
> 6.2 + 6
```

evaluates to 14.2. TCL expressions differ from C expressions in the way that operands are specified, and in that TCL expressions support non-numeric operands and string comparisons.

A TCL expression consists of a combination of operands, operators, and parentheses. White space may be used between the operands and operators and parentheses; it is ignored by the expression processor. Where possible, operands are interpreted as integer values. Integer values may be specified in decimal (the normal case), in octal (if the first character of the operand is **0**), or in hexadecimal (if the first two characters of the operand are **0x**). If an operand does not have one of the integer formats given above, then it is treated as a floating-point number if that is possible. Floating-point numbers may be specified in any of the ways accepted by an ANSI-compliant C compiler (except that the "**f**", "**F**", "**l**" and "**L**" suffixes will not be permitted in most installations). For example, **all** of the following are valid floating-point numbers: **2.1**, **3.**, **6e4**, **7.91e+16**. If no numeric interpretation is possible, then an operand is left as a string (and only a limited set of operators may be applied to it).

Operators may be specified in any of the following ways:

1. As an numeric value, either integer or floating-point.
2. As a TCL variable, using standard $ notation. The variable's value will be used as the operand.
3. As a string enclosed in double-quotes. The expression parser will perform backslash, variable, and command substitutions on the information between the quotes, and use the resulting value as the operand.
4. As a string enclosed in braces. The characters between the open brace and matching close brace will be used as the operand without any substitutions.
5. As a TCL command enclosed in brackets. The command will be executed and its result will be used as the operand.
6. An unquoted string consisting of any number of letters, digits~ and underscores (but a digit may not be the first character).

Where substitutions occur above (e.g., inside quoted strings), they are performed by the expression processor. However, an additional layer of substitution may already have been performed by the command parser before the expression processor was called. As discussed below, it is usually best

to enclose expressions in braces to prevent the command parser from performing substitutions on the contents.

For some examples of simple expressions, suppose the variable **a** has the value **3** and the variable **b** has the value **6**. Then the expression on the left side of each of the lines below will evaluate to the value on the right side of the line:

```
>  3.1 + $a                    6.1

>  2 + "$a.$b"                 5.6

>  4*[Iength "6 2]             8

>  {word one}< “word $a”       0
```

The valid operators are listed in the table below, grouped in decreasing order of precedence.

Table 54Valid TCL Expression Operators

Operators	Description
- ~ !	**Unary minus**, **bit-wise NOT**, l**ogical NOT**. None of these operands may be applied to string operands, and **bit-wise NO**T may be applied only to integers.
* / %	**Multiply**, **divide**, **remainder**. None of these operands may be applied to string operands, and **remainder** may be applied only to integers.
+ -	**Add** and **subtract**. Valid for any numeric operands.
<< >>	**Left** and **right shift**. Valid for integer operands only.
< > <= >=	**Boolean less**, **greater**, **less than or equal**, and **greater than or equal**. Each operator produces **1** if the condition is true, **0** otherwise. These operators may be applied to strings as well as numeric operands, in which case string comparison is used.
== !=	**Boolean equal** and **not equal**. Each operator produces a **zero**/**one** result. Valid for all operand types.
&	**Bit-wise AND**. Valid for integer operands only.

Table 54Valid TCL Expression Operators

Operators	Description
^	**Bit-wise exclusive OR**. Valid for integer operands only.
|	**Bit-wise OR**. Valid for integer operands only.
&&	**Logical AND**. Produces a **1** result if both operands are non-zero, **0** otherwise. Valid for numeric operands only (integers or floating-point).
||	**Logical OR**. Produces a **0** result if both operands are zero, **1** otherwise. Valid for numeric operands only (integers and floating-point).
x?y:z	**If-then-else**, as in C. if ***x*** evaluates to non-zero, then the result is the value of ***y***. Otherwise the result is the value of ***z***. The x operand must have a numeric value.

See the C manual for more details on the results produced by each operator. All of the binary operators group left-to-right within the same precedence level. For example, the expression

```
> 4*2 < 7
```

evaluates to **0**. The **&&**, | |, and **?:** operators have “lazy evaluation,” just as in C, which means that operands are not evaluated if they are not needed to determine the outcome. For example, in

```
> $v ? [a] : (b]
```

only one of **[a]** or **[b]** wfll actually be evaluated, depending on the value of **$v**.

All internal computations involving integers are done with the C type ***long***, and all internal computations involving floating-point are done with the C type ***double***. When converting a string to floating-point, exponent overflow is detected and results in a TCL error. For conversion to integer from **string**, detection of overflow depends on the behavior of some routines in the local C library, so it should be regarded as unreliable. In any case, overflow and underflow are generally not detected reliably for intermediate results.

Conversion among internal representations for integer, floating point, and string operands is done automatically as needed. For arithmetic computations, integers are used until some floating-point number is introduced, after which floating point is used. For example,

```
> 5 / 4
```

yields the result **1**, while

```
> 5 / 4.0
```

```
> 5 / ( [length "abcd* chars] + 0.0)
```

both will yield the result **1.25**

String values may be used as operands of the comparison operators, although the expression evaluator tries to do comparisons as integer or floating point when it can. If one of the operands of a comparison is a string and the other has a numeric value, the numeric operand is converted back to a string using the C ***sprintf*** format specifier **%d** for integers and **%g** for floating-point values. For example, the expressions

```
> "0x03" > 02"
```

```
> “0y" < 0x12"
```

both evaluate to **1**. The first comparison is done using integer comparison, and the second is done using string comparison after the second operand is converted to the string “**18**.” In general it is safest to enclose an expression in braces when entering it in a command: otherwise, if the expression contains any white space then the TCL interpreter will split it among several arguments. For example, the command

```
> expr $a + $b
```

results in three arguments being passed to **expr: $a**, **+**, and **$b**. In addition, if the expression isn't in braces then the TCL interpreter will perform variable and command substitution immediately (it will happen in the command parser rather than in the expression parser). In many cases the expression is being passed to a command that will evaluate the expression later (or even many times if, for example, the expression is to be used to decide when to exit a loop). Usually the desired goal is to redo the variable or command substitutions each time the expression is evaluated, rather than once and for all at the beginning. For example, the command

```
> for {set i 1} $i<=10 {incr i} {...}*** WRONG***
```

is probably intended to iterate over all values of **i** from **1** to **10**. After each iteration of the body of the loop, **for** will pass its second argument to the expression evaluator to see whether or not to continue processing. Unfortunately, in this case the value of **i** in the second argument will be substituted once and for all when the **for** command is parsed. If it was **0** before the for command was invoked then **for**’s second argument will be

```
> 0<=10
```

which will always evaluate to **1**, even though **i**’s value eventually becomes greater than **10**. In the above case the loop will never terminate. Instead, the expression should be placed in braces:

```
> for {set i 1} {$i<=10} {incr i} {...}*** RIGHT***
```

This causes the substitution of **i**'s value to be delayed; it will be redone each time the expression is evaluated, which is the desired result.

Lists

The third major way that strings are interpreted in TCL is as lists. A list is just a string with a list-like structure consisting of fields separated by white space. For example, the string

```
> Al Sue Anne John
```

is a list with four elements or fields. Lists have the same basic structure as command strings, except that a newline character in a list is treated as a field separator just like space or tab. Conventions for braces and quotes and backslashes are the same for lists as for commands. For example, the string:

```
> a b\ c {d e (f g h}}
```

is a list with three elements: **a**, **b c**, and **d e {f g h}**. Whenever an element is extracted from a list, the same rules about braces and quotes and backslashes are applied as for commands.

Thus in the example above when the third element is extracted from the list, the result is

```
> d e (f g h}
```

(when the field was extracted, all that happened was to strip off the outermost layer of braces). Command substitution and variable substitution are never made on a list (at least, not by the list-processing commands; the list can always be passed to the TCL interpreter for evaluation).

The TCL commands **concat**, f**oreach**, **lappend**, **lindex**, **linsert**, **list**, **llength**, **lrange**, **lreplace**, **lsearch**, and **lsort** allow you to build lists, extract elements from them, search them, and perform other list-related functions.

Regular Expressions

TCL provides two commands that support string matching using **egrep**-style regular expressions: **regexp** and **regsub**. Regular expressions

are implemented using Henry Spencer's package, and the description of regular expressions below is copied verbatim from his manual entry.

A regular expression is zero or more ***branches***, separated by "|".It matches anything that matches one of the branches.

A branch is zero or more ***pieces***, concatenated. It matches a match for the first, followed by a match for the second, etc.

A piece is an ***atom*** possibly followed by "*****", "**+**", or "**?**". An atom followed by "*" matches a sequence of **0** or more matches of the atom. An atom followed by "**+**" matches a sequence of **1** or more matches of the atom. An atom followed by "**?**" matches a match of the atom, or the null string.

An atom is a regular expression in parentheses (matching a match for the regular expression), a ***range*** (see below), "**.**" (matching any single character), "^" (matching the null string at the beginning of the input string), "**$**" (matching the null string at the end of the input string), a "\" followed by a single character (matching that character), or a single character with no other significance (matching that character).

A ***range*** is a sequence of characters enclosed in "**[]**". It normally matches any single character from the sequence.

- If the sequence begins with "'^", it matches any single character ***not*** from the rest of the sequence.
- If two characters in the sequence are separated by "**-**", this is shorthand for the full list of ASCII characters between them (e.g., "**[0-9]**" matches any decimal digit).
- To include a literal "**]**" in the sequence, make it the first character following a possible "^").
- To include a literal "**-**", make it the first or last character.
- If a regular expression could match two different parts of a string, it will match the one which begins earliest.
- If both begin in the same place but match different lengths, or match the same length in different ways, life gets messier, as follows.

In general, the possibilities in a list of branches are considered in left-to-right order, the possibilities for "*", "+", and "?" are considered longest-first, nested constructs are considered from the outermost in, and concatenated constructs are considered leftmost-first. The match that will be chosen is the one that uses the earliest possibility in the first choice that has to be made. If there is more than one choice, the next will be made in the same manner (earliest possibility) subject to the decision on the first choice. And so forth.

For example, "**(ab I a)b*c**" could match "'abc" in one of two ways:

- The first choice is between “ab” and “a“; since “ab” is earlier, and does lead to a successful overall match, it is chosen. Since the “'b” is already spoken for, the “b” must match its last possibility the empty string-since it must respect the earlier choice.
- In the particular case where no “|”s are present and there is only one “or I/?”, the net effect is that the longest possible match will be chosen. So “W”, presented with “xabbbby”, will match “abbbb”.

> **Note:** Note that if “**ab***” is tried against “xabyabbbz”, it will match “ab” just after “x”, due to the begins-earliest rule. (In effect, the decision on where to start the match is the first choice to be made, hence subsequent choices must respect it even if this leads them to less-preferred alternatives.)

Command Results

Each command produces two results: a code and a string. The code indicates whether the command completed successfully or not, and the string gives additional information. The valid codes are defined in TCL.h, and are:

TCL_OK

This is the normal return code, and indicates that the command completed successfully. The string gives the command’s return value.

TCL_E‘RROR

Indicates that an error occurred; the string gives a message describing the error. In addition, the global variable **error1nfo** will contain human readable information describing which commands and procedures were being executed when the error occurred, and the global variable **errorCode** will contain machine-readable details about the error, if they are available. See the section Built-in TCL Commands on page 206 for more information.

TCL_RETURN

Indicates that the **return** command has been invoked, and that the current procedure (or top-level command or source command) should return

immediately. The string gives the return value for the procedure or command.

TCL_BREAK

Indicates that the **break** command has been invoked, so the innermost loop should abort immediately. The string should always be empty.

TCL,_CONTINUE

Indicates that the **continue** command has been invoked, so the innermost loop should go on to the next iteration. The string should always be empty.

TCL programmers do not normally need to think about return codes, since **TCL_OK** is almost always returned. If anything else is returned by a command, then the TCL interpreter immediately stops processing commands and returns to its caller. If there are several nested invocations of the TCL interpreter in progress, then each nested command will usually return the error to its caller, until eventually the error is reported to the top-level application code. The application will then display the error message for the user.

In a few cases, some commands will handle certain “error” conditions themselves and not them upwards. For example, the **for** command checks for the **TCL_BREAK** code; if it occurs, then **for** stops executing the body of the loop and returns **TCL_OK** to its caller. The **for** command also handles **TCL_CONTINUE** codes and the procedure interpreter handles **TCL_RETURN** codes. The **catch** command allows TCL programs to catch errors and handle them without aborting command interpretation any further.

Procedures

TCL allows you to extend the command interface by defining procedures. A TCL procedure can be invoked just like any other TCL command (it has a name and it receives one or more arguments). The only difference is that its body isn’t a piece of C code linked into the program; it is a string containing one or more other TCL commands. See the proc command on page 225 for information on how to define procedures and what happens when they are invoked.

Variables—scalars and arrays

TCL allows the definition of variables and the use of their values either through $-style variable substitution, the **set** command, or a few other mechanisms. Variables need not be declared; a new variable will automatically be created each time a new variable name is used.

TCL supports two types of variables: scalars and arrays. A scalar variable has a single value, whereas an array variable can have any number of elements, each with a name (called its “index”) and a value. Array indexes may be arbitrary strings; they need not be numeric. Parentheses are used refer to array elements in TCL commands. For example, the command

```
> set x(first) 44
```

will modify the element of x whose index is first so that its new value is **44**. Two-dimensional arrays can be simulated in TCL by using indexes that contain multiple concatenated values. For example, the commands

```
> not a(2,3) 1

> set a (3,6) 2
```

set the elements of a whose indexes are **2**,**3** and **3**,**6**.

In general, array elements may be used anywhere in TCL that scalar variables may be used. If an array is defined with a particular name, then there may not be a scalar variable with the same name. Similarly, if there is a scalar variable with a particular name then it is not possible to make array references to the variable. To convert a scalar variable to an array or vice versa, remove the existing variable with the **unset** command.

The **array** command provides several features for dealing with arrays, such as querying the names of all the elements of the array and searching through the array one element at a time.

Built-in TCL Commands

The TCL library provides the following built-in commands, which will be available in any application using TCL. In addition to these built-in commands, there may be additional commands defined by each application, plus commands defined as TCL procedures. In the command syntax descriptions below, words in **boldface** are literals that you type verbatim to TCL. Words in *italics* are meta-symbols; they serve as names for any of a range of values that you can type. Optional arguments or groups of argu-

ments are indicated by enclosing them in question marks. Ellipses indicate that any number of additional arguments or groups of arguments may appear, in the same format as the preceding argument(s).

append varName value ?value value … ?

Append all of the value arguments to the current value of variable varName. If **varName** doesn't exist, it is given a value equal to the concatenation of all the value arguments. This command provides an efficient way to build up long variables incrementally. For example, "**append a $b**" is much more efficient than "s**et a ab**" if **$a** is long.

array option arrayName ?arg arg … ?

This command performs one of several operations on the variable given by **arrayName**. **arrayName** must be the name of an existing array variable. The option argument determines what action is carried out by the command. The legal options (which may be abbreviated) are:

array anymore arrayName searchId: Returns **1** if there are any more elements left to be processed in an array search, **0** if all elements have already been returned. **searchId** indicates which search on **arrayName** to check, and must have been the return value from a previous invocation of **array startsearch**. This option is particularly useful if an array has an element with an empty name, since the return value from **array nextelement** won't indicate whether the search has been completed.

array donesearch arrayName searchId: This command terminates an array search and destroys all the state associated with that search. **searchId** indicates which search on **arrayName** to destroy, and must have been the return value from a previous invocation of **array startsearch**. Returns an empty string.

array names arrayName: Returns a list containing the names of all of the elements in the array. If there are no elements in the array then an empty string is returned.

array nextelement arrayName searchId: Returns the name of the next element in **arrayName**, or an empty string if all elements of **arrayName** have already been returned in this search. The **searchId** argument identifies the search, and must have been the return value of an **array startsearch** command.

> **Warning:** If elements are added to or deleted from the array, then all searches are automatically terminated just as if **array donesearch** had been invoked, this will cause **array nextelement** operations to fail for those searches at the end of each line.

array size arrayName: Returns a decimal string giving the number of elements in the array.

array startsearch arrayName: This command initializes an element-by-element search through the array given by **arrayName**, such that invocations of the **array nextelement** command will return the names of the individual elements in the array. When the search has been completed, the **array donesearch** command should be invoked. The return value is a search identifier that must be used in **array nextelement** and **array donesearch** commands; it allows multiple searches to be underway simultaneously for the same array.

break

This command may be invoked only inside the body of a loop command such as **for** or **foreach** or **while**. It returns a **TCL-BREAK** code to signal the innermost containing loop command to return immediately.

case string ?in? {patList body ?patList body...?}

Match **string** against each of the **patList** arguments in order. If one matches, then evaluate the following body argument by passing it recursively to the TCL interpreter, and return the result of that evaluation.

- Each **patList** argument consists of a single pattern or list of patterns.
- Each pattern may contain any of the wildcards described under **string match**.
- If a **patList** argument is default, the corresponding body will be evaluated if no **patList** matches **string**.
- If no **patList** argument matches **string** and no default is given, then the case command returns an empty string.

Two syntaxes are provided. The first uses a separate argument for each of the patterns and commands; this form is convenient if substitutions are desired on some of the patterns or commands.

The second form places all of the patterns and commands together into a single argument; the argument must have proper list structure, with the elements of the list being the patterns and commands. The second form makes it easy to construct multi-line case commands, since the braces around the whole list make it unnecessary to include a backslash at the end of each line. Since the **patList** arguments are in braces in the second form, no command or variable substitutions are performed on them; this makes the behavior of the second form different than the first form in some cases.

Below are some examples of case commands:

```
>  case abc in (a b) (format 1) default (format 2) a*
   (format 3)
```

will return **3**,

```
>  case a in ( (a b) (format 1) default (format 2) a*
   (format 3)
```

will return **1**, and

```
>  case xyz ( (a b) (format 1) default (fo-t 21 a*
   (format 3)
```

will return **2**.

catch command ?varName?

The **catch** command may be used to prevent errors from aborting command interpretation. **catch** calls the TCL interpreter recursively to execute command, and always returns a **TCL_OK** code, regardless of any errors that might occur while executing command. The return value from **catch** is a decimal string giving the code returned by the TCL interpreter valuate after executing command. This will be **0** (**TCL_OK**) if there were no errors in command; otherwise it will have a non-zero value corresponding to one of the exceptional return codes (see **TCL.h** for the definitions of code values). If the **varName** argument is given, then it gives the name of a variable; **catch** will set the value of the variable to the string returned from command (either a result or an error message).

cd ?d!rName?

Change the current working directory to **dirName**, or to the home directory (as specified in the HOME environment variable) if **dirName** is not given. If **dirName** starts with a tilde, then tilde-expansion is done as described for **Tcl_TildeSubst**. Returns an empty string.

This command can potentially be disruptive to an application, so it may be removed in some applications. Closes the file given by **fileId**. **fileId** must be the return value from a previous invocation of the open command; after this command, it should not be used anymore. If **fileId** refers to a command pipeline instead of a file, then **close** waits for the children to complete. The normal result of this command is an empty string, but errors are returned if there are problems in closing the file or waiting for children to complete.

concat arg ?arg ...

This command treats each argument as a list and concatenates them into a single list. It permits any number of arguments. For example, the command **concat a b {c d e} {f {g h}}** will return **a b c d e f {g h}** as its result.

continue

This command may be invoked only inside the body of a loop command such as **for** or **foreach** or **while**. It returns a **TCL_CONTINUE** code to signal the innermost containing loop command to skip the remainder of the loop's body but continue with the next iteration of the loop.

env

This variable is maintained by TCL as an array whose elements are the environment variables for the process.

- Reading an element will return the value of the corresponding environment variable.
- Setting an element of the array will modify the corresponding environment variable or create a new one if it doesn't already exist.
- Unsetting an element of env will remove the corresponding environment variable.
- Changes to the **env** array will affect the environment passed to children by commands like exec.

If the entire **env** array is unset then TCL will stop monitoring **env** accesses and will not update environment variables. **eof fileId** returns **1** if an end-of-file condition has occurred on **fileId**, **0** otherwise. **fileId** must have been the return value from a previous call to open, or it may be **stdin**, **stdout**, or **stderr** to refer to one of the standard I/O channels.

error message ?info7 ?code?

Returns a **TCL_ERROR** code, which causes command interpretation to be unwound. Message is a string that is returned to the application to indicate what went wrong.

If the info argument is provided and is non-empty, it is used to initialize the global variable **errorInfo**. **errorInfo** is used to accumulate a stack trace of what was in progress when an error occurred; as nested commands unwind, the TCL interpreter adds information to **errorInfo**. If the **info** argument is present, it is used to initialize **errorInfo** and the first increment of unwind information will not be added by the TCL interpreter. In other words, the command containing the error command will not appear in **errorInfo**; in its place will be **info**. This feature is most useful in conjunction with the

catch command: if a caught error cannot be handled successfully, **info** can be used to return a stack trace reflecting the original point of occurrence of the error

catch {...} errMsg set savedInfo $errorInfo ... error $errMsg $savedInfo

If the code argument is present, then its value is stored in the **errorCode** global variable. This variable is intended to hold a machine-readable description of the error in cases where such information is available; see the section Built-in TCL Commands below for information on the proper format for the variable. If the code argument is not present, then **errorCode** is automatically reset to "NONE" by the TCL interpreter as part of processing the error generated by the command.

error Code: After an error has occurred, this variable will be set to hold additional information about the erTor in a form that is easy to process with programs. **errorCode** consists of a TCL list with one or more elements. The first element of the list identifies a general class of errors, and determines the format of the rest of the list. The following formats for **errorCode** are used by the TCL core; individual applications may define additional formats.

CHILDKILLED pid sigName msn: This format is used when a child process has been killed because of a signal. The second element of **errorCode** will be the process's identifier (in decimal). The third element will be the symbolic name of the signal that caused the process to terminate; it will be one of the names from the include file **signal.h**, such as SIGPIPE. The fourth element will be a short human-readable message describing the signal, such as "write on pipe with no readers" for SIGPIPE.

CHILDSTATUS pid code: This format is used when a child process has exited with a non-zero exit status. The second element of **errorCode** will be the process's identifier (in decimal) and the third element will be the exit code returned by the process (also in decimal).

CHILDSUSP pid code: This format is used when a child process has been suspended because of a signal. The second element of **errorCode** will be the process's identifier, in decimal. The third element will be the symbolic name of the signal that caused the process to suspend; this will be one of the names from the include file **signal.h**, such as SIGTTIN. The fourth element will be a short human-readable message describing the signal, such as "background tty read" for SIGTTINJ*

*This format is used for errors when no additional information is available for an error besides the message returned with the error. In these cases errorCode will consist of a list containing a single element whose contents are NONE.

UNIX errName rnsg: If the first element of **errorCode** is UNIX, then the error occurred during a UNIX kernel call. The second element of the list will contain the symbolic name of the error that occurred, such as ENOENT; this will be one of the values defined in the include file **errno.h**. The third element of the list will be a human-readable message corresponding to **errName**, such as "no such file or directory" for the ENOENT case. To set **errorCode**, applications should use library procedures such as **Tcl_SetErrorCode** and **Tcl_UnixError**, or they may invoke the error command. If one of these methods has't been used, then the TCL interpreter will reset the variable to NONE after the next error.

errorinfo

After an error has occurred, this string will contain one or more lines identifying the TCL commands and procedures that were being executed when the most recent error occurred. Its contents take the form of a stack trace showing the various nested TCL commands that had been invoked at the time of the error.

eval arg ?arg ... ?

eval takes one or more arguments, which together comprise a TCL command (or collection of TCL commands separated by newlines in the usual way). Eval concatenates all its arguments in the same fashion as the **concat** command, passes the concatenated string to the TCL interpreter recursively, and returns the result of that evaluation (or any error generated by it).

exec arg ? arg .

This command treats its arguments as the specification of one or more UNIX commands to execute as subprocesses. The commands take the form of a standard shell pipeline; "|" arguments separate commands in the pipeline and cause standard output of the preceding command to be piped into standard input of the next command. Under normal conditions the result of the **exec** command consists of the standard output produced by the last command in the pipeline.

- If any of the commands in the pipeline exit abnormally or are killed or suspended, then **exec** will return an error and the error message will include the pipelines output followed by error messages describing the abnormal terminations; the **errorCode** variable will contain additional information about the last abnormal termination encountered.
- If any of the commands writes to its standard error file, then **exec** will return an error, and the error message will include the pipeline's output,

followed by messages about abnormal terminations (if any), followed by the standard error output.

- If the last character of the result or error message is a newline, then that character is deleted from the result or error message for consistency with normal TCL return values.
- If an **arg** has the value ">" then the following argument is taken as the name of a file and the standard output of the last command in the pipeline is redirected to the file. In this situation **exec** will normally return an empty string.
- If an **arg** has the value "<" then the following argument is taken as the name of a file to use for standard input to the first command in the pipeline.
- If an argument has the value "<<" then the following argument is taken as an immediate value to be passed to the first command as standard input. If there is no "<" or "<<" argument then the standard input for the first command in the pipeline is taken from the application's current standard input.
- If the last **arg** is "&" then the command will be executed in background.

In this case the standard output from the last command in the pipeline will go to the application's standard output unless redirected in the command, and error output from all the commands in the pipeline will go to the application's standard error file. Each **arg** becomes one word for a command, except for "|", "<", "<<", ">", and "&" arguments, and the arguments that follow "<", "<<", and ">".

The first word in each command is taken as the command name; tilde-substitution is performed on it, and the directories in the PATH environment variable are searched for an executable by the given name. No "glob" expansion or other shell-like substitutions are performed on the arguments to commands.

exit ?returnCode?

Terminate the process, returning **returnCode** to the parent as the exit status. If **returnCode** isn't specified then it defaults to **0**.

expr arg

Calls the expression processor to evaluate **arg**, and returns the result as a string.

file option name ?arg arg ...?

Operate on a file or a file name. **name** is the name of a file; if it starts with a tilde, then tilde-substitution is done before executing the command (see the manual entry for **TCL_TildeSubst** for details). **option** indicates what to do with the file name. Any unique abbreviation for **option** is acceptable. The valid options are:

file atime name: Return a decimal string giving the time at which file name was last accessed. The time is measured in the standard UNIX fashion as seconds from a fixed starting time (often January 1, 1970). If the file doesn't exist or its access time cannot be queried then an error is generated.

file dirname name: Return all of the characters in name up to but not including the last slash character. If there are no slashes in name then return **1**. If the last slash in **name** is its first character, then return "/".

file executable name: Return **1** if **name** is executable by the current user, **0** otherwise.

file exists name: Return **1** if **name** exists and the current user has search privileges for the directories leading to it, **0** otherwise.

file extension name: Return all of the characters in **name** after and including the last dot in name. If there is no dot in **name** then return the empty string.

file isdirectory name: Return **1** if **name** is a directory, **0** otherwise.

file isfile name: Return 1 if **name** is a regular file, 0 otherwise.

file lstat name varName: Same as **stat** option (see below) except uses the **lstat** kernel call instead of **stat**. This means that if **name** refers to a symbolic link the information returned in **varName** is for the link rather than the file it refers to. On systems that don't support symbolic links this option behaves exactly the same as the **stat** option.

file mtime name: Return a decimal siring giving the time at which file name was last modified. The time is measured in the standard UNIX fashion as seconds from a fixed starting time (often January 1, 1970). If the file doesn't exist or its modified time cannot be queried then an error is generated.

file owned name: Return **1** if file name is owned by the current user, **0** otherwise.

file readable name: Return **1** if file name is readable by the current user, **0** otherwise.

file readlink name: Returns the value of the symbolic link given by **name** (i.e., the name of the file it points to). If **name** isn't a symbolic link or its

value cannot be read, then an error is returned. On systems that don't support symbolic links this option is undefined.

file rootname name: Return all of the characters in **name** up to but not including the last "." character in the name. If **name** doesn't contain a dot, then return **name**.

file size name: Return a decimal string giving the size of file name in bytes. If the file doesn't exist or its size cannot be queried then an error is generated.

file stat name varName: Invoke the **stat** kernel call on name, and use the variable given by **varName** to hold information returned from the kernel call. **varName** is treated as an array variable, and the following elements of that variable are set: **atime**, **ctime**, **dev**, **gid**, **ino**, **mode**, **mtime**, **nlink**, **size**, **type**, **uid**. Each element except **type** is a decimal string with the value of the corresponding field from the stat return structure; see the manual entry for **stat** for details on the meanings of the values. The **type** element gives the type of the file in the same form returned by the command **file type**. This command returns an empty string.

file tail name: Return all of the characters in **name** after the last slash. If **name** contains no slashes then return **name**.

file type name: Returns a string giving the type of file name, which will be one of **file**, **directory**, **characterSpecial**, **blockSpecial**, **fifo**, **link**, or **socket**.

file writable name: Return **1** if file name is writable by the current user, **0** otherwise.

The file commands that return 0/1 results are often used in conditional or looping commands, for example:

```
> if {![file exists foo]} then [error {bad file name}}
```

flush fileId

Flushes any output that has been buffered for **fileId**. **fileId** must have been the return value from a previous call to **open**, or it may be **stdout** or **stderr** to access one of the standard I/O streams; it must refer to a file that was opened for writing. This command returns an empty string.

for start test next body

for is a looping command, similar in structure to the C **for** statement.The **start**, **next**, and **body** arguments must be TCL command strings, and **test** is an expression string. The **for** command first invokes the TCL interpreter to execute **start**. Then it repeatedly evaluates **test** as an expression; if the result is non- zero it invokes the TCL interpreter on **body** then invokes the

TCL interpreter on **next**, then repeats the loop. The command terminates when **test** evaluates to **0**.

If a continue command is invoked within **body** then any remaining commands in the current execution of **body** are skipped; processing continues by invoking the TCL interpreter on **next**, then evaluating **test**, and so on. If a **break** command is invoked within **body** or **next**, then the for command will return immediately. The operation of **break** and **continue** are similar to the corresponding statements in C. **for** returns an empty string.

foreach varname list body

In this command, **varname** is the name of a variable, **list** is a list of values to assign to **varname**, and **body** is a collection of TCL commands. For each field in **list** (in order from left to right), **foreach** assigns the contents of the field to **varname** (as if the **lindex** command had been used to extract the field), then calls the TCL interpreter to execute **body**. The **break** and **continue** statements may be invoked inside **body**, with the same effect as the **for** command. **foreach** returns an empty string.

format formatString ?arg arg ...?

This command generates a formatted string in the same way as the C **sprintf** procedure (it uses **sprintf** in its implementation). **formatString** indicates how to format the result, using **%** fields as in **sprintf**, and the additional arguments, if any, provide values to be substituted into the result. All of the **sprintf** options are valid; see the **sprintf** manual page for looping details.

Each **arg** must match the expected type from the **%** field in **formatString**; the **format** command converts each argument to the correct type (floating, integer, etc.) before passing it to **sprintf** for formatting. The only unusual conversion is for **%c**; in this case the argument must be a decimal string, which will then be converted to the corresponding ASCII character value. **format** does backslash substitution on its **formatString** argument, so backslash sequences in **formatString** will be handled correctly even if the argument is in the standard braces. The return value from format is the formatted string.

gets fileId ?varName?

Reads the next line from the file given by **fileId** and discards the terminating newline character.

- If **varName** is specified, then the line is placed in the variable by that name and the valuates return value is a count of the number of characters read (not including the newline).

- If the end of the file is reached before reading any characters then **-1** is returned and varies when **name** is set to an empty string.
- If **varName** is not specified then the return value will be the line (minus the newline character) or an empty string if the end of the file is reached before reading any characters. An empty string will also be returned if a line contains no characters except the body or newline, so **eof** may have to be used to determine what really happened.
- If the last character in the file is not a newline character, then **gets** behaves as if there were an additional newline character at the end of the file. **fileId** must be **stdin** or the return value from a previous call to **open**; it must refer to a file that was opened for reading.

glob ?-nocomplain? filename ?filename …?

This command performs filename globbing, using **csh** rules. The returned value from **glob** is the list of expanded filenames. If **-nocomplain** is specified as the first argument then the empty list may be returned; otherwise an error is returned if the expanded list is empty. The **-nocomplain** argument must be provided exactly; an abbreviation will not be accepted.

global varname ?varname …?

This command is ignored unless a TCL procedure is being interpreted. If so, then it declares the given varname's to be global variables rather than local ones. For the duration of the current procedure (and only while executing the current procedure), any reference to any of the varnames will be bound to a global variable instead of a local one.

history ?option? ?arg arg …?*

The **history** command performs one of several operations related to recently-executed commands recorded in a history list. Each of these recorded commands is referred to as an "event."

When specifying an event to the **history** command, the following forms may be used:

1. A ***number***. If positive, it refers to the event with that number (all events are numbered starting at **1**). If the number is negative, it selects an event relative to the current event (-1 refers to the previous event, -2 to the one before that, and so on).

*This conunand may not be available In all TCL- based applications. Typically, only those that receive command input in a typescript form will support **history**.

2. A ***string***. Selects the most recent event that matches the string. An event is considered to match the string either if the string is the same as the first characters of the event, or if the **string match** command matches the event in the sense of the string.

history add command ?exec?

Add the command argument to the history list as a new event. If **exec** is specified (or abbreviated) then the command is also executed and its result is returned. If **exec** isn't specified then an empty string is returned as result.

history change newValue ?event?

Replace the value recorded for an event with **newValue**. **event** specifies the event to replace, and defaults to the current event (not event -1). This command is intended for use in commands that implement new forms of **history** substitution and wish to replace the current event (which invokes the substitution) with the command created through substitution. The return value is an empty string.

history event ?event?

Returns the value of the event given by **event**. **event** defaults to -1. This command causes history revision to occur: see below for details.

history Info ?count?

Returns a formatted string (intended for humans to read) giving the event number and contents for each of the events in the history list except the current event. If **count** is specified then only the most recent **count** events are returned.

history keep count

This command may be used to change the size of the history list to **count** events. Initially, 20 events are retained in the history list. This command returns an empty string.

history nextid

Returns the number of the next event to be recorded in the history list. It is useful for things like printing the event number in command-line prompts.

history redo ?event?

Re-execute the command indicated by **event** and return its result. Event defaults to **-1**. This command results in history revision: see below for details.

history substitute old new ?event?

Retrieve the command given by **event** (**-1** by default), replace any occurrences of **old** by **new** in the command (only simple character equality is supported; no wildcards), execute the resulting command, and return the result of that execution. This command results in history revision: see below for details.

history words selector ?event?

Retrieve from the command given by **event** (**0** by default) the words given by **selector**, and return those words in a string separated by spaces. The **selector** argument has three forms:

- If it is a single number then it selects the word given by that number (**0** for the command name, **1** for its first argument, and so on).
- If it consists of two numbers separated by a dash, then it selects all the arguments between those two.
- Otherwise **selector** is treated as a pattern; all words matching that pattern (in the sense of **string match**) are returned. In the numeric forms $ may be used to select the last word of a command.

For example, suppose the most recent command in the history list is:

```
>  format {% is %d years old} Alice [expr $ageInMonths/12]
```

Below are some history commands and the results they would produce:

```
>  history words $          >  [expr $ageInMonths/12]
>  history words 1-2        >  {% is %d years old}Alice
>  history words *a*o*      >  {% is %d years old}[expr $ageInMonths/
                              12]
```

History words results in history revision: see below for details.

The **history** options **event**, **redo**, **substitute**, and **words** result in history revision". When one of these options is invoked then the current event is modified to eliminate the **history** command and replace it with the result of the **history** command.

For example, suppose that the most recent command in the history list is

```
>  set a (expr Sb+23
```

and suppose that the next command invoked is one of the ones on the left side of the examples below. The command actually recorded in the history event will be the corresponding one on the right side of these examples.

```
>  history                    >  set a [expr $b+2]
>  history s a b              >  set b [expr $b+2]
>  set c [history w 2]        >  set c [expr $b+2]
```

History revision is needed because event specifiers like **-1** are only valid at a particular time; once more events have been added to the history list a different event specifier would be needed. History revision occurs even when **history** is invoked indirectly from the current event (e.g., a user types a command that invokes a TCL procedure that invokes **history**); the top-level command whose execution eventually resulted in a **history** command is replaced.

If you wish to invoke commands like **history words** without history revision, you can use **history event** to save the current history event and then use **history change** to restore it later.

if test When? trueBody ?else? ?falseBody?

The **if** command evaluates **test** as an expression (in the same way that **expr** evaluates its argument). The value of the expression must be numeric; if it is non-zero then **trueBody** is called by passing it to the TCL interpreter. Otherwise **falseBody** is executed by passing it to the TCL interpreter. The **then** and **else** arguments are optional “noise words” to make the command easier to read. **falseBody** is also optional; if it isn’t specified then the command does nothing if **test** evaluates to zero. The return value from **if** is the value of the last command executed in **trueBody** or **falseBody**, or the empty string if **test** evaluates to zero and **falseBody** isn’t specified.

incr varName ?increment?

Increment the value stored in the variable whose name is **varName**. The value of the variable must be integral. If **increment** is supplied then its value (which must be an integer) is added to the value of variable **varName**; otherwise 1 is added to **varName**. The new value is stored as a decimal string in variable **varName** and also returned as result.

info option ?arg arg ... ?

Provide information about various internals to the TCL interpreter. The legal options (which may be abbreviated) are:

info args procname: Returns a list containling the names of the arguments to procedure **procname**, in order. **procname** must be the name of a TCL command procedure.

info body procname: Returns the body of procedure **procname**. **procname** must be the name of a TCL command procedure.

info cmdcount: Returns a count of the total number of commands that have been invoked in this interpreter.

info commands ?pattern?: If **pattern** isn't specified, returns a list of names of all the TCL commands, including both the built-in commands written in C and the command procedures defined using the **proc** command. If **pattern** is specified, only those names matching **pattern** are returned. Matching is determined using the same rules as for **string match**.

info default procname arg varname: **procname** must be the name of a TCL command procedure and **arg** must be the name of an argument to that procedure. If **arg** doesn't have a default value then the command returns **0**. Otherwise it returns **I** and places the default value of **arg** into variable **varname**.

info exists varName: Returns **1** if the variable named **varName** exists in the current context (either as a global or local variable), returns **0** otherwise.

info globals ?pattern?: If **pattern** isn't specified, returns a list of all the names of currentlydefined global variables. If **pattern** is specified, only those names matching **pattern** are returned. Matching is determined using the same rules as for **string match**.

info level ?number?: If **number** is not specified, this command returns a number giving the stack level of the invoking procedure, or **0** if the command is invoked at top-level. If **number** is specified, then the result is a list consisting of the name and arguments for the procedure call at level number on the stack. If **number** is positive then it selects a particular stack level (1 refers to the top-most active procedure, 2 to the procedure it called, and so on); otherwise it gives a level relative to the current level (0 refers to the current procedure, -1 to its caller, and so on). See uplevel ?level? command ?command ... ? on page 235 for more information on what stack levels mean.

infolibrary: Returns the name of the library directory in which standard TCL scripts are stored. If there is no such directory defined for the current installation then an error is generated. See the library manual entry for details of the facilities provided by the TCL script library. Normally each application will have its own application-specific library in addition to the TCL script library; the location of the application-specific library should be kept in the **$appLibrary** global variable.

i**nfo locals ?paffern?**: If **pattern** isn't specified, returns a list of all the names of currently-defined local variables, including arguments to the current procedure, if any. Variables defined with the **global** and **upvar** commands will not be returned. If **pattern** is specified, only those names

matching **pattern** are returned. Matching is determined using the same rules as for **string match**.

info procs ?pattern?: If **pattern** isn't specified, returns a list of all the names of TCL command procedures. If **pattern** is specified, only those names matching **pattern** are returned. Matching is determined using the same rules as for **string match**.

info script: If a TCL script file is currently being evaluated (i.e., there is a call to **Tcl_EvalFile** active or there is an active invocation of the source command), then this command returns the name of the innermost file being processed. Otherwise the command returns an empty string.

info telversion: Returns the version number for this version of TCL in the form **x.y**, where changes to **x** represent major changes with probable incompatibilities and changes to **y** represent small enhancements and bug fixes that retain backward compatibility.

info vars ?pattern?: If **pattern** isn't specified, returns a list of all the names of currently-visible variables, including both locals and currently-visible globals. If **pattern** is specified, only those names matching **pattern** are returned. Matching is determined using the same rules as for **string match**.

join list ?JoinString?

The list argument must be a valid TCL list. This command returns the string formed by joining all of the elements of list together with j**oinString** separating each adjacent pair of elements. The **joinString** argument defaults to a space character.

lappend varName value ?value value…?

Treat the variable given by **varName** as a list and append each of the value arguments to that list as a separate element, with spaces between elements. If **varName** doesn't exist, it is created as a list with elements given by the value arguments.

lappend is similar to **append** except that the values are appended as list elements rather than raw text. This command provides a relatively efficient way to build up large lists. For example, "**lappend a $b**" is much more efficient than "**set a [concat $a [Iist $bj]**" when **$a** is long.

lindex list Index: Treats list as a TCL list and returns the indexth element from it (**0** refers to the first element of the list). In extracting the element, **lindex** observes the same rules concerning braces and quotes and backslashes as the TCL command interpreter; however, variable substitution and command substitution do not occur. If **index** is negative or greater than

or equal to the number of elements in value, then an empty string is returned.

linsert list Index element Ulement element ... ?

This command produces a new list from list by inserting all of the element arguments just before the indexth element of list. Each element argument will become a separate element of the new list. If **index** is less than or equal to zero, then the new elements are inserted at the beginning of the list. If **index** is greater than or equal to the number of elements in the list, then the new elements are appended to the list.

list arg ?arg...?

This command returns a list comprised of all the args. Braces and back-slashes get added as necessary, so that the **index** command may be used on the result to re-extract the original arguments, and also so that **eval** may be used to execute the resulting list, with **arg1** comprising the command's name and the other args comprising its arguments. **list** produces slightly different results than **concat; concat** removes one level of grouping before forming the list, while **list** works directly from. the original arguments. For example, the command

```
>  list a b (c d e) (f (g h))
```

will return

```
a b (c d a) (f (g h))
```

while **concat** with the same arguments will return

```
a b c d e f (g h)
```

llength list

Treats **list** as a list and returns a decimal string giving the number of elements in it. **lrange list first last list** must be a valid TCL list. This command will return a new list consisting of elements first through last, inclusive.

last may be **end** (or any abbreviation of it) to refer to the last element of the list. If **first** is less than zero, it is treated as if it were zero. If **last** is greater than or equal to the number of elements in the list, then it is treated as if it were **end**. If **first** is greater than **last** then an empty string is returned.*

*"lrange list first first" does not always produce the same results as "lindex list first" (although it often does for simple fields that aren't enclosed in braces); it does, however, produce the same results as "list [lindex list first]".

lreplace list first last ?element element…?

Returns a new list formed by replacing one or more elements of **list** with the element arguments. **first** gives the index in list of the first element to be replaced. If **first** is less than zero then it refers to the first element of list; the element indicated by **first** must exist in the list. **last** gives the index in list of the last element to be replaced; it must be greater than or equal to **first**. **last** may be **end** (or any abbreviation of it) to indicate that all elements between **first** and the end of the list should be replaced.

The **element** arguments specify zero or more new arguments to be added to the list in place of those that were deleted. Each **element** argument will become a separate element of the list. If no **element** arguments are specified, then the elements between **first** and **last** are simply deleted.

lsearch list pattern

Search the elements of **list** to see if one of them matches **pattern**. If so, the command returns the index of the first matching element. If not, the command returns **-1**.

Pattern matching is done in the same way as for the **string match** command. **lsort list** sorts the elements of list, returning a new list in sorted order. ASCII sorting is used, with the result in increasing order.

open fileName ?access?

Opens a file and returns an identifier that may be used in future invocations of commands like **read**, **write**, and **close**. **fileName** gives the name of the file to open; if it starts with a tilde then tilde substitution is performed as described for **Tcl_TildeSubst**. If the first character of **fileName** is "|" then the remaining characters of **fileName** are treated as a command pipeline to invoke, in the same style as for **exec**. In this case, the identifier returned by **open** may be used to write to the command's input pipe or read from its output pipe.

The access argument indicates the way in which the file (or command pipeline) is to be accessed. It may have any of the following values:

r	Open the file for reading only; the file must already exist.
r+	Open the file for both reading and writing; the file must already exist.
w	Open the file for writing only. Truncate it if it exists. If it doesn't exist, create a new file.
w+	Open the file for reading and writing. Truncate it if it exists. If it doesn't exist, create a new file.
a	Open the file for writing only. The file must already exist, and the file is positioned so that new data is appended the file.
a+	Open the file for reading and writing. The file must already exist, and the initial access position is set to the end of the file.

access defaults to **r**. If a file is opened for both reading and writing, then **seek** must be invoked between a read and a write, or vice versa (this restriction does not apply to command pipelines opened with **open**).

When **fileName** specifies a command pipeline and a write-only access is used, then standard output from the pipeline is directed to the current standard output unless overridden by the command. When **fileName** specifies a command pipeline and a read-only access is used, then standard input from the pipeline is taken from the current standard input unless overridden by the command.

proc name args body

The **proc** command creates a new TCL command procedure, **name**, replacing any existing command there may have been by that name. Whenever the new command is invoked, the contents of **body** will be executed by the TCL interpreter. **args** specifies the formal arguments to the procedure. It consists of a list, possibly empty, each of whose elements specifies one argument. Each argument specifier is also a list with either one or two fields. If there is only a single field in the specifier, then it is the name of the argument; if there are two fields, then the first is the argument name and the second is its default value. Braces and backslashes may be used in the usual way to specify complex default values. When **name** is invoked, a local variable will be created for each of the formal arguments to the procedure; its value will be the value of corresponding argument in the invoking command or the argument's default value. Arguments with default values need not be specified in a procedure invocation. However, there must be enough actual arguments for all the formal arguments that don't have defaults, and there must not be any extra actual arguments.

There is one special case to permit procedures with variable numbers of arguments. If the last formal argument has the name **args**, then a call to the procedure may contain more actual arguments than the procedure has for-

mals. In this case, all of the actual arguments starting at the one that would be assigned to **args** are combined into a list (as if the **list** command had been used); this combined value is assigned to the local variable **args**.

When **body** is being executed, variable names normally refer to local variables, which are created automatically when referenced and deleted when the procedure returns. One local variable is automatically created for each of the procedure's arguments. Global variables can only be accessed by invoking the **global** command.

The **proc** command returns the null string. When a procedure is invoked, the procedure's return value is the value specified in a **return** command. If the procedure doesn't execute an explicit return, then its return value is the value of the last command executed in the procedure's body. If an error occurs while executing the procedure body, then the procedure-as-a-whole will return that same error.

puts fileId string ?nonewline?

Writes the characters given by **string** to the file given by **fileId**. **puts** normally outputs a newline character after **string**, but this feature may be suppressed by specifying the **nonewline** argument. Output to files is buffered internally by TCL; the **flush** command may be used to force buffered characters to be output. **fileId** must have been the return value from a previous call to **open**, or it may be **stdout** or **stderr** to refer to one of the standard I/O channels; it must refer to a file that was opened for writing.

pwd

Returns the path name of the current working directory.

read fileId, read fileId nonewline, read fileId numBytes

In the first form, all of the remaining bytes are read from the file given by **fileId**; they are returned as the result of the command. If **nonewline** is specified as an additional argument, then the last character of the file is discarded if it is a newline. In the third form, the extra argument specifies how many bytes to read; exactly this many bytes will be read and returned, unless there are fewer than **numBytes** bytes left in the file; in this case, all the remaining bytes are returned. **fileId** must be **stdin** or the return value from a previous call to **open**; it must refer to a file that was opened for reading.

regexp ?-Indices? ?-nocase? exp string ?matchVar? ?subMatchVar ...?

Determines whether the regular expression **exp** matches part or all of **string** and returns **1** if it does, **0** if it doesn't. See Regular Expressions on page 202 for complete information on the syntax of **exp** and how it is matched against **string**. If the **-nocase** switch is specified then upper-case characters in **string** are treated as lower case during the matching process.

The **-nocase** switch must be specified before **exp** and may not be abbreviated.If additional arguments are specified after **string** then they are treated as the names of variables to use to return information about which part(s) of **string** matched **exp**. **matchVar** will be set to the range of **string** that matched all of **exp**. The first **subMatchVar** will contain the characters in **string** that matched the leftmost parenthesized subexpression within **exp**, the **nextsubMatchVar** will contain the characters that matched the next parenthesized subexpression to the right in **exp**, and so on.

Normally, **matchVar** and the **subMatchVars** are set to hold the matching characters from **execute string**. However, if the **-indices** switch is specified then each variable will contain a list in the two decimal strings giving the indices in **string** of the first and last characters in the matching range of characters. The **-indices** switch must be specified before the exp argument and may not be abbreviated.

If there are more **subMatchVars** than parenthesized subexpressions within **exp**, or if a particular subexpression in **exp** doesn't match the string (e.g., because it was in a portion of the non-expression that wasn't matched), then the corresponding **subMatchVar** will be set to "**-1 -1**" if **-indices** has been specified, or to an empty string otherwise.

regsub ?-all? ?-nocase? exp string subSpec varName

This command matches the regular expression **exp** against **string** using the rules described in Regular Expressions on page 202. If there is no match, then the command returns **0** and does nothing else. If there is a match, then the command returns **1** and also copies **string** to the variable whose name is given by varName.

When copying **string**, the portion of **string** that matched **exp** is replaced with **subSpec**.

- If **subSpec** contains a "&" or "\0", then it is replaced in the substitution with the portion of **string** that matched **exp**.
- If **subSpec** contains a "\n", where *n* is a digit between 1 and 9, then it is replaced in the substitution with the portion of **string** that matched the n^{th} parenthesized subexpression of **exp**. Additional backslashes may be used in **subSpec** to prevent special interpre-

tation of “&” or “\0” or “\n” or backslash. The use of backslashes in **subSpec** tends to interact badly with the TCL parser’s use of backslashes, so it’s generally safest to enclose **subSpec** in braces if it includes backslashes.

- If the **-all** argument is specified, then all ranges in **string** that match **exp** are found and substitution is performed for each of these ranges; otherwise only the first matching range is found and substituted.
- If **-all** is specified, then “&” and “\n” sequences are handled for each substitution using the information from the corresponding match.
- If the **-nocase** argument is specified, then upper-case characters in **string** are converted to lower-case before matching against **exp**; however, substitutions specified by **subSpec** use the original unconverted form of **string**. The **-all** and **-nocase** arguments must be specified exactly. No abbreviations are permitted.

rename oldName newName

Rename the command that used to be called **oldName** so that it is now called **newName**. If **newName** is an empty string (e.g., **{}**)then **oldName** is deleted. The **rename** command returns an empty string as result.

return ?value?

Return immediately from the current procedure (or top-level command or source command), with **value** as the return value. If **value** is not specified, an empty string will be returned as result.

scan string format varnamel ?varname2 … ?

This command parses fields from an input string in the same fashion as the C **sscanf** procedure. **string** gives the input to be parsed and **format** indicates how to parse it, using **%** fields as C in **sscanf**. All of the **sscanf** options are valid; see the sscanf man page for details. Each **varname** gives the name of a variable; when a field is scanned from **string**, the result is converted back into a string and assigned to the corresponding **varname**. The only unusual conversion is for **%c**. For **%c** conversions a single character value is converted to a decimal string, which is then assigned to the corresponding **varname**; no field width may be specified for this conversion.

seek fileId offset ?origin?

Change the current access position for **fileId**. The **offset** and **origin** arguments specify the position at which the next read or write will occur for

fileId. **offset** must be a number (which may be negative) and **origin** must be one of the following:

- **start:** The new access position will be origin bytes from the start of the file.
- **current**: The new access position will be **origin** bytes from the current access position; a negative **origin** moves the access position backwards in the file.
- **end**: The new access position will be **origin** bytes from the end of the file. A negative **origin** places the access position before the end-of-file, and a positive origin places the access position after the end-of-ffle.

The **origin** argument defaults to **start**. **fileId** must have been the return value from a previous call to **open**, or it may be **stdin**, **stdout**, or **stderr** to refer to one of the standard I/O channels. This command returns an empty string.

set varname ?value?

Returns the value of variable **vamame**. If value is specified, then set the value of **vamame** to **value**, creating a new variable if one doesn't already exist, and return its value. If **varname** contains an open parenthesis and ends with a close parenthesis, then it refers to an array element: the characters before the open parenthesis are the name of the array, and the characters between the parentheses are the index within the array. Otherwise **varname** refers to a scalar variable. If no procedure is active, then **varname** refers to a global variable. If a procedure is active, then **varname** refers to a parameter or local variable of the procedure, unless the global command has been invoked to declare **varname** to be global.

source fileName

Read file **fileName** and pass the contents to the TCL interpreter as a sequence of commands to execute in the normal fashion The return value of **source** is the return value of the last command executed from the file. If an error occurs in executing the contents of the file, then the **source** command will return that error. If a return command is invoked from within the file, the remainder of the file will be skipped and the **source** command will return normally with the result from the **return** command. If f**ileName** starts with a tilde, then it is tilde-substituted as described in the **Tcl_TildeSubst** manual entry.

split string ?splitChars?

Returns a list created by splitting **string** at each character that is in the **splitChars** argument. Each element of the result list will consist of the characters from **string** between instances of the characters in **splitChars**. Empty list elements will be generated if **string** contains adjacent characters in **splitChars**, or if the first or last character of **string** is in **splitChars**.If **splitChars** is an empty string then each character of **string** becomes a separate element of the result list. **SplitChars** defaults to the standard white-space characters.

For example:

```
>  split “comp.unix.misc”
```

returns

```
“comp.unix.misc”
```

and

```
>  split “Hello world" {}
```

returns

```
"H e l l o { } w o r l d"
```

string option arg ?arg ...?

Perform one of several **string** operations, depending on option. The legal options (which may be abbreviated) are:

string compare stringl string2: Perform a character-by-character comparison of **strings**, **stringl** and **string2** in the same way as the C **strcmp** procedure. Return **-1**, **0**, or **1**, depending on whether **stringl** is lexicographically less than, equal to, or greater than **string2**.

string first stringl string2: Search **string2** for a sequence of characters that exactly match the characters in **stringl**. If found, return the index of the first character in the first such match within **string2**. If not found, return **-1**.

string index string charIndex: Returns the **charIndex**th character of the **string** argument. A **charIndex** of **0** corresponds to the first character of the string. If **charIndex** is less than **0** or greater than or equal to the length of the string then an empty string is returned.

string last stringl string2: Search **string2** for a sequence of characters that exactly match the characters in **stringl**. If found, return the index of the first character in the last such match within **string2**. If there is no match, then return **-1**.

string length string: Returns a decimal string giving the number of characters in **string**.

string match pattern string: See if **pattern** matches **string**; return **1** if it does, **0** if it doesn't. Matching is done in a fashion similar to that used by the C-shell. For the two strings to match, their contents must be identical except that the following special sequences may appear in pattern:

*****	Matches any sequence of characters in **string**, including a null string.
?	Matches any single character in **string**.
[chars]	Matches any character in the set given by **chars**. If a sequence of the form **x-y** appears in **chars**, then any character between **x** and **y**, inclusive, will match.
\x	Matches the single character **x**. This provides a way of avoiding the special interpretation of the characters ***?[]** in **pattern**.

string range string first last: Returns a range of consecutive characters from **string**, starting with the character whose index is first and ending with the character whose index is last. An index of **0** refers to the first character of the string. **last** may be **end** (or any abbreviation of it) to refer to the last character of the string. If **first** is less than zero then it is treated as if it were zero, and if **last** is greater than or equal to the length of the string then it is treated as if it were **end**. If **first** is greater than last then an empty string is returned.

string tolower string: Returns a value equal to **string** except that all upper-case letters have been converted to lower-case.

string toupper string: Returns a value equal to **string** except that all lower-case letters have been converted to upper-case.

string trim string ?chars?: Returns a value equal to **string** except that any leading or trailing characters from the set given by **chars** are removed. If **chars** is not specified then white space is removed (spaces, tabs, newlines, and carriage returns).

string trimleft string ?chars?: Returns a value equal to **string** except that any leading characters from the set given by **chars** are removed. If chars is not specified then white space is removed (spaces, tabs, new lines, and carriage returns).

string trimright string ?chars?: Returns a value equal to **string** except that any trailing characters from the set given by chars are removed. If chars is not specified then white space is removed (spaces, tabs, new fines, and carriage returns).

tell fileId

Returns a decimal string giving the current access position in **fileId**. **fileId** must have been the return value from a previous call to **open**, or it may be **stdin**, **stdout**, or **stderr** to refer to one of the standard I/O channels.

time command ?count?

This command will call the TCL interpreter **count** times to execute command (or once if **count** isn't specified). It will then return a string of the form:

```
503 microseconds per iteration
```

which indicates the average amount of time required per iteration, in microseconds. **time** is measured in elapsed time, not CPU time.

trace option ?arg arg ...?

Cause TCL commands to be executed whenever certain operations are invoked. At present only variable tracing is implemented. The legal options (which may be abbreviated) are:

trace variable name ops command: Arrange for command to be executed whenever **variable name** is accessed in one of the ways given by **ops**. **name** may refer to a normal variable, an element of an array, or to an array as a whole (i.e,. **name** may be just the name of an array, with no parenthesized index).

If **name** refers to a whole array, then command is invoked whenever any element of the array is manipulated.

ops indicates which operations are of interest, and consists of one or more of the following letters:

r	Invoke command whenever the variable is read..
w	Invoke command whenever the variable is written.
u	Invoke command whenever the variable is unset.

Variables can be unset explicitly with the **unset** command, or implicitly when procedures return (all of their local variables are unset). Variables are also unset when interpreters are deleted, but traces will not be invoked because there is no interpreter in which to execute them.

When the trace triggers, three arguments are appended to command so that the actual command is as follows:

```
>  command namel name2 op
```

Namel and **name2** give the name(s) for the variable being accessed:

- If the variable is a scalar then **namel** gives the variable's name and **name2** is an empty string;
- If the variable is an array element then **namel** gives the name of the array and **name2** gives the index into the array;
- If an entire array is being deleted and the trace was registered on the overall array, rather than a single element, then **namel** gives the array name and **name2** is an empty string.

op indicates what operation is being performed on the variable, and is one of **r**, **w**, or **u** as defined above.

Command executes in the same context as the code that invoked the traced operation: if the variable was accessed as part of a TCL procedure, then command will have access to the same local variables as code in the procedure. This context may be different than the context in which the trace was created.*

For read and write traces, command can modify the variable to affect the result of the traced operation. If command modifies the value of a variable during a read trace, then the value returned by the traced read operation will be the value of the variable after command completes. For write traces, command is invoked after the variable's value has been changed; it can write a new value into the variable to override the original value specified in the write operation.

The value returned by the traced write operation will be the value of the variable when the command completes. If command returns an error during a read or write trace, then the traced operation is aborted with an error. This mechanism can be used to implement read-only variables, for example. Command's result is always ignored.

While a command is executing during a read or write trace, traces on the variable are temporarily disabled. This means that reads and writes invoked by command will occur directly, without invoking command (or any other traces) again. It is illegal to unset a variable while a trace is active for it. It is also illegal to unset an array if there are traces active for any of the array's elements.

When an unset trace is invoked, the variable has already been deleted: it will appear to be undefined with no traces. If an unset occurs because of a procedure return, then the trace will be invoked in the variable context of the procedure being returned to: the stack frame of the returning procedure

*Note that **namel** may not necessarily be the same as the name used to set the trace on the variable; differences can occur if the access is made through a variable defined with the **upvar** command.

will no longer exist. Traces are not disabled during unset traces, so if an unset trace command creates a new trace and accesses the variable, thetrace will be invoked.

If there are multiple traces on a variable they are invoked in order of creation, most-recent first. If one trace returns an error, then no further traces are invoked for the variable. If an array element has a trace set, and there is also a trace set on the array as a whole, the trace on the overall array is invoked before the one on the element.

Once created, the trace remains in effect either until the trace is removed with the **trace vdelete** command described below, until the variable is unset, or until the interpreter is deleted. Unsetting an element of array will remove any traces on that element, but will not remove traces on the overall array.

trace vdelete name ops command: This command returns an empty string. If there is a trace set on **variable name** with the operations and command given by **ops and** command, then the trace is removed, so that command will never again be invoked.

trace vinfo name: Returns a list containing one element for each trace currently set on **variable name**. Each element of the list is itself a list containing two elements, which are the **ops and** command associated with the trace. If **name** doesn't exist or doesn't have any traces set, then the result of the command will be an empty string.

unknown cmdName ?arg arg … ?

This command doesn't actually exist as part of TCL, but TCL will invoke it if it does exist. If the TCL interpreter encounters a command name for which there is not a defined command, then TCL checks for the existence of a command named **unknown**. If there is no such command, then the interpreter returns an error. If the **unknown** command exists, then it is invoked with arguments consisting of the fully-substituted name and arguments for the original non-existent command.

The **unknown** command typically does things like searching through library directories for a command procedure with the name **cmdName**, or expanding abbreviated command names to full-length, or automatically executing **unknown** commands as UNIX sub-processes. In some cases (such as expanding abbreviations) **unknown** will change the original command slightly and then (re-)execute it. The result of the **unknown** command is used as the result for the original non-existent command.

unset name ?name name ... ?

Remove one or more variables. Each name is a variable name, specified in any of the ways acceptable to the **set** command. If a **name** refers to an element of an array, then that element is removed without affecting the rest of the array. If a **name** consists of an array name with no parenthesized index, then the entire array is deleted. The **unset** command returns an empty string as result. An error occurs if any of the variables doesn't exist, or if any of the variables has an active trace.

uplevel ?level? command ?command ... ?

All of the command arguments are concatenated as if they had been passed to **concat**; the result is then evaluated in the variable context indicated by level. **uplevel** returns the result of that evaluation. If **level** is an integer, then it gives a distance (**up** the procedure calling stack) to move before executing the command. If **level** consists of **#** followed by a number then the number gives an absolute level number. If **level** is omitted then it defaults to **1**. **level** cannot be defaulted if the first command argument starts with a digit or #.

For example, suppose that procedure **a** was invoked from top-level, and that it called **b**, and that **b** called **c**. Suppose that **c** invokes the **uplevel** command. If **level** is **1** or **#2** or omitted, then the command will be executed in the variable context of **b**. If level is **2** or **#1** then the command will be executed in the variable context of **a**. If level is **3** or **#0** then the command will be executed at top-level (only global variables will be visible).

The **uplevel** command causes the invoking procedure to disappear from the procedure calling stack while the command is being executed. In the above example, suppose **c** invokes the command:

```
uplevel 1 {set x 43; d}
```

where **d** is another TCL procedure. The **set** command will modify the variable **x** in **b**'s context, and **d** will execute at level 3, as if called from **b**. If it in turn executes the command:

```
uplevel (set x 42)
```

then the **set** command will modify the same variable **x** in **b**'s context; the procedure **c** does not appear to be on the call stack when **d** is executing. The command **info level** may be used to obtain the level of the current procedure.

uplevel makes it possible to implement new control constructs as TCL procedures (for example, **uplevel** could be used to implement the while construct as a TCL procedure).

upvar ?level? otherVar myVar ?otherVar myVar...?

This command arranges for one or more local variables in the current procedure to refer to variables in an enclosing procedure call or to global variables. **level** may have any of all the forms permitted for the **uplevel** command, and may be omitted if the first letter of the first **otherVar** isn't **#** or a digit (it defaults to **1**). For each **otherVar** argument, **upvar** makes the variable by that name in the procedure frame given by **level** (or at global level, if level is **#0**) accessible in the current procedure by the name given in the corresponding **myVar** argument.

The variable named by **otherVar** need not exist at the time of the call; it will be created the first time **myVar** is referenced, just like an ordinary variable. **upvar** may only be invoked from within procedures. Neither **otherVar** or **myVar** may refer to an element of an array.

upvar returns an empty string.

The **upvar** command simplifies the implementation of call-by-name procedure calling and also makes it easier to build new control constructs as TCL procedures. For example, consider the following procedure:

```
>  proc add2 name {upvar $name x set x [expr $x+2]}
```

add.2 is invoked with an argument giving the name of a variable, and it adds two to the value of that variable. Although **add2** could have been implemented using **uplevel** instead of **upvar**, **upvar** makes it simpler for **add2** to access the variable in the caller's procedure frame.

while test body

The **while** command evaluates **test** as an expression (in the same way that **expr** evaluates its argument). The value of the expression must be numeric; if it is non-zero then **body** is executed by passing it to the TCL interpreter. Once **body** has been executed then **test** is evaluated again, and the process repeats until eventually **test** evaluates to a zero numeric value. **continue** commands may be executed inside **body** to terminate the current iteration of the loop, and **break** commands may be executed inside **body** to cause immediate termination of the **while** command. The **while** command always returns an empty string.

QUANTA Script Examples

```
* QUANTA script that searches through a directory for all
* mol files, reads them into QUANTA, and does 500 steps of
* ABNR minimization on them saving the results. This macro
* illustrates the TCL command foreach, glob, and proc.
```

```
* First set up minimization parameters
charm set mini
> algorithm
> abnr
> nstep
> 510
> nprint
> 5
> tolgrd
> 0.01
> tolval
> 0
> step
> 0.02
> tolstp
> 0
> ok

* minim reads in a mol file and minimizes the structure
!proc minim {molecule} {
data read chem $molecule

%CHARMm Minimization
%Save Changes
> OVER

data write chem all notran $molecule

mole remove
}

* loop over all mol files in the current directory
!foreach file [!glob *.mol] {
!minim $file
}


* QUANTA script that automates the process of performing
* MOPAC calculations on a number of molecules. It reads
* in all the MSF files in the current directory and sends
* them to MOPAC. This procedure illustrates the TCL
* commands foreach glob, and proc. It also shows how to
* obtain the root name of a file using the set and file
* commands.

* qm opens an MSF file and performs a MOPAC
* calculation on it
!proc qm {molecule} {
mole open $molecule.msf
exec mopac
> CALC
> title "MOPAC calculation"
> PM3
> zmatauto
> dummy 0
> symmetry 0
> charge 0
> options 0
> ok
> NONE
> $molecule.ain
> ok
> WAIT
> report 0
> conf 0
> coord 1
> charge 1
```

```
> csr 0
> dipole 0
> map 0
> clean 0
> ok
> OVER
> exit
mole remove
}
*loop over all msf files in the current directory for each
structure file
!foreach file [!glob *.msf] {
!set ext [!file rootname $file]
!qm $ext
}


* QUANTA script to loop through a trajectory, saving a
* color snapshot of each frame. The screen save command
* used is SGI specific. This script assumes that the
* Dynamics Animation module is active, and the trajectory
* has been loaded. This script illustrates the TCL commands
* for, set, and incr.

* This sets up the loop. If there are not 100 frames in the
* trajectory, change the “100” in the next line to the
* number of frames loaded.
!for {!set frame 1} {$frame<=100} {!incr frame} {

* Select a frame of the trajectory to avoid having a ghost
* image.
%Select Frame
> $frame

* Pick a frame and write it to an msf file so that
* graphical objects can be created.
%Write Frame to MSF
* This line should specify the msf that was used to create
* the trajectory.
> /usr/paranoid3/bobf/quanta/TCL/lcrn.msf
> ok
> $frame
> NEW
> traj$frame.msf
> ok
> NEW

* First draw the dot surface
surface dot dots big dens 15 radi 100 % select all

* Answer the dialog box which appears
> NO

* Issue the screen save command
sys scrsave frame$frame 9 918 219 970

* Clean up by deleting the dot surface and removing the
* temporary msf.
surface dele dots big dens 15 radi 100 % select all
sys rm traj$frame.msf

}
* QUANTA script to loop through a trajectory, saving a
* color snap* shot of each frame. The screen save command
* used is SGI-specific. A protein ribbon will be added to
* each frame to help visualize the secondary structure.
* This script assumes that the Dynamics Animation module is
* active, and the trajectory has been loaded. This script
```

```
* illustrates the TCL commands for, set, and incr.

* This sets up the loop. If there are not 100 frames in the
* trajectory, change the “100” in the next line to the
* number of frames loaded.

!for {!set frame 1} {$frame<=100} {!incr frame} {

* Select a frame of the trajectory to avoid having a ghost
* image.
%Select Frame...
> $frame

* Pick a frame and write it to an msf file so that
* graphical objects * can be created.
%Write Frame to MSF...
* This line should specify the msf that was used to create
* the trajectory.
> /usr/pranoid3/bobf/quanta/TCL/lcrn.msf
> ok
> $frame
> NEW
> traj$frame.msf
> ok
> NEW

* First, draw the ribbon
object ribbon name default colo 1 attach

* Answer the dialog box which appears
> NO

* Issue the screen save command
sys scrsave frame 9 918 219 970
sys izoom frame frame$frame 0.5 0.5

* Clean up by removing the temporary msf and the initial
* frame
sys rm traj$frame.msf frame

}
```

F. Known Limitations

The following table contains the known limitations of the QUANTA program as of the present release 2006. Where known, a workaround has been provided.

Accelrys bug no.	Description	Workaround
06072s8pq05, 06059s8pq05	On **Tab 7** of the Dials palette, the color does not refresh when changed in line 2 or 3.	The color refreshes when you click **Tab 7** again or move the Dials palette.
06068dhmq01	After performing a valence check on an atom in ChemNote, the atom disappears when you add a bond.	The atom is not really gone. Performing another operation (e.g., clicking **F** for font) causes the atom to redisplay.
06059bxmq02	ALT + F8 moves the Select Unknown Structure window instead of resizing it.	Double-click the title bar or release the mouse button.
06061bxmq06	Unable to increase or decrease the Rock Speed properly using CTRL + middle or right mouse button.	Use the Dial Set B.
06061bxmq05	Unable to bring Flash speed below 1.00 by clicking the 3D view, but able to bring Flash Speed up to 0.01 using Dial Set A.	Use Dial Set A.
06074s8pq02	When you choose **CHARMm | CHARMm Process | Interactive Status**, the text port reports that CHARMm is not running even after CHARMm Initialization.	To check this, choose **CHARMm | Select CHARMm Host**. A window appears that says "CHARMm Already Running".
05096kjbq02	Sequence Builder in QUANTA does not function without a CHARMm license.	No known workaround.
05108039501	Previous to upgrading IRIX, importing a CCP4 map file with the command **Map | Import | CCP4** works correctly. After the upgrade to IRIX 6.5.27m, importing a map to a new mbk file fails after creating the new file.	If writing the mbk fails the first time, it should work on the second attempt.
05104m3kq01	In ChemNote, **Help | Topics** shows no content or message.	No known workaround.
06062dhmq01	In Chemnote , all bonds of in an aromatic ring change to single bonds when attempting to change one bond to a single bond.	No known workaround.

01352015307	In X-ligand, the option to adjust site volume often grossly overestimates the volume needed for the ligand (> 200%). This problem was encountered in half the examples (two of four) examined. The effect was particularly noticeable and detrimental when two copies of the inhibitor were present in moderate proximity to each other. The ligand was placed in various conformations spanning the two distinct density profiles, instead of within one or the other.	No known workaround.
04362gxmq02	If QUANTA is installed in a directory which has a lengthy path (> ~80 characters), then PROBE calculations fail because the path to the parameter file gets truncated.	No known workaround.
05034hjkq01	If you do not use qlauncher or appropriately set F_UFMTENDIAN, there are problems with external programs doing binary I/O.	To set the environment correctly, use the qlauncher script to run external programs standalone. For example: $HYD_EXE/qlauncher.sh $HYD_EXE/search
05049d2pq01	Successively adding eight or more sp3 hydrogen atoms causes QUANTA to crash.	No known workaround.
02177015301	For some molecules, an improper is defined incorrectly at a chiral center, leading to a planar sp3 chiral center after CHARMm minimization. For example, in the case of a benzene ring attached to a cyclopentane ring by a single bond, where carbon 3 of the cyclopentane ring is a Cl atom and all hydrogens are removed, CHARMm will produce after 1000 cycles of minimization a planar sp3 center.	Add hydrogens before using such a molecule for QUANTA calculations.
05047m3kq01	In lower screen resolutions, some text in the status bar may not appear.	Resize the window.
05058m3kq01	Even after closing the molecule the name appears in the 3D window.	No known workaround.
05054k3bq07	It is possible when attempting to customize colors to modify your .cst file such that the colors make it difficult to use QUANTA (e.g., menus are black).	You can return to the default color settings by entering the command **set colo reset**.

05066gxmq01	When running a solvent accessible surface calculation, a warning may appear in top.log asking the user to “remove the lock file surface.lock by entering the QUANTA command 'SYS rm surface.lock’" if no other such jobs are running.	This warning occurs when a previous surface calculation running in the background has not completed before a new calculation attempts to start. Make sure the background surface calculation has completed before initiating a new calculation.
05071s8pq01	If two templates in the ChemNote window are not bonded correctly, you will get the following error message when attempting to save: “You cannot save two structures.”	Ensure that the structures are bonded correctly before saving.
05075gxmq01	The QUANTA non-graphical and batch options -n & -batch) generate the error: ERROR: Can't open display QUANTA terminated abnormally	Both graphical and non-graphical QUANTA must be run from within an X-windows graphical environment.
05054s1nq01	When **Applications | Protein Health** is selected for a second time no colors appear on the molecule even though the following options are selected: 1. Main Chain Conformation 2. Side Chain onformation 3. Chirality 4. Buried Polar Atom 5. Buried Hydrophilic	Clicking any option again causes the colors to reappear.
05069k3bq05	The IBM Thinkpad may hang when you select the **View | Stereo | Crystal Eyes** option.	This is an SGI feature. Do not select this feature on Linux.
05096r1jq01	The IBM Thinkpad may hang when you select any option under **View | Stereo** three or more times.	No known workaround.

Index

D

E

F

M

N

P

U

V